Name: Syed Saad Ullah Hussaini **Roll No.:** 21K-4736

Machine Learning Test

Section 1: Data Exploration

1. Import Libraries

In [ ]:
```
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
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDispla
from sklearn.linear_model import LassoCV
from sklearn.model_selection import GridSearchCV
```

2. Load Dataset

In [ ]:
```
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

In [ ]:
```
train_data.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lv
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lv
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lv
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lv
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lv

5 rows × 81 columns

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCont
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	

5 rows × 80 columns

3. **Explore Features**

In [ ]:
```
train_data.describe()
```

Out[ ]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	14
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	19
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	18
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	19
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	19
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	20
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	20

8 rows × 38 columns

In [ ]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     588 non-null    object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
```

```
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [ ]:
```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1459 non-null   int64  
 1   MSSubClass     1459 non-null   int64  
 2   MSZoning       1455 non-null   object 
 3   LotFrontage    1232 non-null   float64
 4   LotArea        1459 non-null   int64  
 5   Street         1459 non-null   object 
 6   Alley          107 non-null    object 
 7   LotShape       1459 non-null   object 
 8   LandContour    1459 non-null   object 
 9   Utilities      1457 non-null   object 
 10  LotConfig      1459 non-null   object 
 11  LandSlope      1459 non-null   object 
 12  Neighborhood   1459 non-null   object 
 13  Condition1     1459 non-null   object 
 14  Condition2     1459 non-null   object 
 15  BldgType       1459 non-null   object 
 16  HouseStyle     1459 non-null   object 
 17  OverallQual    1459 non-null   int64  
 18  OverallCond    1459 non-null   int64  
 19  YearBuilt      1459 non-null   int64  
 20  YearRemodAdd   1459 non-null   int64  
 21  RoofStyle      1459 non-null   object 
 22  RoofMatl       1459 non-null   object 
 23  Exterior1st    1458 non-null   object 
 24  Exterior2nd    1458 non-null   object 
 25  MasVnrType     565 non-null    object 
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object 
 28  ExterCond      1459 non-null   object 
 29  Foundation     1459 non-null   object 
 30  BsmtQual       1415 non-null   object 
 31  BsmtCond       1414 non-null   object 
 32  BsmtExposure   1415 non-null   object 
 33  BsmtFinType1   1417 non-null   object 
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object 
 36  BsmtFinSF2     1458 non-null   float64
 37  BsmtUnfSF      1458 non-null   float64
 38  TotalBsmtSF    1458 non-null   float64
 39  Heating        1459 non-null   object 
 40  HeatingQC      1459 non-null   object 
 41  CentralAir     1459 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1459 non-null   int64  
 44  2ndFlrSF       1459 non-null   int64  
 45  LowQualFinSF   1459 non-null   int64  
 46  GrLivArea      1459 non-null   int64  
 47  BsmtFullBath   1457 non-null   float64
 48  BsmtHalfBath   1457 non-null   float64
 49  FullBath       1459 non-null   int64  
 50  HalfBath       1459 non-null   int64  
```

```
 51  BedroomAbvGr   1459 non-null   int64
 52  KitchenAbvGr   1459 non-null   int64
 53  KitchenQual    1458 non-null   object
 54  TotRmsAbvGrd   1459 non-null   int64
 55  Functional     1457 non-null   object
 56  Fireplaces     1459 non-null   int64
 57  FireplaceQu    729 non-null    object
 58  GarageType     1383 non-null   object
 59  GarageYrBlt    1381 non-null   float64
 60  GarageFinish   1381 non-null   object
 61  GarageCars     1458 non-null   float64
 62  GarageArea     1458 non-null   float64
 63  GarageQual     1381 non-null   object
 64  GarageCond     1381 non-null   object
 65  PavedDrive     1459 non-null   object
 66  WoodDeckSF     1459 non-null   int64
 67  OpenPorchSF    1459 non-null   int64
 68  EnclosedPorch  1459 non-null   int64
 69  3SsnPorch      1459 non-null   int64
 70  ScreenPorch    1459 non-null   int64
 71  PoolArea       1459 non-null   int64
 72  PoolQC         3 non-null      object
 73  Fence          290 non-null    object
 74  MiscFeature    51 non-null     object
 75  MiscVal        1459 non-null   int64
 76  MoSold         1459 non-null   int64
 77  YrSold         1459 non-null   int64
 78  SaleType       1458 non-null   object
 79  SaleCondition  1459 non-null   object
dtypes: float64(11), int64(26), object(43)
memory usage: 912.0+ KB
```

In [ ]:
```
train_data['SalesPrice'].hist()
```

In [ ]:
```
for i in train_data.columns:
        if type(train_data[i][0]) == str:
                print(train_data[i].value_counts(normalize=True))
```

In [ ]:
```
for i in test_data.columns:
        if type(test_data[i][0]) == str:
                print(test_data[i].value_counts(normalize=True))
```

4. Handle Missing Data

In [ ]:
```
train_data.isna().sum().to_dict()
```

Out[ ]:
```
{'Id': 0,
 'MSSubClass': 0,
 'MSZoning': 0,
 'LotFrontage': 259,
 'LotArea': 0,
 'Street': 0,
 'Alley': 1369,
 'LotShape': 0,
 'LandContour': 0,
 'Utilities': 0,
 'LotConfig': 0,
 'LandSlope': 0,
 'Neighborhood': 0,
 'Condition1': 0,
 'Condition2': 0,
 'BldgType': 0,
 'HouseStyle': 0,
 'OverallQual': 0,
 'OverallCond': 0,
 'YearBuilt': 0,
 'YearRemodAdd': 0,
 'RoofStyle': 0,
 'RoofMatl': 0,
 'Exterior1st': 0,
 'Exterior2nd': 0,
 'MasVnrType': 872,
 'MasVnrArea': 8,
 'ExterQual': 0,
 'ExterCond': 0,
 'Foundation': 0,
 'BsmtQual': 37,
 'BsmtCond': 37,
 'BsmtExposure': 38,
 'BsmtFinType1': 37,
 'BsmtFinSF1': 0,
 'BsmtFinType2': 38,
 'BsmtFinSF2': 0,
 'BsmtUnfSF': 0,
 'TotalBsmtSF': 0,
 'Heating': 0,
 'HeatingQC': 0,
 'CentralAir': 0,
 'Electrical': 1,
 '1stFlrSF': 0,
 '2ndFlrSF': 0,
 'LowQualFinSF': 0,
 'GrLivArea': 0,
 'BsmtFullBath': 0,
 'BsmtHalfBath': 0,
 'FullBath': 0,
 'HalfBath': 0,
 'BedroomAbvGr': 0,
 'KitchenAbvGr': 0,
 'KitchenQual': 0,
```



```
 'Functional': 0,
```

```
 'Fireplaces': 0,
 'FireplaceQu': 690,
 'GarageType': 81,
 'GarageYrBlt': 81,
 'GarageFinish': 81,
 'GarageCars': 0,
 'GarageArea': 0,
 'GarageQual': 81,
 'GarageCond': 81,
 'PavedDrive': 0,
 'WoodDeckSF': 0,
 'OpenPorchSF': 0,
 'EnclosedPorch': 0,
 '3SsnPorch': 0,
 'ScreenPorch': 0,
 'PoolArea': 0,
 'PoolQC': 1453,
 'Fence': 1179,
 'MiscFeature': 1406,
 'MiscVal': 0,
 'MoSold': 0,
 'YrSold': 0,
 'SaleType': 0,
 'SaleCondition': 0,
 'SalePrice': 0}
```

In [ ]:
```
data_col_drop = train_data.drop(columns=['Alley', 'MasVnrType', 'FireplaceQu', 'Poo
```

In [ ]:
```
test_data_col_drop = test_data.drop(columns=['Alley', 'MasVnrType', 'FireplaceQu',
```

In [ ]:
```
data_col_drop.isna().sum().to_dict()
```

Out[ ]:

```
{'MSSubClass': 0,
 'MSZoning': 0,
 'LotFrontage': 259,
 'LotArea': 0,
 'Street': 0,
 'LotShape': 0,
 'LandContour': 0,
 'Utilities': 0,
 'LotConfig': 0,
 'LandSlope': 0,
 'Neighborhood': 0,
 'Condition1': 0,
 'Condition2': 0,
 'BldgType': 0,
 'HouseStyle': 0,
 'OverallQual': 0,
 'OverallCond': 0,
 'YearBuilt': 0,
 'YearRemodAdd': 0,
 'RoofStyle': 0,
 'RoofMatl': 0,
 'Exterior1st': 0,
 'Exterior2nd': 0,
 'MasVnrArea': 8,
 'ExterQual': 0,
 'ExterCond': 0,
 'Foundation': 0,
 'BsmtQual': 37,
 'BsmtCond': 37,
 'BsmtExposure': 38,
 'BsmtFinType1': 37,
 'BsmtFinSF1': 0,
 'BsmtFinType2': 38,
 'BsmtFinSF2': 0,
 'BsmtUnfSF': 0,
 'TotalBsmtSF': 0,
 'Heating': 0,
 'HeatingQC': 0,
 'CentralAir': 0,
 'Electrical': 1,
 '1stFlrSF': 0,
 '2ndFlrSF': 0,
 'LowQualFinSF': 0,
 'GrLivArea': 0,
 'BsmtFullBath': 0,
 'BsmtHalfBath': 0,
 'FullBath': 0,
 'HalfBath': 0,
 'BedroomAbvGr': 0,
 'KitchenAbvGr': 0,
 'KitchenQual': 0,
 'TotRmsAbvGrd': 0,
 'Functional': 0,
 'Fireplaces': 0,
```



```
 'GarageYrBlt': 81,
```

```
 'GarageFinish': 81,
 'GarageCars': 0,
 'GarageArea': 0,
 'GarageQual': 81,
 'GarageCond': 81,
 'PavedDrive': 0,
 'WoodDeckSF': 0,
 'OpenPorchSF': 0,
 'EnclosedPorch': 0,
 '3SsnPorch': 0,
 'ScreenPorch': 0,
 'PoolArea': 0,
 'MiscVal': 0,
 'MoSold': 0,
 'YrSold': 0,
 'SaleType': 0,
 'SaleCondition': 0,
 'SalePrice': 0}
```

In [ ]:
```
data_col_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 74 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MSSubClass     1460 non-null   int64  
 1   MSZoning       1460 non-null   object 
 2   LotFrontage    1201 non-null   float64
 3   LotArea        1460 non-null   int64  
 4   Street         1460 non-null   object 
 5   LotShape       1460 non-null   object 
 6   LandContour    1460 non-null   object 
 7   Utilities      1460 non-null   object 
 8   LotConfig      1460 non-null   object 
 9   LandSlope      1460 non-null   object 
 10  Neighborhood   1460 non-null   object 
 11  Condition1     1460 non-null   object 
 12  Condition2     1460 non-null   object 
 13  BldgType       1460 non-null   object 
 14  HouseStyle     1460 non-null   object 
 15  OverallQual    1460 non-null   int64  
 16  OverallCond    1460 non-null   int64  
 17  YearBuilt      1460 non-null   int64  
 18  YearRemodAdd   1460 non-null   int64  
 19  RoofStyle      1460 non-null   object 
 20  RoofMatl       1460 non-null   object 
 21  Exterior1st    1460 non-null   object 
 22  Exterior2nd    1460 non-null   object 
 23  MasVnrArea     1452 non-null   float64
 24  ExterQual      1460 non-null   object 
 25  ExterCond      1460 non-null   object 
 26  Foundation     1460 non-null   object 
 27  BsmtQual       1423 non-null   object 
 28  BsmtCond       1423 non-null   object 
 29  BsmtExposure   1422 non-null   object 
 30  BsmtFinType1   1423 non-null   object 
 31  BsmtFinSF1     1460 non-null   int64  
 32  BsmtFinType2   1422 non-null   object 
 33  BsmtFinSF2     1460 non-null   int64  
 34  BsmtUnfSF      1460 non-null   int64  
 35  TotalBsmtSF    1460 non-null   int64  
 36  Heating        1460 non-null   object 
 37  HeatingQC      1460 non-null   object 
 38  CentralAir     1460 non-null   object 
 39  Electrical     1459 non-null   object 
 40  1stFlrSF       1460 non-null   int64  
 41  2ndFlrSF       1460 non-null   int64  
 42  LowQualFinSF   1460 non-null   int64  
 43  GrLivArea      1460 non-null   int64  
 44  BsmtFullBath   1460 non-null   int64  
 45  BsmtHalfBath   1460 non-null   int64  
 46  FullBath       1460 non-null   int64  
 47  HalfBath       1460 non-null   int64  
 48  BedroomAbvGr   1460 non-null   int64  
 49  KitchenAbvGr   1460 non-null   int64  
 50  KitchenQual    1460 non-null   object 
```

```
 51  TotRmsAbvGrd   1460 non-null   int64
 52  Functional     1460 non-null   object
 53  Fireplaces     1460 non-null   int64
 54  GarageType     1379 non-null   object
 55  GarageYrBlt    1379 non-null   float64
 56  GarageFinish   1379 non-null   object
 57  GarageCars     1460 non-null   int64
 58  GarageArea     1460 non-null   int64
 59  GarageQual     1379 non-null   object
 60  GarageCond     1379 non-null   object
 61  PavedDrive     1460 non-null   object
 62  WoodDeckSF     1460 non-null   int64
 63  OpenPorchSF    1460 non-null   int64
 64  EnclosedPorch  1460 non-null   int64
 65  3SsnPorch      1460 non-null   int64
 66  ScreenPorch    1460 non-null   int64
 67  PoolArea       1460 non-null   int64
 68  MiscVal        1460 non-null   int64
 69  MoSold         1460 non-null   int64
 70  YrSold         1460 non-null   int64
 71  SaleType       1460 non-null   object
 72  SaleCondition  1460 non-null   object
 73  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(34), object(37)
memory usage: 844.2+ KB
```

In [ ]:
```
def fill_na_median(data):
    data_dropped = data.copy()
    for i in data_dropped.columns:
            try:
                data_dropped[i] = data_dropped[i].fillna(data_dropped[i].median())
            except:
                 pass
    return data_dropped
```

In [ ]:
```
train_data_dropped = fill_na_median(data_col_drop)
train_data_dropped.head()
```

Out[ ]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub

5 rows × 74 columns

In [ ]:
```
test_data_col_drop.isna().sum().to_dict()
```

Out[ ]:

```
{'MSSubClass': 0,
 'MSZoning': 4,
 'LotFrontage': 227,
 'LotArea': 0,
 'Street': 0,
 'LotShape': 0,
 'LandContour': 0,
 'Utilities': 2,
 'LotConfig': 0,
 'LandSlope': 0,
 'Neighborhood': 0,
 'Condition1': 0,
 'Condition2': 0,
 'BldgType': 0,
 'HouseStyle': 0,
 'OverallQual': 0,
 'OverallCond': 0,
 'YearBuilt': 0,
 'YearRemodAdd': 0,
 'RoofStyle': 0,
 'RoofMatl': 0,
 'Exterior1st': 1,
 'Exterior2nd': 1,
 'MasVnrArea': 15,
 'ExterQual': 0,
 'ExterCond': 0,
 'Foundation': 0,
 'BsmtQual': 44,
 'BsmtCond': 45,
 'BsmtExposure': 44,
 'BsmtFinType1': 42,
 'BsmtFinSF1': 1,
 'BsmtFinType2': 42,
 'BsmtFinSF2': 1,
 'BsmtUnfSF': 1,
 'TotalBsmtSF': 1,
 'Heating': 0,
 'HeatingQC': 0,
 'CentralAir': 0,
 'Electrical': 0,
 '1stFlrSF': 0,
 '2ndFlrSF': 0,
 'LowQualFinSF': 0,
 'GrLivArea': 0,
 'BsmtFullBath': 2,
 'BsmtHalfBath': 2,
 'FullBath': 0,
 'HalfBath': 0,
 'BedroomAbvGr': 0,
 'KitchenAbvGr': 0,
 'KitchenQual': 1,
 'TotRmsAbvGrd': 0,
 'Functional': 2,
 'Fireplaces': 0,
```



```
 'GarageYrBlt': 78,
```

```
 'GarageFinish': 78,
 'GarageCars': 1,
 'GarageArea': 1,
 'GarageQual': 78,
 'GarageCond': 78,
 'PavedDrive': 0,
 'WoodDeckSF': 0,
 'OpenPorchSF': 0,
 'EnclosedPorch': 0,
 '3SsnPorch': 0,
 'ScreenPorch': 0,
 'PoolArea': 0,
 'MiscVal': 0,
 'MoSold': 0,
 'YrSold': 0,
 'SaleType': 1,
 'SaleCondition': 0}
```

In [ ]:
```
test_data_dropped = fill_na_median(test_data_col_drop)
```

In [ ]:
```
test_data_dropped.isna().sum().to_dict()
```

Out[ ]:

```
{'MSSubClass': 0,
 'MSZoning': 4,
 'LotFrontage': 0,
 'LotArea': 0,
 'Street': 0,
 'LotShape': 0,
 'LandContour': 0,
 'Utilities': 2,
 'LotConfig': 0,
 'LandSlope': 0,
 'Neighborhood': 0,
 'Condition1': 0,
 'Condition2': 0,
 'BldgType': 0,
 'HouseStyle': 0,
 'OverallQual': 0,
 'OverallCond': 0,
 'YearBuilt': 0,
 'YearRemodAdd': 0,
 'RoofStyle': 0,
 'RoofMatl': 0,
 'Exterior1st': 1,
 'Exterior2nd': 1,
 'MasVnrArea': 0,
 'ExterQual': 0,
 'ExterCond': 0,
 'Foundation': 0,
 'BsmtQual': 44,
 'BsmtCond': 45,
 'BsmtExposure': 44,
 'BsmtFinType1': 42,
 'BsmtFinSF1': 0,
 'BsmtFinType2': 42,
 'BsmtFinSF2': 0,
 'BsmtUnfSF': 0,
 'TotalBsmtSF': 0,
 'Heating': 0,
 'HeatingQC': 0,
 'CentralAir': 0,
 'Electrical': 0,
 '1stFlrSF': 0,
 '2ndFlrSF': 0,
 'LowQualFinSF': 0,
 'GrLivArea': 0,
 'BsmtFullBath': 0,
 'BsmtHalfBath': 0,
 'FullBath': 0,
 'HalfBath': 0,
 'BedroomAbvGr': 0,
 'KitchenAbvGr': 0,
 'KitchenQual': 1,
 'TotRmsAbvGrd': 0,
 'Functional': 2,
 'Fireplaces': 0,
```



```
 'GarageYrBlt': 0,
```

```
 'GarageFinish': 78,
 'GarageCars': 0,
 'GarageArea': 0,
 'GarageQual': 78,
 'GarageCond': 78,
 'PavedDrive': 0,
 'WoodDeckSF': 0,
 'OpenPorchSF': 0,
 'EnclosedPorch': 0,
 '3SsnPorch': 0,
 'ScreenPorch': 0,
 'PoolArea': 0,
 'MiscVal': 0,
 'MoSold': 0,
 'YrSold': 0,
 'SaleType': 1,
 'SaleCondition': 0}
```

In [ ]:
```
train_data_dropped = train_data_dropped.dropna()
test_data_dropped = test_data_dropped.dropna()
```

I first saw for the most null valued columns , those who had at least 40 percent of the missing values than the total number of observations, I dropped those columns from test and train data both, because the model could not learn if we would try to fill these columns with any strategy.

This way we had columns with few number of rows. Now, I filled the numeric columns with the median of those columns, and dropped the rows, where the categorial features were missing, but these number of missing in categorial features is very few , and dropping those observations does not affect the size of the dataset.

Section 2: Data Preprocessing

1. Feature Engineering

In [ ]:
```
import datetime

train_data_dropped['TotalFlrSF'] = train_data_dropped['1stFlrSF'] + train_data_drop
test_data_dropped['TotalFlrSF'] = test_data_dropped['1stFlrSF'] + test_data_dropped
```

In [ ]:
```
train_data_dropped['SFRatio'] = train_data_dropped['TotalBsmtSF'] / train_data_drop
test_data_dropped['SFRatio'] = test_data_dropped['TotalBsmtSF'] / test_data_dropped
```

We had `TotalBsmtSF` in the dataset, but we had not `TotalFlrSF` for the two floors SF , that's why I created `TotalFlrSF` by adding the `1stFlrSF` and `2ndFlrSF`

furthermore, I could take out the SF ratio by dividing the `TotalBsmtSF` / `TotalFlrSF`

2. Encode Categorical variables

In [ ]:
```
X = train_data_dropped.drop(columns='SalePrice')
y = train_data_dropped[['SalePrice']]
```

In [ ]:
```
encoder = OneHotEncoder()
X_encod = encoder.fit_transform(X)
test_data_dropped_encod = encoder.fit(test_data_dropped)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_encod, y, test_size=0.2, rand
```

I have chose the 20 percent for test_size, becuase we already don't have enough number of observtions, ideally this proportion is correct to validate the model as a winner among others, the train.csv data will be used as unseen data.

Section 3: Regression Models

1. Linear Regression

In [ ]:
```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
```

In [ ]:
```
mse = mean_squared_error(y_test,y_pred_lr)
r2 = r2_score(y_test, y_pred_lr)
print("MSE: ", mse)
print("R2 Score: ", r2)
```

```
MSE:  1329249158.9942062
R2 Score:  0.7872987998251041
```

2. Advanced Regression

In [ ]:
```
ridge = RidgeCV(alphas=[0.001, 0.01, 0.1, 1, 10], cv=5)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)
```

In [ ]:
```
print("MSE: ", mse_ridge)
print("R2 Score: ", r2_ridge)
```

```
MSE:  1305434412.7827523
R2 Score:  0.7911095414507521
```

The model has slightly performed better than the Linear Regression, but not much, thus we can't see the big diiference in performance

3. Ensemble Method

In [ ]:
```
rf = RandomForestRegressor(random_state=0)
rf.fit(X_train, y_train)
```

```
# y_pred_rf = rf.predict(X_test)
```

```
C:\Users\Saadu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\base.py:1151: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

In [ ]:
```
y_pred_rf = rf.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print("MSE_rf: ", mse_rf)
print("r2_rf: ", r2_rf)
```

```
MSE_rf:  1191964263.3547525
r2_rf:  0.8092665865796146
```

The Random Forst Model has performed the so far best, by R2 score of acceptable 0.80. Moeover, the MSE compare to other two models is less.

Section 4: Logistic Regression

1. Transform into Classification Task

In [ ]:
```
logistic = LogisticRegression()
```

2. Logistic Regression

In [ ]:
```
logistic.fit(X_train, y_train)
y_pred_logistic = logistic.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_logistic)
conf_matrix = confusion_matrix(y_test, y_pred_logistic)
```

```
C:\Users\Saadu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Saadu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [ ]:
```
print('logistic Accuracy: ', accuracy)
```

```
logistic Accuracy:  0.007407407407407408
```

3. Discussion

Challenges:

1. **Loss of information:** Converting continuous variables into discrete categories can lead to a loss of information, potentially affecting the accuracy of the model.
2. **Arbitrary binning:** Defining the boundaries between categories can be arbitrary, influencing the model's performance.
3. **Potential for bias:** The binning process can introduce bias if not done carefully, skewing the model's predictions.

Advantages:

1. **Interpretability:** Classification models are often easier to interpret than regression models, making it simpler to understand the relationship between features and the target variable.
2. **Computational efficiency:** Classification algorithms can be computationally more efficient than regression algorithms, especially for large datasets.
3. **Handling outliers:** Classification models are less sensitive to outliers compared to regression models.

Overall, transforming a regression problem into a classification approach can be a useful strategy in certain situations, but it's crucial to weigh the potential loss of information and the challenges of binning against the advantages of interpretability, efficiency, and outlier handling.

Section 5: Advanced Concepts

1. Regularization

In [ ]:
```
reg_logistic = LogisticRegression(penalty='l2')
```

In [ ]:
```
reg_logistic.fit(X_train, y_train)
y_pred_reg_logistic = reg_logistic.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_reg_logistic)
```

```
C:\Users\Saadu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Saadu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [ ]:
```
print("L2 accuracy: ", accuracy)
```

2. Cross-Validation and Hyperparameter Tuning

In [ ]:
```
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}
```

In [ ]:
```
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, n_jobs=-1)
grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
c:\Users\Saadu\OneDrive\Documents\Competition\AI Nexus\notebook.ipynb Cell 65 line 2
      <a href='vscode-notebook-cell:/c%3A/Users/Saadu/OneDrive/Documents/Competitio
n/AI%20Nexus/notebook.ipynb#Y120sZmlsZQ%3D%3D?line=0'>1</a> grid_rf = GridSearchCV(r
f, param_grid_rf, cv=5, n_jobs=-1)
----> <a href='vscode-notebook-cell:/c%3A/Users/Saadu/OneDrive/Documents/Competitio
n/AI%20Nexus/notebook.ipynb#Y120sZmlsZQ%3D%3D?line=1'>2</a> grid_rf.fit(X_train, y_t
rain)
      <a href='vscode-notebook-cell:/c%3A/Users/Saadu/OneDrive/Documents/Competitio
n/AI%20Nexus/notebook.ipynb#Y120sZmlsZQ%3D%3D?line=2'>3</a> best_rf = grid_rf.best_e
stimator_
      <a href='vscode-notebook-cell:/c%3A/Users/Saadu/OneDrive/Documents/Competitio
n/AI%20Nexus/notebook.ipynb#Y120sZmlsZQ%3D%3D?line=3'>4</a> y_pred_rf = best_rf.pred
ict(X_test)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\Loc
alCache\local-packages\Python311\site-packages\sklearn\base.py:1151, in _fit_contex
t.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\Loc
alCache\local-packages\Python311\site-packages\sklearn\model_selection\_search.py:89
8, in BaseSearchCV.fit(self, X, y, groups, **fit_params)
    892     results = self._format_results(
    893         all_candidate_params, n_splits, all_out, all_more_results
    894     )
    896     return results
--> 898 self._run_search(evaluate_candidates)
    900 # multimetric is determined here because in the case of a callable
    901 # self.scoring the return type is only known after calling
    902 first_test_score = all_out[0]["test_scores"]

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\Loc
alCache\local-packages\Python311\site-packages\sklearn\model_selection\_search.py:14
19, in GridSearchCV._run_search(self, evaluate_candidates)
   1417 def _run_search(self, evaluate_candidates):
   1418     """Search all candidates in param_grid"""
-> 1419     evaluate_candidates(ParameterGrid(self.param_grid))

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\Loc
alCache\local-packages\Python311\site-packages\sklearn\model_selection\_search.py:84
5, in BaseSearchCV.fit.<locals>.evaluate_candidates(candidate_params, cv, more_resul
ts)
    837 if self.verbose > 0:
    838     print(
    839         "Fitting {0} folds for each of {1} candidates,"
    840         " totalling {2} fits".format(
                                    _candidates, n_candidates * n_splits
    842         )
```

```
    843     )
--> 845 out = parallel(
    846     delayed(_fit_and_score)(
    847         clone(base_estimator),
    848         X,
    849         y,
    850         train=train,
    851         test=test,
    852         parameters=parameters,
    853         split_progress=(split_idx, n_splits),
    854         candidate_progress=(cand_idx, n_candidates),
    855         **fit_and_score_kwargs,
    856     )
    857     for (cand_idx, parameters), (split_idx, (train, test)) in product(
    858         enumerate(candidate_params), enumerate(cv.split(X, y, groups))
    859     )
    860 )
    862 if len(out) < 1:
    863     raise ValueError(
    864         "No fits were performed. "
    865         "Was the CV iterator empty? "
    866         "Were there no candidates?"
    867     )

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\utils\parallel.py:65, in Parallel.__call__(self, iterable)
     60 config = get_config()
     61 iterable_with_config = (
     62     (_with_config(delayed_func, config), args, kwargs)
     63     for delayed_func, args, kwargs in iterable
     64 )
---> 65 return super().__call__(iterable_with_config)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\joblib\parallel.py:1944, in Parallel.__call__(self, iterable)
   1938 # The first item from the output is blank, but it makes the interpreter
   1939 # progress until it enters the Try/Except block of the generator and
   1940 # reach the first `yield` statement. This starts the aynchronous
   1941 # dispatch of the tasks to the workers.
   1942 next(output)
-> 1944 return output if self.return_generator else list(output)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\joblib\parallel.py:1587, in Parallel._get_outputs(self, iterator, pre_dispatch)
   1584     yield
   1586     with self._backend.retrieval_context():
-> 1587         yield from self._retrieve()
   1589 except GeneratorExit:
   1590     # The generator has been garbage collected before being fully
   1591     # consumed. This aborts the remaining tasks if possible and warn
   1592     # the user if necessary.
   1593     self._exception = True
```

```
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\joblib\parallel.py:1699, in Parallel._retrieve(self)
   1694 # If the next job is not ready for retrieval yet, we just wait for
   1695 # async callbacks to progress.
   1696 if ((len(self._jobs) == 0) or
   1697     (self._jobs[0].get_status(
   1698         timeout=self.timeout) == TASK_PENDING)):
-> 1699     time.sleep(0.01)
   1700     continue
   1702 # We need to be careful: the job list can be filling up as
   1703 # we empty it and Python list are not thread-safe by
   1704 # default hence the use of the lock

KeyboardInterrupt:
```

Note: the Hyperparametred tuning could not be run due to 35 mins extensive time

In [ ]:
```
print("MSE tuned Rf: ", mse_rf)
print("R2_Score: ", r2_score)
```

3. Bias-Variance Tradeoff

Linear and Logistic Regression: Offer simplicity and low variance, but risk high bias from simplifying assumptions. Ideal for small datasets or those with simple relationships.

Random Forests: Powerfully capture complex relationships with low bias, but can overfit, leading to high variance. A good choice for larger datasets with complex relationships.

L2 Regularization: Added to linear and logistic regressions, L2 prevents overfitting, reducing variance but potentially increasing bias.

Choosing the optimal model involves balancing these tradeoffs, considering both model complexity and data characteristics.

For instance, L2-regularized Logistic Regression might offer a good balance between model complexity and training data size.

In conclusion, identifying the model with the best balance between low bias and low variance results in optimized error and generalization.

Reducing Bias:

- **Bigger and better data:** Give your model more examples to learn from.
- **Fancier models:** Capture complex relationships, but watch out for overfitting.
- **Feature tuning:** Create more relevant features for better predictions.

Reducing Variance:

- **Regularization:** Penalize complex models to prevent overfitting.
- **Ensemble learning:** Combine multiple models for more accurate predictions.

- **Early stopping:** Avoid overfitting by stopping training early.