**TITLE**

**FUTURE SALES PREDICTION**

# DEVELOPMENT PART 2

**Import Necessary Libraries:**

Import the required libraries for deep learning, data preprocessing, and model evaluation

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

**Load the Dataset:**

import pandas as pd  
  
df = pd.read\_csv(r"C:)\Sales.csv")  
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

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.

df.shape

(200, 4)

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 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

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

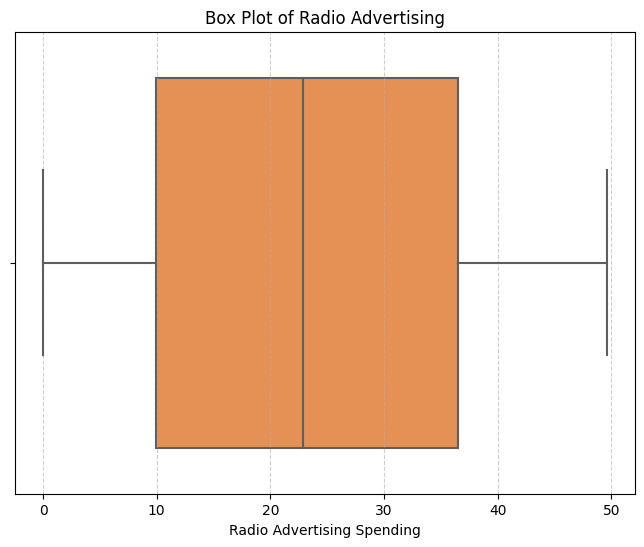
# Calculate the correlation  
correlation = df.corr()  
sales\_correlation = correlation["Sales"].sort\_values(ascending=False)  
  
# Format and style the correlation values  
styled\_sales\_correlation = sales\_correlation.apply(lambda x: f'{x:.2f}')  
styled\_sales\_correlation = styled\_sales\_correlation.reset\_index()  
styled\_sales\_correlation.columns = ["Feature", "Correlation with Sales"]  
styled\_sales\_correlation.style.background\_gradient(cmap='coolwarm', axis=0)

<pandas.io.formats.style.Styler at 0x23785318950>

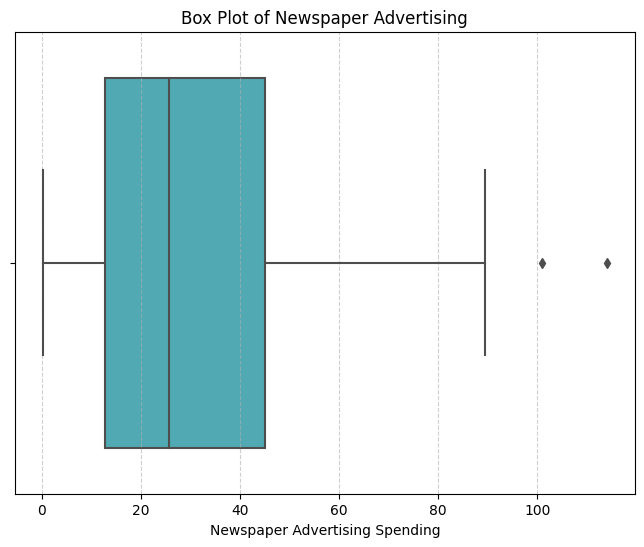
# Data Preprocessing:

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()

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()

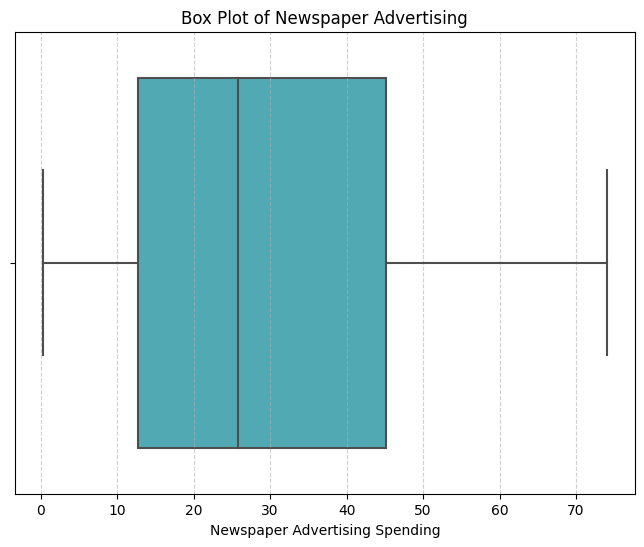


# Create the box plot  
plt.figure(figsize=(8, 6))  
sns.boxplot(x='Newspaper', data=df, palette='YlGnBu')  
plt.title('Box Plot of Newspaper Advertising')  
plt.xlabel('Newspaper Advertising Spending')  
plt.grid(axis='x', linestyle='--', alpha=0.6)  
  
# Show the plot  
plt.show()



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()



#### 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.

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()

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

# 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.

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

from sklearn.model\_selection import cross\_val\_score  
  
num\_folds = 5  
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

# Linear Regression:

In machine learning, a method called linear regression is used to predict a continuous outcome variable, also known as a dependent variable, from one or more predictor factors, also known as independent variables. It makes the assumption that the inputs and the output have a linear relationship. The best-fit line that minimizes the variation between the expected and actual values is the one to be found. For jobs like price forecasting, trend analysis, and sales forecasting, this method is frequently employed.

from sklearn.linear\_model import LinearRegression, Ridge, Lasso  
linear\_model = LinearRegression()

# Ridge Regression:

In machine learning, ridge regression is a regularization approach that keeps linear regression models from overfitting. In order to discourage large coefficients for the input features, it adds a penalty term (L2 regularization) to the linear regression cost function. This facilitates the handling of multicollinearity and yields a more accurate and stable model, particularly in the case of high dimensionality datasets. Ridge regression is a crucial component in enhancing the generalization and performance of models on intricate datasets.

ridge\_model = Ridge(alpha=1.0)  
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")

Ridge Regression:  
Average MSE: 19.67%  
Average RMSE: 11.20%  
Average MAE: 8.54%  
Average R-squared: 89.19%

# Lasso Regression:

A machine learning method called lasso regression is employed for regularization as well as variable selection. It encourages sparsity in the coefficient values by adding a penalty term (L1 regularization) to the linear regression cost function, much like ridge regression. Lasso works to reduce overfitting by setting the coefficients of less significant features to zero, which is especially helpful in high-dimensional datasets. It is useful for selecting features and creating more straightforward, understandable models when there are a lot of input variables.

lasso\_model = Lasso(alpha=1.0)   
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("\n")

Lasso Regression:  
Average MSE: 115.55%  
Average RMSE: 27.51%  
Average MAE: 22.39%  
Average R-squared: 35.98%

# Decision Tree:

In machine learning, a decision tree is a predictive model that associates features with results. It divides the data recursively according to features, forming a structure like a tree, with each internal node denoting a choice made in response to a feature and each leaf node denoting the expected result. Decision trees are useful tools in machine learning for decision-making processes because they are adaptable, comprehensible, and frequently utilized for classification and regression problems.

from sklearn.tree import DecisionTreeRegressor  
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")

Decision Trees:  
Average MSE: 16.73%  
Average RMSE: 10.40%  
Average MAE: 7.56%  
Average R-squared: 90.65%

# Random Forest:

A machine learning ensemble learning technique called Random Forest uses several decision trees. By training each tree on a different subset of the data and features, it creates a forest of trees. When making predictions, it either uses a majority vote (for classification) or averages the outcomes from individual trees (for regression). For ML applications like feature selection, regression, and classification, Random Forests are frequently utilized due to their robustness and ability to manage intricate relationships in data.

from sklearn.ensemble import RandomForestRegressor  
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}%")

Random Forest:  
Average MSE: 10.32%  
Average RMSE: 8.09%  
Average MAE: 5.99%  
Average R-squared: 94.27%