Source code</>

App.py –

import streamlit as st

import pandas as pd

import joblib

# Load the model

best\_model = joblib.load('sales\_forecasting\_pipeline.pkl')

# UI elements

st.title("Sales Forecasting for Furniture Store")

# Collect input data from user

ship\_mode = st.selectbox('Ship Mode', ['Second Class', 'First Class', 'Standard Class', 'Same Day'])

segment = st.selectbox('Segment', ['Consumer', 'Corporate', 'Home Office'])

country = st.text\_input('Country', 'United States')

city = st.text\_input('City', 'New York')

state = st.text\_input('State', 'New York')

region = st.selectbox('Region', ['East', 'West', 'Central', 'South'])

category = st.selectbox('Category', ['Furniture', 'Office Supplies', 'Technology'])

sub\_category = st.text\_input('Sub-Category', 'Chairs')

sales = st.number\_input('Sales', 0.0)

quantity = st.number\_input('Quantity', 1)

discount = st.slider('Discount', 0.0, 1.0)

profit = st.number\_input('Profit', 0.0)

order\_year = st.number\_input('Order Year', 2023)

order\_month = st.number\_input('Order Month', 8)

order\_day = st.number\_input('Order Day', 15)

# Prepare the input data as a DataFrame

new\_data = pd.DataFrame({

'Ship Mode': [ship\_mode],

'Segment': [segment],

'Country': [country],

'City': [city],

'State': [state],

'Region': [region],

'Category': [category],

'Sub-Category': [sub\_category],

'Sales': [sales],

'Quantity': [quantity],

'Discount': [discount],

'Profit': [profit],

'Order Year': [order\_year],

'Order Month': [order\_month],

'Order Day': [order\_day]

})

# Make predictions when the user clicks the button

if st.button('Predict Sales'):

prediction = best\_model.predict(new\_data)

st.write(f"Predicted Sales: {prediction[0]:.2f}")

from flask import Flask, request, jsonify, render\_template

import pandas as pd

import joblib

app = Flask(\_\_name\_\_)

# Load the model

best\_model = joblib.load('sales\_forecasting\_pipeline.pkl')

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

data = request.form.to\_dict()

new\_data = pd.DataFrame([data])

prediction = best\_model.predict(new\_data)

return jsonify({'prediction': prediction[0]})

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

from flask import Flask, request, jsonify, render\_template

import pandas as pd

import joblib

app = Flask(\_\_name\_\_)

# Load the model

best\_model = joblib.load('sales\_forecasting\_pipeline.pkl')

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

data = request.form.to\_dict()

new\_data = pd.DataFrame([data])

prediction = best\_model.predict(new\_data)

return jsonify({'prediction': prediction[0]})

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**Jupityper file –**

import numpy as np

from matplotlib import pyplot as plt

import pandas as pd

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

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestRegressor

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

# Load the dataset with the correct encoding

df = pd.read\_csv('/content/Super\_Store\_data.csv', encoding='ISO-8859-1')

df.head()

#data preprocessing

#Handling missing data (if any)

df.fillna(method='ffill',inplace = True)

#feature Engineering

#Extracting date components

df['Order Year'] = pd.to\_datetime(df['Order Date']).dt.year

df['Order Month'] = pd.to\_datetime(df['Order Date']).dt.month

df['Order Day'] = pd.to\_datetime(df['Order Date']).dt.day

#Selecting Features and Target Variables

features = ['Ship Mode', 'Segment', 'Country', 'City', 'State', 'Region', 'Category', 'Sub-Category',

'Sales', 'Quantity', 'Discount', 'Profit', 'Order Year', 'Order Month', 'Order Day']

target = 'Sales'

X = df[features]

y = df[target]

# Splitting the data to training and testing sets

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

# Preprocessing the pipeline for categorical and numerical values

numerical\_features = ['Sales', 'Quantity', 'Discount', 'Profit', 'Order Year', 'Order Month', 'Order Day']

categorical\_features = ['Ship Mode', 'Segment', 'Country', 'City', 'State', 'Region', 'Category', 'Sub-Category']

numerical\_transformer = StandardScaler()

categorical\_transformer = OneHotEncoder(handle\_unknown='ignore')

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_transformer, numerical\_features),

('cat', categorical\_transformer,categorical\_features)

])

# Model pipeline

model = Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', RandomForestRegressor(random\_state=42))

])

# Hyperparameter tuning using GridSearchCV

param\_grid = {

'regressor\_\_n\_estimators': [100, 200],

'regressor\_\_max\_depth': [10, 20, None],

'regressor\_\_min\_samples\_split': [2, 5, 10]

}

grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X\_train, y\_train)

#best Model

best\_model = grid\_search.best\_estimator\_

#Prediction

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

#evaluate the model

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

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

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

print(f'R2 Score: {r2}')

#feature importance

feature\_importances = best\_model.named\_steps['regressor'].feature\_importances\_

feature\_names = numerical\_features + list(grid\_search.best\_estimator\_.named\_steps['preprocessor'].transformers\_[1][1].get\_feature\_names\_out(categorical\_features))

importance\_df = pd.DataFrame({

'Feature': feature\_names,

'Importance': feature\_importances

}).sort\_values(by='Importance', ascending = False)

print(importance\_df)

#cross validation scores

cv\_results = pd.DataFrame(grid\_search.cv\_results\_)

print(cv\_results[['mean\_test\_score','params']])

# Saving the model

import joblib

joblib.dump(best\_model, 'sales\_forecasting\_model.pkl')

# Save the preprocessor after fitting

joblib.dump(preprocessor, 'preprocessor.pkl')

# Save the entire pipeline (preprocessor + model)

joblib.dump(best\_model, 'sales\_forecasting\_pipeline.pkl')

# Make predictions on new data

def predict\_new\_data(new\_data):

# Use the saved pipeline to make predictions

predictions = best\_model.predict(new\_data)

return predictions

# Load the saved pipeline for future use

best\_model = joblib.load('sales\_forecasting\_pipeline.pkl')

# Example new data (ensure it has the same features as the original training data)

new\_data = pd.DataFrame({

'Ship Mode': ['Second Class', 'First Class'],

'Segment': ['Consumer', 'Corporate'],

'Country': ['United States', 'United States'],

'City': ['New York', 'Los Angeles'],

'State': ['New York', 'California'],

'Region': ['East', 'West'],

'Category': ['Furniture', 'Office Supplies'],

'Sub-Category': ['Chairs', 'Binders'],

'Sales': [500.0, 200.0],

'Quantity': [3, 2],

'Discount': [0.1, 0.2],

'Profit': [50.0, 20.0],

'Order Year': [2023, 2023],

'Order Month': [8, 8],

'Order Day': [15, 16]

})

# Make predictions on the new data

predictions = predict\_new\_data(new\_data)

# Output the predictions

print(predictions)