import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, learning\_curve

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.linear\_model import LinearRegression

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

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import KNNImputer

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import plotly.graph\_objects as go

from scipy import stats

try:

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

statsmodels\_available = True

except ImportError:

statsmodels\_available = False

import warnings

warnings.filterwarnings('ignore')

# Set random seed for reproducibility

np.random.seed(42)

# 1. Load the dataset

def load\_data(url):

try:

df = pd.read\_csv (url)

print("Data loaded successfully.")

print("\nDataset Column Types:")

print(df.dtypes)

return df

except Exception as e:

print(f"Error loading data: {e}")

return None

url = "https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/housing/housing.csv"

df = load\_data(url)

if df is None:

raise SystemExit("Exiting due to data loading failure.")

# Subsample dataset for faster training (comment out to use full dataset)

df = df.sample(frac=0.3, random\_state=42)

print(f"\nSubsampled dataset to {len(df)} rows for faster training.")

# 2. Data Preprocessing

def preprocess\_data(df):

print("\n=== Data Preprocessing ===")

# Display missing values before imputation

print("Missing Values Before Imputation:")

print(df.isnull().sum())

# Separate categorical and numerical columns

categorical\_cols = ['ocean\_proximity'] if 'ocean\_proximity' in df.columns else []

numerical\_cols = [col for col in df.columns if col not in categorical\_cols]

# Debug: Confirm columns

print("\nNumerical Columns for Imputation:", numerical\_cols)

print("Categorical Columns:", categorical\_cols)

# Handle missing values with KNN imputation for numerical columns

try:

if numerical\_cols:

imputer = KNNImputer(n\_neighbors=5)

df[numerical\_cols] = pd.DataFrame(

imputer.fit\_transform(df[numerical\_cols]),

columns=numerical\_cols,

index=df.index

)

except Exception as e:

print(f"Error during KNN imputation: {e}")

print("Falling back to median imputation.")

for col in numerical\_cols:

df[col].fillna(df[col].median(), inplace=True)

# Display missing values after imputation

print("\nMissing Values After Imputation:")

print(df.isnull().sum())

# Remove duplicates

initial\_rows = len(df)

df.drop\_duplicates(inplace=True)

print(f"\nDuplicates Removed: {initial\_rows - len(df)}")

# Cap outliers for numerical columns and track counts

outlier\_counts = {}

def cap\_outliers(series, col\_name):

Q1 = series.quantile(0.25)

Q3 = series.quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = ((series < lower\_bound) | (series > upper\_bound)).sum()

outlier\_counts[col\_name] = outliers

return series.clip(lower\_bound, upper\_bound)

for col in numerical\_cols:

df[col] = cap\_outliers(df[col], col)

print("\nOutliers Detected and Capped:")

for col, count in outlier\_counts.items():

print(f"{col}: {count} outliers")

# Display skewness before log-transformation

print("\nSkewness Before Log-Transformation:")

print(df[numerical\_cols].skew())

# Log-transform skewed features

skewed\_cols = ['total\_rooms', 'population', 'median\_house\_value']

for col in skewed\_cols:

if col in df.columns:

df[col] = np.log1p(df[col])

# Display skewness after log-transformation

print("\nSkewness After Log-Transformation:")

print(df[numerical\_cols].skew())

# Verify median\_house\_value presence

print("\nColumns After Preprocessing:", df.columns.tolist())

return df

df = preprocess\_data(df)

# 3. Feature Engineering

def engineer\_features(df):

print("\n=== Feature Engineering ===")

# Add core features (reduced set)

df['rooms\_per\_household'] = df['total\_rooms'] / df['households']

df['bedrooms\_per\_room'] = df['total\_bedrooms'] / df['total\_rooms']

df['population\_per\_household'] = df['population'] / df['households']

# Removed 'distance\_to\_coast' and 'median\_income\_poly1' to reduce feature count

# Summarize new features

print("New Features Created:")

print(df[['rooms\_per\_household', 'bedrooms\_per\_room', 'population\_per\_household']].head())

return df

df = engineer\_features(df)

# 4. Statistical Analysis

def statistical\_analysis(df):

print("\n=== Statistical Analysis ===")

stat, p\_value = stats.shapiro(df['median\_house\_value'])

normality\_result = f"Shapiro-Wilk Test for median\_house\_value: p-value = {p\_value:.4f}"

# Filter numerical columns for descriptive stats, skewness, and kurtosis

numerical\_cols = df.select\_dtypes(include=['float64', 'int64']).columns

print("\nNumerical Columns for Statistical Analysis:", numerical\_cols.tolist())

desc\_stats = df[numerical\_cols].describe().T

desc\_stats['skewness'] = df[numerical\_cols].skew()

desc\_stats['kurtosis'] = df[numerical\_cols].kurtosis()

# Multcollinearity analysis (VIF)

if statsmodels\_available:

numerical\_df = df[numerical\_cols]

vif\_data = pd.DataFrame()

vif\_data['Feature'] = numerical\_df.columns

vif\_data['VIF'] = [variance\_inflation\_factor(numerical\_df.values, i) for i in range(numerical\_df.shape[1])]

print("\nVariance Inflation Factor (VIF) Analysis:")

print(vif\_data)

return normality\_result, desc\_stats

normality\_result, desc\_stats = statistical\_analysis(df)

print("\nNormality Test:")

print(normality\_result)

print("\nDescriptive Statistics with Skewness and Kurtosis:")

print(desc\_stats)

# 5. Exploratory Data Analysis (EDA)

def perform\_eda(df):

print("\n=== Exploratory Data Analysis ===")

# Correlation matrix

corr\_matrix = df.select\_dtypes(include=['float64', 'int64']).corr()

fig\_corr = go.Figure(data=go.Heatmap(

z=corr\_matrix.values,

x=corr\_matrix.columns,

y=corr\_matrix.columns,

colorscale='RdBu',

zmin=-1, zmax=1,

text=corr\_matrix.values.round(2),

texttemplate="%{text}",

textfont={"size": 10}

))

fig\_corr.update\_layout(title='Interactive Correlation Matrix', width=800, height=800)

fig\_corr.show()

# Scatter plot (fixed syntax error)

fig\_scatter = px.scatter(df, x='median\_income', y='median\_house\_value', title='Median Income vs Median House Value (USD)',

hover\_data=['longitude', 'latitude'],

trendline='ols')

fig\_scatter.update\_yaxes(title\_text='Median House Value (USD)')

fig\_scatter.update\_traces(hovertemplate='Income: %{x}House Value: $%{y:,.2f} USD')

fig\_scatter.show()

# Geographical plot (requires Mapbox token)

px.set\_mapbox\_access\_token('your\_mapbox\_token')

fig\_geo = px.scatter\_mapbox(df, lat='latitude', lon='longitude', color='median\_house\_value',

size='population', zoom=5, mapbox\_style='open-street-map',

title='House Prices by Location (USD)',

color\_continuous\_scale=px.colors.sequential.Plasma)

fig\_geo.update\_coloraxes(colorbar\_title='Median House Value (USD)')

fig\_geo.show()

# Distribution plot

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

sns.histplot(df['median\_house\_value'], kde=True)

plt.title('Distribution of Median House Value (USD)')

plt.xlabel('Median House Value (USD)')

plt.show()

perform\_eda(df)

# 6. Prepare Data for Modeling

def prepare\_data(df):

X = df.drop('median\_house\_value', axis=1)

y = df['median\_house\_value']

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

return X\_train, X\_test, y\_train, y\_test

X\_train, X\_test, y\_train, y\_test = prepare\_data(df)

# 7. Modeling and Impact Analysis

def build\_pipeline(model):

numerical\_cols = X\_train.select\_dtypes(include=['float64', 'int64']).columns

categorical\_cols = ['ocean\_proximity'] if 'ocean\_proximity' in df.columns else []

preprocessor = ColumnTransformer([

('num', StandardScaler(), numerical\_cols),

('cat', OneHotEncoder(drop='first', handle\_unknown='ignore'), categorical\_cols)

])

pipeline = Pipeline([

('preprocessor', preprocessor),

('regressor', model)

])

return pipeline

def plot\_feature\_importance(model, feature\_names):

importances = model.feature\_importances\_

indices = np.argsort(importances)[::-1]

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

plt.bar(range(len(importances)), importances[indices], align='center')

plt.xticks(range(len(importances)), [feature\_names[i] for i in indices], rotation=90)

plt.title('Feature Importance')

plt.tight\_layout()

plt.show()

def plot\_learning\_curves(model, X\_train, y\_train):

train\_sizes, train\_scores, val\_scores = learning\_curve(

model, X\_train, y\_train, cv=3, scoring='r2', n\_jobs=-1, train\_sizes=np.linspace(0.1, 1.0, 5))

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

val\_mean = np.mean(val\_scores, axis=1)

val\_std = np.std(val\_scores, axis=1)

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

plt.plot(train\_sizes, train\_mean, label='Training R²')

plt.plot(train\_sizes, val\_mean, label='Validation R²')

plt.fill\_between(train\_sizes, train\_mean - train\_std, train\_mean + train\_std, alpha=0.1)

plt.fill\_between(train\_sizes, val\_mean - val\_std, val\_mean + val\_std, alpha=0.1)

plt.xlabel('Training Examples')

plt.ylabel('R² Score')

plt.title('Learning Curves')

plt.legend(loc='best')

plt.grid(True)

plt.show()

def train\_and\_evaluate(X\_train, X\_test, y\_train, y\_test):

print("\n=== Model Training and Evaluation ===")

# Define models

models = {

'Linear Regression': LinearRegression(),

'Random Forest': RandomForestRegressor(n\_estimators=50, max\_depth=10, random\_state=42, n\_jobs=-1),

'Gradient Boosting': GradientBoostingRegressor(n\_estimators=50, random\_state=42)

}

results = []

best\_model = None

best\_r2 = -np.inf

numerical\_cols = X\_train.select\_dtypes(include=['float64', 'int64']).columns

categorical\_cols = ['ocean\_proximity'] if 'ocean\_proximity' in df.columns else []

feature\_names = numerical\_cols.tolist() + [f"ocean\_proximity\_{cat}" for cat in OneHotEncoder(drop='first').fit(X\_train[categorical\_cols]).get\_feature\_names\_out()] if categorical\_cols else numerical\_cols.tolist()

for name, model in models.items():

print(f"\nTraining {name}...")

pipeline = build\_pipeline(model)

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

# Predictions

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

# Evaluation metrics (convert back to USD)

y\_test\_usd = np.expm1(y\_test)

y\_pred\_usd = np.expm1(y\_pred)

mae = mean\_absolute\_error(y\_test\_usd, y\_pred\_usd)

rmse = np.sqrt(mean\_squared\_error(y\_test\_usd, y\_pred\_usd))

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

results.append({

'Model': name,

'MAE (USD)': mae,

'RMSE (USD)': rmse,

'R²': r2

})

print(f"\n{name} Performance (in USD):")

print(f"Mean Absolute Error: ${mae:,.2f}")

print(f"Root Mean Squared Error: ${rmse:,.2f}")

print(f"R²: {r2:.4f}")

# Feature importance (for Random Forest and Gradient Boosting)

if name in ['Random Forest', 'Gradient Boosting']:

plot\_feature\_importance(pipeline.named\_steps['regressor'], feature\_names)

# Learning curves

plot\_learning\_curves(pipeline, X\_train, y\_train)

# Residual analysis

residuals = y\_test\_usd - y\_pred\_usd

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

stats.probplot(residuals, dist="norm", plot=plt)

plt.title(f'Q-Q Plot of Residuals ({name})')

plt.show()

# Update best model

if r2 > best\_r2:

best\_r2 = r2

best\_model = pipeline

# Display model comparison

print("\n=== Model Comparison ===")

results\_df = pd.DataFrame(results)

print(results\_df)

return best\_model

best\_model = train\_and\_evaluate(X\_train, X\_test, y\_train, y\_test)

# 8. Gradio Interface for Hugging Face Spaces

try:

import gradio as gr

def predict\_house\_value(longitude, latitude, housing\_median\_age, total\_rooms, total\_bedrooms,

population, households, median\_income, ocean\_proximity):

input\_data = pd.DataFrame({

'longitude': [longitude],

'latitude': [latitude],

'housing\_median\_age': [housing\_median\_age],

'total\_rooms': [np.log1p(total\_rooms)],

'total\_bedrooms': [total\_bedrooms],

'population': [np.log1p(population)],

'households': [households],

'median\_income': [median\_income],

'ocean\_proximity': [ocean\_proximity],

'rooms\_per\_household': [np.log1p(total\_rooms) / households],

'bedrooms\_per\_room': [total\_bedrooms / np.log1p(total\_rooms)],

'population\_per\_household': [np.log1p(population) / households]

})

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

usd\_value = np.expm1(prediction[0]) # Reverse log transformation

return f"Predicted House Value: ${usd\_value:,.2f} USD"

iface = gr.Interface(

fn=predict\_house\_value,

inputs=[

gr.Slider(-124, -114, step=0.1, label="Longitude"),

gr.Slider(32, 42, step=0.1, label="Latitude"),

gr.Slider(0, 52, step=1, label="Housing Median Age"),

gr.Slider(0, 40000, step=100, label="Total Rooms"),

gr.Slider(0, 7000, step=10, label="Total Bedrooms"),

gr.Slider(0, 50000, step=100, label="Population"),

gr.Slider(0, 7000, step=10, label="Households"),

gr.Slider(0, 15, step=0.1, label="Median Income"),

gr.Dropdown(choices=['<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'NEAR BAY', 'ISLAND'], label="Ocean Proximity")

],

outputs="text",

title="California House Price Predictor",

description="Enter features to predict the median house value (in USD)."

)

print("\nGradio interface is ready. Run iface.launch() in a Hugging Face Space to use it.")

except ImportError:

print("\nGradio not installed. Skipping Gradio interface. Install gradio to enable.")