import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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.impute import SimpleImputer

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.tree import DecisionTreeRegressor, plot\_tree

import xgboost as xgb

from sklearn.svm import SVR

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

data = pd.read\_csv('/content/laptop.csv')

data = data.drop(columns=['Unnamed: 0.1', 'Unnamed: 0'], errors='ignore')

print(data.head())

print(data.columns)

print(data.info())

print(data.describe())

target\_column = 'Price'  # Update this if needed based on the correct column name from the dataset

categorical\_features = data.select\_dtypes(include=['object']).columns.tolist()

print(categorical\_features)

# Visualize the distribution of the target variable (price)

plt.figure(figsize=(10, 6))

sns.histplot(data[target\_column], kde=True, color='skyblue')

plt.title('Distribution of Laptop Prices')

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.show()

# Visualize correlations between numerical features and price

numerical\_data = data.select\_dtypes(include=['int64', 'float64'])

plt.figure(figsize=(12, 8))

sns.heatmap(numerical\_data.corr(), annot=True, cmap='viridis', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap', fontsize=16)

plt.show()

# Additional Visualizations

# 1. Box plot of prices by brand

if 'Brand' in data.columns:

    plt.figure(figsize=(14, 8))

    sns.boxplot(x='Brand', y=target\_column, data=data, palette='Set3')

    plt.xticks(rotation=90)

    plt.title('Laptop Prices by Brand')

    plt.xlabel('Brand')

    plt.ylabel('Price')

    plt.show()

# 2. Pairplot of numerical features

sns.pairplot(numerical\_data)

plt.show()

# 3. Violin plot of prices by processor

if 'Processor' in data.columns:

    plt.figure(figsize=(14, 8))

    sns.violinplot(x='Processor', y=target\_column, data=data, palette='Set2')

    plt.xticks(rotation=90)

    plt.title('Laptop Prices by Processor')

    plt.xlabel('Processor')

    plt.ylabel('Price')

    plt.show()

# 4. Bar plot of average prices by operating system

if 'OS' in data.columns:

    plt.figure(figsize=(12, 6))

    avg\_price\_by\_os = data.groupby('OS')[target\_column].mean().sort\_values()

    sns.barplot(x=avg\_price\_by\_os.index, y=avg\_price\_by\_os.values, palette='Set1')

    plt.xticks(rotation=90)

    plt.title('Average Laptop Prices by Operating System')

    plt.xlabel('Operating System')

    plt.ylabel('Average Price')

    plt.show()

# Identify numerical and categorical columns

numerical\_features = data.select\_dtypes(include=['int64', 'float64']).columns.tolist()

numerical\_features.remove(target\_column)

categorical\_features = data.select\_dtypes(include=['object']).columns.tolist()

# Handle missing values in the target column

data = data.dropna(subset=[target\_column])

# Define preprocessing steps for numerical and categorical data

numerical\_transformer = Pipeline(steps=[

    ('imputer', SimpleImputer(strategy='median')),

    ('scaler', StandardScaler())

])

categorical\_transformer = Pipeline(steps=[

    ('imputer', SimpleImputer(strategy='most\_frequent')),

    ('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

# Combine preprocessing steps

preprocessor = ColumnTransformer(

    transformers=[

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

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

    ])

models = {

    'Linear Regression': LinearRegression(),

    'Random Forest': RandomForestRegressor(random\_state=42),

    'Gradient Boosting': GradientBoostingRegressor(random\_state=42),

    'XGBoost': xgb.XGBRegressor(random\_state=42),

    'Decision Tree': DecisionTreeRegressor(random\_state=42),

    'Support Vector Machine': SVR()

}

X = data.drop(columns=[target\_column])

y = data[target\_column]

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

results = {}

for name, model in models.items():

    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])

    pipeline.fit(X\_train, y\_train)

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

    results[name] = {

        'MAE': mean\_absolute\_error(y\_test, y\_pred),

        'MSE': mean\_squared\_error(y\_test, y\_pred),

        'R2': r2\_score(y\_test, y\_pred),

        'Accuracy (%)': r2\_score(y\_test, y\_pred) \* 100

    }

# Display results

results\_df = pd.DataFrame(results).T

print(results\_df)

param\_grid = {

    'model\_\_n\_estimators': [100, 200, 300],

    'model\_\_max\_depth': [None, 10, 20, 30]

}

pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', RandomForestRegressor(random\_state=42))])

grid\_search = GridSearchCV(pipeline, param\_grid, cv=3, scoring='r2', n\_jobs=-1)

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

# Best model and parameters

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

print(f"Best Parameters: {grid\_search.best\_params\_}")

print(f"Best Cross-Validation R2 Score: {grid\_search.best\_score\_}")

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

print('Test Set Performance:')

print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred\_best)}")

print(f"MSE: {mean\_squared\_error(y\_test, y\_pred\_best)}")

print(f"R2: {r2\_score(y\_test, y\_pred\_best)}")

print(f"Accuracy (%): {r2\_score(y\_test, y\_pred\_best) \* 100:.2f}")

if hasattr(best\_model.named\_steps['model'], 'feature\_importances\_'):

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

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

    feature\_importance\_df = pd.DataFrame({

        'Feature': feature\_names,

        'Importance': feature\_importances

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

    plt.figure(figsize=(12, 8))

    sns.barplot(x='Importance', y='Feature', data=feature\_importance\_df, palette='viridis')

    plt.title('Feature Importances')

    plt.show()

if 'Decision Tree' in models:

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

    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', dt\_model)])

    pipeline.fit(X\_train, y\_train)

    # Extract feature names after preprocessing

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

    plt.figure(figsize=(20, 20))

    plot\_tree(pipeline.named\_steps['model'], feature\_names=feature\_names, filled=True, rounded=True, fontsize=15)

    plt.title('Decision Tree Visualization')

    plt.show()

def predict\_price(new\_data):

    """

    Function to predict laptop price for new data

    Args:

    new\_data (pd.DataFrame): DataFrame containing the new laptop data with the same features as the training data

    Returns:

    float: Predicted laptop price

    """

    return best\_model.predict(new\_data)

# Check the column names in the training data

print(X.columns)

# Example of predicting price for a new laptop

new\_laptop = pd.DataFrame([{

    'Company': 'NewBrand',

    'TypeName': 'Ultrabook',

    'Inches': 15.6,

    'ScreenResolution': '1920x1080',

    'Cpu': 'Intel Core i7',

    'Ram': '16GB',

    'Memory': '512GB SSD',

    'Gpu': 'NVIDIA GTX 1650',

    'OpSys': 'Windows 10',

    'Weight': '2.5kg'

}])

predicted\_price = predict\_price(new\_laptop)

print(f"Predicted Price for the new laptop: ${predicted\_price[0]:.2f}")