## Data Engineer INTERN at HACKVEDA LIMITED

**AUTHOR : BANDANA PRAKASH**
**TASK 1 :** Insights-Driven_sales
**PURPOSE :** to analyze sales transactions and derive actionable insights that can enhance sales strategies.
Specifically, the dataset contains information about customers, their demographics, purchase behaviors, and transaction details.

### Here are the key objectives:

**Customer Analysis:** Understand customer demographics, including age, gender, marital status, and location.
**Sales Performance:** Analyze sales data to identify trends in orders and revenue generation.
**Insight Generation:** Generate insights that can inform marketing strategies and improve customer targeting.
**Data Visualization:** Use visual tools to present findings clearly, making it easier to interpret data trends.
Overall, the goal is to leverage this data for informed decision-making that drives sales growth and enhances customer engagement.

### Steps Involved:

**Import Libraries:** Load essential Python libraries like numpy, pandas, matplotlib, and seaborn for data manipulation and visualization.
**Load Dataset:** Import the sales dataset (Insights-Driven_sales-main.csv) into a Pandas DataFrame.
**Check Dataset Dimensions:** Verify the shape of the dataset to understand its size (rows and columns).
**Explore Data:** Inspect the dataset structure, including column names and sample records, to understand its contents.
**Statistical Summary:** Analyze statistical metrics (count, mean, min, max, etc.) for numerical columns like Age, Orders, and Amount.
**Data Cleaning:** Handle missing values or anomalies in the dataset (if required).
**Data Visualization:** Use tools like Matplotlib and Seaborn to create graphs and charts for better insights into sales trends and customer behavior.
**Generate Insights:** Derive actionable insights such as top-performing products, customer demographics, and purchasing patterns.

```
# import python libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt # visualizing data
%matplotlib inline
import seaborn as sns
```

```
# Raw URL of the CSV file
url = 'https://raw.githubusercontent.com/bandanaprakash/finalYearProject/main/MarketMinder%7C%20Real-Time%20Market%20Analysis%20and%20Fraud%20Defense/Ta

# Read the CSV file
df = pd.read_csv(url, encoding='unicode_escape')
```

```
df.shape
```

```
(11251, 15)
```

```
df.head()
```

| | User_ID | Cust_name | Product_ID | Gender | Age Group | Age | Marital_Status | State | Zone | Occupation | Product_Category | Orders | Amount | Status | unnam |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1002903 | Sanskriti | P00125942 | F | 26-35 | 28 | 0 | Maharashtra | Western | Healthcare | Auto | 1 | 23952.0 | NaN | |
| 1 | 1000732 | Kartik | P00110942 | F | 26-35 | 35 | 1 | Andhra Pradesh | Southern | Govt | Auto | 3 | 23934.0 | NaN | |
| 2 | 1001990 | Bindu | P00118542 | F | 26-35 | 35 | 1 | Uttar Pradesh | Central | Automobile | Auto | 3 | 23924.0 | NaN | |
| 3 | 1001425 | Sudevi | P00237842 | M | 0-17 | 16 | 0 | Karnataka | Southern | Construction | Auto | 2 | 23912.0 | NaN | |
| 4 | 1000588 | Joni | P00057942 | M | 26-35 | 28 | 1 | Gujarat | Western | Food Processing | Auto | 2 | 23877.0 | NaN | |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11251 entries, 0 to 11250
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   User_ID           11251 non-null  int64
 1   Cust_name         11251 non-null  object
 2   Product_ID        11251 non-null  object
 3   Gender            11251 non-null  object
 4   Age Group         11251 non-null  object
 5   Age               11251 non-null  int64
 6   Marital_Status    11251 non-null  int64
 7   State             11251 non-null  object
 8   Zone              11251 non-null  object
 9   Occupation        11251 non-null  object
 10  Product_Category  11251 non-null  object
 11  Orders            11251 non-null  int64
 12  Amount            11239 non-null  float64
 13  Status            0 non-null      float64
 14  unnamed1          0 non-null      float64
dtypes: float64(3), int64(4), object(8)
memory usage: 1.3+ MB
```

```
#drop unrelated/blank columns
df.drop(['Status', 'unnamed1'], axis=1, inplace=True)
```

```
#check for null values
pd.isnull(df).sum()
```

| | 0 |
|---|---|
| User_ID | 0 |
| Cust_name | 0 |
| Product_ID | 0 |
| Gender | 0 |
| Age Group | 0 |
| Age | 0 |
| Marital_Status | 0 |
| State | 0 |
| Zone | 0 |
| Occupation | 0 |
| Product_Category | 0 |
| Orders | 0 |
| Amount | 12 |

**dtype:** int64

```
# drop null values
df.dropna(inplace=True)
```

```
# change data type
df['Amount'] = df['Amount'].astype('int')
```

```
df['Amount'].dtypes
```

```
dtype('int64')
```

```
df.columns
```

```
Index(['User_ID', 'Cust_name', 'Product_ID', 'Gender', 'Age Group', 'Age',
       'Marital_Status', 'State', 'Zone', 'Occupation', 'Product_Category',
       'Orders', 'Amount'],
      dtype='object')
```

```
#rename column
df.rename(columns= {'Marital_Status':'Shaadi'})
```

| | User_ID | Cust_name | Product_ID | Gender | Age Group | Age | Shaadi | State | Zone | Occupation | Product_Category | Orders | Amount |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1002903 | Sanskriti | P00125942 | F | 26-35 | 28 | 0 | Maharashtra | Western | Healthcare | Auto | 1 | 23952 |
| 1 | 1000732 | Kartik | P00110942 | F | 26-35 | 35 | 1 | Andhra Pradesh | Southern | Govt | Auto | 3 | 23934 |
| 2 | 1001990 | Bindu | P00118542 | F | 26-35 | 35 | 1 | Uttar Pradesh | Central | Automobile | Auto | 3 | 23924 |
| 3 | 1001425 | Sudevi | P00237842 | M | 0-17 | 16 | 0 | Karnataka | Southern | Construction | Auto | 2 | 23912 |
| 4 | 1000588 | Joni | P00057942 | M | 26-35 | 28 | 1 | Gujarat | Western | Food Processing | Auto | 2 | 23877 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 11246 | 1000695 | Manning | P00296942 | M | 18-25 | 19 | 1 | Maharashtra | Western | Chemical | Office | 4 | 370 |
| 11247 | 1004089 | Reichenbach | P00171342 | M | 26-35 | 33 | 0 | Haryana | Northern | Healthcare | Veterinary | 3 | 367 |
| 11248 | 1001209 | Oshin | P00201342 | F | 36-45 | 40 | 0 | Madhya Pradesh | Central | Textile | Office | 4 | 213 |
| 11249 | 1004023 | Noonan | P00059442 | M | 36-45 | 37 | 0 | Karnataka | Southern | Agriculture | Office | 3 | 206 |
| 11250 | 1002744 | Brumley | P00281742 | F | 18-25 | 19 | 0 | Maharashtra | Western | Healthcare | Office | 3 | 188 |

11239 rows × 13 columns

```
# describe() method returns description of the data in the DataFrame (i.e. count, mean, std, etc)
df.describe()
```

| | User_ID | Age | Marital_Status | Orders | Amount |
|---|---|---|---|---|---|
| count | 1.123900e+04 | 11239.000000 | 11239.000000 | 11239.000000 | 11239.000000 |
| mean | 1.003004e+06 | 35.410357 | 0.420055 | 2.489634 | 9453.610553 |
| std | 1.716039e+03 | 12.753866 | 0.493589 | 1.114967 | 5222.355168 |
| min | 1.000001e+06 | 12.000000 | 0.000000 | 1.000000 | 188.000000 |
| 25% | 1.001492e+06 | 27.000000 | 0.000000 | 2.000000 | 5443.000000 |
| 50% | 1.003064e+06 | 33.000000 | 0.000000 | 2.000000 | 8109.000000 |
| 75% | 1.004426e+06 | 43.000000 | 1.000000 | 3.000000 | 12675.000000 |
| max | 1.006040e+06 | 92.000000 | 1.000000 | 4.000000 | 23952.000000 |

```
# use describe() for specific columns
```

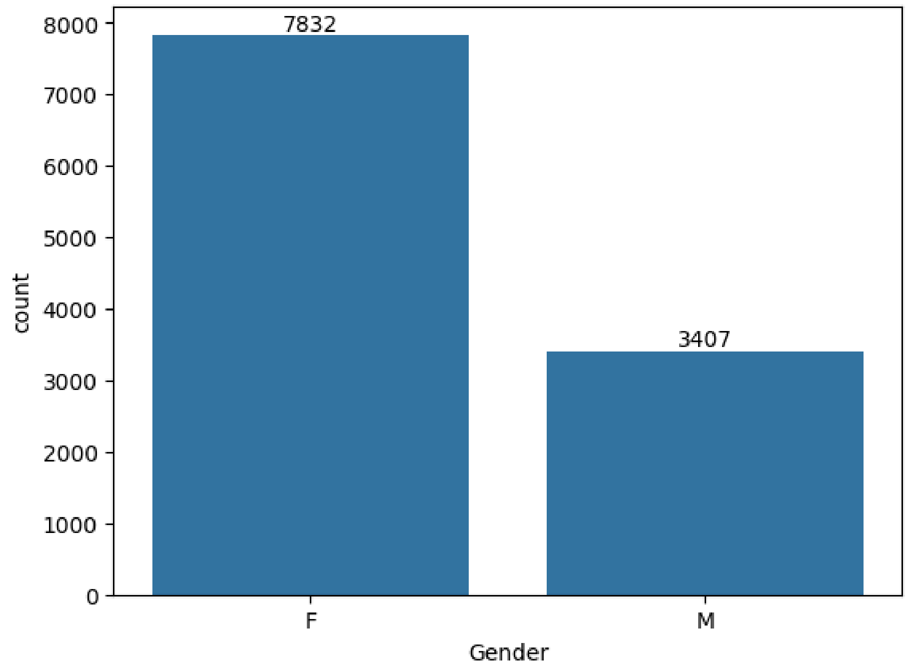
```
df[['Age', 'Orders', 'Amount']].describe()
```

| | Age | Orders | Amount |
|---|---|---|---|
| count | 11239.000000 | 11239.000000 | 11239.000000 |
| mean | 35.410357 | 2.489634 | 9453.610553 |
| std | 12.753866 | 1.114967 | 5222.355168 |
| min | 12.000000 | 1.000000 | 188.000000 |
| 25% | 27.000000 | 2.000000 | 5443.000000 |
| 50% | 33.000000 | 2.000000 | 8109.000000 |
| 75% | 43.000000 | 3.000000 | 12675.000000 |
| max | 92.000000 | 4.000000 | 23952.000000 |

# Exploratory Data Analysis

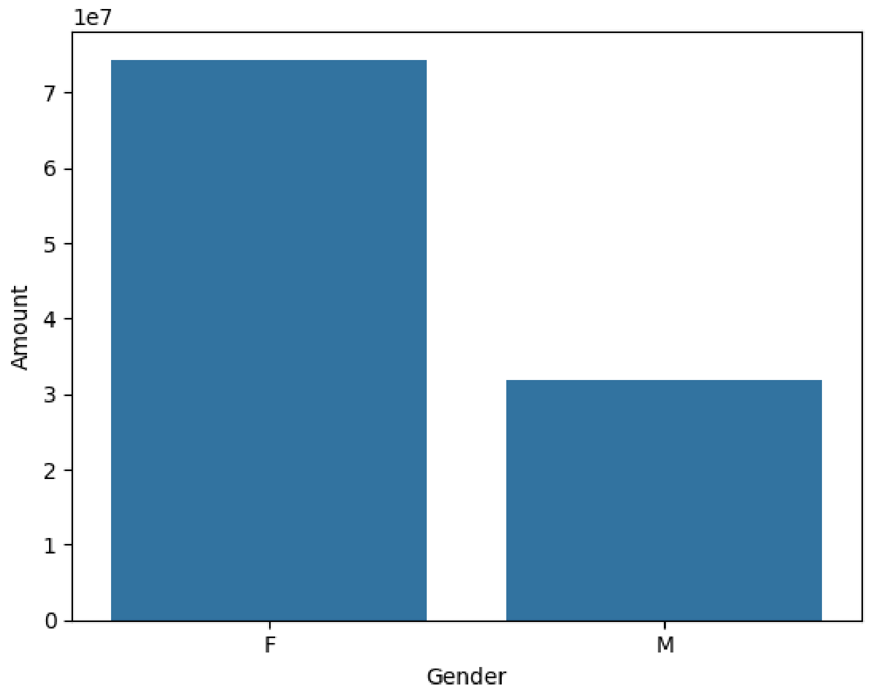
## Gender

```
# plotting a bar chart for Gender and it's count

ax = sns.countplot(x = 'Gender',data = df)

for bars in ax.containers:
    ax.bar_label(bars)
```




```
# plotting a bar chart for gender vs total amount

sales_gen = df.groupby(['Gender'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.barplot(x = 'Gender',y= 'Amount' ,data = sales_gen)
```

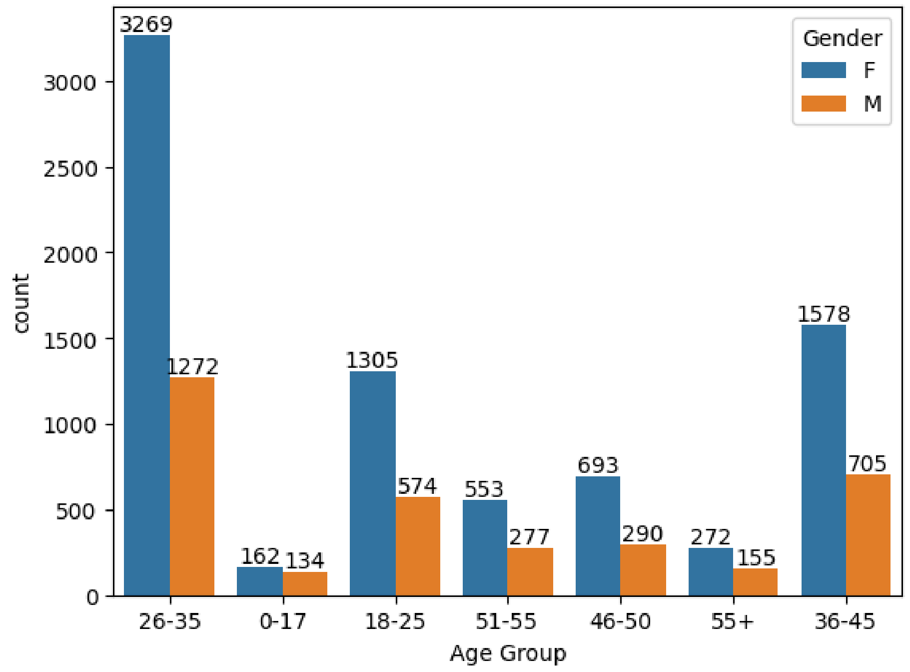
<Axes: xlabel='Gender', ylabel='Amount'>




*From above graphs we can see that most of the buyers are females and even the purchasing power of females are greater than men*

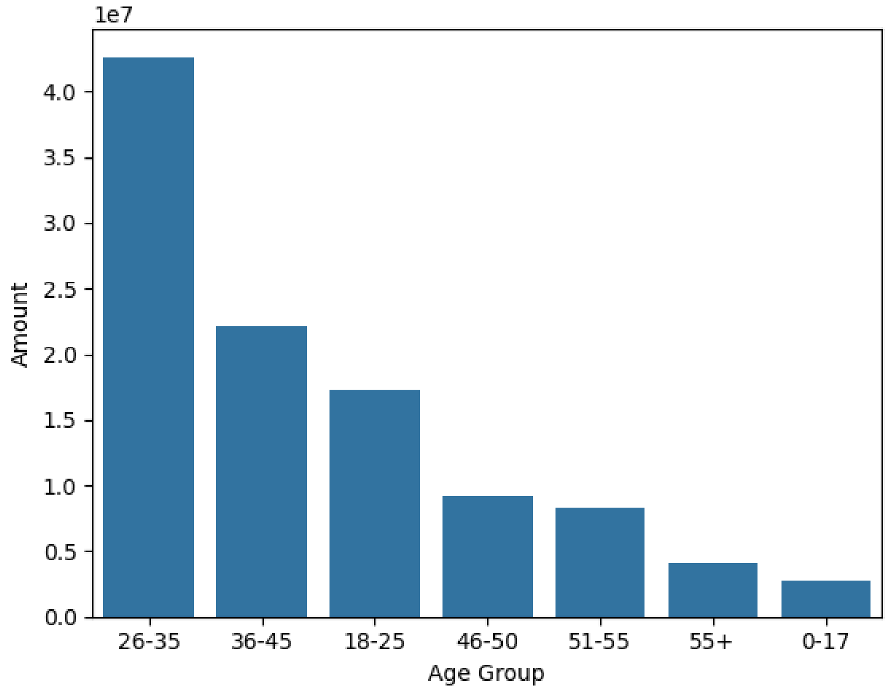
## Age

```
ax = sns.countplot(data = df, x = 'Age Group', hue = 'Gender')

for bars in ax.containers:
    ax.bar_label(bars)
```




```
# Total Amount vs Age Group
sales_age = df.groupby(['Age Group'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.barplot(x = 'Age Group',y= 'Amount' ,data = sales_age)
```

```
<Axes: xlabel='Age Group', ylabel='Amount'>
```

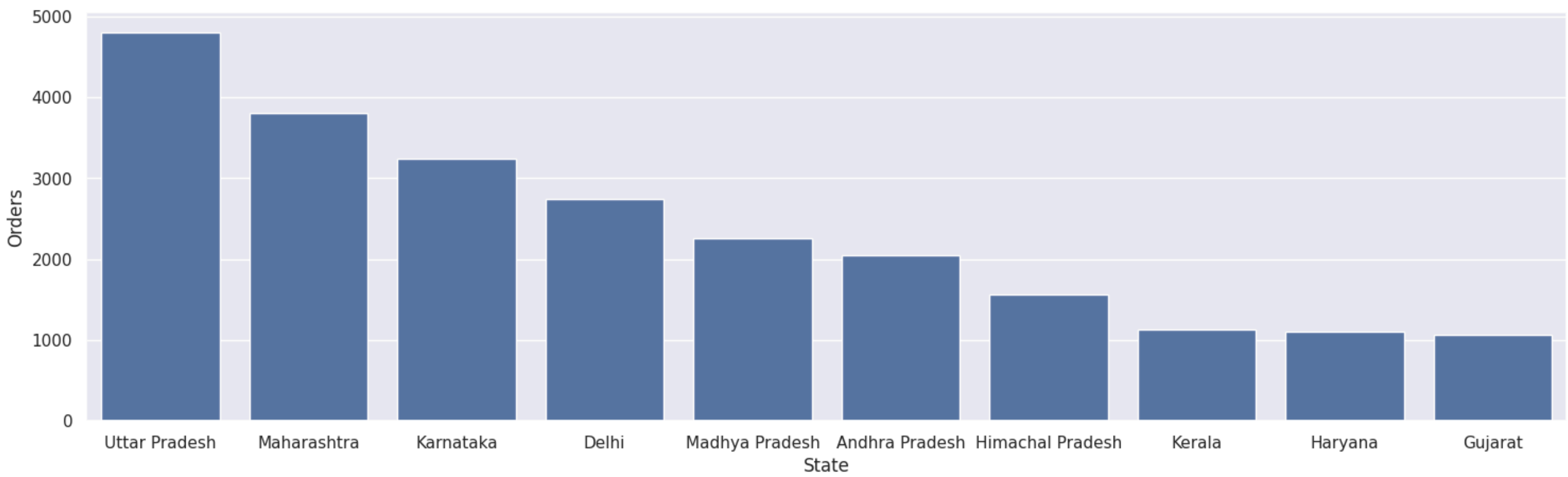



*From above graphs we can see that most of the buyers are of age group between 26-35 yrs female*

## State

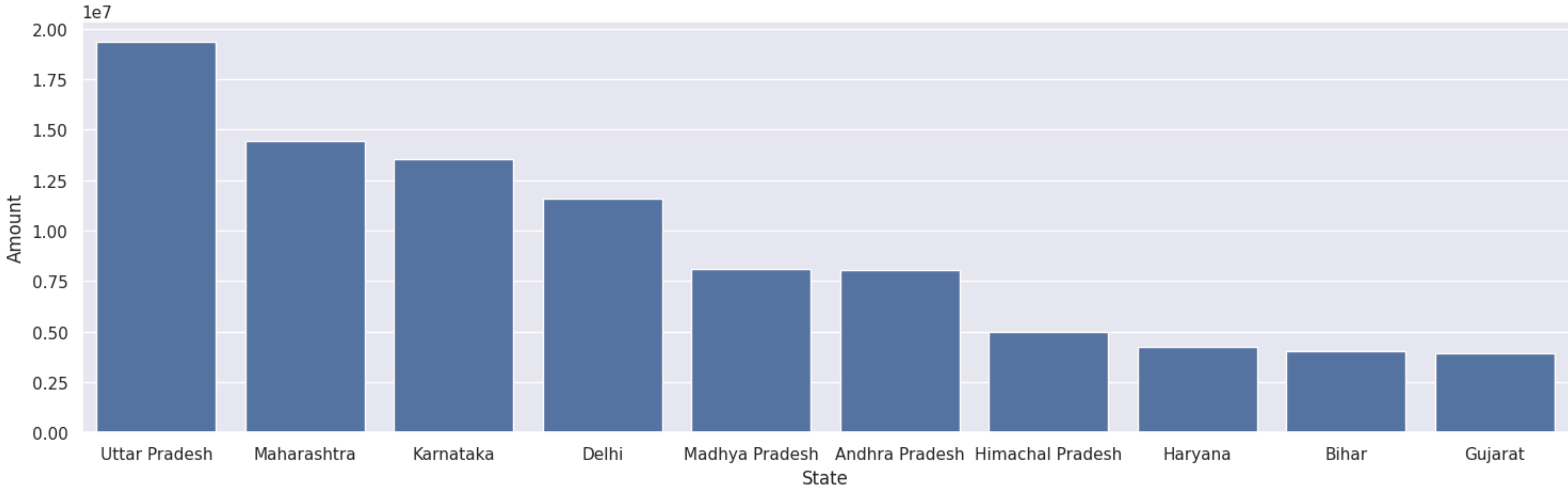
```
# total number of orders from top 10 states

sales_state = df.groupby(['State'], as_index=False)['Orders'].sum().sort_values(by='Orders', ascending=False).head(10)

sns.set(rc={'figure.figsize':(18,5)})
sns.barplot(data = sales_state, x = 'State',y= 'Orders')
```

```
<Axes: xlabel='State', ylabel='Orders'>
```




```
# total amount/sales from top 10 states

sales_state = df.groupby(['State'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False).head(10)

sns.set(rc={'figure.figsize':(18,5)})
sns.barplot(data = sales_state, x = 'State',y= 'Amount')
```

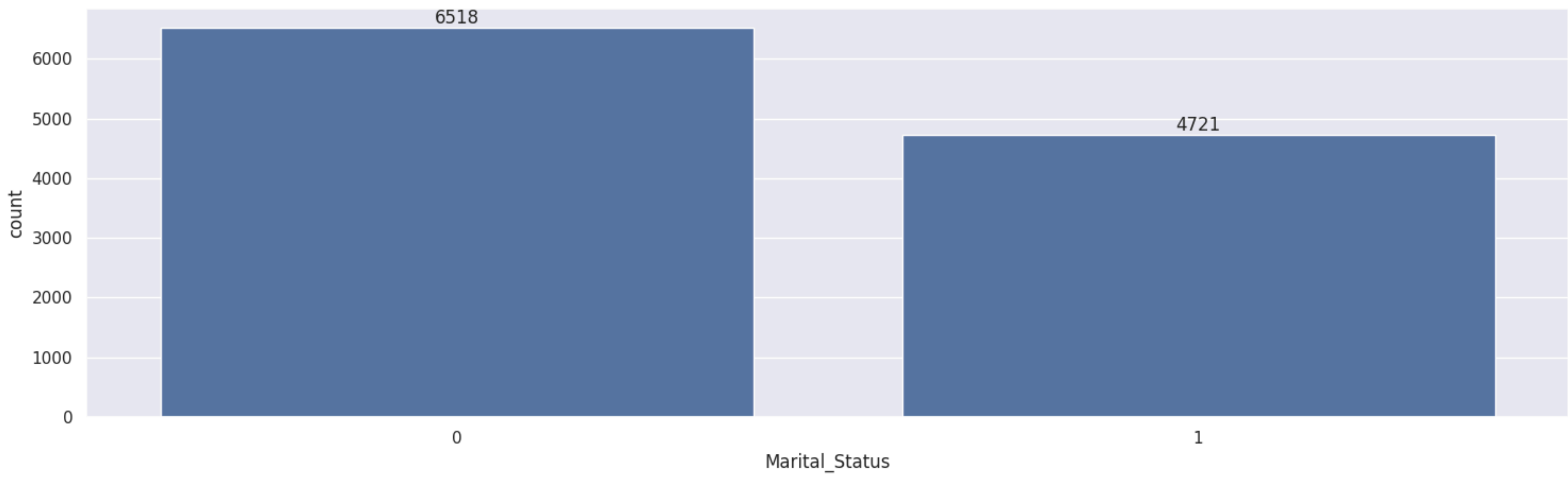
```
<Axes: xlabel='State', ylabel='Amount'>
```




*From above graphs we can see that most of the orders & total sales/amount are from Uttar Pradesh, Maharashtra and Karnataka respectively*

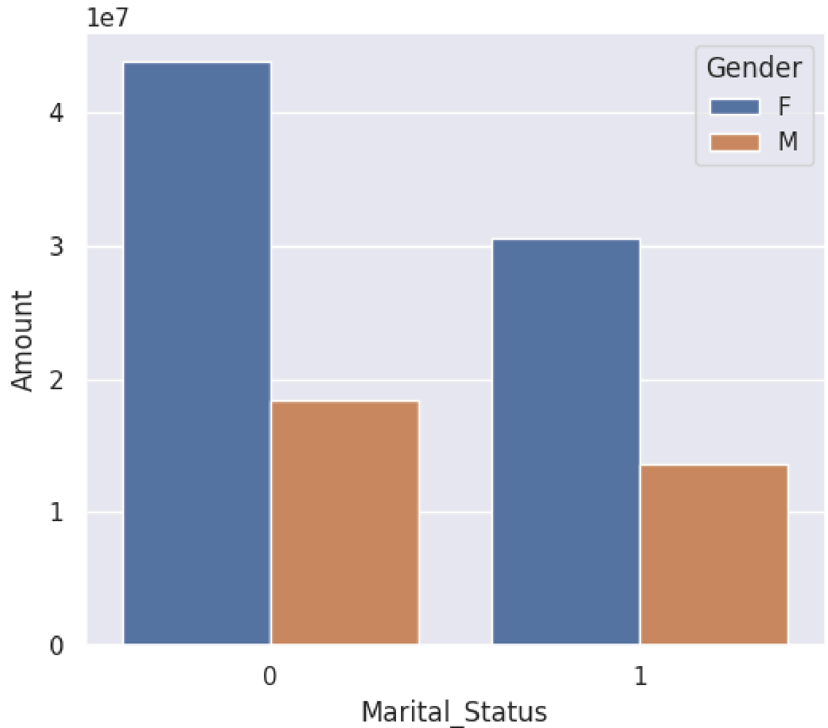
## Marital Status

```
ax = sns.countplot(data = df, x = 'Marital_Status')

sns.set(rc={'figure.figsize':(7,3)})
for bars in ax.containers:
    ax.bar_label(bars)
```




```
sales_state = df.groupby(['Marital_Status', 'Gender'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.set(rc={'figure.figsize':(6,5)})
sns.barplot(data = sales_state, x = 'Marital_Status',y= 'Amount', hue='Gender')
```

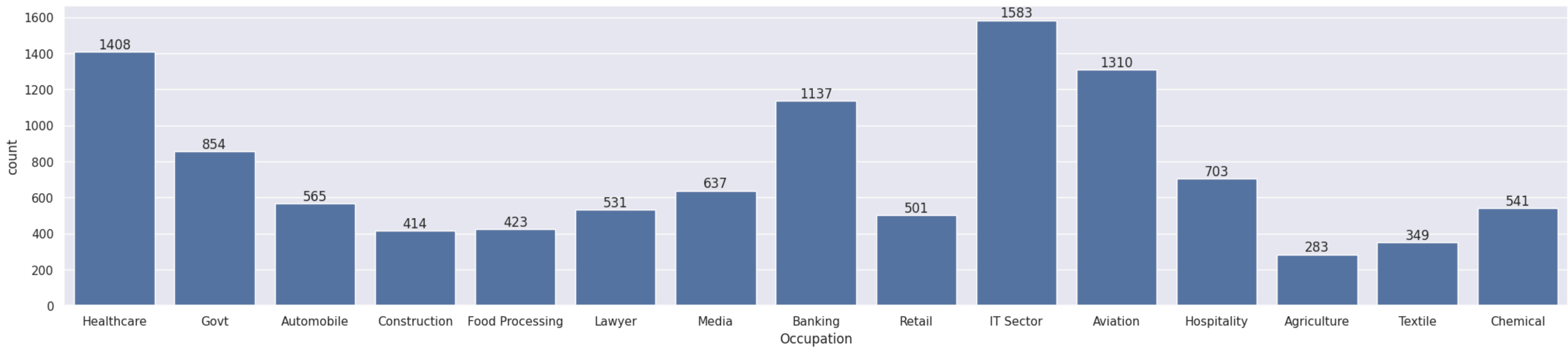
```
<Axes: xlabel='Marital_Status', ylabel='Amount'>
```




*From above graphs we can see that most of the buyers are married (women) and they have high purchasing power*

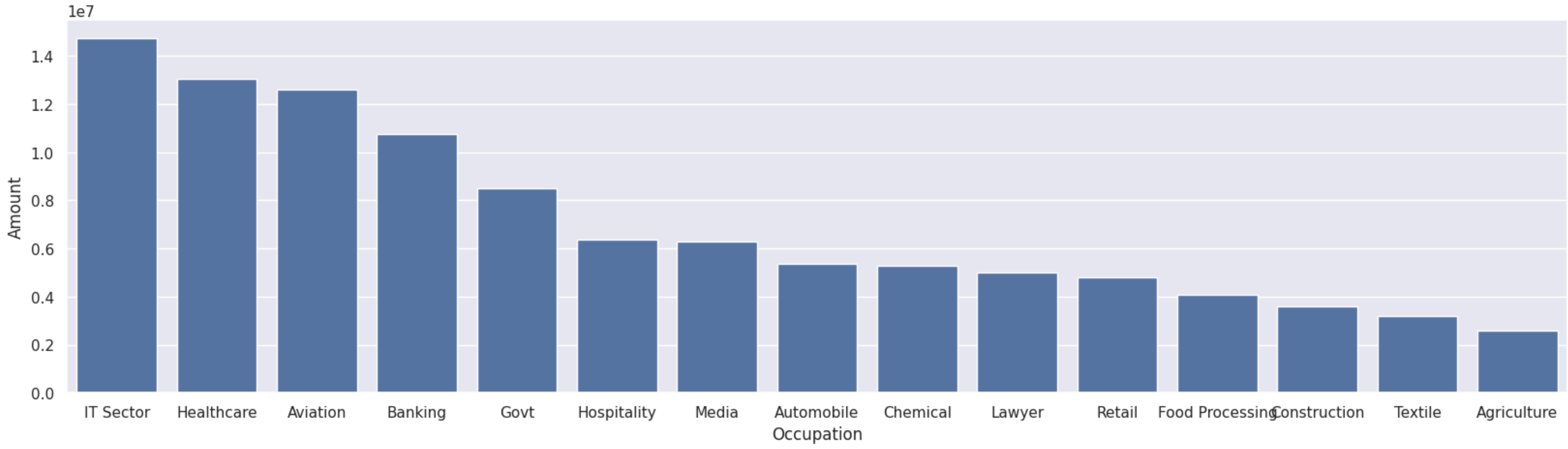
## Occupation

```
sns.set(rc={'figure.figsize':(25,5)})
ax = sns.countplot(data = df, x = 'Occupation')

for bars in ax.containers:
    ax.bar_label(bars)
```




```
sales_state = df.groupby(['Occupation'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Occupation',y= 'Amount')
```

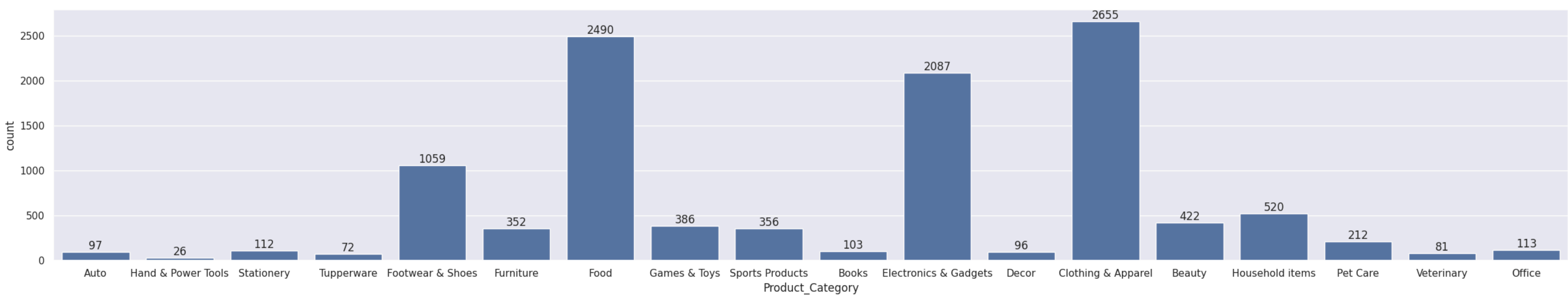
<Axes: xlabel='Occupation', ylabel='Amount'>




*From above graphs we can see that most of the buyers are working in IT, Healthcare and Aviation sector*

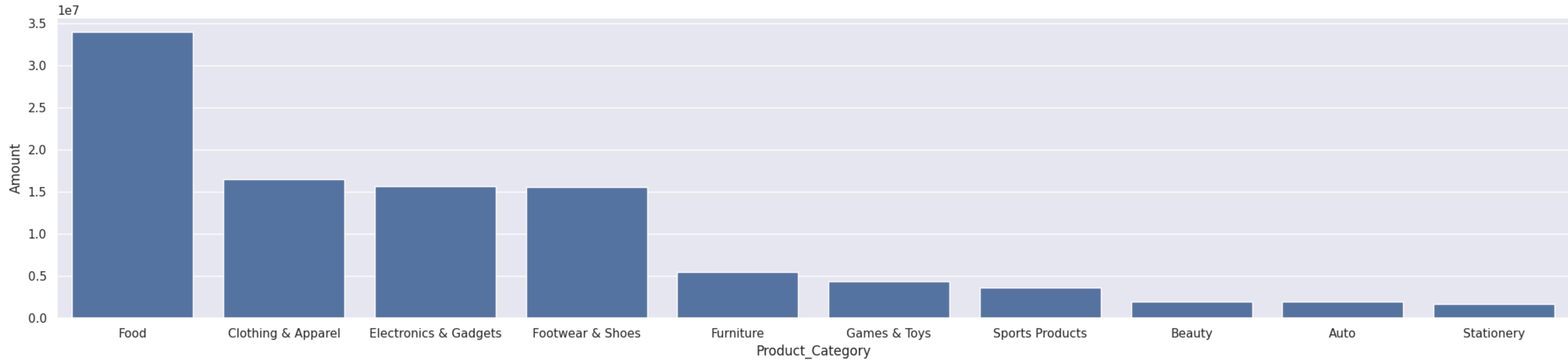
## Product Category

```
sns.set(rc={'figure.figsize':(30,5)})
ax = sns.countplot(data = df, x = 'Product_Category')

for bars in ax.containers:
    ax.bar_label(bars)
```




```
sales_state = df.groupby(['Product_Category'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False).head(10)

sns.set(rc={'figure.figsize':(25,5)})
sns.barplot(data = sales_state, x = 'Product_Category',y= 'Amount')
```

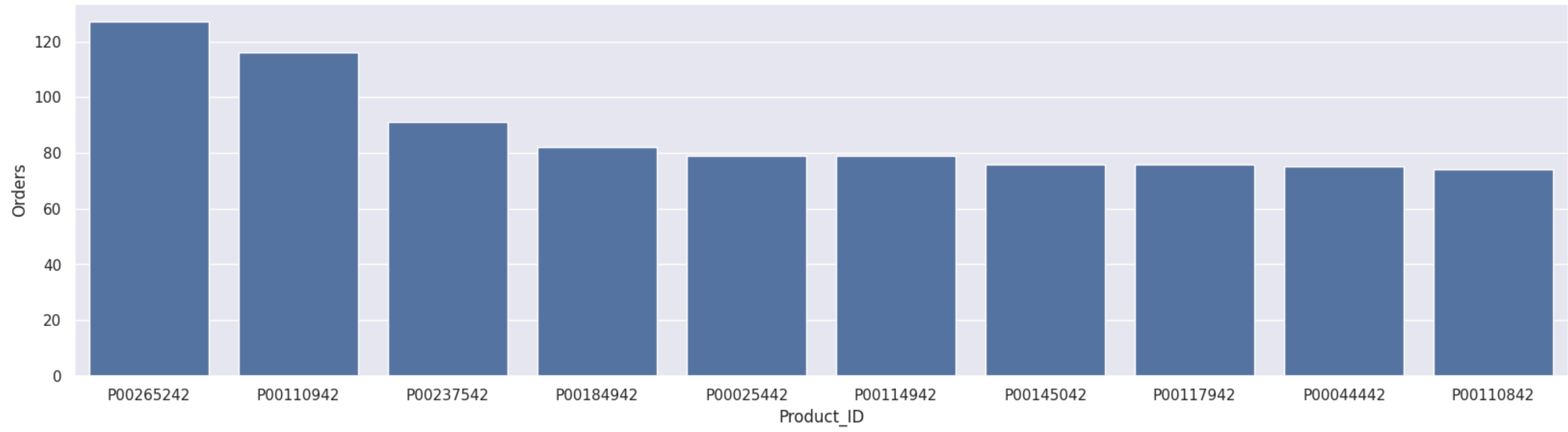
<Axes: xlabel='Product_Category', ylabel='Amount'>




*From above graphs we can see that most of the sold products are from Food, Clothing and Electronics category*

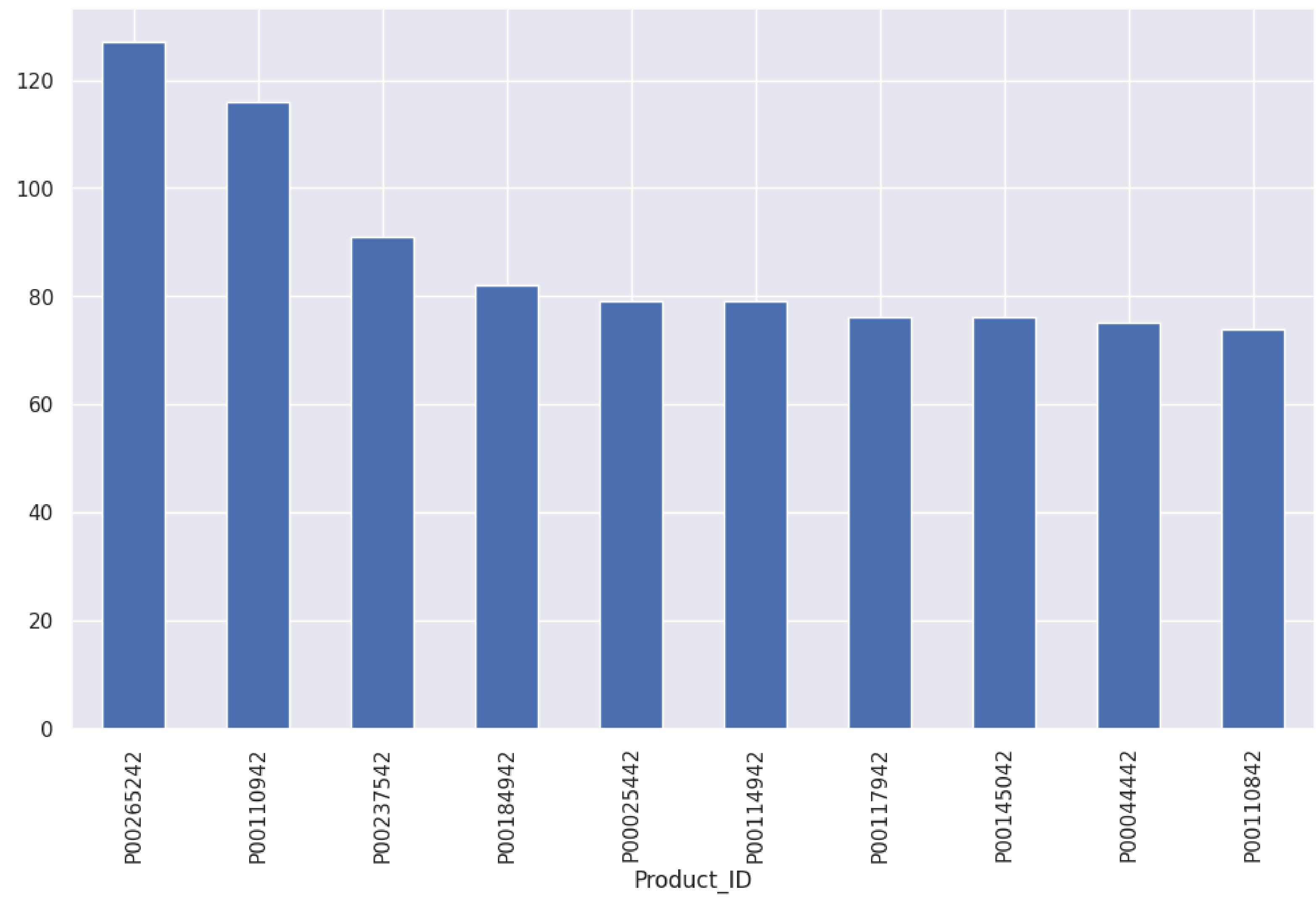
```
sales_state = df.groupby(['Product_ID'], as_index=False)['Orders'].sum().sort_values(by='Orders', ascending=False).head(10)

sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Product_ID',y= 'Orders')
```

<Axes: xlabel='Product_ID', ylabel='Orders'>




```
# top 10 most sold products (same thing as above)

fig1, ax1 = plt.subplots(figsize=(12,7))
df.groupby('Product_ID')['Orders'].sum().nlargest(10).sort_values(ascending=False).plot(kind='bar')
```

<Axes: xlabel='Product_ID'>




## Conclusion:

###

*Married women age group 26-35 yrs from UP, Maharastra and Karnataka working in IT, Healthcare and Aviation are more likely to buy products from Food, Clothing and Electronics category*

Thank you!

# Data Engineer INTERN at HACKVEDA LIMITED

**AUTHOR : BANDANA PRAKASH**
**TASK 2 :** STOCK PREDICTION
**PURPOSE :** TO PREDICT THE STOCK PRICE OF A COMPANY USING LSTM.

## ABOUT DATASET

**Google Stock Prediction** This dataset contains historical data of Google's stock prices and related attributes. It consists of 14 columns and a smaller subset of 1257 rows. Each column represents a specific attribute, and each row contains the corresponding values for that attribute.

## The columns in the dataset are as follows:

**Symbol:** The name of the company, which is GOOG in this case.
**Date:** The year and date of the stock data.
**Close:** The closing price of Google's stock on a particular day.
**High:** The highest value reached by Google's stock on the given day.
**Low:** The lowest value reached by Google's stock on the given day.
**Open:** The opening value of Google's stock on the given day.
**Volume:** The trading volume of Google's stock on the given day, i.e., the number of shares traded.
**adjClose:** The adjusted closing price of Google's stock, considering factors such as dividends and stock splits.
**adjHigh:** The adjusted highest value reached by Google's stock on the given day.
**adjLow:** The adjusted lowest value reached by Google's stock on the given day.
**adjOpen:** The adjusted opening value of Google's stock on the given day.
**adjVolume:** The adjusted trading volume of Google's stock on the given day, accounting for factors such as stock splits.
**divCash:** The amount of cash dividend paid out to shareholders on the given day.
**splitFactor:** The split factor, if any, applied to Google's stock on the given day. A split factor of 1 indicates no split.

## STEPS INVOLVED :

1. IMPORTING LIBRARIES AND DATA TO BE USED
2. GATHERING INSIGHTS
3. DATA PRE-PROCESSING
4. CREATING LSTM MODEL
5. VISUALIZING ACTUAL VS PREDICTED DATA
6. PREDICTING UPCOMING 15 DAYS

STEP 1 : IMPORTING LIBRARIES AND DATA TO BE USED

```
#importing libraries to be used
import numpy as np # for linear algebra
import pandas as pd # data preprocessing
import matplotlib.pyplot as plt # data visualization library
import seaborn as sns # data visualization library
%matplotlib inline
import warnings
warnings.filterwarnings('ignore') # ignore warnings

from sklearn.preprocessing import MinMaxScaler # for normalization
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Bidirectional
```

```
# Raw URL of the CSV file
url = 'https://raw.githubusercontent.com/bandanaprakash/finalYearProject/main/MarketMinder%7C%20Real-Time%20Market%20Analysis%20and%20Fraud%20Defense/Task2_STOCK%20PREDICTION/(

# Read the CSV file
df = pd.read_csv(url)

# Display the first 10 rows of the dataset
print(df.head(10))
```

```
  symbol                       date   close    high       low    open  \
0   GOOG  2016-06-14 00:00:00+00:00  718.27  722.47  713.1200  716.48
1   GOOG  2016-06-15 00:00:00+00:00  718.92  722.98  717.3100  719.00
2   GOOG  2016-06-16 00:00:00+00:00  710.36  716.65  703.2600  714.91
3   GOOG  2016-06-17 00:00:00+00:00  691.72  708.82  688.4515  708.65
4   GOOG  2016-06-20 00:00:00+00:00  693.71  702.48  693.4100  698.77
5   GOOG  2016-06-21 00:00:00+00:00  695.94  702.77  692.0100  698.40
6   GOOG  2016-06-22 00:00:00+00:00  697.46  700.86  693.0819  699.06
7   GOOG  2016-06-23 00:00:00+00:00  701.87  701.95  687.0000  697.45
8   GOOG  2016-06-24 00:00:00+00:00  675.22  689.40  673.4500  675.17
9   GOOG  2016-06-27 00:00:00+00:00  668.26  672.30  663.2840  671.00

    volume  adjClose  adjHigh    adjLow  adjOpen  adjVolume  divCash  \
0  1306065    718.27   722.47  713.1200   716.48    1306065      0.0
1  1214517    718.92   722.98  717.3100   719.00    1214517      0.0
2  1982471    710.36   716.65  703.2600   714.91    1982471      0.0
3  3402357    691.72   708.82  688.4515   708.65    3402357      0.0
4  2082538    693.71   702.48  693.4100   698.77    2082538      0.0
5  1465634    695.94   702.77  692.0100   698.40    1465634      0.0
6  1184318    697.46   700.86  693.0819   699.06    1184318      0.0
7  2171415    701.87   701.95  687.0000   697.45    2171415      0.0
8  4449022    675.22   689.40  673.4500   675.17    4449022      0.0
9  2641085    668.26   672.30  663.2840   671.00    2641085      0.0

   splitFactor
0          1.0
1          1.0
2          1.0
3          1.0
4          1.0
5          1.0
6          1.0
7          1.0
8          1.0
9          1.0
```

STEP 2 : GATHERING INSIGHTS

```
# shape of data
print("Shape of data:",df.shape)
```

```
Shape of data: (1258, 14)
```

```
# statistical description of data
df.describe()
```

| | close | high | low | open | volume | adjClose | adjHigh | adjLow | adjOpen | adjVolume | divCash | splitFactor |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 1258.000000 | 1258.000000 | 1258.000000 | 1258.000000 | 1.258000e+03 | 1258.000000 | 1258.000000 | 1258.000000 | 1258.000000 | 1.258000e+03 | 1258.0 | 1258.0 |
| **mean** | 1216.317067 | 1227.430934 | 1204.176430 | 1215.260779 | 1.601590e+06 | 1216.317067 | 1227.430936 | 1204.176436 | 1215.260779 | 1.601590e+06 | 0.0 | 1.0 |
| **std** | 383.333358 | 387.570872 | 378.777094 | 382.446995 | 6.960172e+05 | 383.333358 | 387.570873 | 378.777099 | 382.446995 | 6.960172e+05 | 0.0 | 0.0 |
| **min** | 668.260000 | 672.300000 | 663.284000 | 671.000000 | 3.467530e+05 | 668.260000 | 672.300000 | 663.284000 | 671.000000 | 3.467530e+05 | 0.0 | 1.0 |
| **25%** | 960.802500 | 968.757500 | 952.182500 | 959.005000 | 1.173522e+06 | 960.802500 | 968.757500 | 952.182500 | 959.005000 | 1.173522e+06 | 0.0 | 1.0 |
| **50%** | 1132.460000 | 1143.935000 | 1117.915000 | 1131.150000 | 1.412588e+06 | 1132.460000 | 1143.935000 | 1117.915000 | 1131.150000 | 1.412588e+06 | 0.0 | 1.0 |
| **75%** | 1360.595000 | 1374.345000 | 1348.557500 | 1361.075000 | 1.812156e+06 | 1360.595000 | 1374.345000 | 1348.557500 | 1361.075000 | 1.812156e+06 | 0.0 | 1.0 |
| **max** | 2521.600000 | 2526.990000 | 2498.290000 | 2524.920000 | 6.207027e+06 | 2521.600000 | 2526.990000 | 2498.290000 | 2524.920000 | 6.207027e+06 | 0.0 | 1.0 |

```
# summary of data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   symbol       1258 non-null   object
 1   date         1258 non-null   object
 2   close        1258 non-null   float64
 3   high         1258 non-null   float64
 4   low          1258 non-null   float64
 5   open         1258 non-null   float64
 6   volume       1258 non-null   int64
 7   adjClose     1258 non-null   float64
 8   adjHigh      1258 non-null   float64
 9   adjLow       1258 non-null   float64
 10  adjOpen      1258 non-null   float64
 11  adjVolume    1258 non-null   int64
 12  divCash      1258 non-null   float64
 13  splitFactor  1258 non-null   float64
dtypes: float64(10), int64(2), object(2)
memory usage: 137.7+ KB
```

```
# checking null values
df.isnull().sum()
```

| | 0 |
|---|---|
| **symbol** | 0 |
| **date** | 0 |
| **close** | 0 |
| **high** | 0 |
| **low** | 0 |
| **open** | 0 |
| **volume** | 0 |
| **adjClose** | 0 |
| **adjHigh** | 0 |
| **adjLow** | 0 |
| **adjOpen** | 0 |
| **adjVolume** | 0 |
| **divCash** | 0 |
| **splitFactor** | 0 |

**dtype:** int64

There are no null values in the dataset

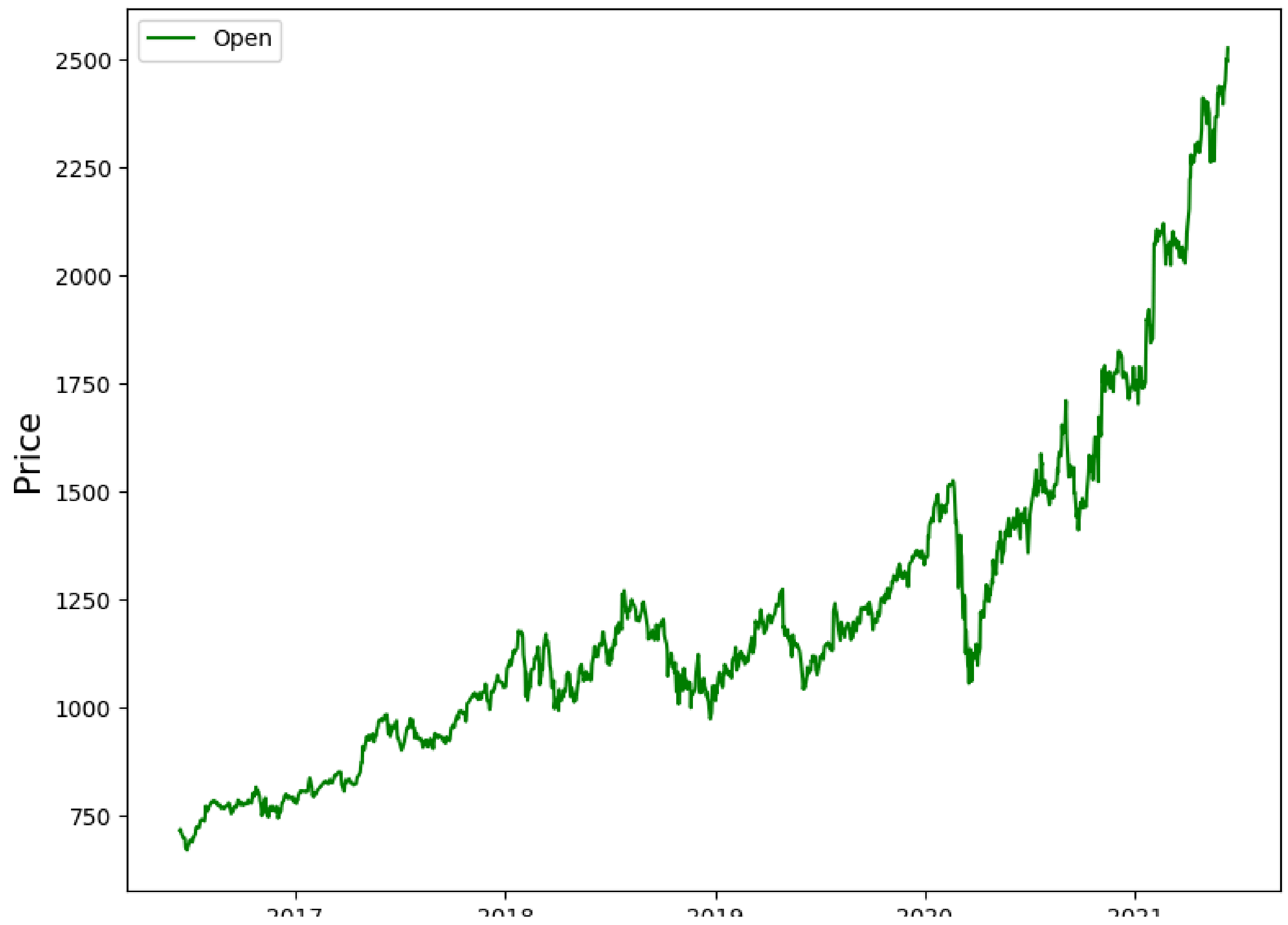
```
df = df[['date','open','close']] # Extracting required columns
df['date'] = pd.to_datetime(df['date'].apply(lambda x: x.split()[0])) # converting object dtype of date column to datetime dtype
df.set_index('date',drop=True,inplace=True) # Setting date column as index
df.head(10)
```

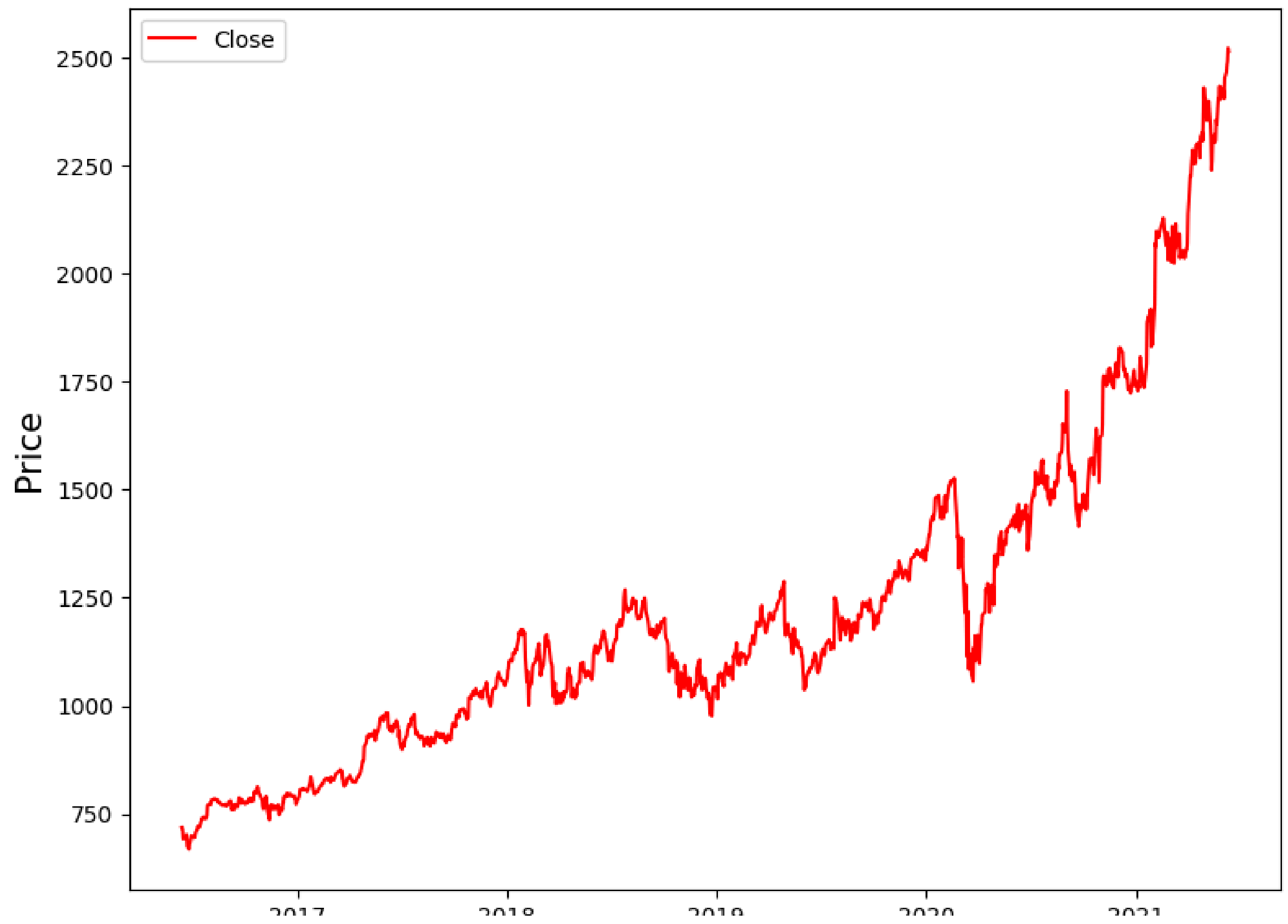
| date | open | close |
|---|---|---|
| **2016-06-14** | 716.48 | 718.27 |
| **2016-06-15** | 719.00 | 718.92 |
| **2016-06-16** | 714.91 | 710.36 |
| **2016-06-17** | 708.65 | 691.72 |
| **2016-06-20** | 698.77 | 693.71 |
| **2016-06-21** | 698.40 | 695.94 |
| **2016-06-22** | 699.06 | 697.46 |
| **2016-06-23** | 697.45 | 701.87 |
| **2016-06-24** | 675.17 | 675.22 |
| **2016-06-27** | 671.00 | 668.26 |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# plotting open and closing price on date index
fig, ax =plt.subplots(1,2,figsize=(20,7))
```

```
ax[0].plot(df['open'],label='Open',color='green')
ax[0].set_xlabel('Date',size=15)
ax[0].set_ylabel('Price',size=15)
ax[0].legend()

ax[1].plot(df['close'],label='Close',color='red')
ax[1].set_xlabel('Date',size=15)
ax[1].set_ylabel('Price',size=15)
ax[1].legend()

fig.show()
```




STEP 3 : DATA PRE-PROCESSING

```
# normalizing all the values of all columns using MinMaxScaler
MMS = MinMaxScaler()
df[df.columns] = MMS.fit_transform(df)
df.head(10)
```

| date | open | close |
|---|---|---|
| 2016-06-14 | 0.024532 | 0.026984 |
| 2016-06-15 | 0.025891 | 0.027334 |
| 2016-06-16 | 0.023685 | 0.022716 |
| 2016-06-17 | 0.020308 | 0.012658 |
| 2016-06-20 | 0.014979 | 0.013732 |
| 2016-06-21 | 0.014779 | 0.014935 |
| 2016-06-22 | 0.015135 | 0.015755 |
| 2016-06-23 | 0.014267 | 0.018135 |
| 2016-06-24 | 0.002249 | 0.003755 |
| 2016-06-27 | 0.000000 | 0.000000 |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# splitting the data into training and test set
training_size = round(len(df) * 0.75) # Selecting 75 % for training and 25 % for testing
training_size
```

```
944
```

```
train_data = df[:training_size]
test_data  = df[training_size:]

train_data.shape, test_data.shape
```

```
((944, 2), (314, 2))
```

```
# Function to create sequence of data for training and testing

def create_sequence(dataset):
  sequences = []
  labels = []

  start_idx = 0

  for stop_idx in range(50,len(dataset)): # Selecting 50 rows at a time
    sequences.append(dataset.iloc[start_idx:stop_idx])
    labels.append(dataset.iloc[stop_idx])
    start_idx += 1
  return (np.array(sequences),np.array(labels))


train_seq, train_label = create_sequence(train_data)
test_seq, test_label = create_sequence(test_data)
train_seq.shape, train_label.shape, test_seq.shape, test_label.shape
```

```
((894, 50, 2), (894, 2), (264, 50, 2), (264, 2))
```

STEP 4 : CREATING LSTM MODEL

```
# imported Sequential from keras.models
model = Sequential()
```

```
# importing Dense, Dropout, LSTM, Bidirectional from keras.layers
model.add(LSTM(units=50, return_sequences=True, input_shape = (train_seq.shape[1], train_seq.shape[2])))

model.add(Dropout(0.1))
model.add(LSTM(units=50))

model.add(Dense(2))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_error'])

model.summary()
```

**Model: "sequential"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm (LSTM) | (None, 50, 50) | 10,600 |
| dropout (Dropout) | (None, 50, 50) | 0 |
| lstm_1 (LSTM) | (None, 50) | 20,200 |
| dense (Dense) | (None, 2) | 102 |

**Total params:** 30,902 (120.71 KB)
**Trainable params:** 30,902 (120.71 KB)
**Non-trainable params:** 0 (0.00 B)

```
# fitting the model by iterating the dataset over 100 times(100 epochs)
model.fit(train_seq, train_label, epochs=100,validation_data=(test_seq, test_label), verbose=1)
```

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - loss: 1.5238e-04 - mean_absolute_error: 0.0089 - val_loss: 0.0030 - val_mean_absolute_error: 0.0448
Epoch 73/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - loss: 1.5071e-04 - mean_absolute_error: 0.0089 - val_loss: 0.0029 - val_mean_absolute_error: 0.0427
Epoch 74/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - loss: 1.5754e-04 - mean_absolute_error: 0.0093 - val_loss: 0.0037 - val_mean_absolute_error: 0.0498
Epoch 75/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - loss: 1.2796e-04 - mean_absolute_error: 0.0084 - val_loss: 0.0030 - val_mean_absolute_error: 0.0449
Epoch 76/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - loss: 1.4075e-04 - mean_absolute_error: 0.0087 - val_loss: 0.0029 - val_mean_absolute_error: 0.0425
Epoch 77/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - loss: 1.5298e-04 - mean_absolute_error: 0.0092 - val_loss: 0.0016 - val_mean_absolute_error: 0.0302
Epoch 78/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - loss: 1.4958e-04 - mean_absolute_error: 0.0088 - val_loss: 0.0017 - val_mean_absolute_error: 0.0318
Epoch 79/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - loss: 1.4603e-04 - mean_absolute_error: 0.0087 - val_loss: 0.0018 - val_mean_absolute_error: 0.0334
Epoch 80/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.4834e-04 - mean_absolute_error: 0.0090 - val_loss: 0.0012 - val_mean_absolute_error: 0.0270
Epoch 81/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - loss: 1.3472e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0050 - val_mean_absolute_error: 0.0594
Epoch 82/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - loss: 1.3095e-04 - mean_absolute_error: 0.0084 - val_loss: 0.0054 - val_mean_absolute_error: 0.0618
Epoch 83/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 52ms/step - loss: 1.4732e-04 - mean_absolute_error: 0.0091 - val_loss: 0.0054 - val_mean_absolute_error: 0.0614
Epoch 84/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 87ms/step - loss: 1.5452e-04 - mean_absolute_error: 0.0088 - val_loss: 0.0036 - val_mean_absolute_error: 0.0494
Epoch 85/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.2542e-04 - mean_absolute_error: 0.0080 - val_loss: 0.0034 - val_mean_absolute_error: 0.0474
Epoch 86/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - loss: 1.3994e-04 - mean_absolute_error: 0.0085 - val_loss: 0.0039 - val_mean_absolute_error: 0.0509
Epoch 87/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.4788e-04 - mean_absolute_error: 0.0086 - val_loss: 0.0014 - val_mean_absolute_error: 0.0284
Epoch 88/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.3139e-04 - mean_absolute_error: 0.0082 - val_loss: 0.0012 - val_mean_absolute_error: 0.0258
Epoch 89/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - loss: 1.4381e-04 - mean_absolute_error: 0.0083 - val_loss: 0.0011 - val_mean_absolute_error: 0.0260
Epoch 90/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - loss: 1.2231e-04 - mean_absolute_error: 0.0079 - val_loss: 0.0027 - val_mean_absolute_error: 0.0405
Epoch 91/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - loss: 1.2541e-04 - mean_absolute_error: 0.0082 - val_loss: 0.0028 - val_mean_absolute_error: 0.0427
Epoch 92/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.1055e-04 - mean_absolute_error: 0.0077 - val_loss: 0.0014 - val_mean_absolute_error: 0.0289
Epoch 93/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - loss: 1.3745e-04 - mean_absolute_error: 0.0085 - val_loss: 0.0019 - val_mean_absolute_error: 0.0348
Epoch 94/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 115ms/step - loss: 1.2206e-04 - mean_absolute_error: 0.0080 - val_loss: 0.0012 - val_mean_absolute_error: 0.0263
Epoch 95/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 4s 82ms/step - loss: 1.4256e-04 - mean_absolute_error: 0.0090 - val_loss: 9.3829e-04 - val_mean_absolute_error: 0.0232
Epoch 96/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 1.1209e-04 - mean_absolute_error: 0.0075 - val_loss: 7.4422e-04 - val_mean_absolute_error: 0.0206
Epoch 97/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - loss: 1.2002e-04 - mean_absolute_error: 0.0078 - val_loss: 0.0021 - val_mean_absolute_error: 0.0376
Epoch 98/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.0345e-04 - mean_absolute_error: 0.0075 - val_loss: 0.0015 - val_mean_absolute_error: 0.0305
Epoch 99/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 1.1212e-04 - mean_absolute_error: 0.0078 - val_loss: 5.8592e-04 - val_mean_absolute_error: 0.0183
Epoch 100/100
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - loss: 1.0687e-04 - mean_absolute_error: 0.0074 - val_loss: 0.0019 - val_mean_absolute_error: 0.0335
<keras.src.callbacks.history.History at 0x7a68db5db9d0>
```

```
# predicting the values after running the model
test_predicted = model.predict(test_seq)
test_predicted[:5]
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step
array([[0.40194753, 0.40702906],
       [0.40166208, 0.4065613 ],
       [0.39763585, 0.40240565],
       [0.4002833 , 0.4054701 ],
       [0.40377948, 0.40909466]], dtype=float32)
```

```
# Inversing normalization/scaling on predicted data
test_inverse_predicted = MMS.inverse_transform(test_predicted)
test_inverse_predicted[:5]
```

```
array([[1416.1785, 1422.6233],
       [1415.6493, 1421.7562],
       [1408.185 , 1414.0544],
       [1413.0931, 1419.734 ],
       [1419.5748, 1426.4515]], dtype=float32)
```

STEP 5 : VISUALIZING ACTUAL VS PREDICTED DATA

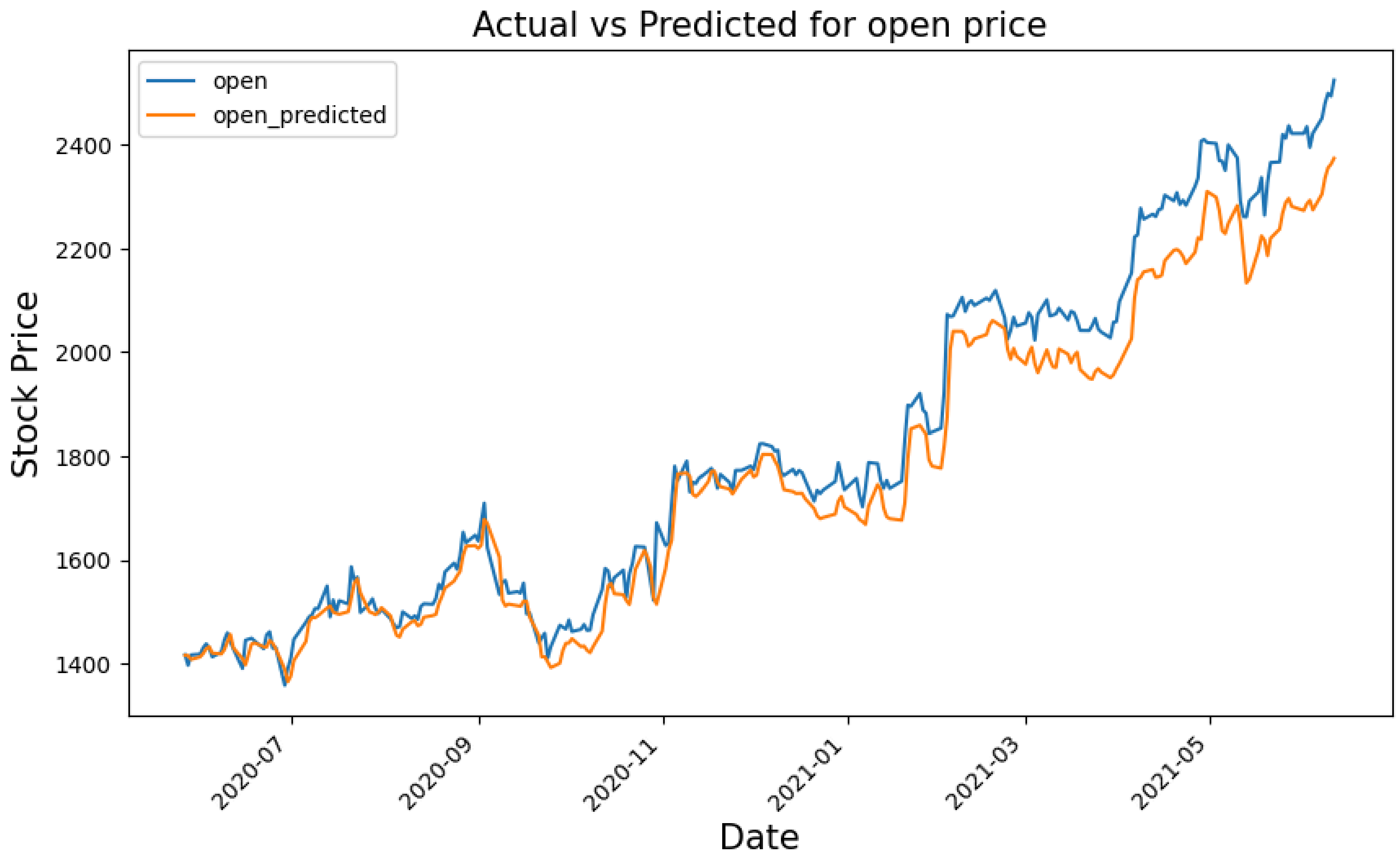
```
# Merging actual and predicted data for better visualization
df_merge = pd.concat([df.iloc[-264:].copy(),
                      pd.DataFrame(test_inverse_predicted,columns=['open_predicted','close_predicted'],
                                   index=df.iloc[-264:].index)], axis=1)
```

```
# Inversing normalization/scaling
df_merge[['open','close']] = MMS.inverse_transform(df_merge[['open','close']])
df_merge.head()
```

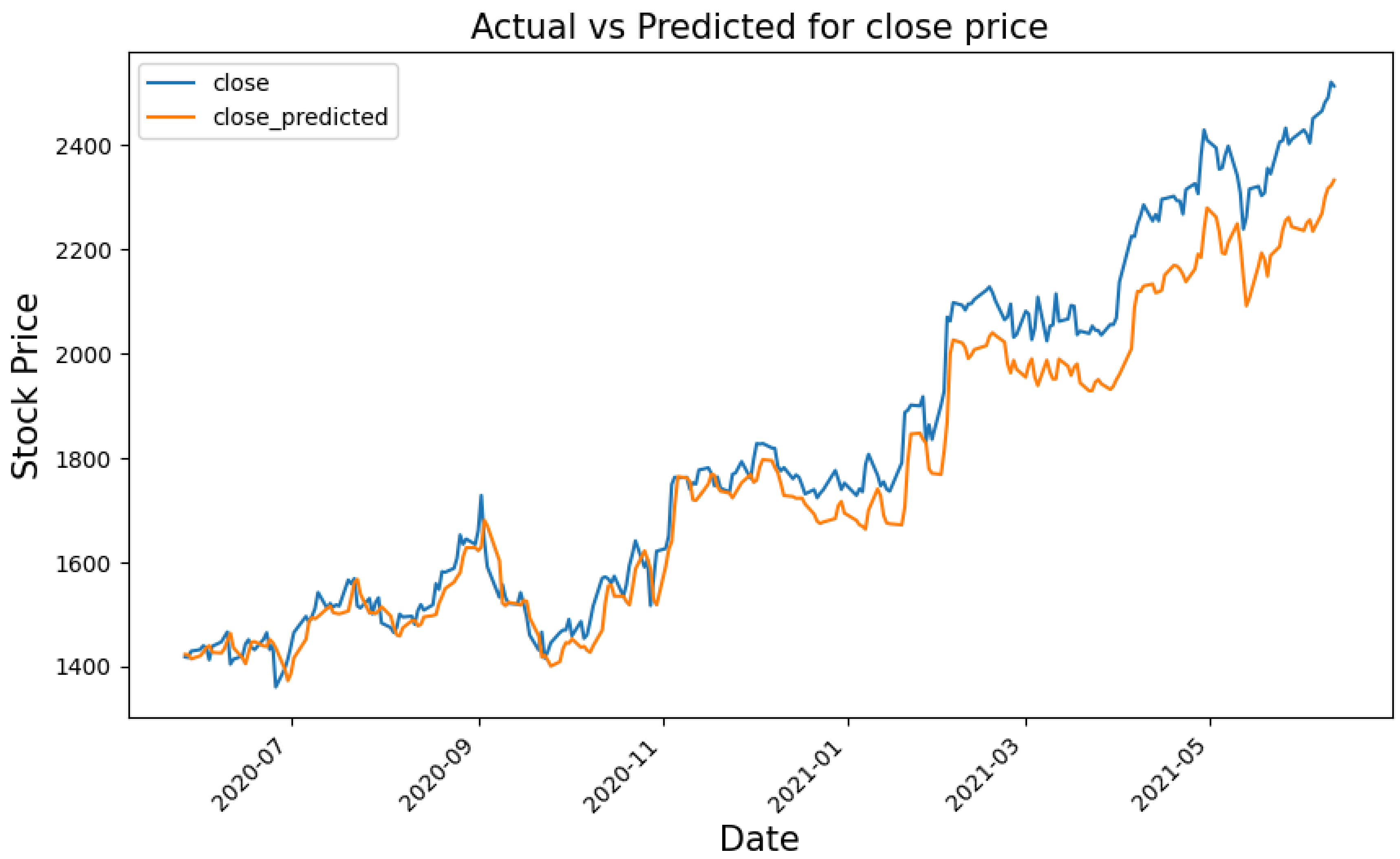
| date | open | close | open_predicted | close_predicted |
|---|---|---|---|---|
| 2020-05-27 | 1417.25 | 1417.84 | 1416.178467 | 1422.623291 |
| 2020-05-28 | 1396.86 | 1416.73 | 1415.649292 | 1421.756226 |
| 2020-05-29 | 1416.94 | 1428.92 | 1408.185059 | 1414.054443 |
| 2020-06-01 | 1418.39 | 1431.82 | 1413.093140 | 1419.734009 |
| 2020-06-02 | 1430.55 | 1439.22 | 1419.574829 | 1426.451538 |

Next steps: Generate code with df_merge | View recommended plots | New interactive sheet

```
# plotting the actual open and predicted open prices on date index
df_merge[['open','open_predicted']].plot(figsize=(10,6))
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for open price',size=15)
plt.show()
```




```
# plotting the actual close and predicted close prices on date index
df_merge[['close','close_predicted']].plot(figsize=(10,6))
plt.xticks(rotation=45)
plt.xlabel('Date',size=15)
plt.ylabel('Stock Price',size=15)
plt.title('Actual vs Predicted for close price',size=15)
plt.show()
```




STEP 6. PREDICTING UPCOMING 10 DAYS

```
new_index = pd.date_range(start=df_merge.index[-1] + pd.Timedelta(days=1), periods=10, freq='D')
new_rows = pd.DataFrame(columns=df_merge.columns, index=new_index)
df_merge = pd.concat([df_merge, new_rows])
```

```
df_merge.loc['2021-06-09':'2021-06-16']
```

| | open | close | open_predicted | close_predicted |
|---|---|---|---|---|
| 2021-06-09 | 2499.50 | 2491.40 | 2355.756104 | 2317.685059 |
| 2021-06-10 | 2494.01 | 2521.60 | 2362.784912 | 2322.546387 |
| 2021-06-11 | 2524.92 | 2513.93 | 2374.123047 | 2333.229492 |
| 2021-06-12 | NaN | NaN | NaN | NaN |
| 2021-06-13 | NaN | NaN | NaN | NaN |
| 2021-06-14 | NaN | NaN | NaN | NaN |
| 2021-06-15 | NaN | NaN | NaN | NaN |
| 2021-06-16 | NaN | NaN | NaN | NaN |

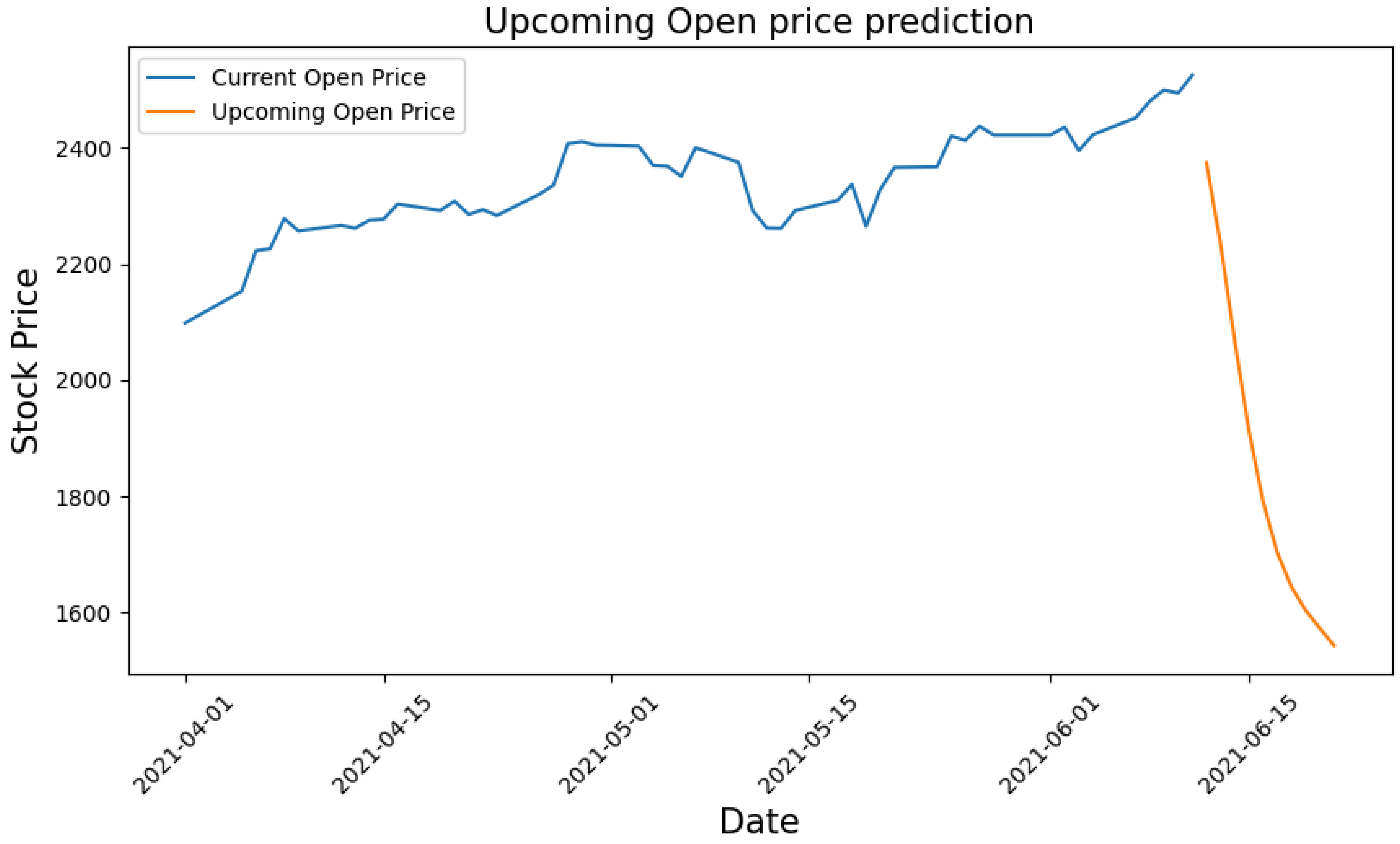
```
# creating a DataFrame and filling values of open and close column
upcoming_prediction = pd.DataFrame(columns=['open','close'],index=df_merge.index)
upcoming_prediction.index=pd.to_datetime(upcoming_prediction.index)
```

```
curr_seq = test_seq[-1:]

for i in range(-10,0):
  up_pred = model.predict(curr_seq)
  upcoming_prediction.iloc[i] = up_pred
  curr_seq = np.append(curr_seq[0][1:],up_pred,axis=0)
  curr_seq = curr_seq.reshape(test_seq[-1:].shape)
```

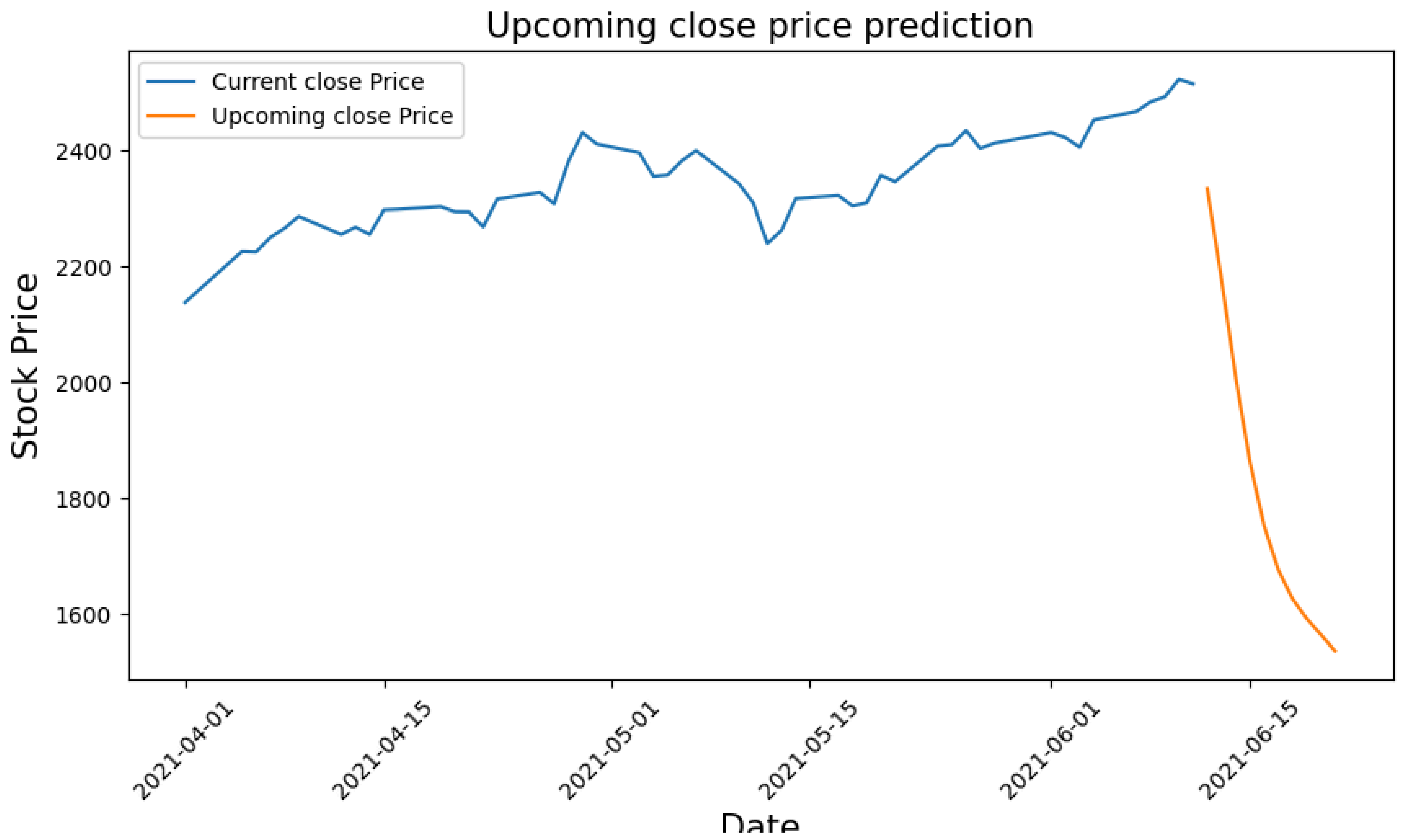
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```

```
# inversing Normalization/scaling
upcoming_prediction[['open','close']] = MMS.inverse_transform(upcoming_prediction[['open','close']])
```

```
# plotting Upcoming Open price on date index
fig,ax=plt.subplots(figsize=(10,5))
ax.plot(df_merge.loc['2021-04-01':,'open'],label='Current Open Price')
ax.plot(upcoming_prediction.loc['2021-04-01':,'open'],label='Upcoming Open Price')
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming Open price prediction',size=15)
ax.legend()
fig.show()
```




```
# plotting Upcoming Close price on date index
fig,ax=plt.subplots(figsize=(10,5))
ax.plot(df_merge.loc['2021-04-01':,'close'],label='Current close Price')
ax.plot(upcoming_prediction.loc['2021-04-01':,'close'],label='Upcoming close Price')
plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_xlabel('Date',size=15)
ax.set_ylabel('Stock Price',size=15)
ax.set_title('Upcoming close price prediction',size=15)
ax.legend()
fig.show()
```


Upcoming close price prediction
Current close Price
Upcoming close Price
Stock Price
2400
2200
2000
1800
1600
2021-04-01
2021-04-15
2021-05-01
2021-05-15
2021-06-01
2021-06-15
Date

# Data Engineer INTERN at HACKVEDA LIMITED

**AUTHOR : BANDANA PRAKASH**
**TASK 3 :** SALES PREDICTION USING PYTHON
**PURPOSE :** Predict sales based on advertising expenditure using the given dataset. The dataset contains information about advertising spending on different platforms (TV, Radio, and Newspaper) and the corresponding sales amount.

IMPORTING IMPORTANT LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

IMPORTING DATASET

```
# Raw URL of the CSV file
url = 'https://raw.githubusercontent.com/bandanaprakash/finalYearProject/main/MarketMinder%7C%20Real-Time%20Market%20Analysi

# Read the CSV file
df = pd.read_csv(url)

# Display the first few rows of the dataset
print(df.head())
```

```
      TV  Radio  Newspaper  Sales
0  230.1   37.8       69.2   22.1
1   44.5   39.3       45.1   10.4
2   17.2   45.9       69.3   12.0
3  151.5   41.3       58.5   16.5
4  180.8   10.8       58.4   17.9
```

Aim:- Sales prediction involves forecasting the amount of a product that customers will purchase, taking into account various factors such as advertising expenditure, target audience segmentation, and advertising platform selection.

Given dataset consist of the advertising platform and the related sales.Let's visulalize each platform

```
df.shape
```

```
(200, 4)
```

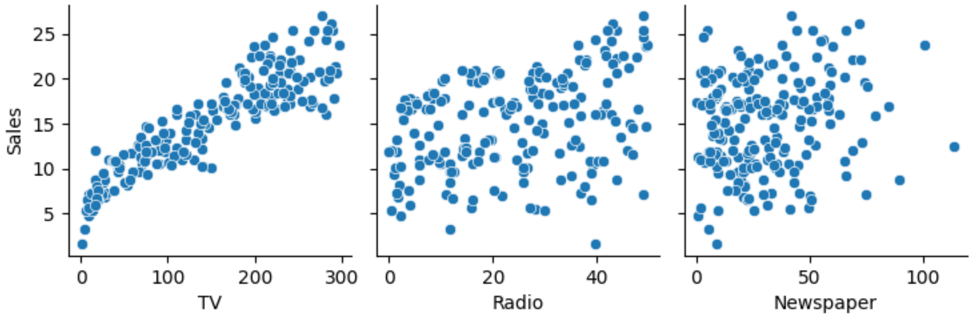
```
df.describe()
```

| | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| **count** | 200.000000 | 200.000000 | 200.000000 | 200.000000 |
| **mean** | 147.042500 | 23.264000 | 30.554000 | 15.130500 |
| **std** | 85.854236 | 14.846809 | 21.778621 | 5.283892 |
| **min** | 0.700000 | 0.000000 | 0.300000 | 1.600000 |
| **25%** | 74.375000 | 9.975000 | 12.750000 | 11.000000 |
| **50%** | 149.750000 | 22.900000 | 25.750000 | 16.000000 |
| **75%** | 218.825000 | 36.525000 | 45.100000 | 19.050000 |
| **max** | 296.400000 | 49.600000 | 114.000000 | 27.000000 |

Basic Observation

Avg expense spend is highest on TV Avg expense spend is lowest on Radio Max sale is 27 and min is 1.6

```
sns.pairplot(df, x_vars=['TV', 'Radio','Newspaper'], y_vars='Sales', kind='scatter')
plt.show()
```

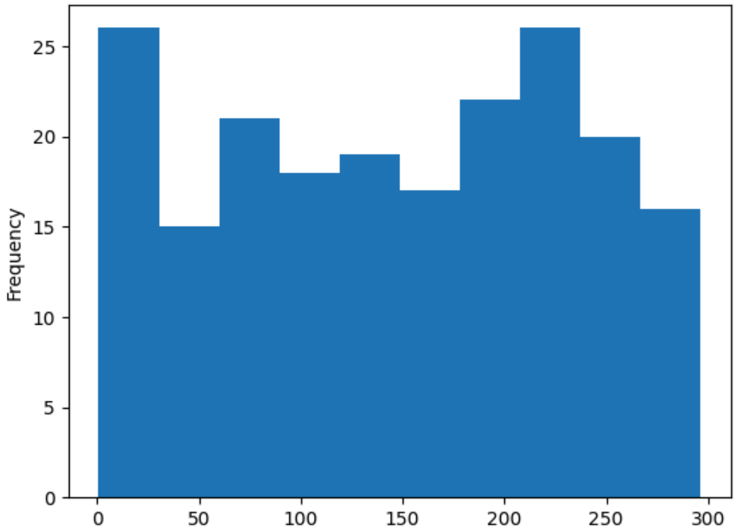



Pair Plot Observation

When advertising cost increases in TV Ads the sales will increase as well. While the for newspaper and radio it is bit unpredictable.

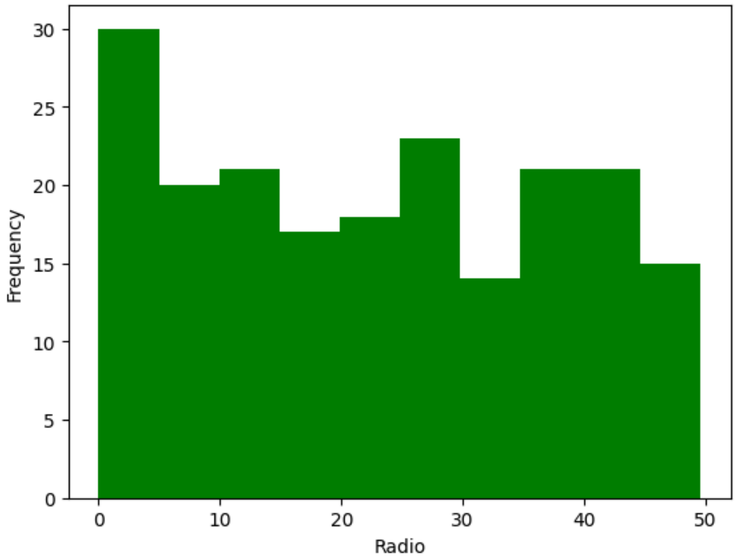
```
df['TV'].plot.hist(bins=10)
```

```
<Axes: ylabel='Frequency'>
```



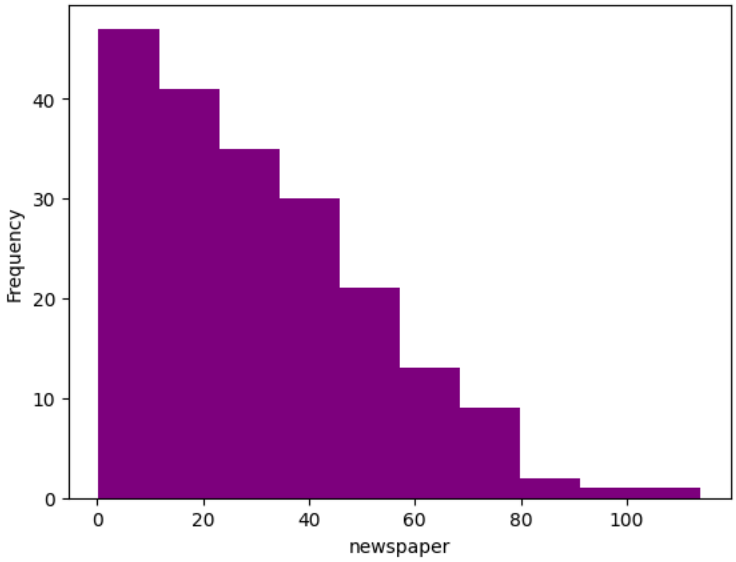

```
df['Radio'].plot.hist(bins=10, color="green", xlabel="Radio")
```

```
<Axes: xlabel='Radio', ylabel='Frequency'>
```




```
df['Newspaper'].plot.hist(bins=10,color="purple", xlabel="newspaper")
```

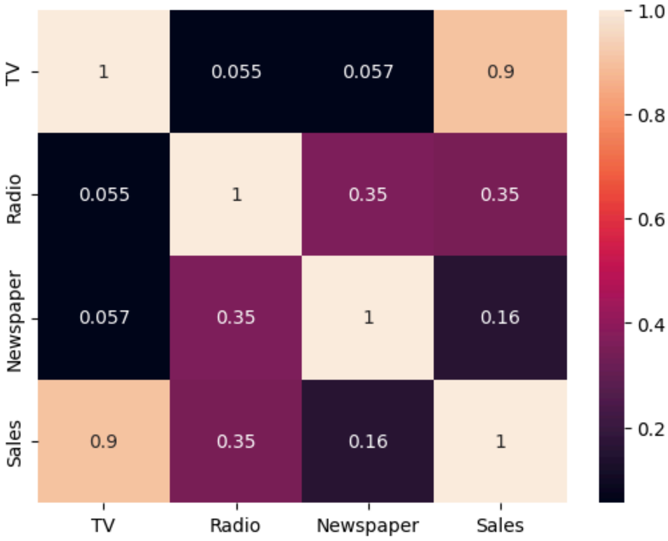
```
<Axes: xlabel='newspaper', ylabel='Frequency'>
```




Histogram Observation

The majority sales is the result of low advertising cost in newspaper

```
sns.heatmap(df.corr(),annot = True)
plt.show()
```




SALES IS HIGHLY COORELATED WITH THE TV

Lets train our model using linear regression as it is coorelated with only one variable TV

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['TV']], df[['Sales']], test_size = 0.3,random_state=0)
```

```
print(X_train)
```

```
        TV
131  265.2
96   197.6
181  218.5
19   147.3
153  171.3
..     ...
67   139.3
192   17.2
117   76.4
47   239.9
172   19.6

[140 rows x 1 columns]
```

```
print(y_train)
```

```
     Sales
131   17.7
96    16.7
181   17.2
19    14.6
153   16.0
..     ...
67    13.4
192    5.9
117    9.4
47    23.2
172    7.6

[140 rows x 1 columns]
```

```
print(X_test)
```

```
107   90.4
98   289.7
177  170.2
182   56.2
5      8.7
146  240.1
12    23.8
152  197.6
61   261.3
125   87.2
180  156.6
154  187.8
80    76.4
7    120.2
33   265.6
130    0.7
37    74.7
74   213.4
183  287.6
145  140.3
45   175.1
159  131.7
60    53.5
123  123.1
179  165.6
185  205.0
122  224.0
44    25.1
16    67.8
55   198.9
150  280.7
111  241.7
22    13.2
189   18.7
129   59.6
4    180.8
83    68.4
106   25.0
134   36.9
66    31.5
26   142.9
113  209.6
168  215.4
63   102.7
8      8.6
75    16.9
118  125.7
143  104.6
71   109.8
124  229.5
184  253.8
97   184.9
149   44.7
24    62.3
30   292.9
160  172.5
40   202.5
56     7.3
```

```
print(y_test)
```

```
107   12.0
98    25.4
177   16.7
182    8.7
5      7.2
```

```
61    24.2
125   10.6
180   15.5
154   20.6
80    11.8
7     13.2
33    17.4
130    1.6
37    14.7
74    17.0
183   26.2
145   10.3
45    16.1
159   12.9
60     8.1
123   15.2
179   17.6
185   22.6
122   16.6
44     8.5
16    12.5
55    23.7
150   16.1
111   21.8
22     5.6
189    6.7
129    9.7
4     17.9
83    13.6
106    7.2
134   10.8
66    11.0
26    15.0
113   20.9
168   17.1
63    14.0
8      4.8
75     8.7
118   15.9
143   10.4
71    12.4
124   19.7
184   17.6
97    20.5
149   10.1
24     9.7
30    21.4
160   16.4
40    16.6
56     5.5
```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
res= model.predict(X_test)
print(res)
```

```
[12.09159447]
[22.99968079]
[16.45920756]
[10.21976029]
[ 7.6199906 ]
[20.28497391]
[ 8.4464437 ]
[17.95886418]
[21.44529217]
[11.91645209]
[15.71485245]
[17.42249065]
[11.32534656]
[13.72260788]
[21.68063975]
[ 7.18213465]
[11.23230217]
[18.82362968]
[22.88474361]
[14.82272095]
[16.72739433]
[14.35202581]
[10.07198391]
[13.88133066]
```

```
[19.40578881]
[ 8.51759529]
[10.85465142]
[18.03001578]
[22.50709285]
[20.3725451 ]
[ 7.86628457]
[ 8.16731053]
[10.40584907]
[17.03936669]
[10.88749061]
[ 8.51212209]
[ 9.16343282]
[ 8.86788005]
[14.96502414]
[18.61564811]
[18.93309367]
[12.76479799]
[ 7.6145174 ]
[ 8.06879294]
[14.02363385]
[12.86878878]
[13.15339515]
[19.70481478]
[21.03480222]
[17.26376787]
[ 9.59034237]
[10.55362545]
[23.17482317]
[16.58509115]
[18.22705095]
[ 7.54336581]]
```

```
model.coef_
```

```
array([[0.05473199]])
```

```
model.intercept_
```

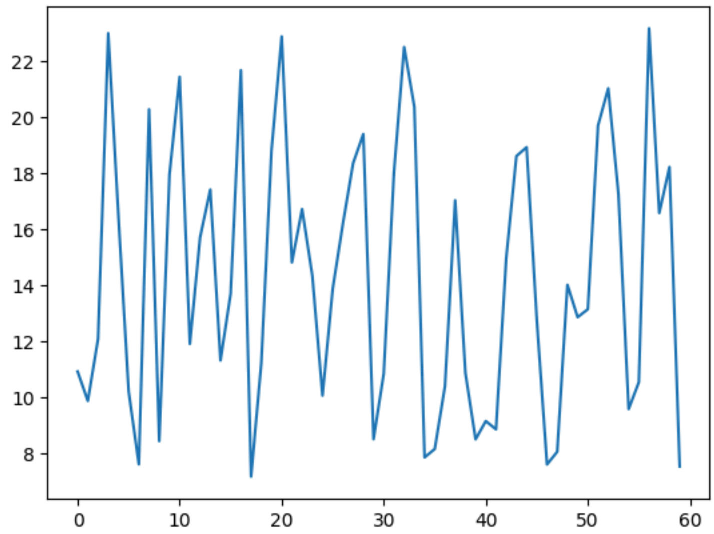
```
array([7.14382225])
```

```
0.05473199* 69.2 + 7.14382225
```

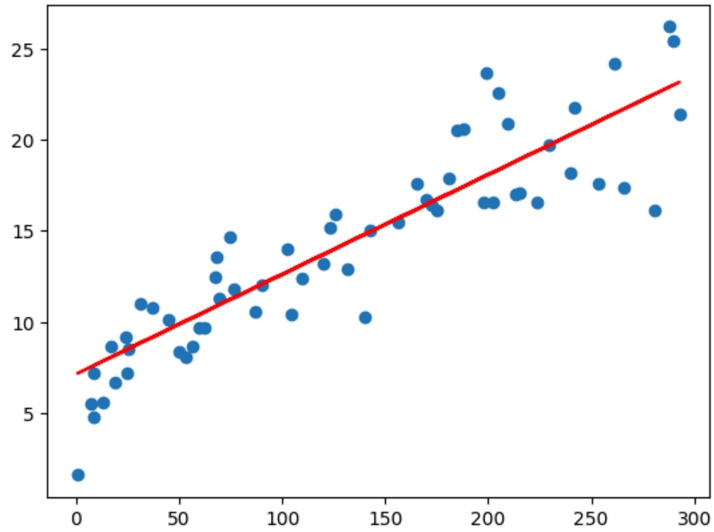
```
10.931275958
```

```
plt.plot(res)
```

```
[<matplotlib.lines.Line2D at 0x7ef74a4f3ad0>]
```




```
plt.scatter(X_test, y_test)
plt.plot(X_test, 7.14382225 + 0.05473199 * X_test, 'r')
plt.show()
```




Concluding with saying that above mention solution is successfully able to predict the sales using advertising platform datasets

# Data Engineer INTERN at HACKVEDA LIMITED

**AUTHOR: BANDANA PRAKASH**
**TASK 4:** Stock_Market_Prediction_Model_Creation
**PURPOSE :** to develop a model for analyzing and predicting stock market trends.

## It involves:

**Data Collection:** Fetching historical stock price data (e.g., Google stock) using yfinance.
**Data Preparation:** Cleaning and structuring the data for analysis.
**Trend Analysis:** Identifying patterns in stock prices over time.
**Prediction Modeling:** Building a predictive model to forecast future stock prices based on historical data.
This project aims to assist in making informed investment decisions by leveraging data-driven insights and predictive analytics.

## Steps Involved

**Import Libraries:** Load necessary libraries such as numpy, pandas, matplotlib, and yfinance.
**Define Timeframe:** Set the start and end dates for the historical data to be analyzed (from January 1, 2012, to December 21, 2022).
**Fetch Data:** Use yfinance to download historical stock data for Google (GOOG).
**Reset Index:** Prepare the data by resetting the index for easier manipulation.
**Data Exploration:** Inspect the dataset to understand its structure and contents.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
```

```
start = '2012-01-01'
end = '2022-12-21'
stock = 'GOOG'

data = yf.download(stock, start, end)
```

```
[*********************100%***********************]  1 of 1 completed
```

```
data.reset_index(inplace=True)
```

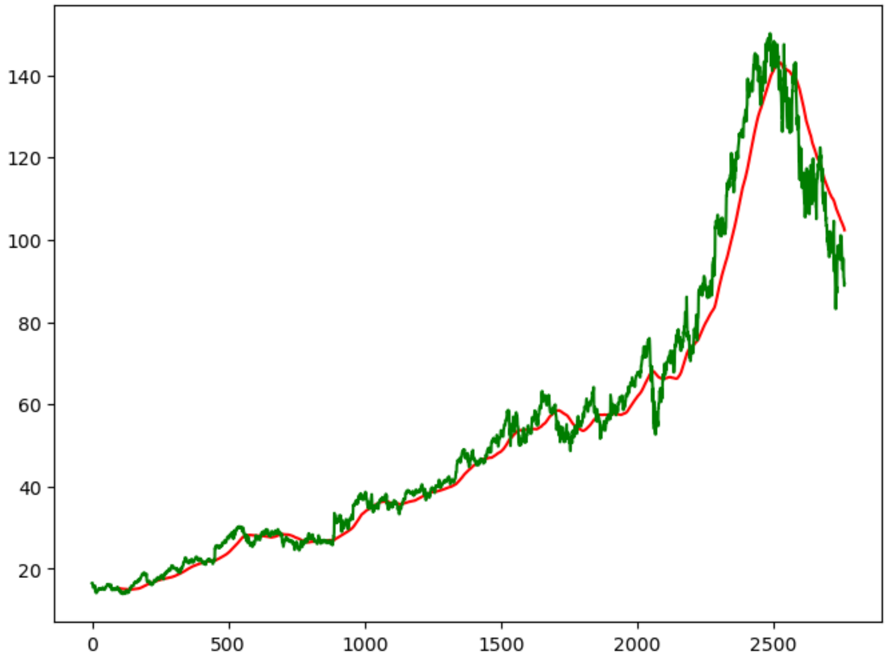
```
data
```

| Price | Date | Close | High | Low | Open | Volume |
|---|---|---|---|---|---|---|
| Ticker | | GOOG | GOOG | GOOG | GOOG | GOOG |
| 0 | 2012-01-03 | 16.513794 | 16.581795 | 16.190173 | 16.204321 | 147611217 |
| 1 | 2012-01-04 | 16.585020 | 16.633911 | 16.394919 | 16.504364 | 114989399 |
| 2 | 2012-01-05 | 16.354961 | 16.478056 | 16.285969 | 16.432392 | 131808205 |
| 3 | 2012-01-06 | 16.131853 | 16.379531 | 16.126144 | 16.358435 | 108119746 |
| 4 | 2012-01-09 | 15.447884 | 16.056905 | 15.417357 | 16.044495 | 233776981 |
| ... | ... | ... | ... | ... | ... | ... |
| 2756 | 2022-12-14 | 94.968765 | 96.871931 | 93.603675 | 95.197945 | 26452900 |
| 2757 | 2022-12-15 | 90.873482 | 93.693352 | 90.106242 | 93.205108 | 28298800 |
| 2758 | 2022-12-16 | 90.534691 | 91.421504 | 89.687736 | 90.873470 | 48485500 |
| 2759 | 2022-12-19 | 88.830826 | 90.873482 | 88.606633 | 90.554628 | 23020500 |
| 2760 | 2022-12-20 | 89.309097 | 89.458562 | 87.724794 | 88.412326 | 21976800 |

2761 rows × 6 columns

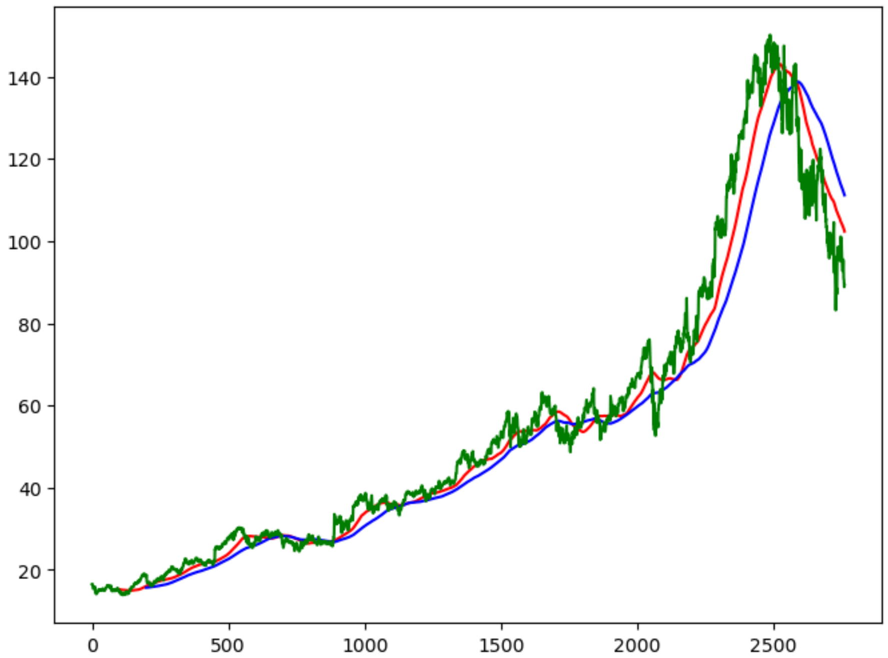
```
ma_100_days = data.Close.rolling(100).mean()
```

```
plt.figure(figsize=(8,6))
plt.plot(ma_100_days, 'r')
plt.plot(data.Close, 'g')
plt.show()
```




```
ma_200_days = data.Close.rolling(200).mean()
```

```
plt.figure(figsize=(8,6))
plt.plot(ma_100_days, 'r')
plt.plot(ma_200_days,'b')
plt.plot(data.Close,'g')
plt.show()
```




```
data.dropna(inplace=True)
```

```
data_train = pd.DataFrame(data.Close[0: int(len(data)*0.80)])
data_test = pd.DataFrame(data.Close[int(len(data)*0.80): len(data)])
```

```
data_train.shape[0]
```

```
2208
```

```
data_test.shape[0]
```

```
553
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))


data_train_scale = scaler.fit_transform(data_train)


x = []
y = []

for i in range(100, data_train_scale.shape[0]):
    x.append(data_train_scale[i-100:i])
    y.append(data_train_scale[i,0])



x, y = np.array(x), np.array(y)


from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential


model = Sequential()
model.add(LSTM(units = 50, activation = 'relu', return_sequences = True,
               input_shape = ((x.shape[1],1))))
model.add(Dropout(0.2))

model.add(LSTM(units = 60, activation='relu', return_sequences = True))
model.add(Dropout(0.3))

model.add(LSTM(units = 80, activation = 'relu', return_sequences = True))
model.add(Dropout(0.4))

model.add(LSTM(units = 120, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(units =1))


model.compile(optimizer = 'adam', loss = 'mean_squared_error')


model.fit(x,y, epochs = 50, batch_size =32, verbose =1)
```

```
Epoch 1/50
2025-01-23 18:37:52.856514: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
66/66 [==============================] - 9s 113ms/step - loss: 0.0372
Epoch 2/50
66/66 [==============================] - 8s 116ms/step - loss: 0.0075
Epoch 3/50
66/66 [==============================] - 8s 118ms/step - loss: 0.0066
Epoch 4/50
66/66 [==============================] - 8s 117ms/step - loss: 0.0053
Epoch 5/50
66/66 [==============================] - 8s 118ms/step - loss: 0.0056
Epoch 6/50
66/66 [==============================] - 8s 118ms/step - loss: 0.0046
Epoch 7/50
66/66 [==============================] - 9s 141ms/step - loss: 0.0044
Epoch 8/50
66/66 [==============================] - 10s 147ms/step - loss: 0.0043
Epoch 9/50
66/66 [==============================] - 10s 153ms/step - loss: 0.0041
Epoch 10/50
66/66 [==============================] - 10s 147ms/step - loss: 0.0040
Epoch 11/50
66/66 [==============================] - 10s 148ms/step - loss: 0.0041
Epoch 12/50
66/66 [==============================] - 10s 149ms/step - loss: 0.0039
Epoch 13/50
66/66 [==============================] - 10s 146ms/step - loss: 0.0037
Epoch 14/50
66/66 [==============================] - 10s 148ms/step - loss: 0.0037
Epoch 15/50
66/66 [==============================] - 9s 139ms/step - loss: 0.0040
Epoch 16/50
66/66 [==============================] - 9s 139ms/step - loss: 0.0032
Epoch 17/50
66/66 [==============================] - 9s 139ms/step - loss: 0.0033
Epoch 18/50
66/66 [==============================] - 9s 139ms/step - loss: 0.0034
Epoch 19/50
66/66 [==============================] - 9s 140ms/step - loss: 0.0030
Epoch 20/50
66/66 [==============================] - 9s 139ms/step - loss: 0.0029
Epoch 21/50
66/66 [==============================] - 9s 143ms/step - loss: 0.0028
```

```
Epoch 22/50
66/66 [==============================] - 9s 136ms/step - loss: 0.0031
Epoch 23/50
66/66 [==============================] - 9s 138ms/step - loss: 0.0032
Epoch 24/50
66/66 [==============================] - 9s 142ms/step - loss: 0.0028
Epoch 25/50
66/66 [==============================] - 9s 144ms/step - loss: 0.0025
Epoch 26/50
66/66 [==============================] - 9s 141ms/step - loss: 0.0027
Epoch 27/50
66/66 [==============================] - 9s 143ms/step - loss: 0.0028
Epoch 28/50
66/66 [==============================] - 9s 133ms/step - loss: 0.0026
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 60)           26640

 dropout_1 (Dropout)         (None, 100, 60)           0

 lstm_2 (LSTM)               (None, 100, 80)           45120

 dropout_2 (Dropout)         (None, 100, 80)           0

 lstm_3 (LSTM)               (None, 120)               96480

 dropout_3 (Dropout)         (None, 120)               0

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 178,761
Trainable params: 178,761
Non-trainable params: 0
_________________________________________________________________
```

```
pas_100_days = data_train.tail(100)
```

```
data_test = pd.concat([pas_100_days, data_test], ignore_index=True)
```

```
data_test_scale  =  scaler.fit_transform(data_test)
```

```
x = []
y = []

for i in range(100, data_test_scale.shape[0]):
    x.append(data_test_scale[i-100:i])
    y.append(data_test_scale[i,0])
x, y = np.array(x), np.array(y)
```

```
y_predict = model.predict(x)
```

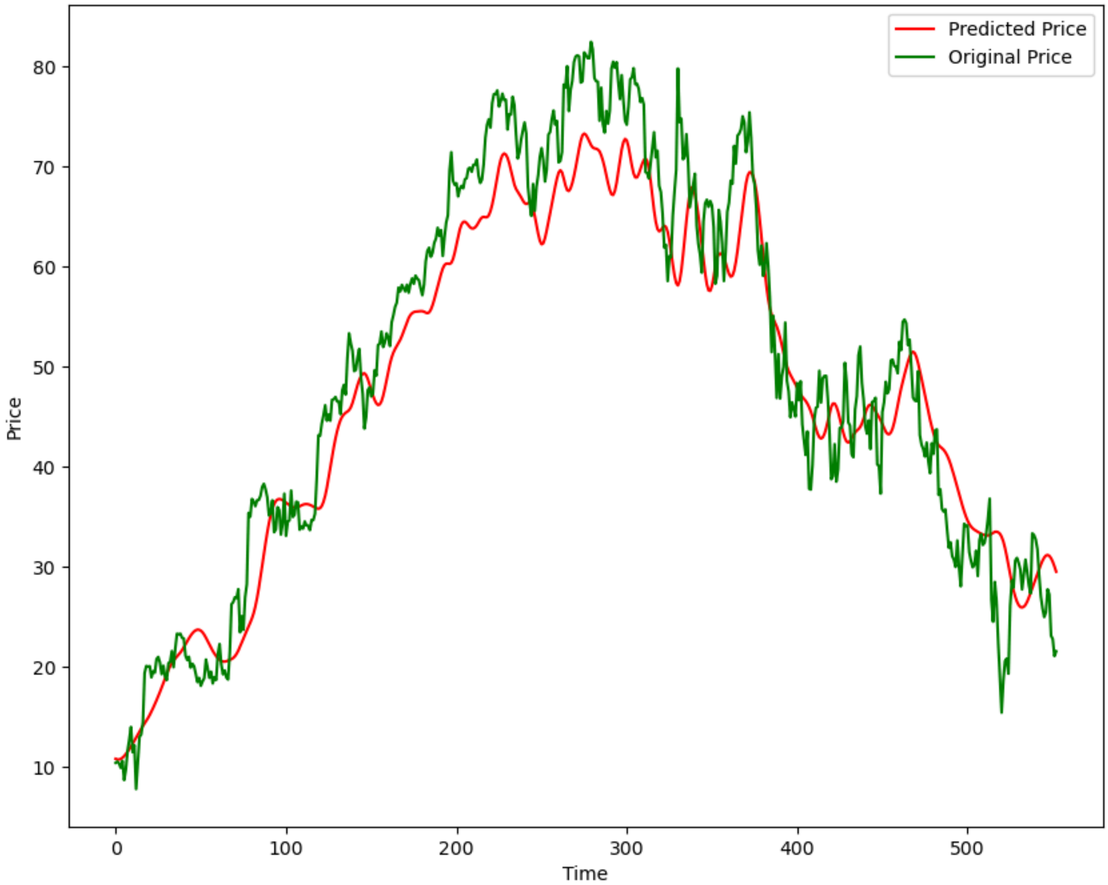
```
18/18 [==============================] - 1s 29ms/step
```

```
scale =1/scaler.scale_
```

```
y_predict = y_predict*scale
```

```
y = y*scale
```

```
plt.figure(figsize=(10,8))
plt.plot(y_predict, 'r', label = 'Predicted Price')
plt.plot(y, 'g', label = 'Original Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```




```
model.save('Stock Predictions Model.keras')
```

# Data Engineer INTERN at HACKVEDA LIMITED

**AUTHOR : BANDANA PRAKASH**

**TASK 5 :** CREDIT CARD FRAUD DETECTION

**PURPOSE :** Build a machine learning model to identify fraudulent credit card transactions.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
file=pd.read_csv("creditcard.csv")
```

```
file.head(10)
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.1104 |
| 1 | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0.1012 |
| 2 | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0.9094 |
| 3 | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | -0.1903 |
| 4 | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | -0.1374 |
| 5 | 2.0 | -0.425966 | 0.960523 | 1.141109 | -0.168252 | 0.420987 | -0.029728 | 0.476201 | 0.260314 | -0.568671 | ... | -0.208254 | -0.559825 | -0.0263 |
| 6 | 4.0 | 1.229658 | 0.141004 | 0.045371 | 1.202613 | 0.191881 | 0.272708 | -0.005159 | 0.081213 | 0.464960 | ... | -0.167716 | -0.270710 | -0.1541 |
| 7 | 7.0 | -0.644269 | 1.417964 | 1.074380 | -0.492199 | 0.948934 | 0.428118 | 1.120631 | -3.807864 | 0.615375 | ... | 1.943465 | -1.015455 | 0.0575 |
| 8 | 7.0 | -0.894286 | 0.286157 | -0.113192 | -0.271526 | 2.669599 | 3.721818 | 0.370145 | 0.851084 | -0.392048 | ... | -0.073425 | -0.268092 | -0.2042 |
| 9 | 9.0 | -0.338262 | 1.119593 | 1.044367 | -0.222187 | 0.499361 | -0.246761 | 0.651583 | 0.069539 | -0.736727 | ... | -0.246914 | -0.633753 | -0.1207 |

10 rows × 31 columns

```
file.describe()
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| count | 284807.000000 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+ |
| mean | 94813.859575 | 1.168375e-15 | 3.416908e-16 | -1.379537e-15 | 2.074095e-15 | 9.604066e-16 | 1.487313e-15 | -5.556467e-16 | 1.213481e- |
| std | 47488.145955 | 1.958696e+00 | 1.651309e+00 | 1.516255e+00 | 1.415869e+00 | 1.380247e+00 | 1.332271e+00 | 1.237094e+00 | 1.194353e+ |
| min | 0.000000 | -5.640751e+01 | -7.271573e+01 | -4.832559e+01 | -5.683171e+00 | -1.137433e+02 | -2.616051e+01 | -4.355724e+01 | -7.321672e+ |
| 25% | 54201.500000 | -9.203734e-01 | -5.985499e-01 | -8.903648e-01 | -8.486401e-01 | -6.915971e-01 | -7.682956e-01 | -5.540759e-01 | -2.086297e- |
| 50% | 84692.000000 | 1.810880e-02 | 6.548556e-02 | 1.798463e-01 | -1.984653e-02 | -5.433583e-02 | -2.741871e-01 | 4.010308e-02 | 2.235804e- |
| 75% | 139320.500000 | 1.315642e+00 | 8.037239e-01 | 1.027196e+00 | 7.433413e-01 | 6.119264e-01 | 3.985649e-01 | 5.704361e-01 | 3.273459e- |
| max | 172792.000000 | 2.454930e+00 | 2.205773e+01 | 9.382558e+00 | 1.687534e+01 | 3.480167e+01 | 7.330163e+01 | 1.205895e+02 | 2.000721e+ |

8 rows × 31 columns

```
file.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
```

```
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
file['Class'].value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

```
normal=file[file.Class==0]
```

```
fraud=file[file.Class==1]
```

```
print(normal.shape)
```

```
(284315, 31)
```

```
print(fraud.shape)
```

```
(492, 31)
```

```
normal.Amount.describe()
```

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
fraud.Amount.describe()
```

```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

```
file.groupby('Class').mean()
```

| Class | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V20 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 94838.202258 | 0.008258 | -0.006271 | 0.012171 | -0.007860 | 0.005453 | 0.002419 | 0.009637 | -0.000987 | 0.004467 | ... | -0.000644 | -0.00 |
| 1 | 80746.806911 | -4.771948 | 3.623778 | -7.033281 | 4.542029 | -3.151225 | -1.397737 | -5.568731 | 0.570636 | -2.581123 | ... | 0.372319 | 0.71 |

2 rows × 30 columns

```
normal_sample=normal.sample(n=492)
```

```
new_file=pd.concat([normal_sample,fraud],axis=0)
```

```
new_file.head(10)
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **32993** | 37045.0 | 1.273090 | -0.744403 | 1.083617 | -0.682701 | -1.537089 | -0.502750 | -1.064374 | 0.039786 | -0.738666 | ... | 0.465367 | 1.22964( |
| **176252** | 122689.0 | -0.619340 | 0.650909 | 0.853761 | -0.441992 | 1.189456 | 0.079074 | 1.075137 | -0.224906 | -0.173498 | ... | -0.337960 | -0.88283( |
| **71866** | 54473.0 | -1.459553 | 0.016956 | 1.063610 | -1.484100 | -0.244744 | -1.080333 | -0.089253 | 0.466594 | 1.243897 | ... | 0.144194 | 0.45999( |
| **189917** | 128610.0 | 1.778519 | -0.112447 | -1.234223 | 0.996746 | 0.961386 | 1.807237 | -0.624287 | 0.614767 | 0.549400 | ... | -0.183639 | -0.42127( |
| **153148** | 98024.0 | -0.557542 | 1.064676 | 0.524862 | -1.771705 | 1.141241 | -0.310842 | 0.624603 | 0.006453 | 1.478922 | ... | -0.514084 | -1.28654( |
| **91669** | 63576.0 | 0.910530 | -1.359016 | 0.862437 | -0.590947 | -1.567072 | 0.005582 | -0.959290 | 0.164846 | -0.678745 | ... | 0.586836 | 1.21331( |
| **248153** | 153810.0 | 2.102483 | -1.302057 | 0.379806 | -0.486967 | -1.830271 | -0.165263 | -1.633904 | 0.141010 | 1.060734 | ... | 0.134621 | 0.69733( |
| **175828** | 122505.0 | -2.783805 | -2.928222 | -1.500618 | -1.979360 | 1.645353 | 0.802380 | 1.036764 | -0.020237 | -1.068077 | ... | 0.242157 | 1.41124( |
| **201518** | 133915.0 | 2.006453 | -1.760294 | -0.688837 | -1.337221 | -1.645638 | -0.848000 | -0.970753 | -0.328486 | -1.233229 | ... | -0.044086 | 0.14975( |
| **57869** | 48115.0 | 1.314915 | -0.980378 | -0.032665 | -2.770975 | -1.047365 | -0.705180 | -0.491240 | -0.076395 | 0.571959 | ... | -0.416346 | -0.48188( |

10 rows × 31 columns

```
new_file['Class'].value_counts()
```

```
Class
0    492
1    492
Name: count, dtype: int64
```

```
new_file.groupby('Class').mean()
```

| Class | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V20 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 96327.323171 | 0.077345 | -0.022423 | 0.081215 | -0.129899 | 0.029804 | -0.009543 | 0.086710 | -0.040480 | -0.027447 | ... | -0.071245 | 0.017 |
| **1** | 80746.806911 | -4.771948 | 3.623778 | -7.033281 | 4.542029 | -3.151225 | -1.397737 | -5.568731 | 0.570636 | -2.581123 | ... | 0.372319 | 0.713 |

2 rows × 30 columns

```
X=new_file.drop(columns='Class',axis=1)

Y=new_file['Class']

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=2)

model=LogisticRegression()

model.fit(X_train,Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
X_train_prediction=model.predict(X_train)

training_data_acuracy=accuracy_score(X_train_prediction,Y_train)*100

print(f"Training Data Accuracy: {training_data_acuracy}%")
```

```
Training Data Accuracy: 93.90088945362135%
```

```
X_test_prediction=model.predict(X_test)

test_data_accuracy=accuracy_score(X_test_prediction,Y_test)*100

print(f"Test Data Accuracy: {test_data_accuracy}%")
```

```
Test Data Accuracy: 91.87817258883248%
```