HOUSING PROJECT

Submitted by:

Khanin Deka

ACKNOWLEDGMENT

- https://datascience.stackexchange.com/
- https://stackoverflow.com/
- FlipRobo Technologies
- https://scikit-learn.org/

INTRODUCTION

- ## Business Problem Framing

  Houses are one of the necessary needs 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 this domain. Data Analytics can be used to help the companies to focus on changing trends in house sales and purchases to increase their overall profit and revenue.
  We focus on building a machine learning model to predict the actual value of prospective properties which can be used by a company to decide whether or not to invest in them.

- ## Conceptual Background of the problem

  The price of a property depends on several factors like type of dwelling, zone, total area, utilities, and many more. To be able to predict the price of a property the machine learning model needs to learn the relationship of various variables and factors affecting the price. Price is a continuous numeric variable. Hence various regression algorithms shall be used to predict the price value.
  We have a dataset of 1460 entries each having 81 variables. We train a machine learning model using 1168 entries and analyze the relationship of various variables with the price of the property. Then we test our model using the remaining 292 entries to see how well the model is performing.

Analytical Problem Framing

- About the Dataset
  1. The dataset contains 1460 rows and 81 columns. We will use 80% of the entries to train a machine learning model.
  2. Out of the 81 columns, 43 are of object data-type, 35 are of integer type and 3 of float type.
  3. Here is a snapshot of the dataset:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
3	110	20	RL	105.0	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1163	289	20	RL	NaN	9819	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
1164	554	20	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
1165	196	160	RL	24.0	2280	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1166	31	70	C (all)	50.0	8500	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
1167	617	60	RL	NaN	7861	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0

1168 rows × 81 columns

  4. Most of the columns have data in text format. A machine learning model doesn't understand text data. Hence, we have to encode these text data into numeric format.
  5. Some columns have missing entries which we have to deal with using various imputation techniques.

- Tools and Libraries used
    1. This project is built using Jupyter-Notebook of Anaconda Navigator.
    2. python language version *3.9.12* is used.
    3. List of libraries include pandas, numpy and sklearn.
    4. For data visualization matplotlib and seaborn is used.

- Data Preprocessing
    1. *Id* column is deleted as it is nominal data to identify a housing property.
    2. *Utilities* column is deleted as it contains only one unique value for the entire column.
    3. *Alley* column has more than 1000 missing entries. It is assumed that these missing values signify no access to alley.
    4. *MasVnrType* column which signifies Masonry veneer type, has some missing values. These missing values are assumed to be of type *None* which is one of the unique values in the column.
    5. *BsmtQual* column which evaluates the height of the basement has missing values. These missing values are assumed to signify non-existence of basement. The missing values in *BsmtCond*, *BsmtExposure, BsmtFinType1* and *BsmtFinType2* columns are also interpreted the same way.
    6. *FireplaceQu* column which signifies fireplace quality has missing entries. These missing values are assumed to signify non-existence of fireplace.
    7. *GarageType* column has missing values. These missing values are assumed to signify non-existence of garage. The missing in columns *GarageFinish*, *GarageQual* and *GarageCond* are also handled the same way.

8. The missing values in *PoolQC, Fence* and *MiscFeature* are also handled the same way.
9. **Encoding, Imputation and Transformation**
    a) Firstly the categorical columns of object data-type which give quality ratings of different infrastructure in the property are identified and for them python dictionaries are defined to map from and encode the values. Here is a screenshot:

```
q_rating1= {'Po':0,'Fa':1,'TA':2,'Gd':3,'Ex':4}
q_rating2= {'NA':0,'Po':1,'Fa':2,'TA':3,'Gd':4,'Ex':5}
q_rating3= {'NA':0,'No':1,'Mn':2,'Av':3,'Gd':4}
q_rating4= {'NA':0,'Unf':1,'LwQ':2,'Rec':3,'BLQ':4,'ALQ':5,'GLQ':6}
q_rating5= {'Sal':0,'Sev':1,'Maj2':2,'Maj1':3,'Mod':4,'Min2':5,'Min1':6,'Typ':7}
q_rating6= {'NA':0,'Unf':1,'RFn':2,'Fin':3}
q_rating7= {'N':1,'P':2,'Y':3}
q_rating8= {'NA':0,'MnWw':1,'GdWo':2,'MnPrv':3,'GdPrv':4}
```

    Encoding of categorical columns using map function:

```
# Encoding using map function and the above defined dictionaries.
df['ExterQual'] = df['ExterQual'].map(q_rating1)
df['ExterCond']= df['ExterCond'].map(q_rating1)
df['BsmtQual']= df['BsmtQual'].map(q_rating2)
df['BsmtCond']= df['BsmtCond'].map(q_rating2)
df['BsmtExposure']= df['BsmtExposure'].map(q_rating3)
df['BsmtFinType1']= df['BsmtFinType1'].map(q_rating4)
df['BsmtFinType2']= df['BsmtFinType2'].map(q_rating4)
df['HeatingQC']= df['HeatingQC'].map(q_rating1)
df['KitchenQual']= df['KitchenQual'].map(q_rating1)
df['Functional']= df['Functional'].map(q_rating5)
df['FireplaceQu']= df['FireplaceQu'].map(q_rating2)
df['GarageFinish']= df['GarageFinish'].map(q_rating6)
df['GarageQual']= df['GarageQual'].map(q_rating2)
df['GarageCond']= df['GarageCond'].map(q_rating2)
df['PavedDrive']= df['PavedDrive'].map(q_rating7)
df['PoolQC']= df['PoolQC'].map(q_rating2)
df['Fence']= df['Fence'].map(q_rating8)
```

    b) For the rest of categorical columns with object data-type Label Encoder is used to encode the values.
    c) KNN Imputer is then used to impute all the missing values in the dataset.
    d) Power Transformer is used on continuous numeric columns with skewed data.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches
    1) Our task is to predict the price of a property which is a regression task. For this multiple algorithmic approaches are tried and their performances are evaluated using relevant metrics.
    2) **Multiple Linear Regression**: In this approach relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data**.**
    3) **Random Forests Regression**: It is an ensemble learning method for regression that operates by constructing a multitude of decision trees at training time. The mean or average prediction of the individual trees is then returned.
    4) **XGBoost Regression**: Extreme Gradient Boosting or XGBoost is an ensemble machine learning algorithm which uses an objective function and base learners. The objective function contains loss function and a regularization term. It tells about the difference between actual values and predicted values, i.e. how far the model results are from the real values.
    5) **KNN Regression**: It approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood.

- Performance Evaluation

  The primary metric used for evaluation of a model is R^2 (coefficient of determination) regression score function of scikit-learn. R^2 score can be interpreted as the proportion of variation in the dependent variable that is predicted by the statistical model. MAE(Mean Absolute Error) and RMSE(Root Mean Squared Error) are also observed during evaluation.

  1) **Multiple Linear Regression**:

     Code:

```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()

for i in range(0,5):
    x_train,x_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=i)

    lr.fit(x_train,y_train)

    y_pred_train = lr.predict(x_train)
    accuracy_train = r2_score(y_train,y_pred_train)
    print("Training Score for sample",i,":",round(accuracy_train*100,2))

    y_pred= lr.predict(x_test)
    accuracy_test = r2_score(y_test,y_pred)
    print("Testing Score for sample",i,":",round(accuracy_test*100,2))

    print("Mean Absolute Error",round(mean_absolute_error(y_test,y_pred),2))
    print("Root Mean Squared Error",round(np.sqrt(mean_squared_error(y_test,y_pred)),2))
    print("\n")
```

     The performance is tested for 5 different train-test splits.

     Results:

```
Training Score for sample 0 : 82.39
Testing Score for sample 0 : 75.89
Mean Absolute Error 22394.33
Root Mean Squared Error 40921.37


Training Score for sample 1 : 82.21
Testing Score for sample 1 : 80.83
Mean Absolute Error 22611.93
Root Mean Squared Error 33262.7


Training Score for sample 2 : 83.67
Testing Score for sample 2 : 69.67
Mean Absolute Error 23557.18
Root Mean Squared Error 44329.17


Training Score for sample 3 : 81.27
Testing Score for sample 3 : 85.64
Mean Absolute Error 21377.88
Root Mean Squared Error 28426.9


Training Score for sample 4 : 84.96
Testing Score for sample 4 : 63.31
Mean Absolute Error 24567.57
Root Mean Squared Error 48875.7
```

Cross-Validation results:

```
# cross-validating the linear regression model.
for i in range(5,11):
    cv_score= cross_val_score(lr,X_scaled,y,cv=i).mean()
    print("the cv score for",i,"fold:", round(cv_score*100,2))
```

```
the cv score for 5 fold: 74.4
the cv score for 6 fold: 75.86
the cv score for 7 fold: 74.36
the cv score for 8 fold: 76.22
the cv score for 9 fold: 73.89
the cv score for 10 fold: 75.82
```

2) **Random Forests Regression**:

Code:

```
from sklearn.ensemble import RandomForestRegressor
rf= RandomForestRegressor(n_jobs=-1)

for i in range(0,5):
    x_train,x_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=i)

    rf.fit(x_train,y_train)

    y_pred_train = rf.predict(x_train)
    accuracy_train = r2_score(y_train,y_pred_train)
    print("Training Score for sample",i,":",round(accuracy_train*100,2))

    y_pred= rf.predict(x_test)
    accuracy_test = r2_score(y_test,y_pred)
    print("Testing Score for sample",i,":",round(accuracy_test*100,2))

    print("Mean Absolute Error",round(mean_absolute_error(y_test,y_pred),2))
    print("Root Mean Squared Error",round(np.sqrt(mean_squared_error(y_test,y_pred)),2))
    print("\n")
```

Train-Test split Results:

```
Training Score for sample 0 : 97.91
Testing Score for sample 0 : 88.51
Mean Absolute Error 17229.14
Root Mean Squared Error 28251.12


Training Score for sample 1 : 97.84
Testing Score for sample 1 : 88.95
Mean Absolute Error 17021.26
Root Mean Squared Error 25255.56


Training Score for sample 2 : 97.53
Testing Score for sample 2 : 88.3
Mean Absolute Error 15854.19
Root Mean Squared Error 27539.77


Training Score for sample 3 : 97.75
Testing Score for sample 3 : 88.4
Mean Absolute Error 17251.1
Root Mean Squared Error 25550.25


Training Score for sample 4 : 97.66
Testing Score for sample 4 : 80.06
Mean Absolute Error 19133.45
Root Mean Squared Error 36030.91
```

Cross-Validation Results:

```
# cross-validating the random-forests regression model.
for i in range(5,11):
    cv_score= cross_val_score(rf,X_scaled,y,cv=i).mean()
    print("the cv score for",i,"fold:", round(cv_score*100,2))
```

```
the cv score for 5 fold: 84.6
the cv score for 6 fold: 85.8
the cv score for 7 fold: 83.5
the cv score for 8 fold: 84.84
the cv score for 9 fold: 84.51
the cv score for 10 fold: 84.74
```

3) **XGBoost Regression**:

Code:

```
import xgboost as xgb
xgb_reg= xgb.XGBRegressor()

for i in range(0,5):
    x_train,x_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=i)

    xgb_reg.fit(x_train,y_train)

    y_pred_train = xgb_reg.predict(x_train)
    accuracy_train = r2_score(y_train,y_pred_train)
    print("Training Score for sample",i,":",round(accuracy_train*100,2))

    y_pred= xgb_reg.predict(x_test)
    accuracy_test = r2_score(y_test,y_pred)
    print("Testing Score for sample",i,":",round(accuracy_test*100,2))

    print("Mean Absolute Error",round(mean_absolute_error(y_test,y_pred),2))
    print("Root Mean Squared Error",round(np.sqrt(mean_squared_error(y_test,y_pred)),2))
    print("\n")
```

Train-Test split Results:

```
Training Score for sample 0 : 100.0
Testing Score for sample 0 : 78.53
Mean Absolute Error 19001.51
Root Mean Squared Error 38614.57


Training Score for sample 1 : 100.0
Testing Score for sample 1 : 87.65
Mean Absolute Error 18024.83
Root Mean Squared Error 26694.85


Training Score for sample 2 : 99.99
Testing Score for sample 2 : 85.39
Mean Absolute Error 17099.34
Root Mean Squared Error 30768.52


Training Score for sample 3 : 100.0
Testing Score for sample 3 : 83.61
Mean Absolute Error 18905.24
Root Mean Squared Error 30368.02


Training Score for sample 4 : 99.99
Testing Score for sample 4 : 87.37
Mean Absolute Error 18389.54
Root Mean Squared Error 28674.09
```

Cross-Validation Results:

```
# cross-validating the xgboost regression model.
for i in range(5,11):
    cv_score= cross_val_score(xgb_reg,X_scaled,y,cv=i).mean()
    print("the cv score for",i,"fold:", round(cv_score*100,2))
```

```
the cv score for 5 fold: 83.36
the cv score for 6 fold: 83.81
the cv score for 7 fold: 82.14
the cv score for 8 fold: 82.21
the cv score for 9 fold: 83.79
the cv score for 10 fold: 84.21
```

4) **KNN Regression**:

Code:

```
from sklearn.neighbors import KNeighborsRegressor
knr= KNeighborsRegressor()

for i in range(0,5):
    x_train,x_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=i)

    knr.fit(x_train,y_train)

    y_pred_train = knr.predict(x_train)
    accuracy_train = r2_score(y_train,y_pred_train)
    print("Training Score for sample",i,":",round(accuracy_train*100,2))

    y_pred= knr.predict(x_test)
    accuracy_test = r2_score(y_test,y_pred)
    print("Testing Score for sample",i,":",round(accuracy_test*100,2))

    print("Mean Absolute Error",round(mean_absolute_error(y_test,y_pred),2))
    print("Root Mean Squared Error",round(np.sqrt(mean_squared_error(y_test,y_pred)),2))
    print("\n")
```

Train-Test split Results:

```
Training Score for sample 0 : 86.11
Testing Score for sample 0 : 67.17
Mean Absolute Error 23400.72
Root Mean Squared Error 47745.1


Training Score for sample 1 : 82.66
Testing Score for sample 1 : 78.81
Mean Absolute Error 21788.9
Root Mean Squared Error 34963.1


Training Score for sample 2 : 85.03
Testing Score for sample 2 : 64.5
Mean Absolute Error 22734.58
Root Mean Squared Error 47962.65


Training Score for sample 3 : 82.4
Testing Score for sample 3 : 78.09
Mean Absolute Error 23068.06
Root Mean Squared Error 35109.72


Training Score for sample 4 : 83.25
Testing Score for sample 4 : 76.06
Mean Absolute Error 24038.25
Root Mean Squared Error 39476.01
```

Cross-Validation Results:

```
# cross-validating the knn regression model.
for i in range(5,11):
    cv_score= cross_val_score(knr,X_scaled,y,cv=i).mean()
    print("the cv score for",i,"fold:", round(cv_score*100,2))
```

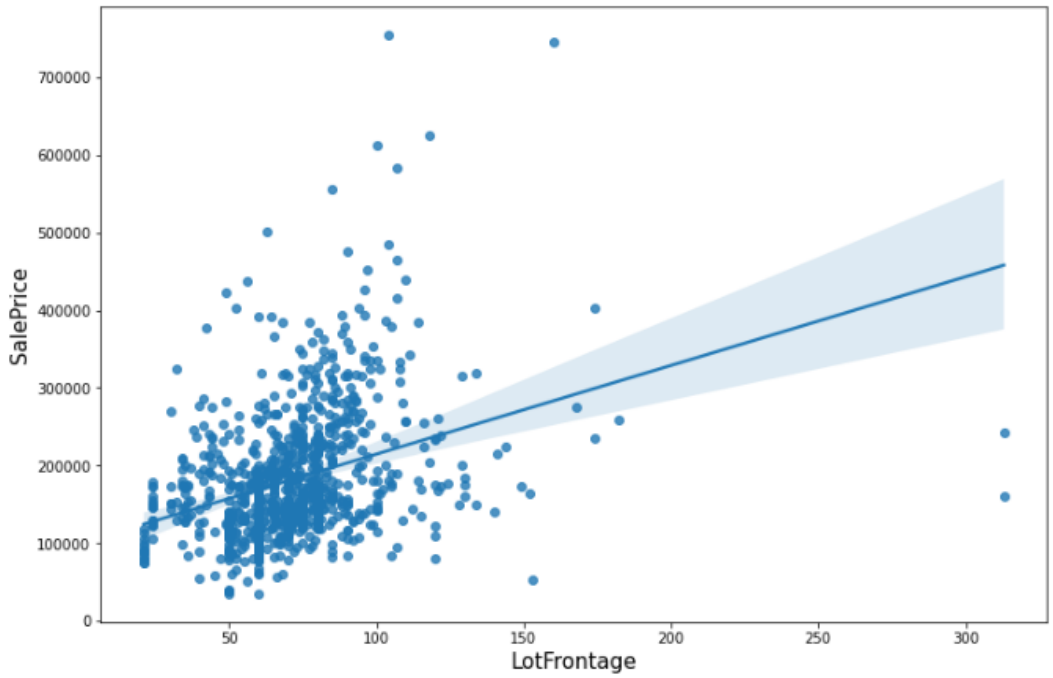
```
the cv score for 5 fold: 75.3
the cv score for 6 fold: 75.11
the cv score for 7 fold: 75.21
the cv score for 8 fold: 75.46
the cv score for 9 fold: 76.24
the cv score for 10 fold: 75.73
```

Based on the R^2 score in train-test split and cross-validation, Random-Forests is performing best for our dataset.

- ## Visualizations

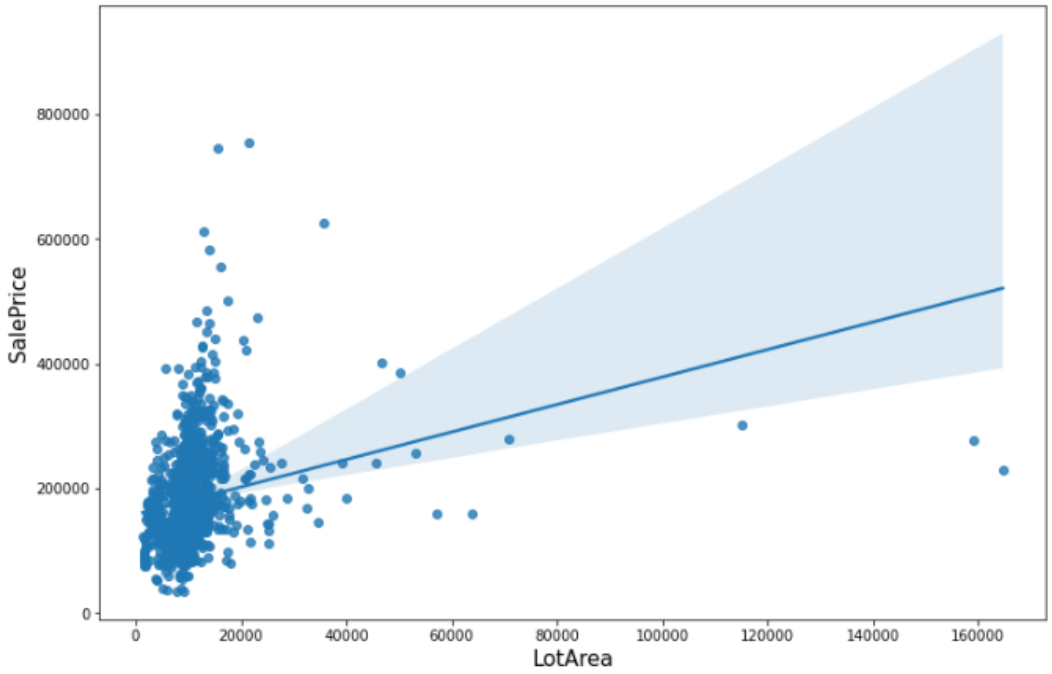
  To analyze the relation between sale price and the independent variables, various plots are made. Let's see them.

  Regression plot of *LotFrontage* vs *SalePrice*:



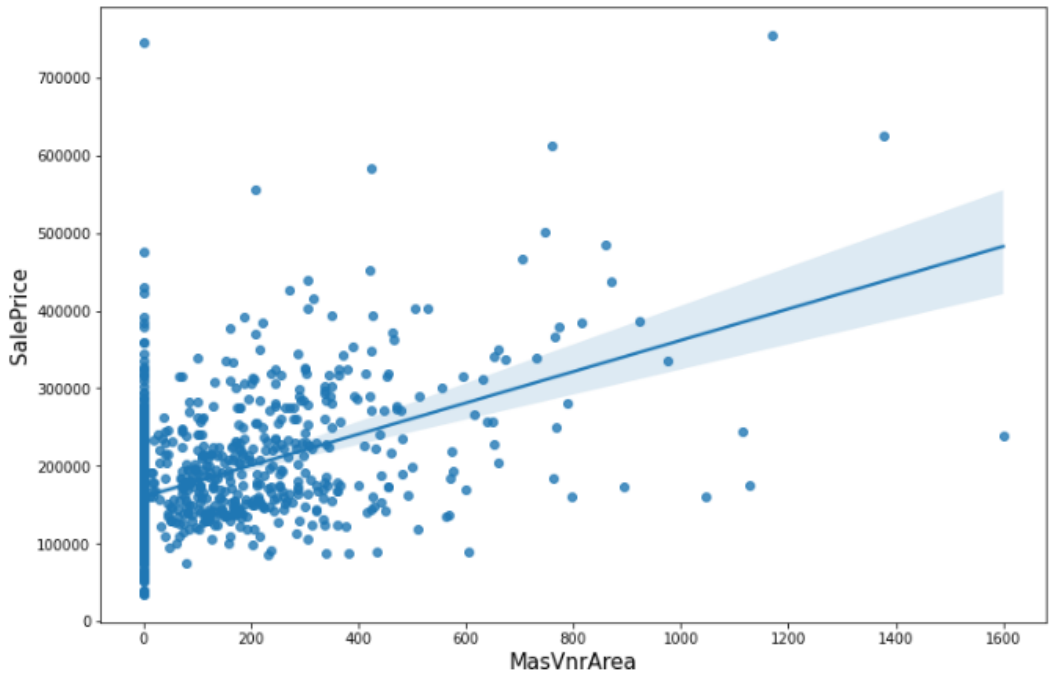
  The slope of the regression-fit line shows positive linear relationship.

Regression plot of *LotArea* vs *SalePrice*:



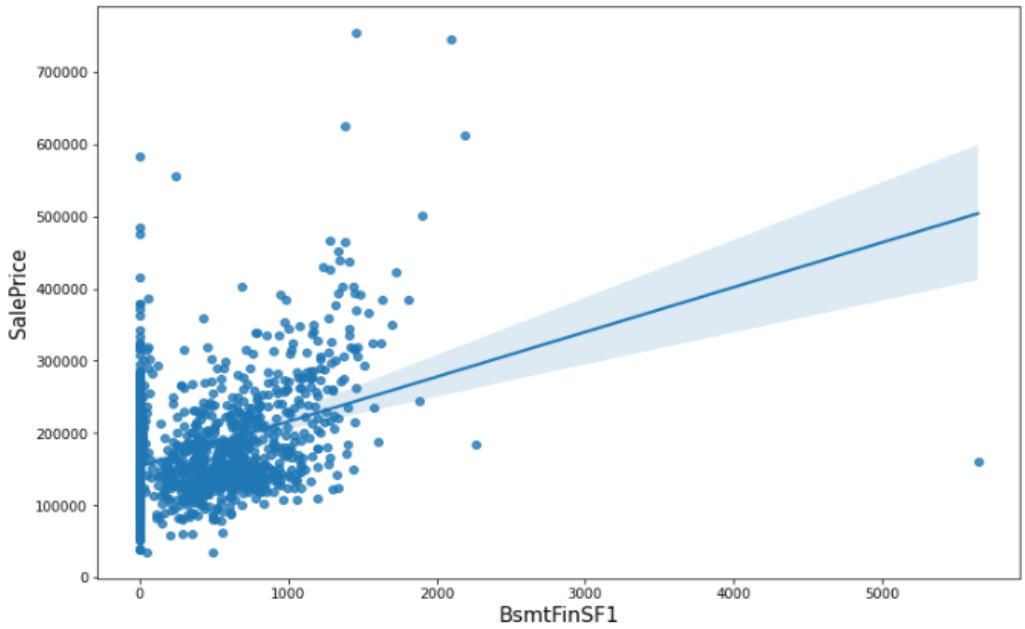
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *MasVnrArea* vs *SalePrice*:



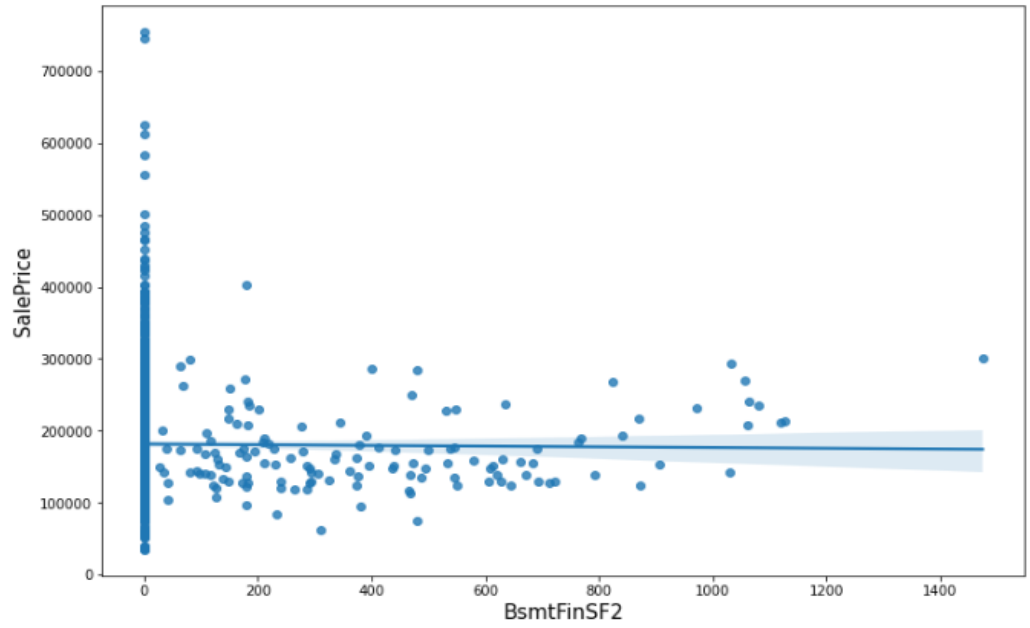
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *BsmtFinSF1* vs *SalePrice*:



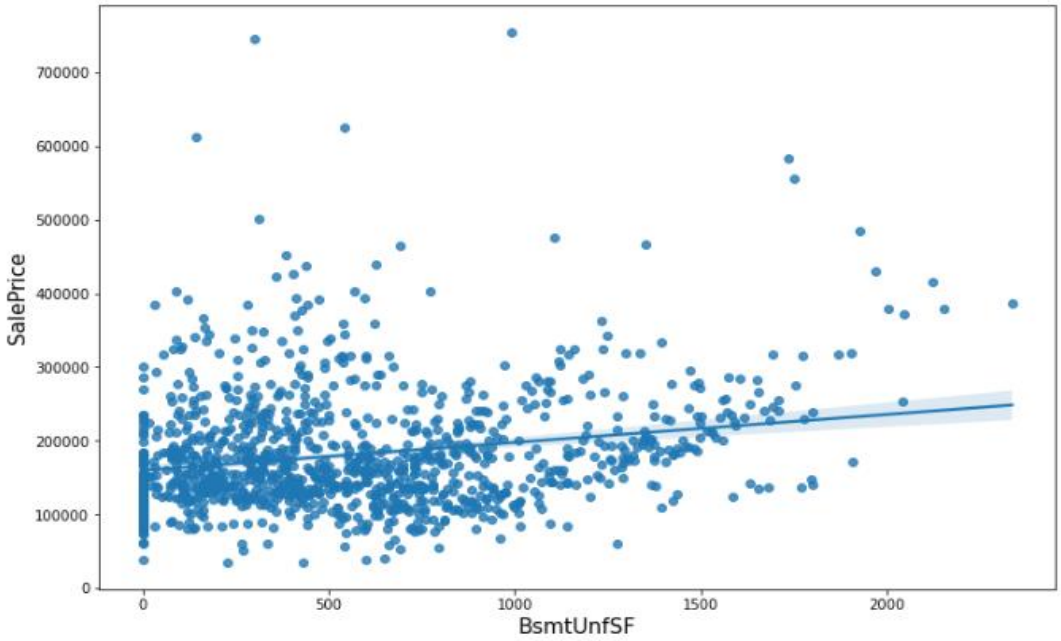
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *BsmtFinSF2* vs *SalePrice*:



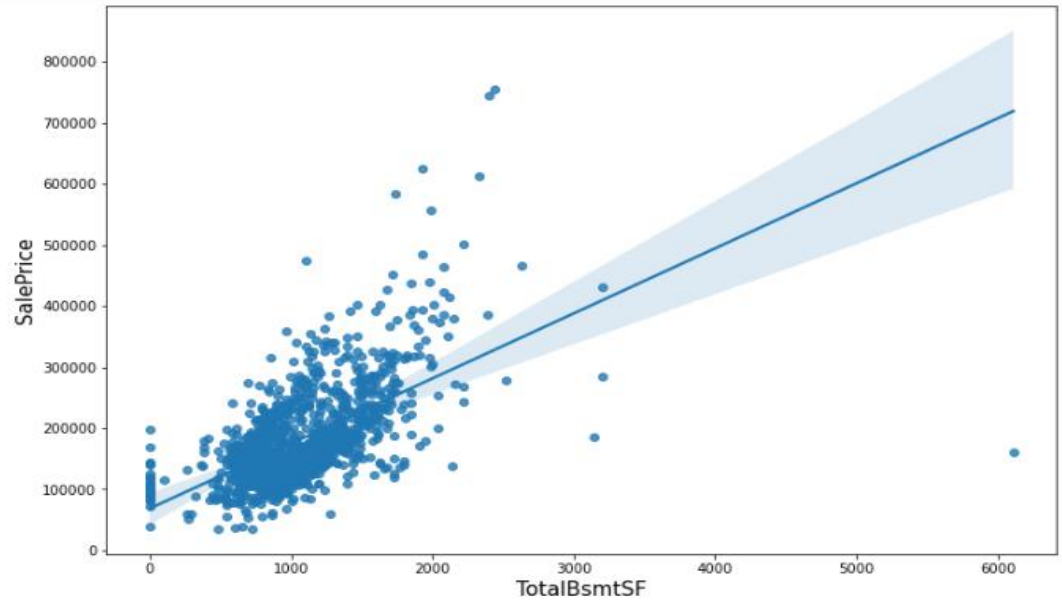
Since the regression-fit line is almost horizontal, it can be said that the relation is very weak.

Regression plot of *BsmtUnfSF* vs *SalePrice*:



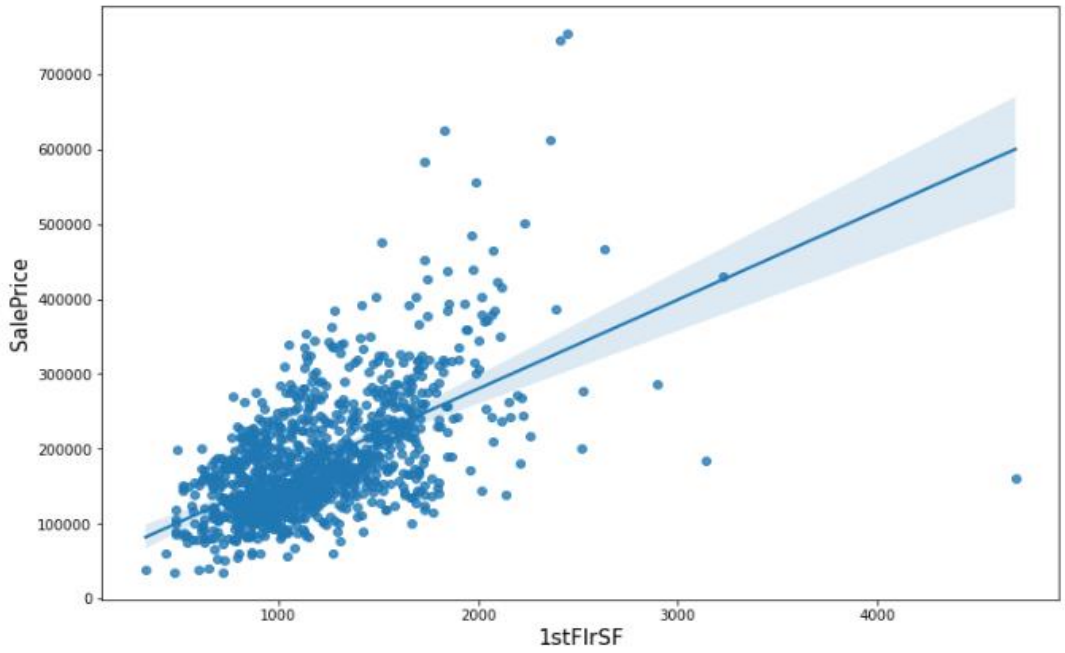
The slope of the regression-fit line shows positive linear relationship. But since the slope is not very significant hence the relation is weak.

Regression plot of *TotalBsmtfSF* vs *SalePrice*:



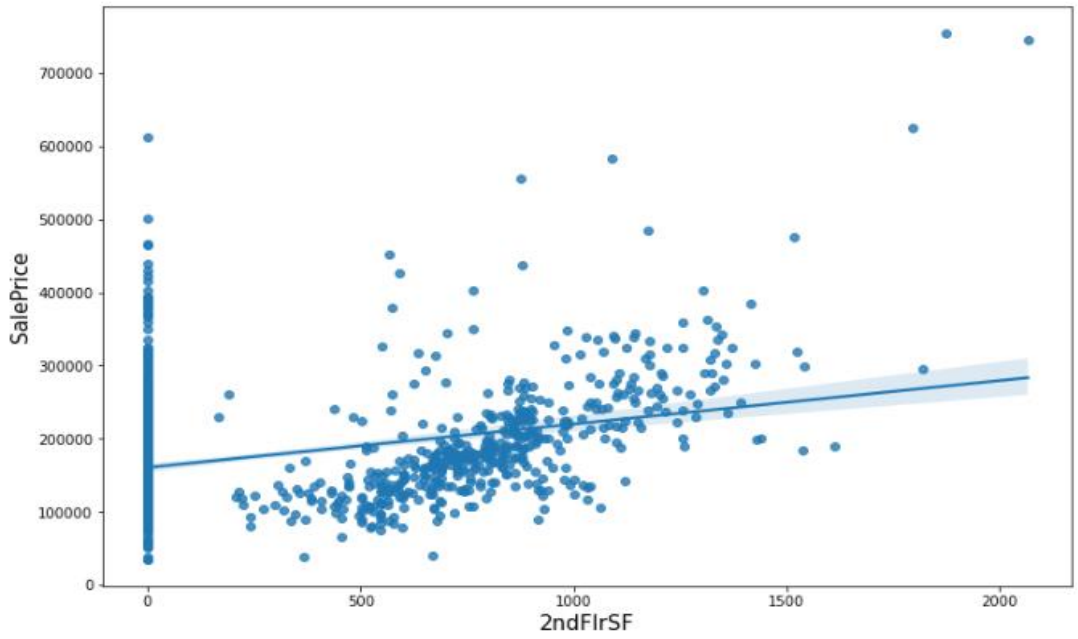
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *1stFlrSF* vs *SalePrice*:



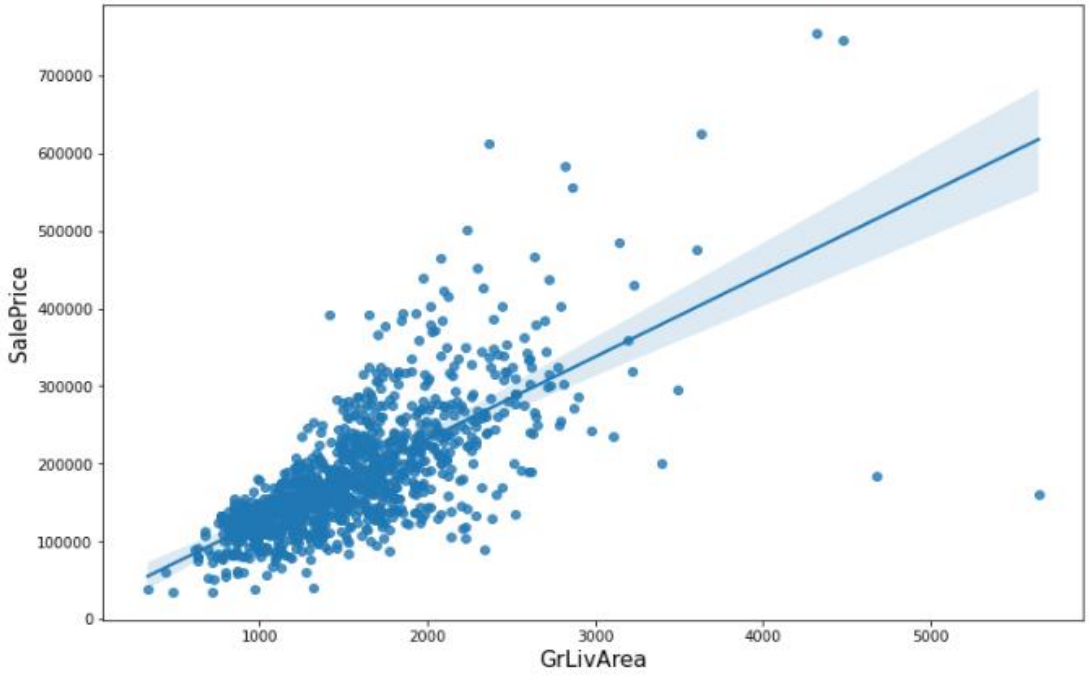
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *2ndFlrSF* vs *SalePrice*:



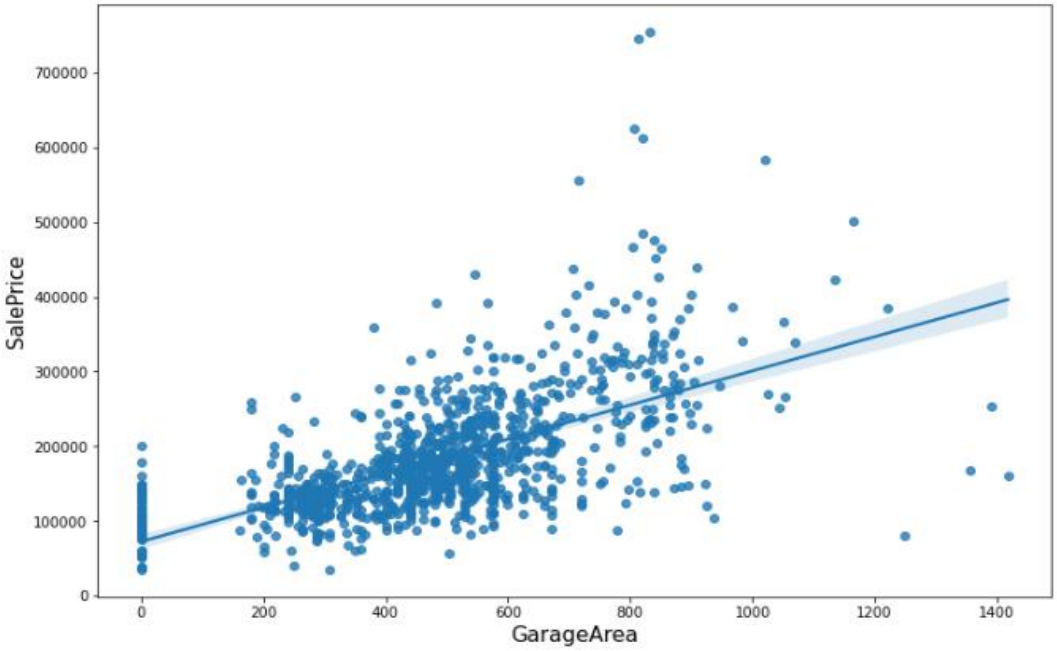
The slope of the regression-fit line shows positive linear relationship but weaker compared to 1st floor surface area.

Regression plot of *GrLivArea* vs *SalePrice*:



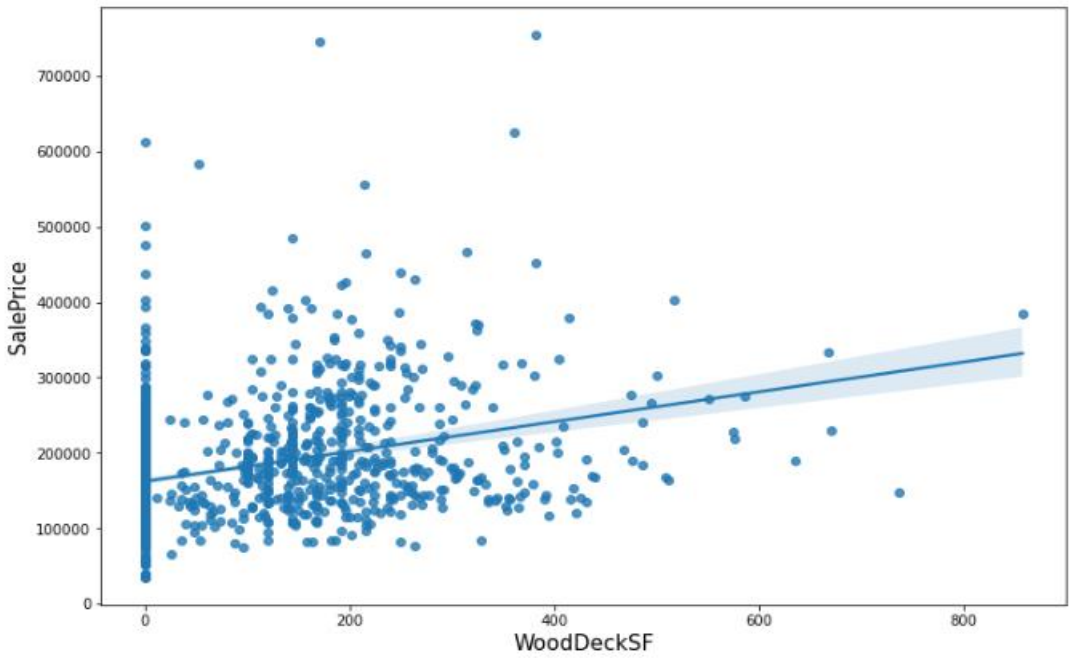
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *GarageArea* vs *SalePrice*:



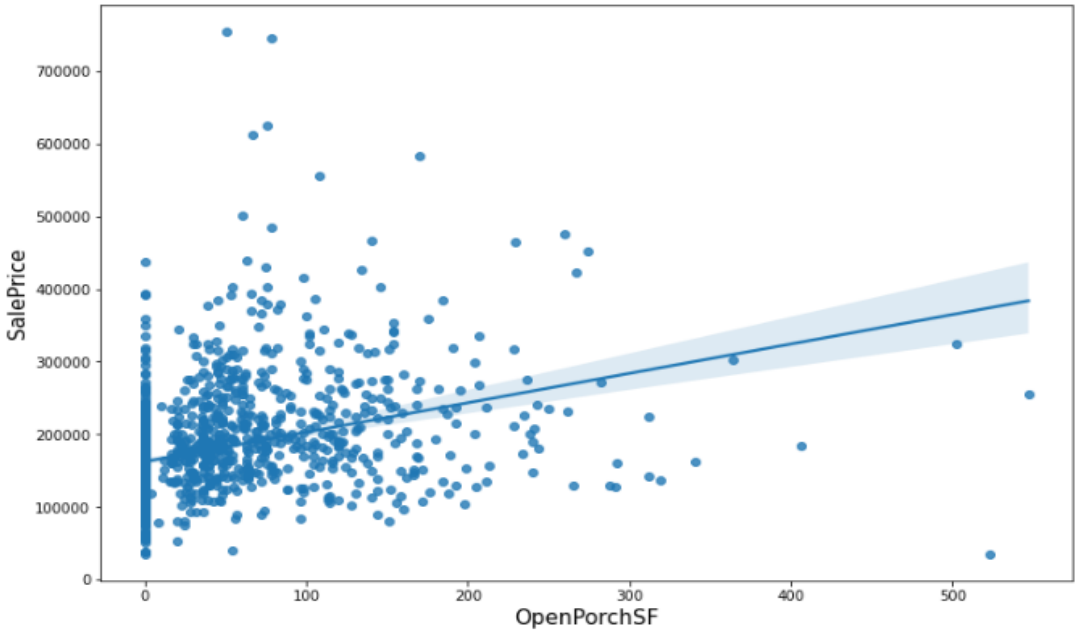
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *WoodDeckSF* vs *SalePrice*:



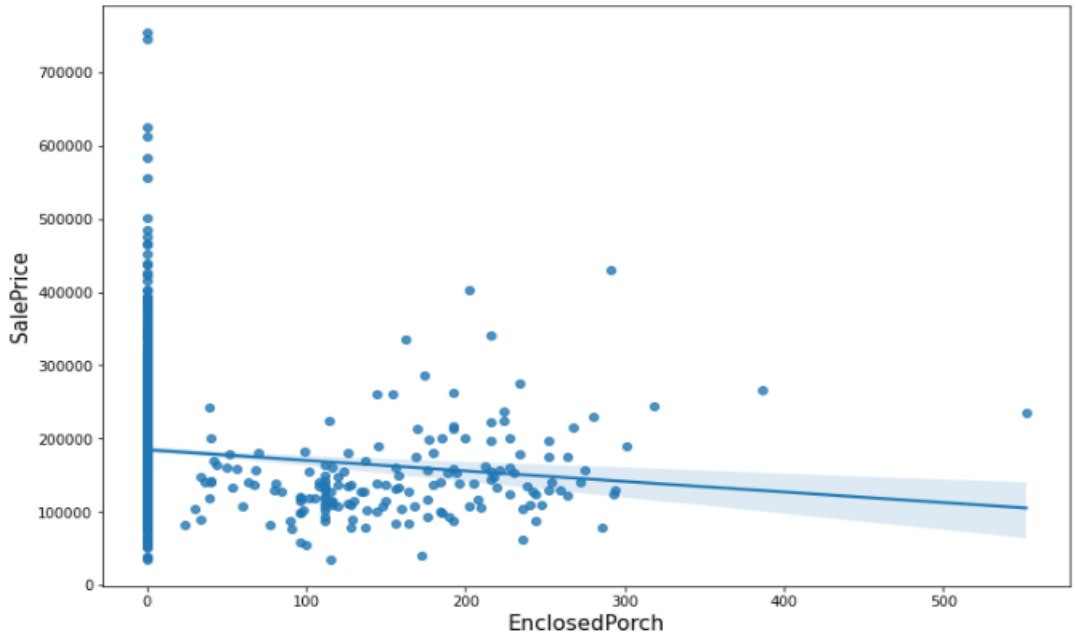
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *OpenPorchSF* vs *SalePrice*:



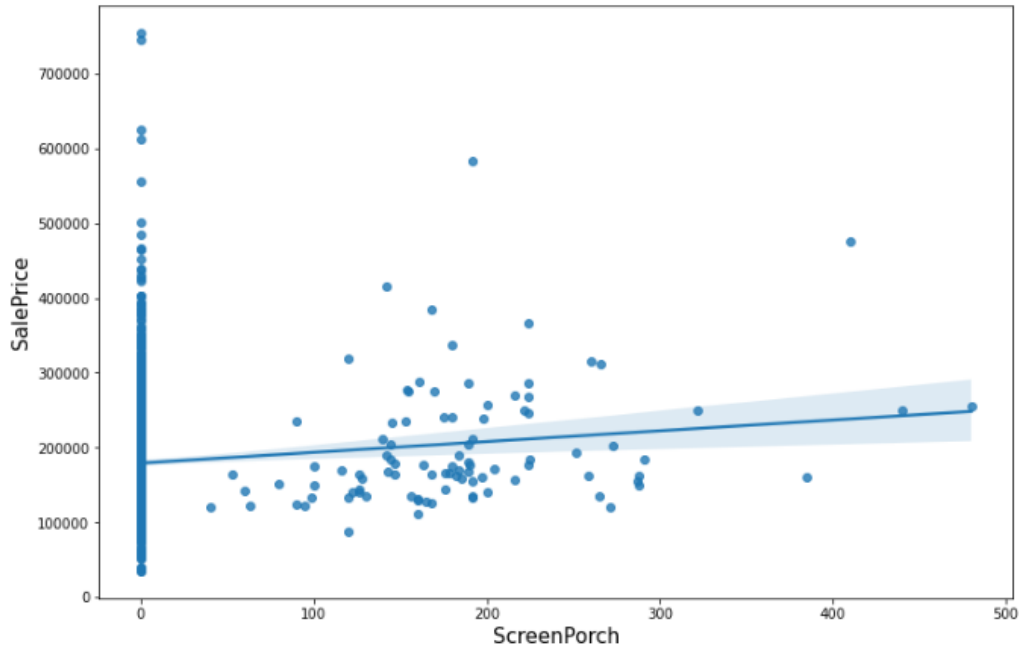
The slope of the regression-fit line shows positive linear relationship.

Regression plot of *EnclosedPorch* vs *SalePrice*:



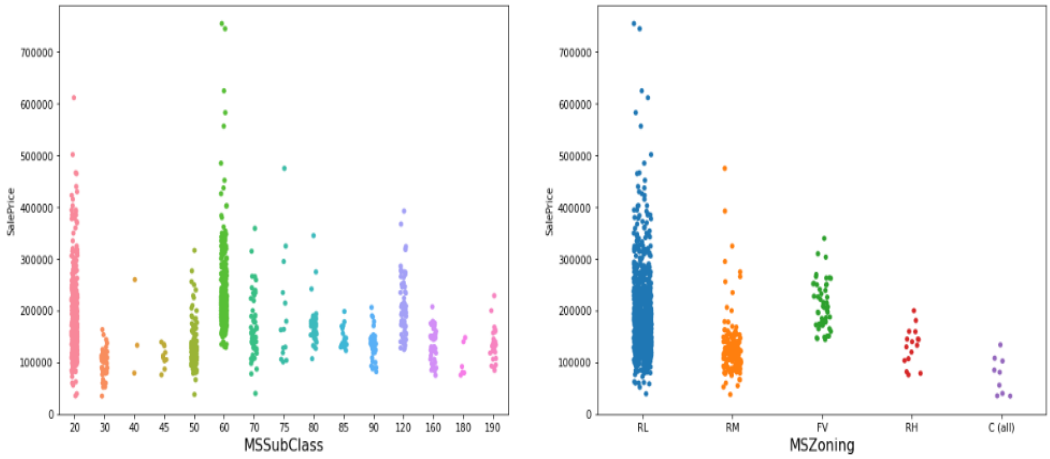
The slope of the regression-fit line shows a weak negative linear relationship.

Regression plot of *ScreenPorch* vs *SalePrice*:



Weak positive linear relation can be seen.

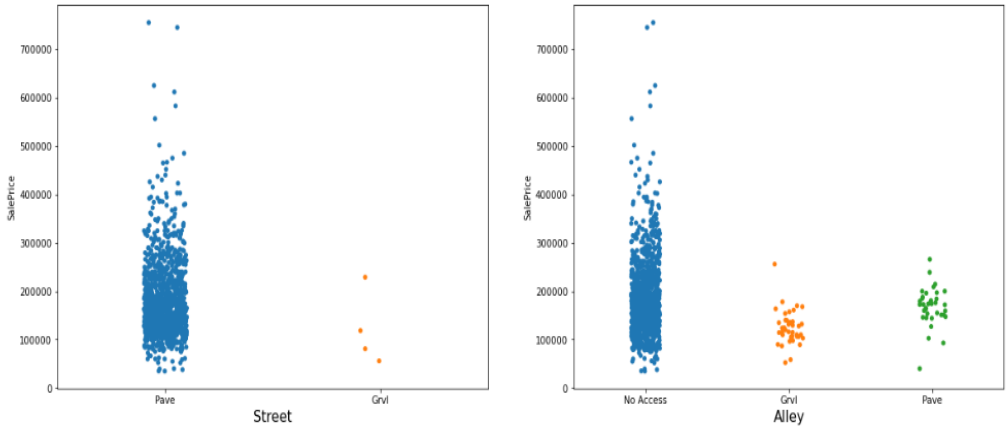
Strip plot of *MSSubClass* and *MSZoning* with respect to *SalePrice*:



MSSubClass identifies the type of dwelling. Type 60(2-STORY 1946 & NEWER), 75(2-1/2 STORY ALL AGES), 80(SPLIT OR MULTI-LEVEL), 85(SPLIT FOYER) and 120(1-STORY PUD - 1946 & NEWER) have a higher starting price then other types. Type 60 have the costliest properties.

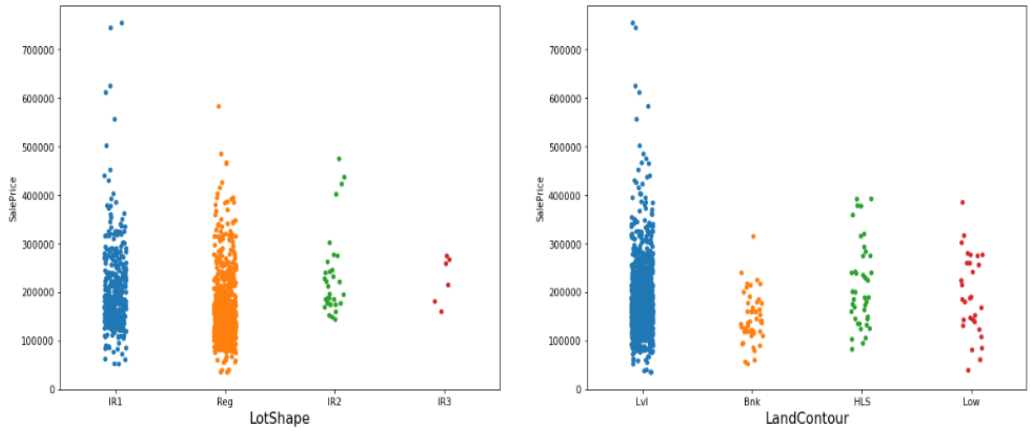
MSZoning identifies the general zoning classification of the sale. Majority of the properties belong to RL (Residential Low Density) type.

Strip plot of *Street* and *Alley* with respect to *SalePrice*:



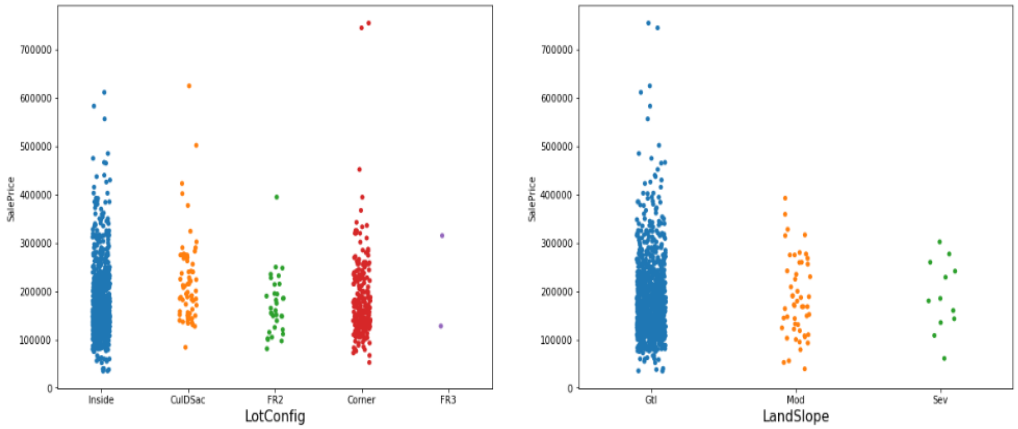
A high majority of properties have paved street and no access to alley.

Strip plot of *LotShape* and *LandContour* with respect to *SalePrice*:



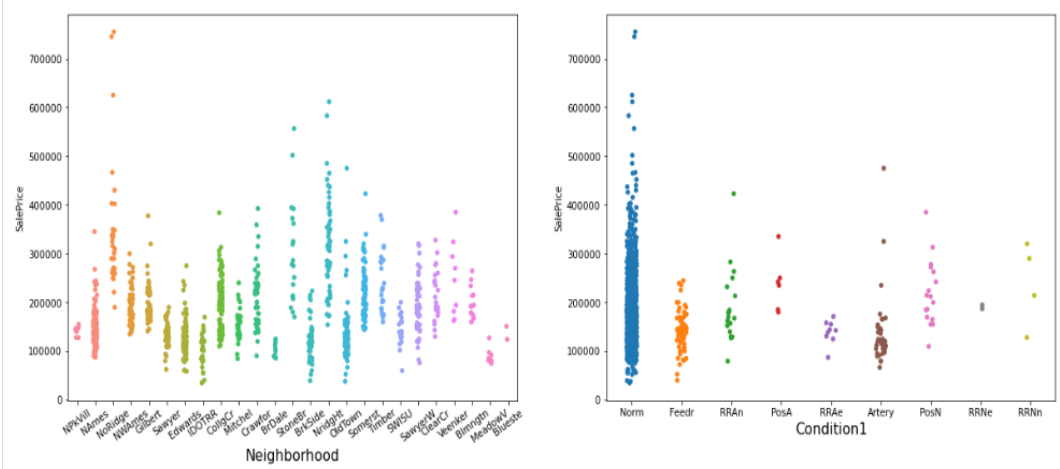
A high majority of properties have regular or slightly irregular shape and leveled land contour. The price of properties on unleveled land contour doesn't go beyond 400K. Irregular-shaped (IR3) properties doesn't go beyond 300K.

Strip plot of *LotConfig* and *LandSlope* with respect to *SalePrice*:



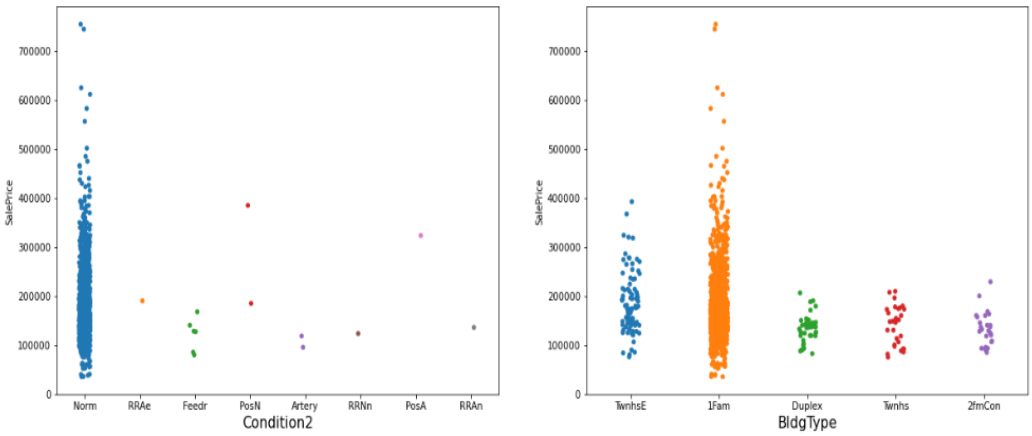
A high majority of properties have gentle land slope. Price of properties with moderate or severe land slope doesn't go beyond 400K

Strip plot of *Neighborhood* and *Condition1* with respect to *SalePrice*:



The price range of property varies with respect to neighborhood. In neighborhoods like *NorthRidge* the starting price itself is higher than the maximum price in *Briardale* or *Meadow Village.*

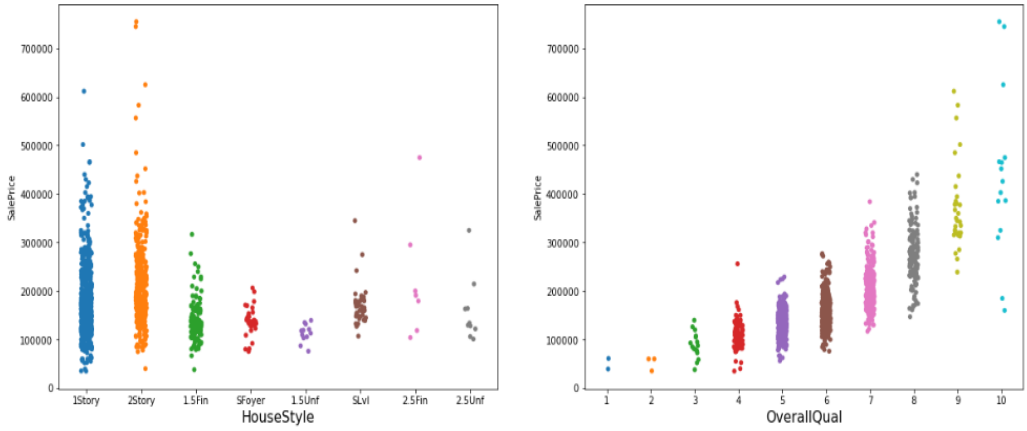
Strip plot of *Condition2* and *BldgType* with respect to *SalePrice*:



A high majority of properties are not adjacent or in very close proximity to railroads or arterial streets.

Majority of properties are of 1Fam(single-family-detached) type.

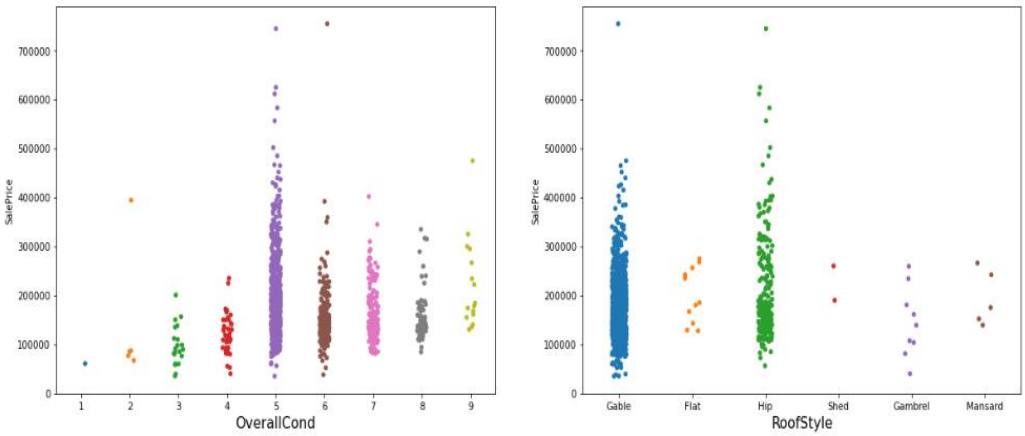
Strip plot of *HouseStyle* and *OverallQual* with respect to *SalePrice*:



Majority of properties are of either 1Story or 2Story type and they have a higher price range compared to others.

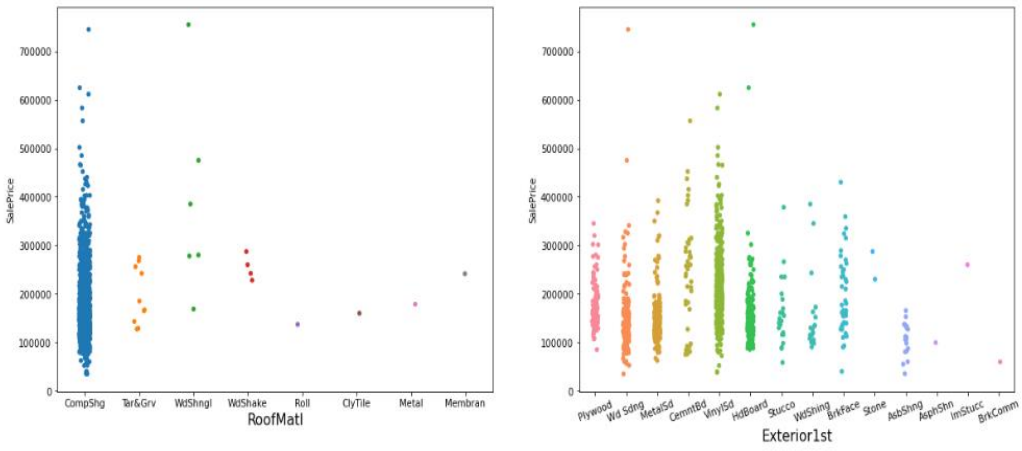
Better the overall quality higher is the starting price as well as maximum price.

Strip plot of *OverallCond* and *RoofStyle* with respect to *SalePrice*:



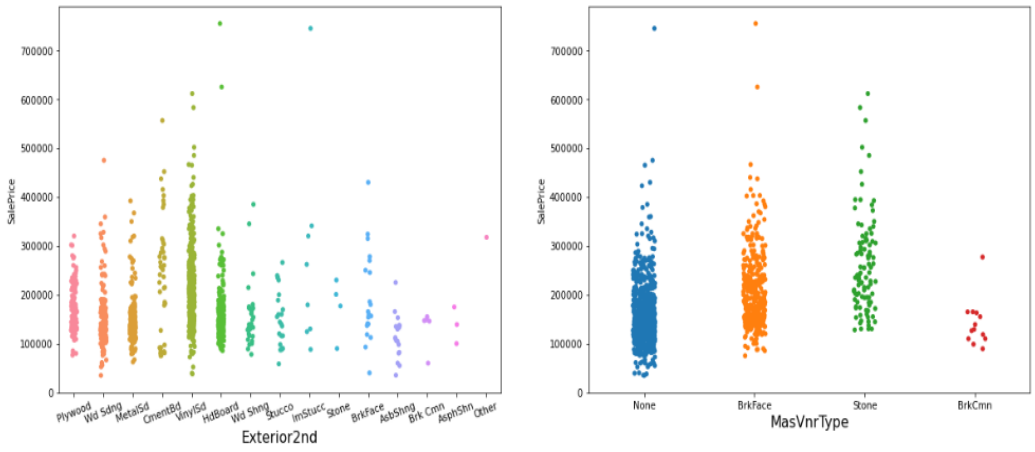
A high majority of properties have roof style either gable or hip and they also have a much higher price range.

Strip plot of *RoofMatl* and *Exterior1st* with respect to *SalePrice*:

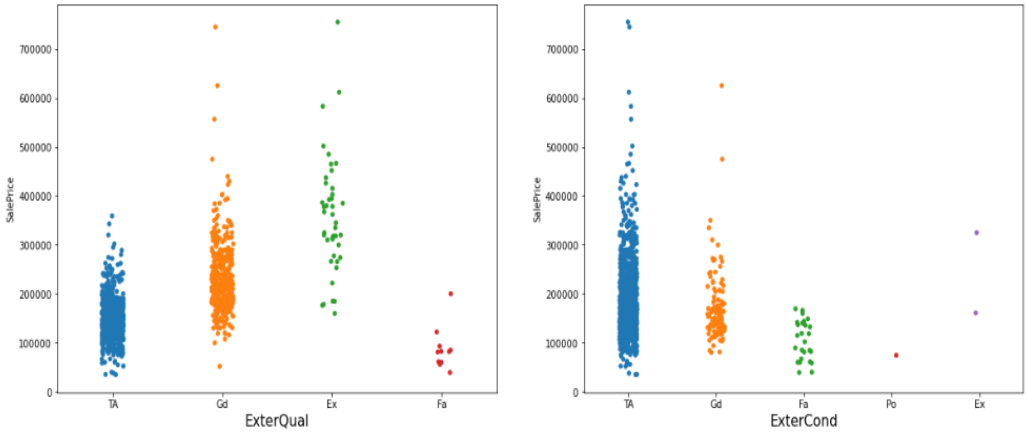


A high majority of properties uses CompShg (Standard Composite Shingle) type of roof material.

Strip plot of *Exterior2nd* and *MasVnrType* with respect to *SalePrice*:



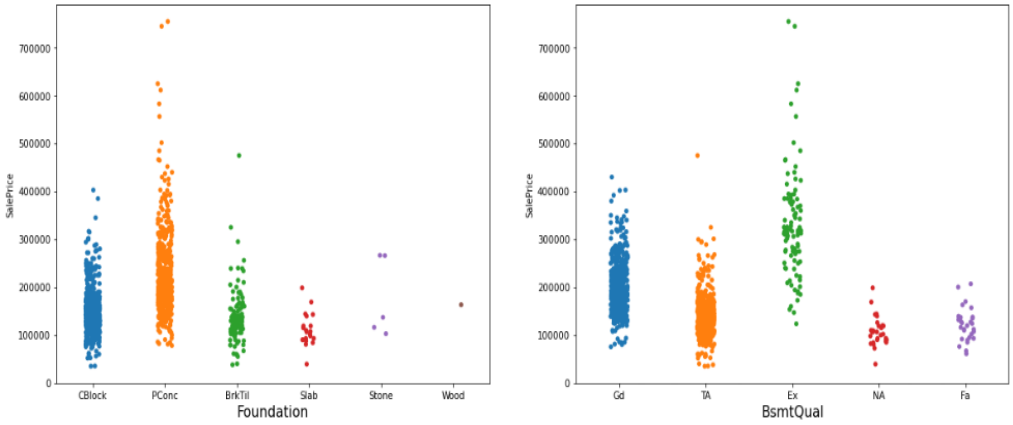
Strip plot of *ExterQual* and *ExterCond* with respect to *SalePrice*:



Properties with better quality material in the exterior have higher starting price and maximum price. Those with average quality doesn't go beyond 400K.

Out of all the properties in the dataset very few have currently excellent or poor condition of exterior materials. A high majority are in average or good condition.

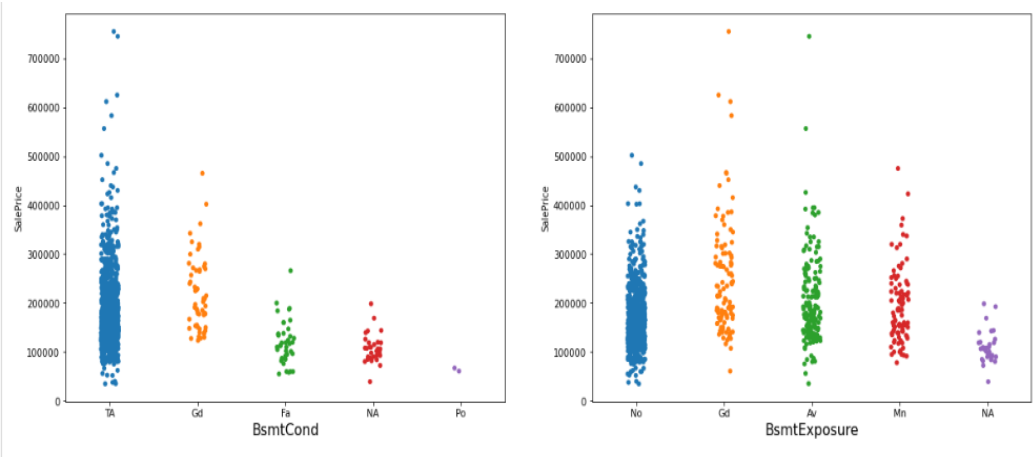
Strip plot of *Foundation* and *BsmtQual* with respect to *SalePrice*:



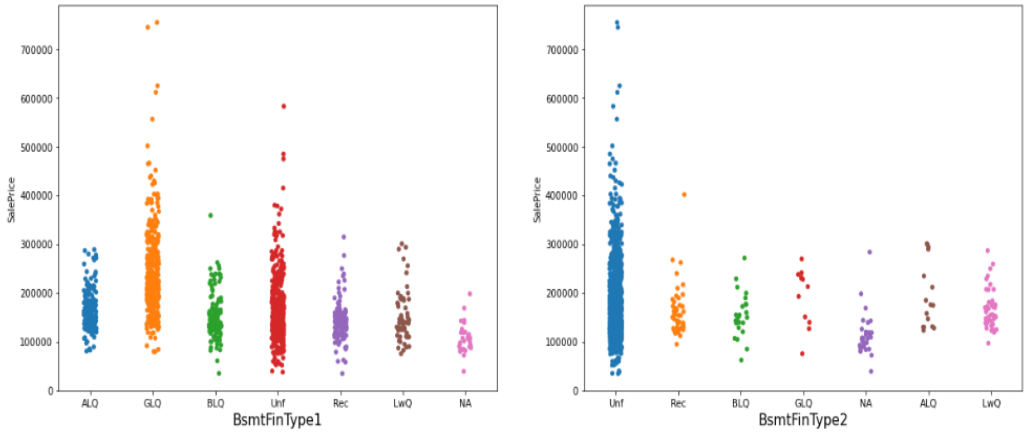
Most of the properties have either poured concrete, or cinder block or brick-tile in their foundation. Starting price and maximum price of poured concrete type foundation is generally higher.

Properties with excellent basement have higher starting price and the maximum price goes beyond 700K. Those in good or average condition doesn't go beyond 500K.

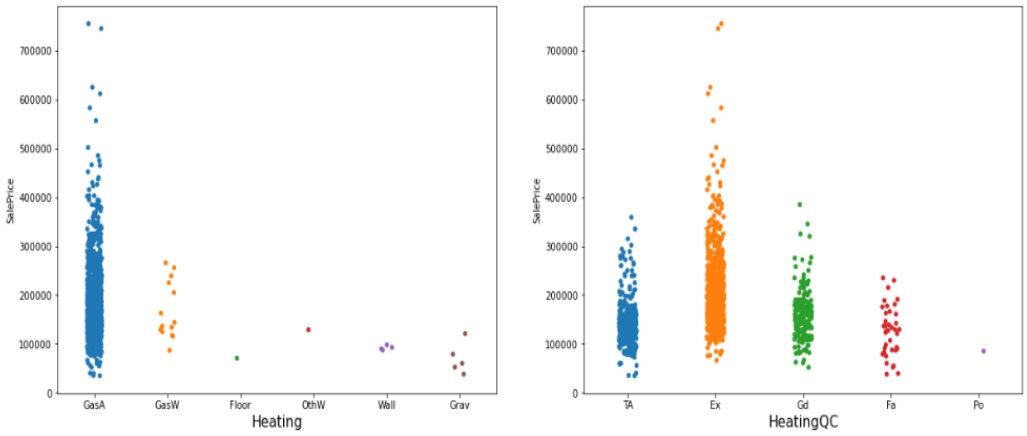
Strip plot of *BsmtCond* and *BsmtExposure* with respect to *SalePrice*:



Strip plot of *BsmtFinType1* and *BsmtFinType2* with respect to *SalePrice*:



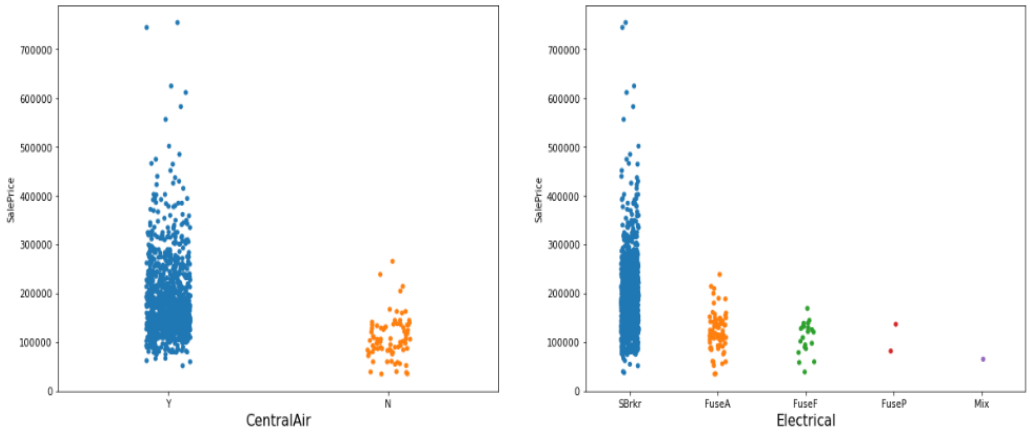
Strip plot of *Heating* and *HeatingQC* with respect to *SalePrice*:



A high majority of properties have GasA (Gas forced warm air furnace) type heating. Its price range is much higher than the other types. All the properties having price higher than 300K have this type of heating system.

Houses with better heating quality yield higher prices.

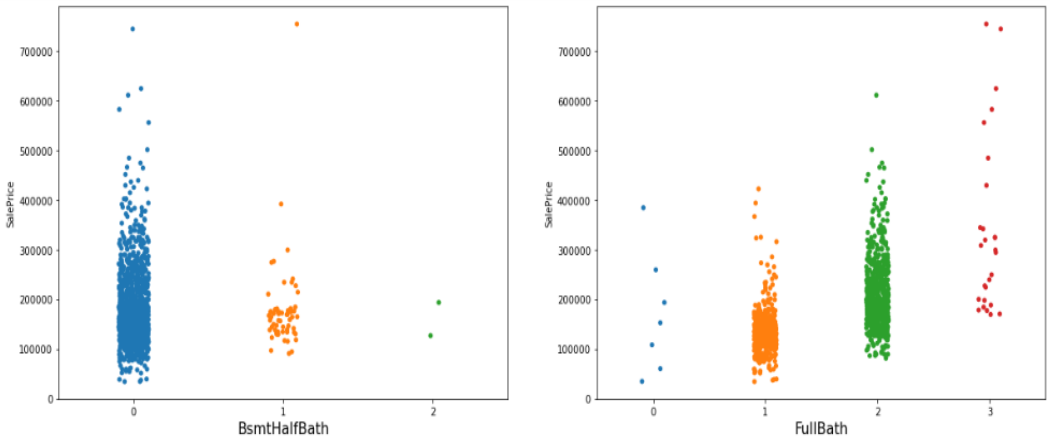
Strip plot of *CentralAir* and *Electrical* with respect to *SalePrice*:



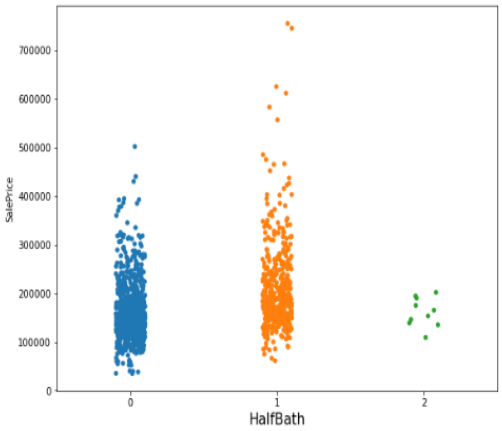
Most of the properties have central air. Those with no central air have much lesser price and don't go beyond 300K.

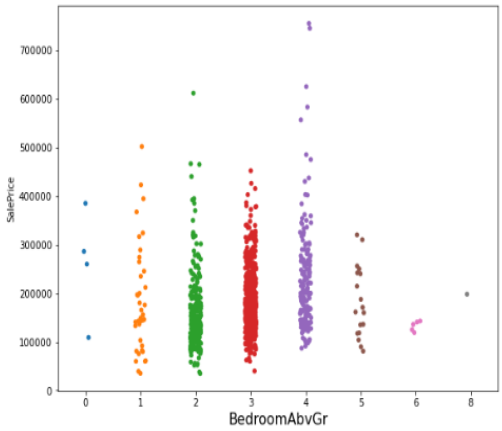
Most of the properties have SBrkr (Standard Circuit Breakers & Romex) type electrical system. All the properties of price higher than 250K have this type of electrical system.

Strip plot of *BsmtHalfBath* and *FullBath* with respect to *SalePrice*:

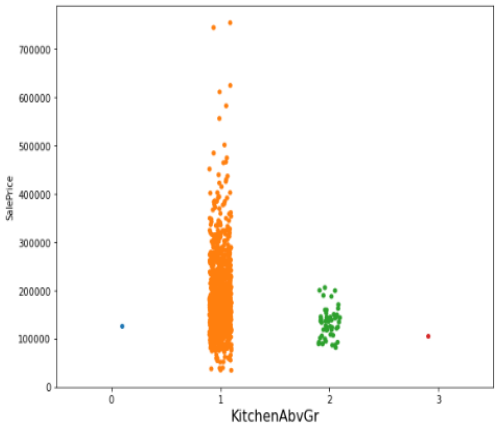


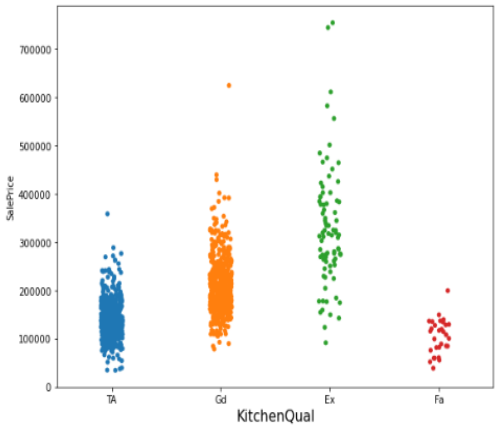
Strip plot of *HalfBath* and *BedroomAbvGr* with respect to *SalePrice*:





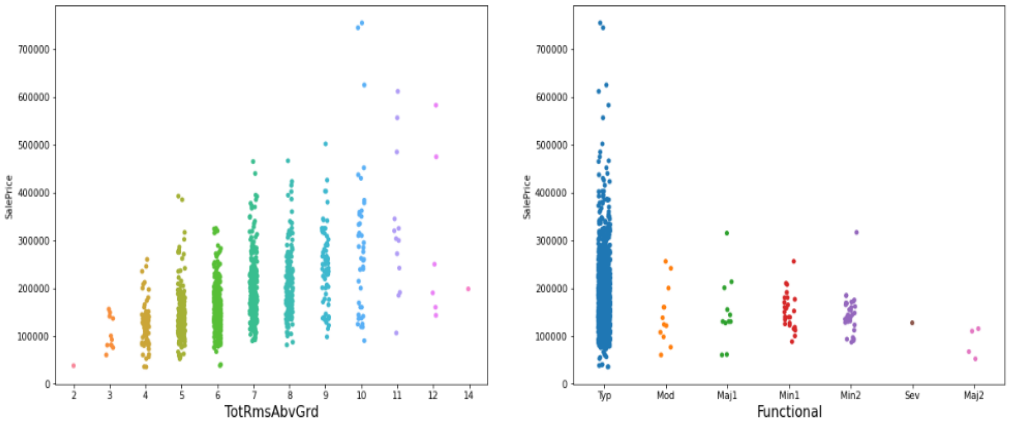
Strip plot of *KitchenAbvGr* and *KitchenQual* with respect to *SalePrice*:





A high majority of properties have 1 kitchen above grade. Better the kitchen quality better is the price and range of price.

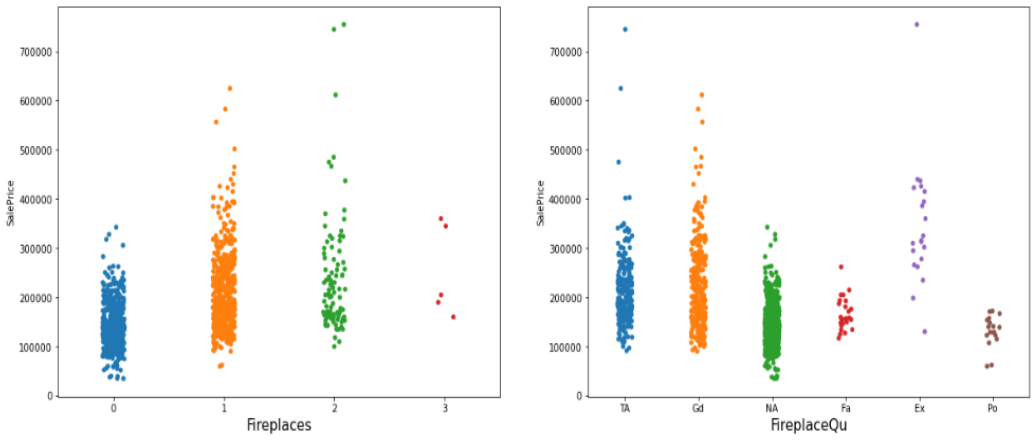
Strip plot of *TotRmsAbvGrd* and *Functional* with respect to *SalePrice*:



Total rooms above grade varies from 2 to 14.

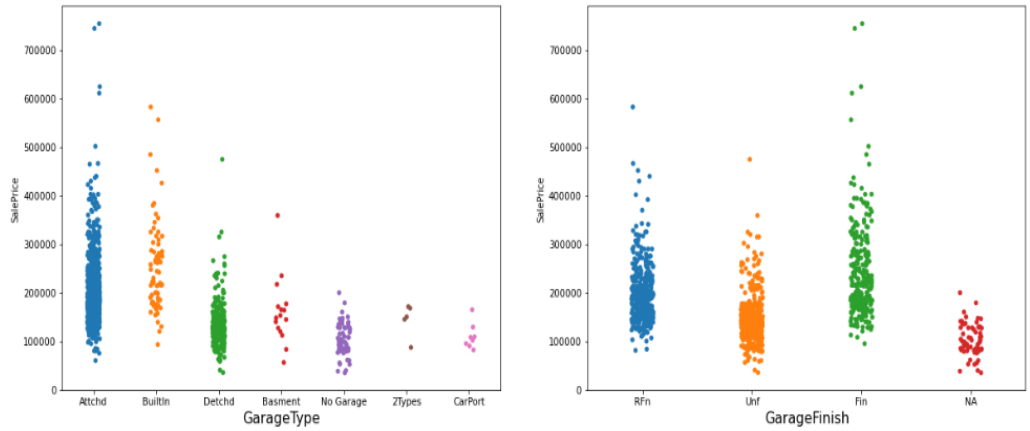
Most of the properties are of typical home functionality and no deductions are warranted. The price range is also much higher.

Strip plot of *Fireplaces* and *FireplaceQu* with respect to *SalePrice*:



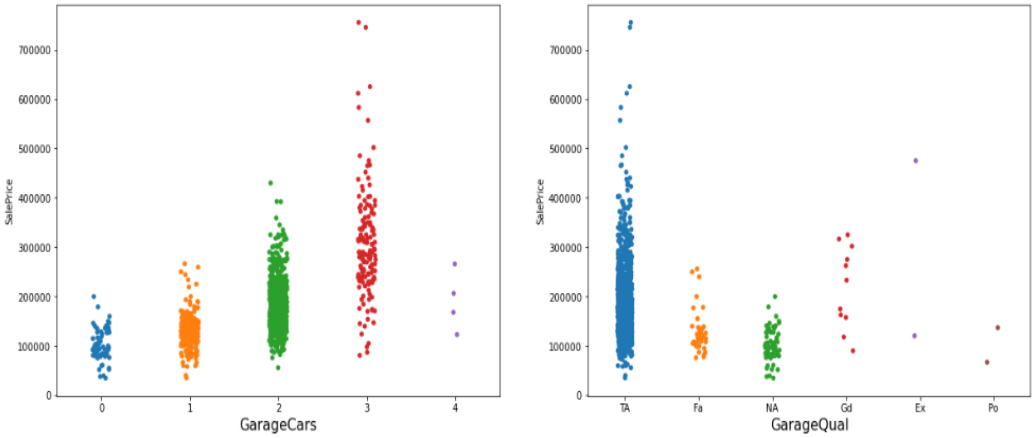
Very few houses have more than 2 fireplaces. Having fireplace enhances the price.

Strip plot of *GarageType* and *GarageFinish* with respect to *SalePrice*:



Houses with attached or built-in garage have higher starting price. Almost all the houses beyond 350K have either attached or built-in garage.

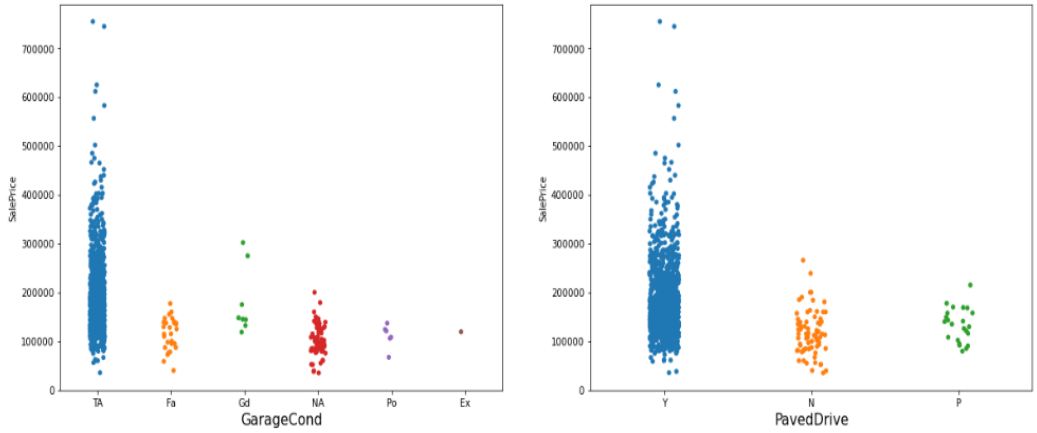
Strip plot of *GarageCars* and *GarageQual* with respect to *SalePrice*:



Properties of price above 300K have space for multiple cars in the garage.

Most of the properties have the garage of average quality.

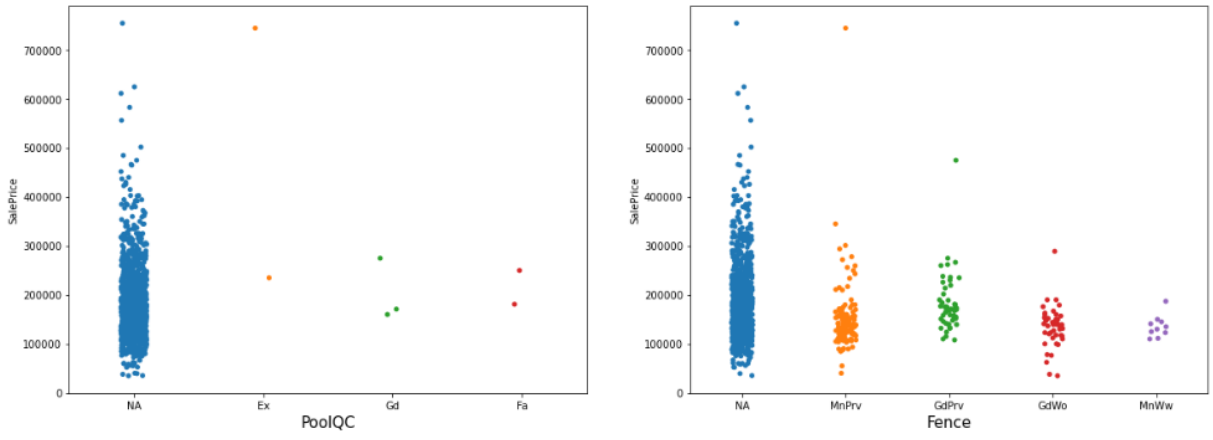
Strip plot of *GarageCond* and *PavedDrive* with respect to *SalePrice*:



Most of the properties have the garage in average condition.

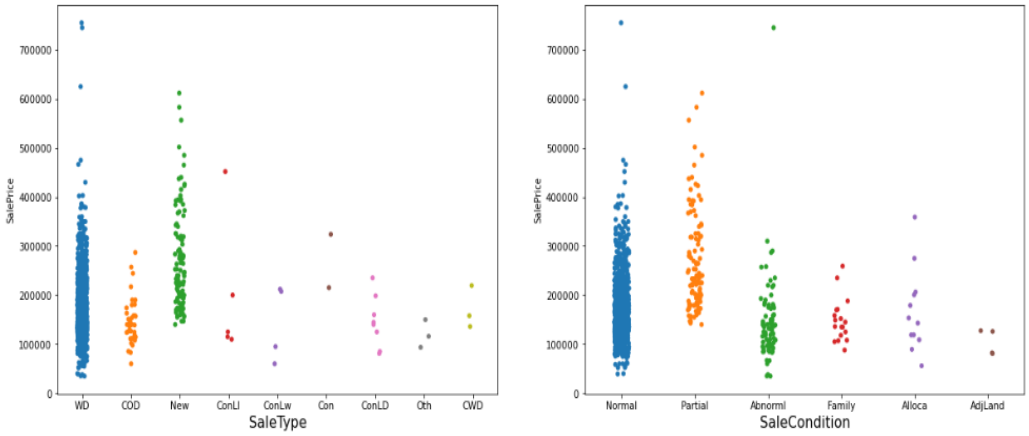
Most of the properties have paved driveway. Those with dirt/gravel or partial pavement have lower price. All the properties of price higher than 300K have paved driveway.

Strip plot of *PoolQC* and *Fence* with respect to *SalePrice*:



Majority of properties doesn't have pool and fence. However this doesn't affect the price much as we can see numerous data points with good sale price.

Strip plot of *SaleType* and *SaleCondition* with respect to *SalePrice*:



Majority of the sale type is of WD(Warranty Deed - Conventional) or New(Home just constructed and sold). New houses have a higher starting price compared to others.

Majority of sale condition is of type Normal (Normal Sale) or Partial (Home was not completed when last assessed (associated with New Homes)). Partial type has higher starting price.

CONCLUSION

- Key Findings and Conclusions of the Study
  1) The sale price of a property depends on a lot of factors. The top 16 features affecting the sale price of a property are shown below in a list along with the Pearson standard correlation coefficient.

	Feature	Corr_with_SalePrice
0	OverallQual	0.789185
1	GrLivArea	0.694691
2	ExterQual	0.672665
3	KitchenQual	0.659228
4	GarageCars	0.628329
5	GarageArea	0.619000
6	TotalBsmtSF	0.585797
7	BsmtQual	0.585348
8	1stFlrSF	0.573479
9	FullBath	0.554988
10	GarageFinish	0.550624
11	TotRmsAbvGrd	0.528363
12	YearBuilt	0.514408
13	FireplaceQu	0.513185
14	YearRemodAdd	0.507831
15	GarageYrBlt	0.500487

  2) Random Forests Regression Algorithm is giving the best fit results for prediction of sale price. It gives R^2 score of around 85% in cross-validation.

- Limitations of this work and Scope for Future Work
    1) For better understanding of the housing market, model should be trained with more number of entries.
    2) Some of the continuous numeric columns have highly skewed data. In this project only power transformation is used to tackle skewness. Various other data-transformation techniques can be tested to improve the quality of data.
    3) The hyperparameters of our final model can be further tuned.
    4) In this project we worked on only 4 algorithms. There are numerous other regression algorithms with which we can try to make a better model.