import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from sklearn.ensemble import RandomForestRegressor

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

# Ensure inline plotting in Jupyter notebooks

%matplotlib inline

# Load the dataset

milk = pd.read\_csv('milk\_quality\_data.csv')

print(milk.head())

print(milk.describe())

print(milk.info())

# Data visualization

fig, axes = plt.subplots(3, 4, figsize=(15, 10))

sns.barplot(ax=axes[0, 0], x='quality', y='density', data=milk)

sns.barplot(ax=axes[0, 1], x='quality', y='pH', data=milk)

sns.barplot(ax=axes[0, 2], x='quality', y='sulfates', data=milk)

sns.barplot(ax=axes[0, 3], x='quality', y='alcohol', data=milk)

sns.barplot(ax=axes[1, 0], x='quality', y='residual\_sugar', data=milk)

sns.barplot(ax=axes[1, 1], x='quality', y='chlorides', data=milk)

sns.barplot(ax=axes[1, 2], x='quality', y='free\_sulfur\_dioxide', data=milk)

sns.barplot(ax=axes[1, 3], x='quality', y='total\_sulfur\_dioxide', data=milk)

sns.barplot(ax=axes[2, 0], x='quality', y='citric\_acid', data=milk)

sns.barplot(ax=axes[2, 1], x='quality', y='fixed\_acidity', data=milk)

sns.barplot(ax=axes[2, 2], x='quality', y='volatile\_acidity', data=milk)

plt.tight\_layout()

# Categorize quality

ranges = (0, 4, 7)

groups = ['low', 'high']

milk['quality'] = pd.cut(milk['quality'], bins=ranges, labels=groups)

le = LabelEncoder()

milk['quality'] = le.fit\_transform(milk['quality'])

print(milk.head())

# Balance classes

high\_quality = milk[milk['quality'] == 1]

low\_quality = milk[milk['quality'] == 0].sample(frac=1).head(len(high\_quality))

balanced\_data = pd.concat([high\_quality, low\_quality]).sample(frac=1)

# Train-test split

X = balanced\_data.drop('quality', axis=1)

y = balanced\_data['quality']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

# Standardize features

scaler = StandardScaler()

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

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

# Model training and evaluation

param\_grid = {'n\_estimators': [100, 200, 300, 400, 500]}

grid\_rf = GridSearchCV(RandomForestRegressor(), param\_grid, scoring='r2', cv=10)

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

print('Best parameters:', grid\_rf.best\_params\_)

predictions = grid\_rf.predict(X\_test)

print('RMSE:', mean\_squared\_error(y\_test, predictions, squared=False))

print('R2 Score:', r2\_score(y\_test, predictions))