IBM NAAN MUDHALVAN PROJECT SUBMISSION - PHASE_5

TECHONOLOGY: APPLIED DATA SCIENCE

TITLE: STOCK PRICE PREDICTION

STOCK PRICE PREDICTION

PROJECT OVERVIEW:

1. Studying data science: It means a huge amount of information. It includes lots and lots of different pieces of information, like records or other large sets of data.

2. Using Data bases: It is like a special tool that helps us to organize and manage all the data sets.

3. Finding Important Stuff: The data need to be completely checked out and finds only the important and useful bits.

4. Making a plan: A step by step plan should be made to work on this project.

5. Setting up Our Tools: Databases should be set to work efficiently.

6. Analysing and showing the results: All the data needs to be analysed carefully and it should be presented in such a way that it is easy to be understandable.

DESIGN THINKING:

1. Data Selection: We have chosen the programming language dataset which is at the current trend.

2. Database setup: we have to set up the dataset for storing and managing the dataset.

3. Data Exploration: We are in the middle of developing queries and scripts to explore the datasets, extract relevant information, and identify patterns.

4. Analysis Techniques: We are yet to apply appropriate analysis techniques, like statistical analysis or machine learning, to uncover insights.

5. Visualization: We will design a visualization chart to present the analysis results in an understandable and impactful manner.

6. Business Insights: We will interpret the analysis findings to derive valuable business intelligence and actionable recommendations.

Dataset is taken from kaggle competition and it can be downloaded from here:

https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocksdataset

Stock Price Prediction is the process of predicting what the demand for certain products will be in the future. This helps manufacturers to decide what they should produce and guides retailers toward what they should stock.

Stock predicting is the process of estimating future revenue by predicting how much of a product or service will sell in the next week, month, quarter, or year. At its simplest, a sales forecast is a projected measure of how a market will respond to a company's go-to-market efforts.

One of the most common methods used to predict stock is regression analysis. This method involves using historical sales data to train a model that can predict future sales.

The model can take into account factors such as past sales, marketing campaigns, and economic indicators to make its predictions.

Demand forecasting is aimed at improving the following process:

- Supplier relationship management
- Customer relationship management
- Order fulfillment and logistics
- Marketing campaigns
- Manufacturing flow management

DATA GATHERING:

Dataset is taken from kaggle competition and be downloaded from here:

https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocksdataset

DATA DESCRIPTION:

You are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. Note that the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge.

Machine Learning Algorithms:

Decision Trees and Random Forests: Useful for capturing complex relationships in the data.

Gradient Boosting Models (e.g., XG Boost, Light GBM): Excellent for predictive modeling and handling non-linear relationships.

Predicting future sales involves using data science techniques. A simple algorithmic approach could involve:

1. Data Collection:

Gather historical sales data, including timestamps.

Include relevant features like promotions, holidays, and other factors affecting sales.

2. Data Preprocessing:

Handle missing data and outliers.

Convert categorical variables into numerical representations.

Normalize or scale numerical features.

3. Feature Engineering:

Create new features, like moving averages or seasonality indicators.

Extract relevant information from timestamps, such as day of the week or month.

4. Model Selection:

Choose a regression model suitable for time-series data, like linear regression, decision trees, or more advanced models like ARIMA, SARIMA, or Prophet.

5. Training the Model:

Split data into training and validation sets.

Train the model on historical data.

6. Model Evaluation:

Evaluate the model's performance using metrics like Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) on the validation set.

7. Hyperparameter Tuning:

Fine-tune model parameters to improve performance.

8. Prediction:

Use the trained model to predict future sales based on new data.

9. Monitoring and Updating:

Regularly monitor the model's performance and update it as needed with new data.

ALGORITHMS:

1. List out the goods and services you sell.

2. Estimate how much of each you expect to sell.

3. Define the unit price or dollar value of each good or service sold.

4. Multiply the number sold by the price.

5. Determine how much it will cost to produce and sell each good or service

Importing Libraries:

 # EDA Libraries: import pandas as pd

import numpy as np import

matplotlib.colors as col from

mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import datetime from

pathlib import Path

import random **# Scikit-**

Learn models:

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error,

mean_absolute_error, r2_score

from sklearn.ensemble import RandomForestRegressor from

xgboost.sklearn import XGBRegressor

from sklearn.model_selection import KFold, cross_val_score, train_test_split

from sklearn.model_selection import KFold, cross_val_score, train_test_split **#**

LSTM:

import keras

from keras.layers import Dense from

keras.models import Sequential from

keras.callbacks import EarlyStopping

from keras.utils import np_utils from

keras.layers import LSTM **# ARIMA**

Model:

import statsmodels.tsa.api as smt import

statsmodels.api as sm

from statsmodels.tools.eval_measures import rmse

import pickle import warnings

2. Loading and Exploration of the Data

3. EDA (Exploratory Data Analysis)

4. Determining Time Series Stationary

5. Differencing

6. Scaling Data

7. Prediction Dataframe

Machine Learning Algorithms:

Decision Trees and Random Forests: Useful for capturing complex relationships in the data.

Gradient Boosting Models (e.g., XGBoost, LightGBM): Excellent for predictive modeling and handling non-linear relationships.

Predicting future sales involves using data science techniques. A simple algorithmic approach could involve:

Scikit-Research:

A broadly used python library that gives various feature selection, extraction, and preprocessing tools. It provides a steady API, making enforcing numerous feature engineering strategies easy. Its wide adoption guarentees tremendous community support and resources.

Applications: Handling missing values, transforming categorical variables using one-hot encoding, and standardizing features with scaling strategies.

Scikit-Learn Models: from sklearn.preprocessing

import MinMaxScaler from sklearn.linear_model

import LinearRegression

from sklearn.metrics importmean_squared_error,mean_absolute_error, r2_score

from sklearn.ensemble import RandomForestRegress

from xgboost.sklearn import XGBRegressor

from sklearn.model_selection import KFold, cross_val_score, train_test_split

Data set: we provide the screenshots of our data set which is

downloaded here:

https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocksdataset

	A	B	C	D	E	F	G
1	TV	Radio	Newspape	Sales			
2	230.1	37.8	69.2	22.1			
3	44.5	39.3	45.1	10.4			
4	17.2	45.9	69.3	12			
5	151.5	41.3	58.5	16.5			
6	180.8	10.8	58.4	17.9			
7	8.7	48.9	75	7.2			
8	57.5	32.8	23.5	11.8			
9	120.2	19.6	11.6	13.2			
10	8.6	2.1	1	4.8			
11	199.8	2.6	21.2	15.6			
12	66.1	5.8	24.2	12.6			
13	214.7	24	4	17.4			
14	23.8	35.1	65.9	9.2			
15	97.5	7.6	7.2	13.7			
16	204.1	32.9	46	19			
17	195.4	47.7	52.9	22.4			
18	67.8	36.6	114	12.5			
19	281.4	39.6	55.8	24.4			
20	69.2	20.5	18.3	11.3			
21	147.3	23.9	19.1	14.6			
22	218.4	27.7	53.4	18			
23	237.4	5.1	23.5	17.5			
24	13.2	15.9	49.6	5.6			
25	228.3	16.9	26.2	20.5			
26	62.3	12.6	18.3	9.7			
27	262.9	3.5	19.5	17			
28	142.9	29.3	12.6	15			
29	240.1	16.7	22.9	20.9			
30	248.8	27.1	22.9	18.9			
31	70.6	16	40.8	10.5			
32	292.9	28.3	43.2	21.4			
33	112.9	17.4	38.6	11.9			
34	97.2	1.5	30	13.2			
35	265.6	20	0.3	17.4			
36	95.7	1.4	7.4	11.9			
37	290.7	4.1	8.5	17.8			

38	266.9	43.8	5	25.4
39	74.7	49.4	45.7	14.7
40	43.1	26.7	35.1	10.1
41	228	37.7	32	21.5
42	202.5	22.3	31.6	16.6
43	177	33.4	38.7	17.1
44	293.6	27.7	1.8	20.7
45	206.9	8.4	26.4	17.9
46	25.1	25.7	43.3	8.5
47	175.1	22.5	31.5	16.1
48	89.7	9.9	35.7	10.6
49	239.9	41.5	18.5	23.2
50	227.2	15.8	49.9	19.8
51	66.9	11.7	36.8	9.7
52	199.8	3.1	34.6	16.4
53	100.4	9.6	3.6	10.7
54	216.4	41.7	39.6	22.6
55	182.6	46.2	58.7	21.2
56	262.7	28.8	15.9	20.2
57	198.9	49.4	60	23.7
58	7.3	28.1	41.4	5.5
59	136.2	19.2	16.6	13.2
60	210.8	49.6	37.7	23.8
61	210.7	29.5	9.3	18.4
62	53.5	2	21.4	8.1
63	261.3	42.7	54.7	24.2
64	239.3	15.5	27.3	20.7
65	102.7	29.6	8.4	14
66	131.1	42.8	28.9	16
67	69	9.3	0.9	11.3
68	31.5	24.6	2.2	11
69	139.3	14.5	10.2	13.4
70	237.4	27.5	11	18.9
71	216.8	43.9	27.2	22.3
72	199.1	30.6	38.7	18.3
73	109.8	14.3	31.7	12.4

74	26.8	33	19.3	8.8
75	129.4	5.7	31.3	11
76	213.4	24.6	13.1	17
77	16.9	43.7	89.4	8.7
78	27.5	1.6	20.7	6.9
79	120.5	28.5	14.2	14.2
80	5.4	29.9	9.4	5.3
81	116	7.7	23.1	11
82	76.4	26.7	22.3	11.8
83	239.8	4.1	36.9	17.3
84	75.3	20.3	32.5	11.3
85	68.4	44.5	35.6	13.6
86	213.5	43	33.8	21.7
87	193.2	18.4	65.7	20.2
88	76.3	27.5	16	12
89	110.7	40.6	63.2	16
90	88.3	25.5	73.4	12.9
91	109.8	47.8	51.4	16.7
92	134.3	4.9	9.3	14
93	28.6	1.5	33	7.3
94	217.7	33.5	59	19.4
95	250.9	36.5	72.3	22.2
96	107.4	14	10.9	11.5
97	163.3	31.6	52.9	16.9
98	197.6	3.5	5.9	16.7
99	184.9	21	22	20.5
100	289.7	42.3	51.2	25.4
101	135.2	41.7	45.9	17.2
102	222.4	4.3	49.8	16.7
103	296.4	36.3	100.9	23.8
104	280.2	10.1	21.4	19.8
105	187.9	17.2	17.9	19.7
106	238.2	34.3	5.3	20.7
107	137.9	46.4	59	15
108	25	11	29.7	7.2
109	90.4	0.3	23.2	12

110	13.1	0.4	25.6	5.3
111	255.4	26.9	5.5	19.8
112	225.8	8.2	56.5	18.4
113	241.7	38	23.2	21.8
114	175.7	15.4	2.4	17.1
115	209.6	20.6	10.7	20.9
116	78.2	46.8	34.5	14.6
117	75.1	35	52.7	12.6
118	139.2	14.3	25.6	12.2
119	76.4	0.8	14.8	9.4
120	125.7	36.9	79.2	15.9
121	19.4	16	22.3	6.6
122	141.3	26.8	46.2	15.5
123	18.8	21.7	50.4	7
124	224	2.4	15.6	16.6
125	123.1	34.6	12.4	15.2
126	229.5	32.3	74.2	19.7
127	87.2	11.8	25.9	10.6
128	7.8	38.9	50.6	6.6
129	80.2	0	9.2	11.9
130	220.3	49	3.2	24.7
131	59.6	12	43.1	9.7
132	0.7	39.6	8.7	1.6
133	265.2	2.9	43	17.7
134	8.4	27.2	2.1	5.7
135	219.8	33.5	45.1	19.6
136	36.9	38.6	65.6	10.8
137	48.3	47	8.5	11.6
138	25.6	39	9.3	9.5
139	273.7	28.9	59.7	20.8
140	43	25.9	20.5	9.6
141	184.9	43.9	1.7	20.7
142	73.4	17	12.9	10.9
143	193.7	35.4	75.6	19.2
144	220.5	33.2	37.9	20.1
145	104.6	5.7	34.4	10.4

146	96.2	14.8	38.9	12.3
147	140.3	1.9	9	10.3
148	240.1	7.3	8.7	18.2
149	243.2	49	44.3	25.4
150	38	40.3	11.9	10.9
151	44.7	25.8	20.6	10.1
152	280.7	13.9	37	16.1
153	121	8.4	48.7	11.6
154	197.6	23.3	14.2	16.6
155	171.3	39.7	37.7	16
156	187.8	21.1	9.5	20.6
157	4.1	11.6	5.7	3.2
158	93.9	43.5	50.5	15.3
159	149.8	1.3	24.3	10.1
160	11.7	36.9	45.2	7.3
161	131.7	18.4	34.6	12.9
162	172.5	18.1	30.7	16.4
163	85.7	35.8	49.3	13.3
164	188.4	18.1	25.6	19.9
165	163.5	36.8	7.4	18
166	117.2	14.7	5.4	11.9
167	234.5	3.4	84.8	16.9
168	17.9	37.6	21.6	8
169	206.8	5.2	19.4	17.2
170	215.4	23.6	57.6	17.1
171	284.3	10.6	6.4	20
172	50	11.6	18.4	8.4
173	164.5	20.9	47.4	17.5
174	19.6	20.1	17	7.6
175	168.4	7.1	12.8	16.7
176	222.4	3.4	13.1	16.5
177	276.9	48.9	41.8	27
178	248.4	30.2	20.3	20.2
179	170.2	7.8	35.2	16.7
180	276.7	2.3	23.7	16.8
181	165.6	10	17.6	17.6

182	156.6	2.6	8.3	15.5
183	218.5	5.4	27.4	17.2
184	56.2	5.7	29.7	8.7
185	287.6	43	71.8	26.2
186	253.8	21.3	30	17.6
187	205	45.1	19.6	22.6
188	139.5	2.1	26.6	10.3
189	191.1	28.7	18.2	17.3
190	286	13.9	3.7	20.9
191	18.7	12.1	23.4	6.7
192	39.5	41.1	5.8	10.8
193	75.5	10.8	6	11.9
194	17.2	4.1	31.6	5.9
195	166.8	42	3.6	19.6
196	149.7	35.6	6	17.3
197	38.2	3.7	13.8	7.6
198	94.2	4.9	8.1	14
199	177	9.3	6.4	14.8
200	283.6	42	66.2	25.5
201	232.1	8.6	8.7	18.4

We build the **Stock Price Prediction** model by loading and preprocessing the dataset and we load the historical sales dataset and preprocess the data for analysis.

In the analysis of the "STOCK PRICE PREDICTION" dataset, we conducted a comprehensive series of data analysis steps to create an accurate prediction model. The process began with Exploratory Data Analysis (EDA) to understand the dataset's characteristics. Subsequently, we performed data preprocessing, including outlier detection and handling using the winzoring technique, as well as data normalization using the min-max method. We then developed multiple models, including Linear Regression, Ridge Regression, Lasso Regression, Decision Tree, and Random Forest, all of which were evaluated through cross-validation. The model evaluation results revealed that Random Forest outperformed others, yielding an average Mean Squared Error (MSE) of 10.32%, Root Mean Squared Error (RMSE) of 8.09%, Mean

Absolute Error (MAE) of 5.99%, and an Rsquared value of 94.27%. Additionally, we conducted classic assumption tests, including tests for linearity, homoscedasticity, normality, multi collinearity, outliers, and independence, to ensure the validity of our model.These results provide in-depth insights into the quality of our prediction model and its relevance in the context of future sales forecasting.

Feature Engineering:

Feature engineering includes remodeling raw data into a format that successfully represents the underlying patterns within the data. It involves selecting, combining, and crafting attributes that capture the relationships between variables, enhancing the predictive power of machine learning models. These engineered features acts as the input for algorithms, using progressed performance and robustness.

Feature Tools:

A python library centered on automated feature engineering, particularly for timeseries and relational data. It automates producing new features by leveraging domain-specific knowledge and entity relationships.

Applications: Creating time-based features, aggregating data over different time intervals, and handling a couple of associated data tables.

Features explanation:

- **TV:** This feature represents the amount of advertising budget spent on television media for a product or service in a certain period, for example in thousands of dollars (USD).
- **Radio:** This feature represents the amount of advertising budget spent on radio media in the same period as TV.

- **Newspaper:** This feature represents the amount of advertising budget spent in newspapers or print media in the same period as TV and Radio.
- **Sales:** This feature represents product or service sales data in the same period as advertising expenditure on TV, Radio and Newspaper.

Load Data Input:

1. import pandas as pd df = pd.read_csv('/kaggle/input/stock price prediction/Sales.csv') df.head()

2. import pandas as pd

   df = pd.read_csv('/kaggle/input/stock price prediction/Sales.csv')

   df.shape()

3. import pandas as pd

   df = pd.read_csv('/kaggle/input/stock price prediction/Sales.csv')

   df.info()

4. import pandas as pd

   df = pd.read_csv('/kaggle/input/stock price prediction/Sales.csv')

   df.describe()

Exploratory Data Analysis (EDA) 5.

import plotly.express as px

figure = px.scatter(df, x='Sales', y='TV', size='TV', trendline='ols', title='Relationship Between Sales and TV Advertising')

figure.update_traces(marker=dict(line=dict(width=2, color='DarkSlateGrey')),

selector=dict(mode='markers')) figure.update_layout(xaxis_title='Sales',

yaxis_title='TV Advertising', legend_title='TV Ad Size', plot_bgcolor='white'

)

figure.show()

6. figure= px.scatter(df, x='Sales', y='Newspaper', size='Newspaper', trendline='ols', title='Relationship Between Sales and Newspaper Advertising')

figure.update_traces(marker=dict(line=dict(width=2,
color='DarkSlateGrey')), selector=dict(mode='markers'))

figure.update_layout(xaxis_title='Sales',

yaxis_title='Newspaper Advertising',

legend_title='Newspaper Ad Size',

plot_bgcolor='white'

)

figure.show()

7. px.scatter(df, x='Sales', y='Radio', size='Radio', trendline='ols', title='Relationship Between Sales and Radio Advertising')

```
figure.update_traces(marker=dict(line=dict(width=2,       color='DarkSlateGrey')),
selector=dict(mode='markers')) figure.update_layout(     xaxis_title='Sales',
    yaxis_title='Radio Advertising',
legend_title='Radio Ad Size',
plot_bgcolor='white'
)
figure.show()
```

8. ***Calculate the correlation*** correlation = df.corr()

```
sales_correlation = correlation["Sales"].sort_values(ascending=False)

# Format and style the correlation values
styled_sales_correlation = sales_correlation.apply(lambda x: f'{x:.2f}')

styled_sales_correlation = styled_sales_correlation.reset_index()

styled_sales_correlation.columns = ["Feature", "Correlation with Sales"]

styled_sales_correlation.style.background_gradient(cmap='coolwarm', axis=0)
```

Data Preprocessing

9. import seaborn as sns

```
import matplotlib.pyplot as plt

# Create the box plot
plt.figure(figsize=(8, 6))

sns.boxplot(x='TV', data=df, palette='Blues')

plt.title('Box Plot of TV Advertising')
```

```python
plt.xlabel('TV Advertising Spending')

plt.grid(axis='x', linestyle='--', alpha=0.6)


# Show the plot plt.show()
```

10. ***# Create the box plot***

```python
plt.figure(figsize=(8, 6))

sns.boxplot(x='Radio', data=df, palette='Oranges')

plt.title('Box Plot of Radio Advertising')

plt.xlabel('Radio Advertising Spending')

plt.grid(axis='x', linestyle='--', alpha=0.6)


# Show the plot
plt.show()
```

11. ***# Create the box plot***

```python
plt.figure(figsize=(8, 6))

sns.boxplot(x='Newspaper', data=df, palette='YlGnBu')

plt.title('Box Plot of Newspaper Advertising')

plt.xlabel('Newspaper Advertising Spending')

plt.grid(axis='x', linestyle='--', alpha=0.6)

# Show the plot
plt.show()
```

There are outliers in the Newspaper feature. To overcome this, we use the Winsorizing technique. Winsorizing is a technique that replaces outlier values with certain predetermined threshold values. We set the threshold value for the Newspaper feature at 2.

12.

```
import numpy as np

er_threshold = 2 * np.std(df['Newspaper']) +
np.mean(df['Newspaper'])

df[ an Winsorizing pada kolom 'Newspaper'
Newspaper'
f[ ] = np.where(df['Newspaper'] > upper_threshold,
'Newspaper' upper_threshold, d

13. ])

plt. igure(figsize e the box plot

sns. oxplot(x =(8, 6))

plt. tle( ='Newspaper', data=df, palette='YlGnBu')

plt. label( Plot of Newspaper Advertising')

Newspaper Advertising Spending')
plt.grid(axis='x', linestyle='--', alpha=0.6)

# Show the plot
plt.show()
```

Data normalization

At this stage, we use the min-max technique. Min-Max is a data preprocessing technique used in data analysis and machine learning to convert values in a dataset into a certain range, usually between 0 and 1.

```
14. from sklearn.preprocessing import MinMaxScaler
```

Creating a MinMaxScaler object:

```
scaler = MinMaxScaler()

# Columns to be normalized (e.g., TV, Radio, Newspaper)

columns_to_normalize = ['TV', 'Radio', 'Newspaper']

# Apply Min-Max normalization to the selected columns

df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])

df.head()
```

Modelling and Evaluation

At the modeling stage, we use 5 algorithms for comparison, namely Linear Regression, Ridge Regression, Lasso Regression, Decision Tree, and Random Forest. And for evaluation using MSE, RMSE, MAE and R-Squared.

15. X = df[['TV', 'Radio', 'Newspaper']]

y = df['Sales']

16. from sklearn.model_selection import cross_val_score num_folds = 5

Cross validation function is performed and the metrics were calculated in percentage

```
def perform_cross_validation(model, X, y, num_folds):

    mse_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_mean_squared_error')

    rmse_scores = np.sqrt(mse_scores)

    mae_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_mean_absolute_error')

    r2_scores = cross_val_score(model, X, y, cv=num_folds, scoring='r2')

    return mse_scores, rmse_scores, mae_scores, r2_scores
```

17. from sklearn.linear_model import LinearRegression, Ridge, Lasso

 # Linear Regression

```
linear_model = LinearRegression()

linear_mse, linear_rmse, linear_mae, linear_r2 = perform_cross_validation(linear_model, X, y, num_folds)
```

```
print("Linear Regression:")

print(f"Average MSE: {np.mean(linear_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(linear_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(linear_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(linear_r2) * 100:.2f}%")

print("\n")
```

18. ***# Ridge Regression*** ridge_model = Ridge(alpha=1.0) *# You can adjust alpha as needed*

```
ridge_mse, ridge_rmse, ridge_mae, ridge_r2 = perform_cross_validation(ridge_model, X, y, num_folds)

print("Ridge Regression:")

print(f"Average MSE: {np.mean(ridge_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(ridge_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(ridge_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(ridge_r2) * 100:.2f}%")

print("\n")
```

19. ***# Lasso Regression*** lasso_model = Lasso(alpha=1.0) *# You can adjust alpha as needed*

```
lasso_mse, lasso_rmse, lasso_mae, lasso_r2 = perform_cross_validation(lasso_model, X, y, num_folds)
```

```
print("Lasso Regression:")

print(f"Average MSE: {np.mean(lasso_mse) / np.mean(y) * 100:.2f}%")
```

```
print(f"Average RMSE: {np.mean(lasso_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(lasso_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(lasso_r2) * 100:.2f}%")

print("\n")
```

20. from sklearn.tree import DecisionTreeRegressor

```
# Decision Trees
tree_model = DecisionTreeRegressor(max_depth=None, random_state=0)

tree_mse, tree_rmse, tree_mae, tree_r2 = perform_cross_validation(tree_model, X,
y, num_folds)

print("Decision Trees:")

print(f"Average MSE: {np.mean(tree_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(tree_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(tree_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(tree_r2) * 100:.2f}%")

print("\n")
```

21. from sklearn.ensemble import RandomForestRegressor

```
# Random Forest
forest_model = RandomForestRegressor(n_estimators=100, random_state=0)

forest_mse, forest_rmse, forest_mae, forest_r2 = perform_cross_validation(forest_
model, X, y, num_folds)
```

```
print("Random Forest:")

print(f"Average MSE: {np.mean(forest_mse) / np.mean(y) * 100:.2f}%")

print(f"Average RMSE: {np.mean(forest_rmse) / np.mean(y) * 100:.2f}%")

print(f"Average MAE: {np.mean(forest_mae) / np.mean(y) * 100:.2f}%")

print(f"Average R-squared: {np.mean(forest_r2) * 100:.2f}%")
```

Classic assumption test

At the classical assumption testing stage, 5 assumption tests are used, namely linearity test, homoscedasticity test, normality test, multicollinearity test, outliers test, and independent test.

22.

```
import statsmodels.api as sm

import statsmodels.stats.api as sms

# Adding a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the regression model model
= sm.OLS(y, X).fit()

# Residuals (model residuals)
residuals = model.resid
```

23. *Linearity is checked by using Residual vs Fitted values plot*

```
import matplotlib.pyplot as plt
```

```
plt.scatter(model.fittedvalues, residuals)

plt.xlabel("Fitted Values")

plt.ylabel("Residuals")

plt.title("Linearity Check")

plt.show()
```

24. *# Assumption 2: Homoskedasticity*

```
# You can check homoskedasticity using Breusch-Pagan test _,
p_homo, _, _ = sms.het_breuschpagan(residuals, X)

print(f"Homoskedasticity (Breusch-Pagan): p-value = {p_homo:.4f}")
```

25. *# Assumption 3: Independence (Serial Correlation)*

```
# You can check for serial correlation using Durbin-Watson test
from statsmodels.stats.stattools import durbin_watson

dw_stat = durbin_watson(residuals)

print(f"Serial Correlation (Durbin-Watson): DW Statistic = {dw_stat:.2f}")
```

26. *# Assumption 4: Normality*

```
# You can check normality using a normal probability plot (Q-Q plot) import
scipy.stats as stats

fig, ax = plt.subplots(figsize=(6, 4))
_, (__, ___, r) = stats.probplot(residuals, plot=ax, fit=True)

ax.get_lines()[0].set_markerfacecolor('C0')

ax.get_lines()[0].set_markersize(5.0)
```

```
ax.get_lines()[1].set_linewidth(3.0)

plt.title("Normal Probability Plot")

plt.show()
```

27. *# Assumption 5: Multicollinearity*

```
# You can check multicollinearity using the Variance Inflation Factor (VIF)

from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()

vif["Features"] = X.columns

vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print("Multicollinearity (VIF):")

print(vif)
```

28. *# Assumption 6: Outliers*

```
# You can check for outliers using the studentized residuals and the Cook's Distance

student_resid = model.get_influence().resid_studentized_internal

cooks_d = model.get_influence().cooks_distance[0]

outliers = pd.DataFrame({'Studentized Residuals': student_resid, "Cook's Distance": cooks_d})

outliers.index = X.index

print("Outliers:")

print(outliers[outliers['Studentized Residuals'].abs() > 2])   # You can adjust the threshold as needed
```

```
Out[1]:
```

	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

Features explanation:

- **TV**: this feature represents the amount of advertising budget spent on television media for a product or service in a certain period, for example in thousands of dollars (USD).
- **Radio**: this feature represents the amount of advertising budget spent on radio media in th

Newspaper.

In [2]:

```
df.shape
```

Out[2]:

```
(200, 4)
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float6
4
 1   Radio      200 non-null    float6
4
 2   Newspaper  200 non-null    float
4
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float6
4
 1   Radio      200 non-null    float6
4
 2   Newspaper  200 non-null    float6
4
 3   Sales      200 non-null    float6
4
dtypes: float64(4)
memory usage: 6.4 KB
```

In [4]:

```
df.describe().T
```

Out[4]: