**TITLE**

**FUTURE SALES PREDICTION**

# DEVELOPMENT PART 1

**INTRODUCTION:-**

* The development phase of creating a future sales prediction model is where the theoretical concepts and strategies begin to take shape. In this phase, you will work with your data, select and train machine learning models, and optimize their performance.
* The overarching goal is to transform historical sales data into actionable insights for better business planning.
* This phase typically encompasses data preprocessing, model selection, training, evaluation, and deployment.

**Import Necessary Libraries**

* Start by importing the required Python libraries for data manipulation and analysis:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

**Load the Historical Sales Dataset**

* Load your historical sales dataset into a pandas DataFrame. You can use various file formats, but in this example, we'll assume the dataset is in a CSV file:

# Replace 'your\_dataset.csv' with the actual file path or URL of your dataset.

data = pd.read\_csv(‘sales.csv')

**Step 3: Data Exploration**

* Before preprocessing, explore the data to understand its

structure, identify missing values, and gain insights into its characteristics

# Display the first few rows of the dataset

print(data.head())

# Check for missing values.

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

# Get summary statistics of the data.

print(data.describe())

**Step 4: Data Preprocessing**

**Step 3: Data Preprocessing for LSTM**

LSTM models require specific data preprocessing for time series data:

**3.1: Data Sorting**

Ensure that your data is sorted by time, with the earliest time points at the beginning:

data['date'] = pd.to\_datetime(data['date']) # Convert the date column to a datetime format. data = data.sort\_values(by='date')

**Scaling Data**

LSTM models benefit from feature scaling, usually in the range [0, 1]. Min-max scaling is a common choice:

scaler = MinMaxScaler()

data['sales\_scaled'] = scaler.fit\_transform(data['sales'].values.reshape(-1, 1))

**Sequence Data Preparation**

Prepare sequences of data, where each sequence contains a series of historical sales values and the target is the sales value at the next time step. The choice of the sequence length depends on your specific problem

sequence\_length = 10 # Example sequence length.

sequences = []

targets = []

for i in range(len(data) - sequence\_length):

sequences.append(data['sales\_scaled'].values[i:i+sequence\_length])

targets.append(data['sales\_scaled'].values[i+sequence\_length])

**Train-Test Split**

Split your data into training and testing sets for model evaluation

split\_ratio = 0.8

split\_index = int(split\_ratio \* len(sequences))

X\_train = np.array(sequences[:split\_index])

y\_train = np.array(targets[:split\_index])

X\_test = np.array(sequences[split\_index:])

y\_test = np.array(targets[split\_index:])

**Build the LSTM Model**

Construct the LSTM model using Keras. You can adjust the architecture, such as the number of layers, units, and activation functions, to suit your specific problem:

model = Sequential()

model.add(LSTM(50, activation='relu', input\_shape=(sequence\_length,

1)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

**Model Training**

Train the LSTM model on the training data

model.fit(X\_train, y\_train, epochs=50, batch\_size=32)

**Model Evaluation and Predictions**

Evaluate your model on the test data and make predictions for future sales values. You can use evaluation metrics like mean squared error (MSE) or root mean squared error (RMSE) to assess the model's performance.

**Deployment and Integration**

Once satisfied with the model's performance, you can deploy it to make real-time predictions or integrate it into your business processes.

Building a future sales prediction model using LSTM is an iterative process that may require fine-tuning and experimentation with various hyperparameters and model architectures to achieve the best results.