Phase 1: Problem Definition and Design Thinking

Name: C.Arthi
Register Number : 312621243003
College Name :Thangavelu Engineering College

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.

Design Thinking:

Data Source: Utilize a dataset containing historical sales data. Here a csv file is converted to a DataFrame and the pandas object is used.

Data preprocessing:

- 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 .

- 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.
- 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.
- The drop_duplicates() method removes duplicate rows. Use the subset parameter if only some specified columns should be considered when looking for duplicates.
- The strategy is to convert each category into a column and assign it a 1 or 0 value. It is a process of creating dummy variables. We can see from the table above that all the unique categories were assigned a new column. If a category is present, we have 1 in the column and 0 for others.

Feature engineering:

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 .

Model Selection:

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 **Model Training:**

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.

Evaluation:

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.

Code :

The code should be run in jupyter or collab.

#Data Source utilize the dataset

```
import pandas as pd
data=pd.read_csv(r'Sales.csv')
data
```

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

200 rows × 5 columns

#Data Preprocessing #describe() method

```
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
print(data.describe())
```

	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

#to check any missing values print(data.isnull().sum())

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

#if missing values are their then use this code

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

#to remove duplicate values data

= data.drop_duplicates()

#Categorical column labelencoder

= LabelEncoder()

data['class']=labelencoder.fit_transform(data['Sales']) data.tail(5)

	TV	Radio	Newspaper	Sales	class
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

data['Total_Spent'] = data['TV'] + data['Radio'] + data['Newspaper']

print(data)

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

[200 rows x 5 columns]
```

#Model Selection

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

model = ARIMA(y, order=(p, d, q)) # Create the ARIMA model

model_fit = model.fit() # Fit the model to the data

print(model_fit.summary()) # Summary of the model

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                  200
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -616.270
Date:                Sat, 30 Sep 2023   AIC                           1238.541
Time:                        08:39:18   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).
```

#Model training

train_size = int(len(data) * 0.8) train,

test = data['Sales'][:train_size],

data['Sales'][train_size:] # Initialize and fit the ARIMA model on the training data model = ARIMA(train, order=order) model_fit = model.fit()

Print the summary of the model

print(model_fit.summary())

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                  160
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -492.777
Date:                Sat, 30 Sep 2023   AIC                            995.554
Time:                        11:33:08   BIC                           1010.898
Sample:                             0   HQIC                          1001.785
                                - 160
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7420      2.110     -0.352      0.725      -4.878       3.394
ar.L2         -0.0002      0.123     -0.001      0.999      -0.242       0.242
ma.L1         -0.2499      2.115     -0.118      0.906      -4.396       3.896
ma.L2         -0.7060      2.049     -0.345      0.730      -4.721       3.309
sigma2        28.2650      4.001      7.064      0.000      20.423      36.107
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 3.55
Prob(Q):                              0.96   Prob(JB):                         0.17
Heteroskedasticity (H):               1.25   Skew:                            -0.09
Prob(H) (two-sided):                  0.42   Kurtosis:                         2.29
===================================================================================

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

#model evaluation

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

output

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): 4.589596699334463
Mean Squared Error (MSE): 29.66771325808453
Root Mean Squared Error (RMSE): 5.446807620807305
```