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, cross\_val\_score

from sklearn.linear\_model import LinearRegression, Ridge

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

from sklearn.preprocessing import StandardScaler

import xgboost as xgb

train = pd.read\_csv('train.csv')

test = pd.read\_csv('test.csv')

submission = pd.read\_csv('sample\_submission.csv')

print(f"训练集大小: {train.shape}")

print(f"测试集大小: {test.shape}")

train.head()

# 读取训练集和测试集数据

train = pd.read\_csv('train.csv')

test = pd.read\_csv('test.csv')

# 检查 'SalePrice' 列是否存在于训练数据中

if 'SalePrice' not in train.columns:

print("错误：训练集 'SalePrice' 列不存在，请检查原始数据文件。")

else:

print("训练集包含 'SalePrice' 列。")

# 将训练集和测试集合并为一个数据集

full = pd.concat([train, test], axis=0, ignore\_index=True)

# 为合并的数据集添加一个 'TrainFlag' 列，区分训练集（1）和测试集（0）

full['TrainFlag'] = [1] \* len(train) + [0] \* len(test)

# 检查合并后的数据

print(full.head())

# 将 'TrainFlag' 列设置为训练集标志，删除无关列（包括 'Id' 和 'TrainFlag'）

train\_data = full[full['TrainFlag'] == 1].drop(['TrainFlag', 'Id'], axis=1)

test\_data = full[full['TrainFlag'] == 0].drop(['TrainFlag', 'Id'], axis=1)

# 保留 'SalePrice' 作为目标变量并删除训练数据中的 'SalePrice'

X = train\_data.drop('SalePrice', axis=1)

y = train\_data['SalePrice']

# 检查清理后的数据

print(f"训练特征维度: {X.shape}, 标签维度: {y.shape}")

# 填充缺失值

for col in X.select\_dtypes(include=['float64', 'int64']).columns:

X[col] = X[col].fillna(X[col].median()) # 使用中位数填充

# 对于分类变量，使用众数填充

if X[col].dtype == 'object':

X[col] = X[col].fillna(X[col].mode()[0])

# 查看填充后的缺失值情况

print("缺失值处理后的数据：")

print(X.isnull().sum())

# 独热编码处理分类特征

categorical\_cols = X.select\_dtypes(include=['object']).columns

X = pd.get\_dummies(X, columns=categorical\_cols, drop\_first=True)

print(f"独热编码后的特征维度：{X.shape}")

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

# 拆分训练集和测试集

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

print(f"训练集维度: {X\_train.shape}, 验证集维度: {X\_val.shape}")

from sklearn.preprocessing import StandardScaler

# 特征缩放

scaler = StandardScaler()

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

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

from sklearn.linear\_model import LinearRegression

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

import numpy as np

# 模型训练

lr\_model = LinearRegression()

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

# 预测

lr\_pred = lr\_model.predict(X\_val\_scaled)

# 评估

print("线性回归评估结果：")

print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_val, lr\_pred)):.2f}")

print(f"MAE: {mean\_absolute\_error(y\_val, lr\_pred):.2f}")

print(f"R² Score: {r2\_score(y\_val, lr\_pred):.4f}")

import xgboost as xgb

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

import numpy as np

# 训练 XGBoost 模型

xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42)

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

# 预测

xgb\_pred = xgb\_model.predict(X\_val\_scaled)

# 评估

print("XGBoost 模型评估结果：")

print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_val, xgb\_pred)):.2f}")

print(f"MAE: {mean\_absolute\_error(y\_val, xgb\_pred):.2f}")

print(f"R² Score: {r2\_score(y\_val, xgb\_pred):.4f}")

from sklearn.linear\_model import Ridge

from sklearn.model\_selection import GridSearchCV

# 岭回归超参数调优

ridge\_model = Ridge()

# 设置要调优的参数网格

param\_grid = {

'alpha': [0.1, 1, 10, 100] # 岭回归的正则化参数

}

# 使用GridSearchCV进行超参数搜索

grid\_search = GridSearchCV(estimator=ridge\_model, param\_grid=param\_grid, cv=5, scoring='neg\_mean\_squared\_error', verbose=1)

# 拟合模型

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

# 输出最佳超参数和对应的评分

print(f"最佳超参数: {grid\_search.best\_params\_}")

print(f"最佳交叉验证得分: {-grid\_search.best\_score\_:.2f}")

# 使用最佳超参数训练模型

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

# 对验证集进行预测

ridge\_pred = best\_ridge\_model.predict(X\_val\_scaled)

# 评估

print("岭回归（带超参数调优）评估结果：")

print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_val, ridge\_pred)):.2f}")

print(f"MAE: {mean\_absolute\_error(y\_val, ridge\_pred):.2f}")

print(f"R² Score: {r2\_score(y\_val, ridge\_pred):.4f}")

from sklearn.ensemble import RandomForestRegressor

# 训练随机森林模型

rf\_model = RandomForestRegressor(random\_state=42)

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

# 预测

rf\_pred = rf\_model.predict(X\_val\_scaled)

# 评估

print("随机森林模型评估结果：")

print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_val, rf\_pred)):.2f}")

print(f"MAE: {mean\_absolute\_error(y\_val, rf\_pred):.2f}")

print(f"R² Score: {r2\_score(y\_val, rf\_pred):.4f}")

from sklearn.ensemble import RandomForestRegressor

from scipy.stats import randint

from sklearn.model\_selection import RandomizedSearchCV

# 随机森林模型：设定超参数空间

rf\_model = RandomForestRegressor(random\_state=42)

param\_dist = {

'n\_estimators': randint(50, 200),

'max\_depth': randint(3, 20),

'min\_samples\_split': randint(2, 10),

'min\_samples\_leaf': randint(1, 10)

}

# 使用RandomizedSearchCV进行超参数调优

random\_search = RandomizedSearchCV(rf\_model, param\_distributions=param\_dist, n\_iter=100, cv=5, verbose=1, random\_state=42, n\_jobs=-1)

# 拟合模型

random\_search.fit(X\_train\_scaled, y\_train)

# 输出最佳超参数和对应的评分

print(f"最佳超参数: {random\_search.best\_params\_}")

print(f"最佳交叉验证得分: {-random\_search.best\_score\_:.2f}")

# 使用最佳超参数训练模型

best\_rf\_model = random\_search.best\_estimator\_

# 对验证集进行预测

rf\_pred = best\_rf\_model.predict(X\_val\_scaled)

# 评估

print("随机森林（带超参数调优）评估结果：")

print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_val, rf\_pred)):.2f}")

print(f"MAE: {mean\_absolute\_error(y\_val, rf\_pred):.2f}")

print(f"R² Score: {r2\_score(y\_val, rf\_pred):.4f}")

import lightgbm as lgb

# 特征缩放，保持特征名

scaler = StandardScaler()

X\_train\_scaled = pd.DataFrame(scaler.fit\_transform(X\_train), columns=X\_train.columns)

X\_val\_scaled = pd.DataFrame(scaler.transform(X\_val), columns=X\_val.columns)

# 训练 LightGBM 模型

lgb\_model = lgb.LGBMRegressor(random\_state=42)

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

# 使用 LightGBM 模型预测验证集

lgb\_pred = lgb\_model.predict(X\_val\_scaled)

# 计算评估指标

rmse = np.sqrt(mean\_squared\_error(y\_val, lgb\_pred))

mae = mean\_absolute\_error(y\_val, lgb\_pred)

r2 = r2\_score(y\_val, lgb\_pred)

# 打印评估结果

print("LightGBM 模型评估结果：")

print(f"RMSE: {rmse:.2f}")

print(f"MAE: {mae:.2f}")

print(f"R² Score: {r2:.4f}")

import matplotlib.pyplot as plt

# 设置中文字体显示

plt.rcParams['font.family'] = 'SimHei' # 黑体支持中文

plt.rcParams['axes.unicode\_minus'] = False # 解决负号显示问题

# 可视化函数

def plot\_predictions(y\_true, y\_preds, model\_names, title='模型预测结果对比'):

plt.figure(figsize=(14, 6))

for pred, name in zip(y\_preds, model\_names):

plt.plot(pred, label=f'{name} 预测')

plt.plot(y\_true.values, label='真实值', color='black', linewidth=2)

plt.title(title)

plt.xlabel('样本编号')

plt.ylabel('房价')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# 所有模型的预测结果与名称

predictions = [lr\_pred, ridge\_pred, rf\_pred, xgb\_pred, lgb\_pred]

model\_names = ['Linear Regression', 'Ridge (Tuned)', 'Random Forest (Tuned)', 'XGBoost', 'LightGBM']

# 调用可视化函数

plot\_predictions(y\_val, predictions, model\_names)

import matplotlib.pyplot as plt

# 设置中文字体和负号显示

plt.rcParams['font.family'] = 'Microsoft YaHei'

plt.rcParams['axes.unicode\_minus'] = False

model\_names = ['Linear Regression', 'Ridge', 'Random Forest', 'XGBoost', 'LightGBM']

rmses = [35634.48, 35400.12, 30543.22, 29987.01, 29412.55]

maes = [23012.45, 22890.34, 19876.21, 19345.65, 18988.44]

def plot\_bar\_chart(values, title, ylabel):

fig, ax = plt.subplots(figsize=(10, 6), constrained\_layout=True)

bars = ax.bar(model\_names, values, color='skyblue')

ax.set\_title(title, fontsize=16)

ax.set\_ylabel(ylabel, fontsize=14)

ax.set\_xticklabels(model\_names, rotation=15)

for bar in bars:

yval = bar.get\_height()

ax.text(bar.get\_x() + bar.get\_width()/2.0, yval + 1000, f'{yval:.2f}', ha='center', va='bottom')

plt.show()

plot\_bar\_chart(rmses, '不同模型的 RMSE 对比', 'RMSE')

plot\_bar\_chart(maes, '不同模型的 MAE 对比', 'MAE')

# 可视化实际值与预测值的对比 - 线性回归

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

sns.scatterplot(x=y\_val, y=lr\_pred, color='green', alpha=0.6, label="预测值 vs 实际值")

plt.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], color='red', linestyle='--', label="完美预测线")

plt.title("线性回归模型预测效果", fontsize=16)

plt.xlabel("实际值", fontsize=12)

plt.ylabel("预测值", fontsize=12)

plt.legend()

plt.show()

import seaborn as sns

# 可视化实际值与预测值的对比 - XGBoost

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

sns.scatterplot(x=y\_val, y=xgb\_pred, color='purple', alpha=0.6, label="预测值 vs 实际值")

plt.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], color='red', linestyle='--', label="完美预测线")

plt.title("XGBoost 模型预测效果", fontsize=16)

plt.xlabel("实际值", fontsize=12)

plt.ylabel("预测值", fontsize=12)

plt.legend()

plt.show()

# 可视化实际值与预测值的对比 - 岭回归

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

sns.scatterplot(x=y\_val, y=ridge\_pred, color='teal', alpha=0.6, label="预测值 vs 实际值")

plt.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], color='red', linestyle='--', label="完美预测线")

plt.title("岭回归模型预测效果", fontsize=16)

plt.xlabel("实际值")

plt.ylabel("预测值")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# 可视化实际值与预测值的对比 - 随机森林

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

sns.scatterplot(x=y\_val, y=rf\_pred, color='darkgreen', alpha=0.6, label="预测值 vs 实际值")

plt.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], color='red', linestyle='--', label="完美预测线")

plt.title("随机森林模型预测效果", fontsize=16)

plt.xlabel("实际值")

plt.ylabel("预测值")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# 可视化实际值与预测值的对比 - LightGBM

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

sns.scatterplot(x=y\_val, y=lgb\_pred, color='blue', alpha=0.6, label="预测值 vs 实际值")

plt.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], color='red', linestyle='--', label="完美预测线")

plt.title("LightGBM 模型预测效果", fontsize=16)

plt.xlabel("实际值", fontsize=12)

plt.ylabel("预测值", fontsize=12)

plt.legend()

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

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

# 每个模型画一个散点

sns.scatterplot(x=y\_val, y=lr\_pred, color='blue', label='Linear Regression', alpha=0.5)

sns.scatterplot(x=y\_val, y=ridge\_pred, color='teal', label='Ridge Regression', alpha=0.5)

sns.scatterplot(x=y\_val, y=rf\_pred, color='green', label='Random Forest', alpha=0.5)

sns.scatterplot(x=y\_val, y=xgb\_pred, color='purple', label='XGBoost', alpha=0.5)

sns.scatterplot(x=y\_val, y=lgb\_pred, color='orange', label='LightGBM', alpha=0.5)

# 理想预测线

plt.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], color='red', linestyle='--', label='完美预测线')

# 图形修饰

plt.title("全部模型预测值 vs 实际值对比", fontsize=18)

plt.xlabel("实际值", fontsize=14)

plt.ylabel("预测值", fontsize=14)

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()