**Phase 3: Development Part 1**



**Name: P.Kaviyalaxmi**

**Register Number : 312621243017**

**College Name : Thangavelu Engineering College**

**Project 3 : Future Sales Prediction**

**Project 3: Future Sales Prediction**

**Objective:**

The objective is to create a tool that enables the company to optimize inventory management and make informed business decisions based on datadriven sales predictions In this part we understand the problem statement and we created a document on what have we understood and we proceeded ahead with solving the problem. The problem is to develop a predictive model that uses historical sales data to forecast future sales for a retail company.

**Problem Definition:**

The problem is to develop a predictive model that uses historical sales data to forecast future sales for a retail company. This project involves data preprocessing, feature engineering, model selection, training, and evaluation. This model will predict sales on a certain day after being provided with a certain set of inputs.

**Code and Explanation :**

Utilize a dataset containing historical sales data. Here a csv file is converted to a DataFrame and the pandas object is used. The This code will create a DataFrame using the provided data and column names. Remember to replace the placeholder data with your actual dataset.

This dataset seems to be related to advertising expenditures and their impact on sales. Here are the column meanings:

TV: Advertising budget spent on TV ads.

Radio: Advertising budget spent on radio ads.

Newspaper: Advertising budget spent on newspaper ads.

Sales: Sales generated as a result of the advertising campaign.

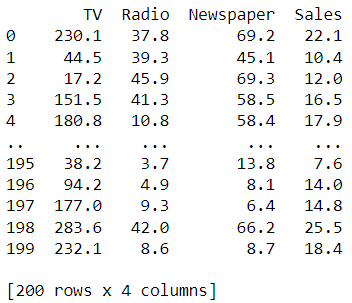
Here's how you can implement this in Python using pandas:

#Data Source utilize the dataset

import pandas as pd

df=pd.read\_csv(r'Sales.csv')

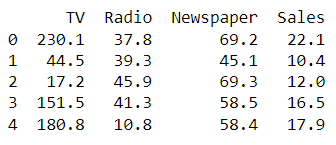
print(df)



You can use df. head() to get the first N rows in Pandas DataFrame.

# Print the first few rows of the DataFrame

print(df.head())

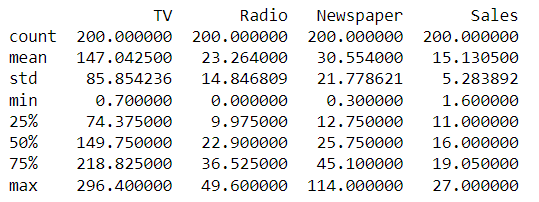


We calculate and print the summary statistics of the dataset using df.describe() function . The describe() method returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column such as count , mean , std , min , 25% , 50% , 75% , max .

# Summary statistics

summary\_stats = df.describe()

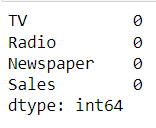
print(summary\_stats)



You can use the isnull() or isna() method of pandas. DataFrame and Series to check if each element is a missing value or not. isnull() is an alias for isna() , and both are used interchangeably.value.

#to check any missing values

print(df.isnull().sum())



The fillna() method replaces the NULL values with a specified value. The fillna() method returns a new DataFrame object unless the inplace parameter is set to True , in that case the fillna() method does the replacing in the original DataFrame instead.

#if missing values are their then use this code

df.fillna(df.mean(), inplace=True)

The drop\_duplicates() method removes duplicate rows. Use the subset parameter if only some specified columns should be considered when looking for duplicates.

#to remove duplicate values

df = df.drop\_duplicates()

Label encoding is a technique used in machine learning and data analysis to convert categorical variables into numerical format. It is particularly useful when working with algorithms that require numerical input, as most machine learning models can only operate on numerical data.

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler, LabelEncoder

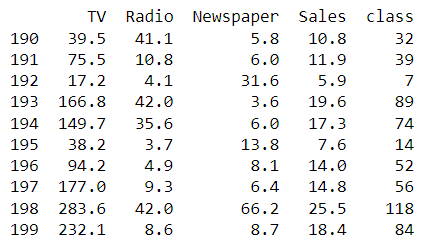
from sklearn.preprocessing import StandardScaler

#to convert categorical variables into numerical format

labelencoder = LabelEncoder()

df['class']=labelencoder.fit\_transform(df['Sales'])

print(df.tail(10))

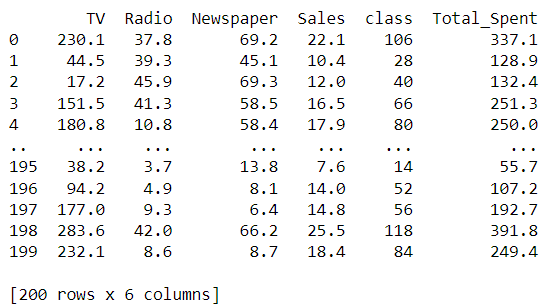


Feature engineering involves a set of techniques that enable us to create new features by combining or transforming the existing ones. These techniques help to highlight the most important patterns and relationships in the data.Here the Total spent is added as a feature using the datas of TV , Radio , Newspaper in the data set .

#adding the new feature as new column

df['Total\_Spent'] = df['TV'] + df['Radio'] + df['Newspaper']

print(df)



Feature Scaling or Standardization: It is a step of Data Pre Processing that is applied to independent variables or features of data. It helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm .

# Scaling features (optional, but can be important for some models)

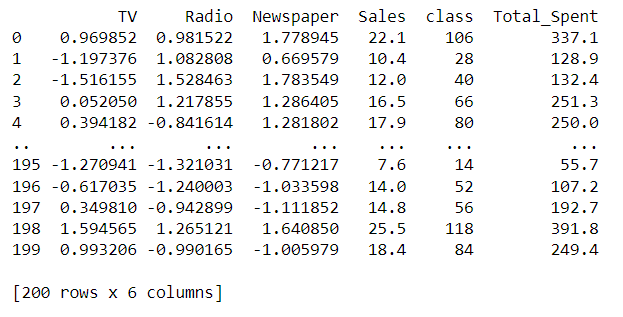
scaler = StandardScaler()

df[['TV', 'Radio', 'Newspaper']] = scaler.fit\_transform(df[['TV', 'Radio',

'Newspaper']])

# Now, the data is preprocessed and ready for modelling

print(df)

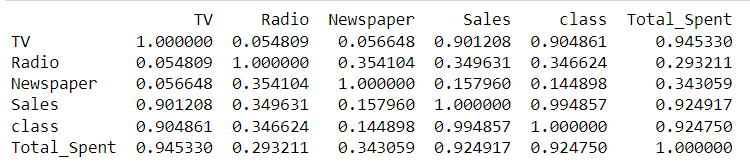


A correlation matrix is a table containing correlation coefficients for many variables. Each cell in the table represents the correlation between two variables. The value might range between -1 and 1.

# Correlation matrix

correlation\_matrix = df.corr()

print(correlation\_matrix)



Model selection is the process of selecting one final machine learning model from among a collection of candidate machine learning models for a training dataset. Here the ARIMA model is selected . An autoregressive integrated moving average, or ARIMA, is a statistical analysis model that uses time series data to either better understand the data set or to predict future trends . Autoregressive Integrated Moving Average (ARIMA ) is a commonly-used local statistical algorithm for time-series forecasting .

#import libraries

from statsmodels.tsa.arima.model import ARIMA

from itertools import product

import itertools

p = 1 # Example value

d = 1 # Example value

q = 1 # Example value

# Create the ARIMA model

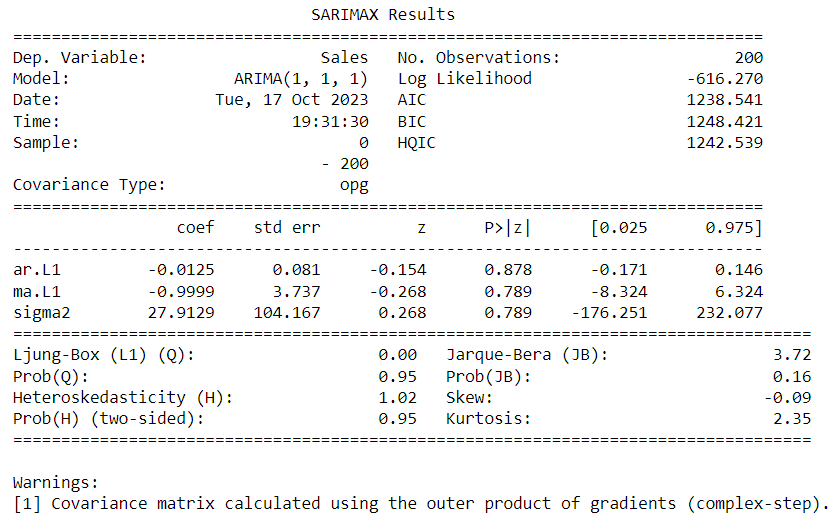
model = ARIMA(y, order=(p, d, q))

# Fit the model to the data

model\_fit = model.fit()

# Summary of the model

print(model\_fit.summary())



Train/Test is a method to measure the accuracy of your model. It is called Train/Test because you split the data set into two sets: a training set and a testing set. 80% for training, and 20% for testing. You train the model using the training set. You test the model using the testing set.

# Extract 'Sales' column as a time series

sales\_ts = df['Sales']

# Train an ARIMA model

order = (1, 1, 1) # ARIMA(p,d,q) parameters (you may need to tune these)

model = ARIMA(sales\_ts, order=order)

results = model.fit()

# Print model summary

print(results.summary())

# Optionally, you can make forecasts with the trained model

# Number of steps to forecast

forecast\_steps = 10

forecast = results.get\_forecast(steps=forecast\_steps)

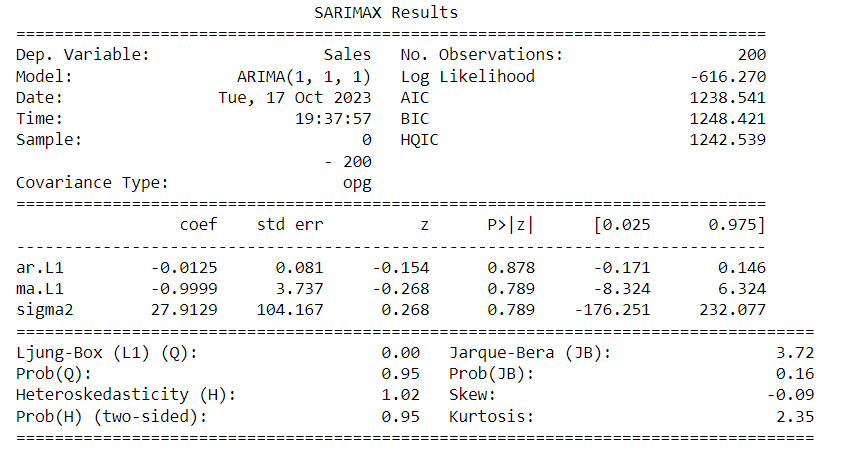
forecast\_mean = forecast.predicted\_mean

forecast\_ci = forecast.conf\_int()

# Print the forecasts

print(forecast\_mean)

print(forecast\_ci)



Currently, the most popular metrics for evaluating time series forecasting models are MAE, RMSE and AIC. To briefly summarize, both MAE and RMSE measures the magnitude of errors in a set of predictions. The major difference between MAE and RMSE is the impact of the large errors.

**MAE:** absolute error refers to the magnitude of difference between the prediction of an observation and the true value of that observation. MAE takes the average of absolute errors for a group of predictions and observations as a measurement of the magnitude of errors for the entire group.

**MSE:** Mean Squared Error (MSE) measures the amount of error in a statistical model. Evaluate the mean squared difference between observed and predicted values. If the model has no errors, the MSE is zero. Its value increases as the model error increases.

**RMSE:** In machine learning, it is extremely helpful to have a single number to judge a model's performance, whether it be during training, cross-validation, or monitoring after deployment. Root mean square error is one of the most widely used measures for this.

# Make predictions on the test set

predictions = model\_fit.forecast(len(test))

# Calculate MAE, MSE, RMSE

mae = mean\_absolute\_error(test, predictions)

mse = mean\_squared\_error(test, predictions)

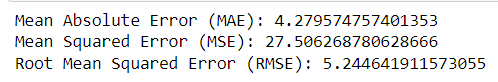
rmse = math.sqrt(mse)

#Print the values

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Mean Squared Error (MSE): {mse}')

print(f'Root Mean Squared Error (RMSE): {rmse}')



A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

import matplotlib.pyplot as plt

# Scatter plot between 'TV' and 'Sales'

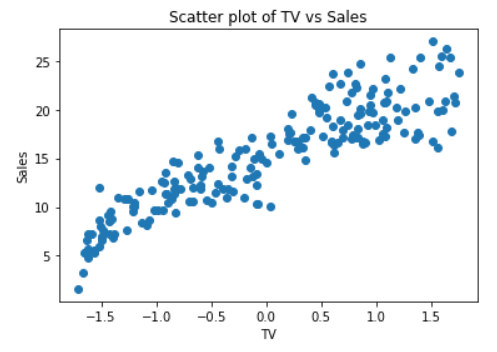
plt.scatter(df['TV'], df['Sales'])

plt.xlabel('TV')

plt.ylabel('Sales')

plt.title('Scatter plot of TV vs Sales')

plt.show()



A histogram is a graph that shows the frequency of numerical data using rectangles. The height of a rectangle (the vertical axis) represents the distribution frequency of a variable (the amount, or how often that variable appears).

# Histogram of 'Sales'

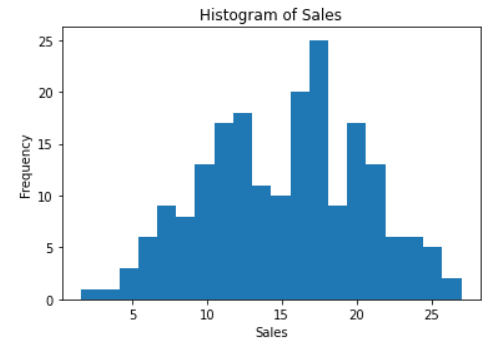
plt.hist(df['Sales'], bins=20)

plt.xlabel('Sales')

plt.ylabel('Frequency')

plt.title('Histogram of Sales')

plt.show()



Linear Regression is a supervised learning algorithm in machine learning that supports finding the linear correlation among variables. The result or output of the regression problem is a real or continuous value.

from sklearn.linear\_model import LinearRegression

# Assuming X and y are your features and target variables

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

y = df['Sales']

# Initialize and train a Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Get coefficients and intercept

coefficients = model.coef\_

intercept = model.intercept\_

print(f'Coefficients: {coefficients}')

print(f'Intercept: {intercept}')

