FUTURE SALES PREDICTION

PHASE 3 : DEVELOPMENT PHASE PART-1

LOADING AND DATA PREPROCESSING

INTRODUCTION

Future sales prediction, often referred to as sales forecasting, is a crucial aspect of business strategy. It involves the use of statistical models and predictive analytics to estimate a company's future sales performance. By examining historical sales data, market trends, and other relevant factors, businesses can make informed decisions regarding inventory management, resource allocation, and overall growth strategies. Sales predictions enable companies to adapt to changing market conditions, optimize marketing efforts, and ensure sustainable success in a competitive business landscape.

WORKS DONE IN PREVIOUS PHASES

DEFINITION PHASE

In this phase we defined the problem that develop a predictive model that uses historical sales data to forecast future sales for a retail company. The objective is to create a tool that enables the company to optimize inventory management and make informed business decisions based on data driven sales predictions. These involves data preprocessing, feature engineering, model selection, training, and evaluation.

INNOVATION PHASE

In the innovation phase of our future sales prediction project, you can explore advanced techniques and methods to improve the accuracy of your sales forecasting model like Prophet Forecasting model and LSTM

PHASE 3

DEVELOPMENT PHASE

These phase can be executed using three parts

- Loading and Pre-processing data
- Training and Testing data
- Model testing and Displaying Output

LOADING DATA

For loading the data we can use the dataset link

https://www.kaggle.com/datasets/chakradharmattapalli/future-salesprediction

IMPORTING LIBRARIES

We importing the necessary Python libraries, such as

- Pandas for data manipulation
- NumPy for analysis,
- Matplotlib for visualization.

Here is the code:

```
import pandas as pd
import numpy as np
import warnings
```

LOADING THE DATASET

- To load data points from a file (e.g., a CSV file), you can use the **pd.read.csv()** function.
- This function reads the data from the file and stores it in a Pandas DataFrame.

Here is the code:

```
data=pd.read_csv(r"C:\Users\ridha\Downloads\sales.csv")
```

EXPLORE AND CLEAN DATA

head():

- The head() function allows you to view the first few rows of the DataFrame.
- We can specify the number of rows you want to see by passing an integer as an argument to head().
- By Default, it shows first 5 rows

Here is the code:

```
data.head(10)
```

Output:

	S.no	TV	Radio	Newspaper	Sales
0	1.0	230.1	37.8	69.2	22.1
1	2.0	44.5	39.3	45.1	10.4
2	3.0	17.2	45.9	69.3	9.3
3	4.0	151.5	41.3	58.5	18.5
4	5.0	180.8	10.8	58.4	12.9
5	6.0	8.7	48.9	75.0	7.2
6	7.0	57.5	32.8	23.5	11.8
7	8.0	120.2	19.6	11.6	13.2

8	9.0	8.6	2.1	1.0	4.8
9	10.0	199.8	2.6	21.2	10.6

tail():

- The data.tail() function allows you to view the last few rows of the DataFrame.
  .
- You can specify the number of rows you want to see by passing an integer as an argument to data.tail().
- By default, it displays the last five rows

Here is the code:

```
data.tail(4)
```

Output:

	S.no	TV	Radio	Newspaper	Sales
196	197.0	94.2	4.9	8.1	9.7
197	198.0	177.0	9.3	6.4	12.8
198	199.0	283.6	42.0	66.2	25.5
199	13.4	13.4	13.4	13.4	13.4

shape:

- To retrieve the dimensions (number of rows and columns) of the DataFrame, you can access the shape attribute directly
- The data.shape attribute is a valuable tool when we need to know the size of your DataFrame.
- It's useful for data exploration, data preprocessing, and understanding the structure of your dataset.

- It's useful for data exploration, data preprocessing, and understanding the structure of your dataset.

Here is the code:

```
data.shape
```

Output:

(200,5)

columns:

- To retrieve the column names (column labels) of the DataFrame, we can access the columns attribute directly
- The data.columns attribute is a useful tool when you need to access and manipulate column names in a Pandas DataFrame.
- It's commonly used for data analysis and data manipulation tasks.

Here is the code:

```
data.columns
```

Output:

Index(['S.no', 'TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')

column to list()

- You can use this list to perform operations based on column names, as needed for your data analysis tasks.
- Using data.columns.values.tolist() is a convenient way to convert column names into a standard Python list, making it easier to work with column names in a more Pythonic manner.

Here is the code:

```
data.columns.values.tolist()
```

Output:

['S.no', 'TV', 'Radio', 'Newspaper', 'Sales']

Info()

To obtain an overview of the DataFrame's information, you can simply call the info() method on the DataFrame

Here is the code:

```
data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   S.no       200 non-null    float64
 1   TV         200 non-null    float64
 2   Radio      200 non-null    float64
 3   Newspaper  200 non-null    float64
 4   Sales      200 non-null    float64
dtypes: float64(5)
memory usage: 7.9 KB
```

When we run this, we'll see an output that includes the following details:

- The number of non-null (non-missing) values in each column.

- The data type of each column (e.g., int64, float64, object, datetime64, etc.).
- The memory usage of the DataFrame.

Additionally, it provides a summary count of the non-null entries and the memory footprint, which is useful for understanding the overall structure and size of our data.

describe():

To generate summary statistics for the numerical columns in the DataFrame, we can call the describe() method

Here is the code:

```
data.describe()
```

Output:

S.no	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	99.567000	145.949000	23.288000	30.577500
std	57.771081	86.157663	14.826852	21.757446
min	1.000000	0.700000	0.000000	0.300000
25%	49.750000	72.700000	10.075000	12.875000
50%	99.500000	148.500000	22.900000	25.750000
75%	149.250000	218.425000	36.525000	45.100000

isnull().sum()

To count the number of missing values in each column of the DataFrame, we can apply the isnull() method followed by sum()

Here is the code:

```
data.isnull().sum()
```

Output:

S.no 0

TV 0

Radio 0

Newspaper   0

Sales   0

dtype: int64

VISUALISE DATA

Matplotlib and Seaborn:

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- It provides a wide range of functions for creating various types of plots, from basic line charts to complex 3D visualizations.
- To create a basic plot, you typically use plt as an alias for matplotlib.pyplot, and you can call various functions to customize your plot.
- Seaborn is a high-level data visualization library that works on top of Matplotlib.
- It's designed for creating informative and attractive statistical graphics.
- Seaborn provides a simpler interface for creating a variety of statistical plots, and it's particularly well-suited for visualizing complex datasets.
- It also offers built-in themes and color palettes to make your plots aesthetically pleasing.

Here is the code:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Box Plot (Box-and-Whisker Plot)

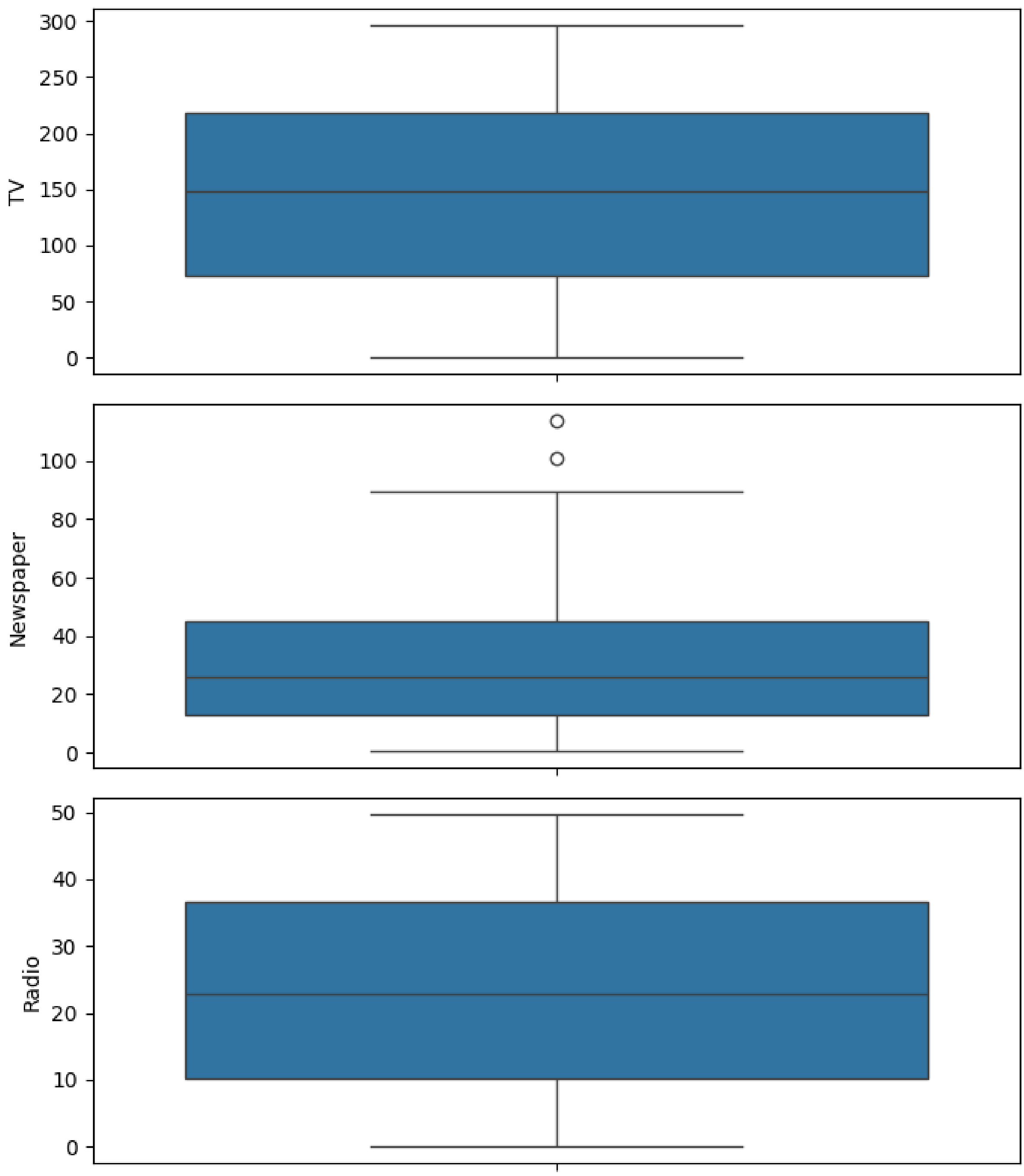
- Box plots provide a summary of a dataset's distribution, including the median, quartiles, and potential outliers.

- They are useful for identifying the spread and skewness of data.

Here is the code:

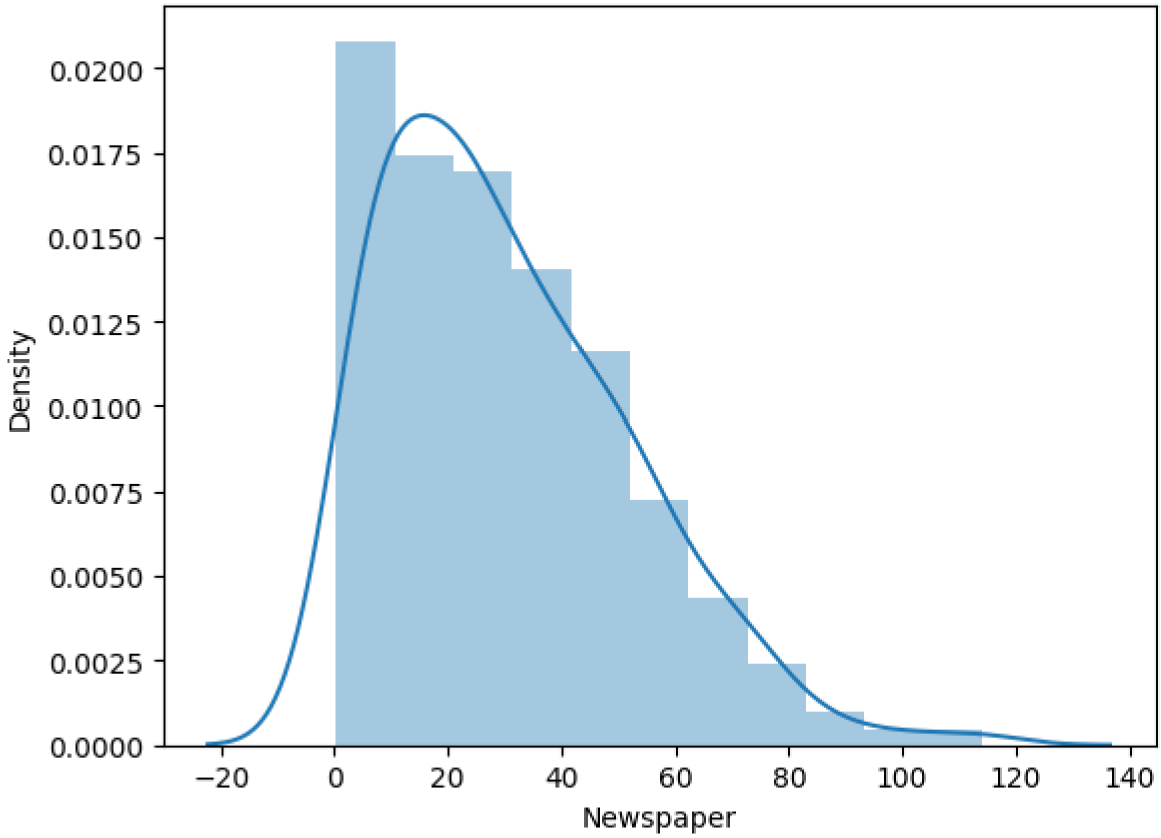
```
fig, axs=plt.subplots(3,figsize=(7,8))
plt1=sns.boxplot(data["TV"],ax=axs[0])
plt2=sns.boxplot(data["Newspaper"],ax=axs[1])
plt3=sns.boxplot(data["Radio"],ax=axs[2]
plt.tight_layout()
```

Output:




```
sns.distplot(data['Newspaper'])
<Axes: xlabel='Newspaper', ylabel='Density'>
```

Output



Finding the quantile:

```
iqr = data["Newspaper"].quantile(0.75) - data["Newspaper"].quantile(0.25)
lower_bridge=data['Newspaper'].quantile(0.25)-(iqr*1.5)

upper_bridge=data['Newspaper'].quantile(0.75)+(iqr*1.5)
lower_bridge
```

Output:

-35.46250000000006

```
upper_bridge
```

Output:

93.4375

```
data.loc[data['Newspaper']>=93,'Newspaper']=93
```

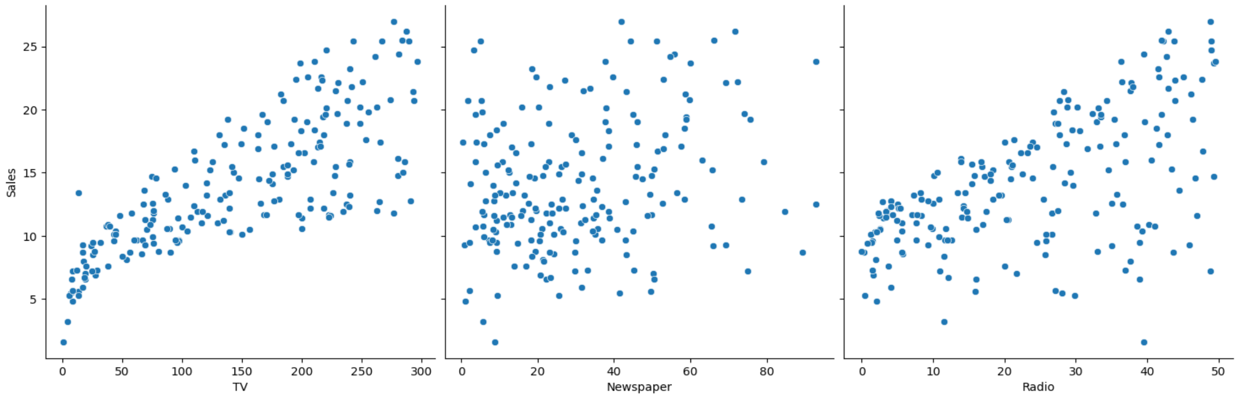
Pairplot()

- We can create a pair plot by calling the pairplot function and passing in your DataFrame.

- By default, it will create scatterplots for all pairs of numerical variables in the DataFrame and histograms along the diagonal for each variable.

```
sns.pairplot(data, x_vars=['TV', 'Newspaper', 'Radio'], y_vars='Sales',
height=5, aspect=1, kind='scatter')
plt.show()
```

Output:



heatmap():

- We use sns.heatmap() to create the heatmap. The annot=True argument adds annotations (numbers) to the cells, and cmap specifies the color map (you can choose from various color maps).

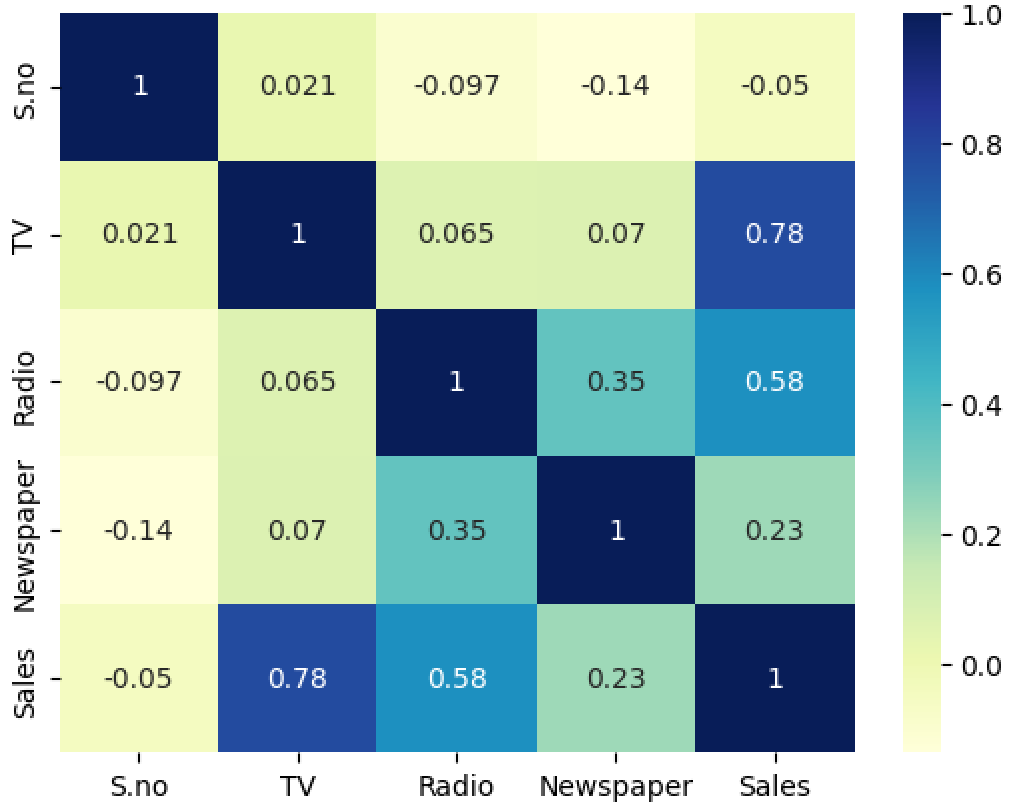
- We add labels to the X and Y axes using plt.xlabel() and plt.ylabel().

- We set a title for the heatmap using plt.title().

Here is the code:

```
sns.heatmap(data.corr(), cmap='YlGnBu', annot=True)
plt.show()
```

Output:




```
df=pd.read_csv(r"C:\Users\ridha\Downloads\sales.csv")
important_features = list(df.corr()['Sales'][(df.corr()['Sales']) >
0.5].index)
important_features
```

Output:

['TV', 'Radio', 'Sales']

```
x=df["TV"]
y=df["Sales"]
x=x.values.reshape(-1,1)
x
```

Output:

array([[230.1],

[44.5],

```
[ 17.2],
[151.5],
[180.8],
[  8.7],
[ 57.5],
[120.2],
[  8.6],
[199.8],
[ 66.1],
[214.7],
[ 23.8],
[ 97.5],
[204.1],
[195.4],
[ 67.8],
[281.4],
[ 69.2],
[147.3],
[218.4],
[237.4],
```

```
y
```

Output:

```
0     22.1
1     10.4
2      9.3
3     18.5
4     12.9
      ...
```

```
195     7.6
196     9.7
197    12.8
198    25.5
199    13.4
Name: Sales, Length: 200, dtype: float64
```

Conclusion:

The code which we executed above is the loading and pre processing of the dataset which is provided