Final Project Report

Usable-AI
SP22-BL-INFO-I513-14356

Kiran Naik
Sowmya Priya

Subject of Focus:

Sale Price Prediction

Data Collection:

Data collection for this analysis is taken from the Kaggle.com website under competitions section- https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques . Datasets consist of the train and test data. This dataset is not changed and can be downloaded any time from the Kaggle Website. Below are the attributes(features) of dataset downloaded as CSV.

Id	YearRemodAdd	HeatingQC	GarageFinish
MSSubClass	RoofStyle	CentralAir	GarageCars
MSZoning	RoofMatl	Electrical	GarageArea
LotFrontage	Exterior1st	1stFlrSF	GarageQual
LotArea	Exterior2nd	2ndFlrSF	GarageCond
Street	MasVnrType	LowQualFinSF	PavedDrive
Alley	MasVnrArea	GrLivArea	WoodDeckSF
LotShape	ExterQual	BsmtFullBath	OpenPorchSF
LandContour	ExterCond	BsmtHalfBath	EnclosedPorch
Utilities	Foundation	FullBath	3SsnPorch
LotConfig	BsmtQual	HalfBath	ScreenPorch
LandSlope	BsmtCond	BedroomAbvGr	PoolArea
Neighborhood	BsmtExposure	KitchenAbvGr	PoolQC
Condition1	BsmtFinType1	KitchenQual	Fence
Condition2	BsmtFinSF1	TotRmsAbvGrd	MiscFeature
BldgType	BsmtFinType2	Functional	MiscVal
HouseStyle	BsmtFinSF2	Fireplaces	MoSold
OverallQual	BsmtUnfSF	FireplaceQu	YrSold
OverallCond	TotalBsmtSF	GarageType	SaleType
YearBuilt	Heating	GarageYrBlt	SaleCondition
SalePrice			

Data Management:

Post data collection, data cleaning is performed on the csv data using Python. Below are the python modules used to clean and recode few variables.

Python Modules
Pandas
NumPy
Pandas
matplotlib
Altair
sklearn. metrics
seaborn

The CSV data has been cleaned and all the string values have been converted to numeric. Below code explains everything on the same. Conversion of all the categorical values will help in the later statistics. No other data was merged or manually updated with the Kaggle dataset.

```
import pandas as pd
import numpy as np
import altair as alt
import matplotlib.pyplot as plt
import seaborn as sns

filename="train.csv"
df = pd.read_csv(filename)

import numpy

def getEncodings(dataframe):
    encodings = {}
    for column in dataframe.columns:
        column_values = df[column]

        if isinstance(column_values[0], str) or column=="Alley" or
column=="FireplaceQu" or column=="PoolQC" or column=="Fence" or
column=="MiscFeature":
            encoder = {}
            counter = 1
            for value in column_values.unique():
                encoder[value]=counter
```

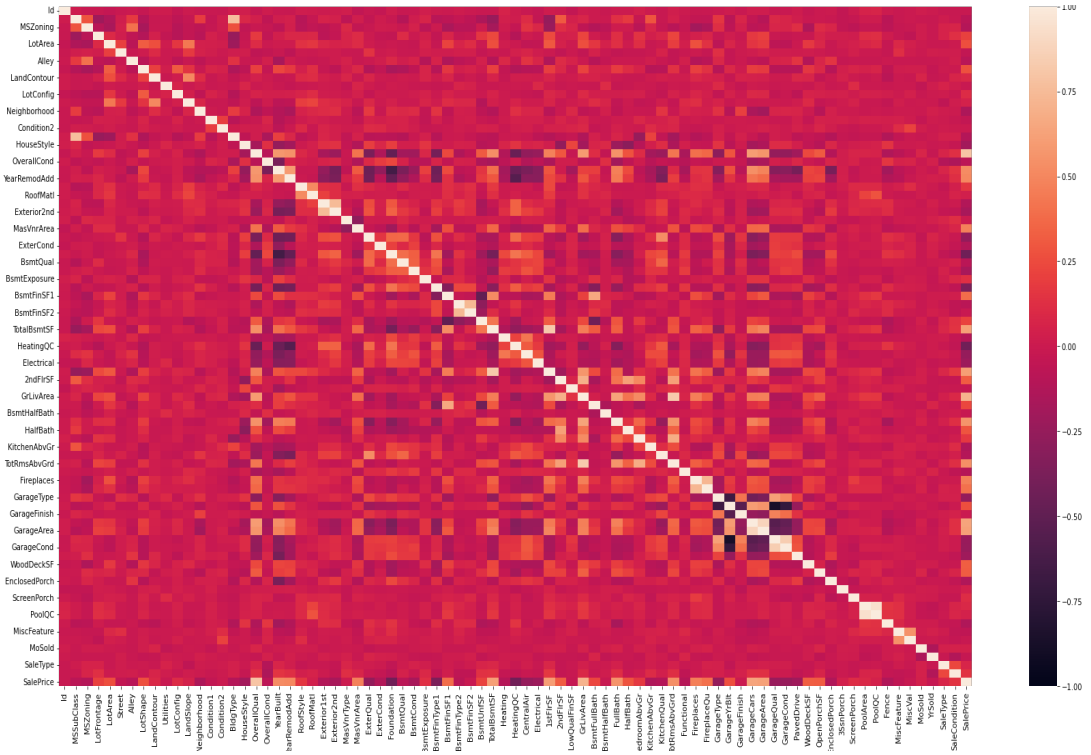
```
                counter = counter + 1
            encodings[column]= encoder
    return encodings

encodings = getEncodings(df) # conversion of each string values to the
number.
print(encodings)


def applyEncodings(dataframe,encodings):
    return dataframe.replace(encodings)
clean_column=applyEncodings(df,encodings).fillna(0)


OUTPUT:
```

{'MSZoning': {'RL': 1, 'RM': 2, 'C (all)': 3, 'FV': 4, 'RH': 5}, 'Street': {'
Pave': 1, 'Grvl': 2}, 'Alley': {nan: 1, 'Grvl': 2, 'Pave': 3}, 'LotShape': {'
Reg': 1, 'IR1': 2, 'IR2': 3, 'IR3': 4}, 'LandContour': {'Lvl': 1, 'Bnk': 2, '
Low': 3, 'HLS': 4}, 'Utilities': {'AllPub': 1, 'NoSeWa': 2}, 'LotConfig': {'I
nside': 1, 'FR2': 2, 'Corner': 3, 'CulDSac': 4, 'FR3': 5}, 'LandSlope': {'Gtl
': 1, 'Mod': 2, 'Sev': 3}, 'Neighborhood': {'CollgCr': 1, 'Veenker': 2, 'Craw
for': 3, 'NoRidge': 4, 'Mitchel': 5, 'Somerst': 6, 'NWAmes': 7, 'OldTown': 8,
'BrkSide': 9, 'Sawyer': 10, 'NridgHt': 11, 'NAmes': 12, 'SawyerW': 13, 'IDOTR
R': 14, 'MeadowV': 15, 'Edwards': 16, 'Timber': 17, 'Gilbert': 18, 'StoneBr':
19, 'ClearCr': 20, 'NPkVill': 21, 'Blmngtn': 22, 'BrDale': 23, 'SWISU': 24, '
Blueste': 25}, 'Condition1': {'Norm': 1, 'Feedr': 2, 'PosN': 3, 'Artery': 4,
'RRAe': 5, 'RRNn': 6, 'RRAn': 7, 'PosA': 8, 'RRNe': 9}, 'Condition2': {'Norm'
: 1, 'Artery': 2, 'RRNn': 3, 'Feedr': 4, 'PosN': 5, 'PosA': 6, 'RRAn': 7, 'RR
Ae': 8}, 'BldgType': {'1Fam': 1, '2fmCon': 2, 'Duplex': 3, 'TwnhsE': 4, 'Twnh
s': 5}, 'HouseStyle': {'2Story': 1, '1Story': 2, '1.5Fin': 3, '1.5Unf': 4, 'S
Foyer': 5, 'SLvl': 6, '2.5Unf': 7, '2.5Fin': 8}, 'RoofStyle': {'Gable': 1, 'H
ip': 2, 'Gambrel': 3, 'Mansard': 4, 'Flat': 5, 'Shed': 6}, 'RoofMatl': {'Comp
Shg': 1, 'WdShngl': 2, 'Metal': 3, 'WdShake': 4, 'Membran': 5, 'Tar&Grv': 6,
'Roll': 7, 'ClyTile': 8}, 'Exterior1st': {'VinylSd': 1, 'MetalSd': 2, 'Wd Sdn
g': 3, 'HdBoard': 4, 'BrkFace': 5, 'WdShing': 6, 'CemntBd': 7, 'Plywood': 8,
'AsbShng': 9, 'Stucco': 10, 'BrkComm': 11, 'AsphShn': 12, 'Stone': 13, 'ImStu
cc': 14, 'CBlock': 15}, 'Exterior2nd': {'VinylSd': 1, 'MetalSd': 2, 'Wd Shng'
: 3, 'HdBoard': 4, 'Plywood': 5, 'Wd Sdng': 6, 'CmentBd': 7, 'BrkFace': 8, 'S
tucco': 9, 'AsbShng': 10, 'Brk Cmn': 11, 'ImStucc': 12, 'AsphShn': 13, 'Stone
': 14, 'Other': 15, 'CBlock': 16}, 'MasVnrType': {'BrkFace': 1, 'None': 2, 'S
tone': 3, 'BrkCmn': 4, nan: 5}, 'ExterQual': {'Gd': 1, 'TA': 2, 'Ex': 3, 'Fa'
: 4}, 'ExterCond': {'TA': 1, 'Gd': 2, 'Fa': 3, 'Po': 4, 'Ex': 5}, 'Foundation
': {'PConc': 1, 'CBlock': 2, 'BrkTil': 3, 'Wood': 4, 'Slab': 5, 'Stone': 6},
'BsmtQual': {'Gd': 1, 'TA': 2, 'Ex': 3, nan: 4, 'Fa': 5}, 'BsmtCond': {'TA':
1, 'Gd': 2, nan: 3, 'Fa': 4, 'Po': 5}, 'BsmtExposure': {'No': 1, 'Gd': 2, 'Mn
': 3, 'Av': 4, nan: 5}, 'BsmtFinType1': {'GLQ': 1, 'ALQ': 2, 'Unf': 3, 'Rec':
4, 'BLQ': 5, nan: 6, 'LwQ': 7}, 'BsmtFinType2': {'Unf': 1, 'BLQ': 2, nan: 3,
'ALQ': 4, 'Rec': 5, 'LwQ': 6, 'GLQ': 7}, 'Heating': {'GasA': 1, 'GasW': 2, 'G
rav': 3, 'Wall': 4, 'OthW': 5, 'Floor': 6}, 'HeatingQC': {'Ex': 1, 'Gd': 2, '
TA': 3, 'Fa': 4, 'Po': 5}, 'CentralAir': {'Y': 1, 'N': 2}, 'Electrical': {'SB
rkr': 1, 'FuseF': 2, 'FuseA': 3, 'FuseP': 4, 'Mix': 5, nan: 6}, 'KitchenQual'

: {'Gd': 1, 'TA': 2, 'Ex': 3, 'Fa': 4}, 'Functional': {'Typ': 1, 'Min1': 2, '
Maj1': 3, 'Min2': 4, 'Mod': 5, 'Maj2': 6, 'Sev': 7}, 'FireplaceQu': {nan: 1,
'TA': 2, 'Gd': 3, 'Fa': 4, 'Ex': 5, 'Po': 6}, 'GarageType': {'Attchd': 1, 'De
tchd': 2, 'BuiltIn': 3, 'CarPort': 4, nan: 5, 'Basment': 6, '2Types': 7}, 'Ga
rageFinish': {'RFn': 1, 'Unf': 2, 'Fin': 3, nan: 4}, 'GarageQual': {'TA': 1,
'Fa': 2, 'Gd': 3, nan: 4, 'Ex': 5, 'Po': 6}, 'GarageCond': {'TA': 1, 'Fa': 2,
nan: 3, 'Gd': 4, 'Po': 5, 'Ex': 6}, 'PavedDrive': {'Y': 1, 'N': 2, 'P': 3}, '
PoolQC': {nan: 1, 'Ex': 2, 'Fa': 3, 'Gd': 4}, 'Fence': {nan: 1, 'MnPrv': 2, '
GdWo': 3, 'GdPrv': 4, 'MnWw': 5}, 'MiscFeature': {nan: 1, 'Shed': 2, 'Gar2':
3, 'Othr': 4, 'TenC': 5}, 'SaleType': {'WD': 1, 'New': 2, 'COD': 3, 'ConLD':
4, 'ConLI': 5, 'CWD': 6, 'ConLw': 7, 'Con': 8, 'Oth': 9}, 'SaleCondition': {'
Normal': 1, 'Abnorml': 2, 'Partial': 3, 'AdjLand': 4, 'Alloca': 5, 'Family':
6}}

To understand what columns are related to each other, we used the correlation function on the dataset and tried to analyze it via the heatmap.

```
#Sns. heatmap
plt.figure(figsize=(30, 15))
heatmap = sns.heatmap(clean_column.corr(), vmin=-1, vmax=1)
```



Positive Correlation			Positive Correlation		
Feature1	**Feature2**	**Correlation**	**Feature1**	**Feature2**	**Correlation**
2ndFlrSF	BedroomAbvGr	0.50290061	YearRemodAdd	YearBuilt	0.59285498

YearRemodAdd	SalePrice	0.50710097	GrLivArea	OverallQual	0.59300743
LandContour	LandSlope	0.50720325	GarageCars	GarageYrBlt	0.59800465
GarageType	GarageCond	0.50933536	OverallQual	GarageCars	0.60067072
RoofMatl	RoofStyle	0.50973252	SalePrice	1stFlrSF	0.60585218
GrLivArea	BedroomAbvGr	0.52126951	HalfBath	2ndFlrSF	0.6097073
BsmtFinSF1	TotalBsmtSF	0.52239605	GarageType	GarageQual	0.61301033
SalePrice	YearBuilt	0.52289733	SalePrice	TotalBsmtSF	0.61358055
KitchenQual	ExterQual	0.52837788	2ndFlrSF	TotRmsAbvGrd	0.61642264
TotRmsAbvGrd	SalePrice	0.53372316	GarageArea	SalePrice	0.62343144
TotalBsmtSF	OverallQual	0.5378085	FullBath	GrLivArea	0.63001165
YearBuilt	GarageCars	0.53785009	GarageCars	SalePrice	0.6404092
MiscFeature	MiscVal	0.54789083	BsmtFullBath	BsmtFinSF1	0.64921175
FullBath	OverallQual	0.55059971	TotRmsAbvGrd	BedroomAbvGr	0.67661994
YearRemodAdd	OverallQual	0.55068392	GrLivArea	2ndFlrSF	0.68750106
FullBath	TotRmsAbvGrd	0.55478425	SalePrice	GrLivArea	0.70862448
FullBath	SalePrice	0.56066376	GrLivArea	SalePrice	0.70862448
GarageArea	GarageYrBlt	0.56078287	Fireplaces	FireplaceQu	0.72439314
GarageArea	OverallQual	0.56202176	BsmtFinSF2	BsmtFinType2	0.72952382
GrLivArea	1stFlrSF	0.56602397	Exterior2nd	Exterior1st	0.74959947
YearBuilt	OverallQual	0.57232277	BldgType	MSSubClass	0.77083966
GarageQual	GarageCond	0.83839591	SalePrice	OverallQual	0.7909816
GarageCars	GarageArea	0.88247541	1stFlrSF	TotalBsmtSF	0.81952998
PoolQC	PoolArea	0.93840162	GrLivArea	TotRmsAbvGrd	0.82548937

Negative Correlation		
Feature1	**Feature2**	**Correlation**
GarageCond	GarageYrBlt	-0.722032
GarageType	GarageYrBlt	-0.6761107
Foundation	YearBuilt	-0.6478649
HeatingQC	YearRemodAdd	-0.5500173
GarageQual	GarageCars	-0.5444106
GarageYrBlt	GarageFinish	-0.5392388

As the Feature1 and Feature 2 has the higher correlation, we can remove few features and apply the predicting technic for Sale price. Below is the respective code and output.

In the above columns, the one that are marked in Red can be discarded as they are not adding any value to the predicting output (Sale Price)

For Instance: Fireplaces v/s FireplaceQU (Quality of fireplace), both are related, and Fireplace quality exist only if there is fireplace so we can remove Fireplaces.

In this way we have discarded few features, based on our lookup with housing data.

Below is the code snippet which drops the marked features.

```
# clean_column variable contains the values, after all the features values
are been set to numeric.


clean_column.drop(['MiscFeature','MiscVal','Fireplaces','BsmtFinType2','PoolA
rea','TotRmsAbvGrd','GarageYrBlt','Id'], inplace=True, axis=1)
features=clean_column #copy of the clean_column

main_target_price = clean_column.filter(['SalePrice']) #capturing the
predicting training data in separate variable

features.drop('SalePrice', inplace=True, axis=1) #drop the saleprice from the
dataframe.
```

Analysis:

During the analysis, we try to fit the training data to various models and try to predict the sale price of the inputs. Accuracy score let us know on how much the predicted data is matching with the actual data.

Initially we split our training data into two i.e., training and testing data. Training data will be used to train the model and testing data will be used to train the model.

The values need to be normalized and this will perform using the standard scalar function.

As the dimensions are high, we will try to reduce the dimensions(features) by applying Principal component analysis. As the data has been transformed with less dimensions, we can apply linear, logistic, K – Nearest neighbors etc. models and try to find the accuracy for each.

Below is the code snippet used to get the accuracy for each.

```
import pandas as pd
import numpy as np
import altair as alt
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier


# From the data management we can see that the target and features data are being separated. Same has been used here.

#linear regression

train_data, test_data, train_target, test_target = train_test_split(
features, main_target_price, test_size=0.3, random_state=0)


scaler = StandardScaler()

#Fit the data to standard scalar
scaler.fit(features)
train_data = scaler.transform(train_data)
test_data = scaler.transform(test_data)

#apply PCA to reduce the dimension with variance of 0.95
pca = PCA(0.95)
pca.fit(train_data)
print(pca.n_components_)

# transform the pca to test and train data
train_data = pca.transform(train_data)
test_data = pca.transform(test_data)

#apply linear regression model and check for the accuracy score

model = LinearRegression()
model.fit(train_data,train_target)
model.predict(test_data[0].reshape(1,-1))
model.score(test_data,test_target)
```

Output:

```
0.7123080763360028
```

```
#logistic regression
```

```
train_data, test_data, train_target, test_target = train_test_split(
features, main_target_price, test_size=0.3, random_state=0)


scaler = StandardScaler()

#Fit the data to standard scalar
scaler.fit(features)
train_data = scaler.transform(train_data)
test_data = scaler.transform(test_data)

#apply PCA to reduce the dimension with variance of 0.95
pca = PCA(0.95)
pca.fit(train_data)
pca.n_components_

# transform the pca to test and train data
train_data = pca.transform(train_data)
test_data = pca.transform(test_data)

logistic_regression = LogisticRegression(solver = 'lbfgs')
logistic_regression.fit(train_data,train_target)
logistic_regression.score(test_data, test_target)
```

Output:

```
0.0091324200913242
```

```
#SVM


train_data, test_data, train_target, test_target = train_test_split(
features, main_target_price, test_size=0.3, random_state=0)


scaler = StandardScaler()

#Fit the data to standard scalar
scaler.fit(features)
train_data = scaler.transform(train_data)
test_data = scaler.transform(test_data)

#apply PCA to reduce the dimension with variance of 0.95
pca = PCA(0.95)
pca.fit(train_data)
print("pca componenet",pca.n_components_)

# transform the pca to test and train data
train_data = pca.transform(train_data)
test_data = pca.transform(test_data)

#apply SVM
svm= SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(train_data,train_target)
```

```
pred_sale=svm.predict(test_data)
print(f'Accuracy: {accuracy_score(test_target, pred_sale)}')
```

Output:

```
Accuracy: 0.0091324200913242
```

```
#knn neighbors

train_data, test_data, train_target, test_target = train_test_split(
features, main_target_price, test_size=0.3, random_state=0)


scaler = StandardScaler()

#Fit the data to standard scalar
scaler.fit(features)
train_data = scaler.transform(train_data)
test_data = scaler.transform(test_data)

#apply PCA to reduce the dimension with variance of 0.95
pca = PCA(0.95)
pca.fit(train_data)
print("pca componenet",pca.n_components_)

# transform the pca to test and train data
train_data = pca.transform(train_data)
test_data = pca.transform(test_data)

#apply knn
knn= KNeighborsClassifier(n_neighbors=5,metric='euclidean')

knn.fit(train_data,train_target)

pred_sale= knn.predict(test_data)

print(f'Accuracy: {accuracy_score(test_target, pred_sale)}')
```

Output:

```
Accuracy: 0.00228310502283105
```

After this, we tried to work with Random Forest Classifier, so that we can discard few unnecessary features, in an urge of getting more higher accuracy and tried to see which feature contributes more.

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression(max_iter=150)
```

```
#print(iris)
scores = cross_val_score(logreg, iris.data, iris.target, cv=10)
print("10-fold cross validation accuracy scores: {}".format(scores))
print("10-fold cross validation average accuracy scores:
{:.2f}".format(scores.mean(axis=0)))
```

Output:

```
10-fold cross validation accuracy scores: [1.         0.93333333 1.         1
.         0.93333333 0.93333333
 0.93333333 1.         1.         1.        ]
10-fold cross validation average accuracy scores: 0.97
```

```
logreg = LinearRegression()

scores = cross_val_score(logreg, train_data,train_target, cv=10)
print("10-fold cross validation accuracy scores: {}".format(scores))
print("10-fold cross validation average accuracy scores:
{:.2f}".format(scores.mean(axis=0)))
```

Output:

```
10-fold cross validation accuracy scores: [0.86541225 0.6011684  0.83201005 0
.90878463 0.89406031 0.80093475
 0.86982517 0.83547309 0.9097381  0.83132779]
10-fold cross validation average accuracy scores: 0.83
```

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

select = SelectFromModel(RandomForestClassifier(n_estimators=30,
random_state=42))
select.fit(train_data, train_target)
X_train_rf = select.transform(train_data)
print("X_train.shape: {}".format(train_data.shape))
print("X_train_rf.shape: {}".format(X_train_rf.shape))
```

```
X_train.shape: (1022, 72)
X_train_rf.shape: (1022, 24)
```

```
mask = select.get_support()
#print(mask)

for i in range(0,len(mask)):
    if mask[i]:
        print(train_data.columns[i])
# selected_feat= train_data.columns[(select.get_support())]
# len(selected_feat)
```

```
24 selected features are
```

LotFrontage	Exterior2nd	GrLivArea
LotArea	MasVnrArea	BedroomAbvGr
Neighborhood	BsmtFinType1	FireplaceQu
OverallQual	BsmtFinSF1	GarageArea
OverallCond	BsmtUnfSF	WoodDeckSF
YearBuilt	TotalBsmtSF	OpenPorchSF
YearRemodAdd	1stFlrSF	MoSold
Exterior1st	2ndFlrSF	YrSold

```
lr = LinearRegression()
lr.fit(train_data, train_target)
print("Score with all features: {:.3f}".format(lr.score(test_data,
test_target)))

# transform test data
X_test_rf = select.transform(test_data)
score = lr.fit(X_train_rf, train_target).score(X_test_rf, test_target)
print("Score with only selected features: {:.3f}".format(score))
```

```
Score with all features: 0.712
Score with only selected features: 0.709
```

The score was almost similar.

Conclusion:

From above we could understand that the linear regression has the higher value of 0.712.

ARGUMENT:

Prediction can be highly accurate in some instances. However, there are certain houses where the predicted value is incorrect. This common in ML that model sometimes goes wrong. There might be many reasons like not enough data, there are frequent errors in the data and a model that is either too simple or too complex. In addition, let's look at one feature called Overall Quality, as well as other

features. Each has a varied coefficient since some characteristics contribute less than others. Overall Quality, among other factors, played a larger role in this. When tried with different models like Linear regression, Logistic regression, Support Vector Machine and KNN we got less accuracy. For better accuracy, we have tried fitting the data using Random Forest Classifier and Principle Component Analysis (PCA) for reducing the dimensions of features. All these were tested and these results in 70 percent of accuracy

STAKEHOLDER ANALYSIS:

House price analysis is targeted for home buyers and sellers. A home buyer would always want to get a good value for the house that he/she buys. One important question while buying a house is, for given home features, what is the expected price. Knowing this, they can prioritize the important features while buying same house. Sometimes, the buyer can also buy a house with less features for lower price and then after purchase they can remodify the home, so that the total cost is reduced. Hence the relation of different features to sales price and prediction of sales price, given the features are important.

For house sellers, it helps to understand the importance of certain features of house, so that they can get more profit while selling. For example, if the predicted sales price can go very high if the house has garage, then the seller may build the garage before selling the house.

Airbnb hosts are the new set of stakeholders, who might play a major role in this house market. Airbnb houses are getting more popular and the profit from renting the house is directly related to the features and the amenities that are provided. Hence Airbnb host may be more interested in sophisticated features while buying the house.

In all the above cases, selling price of house may appear as single importance variable, but the presence of a certain miscellaneous feature and their influence on selling price may also play an important role. This analysis is also targeted on analyzing such cases.

In all the above cases, stakeholders usually are general public or early Airbnb hosts, and they might have casual to frequent computer experience.

With this Analysis, they can clearly understand the importance of house features, relation between them and leverage this knowledge to improve the buying/selling experience.

CONTEXT:

For our use case, at first the data is cleaned and ready for analysis. First part of the analysis can help stakeholders to understand the relation between the different variables and how they co-relate with one another. As selling price of the house is an important dependent variable, these correlation plots can also help them to visually understand the linear relation between certain variables and Selling price.

With help of correlation plots, we have also explained how the certain features can be removed from the analysis, as they are redundant or irrelevant.

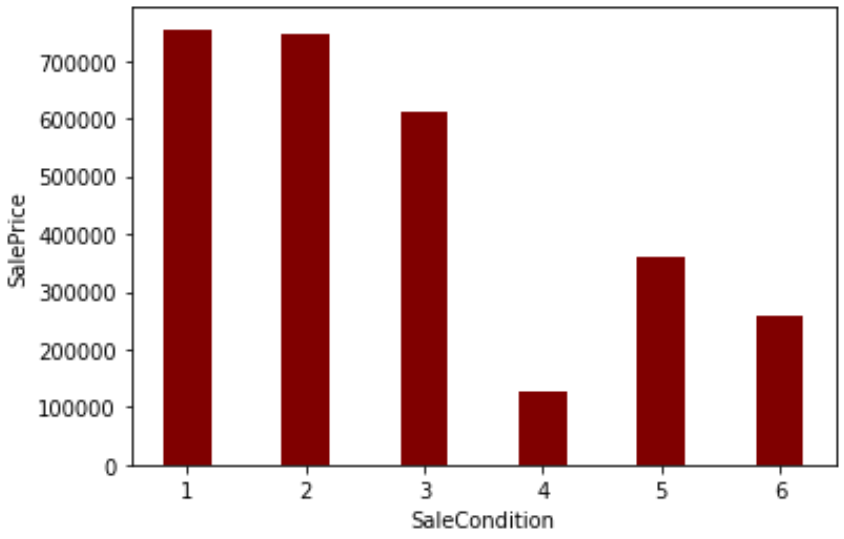
As this is public data, there is no privacy or trust issue that needs to be considered.

Because of the context of use, requirements to provide more clear data visualization and clear explanation of decision variables is raised up. We have also added few visualizations that are targeted for stakeholders.

Data Visualizations:

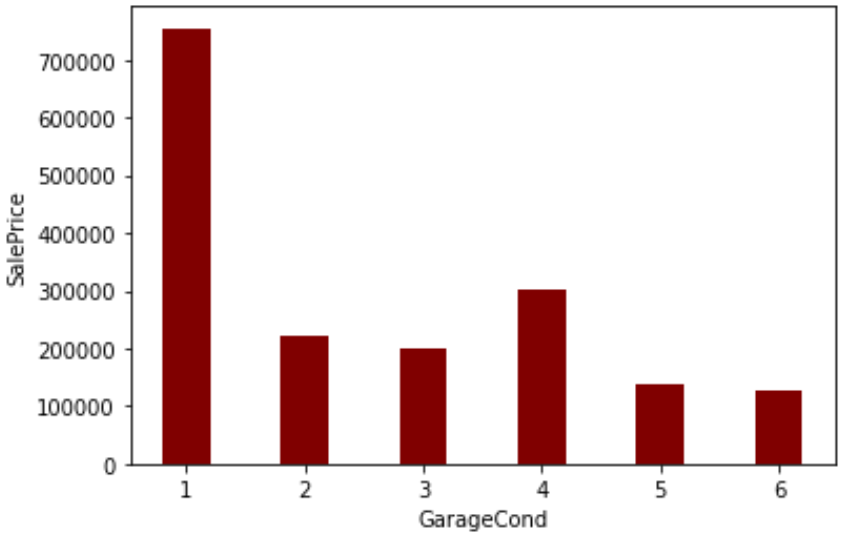
Below is the code snippet used to visualize the data

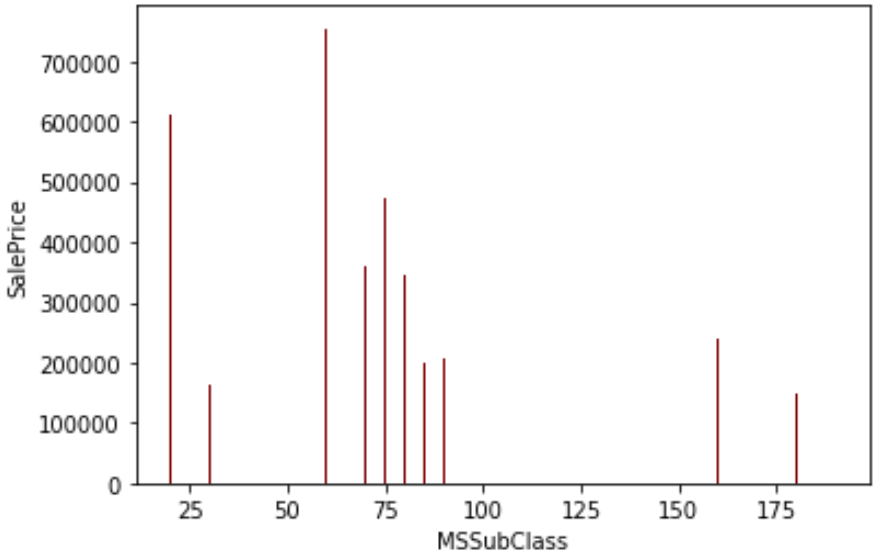
```
column= ['SaleCondition',
'GarageCond','Neighborhood','MSSubClass','OverallCond']
for cols in column:
    plt.bar(clean_column[cols], clean_column['SalePrice'], color ='maroon',
        width = 0.4)
    plt.xlabel(cols)
    plt.ylabel('SalePrice')
    plt.show()
```

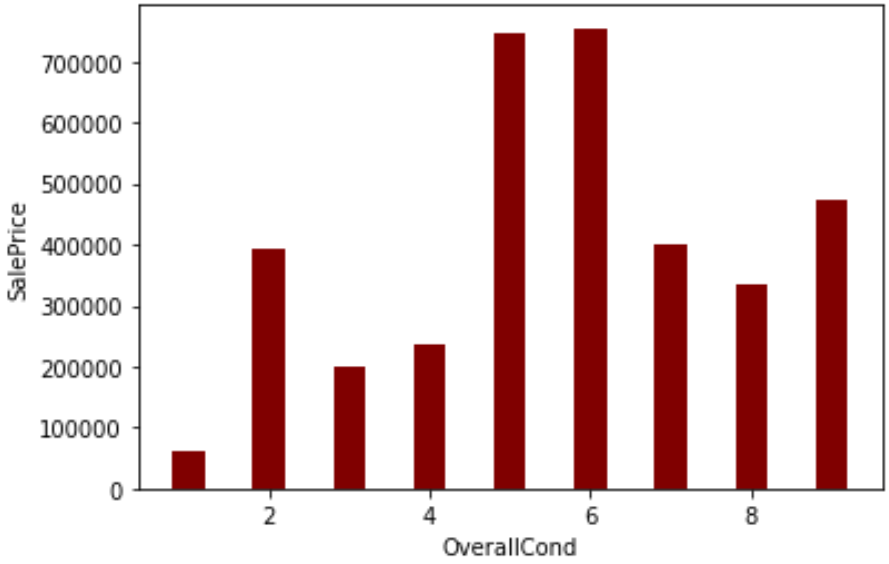


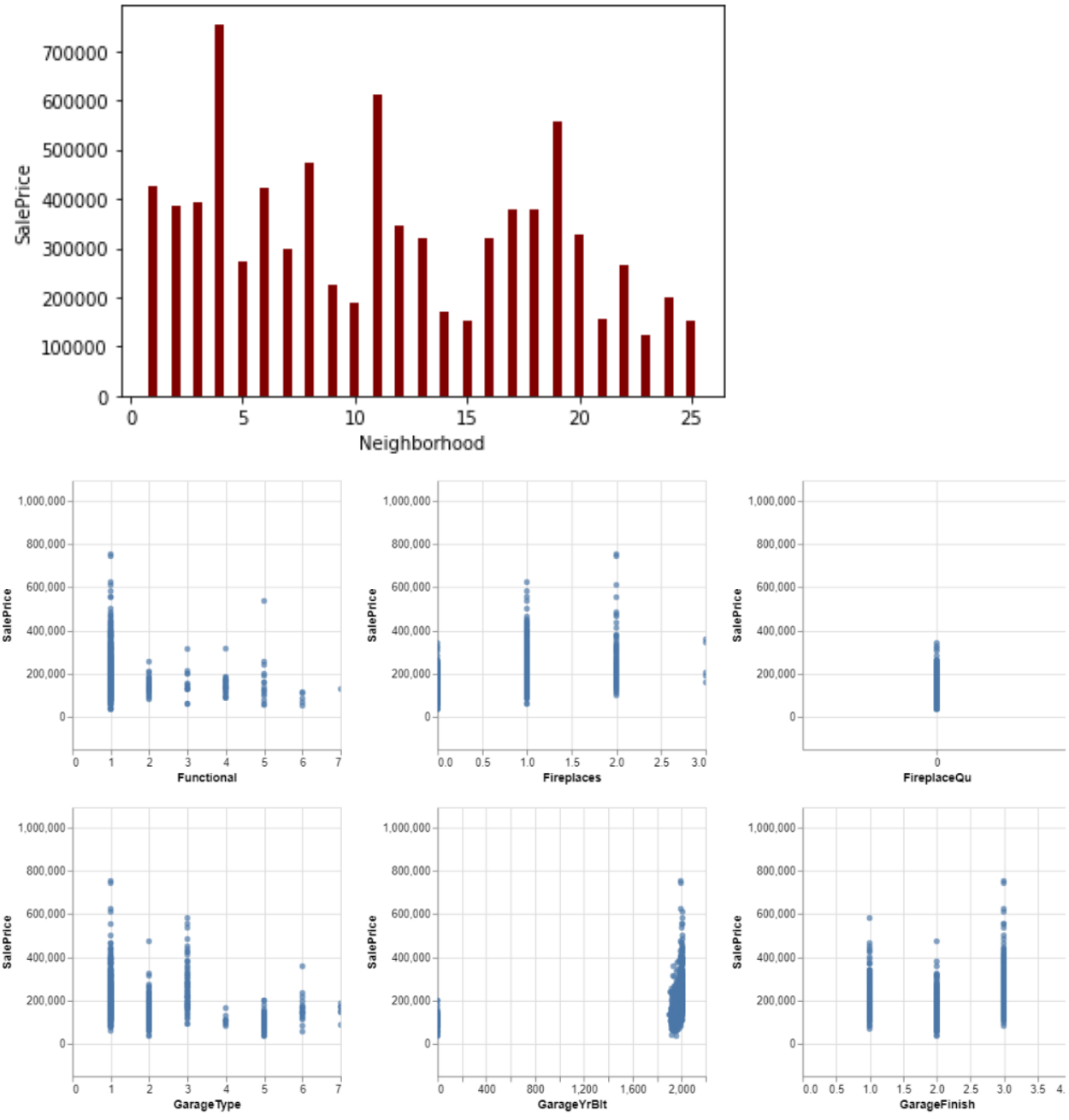
Sale condition:

1- Normal
2- Abnormal
3- Partial
4- Adjoined Land purchase
5- Condo with garage
6- Family


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
GarageCond


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
25
50
75
100
125
150
175
MSSubClass


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
2
4
6
8
OverallCond

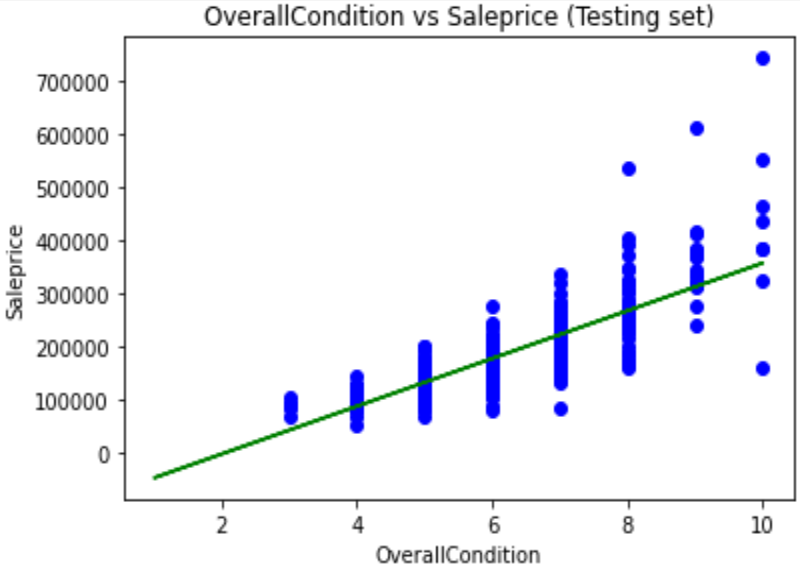


Please refer the ipynb file for more of these graphs.

After the modeling was completed, we plotted a graph, with the higher co-efficient feature i.e, Overall Condition

```
plt.scatter(test_data, test_target, color = "blue")
```

```
plt.plot(train_data.to_numpy(), y, color = "green")
plt.title("OverallCondition vs Saleprice (Testing set)")
plt.xlabel("OverallCondition")
plt.ylabel("Saleprice")
plt.show()
```



ETHICS:

As we are dealing with public data and as it does not include any personal data, any unintended and negative consequences are highly improbable. With our design and structure of data analysis, we emphasize the benevolent nature of the analysis results for both the parties. i.e., House buyers and sellers.

It would be unethical if the house sellers fudge their housing data while selling. Though of the features like NoOfBathrooms are directly related to house price, the stakeholders may try to hide certain subtle faults in the important but non-evident variables (Eg: GarageSpace)

Our design does not disproportionally affect underserved, marginalized, low-resourced, and underrepresented populations.

As our design can help both buyers and sellers to understand the important features, it will promote the positive impact on both the sides.

REFERENCES:

Kaggle dataset https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques