GUJARAT TECHNOLOGICAL UNIVERSITY

Academic year

(2022-2023)

Gujarat Technical University

VPMP POLYTECHNIC, GANDHINAGAR

INTERSHIP

REPORTUNDER

SUBJECT OF

SUMMER INTERSHIP (4350704)

DIPLOMA SEMESTER-**V**

BRANCH-Computer

EngineeringSubmitted By: -

Deep Jain ManojBhai

STYPIX

Harvi Jivani

(Internal Guide)

Sagar Khatri

(External Guide)

Company Profile

Company Name: STYPIX

Address: A-903, Siddhivinayak Business Tower Nr, Vasna Telephone Exchange, Makarba, Ahmedabad, Gujarat 380051

Contact No: +917433050303

Email Id: harvijivani@gmail.com

Website: https://www.stypix.co.in/

About Us

STYPIX delivers the best-in-class custom software solutions, elite software development teams and innovative cloud software to enterprise businesses across numerous industries. We believe new technologies are the lifeline of every business in the modern age and aim to connect businesses across all industries to innovative software, technological development, solutions, and services, in a manner that's faster, easier, and better than ever before. Our mission is to create leading, innovative software that creates economic and social value on a global scale, collaborating closely with our clients to help them achieve both their short- and longer-term goals.

Vision

To become the most trusted and preferred offshore IT solutions partner for Startups, SMBs and Enterprises through innovation and technology leadership. Understanding your ambitious vision,homing in on its essence, creating a design strategy, and knowing how to technically execute it iswhat we do best. Our promise? The integrity of your vision will maintain, and we'll enhance it to best reach your target customers. With our primary focus on creating amazing user experiences, we'll help you understand the tradeoffs, prioritize features, and distill valuable functionality. It's an art form we care about getting right.

JOINING LETTER

STYPIX

Offer Letter

To: Deep Jain (216540307044)

VPMP Polytechnic

We are delighted to report that you have been accepted for a **Data Science & Machine Learning** internship at stypix. The following are your responsibilities and assignments for this position: working with several modules of the relevant technology

27th Jul-2023 will be your first day of employment. Five days a week will be dedicated to work. You will report to the appropriate Developer.

Hadhaniya
STYPIX

Regards
Stypix

Address: A-903, Siddhivinayak Business Tower Nr, Vasna Telephone Exchange, Makarba, Ahmedabad, Gujarat 380051
Email: harvijivani@gmail.com Contact: 7046976500

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all those who provided me the possibility to the completion of the internship. A special gratitude of thanks I give to our Assistant Professor, Prof. Vaishali Mehta, whose contribution in stimulating suggestions and encouragement helped me to coordinate the internship especially in drafting this report.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the Head of Department, Kiran Jha who gave the permission to use all required equipment and the necessary material to fulfill the task. Last butnot the least, many thanks go to the teachers and my friends and families who have invested their full effort in guiding us in achieving the goal.

Also, I appreciate the guidance given by the developer at Brainy Beam, Mr. Sagar Jasani as well as the panels especially for the internship that has advisedme and gave guidance at every moment of the internship.

Abstract

The project focuses on analyzing residential property prices to gain insights into the factors influencing the housing market. By examining various variables such as property characteristics, location, market trends, and economic indicators, this project aims to provide a comprehensive understanding of the factors driving home prices and identify key trends and patterns. The project utilizes a dataset containing information on residential properties, including features such as square footage, number of bedrooms and bathrooms, location attributes (e.g., neighborhood, proximity to amenities), and corresponding sale prices. By analyzing these variables, the project seeks to uncover correlations and trends that can help stakeholders understand the dynamics of the housing market and make informed decisions regarding real estate investments, pricing strategies, and policy development. Through exploratory data analysis and statistical modeling techniques, the project aims to identify the key factors influencing home prices. This includes assessing the impact of property characteristics (e.g., size, amenities), location factors (e.g., proximity to schools, transportation), and market trends (e.g., supply and demand, interest rates) on property values. Additionally, the project explores the relationships between economic indicators (e.g., GDP growth, employment rates) and home prices to understand the broader macroeconomic influences on the housing market. The outcomes of this project have implications for real estate professionals, investors, policymakers, and prospective homeowners. Real estate professionals can utilize the findings to better understand market trends, advise clients, and develop effective pricing strategies. Investors can gain insights into factors that drive property value appreciation and identify lucrative investment opportunities. Policymakers can use the findings to inform housing policies, zoning regulations, and urban planning initiatives. Prospective homeowners can make more informed decisions regarding property purchases, considering factors that influence home prices and long-term value appreciation.

DATE- 27 JUL

Basic Introduction of Data Science

Explain about workflow of whole internship. Also discuss some basic domain knowledge.

What is Data Science?

Data science is the study of data to extract meaningful insights for business. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data. This analysis helps data scientists to ask and answer questions like what happened, why it happened, what will happen, and what can be done with the results.

Which software are used in data science?

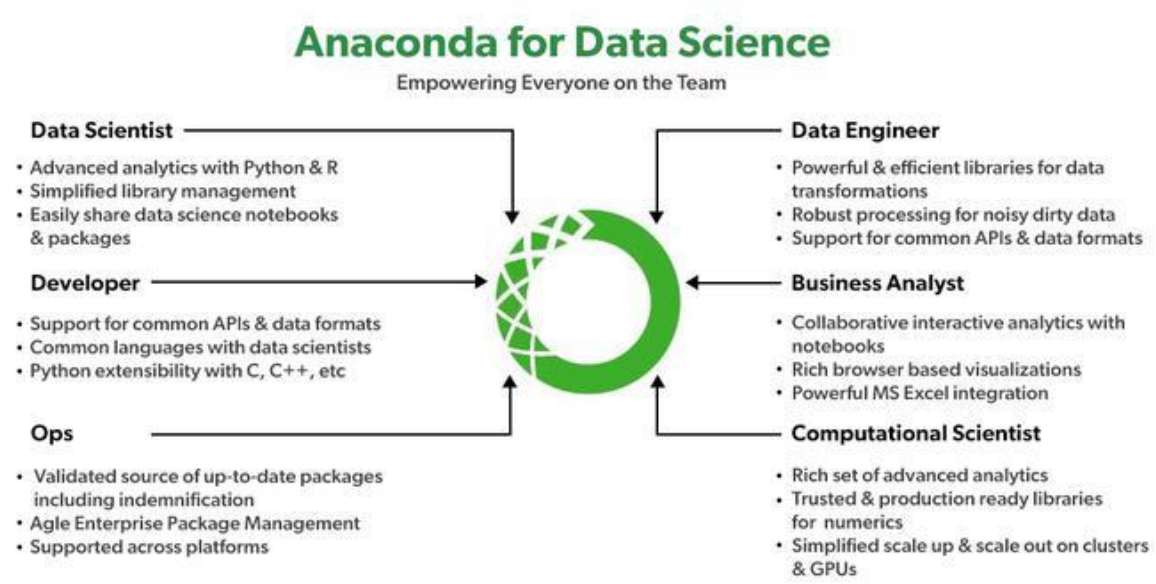
- ✓ Microsoft Excel
- ✓ Python
- ✓ excel MySQL.
- ✓ SAS
- ✓ Jupiter Notebook

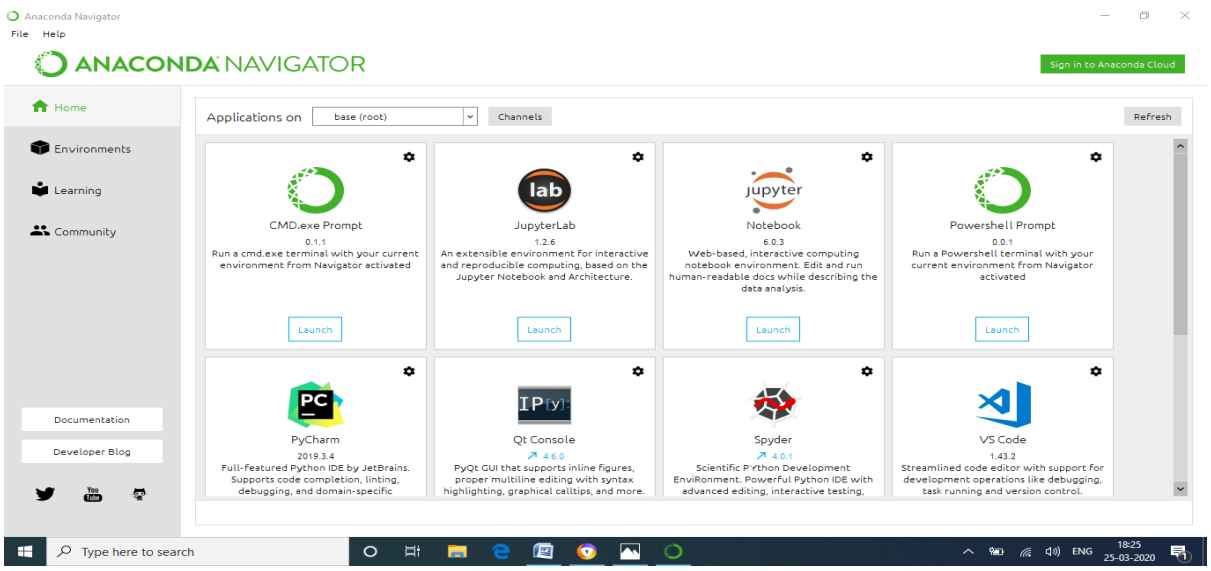
What is Machine Learning?

Machine learning (ML) is an umbrella term for solving problems for which development of algorithms by human programmers would be cost-prohibitive, and instead the problems are solved by helping machines 'discover' their 'own' algorithms, without needing to be explicitly told what to do by any human-developed algorithms.

How to install Anaconda:

Head over to anaconda.com and install the latest version of Anaconda. Make sure to download the “Python 3.7 Version” for the appropriate architecture.


Anaconda for Data Science
Empowering Everyone on the Team
Data Scientist
• Advanced analytics with Python & R
• Simplified library management
• Easily share data science notebooks & packages
Developer
• Support for common APIs & data formats
• Common languages with data scientists
• Python extensibility with C, C++, etc
Ops
• Validated source of up-to-date packages including indemnification
• Agile Enterprise Package Management
• Supported across platforms
Data Engineer
• Powerful & efficient libraries for data transformations
• Robust processing for noisy dirty data
• Support for common APIs & data formats
Business Analyst
• Collaborative interactive analytics with notebooks
• Rich browser based visualizations
• Powerful MS Excel integration
Computational Scientist
• Rich set of advanced analytics
• Trusted & production ready libraries for numerics
• Simplified scale up & scale out on clusters & GPUs

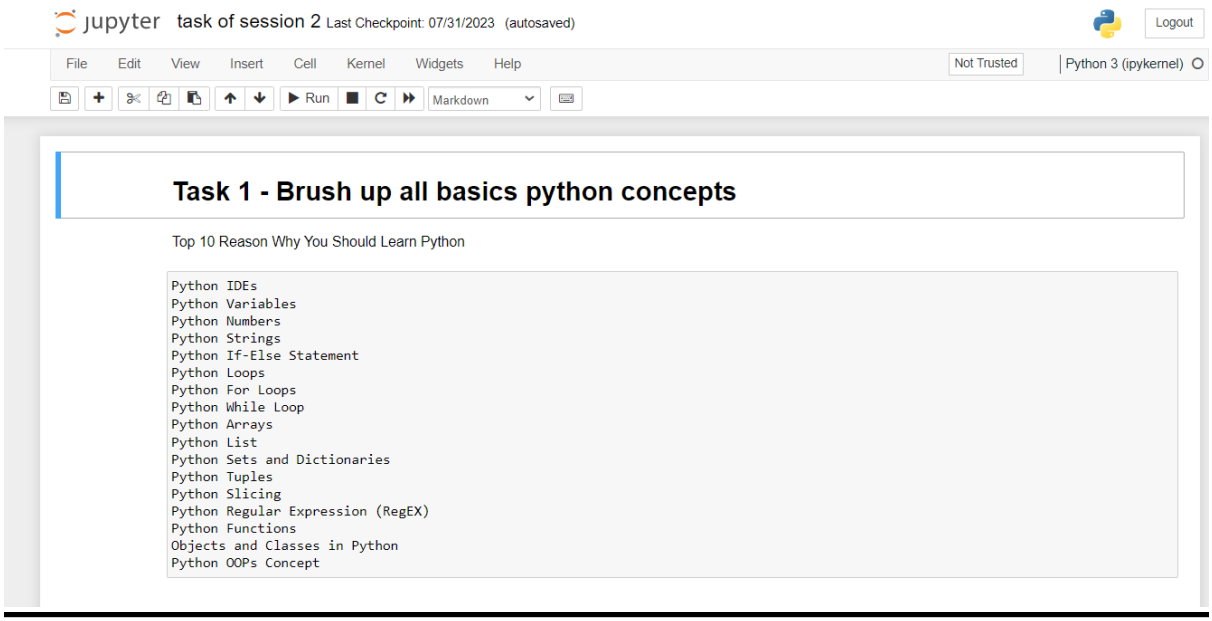

Anaconda Navigator
File Help
ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Learning
Community
Documentation
Developer Blog
Applications on
base (root)
Channels
Refresh
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
JupyterLab
1.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.0.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
PyCharm
2019.3.4
Full-featured Python IDE by JetBrains. Supports code completion, linting, debugging, and domain-specific
Qt Console
4.6.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Spyder
4.0.1
Scientific Python Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing,
VS Code
1.43.2
Streamlined code editor with support for development operations like debugging, task running and version control.
Type here to search
ENG
18:25
25-03-2020

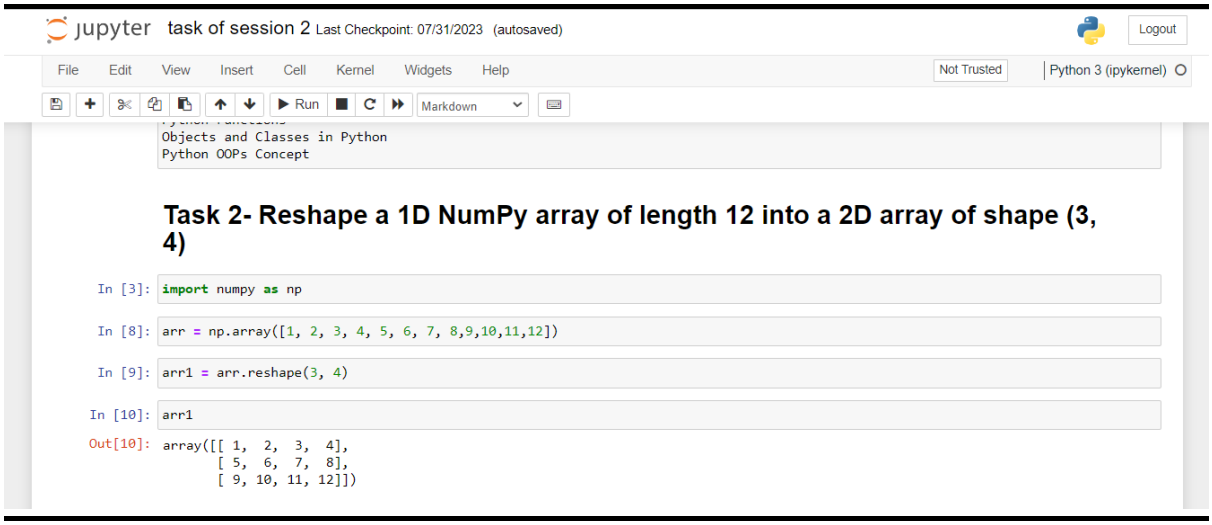
DATE -28 JUL

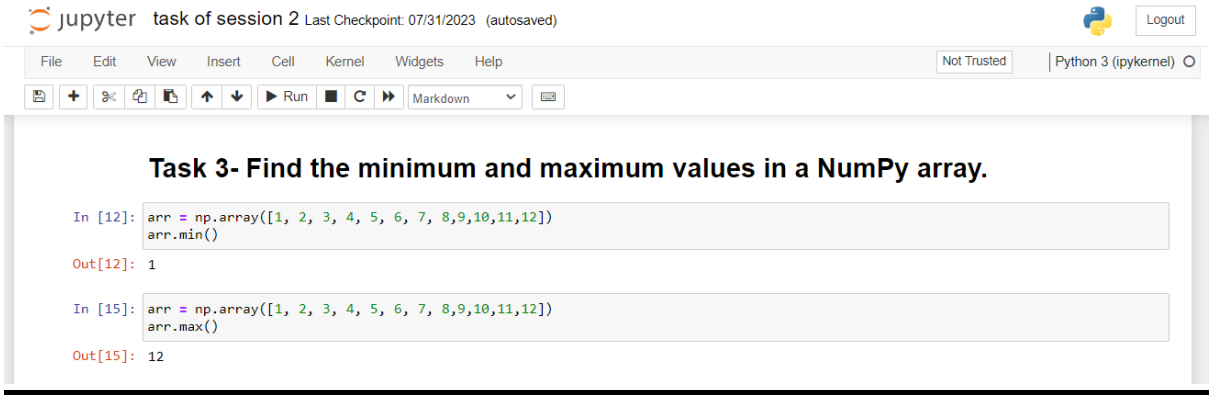
What is NumPy?

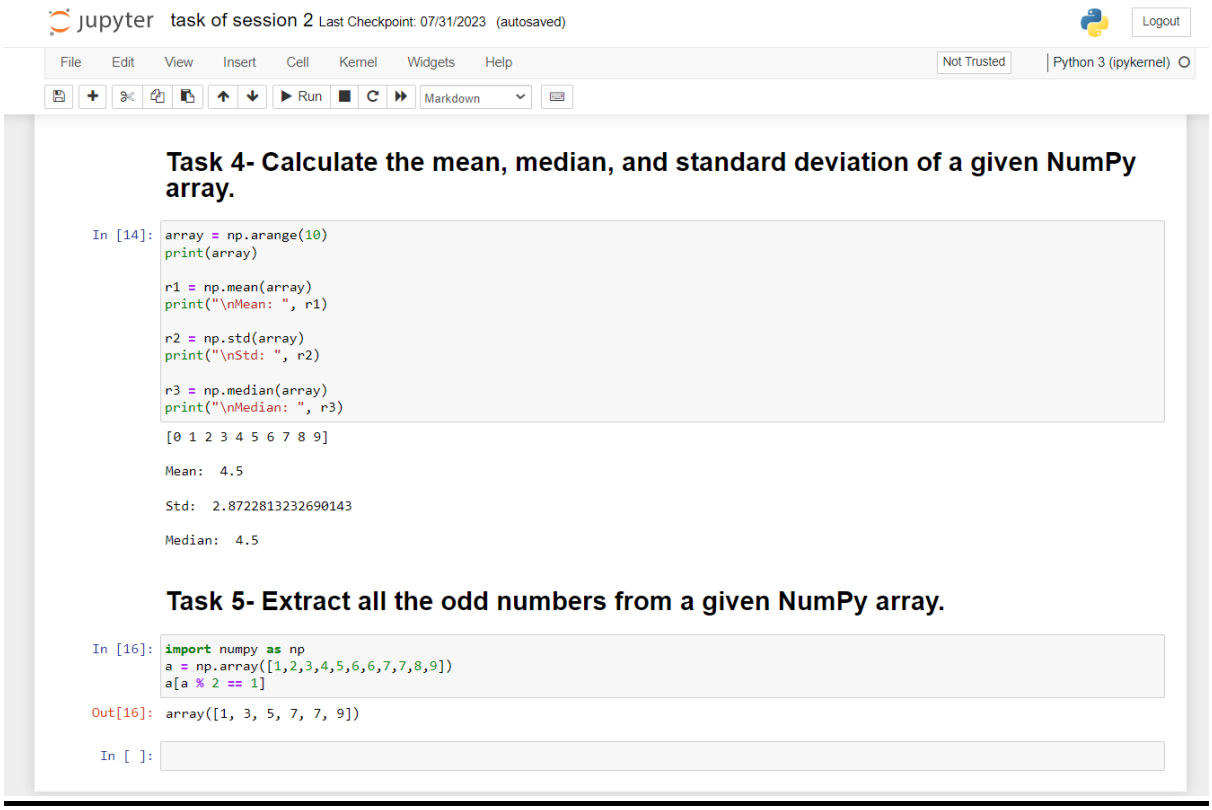
NumPy is a Python library used for **working with arrays**. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open-source project, and you can use it freely. NumPy stands for Numerical Python.

Tasks of NumPy:









Conclusion:

Using a NumPy library of python we can,

- ✓ Create arrays.
- ✓ Sorting arrays
- ✓ Concatenating arrays
- ✓ Slicing arrays
- ✓ Print random numbers.

DATE-29,30 JUL(sat-sun)

DATE - 31 JUL,1 AUG,2 AUG (3 DAY)

Introduction of Pandas

- A panda is a Python library used for working with data sets.
- It has functions for analysing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- Pandas allow us to analyse big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets and make them readable and relevant.
- Relevant data is very important in data science.

Introduction of Data frame:

- Now, Data Frames in Python are very similar: they come with the panda's library, and they are defined as two-dimensional labeled data structures with columns of potentially different types. In general, you could say that the pandas Data Frame consists of three main components: the data, the index, and the columns.

What is in place?

- In-place operation is an operation that directly changes the content of a given linear algebra, vector, matrices (Tensor) without making a copy. The operators which helps to do the operation are called in-place operators. E.g.: a+= b is equivalent to a= operator.
- Using the in place=True keyword in a panda's method changes the default behavior such that the operation on the data frame doesn't return anything, it instead 'modifies the underlying data' (more on that later). It mutates the actual object which you apply it to.

What does axis mean python?

- Axis is defined for arrays with more than one dimension. A 2-dimensional array has two corresponding axes: the first running vertically downwards across rows (axis 0),

and the second running horizontally across columns (axis 1). Much operation can take place along one of these axes.

- The axis parameter specifies the index of the new axis in the dimensions of the result. For example, if axis=0 it will be the first dimension and if axis=-1 it will be the last dimension.

Introduction of loc and iloc:

- Loc and iloc are two functions in Pandas that are used to slice a data set in a Pandas Data Frame. The function. loc is typically used for label indexing and can access multiple columns, while. Iloc is used for integer indexing.
- The panda's library in Python is used to work with data frames that structure data in rows and columns. It is widely used in data analysis and machine learning. The loc operator is used to index a portion of the data frame. Loc supports indexing both by row and column names and by using Boolean expressions.
- The iloc () function in python is one of the functions defined in the Pandas module that helps us to select a specific row or column from the data set. Using the iloc () function in python, we can easily retrieve any particular value from a row or column using index values.

Examples of loc and iloc:

```
df.loc [0]
df.loc [df ["Age"]>12]
df.Iloc [:]
df.Iloc [0:4, 0:1]
```

How to import file into dataframe:

Pandas read_csv () function imports a CSV file to Data Frame format. Header: this allows you to specify which row will be used as column names for your data frame. Expected an int value or a list of int values. Default value is header=0 , which means the first row of the CSV file will be treated as column names.

Read A CSV File Using Python

- Using the CSV Library. import csv with open("./bwq.csv", 'r') as file: csvreader = csv.reader(file) for row in csvreader: print(row) ...

Using the Pandas Library. Import pandas as pd data = pd.read_csv("bwq.csv") data.

Isnull():

- To check missing values in Pandas Data Frame, we use a function is null () and not null (). Both functions help in checking whether a value is NaN or not. These functions can also be used in Pandas Series in order to find null values in a series.

Isna:

- isna() method checks whether the objects of a Data frame or a Series contain missing or null values (NA, NaN) and returns a new object with the same shape as the original but with Boolean values True or False as the elements.

Isnull.sum():

- Sum () calculates the sum of elements for each row and column. Since sum () calculates as True=1 and False=0, you can count the number of NaN in each row and column by calling sum () on the result of isnull (). You can count NaN in each column by default, and in each row with axis=1.

Fillna():

- The fillna () method replaces the NULL values with a specified value. The fillna () method returns a new Data Frame object unless the in place parameter is set to True , in that case the fillna() method does the replacing in the original Data Frame instead

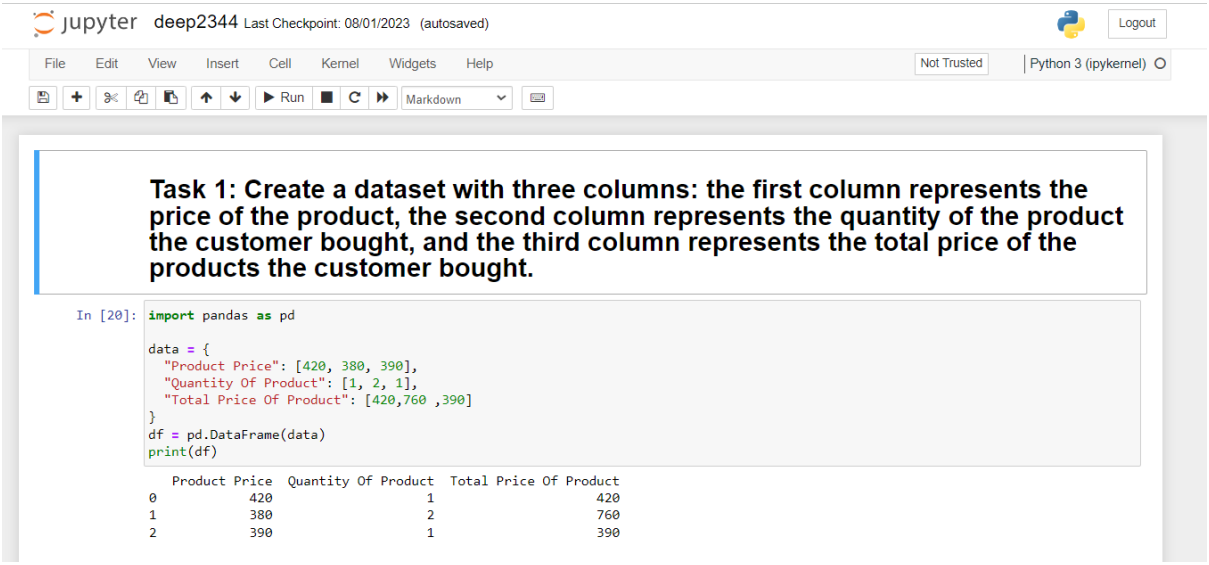
Info ():

- The info () method prints information about the Data Frame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values). Note: the info () method prints the info.

Examples of isnull(),fillna,isna():

```
df.isnull().sum()
df["Age"].fillna(df['Age'].mean,inplace=True)
df["Age"]=df["Age"].astype(int)
```

Tasks of pandas:



jupyter deepdive Last Checkpoint: 08/01/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

Task 2: Change data type of any column using train.csv file

In [21]:
```
import numpy as np
df=pd.read_csv("train.csv")
print(df.dtypes)
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [28]:
```
df1 = df.copy()
df1["SibSp"] = df1["SibSp"].astype("float64")
df1.head()
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    float64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(3), int64(4), object(5)
memory usage: 83.7+ KB
```

jupyter deepdive Last Checkpoint: 08/01/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

memory usage: 83.7+ KB

Task 3: Calculating descriptive statistics for numeric column

In [13]:
```
df.describe()
```

Out[13]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Task4: Counting unique values in a column.

In [16]:
```
df.Age.unique()
```

Out[16]:
```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

DATE - 3 AUG

Simple Imputer:

- Simple Imputer is a class in the sklearn. Impute module that can be used to replace missing values in a dataset, using a variety of input strategies. Simple Imputer is designed to work with numerical data but can also handle categorical data represented as strings.
- Using the strategy “Median” in the Simple Imputer allows us to impute the missing value by the median value of the dataset. This strategy can only be used on the numerical dataset. Let's suppose we have a column age in our dataset in which we have a missing value.

Example:

In [6]:
```
from sklearn.impute import SimpleImputer
```

In [7]:
```
sp=SimpleImputer()
```

In [ ]:
```
# age=df["Age"].reshape(-1,1)
```

In [11]:
```
df["Age"]=sp.fit_transform(df[["Age"]])
```

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

DATE – 4 AUG

Standard Scaler:

StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way. StandardScaler can be influenced by outliers (if they exist in the dataset) since it involves the estimation of the empirical mean and standard deviation of each feature.

Example:

In [23]:
```
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
st=StandardScaler()
```

In [25]:
```
x_train_Scaler=st.fit_transform(x_train)
x_test_Scaler=st.transform(x_test)
```

In [26]:
```
x_train_Scaler
```

Out[26]:
```
array([[-1.11114426,  0.98099823],
       [-1.33985407, -0.46963364],
       [-1.28267662, -0.40613632],
       ...,
       [ 1.00442157, -0.34778742],
       [-0.53936971,  1.72907416],
       [-0.13912752,  0.8913508 ]])
```

In [27]:
```
x_train_Scaler=pd.DataFrame(x_train_Scaler,columns=x_train.columns)
```

In [29]:
```
x_train_Scaler.describe()
```

Out[29]:

	Age	Fare
count	6.230000e+02	6.230000e+02
mean	-7.925709e-17	1.799880e-17
std	1.000804e+00	1.000804e+00
min	-1.339854e+00	-6.244945e-01
25%	-1.111144e+00	-4.690609e-01
50%	3.240484e-02	-3.410033e-01
75%	6.613568e-01	-2.139256e-02
max	3.234342e+00	9.423855e+00

DATE – 5,6 AUG (SAT-SUN)

DATE – 7 AUG

Normalization:

Data Normalization could also be a typical practice in machine learning which consists of transforming numeric columns to a standard scale. In machine learning, some feature values differ from others multiple times. The features with higher values will dominate the learning process.

One-Hot encoder:

One hot encoding is a technique that we use to represent categorical variables as numerical values in a machine learning model.

Label encoder:

Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine-learning project.

Tasks:

jupyter onelabel and normalization Last Checkpoint: 08/10/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

(Tasks for one hot and label encoder) Apply the same task in different datasets using kaggle

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df=pd.read_csv("train.csv")
df
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [3]:
```
from sklearn.preprocessing import LabelEncoder
```

In [4]:
```
l=LabelEncoder()
```

jupyter onelabel and normalization Last Checkpoint: 08/10/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [5]:
```
df["Sex"]=l.fit_transform(df[["Sex"]])
df
```

```
C:\Users\hp\anaconda3\Lib\site-packages\sklearn\preprocessing\_label.py:116: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	1	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	0	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	0	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	1	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	1	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [6]:
```
df1=pd.get_dummies(df,columns=["Embarked"])
df1
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked_C	Embarked_Q	Embarked_S
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	0	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	1	0	0
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	0	0	1

jupyter onelabel and normalization Last Checkpoint: 08/10/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

890	891	0	3	Dooley, Mr. Patrick	1	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [6]:
```
df1=pd.get_dummies(df,columns=["Embarked"])
df1
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked_C	Embarked_Q	Embarked_S
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	0	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	1	0	0
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	0	0	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	0	0	1
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	1	27.0	0	0	211536	13.0000	NaN	0	0	1
887	888	1	1	Graham, Miss. Margaret Edith	0	19.0	0	0	112053	30.0000	B42	0	0	1
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	0	NaN	1	2	W./C. 6607	23.4500	NaN	0	0	1
889	890	1	1	Behr, Mr. Karl Howell	1	26.0	0	0	111369	30.0000	C148	1	0	0
890	891	0	3	Dooley, Mr. Patrick	1	32.0	0	0	370376	7.7500	NaN	0	1	0

891 rows × 14 columns

jupyter onelabel and normalization Last Checkpoint: 08/10/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

(Task for normalization and z-score) Find 10000 row dataset and apply same task in the 10000 row dataset.

In [8]:
```
from sklearn import preprocessing
import pandas as pd

from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing(as_frame=True)

d = preprocessing.normalize(california_housing.data, axis=0)
scaled_df = pd.DataFrame(d, columns=california_housing.data.columns)
print(scaled_df)
```

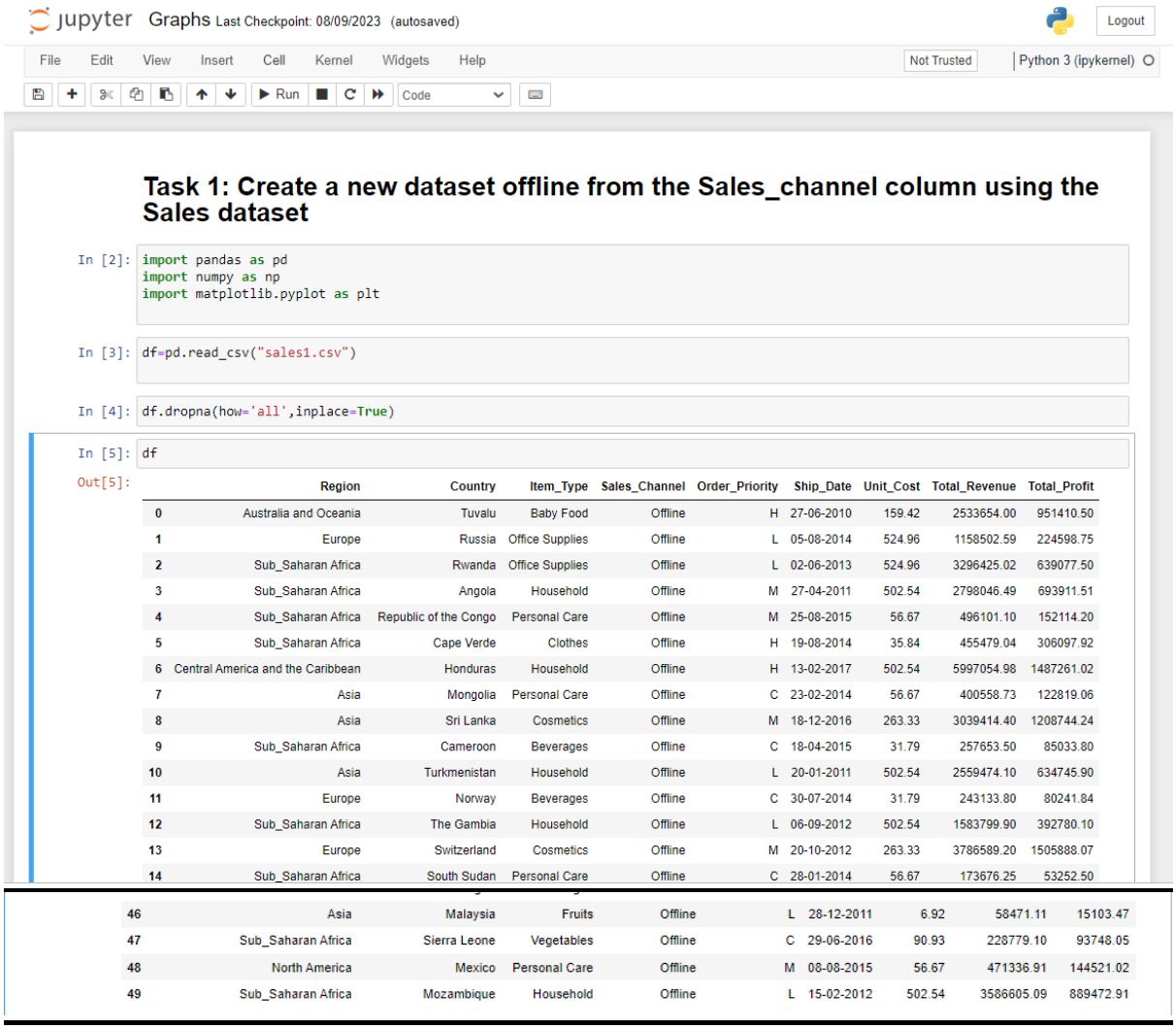
```
         MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
0      0.013440  0.009123  0.008148   0.005965    0.001231  0.001642
1      0.013401  0.004673  0.007278   0.005662    0.009180  0.001356
2      0.011716  0.011570  0.009670   0.006254    0.001896  0.001801
3      0.009110  0.011570  0.006787   0.006252    0.002133  0.001638
4      0.006209  0.011570  0.007329   0.006299    0.002160  0.001402
...         ...       ...       ...        ...         ...       ...
20635  0.002519  0.005563  0.005886   0.006603    0.003231  0.001646
20636  0.004128  0.004005  0.007133   0.007666    0.001361  0.002007
20637  0.002744  0.003783  0.006073   0.006526    0.003850  0.001495
20638  0.003014  0.004005  0.006218   0.006828    0.002833  0.001365
20639  0.003856  0.003560  0.006131   0.006772    0.005303  0.001682

       Latitude  Longitude
0      0.007386  -0.007114
1      0.007383  -0.007114
2      0.007381  -0.007115
3      0.007381  -0.007116
4      0.007381  -0.007116
...         ...        ...
20635  0.007698  -0.007048
20636  0.007700  -0.007055
20637  0.007689  -0.007056
20638  0.007689  -0.007061
20639  0.007677  -0.007057

[20640 rows x 8 columns]
```

DATE – 8 AUG

Matplotlib Library:

Matplotlib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consist of several plots like line chart, bar chart, histogram, etc. It provides a lot of flexibility but at the cost of writing more code.

Tasks:



	Region	Country	Item_Type	Sales_Channel	Order_Priority	Ship_Date	Unit_Cost	Total_Revenue	Total_Profit
0	Australia and Oceania	Tuvalu	Baby Food	Offline	H	27-06-2010	159.42	2533654.00	951410.50
1	Europe	Russia	Office Supplies	Offline	L	05-08-2014	524.96	1158502.59	224598.75
2	Sub_Saharan Africa	Rwanda	Office Supplies	Offline	L	02-06-2013	524.96	3296425.02	639077.50
3	Sub_Saharan Africa	Angola	Household	Offline	M	27-04-2011	502.54	2798046.49	693911.51
4	Sub_Saharan Africa	Republic of the Congo	Personal Care	Offline	M	25-08-2015	56.67	496101.10	152114.20
5	Sub_Saharan Africa	Cape Verde	Clothes	Offline	H	19-08-2014	35.84	455479.04	306097.92
6	Central America and the Caribbean	Honduras	Household	Offline	H	13-02-2017	502.54	5997054.98	1487261.02
7	Asia	Mongolia	Personal Care	Offline	C	23-02-2014	56.67	400558.73	122819.06
8	Asia	Sri Lanka	Cosmetics	Offline	M	18-12-2016	263.33	3039414.40	1208744.24
9	Sub_Saharan Africa	Cameroon	Beverages	Offline	C	18-04-2015	31.79	257653.50	85033.80
10	Asia	Turkmenistan	Household	Offline	L	20-01-2011	502.54	2559474.10	634745.90
11	Europe	Norway	Beverages	Offline	C	30-07-2014	31.79	243133.80	80241.84
12	Sub_Saharan Africa	The Gambia	Household	Offline	L	06-09-2012	502.54	1583799.90	392780.10
13	Europe	Switzerland	Cosmetics	Offline	M	20-10-2012	263.33	3786589.20	1505888.07
14	Sub_Saharan Africa	South Sudan	Personal Care	Offline	C	28-01-2014	56.67	173676.25	53252.50
46	Asia	Malaysia	Fruits	Offline	L	28-12-2011	6.92	58471.11	15103.47
47	Sub_Saharan Africa	Sierra Leone	Vegetables	Offline	C	29-06-2016	90.93	228779.10	93748.05
48	North America	Mexico	Personal Care	Offline	M	08-08-2015	56.67	471336.91	144521.02
49	Sub_Saharan Africa	Mozambique	Household	Offline	L	15-02-2012	502.54	3586605.09	889472.91

Task 2: According to task 2 now plot bar graph of unit cost and No_product using offline.

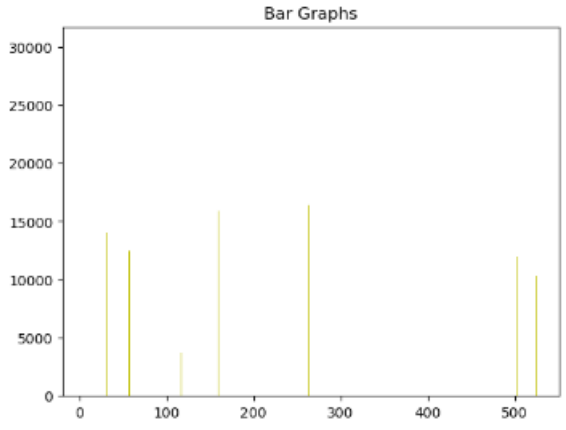
In [6]:
```
df["No_product"]=df["Total_Revenue"]/df["Unit_Cost"]
```

In [7]:
```
df
```

Out[7]:

	Region	Country	Item_Type	Sales_Channel	Order_Priority	Ship_Date	Unit_Cost	Total_Revenue	Total_Profit	No_product
0	Australia and Oceania	Tuvalu	Baby Food	Offline	H	27-08-2010	159.42	2533654.00	951410.50	15892.949442
1	Europe	Russia	Office Supplies	Offline	L	05-08-2014	524.96	1158502.59	224598.75	2206.839740
2	Sub-Saharan Africa	Rwanda	Office Supplies	Offline	L	02-06-2013	524.96	3296425.02	639077.50	6279.383229
3	Sub-Saharan Africa	Angola	Household	Offline	M	27-04-2011	502.54	2798046.49	693911.51	5567.808513
4	Sub-Saharan Africa	Republic of the Congo	Personal Care	Offline	M	25-08-2015	56.67	496101.10	152114.20	8754.210341
5	Sub-Saharan Africa	Cape Verde	Clothes	Offline	H	19-08-2014	35.84	455479.04	306097.92	12708.678571
6	Central America and the Caribbean	Honduras	Household	Offline	H	13-02-2017	502.54	5997054.98	1487261.02	11933.487842

In [8]:
```
plt.bar(df['Unit_Cost'],df['No_product'],color='y')
plt.title("Bar Graphs")
plt.show()
```



jupyter Graphs Last Checkpoint: 08/09/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

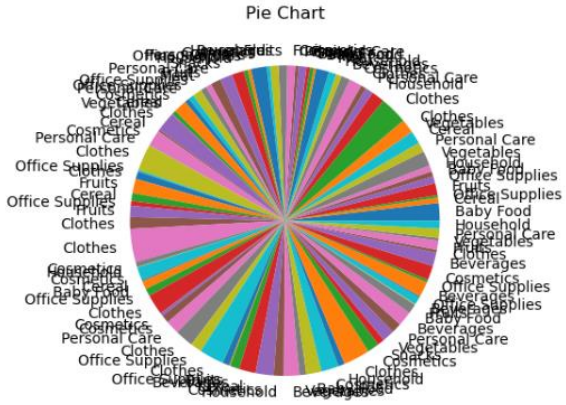
Not Trusted | Python 3 (ipykernel)

Run Code

Task 3: Plot pie chart for item_type and No_product column using Sales dataset.

In [9]:
```
df1=pd.read_csv("sales.csv")
df1["No_product"]=df1["Total_Revenue"]/df1["Unit_Cost"]
```

In [10]:
```
x=plt.pie(df1['No_product'],labels=df1['Item_Type'])
plt.title("Pie Chart")
plt.show()
```



jupyter Graphs Last Checkpoint: 08/09/2023 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

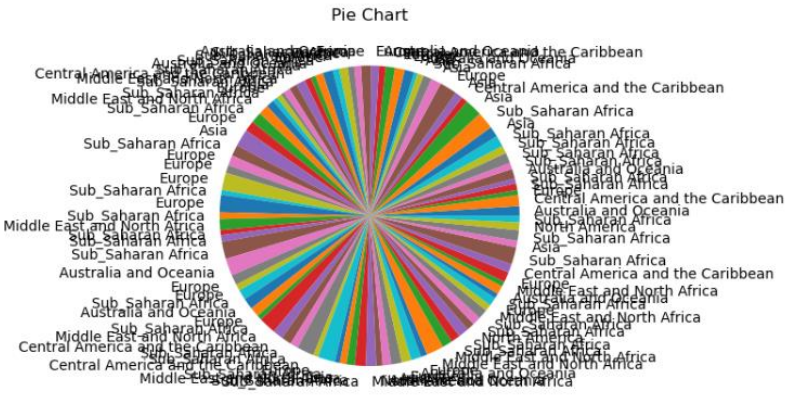
Not Trusted | Python 3 (ipykernel)

Run Code

Task 4: Plot pie chart for Region and Gross_margin column using Sales dataset.

In [11]:
```
df1["Gross_margin"]=df1["Total_Profit"]/df1["Total_Revenue"]
```

In [12]:
```
y=plt.pie(df1['Gross_margin'],labels=df1['Region'])
plt.title("Pie Chart")
plt.show()
```



jupyter Graphs Last Checkpoint: 08/09/2023 (autosaved)

Logout

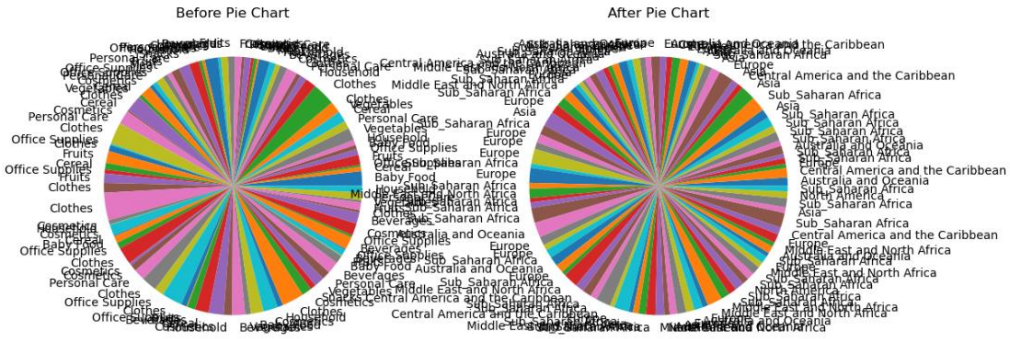
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

Task 5: According to Task 3,4 plot subplot of it.

In [16]:
```
fig,(a1,a2)=plt.subplots(ncols=2,figsize=(12,5))
a1.pie(df1['No_product'],labels=df1['Item_Type'])
a1.set_title("Before Pie Chart")
a2.pie(df1['Gross_margin'],labels=df1['Region'])
a2.set_title("After Pie Chart")
plt.show()
```



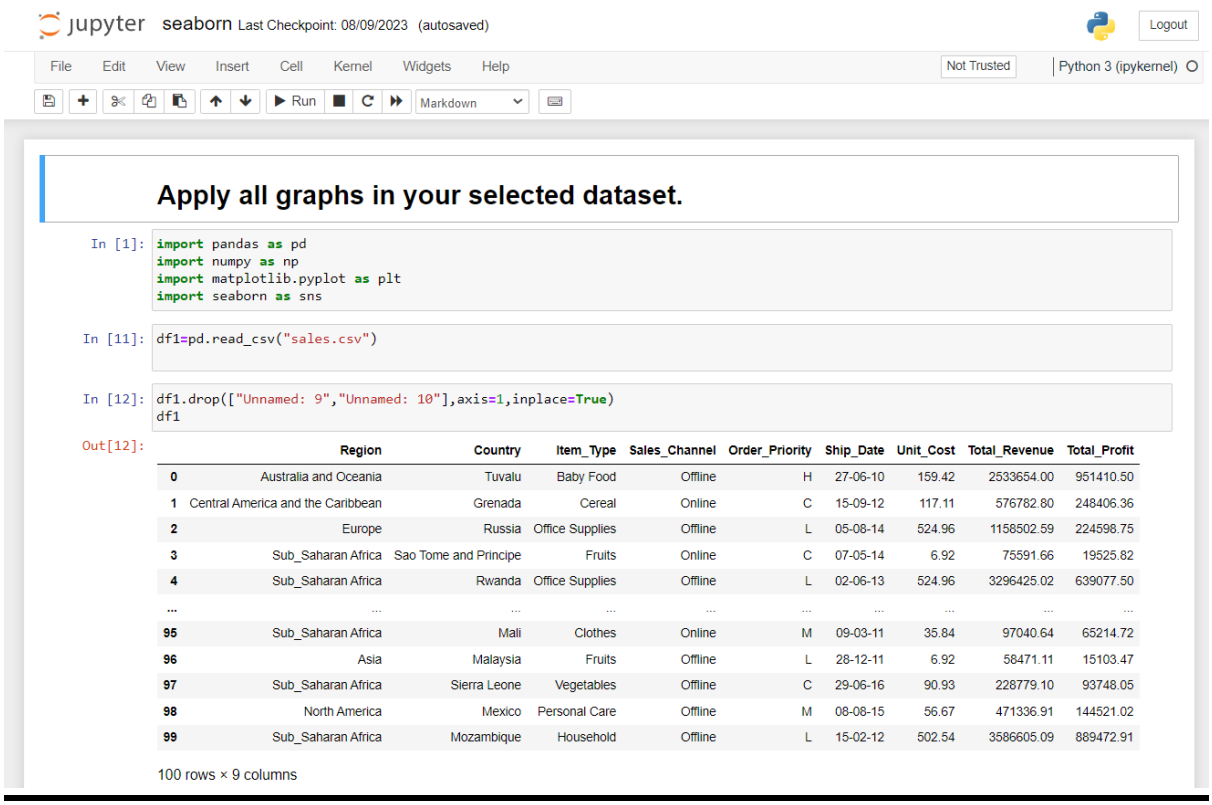
In [ ]:

DATE - 9 AUG

Seaborn Library:

Python Seaborn library is a widely popular data visualization library t hat is commonly used for data science and machine learning tasks. You build it on top of the matplotlib data visualization library and can perform exploratory analysis. You can create interactive plots to ans wer questions about your data.

Tasks:



jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved)

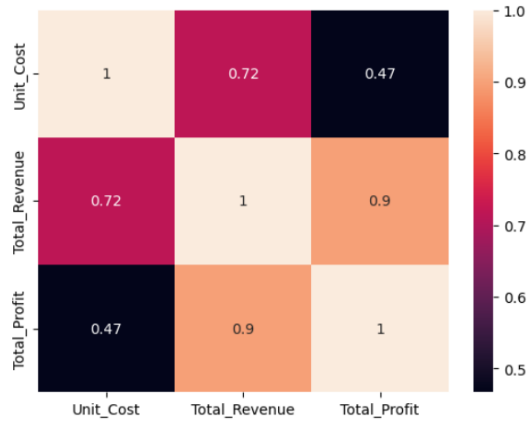
Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

In [24]:
```
sns.heatmap(cr,annot=True,cmap="rocket")
```

Out[24]: <Axes: >



jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved)

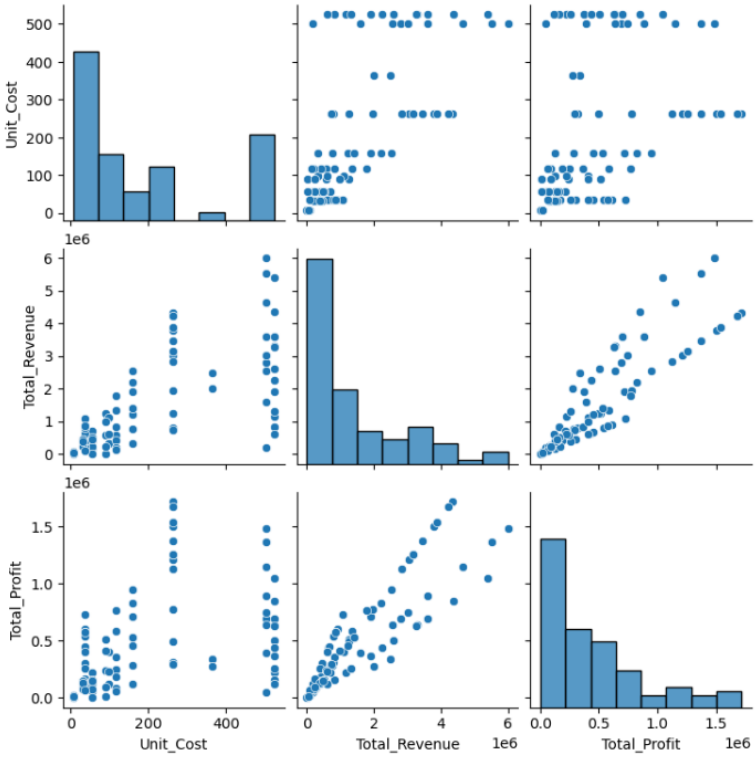
Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

In [17]:
```
sns.pairplot(df1)
```

Out[17]: <seaborn.axisgrid.PairGrid at 0x1a6a7ae02d0>



jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved)

Logout

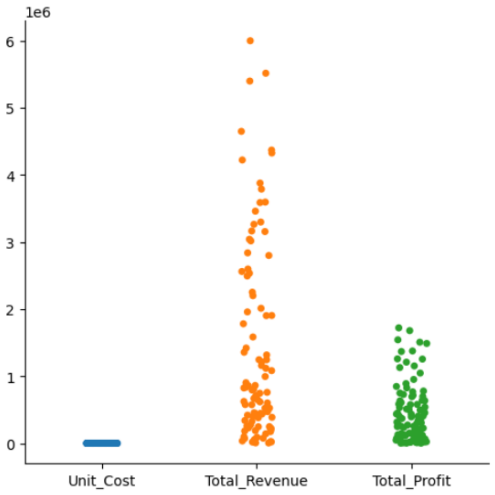
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [18]:
```
sns.catplot(df1)
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x1a6a836ea90>



jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved)

Logout

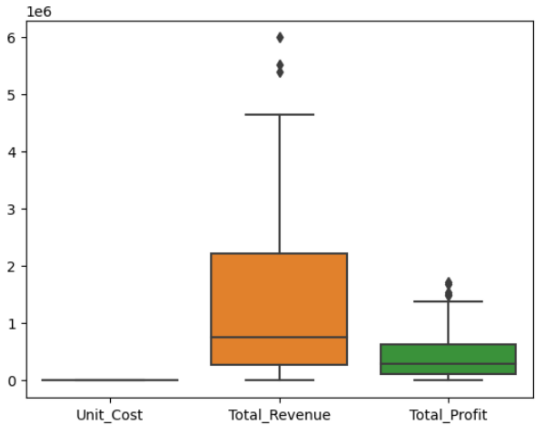
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

In [19]:
```
sns.boxplot(df1)
```

Out[19]: <Axes: >



jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved)

Logout

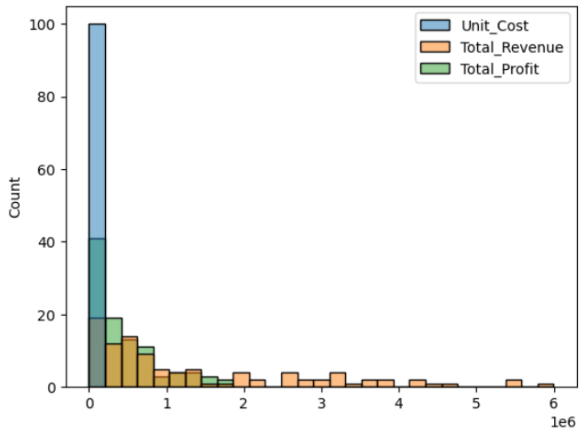
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Markdown

```
In [20]: sns.histplot(df1)
```

```
Out[20]: <Axes: ylabel='Count'>
```



jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved)

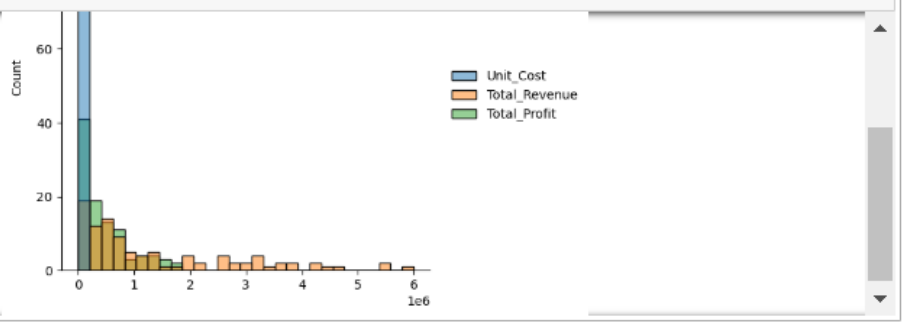
Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

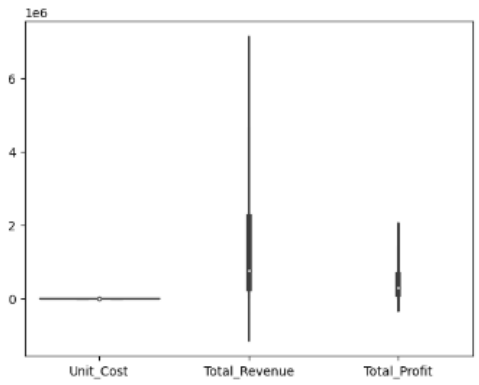
Run Code

```
In [21]: sns.displot(df1)
```




```
In [22]: sns.violinplot(df1)
```

```
Out[22]: <Axes: >
```



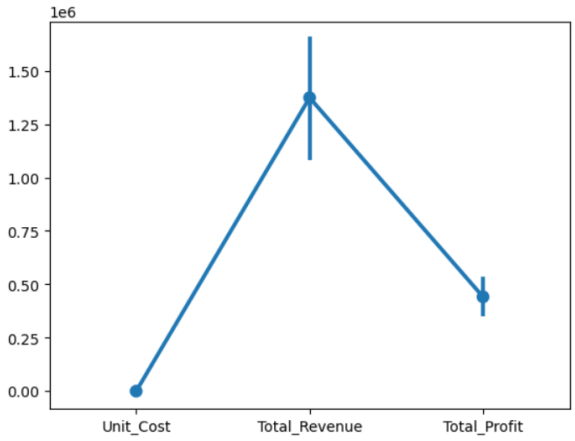
jupyter seaborn Last Checkpoint: 08/09/2023 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [23]:
```
sns.pointplot(df1)
```

Out[23]: <Axes: >

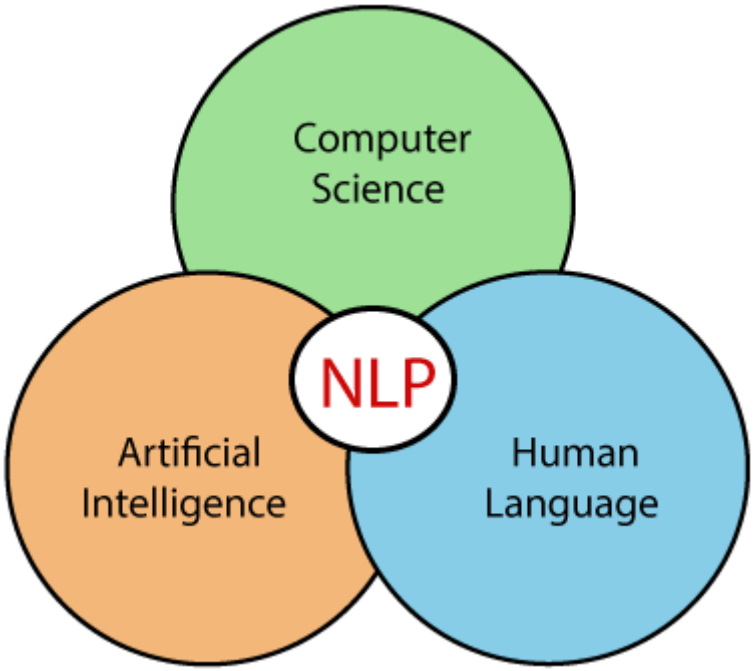


In [ ]:

DATE - 10,11,14 AUG (3 DAY)

What is NLP?

NLP stands for **Natural Language Processing**, which is a part of **Computer Science, Human language,** and **Artificial Intelligence**. It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,** and **topic segmentation**.



Work Tokenize:

With the help of **`nltk.tokenize.word_tokenize()`** method, we are able to extract the tokens from string of characters by using `tokenize.word_tokenize()` method. It actually returns the **syllables** from a single word. A single word can contain one or two syllables.

Example:

```
from nltk.tokenize import word_tokenize
In [10]:tokens=word_tokenize(data)
In [12]:tokens
```

Porter stemmer:

The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems.

Example:

```
from nltk.stem import PorterStemmer
In [27]:
p=PorterStemmer()
for i in tokens:
    print(i+"=="+p.stem(i))
```

Stopwords:

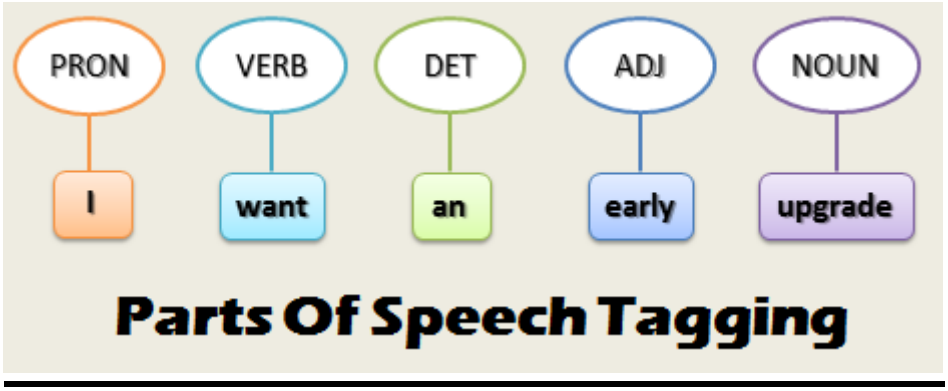
Stop words are words that are so common they are basically ignored by typical tokenizers. By default, NLTK (Natural Language Toolkit) includes a list of 40 stop words, including: “a”, “an”, “the”, “of”, “in”, etc. The stopwords in nltk are the most common words in data.

Example:

Pos Tags:

POS tagging is a technique used in Natural Language Processing. It categorizes the tokens in a text as nouns, verbs, adjectives, and so on. In Python, you can use the NLTK library for this purpose. import nltk. from nltk import word_tokenizetext = "This is one simple example."tokens = word_tokenize(text)

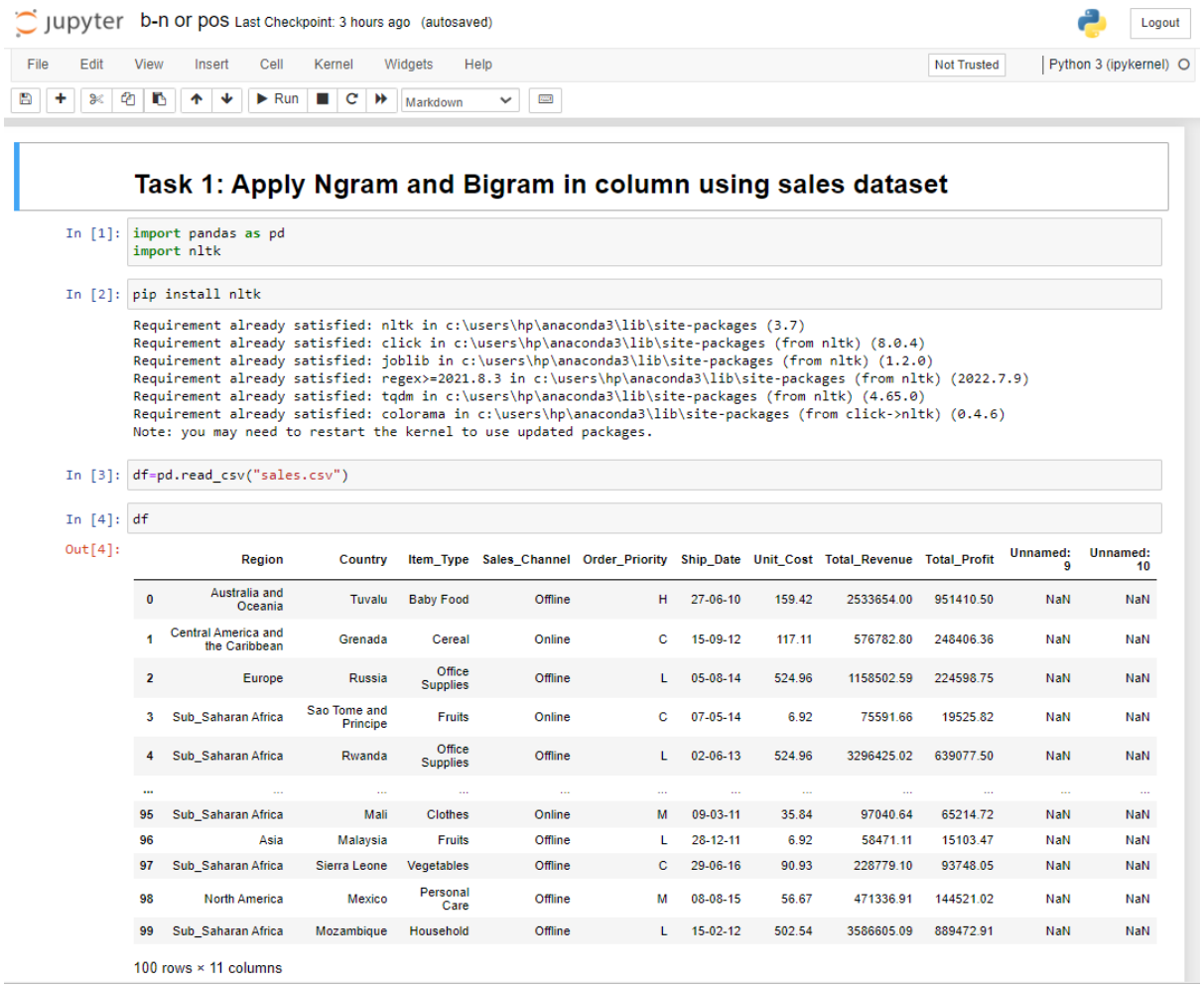
Image How to Use:



Example:

```
In [55]: nltk.pos_tag(tokens)

Out[55]: [('Cows', 'NNS'),
          ('are', 'VBP'),
          ('considered', 'VBN'),
          ('to', 'TO'),
          ('be', 'VB'),
          ('a', 'DT'),
          ('sacred', 'JJ'),
          ('animal', 'NN'),
          ('in', 'IN'),
          ('the', 'DT'),
          ('Hindu', 'NNP'),
          ('religion', 'NN'),
          ('.', '.'),
          ('The', 'DT'),
          ('ardent', 'JJ'),
          ('followers', 'NNS'),
          ('of', 'IN'),
          ('religion', 'NN'),
          ('worship', 'NN'),
          ('this', 'DT'),
          ('animal', 'NN'),
          ('like', 'IN'),
          ('a', 'DT'),
          ('Goddess', 'NNP'),
          ('.', '.'),
          ('A', 'NNP'),
          ('cow', 'NN'),
          ('has', 'VBZ'),
          ('been', 'VBN'),
          ('honored', 'VBN'),
          ('with', 'IN'),
          ('the', 'DT'),
          ('status', 'NN'),
          ('of', 'IN'),
          ('a', 'DT'),
```

Task of NLP:



Task 2: Apply pos_tag using sale dataset

```
In [9]: from nltk.tokenize import word_tokenize

In [10]: tokens=word_tokenize(data)

In [12]: tokens

In [55]: nltk.pos_tag(tokens)

Out[55]: [('Cows', 'NNS'),
 ('are', 'VBP'),
 ('considered', 'VBN'),
 ('to', 'TO'),
 ('be', 'VB'),
 ('a', 'DT'),
 ('sacred', 'JJ'),
 ('animal', 'NN'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('Hindu', 'NNP'),
 ('religion', 'NN'),
 ('.', '.'),
 ('The', 'DT'),
 ('ardent', 'JJ'),
 ('followers', 'NNS'),
 ('of', 'IN'),
 ('religion', 'NN'),
 ('worship', 'NN'),
 ('this', 'DT'),
 ('animal', 'NN'),
 ('like', 'IN'),
 ('a', 'DT'),
 ('Goddess', 'NNP'),
 ('.', '.'),
 ('A', 'NNP'),
 ('cow', 'NN'),
 ('has', 'VBZ'),
 ('been', 'VBN'),
 ('honored', 'VBN'),
 ('with', 'IN'),
 ('the', 'DT'),
 ('status', 'NN'),
 ('of', 'IN'),
 ('a', 'DT'),
```

DATE – 12,13 AUG (SAT-SUN)

DATE – 15 AUG (HOLIDAY)

DATE - 16 AUG

Linear Regression:

Linear regression is a statistical method for modeling relationships between a dependent variable with a given set of independent variables. Note: In this article, we refer to dependent variables as responses and independent variables as features for simplicity.

Example & Task:

```
In [15]: from sklearn.linear_model import LinearRegression

In [16]:  =LinearRegression()

In [17]: model.fit(x,y)

Out[17]: ▾ LinearRegression
         LinearRegression()

In [18]: testing=np.array([11,951410,12]).reshape(-1,1)

In [19]: model.predict(testing)

Out[19]: array([[ 71497.92952284],
                [327922.18745098],
                [ 71498.19904621]])
```

DATE - 17 AUG

(Presentation)

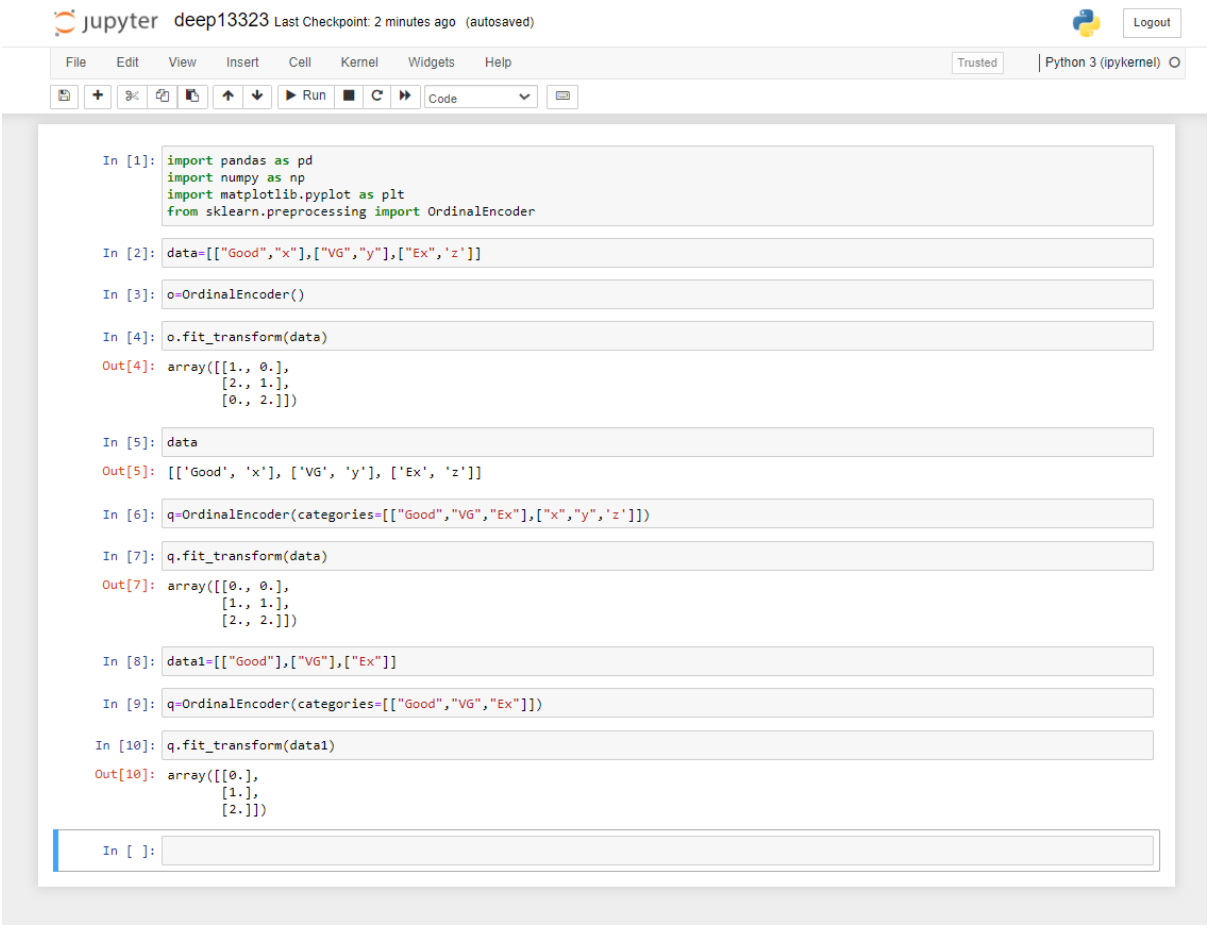
DATE – 18 AUG

Ordinal Encoding Or (Presentation) :

	cost	size	size_endoced
0	50	large	1.0
1	35	small	3.0
2	75	extra large	0.0
3	42	medium	2.0
4	54	large	1.0
5	71	extra large	0.0

Ordinal encoding is a technique to transform categorical features into a numerical format. In ordinal encoding, labels are translated to numbers based on their ordinal relationship to one another.

Example:



DATE – 19,20 AUG (SAT-SUN)

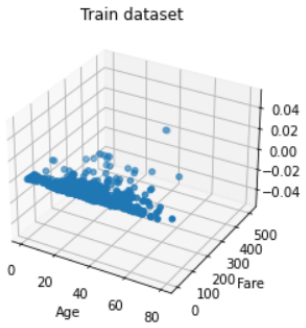
DATE – 21 AUG

Column Transformer:

Column Transformer is a sciket-learn class used to create and apply separate transformers f or numerical and categorical data.

Example:

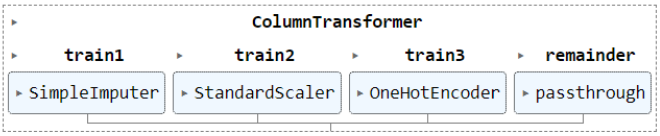
In [4]:
```
fig=plt.figure()
ax=fig.add_subplot(111,projection="3d")
ax.scatter(df["Age"],df["Fare"])
ax.set_xlabel("Age")
ax.set_ylabel("Fare")
ax.set_title("Train dataset")
plt.show()
```



In [5]:
```
transformer=ColumnTransformer(transformers=[
    ("train1",SimpleImputer(),["Age"]),
    ("train2",StandardScaler(),["Fare"]),
    ("train3",OneHotEncoder(sparse_output=True,drop="first"),["Embarked"])
],remainder="passthrough")
```

In [6]:
```
transformer
```

Out[6]:



DATE - 22 AUG

(Presentation)

DAY - 23 AUG

Power Transforms:

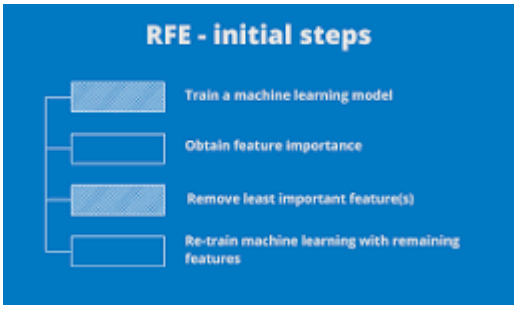
Power transformations are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

Example:

```
In [12]: pt=PowerTransformer()

In [13]: df["Total_Profit"]=pt.fit_transform(df[["Total_Profit"]])

In [14]: x_train,x_test,y_train,y_test=train_test_split(df.drop(["Total_Profit"],axis=1),df["Total_Profit"],test_size=0.3,random_state=42)

In [15]: x_train=pt.fit_transform(x_train)

In [16]: x_test=pt.transform(x_test)

In [17]: y_train

Out[17]: 11    0.058821
         47   -1.794596
         85   -0.416850
         28   -0.839271
         93    1.596595
                 ...
         60   -0.988018
         71   -2.103431
         14   -0.591319
         92   -0.085187
         51   -1.407619
         Name: Total_Profit, Length: 70, dtype: float64
```

DATE – 24 AUG

REF:



Recursive Feature Elimination (RFE) in R is a feature selection technique that iteratively eliminates less important features based on an algorithm and importance-ranking metric to identify the most relevant subset of features.

Example & Tasks:

- Apply all the technique, which is explained in lectures, In your selected dataset.

In [20]:
```
from sklearn.linear_model import LinearRegression
```

In [21]:
```
lr=LinearRegression()
```

In [30]:
```
from sklearn.feature_selection import RFE
r=RFE(lr,n_features_to_select=3)
r.fit(x_train,y_train)
```

Out[30]:

RFE
estimator: LinearRegression
LinearRegression

In [31]:
```
best=r.get_support(indices=True)
best_f=df.columns[best]
```

In [32]:
```
best_f
```

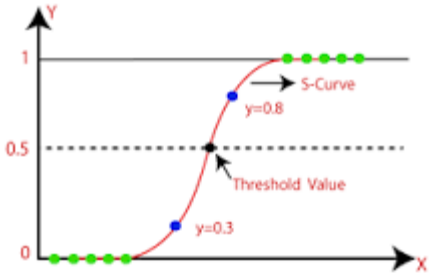
Out[32]:
```
Index(['Unit_Cost', 'Total_Revenue', 'Total_Profit'], dtype='object')
```

DATE – 25 AUG
(Presentation)

DATE – 26,27 AUG (SAT – SUN)

DATE – 28 AUG

Logistic Regression:



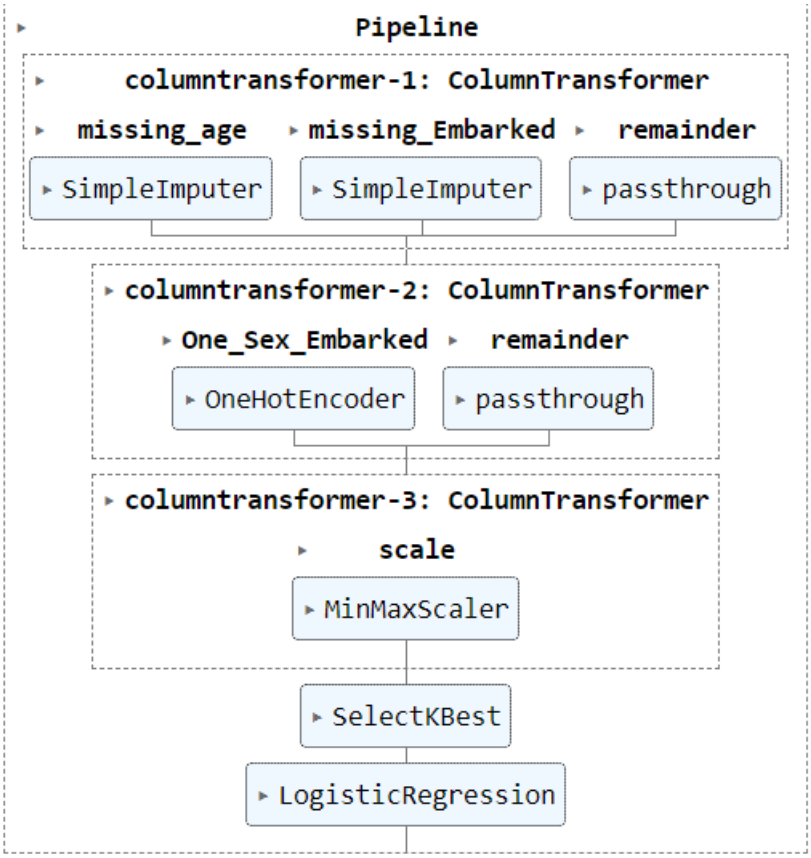
Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable.

Example:

Input:

```
In [26]: from sklearn.linear_model import LogisticRegression
         tr5=LogisticRegression()

In [27]: pipe=make_pipeline(tr1,tr2,tr3,tr4,tr5)

In [28]: pipe.fit(x_train,y_train)
```

Output:



DATE – 29 AUG
(Report Explanation)

Project

- **Project detail:**

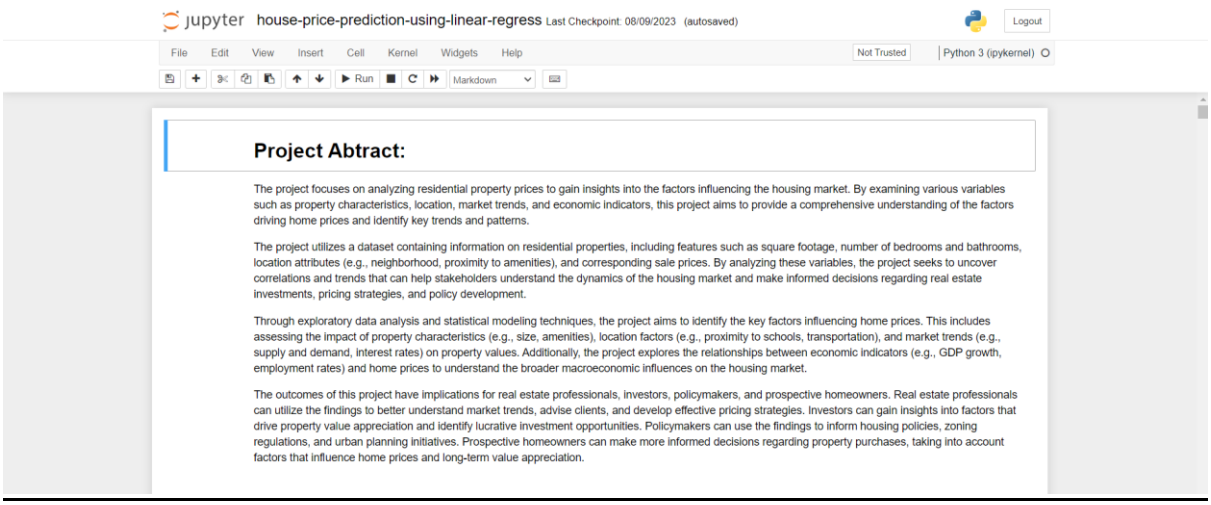
Title:

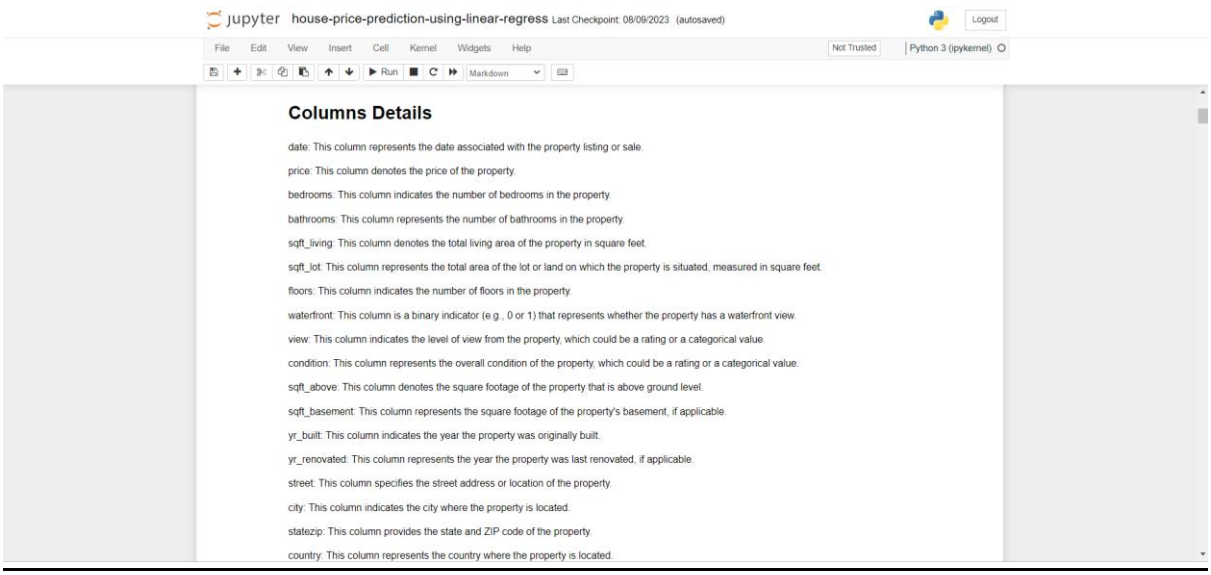
Home Price Analysis

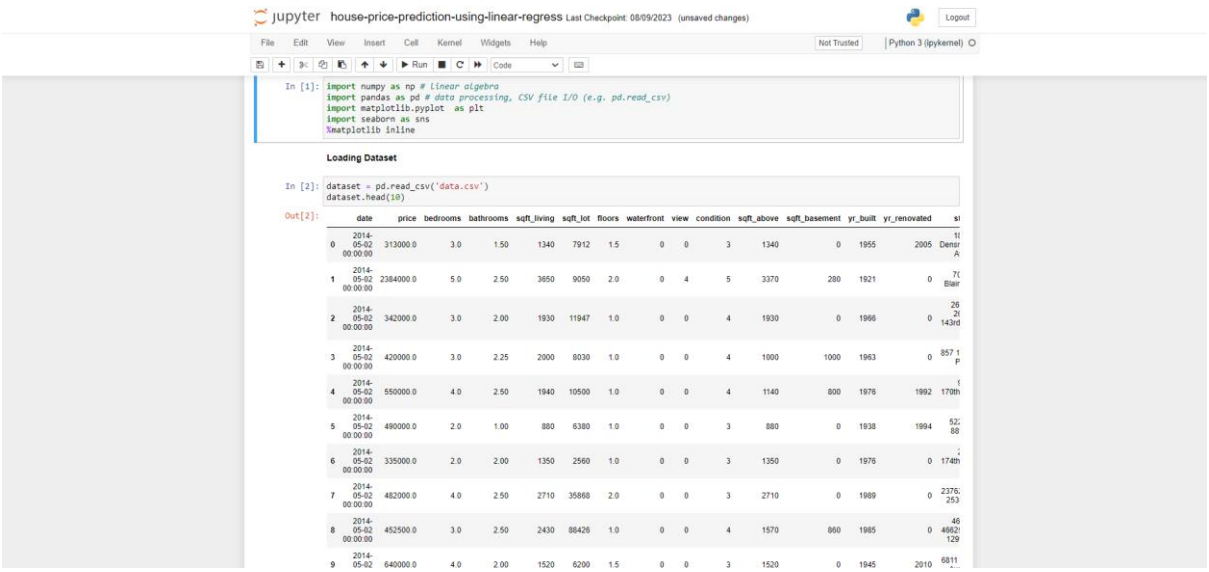
Abstract:

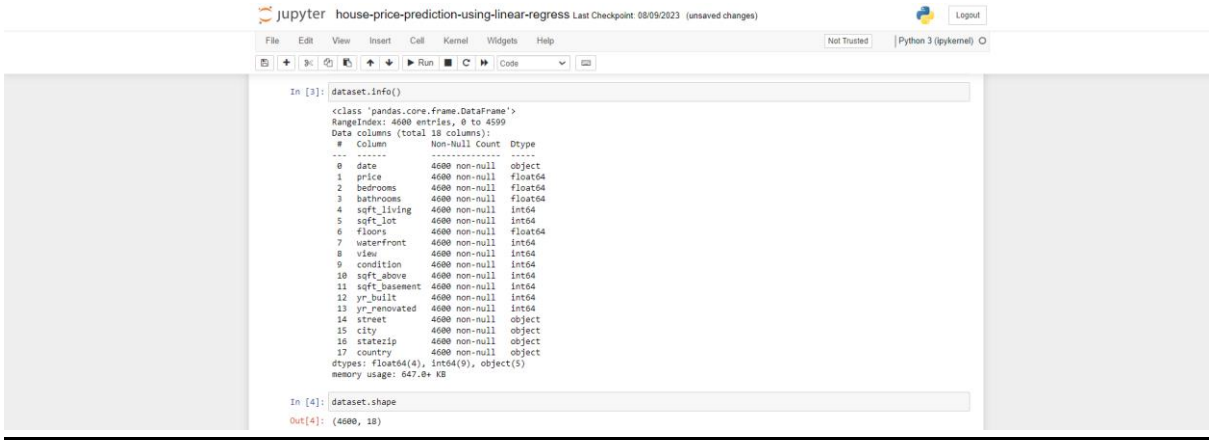
The project focuses on analyzing residential property prices to gain insights into the factors influencing the housing market. By examining various variables such as property characteristics, location, market trends, and economic indicators, this project aims to provide a comprehensive understanding of the factors driving home prices and identify key trends and patterns. The project utilizes a dataset containing information on residential properties, including features such as square footage, number of bedrooms and bathrooms, location attributes (e.g., neighborhood, proximity to amenities), and corresponding sale prices. By analyzing these variables, the project seeks to uncover correlations and trends that can help stakeholders understand the dynamics of the housing market and make informed decisions regarding real estate investments, pricing strategies, and policy development. Through exploratory data analysis and statistical modeling techniques, the project aims to identify the key factors influencing home prices. This includes assessing the impact of property characteristics (e.g., size, amenities), location factors (e.g., proximity to schools, transportation), and market trends (e.g., supply and demand, interest rates) on property values. Additionally, the project explores the relationships between economic indicators (e.g., GDP growth, employment rates) and home prices to understand the broader macroeconomic influences on the housing market. The outcomes of this project have implications for real estate professionals, investors, policymakers, and prospective homeowners. Real estate professionals can utilize the findings to better understand market trends, advise clients, and develop effective pricing strategies. Investors can gain insights into factors that drive property value appreciation and identify lucrative investment opportunities. Policymakers can use the findings to inform housing policies, zoning regulations, and urban planning initiatives. Prospective homeowners can make more informed decisions regarding property purchases, considering factors that influence home prices and long-term value appreciation.

Project Images:


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Project Abtract:
The project focuses on analyzing residential property prices to gain insights into the factors influencing the housing market. By examining various variables such as property characteristics, location, market trends, and economic indicators, this project aims to provide a comprehensive understanding of the factors driving home prices and identify key trends and patterns.
The project utilizes a dataset containing information on residential properties, including features such as square footage, number of bedrooms and bathrooms, location attributes (e.g., neighborhood, proximity to amenities), and corresponding sale prices. By analyzing these variables, the project seeks to uncover correlations and trends that can help stakeholders understand the dynamics of the housing market and make informed decisions regarding real estate investments, pricing strategies, and policy development.
Through exploratory data analysis and statistical modeling techniques, the project aims to identify the key factors influencing home prices. This includes assessing the impact of property characteristics (e.g., size, amenities), location factors (e.g., proximity to schools, transportation), and market trends (e.g., supply and demand, interest rates) on property values. Additionally, the project explores the relationships between economic indicators (e.g., GDP growth, employment rates) and home prices to understand the broader macroeconomic influences on the housing market.
The outcomes of this project have implications for real estate professionals, investors, policymakers, and prospective homeowners. Real estate professionals can utilize the findings to better understand market trends, advise clients, and develop effective pricing strategies. Investors can gain insights into factors that drive property value appreciation and identify lucrative investment opportunities. Policymakers can use the findings to inform housing policies, zoning regulations, and urban planning initiatives. Prospective homeowners can make more informed decisions regarding property purchases, taking into account factors that influence home prices and long-term value appreciation.


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Columns Details
date: This column represents the date associated with the property listing or sale.
price: This column denotes the price of the property.
bedrooms: This column indicates the number of bedrooms in the property.
bathrooms: This column represents the number of bathrooms in the property.
sqft_living: This column denotes the total living area of the property in square feet.
sqft_lot: This column represents the total area of the lot or land on which the property is situated, measured in square feet.
floors: This column indicates the number of floors in the property.
waterfront: This column is a binary indicator (e.g., 0 or 1) that represents whether the property has a waterfront view.
view: This column indicates the level of view from the property, which could be a rating or a categorical value.
condition: This column represents the overall condition of the property, which could be a rating or a categorical value.
sqft_above: This column denotes the square footage of the property that is above ground level.
sqft_basement: This column represents the square footage of the property's basement, if applicable.
yr_built: This column indicates the year the property was originally built.
yr_renovated: This column represents the year the property was last renovated, if applicable.
street: This column specifies the street address or location of the property.
city: This column indicates the city where the property is located.
statezip: This column provides the state and ZIP code of the property.
country: This column represents the country where the property is located.


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [1]:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
Loading Dataset
In [2]:
dataset = pd.read_csv('data.csv')
dataset.head(10)
Out[2]:
date price bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition sqft_above sqft_basement yr_built yr_renovated
0 2014-05-02 00:00:00 313000.0 3.0 1.50 1340 7912 1.5 0 0 3 1340 0 1955 2005
1 2014-05-02 00:00:00 2384000.0 5.0 2.50 3650 9050 2.0 0 4 5 3370 280 1921 0
2 2014-05-02 00:00:00 342000.0 3.0 2.00 1930 11947 1.0 0 0 4 1930 0 1966 0
3 2014-05-02 00:00:00 420000.0 3.0 2.25 2000 8030 1.0 0 0 4 1000 1000 1963 0
4 2014-05-02 00:00:00 550000.0 4.0 2.50 1940 10500 1.0 0 0 4 1140 800 1976 1992
5 2014-05-02 00:00:00 490000.0 2.0 1.00 880 6380 1.0 0 0 3 880 0 1938 1994
6 2014-05-02 00:00:00 335000.0 2.0 2.00 1350 2560 1.0 0 0 3 1350 0 1976 0
7 2014-05-02 00:00:00 482000.0 4.0 2.50 2710 35868 2.0 0 0 3 2710 0 1989 0
8 2014-05-02 00:00:00 452500.0 3.0 2.50 2430 88426 1.0 0 0 4 1570 860 1985 0
9 2014-05-02 640000.0 4.0 2.00 1520 6200 1.5 0 0 3 1520 0 1945 2010

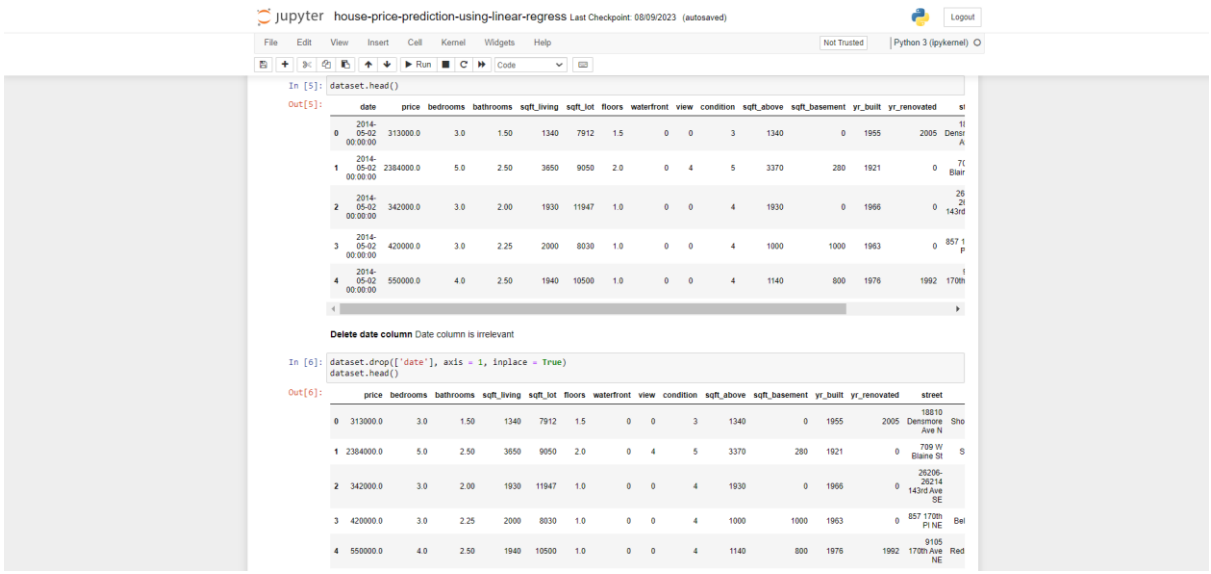


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

```
In [3]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64
 5   sqft_lot       4600 non-null   int64
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64
 8   view           4600 non-null   int64
 9   condition      4600 non-null   int64
 10  sqft_above     4600 non-null   int64
 11  sqft_basement  4600 non-null   int64
 12  yr_built       4600 non-null   int64
 13  yr_renovated   4600 non-null   int64
 14  street         4600 non-null   object
 15  city           4600 non-null   object
 16  statezip       4600 non-null   object
 17  country        4600 non-null   object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB

In [4]: dataset.shape
Out[4]: (4600, 18)
```



jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

```
In [5]: dataset.head()
```

Out[5]:

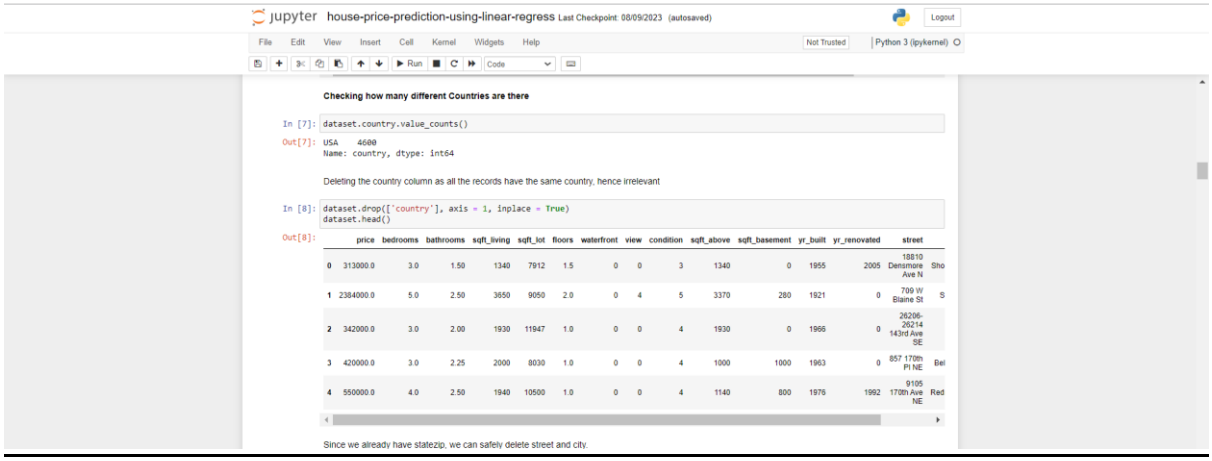
	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	st
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	1(Densr A
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	7(Blair
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26 2(143rd
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 1 P
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	1 170th

Delete date column Date column is irrelevant

```
In [6]: dataset.drop(['date'], axis = 1, inplace = True)
        dataset.head()
```

Out[6]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	street	
0	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18810 Densmore Ave N	Sho
1	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	709 W Blaine St	S
2	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26206-26214 143rd Ave SE	
3	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 170th Pl NE	Bel
4	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9105 170th Ave NE	Red



jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Checking how many different Countries are there

```
In [7]: dataset.country.value_counts()
Out[7]: USA    4600
        Name: country, dtype: int64
```

Deleting the country column as all the records have the same country, hence irrelevant

```
In [8]: dataset.drop(['country'], axis = 1, inplace = True)
        dataset.head()
```

Out[8]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	street	
0	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18810 Densmore Ave N	Sho
1	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	709 W Blaine St	S
2	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26206-26214 143rd Ave SE	
3	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 170th Pl NE	Bel
4	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9105 170th Ave NE	Red

Since we already have statezip, we can safely delete street and city.

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

Since we already have statezip, we can safely delete street and city.

```
In [9]: dataset.drop(['street', 'city'], axis = 1, inplace = True)
        dataset.head()
```

Out[9]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	statezip
0	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	WA 98133
1	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	WA 98119
2	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	WA 98042
3	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	WA 98008
4	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	WA 98052

Checking for null values

```
In [10]: dataset.isnull().sum()
```

```
Out[10]: price          0
         bedrooms       0
         bathrooms      0
         sqft_living    0
         sqft_lot       0
         floors         0
         waterfront     0
         view           0
         condition      0
         sqft_above     0
         sqft_basement  0
         yr_built       0
         yr_renovated   0
         statezip       0
         dtype: int64
```

No null values present

General corellation analysis

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

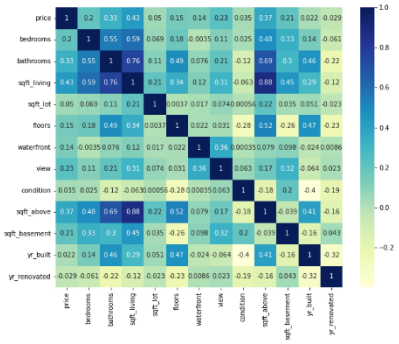
File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

No null values present

General corellation analysis

```
In [11]: a4_dims = (10, 8)
         fig, ax = plt.subplots(figsize=a4_dims)
         cor = dataset.corr()
         sns.heatmap(cor, annot = True, cmap="YlGnBu")
```

```
Out[11]: <Axes: >
```



jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

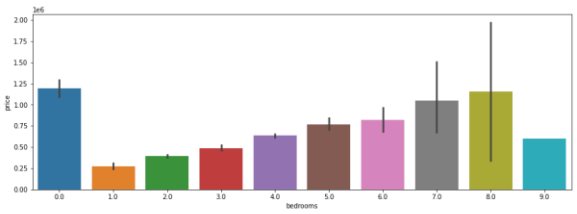
File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

Analysis on number of bedroom feature

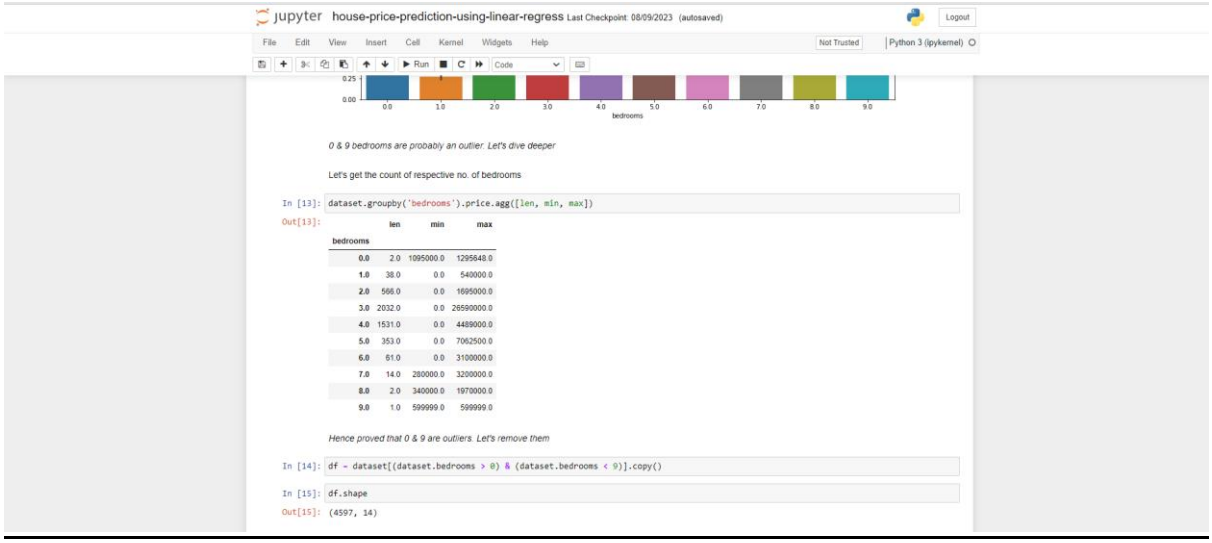
corellation of price with no. of bedrooms

```
In [12]: a4_dims = (15, 5)
         fig, ax = plt.subplots(figsize=a4_dims)
         sns.barplot(x = dataset.bedrooms, y = dataset.price)
```

```
Out[12]: <Axes: xlabel='bedrooms', ylabel='price'>
```



0 & 9 bedrooms are probably an outlier. Let's dive deeper



jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

0 & 9 bedrooms are probably an outlier. Let's dive deeper

Let's get the count of respective no. of bedrooms

```
In [13]: dataset.groupby('bedrooms').price.agg([len, min, max])
```

Out[13]:

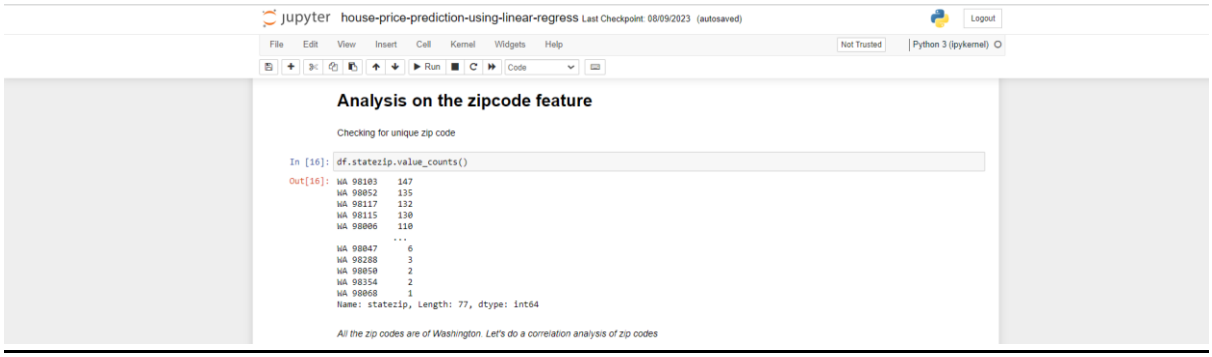
bedrooms	len	min	max
0.0	2.0	1095000.0	1295648.0
1.0	38.0	0.0	540000.0
2.0	566.0	0.0	1695000.0
3.0	2032.0	0.0	26590000.0
4.0	1531.0	0.0	4489000.0
5.0	353.0	0.0	7062500.0
6.0	61.0	0.0	3100000.0
7.0	14.0	280000.0	3200000.0
8.0	2.0	340000.0	1970000.0
9.0	1.0	599999.0	599999.0

Hence proved that 0 & 9 are outliers. Let's remove them

```
In [14]: df = dataset[(dataset.bedrooms > 0) & (dataset.bedrooms < 9)].copy()
```

```
In [15]: df.shape
```

```
Out[15]: (4597, 14)
```



jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

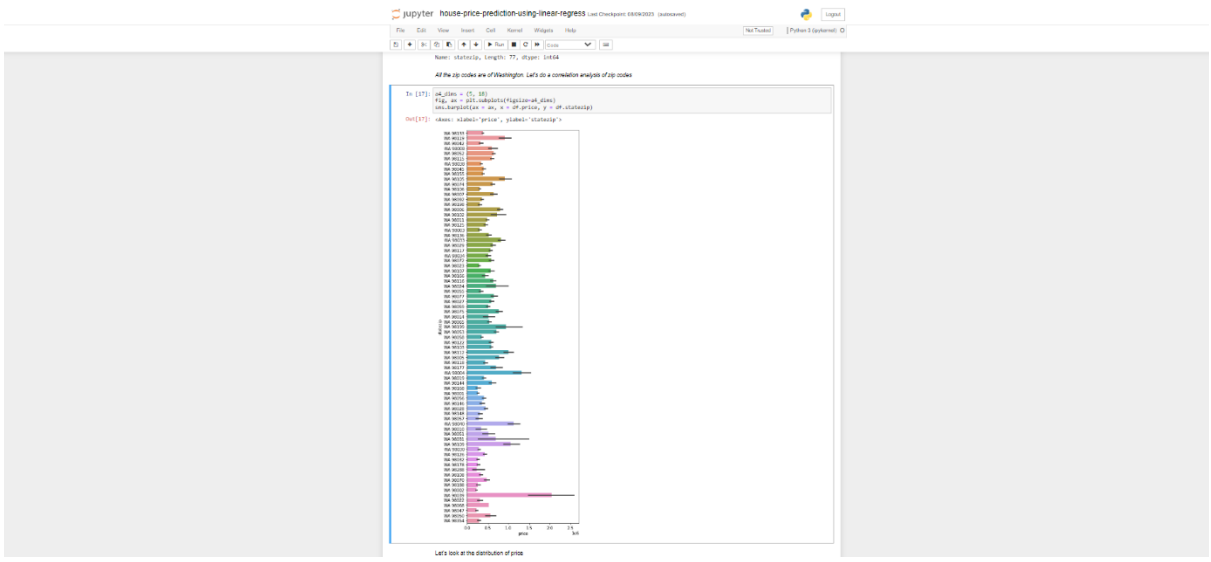
Analysis on the zipcode feature

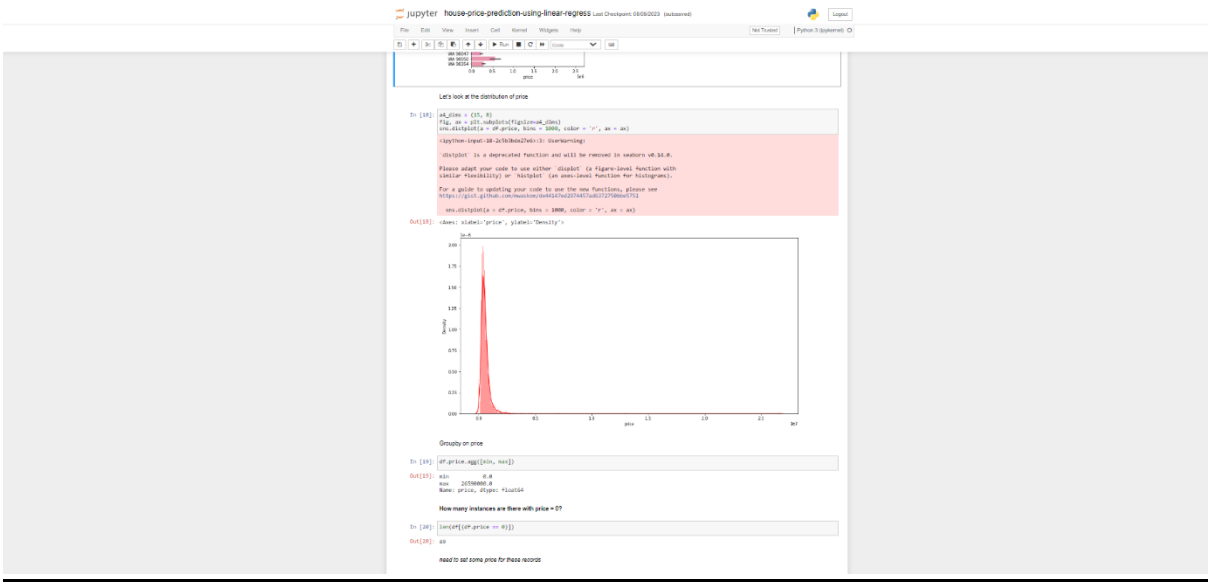
Checking for unique zip code

```
In [16]: df.statezip.value_counts()
```

```
Out[16]: WA 98103    147
         WA 98052    135
         WA 98117    132
         WA 98115    130
         WA 98006    110
                    ...
         WA 98047      6
         WA 98288      3
         WA 98050      2
         WA 98354      2
         WA 98068      1
         Name: statezip, Length: 77, dtype: int64
```

All the zip codes are of Washington. Let's do a correlation analysis of zip codes





Let's look at the distribution of price

```
In [18]: a4_dims = (15, 8)
         fig, ax = plt.subplots(figsize=a4_dims)
         sns.distplot(a = df.price, bins = 1000, color = 'r', ax = ax)
```

```
<ipython-input-18-2cf63bde27e6>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(a = df.price, bins = 1000, color = 'r', ax = ax)
```

```
Out[18]: <Axes: xlabel='price', ylabel='Density'>
```


Groupby on price

```
In [19]: df.price.agg([min, max])
```

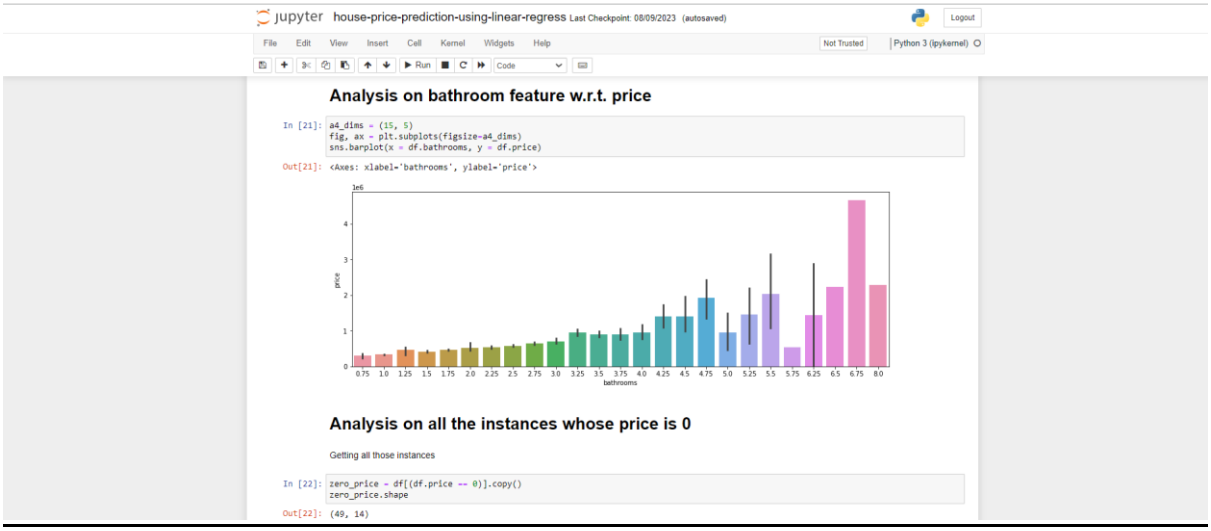
```
Out[19]: min    0.0
         max    26590000.0
         Name: price, dtype: float64
```

How many instances are there with price = 0?

```
In [20]: len(df[(df.price == 0)])
```

```
Out[20]: 49
```

need to set some price for these records



jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

Run Code

Analysis on bathroom feature w.r.t. price

```
In [21]: a4_dims = (15, 5)
         fig, ax = plt.subplots(figsize=a4_dims)
         sns.barplot(x = df.bathrooms, y = df.price)
```

```
Out[21]: <Axes: xlabel='bathrooms', ylabel='price'>
```


Analysis on all the instances whose price is 0

Getting all those instances

```
In [22]: zero_price = df[(df.price == 0)].copy()
         zero_price.shape
```

```
Out[22]: (49, 14)
```

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

```
In [23]: zero_price.head()
```

Out[23]:

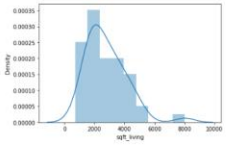
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	statezip
4354	0.0	3.0	1.75	1490	10125	1.0	0	0	4	1490	0	1962	0	WA 98001
4356	0.0	4.0	2.75	2800	5390	1.0	0	0	4	1300	1300	1960	2001	WA 98198
4357	0.0	6.0	2.75	3200	9200	1.0	0	2	4	1600	1600	1953	1983	WA 98146
4358	0.0	5.0	3.50	3480	36615	2.0	0	0	4	2490	990	1983	0	WA 98075
4361	0.0	5.0	1.50	1500	7112	1.0	0	0	5	780	740	1920	0	WA 98166

Let's get the unique value of the most important features

```
In [24]: sns.distplot(zero_price.sqft_living)
```

```
<ipython-input-24-8e1fecfc87a9>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(zero_price.sqft_living)
```

Out[24]: <Axes: xlabel='sqft_living', ylabel='Density'>



Most of the 0 price houses are in the range 1000 - 5000 sqft

Let's find more correlation between the 0 price houses

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

```
In [25]: zero_price.agg(['min', 'max', 'mean', 'median'])
```

Out[25]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovat
min	0.0	1.000000	1.00000	720.000000	3600.000000	1.0	0.000000	0.000000	2.000000	720.000000	0.000000	1920.000000	0.0000
max	0.0	6.000000	6.25000	8020.000000	166320.000000	3.0	1.000000	4.000000	5.000000	8020.000000	1960.000000	2013.000000	2009.0000
mean	0.0	3.870592	2.66998	2787.142857	16453.306122	1.5	0.061224	0.795918	3.673469	2295.714286	491.428571	1969.918367	812.7142
median	0.0	4.000000	2.50000	2600.000000	9000.000000	1.5	0.000000	0.000000	3.000000	1990.000000	0.000000	1962.000000	0.0000

We are going to use common ranges from the above table to get similar records from the original dataset and non-zero price to set the values of 0 price instances

```
In [26]: sim_from_ori = df[(df.bedrooms == 4) & (df.bathrooms > 1) & (df.bathrooms < 4) & (df.sqft_living > 2500) & (df.sqft_living < 300
```

```
In [27]: sim_from_ori.shape
```

Out[27]: (79, 14)

```
In [28]: sim_from_ori.head()
```

Out[28]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	statezip
11	1400000.0	4.0	2.50	2920	4000	1.5	0	0	5	1910	1010	1909	1988	WA 98105
172	407000.0	4.0	2.25	2810	23400	1.0	0	1	3	1710	1100	1959	2004	WA 98198
207	360000.0	4.0	2.00	2580	18768	1.0	0	0	5	2580	0	1965	0	WA 98001
257	749950.0	4.0	2.75	2600	6050	2.0	0	0	5	1960	640	1949	0	WA 98116
261	581000.0	4.0	2.00	2510	13695	1.0	0	0	4	1280	1230	1961	2001	WA 98033

Get the average price of these instances

```
In [29]: sim_from_ori.price.mean()
```

Out[29]: 735475.0378705189

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)

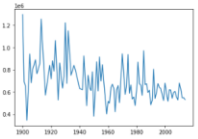
File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

Let's confirm this by comparing with the other house price of the same yr_built and having similar sq_ft

```
In [30]: yr_sqft = df[(df.sqft_living > 2499) & (df.sqft_living < 2900)].copy()
         yr_price_avg = yr_sqft.groupby('yr_built').price.agg('mean')
```

```
In [31]: plt.plot(yr_price_avg)
```

Out[31]: [<matplotlib.lines.Line2D at 0x2064f188850>]



This confirms our assumption. The avg. pricing of such houses is between 600000 to 800000

Replacing all 0 price values with $730000

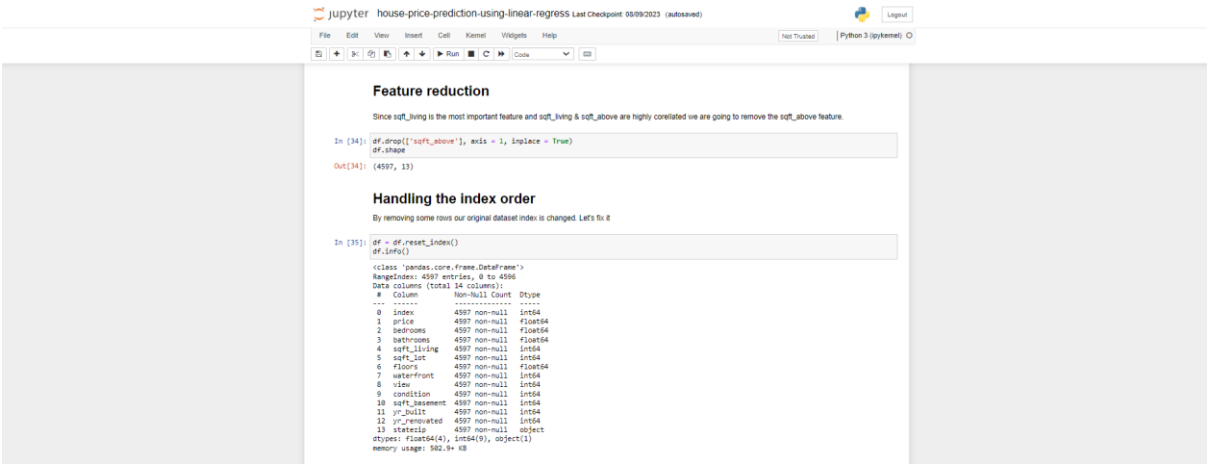
```
In [32]: df.price.replace(to_replace = 0, value = 735000, inplace = True)
         len(df[(df.price == 0)])
```

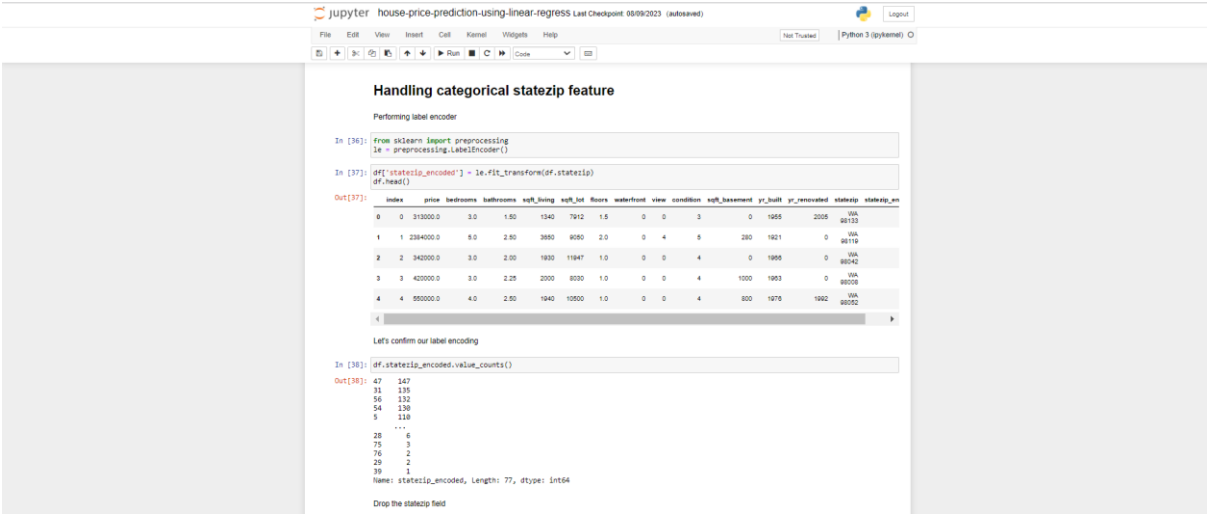
Out[32]: 0

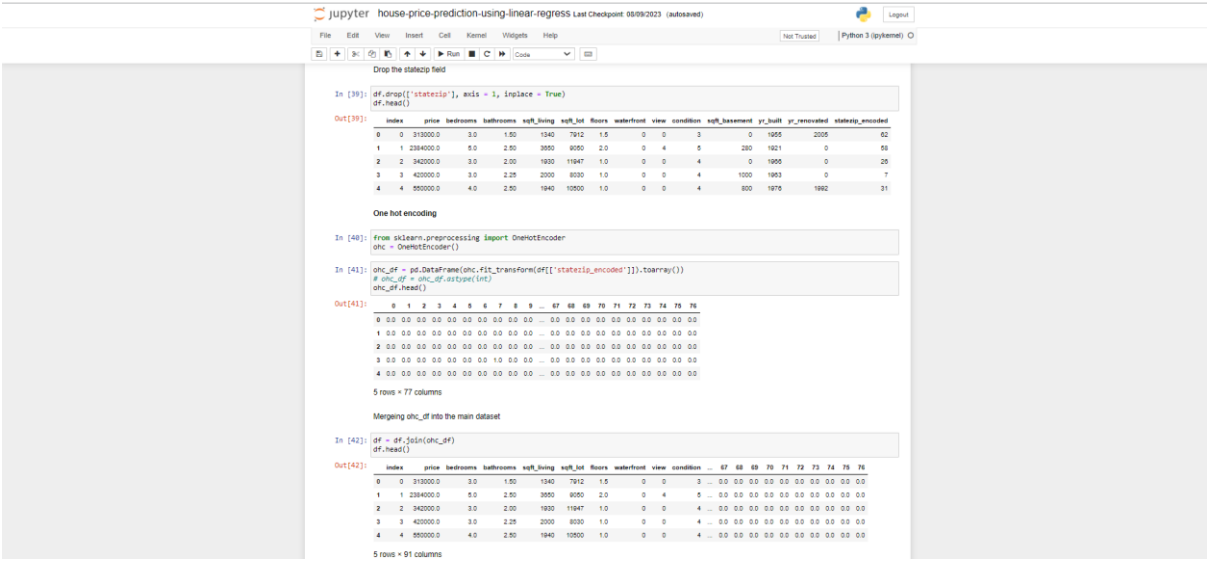
```
In [33]: df.head()
```

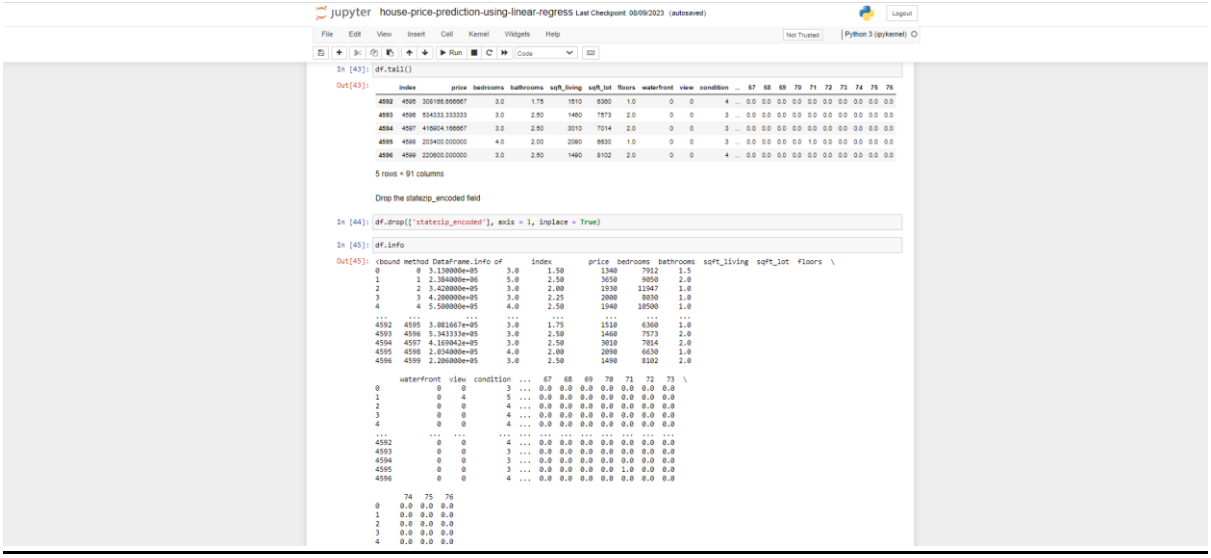
Out[33]:

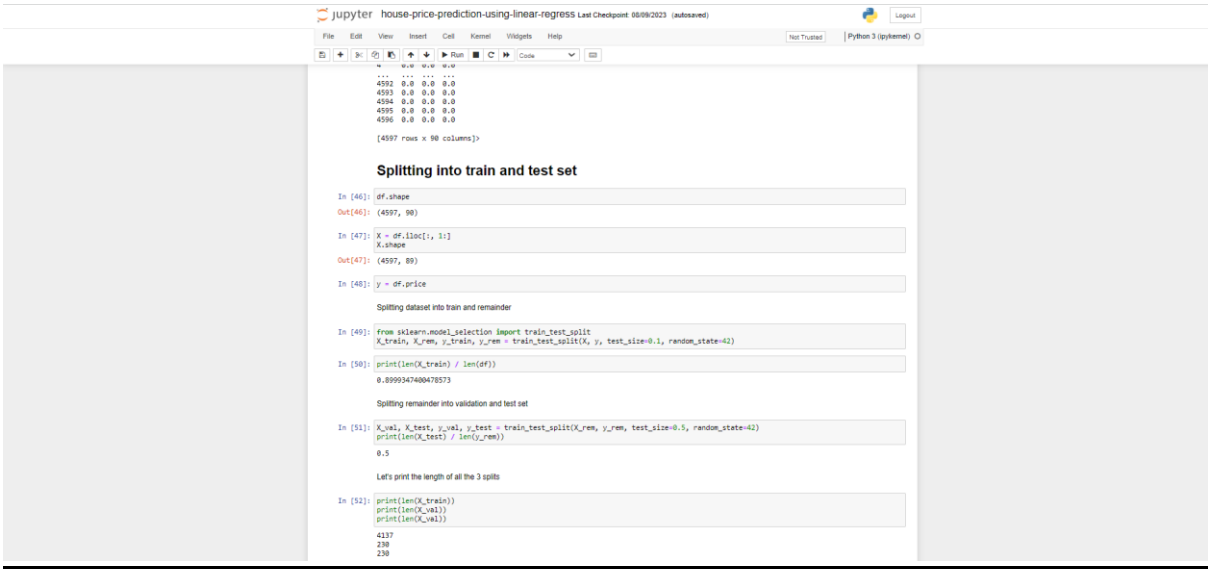
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	statezip
0	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	WA 98133
1	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	WA 98119
2	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	WA 98042
3	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	WA 98008
4	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	WA 98052

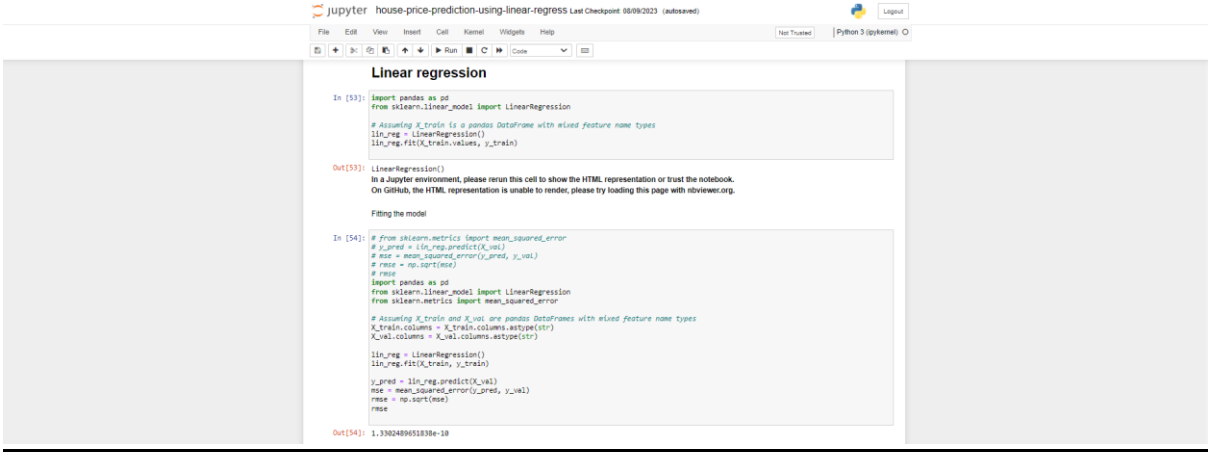

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Feature reduction
Since sqft_living is the most important feature and sqft_living & sqft_above are highly corellated we are going to remove the sqft_above feature.
In [34]:
df.drop(['sqft_above'], axis = 1, inplace = True)
df.shape
Out[34]:
(4597, 13)
Handling the index order
By removing some rows our original dataset index is changed. Let's fix it
In [35]:
df = df.reset_index()
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4597 entries, 0 to 4596
Data columns (total 14 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 index 4597 non-null int64
1 price 4597 non-null float64
2 bedrooms 4597 non-null float64
3 bathrooms 4597 non-null float64
4 sqft_living 4597 non-null int64
5 sqft_lot 4597 non-null int64
6 floors 4597 non-null float64
7 waterfront 4597 non-null int64
8 view 4597 non-null int64
9 condition 4597 non-null int64
10 sqft_basement 4597 non-null int64
11 yr_built 4597 non-null int64
12 yr_renovated 4597 non-null int64
13 statezip 4597 non-null object
dtypes: float64(4), int64(9), object(1)
memory usage: 502.9+ KB

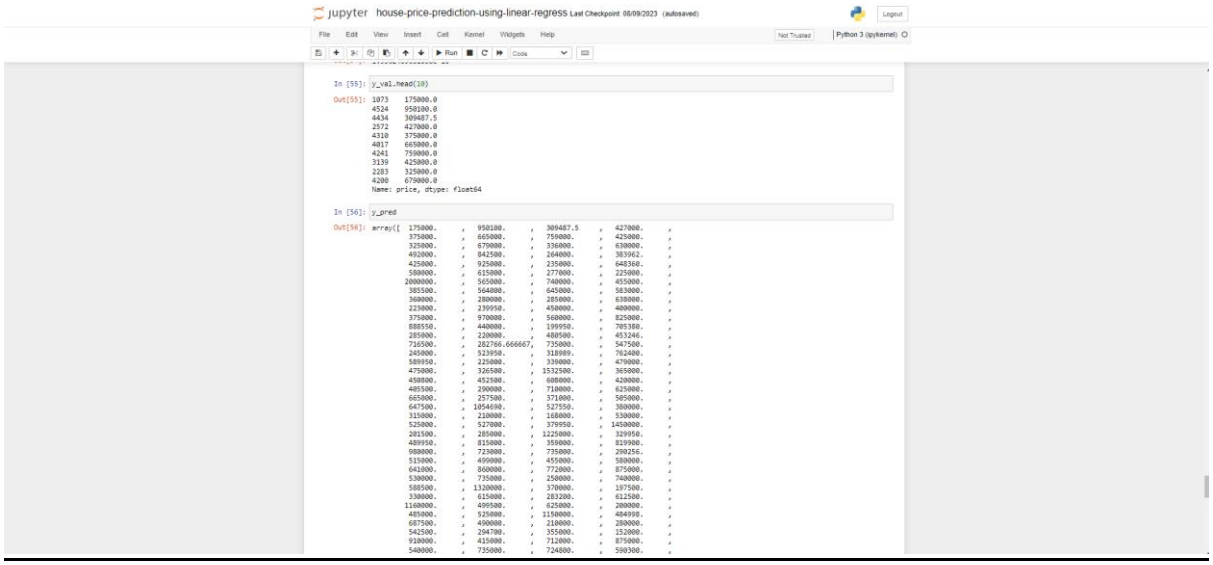

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Handling categorical statezip feature
Performing label encoder
In [36]:
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
In [37]:
df['statezip_encoded'] = le.fit_transform(df.statezip)
df.head()
Out[37]:
index price bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition sqft_basement yr_built yr_renovated statezip statezip_en
0 0 313000.0 3.0 1.50 1340 7912 1.5 0 0 3 0 1955 2005 WA 98133
1 1 2384000.0 5.0 2.50 3650 9050 2.0 0 4 5 280 1921 0 WA 98119
2 2 342000.0 3.0 2.00 1930 11947 1.0 0 0 4 0 1966 0 WA 98042
3 3 420000.0 3.0 2.25 2000 8030 1.0 0 0 4 1000 1963 0 WA 98008
4 4 550000.0 4.0 2.50 1940 10500 1.0 0 0 4 800 1976 1992 WA 98052
Let's confirm our label encoding
In [38]:
df.statezip_encoded.value_counts()
Out[38]:
47 147
31 135
56 132
54 130
5 110
...
28 6
75 3
76 2
29 2
39 1
Name: statezip_encoded, Length: 77, dtype: int64
Drop the statezip field

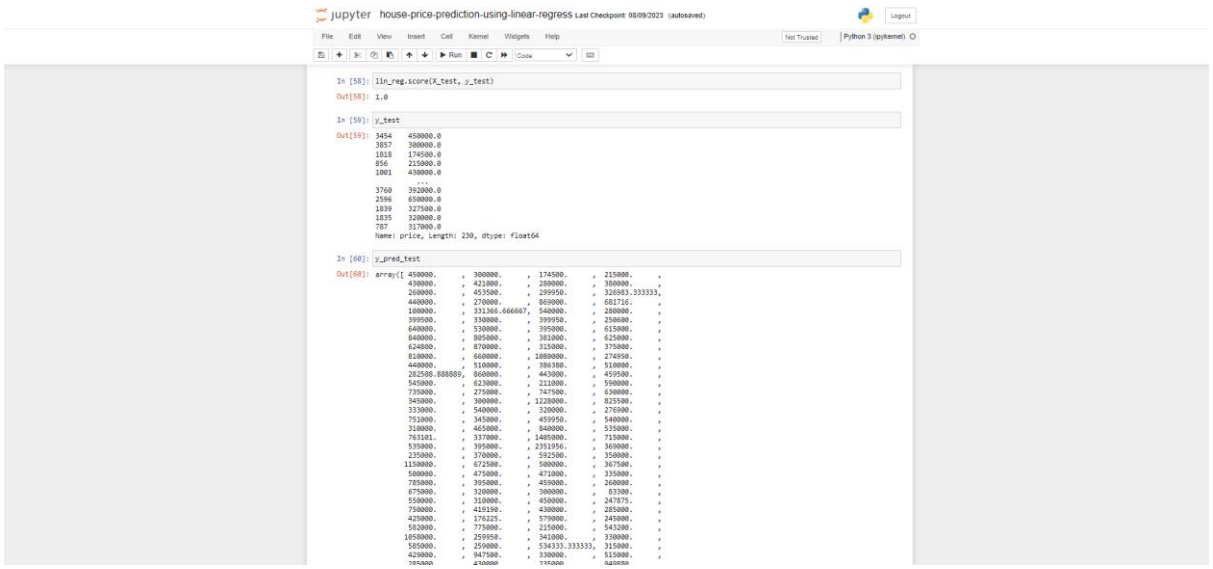

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Drop the statezip field
In [39]:
df.drop(['statezip'], axis = 1, inplace = True)
df.head()
Out[39]:
index price bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition sqft_basement yr_built yr_renovated statezip_encoded
0 0 313000.0 3.0 1.50 1340 7912 1.5 0 0 3 0 1955 2005 62
1 1 2384000.0 5.0 2.50 3650 9050 2.0 0 4 5 280 1921 0 68
2 2 342000.0 3.0 2.00 1930 11947 1.0 0 0 4 0 1966 0 26
3 3 420000.0 3.0 2.25 2000 8030 1.0 0 0 4 1000 1963 0 7
4 4 550000.0 4.0 2.50 1940 10500 1.0 0 0 4 800 1976 1992 31
One hot encoding
In [40]:
from sklearn.preprocessing import OneHotEncoder
ohc = OneHotEncoder()
In [41]:
ohc_df = pd.DataFrame(ohc.fit_transform(df[['statezip_encoded']]).toarray())
# ohc_df = ohc_df.astype(int)
ohc_df.head()
Out[41]:
0 1 2 3 4 5 6 7 8 9 ... 67 68 69 70 71 72 73 74 75 76
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5 rows × 77 columns
Mergeing ohc_df into the main dataset
In [42]:
df = df.join(ohc_df)
df.head()
Out[42]:
index price bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition ... 67 68 69 70 71 72 73 74 75 76
0 0 313000.0 3.0 1.50 1340 7912 1.5 0 0 3 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 1 2384000.0 5.0 2.50 3650 9050 2.0 0 4 5 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 2 342000.0 3.0 2.00 1930 11947 1.0 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 3 420000.0 3.0 2.25 2000 8030 1.0 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 4 550000.0 4.0 2.50 1940 10500 1.0 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5 rows × 91 columns

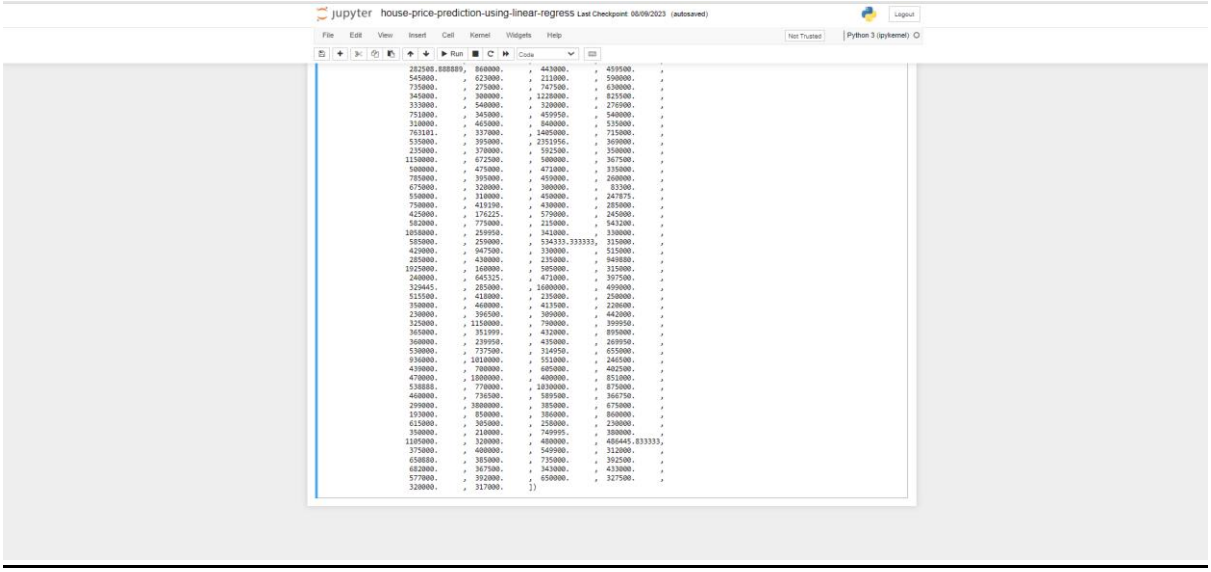

jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [43]: df.tail()
Out[43]:
index price bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition ... 67 68 69 70 71 72 73 74 75 76
4592 4595 308166.666667 3.0 1.75 1510 6360 1.0 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4593 4596 534333.333333 3.0 2.50 1460 7573 2.0 0 0 3 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4594 4597 416904.166667 3.0 2.50 3010 7014 2.0 0 0 3 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4595 4598 203400.000000 4.0 2.00 2090 6630 1.0 0 0 3 ... 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
4596 4599 220600.000000 3.0 2.50 1490 8102 2.0 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5 rows × 91 columns
Drop the statezip_encoded field
In [44]: df.drop(['statezip_encoded'], axis = 1, inplace = True)
In [45]: df.info
Out[45]: <bound method DataFrame.info of index price bedrooms bathrooms sqft_living sqft_lot floors \
0 0 3.130000e+05 3.0 1.50 1340 7912 1.5
1 1 2.384000e+06 5.0 2.50 3650 9050 2.0
2 2 3.420000e+05 3.0 2.00 1930 11947 1.0
3 3 4.200000e+05 3.0 2.25 2000 8030 1.0
4 4 5.500000e+05 4.0 2.50 1940 10500 1.0
... ...
4592 4595 3.081667e+05 3.0 1.75 1510 6360 1.0
4593 4596 5.343333e+05 3.0 2.50 1460 7573 2.0
4594 4597 4.169042e+05 3.0 2.50 3010 7014 2.0
4595 4598 2.034000e+05 4.0 2.00 2090 6630 1.0
4596 4599 2.206000e+05 3.0 2.50 1490 8102 2.0
waterfront view condition ... 67 68 69 70 71 72 73 \
0 0 0 3 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1 0 4 5 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
... ...
4592 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4593 0 0 3 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4594 0 0 3 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4595 0 0 3 ... 0.0 0.0 0.0 0.0 1.0 0.0 0.0
4596 0 0 4 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
74 75 76
0 0.0 0.0 0.0
1 0.0 0.0 0.0
2 0.0 0.0 0.0
3 0.0 0.0 0.0
4 0.0 0.0 0.0


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
4592 0.0 0.0 0.0
4593 0.0 0.0 0.0
4594 0.0 0.0 0.0
4595 0.0 0.0 0.0
4596 0.0 0.0 0.0
[4597 rows x 90 columns]
Splitting into train and test set
In [46]: df.shape
Out[46]: (4597, 90)
In [47]: X = df.iloc[:, 1:]
X.shape
Out[47]: (4597, 89)
In [48]: y = df.price
Splitting dataset into train and remainder
In [49]: from sklearn.model_selection import train_test_split
X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.1, random_state=42)
In [50]: print(len(X_train) / len(df))
0.8999347400478573
Splitting remainder into validation and test set
In [51]: X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.5, random_state=42)
print(len(X_test) / len(y_rem))
0.5
Let's print the length of all the 3 splits
In [52]: print(len(X_train))
print(len(X_val))
print(len(X_val))
4137
230
230


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Linear regression
In [53]: import pandas as pd
from sklearn.linear_model import LinearRegression
# Assuming X_train is a pandas DataFrame with mixed feature name types
lin_reg = LinearRegression()
lin_reg.fit(X_train.values, y_train)
Out[53]: LinearRegression()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
Fitting the model
In [54]: # from sklearn.metrics import mean_squared_error
# y_pred = lin_reg.predict(X_val)
# mse = mean_squared_error(y_pred, y_val)
# rmse = np.sqrt(mse)
# rmse
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Assuming X_train and X_val are pandas DataFrames with mixed feature name types
X_train.columns = X_train.columns.astype(str)
X_val.columns = X_val.columns.astype(str)
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_val)
mse = mean_squared_error(y_pred, y_val)
rmse = np.sqrt(mse)
rmse
Out[54]: 1.3302489651838e-10


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
In [55]: y_val.head(10)
Out[55]: 1073 175000.0
4524 950100.0
6434 309487.5
2572 427000.0
4310 375000.0
4017 665000.0
4241 759000.0
3139 425000.0
2283 325000.0
4200 679000.0
Name: price, dtype: float64
In [56]: y_pred
Out[56]: array([175000. , 950100. , 309487.5 , 427000. ,
375000. , 665000. , 759000. , 425000. ,
325000. , 679000. , 336000. , 630000. ,
402000. , 842500. , 264000. , 383962. ,
425000. , 925000. , 235000. , 648360. ,
580000. , 615000. , 277000. , 225000. ,
2000000. , 565000. , 740000. , 455000. ,
385500. , 564000. , 645000. , 583000. ,
360000. , 280000. , 285000. , 630000. ,
223000. , 239950. , 450000. , 400000. ,
375000. , 970000. , 560000. , 825000. ,
888550. , 440000. , 199950. , 795388. ,
285000. , 220000. , 480500. , 453246. ,
716500. , 282766.666667, 735000. , 547500. ,
245000. , 523950. , 318989. , 762400. ,
589950. , 225000. , 339000. , 479000. ,
475000. , 326500. , 1532500. , 365000. ,
458000. , 452500. , 608000. , 420000. ,
405500. , 290000. , 710000. , 625000. ,
665000. , 257500. , 371000. , 595000. ,
647500. , 1054690. , 527550. , 380000. ,
315000. , 210000. , 168000. , 530000. ,
525000. , 527000. , 379950. , 1450000. ,
201500. , 285000. , 1225000. , 329950. ,
489950. , 815000. , 359000. , 819900. ,
980000. , 723000. , 735000. , 290256. ,
515000. , 499000. , 455000. , 580000. ,
641000. , 860000. , 772000. , 875000. ,
530000. , 735000. , 250000. , 740000. ,
588500. , 1320000. , 370000. , 197500. ,
330000. , 615000. , 283200. , 612500. ,
1160000. , 499500. , 625000. , 200000. ,
485000. , 525000. , 1150000. , 484998. ,
687500. , 490000. , 210000. , 280000. ,
542500. , 294700. , 355000. , 152000. ,
910000. , 415000. , 712000. , 875000. ,
540000. , 735000. , 724800. , 590300. ,
288400. , 570000. , 490000. , 530000. ,
375000. , 669000. , 1300000. , 890000. ,
300000. , 671000. , 812000. , 487028. ,
285000. , 375000. , 899000. , 405000. ,
712000. , 359950. , 698500. , 735000. ,
1325000. , 1100000. , 910000. , 209950. ,
550000. , 451000. , 695000. , 405000. ,
385000. , 384400. , 519000. , 695000. ,
1220000. , 331950. , 885000. , 282000. ,
315000. , 255000. , 960000. , 875000. ,
410000. , 263900. , 210000. , 397000. ,
580000. , 455000. , 520000. , 405000. ,
482000. , 635200. , 469000. , 645000. ,
413000. , 985000. , 391400. , 180000. ,
584000. , 435000. , 26590000. , 337500. ,
385000. , 268971.875 , 2280000. , 255000. ,
790000. , 418000. , 772650. , 849900. ,
1225000. , 810000. , 705000. , 799000. ,
306000. , 265000. , 365000. , 688000. ,
234950. , 440000. , 330000. , 320000. ,
232603. , 433111.111111])
In [57]: # y_pred_test = (lin_reg.predict(X_test)
# mse = mean_squared_error(y_pred_test, y_test)
# rmse = np.sqrt(mse)
# rmse
# import pandas as pd
# from sklearn.metrics import mean_squared_error
# # Assuming X_test is a pandas DataFrame with mixed feature name types
X_test.columns = X_test.columns.astype(str)
y_pred_test = lin_reg.predict(X_test)
mse = mean_squared_error(y_pred_test, y_test)
rmse = np.sqrt(mse)
rmse
Out[57]: 1.1442510148993517e-10


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
In [58]: lin_reg.score(X_test, y_test)
Out[58]: 1.0
In [59]: y_test
Out[59]: 3454 450000.0
3857 300000.0
1818 174500.0
856 215000.0
1001 430000.0
...
3760 392000.0
2596 650000.0
1839 327500.0
1835 320000.0
787 317000.0
Name: price, Length: 230, dtype: float64
In [60]: y_pred_test
Out[60]: array([450000. , 300000. , 174500. , 215000. ,
430000. , 421000. , 280000. , 380000. ,
260000. , 453500. , 299950. , 326983.333333,
440000. , 270000. , 869000. , 681716. ,
180000. , 331366.666667, 540000. , 280000. ,
369500. , 330000. , 399950. , 250680. ,
640000. , 530000. , 395000. , 615000. ,
840000. , 805000. , 381000. , 625000. ,
624000. , 870000. , 315000. , 375000. ,
810000. , 660000. , 1080000. , 274950. ,
440000. , 510000. , 386380. , 510000. ,
282596.888889, 860000. , 443000. , 459500. ,
545000. , 623000. , 211000. , 590000. ,
735000. , 275000. , 747500. , 630000. ,
345000. , 300000. , 1220000. , 825500. ,
333000. , 540000. , 320000. , 276900. ,
751000. , 345000. , 459950. , 540000. ,
310000. , 465000. , 840000. , 535000. ,
763101. , 337000. , 1405000. , 715000. ,
535900. , 395000. , 2351956. , 369000. ,
235000. , 370000. , 592500. , 350000. ,
1150000. , 672500. , 500000. , 367500. ,
580000. , 475000. , 471000. , 335000. ,
785000. , 395000. , 459000. , 260000. ,
675000. , 320000. , 300000. , 83300. ,
550000. , 310000. , 450000. , 247875. ,
750000. , 419190. , 430000. , 285000. ,
425000. , 176225. , 579000. , 245000. ,
582000. , 775000. , 215000. , 543200. ,
1058000. , 259950. , 341000. , 330000. ,
585000. , 259000. , 534333.333333, 315000. ,
420000. , 947500. , 330000. , 515000. ,
285000. , 430000. , 335000. , 940000. ,


jupyter house-price-prediction-using-linear-regress Last Checkpoint: 08/09/2023 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

---THE END---