

# DATA PRE PROCESSING

Team id	PNT2022TMID14821
Project name	AI powered Food Demand Forecaster

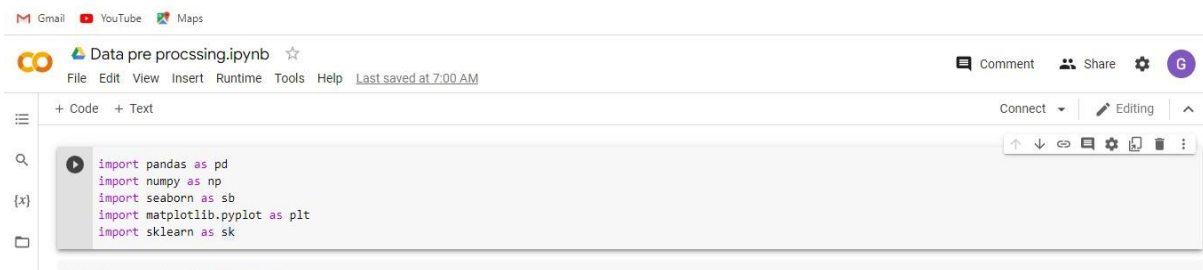
## 1.Importing The Libraries:

**Pandas:** It is a python library mainly used for data manipulation.

**NumPy:** This python library is used for numerical analysis.

**Matplotlib and Seaborn:** Both are the data visualization library used for plotting graph which will help us for understanding the data.

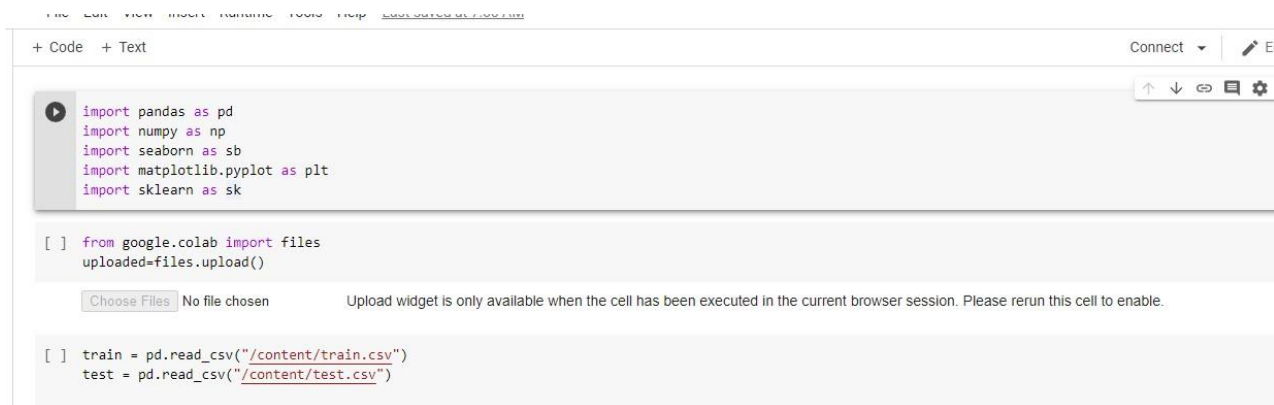
**Pickle:**to serialize your machine learning algorithms and save the serialized format to a file.



```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import sklearn as sk
```

## 2.Reading The Dataset:

- first step will be to read it into a data structure that's compatible with pandas.
- Let's load a .csv data file into pandas. There is a function for it, called **read\_csv()**. We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).



```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import sklearn as sk

[ ] from google.colab import files
    uploaded=files.upload()

[ ] train = pd.read_csv("/content/train.csv")
    test = pd.read_csv("/content/test.csv")
```

## 3.Exploratory Data Analysis:

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods and used for determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions. **head()** :To check first five rows of dataset, we have a function call **head()**.

```
train.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
0	1379560	1.0	55.0	1885.0	136.83	152.29	0.0	0.0	177.0
1	1466964	1.0	55.0	1993.0	136.83	135.83	0.0	0.0	270.0
2	1346989	1.0	55.0	2539.0	134.86	135.86	0.0	0.0	189.0
3	1338232	1.0	55.0	2139.0	339.50	437.53	0.0	0.0	54.0
4	1448490	1.0	55.0	2631.0	243.50	242.50	0.0	0.0	40.0

```
test.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured
0	1028232	146	55	1885	158.11	159.11	0	0
1	1127204	146	55	1993	160.11	159.11	0	0
2	1212707	146	55	2539	157.14	159.14	0	0
3	1082698	146	55	2631	162.02	162.02	0	0
4	1400926	146	55	1248	163.93	163.93	0	0

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103621 entries, 0 to 103620
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   id                     103621 non-null int64  
1   week                   103620 non-null float64 
2   center_id              103620 non-null float64 
3   meal_id                103620 non-null float64 
4   checkout_price          103620 non-null float64 
5   base_price              103620 non-null float64 
6   emailer_for_promotion   103620 non-null float64 
7   homepage_featured       103620 non-null float64 
8   num_orders              103620 non-null float64 
dtypes: float64(8), int64(1)
memory usage: 7.1 MB
```

```
train['num_orders'].describe()
```

count	103620.000000
mean	261.858483
std	433.910688
min	13.000000
25%	54.000000
50%	136.000000
75%	323.000000
max	24299.000000
Name:	num_orders, dtype: float64

```
train['num_orders'].describe()
```

#### 4. Checking For Null Values:

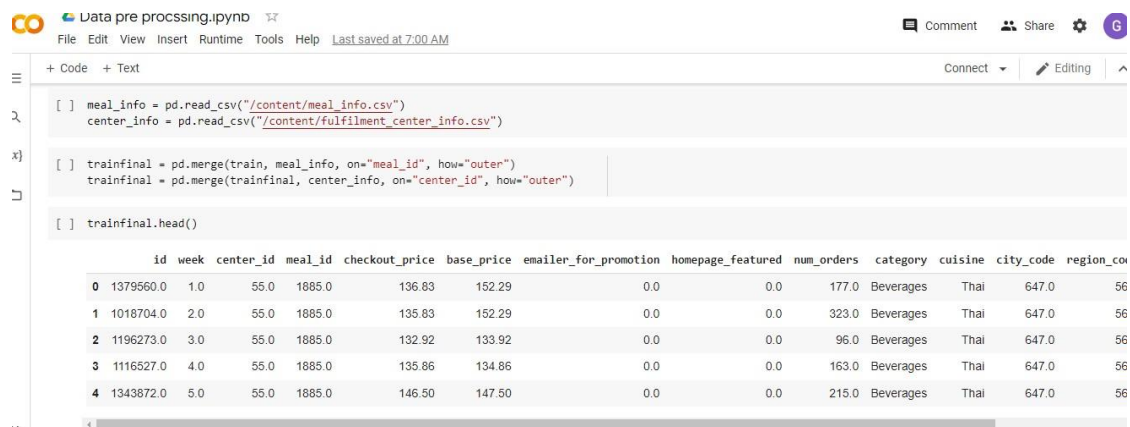
- Imputing data using Imputation method in **sklearn**
- Filling **NaN** values with mean, median and mode using **fillna()** method.

We will be using **isnull().sum()** method to see which total number of missing values.

```
[ ] train.isnull().sum()
```

```
id          0
week        1
center_id   1
meal_id     1
checkout_price 1
base_price  1
emailer_for_promotion 1
homepage_featured 1
num_orders  1
dtype: int64
```

## 5. Reading And Merging .Csv Files:



The screenshot shows a Jupyter Notebook interface with the following code and output:

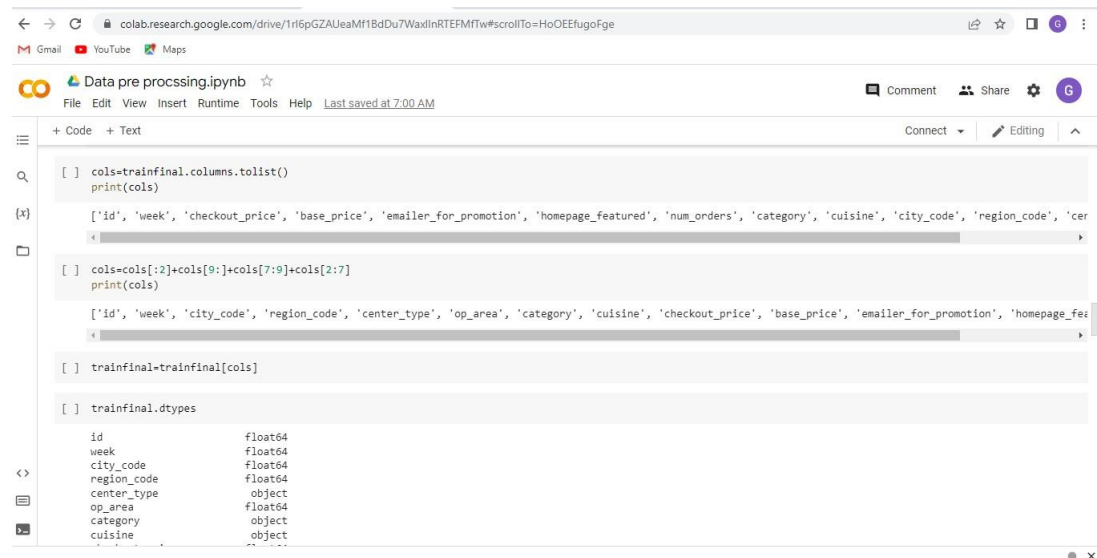
```
[ ] meal_info = pd.read_csv("/content/meal_info.csv")
    center_info = pd.read_csv("/content/fulfilment_center_info.csv")

[ ] trainfinal = pd.merge(train, meal_info, on="meal_id", how="outer")
    trainfinal = pd.merge(trainfinal, center_info, on="center_id", how="outer")

[ ] trainfinal.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	category	cuisine	city_code	region_code
0	1379560.0	1.0	55.0	1885.0	136.83	152.29	0.0	0.0	177.0	Beverages	Thai	647.0	56
1	1018704.0	2.0	55.0	1885.0	135.83	152.29	0.0	0.0	323.0	Beverages	Thai	647.0	56
2	1196273.0	3.0	55.0	1885.0	132.92	133.92	0.0	0.0	96.0	Beverages	Thai	647.0	56
3	1116527.0	4.0	55.0	1885.0	135.86	134.86	0.0	0.0	163.0	Beverages	Thai	647.0	56
4	1343872.0	5.0	55.0	1885.0	146.50	147.50	0.0	0.0	215.0	Beverages	Thai	647.0	56

## 6. Dropping Columns:



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[ ] cols=trainfinal.columns.tolist()
    print(cols)

[ ] cols=cols[:2]+cols[9:]+cols[7:9]+cols[2:7]
    print(cols)

[ ] trainfinal=trainfinal[cols]

[ ] trainfinal.dtypes
```

	id	week	city_code	region_code	center_type	op_area	category	cuisine	checkout_price	base_price	emailer_for_promotion	homepage_featured
id	float64	float64	float64	float64	object	float64	object	object	float64	float64	float64	float64

## 7. Label Encoding:

Gmail YouTube Maps

Data pre procssing.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 7:00 AM

+ Code + Text

```
[ ] lb1=LabelEncoder()
trainfinal['center_type']=lb1.fit_transform(trainfinal['center_type'])
lb2=LabelEncoder()
trainfinal['category']=lb1.fit_transform(trainfinal['category'])
lb1=LabelEncoder()
trainfinal['cuisine']=lb1.fit_transform(trainfinal['cuisine'])
```

trainfinal.head()

	id	week	city_code	region_code	center_type	op_area	category	cuisine	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_order
0	1379560.0	1.0	647.0	56.0	2	2.0	0	3	136.83	152.29	0.0	0.0	177
1	1018704.0	2.0	647.0	56.0	2	2.0	0	3	135.83	152.29	0.0	0.0	323
2	1196273.0	3.0	647.0	56.0	2	2.0	0	3	132.92	133.92	0.0	0.0	96
3	1116527.0	4.0	647.0	56.0	2	2.0	0	3	135.86	134.86	0.0	0.0	163
4	1343872.0	5.0	647.0	56.0	2	2.0	0	3	146.50	147.50	0.0	0.0	215

trainfinal.shape

(103624, 13)

## 8.Data Visualization

colab.researchn.google.com/drive/1nbpGZA0EamITBdDu7WaxiINKIEFM

Gmail YouTube Maps

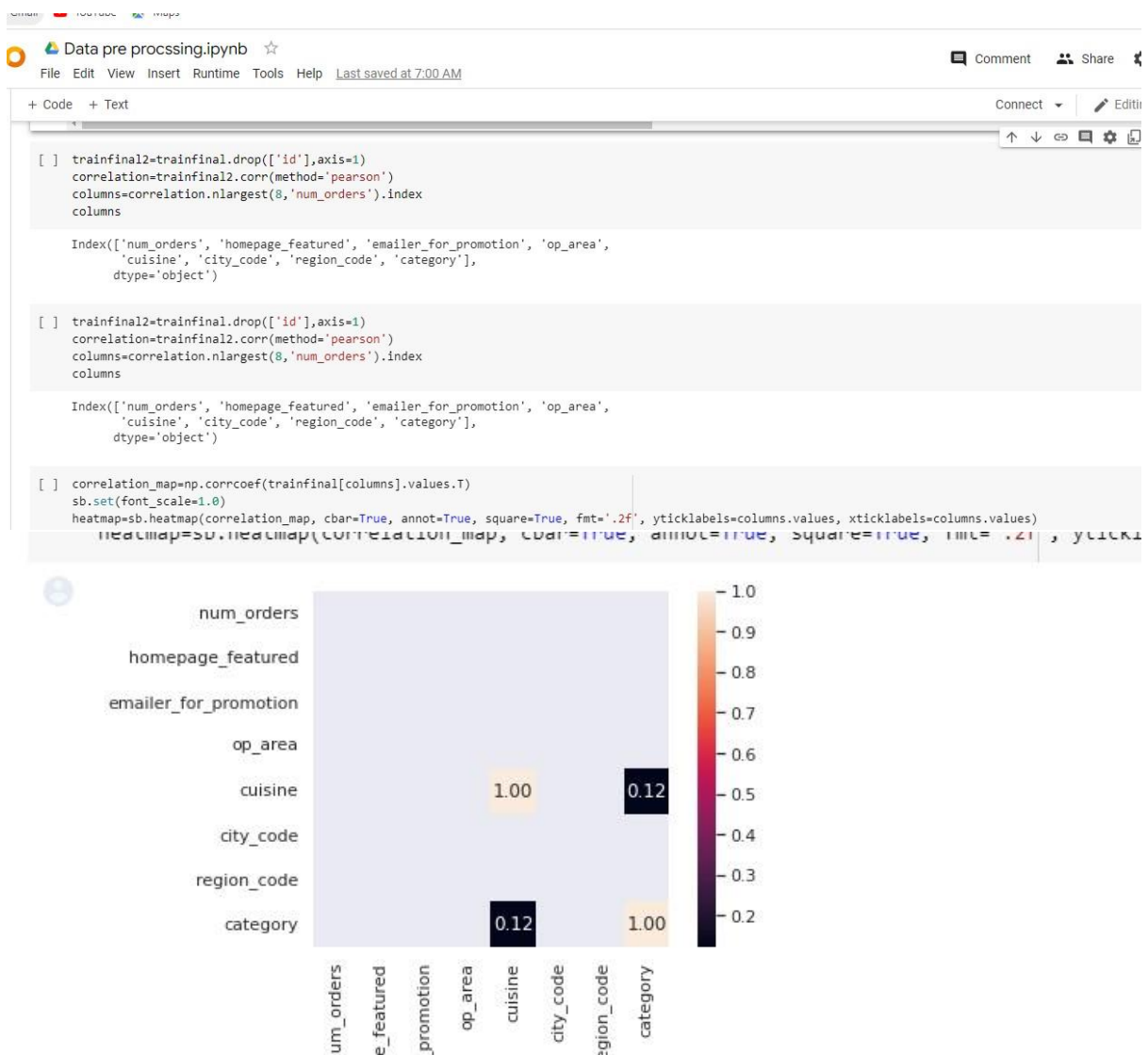
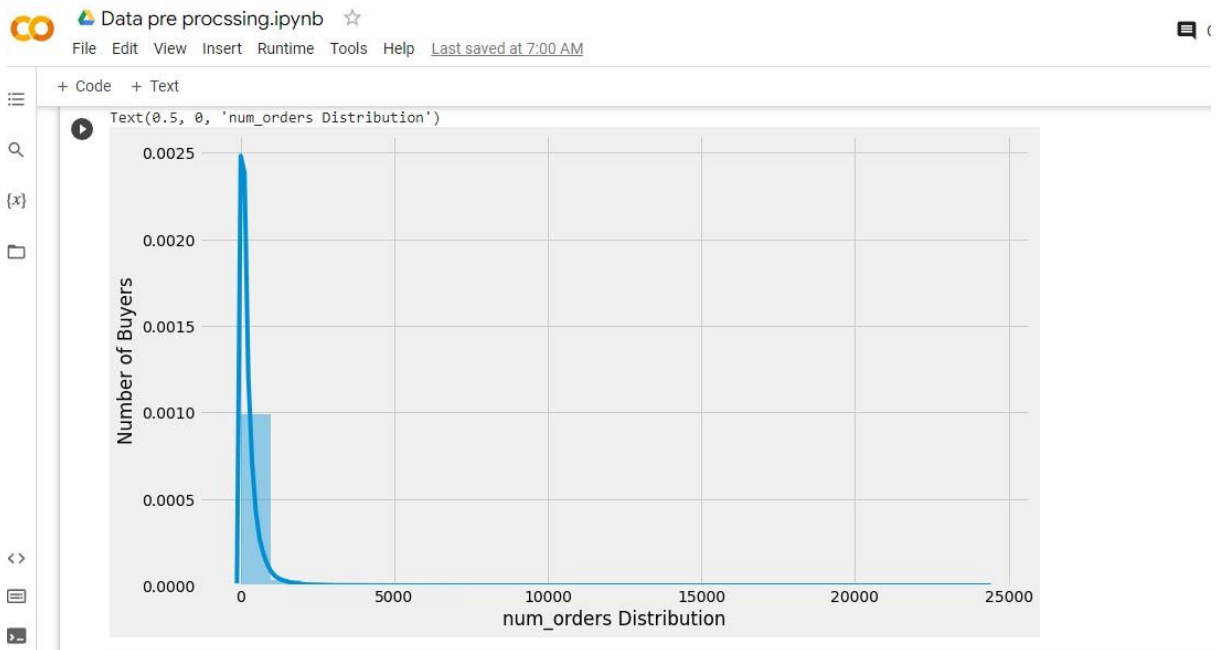
Data pre procssing.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 7:00 AM

+ Code + Text

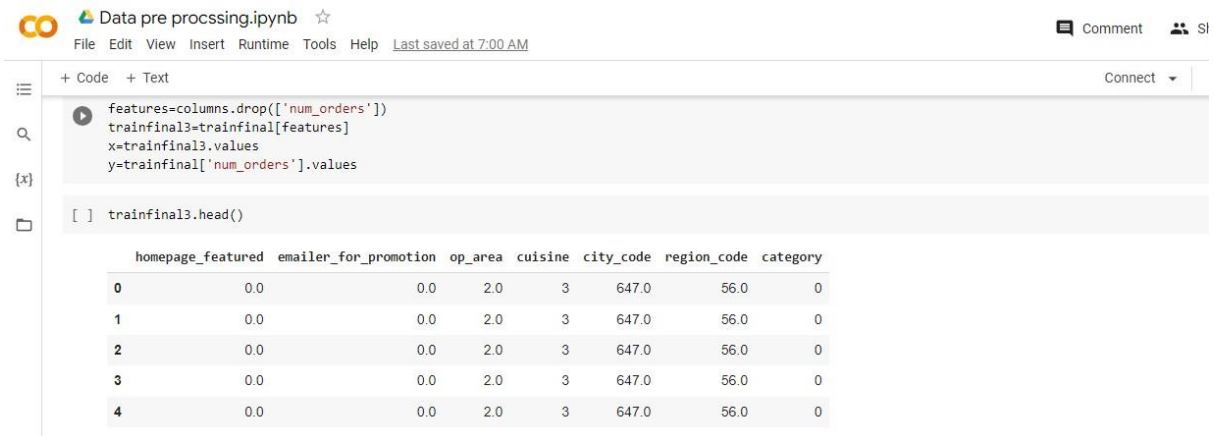
```
[ ] plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,7))
sb.distplot(trainfinal.num_orders,bins=25)
plt.xlabel("num_orders")
plt.ylabel("Number of Buyers")
plt.xlabel("num_orders Distribution")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:
warnings.warn(msg, FutureWarning)
Text(0.5, 0, 'num orders Distribution')
```



## 9.Splitting The Dataset Into Dependent And Independent Variable:

1. The independent variable in the dataset would be considered as 'x' and the 'homepage\_featured', 'emailer\_for\_promotion', 'op\_area', 'cuisine', 'city\_code', 'region\_code', 'category' columns would be considered as independent variable.
2. The dependent variable in the dataset would be considered as 'y' and the 'num\_orders' column is considered as dependent variable.



The screenshot shows a Jupyter Notebook titled "Data pre procssing.ipynb". The code cell contains the following Python code:

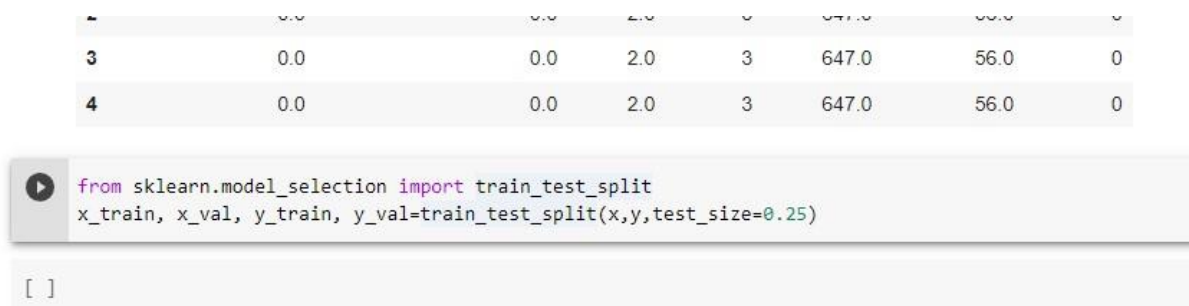
```
features=columns.drop(['num_orders'])
trainfinal3=trainfinal[features]
x=trainfinal3.values
y=trainfinal['num_orders'].values
```

The output cell shows the result of `trainfinal3.head()`, which is a table with 8 columns: `homepage_featured`, `emailer_for_promotion`, `op_area`, `cuisine`, `city_code`, `region_code`, and `category`. The table displays the first 5 rows of data.

	homepage_featured	emailer_for_promotion	op_area	cuisine	city_code	region_code	category
0	0.0	0.0	2.0	3	647.0	56.0	0
1	0.0	0.0	2.0	3	647.0	56.0	0
2	0.0	0.0	2.0	3	647.0	56.0	0
3	0.0	0.0	2.0	3	647.0	56.0	0
4	0.0	0.0	2.0	3	647.0	56.0	0

## Split The Dataset Into Train Set And Test Set:

When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.



The screenshot shows a Jupyter Notebook with a code cell containing the following Python code:

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val=train_test_split(x,y,test_size=0.25)
```

The output cell shows the result of the code execution, which is an empty list: `[ ]`.