FUTURE SALES PREDICTION

**NAAN MUDHALVAN PHASE-3**

Importing the required libraries:

We need a combination of libraries for data manipulation, data analysis, and machine learning. Here are some common libraries we might need:

1.**Pandas**: Pandas is a powerful library for data manipulation and analysis. It's great for loading, cleaning, and exploring your data.

import pandas as pd

2.**NumPy**: NumPy provides support for working with arrays and matrices of numerical data. It's often used for mathematical and statistical operations.

import numpy as np

3.**Matplotlib and Seaborn**: These libraries are helpful for data visualization, which is important for understanding your sales data.

import matplotlib.pyplot as plt

import seaborn as sns

4. **Scikit-Learn (sklearn)**: Scikit-Learn is a powerful machine learning library that includes tools for regression and classification. You can use it for building sales prediction models.

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

from sklearn.linear\_model import LinearRegression # Example algorithm

from sklearn.metrics import mean\_squared\_error, r2\_score

5. **Statsmodels**: Statsmodels is a library for estimating and interpreting models for statistical analysis.

import statsmodels.api as sm

Importing the data set (read data set: create matrix):

To import a dataset for future sales prediction, use the Pandas library to read the data and create a DataFrame. Here we see how its implemented

import pandas as pd

- Replace 'your\_dataset.csv' with the actual path to your dataset file

dataset = pd.read\_csv('your\_dataset.csv')

- Optionally, you can take a quick look at the first few rows of the dataset

print(dataset.head())

In above code:-

- `import pandas as pd` imports the Pandas library as `pd`.

- `pd.read\_csv('your\_dataset.csv')` reads your dataset from a CSV file. Replace `'your\_dataset.csv'` with the actual path to your dataset file.

Once you've read your dataset into a Pandas DataFrame, you can start exploring and analyzing the data. You may also need to preprocess the data, perform feature engineering, and split it into training and testing sets for future sales prediction tasks.

Handling the Missing Data:

Handling missing data using scikit-learn's `SimpleImputer` is a straightforward and effective method. Here's how to use it to handle missing data in a dataset for future sales prediction:

First, ensure you've imported the required libraries, including scikit-learn:

import pandas as pd

from sklearn.impute import SimpleImputer

Assuming you've already read your dataset into a Pandas DataFrame (`dataset`), you can follow these steps:

1. **Identify Missing Data**:

# Check for missing values in your dataset

missing\_data = dataset.isnull().sum()

print(missing\_data)

2. **Instantiate and configure the `SimpleImputer`**:

- Choose a strategy for imputation, such as 'mean,' 'median,' 'most\_frequent,' or 'constant' (to replace with a specific value).

- Specify which columns you want to impute using the `strategy` parameter.

- Create an instance of `SimpleImputer` and fit it on your data.

# Choose a strategy (e.g., 'mean', 'median', 'most\_frequent', or 'constant')

strategy = 'mean'

# Specify columns with missing values that you want to impute

columns\_to\_impute = ['column1', 'column2']

# Create the imputer

imputer = SimpleImputer(strategy=strategy)

# Fit the imputer on the specified columns

imputer.fit(dataset[columns\_to\_impute])

3. **Apply the imputer to the missing values and replace them in the DataFrame**:

# Transform and replace missing values in the DataFrame

dataset[columns\_to\_impute]=imputer.transform(dataset[columns\_to\_impute])

Now, the dataset should have missing values replaced with the strategy that selected. You can customize the strategy based on your dataset's characteristics.

**After handling missing data, you can proceed with your future sales prediction tasks using the cleaned dataset.**

Encoding Categorical Data. (one-hot encoding):

One-hot encoding categorical data is a common preprocessing step for future sales prediction. Here's how you can use Python to perform one-hot encoding on your categorical data:

Assuming you have a dataset in a Pandas DataFrame and a list of categorical columns to encode (`categorical\_columns`), follow these steps:

1. **Import the necessary libraries**:

import pandas as pd

2. **Identify your categorical columns**:

# List of categorical columns that need one-hot encoding

categorical\_columns = ['category1', 'category2', 'location', 'product\_type']

3. **Perform one-hot encoding using Pandas' `get\_dummies` function**:

# Perform one-hot encoding for the specified categorical columns

encoded\_data = pd.get\_dummies(dataset, columns=categorical\_columns, drop\_first=True)

- `pd.get\_dummies` will convert the specified categorical columns into a set of binary (0 or 1) columns for each category.

- The `drop\_first=True` parameter is used to drop the first category in each feature to avoid multicollinearity issues. This is often necessary to prevent redundancy in the encoding.

After applying one-hot encoding, you will have a new DataFrame, `encoded\_data`, which contains the one-hot encoded features.

Here's an example of how you can use this encoded data to prepare your features and target variable:

# Assuming your target column is 'sales'

X = encoded\_data.drop('sales', axis=1) # Features

Y = encoded\_data['sales'] # Target variable

Make sure to replace `dataset` with the actual name of your DataFrame and adjust the list of `categorical\_columns` according to your dataset's structure.

Encoding Categorical Data. (one-hot encoding):

Splitting your dataset into a training set and a test set is a crucial step in preparing data for future sales prediction.

You can use the `train\_test\_split` function from scikit-learn to achieve this. Here's how to do it:

Step1- make sure you've imported the necessary libraries:

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

Step 2 - assuming you have your features `X` and the target variable `Y` (typically 'sales') ready, you can split the data as follows:

# Split the data into a training set and a test set

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

**In this code:**

- `X` represents your feature matrix.

- `Y` represents your target variable, which is often the 'sales' column.

- `test\_size` specifies the proportion of the data to include in the test set. Here, it's set to 0.2, meaning 20% of the data will be used for testing, and the remaining 80% will be used for training. You can adjust this ratio according to your needs.

- `random\_state` is an optional parameter that allows you to set a seed for the random number generator. This ensures that the split is reproducible.

After splitting your data, there are four datasets:

- `X\_train`: The feature matrix for the training set.

- `X\_test`: The feature matrix for the test set.

- `Y\_train`: The target variable for the training set.

- `Y\_test`: The target variable for the test set.

**You can then use these datasets to train and evaluate your future sales prediction models.**

Feature Scaling. (import StandardScaler):

Future scaling is used for more preprocess. The `StandardScaler` from scikit-learn is a common choice for standardizing your features. Here's how to use it:

importing the necessary libraries:

from sklearn.preprocessing import StandardScale

Assuming you have your feature matrices `X\_train` and `X\_test` already split into training and testing sets, you can apply feature scaling as follows:

# Instantiate the StandardScaler

scaler = StandardScaler()

# Fit the scaler on the training data and transform it

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

# Transform the test data using the same scaler

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

**In this code:**

- `StandardScaler` scales your features by centering them around the mean and scaling them to have a standard deviation of 1. This is done independently for each feature.

- `scaler.fit\_transform(X\_train)` fits the scaler on the training data (`X\_train`) and then applies the transformation. The training data is used to estimate the mean and standard deviation for each feature, and the transformation is applied to both the training and test data.

- `scaler.transform(X\_test)` applies the same transformation to the test data, ensuring that the scaling is consistent between the training and test datasets.

After feature scaling, your `X\_train\_scaled` and `X\_test\_scaled` contain the standardized features, which are ready to be used for training and evaluating your future sales prediction models.