**FUTURE SALES PREDICTION - PHASE 3**

***Introduction:***

In the world of business, the ability to predict future sales is of paramount importance. Accurate sales forecasts empower companies to make informed decisions, allocate resources efficiently, and stay ahead in the competitive landscape. One powerful tool at the disposal of businesses for this purpose is time series forecasting. Time series forecasting is a specialized field within data analysis that focuses on understanding and predicting future values of a variable based on its historical performance over time.

***CODE:***

import pandas as pd

df = pd.read\_csv('/kaggle/input/future-sales-prediction/Sales.csv')

df.head()

**output:**

*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*

df.shape

**output:**

(200, 4)

df.info()

**output:**

<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 float64

1 Radio 200 non-null float64

2 Newspaper 200 non-null float64

3 Sales 200 non-null float64

dtypes: float64(4)

memory usage: 6.4 KB

df.describe() . T

**output:**

count mean std min 25% 50% 75% max

TV 200.0 147.0425 85.854236 0.7 74.375 149.75 218.825 296.4

Radio 200.0 23.2640 14.846809 0.0 9.975 22.90 36.525 49.6

Newspaper 200.0 30.5540 21.778621 0.3 12.750 25.75 45.100 114.0

Sales 200.0 15.1305 5.283892 1.6 11.000 16.00 19.050 27.0

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()

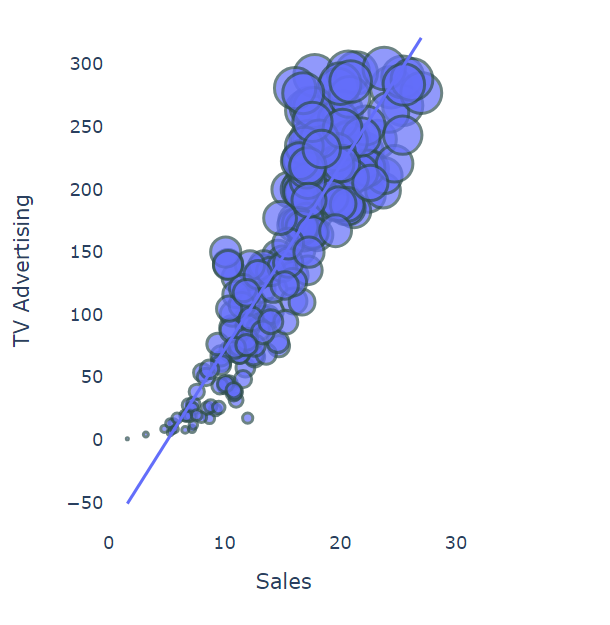


figure = px.scatter(df, x='Sales', y='Newspaper', size='Newspaper', trendline='ols', title='Relationship Between Sales and Newspaper Advertising')

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()

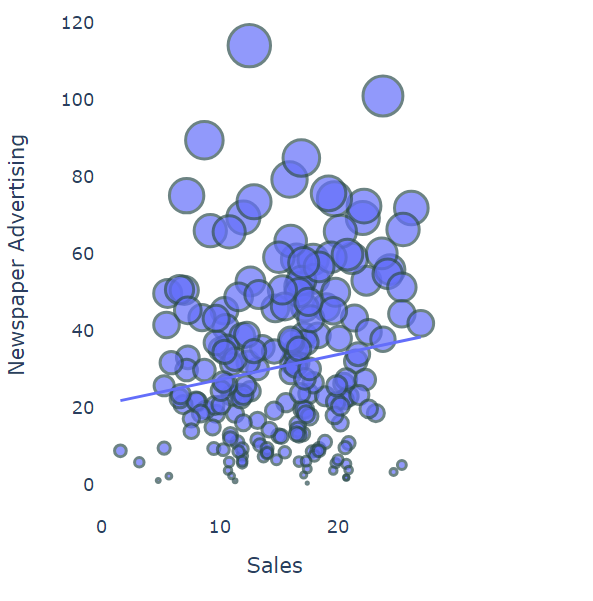


figure = px.scatter(df, x='Sales', y='Radio', size='Radio', trendline='ols', title='Relationship Between Sales and Radio Advertising')

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

figure.update\_layout(

xaxis\_title='Sales',

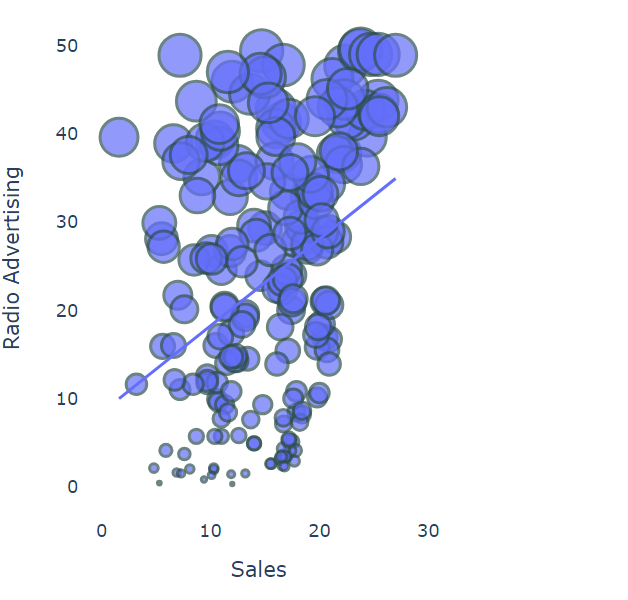
yaxis\_title='Radio Advertising',

legend\_title='Radio Ad Size',

plot\_bgcolor='white'

)

figure.show()



# 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)

**output:**

Feature Correlation with Sales

0 Sales 1.00

1 TV 0.90

2 Radio 0.35

3 Newspaper 0.16

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')

plt.xlabel('TV Advertising Spending')

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

# Show the plot

plt.show()

A graph with a blue rectangle

Description automatically generated

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')

plt.xlabel('TV Advertising Spending')

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

# Show the plot

plt.show()

A graph with a rectangular object

Description automatically generated with medium confidence

# Create the box plot

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()

A graph with a blue rectangle

Description automatically generated

import numpy as np

# Ambang batas atas (threshold) untuk Winsorizing

upper\_threshold = 2 \* np.std(df['Newspaper']) + np.mean(df['Newspaper'])

# Menerapkan Winsorizing pada kolom 'Newspaper'

df['Newspaper'] = np.where(df['Newspaper'] > upper\_threshold, upper\_threshold, df['Newspaper'])

# Create the box plot

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

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

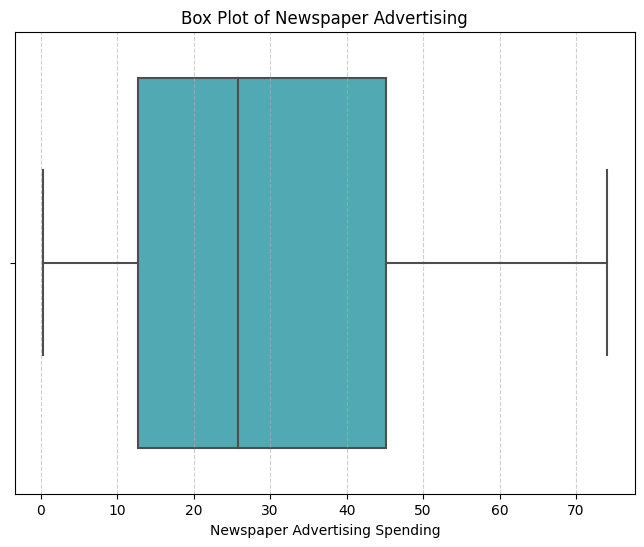
plt.title('Box Plot of Newspaper Advertising')

plt.xlabel('Newspaper Advertising Spending')

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

# Show the plot

plt.show()



from sklearn.preprocessing import MinMaxScaler

# Create 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()

**output:**

TV Radio Newspaper Sales

0 0.775786 0.762097 0.934843 22.1

1 0.148123 0.792339 0.607851 10.4

2 0.055800 0.925403 0.936200 12.0

3 0.509976 0.832661 0.789664 16.5

4 0.609063 0.217742 0.788307 17.9

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

y = df['Sales']

from sklearn.model\_selection import cross\_val\_score

# Performing 5-fold cross-validation (can be adjusted to the desired number of folds)

num\_folds = 5

# Function to perform cross-validation and calculate metrics 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

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")

**output:**

Linear Regression:

Average MSE: 18.90%

Average RMSE: 11.01%

Average MAE: 8.38%

Average R-squared: 89.53%

# 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")

**output:**

Ridge Regression:

Average MSE: 19.67%

Average RMSE: 11.20%

Average MAE: 8.54%

Average R-squared: 89.19%

# 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")

**output:**

Lasso Regression:

Average MSE: 115.55%

Average RMSE: 27.51%

Average MAE: 22.39%

Average R-squared: 35.98%

from sklearn.tree import DecisionTreeRegressor

# Decision Trees

tree\_model = DecisionTreeRegressor(max\_depth=None, random\_state=0) # You can adjust parameters as needed

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")

**output:**

Decision Trees:

Average MSE: 16.73%

Average RMSE: 10.40%

Average MAE: 7.56%

Average R-squared: 90.65%

from sklearn.ensemble import RandomForestRegressor

# Random Forest

forest\_model = RandomForestRegressor(n\_estimators=100, random\_state=0) # You can adjust parameters as needed

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}%")

**output:**

Random Forest:

Average MSE: 10.32%

Average RMSE: 8.09%

Average MAE: 5.99%

Average R-squared: 94.27%

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

#Assumption 1: Linearity

# You can check linearity 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()

**output:**

A graph with blue dots

Description automatically generated

# 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}")

**output**:

Homoskedasticity (Breusch-Pagan): p-value = 0.2634

# 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}")

**OUTPUT:**

Serial Correlation (Durbin-Watson): DW Statistic = 2.25

# 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)

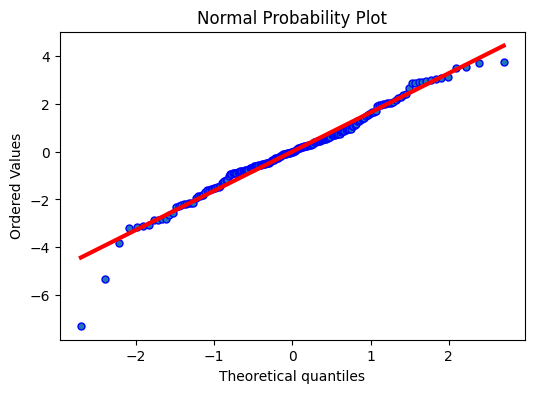
ax.get\_lines()[0].set\_markerfacecolor('C0')

ax.get\_lines()[0].set\_markersize(5.0)

ax.get\_lines()[1].set\_linewidth(3.0)

plt.title("Normal Probability Plot")

plt.show()



# 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)

**OUTPUT:**

Multicollinearity (VIF):

Features VIF

0 const 6.898975

1 TV 1.005037

2 Radio 1.150018

3 Newspaper 1.150920

# Assumption 6: Outliers

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])

**OUTPUT:**

Outliers:

Studentized Residuals Cook's Distance

10 2.272004 0.021004

33 -2.322006 0.037363

97 2.148943 0.007995

130 -4.468814 0.195094

150 -3.233182 0.056641

154 2.120557 0.013399

196 2.261963 0.021244

**Conclusion:**

Time series forecasting is a powerful tool for businesses seeking to predict future sales. By analyzing historical sales data and understanding the underlying patterns, it enables organizations to make data-driven decisions, allocate resources efficiently, and maintain a competitive edge in their respective industries. In this rapidly changing business landscape, mastering the art of sales prediction through time series forecasting is essential for sustainable growth and success.