FLIP ROBO

HOUSING: PRICE PREDICTION

Submitted by:

Dipak Someshwar

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Sapna Verma.

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thank and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Dipak Someshwar.

INTRODUCTION

- Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate.

- market is one of the markets which is one of the major contributors in the world’s economy. It is a very large market and there are various companies working in the domain.

- Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases.

- Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies.

- The company is looking at prospective properties to buy houses to enter the market. You are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know:

Analytical Problem Framing

- Import library and load the dataset

```
# import python libraries

# data analysis
import numpy as np
import pandas as pd

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# sklearn utilities
from sklearn.feature_selection import VarianceThreshold
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
# prediction
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor

import warnings
warnings.filterwarnings('ignore')
```

```
# Loading the dataset:
test_data = pd.read_csv("train.csv")
train_data= pd.read_csv("test.csv")
```

```
train_test_data = [train_data, test_data]
print('Training data shape: ', train_data.shape)
print('Test data shape: ', test_data.shape)
```

```
Training data shape:  (292, 80)
Test data shape:  (1168, 81)
```

```
test_data.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	Mo
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	889	20	RL	95.00	15865	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	793	60	RL	92.00	9920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	110	20	RL	105.00	11751	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

- Display all column name of dataset.

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1168 non-null   int64
 1   MSSubClass     1168 non-null   int64
 2   MSZoning       1168 non-null   object
 3   LotFrontage    954 non-null    float64
 4   LotArea        1168 non-null   int64
 5   Street         1168 non-null   object
 6   Alley          77 non-null     object
 7   LotShape       1168 non-null   object
 8   LandContour    1168 non-null   object
 9   Utilities      1168 non-null   object
 10  LotConfig      1168 non-null   object
 11  LandSlope      1168 non-null   object
 12  Neighborhood   1168 non-null   object
 13  Condition1     1168 non-null   object
 14  Condition2     1168 non-null   object
 15  BldgType       1168 non-null   object
 16  HouseStyle     1168 non-null   object
 17  OverallQual    1168 non-null   int64
 18  OverallCond    1168 non-null   int64
 19  YearBuilt      1168 non-null   int64
 20  YearRemodAdd   1168 non-null   int64
 21  RoofStyle      1168 non-null   object
 22  RoofMatl       1168 non-null   object
 23  Exterior1st    1168 non-null   object
```

```
train_data.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeatu
0	337	20	RL	86.00	14157	Pave	NaN	IR1	HLS	AllPub	...	0	0	NaN	NaN	Na
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	Na
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	NaN	Na
3	1148	70	RL	75.00	12000	Pave	NaN	Reg	Bnk	AllPub	...	0	0	NaN	NaN	Na
4	1227	60	RL	86.00	14598	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	Na

5 rows × 80 columns

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 292 entries, 0 to 291
Data columns (total 80 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Id            292 non-null    int64
 1   MSSubClass    292 non-null    int64
 2   MSZoning      292 non-null    object
 3   LotFrontage   247 non-null    float64
 4   LotArea       292 non-null    int64
 5   Street        292 non-null    object
 6   Alley         14 non-null     object
```

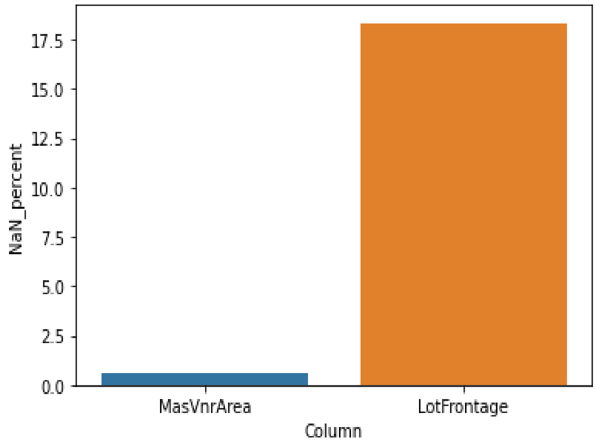
- Display statistical summary.

```
test_data.describe().style.background_gradient()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1168.000000	1168.000000	954.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1161.000000	1168.000000	1168.000000
mean	724.136130	56.767979	70.988470	10484.749144	6.104452	5.595890	1970.930651	1984.758562	102.310078	444.726027	46.647260
std	416.159877	41.940650	24.828750	8957.442311	1.390153	1.124343	30.145255	20.785185	182.595606	462.664785	163.520016
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000
25%	360.500000	20.000000	60.000000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000
50%	714.500000	50.000000	70.000000	9522.500000	6.000000	5.000000	1972.000000	1993.000000	0.000000	385.500000	0.000000
75%	1079.500000	70.000000	80.000000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000
max	1460.000000	190.000000	313.000000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

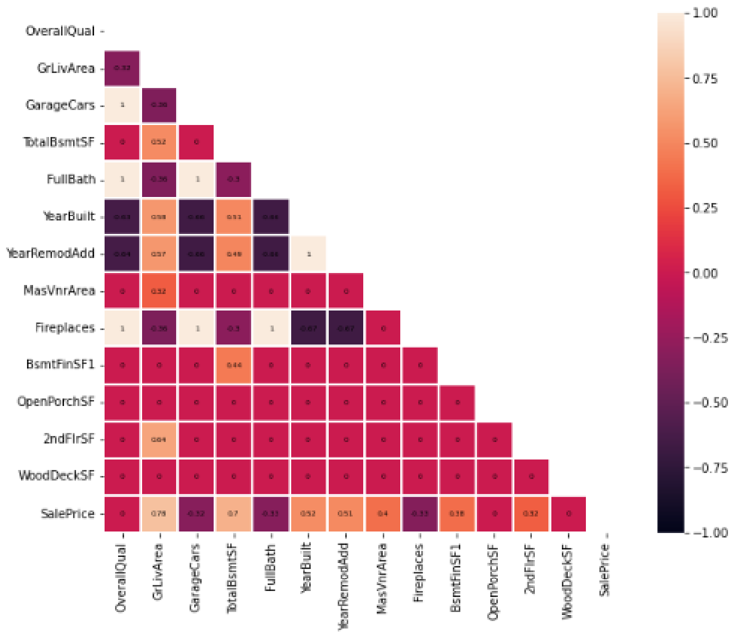
- Display barplot of all columns.

```
test_nan_cols = test_data_num.columns[test_data_num.isnull().any()].tolist()
test_nan = pd.DataFrame({ 'Column': test_nan_cols,
                          'NaN_percent': [ test_data_num[col].isnull().sum() * 100 / len(test_data_num)
                                          for col in test_nan_cols] })
sns.barplot(data=test_nan, x='Column', y='NaN_percent');
```



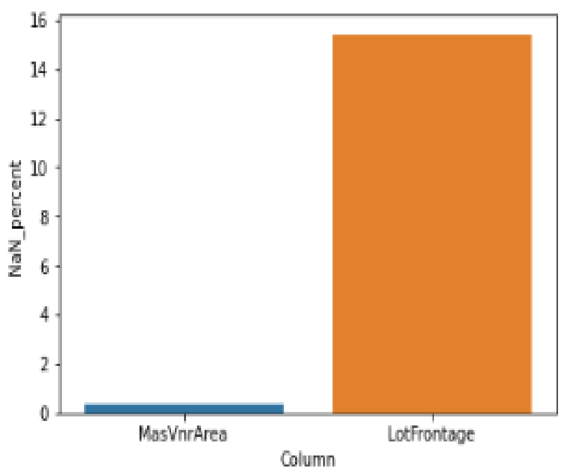
- Display correlation of columns using heatmap.

```python
pd.options.display.float_format = "{:,.2f}".format
corr_mat = test_data_num.corr('pearson')
# replace very weak correlation
corr_mat[(corr_mat < 0.3) & (corr_mat > -0.3)] = 0
# define triangular mask for better visibility
mask = np.triu(np.ones_like(corr_mat, dtype=bool))
plt.figure(figsize=(10, 8))
sns.heatmap(corr_mat, mask=mask, vmax=1.0, vmin=-1.0, square=True, annot=True, annot_kws={"size": 6, "color": "black"}, linewidt
```



- Display barplot of all columns.

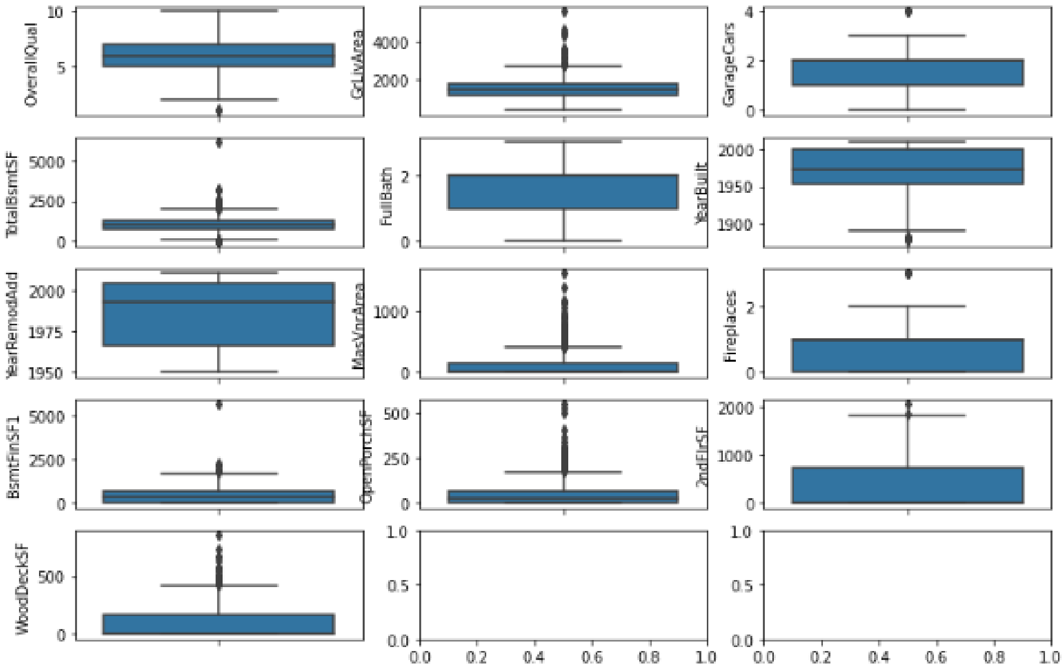
```python
train_nan_cols = train_data_num.columns[train_data_num.isnull().any()].tolist()
train_nan = pd.DataFrame({ 'Column': train_nan_cols,
                           'NaN_percent': [ train_data_num[col].isnull().sum() * 100 / len(train_data_num)
                                            for col in train_nan_cols] })
sns.barplot(data=train_nan, x='Column', y='NaN_percent');
```



- Display outliers of all columns.

```
# outliers
fig, ax = plt.subplots(5, 3, figsize=(12, 8))
test_num_cols = test_data_num.columns.tolist()[:-1]

for i, ax in enumerate(fig.axes):
    if i < len(test_num_cols):
        sns.boxplot(data=test_data_num, y=test_num_cols[i], ax=ax)
```

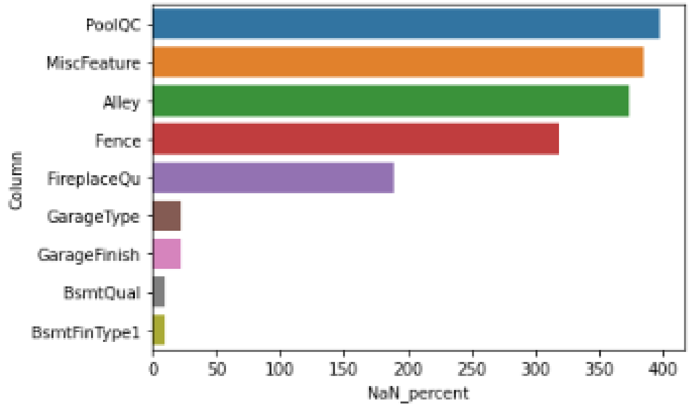


- Filling empty values.

```
# filling empty values:
cols_null_test = test_data_cat.columns[test_data_cat.isnull().any()]

nan_counts_train = pd.DataFrame({
    'Column': cols_null_test,
    'NaN_percent': [test_data_cat[col].isnull().sum()*100 / len(train_data_cat)
                    for col in cols_null_test]
})

nan_counts_train.sort_values('NaN_percent', ascending=False, inplace=True, ignore_index=True)
sns.barplot(data=nan_counts_train, y='Column', x='NaN_percent');
```



Model/s Development and Evaluation

- **Feature engeenering:**

```
# Feature engeenering:
# Age of house from the year of construction
train_data_new['Age'] = train_data_new['YearBuilt'].max() - train_data_new['YearBuilt']
test_data_new['Age'] = test_data_new['YearBuilt'].max() - test_data_new['YearBuilt']
```

```
# Age since renovating
train_data_new['Renovate'] = train_data_new['YearRemodAdd'] - train_data_new['YearBuilt']
test_data_new['Renovate'] = test_data_new['YearRemodAdd'] - test_data_new['YearBuilt']

train_data_new['Renovate'] = np.where(train_data_new['Renovate'] < 0, 0, train_data_new['Renovate'])
test_data_new['Renovate'] = np.where(test_data_new['Renovate'] < 0, 0, test_data_new['Renovate'])

# Drop YearBuilt
train_data_new.drop(['YearBuilt'], axis=1, inplace=True)
test_data_new.drop(['YearBuilt'], axis=1, inplace=True)
# Drop YearRemodAdd
train_data_new.drop(['YearRemodAdd'], axis=1, inplace=True)
test_data_new.drop(['YearRemodAdd'], axis=1, inplace=True)
```

```
# Artificial feature combines OverallQual and GrLivArea
train_data_new['Qual_Area'] = train_data_new['OverallQual'] * train_data_new['GrLivArea']
test_data_new['Qual_Area'] = test_data_new['OverallQual'] * test_data_new['GrLivArea']
```

```
cont_features = test_data_new.select_dtypes(include=['int', 'float']).drop(['SalePrice'], axis=1).columns.tolist()

cont_data = test_data_new.loc[:, cont_features]
cont_data.head()
```

- Testing of Identified Approaches (Algorithms)

```
# Modeling
#Preparing data
X = test_data_new.drop(['SalePriceLog'], axis=1)
y = test_data_new['SalePriceLog']

print('X shape: ', X.shape)
print('y shape: ', y.shape)
```

```
X shape:  (1168, 153)
y shape:  (1168,)
```

```
# Standardize data
scaler = StandardScaler().fit(X)
```

```
import statsmodels.api as sm

def backward_elimination(X, y, threshold=0.05):
    features = X.columns.tolist()

    while True:
        changed = False
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[features]))).fit()
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max()
        if worst_pval > threshold:
            changed = True
            worst_fet = pvalues.idxmax()
            features.remove(worst_fet)
        if not changed:
            break

    return features
```



```
selected_features = backward_elimination(X, y)
selected_features
```

- Run and Evaluate selected models:

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15)
print('Train size:', X_train.shape, y_train.shape)
print('Validation size:', X_val.shape, y_val.shape)
```

```
Train size: (992, 80) (992,)
Validation size: (176, 80) (176,)
```

```
# Creating RMSE

def rmse_score(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# Creating estimating function

r2_list = []
rmse_list = []

def get_metrics(model):
    r2 = model.score(X_val, y_val)
    rmse = rmse_score(y_val, model.predict(X_val))
    r2_list.append(r2)
    rmse_list.append(rmse)
    print('Cross validation score:', cross_val_score(model, X_train, y_train, cv=5))
    print('R2 score:', r2)
    print('RMSE:', rmse)
```

```
#Linear Regression:
linreg = LinearRegression()
linreg.fit(X_train, y_train)

get_metrics(linreg)
```

```
Cross validation score: [0.99441619 0.98766306 0.99606524 0.99097402 0.99331225]
R2 score: 0.9931933012994151
RMSE: 0.16921169213566375
```

- Creating RMSE:

```
# Creating RMSE

def rmse_score(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# Creating estimating function

r2_list = []
rmse_list = []

def get_metrics(model):
    r2 = model.score(X_val, y_val)
    rmse = rmse_score(y_val, model.predict(X_val))
    r2_list.append(r2)
    rmse_list.append(rmse)
    print('Cross validation score:', cross_val_score(model, X_train, y_train, cv=5))
    print('R2 score:', r2)
    print('RMSE:', rmse)
```

```
#Linear Regression:
linreg = LinearRegression()
linreg.fit(X_train, y_train)

get_metrics(linreg)
```

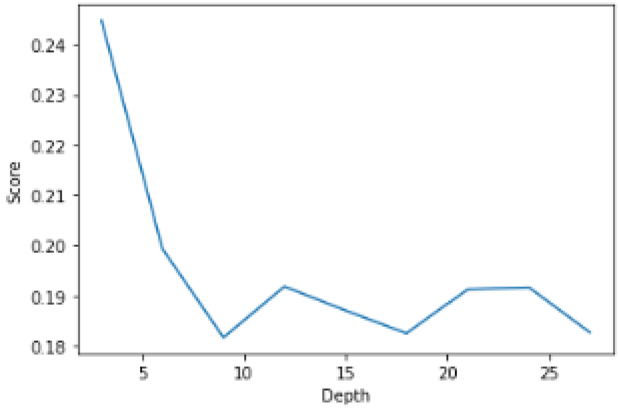
```
Cross validation score: [0.99441619 0.98766306 0.99606524 0.99097402 0.99331225]
R2 score: 0.9931933012994151
RMSE: 0.16921169213566375
```

Decision Tree:

```
# Decision Tree:
depths = []
scores = []

for d in range(3, 30, 3):
    m = DecisionTreeRegressor(max_depth=d).fit(X_train, y_train)
    depths.append(d)
    scores.append(rmse_score(y_val, m.predict(X_val)))

dt_scores = pd.DataFrame({
    'Depth': depths,
    'Score': scores
})
sns.lineplot(data=dt_scores, x='Depth', y='Score');
```



- ## Interpretation of the Results:

```
model_list = ['linreg', 'ridge', 'lasso', 'svr', 'dt', 'rf', 'xgb', 'gbr', 'cbr']

summary = pd.DataFrame({
    'Model': model_list,
    'R2': r2_list,
    'RMSE': rmse_list
})
summary.sort_values('RMSE')
```

	Model	R2	RMSE
7	gbr	1.00	0.13
5	rf	1.00	0.14
8	cbr	1.00	0.14
6	xgb	0.99	0.15
0	linreg	0.99	0.17
1	ridge	0.99	0.17
4	dt	0.99	0.18
2	lasso	0.99	0.18
3	svr	0.89	0.67

Hardware and Software Requirements and Tools Used:

- **Language :-** Python
- **Tool:-** Jupyter
- **OS:-** Windows 10
- **RAM:-** 8gb

CONCLUSION:

This Kernel investigates different models for housing price prediction. Different types of Machine Learning methods including CatBoostRegressor, GradientBoostingRegressor and LightGBM and two techniques in machine learning are compared and analyzed for optimal solutions. Even though all of those methods achieved desirable results, different models have their own pros and cons.

The GradientBoostingRegressor is probably the best one and has been selected for this problem. The BayesianOptimization method is simple but performs lot better than the three other availabel methods due to the generalization.

Finally, the CatBoostRegressor is the best choice when parameterization is the top priority.