**PROJECT: FUTURE SALES PREDICTION**

**Phase-5**

**PROCESS:**

* + 1. Problem statement
    2. Design thinking process and phases development
    3. Dataset description
    4. Usage of dataset
    5. Data pre-processing
    6. Model training
    7. Forecasting
    8. Evaluation

**1.PROBLEM STATEMNT:**

Predicting sales of a company needs time series data of that company and based on thatdata the model can predict the future sales of that company or product. So, in this research project we will Analyze the time series sales data of a company and will predict the sales of the company for the coming quarter and for a specific product.

The problem definition of future sales prediction involves using historical sales data, market trends, and other relevant factors to forecast future sales for a specific product or service within a certain time frame. This prediction is essential for businesses to make informed decisions regarding inventory management, marketing strategies and financial planning. This primary goal is to develop a reliable model or algorithm that can accurately estimate future sales volumes helping businesses optimize their operations and resources.

**2.DESIGN THINKING PROCESS AND PHASES DEVELOPMENT:**

1. Data Source

2 . Data Pre-processing

3. Feature Engineering

4. Model Selection

5. Model Training

6. Evaluation

**1.Data source:**

1.Historical Sales Data:

This is a fundamental source. It includes past sales records, which can reveal trends, seasonality, and patterns.

2.Market Data :

Information about the overall market conditions, such as economic indicators, industry trends, and competitor data, can be valuable.

3.Customer Data:

Understanding customer behaviour, preferences, and demographics can aid in predicting future sales. This can include data from CRM systems, surveys, and social media.

4.Product Data :

Information about your products or services, including specifications, pricing, and any changes or updates, is crucial.

5.External Data:

Utilize external data sources like weather data (for weather-dependent sales), social media sentiment, or news events that might impact sales.

2.**Data Pre-processing:**

1.Data collection: Gather historical sales data, including transaction records, product information, customer data, and any other relevant variables.

2.Data cleaning: Remove or handle missing values, duplicate entries, and outliers to ensure data quality

. 3.Future Engineering: Create new features or transform existing ones to extract valuable information, such as seasonality, trends, and customer segmentation.

4.Data scaling and normalization: Standardize numerical features to have a consistent scale, preventing certain features from dominating the model.

5.Encoding Categorical data : Convert categorical variables into numerical format using techniques like one-hot encoding or label encoding.

**3.Feature Engineering:**

Feature engineering, in simple terms, is the act of converting raw observations into desired features using statistical or machine learning approaches.

**Ideas:**

1. Historical Sales Data: Include past sales data, such as daily, weekly, or monthly sales figures. You can calculate various statistics like moving averages, seasonality, and trends from this data.

2. Calendar Features: Incorporate calendar-related features like holidays, day of the week, and month to capture seasonal trends.

3. Lagged Variables: Create lag features, representing past sales for different time periods (e.g., lag of 1 month, 3 months) to capture autocorrelation.

4. Product Attribute: Include information about the products, such as category, brand, price, and any promotions or discounts.

5. Store Information: Consider store-specific details like location, size, and foot traffic. This can help account for variations in different store location

**4.Model selection:**

Statistical Methods:

ARIMA (Auto Regressive Integrated Moving Average): Suitable for stationary time series data. It can capture auto-regressive and moving average patterns in the data. ARIMA models are effective when the data has a clear trend and seasonality.

State Space Models:

Kal man Filter: Suitable for dynamic systems with linear relationships. It can be used for state-space modeling and time series prediction.

Deep Learning with Transformers:

Transformer-based Models: These models, like GPT (Generative Pre-trained Transformer), have shown promise in time series forecasting tasks, especially when dealing with sequences of varying lengths and complex pattern.

**5.Model training:**

Model training is the phase in the data science development lifecycle where practitioners try to fit the best combination of weights and bias to a machine learning algorithm to minimize a loss function over the prediction range.

Steps:

1. Data Collection: Gather historical sales data, including information on products, sales channels, time periods, and any relevant external factors like marketing campaigns or economic indicators.

2. Data Preprocessing: Clean and prepare the data by handling missing values, outliers, and converting categorical variables into numerical representations through techniques like one-hot encoding.

3. Feature Engineering: Create relevant features that can help the model better understand the data. This might include creating lag features, aggregating data by time periods, or incorporating external data sources.

4. Splitting Data: Divide your dataset into training, validation, and test sets to evaluate the model's performance accurately.

5. Selecting a Model: Choose an appropriate machine learning model for sales prediction. Common choices include linear regression, decision trees, random forests, or more advanced techniques like neural networks.

**6. Evaluation:**

Definition: Evaluation is the process of assessing or appraising something based on certain criteria or standards to determine its value, effectiveness, quality, or significance

1. Performance Evaluation: Assessing an individual's or a group's performance in a specific role or task, often used in workplaces to determine promotions, bonuses, or areas for improvement.

2. Program Evaluation: Analyzing the effectiveness and efficiency of a program, project, or intervention, usually to determine whether it achieves its intended goals and if resources are used appropriately.

3. Product Evaluation: Assessing the quality, functionality, and overall performance of a product or service to determine its suitability for a particular purpose.

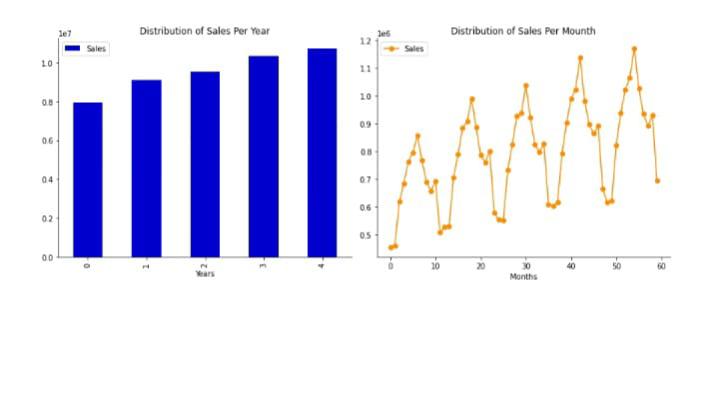
1.Mean Absolut Error: ���=1�∑�=1�∣��−�^�∣MAE=n1∑i=1n∣Yi−Y^i∣ 3.Mean Bias Deviation (MBD): Formula: ���=1�∑�=1�(�^�−��)MBD=n1∑i=1n(Y^i−Yi)

2. Mean Absolute Percentage Error (MAPE): Formula:����=100�∑�=1�∣��−�^�∣��MAPE=n100∑i=1nYi∣Yi−Y^i∣ THANKING YO

**3.DATASET DESCRIPTION:**

Predicting the future sales of a product helps a business manage the manufacturing and advertising cost

of the product. There are many more benefits of predicting the future sales of a product.



The dataset given here contains the data about the sales of the product. The dataset is about the advertising cost incurred by the business on various advertising platforms. Below is the description of all.

The columns in the dataset:

TV: Advertising cost spent in dollars for advertising on TV;

Radio: Advertising cost spent in dollars for advertising on Radio;

Newspaper: Advertising cost spent in dollars for advertising on Newspaper;

Sales: Number of units sold;

So, in the above dataset, the sales of the product depend on the advertisement cost of the product. I hope you now have understood everything about this dataset. Now in the section below, I will take you through the task of future sales prediction with machine learning using Python.

**Columns details:**

Column explanation in the dataset:

The dataset we mentioned appears to be related to predicting sales based on different advertising mediums, such as television, radio, and newspapers. Here’s a brief explanation of the columns typically found in such a dataset:

1. Television: This column likely contains data on the amount of money spent on advertising via television for a specific period, such as a week or month.

2. Radio: Similarly, this column contains data on advertising expenses for radio advertising.

3. Newspaper: This column contains data on advertising expenses for newspaper advertisements.

4. Sales: This column usually represents the sales figures for a product or service during the same period for which advertising expenses are recorded.

To predict future sales based on these advertising mediums, you can use regression analysis or machine learning techniques.

**Libraries:**

NumPy: NumPy is used for numerical operations in Python. You can install it using pip:

Pip install numpy

Pandas: Pandas is used for data manipulation and analysis. Install it with pip:

Pip install pandas

Matplotlib and seaborn: These libraries are essential for data visualization:

Pip install matplotlib seaborn

Scikit\_Learn: Scikit\_learn is a powerful library for machine learning. You can use it for building predictive models:

Pip install scikit-learn

Tensorflow Or Torch: If you plan to work with deep learning models, you’ll need one of these Libraries:

Pip install Tensorflow or Pip install torch

statsmodels: Statsmodels is useful for statistical modeling and Hypothesis testing:

Pip install statsmodels

prophet: If you want to use a library specifically designed for Data science and model development install it:

Pip install fbprophet

Jupyter notebook: While not a library, Jupyter Notebook it an excellent interactive environment for data science and model development. Install it with:

Pip install jupyter

Others data processing libraries: Depending on your data, you might need other libraries like scikit-learn’s preprocessing module (`sklearn.preprocessing`) for data preprocessing tasks.

Database Libraries: your data is stored in a database, consider using libraries like SQLAlchemy or PyMySQL to connect to and fetch data from the database.

 Remember to keep your libraries and dependencies up-to-date. Pip Install-upgrade library-name. Once you have these libraries installed, you can start working on your future sales prediction project

using the various data.

**4.USAGES OF DATASET:**

Import pandas as pd

Import numpy as np

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

From sklearn.linear\_model import LLinearRegression

Data\_pd.read\_csv(https://raw.githubusercontent.com/amankharwal/Website-

data/master/advertising.csv)

Print(data.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

Our goal is to predict monthly sales, so we will first consolidate all stores and days into total monthly sales.

Dataset = pd.read\_csv('/input/demand\_forecasting\_kernals\_only/sample\_submission.csv’)

Df = dataset.copy()

Df.head()

Id sales

0 52

1 52

2 52

3 52

4 52

Def load\_data(file\_name):

"""Returns a pandas dataframe from a csv file.""""

Return pd.read\_csv(file\_name)

Sales\_data = load\_data('../input/demand-forecasting-kernels-only/train.csv')

Df\_s.sales\_data.copy(

Df\_s.info()

<class 'pandas.core.frame.DataFrame">

RangeIndex: 913000 entries, 0 to 912999

Data columns (total 4 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

date 913000 non-null object

store 913000 non-null int64

item 913000 non-null int64

sales 913000 non-null int64

Dtypes: int64(3), object(1)

Memory usage: 27.9+ MB

Df\_s.tail()

Date store item sales

912995 2017-12-27 10 50 63

912996 2017-12-28 10. 50. 59

912997 2017-12-29 10 50 74

912998 2017-12-30 10 50 62

912999 2017-12-31 10 50 82

# To view basic statistical details about dataset:

Df\_s[‘sales’].describe()

Count 913000.000000

Mean 52.250287

Std 28.801144

Min 0.000000

25% 30.000000

50% 47.000000

75% 70.000000

Max 231.000000

Name: sales, dtype: float64

Sales seem to be unbalanced!

Df\_s[‘sales’].plot()

**Train data and Test data:**

1. Training Data: This is the portion of the dataset used to train a machine learning model. The model learns patterns, relationships, and features from this data. It's like a teacher showing examples to a student.
2. Testing Data: After training, the model is tested on a separate dataset called the testing data.

# Split the data into training and testing sets

Train\_size = int(len(sales\_data) \* 0.8)

Train\_data, test\_data = sales\_data[:train\_size], sales\_data[train\_size:]

# Fit an ARIMA model to the training data

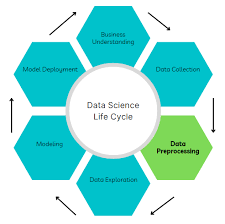
Model = ARIMA(train\_data, order=(5,1,0)) # You can adjust the order parameters

Model\_fit = model.fit (disp=0)

# Make predictions on the test set

Predictions = model\_fit.forecast(steps=len(test\_data))[0]

**5.DATA PREPROCESSING:**

****

**Definition:**

Data preprocessing is a critical step in future sales prediction using data science. Here are key steps to consider:

1.**Data collection:**

Gather historical sales data, including transaction records, product information, customer data, and any other relevant variables.

2.**Data cleaning**:

Remove or handle missing values, duplicate entries, and outliers to ensure data quality.

3.**Future Engineering:**

Create new features or transform existing ones to extract valuable information, such as seasonality, trends, and customer segmentation.

4.**Data scaling and normalization**:

Standardize numerical features to have a consistent scale, preventing certain features from dominating the model.

5.**Encoding Categorical data** :

Convert categorical variables into numerical format using techniques like one-hot encoding or label encoding .

**6**.**MODEL TRAINING:**

Model training is the key step in machine learning that results in a model ready to be validated, tested, and deployed. The performance of the model determines the quality of the applications that are built using it. Quality of training data and the training algorithm are both important assets during the model training phase. Typically, training data is split for training, validation and testing. The training algorithm is chosen based on the end use case. There are a number of trade off points in deciding the best algorithm–model complexity, interpretability, performance, compute requirements, etc. All these aspects of model training make it both an involved and important process in the overall machine learning development cycle.

Linear regression:

Training a model like linear regression typically involves using a dataset to find the best-fit line that minimizes the error between the predicted values and the actual values. Here's a simplified example in Python using the scikit-learn library:

```python

import numpy as np

from sklearn.linear\_model import LinearRegression

# Sample dataset

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1) # Input features

y = np.array([2, 4, 5, 4, 5]) # Target values

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Make predictions

predictions = model.predict(X)

# Print the model parameters

print("Coefficients:", model.coef\_) # Slope of the line

print("Intercept:", model.intercept\_) # Intercept of the line

```

This code creates a linear regression model, fits it to the dataset, and makes predictions.

Random forest:

Training a model like a Random Forest involves using a dataset to build an ensemble of decision trees. Here's a simplified example in Python using the scikit-learn library:

```python

from sklearn.ensemble import RandomForestRegressor

from sklearn.datasets import make regression

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

from sklearn.metrics import mean\_squared\_error

# Generate a sample dataset (you can replace this with your own data)

X, y = make\_regression(n\_samples=100, n\_features=1, noise=0.2, random\_state=42)

# Split the data into training and testing sets

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

# Create and train a Random Forest Regressor

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

rf\_model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = rf\_model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

```In this example, we generate a synthetic dataset, split it into training and testing sets, create a Random Forest Regressor, and make predictions.

Decision tree:

Training a Decision Tree model involves using a dataset to create a tree-like structure of decisions to make predictions. Here's a simplified example in Python using the scikit-learn library:

```python

from sklearn.tree import DecisionTreeRegressor

from sklearn.datasets import make\_regression

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

from sklearn.metrics import mean\_squared\_error

# Generate a sample dataset (you can replace this with your own data)

X, y = make\_regression(n\_samples=100, n\_features=1, noise=0.2, random\_state=42)

# Split the data into training and testing sets

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

# Create and train a Decision Tree Regressor

tree\_model = DecisionTreeRegressor(random\_state=42)

tree\_model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = tree\_model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

```In this example, we generate a synthetic dataset, split it into training and testing sets, create a Decision Tree Regressor, and make predictions.

### **Performing Simple Linear Regression**

Equation of linear regression  
y=c+m1x1+m2x2+...+mnxny=c+m1x1+m2x2+...+mnxn

yy is the response

cc is the intercept

m1m1 is the coefficient for the first feature

mnmn is the coefficient for the nth feature

In our case:

y=c+m1×TVy=c+m1×TV

The mm values are called the model **coefficients** or **model parameters**.

### **Generic Steps in model building using statsmodels**

We first assign the feature variable, TV, in this case, to the variable X and the response variable, Sales, to the variable y.

X = advertising['TV']

y = advertising['Sales']

#### **Train-Test Split**

You now need to split our variable into training and testing sets. You'll perform this by importing train\_test\_split from the sklearn.model\_selection library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size = 0.7, test\_size = 0.3, random\_state = 0;

# Let's now take a look at the train dataset

X\_train.head()

74 213.4

3 151.5

185 205.0

26 142.9

90 134.3

Name: TV, dtype: float64

y\_train.head()

74 17.0

3 16.5

185 22.6

26 15.0

90 14.0

Name: Sales, dtype: float64

**Building a Linear Model**

You first need to import the statsmodel.api library using which you'll perform the linear regression.

import statsmodels.api as sm

By default, the statsmodels library fits a line on the dataset which passes through the origin. But in order to have an intercept, you need to manually use the add\_constant attribute of statsmodels. And once you've added the constant to your X\_train dataset, you can go ahead and fit a regression line using the OLS (Ordinary Least Squares) attribute of statsmodels as shown below

*#* Add a constant to get an intercept

X\_train\_sm = sm.add\_constant(X\_train)

*# Fit the resgression line using 'OLS'*

lr = sm.OLS(y\_train, X\_train\_sm).fit()

# Print the parameters, i.e. the intercept and the slope of the regression line fitted

lr.params

const 6.948683

TV 0.054546

dtype: float64

In [19]:

# Performing a summary operation lists out all the different parameters of the regression line fitted

print(lr.summary())

OLS Regression Results

==============================================================================

Dep. Variable: Sales R-squared: 0.816

Model: OLS Adj. R-squared: 0.814

Method: Least Squares F-statistic: 611.2

Date: Thu, 07 Mar 2019 Prob (F-statistic): 1.52e-52

Time: 06:21:53 Log-Likelihood: -321.12

No. Observations: 140 AIC: 646.2

Df Residuals: 138 BIC: 652.1

Df Model: 1

Covariance Type: nonrobust

==============================================================================

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

------------------------------------------------------------------------------

const 6.9487 0.385 18.068 0.000 6.188 7.709

TV 0.0545 0.002 24.722 0.000 0.050 0.059

==============================================================================

Omnibus: 0.027 Durbin-Watson: 2.196

Prob(Omnibus): 0.987 Jarque-Bera (JB): 0.150

Skew: -0.006 Prob(JB): 0.928

Kurtosis: 2.840 Cond. No. 328

**7.FORECASTING:**

Forecasting future sales of a product offers many advantages. Predicting future sales of a product helps a company manage the cost of manufacturing and marketing the product. In this notebook, we will try to you through the task of future sales prediction with machine learning using Python.

EDA Libraries:

Import pandas as pd

Import numpy as np

Import matplotlib.colors as col

From mpl\_toolkits.mplot3d import Axes3D

Import matplotlib.pyplot as plt

Import seaborn as sns

%matplotlib inline

Import datetime

From pathlib import Path

Import random

**8.EVALUATION:**

Definition: Evaluation is the process of assessing or appraising something based on certain criteria or standards to determine its value, effectiveness, quality, or significance

Certainly! Here's a basic example of how to perform model evaluation using Python and the scikit-learn library. We'll use a simple classification model for illustration:

```python

# Import necessary libraries

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# Assuming you have a dataset with features X and labels y

# Split the dataset into training and testing sets

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

# Create and train a model (Logistic Regression in this case)

model = LogisticRegression()

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

# Make predictions on the test set

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

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print the evaluation metrics

print("Accuracy: ", accuracy)

print("Precision: ", precision)

print("Recall: ", recall)

print("F1 Score: ", f1)

print("Confusion Matrix:\n", conf\_matrix)

```In this example, we use the `LogisticRegression` model, split the data into training and testing sets, make predictions on the test set, and calculate common classification metrics such as accuracy, precision, recall, F1 score, and the confusion matrix. You can replace `LogisticRegression` with any other machine learning algorithm that suits your problem.

## **Model Evaluation**

### **Residual analysis**

To validate assumptions of the model, and hence the reliability for inference

**Distribution of the error terms**:We need to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like,

y\_train\_pred = lr.predict(X\_train\_sm)

res = (y\_train - y\_train\_pred)

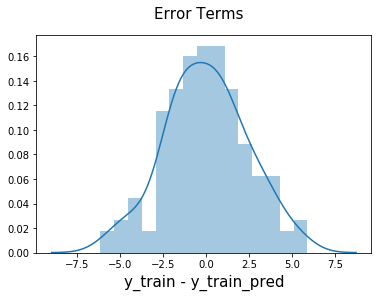
fig = plt.figure()

sns.distplot(res, bins = 15)

fig.suptitle('Error Terms', fontsize = 15) *# Plot heading*

plt.xlabel('y\_train - y\_train\_pred', fontsize = 15) *# X-label*

plt.show()



The residuals are following the normally distributed with a mean 0. All good!

#### **Looking for patterns in the residuals**

plt.scatter(X\_train,res)

plt.show()

We are confident that the model fit isn't by chance, and has decent predictive power. The normality of residual terms allows some inference on the coefficients.

Although, the variance of residuals increasing with X indicates that there is significant variation that this model is unable to explain.

As you can see, the regression line is a pretty good fit to the data

### Image

### **Predictions on the Test Set**

Now that you have fitted a regression line on your train dataset, it's time to make some predictions on the test data. For this, you first need to add a constant to the X\_test data like you did for X\_train and then you can simply go on and predict the y values corresponding to X\_test using the predict attribute of the fitted regression line.

# Add a constant to X\_test

X\_test\_sm = sm.add\_constant(X\_test)

# Predict the y values corresponding to X\_test\_sm

y\_pred = lr.predict(X\_test\_sm)

y\_pred.head()

126 7.374140

104 19.941482

99 14.323269

92 18.823294

111 20.132392

dtype: float64

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score

**looking at the RMSE:**

#Returns the mean squared error; we'll take a square root

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

Out[27]:

2.019296008966232

Checking the R-squared on the test set

In [28]:

r\_squared = r2\_score(y\_test, y\_pred)

r\_squared

Out[28]:

0.792103160124566

Visualizing the fit on the test set

In [29]:

plt.scatter(X\_test, y\_test)

plt.plot(X\_test, 6.948 + 0.054 \* X\_test, 'r')

plt.show()

