Phase 3: Development Part 1

Name: B.Abinaya
Register Number : 312621243001
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)
```

```
        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
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1   14.0
197  177.0    9.3        6.4   14.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   18.4

[200 rows x 4 columns]
```

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())

	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

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)

	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	15.130500
std	85.854236	14.846809	21.778621	5.283892
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	11.000000
50%	149.750000	22.900000	25.750000	16.000000
75%	218.825000	36.525000	45.100000	19.050000
max	296.400000	49.600000	114.000000	27.000000

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())
```

```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

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))
```

```
        TV  Radio  Newspaper  Sales  class
190   39.5   41.1        5.8   10.8     32
191   75.5   10.8        6.0   11.9     39
192   17.2    4.1       31.6    5.9      7
193  166.8   42.0        3.6   19.6     89
194  149.7   35.6        6.0   17.3     74
195   38.2    3.7       13.8    7.6     14
196   94.2    4.9        8.1   14.0     52
197  177.0    9.3        6.4   14.8     56
198  283.6   42.0       66.2   25.5    118
199  232.1    8.6        8.7   18.4     84
```

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 .

```python
#adding the new feature as new column
df['Total_Spent'] = df['TV'] + df['Radio'] + df['Newspaper']
print(df)
```

```
        TV  Radio  Newspaper  Sales  class  Total_Spent
0    230.1   37.8       69.2   22.1    106        337.1
1     44.5   39.3       45.1   10.4     28        128.9
2     17.2   45.9       69.3   12.0     40        132.4
3    151.5   41.3       58.5   16.5     66        251.3
4    180.8   10.8       58.4   17.9     80        250.0
..     ...    ...        ...    ...    ...          ...
195   38.2    3.7       13.8    7.6     14         55.7
196   94.2    4.9        8.1   14.0     52        107.2
197  177.0    9.3        6.4   14.8     56        192.7
198  283.6   42.0       66.2   25.5    118        391.8
199  232.1    8.6        8.7   18.4     84        249.4

[200 rows x 6 columns]
```

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 .

```python
# 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)
```

```
              TV      Radio  Newspaper  Sales  class  Total_Spent
0     0.969852  0.981522   1.778945   22.1    106        337.1
1    -1.197376  1.082808   0.669579   10.4     28        128.9
2    -1.516155  1.528463   1.783549   12.0     40        132.4
3     0.052050  1.217855   1.286405   16.5     66        251.3
4     0.394182 -0.841614   1.281802   17.9     80        250.0
..         ...       ...        ...    ...    ...          ...
195 -1.270941 -1.321031  -0.771217    7.6     14         55.7
196 -0.617035 -1.240003  -1.033598   14.0     52        107.2
197  0.349810 -0.942899  -1.111852   14.8     56        192.7
198  1.594565  1.265121   1.640850   25.5    118        391.8
199  0.993206 -0.990165  -1.005979   18.4     84        249.4

[200 rows x 6 columns]
```

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)
```

```
                   TV     Radio  Newspaper     Sales     class  Total_Spent
TV           1.000000  0.054809   0.056648  0.901208  0.904861     0.945330
Radio        0.054809  1.000000   0.354104  0.349631  0.346624     0.293211
Newspaper    0.056648  0.354104   1.000000  0.157960  0.144898     0.343059
Sales        0.901208  0.349631   0.157960  1.000000  0.994857     0.924917
class        0.904861  0.346624   0.144898  0.994857  1.000000     0.924750
Total_Spent  0.945330  0.293211   0.343059  0.924917  0.924750     1.000000
```

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())
```

```
                            SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                  200
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -616.270
Date:                Tue, 17 Oct 2023   AIC                           1238.541
Time:                        19:31:30   BIC                           1248.421
Sample:                             0   HQIC                          1242.539
                                - 200
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0125      0.081     -0.154      0.878      -0.171       0.146
ma.L1         -0.9999      3.737     -0.268      0.789      -8.324       6.324
sigma2        27.9129    104.167      0.268      0.789    -176.251     232.077
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 3.72
Prob(Q):                              0.95   Prob(JB):                         0.16
Heteroskedasticity (H):               1.02   Skew:                            -0.09
Prob(H) (two-sided):                  0.95   Kurtosis:                         2.35
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

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)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                  200
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -616.270
Date:                Tue, 17 Oct 2023   AIC                           1238.541
Time:                        19:37:57   BIC                           1248.421
Sample:                             0   HQIC                          1242.539
                                - 200
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0125      0.081     -0.154      0.878      -0.171       0.146
ma.L1         -0.9999      3.737     -0.268      0.789      -8.324       6.324
sigma2        27.9129    104.167      0.268      0.789    -176.251     232.077
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 3.72
Prob(Q):                              0.95   Prob(JB):                         0.16
Heteroskedasticity (H):               1.02   Skew:                            -0.09
Prob(H) (two-sided):                  0.95   Kurtosis:                         2.35
===================================================================================
```

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.

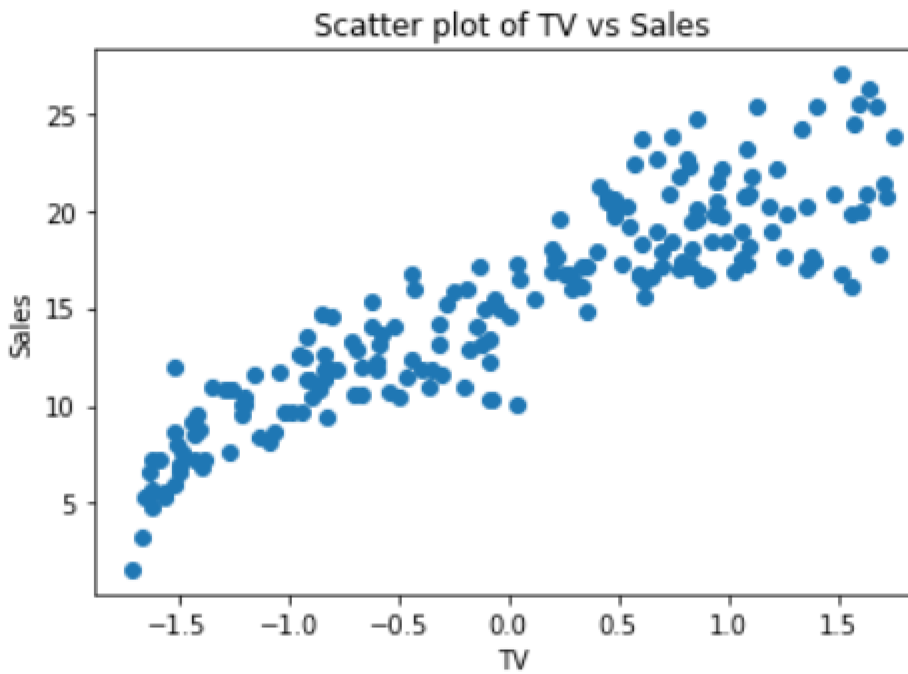
RMSE: In machine learning, it is extremely helpful to have a single number to judge a model's performance, whether it be during training, cross-validation, or monitoring after deployment. Root mean square error is one of the most widely used measures for this.

```
# 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}')
```

```
Mean Absolute Error (MAE): 4.279574757401353
Mean Squared Error (MSE): 27.506268780628666
Root Mean Squared Error (RMSE): 5.244641911573055
```

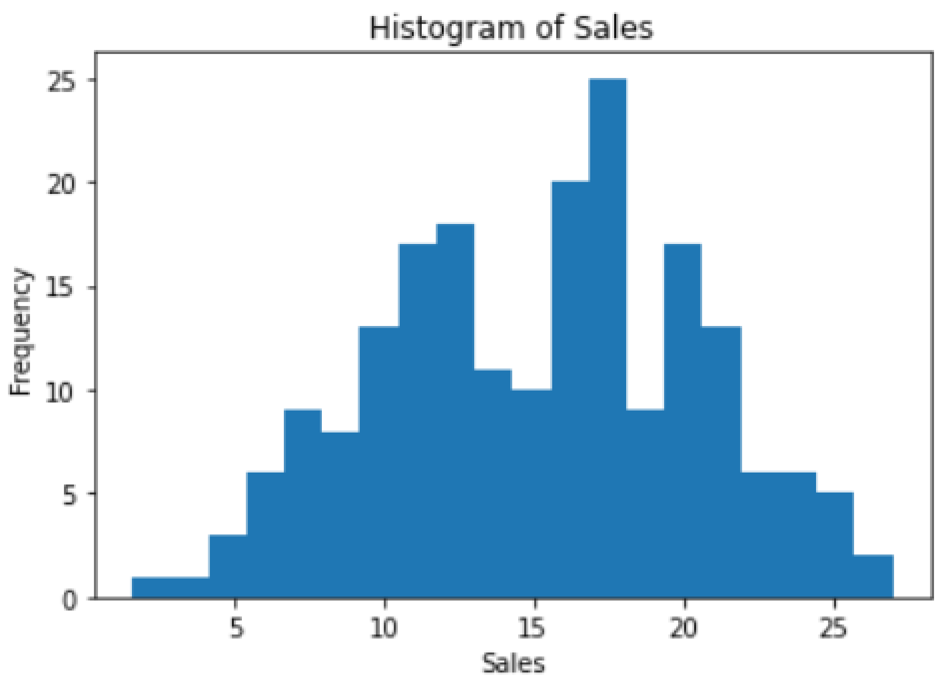
A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

```
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.

```python
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}')
```

```
Coefficients: [4.66270025 1.58465027 0.00729187]
Intercept: 15.130500000000001
```