ПВНЗ Університет Короля Данила

Кафедра Інформаційних технологій та програмної інженерії

УДК 004.415

КВАЛІФІКАЦІЙНА РОБОТА

Тема "Розробка статистичної моделі системи прогнозування цін на ринку нерухомості"

Спеціальність 121– "Інженерія програмна забезпечення"
(код і назва спеціальності)

ПОЯСНЮВАЛЬНА ЗАПИСКА

КР.ІПЗ – 17.00.000 ПЗ
(позначення)

Студентка

Філяк Г.Я.
(підпис) (дата) (розшифрування підпису)

Керівник проєкту

Пашкевич О.П.
(посада) (підпис) (дата) (розшифрування підпису)

Допускається до захисту
Завідувач кафедри
к.т.н. Мануляк І.З.
(посада) (підпис) (дата) (розшифрування підпису)

2020

ПЕРЕЛІК ПРИЙНЯТИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API – applicatuion public interface (прикладний програмний інтерфейс)

Data – дані

DB – data bases (база даних)

Lat – координати широти

Long – координати довготи

OS – operating system (операційна система)

Pip – система управління пакетами в Python 3

User – користувач

МН – машинне навчання

Фіча – додатковий атрибут що на щось впливає

ООП – об’єктно орієнтоване програмування

ШІ – штучний інтелект

ВСТУП

Найважливіша частина процесу машинного навчання – це інтерпритація даних і використання цих даних до задачі яку потрібно вирішити. Вибрати випадковим чином алгоритм і видати йому свої дані – неефективне рішення. Перед тим як приступити до побудови моделі, необхідно зрозуміти що являє собою набір даних. Кожний алгоритм відрізняється з точки зору типу оброблених даних і виду вирішуваних задач. В більш широкому контексті, алгоритми і методи машинного навчання являються лиш етапом більш маштабнішого процесу що буде вирішувати певну задачу, і тому необхідно завжди тримати схему цього процесу в голові. Поглиблюючись в технічні аспекти машинного навчання, легко упустити із виду кінцеву мету.

Динаміка зміни цін на ринку нерухомості в умовах кризи кардинально відрізняється від динаміки зміни вартості в умовах економічного зростання, або економічної стабільності. Відомо, що основою ціноутворення на будь-якому ринку, у ринку нерухомості, є співвідношення мікроекономічних чинників – попиту і пропозиції, які є похідною від макроекономічних показників розвитку країни, регіону, міста.

Регресійні моделі використовуються для передбачення цільових змінних в безперервній шкалі, що робить їх привабливими для вирішення багатьох питань в науці і для додатків в інформаційній галузі, такими як розуміння зв'язків між змінними, оцінювання тенденцій або створення прогнозів. Одним із прикладів може бути прогнозування продаж компаній в майбутні місяці.

Задачею повстало розглянути сучасні методи та засобми прогнозування цін на нерухомість, зробити аналіз даних та візуалізацію, розробити модель прогнозування, перевірити ефективність та якість моделі на реальних даних.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ТА ПРОЄКТНИХ РІШЕНЬ ПРИ СТВОРЕННІ МОДЕЛІ ПРОГНОЗУВАННЯ

1.1 Огляд аналогічних моделей

Огляд аналогічних моделей – невід'ємна частина розробки, адже проаналізувавши всі існуючі моделі можна створити оптимальний і в ідеальності кращий продукт. Якщо розглядати з економічного аналізу то існує модель економічного розрахунку.

Економічний розрахунок

В моделюванні динаміки ринку використовуються деякі положення методології розрахунку індексу вартості житлової нерухомості (загальноміського рівня цін на житло).

В основі цієї методології лежать загальновідомі положення, а саме:

1. Оскільки ринок нерухомості є інертною системою, то йому не притаманні різкі коливання цін зі змінною тенденцією від місяця до місяця. Термін інерційності – два-три місяці. І тому підхід прямого розрахунку зміни середньої ціни по місяцях і навіть по тижнях, що досить часто практикується, може призвести до об'єктивно хибних висновків, якщо статистичний "шум" приймається як коливання цін.

2. Найбільш адекватним показником загально ринкових тенденцій є вторинний ринок, оскільки він характеризується конкурентною здатністю і великою кількістю незалежних продавців. Новобудови (чинник пропозиції), хоча і суттєво впливають на ринок нерухомості, проте не є характерними для виявлення загально ринкових тенденцій. Ціна на новобудови є скоріше функцією обсягів продажів, який, у свою чергу, визначається маркетинговою політикою забудовника.

3. Оскільки на ринок виставляється нерухомість, що відноситься до різних цінових діапазонів (житло: "соціалка", "економ-варіант", "преміям", "люкс-клас"),

то криві вартості для різних типів нерухомості хоча і мають свої особливості, проте в першому наближенні змінюються синхронно.

4. Однією з гіпотез формування цін на ринку нерухомості є уявлення щодо існування гладкої кривої, яка виражає зміни загального рівня цін на ринку. За логікою, індекс вартості нерухомості власне і є функцією, яка характеризує загальний рівень цін на житло у даному місті. Індекс вартості – загально ринковий показник, що за своєю структурою характеризує загальні тенденції ринку щодо збільшення або зменшення цін. В межах одного міста ціни на всі типи нерухомості або зростають, або зменшуються, або є у межах якогось певного стабільного рівня. Причому подібні зміни відбуваються приблизно пропорційно одне до одного. Висновком щодо цього є можливість поділу усіх чинників, що впливають на ціну, на дві основні групи.

Перша група – локальні чинники. Вони призводять до того, що ціни на нерухомість відрізняються між собою. Локальні чинники, впливаючи на ціноутворення, не залежать від часу. Вони більш суттєво впливають на вартість, коли загальний рівень цін залишається приблизно постійним, але майже зникають на тлі сильного зростання чи зниження цін.

Друга група чинників, що впливають на ціноутворення, – це глобальні причини. Вони пов’язані з макроекономічними параметрами, якими є рівень розвитку економіки та бізнесу в країні, регіоні, місті. Це дозволяє говорити про порівнянність загального рівня цін в одному регіоні, місті з рівнем цін в іншому, й стверджувати, що співвідношення цін на аналогічну нерухомість у різних містах буде приблизно пропорційним співвідношенню загальному рівню цін у цих містах. Загальний рівень цін є постійною складовою в ціні кожного об’єкта нерухомого майна, не залежно від його характеристик, параметрів та фізичного опису.

Загальний рівень цін (індекс вартості) є відображенням впливу зміни макроекономічних чинників у часі і не залежить від локальних чинників. З цього випливає, що загальний рівень цін є тільки функцією часу. Це твердження особливо актуальне в умовах фінансово-економічної кризи.

Stocker

Stocker – інструмент Python для прогнозування ринку. Як тільки будуть встановлені необхідні бібліотеки (див. Документацію), можна запустити Jupyter Notebook в тій же папці, що і скрипт, і імпортувати клас Stocker:

from stocker import Stocker

Клас тепер доступний для сеансу Jupyter. Створюється об'єкт класу Stocker, передаючи йому будь-який дійсний тікер, наприклад:

amazon = Stocker (“AMZN”)

AMZN Stocker Initialized. Data covers 1997-05-16 to 2018-01-18.

Тепер в розпорядження потрапили 20 років щоденних даних по акціях Amazon для досліджень. Stocker побудований на фінансової бібліотеці Quandl і містить більше 3000 курсів акцій для використання (рис.1.1). Будується простий графік курсу, викликаючи метод plot_stock:

amazon.plot_stock ()

Maximum Adj. Close = 1305.20 on 2018-01-12.

Minimum Adj. Close = 1.40 on 1997-05-22.

Current Adj. Close = 1293.32.

Рисунок 1.1 – Графік курсу на Stocker

Stocker застосовують для виявлення і аналізу загальних трендів і закономірностей, але зараз зосереджується на прогнозуванні майбутньої ціни. Прогнози в Stocker виробляються з використанням адитивної моделі, яка розглядає тимчасові ряди як комбінацію тренда і сезонних змін в різних часових масштабах (щоденний, щотижневий і щомісячний). Stocker використовує “Прогнозування” пакет, розроблений Facebook для адитивного моделювання (рис.1.2). Створення моделі і прогнозування можна виконати в Stocker одним рядком:

Прогнозувати на дні вперед

model, model_data = amazon.create_prophet_model (days = 90)

Predicted Price on 2018-04-18 = $ 1336.98

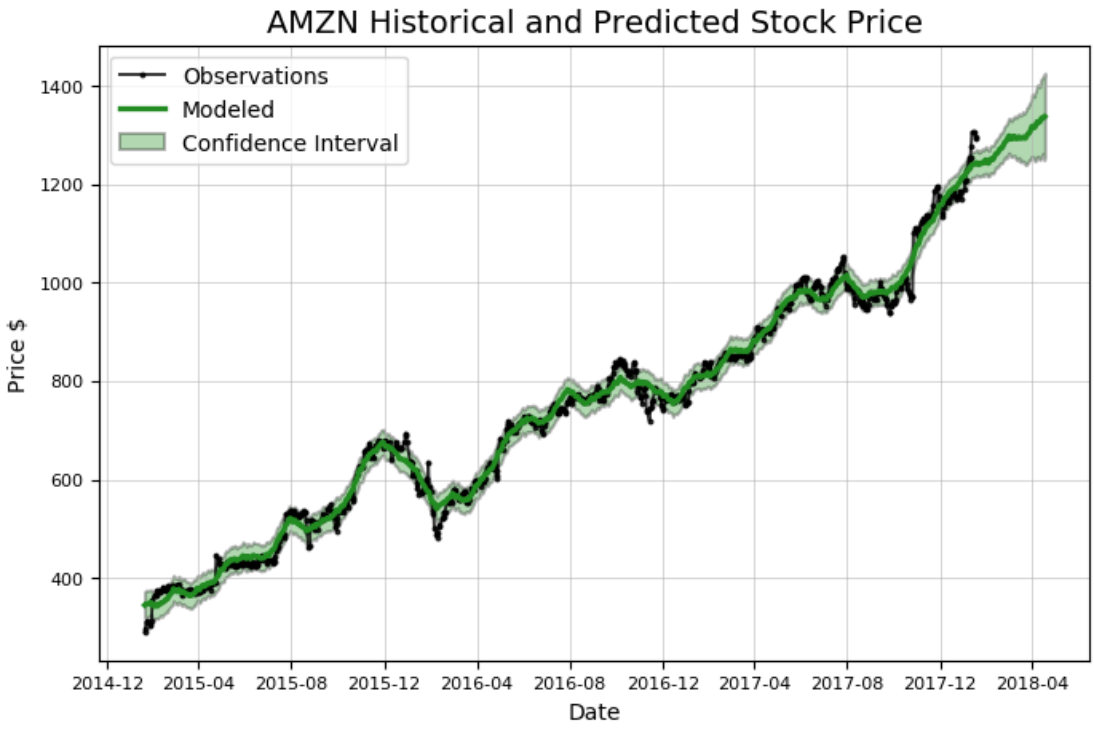


Рисунок 1.2 – Графік прогнозу по даті на Stocker

Прогнозування часових рядів за допомогою Prophet від Facebook

Прогнозування часових рядів – це дуже популярна аналітична задача. Для прогнозування часових рядів в Python використовують такі підходи, як ARIMA, ARCH і т. д. Але підбір параметрів для ARIMA – складний і трудомісткий процес.

Однак 23-го лютого 2017 роки команда Core Data Science з Facebook випустила нову бібліотеку для роботи з тимчасовими рядами - Prophet.

Прогнозується коливання варіації валюти для пари UAH / USD (рис.1.3). Prophet надає API для Python і R, використовується Python. Установку необхідних бібліотек виконується за допомогою пакетного менеджера Conda. Для аналізу використовуються дані за період 01.02.2012 / 05.09.2017.

Цільова змінна буде ціни закриття (Adj Close). Вигляд вхідних даних:

Рисунок 1.3 – Графік прогнозування валют на Prophet

Отже аналогічних моделей не так багато, але їх достатня кількість для аналізу і хороших ідей до майбутнього створення прогнозування.

1.2 Основи архітектури моделі

Зазвичай вирішення залежить від того чи буде значення конкретного признаку більше або менше порогового. Але це підходить тільки для найпростіших задач.

В даному випадку випробовується всі комбінації признаку і порогового значення. Неважко зрозуміти, що з збільшенням складності моделі і розміру набору даних повний перебір всіх комбінацій стає неможливим і приходиться використовувати приближене рішення. Тому для пошуку хорошого вирішення часто використовується змішані методи оптимізації (на щастя, в scikit-learn уже реалізовано достатньо так, що використати їх дуже просто навіть якщо внутрішній код дуже важкий (рис.1.4)).

Рисунок 1.4 – Архітектура видачі результату

Не часто удається знайти ідеальну модель, яка взагалі не робить помилок, отже потрібно вирішити яку саме використовувати. Порогові моделі – одні з найпростіших в машинному навчанні, вони підходять тільки для зовсім тривіальних задач типу класифікації квітів чи щось подібне. Вибір функції виграшу або програшу завжди залежить від конкретної задачі.

Представляючи універсальний алгоритм ми часто намагаємось мінімізувати число помилок, тобто результату максимальної точності. Але якщо одні помилки обходяться дорожче інших то можливо варто змиритись з зменшенням загальної точності на користь мінімізації загальних затримок.

До прикладу архітектура моделі виглядає по різному, але сутність лишається одна:

Алгоритм машинного навчання + Дані = Прогнозуюча модель

Процес побудови такий:

Як можна помітити, вона складається з алгоритму машинного навчання, “натренованого” на даних. З них формується модель прогнозування, далі видається відповідний результат на рис.1.5.

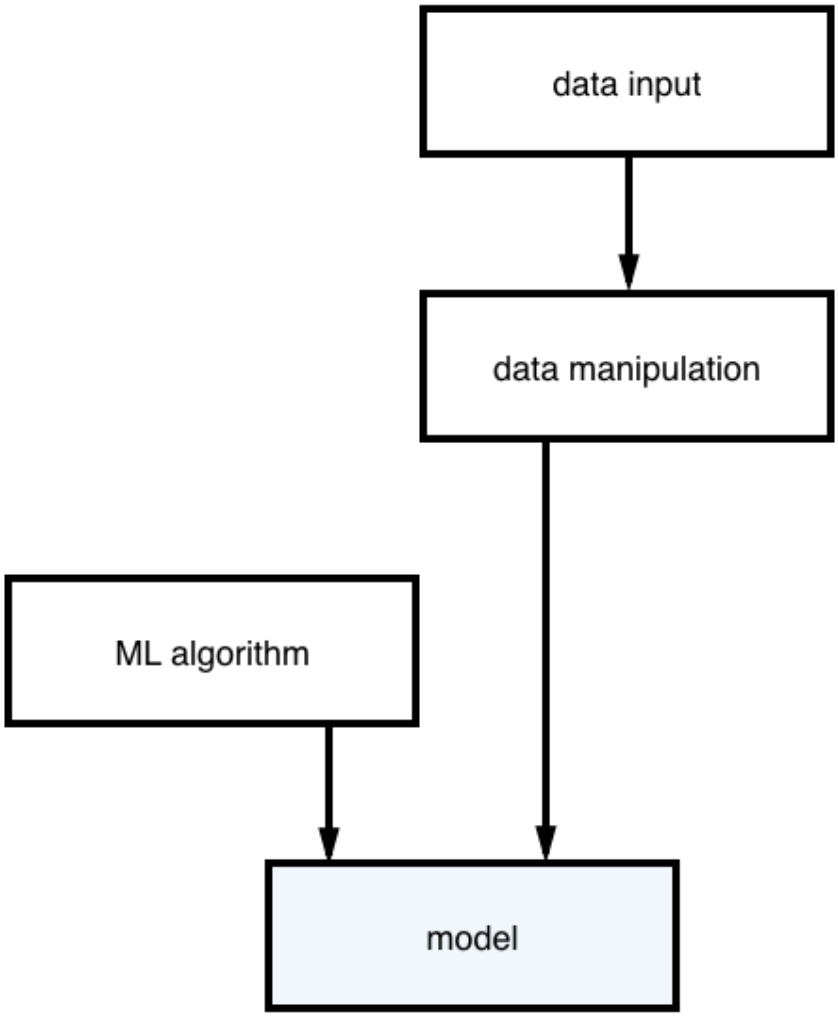


Рисунок 1.5 – Архітектура простої моделі

Робочий процес прогнозування

Мета моделі, яка створюється, полягатиме в класифікації даних за категоріями, їх можна задати:

введення: текст;

результат: категорія.

Існує тренувальний набір даних, в якому усі дані помічені (кожна мітка вказує, до якої категорії він належить). У машинному навчанні такий підхід називається навчанням з учителем.

Класифікуються дані по категоріях, отже, це завдання класифікації.

Для створення моделі можна використати нейронні мережі.

Регресійні моделі в Python:

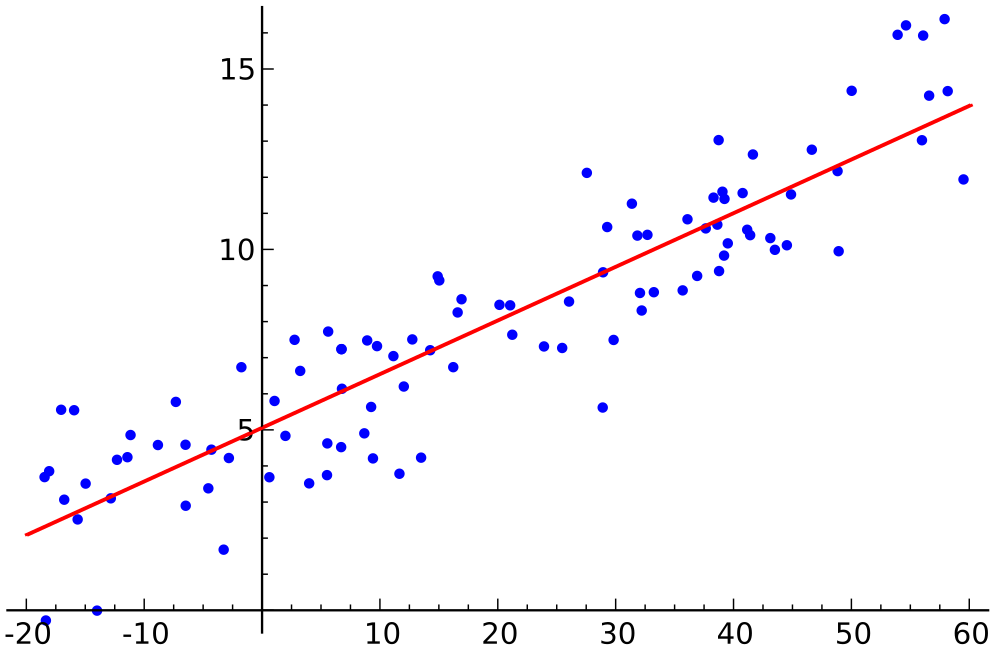


Рисунок 1.6 – Графік регресійної моделі

Лінійна регресія: Коли прогнозується безперервна модель, і ціль змінюється між -∞ і + ∞ (наприклад, температура), найкращою моделлю буде лінійна регресія (рис.1.6). Залежно від того, скільки передбачувачів (функцій) може бути, можна використовувати Просту лінійну регресію (ЗЛР) або Багатолінійну регресію (MLR). Обидві вони використовують один і той же пакет у Python: sklearn.linear_model.LinearRegression (). Документацію для цього можна знайти в інтернеті.

Гамма-регресія: Коли прогнозування виконується для цілі, яка має розподіл від 0 до + ∞, то крім лінійної регресії, для прогнозування може використовуватися Узагальнена лінійна модель (GLM) з розподілом гами. Деталі про GLM можна знайти в інтернеті.

Моделі класифікації в Python

Python пропонує безліч моделей класифікації.

Логістична регресія (LogReg): Ця модель використовується при прогнозуванні цілі в декількох класах. На відміну від K_Nearest Neighbors (kNN),

ця модель добре працює в лінійних випадках. SciKit-Learn пропонує пакет у своїй лінійній бібліотеці моделей: sklearn.linear_model.LogisticRegression () Документацію для цього можна знайти в інтернеті.

KNN (або K-Найближчі сусіди) – непараметрична модель, де логістична регресія є параметричною моделлю. Взагалі кажучи, KNN менш ефективна, ніж модель LogReg і підтримує нелінійні рішення. Ця модель класифікує цілі на основі кількості найближчих сусідів (як випливає з назви) до конкретного класу. Документацію на sklearn.neighbors.KNeighborsClassifer можна знайти в інтернеті. Слід зазначити, що sklearn також пропонує KNeighborsRegressor.

Матриця плутанини для задач класифікації

Оцінка проблем класифікації машинного навчання в Python: 5 + 1 метрик, які мають значення надає огляд показників ефективності класифікації та визначення матриці плутанини і конфузії для цих моделей (рис.1.7).

	Predicted Negative	**Predicted Positive**
Actual Negative	TN	FP
Actual Positive	FN	TP

Рисунок 1.7 – Оцінка класифікацій машинного навчання

Розширені моделі класифікаторів / регресорів

Бібліотеки Python пропонують багато алгоритмів, таких як SciKit-Learn, XGBoost та інші Деякі з цих алгоритмів пропонують і класифікатори, і регресори, а також надають багато параметрів для налаштування.

Дерева рішень: Дерева рішень пропонують настроювані моделі, а також служать основою для більш оптимізованих моделей, таких як RandomForest або GradientBoosting. Документацію можна знайти в інтернеті. Дерева рішень є непараметричним контрольованим навчанням і, таким чином, здатні обробляти нестандартні, а також взаємозалежні змінні. Однак вони можуть легко

переповнювати навчальний набір даних, і користувач повинен бути обережним щодо цього питання (рис.1.8).

Рисунок 1.8 – Схема побудови дерев рішень

Моделі бейджингу (або ансамблеві): Класифікатори бейдженгу встановлюють базовий класифікатор (наприклад, дерево рішення чи будь-який інший класифікатор) на випадкові підмножини оригінального набору даних, а потім агрегують, щоб отримати остаточний прогноз. Це можна зробити шляхом голосування або усередненням.

Підмножина намальованих зразків для кожного базового оцінювача. Повертає динамічно генерований перелік індексів, що ідентифікують зразки, які використовуються для пристосування кожного члена ансамблю, тобто зразків у сумці. Примітка: список створюється заново при кожному виклику властивості, щоб зменшити слід пам'яті об'єкта, не зберігаючи дані вибірки. Таким чином, отримання властивості може бути повільніше, ніж очікувалося.

Детальніше про sklearn.ensemble.BaggingClassifier можна знайти в документації (рис.1.9).

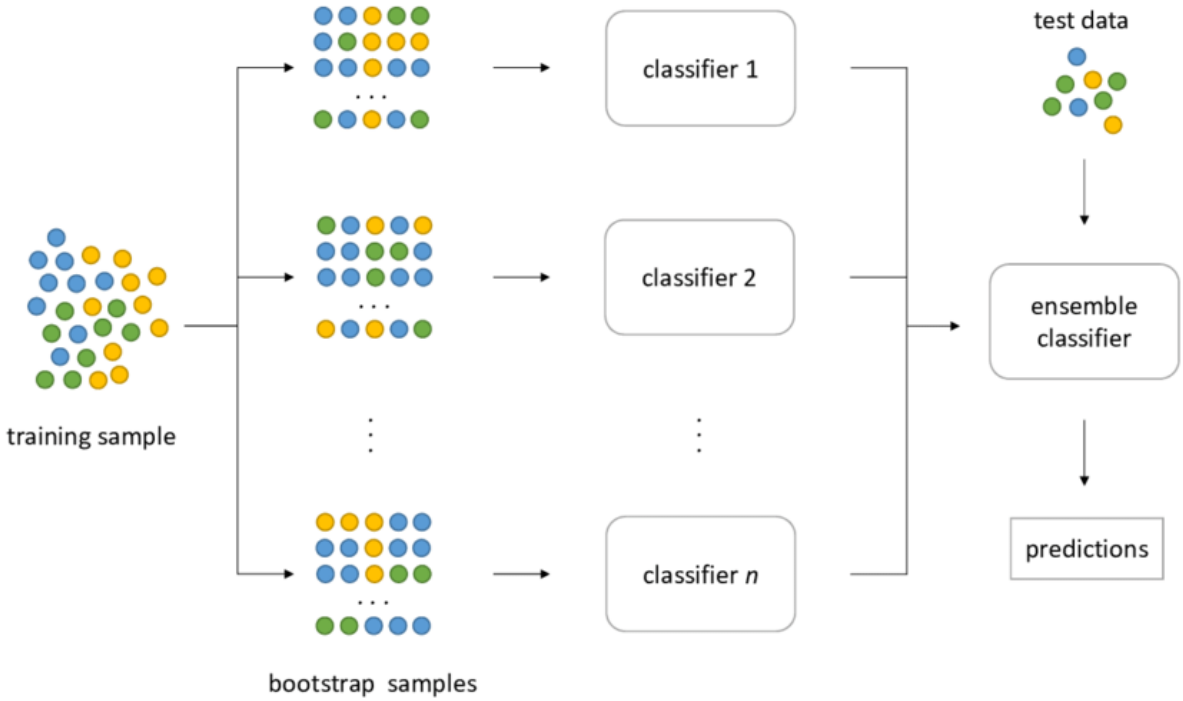


Рисунок 1.9 – Схема ансамблевої моделі

Випадковий ліс: Модель випадкових лісів схожа на ансамблеву модель, але з різницею. Відповідно до документації Sklearn для sklearn.ensemble. RandomForstClassifier: “Випадковий ліс” – це метаоцінювач, який підходить до ряду класифікаторів дерева рішень на різних підвідразках набору даних і використовує усереднення для підвищення точності прогнозування та контролю над пристосуванням. Розмір субпроби завжди той самий, що і вихідний розмір вхідної вибірки, але зразки відбираються із заміною. У цього типу моделей є багато переваг, серед яких висока швидкість навчання, масштабованість, неітераційний характер моделі (вона завжди сходиться). Ще одна важлива перевага моделі полягає в тому, що вона може вирішувати незбалансовані випадки і може використовувати завантажувальну систему для обробки таких випадків. Однак модель може займати багато пам’яті та може переповнювати навчальний набір даних. Якщо даних багато, легко впертися в ліміти зовнішніх

API. Та й отримувати інформацію по HTTP – далеко не завжди оптимальне за швидкістю рішення на рис.1.10.

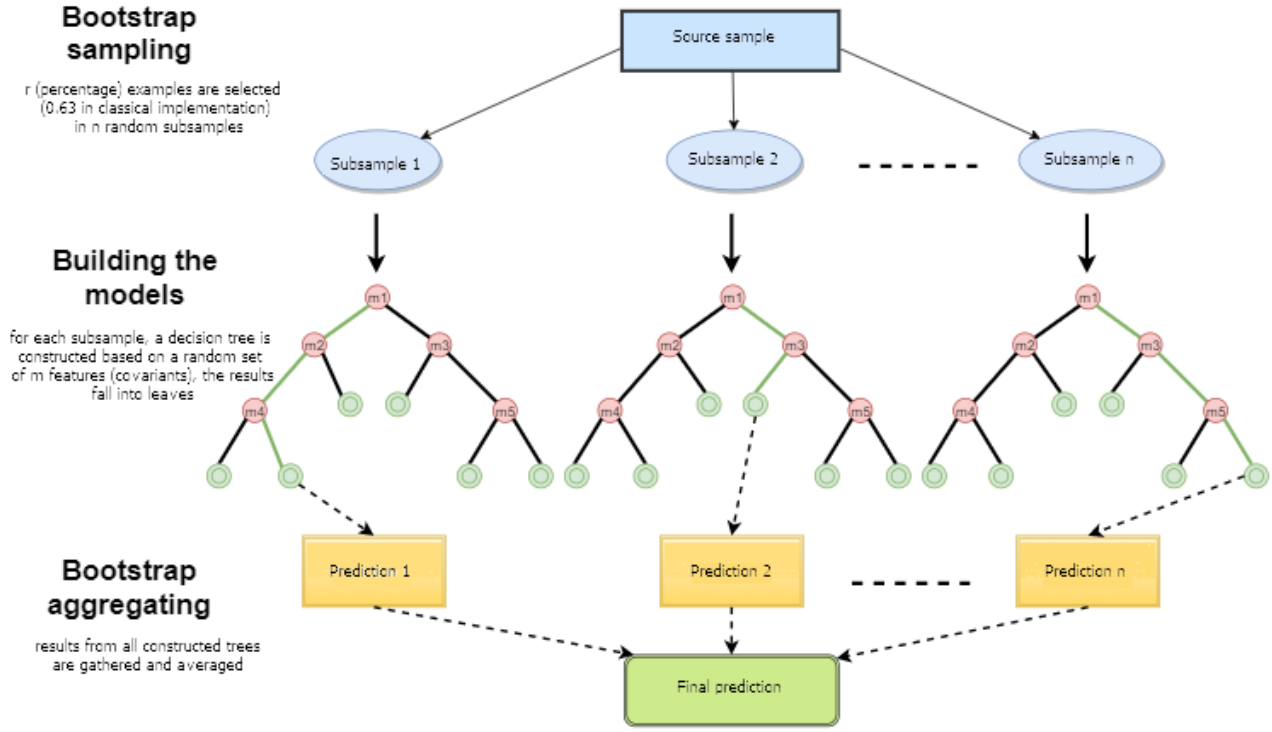


Рисунок 1.10 – Схема моделі випадкового лісу

Моделі голосування: Моделі голосування можна використовувати для упаковки декількох моделей під одну модель. Документація Sklearn називає це “класифікатором м'якого голосування / Мажоритарного правила для непридатних оцінювачів”. У цій моделі кожній моделі голосування може бути присвоєна вага, і тому непридатні моделі будуть знижені. Це схоже на виготовлення мішків, але для різних моделей і з різною вагою (Ансамблева модуль працює лише з однією базовою моделлю, а потім в середньому передбачає прогнози). Sklearn .ensemble.Voting Classifier має більш детальну інформацію про цю модель.

Моделі прискорення: у прискорених моделях кожне дерево отримує важливу вагу на основі своєї точності. Більш точні моделі мали б більшу вагу і, таким чином, більше сприятимуть остаточному прогнозуванню. Вкрай неточні моделі зазнавали б негативної ваги, а це означає, що їх передбачення буде перетворене в остаточному прогнозуванні.

Існує декілька прискорених моделей, але слід зазначити: sklearn. ensemble. GradientBoostingClassifier і sklearn.ensemble.AdaBoostingClassifier.

Scikit-Learn розробила блок-схему вибору правильної моделі (рис.1.11) для проблеми машинного навчання на основі характеристик вибірок, особливостей (або прогнозів) та цілі.

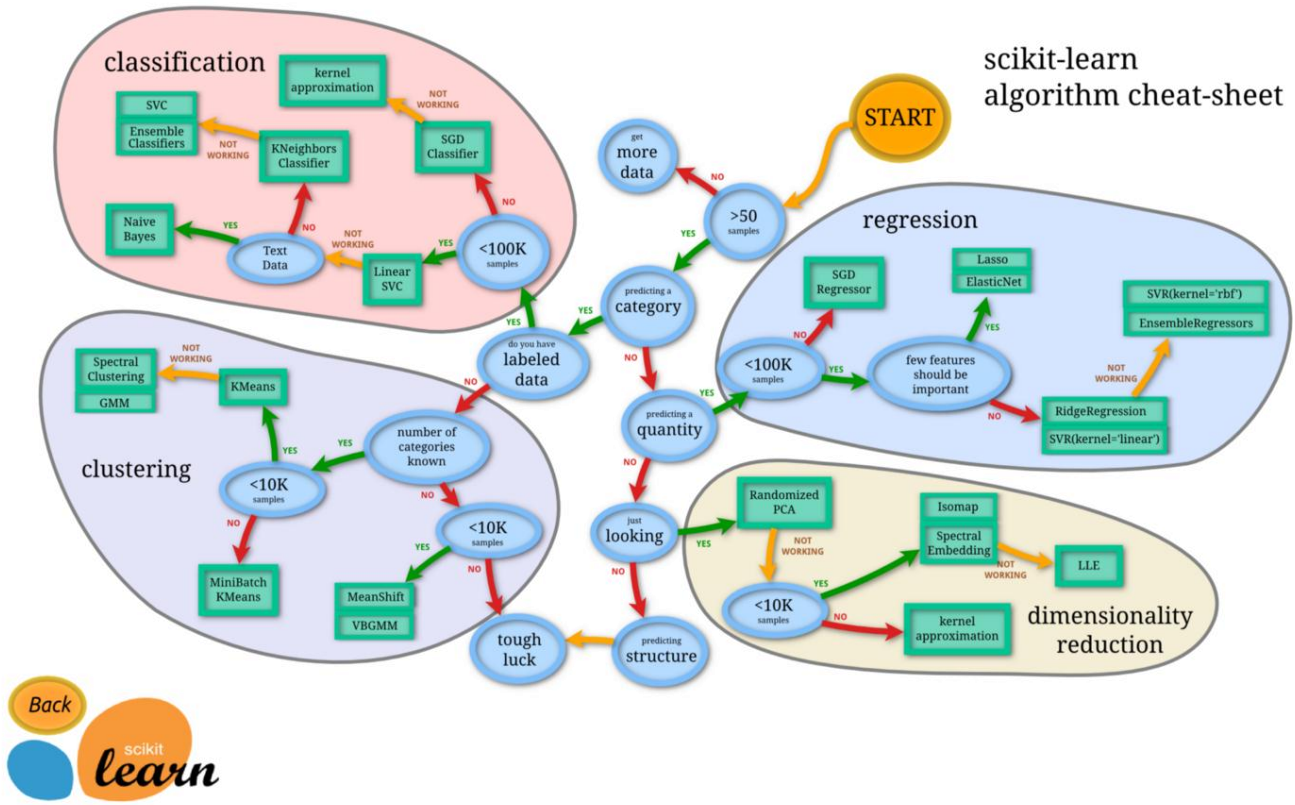


Рисунок 1.11 – Модель розроблена Scikit-Learn

1.3 Постановка задачі та кваліфікаційне проєктування

Завданням кваліфікаційної роботи є створення статистичної моделі для прогнозування ціни на житло за допомогою Лінійної регресії. Аналіз даних на Pandas, Numpy, Sklearn з використанням візуалізації Matplotlib, Pylab, Folium та інших допоміжних бібліотек. Кластеризація даних з KMeans.

Опрацювання даних і програмний код реалізовано на мові Python 3 з використанням інструменту Jupyter notebook.

Модель протестовано на Zillow – найпростіший спосіб знайти будинки для продажу та квартири в оренду. Сайт що дозволяє отримати детальну інформацію про об’єкт, про оренду та викупи, які не можна знайти в MLS.

Набір даних взято з сайту Kaggle.

Висновки до розділу 1

У даному розділі було розглянуто актуальність прогнозування цін ринку економіки та нерухомості, здійснено загальний аналіз існуючих рішень. Крім того, проведено аналіз існуючих моделей та алгоритмів їх реалізації, на основі чого було виділено ряд недоліків та переваг кожного.

Ринок нерухомості, на відміну від фондового або товарного, дуже далекий від досконалого за багатьма показниками, що і визначає особливості його дослідження. Необхідно відзначити, що ринок нерухомості України характеризується безпрецедентною інформаційною закритістю, яка і визначає відсутність серйозної його аналітики.

Моніторинг цінового лістингу пропозицій нерухомості в кращому випадку може відображати флуктуації ринку, які часто формуються його гравцями. Предметом же дослідження ринку є визначення тенденцій його змін в середньостроковому і довгостроковому періодах.

Вивчаючи майже вікову історію функціонування ринків нерухомості в розвинених країнах світу, можна прийти до висновку про те, що недостатня увага, що приділялася до якості аналітичної роботи, – одна з головних причин більшості великих криз на ринках нерухомості.

Розглянувши різні аналогові моделі можна зробити висновок, що є дуже багато можливостей для створення найкращих та найякісніших моделей. Зокрема, експерт в області data science і керівник компанії STATWORX Себастьян Хайнц опублікував на Medium керівництво по створенню моделі глибокого навчання для прогнозування цін акцій на біржі з використанням фреймворку TensorFlow.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ, ПІДКЛЮЧЕННЯ БІБЛІОТЕК ТА ОГЛЯД ДАНИХ

2.1 Вибір технологій

Якщо людина розуміє принцип роботи певного алгоритму, то будь-яка мова програмування може підійти для реалізації того чи іншого методу і вирішення прикладної задачі. Звісно, це буде потребувати додаткового часу, можливо ефективність реалізації буде гіршою за аналоги. Але розуміння логіки процесу робить вибір мови чи сфери застосування алгоритму – необмеженими.

Шту́чний інтеле́кт (англ. Artificial intelligence, AI) – розділ комп'ютерної лінгвістики та інформатики, що опікується формалізацією проблем та завдань, які подібні до дій, які виконує людина.

Машинне навчання – це розділ штучного інтелекту, що має за основу побудову та дослідження систем, які можуть самостійно навчатись з даних. Наприклад, система машинного навчання може бути натренована на електронних повідомленнях для розрізняння спаму і прийнятних повідомлень. Після навчання вона може бути використана для класифікації нових повідомлень електронної пошти на спам та не-спам. В основі машинного навчання розглядаються уявлення та узагальнення. Представлення даних і функцій оцінки цих даних є частиною всіх систем машинного навчання, наприклад, у наведеному вище прикладі, повідомлення електронною поштою, ми можемо уявити лист як набір англійських слів, просто відмовившись від порядку слів. Узагальнення є властивістю, яку система буде застосовувати добре на невидимих примірниках даних; умови, за яких це може бути гарантовано, є ключовим об’єктом вивчення в полі обчислювальної теорії навчання. Існує широкий спектр завдань машинного навчання та успішних застосувань. Оптичне розпізнавання символів, в яких друковані символи розпізнаються автоматично, та ґрунтуються на попередніх прикладах, є класичним підходом техніки машинного навчання. 1959 року Артур

Самуїл визначив машинне навчання як “Поле дослідження, яке дає комп'ютерам можливість навчатися, не будучи явно запрограмованими”.

Говорячи про архітектуру систем ШІ, перш за все розуміють організацію структури, в рамках якої відбувалося б застосування знань і вирішення проблем в конкретній предметній області. Вибір відповідної структури, властивості і функції компонентів систем ІІ, особливо виробничих, визначається і спрямовується за принципами інженерії знань. На формування цих принципів в значній мірі впливають як специфіка предметної області, так і характер завдань і функцій, вирішення яких покладається на інтелектуальні системи.

За останні роки виникла ціла екосистема бібліотек, в яких досить грамотно реалізовані найбільш відомі базові алгоритми МН. Сьогодні не обов’язково реалізовувати кожен з алгоритмів з нуля. Зрозуміло, що для людини, яка займається МН, потрібно розуміти, як працює кожен із алгоритмів.

Однією з мов програмування, яка максимально використовується для вирішення прикладних задач, є мова Python. Ця мова має декілька переваг. Вона досить проста у вивченні і, як правило, це мова, яка потребує низького рівня входу.

Людина, яка має базові знання з теорії алгоритмів і математики, досить просто може освоїти базовий функціонал, методи і синтаксис для того, щоб вирішувати прикладні задачі.

Саме на базі цієї мови реалізована велика кількість бібліотек, які надають у зручному виді більшість доступних алгоритмів.

Python

Python – інтерпретована об’єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв’язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду (рис.2.1). Інтерпретатор Python та стандартні бібліотеки доступні як у

скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об’єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Рисунок 2.1 – Логотип Python

Серед основних її переваг можна назвати такі:

1. Чистий синтаксис (для виділення блоків слід використовувати відступи).

2. Переносність програм (що властиве більшості інтерпретованих мов).

3. Стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу).

4. Можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв’язання простих задач).

5. Стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python.

6. Зручний для розв’язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор).

7. Відкритий код (можливість редагувати його іншими користувачами).

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована

мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови Python і багата Стандартна бібліотека (як вихідні тексти, так і бінарні дистрибутиви для всіх основних операційних систем) можуть бути отримані з сайту Python www.python.org, і можуть вільно розповсюджуватися. Цей самий сайт має дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Розробка мови Python була розпочата в кінці 1980-х років співробітником голландського інституту CWI Гвідо ван Россумом. Для розподіленої ОС Amoeba потрібна була розширювана скриптова мова, і Гвідо почав писати Python на дозвіллі, запозичивши деякі напрацювання для мови ABC (Гвідо брав участь у розробці цієї мови, орієнтованої на навчання програмування). У лютому 1991 року Гвідо опублікував вихідний текст в групі новин alt.sources. Мова почала вільно поширюватися через Інтернет і сподобалася іншим програмістам. З 1991 року Python є цілком об’єктно-орієнтованим. Python також запозичив багато рис таких мов, як C, C++, Modula-3 і Icon, й окремі риси функціонального програмування з Ліспу.

З грудня 2008 року, після тривалого тестування, вийшла перша версія Python 3000 (або Python 3.0, також використовується скорочена Py3k). У Python 3000 усунено багато недоліків архітектури з максимально можливим (але не повним) збереженням сумісності зі старішими версіями. На сьогодні підтримуються Python версії 3.

Розробники мови Python є прихильниками певної філософії програмування, яку називають “The Zen of Python” (“Дзен Пайтона”). Її текст можна отримати у інтерпретаторі Python за допомогою команди import this (лише один раз за сесію). Автором цієї філософії вважається Тім Пейтерс.

З'явившись порівняно пізно, Python створювався під впливом багатьох мов програмування:

1. ABC – відступи (поля) для групування операторів, високорівневі структури даних (map) (фактично, Python створювався як спроба виправити помилки, допущені при проектуванні ABC);

2. Modula-3 – пакети, модулі, використання else спільно з try та except, іменовані аргументи функцій (на це також вплинув Common Lisp);

3. C, C++ – деякі синтаксичні конструкції (як пише сам Гвідо ван Россум – він використовував найбільш несуперечливі конструкції з C, щоб не викликати неприязнь у Сі-програмістів до Python;

4. Smalltalk – об’єктно-орієнтоване програмування;

5. Lisp – окремі риси функціонального програмування (lambda, map, reduce, filter та інші);

6. Fortran – зрізи масивів, комплексна арифметика;

7. Miranda – спискові вирази;

8. Java – модулі logging, unittest, threading (частина можливостей оригінального модуля не реалізована), xml.sax стандартної бібліотеки, спільне використання finally та except при обробці винятків, використання @ для декораторів;

9. Icon – генератори.

Більша частина інших можливостей Python (наприклад, байт-компіляція вихідного коду) також була реалізована раніше в інших мовах.

Python портований і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, всі варіанти UNIX (включаючи FreeBSD та GNU/Linux), Plan 9, Mac OS та Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 та навіть OS/390, Symbian та Android.

У міру старіння платформи її підтримка в основній гілці мови припиняється. Наприклад, з версії 2.6 припинена підтримка Windows 95, Windows 98 та Windows ME. Однак на цих платформах можна використовувати попередні

версії Python – спільнота активно підтримує версії Python починаючи від 2.3 (для них виходять виправлення).

При цьому, на відміну від багатьох портованих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Навіть більше, існує спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретатору виконуватися на будь-якій системі, яка підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python й навіть бути написаними на ньому. Також кілька проєктів забезпечують інтеграцію з платформою Microsoft.NET, основні з яких – IronPython та Python.Net.

Дизайн мови Python побудований навколо об’єктно-орієнтованої моделі програмування. Реалізація ООП в Python є елегантною, потужною та добре продуманою, але разом з тим, достатньо специфічною в порівнянні з іншими об’єктно-орієнтованими мовами.

Можливості та особливості:

1. Класи є одночасно об'єктами з усіма нижче наведеними можливостями.

2. Успадкування, в тому числі множинне.

3. Поліморфізм (всі функції віртуальні).

4. Інкапсуляція (два рівні – загальнодоступні та приховані методи і поля). Особливість – приховані члени доступні для використання та помічені як приховані лише особливими іменами.

5. Спеціальні методи, що керують життєвим циклом об’єкта: конструктори, деструктори, розподільники пам'яті.

6. Перевантаження операторів (усіх, крім is, “.”, “=” і символьних логічних).

7. Властивості (імітація поля за допомогою функцій).

8. Управління доступу до полів (емуляція полів і методів, частковий доступ тощо).

9. Методи для управління найпоширенішими операціями (істиннісне значення, len(), глибоке копіювання, серіалізація, ітерація по об’єкту, …)

10. Мета програмування (управління створенням класів, тригери на створення класів, та ін)

11. Повна інтроспекція.

12. Класові та статичні методи, класові поля.

13. Класи, вкладені у функції та інші класи.

Python підтримує парадигму функціонального програмування, зокрема:

1. Функція є об'єктом.
2. Функції вищих порядків.
3. Рекурсія.
4. Розвинена обробка списків (спискові вирази, операції над послідовностями, ітератори).
5. Аналог замикань (closures).
6. Часткове застосування функції.
7. Можливість реалізації інших засобів на самій мові (наприклад, каррінг).

Модулі та пакети

Програмне забезпечення (застосунок або бібліотека) на Python оформлюється у вигляді модулів, які у свою чергу можуть бути зібрані в пакунки. Модулі можуть розташовуватися як у каталогах, так і в ZIP-архівах. Модулі можуть бути двох типів за своїм походженням: модулі, написані на “чистому” Python, і модулі розширення (extension modules), написані на інших мовах програмування. Наприклад, в стандартній бібліотеці є “чистий” модуль pickle і його аналог на Сі: cPickle. Модуль оформляється у вигляді окремого файлу, а пакет – у вигляді окремого каталогу. Підключення модуля до програми здійснюється оператором import. Після імпорту модуль представлений окремим об’єктом, що дає доступ до простору імен модуля. У ході виконання програми модуль можна перезавантажити функцією reload().

Інтроспекція

Python підтримує повну інтроспекцію часу виконання. Це означає, що для будь-якого об’єкта можна отримати всю інформацію про його внутрішню структуру.

Застосування інтроспекції (метапрограмування) є важливою частиною того, що називають “pythonic style”, і широко застосовується в бібліотеках і фреймворках Python, таких як PyRO, Pyro, PLY, CherryPy, Django та інших, заощаджуючи час програміста, що ними користується.

Стандартна бібліотека

Python поставляється “з батарейками в комплекті”.

Багата стандартна бібліотека є однією з привабливих сторін Python. Тут є засоби для роботи з багатьма мережевими протоколами та форматами Інтернету, наприклад, модулі для написання HTTP-серверів та клієнтів, для розбору та створення поштових повідомлень, для роботи з XML, тощо. Набір модулів для роботи з операційною системою дозволяє писати крос-платформні застосунки. Існують модулі для роботи з регулярними виразами, текстовими кодуваннями, мультимедійними форматами, криптографічними протоколами, архівами, серіалізацією даних, юніт-тестуванням та ін.

Модулі розширення та програмні інтерфейси

Крім стандартної бібліотеки існує багато інших, що надають інтерфейс до всіх системних викликів на різних платформах; зокрема, на платформі Win32 підтримуються всі виклики Win32 API, а також COM в обсязі не меншому, ніж у Visual Basic або Delphi. Існує велика кількість прикладних бібліотек для Python у різноманітних галузях: веб-розробка, бази даних, обробка зображень, обробка тексту, чисельні методи, програми операційної системи тощо.

Для Python прийнята специфікація програмного інтерфейсу до баз даних DB-API 2 та розроблено відповідні цій специфікації пакети для доступу до різних СУБД: PostgreSQL, Oracle, Sybase, Firebird (Interbase), Informix, Microsoft SQL Server, MySQL та sqlite. На платформі Microsoft Windows доступ до БД можливий через ADO (ADOdb). Комерційний пакет mxODBC для доступу до СУБД через ODBC для платформ Windows і UNIX розроблений eGenix. Для Python написано багато ORM: (SQLObject, SQLAlchemy, Dejavu, Django), виконані програмні каркаси для розробки веб-застосунків (Django, Pylons).

Бібліотека NumPy для роботи з багатовимірними масивами дозволяє досягти продуктивності наукових розрахунків, порівнянної зі спеціалізованими пакетами. SciPy використовує NumPy і надає доступ до великого спектра математичних алгоритмів (матрична алгебра – BLAS, level 1-3 і LAPACK; ШПФ).

Бібліотека WSGI – інтерфейс шлюзу з веб-сервером (Python Web Server Gateway Interface).

Python надає простий і зручний програмний інтерфейс C API для написання власних модулів на мовах C та C++. Інструмент SWIG дозволяє майже автоматично отримувати прив'язки для використання C/C++ бібліотек у коді на Python. Можливості цього та інших інструментів варіюються від автоматичної генерації (C/C++/Fortran) – Python інтерфейсів за спеціальними файлами (SWIG, pyste, SIP, pyfort) до надання зручніших API (boost::python, CXX та ін.) Інструмент стандартної бібліотеки ctypes дозволяє програмам Python безпосередньо викликати функції з динамічних бібліотек/DLL, написаних на C. Існують модулі, що дозволяють вбудовувати код на C/C++ прямо у вихідні файли Python, створюючи розширення “на льоту” (pyinline, weave). Для підключення математичних функцій, особливо із застосуванням NumPy, наразі офіційно рекомендованим є Cython.

Інший підхід полягає у вбудовуванні інтерпретатора Python у застосунки. Python легко вбудовується в програми на Java, C/C++, Ocaml. Взаємодія Python-застосунків з іншими системами можлива також за допомогою CORBA, XML-RPC, SOAP, COM.

За допомогою Pyrex можлива компіляція Python-подібної мови (додано можливість типізації) в еквівалентний Сі-код і зв'язування із зовнішніми модулями.

Експериментальний проєкт shed skin передбачає створення компілятора для трансформації неявно типізованих Python програм в оптимізований C++ код. Починаючи з версії 0.22 shed skin дозволяє компілювати окремі функції в модулі розширень. Повна компіляція (станом на 1 липня 2007) далека від завершення.

Python та переважна більшість бібліотек до нього безкоштовні й поставляються у вихідних кодах. Навіть більше, на відміну від багатьох відкритих систем, ліцензія ніяк не обмежує використання Python у комерційних розробках та не накладає ніяких зобов'язань, крім зазначення авторських прав.

Графічні бібліотеки

З Python поставляється бібліотека tkinter на основі Tcl/Tk для створення крос-платформних програм з графічним інтерфейсом.

Для науково-технічної мети найбільшого поширення набуло використання matplotlib – бібліотеки з інтерфейсом, аналогічним MATLAB Plot Tool.

Існують розширення, що дозволяють використовувати всі основні GUI бібліотеки – wxPython, засноване на бібліотеці wxWidgets, PyGTK для GTK+, PyQt та PySide для Qt та інші. Деякі з них також надають широкі можливості для роботи з базами даних, графікою та мережами, використовуючи всі можливості бібліотеки, на якій базуються.

Для створення ігор та програм, що вимагають нестандартного інтерфейсу, можна використовувати бібліотеку Pygame. Вона також надає великі засоби роботи з мультимедіа: з її допомогою можна керувати звуком і зображеннями, відтворювати відео. Надаване pygame апаратне прискорення графіки OpenGL має більш високорівневий інтерфейс в порівнянні з PyOpenGL, що копіює семантику C-бібліотеки для OpenGL. Є також PyOgr, що забезпечує прив'язку до OGRE – високорівневої об'єктно-орієнтованої бібліотеки 3D-графіки. Крім того, існує бібліотека pythonOCC, що забезпечує прив'язку до середовища 3D-моделювання та симуляції OpenCascade.

Для роботи з растровою графікою використовується бібліотека Python Imaging Library.

Порівняння з іншими мовами

Найчастіше Python порівнюють з Perl та Ruby. Ці мови також є інтерпретованими та мають приблизно однакову швидкість виконання програм. Як і Perl, Python може успішно застосовуватися для написання скриптів (сценаріїв). Як і Ruby, Python є добре продуманою системою для ООП.

Засоби функціонального програмування частково запозичені з Scheme та Icon.

У середовищі комерційних застосунків швидкість виконання програм на Python можуть порівнювати з Java-застосунками.

Попри те, що Python має досить самобутній синтаксис, одним із принципів дизайну цієї мови є принцип найменшого подиву.

Недоліки

Низька швидкодія: Python, як і багато інших інтерпретованих мов, які не застосовують, наприклад, JIT-компілятори, мають загальний недолік – порівняно невисоку швидкість виконання програм. Однак, у випадку з Python цей недолік компенсується зменшенням часу розробки програми. У середньому, програма, написана на Python, в 2-4 рази компактніша, ніж її аналог на C++ або Java. Збереження байт-коду (файли .pyc і .pyo) дозволяє інтерпретатору не витрачати зайвий час на перекомпіляцію коду модулів при кожному запуску, на відміну, наприклад, від мови Perl. Крім того, існує спеціальна JIT-бібліотека psyco(проте призводить до збільшення споживання оперативної пам'яті). Ефективність psyco в значній мірі залежить від архітектури програми.

Існують проєкти реалізацій мови Python, що вводять високопродуктивні віртуальні машини (ВМ) як компілятора заднього плану. Прикладами таких реалізацій може служити PyPy, що базується на LLVM; більш ранньою ініціативою є проєкт Parrot. Очікується, що використання ВМ типу LLVM призведе до тих самих результатів, що й використання аналогічних підходів для реалізацій мови Java, де низька обчислювальна продуктивність в основному подолана.

Безліч програм/бібліотек для інтеграції з іншими мовами програмування (див. вище) надають можливість використовувати іншу мову для написання критичних ділянок.

У найпопулярнішій реалізації мови Python інтерпретатор досить великий і більш вимогливий до ресурсів, ніж в аналогічних популярних реалізаціях Tcl, Forth, LISP або Lua, що обмежує його застосування у вбудованих системах. Тим

не менше, Python знайшов застосування в КПК і деяких моделях мобільних телефонів.

Відсутність статичної типізації: Відсутність статичної типізації є не стільки вадою інтерпретатора, скільки вибором розробника мови. Справа в тому, що в Python прийнята так звана “качина типізація”. У силу цього типи переданих значень недоступні на етапі компіляції, та помилки на зразок AttributeError можуть виникати під час виконання. Відсутність статичної типізації також є однією з основних причин низької швидкодії.

Існують модулі, які дозволяють контролювати типи параметрів функцій на етапі виконання, наприклад typecheck або method signature checking decorators. Додавання необов'язковою статичної типізації параметрів функції заплановано для Python3000. При цьому, однак, безпосередньо інтерпретатор не буде перевіряти типи, а тільки додавати відповідну інформацію до метаданих функції для її (інформації) подальшого використання модулями розширень.

Відсутність статичної типізації і деякі інші причини не дозволяють реалізувати в Python механізм перевантаження функцій на етапі компіляції. Можливості Python дозволяють реалізувати динамічну перевантаження на етапі виконання, що, звичайно, уповільнює виклик, так як вирішення яку саме функцію викликати проводиться при кожному зверненні і є, в загальному випадку, досить складною процедурою. Відсутність перевантаження в Python намагаються компенсувати використанням віртуальних функцій.

len = lambda x : x.__len__() # це лише приклад

Реалізації та опис, реалізації простого перевантаження також є в прикладах програм на Python .

Плани з підтримки перевантаження в Python3: Перевантаження функцій реалізована різними сторонніми бібліотеками, в тому числі PEAK надає надзвичайно багатий можливостями механізм перевантаження функцій з використанням довільних правил.

Неможливість модифікації вбудованих класів: У порівнянні з Ruby та деякими іншими мовами, в Python відсутня можливість модифікувати вбудовані

класи, такі, як int, str, float, list та інші, що, однак, дозволяє Python споживати менше оперативної пам’яті і швидше працювати. Ще однією причиною введення такого обмеження є необхідність узгодження з модулями розширення. Багато модулів (з метою оптимізації швидкодії) перетворять Python-об’єкти елементарних типів до відповідних Сі-типів замість маніпуляцій з ними за допомогою Сі-API.

Глобальне блокування інтерпретатора (GIL): GIL (Global Interpreter Lock) – проблема, притаманна CPython, Stackless та PyPy, але відсутня в Jython та IronPython. При своїй роботі основний інтерпретатор Python постійно використовує велику кількість нитко-небезпечних даних. В основному це словники, в яких зберігаються атрибути об’єктів. Для уникнення руйнування цих даних при спільній модифікації з різних потоків перед початком виконання декількох інструкцій (за замовчуванням 100) потік інтерпретатора захоплює GIL, а після закінчення звільняє. Внаслідок цієї особливості в кожен момент часу може виконуватися тільки одна нить Python коду, навіть якщо на комп’ютері є кілька процесорів або процесорних ядер (GIL також звільняється на час виконання блокуючих операцій, таких як введення-виведення, зміни/перевірка стану синхронізуючих примітивів та інших – таким чином, якщо одна нить блокується, інші можуть виконуватися). Була зроблена спроба переходу до більш гранульованої синхронізації, проте через часті захоплення/звільнення блокувань ця реалізація виявилася занадто повільною. У найближчому майбутньому перехід від GIL до інших технік не передбачається, однак є python-safethread – CPython без GIL і з деякими іншими змінами (за твердженнями його авторів, на однонитевих застосунках швидкість відповідає 60-65% від швидкості оригінальному CPython).

Ця проблема має два основних варіанти вирішення. Перший – відмова від спільного використання змінюваних даних. При цьому дані дублюються в нитях і необхідність забезпечення їхньої синхронізації (якщо така потрібна) лягає на програміста. Цей підхід веде до збільшення споживання оперативної пам'яті (однак не настільки сильно, як при використанні процесів).

Другий підхід – забезпечення більш гранульованої синхронізації – для окремих структур даних. У цьому випадку падає продуктивність внаслідок збільшення числа звільнень/захоплень блокувань.

Якщо необхідно паралельне виконання декількох нитей Python-коду, то можна скористатися процесами, наприклад, модулем processing, який імітує семантику стандартного модуля threading, але використовує процеси замість нитей. Є безліч модулів, що спрощують написання паралельних та/або розподілених застосунків на Python, таких як parallelpython, Pypar, pympi та інші. GIL звільняється при виконанні коду більшості розширень, наприклад, NumPy/SciPy, дозволяючи на час розрахунків виконуватися іншому Python-нитi. Іншим рішенням може бути використання IronPython або Jython, позбавлених даного недоліку.

Реалізації: Python портований на всі відомі платформи – від КПК до мейнфреймів. Існують порти під Windows, всі варіанти UNIX (включно з Linux), Plan 9, Mac OS і Mac OS X, Palm OS, OS/2, Amiga, AS/400 і навіть OS/390 і Symbian.

При цьому, на відміну від багатьох портованих систем, на кожній платформі Python підтримує характерні для даної платформи технології (наприклад, Microsoft COM). Крім того, існує спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретатору виконуватися на будь-якій системі, що підтримує Java, класи Java можуть безпосередньо використовуватися з Python і навіть бути написаними на Python. Нещодавно почалася розробка системи, спрямованої на повнішу інтеграцію з платформою .NET – Iron Python.

Подальша розробка: Python Enhancement Proposal (“PEP”) – це документ зі стандартизованим дизайном, що надає загальну інформацію про мову Python, включаючи нові пропозиції, описи та роз'яснення можливостей мови. PEP пропонуються як основне джерело для пропозиції нових можливостей і для роз'яснення вибору того або іншого дизайну для основних елементів мови. Видатні PEP рецензуються і коментуються BDFL.

Графік і сумісність: Серії Python 2.x і Python 3.x протягом кількох випусків будуть існувати паралельно, при цьому серія 2.x буде використовуватися для забезпечення сумісності та швидше за все в неї будуть включені деякі можливості серії 3.x. PEP 3000 містить більше інформації про плановані випуски.

Python 3.0 обернено не сумісний з попередньою серією 2.x. Код Python 2.x швидше за все буде видавати помилки при виконанні в Python 3.0. Динамічна типізація Python, разом з планами зміни декількох методів словників, робить механічний переклад з Python 2.x в Python 3.0 дуже складним. Однак, утиліта «2to3» вже здатна зробити більшість роботи з перекладу коду, вказуючи на підозрілі їй частини за допомогою коментарів і попереджень. PEP 3000 рекомендує тримати вихідний код для серії 2.x, і робити випуски для Python 3.x за допомогою “2to3”. Отриманий код не слід редагувати, поки програма повинна бути працездатною в Python 2.x.

Нещодавно розробники оголосили про офіційне припинення розвитку гілки Python 2.x. Остання випущена версія Python 2.7. Далі розробка буде вестися лише у гілці Python 3.x.

Можливості: Синтаксична можливість для анотації параметрів і результату функцій (наприклад, для передачі інформації про тип або документування).

Повний перехід на unicode для рядків.

Введення нового типу “незмінні байти” і типу “змінюваний буфер”. Обидва необхідні для подання двійкових даних.

Нова підсистема вводу-виводу (модуль io), що має окремі вигляди для бінарних і текстових даних.

Абстрактні класи, абстрактні методи (є вже в 2.6).

Ієрархія типів для чисел.

Вирази для словників і множин {k: v for k, v in a_dict} і {el1, el2, el3} (за аналогією зі списковими виразами).

Зміни print з вбудованого виразу у вбудовану функцію. Це дозволить модулям робити зміни, підлаштовуючись під різне використання функції, а також

спростить код. У Python 2.6 ця можливість активується введенням from __future__ import print_function.

Переміщення reduce (але не map або filter) з вбудованого простору в модуль functools (використання reduce істотно менш читабельне в порівнянні з циклом).
Видалення деяких застарілих можливостей, які підтримуються у гілці 2.x для сумісності, зокрема: класи старого стилю, цілочисельний поділ з обрізанням результату як поведінка за вмовчанням, рядкові винятки, неявний відносний імпорт, оператор exec тощо.

Реорганізація стандартної бібліотеки: Новий синтаксис для метакласів.
Змінений синтаксис присвоєння. Стало можливим, наприклад, надання (a, * rest, b) = range(5). З іншого боку, формальні параметри функцій на зразок def foo (a, (b, c)) більше неприпустимі.
Основні пропозиції: Перелік всіх пропозицій наведено на офіційному сайті разом із їх статусом. Нижче наведено ті, які здобули найбільше поширення:

PEP8 Настанова щодо стилю оформлення коду.

Спеціалізовані підмножини/розширення Python

На основі Python було створено кілька спеціалізованих підмножин мови, в основному призначених для статичної компіляції в машинний код. Деякі з них:

RPython – створена в рамках проекту PyPy значно обмежена реалізація Python без динамізму часу виконання та деяких інших можливостей. RPython код можна компілювати в безліч інших мов/платформ – C, JavaScript, Lisp, .NET, LLVM. На RPython написаний інтерпретатор PyPy.

Pyrex – обмежена реалізація Python, але трохи менше, ніж RPython. PyReX розширено можливостями статичної типізації типами з мови C і дозволяє вільно змішувати типізований та не типізований код. Призначений для написання модулів розширень, компілюється в код на мові C.

Cython – розширена версія Pyrex.

Pyastra – компілятор Python коду в асемблер для PIC архітектури.

Shed Skin – призначений для компіляції неявно статично типізованого Python коду в оптимізований код на мові C++, проєкт далекий від завершення.

Застосування

Python – стабільна та поширена мова. Вона використовується в багатьох проєктах та в різних якостях: як основна мова програмування або для створення розширень та інтеграції додатків. На Python реалізована велика кількість проєктів, також вона активно використовується для створення прототипів майбутніх програм.

Python використовується в багатьох великих компаніях.

Мова програмування Python останнім часом все частіше використовується для аналізу даних, як в науці, так і комерційній сфері. Цьому сприяє простота мови, а також велика різноманітність відкритих бібліотек.

Тож даний проєкт написаний на мові Python 3.6, що є оптимальним варіантом серед інших мов.

Jupyter Notebook

Jupyter Notebook – це командна оболонка для інтерактивних обчислень (рис.2.2). Цей інструмент може використовуватися не лише на Python, але й на інших мовах програмування: Julia, R, Haskell і Ruby. Він часто використовується для роботи з даними, статистичним моделюванням і машинним навчанням.

Рисунок 2.2 – Логотип Jupyter Notebook

Цей інструмент допоможе створювати файли (notebooks), які містять не лише комп’ютерний код, але й інші елементи (замітки, рівняння, діаграми, посилання і так далі), якими можна потім поділитися із замовниками або друзями.

Вимоги

Знадобиться середовище програмування для Python 3, встановлене або на локальній машині, або на сервері Ubuntu 16.04 чи новітніші версії.

Jupyter Notebook – неймовірно потужний інструмент для інтерактивної розробки та подання проектів в області наук про дані.

Блокнот об'єднує код і його висновок в єдиний документ, який об'єднує візуалізацію, розповідний текст, математичні рівняння та інші мультимедійні. Цей інтуїтивно зрозумілий робочий процес сприяє итеративній і швидкій розробці, що робить ноутбуки все більш популярним вибором для подання в даних і їх аналізу.

Найкраще те, що в рамках проєкту з відкритим вихідним кодом Project Jupyter він повністю безкоштовний.

Проєкт Jupyter є наступником більш раннього проєкту IPython Notebook, який вперше був опублікований в якості прототипу в 2010 році. Хоча в Jupyter Notebooks можна використовувати з багатьма різними мовами програмування, в дипломному проєкті основну увагу буде приділено Python, оскільки він є найбільш поширений варіантом використання.

Щоб отримати максимальну віддачу від цього уроку, потрібно бути знайомим з програмуванням, особливо з Python і pandas. Проте, якщо є досвід роботи з іншою мовою, Python не буде занадто складним.

Для новачка найпростіше почати роботу з Jupyter Notebooks, встановивши дистрибутив Anaconda. Anaconda є найбільш широко використовуваним дистрибутивом Python для роботи з даними і поставляється з попередньо встановленими найбільш популярними бібліотеками та інструментами. Деякі з найбільших бібліотек Python, включених в Anaconda, включають NumPy, pandas і Matplotlib, хоча можна поглянути на повний список з більш ніж 1000+ пакетів. Це дозволить приступити до роботи, без клопоту управління незліченними установками або занепокоєння про залежності і проблемах установки, пов'язаних з ОС.

kernel (Ядро) – це "обчислювальний движок", який виконує код, що міститься в документі ноутбука.

cell (Осередок) – це контейнер для тексту, який буде відображатися в записнику, або код, який буде виконуватися ядром записної книжки.

Осередки утворюють структуру ноутбука.

Є два основних типи осередків:

1. Осередок коду містить код, який повинен бути виконаний в ядрі, і відображає його висновок нижче.

2. Осередок Markdown містить текст, відформатований з використанням Markdown, і відображає його висновок на місці під час запуску.

Ядра (Kernels)

За кожним ноутбуком працює ядро. Коли запускається осередок коду, цей код виконується в ядрі, і будь-який висновок повертається назад в осередок для відображення. Стан ядра зберігається в часі і між осередками – воно відноситься до документа в цілому, а не до окремих осередків.

Наприклад, якщо імпортувати бібліотеки або оголосити змінні в одній комірці, вони будуть доступні в інший. Таким чином, можна думати про документ блокнота як про щось порівнянному з файлом сценарію, за винятком того, що він є мультимедійним.

Існують не тільки ядра для різних версій Python, але і більш 100 мов, включаючи Java, C і навіть Fortran. Дослідники даних можуть бути особливо зацікавлені в ядрах для R і Julia, а також в imatlab і ядрі Calysto MATLAB Kernel для Matlab. Ядро SoS забезпечує багатомовну підтримку в межах одного ноутбука. Кожне ядро має свої власні інструкції по установці, але, ймовірно, зажадає виконання деяких команд на комп'ютері.

2.2 Бібліотеки аналізу даних

Pandas

Pandas – програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу (рис.2.3). Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та

часовими рядами. pandas є вільним програмним забезпеченням, що випускається за трипунктовою ліцензією BSD. Ця назва походить від терміну “панельні дані” (англ. panel data), який в економетрії позначає багатовимірні структуровані набори даних.

Рисунок 2.3 – Логотип Pandas

Можливості бібліотеки:

1. Об’єкт DataFrame із вбудованим індексуванням для маніпулювання даними.

2. Інструменти для зчитування та записування даних між структурами даних у пам'яті та різними форматами файлів.

3. Вирівнювання даних та вбудована підтримка пропущених даних.

4. Переформатовування для отримання зведених наборів даних.

5. Отримання зрізів за мітками, індексування з розширеними можливостями та отримання піднаборів з великих наборів даних.

6. Вставляння та вилучення стовпчиків у структурах даних.

7. Рушій групування, що дозволяє робити з наборами даних операції розділення-зміни-об’єднання (англ. split-apply-combine).

8. Злиття та з’єднання наборів даних.

9. Ієрархічне індексування осей для роботи з даними високої вимірності в структурі даних нижчої вимірності.

10. Функціональність для часових рядів: породження діапазонів дат та перетворення частоти, статистики рухливого вікна, лінійні регресії рухливого вікна, зсування дат та запізнювання.

Цю бібліотеку сильно оптимізовано за продуктивністю, критичні ланцюжки коду написано мовами Cython та C.

Історія

Розробник Вес Маккінні почав працювати над pandas 2008 року, коли був у AQR Capital Management, через потребу у високопродуктивному, гнучкому інструменті для здійснення кількісного аналізу фінансових даних. Перед тим, як покинути AQR, він зміг переконати керівництво дозволити йому віддати цю бібліотеку у відкритий доступ.

Інший співробітник AQR, Чан Ше (англ. Chang She), приєднався до цих зусиль 2012 року як другий з основних за внеском до цієї бібліотеки.

DataFrame і Series: Щоб ефективно працювати з pandas, необхідно освоїти найголовніші структури даних бібліотеки: DataFrame і Series. Без розуміння що вони з себе представляють, неможливо в подальшому проводити якісний аналіз.

Series: Структура / об'єкт Series вдає із себе об'єкт, схожий на одновимірний масив (пітонівскій список, наприклад), але відмітною його рисою є наявність асоційованих міток, т.зв. індексів, уздовж кожного елемента зі списку. Така особливість перетворює його в асоціативний масив або словник в Python.

У строковому поданні об'єкта Series, індекс знаходиться зліва, а сам елемент праворуч. Якщо індекс явно не заданий, то pandas автоматично створює RangeIndex від 0 до N-1, де N загальна кількість елементів. Також варто звернути, що у Series є тип збережених елементів, в цьому випадку це int64, тому що передаються цілочисельні значення.

У об'єкта Series є атрибути через які можна отримати список елементів і індекси, це values і index відповідно.

DataFrame

Об'єкт DataFrame найкраще уявляти собі у вигляді звичайної таблиці і це правильно, адже DataFrame є табличній структурою даних. У будь-якій таблиці завжди присутні рядки і стовпці. Стовпцями в об'єкті DataFrame виступають об'єкти Series, рядки яких є їхніми безпосередніми елементами.

Угруповання і агрегування в pandas: Угруповання даних один з найбільш часто використовуваних методів при аналізі даних. У pandas за угруповання відповідає метод groupby().

Зведені таблиці в pandas: Термін “зведена таблиця” добре відомий тим, хто не з чуток знайомий з інструментом Microsoft Excel або будь-яким іншим, призначеним для обробки і аналізу даних. У pandas зведені таблиці будуються через метод pivot_table().

NumPy

Numpy – розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами. Попередник Numpy, Numeric, був спочатку створений Jim Hugunin. Numpy – відкрите програмне забезпечення і має багато розробників.

Оскільки Python – інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм, який може бути виражений в основному як послідовність операцій над масивами і матрицями, працює так само швидко, як еквівалентний код, написаний на C.

NumPy можна розглядати як гарну вільну альтернативу MATLAB, оскільки мова програмування MATLAB зовні нагадує NumPy: обидві вони інтерпретовані, і обидві дозволяють користувачам писати швидкі програми поки більшість операцій проводяться над масивами або матрицями, а не над скалярами. Перевага MATLAB у великій кількості доступних додаткових тулбоксів, включаючи такі як пакет Simulink. Основні пакети, що доповнюють NumPy, це: SciPy – бібліотека, що додає більше MATLAB-подібної функціональності; Matplotlib – пакет для створення графіки в стилі MATLAB. Внутрішньо як MATLAB, так і NumPy базується на бібліотеці LAPACK, призначеної для вирішення основних задач лінійної алгебри.

У NumPy доступно багато об’єктів, найважливіші з яких: Одновимірні масиви.

N-вимірний масив, також відомий як ndarray. Усі елементи ndarray повинні бути одного типу, на зразок цілих чисел.

Ви можете уявляти одновимірний масив як колонку чи рядок таблиці, що містить один чи більше елементів.

Багатовимірні масиви: У багатовимірних масивах колонок більше за одну.

Уявіть таблиці Excel – там є колонки та рядки. Кожна колонка може розглядатися як вимір.

Існує декілька різних способів створити масив.

Агрегатори – це методи NumPy дозволяють замінювати дані інтегральними характеристиками вздовж деяких осей. Наприклад, можна порахувати середнє значення, максимальне, мінімальне, варіацію або ще якусь характеристику уздовж будь-якої осі або осей і сформувати з цих даних новий масив. Форма нового масиву буде містити всі осі вихідного масиву, крім тих, уздовж яких підраховується агрегатор.

Scikit-learn

Бібліотека Scikit-learn – найпоширеніший вибір для вирішення завдань класичного машинного навчання. Вона надає широкий вибір алгоритмів навчання з учителем і без вчителя. Навчання з учителем передбачає наявність розміченого датасета, в якому відомо значення цільового показника. У той час як навчання без вчителі не передбачає наявності розмітки в датасета – потрібно навчитися отримувати корисну інформацію з довільних даних. Одне з основних переваг бібліотеки полягає в тому, що вона працює на основі декількох поширених математичних бібліотек, і легко інтегрує їх один з одним. Ще однією перевагою є широка спільнота і докладна документація. Scikit-learn широко використовується для промислових систем, в яких застосовуються алгоритми класичного машинного навчання, для досліджень, а так само для новачків, які тільки робить перші кроки в області машинного навчання.

Для своєї роботи, scikit-learn використовує такі популярні бібліотеки на рис. 2.4:

1. NumPy: математичні операціїї і операції над тензорами.
2. SciPy: науково-технічні обчислення.
3. atplotlib: візуалізація даних.
4. IPython: інтерактивна консоль для Python.
5. SymPy: символьна математика.
6. Pandas: обробка, маніпуляції і аналіз даних.

Рисунок 2.4 – Логотип Scikit-learn

До завдань бібліотеки не входить завантаження, обробка, маніпуляція даними і їх візуалізація. З цими завданнями відмінно справляються бібліотеки Pandas і NumPy. Scikit-learn спеціалізується на алгоритмах машинного навчання для вирішення завдань навчання з учителем: класифікації (прогноз ознаки, безліч допустимих значень якого обмежена) і регресії (прогноз ознаки з речовими значеннями), а також для задач навчання без учителя: кластеризації (розбиття даних по класах, які модель визначить сама), зниження розмірності (подання даних в просторі меншої розмірності з мінімальними втратами корисної інформації) і детектування аномалій.

Бібліотека реалізує наступні основні методи:

Лінійні: моделі, завдання яких побудувати розділяє (для класифікації) або аппроксиміруючу (для регресії) гіперплоскість.

Метричні: моделі, які обчислюють відстань по одній з метрик між об’єктами вибірки, і приймають рішення в залежності від цього відстані (K найближчих сусідів).

Дерева рішень: навчання моделей, що базуються на безлічі умов, оптимально обраних для вирішення завдання.

Ансамблеві методи: методи, засновані на деревах рішень, які комбінують міць безлічі дерев, і таким чином підвищують їх якість роботи, а також дозволяють проводити відбір ознак (бустінг, беггінг, випадковий ліс, мажоритарну голосування).

Нейронні мережі: комплексний нелінійний метод для задач регресії і класифікації.

SVM: нелінійний метод, який навчається визначати межі прийняття рішень.

Наївний Байес: пряме розподіл усіх моделювання для задач класифікації.

PCA: лінійний метод зниження розмірності і відбору ознак

t-SNE: нелінійний метод зниження розмірності.

K-середніх: найпоширеніший метод для кластеризації, потрібних на вхід число кластерів, за якими повинні бути розподілені дані.

Крос-валідація: метод, при якому для навчання використовується весь датасета (на відміну від розбиття на вибірки train / test), проте навчання відбувається багаторазово, і в якості валідаційної вибірки на кожному кроці виступають різні частини датасета. Підсумковий результат уявляють собою усереднення отриманих результатів.

Grid Search: метод для знаходження оптимальних гіперпараметров моделі шляхом побудови сітки з значень гіперпараметров і послідовного навчання моделей з усіма можливими комбінаціями гіперпараметров з сітки.

Це лише базовий список. Крім цього, Scikit-learn містить функції для розрахунку значень метрик, вибору моделей, препроцессінга даних та інші.

Scikit-learn (раніше scikits.learn і також відомий як sklearn) – це вільна бібліотека машинного навчання для мови програмування Python. Він містить різні алгоритми класифікації, регресії та кластеризації, включаючи підтримуючі

векторні машини, випадкові ліси, збільшення градієнта, k-засоби та DBSCAN, і призначений для взаємодії з числовими та науковими бібліотеками Python NumPy та SciPy.

Проєкт scikit-learn розпочався як scikits.learn, проєкт Девід Курно. Його назва походить від поняття, що це “SciKit” (SciPy Toolkit), окремо розроблене та розповсюджене сторонне розширення для SciPy. Оригінальну кодову базу згодом переписали інші розробники. У 2010 році Фабіан Педрегоса, Гаель Варуко, Олександр Грамфорт та Вінсент Мішель, усі з Французького інституту досліджень комп’ютерних наук та автоматики в Рокенкорті, Франція, взяли керівництво проектом і зробили перший публічний реліз 1 лютого 2010 року. У листопаді 2012 року серед різних науковців науковці, як і наукові образи, були описані як “добре підтримувані та популярні”. Scikit-learn – одна з найпопулярніших бібліотек машинного навчання на GitHub.

Scikit-learn багато в чому написаний на Python, і широко використовує numpy для високоефективних лінійних алгебр та операцій з масивом. Крім того, деякі основні алгоритми написані в Cython для підвищення продуктивності. Векторні машини підтримки реалізовані обгорткою Cython навколо LIBSVM; логістична регресія та лінійні верстати підтримки машин аналогічною обгорткою навколо LIBLINEAR. У таких випадках розширення цих методів за допомогою Python може бути неможливим.

Scikit-learn добре інтегрується з багатьма іншими бібліотеками Python, такими як matplotlib і plotly для побудови графіків, numpy для векторної масиви масивів, фрейми даних панди, sciру та багато іншого.

Scikit-learn спочатку був розроблений Девідом Курнопо як літній проект коду Google у 2007 році. Пізніше Маттьє Брюхер приєднався до проєкту і почав використовувати його як частину своєї дипломної роботи. У 2010 році INRIA, Французький інститут досліджень комп’ютерних наук та автоматики, взяв участь, і перший публічний реліз (v0.1 beta) був опублікований наприкінці січня 2010 року.

1. Травень 2019. scikit-learn 0,21,0

2. Вересень 2018. scikit-learn 0.20.0

3. Липень 2017. scikit-learn 0.19.0

4. Вересень 2016. scikit-learn 0.18.0

5. Листопад 2015. scikit-learn 0.17.0

6. Березень 2015 р. Scikit-learn 0.16.0

7. Липень 2014 р. Scikit-learn 0,15,0

8. Серпень 2013. scikit-learn 0,14

Оптимізація гіперпараметрів – задача машинного навчання по вибору множини оптимальних гіперпараметрів для алгоритму машинного навчання. Гіперпараметр є параметром, значення якого використовується для керування процесом навчання. На відміну від значень інших параметрів (наприклад, вагових коефіцієнтів), які потрібно вивчити.

Одні й ті ж види моделей машинного навчання можуть мати різні обмеження, ваги або потребувати певної швидкості навчання для різних видів даних. Ці параметри називаються гіперпараметрами і їх слід підбирати так, щоб модель могла оптимально вирішити завдання навчання. Для цього знаходиться кортеж гіперпараметрів, який дає оптимальну модель, що оптимізує задану функцію втрат на заданих незалежних даних. Цільова функція бере кортеж гіперпараметрів і повертає пов'язані з ними втрати. Часто використовується перехресна перевірка для оцінки цієї узагальнюючої здатності.

Matplotlib

Matplotlib – бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується). Отримувані зображення можуть бути використані як ілюстрації в публікаціях.

matplotlib написана і підтримується в основному Джоном Хантером і поширюється на умовах BSD-подібної ліцензії. Зображення, які генеруються в різних форматах, можуть бути використані в інтерактивній графіці, наукових публікаціях, графічному інтерфейсі користувача, веб-додатках, де потрібно будувати діаграми (англ. plotting). В документації автор зізнається, що Matplotlib

починався з імітування графічних команд MATLAB (рис.2.5), але є незалежним від нього проєктом.

Рисунок 2.5 – Логотип matplotlib

Версія 1.2.0 – остання стабільна – потребує Python версії від 2.6 і вище і версію NumPy від 1.4 і вище.

Бібліотека Matplotlib побудована на принципах ООП, але має процедурний інтерфейс pylab, який надає аналоги команд MATLAB.

Matplotlib є гнучким, легко конфігурованим пакетом, який разом з NumPy, SciPy і IPython надає можливості, подібні до MATLAB. В даний час пакет працює з декількома графічними бібліотеками, включаючи wxWindows і PyGTK.

Пакет підтримує багато видів графіків і діаграм:

1. Графіки (line plot).
2. Діаграми розсіювання (scatter plot).
3. Стовпчасті діаграми (bar chart) і гістограми (histogram).
4. Секторні діаграми (pie chart).
5. Діаграми “Стовбур-листя” (stem plot).
6. Контурні графіки (contour plot).
7. Поля градієнтів (quiver).
8. Спектральні діаграми (spectrogram).

Користувач може вказати осі координат, сітку, додати підписи і пояснення, використовувати логарифмічну шкалу або полярні координати.

Нескладні тривимірні графіки можна будувати з допомогою набору інструментів (toolkit) mplot3d. Існують і інші набори інструментів: для картографії, для роботи з Excel, утиліти для GTK та інші.

З допомогою Matplotlib можна створювати і анімовані зображення.

Інтерфейс pylab дозволяє легко використовувати matplotlib досвідченими користувачами MATLAB.

Нижче наведені деякі переваги використання matplotlib, як аналогу MATLAB:

1. Вбудована підтримка SVG.

2. Є відкритим програмним забезпеченням.

3. Безкоштовний

Plotly

Plotly – надзвичайно потужний інструмент, але налаштування та створення графіків забирають багато часу й не є інтуїтивно зрозумілими.

Ось найпомітніші мінуси на початку роботи з Plotly:

1. Для нього потрібний ключ API і реєстрація, а не просто встановлення пакета.

2. Він створює об’єкти даних/макетів, які є унікальними для Plotly, але не інтуїтивними.

3. Розмітка діаграми не спрацювала (40 рядків коду – буквально ні для чого).

Проте за всіх його мінусів існують плюси та обхідні шляхи:

1. Можна редагувати графіки на сайті Plotly, а також у середовищі Python.

2. Добре підтримуються інтерактивні графіки/панелі.

3. Plotly співпрацює з Mapbox, даючи змогу налаштовувати карти.

4. Існує чудовий загальний потенціал для неперевершених графіків.

З боку було б несправедливо просто висміювати свої колізії з Plotly, не показуючи жодного коду й результату, на відміну від того, що зробили люди з більшими здібностями до цього пакета.

Folium

Folium – це бібліотека, що будує оброблені дані в Python на сторінки відображення бібліотеки leaflet.js. Маніпулює своїми даними в Python, а потім візуалізує їх на карті Leaflet через folium.

Концепції

folium дозволяє легко візуалізувати дані, якими маніпулювали в Python на інтерактивній карті. Це дає змогу прив'язувати дані до карти для візуалізації виборів, а також передавати візуалізацію на вектор / растр / HTML як маркери на карті.

Бібліотека має ряд вбудованих наборів даних з OpenStreetMap, Mapbox та Stamen, а також підтримує користувальницькі тилетки з Mapbox або Cloudmade API. folium підтримує накладання зображень, відео, GeoJSON і TopoJSON.

Seaborn

Seaborn – бібліотека візуалізації Python, заснована на matplotlib. Він забезпечує інтерфейс високого рівня для малювання привабливої статистичної графіки.

Онлайн-документація доступна на сайті seaborn.pydata.org.

Документи включають підручник, приклад галереї, посилання на API та іншу корисну інформацію.

Seaborn підтримує Python 3.6+ і більше і більше не підтримує Python 2.

Для встановлення потрібні numpy, scipy, панди та matplotlib. Деякі функції необов'язково використовуватимуть статистичні моделі, якщо вони встановлені.

Datetime

Бібліотека datetime використовується для роботи в Python з часом і датами, дозволяючи представляти дану інформацію в найбільш зручній формі.

Вона складається з декількох класів. Завдяки їх наявності, програміст отримує доступ до багатьох корисних методів:

1. Отримання поточних системних дати і часу.

2. Обчислення різниці між датами та інші арифметичні операції.

3. Операціями, які дозволяють порівнювати час.

4. Форматований вивід інформації про дату і час.

У модулі використовуються константи MINYEAR і MAXYEAR, які дорівнюють 1 і 9999 відповідно. Це мінімальне і максимально можливе значення року, використовувані в бібліотеці.

Модуль datetime включає в себе кілька різних класів, кожен з яких володіє власними методами і властивостями, а також служить для певних цілей. Всі вони представлені в наступній таблиці, де містяться їх назви та коротка характеристика.

date є датою, повністю виключаючи дані про час, на основі Григоріанського календаря

time включає дані про час, повністю ігноруючи відомості про дату

datetime містить інформацію про час і дату, грунтуючись на даних з Григоріанського календаря

timedelta описує певний період у часі, який знаходиться між двома різними моментами

tzinfo представляє різні відомості про часовий пояс

timezone описує час, керуючись стандартом UTC

Далі буде розглянуто як за допомогою цих класів можна в Python працювати з датою і часом. Розберемо основні приклади використання бібліотеки datetime.

Date: Клас date використовується для представлення даних про дату, які включають рік, місяць і число. Щоб мати можливість застосовувати цей клас, необхідно попередньо імпортувати модуль datetime, помістивши відповідну інструкцію в початок файлу з програмним кодом. Для створення об'єктів типу date слід зробити виклик однойменного конструктора, вказавши йому в якості параметрів відомості про дату. При цьому не можна забувати про порядок, в якому знаходяться аргументи: рік, потім місяць і число.

Наступний приклад демонструє створення об'єкта date, де змінна під назвою a отримує три зазначених вище значення. Після цього відомості про дату виводяться на екран за допомогою методу print, а функція type дозволяє отримати ім'я класу для цього об’єкта.

Time: Клас time служить для демонстрації даних про час, повністю ігноруючи дату. Як і у випадку з попереднім класом date, слід імпортувати модуль datetime за допомогою ключового слова import. Створити об'єкт, що належить до

типу time можна за допомогою конструктора, який приймає такі аргументи як кількість годин, хвилин, секунд і мікросекунд. Вказуючи дані параметри, не варто забувати про їх необхідному порядку, розставляючи числа на правильних позиціях. Також потрібно пам’ятати, що існують обмеження, наприклад, не можна відображати час із значенням понад 59 на місці секунд.

У наступному фрагменті коду створюється змінна a, якій присвоюється об’єкт типу time з чотирма різними властивостями: число годин, хвилин, секунд і мікросекунд. Завдяки методу print виводиться інформація про цей об’єкт, а функція type відображає його клас.

Datetime*:* Клас datetime дозволяє описувати дані про певний моменті в часі, який враховує не тільки годинник і хвилини, але і відомості про дату. Як і у випадку з попередніми типами об’єктів, тут використовується конструктор з декількома аргументами під кожне значення. У наступному прикладі демонструється програма, де змінна під ім’ям a отримує об'єкт зі стандартного конструктора datetime, після виводить своє значення і тип.

Існує певний формат, в якому дата і час виводяться за замовчуванням, але дане подання не завжди задовольняє запити користувача.

Щоб в Python перетворити дату і час в рядок потрібного формату, слід скористатися методом strftime, вказавши йому в якості аргументу параметри форматування. Як можна зрозуміти з прикладу, який знаходиться нижче, за наступне поширення інформації про кількість днів або хвилин відповідають спеціальні зарезервовані символи, що йдуть слідом за знаком відсотка.

Варто зауважити, що тут створюються два абсолютно ідентичних об'єкта за допомогою методу today. Однак, завдяки роботі функції strftime, кожен з них отримує свою форму для виведення на екран (день, місяць і рік для змінної a, година, хвилина, секунда для змінної b). Після цього метод print відображає відомості в заданому раніше форматі.

Операції: Користуючись класом datetime, можна легко знаходити різницю між двома різними датами. Наступним приклад демонструє створення двох об'єктів. Мінлива a є дату, яку передає їй метод now, a b задається за допомогою

конструктора вручну. Виконавши просту операцію знаходження різниці між об'єктами a і b, можна отримати третій об’єкт c. Застосування функції print дозволяє безпосередньо вивести на екран його властивості за рахунок оператора точки.

Timedelta: Клас timedelta призначений для зручного виконання різних маніпуляцій над датами і часом. Можна створити об'єкт даного класу, скориставшись конструктором. Аргументами є дні, години, хвилини, секунди, мікросекунди і тижні, задати які можна за допомогою прямого звернення до імен властивостей, як це показано в наступному прикладі. Як і раніше, метод print дозволяє виводити відомості про дату на екран, а функція type відображає тип створеного об’єкта.

Tzinfo і timezone: Класи tzinfo і timezone застосовуються для роботи з інформацією, яка містить відомості про часові пояси. Створити об’єкт, що належить типу tzinfo неможливо, оскільки цей клас є абстрактним. Однак можна скористатися спадкуванням, створивши власний клас на основі tzinfo. При цьому слід пам’ятати, що для роботи з такими об'єктами доведеться реалізувати кілька абстрактних методів, до числа яких відносяться utcoffset (зміщення за місцевим часом з UTC), dst (настройка переходу на літній час), а також функція tzname (ім’я часового поясу у вигляді рядки).

Таким чином, користуючись класами з бібліотеки datetime, можна підвищити ефективність роботи програм, які взаємодіють з часом і датою. Можливості вбудованих методів дозволяють не тільки відображати потрібні відомості в зручному форматі, але також по-різному маніпулювати ними, наприклад, додаючи або віднімаючи години, хвилини і секунди.

OS

Модуль os зі стандартної бібліотеки мови програмування Python зазвичай використовується для роботи зі встановленою ОС, а також файлової системою ПК. Він містить масу корисних методів для взаємодії з файлами і папками на жорсткому диску. Програми, що працюють з модулем os, що не залежать від типу ОС і є легко переносяться на іншу платформу.

Модуль os в Python – це бібліотека функцій для роботи з операційною системою. Методи, включені в неї дозволяють визначати тип операційної системи, отримувати доступ до змінних оточення, управляти директоріями і файлами:

1. Перевірка існування об'єкта по заданому шляху.

2. Визначення розміру в байтах.

3. Видалення.

4. Перейменування та ін.

При виклику функцій os необхідно враховувати, що деякі з них можуть не підтримуватися поточної ОС.

Щоб користуватися методами з os, потрібно підключити бібліотеку. Для цього в Python використовується import os, який необхідно описати в файлі до першого звернення до модуля.

Рекомендується використовувати цю інструкцію на початку файлу з вихідним кодом.

Методи з бібліотеки os можуть застосовуватися користувачем для різних цілей. Нижче показані найбільш популярні з них, що дозволяють отримувати дані про операційну систему. Також отримувати відомості про файли та папки, що зберігаються в пам'яті на жорсткому диску ПК.

Щоб дізнатися ім’я поточної ОС, досить скористатися методом name. Залежно від встановленої платформи, він поверне її короткий найменування в строковому поданні. Наступна програма була запущена на ПК з ОС Windows 10, тому результатом роботи функції name є рядок nt.

За замовчуванням робочої Директорією програми є каталог, де міститься документ з її вихідним кодом. Завдяки цьому, можна не вказувати абсолютний шлях до файлу, якщо той перебуває саме в цій папці. Отримати відомості про поточну директорії дозволяє функція getcwd, яка повертає повну адресу робочого каталогу на жорсткому диску. У наступному фрагменті коду показано що буде, якщо передати результат роботи цього методу в print. Як можна помітити, робочої Директорією є каталог program на системному диску C.

Вбудовані функції бібліотеки os дозволяють запускати окремі файли і папки прямо з програми. З цим завданням чудово справляється метод startfile, якому варто передати адресу необхідно об’єкта. Програмне забезпечення, яке використовується для відкриття документа, визначається середовищем автоматично. Наприклад, при запуску звичайного файлу test.txt, як це зроблено в наступному прикладі, задіюється стандартний блокнот. Якщо передати функції посилання на директорію, вона буде відкрита за допомогою вбудованого менеджера файлів.

Іноді для взаємодії з документом необхідно отримати його повне ім’я, яке включає дозвіл, але не абсолютний шлях до нього на диску. Перетворити адреса об'єкта в назву дозволяє функція basename, яка міститься в подмодуле path з бібліотеки os.

Зворотна ситуація виникає тоді, коли користувачеві потрібно отримати тільки шлях до файлу, без самої назви об’єкта. Це допоможе зробити метод dirname, який повертає шлях до заданого документу в строковому поданні, як це продемонстровано в невеликому прикладі нижче. Тут print виводить на екран адреса текстового документа в папці folder.

Повертаючись до класу path з бібліотеки os, варто взяти до уваги функцію split, що дозволяє дуже легко роз’єднувати шлях до файлу та ім’я файлу в різні строки.

Зворотна дія виконує функція join, дозволяючи легко поєднати шлях до документа з його назвою. Як видно з результатів роботи даного коду, завдяки print на екрані буде відображатися шлях, який посилається на текстовий файл test.txt в каталозі folder на D.

Це були базові можливості модуля os, що реалізуються в програмах на мові Python за рахунок безлічі вбудованих методів по управлінню встановленої ОС. Таким чином, вони дають можливість не тільки отримувати корисні відомості про платформу, але і працювати з вмістом диска, створюючи нові директорії і файли, перейменовуючи і повністю видаляючи їх.

Warnings

Попереджувальні повідомлення, як правило, видаються у випадках, коли корисно попередити користувача про певний стан програми, коли ця умова (як правило) не вимагає збільшення винятку та припинення програми. Наприклад, можна створити попередження, коли програма використовує застарілий модуль.

Програмісти Python видають попередження, викликаючи функцію warn (), визначену в цьому модулі. (Програмісти на C використовують PyErr_WarnEx ();

Попереджувальні повідомлення зазвичай пишуться на sys.stderr, але їх розпорядження можна змінювати гнучко, від ігнорування всіх попереджень до перетворення їх у винятки. Розподіл попереджень може змінюватись залежно від категорії попередження, тексту попереджувального повідомлення та місця джерела, де воно видане. Повторення конкретного попередження для одного і того ж місця джерела, як правило, придушується.

Існує два етапи управління попередженням: по-перше, кожного разу, коли видається попередження, визначається, чи слід видавати повідомлення чи ні; далі, якщо повідомлення має бути видане, воно форматується та друкується за допомогою гачка, встановленого користувачем.

Визначення того, чи потрібно видавати попереджувальне повідомлення, контролюється фільтром попередження, який є послідовністю відповідності правил та дій. Правила можна додати до фільтра, викликавши filterwarnings () і повернути до його типового стану, викликавши resetwarnings ().

Друк попереджувальних повідомлень здійснюється за допомогою виклику showwarning (), який може бути відмінено; реалізація за замовчуванням цієї функції формулює повідомлення за допомогою виклику formatwarning (), який також доступний для використання у користувацьких реалізаціях.

2.3 Обробка, очистка та інтеграція даних

Значна частина часу програміста, що займається аналізом і моделіруванням даних, йде на підготовку даних: завантаження, очищення, перетворення та реорганізацію.

Іноді спосіб зберігання даних в файлах або в базі не узгоджується з алгоритмом обробки. Багато хто віддає перевагу писати перетворення даних з однієї форми в іншу на якійсь універсальній мові програмування типу Python, Perl, R або Java або за допомогою наявних в UNIX засобів обробки тексту типу sed або awk. На щастя, pandas доповнює стандартну бібліотеку Python високорівневими, гнучкими і розробленими базовими перетвореннями і алгоритмами, які дозволяють переформатувати дані без особливих проблем.

Взагалі, багато в pandas – в частині як проектування, так реалізації ШІ-обумовлено потребами реальних додатків. Впорядкування даних – термін, що охоплює широкий діапазон тем і часто неофіційно використовується для опису процесу перетворення сирих даних в чистий і організований формат, готовий до використання. Впорядкування даних – це лише один з кроків в передобробці даних, але це важливий крок.

Розробляючи модель, спершу потрібно імпортувати всі необхідні модулі (рис.2.6 та рис.2.7) та бібліотеки, загальні були описані у попередньому розділі, скачати та встановити їх при неохідності з офіційного сайту.

На першому етапі потрібно створити фігуру-каркас за допомогою %matplotlib.inline та задаються у методі необхідні параметри розмірів фігури.

Наступний етап це зчитування даних з якогось набору.

Набір даних взято з сайту Kaggle. Цей набір даних містить ціни продажу будинку для округу Кінг, Сіетлі. Сюди входять будинки, продані в період з травня 2017 року по травень 2018 року. Це чудовий набір даних для оцінки простих регресійних моделей.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import axes3d, Axes3D
        import seaborn as sb
        sb.set();
        from pylab import rcParams
        import sklearn
        from sklearn.linear_model import LinearRegression
        from sklearn.preprocessing import scale
        from collections import Counter
        from sklearn import linear_model
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error, r2_score
        from mpl_toolkits.mplot3d import Axes3D
        import folium
        from folium.plugins import HeatMap
        %matplotlib inline
        import warnings
        warnings.filterwarnings('ignore')
        import plotly.offline as ply
        import plotly.graph_objs as go
        ply.init_notebook_mode(connected=True)
        from math import sqrt, log, exp
        import os
```

Рисунок 2.6 – Імпорт необхідних бібліотек (Початок)

```
In [1]: import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np
        import mpl_toolkits as mpl_toolkits
        from math import radians, cos, sin, asin, sqrt, log
        from datetime import datetime
        from sklearn import datasets, linear_model
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error, r2_score
        #for decision tree
        from sklearn.tree import DecisionTreeRegressor
        #for clustering\graphics
        from sklearn.datasets import make_blobs
        from sklearn.cluster import KMeans
        from sklearn.cluster import DBSCAN
```

Рисунок 2.7 – Імпорт необхідних бібліотек (Кінець)

Файл містить дуже багато даних, називається kc_house_data.csv. Використовуючи метод read_csv(), передається назва файлу (рис.2.8) та шлях як параметри – дані обробляються та за допомогою метода head() і виводиться результат.

```
In [2]: %matplotlib inline
        rcParams['figure.figsize'] = 5, 4
        sb.set_style('whitegrid')

In [3]: houses = pd.read_csv("kc_house_data.csv")
        houses.head()
```

Рисунок 2.8 – Зчитування даних з файлу

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_l
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1

5 rows × 21 columns

id ідентифікатор date дата коли був проданий будинок price ціна(цільовий показник) bedrooms кількість спалень bathrooms кількість ван sqft_living житлова площа sqft_lot площа лоту floors кількість поверхів waterfront будинок з видом на набережну view перегляд condition наскільки хороші умови в цілому grade загальна оцінка sqft_above квадратні метри крім підвалу sqft_basement квадратний метр підвалу yr_built рік побудови yr_renovated рік реставрації zipcode код індекс lat координати широт long координати довготи sqft_living15 додаткова площа забудови sqft_lot15 додаткова площа лоту

Рисунок 2.9 – Результат виводу

Порожні дані та дані які мають нулюве значення не використовуються , а просто їх очищається (рис.2.10). А наступним кроком виведеться тип даних по кожному рядку.

Впорядкування даних – термін, що охоплює широкий діапазон тем і часто неофіційно використовується для опису процесу перетворення сирих даних в чистий і організований формат, готовий до використання.

Тепер, для того, щоб краще зрозуміти, як можна оптимізувати використані пам’яті цих об'єктів DataFrame, представлене зображення внутрішнього відображення.

```
In [4]: print(houses.isnull().any())
        print(houses.dtypes)

id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
id                int64
date             object
price           float64
bedrooms          int64
bathrooms       float64
sqft_living       int64
sqft_lot          int64
floors          float64
waterfront        int64
view              int64
condition         int64
grade             int64
sqft_above        int64
sqft_basement     int64
yr_built          int64
yr_renovated      int64
zipcode           int64
lat             float64
long            float64
sqft_living15     int64
sqft_lot15        int64
dtype: object
```

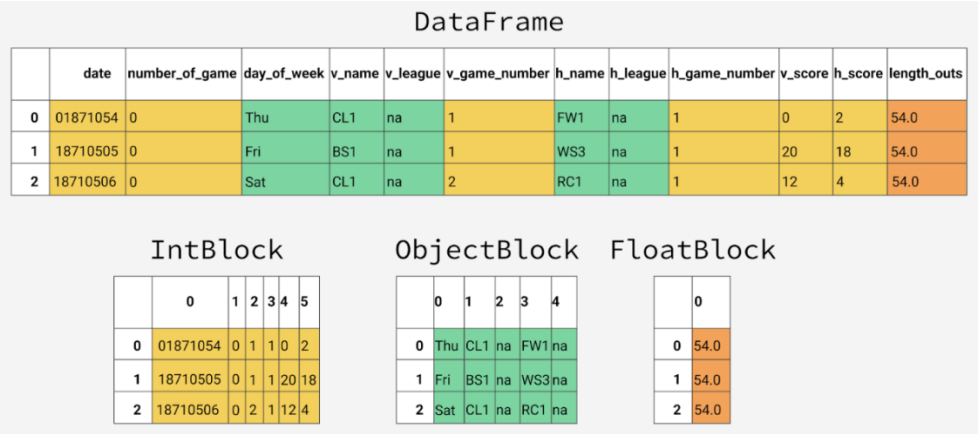
Рисунок 2.10 – Вивід типів даних кожного значення

Можна помітити, що блоки не зберігають відомості про імена стовпців. Відбувається це через те, що блоки оптимізовані для зберігання значень, наявних в осередках таблиці об’єкта DataFrame.

За зберігання відомостей про відповідність між індексами рядків і стовпців набору даних і того, що зберігається в блоках однотипних даних, відповідає клас BlockManager. Він грає роль API, який надає доступ до базових даними. Коли ми читаємо, редагуємо чи видаляємо значення, клас DataFrame взаємодіє з класом BlockManager для перетворення наших запитів в виклики функцій і методів.

Кожен тип даних має спеціалізований клас в модулі pandas.core.internals. Наприклад, pandas використовує клас ObjectBlock для подання блоків, що містять рядкові стовпці, і клас FloatBlock для подання блоків, що містять стовпці, що зберігають числа з плаваючою крапкою (рис.2.10).

Всередині pandas настільних даних групи групуються в заблокованих із значеннями одинакового типу. Далі, наприклад, в pandas зберігаються первісні 12 об'єктів DataFrame (рис.2.11).



	date	number_of_game	day_of_week	v_name	v_league	v_game_number	h_name	h_league	h_game_number	v_score	h_score	length_outs
0	01871054	0	Thu	CL1	na	1	FW1	na	1	0	2	54.0
1	18710505	0	Fri	BS1	na	1	WS3	na	1	20	18	54.0
2	18710506	0	Sat	CL1	na	2	RC1	na	1	12	4	54.0

	0	1	2	3	4	5
0	01871054	0	1	1	0	2
1	18710505	0	1	1	20	18
2	18710506	0	2	1	12	4

	0	1	2	3	4
0	Thu	CL1	na	FW1	na
1	Fri	BS1	na	WS3	na
2	Sat	CL1	na	RC1	na

	0
0	54.0
1	54.0
2	54.0

Рисунок 2.11 - Внутрішнє представлення об'єкта DataFrame

Для блоків, що представляють числові значення, що виглядають як цілі числа або числа з плаваючою точкою, pandas комбінує стовпці і зберігає їх у вигляді структури даних ndarray бібліотеки NumPy. Ця структура даних побудована на основі масиву C, значення зберігаються в безперервному блоці пам’яті. Завдяки такій схемі зберігання даних доступ до фрагментів даних здійснюється дуже швидко.

Дальше можна розглянути перших 10 даних для кращого розуміння всєї ситуації про об’єкти.

Щоб подивитися загальну інформацію (рис.2.12) по датафрейму і всіма ознаками, скористаємося методом info() (рис.2.13):

```
In [6]: houses.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id               21613 non-null int64
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21613 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21613 non-null float64
waterfront       21613 non-null int64
view             21613 non-null int64
condition        21613 non-null int64
grade            21613 non-null int64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Рисунок 2.12 – Вивід загальної інформації по дата фрейму

```
In [6]: houses.head(10).T
Out[6]:
```

	0	1	2	3	4	5	6	
id	7129300520	6414100192	5631500400	2487200875	1954400510	7237550310	1321400060	200800
date	20141013T000000	20141209T000000	20150225T000000	20141209T000000	20150218T000000	20140512T000000	20140627T000000	20150115T00
price	221900	538000	180000	604000	510000	1.225e+06	257500	291
bedrooms	3	3	2	4	3	4	3	
bathrooms	1	2.25	1	3	2	4.5	2.25	
sqft_living	1180	2570	770	1960	1680	5420	1715	1
sqft_lot	5650	7242	10000	5000	8080	101930	6819	9
floors	1	2	1	1	1	1	2	
waterfront	0	0	0	0	0	0	0	
view	0	0	0	0	0	0	0	
condition	3	3	3	5	3	3	3	
grade	7	7	6	7	8	11	7	
sqft_above	1180	2170	770	1050	1680	3890	1715	1
sqft_basement	0	400	0	910	0	1530	0	
yr_built	1955	1951	1933	1965	1987	2001	1995	1
yr_renovated	0	1991	0	0	0	0	0	
zipcode	98178	98125	98028	98136	98074	98053	98003	98
lat	47.5112	47.721	47.7379	47.5208	47.6168	47.6561	47.3097	47.4
long	-122.257	-122.319	-122.233	-122.393	-122.045	-122.005	-122.327	-122
sqft_living15	1340	1690	2720	1360	1800	4760	2238	1
sqft_lot15	5650	7639	8062	5000	7503	101930	6819	9

Рисунок 2.13 – Вивід 10 даних з файлу

Метод describe() показує основні статистичні характеристики даних (рис.2.14) по кожному числовому ознакою (типи int64 і float64): число непропущених значень, середнє, стандартне відхилення, діапазон, медіану, 0.25 і 0.75 квартили.

```
houses.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id	21613.0	4.580302e+09	2.876566e+09	1.000102e+06	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	21613.0	5.400881e+05	3.671272e+05	7.500000e+04	3.219500e+05	4.500000e+05	6.450000e+05	7.700000e+06
bedrooms	21613.0	3.370842e+00	9.300618e-01	0.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
bathrooms	21613.0	2.114757e+00	7.701632e-01	0.000000e+00	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
sqft_living	21613.0	2.079900e+03	9.184409e+02	2.900000e+02	1.427000e+03	1.910000e+03	2.550000e+03	1.354000e+04
sqft_lot	21613.0	1.510697e+04	4.142051e+04	5.200000e+02	5.040000e+03	7.618000e+03	1.068800e+04	1.651359e+06
floors	21613.0	1.494309e+00	5.399889e-01	1.000000e+00	1.000000e+00	1.500000e+00	2.000000e+00	3.500000e+00
waterfront	21613.0	7.541757e-03	8.651720e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
view	21613.0	2.343034e-01	7.663176e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
condition	21613.0	3.409430e+00	6.507430e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	5.000000e+00
grade	21613.0	7.656873e+00	1.175459e+00	1.000000e+00	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
sqft_above	21613.0	1.788391e+03	8.280910e+02	2.900000e+02	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03

Рисунок 2.14 – Вивід основних статистичних характеристик

Висновки до розділу 2

У даному розділі було детально розглянуто всі інтструменти та бібліотеки які використовуються в розробці статистичної моделі, проаналізовано переваги та недоліки, а також розглянуто перший крок створення моделі, імпорт бібліотек та модулей, розроблено зчитування даних з датасету, аналіз даних, очистка та середня статистична оцінка.

РОЗДІЛ 3. РОЗРОБКА СТАТИСТИЧНОЇ МОДЕЛІ ПРОГНОЗУВАННЯ

3.1 Кореляція даних

Кореляційний аналіз – це статистичне дослідження (стохастичної) залежності між випадковими величинами (англ. correlation – взаємозв’язок). У найпростішому випадку досліджують дві вибірки (набори даних), у загальному – їх багатовимірні комплекси (групи).

Мета кореляційного аналізу – виявити чи існує істотна залежність однієї змінної від інших.

Головні завдання кореляційного аналізу:

1. Оцінка за вибірковими даними коефіцієнтів кореляції.
2. Перевірка значущості вибіркових коефіцієнтів кореляції або кореляційного відношення.
3. Оцінка близькості виявленого зв’язку до лінійного.
4. Побудова довірчого інтервалу для коефіцієнтів кореляції.
5. Обмеження кореляційного аналізу.

Кореляція відображає лише лінійну залежність величин, але не відображає їх функціональної зв’язності. Наприклад, якщо обчислити коефіцієнт кореляції між величинами A = sin(x) та B = cos(x), він буде наближений до нуля, тобто залежність між величинами відсутня. Між тим, величини A та B очевидно зв'язані між собою за законом $\sin^2(x) + \cos^2(x) = 1$.

Використання можливе у випадку наявності достатньої кількості випадків для вивчення: для конкретного типу коефіцієнту кореляції становить від 25 до 100 пар спостережень.

Кореляція не означає причинність.

У статистиці – залежність або пов’язаність є будь-яким статистичним відношенням, чи каузальним, чи ні, між двома випадковими величинами або біваріантними даними. До поняття кореляція відноситься будь-яке статистичне

відношення із широкого класу відношень, що задають залежність величин, хоча зазвичай про кореляцію говорять тоді, коли дві величини перебувають у лінійному відношенні між собою. При цьому зміна однієї або кількох цих величин призводить до систематичної зміни іншої або інших величин. Відомими прикладами залежних феноменів є кореляція між фізичними параметрами батьків та їхніх дітей і кореляція між попитом на товар і його ціною.

Користь кореляцій у тому, що вони можуть указувати на відношення, яке може носити передбачуваний характер і тому мати практичне застосування. Наприклад, електрогенеруюча компанія може виробляти менше електрики у періоди з гарною погодою, базуючись на кореляції між попитом на електрику та погодою. У цьому випадку існує причинно-наслідковий зв'язок, тому що в екстремальну погоду люди використовують більше електрики для опалювання або охолодження. Однак зазвичай самої лише наявності кореляції недостатньо для того, щоб зробити висновок про наявність причинно-наслідкового зв'язку (що часто формулюють фразою “кореляція не означає причинності”.

Кореляція може бути позитивною та негативною (можлива також ситуація відсутності статистичного зв’язку – наприклад, для незалежних випадкових величин). Від'ємна кореляція – кореляція, при якій збільшення однієї змінної пов'язане зі зменшенням іншої, при цьому коефіцієнт кореляції від’ємний. Додатна кореляція – кореляція, при якій збільшення однієї змінної пов'язане зі збільшенням іншої, при цьому коефіцієнт кореляції додатний.

Найбільш відомою мірою залежності двох величин є коефіцієнт кореляції Пірсона, який зазвичай називають спрощено коефіцієнтом кореляції. Він розраховується як відношення коваріації двох випадкових величин на добуток їх стандартних відхилень. Карл Пірсон розробив цей коефіцієнт, взявши за основу подібну, але дещо відмінну ідею Френсіса Гальтона.

У даній моделі за допомогою методу corr() можна вивести загальну кореляцію всіх даних (рис.3.1):

```
In [8]: f, ax = plt.subplots(figsize=(18, 14))
        sb.heatmap(houses.corr(), annot=True,annot_kws={'size': 12}, linewidths=.5, fmt='.2f', ax=ax)
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa26d396128>
```

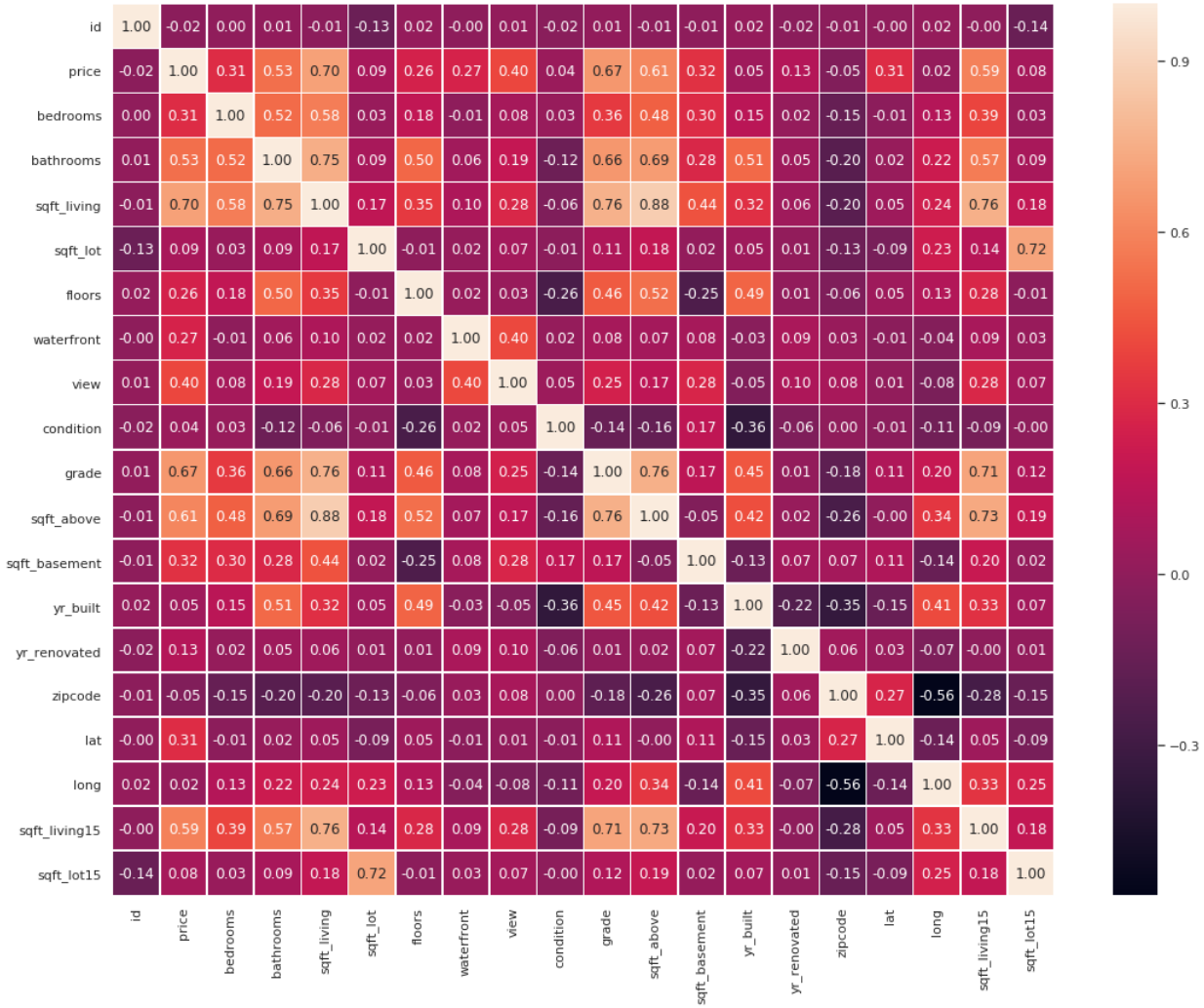


Рисунок 3.1 – Вивід кореляції даних

Отже, розрахований коефіцієнт кореляції свідчить про наявність значного зв’язку між рівнем ціни та інших значень що приближені до 1.

Оскільки обчислений коефіцієнт кореляції більший за критичне його значення (0,6 > 0,3809), то з вірогідністю 0,95 можна стверджувати про статистично достовірну залежність між даними.

Шкала оцінки тісноти зв’язку за коефіцієнтом кореляції та критичне його.

Коефіцієнт кореляції – Тіснота зв’язку

1,00 – Зв’язок функціональний

0,90 – 0,99 – Дуже сильний

0,70 – 0,89 – Сильний

0,50 – 0,69 – Значний

0,30 – 0,49 – Помірний

0,10 – 0,29 – Слабкий

0,00 – Зв'язок відсутній

Крім відображення щільності зв’язку, коефіцієнт кореляції відіграє ще одну важливу роль – через коефіцієнт детермінації (D) він характеризує розмір впливу факторів на результативну ознаку. Можна також зробити кореляцію (рис.3.2) що вже конкретно буде залежати від певних ознак (рис.3.3):

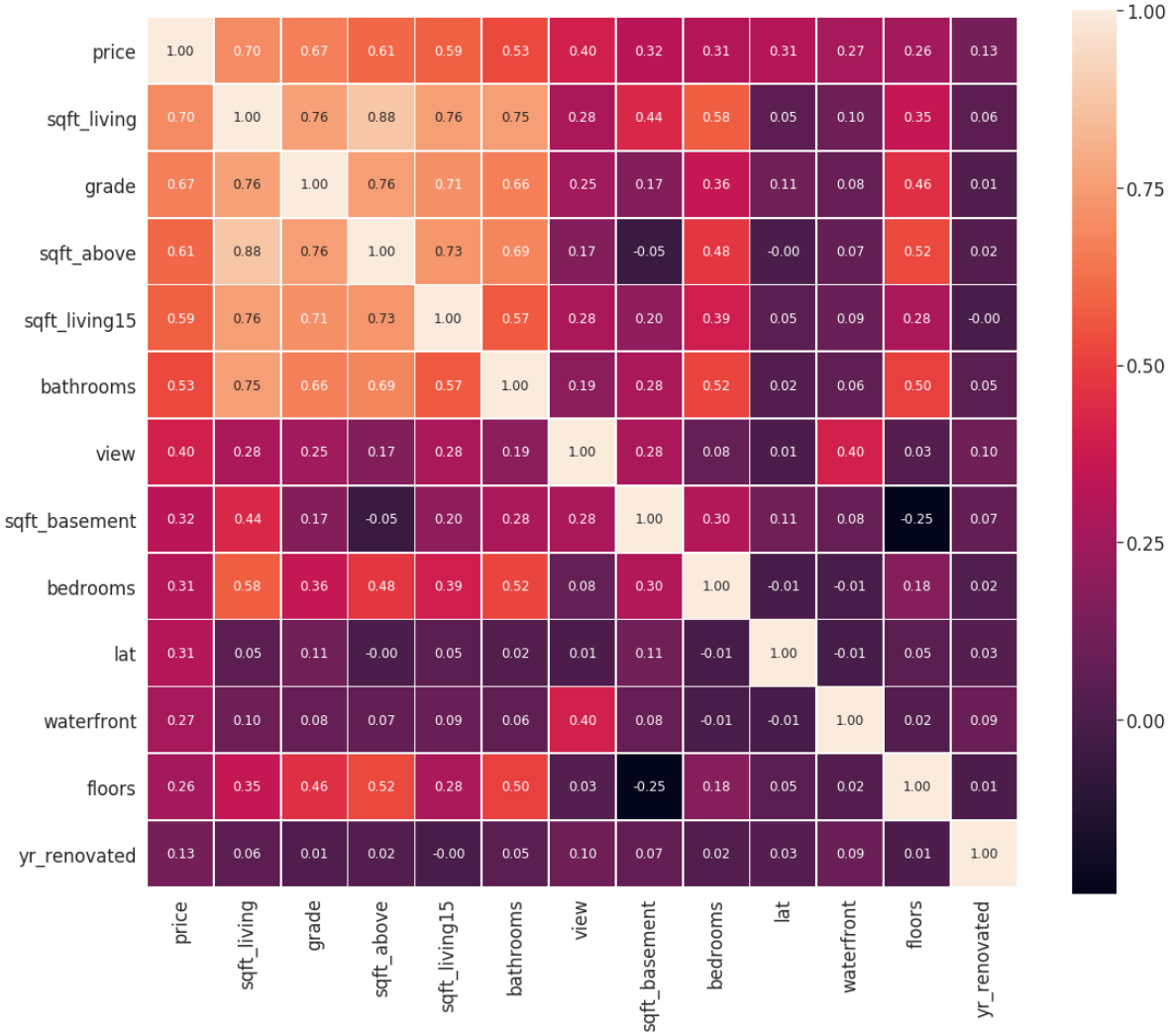


Рисунок 3.2 – Кореляція від одного атрибута

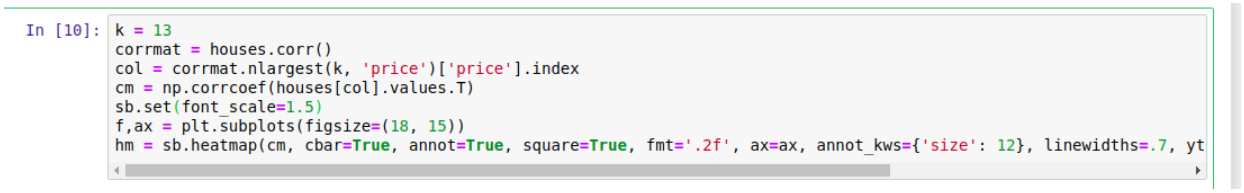


Рисунок 3.3 – Код виклику методів

Кореляція може бути позитивною та негативною (можлива також ситуація відсутності статистичного зв'язку – наприклад, для незалежних випадкових величин). Від’ємна кореляція – кореляція, при якій збільшення однієї змінної пов'язане зі зменшенням іншої, при цьому коефіцієнт кореляції від'ємний. Додатна кореляція – кореляція, при якій збільшення однієї змінної пов'язане зі збільшенням іншої, при цьому коефіцієнт кореляції додатний.

Тоді результат буде відображати вже пряму залежність кожного атрибута від ціни:

3.2 Лінійна регресія

Можна вважати, що модель – це спрощене теоретичне припущення наближених до складної реальності. Тому в моделі завжди існує неточність, яку називають похибкою апроксимації. Саме величина похибки і дозволяє вибрати відповідну модель з безлічі можливих. Похибка вираховується як квадрат відстані між реальнимн і передбаченими моделлю даними; якщо f – навчена модельна функція, то похибка дорівнює:

def error (f, x, y):

return sp.sum ((f (x) -y) ** 2)

Вектори x і y містять підготовлені раніше статистичні даних про запити. Тут наочно відображається зручність векторних функцій в SciPy. Передбачається, що навчена модель приймає вектор і повертає результати у вигляді вектора того ж розміру, так що різниця між результатом і вектором є коректно визначена. Вірність на навчальних даних майже завжди дає надмірну, але оптимістичну

оцінку якості алгоритму. вимірювати слід вірність на тестових даних, тобто на прикладах, що не пред'являлися на етапі навчання.

Факт залишається фактом: лінійна регресія і її розширення – продовжує залишатися поширеним і корисним методом передбачення, коли вектор цілей є кількісним значенням (наприклад, ціна будинку, вік). Лінійна регресія виходить з того, що зв’язок між ознаками і вектором цілей є приблизно лінійною, т. е. ефект (також званий коефіцієнтом тому, вагою або параметром) ознак на вектор цілей є постійним.

У статистиці лінійна регресія (рис.3.4) – це метод моделювання залежності між скаляром y та векторною (у загальному випадку) змінною X. У разі, якщо змінна X також є скаляром, регресію називають простою.

Загалом лінійна регресійна модель визначається у виді:

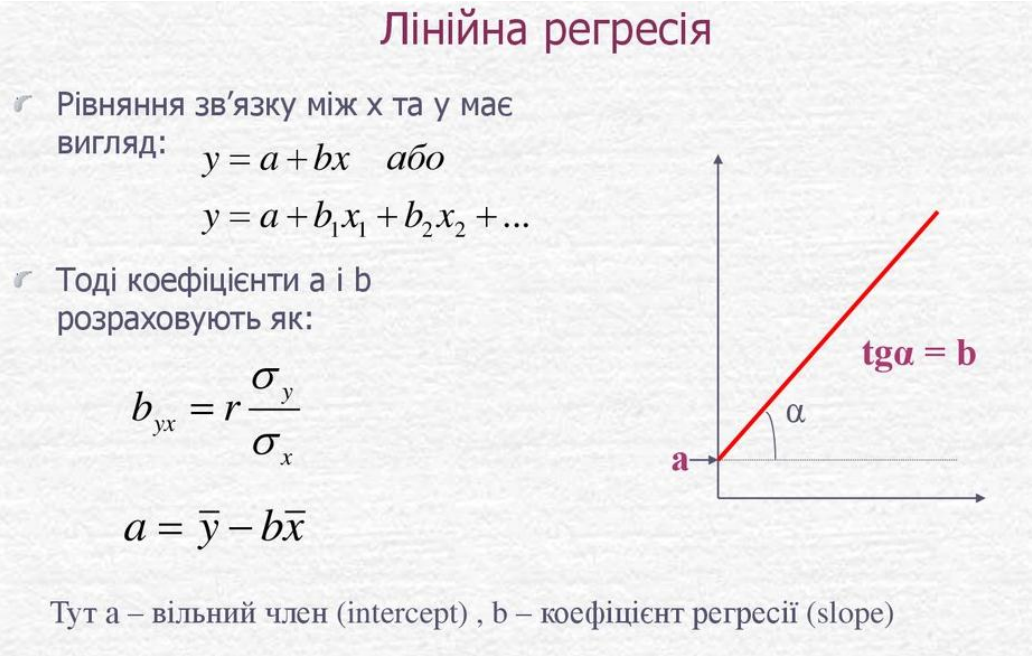


Рисунок 3.4 – Загальний вигляд Лінійної Регресії

При використанні лінійної регресії взаємозв’язок між даними моделюється за допомогою лінійних функцій, а невідомі параметри моделі оцінюються за вхідними даними. Подібно до інших методів регресійного аналізу лінійна регресія

повертає розподіл умовної імовірності y в залежності від X, а не розподіл спільної імовірності y та X, що стосується області мультиваріативного аналізу.

При розрахунках параметрів моделі лінійної регресії зазвичай застосовується метод найменших квадратів (МНК), але також можуть бути використані інші методи. Так само метод найменших квадратів може бути використаний і для нелінійних моделей. Тому МНК та лінійна регресія хоч і є тісно пов'язаними, але не є синонімами.

У машинному навчанні побудова алгоритмів, які вміють навчатись та робити передбачення на даних є поширеною задачею. Такі алгоритми працюють шляхом створення прогнозів або прийняття рішень: 2 заснованих на даних, які використовуються для побудови відповідної математичної моделі. Дані, які використовуються для побудови остаточної моделі беруться з різних наборів даних. Зокрема, три набори даних зазвичай використовуються на різних етапах створення моделі.

Модель спочатку будується за допомогою навчального набору даних (англ. training dataset), який є множиною прикладів, які використовуються для налаштування параметрів (наприклад, це можуть бути ваги зв'язків між нейронами в штучних нейронних мережах) моделі. За допомогою навчання з учителем, модель (наприклад, нейронна мережа або наївний баєсів класифікатор) “тренується” на навчальних даних (це може бути градієнтний спуск або стохастичний градієнтний спуск). На практиці, навчальний набір даних часто складається з пар – вхідного вектора питання та відповіді у вигляді вектора або скаляра, яка зазвичай позначається як ціль. Модель запускається з набором навчальних даних і продукує результат, який потім порівнюється з ціллю для кожного вхідного вектора в тренувальному наборі. На підставі результату порівняння та відповідно до обраного алгоритму навчання, параметри моделі коригуються. Налаштування моделі може містити як обирання ознак, так і оцінку параметрів.

Далі, підібрана модель використовується для прогнозування відповідей для спостережень у другому наборі даних, що називається набором даних для

затверджування (англ. validation dataset). Набір даних для затверджування забезпечує об'єктивну оцінку моделі, яка відповідає навчальному набору при налаштуванні гіперпараметрів моделі (наприклад, число прихованих шарів нейронної мережі). Набори даних для затверджування можуть бути використані для регуляризації шляхом ранньої зупинки: навчання переривається, коли помилка на наборі даних для затверджування збільшується, оскільки це є ознакою перенавчання на навчальному наборі даних. Ця проста процедура на практиці ускладнюється тим, що помилка набору даних для затверджування може коливатися під час тренування, утворюючи декілька локальних мінімумів. Це ускладнення призвело до появи багатьох ad-hoc правил для визначення того, що перенавчання справді почалось на рис.3.5.

```
In [17]: train_data, test_data = train_test_split(houses, train_size =0.8, random_state = 3)
reg = linear_model.LinearRegression()
x_train = np.array(train_data['sqft_living']).reshape(-1,1)
y_train = np.array(train_data['price']).reshape(-1, 1)
reg.fit(x_train, y_train)
#evaluate simple model
x_test = np.array(test_data['sqft_living']).reshape(-1, 1)
y_test = np.array(test_data['price']).reshape(-1, 1)
pred = reg.predict(x_test)
print('Simple Model')


print('R-squared (training) ', round(reg.score(x_train, y_train), 3))
print('R-squared (testing) ', round(reg.score(x_test, y_test), 3))
print('Intercept: ', reg.intercept_)
print('Coefficient:', reg.coef_)

Simple Model
R-squared (training)  0.492
R-squared (testing)  0.496
Intercept:  [-47235.8113029]
Coefficient: [[282.2468152]]

In [18]: _, ax = plt.subplots(figsize= (10, 12))
plt.scatter(x_test, y_test, color= 'darkgreen', label = 'data')
plt.plot(x_test, reg.predict(x_test), color='red', label= ' Predicted Regression line')
plt.xlabel('Living Space (sqft)')
plt.ylabel('price')
plt.legend()
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```

Рисунок 3.5 – Код створення простої моделі

Нарешті, тестовий набір даних (англ. test dataset) – це набір даних, який використовується для забезпечення об'єктивної оцінки кінцевої моделі, яка відповідає навчальному набору даних.

На жаль, терміни тестовий набір даних та навчальний набір даних іноді заміняють один одного, що може призвести до плутанини. Як наслідок, стає загальноприйнятою посилання на набір, що використовується в ітераційному тренінгу, як набір тестів/перевірок (англ. test/validation set) та набір, який використовується для налаштування гіперпараметрів як відсторонений набір (англ. holdout set).

В даному випадку також ділиться датасет на треновані та тестові дані.

За допомогою метода fit() модель навчається, викликається метод predict()-результат прогнозування з використанням Простої Лінійної Регресії (рис.3.6).

Результат можна побудувати на графіку :

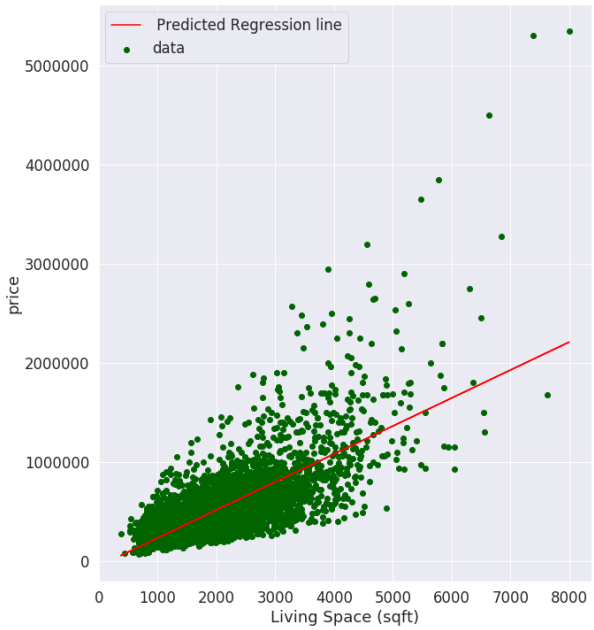


Рисунок 3.6 – Лінійна регресія

Але насправді результат не зовсім задовільняє вимоги, тому буде непоганим варіантом покращити модель з використанням аналізу додаткових “фічей” (другорядних атрибутів що є в даних). Ці ознаки, наприклад розміщення об’єктів на карті, координати широта та довготи, житлова площа, поштовий індекс – дуже допоможуть підвищити якісь результату до більш наближеного (рис.3.7 та рис.3.8).

```
n [25]: #split in X,y
y=houses.loc[:,"price"].apply(lambda x: log(x))

X=houses.loc[:,["sqft_living","grade","floors","lat","long","bathrooms","bedrooms","yr_built","yr_renovated","view"
X["sqft_living"]= X["sqft_living"].apply(lambda x: log(x))
X["lat"]= X["lat"].apply(lambda x: abs(47.63-x))
X["long"]= X["long"].apply(lambda x: log(abs(x)))
X["yr_built"]= X["yr_built"].apply(lambda x: log(abs(x-1955)+1))
X["yr_renovated"]= X["yr_renovated"].apply(lambda x: log(x+1))
X["sqft_lot"]= X["sqft_lot"].apply(lambda x: log(x))

X["lat*long"]=X["lat"]*X["long"]
X["sqft_living*sqft_lot"]=X["sqft_living"]*X["sqft_lot"]

# split in train, test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=123)
#split in validation, test
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=123)
```

Рисунок 3.7 – Код вдосконалення моделі

```
In [26]: # Validation Set
regr= linear_model.LinearRegression()
regr.fit(X_train,y_train)
y_pred= regr.predict(X_val)

print("Validation Set")
print("Root Mean Square Error: %6.f" % sqrt(mean_squared_error(np.exp(y_val.values),np.exp(y_pred))))
print('R2 score: %.6f' % r2_score(np.exp(y_val.values),np.exp(y_pred)))

Validation Set
Root Mean Square Error: 142748
R2 score: 0.841722

In [27]: #Test set
y_pred= regr.predict(X_test)
print("Test Set")
print("Root Mean Square Error (squared): %2.f" % sqrt(mean_squared_error(np.exp(y_test.values),np.exp(y_pred))))
print('R2 score: %.2f' % r2_score(np.exp(y_test.values),np.exp(y_pred)))

Test Set
Root Mean Square Error (squared): 154262
R2 score: 0.83
```

Рисунок 3.8 – Результат навчальних і тестових даних

Результат на наборі для перевірки та тестових даних набагато піднявся до кращої оцінки (рис.3.9), але тепер час перевірити модель на реальних тренувальних даних:

```
In [28]: # Predicting real life data (Training)
#split in X,y
y=houses.loc[:,"price"].apply(lambda x: log(x))

X=houses.loc[:,["sqft_living","sqft_lot","yr_built","bedrooms","bathrooms","lat","long","waterfront"]]
X["sqft_living"]= X["sqft_living"].apply(lambda x: log(x))
X["sqft_lot"]= X["sqft_lot"].apply(lambda x: log(x))
X["lat"]= X["lat"].apply(lambda x: abs(47.63-x))
X["long"]= X["long"].apply(lambda x: abs(x))
X["yr_built"]= X["yr_built"].apply(lambda x: log(x))


X["lat*long"]=X["lat"]*X["long"]
X["sqft_living*sqft_lot"]=X["sqft_living"]*X["sqft_lot"]

# split in train, test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=123)

regr_test= linear_model.LinearRegression()
regr_test.fit(X_train,y_train)
y_pred= regr_test.predict(X_test)
p = r2_score(np.exp(y_test.values),np.exp(y_pred))
print(p)
print("Root Mean Square Error: %2.f" % sqrt(mean_squared_error(np.exp(y_test.values),np.exp(y_pred))))
print('R2 score: %.6f' % p)

0.7622146910961412
Root Mean Square Error: 179697
```

Рисунок 3.9 – Результат тестих даних

Результат, показаний на навчальних даних (підмножині всього набору даних) виявився навіть краще, ніж раніше. Але важливо відзначити, що результат на тестових даних гірше. Початківцю це може здатися дивним, але взагалі немає нічого несподіваного в тому, що вірність на тестових даних нижче, ніж на навчальних. Щоб зрозуміти, чому це так, погляньте на графік, де показана вирішальна межа. Важливість цих понять зростає з збільшенням складності моделі. В даному прикладі відмінність між вірністю на навчальних і на тестових даних невелика. Але якщо модель складна, то цілком можливо на навчальних даних отримати стовідсоткову вірність, а на тестових – не більшу, ніж при випадковому вгадуванні.

Вище, резервуючи дані, використовувати для навчання тільки половину наявних даних. Бути може, було б краще збільшити цю частку. З іншого боку, якщо залишити для тестування занадто мало даних, то може не вистачити прикладів для оцінювання похибки. В ідеалі було б добре використовувати всі дані як для навчання, так і для тестування, але це неможливо.

Ідеальний результат може дати метод перехресної перевірки. Одна з його форм називається перехресною перевіркою з виключенням по одному. В даній моделі вибирається який-небудь приклад з наявних даних, навчається модель на всіх даних, крім цього прикладу, а потім перевіряється, чи правильно модель класифікує цей приклад. Цей процес повторюється для всіх елементів набору даних.

У цьому циклі тестується послідовність моделей на всіх прикладах, а по його завершенні друкується середній результат. При використанні перехресної перевірки зацикленість не виникає, тому що кожна модель тестується на прикладі, який вона не бачила при навчанні. Тому прогнозуючу таким чином оцінку можна вважати надійним показником загальної моделі на нові дані.

Основна проблема перехресної перевірки з вилученням по одному полягає в тому, що приходиться пробляти набагато більше роботи. По суті, потрібно навчати нову модель для кожного прикладу, і з ростом набору даних витрати виявляються дуже великі. Отримати переваги виключення по одному з набагато

меншими витратами дозволяє x-прохідна перехресна перевірка, де x - невелике ціле число. Наприклад, вразі точкової прохідної перевірки розбиваються дані на 5 груп.

Потім навчаються п’ять моделей, кожен раз виділяється з навчального набору одну групу. Добутий код резервується відразу 20 % даних, а не один елемент. Всі навчені моделі тестуються на даних, що входять в виключену групу, і результати усереднюються.

Після виконання тренування нашої моделі можна поглянути на значення кожного параметра. Наприклад, P0, так зване зсув або перетин, можна переглянути за допомогою атрибута intercept()

Використовуючи метод predict(), можна передбачити значення для цього будинку.

Основною перевагою лінійної регресії є її інтерпритованість (рис.3.10) в значній мірі тому, що модельні коефіцієнти являють собою хорошими показниками.

```
actual_values = y_test
predictions = y_pred
sb.scatterplot(actual_values, predictions)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa25e70d9e8>
```

Рисунок 3.10 – Код для відображення графіка

Цей результат на даний момент являється найкращим (рис.3.11), щож можна спробувати зообразити це за допомогою графіка:

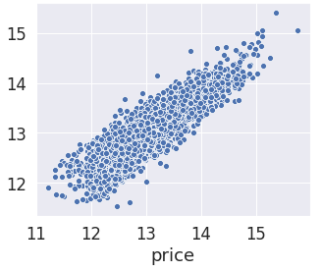


Рисунок 3.11 – Графік моделі

А тепер ще раз використовується Лінійна регресія (рис.3.12 та рис.3.13) для кращої демонстрації залежності даних:

```
In [30]: y_train_pred = regr_test.predict(X_train)
         y_test_pred = regr_test.predict(X_test)

In [31]: # Plot residuals
         plt.scatter(y_train_pred, y_train_pred - y_train, c = "blue", marker = "s", label = "Training data")
         plt.scatter(y_test_pred, y_test_pred - y_test, c = "lightgreen", marker = "s", label = "Validation data")
         plt.title("Linear regression")
         plt.xlabel("Predicted values")
         plt.ylabel("Residuals")
         plt.legend(loc = "upper left")
         plt.hlines(y = 0, xmin = 11.5, xmax = 15.5, color = "red")
         plt.show()
```

Рисунок 3.12 – Код програми для повторної Лінійної Регресії

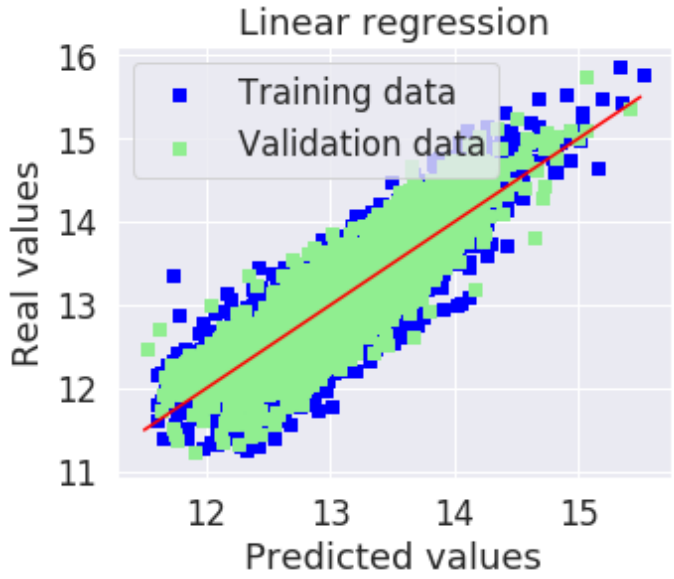


Рисунок 3.13 – Лінійна регресія для навчальних та тренувальних даних

Сортуються дані до максимальних, мінімальних, середніх значень для кожного атрибута(параметра) на рис.3.14 та рис.3.15.

```
In [33]: index=["min","max","mean","coef"]
         columns=list(X_test)
         test = pd.DataFrame(index=index, columns=columns)
         test.loc["min"]=list(X_test.min())
         test.loc["max"]=list(X_test.max())
         test.loc["mean"]=list(X_test.mean())
         test.loc["coef"]=regr_test.coef_
```

Рисунок 3.14 – Код програми для виводу середньої статистики

In [34]:
```
test
```

Out[34]:

	sqft_living	sqft_lot	yr_built	bedrooms	bathrooms	lat	long	waterfront	lat*long	sqft_living*sqft_lot
min	6.04025	6.39693	7.54961	0	0	0.0001	121.676	0	0.0121911	44.3775
max	9.19928	13.9681	7.60837	10	7.75	0.4524	122.515	1	55.1978	113.265
mean	7.55232	8.99356	7.58617	3.37276	2.11636	0.122584	122.215	0.0089451	14.9804	68.0463
coef	0.636933	0.00800451	-0.348176	-0.0477763	0.0823561	682.224	1.05851	0.728813	-5.60684	0.00903068

Рисунок 3.15 – Вивід результату

3.3 3D – візуалізація з Axes3D

Усі класи для роботи з трьохмірними графіками знаходяться у пакеті mpl_toolkits.mplot3d, з якого потрібно буде їх імпортувати.

Для того, щоб намалювати трьохмірний графік, в першому очевидно потрібно створити трьохмірні осі. Щоб їх реалізувати, створюється екземпляр класу mpl_toolkits.mplot3d.Axes3D, конструктор якого реалізує як мінімум один параметр – екземпляр класу matplotlib.figure.Figure, який, у своїй черзі, можна створити викликавши pylab.figure (). У конструкторі класу matplotlib.figure.Figure є ще й інші незвичайні параметри, але вони не показують їх.

Існують інші параметри методу Axes3D.plot_surface (X, Y, Z, * args, ** kwargs), їх не так багато:

X, Y і Z – ці параметри задають сітку і значення функції в вузлах.

rstride і cstride задають крок виведення графіка. Чим менший крок, тим точніше відображається функція, але тим довше відбувається малювання.

color – задає колір графіка

cmap – задає градієнт кольорів, щоб колір графіка залежав від значення функції в цій області.

Побудова графіків, а також статична або інтерактивна візуалізація – одна з найважливіших завдань аналізу даних. Вони можуть бути частиною процесу дослідження, наприклад, застосовуватися для виявлення викидів, визначення необхідних перетворень даних або пошуку ідей для побудови моделей.

В інших випадках побудова інтерактивної візуалізації для веб-сайту, наприклад за допомогою бібліотеки d3.js, може бути кінцевою метою.

Matplotlib – це пакет для побудови графіків (головним чином, двовимірних) поліграфічної якості. Проєкт був заснований Джоном Хантером в 2002 році з метою реалізувати на Python інтерфейс, аналогічний MATLAB. Згодом він сам, Фернандо Перес (з групи розробників IPython) та інші люди протягом багатьох років працювали над тим, щоб зробити комбінацію IPython і matplotlib максимально функціональним і продуктивним середовищем для наукових розрахунків. При використанні в поєднанні з якою-небудь бібліотекою ГІП (наприклад, всередині IPython), matplotlib набуває інтерактивних можливостей: панорамування, масштабування і інші. Цей пакет підтримує різноманітні системи ГІП у всіх операційних системах, а також вміє експортувати графічні дані у всіх векторних і растрових форматах: PDF, SVG, JPG, PNG, BMP, GIF і т. Д. З його допомогою побудувано майже всі малюнки (рис.3.16) для статистичної моделі.

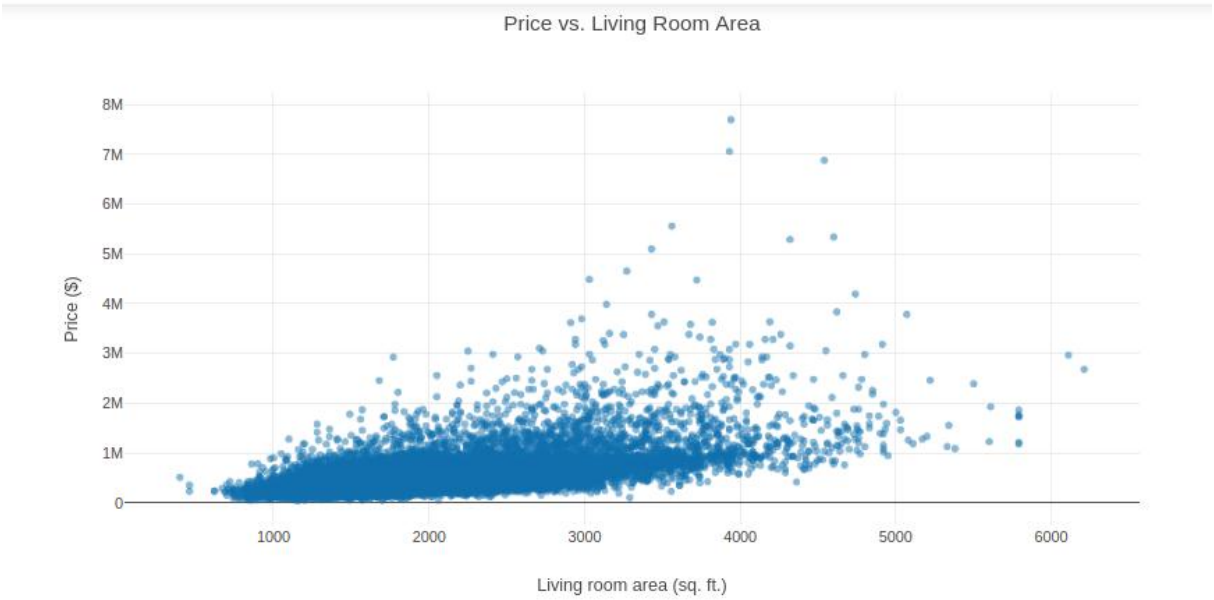


Рисунок 3.16 – Результат графіку

Для matplotlib є цілий ряд додаткових бібліотек, наприклад mplot3d для побудови трьохмірних графіків і basemap для побудови карт і проекцій.

Набір даних має гарну суміш безперервних і категоричних незалежних змінних та безперервну залежну змінну (ціну). Можна спробувати скласти ціну щодо площі вітальні (sqft_living15) (рис.3.17). Використовується scattergl замість splitter, оскільки він має кращі показники. Для менших наборів даних із меншою кількістю точок даних, розкидання має справно працювати.

```
In [12]: trace1 = go.Scattergl(
             x=houses.sqft_living15,
             y=houses.price,
             mode='markers',
             marker=dict(
                 opacity=0.5
             ),
             #showlegend=True
         )
         data=[trace1]

         layout = go.Layout(
             title='Price vs. Living Room Area',
             xaxis=dict(
                 title='Living room area (sq. ft.)'
             ),
             yaxis=dict(
                 title='Price ($)'
             ),
             hovermode='closest',
             #showlegend=True
         )

         figure = go.Figure(data=data, layout=layout)

         ply.iplot(figure)
```

Рисунок 3.17 – Код програми першого графіку

Це виглядає як приємний сюжет, який показує деяке співвідношення між площею вітальні та ціною. Співвідношення було б краще продемонстровано, якщо замість цього використати для побудови ціну.

В коді створюється макет фігури, налаштовуються її розміри, задаються значення, колір та назва, викликається метод де передаються параметри по яких малювати зміст, а також всі подальші зміни будуть оновлятись по мірі оновлення (рис.3.18). Єдиний нюанс – це внесення даних, від них суттєво залежатиме графік відображення (рис.3.19).

```
In [14]: dataPoints = go.Scattergl(
             x=houses.sqft_living15,
             y=np.log(houses.price),
             mode='markers',
             marker=dict(
                 opacity=0.25,
                 line=dict(
                     color='white'
                 )
             ),
             name='Data points'
         )

         data=[dataPoints]

         layout.update(
             yaxis=dict(
                 title='Log(Price)'
             )
         )

         figure.update(
             data=data,
             layout=layout
         )
         ply.iplot(figure)
```

Рисунок 3.18 – Код з використанням ціни

Рисунок 3.18 – Зображення графіку з використанням ціни

Намальовано графік з ціною в порівнянні з деякими особливостями і, здається, немає ідеального лінійного зв’язку між ціною та цими особливостями (рис.3.19 та рис.3.20).

```
fig = plt.figure(figsize=(19,12.5))
ax = Axes3D(fig)
ax.scatter(houses['floors'],houses['bedrooms'],houses['bathrooms'],c="darkgreen",alpha=.5)
ax.set(xlabel='\nFloors',ylabel='\nBedrooms',zlabel='\nBathrooms / Bedrooms')
ax.set(ylim=[0,12])
```

Рисунок 3.19 – Код програми з використанням лінії

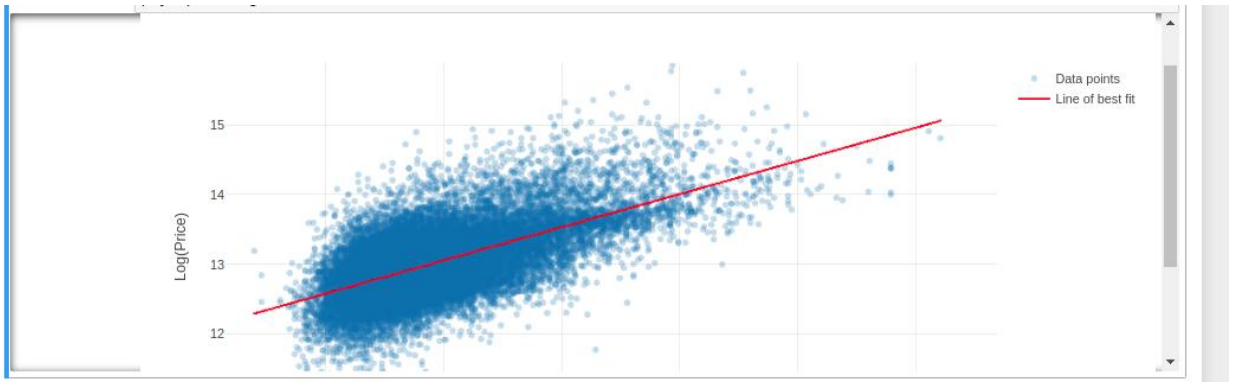


Рисунок 3.20 – Графік що відображає лінійний зв’язок

З іншого боку, а як застосувати це між собою. Щоб показати це, віддається перевага 3D-графікам. Крім того, використовується світло-зелений колір як точковий колір. Темно-зелені частини означають високу щільність (рис.3.21), багато світло-зелених точок перекриваються і стають темнішими.

На наведених нижче діаграмах видно, що при збільшенні житлової площі збільшується площа лоту та кількість поверхів або ванної / спальні. Однак кількість поверхів та ванні кімнати / спальні або житлова площа не мають подібних стосунків.

Альтернативою може стати відновлення 3D структури з урахуванням співвідношень складових частин об'єкта (рис.3.22). Найбільшу популярність в області 3D ML отримали методи глибокого навчання. Це можна пояснити тим, що для роботи більшості класичних алгоритмів машинного навчання, необхідні аналітичні функції для маніпулювання над даними. Наприклад, для того, щоб побудувати метричні алгоритми класифікації / кластеризації, потрібно ввести метрику в просторі 3D моделей.

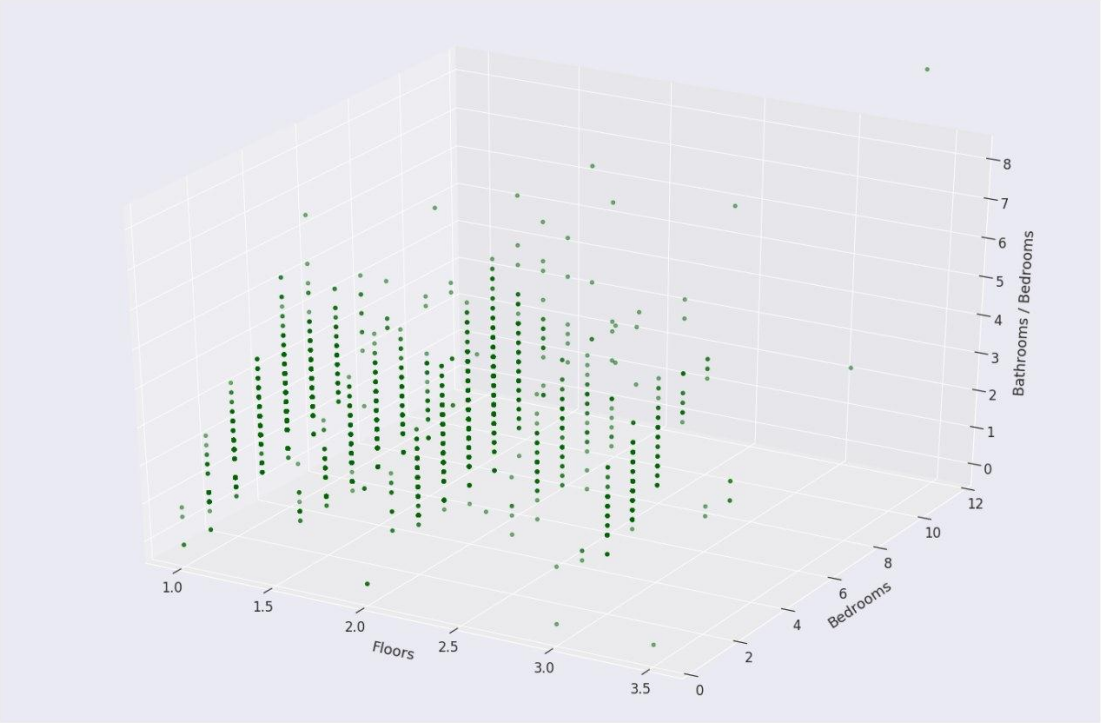


Рисунок 3.21 – 3D Графік

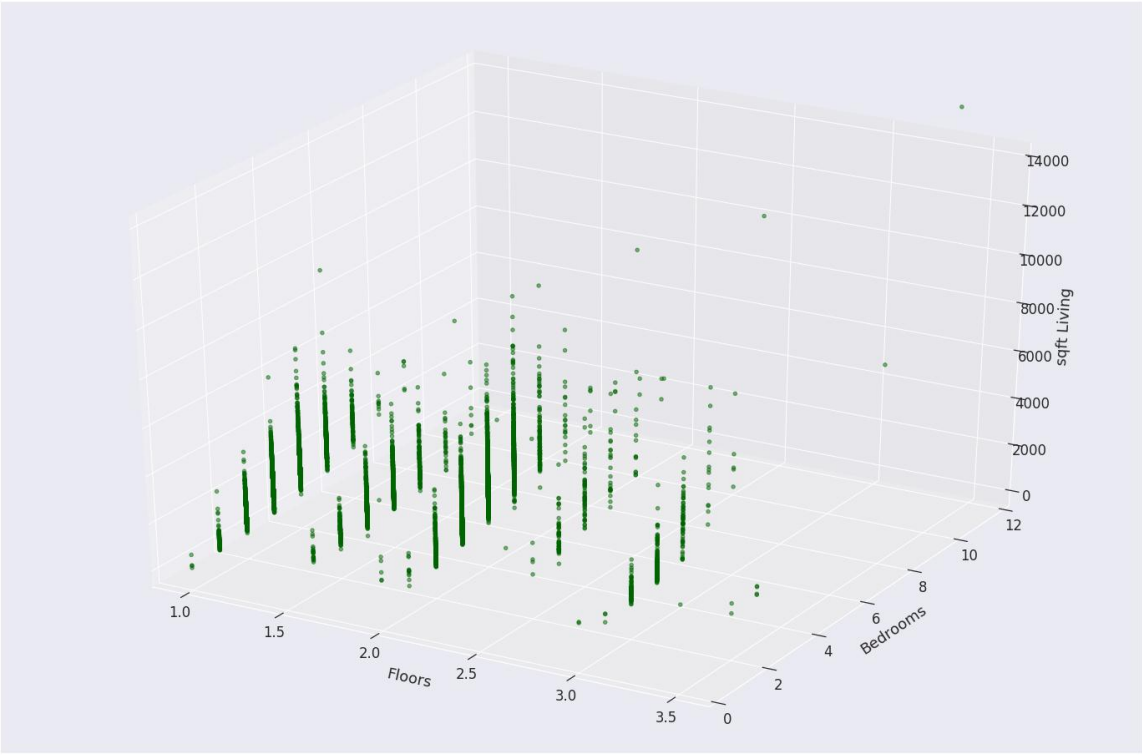


Рисунок 3.22 – 3D Графік

Також можна побачити співвідношення житлової площі до житлового лоту і спалень з ваннами (рис.3.23):

Сегментація 3D-моделі – непросте завдання сама по собі. Навіть якщо сегментація задана попередньо (рис.3.24), складно з точністю визначити місця стикування, симетрію і паралелізм в деталях зображення.

```
In [20]: fig = plt.figure(figsize=(19,12.5))
         ax=Axes3D(fig)
         ax.scatter(houses['sqft_living'],houses['sqft_lot'],houses['bathrooms'],c="darkgreen",alpha=.5)
         ax.set(xlabel='\n sqft Living',ylabel='\nsqft Lot',zlabel='\nBathrooms / Bedrooms')
         ax.set(ylim=[0,250000])

Out[20]: [(0, 250000)]
```

Рисунок 3.23 – Код програми

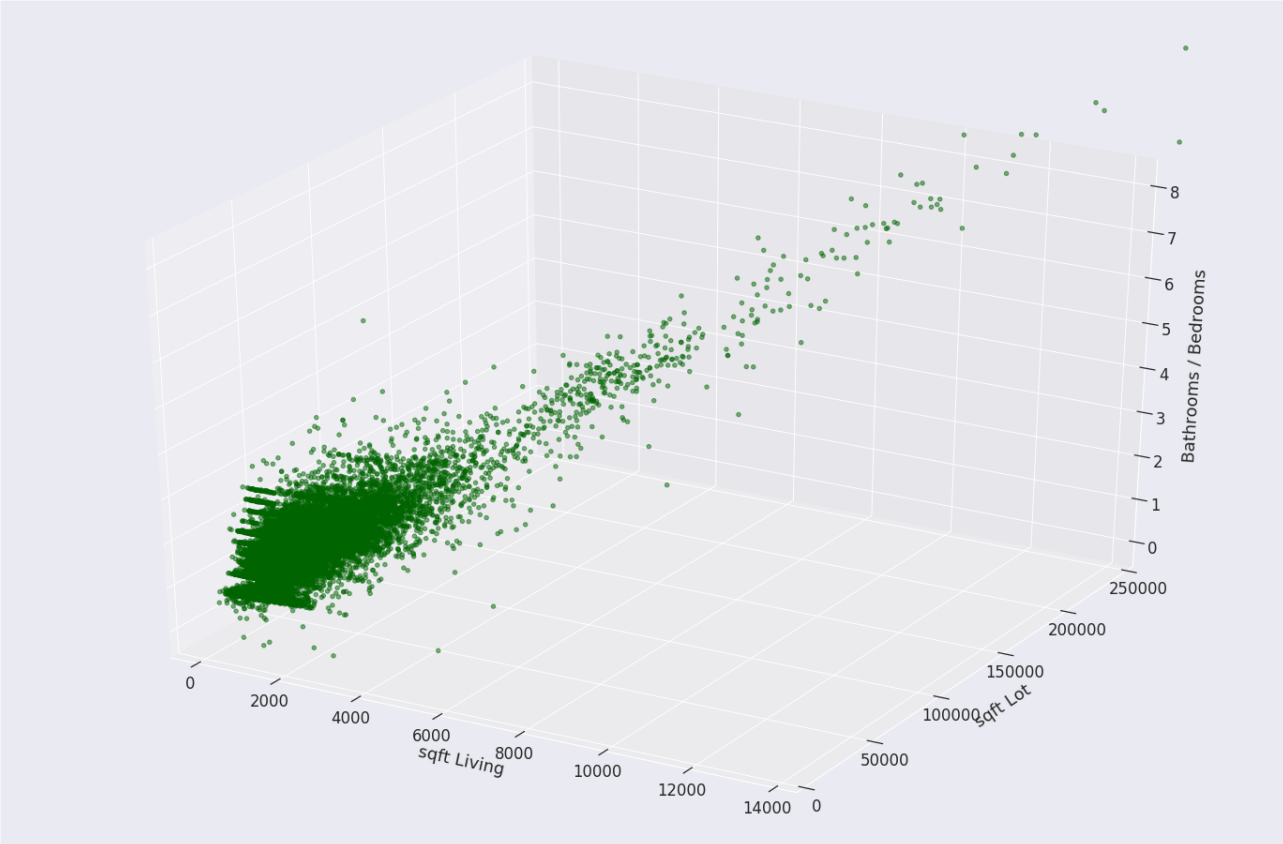


Рисунок 3.24 – 3D Графік

Дальше створено 3D-графік, щоб визначити співвідношення між видом, загальною оцінкою та роком побудови (рис.3.25). На графіку нижче показано, що нові будинки мають кращі оцінки, але про зміни в погляді неможна сказати багато (рис.3.26).

```
In [22]: fig=plt.figure(figsize=(9.5,6.25))
         ax=Axes3D(fig)
         ax.scatter(train_data['view'],train_data['grade'],train_data['yr_built'],c="darkgreen",alpha=.5)
         ax.set(xlabel='\nView',ylabel='\nGrade',zlabel='\nYear Built');
```

Рисунок 3.25 – Код програми

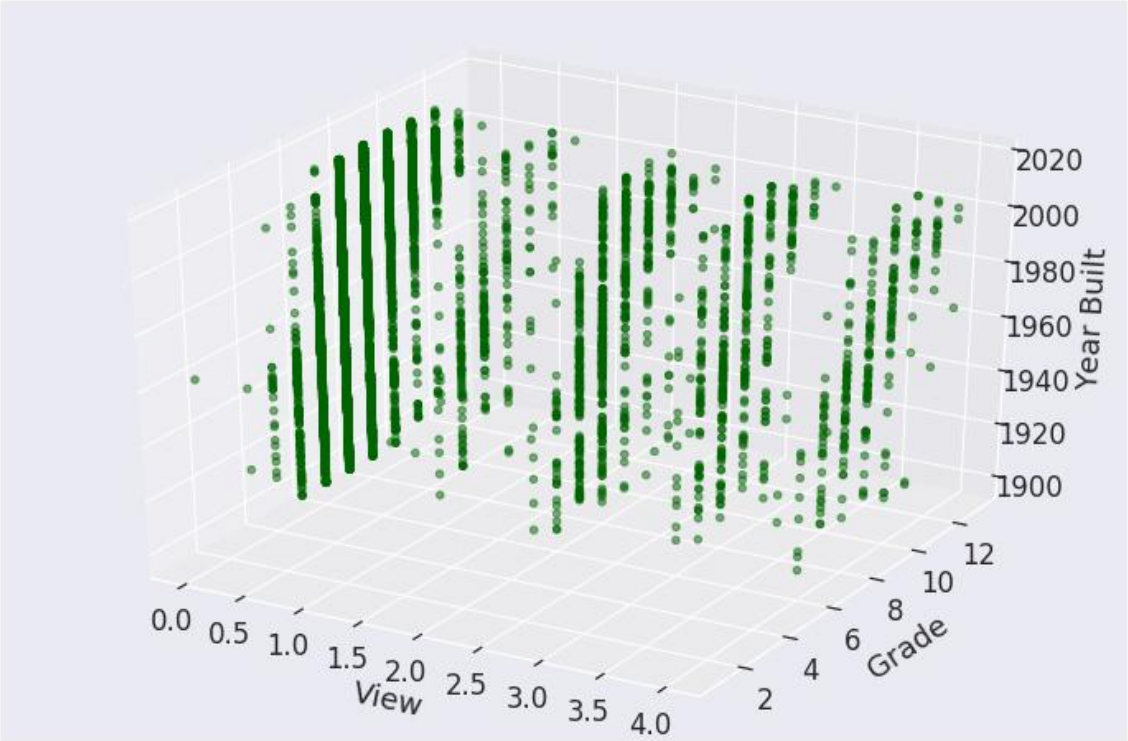


Рисунок 3.26 – 3D візуалізація

Зв’язування даних (data binning) (рис.3.29) – це метод попередньої обробки, що використовується для зменшення наслідків (рис.3.27) незначних помилок спостереження. Не варто застосувати до деяких стовпців цього набору даних. В даній моделі використовуються yr_built та yr_renovated. Додано віки та епоху ремонту будинків, коли вони продавались. Також розділено ці стовпці на інтервали, і можна це спостерігати на наведених нижче гістограмах (3.28).

```
In [24]: df_dm=houses.copy()
df_dm.describe()
# just take the year from the date column
df_dm['sales_yr']=df_dm['date'].astype(str).str[:4]

# add the age of the buildings when the houses were sold as a new column
df_dm['age']=df_dm['sales_yr'].astype(int)-df_dm['yr_built']
# add the age of the renovation when the houses were sold as a new column
df_dm['age_rnv']=0
df_dm['age_rnv']=df_dm['sales_yr'][df_dm['yr_renovated']!=0].astype(int)-df_dm['yr_renovated'][df_dm['yr_renovated'
df_dm['age_rnv'][df_dm['age_rnv'].isnull()]=0

# partition the age into bins
bins = [-2,0,5,10,25,50,75,100,100000]
labels = ['<1','1-5','6-10','11-25','26-50','51-75','76-100','>100']
df_dm['age_binned'] = pd.cut(df_dm['age'], bins=bins, labels=labels)
# partition the age_rnv into bins
bins = [-2,0,5,10,25,50,75,100000]
labels = ['<1','1-5','6-10','11-25','26-50','51-75','>75']
df_dm['age_rnv_binned'] = pd.cut(df_dm['age_rnv'], bins=bins, labels=labels)

# histograms for the binned columns
f, axes = plt.subplots(1, 2,figsize=(15,5))
p1=sb.countplot(df_dm['age_binned'],ax=axes[0])
for p in p1.patches:
    height = p.get_height()
    p1.text(p.get_x()+p.get_width()/2,height + 50,height,ha="center")
p2=sb.countplot(df_dm['age_rnv_binned'],ax=axes[1])
```

Рисунок 3.27 – Код програми з зв’язуванням даних(Початок)

```
axes[0].set(xlabel='Age')
axes[0].yaxis.tick_left()
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
axes[1].set(xlabel='Renovation Age');

# transform the factor values to be able to use in the model
df_dm = pd.get_dummies(df_dm, columns=['age_binned','age_rnv_binned'])
```

Рисунок 3.28 – Код програми з зв’язуванням даних (Кінець)

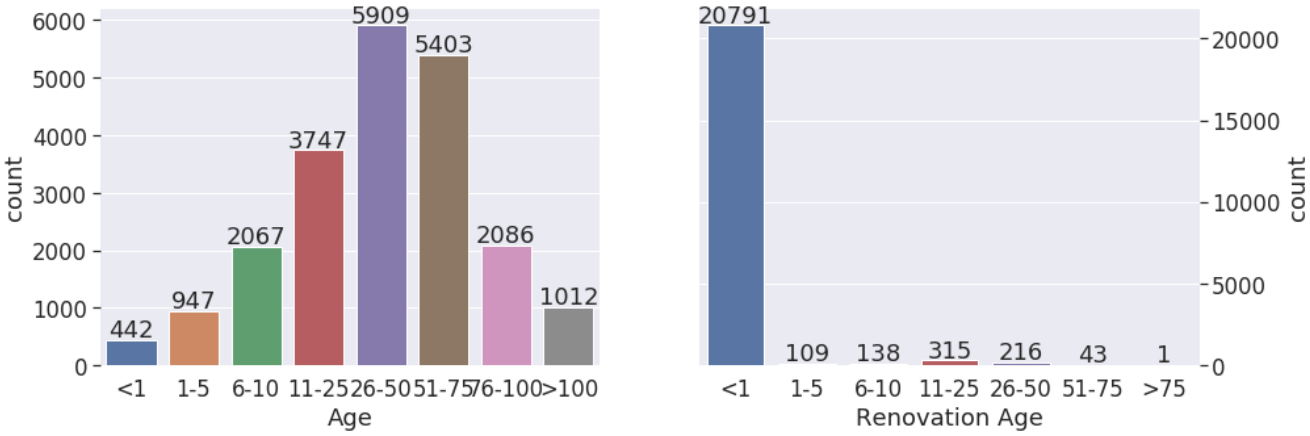


Рисунок 3.29 – Візуалізація зв’язування даних

3.4 Локалізація даних з Folium

Створюється веб-карта, використовуючи Python та Folium.

Folium – це Python-бібліотека для візуалізації географічних даних та інформації, яка містить координати та місцеположення. Детальний опис функцій можна знайти на офіційному веб-сайті проєкту.

Для встановлення бібліотеки використовується pip - систему управління пакетами та бібліотеками.

Примітка: після виконання команд map.save () карта зберігається в поточному робочому каталозі.

Маркер, вказаний на карті, є кращою важливою штукою – без нього орієнтуватися було значно важче. Додається маркер на карту із підтримкою folium.Marker, який пропонує місце, де потрібно встановити маркер, задати зовнішній координати та вигляд вікна та інших користувачів.

У цьому наборі даних є інформація про широту та довготу (рис.3.30) для будинків. Використовуючи lat та довгі стовпці, визначається нижня теплова карта. Крім того, можна вважати, що стовпчик з поштовим індексом дає схожу інформацію про місцеположення (рис.3.31).

```
In [23]: # find the row of the house which has the highest price
maxpr=houses.loc[houses['price'].idxmax()]

# define a function to draw a basemap easily
def generateBaseMap(default_location=[47.5112, -122.257], default_zoom_start=9.4):
    base_map = folium.Map(location=default_location, control_scale=True, zoom_start=default_zoom_start)
    return base_map

df_copy = houses.copy()
# select a zipcode for the heatmap
#set(df['zipcode'])
#df_copy = df[df['zipcode']==98001].copy()
df_copy['count'] = 1
basemap = generateBaseMap()
# add carton position map
folium.TileLayer('cartodbpositron').add_to(basemap)
s=folium.FeatureGroup(name='icon').add_to(basemap)
# add a marker for the house which has the highest price
folium.Marker([maxpr['lat'], maxpr['long']],popup='Highest Price: $'+str(format(maxpr['price'],'.0f')),
              icon=folium.Icon(color='green')).add_to(s)
# add heatmap
HeatMap(data=df_copy[['lat','long','count']].groupby(['lat','long']).sum().reset_index().values.tolist(),
        radius=8,max_zoom=13,name='Heat Map').add_to(basemap)
folium.LayerControl(collapsed=False).add_to(basemap)
basemap
```

Рисунок 3.30 – Код програми для підключення карти

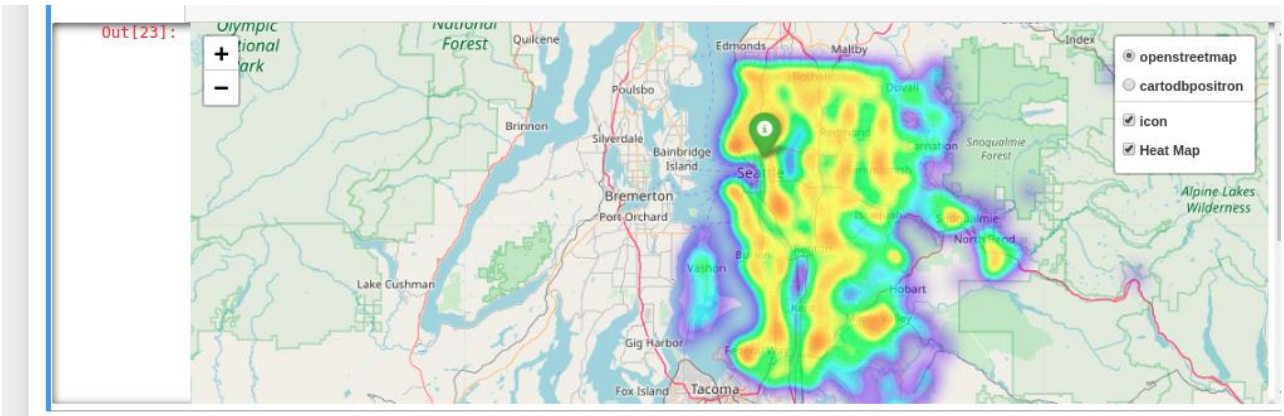


Рисунок 3.31 – Зображення на карті

За аналізами просторових даних, найкраще використовувати всі карти.

Folium – це бібліотека Python, за допомогою якої можна створити кілька типових карт. З її допомогою можна з легкістю згенерувати карту штату, створивши об’єкт Folium Map. Використовуючи розташування аргументу, можна центрирувати карту у визначеному місці (в даному випадку будинку для округу Кінг, Сіетлі). У цьому місці також можна забезпечити початковий рівень масштабування для збільшення карти в центрі.

Сгенерована інтерактивна карта (можна збільшувати і зменшувати масштаб).

Однією з особливостей Folium є можливість створення анімованих теплових карт, що змінюють відображаються дані на основі певного виміру (наприклад, цін). Для створення такої карти також можна використовувати метод класу HeatMapWithTime ().

Не менш привабливим є те, ща на маркері (рис.3.32) можна побачити детальну інформацію (в даному прикладі ціну).

Рисунок 3.32 – Відображення на маркері детальної інформації

3.5 Кластеризація даних

Кластерний аналіз – задача розбиття заданої вибірки об'єктів (ситуацій) на підмножини, які називаються кластерами, так, щоб кожен кластер складався з схожих об’єктів, а об’єкти різних кластерів істотно відрізнялися. Завдання кластеризації відноситься до статистичної обробки, а також до широкого класу завдань навчання без вчителя.

Кластерний аналіз – це не якийсь один алгоритм, а загальна задача, для розв’язання якої використовуються різні підходи. Зокрема, алгоритми побудови кластерів можуть суттєво відрізнятись у розумінні того, що відносити в один кластер і як їх ефективно шукати. Серед популярних концепцій кластерів є групи з елементами, які утворюються ґрунтуючись на відстані між ними, на щільності ділянок у просторі даних, інтервалах або на конкретних статистичних розподілах. Тому кластеризація може бути сформульована як задача багатокритеріальної оптимізації. Відповідний алгоритм кластеризації та вибору параметрів (включаючи такі параметри, як функція відстані, порогове значення щільності або кількість очікуваних кластерів) залежать від конкретного набору даних та мети використання результатів. Кластерний аналіз як такий є не автоматизованим завданням, а ітераційним процесом виявлення знань або інтерактивної багатокритеріальної оптимізації, який містить спроби та невдачі. Часто доводиться змінювати процес опрацювання даних та параметри моделі поки не буде отримано з результат з заданими властивостями.

Окрім терміну кластеризація існує багато термінів з аналогічним значенням, серед яких автоматична класифікація, числова таксономія та типологічний аналіз. Тонкі розбіжності часто полягають у використанні результатів: для добування даних, отримані групи є предметом інтересу, при автоматичній класифікації, навпаки, більш важливий степінь розбіжності.

Кластерний аналіз походить з антропології, де він був започаткований Драйвером (англ. Driver) і Крьобером (англ. Kroeber) у 1932 році. В психологію він був введений Зубіним у 1938 році і Робертом Тріоном у 1939. Став відомий завдяки використанню Кеттелем для класифікації теорії ознак в психології особистості, починаючи з 1943 року.

Кластерний аналіз – це багатовимірна статистична процедура, яка виконує збір даних, що містять інформацію про вибірку об'єктів і потім упорядковує об’єкти в порівняно однорідні групи – кластери (Q-кластеризація, або Q-техніка, власне кластерний аналіз).

Основна мета кластерного аналізу – знаходження груп схожих об'єктів у вибірці. Спектр застосувань кластерного аналізу дуже широкий: його використовують в археології, антропології, медицині, психології, хімії, біології, державному управлінні, філології, маркетингу, соціології та інших дисциплінах. Однак універсальність застосування привела до появи великої кількості несумісних термінів, методів і підходів, що ускладнюють однозначне використання і несуперечливу інтерпретацію кластерного аналізу.

Незалежно від конкретної сфери, застосування кластерного аналізу передбачає наступні етапи:

1. Відбір вибірки для кластеризації.

2. Визначення множини характеристик, по яких будуть оцінюватися об'єкти у вибірці.

3. Обчислення значень тієї чи іншої міри схожості між об'єктами.

4. Застосування одного з методів кластерного аналізу для створення груп схожих об'єктів.

5. Перевірка достовірності результатів кластеризації.

Якщо кластерному аналізу передує факторний аналіз, то вибірка не потребує коректування – викладені вимоги виконуються автоматично самою процедурою факторного моделювання. В іншому випадку вибірку потрібно коректувати.

Оскільки поняття “кластеру” не може бути точно визначено, то це є однією з причин чому існує так багато різних методів кластеризації. Але є і спільна риса – це об’єднання схожих об'єктів у групи. Однак, різні дослідники використовують різні моделі кластерів і для кожної з цих моделей можуть бути застосовані різні алгоритми. Поняття кластера, які отримуються у різних алгоритмах, різняться властивостями. Розуміння цих “кластерних моделей” є ключовим для розуміння відмінностей між різними алгоритмами. Типовими кластерними моделями є:

Моделі зв’язності. Наприклад, ієрархічна кластеризація або таксономія будуються на основі відстані між вузлами.

Центроїдні моделі. Наприклад, метод K-середніх (K-means) представляє кожен кластер єдиним усередненим вектором.

Статистичні моделі. Кластери будуються ґрунтуючись на статистичних розподілах. Таких як багатовимірний нормальний розподіл з допомогою EM-алгоритму.

Моделі засновані на щільності. Наприклад, в DBSCAN і в OPTICS кластери визначаються як зв’язані області відповідної щільності у просторі даних.

Групові моделі. Деякі алгоритми не забезпечують вдосконалену модель для своїх результатів, а просто описують групування об’єктів.

Графові моделі. Поняття кліки (така підмножина вершин, в якій кожна пара вершин з’єднана ребром) у графі слугує прототипом кластеру. Пом’якшення вимоги до повної зв’язності (тобто, частина ребер може бути відсутня) призводить до поняття відомого як квазі-кліка. Вони будуються алгоритмом HCS.

Нейронні моделі. Найбільш відомою моделлю нейронної мережі з навчанням без учителя є нейронна мережа Кохонена. Ці моделі, як правило, можна охарактеризувати як схожі на одну або подібні якійсь з наведених вище моделей, включаючи моделі у підпросторах, коли нейронні мережі реалізують метод головних компонент або аналіз незалежних компонент.

“Кластеризацією” зазвичай вважають такий набір кластерів, які містять усі об'єкти набору даних. Додатково, можна розглянути відношення між кластерами. Наприклад, ієрархію вкладеності кластерів один у одного. Грубо можна виділити такі кластеризації:

Жорстка кластеризація. Кожен об'єкт або належить кластеру або ні.

М’яка кластеризація (також нечітка кластеризація). Кожен об’єкт належить кожному кластеру до певної міри. Наприклад, це ймовірність належності кластеру.

Серед них виділяють декілька доладних:

Жорстке розбиття на кластери. Кожен об’єкт належить рівно одному кластеру.

Жорстке розбиття на кластери з викидами. Об’єкт може не належати жодному кластеру і розглядається як викид.

Кластери з перетином. Об’єкт може належати більш ніж одному кластеру.

Ієрархічна кластеризація. Якщо об'єкт належить нащадку, то він також належить і предку.

Підпросторова кластеризація. Хоч кластери і можуть перетинатись, проте в межах визначеного підпростору кластери не перетинаються.

1. Кластерний аналіз висуває наступні вимоги до даних.

2. Об’єкти не повинні корелювати між собою.

3. Об’єкти мають бути безрозмірними.

4. Розподіл об'єктів має бути близьким до нормального.

5. Об'єкти повинні відповідати вимозі стійкості, під якою розуміється відсутність впливу на їх значення випадкових чинників.

6. Вибірка повинна бути однорідна.

7. Інтерпретація результатів

Результатом кластеризації є групи об’єктів, об'єднані за певною характеристикою чи характеристиками. Однак ці результати можуть бути інтерпретовані по-різному. Зокрема, при аналізі результатів соціологічних досліджень рекомендується здійснювати аналіз ієрархічними методами, наприклад методом Уорда, при якому всередині кластерів оптимізується мінімальна дисперсія і в результаті створюються кластери приблизно рівних розмірів. Як міра відмінності між кластерами використовується квадратична евклідова відстань, що сприяє збільшенню контрастності кластерів.

Тепер виникає питання стійкості знайденого кластерного рішення. По суті, перевірка стійкості кластеризації зводиться до перевірки її достовірності. Тут існує емпіричне правило – стійка типологія зберігається при зміні методів кластеризації. Результати ієрархічного кластерного аналізу можна перевіряти ітеративним кластерним аналізом методом k-середніх. Якщо при порівнянні групи збігаються більше, ніж на 70 % (понад 2/3 збігів), то кластерне рішення приймається.

Перевірити адекватність рішення, не вдаючись до допомоги інших видів аналізу, не можна. Принаймні, в теоретичному плані ця проблема не вирішена. Деякі додаткові методи перевірки стійкості відкидаються з певних причин:

Кофенетична кореляція – не рекомендується і обмежена у використанні.

Тести значущості (дисперсійний аналіз) – завжди дають значущий результат.

Метод повторних випадкових вибірок – не доводить правильність рішення.

Тести значущості для зовнішніх ознак – придатні тільки для повторних вимірювань.

Методи Монте-Карло – дуже складні і доступні тільки досвідченим математикам.

Кластериза́ція ме́тодом k-сере́дніх (англ. k-means clustering) – популярний метод кластеризації, – впорядкування множини об'єктів в порівняно однорідні групи. Винайдений в 1950-х роках математиком Гуґо Штайнгаузом і майже одночасно Стюартом Ллойдом. Особливу популярність отримав після виходу роботи МакКвіна.

Мета методу – розділити n спостережень на k кластерів, так щоб кожне спостереження належало до кластера з найближчим до нього середнім значенням. Метод базується на мінімізації суми квадратів відстаней між кожним спостереженням та центром його кластера.

Маємо масив спостережень (об’єктів), кожен з яких має певні значення по ряду ознак. Відповідно до цих значень об'єкт розташовується у багатовимірному просторі.

Дослідник визначає кількість кластерів, що необхідно утворити.

Випадковим чином обирається k спостережень, які на цьому кроці вважаються центрами кластерів.

Кожне спостереження “приписується” до одного з n кластерів – того, відстань до якого найкоротша.

Розраховується новий центр кожного кластера як елемент, ознаки якого розраховуються як середнє арифметичне ознак об’єктів, що входять у цей кластер

Відбувається така кількість ітерацій (повторюються кроки 3-4), поки кластерні центри стануть стійкими (тобто при кожній ітерації в кожному кластері опинятимуться одні й ті самі об’єктb), дисперсія всередині кластера буде мінімізована, а між кластерами – максимізована.

Алгоритм методу “Кластеризація за схемою к-середніх”:

1. Вибрати k інформаційних точок як центри кластерів поки не завершиться процес зміни центрів кластерів.

2. Зіставити кожну інформаційну точку з кластером, відстань до центра якого мінімальна.

3. Переконатися, що в кожному кластері міститься хоча б одна точка. Для цього кожний порожній кластер потрібно доповнити довільною точкою, що розташована “далеко” від центра кластера.

4. Центр кожного кластера замінити середнім від елементів кластера.

5. Кінець.

Головні переваги методу k-середніх – його простота та швидкість виконання. Метод k-середніх більш зручний для кластеризації великої кількості спостережень, ніж метод ієрархічного кластерного аналізу (у якому дендограми стають перевантаженими і втрачають наочність).

Одним із недоліків простого методу є порушення умови зв'язності елементів одного кластера, тому розвиваються різні модифікації методу, а також його нечіткі аналоги (англ. fuzzy k-means methods), у яких на першій стадії алгоритму допускається приналежність одного елемента множини до декількох кластерів (із різним ступенем приналежності).

Незважаючи на очевидні переваги методу, він має суттєві недоліки:

1. Результат класифікації сильно залежить від випадкових початкових позицій кластерних центрів.

2. Алгоритм чутливий до викидів, які можуть викривлювати середнє.

3. Кількість кластерів повинна бути заздалегідь визначена дослідником.

Метод k-середніх є доволі простим і прозорим, тому успішно використовується у різноманітних сферах – маркетингових сегментаціях, геостатистиці, астрономії, сільському господарстві тощо.

В прикладі відображається знаходження віку (року) будинку починаючи з дати ремонту якщо будинок не ремонтували – починаючи з року будівництва.

По аналізі даних, yr_renovated має 0 значень, тож можна замінити їх значеннями yr_build і за формулою отримається вік (рис.3.33):

Age = yr_renovated – current_year

В коді створюється новий фрейм і, циклом перебераються дані та записуються:

```
In [4]: current_year = datetime.now().year
        for i, x in frame[['yr_built']].itertuples():
            if(frame.at[i, 'yr_renovated'] == 0):
                frame.at[i, 'yr_renovated'] = x
            frame.at[i, 'age'] = current_year - frame.at[i, 'yr_renovated']
```

Рисунок 3.33 – Код реалізації знаходження віку будинків

Знаходження відстані: Використовуються координати довготи і широти та підраховується відстань від центру Сієтла

Висновок – чим будинок далі від центру – тим ціна на будинок нижча.

Обчислюється велика відстань кола між двома точками (рис.3.34) на землі (вказано в десяткових градусах)

long – Довгота

lat – Широта

r = 6371 – радіус Землі в кілометрах

```
In [5]: def haversine(lon2, lat2):
            # Seattle center coordinates
            centerLon = -122.352033
            centerLat = 47.622451

            # convert decimal degrees to radians
            lon1, lat1, lon2, lat2 = map(radians, [centerLon, centerLat, lon2, lat2])

            # haversine formula
            dlon = lon2 - lon1
            dlat = lat2 - lat1
            a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
            c = 2 * asin(sqrt(a))
            r = 6371 # Radius of earth in kilometers, 3956 for miles
            return c * r

In [6]: for i, lat, long in frame[['lat','long']].itertuples():
            frame.at[i,'distance'] = haversine(long, lat)
```

Рисунок 3.34 – Код знаходження відстані

Знаходження середньої площі: Якщо будинок має середні квадратні метражі по кімнаті чи поверху n, (рис.3.35) цей будинок може бути гіршим, ніж будинок із середнім квадратним метром m, де m > n.

```
In [7]: for i, f, a, b, c, d in frame[['floors','bedrooms','bathrooms','sqft_above','sqft_basement']].itertuples():
            frame.at[i,'average_sqft_by_room'] = ((c+d)/((a+b)+1))#+1 because house can be without bedrooms or bathrooms
            frame.at[i,'average_sqft_by_floor'] = ((c+d)/f)
```

Рисунок 3.35 – Код знаходження Середньої площі

Середня площа підвалів: Є поле sqft_basement – інформація про підвал будинку, також дано кількість поверхів будинку (рис.3.36), можна помножити ці параметри та отримати будинок sqft для всіх поверхів

Припущення: Чим більше поверхів, тим найбільша ціна будинку

```
In [8]: for i, f, sqft in frame[['floors','sqft_basement']].itertuples():
            frame.at[i,'sqft_floors_mult_basement'] = f*sqft
```

Рисунок 3.36 – Код для знаходження середньої площі підвалів

Не ефективно конвертувати дату в мілі, секунди або дні, тому що загалом ціни на житло не можуть змінюватися щосекунди або кожного дня. Кожен рік – занадто великий, а кожен місяць (рис.3.37) – досить непогано.

```
In [9]: frame['date'] = pd.to_datetime(frame['date'])
        for i, d in frame[['date']].itertuples():
            frame.at[i, 'mnths'] = (d.year - 2014)*12 + d.month

        #Delete useless field
        del frame['date']
```

Рисунок 3.37 – Код програми

Не можна створити регресію за об’єктом дати продаж бо цього не вказано в базі. Різниця в площі: Створена різниця між sqft за 15 років та sqft (рис.3.38).

```
In [10]: #Convert difference between sqft_living15 and sqft_living to ren_living_diff - sqft difference after renovation
         for i, l, lo, l15, lo15 in frame[['sqft_living','sqft_lot','sqft_living15','sqft_lot15']].itertuples():
             frame.at[i,'ren_living_diff'] = l15-l
             frame.at[i,'ren_lot_diff'] = lo15-lo
```

Рисунок 3.38 – Код знаходження різниці

Сіетл має 7 районів (рис.3.42 та рис.3.43). Отже, розділяються партії на сім кластерів за координатами. По-перше, розділюється на кластери за місцем розташування і ділиться за ціною та відстанню, часто, може бути співвідношення цін на відстань від центру (рис.3.41 та рис.3.39).

```
In [11]: # Simple points visualization by x and y coordinates
         def draw(x, y):
             data = frame[[x,y]].values
             plt.scatter(data[:, 0], data[:, 1], s=5, alpha=.4)
             plt.xlabel(x)
             plt.ylabel(y)
             plt.show()
             return data
```

```
In [12]:  #Distance by price visualization
         dist_price = draw('price','distance')
```

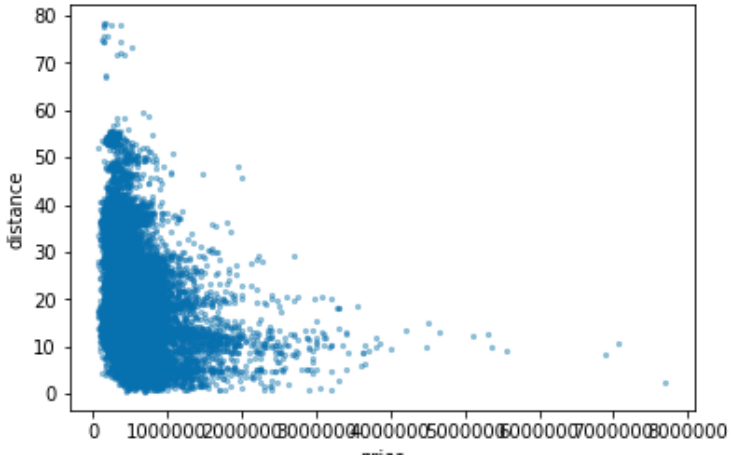


Рисунок 3.39 – Реалізація та вивід результату

```
In [13]: # Train and centers initializing
         k_means_district = KMeans(n_clusters=7)
         k_means_district.fit(dist_price)
         centers = k_means_district.cluster_centers_
```

```
In [14]: # Districts clustering by price visualization
         plt.xlabel("price")
         plt.ylabel("distance")
         plt.scatter(dist_price[:, 0], dist_price[:, 1], c=k_means_district.predict(dist_price), s=10, alpha=1)
         plt.scatter(centers[:, 0], centers[:, 1], c='red', s=30, alpha=1)
```

Рисунок 3.40 – Код візуалізації по кластерам

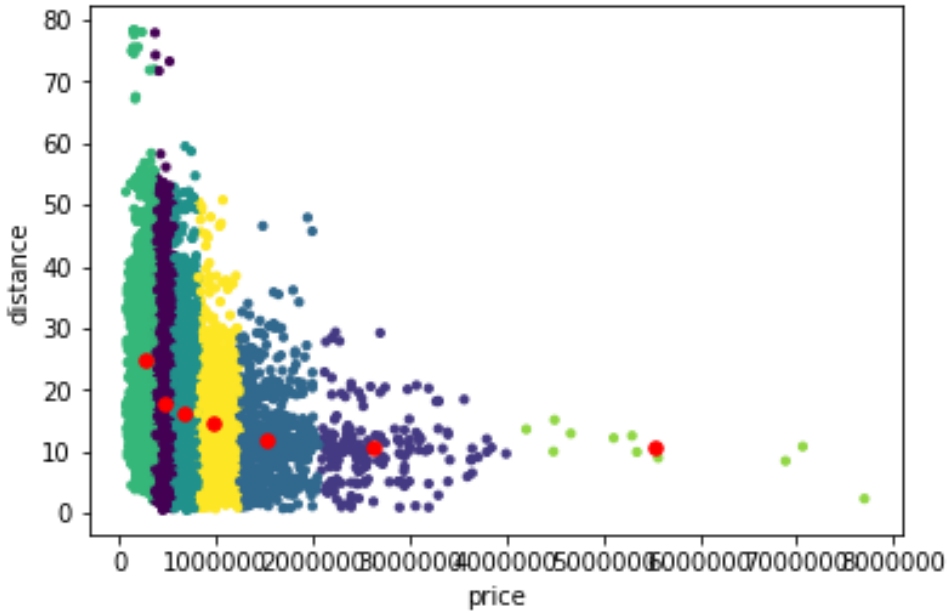


Рисунок 3.41 – Графік розбиття на кластери

```
In [15]: # Splitting lots into districts
distance_clusters = k_means_district.predict(dist_price)
for i, d in frame[['distance']].itertuples():
    frame.at[i,'price_district'] = distance_clusters[i]

pd.DataFrame({
    'price': frame['price'],
    'price_district': frame['price_district']
    }).groupby('price_district').sum().plot()
```

Рисунок 3.42 – Код програми

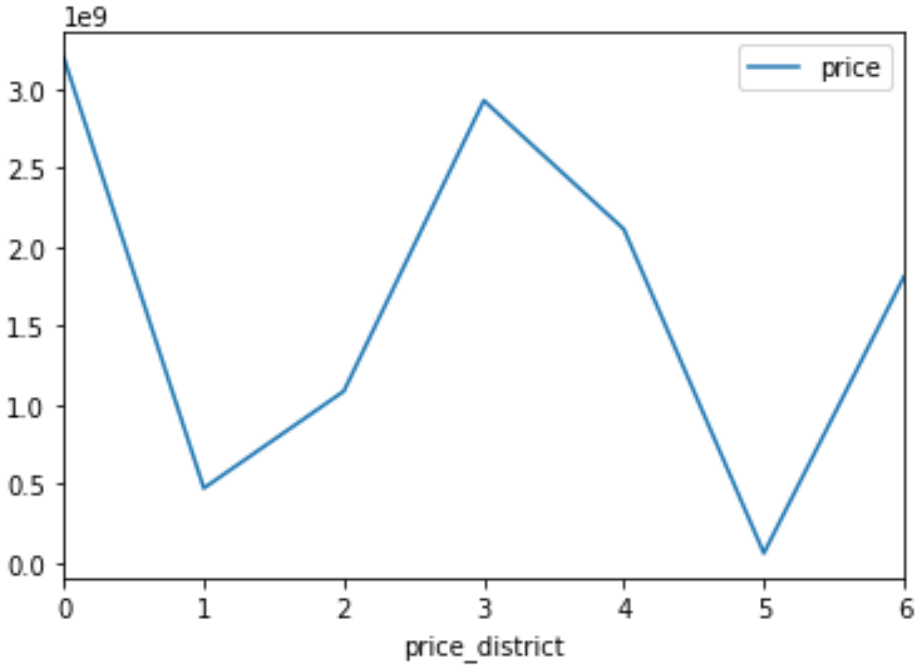


Рисунок 3.43 – Графік співвідношення ціни та районів

```
In [16]: # Coordinates visualization
         coordinates = draw('long','lat')
```

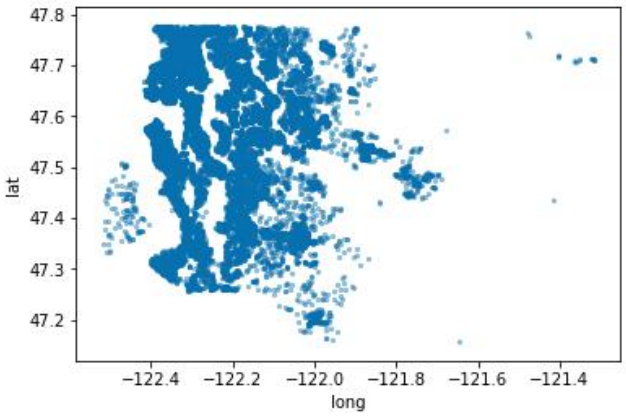


Рисунок 3.44 – Графік відображення карти

```
In [17]: # Train and centers initialization
         k_means_district.fit(coordinates)
         centers = k_means_district.cluster_centers_

In [18]: plt.xlabel("long")
         plt.ylabel("lat")
         plt.scatter(coordinates[:, 0], coordinates[:, 1], c=k_means_district.predict(coordinates), s=10, alpha=1)
         plt.scatter(centers[:, 0], centers[:, 1], c='red', s=10, alpha=1)
```

Рисунок 3.45 – Код реалізації кластеризації на карті

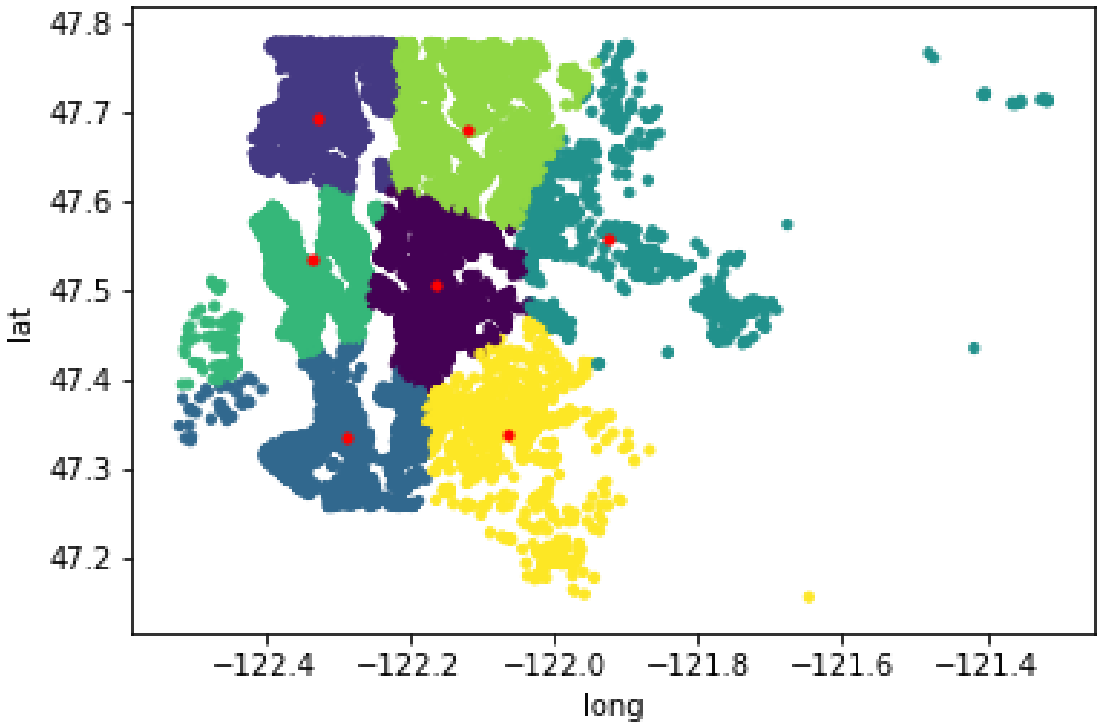


Рисунок 3.46 – Кластери на карті

```
In [21]:  #Train and centers initializing
k_means_neighborhoods = KMeans(n_clusters=127)
k_means_neighborhoods.fit(dist_price)
centers = k_means_neighborhoods.cluster_centers_

In [22]: # Neigborhoods clustering by price visualization
plt.xlabel("price")
plt.ylabel("distance")
plt.scatter(dist_price[:, 0], dist_price[:, 1], c=k_means_neighborhoods.predict(dist_price), s=10, alpha=1)
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=10, alpha=1)
```

Рисунок 3.47 – Код програми

Також Сіетл має 127 мікрорайонів (рис.3.47). Отже, розділяються координати на 127 кластерів, тому що кластеризація по мікрорайонах може дати більше точності, ніж кластеризація по районах (рис.3.47).

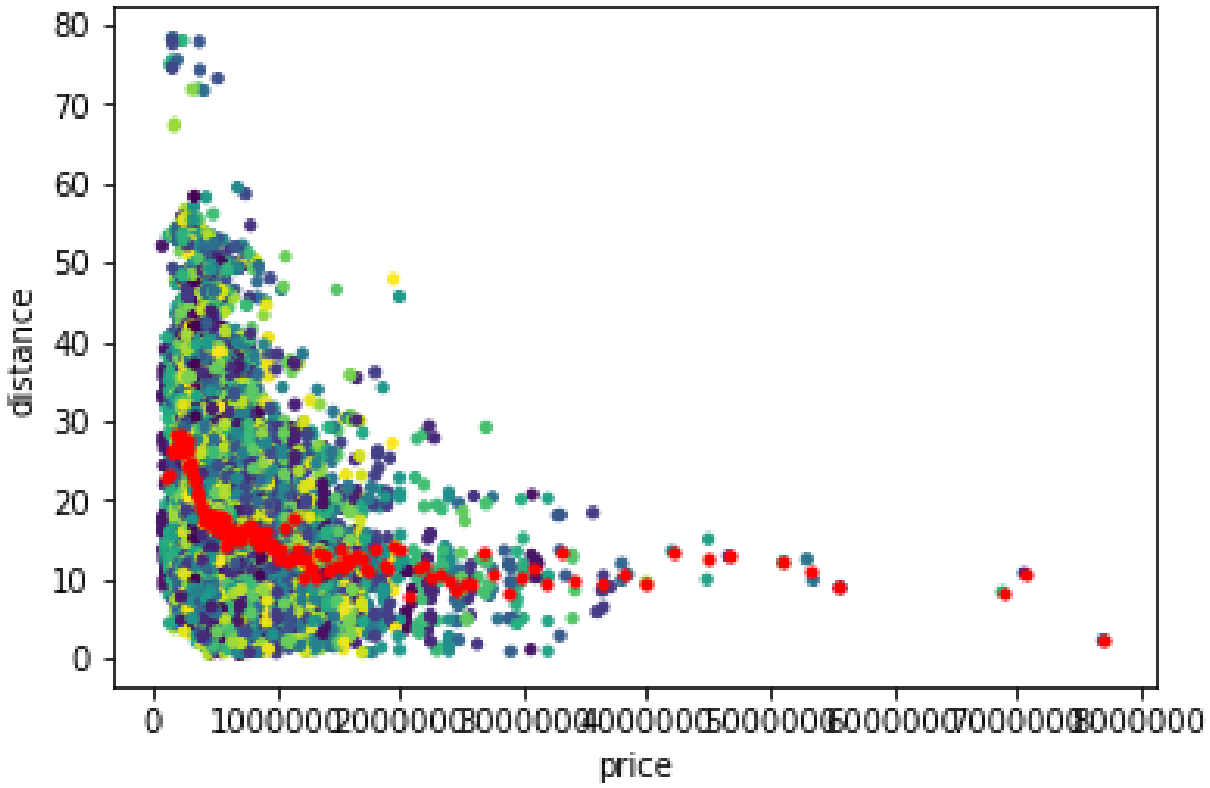


Рисунок 3.48 – Графік представлення

Якщо конвеєр кластеризації вийшов занадто повільним, швидкість можна підвищити, переключившись з модуля nltk.cluster на реалізацію MiniBatchKMeans з sklearn. cluster.

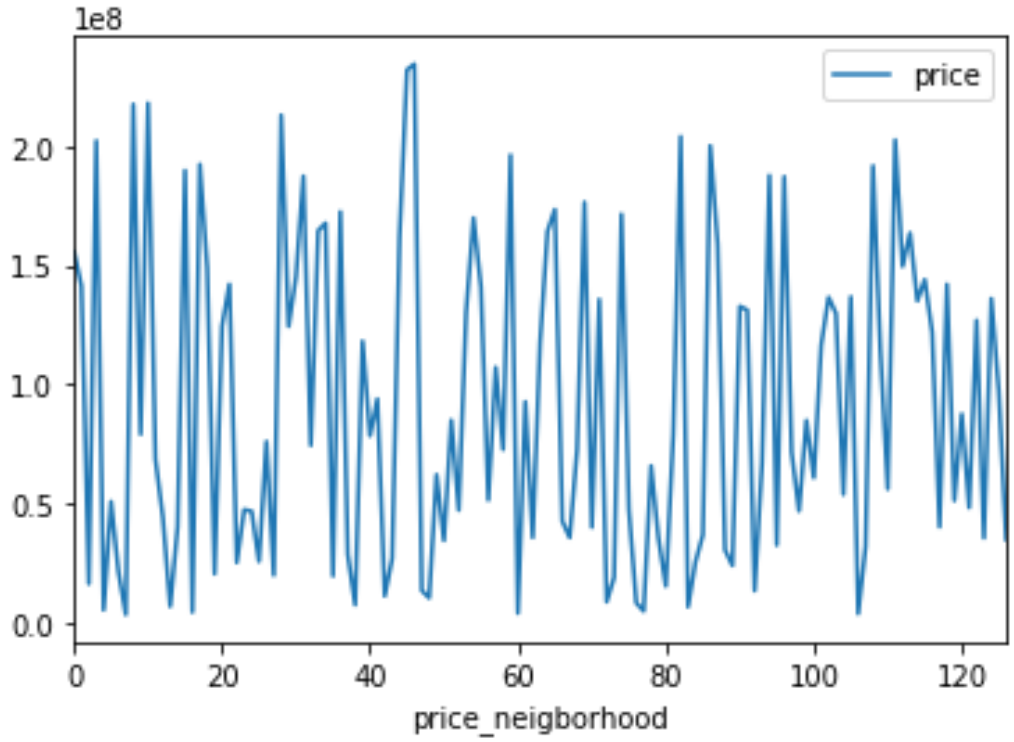


Рисунок 3.49 – Результат кластеризації по мікрорайонах

На рисунках чітко зображено поділ на кластери та результат (рис.3.49) кластеризації по мікрорайонах у співвідношення до ціни(рис.3.50 та рис.3.51).

```
In [23]: # Splitting lots into neigborhoods
distance_clusters = k_means_neighborhoods.predict(dist_price)
for i, d in frame[['distance']].itertuples():
    frame.at[i,'price_neigborhood'] = distance_clusters[i]

#Visualisation
pd.DataFrame({
    'price': frame['price'],
    'price_neigborhood': frame['price_neigborhood']
    }).groupby('price_neigborhood').sum().plot()
```

Рисунок 3.50 – Код програми

```
In [24]: # Train and centers initialization
k_means_neighborhoods.fit(coordinates)
centers = k_means_neighborhoods.cluster_centers_

In [25]: plt.xlabel("long")
plt.ylabel("lat")
plt.scatter(coordinates[:, 0], coordinates[:, 1], c=k_means_neighborhoods.predict(coordinates), s=10, alpha=1)
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=10, alpha=1)
```

Рисунок 3.51 – Код кластеризації по мікрорайонах на карті

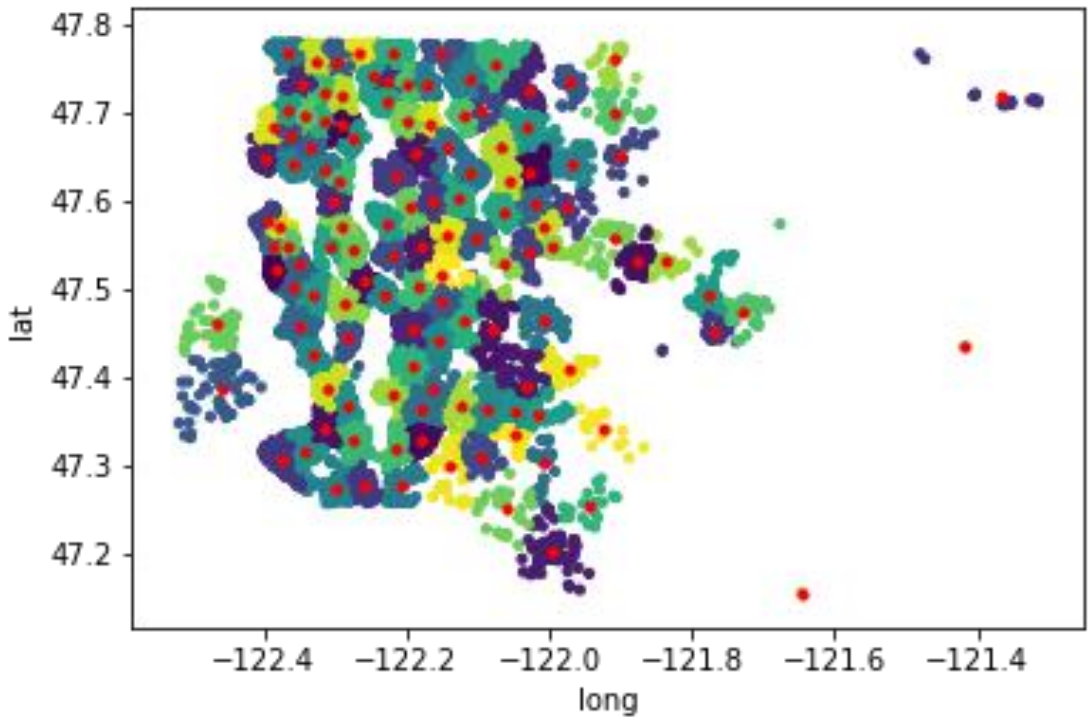


Рисунок 3.52 – Кластеризація по мікрорайонах

Як можна “поліпшити” модель (рис.3.52) кластеризації – у цьому випадку під цим розуміється: як зробити результати більш корисними і простими в інтерпретації. Спростити інтерпретацію можна, експериментуючи з різними значеннями k. У кластеризації методом k-середніх вибір k – часто ітеративний процес; незважаючи на існування деяких емпіричних правил, початковий вибір часто робиться довільно до певної степені. Мається на увазі, що алгоритм k-середніх не є легковажним і може працювати досить повільно на даних з великою кількістю вимірювань, таких як текстові дані

MiniBatchKMeans – це варіант алгоритму k-середніх, який використовує випадково вибрані підмножини (Або “міні-пакети”) з усього навчального набору для оптимізації тієї ж цільової функції, але за менший час.

3.6 Прогнозування реальних даних

Прийшов час протестувати модель на реальних даних. Нехай для прикладу використовується один з відомих сайтів продажу будинків.

Zillow – найпростіший спосіб знайти будинки для продажу та квартири в оренду. Сайт що дозволяє отримати детальну інформацію про об’єкт, про оренду та викупи, які не можна знайти в MLS. Купувати будинки біля пляжу, шукати нерухомість у центрі міста або знайти поблизу школи чи гімназії. Тому тестування на реальних даних буде використовувати цей сайт (рис.3.53 та рис.3.54).

Спершу в логіці програми розбивається на продані будинки, та ці що ще стоять на продажі для кращого тестування моделі. Забивається координати будь яких будинків, а також ціну за яку їх продали чи та яка зараз стоїть актуальна.

```
In [35]: # Predicting real life data (real examples)

rl_X=pd.DataFrame(columns=["sqft_living","sqft_lot","yr_built","bedrooms","bathrooms","lat","long","waterfront"])
rl_y=np.empty(10)

## Sold Price
# https://www.zillow.com/homes/recently_sold/King-County-WA/49015189_zpid/207_rid/globalrelevanceex_sort/47.820992,
rl_X.loc[len(rl_X)]=[1770,7840,1968,4,3,47.7397,-122.185,0]
rl_y[len(rl_X)-1]=720000
# https://www.zillow.com/homes/recently_sold/King-County-WA/49120416_zpid/207_rid/globalrelevanceex_sort/47.820992,
rl_X.loc[len(rl_X)]=[1950,4887,1911,2,2,47.535,-122.388, 1]
rl_y[len(rl_X)-1]=865000
# https://www.zillow.com/homes/recently_sold/King-County-WA/48829308_zpid/207_rid/200000-600000_price/830-2490_mp/g
rl_X.loc[len(rl_X)]=[1810,17424,1994,3,3,47.364,-122.043, 0]
rl_y[len(rl_X)-1]=432000
# https://www.zillow.com/homes/recently_sold/King-County-WA/84756911_zpid/207_rid/globalrelevanceex_sort/47.633354,
rl_X.loc[len(rl_X)]=[1289,1000,2009,2,3,47.532,-122.072, 0]
rl_y[len(rl_X)-1]=520000

i_sale=len(rl_X)
```

Рисунок 3.53 – Код програми внесення реальних даних (Початок)

```
## Sale Price
# https://www.zillow.com/homes/for_sale/King-County-WA/49127321_zpid/207_rid/globalrelevanceex_sort/47.820992,-121.
rl_X.loc[len(rl_X)]=[2760,4839,1923,4,3,47.557,-122.375, 0]
rl_y[len(rl_X)-1]=550000
# https://www.zillow.com/homes/for_sale/King-County-WA/48662094_zpid/207_rid/200000-600000_price/830-2490_mp/global
rl_X.loc[len(rl_X)]=[1290,4791,1925,1,1.5,47.513,-122.387, 0]
rl_y[len(rl_X)-1]=399000
# https://www.zillow.com/homedetails/1102-E-Hemlock-St-Kent-WA-98030/49077132_zpid/
rl_X.loc[len(rl_X)]=[2020,7701,1959,5,2,47.374,-122.220, 0]
rl_y[len(rl_X)-1]=367500
# https://www.zillow.com/homes/for_sale/King-County-WA/48702491_zpid/207_rid/globalrelevanceex_sort/47.633354,-121.
rl_X.loc[len(rl_X)]=[2040,8119,1963,4,2,47.502,-122.167, 0]
rl_y[len(rl_X)-1]=435000


rl_X["sqft_living"]= rl_X["sqft_living"].apply(lambda x: log(x))
rl_X["lat"]= rl_X["lat"].apply(lambda x: abs(47.63-x))
rl_X["long"]= rl_X["long"].apply(lambda x: abs(x))
rl_X["yr_built"]= rl_X["yr_built"].apply(lambda x: log(x))
rl_X["sqft_lot"]= rl_X["sqft_lot"].apply(lambda x: log(x))
rl_X["lat*long"]=rl_X["lat"]*rl_X["long"]
rl_X["sqft_living*sqft_lot"]=rl_X["sqft_living"]*rl_X["sqft_lot"]
```

Рисунок 3.54 - Код програми внесення реальних даних (Кінець)

Та отримується результат прогнозування на рис.3.55:

```
Sold
House 0 for 720000; predicted -> 463289
House 1 for 865000; predicted -> 1179232
House 2 for 432000; predicted -> 355828
House 3 for 520000; predicted -> 338729

On Sale
House 4 for 560000; predicted -> 776516
House 5 for 430000; predicted -> 388952
House 6 for 393500; predicted -> 291041
House 7 for 456000; predicted -> 447304
```

Рисунок 3.55 – Результат прогнозування

Можна розглянути детальніше, якщо взяти один з прикладів рис.3.56:

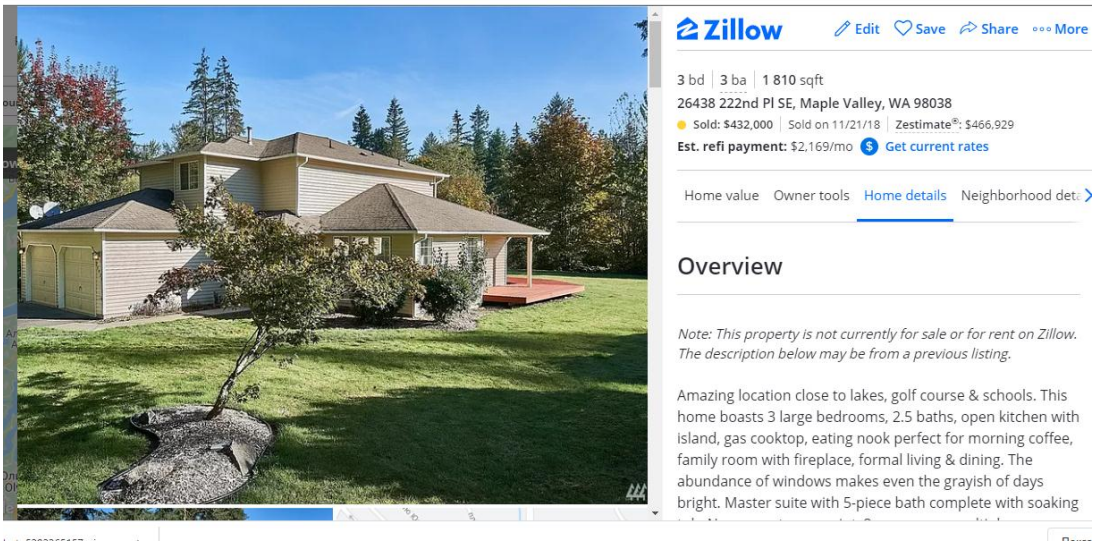


Рисунок 3.56 – Будинок з сайту №2

Реальна ціна за яку продали будинок №2 = 432 000 $, а модель прогнозує 355 828 $ – це не дуже хороший результат. Але якщо розглянути всі приклади продаж, то виходить досить непоганий результат прогнозу для такої простої моделі.

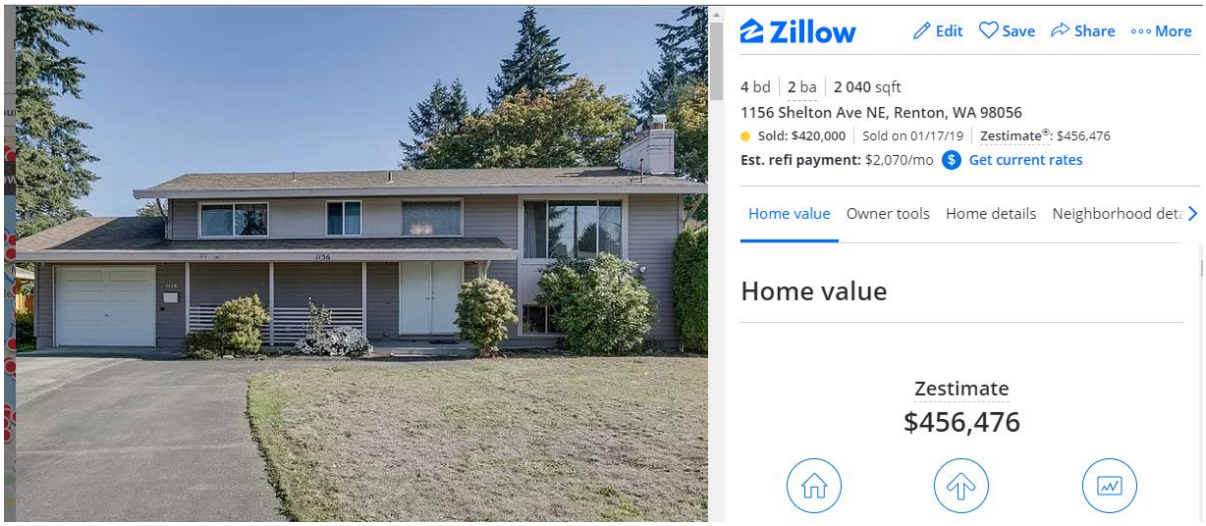


Рисунок 3.57 – Будинок з сайту № 7

Тут доволі гарний результат, адже будинок в середньому (на рис.3.57) коштує 456 000 $ і прогноз моделі відповідає 447 000 $.

Хоча, звісно, така велика прогрішніть недопустима в реальному житті, адже ці помилки коштують великих втрат з боку фінансування.

Висновки до розділу 3

В даному розділі було описано розробку статистичної моделі прогнозування цін на нерухомість. Зокрема розглянуто кореляцію даних, побудовано Лінійну регресію на основі аналізу, виконана візуалізація даних з використанням різних бібліотек. Аналізовані дані відображено на карті за допомогою Folium. Реалізовано кластеризацію даних та тестування на реальних даних, що показало доволі непогані результати прогнозу ціни.

ВИСНОВКИ

У даній роботі було здійснено розробку статистичної моделі прогнозування цін на нерухомість. Результат – 76 % точності. Розроблено додатковим функціоналом аналіз даних, візуалізацію, кластеризацію для кращого відображення всіх основних частин. Для економіки нашої країни є досить актуальним питання структурного аналізу даного виду ринку і подальшого активного розвитку як ринку житла, так і споріднених йому видів ринку. Тому постала необхідність розробки концепції прогнозування цінової політики на ринку житлової нерухомості з урахуванням не тільки загально визначених факторів, таких як середня заробітна плата, валютний курс, безробіття, обсяг виконаних будівельних робіт тощо, які в багатьох випадках представлені трендовими багатофакторними моделями, а й врахувати такий значущий фактор, як привабливість району, в якому представлений досліджуваний вид нерухомості.

Не применшуючи значення робіт, присвячених проблемам на ринку нерухомості, слід зазначити, що велика кількість теоретичних питань функціонування ринку не мають на сьогоднішній день розв’язання, також значна кількість моделей та методів ціноутворення ще не знайшла своєї реалізації саме на ринку нерухомості.

На основі аналізу даних була розроблена концепція прогнозування ціни об’єкта на ринку житлової нерухомості, яка складається з п’яти основних контурів, а саме: вхідні дані, оцінка впливу факторів, контур моделювання, синтез моделей та, безпосередньо, прогнозування ціни. Також розглянуто сучасні методи та засоби прогнозування цін на нерухомість.

Побудова концептуальної моделі дозволяє систематизувати загальні положення щодо прогнозування та моделювання політики ціноутворення та дає змогу не тільки спрогнозувати середній показник рівня ціни для наступного періоду, але й ввести в модель принципово новий корегуючий коефіцієнт.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Андреас Мюллер, Сара Гвидо, Введение в машинное обучение с помощью Python. Руководство для специалістов по роботе с данными. – Москва. – 2016 – 2017. – 340 с.

2. Forecasting in Python with Facebook Prophet. [Електронний ресурс]. – Режим доступу:https://towardsdatascience.com/forecasting-in-python-with-facebook-prophet-29810eb57e66 – Назва з екрану.

3. Матеріали міжнародної наукової конференції, Іжевськ, 13–17 липня 2006. – с. 98 – 103.

4. Ресурс даних [Електронний ресурс]. – Режим доступу: https://www. kaggle.com/ – Назва з екрану.

5. Техническая документацыя по stoker. [Електронний ресурс]. – Режим доступу: https://pypi.org/project/stocker/ – Назва з екрану.

6. Введение в Scikit-learn [Електронний ресурс]. – Режим доступу: https://neurohive.io/ru/osnovy-data-science/vvedenie-v-scikit-learn/ – Назва з екрану.

7. Ной Гифт, Прагматичный ИИ.Машинное обучение и облачныие технологии. – СПб.: Питер. – 2019. – с.23 – 154.

8. Лінійна регресія [Електронний ресурс]. – Режим доступу: https://uk wikipedia.org/ – Назва з екрану. Sandro Skansi, Introduction to Deep Lerning. – Springer. – 2018.

9. Элбон Крис, Машинное обучение с использованием Python. Сборник рецептов. – СПб.: БХВ-Петербург. — 2019. – 890 с.

10. Sebastian Raschka and Vahid Mirjalili, Python Machile Learning. – Packt, Mumbai. – 2017. – pp. 469 – 473.

11. Практик Джоши, Искусственный интеллект с примерами на Python. – СПб.: ООО “Диалектика”. – 2019. – с.144 – 156.

12. Алекс Мартелли, Анна Рейвенскрофт, Стив Холден, Python справочник. – СПб.: ООО “Диалектика”. – 2019. – с. 365 – 786.

13. Дэви Силен, Арно Мейсман, Мохамед Али, Основы Data Science и Big Data. – Спб.: Питер. – 2017. – с.876 – 889.

14. Ян Гудфеллоу, Иошуа Бенджио, Аарон Курвилль, Глубокое обучение. – М.: ДМК Пресс – 2018. – с.123 – 134.

15. 27 шпаргалок по машинному обучению и Python. [Електронний ресурс]. – Режим доступу: https://proglib.io/p/ds-cheatsheets - Назва з екрану.

16. Харрисон Мэтт, Как устроен Python. Гид для разработчиков, программистов и интересующихся. – СПб.:Питер. – 2019. – с. 56 – 89.

17. Способи розпізнавання тексту [Електронний ресурс]. – Режим доступу:http:// mcgrp.ru/ sposobyi-raspoznavaniya-otsifrovki-teksta-article.html – Назва з екрану.

18. Андреас Мюллер, Сара Гвидо, Введение в машинное обучение с помощью Python. Руководство для специалістов по роботе с данными. – Москва. – 2016 – 2017. – с.56 – 93.

19. Francois Chollet, Deep Learning with Python. – Manning Shelter Island. – 2019. pp. 34 – 56.

20. Дж.Вандер Плас, Python для сложных задач, наука о данных и машинное обучение. – СПб.: Питер. – 2018. – с.34 – 87.

21. Бенджамин Бенгфорт, Ребекка Билбро и Тони Охеда, Прикладной анализ текстовых данных на Python. – СПб.: Питер. – 2019. – с.560

22. Bishop C. M. Pattern recognition and machine learning. – Springer. – 2006.

23. Уэс Маккинни, Python и анализ данніх. – М.: ДМК Пресс. – 2015. – с.67 – 458.

24. Себастьян Рашка, Python и машинное обучение. – М.: ДМК Пресс. – 2017. – с.78 – 90.

25. Підбір датасетів для машинного навчання [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/452392 – Назва з екрану.

26. David Julian, Designing Machine Learning Systems with Python. – Packt. – 2016. pp. 499 – 773.

27. Штучний інтелект [Електронний ресурс]. – Режим доступу:– Режим доступу: https://habr.com/ru/post/452392 – Назва з екрану.

28. Ervin Varga, Practical Data Science with Python 3. – Copyright. – 2019. – 567 с.

29. Chin – Sheng Chen, Jian-Jhe Huang, Chien – Liang Huang, Template Matching using Statistical Model and Parametric Template for Multi-Template – Journal of Signal and Information Processing, August 2013. pp.52 – 57.

30. T. Mahalakshmi, R. Muthaiah and P. Swaminathan, Review Article: An Overview of Template Matching Technique in Image Processing Research Journal of Applied Sciences, Engineering and Technology, December 2012, ISSN: 2040-7467. pp. 469 – 473.