**Sales and Stock Prediction Code Document:**

This document provides an overview and explanation of the Python code developed for sales and stock predictions based on historical data. The code incorporates 30 prediction models, each designed to generate predictions for sales and stock levels for individual items or dates. The results are processed and displayed through a Streamlit application in a tabular format. The historical input is expected to be provided as a CSV file, either online or from a local system path.

Code Structure

* + Data Preprocessing
  + Model Training
  + Prediction Generation
  + Results Aggregation
  + Streamlit Application

**Packages and Modules**:

Packages : SK LEARN , STATSMODELS , TENSORFLOW

Modules:

1. Prophet: Time series forecasting model developed by Facebook.

2. Multiple Regression: Statistical technique for analyzing the relationship between multiple independent variables and a dependent variable.

3. Decision Tree: Machine learning algorithm for classification and regression tasks based on hierarchical, tree-like structures.

4. KMeans: Clustering algorithm that partitions data into K clusters based on similarity.

5. Logistic Regression: Statistical method for binary classification problems.

6. SVR (Support Vector Regression): Regression technique that uses support vector machines for prediction.

7. BGM (Bayesian Gaussian Mixture): Bayesian approach to Gaussian mixture models for clustering.

8. GRU (Gated Recurrent Unit): Recurrent neural network (RNN) variant for sequence modeling.

9. Ridge Regression: Linear regression with regularization to prevent overfitting.

10. Lasso Regression: Linear regression with L1 regularization to encourage sparsity in the model.

11. SVM (Support Vector Machine):Machine learning algorithm for classification and regression that finds a hyperplane to separate data into distinct classes while maximising the margin.

12. MLP Reg (Multilayer Perceptron Regression): Neural network model for regression tasks.

13. ARD Reg (Automatic Relevance Determination Regression): Bayesian linear regression with automatic variable selection.

14. Stack Reg (Stacked Regression): Ensemble technique that combines multiple regression models.

15. PLS Reg (Partial Least Squares Regression): Regression method that finds latent variables to predict the target.

16. Poisson Reg: Regression model for count data, assuming a Poisson distribution.

17. LSTM (Long Short-Term Memory): Recurrent neural network architecture designed for handling long-range dependencies.

18. Gaussian Process: Probabilistic model that defines a distribution over functions for regression tasks.

19. Neural Networks: Computational models inspired by the human brain, commonly used for various machine learning tasks.

20. ARIMA (AutoRegressive Integrated Moving Average): Time series forecasting model combining autoregression, differencing, and moving averages.

21. SARIMA (Seasonal AutoRegressive Integrated Moving Average): Extended version of ARIMA with seasonal components.

22. Linear Regression: Simple linear model for predicting a continuous target variable based on linear relationships with input features.

23. Random Forest: Ensemble learning method consisting of multiple decision trees for classification and regression.

24. Exponential Smoothing: Time series forecasting method that assigns exponentially decreasing weights to past observations.

25. XGBoost: Gradient boosting algorithm known for its efficiency and high performance.

26. LightGBM Reg (Light Gradient Boosting Machine Regression): Gradient boosting framework optimized for speed and memory efficiency.

27. CatBoost: Gradient boosting library designed for categorical feature support.

28 PCA (Principal Component Analysis): Dimensionality reduction technique for capturing essential features in data.

29. ICA (Independent Component Analysis): Blind source separation method to extract independent components from mixed signals.

30.Elastic Net Reg (Elastic Net Regression): Hybrid regression model combining L1 and L2 regularization

Usage

To use the code, follow these steps:

* Prepare historical sales and stock data in CSV format, either online or from a local system path.
* Input the path or URL of the CSV file in the code.
* Run the code to execute the training, prediction generation, and Streamlit application processes.
* Access the Streamlit application to view the aggregated predictions in a tabular format and we can download as csv file.

***CODE :***

pip install streamlit

import streamlit as st

import pandas as pd

from prophet import Prophet

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

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVR

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear\_model import Lasso

from sklearn.neural\_network import MLPRegressor

from sklearn.ensemble import StackingRegressor

from datetime import timedelta

from sklearn.linear\_model import ARDRegression

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from sklearn.linear\_model import PoissonRegressor

from sklearn.ensemble import RandomForestRegressor

from math import sqrt

from statsmodels.tsa.holtwinters import ExponentialSmoothing

#from tensorflow.keras.models import Sequential

#from tensorflow.keras.layers import LSTM, Dense

from sklearn.gaussian\_process import GaussianProcessRegressor

from sklearn.gaussian\_process.kernels import RBF, ConstantKernel as C

from sklearn.preprocessing import StandardScaler

#from tensorflow.keras.models import Sequential

#from tensorflow.keras.layers import Dense

#from tensorflow.keras.optimizers import Adam

#from tensorflow.keras.models import Sequential

#from tensorflow.keras.layers import GRU, Dense

#Prophet

def prophet\_prediction(file\_path):

df\_prophet = pd.read\_csv(file\_path)

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

model\_prophet = Prophet()

model\_prophet.fit(df\_prophet)

future\_prophet = model\_prophet.make\_future\_dataframe(periods=365, freq='D')

forecast\_prophet = model\_prophet.predict(future\_prophet)

prophet\_output = forecast\_prophet[['ds', 'yhat']].tail(365)

prophet\_output.columns = ['Date', 'prophet']

return prophet\_output

# Multiple Regressor

def regressor\_prediction(file\_path):

df\_regressor = pd.read\_csv(file\_path)

df\_regressor['Date'] = pd.to\_datetime(df\_regressor['Date'])

df\_regressor['Day'] = df\_regressor['Date'].dt.day

df\_regressor['Month'] = df\_regressor['Date'].dt.month

df\_regressor['Year'] = df\_regressor['Date'].dt.year

df\_regressor.set\_index('Date', inplace=True)

X\_regressor = df\_regressor[['Day', 'Month', 'Year']]

y\_regressor = df\_regressor['Sales']

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

model\_regressor = LinearRegression()

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

future\_dates\_regressor = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

future\_features\_regressor = pd.DataFrame({

'Day': future\_dates\_regressor.day,

'Month': future\_dates\_regressor.month,

'Year': future\_dates\_regressor.year

})

future\_predictions\_regressor = model\_regressor.predict(future\_features\_regressor)

future\_predictions\_df\_regressor = pd.DataFrame({

'Date': future\_dates\_regressor,

'Mul\_Reg': future\_predictions\_regressor

})

return future\_predictions\_df\_regressor

# Decision Tree

def decision\_tree\_prediction(file\_path):

df\_tree = pd.read\_csv(file\_path)

df\_tree['Date'] = pd.to\_datetime(df\_tree['Date'])

df\_tree['Day'] = df\_tree['Date'].dt.day

df\_tree['Month'] = df\_tree['Date'].dt.month

df\_tree['Year'] = df\_tree['Date'].dt.year

df\_tree.set\_index('Date', inplace=True)

X\_tree = df\_tree[['Day', 'Month', 'Year']]

y\_tree = df\_tree['Sales']

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

model\_tree = DecisionTreeRegressor()

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

future\_dates\_tree = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

future\_features\_tree = pd.DataFrame({

'Day': future\_dates\_tree.day,

'Month': future\_dates\_tree.month,

'Year': future\_dates\_tree.year

})

future\_predictions\_tree = model\_tree.predict(future\_features\_tree)

future\_predictions\_df\_tree = pd.DataFrame({

'Date': future\_dates\_tree,

'Decision\_tree': future\_predictions\_tree

})

return future\_predictions\_df\_tree

# K-Means

def kmeans\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['Day'] = df['Date'].dt.day

df['Month'] = df['Date'].dt.month

df['Year'] = df['Date'].dt.year

df.set\_index('Date', inplace=True)

X = df[['Day', 'Month', 'Year']]

y = df['Sales']

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

model = KNeighborsRegressor(n\_neighbors=5)

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

future\_dates = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

future\_features = pd.DataFrame({

'Day': future\_dates.day,

'Month': future\_dates.month,

'Year': future\_dates.year

})

future\_predictions\_kmeans = model.predict(future\_features)

future\_predictions\_df\_kmeans = pd.DataFrame({

'Date': future\_dates,

'K\_Means': future\_predictions\_kmeans

})

return future\_predictions\_df\_kmeans

# Logistic Regression

def logreg\_prediction(file\_path):

sales\_data = pd.read\_csv(file\_path)

X = pd.to\_numeric(pd.to\_datetime(sales\_data['Date'])).values.reshape(-1, 1)

y = sales\_data['Sales']

scaler = StandardScaler()

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

model = LinearRegression()

model.fit(X\_scaled, y)

new\_dates = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

new\_dates\_numeric = pd.to\_numeric(new\_dates).values.reshape(-1, 1)

new\_dates\_scaled = scaler.transform(new\_dates\_numeric)

new\_sales\_pred = model.predict(new\_dates\_scaled)

predictions\_df\_logreg = pd.DataFrame({'Date': new\_dates, 'Log\_Reg': new\_sales\_pred})

return predictions\_df\_logreg

def svr\_prediction(file\_path):

sales\_data = pd.read\_csv(file\_path)

X = pd.to\_numeric(pd.to\_datetime(sales\_data['Date'])).values.reshape(-1, 1)

y = sales\_data['Sales']

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

scaler = StandardScaler()

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

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

# Support Vector Regression model

model = SVR()

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

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

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

print(f"Mean Squared Error: {mse}")

new\_dates = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

new\_dates\_numeric = pd.to\_numeric(new\_dates).values.reshape(-1, 1)

new\_dates\_scaled = scaler.transform(new\_dates\_numeric)

# Predict sales for the new dates

new\_sales\_pred = model.predict(new\_dates\_scaled)

predictions\_df\_svr = pd.DataFrame({'Date': new\_dates, 'SVR': new\_sales\_pred})

return predictions\_df\_svr

def gbm\_prediction(file\_path):

sales\_data = pd.read\_csv(file\_path)

X = pd.to\_numeric(pd.to\_datetime(sales\_data['Date'])).values.reshape(-1, 1)

y = sales\_data['Sales']

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

scaler = StandardScaler()

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

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

# Gradient Boosting Regressor model

model = GradientBoostingRegressor()

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

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

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

print(f"Mean Squared Error: {mse}")

# Predict sales for the next year

new\_dates = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

new\_dates\_numeric = pd.to\_numeric(new\_dates).values.reshape(-1, 1)

new\_dates\_scaled = scaler.transform(new\_dates\_numeric)

new\_sales\_pred = model.predict(new\_dates\_scaled)

predictions\_df\_gbm = pd.DataFrame({'Date': new\_dates, 'GBM': new\_sales\_pred})

return predictions\_df\_gbm

# Function to perform sales predictions using GRU

#def gru\_prediction(file\_path):

# Read data from CSV file

#df = pd.read\_csv(file\_path)

# Assuming you have a 'Date' column in your data

# Convert 'Date' column to datetime format

#df['Date'] = pd.to\_datetime(df['Date'])

# Set 'Date' column as the index of the DataFrame

# df.set\_index('Date', inplace=True)

# Extract the 'Sales' column as a NumPy array

#sales\_data = df['Sales'].values.reshape(-1, 1)

# Normalize the data

#scaler = MinMaxScaler()

#sales\_data\_normalized = scaler.fit\_transform(sales\_data)

# Define a function to create input sequences for the GRU model

#def create\_sequences(data, seq\_length):

#sequences = []

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

# seq = data[i:i + seq\_length, 0]

#label = data[i + seq\_length, 0]

# sequences.append((seq, label))

# return np.array(sequences)

# Set the sequence length (adjust as needed)

# sequence\_length = 10

# Create input sequences and labels

#sequences = create\_sequences(sales\_data\_normalized, sequence\_length)

# Split the data into training and testing sets

# train\_size = int(len(sequences) \* 0.8)

# train\_sequences, test\_sequences = sequences[:train\_size], sequences[train\_size:]

# X\_train = np.array([seq[0] for seq in train\_sequences])

# y\_train = np.array([seq[1] for seq in train\_sequences])

# X\_test = np.array([seq[0] for seq in test\_sequences])

# y\_test = np.array([seq[1] for seq in test\_sequences])

# Build the GRU model

#model = Sequential([

# GRU(units=50, activation='tanh', input\_shape=(X\_train.shape[1], 1)),

#Dense(units=1)

#])

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

# Train the model

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

# Predict using the trained model for the year 2024

#future\_dates = pd.date\_range(start='2024-01-01', end='2024-12-31', freq='D')

#future\_data\_normalized = scaler.transform(sales\_data[-sequence\_length:])

#predicted\_sales = []

#for i in range(len(future\_dates)):

# X\_future = future\_data\_normalized.reshape((1, sequence\_length, 1))

# predicted\_sales\_normalized = model.predict(X\_future)

#predicted\_sales.append(predicted\_sales\_normalized[0, 0])

#future\_data\_normalized = np.append(future\_data\_normalized[1:], predicted\_sales\_normalized.reshape(1, 1))

# Invert the normalization

#predicted\_sales = scaler.inverse\_transform(np.array(predicted\_sales).reshape(-1, 1))

# Convert the predicted sales array to a DataFrame with dates

#predicted\_sales\_df\_GRU = pd.DataFrame(data={'Date': future\_dates, 'GRU': predicted\_sales.flatten()})

#return predicted\_sales\_df\_GRU

def ridge\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['Days'] = (df['Date'] - df['Date'].min()).dt.days

X = df[['Days']]

y = df['Sales']

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

scaler = StandardScaler()

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

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

# Initialize Ridge Regression model

ridge\_model = Ridge(alpha=1.0)

ridge\_model.fit(X\_train\_scaled, y\_train)

y\_pred = ridge\_model.predict(X\_test\_scaled)

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

print(f'Mean Squared Error on Test Set: {mse}')

last\_date = df['Date'].max()

next\_year\_dates = pd.date\_range(last\_date + timedelta(days=1), periods=365, freq='D')

X\_next\_year = pd.DataFrame({'Days': (next\_year\_dates - df['Date'].min()).days})

X\_next\_year\_scaled = scaler.transform(X\_next\_year)

predictions\_next\_year = ridge\_model.predict(X\_next\_year\_scaled)

next\_year\_predictions\_df\_ridge = pd.DataFrame({'Date': next\_year\_dates, 'Ridge\_reg': predictions\_next\_year})

return next\_year\_predictions\_df\_ridge

# 10. Lasso Regression

def lasso\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['Days'] = (df['Date'] - df['Date'].min()).dt.days

X = df[['Days']]

y = df['Sales']

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

scaler = StandardScaler()

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

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

lasso\_model = Lasso(alpha=0.01)

lasso\_model.fit(X\_train\_scaled, y\_train)

y\_pred = lasso\_model.predict(X\_test\_scaled)

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

print(f'Mean Squared Error on Test Set: {mse}')

last\_date = df['Date'].max()

next\_year\_dates = pd.date\_range(last\_date + timedelta(days=1), periods=365, freq='D')

X\_next\_year = pd.DataFrame({'Days': (next\_year\_dates - df['Date'].min()).days})

X\_next\_year\_scaled = scaler.transform(X\_next\_year)

predictions\_next\_year = lasso\_model.predict(X\_next\_year\_scaled)

next\_year\_predictions\_df\_lasso = pd.DataFrame({'Date': next\_year\_dates, 'Lasso\_reg': predictions\_next\_year})

return next\_year\_predictions\_df\_lasso

# 11. SVM

def svm\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

svr\_model = SVR(kernel='rbf')

svr\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = svr\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_SVM = pd.DataFrame({'Date': next\_year\_dates, 'SVM': sales\_predictions})

return predictions\_df\_SVM

# 12. MLP Regression

def mlp\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

mlp\_model = MLPRegressor(hidden\_layer\_sizes=(100,), max\_iter=1000, random\_state=42)

mlp\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = mlp\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_MLP = pd.DataFrame({'Date': next\_year\_dates, 'MLP': sales\_predictions})

return predictions\_df\_MLP

# 13. ARD Regression

def ard\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

ard\_model = ARDRegression()

ard\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = ard\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_ARD = pd.DataFrame({'Date': next\_year\_dates, 'ARD': sales\_predictions})

return predictions\_df\_ARD

# 14. Stacking Regression

def stack\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

linear\_reg = LinearRegression()

mlp\_reg = MLPRegressor(hidden\_layer\_sizes=(100,), max\_iter=1000, random\_state=42)

svr\_reg = SVR(kernel='rbf')

stacked\_model = StackingRegressor(

estimators=[('linear', linear\_reg), ('mlp', mlp\_reg), ('svr', svr\_reg)],

final\_estimator=LinearRegression()

)

stacked\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = stacked\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_stack = pd.DataFrame({'Date': next\_year\_dates, 'STACK': sales\_predictions})

return predictions\_df\_stack

def linear\_reg\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

linear\_model = LinearRegression()

linear\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = linear\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_linear = pd.DataFrame({'Date': next\_year\_dates, 'Linear': sales\_predictions.flatten()})

return predictions\_df\_linear

def poisson\_reg\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

poisson\_model = PoissonRegressor()

poisson\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = poisson\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_poisson = pd.DataFrame({'Date': next\_year\_dates, 'Poisson': sales\_predictions})

return predictions\_df\_poisson

#def lstm\_prediction(file\_path):

#df = pd.read\_csv(file\_path)

# df['Date'] = pd.to\_datetime(df['Date'])

#df['NumericDate'] = df['Date'].dt.dayofyear

# data = df[['NumericDate', 'Sales']].values

# scaler = MinMaxScaler(feature\_range=(0, 1))

# data\_scaled = scaler.fit\_transform(data)

# SEQUENCE\_LENGTH = 10

# EPOCHS = 50

# BATCH\_SIZE = 32

# FUTURE\_STEPS = 365

# sequences = create\_sequences(data\_scaled, SEQUENCE\_LENGTH)

# train\_size = int(len(sequences) \* 0.8)

# train\_data = sequences[:train\_size]

# test\_data = sequences[train\_size:]

# X\_train, y\_train = train\_data[:, :-1, :], train\_data[:, -1, 1]

#X\_test, y\_test = test\_data[:, :-1, :], test\_data[:, -1, 1]

# model = Sequential()

# model.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

# model.add(LSTM(units=50))

# model.add(Dense(units=1))

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

# model.fit(X\_train, y\_train, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)

# future\_predictions = []

# current\_sequence = X\_test[-1]

# for \_ in range(FUTURE\_STEPS):

# current\_sequence\_reshaped = current\_sequence.reshape((1, SEQUENCE\_LENGTH - 1, 2))

# next\_prediction = model.predict(current\_sequence\_reshaped)

# future\_predictions.append(next\_prediction[0, 0])

# current\_sequence = np.append(current\_sequence[1:], [[next\_prediction[0, 0], 0]], axis=0)

# scaled\_future\_predictions = np.column\_stack((np.zeros\_like(future\_predictions), future\_predictions))

# scaled\_future\_predictions = scaler.inverse\_transform(scaled\_future\_predictions)[:, 1]

# future\_dates = pd.date\_range(start=df['Date'].max() + pd.DateOffset(days=1), periods=FUTURE\_STEPS, freq='D')

# predictions\_df\_lstm = pd.DataFrame({'Date': future\_dates, 'LSTM': scaled\_future\_predictions})

# return predictions\_df\_lstm

def gaussian\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

kernel = C(1.0, (1e-3, 1e3)) \* RBF(1.0, (1e-2, 1e2))

gp\_model = GaussianProcessRegressor(kernel=kernel, n\_restarts\_optimizer=10, random\_state=42)

gp\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions, sigma = gp\_model.predict(next\_year\_dates\_scaled, return\_std=True)

predictions\_df\_gaussian = pd.DataFrame({'Date': next\_year\_dates, 'Gaussian': sales\_predictions})

return predictions\_df\_gaussian

#def neural\_prediction(file\_path):

# df = pd.read\_csv(file\_path)

# df['Date'] = pd.to\_datetime(df['Date'])

# df['NumericDate'] = df['Date'].dt.dayofyear

# X = df['NumericDate'].values.reshape(-1, 1)

# y = df['Sales'].values

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

# scaler = StandardScaler()

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

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

# model = Sequential()

# model.add(Dense(64, activation='relu', input\_dim=1))

# model.add(Dense(32, activation='relu'))

# model.add(Dense(1, activation='linear'))

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

# model.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test\_scaled, y\_test))

# next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

# next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

# next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

# sales\_predictions = model.predict(next\_year\_dates\_scaled)

# predictions\_df\_neural = pd.DataFrame({'Date': next\_year\_dates, 'Neural\_networks': sales\_predictions.flatten()})

# return predictions\_df\_neural

def arima\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace=True)

model = ARIMA(df['Sales'], order=(5, 1, 2))

results = model.fit()

next\_month\_forecast = results.get\_forecast(steps=365)

next\_month\_index = pd.date\_range(start=df.index[-1] + timedelta(days=1), periods=365)

next\_month\_sales = next\_month\_forecast.predicted\_mean.values

rounded\_sales = pd.Series(next\_month\_sales).round().astype(int)

rounded\_sales\_df\_arima = pd.DataFrame({'Date': next\_month\_index, 'ARIMA': rounded\_sales})

return rounded\_sales\_df\_arima

def pls\_prediction(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df['NumericDate'] = df['Date'].dt.dayofyear

X = df['NumericDate'].values.reshape(-1, 1)

y = df['Sales'].values

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

scaler = StandardScaler()

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

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

linear\_model = LinearRegression()

linear\_model.fit(X\_train\_scaled, y\_train)

next\_year\_dates = pd.date\_range(df['Date'].max(), periods=365, freq='D')

next\_year\_numeric\_dates = next\_year\_dates.dayofyear.values.reshape(-1, 1)

next\_year\_dates\_scaled = scaler.transform(next\_year\_numeric\_dates)

sales\_predictions = linear\_model.predict(next\_year\_dates\_scaled)

predictions\_df\_PLS = pd.DataFrame({'Date': next\_year\_dates, 'PLS': sales\_predictions.flatten()})

return predictions\_df\_PLS

def sarima\_forecasting(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace=True)

order = (1, 1, 1)

seasonal\_order = (1, 1, 1, 12)

model\_sarima = SARIMAX(df['Sales'], order=order, seasonal\_order=seasonal\_order)

results\_sarima = model\_sarima.fit(disp=False)

next\_year\_forecast = results\_sarima.get\_forecast(steps=365)

next\_year\_index = pd.date\_range(start=df.index[-1] + timedelta(days=1), periods=365)

next\_year\_sales = next\_year\_forecast.predicted\_mean

rounded\_sales = pd.Series(next\_year\_sales).round().astype(int)

rounded\_sales\_df = pd.DataFrame({'Date': next\_year\_index, 'Sarima': rounded\_sales})

rounded\_sales\_df.to\_csv('sarima\_output.csv', index=False)

return rounded\_sales\_df

def exponential\_smoothing\_forecasting(file\_path):

df = pd.read\_csv(file\_path)

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace=True)

model = ExponentialSmoothing(df['Sales'], seasonal='add', seasonal\_periods=12)

result = model.fit()

next\_year\_start = df.index[-1] + pd.DateOffset(days=1)

next\_year\_end = next\_year\_start + pd.DateOffset(years=1)

expon = result.predict(start=next\_year\_start, end=next\_year\_end)

output\_df = pd.DataFrame({'Date': expon.index, 'Expo': expon.values})

print(output\_df)

output\_df.to\_csv("expo\_output.csv", index=False)

return output\_df

# Streamlit App

def main():

st.title("Sales Forecasting Dashboard")

# File path input

file\_path = st.text\_input('Enter the file path for sales data:', 'sales\_data.csv')

if st.button('Generate Predictions for one year'):

# Perform predictions using each model

prophet\_output = prophet\_prediction(file\_path)

regressor\_output = regressor\_prediction(file\_path)

decision\_tree\_output = decision\_tree\_prediction(file\_path)

kmeans\_output = kmeans\_prediction(file\_path)

logreg\_output = logreg\_prediction(file\_path)

svr\_output=svr\_prediction(file\_path)

gbm\_output=gbm\_prediction(file\_path)

# gru\_output=gru\_prediction(file\_path)

lasso\_output=lasso\_prediction(file\_path)

svm\_output=svm\_prediction(file\_path)

mlp\_output=mlp\_prediction(file\_path)

ard\_output=ard\_prediction(file\_path)

stack\_output=stack\_prediction(file\_path)

linear\_output=linear\_reg\_prediction(file\_path)

poisson\_output=poisson\_reg\_prediction(file\_path)

#lstm\_output=lstm\_prediction(file\_path)

gaussian\_output=gaussian\_prediction(file\_path)

#neural\_output=neural\_prediction(file\_path)

arima\_output=arima\_prediction(file\_path)

pls\_output= pls\_prediction(file\_path)

sarima\_output=sarima\_forecasting(file\_path)

expo\_output=exponential\_smoothing\_forecasting(file\_path)

# Combine all results into a single dataframe

final\_results = pd.merge(prophet\_output, regressor\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, decision\_tree\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, kmeans\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, logreg\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, svr\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, gbm\_output, on='Date', how='outer')

#final\_results = pd.merge(final\_results, gru\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, lasso\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, svm\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, mlp\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, ard\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, stack\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, linear\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, poisson\_output, on='Date', how='outer')

# final\_results = pd.merge(final\_results, lstm\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, gaussian\_output, on='Date', how='outer')

# final\_results = pd.merge(final\_results, neural\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, arima\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, pls\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, sarima\_output, on='Date', how='outer')

final\_results = pd.merge(final\_results, expo\_output, on='Date', how='outer')

final\_results = final\_results.round(0)

# Display results in a single table

st.write("Sales Predictions:")

st.write(final\_results.set\_index('Date'))

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Screenshots :**









