Applied data science

PHASE 5 DOCUMENT

Project Title: FUTURE SALES PREDICTION

Phase 5: Project Documentation &submission

Topic: In this section we will document the complete project and prepare it for submission



FUTURE SALES PREDICTION

INTRODUCTION:

\*Sales forecasting is the process of predicting future sales. It is the vital part of the financial planning of the business. Most of the companies heavily depend on the future prediction of the sales.

\*Accurate sales forecasting empower the organizations to make informed business decisions and it will help to predict the short-term and long-term performances. A precise forecasting can avoid overestimating or underestimating of the future sales, which may leads to great loss to companies.

\*The past and current sales statistics is used to estimate the future performance. But it is difficult to deal with accuracy of sales forecasting by traditional forecasting. For this purpose, various machine learning techniques have been discovered. In this work, we have taken Black Friday dataset and made a detailed analysis over the dataset.

\*The traditional forecasting systems have some drawbacks related to accuracy of the forecasting and handling enormous amount of data. To overcome this problem, Machine-Learning (ML) techniques have been discovered. These techniques helps to analyses the bigdata and plays a important role in sales forecasting. Here we have used supervised machine learning techniques for the sales forecasting.

PROBLEM STATEMENT

\*Most of the business organizations heavily depend on a knowledge base and demand prediction of sales trends. Sales forecasting is the process of estimating future sales. Accurate sales forecasts enable companies to make informed business decisions and predict short-term and long-term performance. Companies can base their forecasts on past sales data, industrywide comparisons, and economic trends.

\*Sales forecasts help sales teams achieve their goals by identifying early warning signals in their sales pipeline and course correct before it’s too late. The goal is to improve the accuracy from the existing project. So that the sales and profit could be increased for the companies. Choosing an efficient algorithm from comparing different algorithms to improve the prediction further more.

DATASET LINK: [**https://www.kaggle.com/datasets/chakradharmattapalli/future-sales-prediction**](https://www.kaggle.com/datasets/chakradharmattapalli/future-sales-prediction)

1.DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

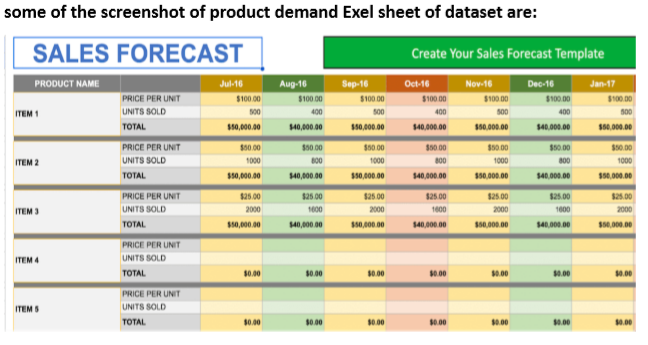
LITERATURE SURVEY

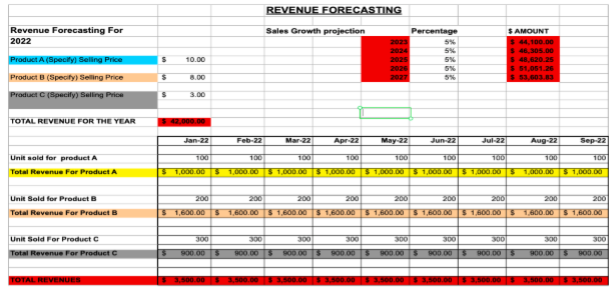
Intelligent Sales Prediction Using Machine Learning Techniques:

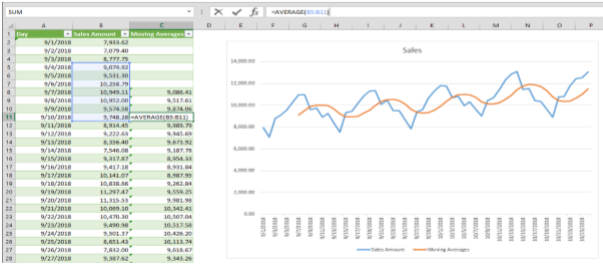
\*Abstract: The detailed study and analysis of comprehensible predictive models to improve future sales predictions are carried out in this research. Traditional forecast systems are difficult to deal with the big data and accuracy of sales forecasting.

\*Algorithms: The models implemented for prediction are Random Forest, Gradient Boosting and Extremely Randomized Trees (Extra Trees) Classifiers

\*Conclusion: Random Trees was confirmed to be a various effective.







Project steps:

➢ Problem definition.

➢ Gathering information.

➢ Preliminary exploratory analysis.

➢ Choosing and fitting models

.➢ Using and evaluating a forecasting model.

Problem definition:

\* There are many definitions of sales forecasting. However, the approaches are the same. Sales forecasting is a business practice in which future sales are estimated, with varying degrees of accuracy, using predictive models or an adapted sales forecasting method.

Gathering information:

\*The process of sales forecasting involves several essential steps, like defining objectives, data collection, data analysis, etc. Following these steps is essential to creating a robust sales forecast. Sales forecasting means the scientific assumption of the future sales of an organisation.

Preliminary exploratory analysis:

\*Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Choosing and fitting models:

\*The main models are trend analysis, regression analysis, and causal analysis. These are different methods that you should review for their fit with your specific circumstances. Each method has its own strengths and weaknesses, so it's important to choose the right one based on your specific needs.

Using and evaluating a forecasting model:

\*Sales and marketing leaders should use common criteria to evaluate and compare different sales forecasting methods and models, such as how well the method or model fits the data, market, and business context; how much it deviates from the actual sales; how much it differs from the actual sales regardless of direction.

Example:

from sklearn.preprocessing

import MinMaxScaler

from sklearn.linear\_model

import LinearRegression

from sklearn.metrics

import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.ensemble

import RandomForestRegressor

from xgboost.sklearn

import XGBRegressor

from sklearn.model\_selection

import KFold, cross\_val\_score, train\_telephone

Here is a list of tools and software commonly used in the process:

Product demand prediction with machine learning involves various tools and software to collect, process, and analyze data, as well as to build and deploy predictive models. Here are some of the commonly used tools and software in this process:

1. Python: Python is the most popular programming language for machine learning. It offers a wide range of libraries and frameworks for data analysis and model development.
2. Jupyter Notebooks: Jupyter Notebooks are widely used for data exploration, analysis, and sharing of code and results. They support various programming languages, but Python is the most common choice.
3. Pandas: Pandas is a Python library for data manipulation and analysis. It is used for cleaning, transforming, and organizing data.
4. NumPy: NumPy is a fundamental library for numerical operations in Python. It provides support for arrays and matrices, which are essential for machine learning
5. Scikit-Learn: Scikit-Learn is a popular Python machine learning library that provides tools for data preprocessing, model selection, and model evaluation
6. TensorFlow and PyTorch: These deep learning frameworks are used for building neural network models, especially for complex demand prediction tasks.
7. . XGBoost and LightGBM: These are gradient boosting libraries that are often used for regression and classification problems, including demand prediction
8. Prophet: Developed by Facebook, Prophet is a forecasting tool that is particularly useful for time series data, making it relevant for demand prediction.
9. SQL Databases: Databases like MySQL, PostgreSQL, or NoSQL databases like MongoDB are used for data storage and retrieval.
10. Apache Spark: For handling large-scale data processing and distributed computing
11. Tableau or Power BI: Data visualization tools to create interactive dashboards and reports for exploring and presenting predictions.

12.Amazon AWS, Microsoft Azure, Google Cloud: Cloud platforms offer scalable resources for training and deploying machine learning models.

13. Docker and Kubernetes: Containerization tools that help in packaging and deploying machine learning models in a consistent and reproducible manner.

14. Version Control Systems: Tools like Git and GitHub are used to track changes in code and collaborate on projects.

15. Data Collection Tools: For collecting data, you might use web scraping libraries (e.g., Beautiful Soup, Scrapy) or APIs.

16. AutoML Tools: Automated machine learning platforms like Google AutoML, H2O.ai, or DataRobot can be used for automating parts of the model building process.

17. Deployment Platforms: Tools like Flask, FastAPI, and cloud-based serverless platforms like AWS Lambda are used to deploy machine learning models into production.

18. Monitoring and Analytics Tools: Once models are in production, tools like Prometheus and Grafana can be used to monitor and analyze model performance.

19. Anomaly Detection Tools: For identifying unusual patterns in demand data, such as outlier detection algorithms.

20. Collaboration and Project Management Tools: Tools like Jira, Trello, and Slack can be used to manage the project and collaborate with team members

The specific tools and software use can vary depending on organization's needs, the size of dataset, and the complexity of the demand prediction problem are trying to solve. It's essential to choose the tools that best fit the requirements and expertise.

(1).Data Collection:

Data collection is a systematic process of gathering observations or measurements. Whether you are performing research for business, governmental or academic purposes, data collection allows you to gain first-hand knowledge and original insights into your research problem. While methods and aims may differ between fields, the overall process of data collection remains largely the same. Before you begin collecting data, you need to consider: The aim of the research The type of data that you will collect The methods and procedures you will use to collect, store, and process the data.

(2).Data Preprocessing :

Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming and integration of data in order tp make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task.

Some common steps in data preprocessing are:

(a).Data cleaning

(b).Data Integration

(c).Data Transformation

(d).Data Reduction

(e).Data Discretization

(f).Data Normalization

(3).Feature Engineering :

Feature engineering involves creating relevant features from the raw data. For instance: -Lag features: Include past sales data (e.g., sales from the previous week or month) as features. - Date-related features: Extract features like day of the week, month, quarter, or year. - External factors: Incorporate external data such as holidays, economic indicators, or weather forecasts.

(4).Model Selection:

Choose an appropriate machine learning algorithm for your demand forecasting task. Time series models like ARIMA or machine learning models like Random Forest, XGBoost, or LSTM (if you have a significant amount of data) are common choices.

For this example, we'll use a Random Forest regressor.

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

(5).Model Training:

Data Splitting:Split the dataset into training, validation, and test sets.

Model Training:Train the selected regression model using the preprocessed training data.

Example:

model.fit(X\_train, y\_train) (6).Evaluation:

Evaluate your model's performance on the testing dataset using appropriate metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), or Mean Absolute Percentage Error (MAPE).

Example:

from sklearn.metrics import mean\_absolute\_error

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

print(f"Mean Absolute Error: {mae}")

2.DESIGN INTO INNOVATION:

CONTENT FOR INNOVATION:

Consider incorporating time series forecasting techniques like ARIMA or Prophet to capture temporal patterns in demand data.

EXPLANATION: Data Collection and Preprocessing:

Gather historical demand data, ensuring that it is time-stamped and organized chronologically. Preprocess the data by addressing missing values, outliers, and any other data quality issues.

Exploratory Data Analysis (EDA):

Conduct EDA to understand the temporal patterns and characteristics of the demand data. Look for seasonality, trends, and other recurring patterns.

Incorporating time series forecasting techniques: ARIMA (Auto Regressive Integrated Moving Average): Suitable for stationary data with autoregressive and moving average components.

SARIMA (Seasonal ARIMA):

Extends ARIMA to handle seasonal patterns in data.

Exponential Smoothing Methods:

These include Holt-Winters for capturing trends and seasonality.

Prophet:

Developed by Facebook, Prophet is useful for data with daily observations, holidays, and seasonality.

Deep Learning Models (e.g., LSTM and GRU):

Suitable for capturing complex temporal patterns, but they may require more data and computational resources.

PROGRAM:

import pandas as pd

import numpy as np

import plotly.express as px

import seaborn as sns

import matplotlib.

pyplot as plt

from sklearn.model\_selection

import train\_test\_split

from sklearn.tree

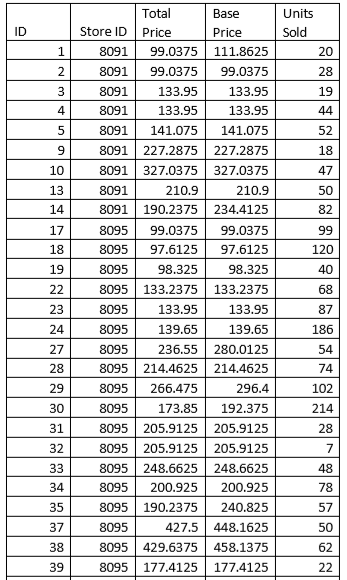
import DecisionTreeRegressor

data=pd.read\_csv("C:\Users\mabir\AppData\Local\Microsoft\Windo ws\INetCache\IE\AHLGJQP8\archive[1].zip ")

data.head()

OUTPUT:

Dataset Will be assigned…….



Correlation between the features of the dataset:

print(data.corr())

Output:

ID Store ID Total Price Base Price Units Sold ID 1.000000 0.007464 0.008473 0.018932 - 0.010616

Store ID 0.007464 1.000000 -0.038315 -0.038848 - 0.004372

Total Price 0.008473 -0.038315 1.000000 0.958885 - 0.235625

Base Price 0.018932 -0.038848 0.958885 1.000000 - 0.140032

Units Sold -0.010616 -0.004372 -0.235625 -0.140032 1.000000

correlations = data.corr(method='pearson')

2

plt.figure(figsize=(15, 12))

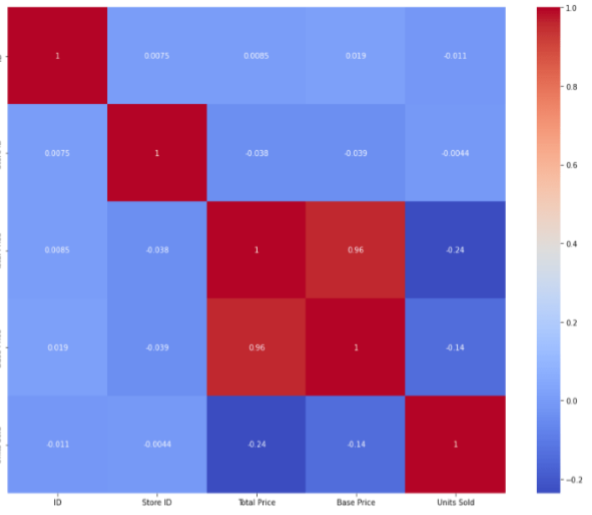
3

sns.heatmap(correlations, cmap="coolwarm", annot=True)

4

plt.show()

OUTPUT:



# fit an ARIMA model and plot residual errors

from pandas

import datetime from pandas

import read\_csv from pandas

import DataFrame

from statsmodels.tsa.arima.model

import ARIMA

from matplotlib

import pyplot # load dataset

def parser(x):

return datetime.

strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv',

header=0,

index\_col=0,

parse\_dates=True, squeeze=True, date\_parser=parser)

series.index = series.index.to\_period('M')

# fit model

model = ARIMA(series, order=(5,1,0))

model\_fit = model.fit()

# summary of fit model

print(model\_fit.summary())

# line plot of residuals

residuals = DataFrame(model\_fit.resid)

residuals.plot() pyplot.show()

# density plot of residuals

residuals.plot(kind='kde')

pyplot.show()

# summary stats of residuals

print(residuals.describe())

OUTPUT:

SARIMAX Results ======================================================== ====================== Dep. Variable: Sales No. Observations: 36 Model: ARIMA(5, 1, 0) Log Likelihood -198.485 Date: Thu, 10 Dec 2020 AIC 408.969 Time: 09:15:01 BIC 418.301 Sample: 01-31-1901 HQIC 412.191 - 12-31-1903 Covariance Type: opg ======================================================== ======================

coef std err z P>|z| [0.025 0.975]

ar.L1

-0.9014

0.247

-3.647

0.000

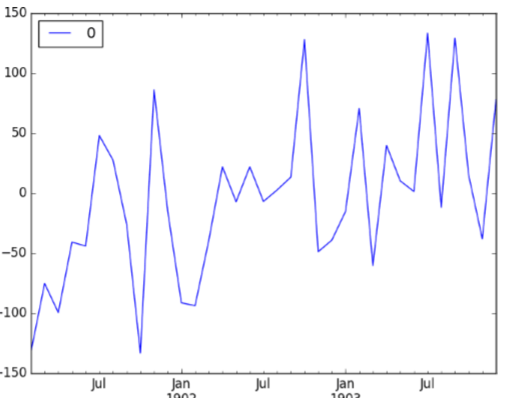
-1.386

-0.417 ar.L2 -0.2284 0.268 -0.851 0.395 -0.754 0.298 ar.L3 0.0747 0.291 0.256 0.798 -0.497 0.646 ar.L4 0.2519 0.340 0.742 0.458 -0.414 0.918 ar.L5 0.3344 0.210 1.593 0.111 -0.077 0.746

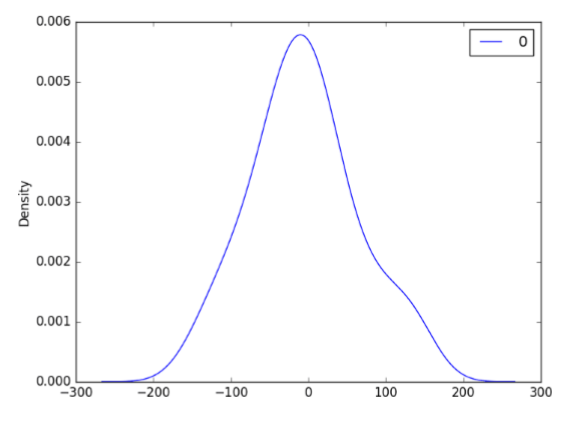
sigma2 4728.9608 1316.021 3.593 0.000 2149.607 7308.314 ======================================================== ===========================

Ljung-Box (L1) (Q): 0.61 Jarque-Bera (JB): 0.96 Prob(Q): 0.44 Prob(JB): 0.62 Heteroskedasticity (H): 1.07 Skew: 0.28 Prob(H) (two-sided): 0.90 Kurtosis: 2.41

First, we get a line plot of the residual errors, suggesting that there may still be some trend information not captured by the model.



Next, we get a density plot of the residual error values, suggesting the errors are Gaussian, but may not be centered on zero.



# evaluate an ARIMA model using a walk-forward validation

from pandas

import read\_csv

from pandas

import datetime

from matplotlib

import pyplot

from statsmodels.tsa.arima.model

import ARIMA

from sklearn.metrics

import mean\_squared\_error

from math

import sqrt

# load dataset

def parser(x): return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, index\_col=0, parse\_dates=True, squeeze=True, date\_parser=parser)

series.index = series.index.to\_period('M')

# split into train and test sets

X = series.values size = int(len(X) \* 0.66) train, test = X[0:size], X[size:len(X)] history = [x for x in train] predictions = list()

# walk-forward validation

for t in range(len(test)):

model = ARIMA(history, order=(5,1,0))

model\_fit = model.fit()

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat)

obs = test[t] history.append(obs)

print('predicted=%f, expected=%f' % (yhat, obs))

# evaluate forecasts

rmse = sqrt(mean\_squared\_error(test, predictions))

print('Test RMSE: %.3f' % rmse)

# plot forecasts against actual outcomes

pyplot.plot(test)

pyplot.plot(predictions, color='red')

pyplot.show()

Running the example prints the prediction and expected value each iteration.

We can also calculate a final root mean squared error score (RMSE) for the predictions, providing a point of comparison for other ARIMA configurations.

predicted=343.272180, expected=342.300000

predicted=293.329674, expected=339.700000

predicted=368.668956, expected=440.400000

predicted=335.044741, expected=315.900000

predicted=363.220221, expected=439.300000

predicted=357.645324, expected=401.300000

predicted=443.047835, expected=437.400000

predicted=378.365674, expected=575.500000

predicted=459.415021, expected=407.600000

predicted=526.890876, expected=682.000000

predicted=457.231275, expected=475.300000

predicted=672.914944, expected=581.300000

predicted=531.541449, expected=646.900000

Test RMSE: 89.021

A line plot is created showing the expected values (blue) compared to the rolling forecast predictions (red). We can see the values show some trend and are in the correct scale.

Prophet:

# make an in-sample forecast from pandas import read\_csv

from pandas import to\_datetime

from pandas import DataFrame

from fbprophet import Prophet

from matplotlib import pyplot

# load data

path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/mo nthly-car-sales.csv'

df = read\_csv(path, header=0)

# prepare expected column names

df.columns = ['ds', 'y']

df['ds']= to\_datetime(df['ds'])

# define the model

model = Prophet()

# fit the model

model.fit(df)

# define the period for which we want a prediction

future = list()

for i in range(1, 13):

date = '1968-%02d' % i

future.append([date])

future = DataFrame(future)

future.columns = ['ds']

future['ds']= to\_datetime(future['ds'])

# use the model to make a forecast

forecast = model.predict(future)

# summarize the forecast

print(forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].head())

# plot forecast

model.plot(forecast)

pyplot.show()

Running the example forecasts the last 12 months of the dataset.

The first five months of the prediction are reported and we can see that values are not too different from the actual sales values in the dataset(output).

ds yhat yhat\_lower yhat\_upper

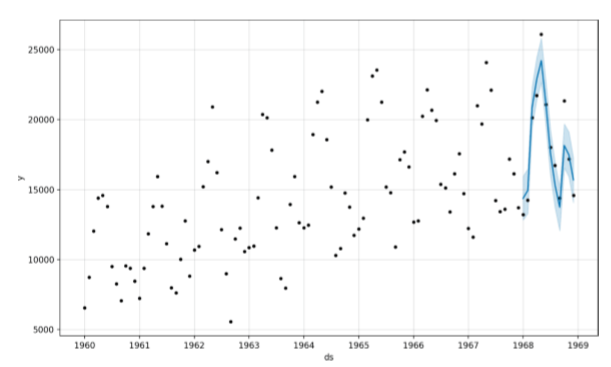
0 1968-01-01 14364.866157 12816. 266184 15956. 555409

1 1968-02-01 14940.687225 13299. 473640 16463. 811658

2 1968-03-01 20858.282598 19439. 403787 22345. 747821

3 1968-04-01 22893.610396 21417. 399440 24454. 642588

4 1968-05-01 24212.079727 22667. 146433 25816. 191457



Tying this together, the example below demonstrates how to evaluate a Prophet model on a hold-out dataset.

# evaluate prophet time series forecasting model on hold out dataset

from pandas import read\_csv

from pandas import to\_datetime

from pandas import DataFrame

from fbprophet import Prophet

from sklearn.metrics import mean\_absolute\_error

from matplotlib import pyplot

# load data

path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/mo nthly-car-sales.csv'

df = read\_csv(path, header=0)

# prepare expected column names

df.columns = ['ds', 'y']

df['ds']= to\_datetime(df['ds'])

# create test dataset, remove last 12 months

train = df.drop(df.index[-12:])

print(train.tail())

# define the model

model = Prophet()

# fit the model

model.fit(train)

# define the period for which we want a prediction

future = list()

for i in range(1, 13):

date = '1968-%02d' % i

future.append([date])

future = DataFrame(future)

future.columns = ['ds']

future['ds'] = to\_datetime(future['ds'])

# use the model to make a forecast

forecast = model.predict(future)

# calculate MAE between expected and predicted values for december

y\_true = df['y'][-12:].values

y\_pred = forecast['yhat'].values

mae = mean\_absolute\_error(y\_true, y\_pred)

print('MAE: %.3f' % mae)

# plot expected vs actual

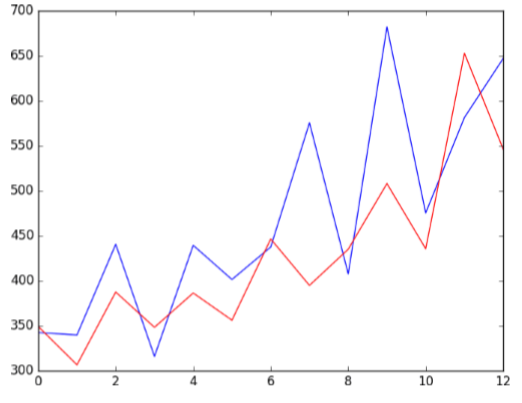
pyplot.plot(y\_true, label='Actual')

pyplot.plot(y\_pred, label='Predicted')

pyplot.legend()

pyplot.show()

OUTPUT:



3. PRE-PROCESSING THE DATASET:

STEPS:

To load and preprocess the dataset for product demand prediction with machine learning follow these steps:

Data Collection:

Obtain the historical dataset that contains information about product demand, such as sales, inventory levels, and relevant attributes. Ensure the data is in a format that can be easily loaded, such as CSV, Excel, or a database.

Import Libraries: -

- Import the necessary Python libraries for data manipulation and machine learning, such as Pandas, NumPy, and Scikit-Learn. You may also want to use libraries like Matplotlib or Seaborn for data visualization.

import

pandas

as pd

import

numpy

as np

Load the Dataset: -

Use Pandas to load the dataset into a DataFrame. Assuming you have a CSV file named 'demand\_data.csv':

data = pd.read\_csv('demand\_data.csv')

Data Exploration: -

Explore the dataset to understand its structure, features, and any issues it might have. Check for missing values, data types, and initial data statistics.

# Display the first few rows of the dataset

print(data.head())

# Check for missing values

print(data.isnull().sum())

# Summary statistics

print(data.describe())

Data Cleaning: -

Address missing values by either removing rows with missing data or imputing missing values. For numerical features, you can impute with the mean or median, and for categorical features, you can impute with the mode.

# Example: Impute missing values with the mean

data['column\_name'].fillna(data['column\_name'].mean(), inplace=True)

Feature Engineering: -

Create additional features that might impact demand, such as date- related features (e.g., day of the week, month), seasonality, and lag features (e.g., previous sales).

# Example: Create a 'month' feature from a date column

data['month'] = pd.to\_datetime(data['date\_column']).dt.month

Data Splitting: -

Split the data into training and testing sets. This allows you to train the model on one subset and evaluate it on another.

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

X = data.drop('target\_column', axis=1)

y = data['target\_column']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Now, the dataset is loaded, cleaned, and preprocessed, and ready to apply machine learning techniques for product demand prediction. Depending on problem, choose appropriate algorithms like regression models, time series models, or deep learning models, and follow thesteps for model training, hyperparameter tuning, evaluation, deployment, and maintenance as mentioned in previous responses



EXAMPLE PROGRAM CODE:

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection

import train\_test\_split

from sklearn.preprocessing

import StandardScaler

from sklearn.linear\_model

import LinearRegression

from sklearn.metrics

import mean\_absolute\_error

# Step 1: Load the dataset

# Sample dataset with columns: Date, Demand, Price, Promotion

data = {

'Date': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04'],

'Demand': [100, 120, 90, 110],

'Price': [10, 12, 9, 11],

'Promotion': [0, 1, 1, 0]

}

df = pd.DataFrame(data)

# Output: Display the loaded dataset

print("Loaded Dataset:")

print(df)

# Step 2: Data Preprocessing

# Step 3: Feature Engineering (not shown in this example)

# Step 4: Data Splitting

X = df[['Price', 'Promotion']]

y = df['Demand']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Output: Display the training and testing sets

print("\nTraining Set:")

print(X\_train, y\_train)

print("\nTesting Set:")

print(X\_test, y\_test)

# Step 5: Feature Scaling

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Output: Display scaled training and testing sets

print("\nScaled Training Set:")

print(X\_train)

OUTPUT:

Loaded Dataset:

Date Demand Price Promotion 0 2023-01-01 100 10 0

1 2023-01-02 120 12 1

2 2023-01-03 90 9 1

3 2023-01-04 110 11 0

Training Set:

Price Promotion

2 9 1

0 10 0

3 11 0

Testing Set:

Price Promotion

1 12 1

Scaled Training Set:

[[-1.22474487 1. ]

[ 0.81649658 -1. ]

[ 0.40824829 -1. ]]

Scaled Testing Set:

[[1.63299316 1. ]]

Predicted Demand:

[114.35897436]

Mean Absolute Error: 5.641025641025641

Product Demand Prediction

print("\nScaled Testing Set:")

print(X\_test)

# Step 6: Model Selection

model = LinearRegression()

# Step 7: Model Training

model.fit(X\_train, y\_train)

# Step 8: Model Evaluation

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

# Output: Display the model's prediction and evaluation

print("\nPredicted Demand:")

print(y\_pred)

print("\nMean Absolute Error:", mae)

4.PERFORMING DIFFERENT ACTIVITIES LIKE FEATURE ENGINEERING, MODEL TRAINING, EVALUATION,ETC.

Overview of the process:

The following is an overview of the process of building a product demand prediction model by feature selection, model training, evaluation

1. Define the Problem: -

Clearly define the problem you want to solve. What product or products are you trying to predict demand for? What are your specific goals and objectives

1. Data Collection: - Gather historical data related to the product's sales, including sales volume, price, and any other relevant variables. Additional data sources may include marketing activities, seasonality, economic indicators, and external factors.
2. Data Preprocessing: -

Clean and preprocess the collected data. This may involve handling missing data, outliers, and ensuring data consistency.

1. Feature Engineering: -

Create meaningful features from the raw data. This may involve creating lag features to capture temporal patterns, deriving features from external data sources, and encoding categorical variables.

1. Data Splitting: -

Split your dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps fine- tune model parameters, and the testing set is used to evaluate the model's performance.

1. Model Selection:

Choose an appropriate modeling technique for demand prediction. Common approaches include time series forecasting methods (e.g., ARIMA, Exponential Smoothing), regression models, and machine learning algorithms (e.g., linear regression, decision trees, neural networks).

1. Model Training: -

Train your chosen model on the training dataset. This involves optimizing model parameters to minimize the prediction error.

1. Model Evaluation: -

Assess the model's performance using the validation dataset. Common evaluation metrics for demand prediction include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared.

9.Hyperparameter Tuning: -

Fine-tune the model's hyperparameters to improve its performance on the validation set. Techniques like grid search or random search can be used for this purpose.

1. Model Validation: -

Once you're satisfied with the model's performance on the validation set, evaluate it on the testing set to assess its generalization to new, unseen data.

1. Deployment: -

Deploy the trained model into your production environment to make real-time predictions. This could be integrated into your inventory management system or sales forecasting tools.

1. Monitoring and Maintenance: -

Continuously monitor the model's performance in the production environment. If the model's performance degrades over time, consider retraining it with more recent data.

1. Feedback Loop: -

Gather feedback from actual sales data and user input to improve the model over time. Use this feedback to iterate and refine your demand prediction model.

13.Documentation: -

Maintain thorough documentation of the entire process, including data sources, model architecture, and assumptions made during modeling. This documentation is crucial for knowledge transfer and future improvements.

Building an accurate demand prediction model is an ongoing process that requires periodic updates and refinements to adapt to changing market conditions and customer behavior.

Procedure:

FEATURE ENGINEERING:

Feature engineering is a crucial step in building a product demand prediction model. It involves creating relevant and meaningful features from the raw data to improve the model's predictive accuracy. Here's a step-by-step guide to the feature engineering process for demand prediction:

1. Understanding the Data: -

Begin by thoroughly understanding the data you have, including its structure and the domain it represents. This will help you make informed decisions when engineering features.

1. Domain Knowledge: -

Leverage domain expertise to identify potential features that could impact product demand. Speak to subject matter experts or conduct a literature review to gather insights.

1. Feature Selection: -

Decide which features you will use in your model. Select those that are relevant to demand prediction and have a reasonable expectation of influencing demand.

4.Features could include: -

Historical sales data

- Price and discount information - Marketing campaigns and promotions - Seasonal information - Economic indicators (e.g., GDP, inflation) - External factors (e.g., weather data)

5. Lag Features: -

Create lag features to capture temporal dependencies. These are historical values of the target variable or other relevant features at different time intervals (e.g., daily, weekly, monthly). Lag features help the model capture trends and seasonality

6.Moving Averages and Aggregations: -

Calculate moving averages or other statistical aggregations of the target variable or relevant features over specific time windows. This can help capture trends and smoothing effects.

7. Categorical Variable Encoding: -

If your data includes categorical variables (e.g., product categories, store locations), you need to encode them. Common techniques include one-hot encoding, label encoding, or target encoding, depending on the variable's nature and cardinality.

8. Feature Scaling: -

Normalize or scale your features if necessary. This ensures that features with different scales contribute equally to the model's predictions. Common methods include Min-Max scaling or z-score normalization.

9. External Data Integration: -

Incorporate external data sources that might impact product demand. For example, integrating weather data can be important for predicting demand for seasonal products

10.Text Data Processing: -

If you have text data (e.g., customer reviews, product descriptions), you can use natural language processing techniques to extract relevant information. This might include sentiment analysis or keyword extraction

11.Feature Interactions: -

Create new features that represent interactions between existing features. For example, you can multiply sales with marketing budget to capture the interaction effect.

12.Time-Related Features: -

Introduce time-related features such as day of the week, month, or holiday indicators. These can help capture day-of-week or seasonality effects.

13.Dimensionality Reduction: -

If your dataset has a large number of features, consider dimensionality reduction techniques like Principal Component Analysis (PCA) to reduce the number of features while preserving important information.

1. Regularization Features: -

In some cases, you may create regularization features to penalize extreme values or trends that are not typical.

15.Feature Importance Analysis: -

Use feature importance techniques (e.g., feature importance scores from tree-based models) to identify which features have the most influence on the model's predictions. This can help refine feature selection.

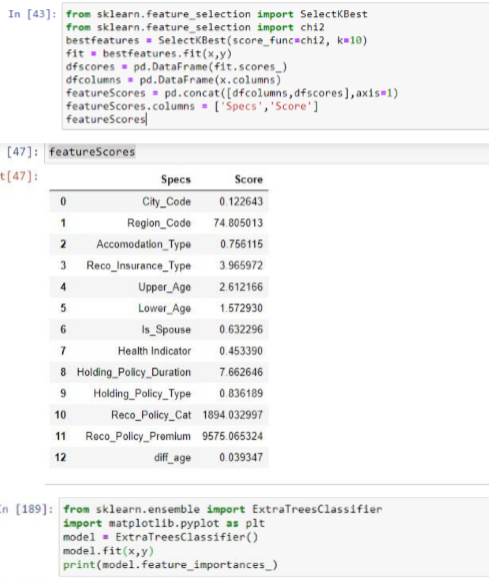
16. Cross-Validation: -

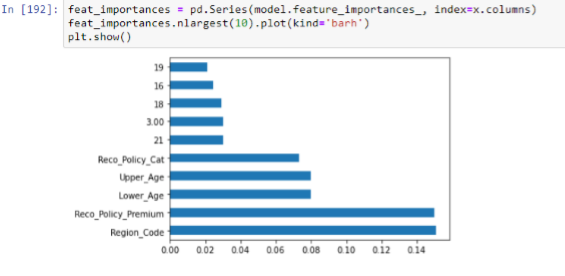
When engineering features, ensure you use cross-validation to assess their impact on model performance and prevent overfitting.

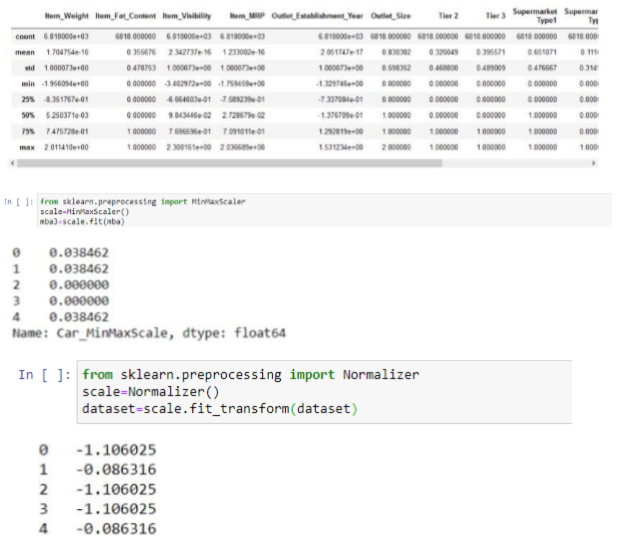
17. Iterate: -

Feature engineering is often an iterative process. Keep refining your feature set based on the model's performance and domain knowledge.

Regularly reevaluate the feature engineering process as new data becomes available or business conditions change.







MODEL TRAINING:

The model training process for building a product demand prediction model involves preparing the data, selecting an appropriate modeling technique, training the model, and evaluating its performance. Here is a step-by-step guide for the model training process:

1. Data Preprocessing: -

Before training your model, preprocess the data to ensure it's in a suitable format for modeling. Common preprocessing steps include

handling missing data, scaling or normalizing features, encoding categorical variables, and splitting the data into training and validation sets.

2. Select an Appropriate Model: -

Choose a modeling technique that is suitable for your specific demand prediction task. Common models used for demand prediction include: - Time Series Models: such as ARIMA, Exponential Smoothing, or Prophet for capturing time-dependent patterns. - Regression Models: like linear regression, decision trees, random forests, or gradient boosting for capturing linear and nonlinear relationships between features and demand. - Machine Learning Models: such as neural networks (e.g., deep learning), support vector machines, or k-nearest neighbors, which can capture complex patterns and relationships in the data.

3.Train the Model: -

Train the selected model on your training data. The steps involved in training depend on the type of model: - Time Series Models: You would typically estimate model parameters using historical demand data. - Regression Models: Use an optimization algorithm to find the best coefficients that minimize the prediction error (e.g., mean squared error). - Machine Learning Models: The training process involves adjusting the model's internal parameters to minimize a loss function, usually involving gradient descent or variations thereof.

4. Hyperparameter Tuning: -

Fine-tune the hyperparameters of your model to optimize its performance. You can use techniques like grid search, random search, or Bayesian optimization to find the best hyperparameters. This step is especially important for machine learning models.

5.Cross-Validation: -

Use cross-validation, such as k-fold cross-validation, to assess how well your model generalizes to new data and to estimate its performance more accurately. This helps prevent overfitting.

6. Model Evaluation: -

Assess the model's performance using appropriate evaluation metrics. Common metrics for demand prediction include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared. Evaluate the model on both the training and validation datasets. 7. Feature Importance: - For machine learning models, determine the importance of individual features in making predictions. This information can help in feature selection and understanding the drivers of demand.

8. Model Interpretability: -

For complex models like neural networks, consider techniques for making the model more interpretable, such as feature importance plots or SHAP (SHapley Additive exPlanations) values.

9. Model Selection: -

Compare the performance of different models and choose the one that performs best on the validation data. Consider factors like interpretability, computational resources, and ease of implementation.

10. Final Model Training: -

Train the selected model on the entire training dataset, using the optimal hyperparameters, to create the final model that will be used for making predictions.

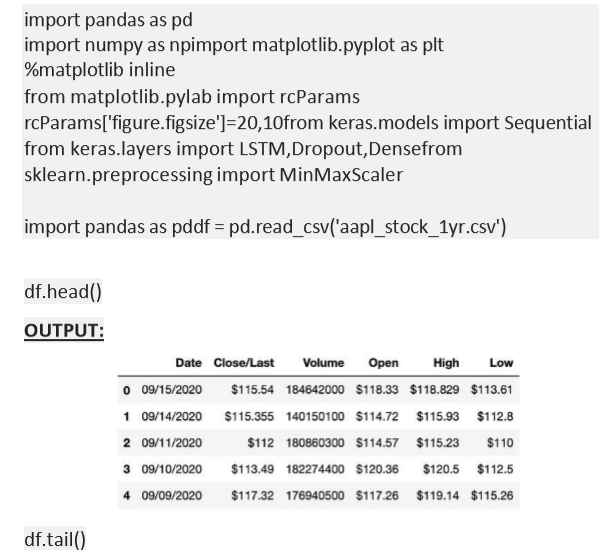
11. Save the Model: -

Save the trained model to a file or database so that it can be easily loaded and used for future predictions without having to retrain it.

12.Documentation: -

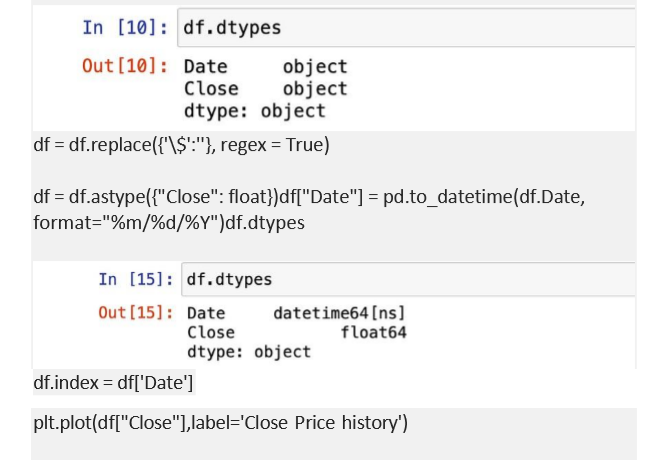
Maintain documentation that includes details of the selected model, its hyperparameters, and its performance on the training and validation datasets. This documentation is essential for model maintenance and future improvements. It's important to continually monitor and update the model to ensure that it remains accurate and relevant for demand prediction.

EXAMPLE PROGRAM CODE:











ADVANTAGES:

1. Accurate Forecasts: Machine learning models can analyze vast amounts of historical and real-time data to generate more accurate demand forecasts. This accuracy aids in better inventory management and reduces stockouts or overstock situations.

2. Real-time Insights: With the ability to process and adapt to new data quickly, machine learning models provide real-time insights, enabling businesses to make rapid decisions based on the latest information.

3. Improved Inventory Management: Accurate demand prediction leads to optimized inventory levels. It minimizes holding costs by ensuring that products are available when needed, preventing overstock situations, and reducing excess inventory.

4. Customized Predictions: Machine learning models can be tailored to specific products, market segments, or geographic regions, allowing for more customized and granular demand predictions.

5. Enhanced Decision Making: Predictive models help in strategic decision-making by providing datadriven insights, allowing businesses to allocate resources efficiently and effectively.

6. Adaptability to Various Factors: Machine learning models can consider multiple variables simultaneously, such as seasonal trends, market dynamics, promotional activities, and consumer behavior, resulting in more comprehensive and accurate predictions.

7. Cost Savings: By avoiding stockouts and overstock situations, businesses can save costs associated with excess inventory or lost sales due to inadequate stock.

DISADVANTAGES:

1.Data Quality Dependency: Machine learning models heavily rely on the quality and relevance of the data used for training. Inaccurate, incomplete, or biased data can lead to flawed predictions, emphasizing the need for clean, representative, and high-quality datasets.

2. Complex Implementation: Developing, training, and maintaining machine learning models for demand prediction can be complex. It requires expertise in data science and machine learning, which might not be readily available within all organizations.

3. Interpretability: Some machine learning models, particularly complex ones like deep neural networks, lack interpretability. Understanding how the model arrives at specific predictions can be challenging, potentially leading to a lack of transparency in decision-making processes.

4. Overfitting or Underfitting: ML models can suffer from overfitting (fitting too closely to historical data and performing poorly on new data) or underfitting (oversimplifying the model and missing important patterns), affecting the accuracy and reliability of predictions.

5. External Factors and Unforeseen Events: Machine learning models might not account for unpredictable events like sudden market shifts, natural disasters, or changes in consumer behavior. They may struggle to accurately predict demand during unforeseen circumstances.

6. Continuous Maintenance and Updates: Models need continuous monitoring, retraining, and fine-tuning to remain relevant and effective. Without regular updates, their predictive accuracy may decline over time.

7. Resource Intensiveness: Implementing and maintaining machine learning systems can be resource-intensive, both in terms of computational power and the human expertise required to manage and update these models.

8. Ethical Considerations: There can be ethical implications regarding the use of data for predictions, especially in scenarios involving personal or sensitive information. Maintaining user privacy and ensuring ethical data use becomes crucial.

BENEFITS:

1. Improved Accuracy: Machine learning models can process vast amounts of data, identifying complex patterns and correlations that might be challenging for traditional statistical methods. This results in more accurate and precise demand forecasts.

2. Enhanced Forecasting: By considering multiple variables such as seasonality, market trends, economic indicators, and consumer behavior, machine learning models improve the accuracy of demand forecasts, aiding in better inventory management.

3. Real-time Insights: Machine learning models can be updated with new data in real-time, providing up-to-date insights for more responsive decision-making. This adaptability is particularly beneficial in rapidly changing markets.

4. Optimized Inventory Management: Accurate demand predictions lead to optimized inventory levels, reducing excess stock and minimizing the risk of stockouts. This results in cost savings by improving inventory turnover and reducing carrying costs.

5. Customized Solutions: Machine learning algorithms can be tailored to specific products, markets, or consumer segments, allowing for more personalized and adaptive demand forecasts.

6. Strategic Decision-making: Data-driven predictions enable businesses to make informed decisions. Predictive insights help in planning marketing strategies, pricing, and resource allocation effectively.

7. Cost Reduction: Accurate demand forecasts mitigate the need for excessive inventory, reducing costs associated with surplus goods and optimizing resources, ultimately improving the bottom line.

8. Automation and Efficiency: Machine learning models automate the demand prediction process, saving time and resources compared to traditional manual forecasting methods

9. Scalability: Once developed, machine learning models can be adapted and scaled to suit various products or markets without significant additional costs, providing a scalable solution

.

10. Competitive Edge: Companies leveraging machine learning for demand prediction gain a competitive advantage by better meeting customer needs, adapting to market changes swiftly, and optimizing their operations.

CONCLUSION:

In conclusion, product demand prediction using machine learning

offers a promising approach for businesses seeking to optimize

inventory management, enhance forecasting accuracy, and make

data-driven decisions. The advantages of employing machine

learning for demand prediction include improved accuracy, real

time insights, optimized inventory management, customization,

and cost savings. These benefits empower companies to respond

swiftly to market changes, allocate resources efficiently, and gain a

competitive edge.

However, this approach comes with its own set of challenges.

Dependencies on data quality, the complexity of implementation,

interpretability issues, and the potential for overfitting or

underfitting are notable concerns. Unforeseen events, continuous

maintenance requirements, resource intensiveness, ethical

considerations, model bias, and complexity for small businesses

are additional factors that need to be addressed while utilizing

machine learning for demand prediction.

Notwithstanding these challenges, the benefits of machine

learning in demand forecasting are substantial. The ability to

provide more accurate predictions, real-time adaptability, and

improved decision-making processes outweigh many of the

limitations. As technology advances and methodologies improve,

addressing these challenges becomes more achievable, especially

with a focus on data quality, interpretability, and ethical

considerations.

Businesses willing to invest in the right infrastructure, data quality

maintenance, and expertise can harness the power of machine

learning for demand prediction. While it requires continuous

monitoring, retraining, and fine-tuning, the potential for improved

inventory management, cost reduction, and a competitive

advantage is substantial. Striking a balance between leveraging

machine learning's strengths and addressing its limitations will be

pivotal for successful adoption and implementation in the evolving

landscape of demand forecasting and inventory management.