# 哈尔滨工业大学（深圳）

# Harbin Institute of Technology (Shenzhen)

实验名称: _____

姓名: _____

学号: _____

日期: _____

# 1.实验内容

1. 利用各方各面的描述变量，采用特征工程、回归预测等关键技术预测爱荷华州艾姆斯每套住宅的最终价格。
2. 比赛网站：House Prices - Advanced Regression Techniques | Kaggle
3. 评价指标：使用均方根误差(RMSE)为唯一评价指标，通过 log()运算将价位高低不同的预测误差对最终结果的影响统一起来。

# 2.实验方法

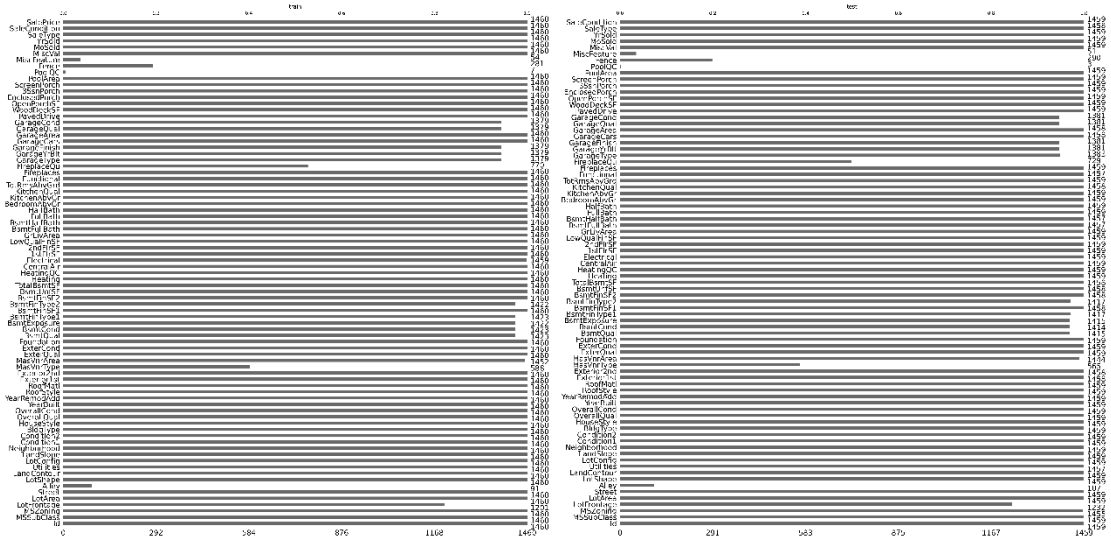
1. 观察初始数据

(1)基本信息

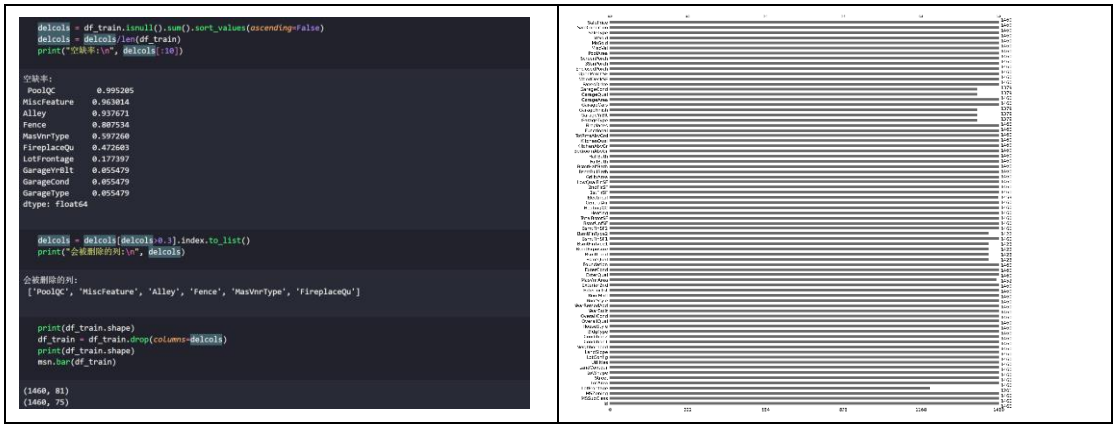
对于训练集，包括 Id 以及 SalePrice 在内，共有 81 个字段。训练集大小为 1461。

(2)空缺值统计

统计 train.csv 和 test.csv 中各属性的 null 值，结果如下。可以很明显看出，无论是在训练集和测试集当中（左边为训练集，右边为测试集），部分字段的空缺值占比都很多。比如 'PoolQC', 'MiscFeature', 'Alley', 'Fence', 'MasVnrType', 'FireplaceQu'6 个字段，空缺值占比几乎都在一半左右或者一半以上。



采取以 0.3 为空缺值删除阈值，删除空缺率大于 0.3 的字段。下左图为具体代码实现，右图为删除空缺值占比过大的字段之后的训练集。



(3)划分离散型属性和连续型属性

对于离散型属性与连续型属性，有着不同的特征工程的实现方法。通过对属性的类型进行简单判断，来确定属性是离散型还是连续型。如下，列属性只有 int64,float64,object 三种(object 一般对应一些字符串描述类型的字段,可以看作是离散型变量)。进行简单的划分，前两者数据类型看作是连续型属性，后者看作是离散型属性。划分结果：

- 连续型属性：38 个

当然连续型属性中的 Id 和 SalePrice 不参与训练，后续从训练集中删除。

['Id', 'MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice', 'LotFrontage', 'MasVnrArea', 'GarageYrBlt']

- 离散型属性：37 个

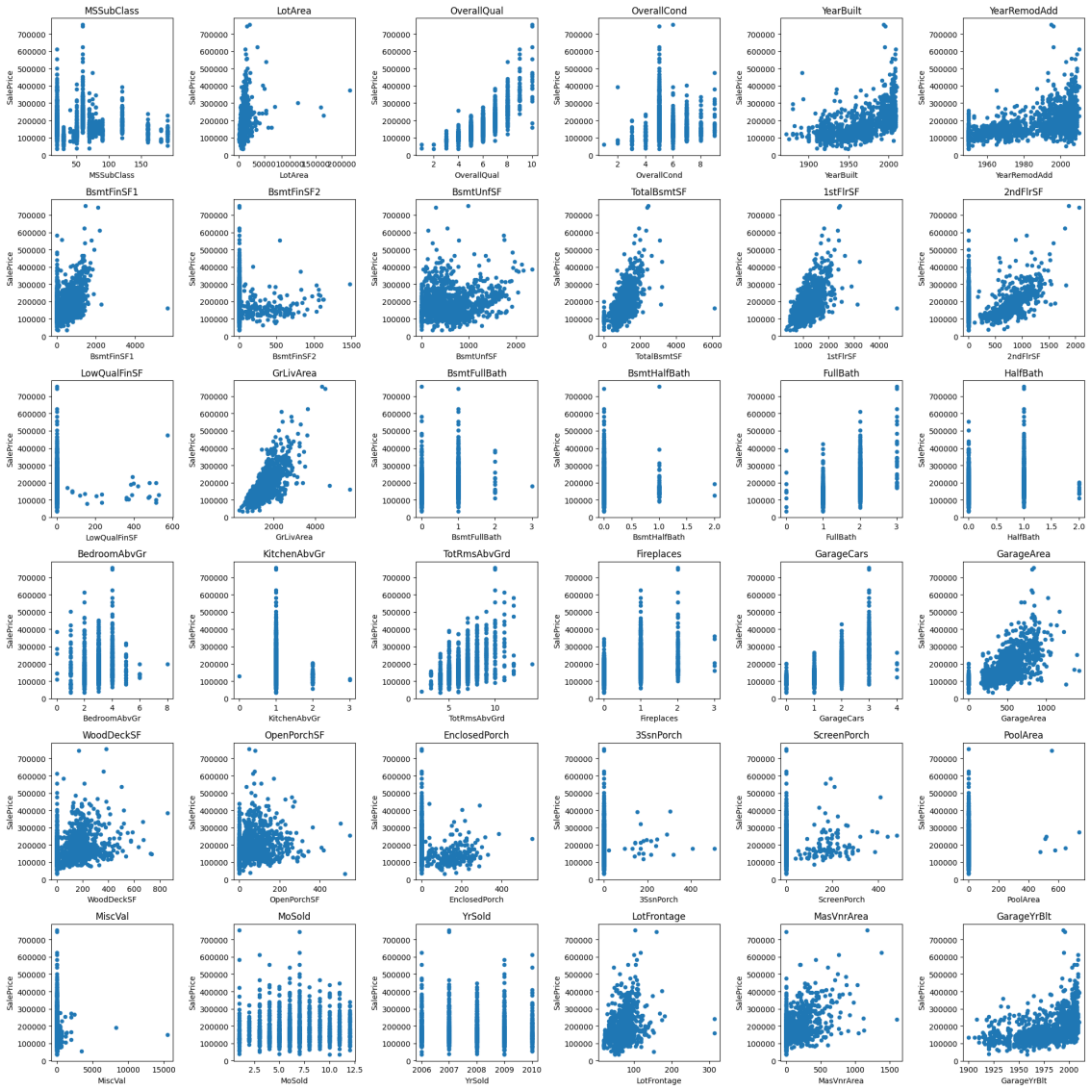
['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'SaleType', 'SaleCondition']


2. 数据预处理

(1)连续型属性和离散型属性的数值分布

- 连续型属性:

由下图可以看出,虽然一些字段数据类型是 int 是 float,但实际上还是表示的不连续的数。比如第三行中倒数四个子图所对应的属性。




➢ 离散型属性：

一些离散型属性对应的字段数值很少，一些离散型属性对应的字段值很多。

```
for ele in type_col['discrete']:
    print(ele, df_train[ele].value_counts().to_dict())
```

```
MSZoning {'RL': 1151, 'RM': 218, 'FV': 65, 'RH': 16, 'C (all)': 10}
Street {'Pave': 1454, 'Grvl': 6}
LotShape {'Reg': 925, 'IR1': 484, 'IR2': 41, 'IR3': 10}
LandContour {'Lvl': 1311, 'Bnk': 63, 'HLS': 50, 'Low': 36}
Utilities {'AllPub': 1459, 'NoSeWa': 1}
LotConfig {'Inside': 1052, 'Corner': 263, 'CulDSac': 94, 'FR2': 47, 'FR3': 4}
LandSlope {'Gtl': 1382, 'Mod': 65, 'Sev': 13}
Neighborhood {'NAmes': 225, 'CollgCr': 150, 'OldTown': 113, 'Edwards': 100, 'Somerst': 86, 'Gilbert': 79, 'NridgHt': 77, 'Sawyer': 74, 'NWAmes': 73, 'SawyerW': 59, 'BrkSide': 58, 'Crawfo
Condition1 {'Norm': 1260, 'Feedr': 81, 'Artery': 48, 'RRAn': 26, 'PosN': 19, 'RRAe': 11, 'PosA': 8, 'RRNn': 5, 'RRNe': 2}
Condition2 {'Norm': 1445, 'Feedr': 6, 'Artery': 2, 'RRNn': 2, 'PosN': 2, 'PosA': 1, 'RRAn': 1, 'RRAe': 1}
BldgType {'1Fam': 1220, 'TwnhsE': 114, 'Duplex': 52, 'Twnhs': 43, '2fmCon': 31}
HouseStyle {'1Story': 726, '2Story': 445, '1.5Fin': 154, 'SLvl': 65, 'SFoyer': 37, '1.5Unf': 14, '2.5Unf': 11, '2.5Fin': 8}
RoofStyle {'Gable': 1141, 'Hip': 286, 'Flat': 13, 'Gambrel': 11, 'Mansard': 7, 'Shed': 2}
RoofMatl {'CompShg': 1434, 'Tar&Grv': 11, 'WdShngl': 6, 'WdShake': 5, 'Metal': 1, 'Membran': 1, 'Roll': 1, 'ClyTile': 1}
Exterior1st {'VinylSd': 515, 'HdBoard': 222, 'MetalSd': 220, 'Wd Sdng': 206, 'Plywood': 108, 'CemntBd': 61, 'BrkFace': 50, 'WdShing': 26, 'Stucco': 25, 'AsbShng': 20, 'BrkComm': 2, 'Stor
Exterior2nd {'VinylSd': 504, 'MetalSd': 214, 'HdBoard': 207, 'Wd Sdng': 197, 'Plywood': 142, 'CmentBd': 60, 'Wd Shng': 38, 'Stucco': 26, 'BrkFace': 25, 'AsbShng': 20, 'ImStucc': 10, 'Brk
ExterQual {'TA': 906, 'Gd': 488, 'Ex': 52, 'Fa': 14}
ExterCond {'TA': 1282, 'Gd': 146, 'Fa': 28, 'Ex': 3, 'Po': 1}
Foundation {'PConc': 647, 'CBlock': 634, 'BrkTil': 146, 'Slab': 24, 'Stone': 6, 'Wood': 3}
BsmtQual {'TA': 649, 'Gd': 618, 'Ex': 121, 'Fa': 35}
BsmtCond {'TA': 1311, 'Gd': 65, 'Fa': 45, 'Po': 2}
BsmtExposure {'No': 953, 'Av': 221, 'Gd': 134, 'Mn': 114}
BsmtFinType1 {'Unf': 430, 'GLQ': 418, 'ALQ': 220, 'BLQ': 148, 'Rec': 133, 'LwQ': 74}
BsmtFinType2 {'Unf': 1256, 'Rec': 54, 'LwQ': 46, 'BLQ': 33, 'ALQ': 19, 'GLQ': 14}
Heating {'GasA': 1428, 'GasW': 18, 'Grav': 7, 'Wall': 4, 'OthW': 2, 'Floor': 1}
HeatingQC {'Ex': 741, 'TA': 428, 'Gd': 241, 'Fa': 49, 'Po': 1}
CentralAir {'Y': 1365, 'N': 95}
Electrical {'SBrkr': 1334, 'FuseA': 94, 'FuseF': 27, 'FuseP': 3, 'Mix': 1}
KitchenQual {'TA': 735, 'Gd': 586, 'Ex': 100, 'Fa': 39}
```

(2) 空缺值填充

对于空缺值的填充根据连续型和离散型采取不同的填充策略。

➢ 连续型：

采取列均值进行填充。

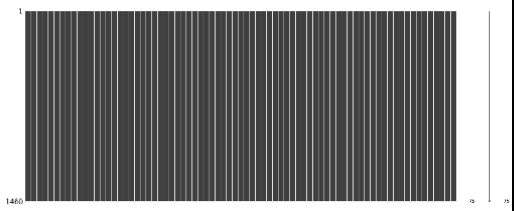
➢ 离散型：

采取该列中的出现次数最多的字段值进行填充（null 除外）

以下左图是填充实现代码以及含有空缺值的列，右图为填充之后的结果。

```
for colname in df_train.columns:
    if df_train[colname].isna().sum() != 0:
        print(df_train[colname].dtype, colname, '含有空值' ,df_train[colname].isna().sum())
        if df_train[colname].dtype == 'object':
            df_train[colname] = df_train[colname].fillna(df_train[colname].mode()[0])
        elif df_train[colname].dtype == 'int64' or df_train[colname].dtype == 'float64':
            df_train[colname] = df_train[colname].fillna(df_train[colname].mean())
        else:
            raise ValueError("未知数据类型")
```

```
float64 LotFrontage 含有空值 259
float64 MasVnrArea 含有空值 8
object BsmtQual 含有空值 37
object BsmtCond 含有空值 37
object BsmtExposure 含有空值 38
object BsmtFinType1 含有空值 37
object BsmtFinType2 含有空值 38
object Electrical 含有空值 1
object GarageType 含有空值 81
float64 GarageYrBlt 含有空值 81
object GarageFinish 含有空值 81
object GarageQual 含有空值 81
object GarageCond 含有空值 81
```




(3) 划分训练集中训练数据与实际房价结果

删除df_train中Id字段，划分属性集合与最终结果SalePrice

- 也对连续型属性和离散型属性的字典进行修改

```
X_train, y_train = df_train.drop(columns=['SalePrice', 'Id']), df_train['SalePrice']
```

```
print(type_col['continue'])
type_col['continue'].remove('SalePrice')
type_col['continue'].remove('Id')
print(type_col['continue'])
```

```
['Id', 'MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2',
['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'Bsmt
```

(4) 对连续型属性进行数据标准化

对于连续型属性，采取 zscore 方法进行标准化，消除不同量纲大小对于数值的影响。

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train[type_col['continue']] = scaler.fit_transform(X_train[type_col['continue']])
display(X_train[type_col['continue']].head())
```

| | MSSubClass | LotArea | OverallQual | OverallCond | YearBuilt | YearRemodAdd | BsmtFinSF1 | BsmtFinSF2 | BsmtUnfSF | TotalBsmtSF |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.073375 | -0.207142 | 0.651479 | -0.517200 | 1.050994 | 0.878668 | 0.575425 | -0.288653 | -0.944591 | -0.459303 |
| 1 | -0.872563 | -0.091886 | -0.071836 | 2.179628 | 0.156734 | -0.429577 | 1.171992 | -0.288653 | -0.641228 | 0.466465 |
| 2 | 0.073375 | 0.073480 | 0.651479 | -0.517200 | 0.984752 | 0.830215 | 0.092907 | -0.288653 | -0.301643 | -0.313369 |
| 3 | 0.309859 | -0.096897 | 0.651479 | -0.517200 | -1.863632 | -0.720298 | -0.499274 | -0.288653 | -0.061670 | -0.687324 |
| 4 | 0.073375 | 0.375148 | 1.374795 | -0.517200 | 0.951632 | 0.733308 | 0.463568 | -0.288653 | -0.174865 | 0.199680 |

5 rows × 36 columns

(5) 对于离散型属性的特征工程 onehot 编码

在此若采取字典映射的方式，即将一个属性中的不同值映射为不同的数字值，可能会对回归模型产生影响。因为该方式隐藏着同一属性下不同的值在“数值”方面有着固定的大小先后顺序。因此，对于离散型属性采取 onehot 编码方式。

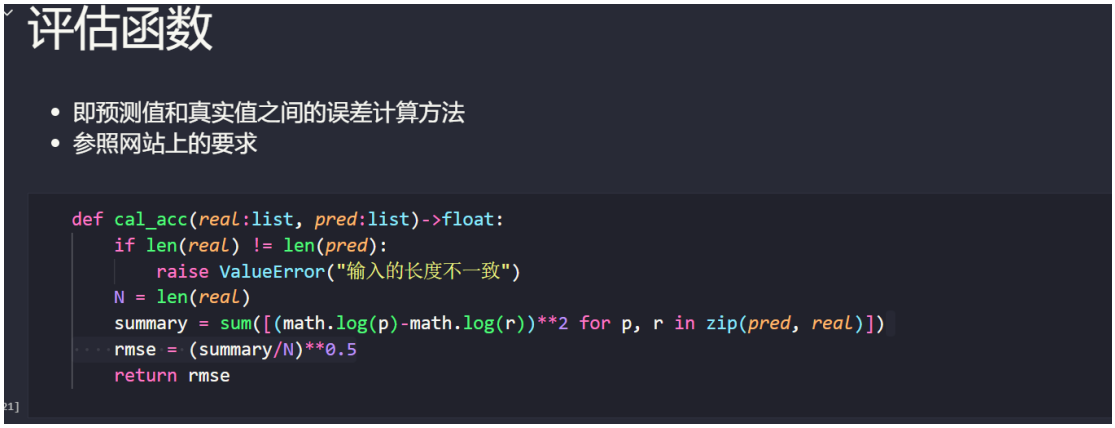
```
from sklearn.preprocessing import OneHotEncoder


encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_discrete_arr = encoder.fit_transform(X_train[type_col['discrete']])
encoded_discrete_df = pd.DataFrame(encoded_discrete_arr, columns=encoder.get_feature_names_out(type_col['discrete']))
X_train = pd.concat([X_train.drop(columns=type_col['discrete']), encoded_discrete_df], axis=1)
```

3. 评估函数

采用均方根误差(RMSE)为唯一评价指标。同时用 log() 运算将价位高低不同的预测误差对最终结果的影响统一起来。

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\log \hat{y}_i - \log y_i)^2}{N}}$$




4. 训练模型

在回归算法的选择中，选取了随机森林算法进行回归预测。随机森林由多个决策树组成，在回归任务中通过对所有树的预测结果进行平均，通过组合多个决策树的结果来减少过拟合，提升预测性能。

- 该随机森林中一共有 100 个决策树；
- 每个决策树的构建不限制决策树的深度，即会一直进行节点分裂直至节点是纯为止；
- 衡量节点分裂的质量采取的标准是均方误差 squared_error。

模型在训练集上的精度为 0.05876。

```
from sklearn.ensemble import RandomForestRegressor


model = RandomForestRegressor(n_estimators=100, random_state=13)


model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=13)
```

```
y_train_pred = model.predict(X_train)
print("在训练集上的误差", cal_acc(y_train.to_list(), y_train_pred))
```

```
在训练集上的误差 0.05876237258950698
```

5. 利用模型对测试集进行预测

(1) 首先按照处理训练集的方法对测试集进行数据预处理

- 删除空缺值过多的列
- 对连续型属性和离散型属性的空缺值填补
- 对连续型属性的数据标准化
- 对离散型属性的 onehot 编码

(2) 调用 model.predict()函数对测试集进行预测并将预测见过保存至文件

```
y_test_pred = model.predict(X_test)
```
[35] ✓ 0.0s

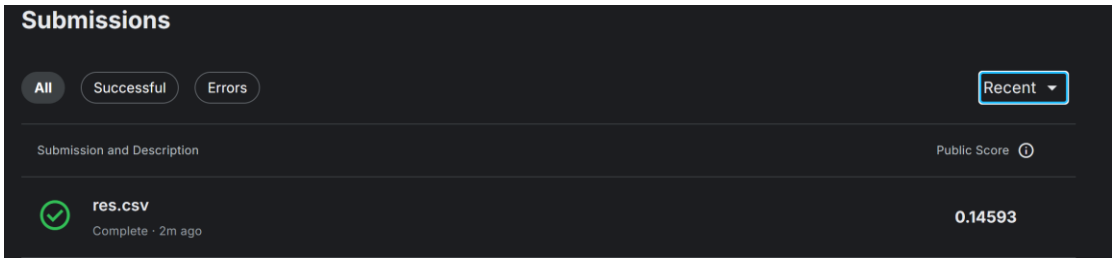
```
y_test_id = df_test['Id'].to_list()
if len(y_test_id)!=len(y_test_pred):
    raise ValueError('')
```
[36] ✓ 0.0s

```
res = pd.DataFrame(data={
    'Id': y_test_id,
    'SalePrice': y_test_pred
})
res.to_csv(f'res.csv', index=False)
res.head()
```
[37] ✓ 0.0s 在 Data Wrangler 中打开"res"

| | Id | SalePrice |
|---|---|---|
| 0 | 1461 | 128350.00 |
| 1 | 1462 | 156391.32 |
| 2 | 1463 | 181188.60 |
| 3 | 1464 | 180211.35 |
| 4 | 1465 | 196018.89 |

## 3.实验结果和分析

预测结果文件保存在 res.csv 文件当中。其中测试集预测房价与测试集实际房价的误差为 0.14593。测试集预测房价与实际房价之间误差较小，模型较优。下为 kaggle 评测截图。



## 4.总结

在本次实验中，通过建立随机森林模型，对测爱荷华州艾姆斯每套住宅的最终价格进行了预测。训练数据的质量决定了模型的上限，模型的选择和参数的调优是不断的接近这个上限的过程。与实验课中构建决策树实验提供的数据集不同，该实验提供的数据集，无论是训练集还是测试集都有许多空缺值。因此在特征工程方面，对数据集进行了规范处理。对于连续型属性，采取均值填充空缺值，进行 zscore 数据标准化；对于离散型属性，采取众数填充空缺值，onehot 编码的方式进行规范。

通过本次实验，自己的代码能力有了很大提高，了解了 scikit-learn 的使用，熟悉了数据预处理、参数选择、模型训练、模型评估等机器学习主要流程，对随机森林的应用有了更深刻的理解。