CASE STUDY
House Rent Prediction

TEAM MEMBERS:

DHANUSHKA SHREE : CB.SC.I5DAS19036

Table of content:

Abstract

Building machine learning model for correctly predicting the house price with the help of many factors like location, sqft of basement, nuil

Problem statement

Main aim is to predict The King County's home prices. This dataset contains house sale prices for King County, which includes Seattle given the houses sold between May 2014 and May 2015.

Motivation

- To help the developer determine the selling price of a house and can help the customer to arrange the right houses to have in their choices.
- To build model with house type and different facilities

Requirements

- NUMPY
- PANDAS
- MATPLOTLIB
- SCIKIT
- SKLEARN

Challenges

- As the initial house price prediction were challenging, it required some best method to get accurate prediction.
- Data quality is a key factor to predict the house prices and missing features(NaN))are a difficult aspect to handle in machine learning models let alone house prediction model.
- In this study, several methods of prediction were compared to finding the best predicted results in determining a house's selling price compared to the actual price.
- This project brings the latest research on regression technique that can be used for house prediction such as Linear regression, KNN , Logistic regression , Random Forest
- To find the best algorithm among these algorithm R^2 method is used to find the accuracy.

About Data

The dataset have most of the possible attributes which influence the price of the house .

Namely : no. of bedrooms & bathrooms ,view ; sqft measurements of basement , above , living etc and also latitude and longitude

shape : (21613 , 21)

column names : 'id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15'

```
[ ] house.head()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_
0	245000.0	3	1.00	1180	5650.0	1.0	0	0	3	7	1180	0	1955	0	98155	47.5112	-122.257	
1	538000.0	3	2.25	2570	7242.0	2.0	0	0	3	7	2170	400	1951	0	98125	47.7210	-122.319	
2	245000.0	2	1.00	1090	10000.0	1.0	0	0	3	6	970	0	1933	0	98028	47.7279	-122.233	
3	604000.0	4	3.00	1960	5000.0	1.0	0	0	4	7	1050	910	1965	0	98136	47.5208	-122.372	
4	510000.0	3	2.00	1680	8080.0	1.0	0	0	3	8	1680	0	1987	0	98074	47.6168	-122.045	

Data pre-processing

Duplicate Handling

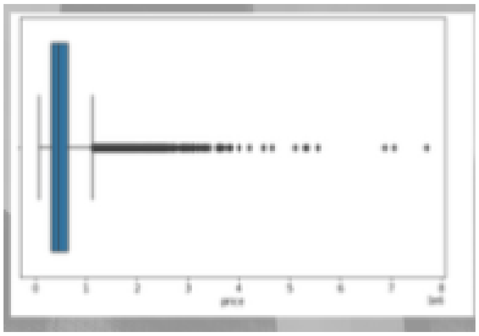
```
[ ] house.duplicated().sum()

    # NO DUPLICATE VALUES

    0
```

Handling Null values

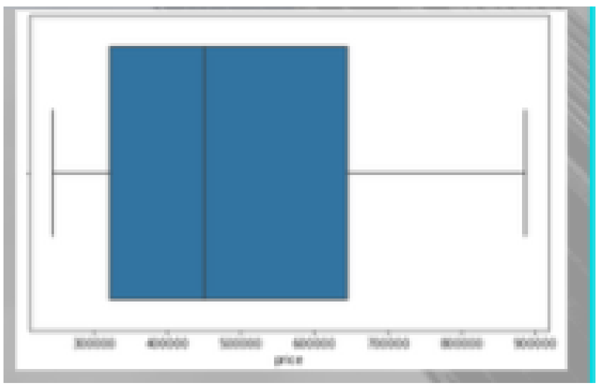
```
house.isnull().sum()

# NO NULL VALUES

id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Outlier Treatment

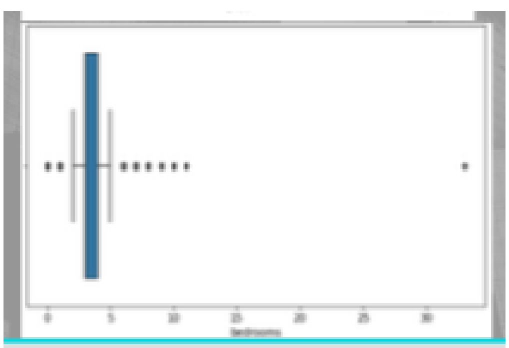
Before:



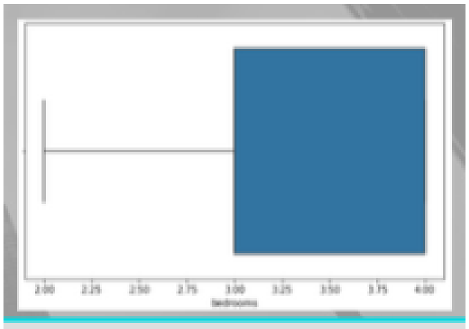
After:



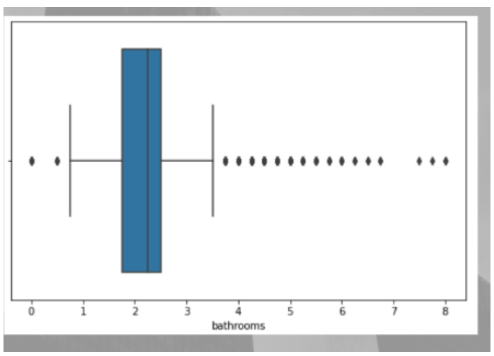
Before:



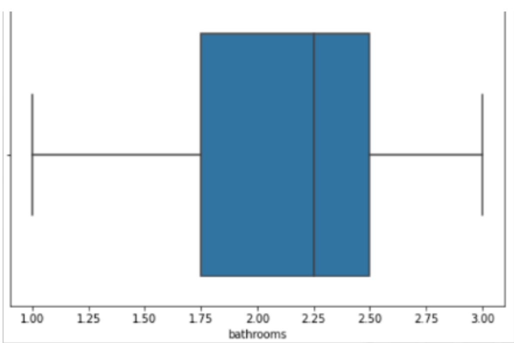
After:



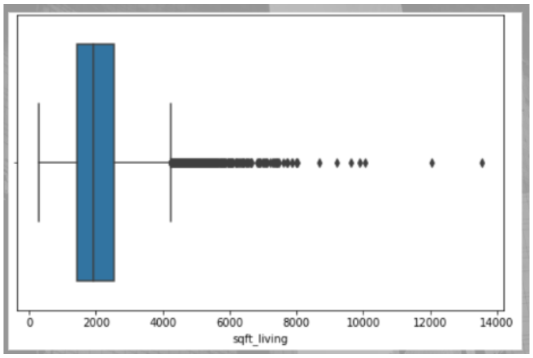
Before:



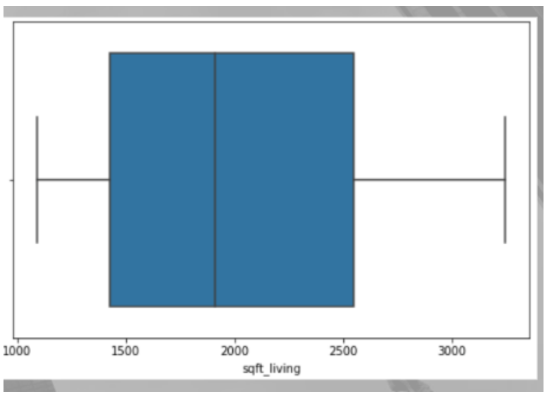
After:



Before:



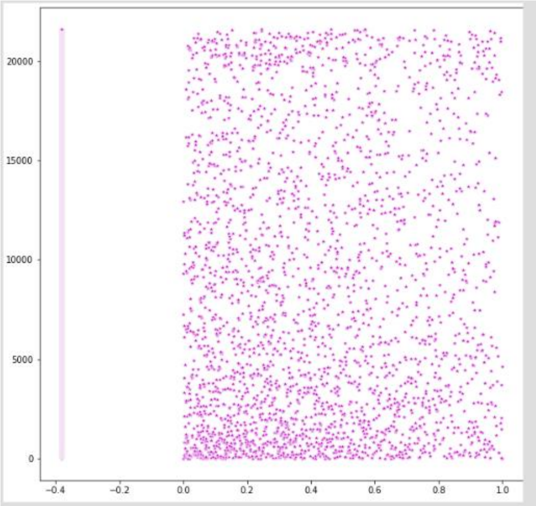
After:



Encoding Variable

Conversion of categorical variable into numerical variable so that it can be used in exploratory data analysis. Then we can get relationship between other attributes.

Normalization



After normalizing we can see that the values of price column are within 0 and 1.

Model Fitting

Random Forest

```
rfr_model=RandomForestRegressor(n_estimators=200)
rfr_model.fit(x_train,y_train)
print("Random Forest: ",rfr_model.score(x_test,y_test))
```

```
Random Forest:  0.8899202077312407
```

```
l=[i for i in range(1,101)]
kfold = KFold(n_splits=10, random_state=None)
parameter= {"max_depth": [2,7,9,11,13,15,None],
            "max_features":['auto', 'sqrt', 'log2',None],
            "max_leaf_nodes":l,
            'min_samples_leaf':l}
rfr_model1= RandomForestRegressor()
rfr_model1_tuning= RandomizedSearchCV(rfr_model1, parameter, cv = 5)
rfr_model1_tuning.fit(x_train, y_train)
print("Tuned Random forest classifier Parameters: {}".format(rfr_model1_tuning.best_params_))
print("Best score is {}".format(rfr_model1_tuning.best_score_))
```

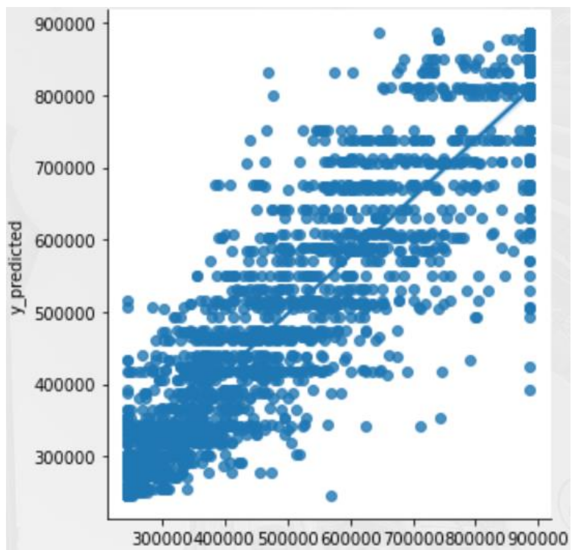
```
Tuned Random forest classifier Parameters: {'min_samples_leaf': 21, 'max_leaf_nodes': 75, 'max_features': 'auto', 'max_depth': 11}
Best score is 0.8440442222786213
```

Decision Tree

```
decision_tree=DecisionTreeRegressor(min_samples_leaf=.01)
decision_tree.fit(x_train1,y_train1)
y_preds=decision_tree.predict(x_test1)
accuracy.append(r2_score(y_test1,y_preds)*100)
model.append('Decision Tree')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds))
print("Accuracy of Decision Tree Regressor without PCA: ",r2_score(y_test1,y_preds)*100)
print("The mean squared error of Decision tree regressor without pca is: ",mean_squared_error(y_test1,y_preds))
print(" ")

Accuracy of Decision Tree Regressor without PCA:  80.41872903515707
The mean squared error of Decision tree regressor without pca is:  8447306952.786092
```

	y	y_predicted
13180	335000.0	380774.600000
19315	539950.0	642008.250951
5902	887000.0	877226.838235
15546	295000.0	276797.359712
13964	720000.0	510214.883249
...	...	...
8472	452000.0	464674.857520
11406	387500.0	549503.661692
19803	715000.0	738661.015385
6815	473975.0	589866.760234
16366	399000.0	391724.322835

2162 rows × 2 columns



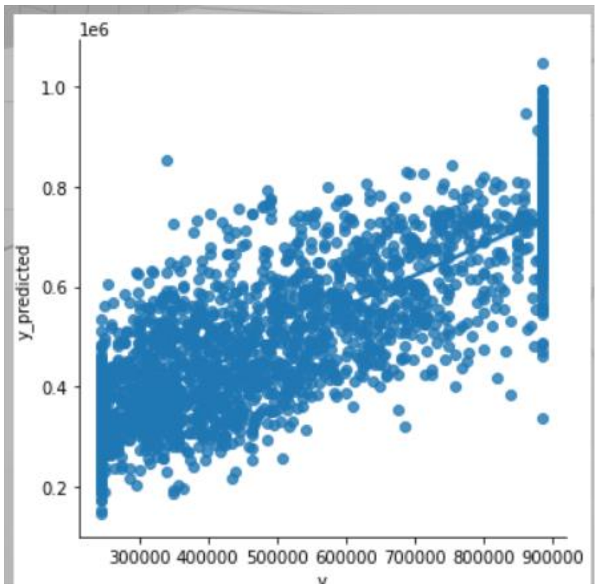
Linear Regression:

```
mlrm1=LinearRegression()
num1=90
model2= BaggingRegressor(base_estimator=mlrm1, n_estimators=num1)
model2.fit(x_train2, y_train2)
y_preds4= model2.predict(x_test2)
print('Accuracy of linear regression method is ',r2_score(y_test2,y_preds4)*100)
print("the mean squared error of the decision tree classifier with PCA and bagging is: ",mean_squared_error(y_test2,y_preds4))
mse.append(mean_squared_error(y_test2,y_preds4))
model.append('Linear regression')
with_pca.append(1)
accuracy.append(r2_score(y_test2,y_preds4)*100)
```

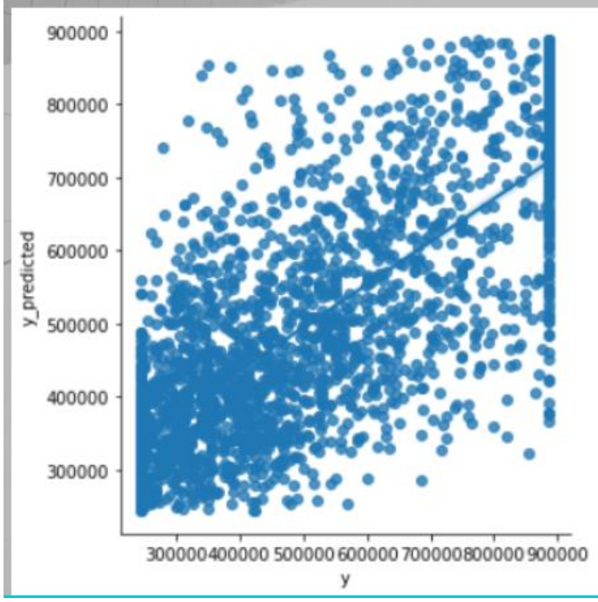
```
Accuracy of linear regression method is  62.57315562583131
the mean squared error of the decision tree classifier with PCA and bagging is:  16515702772.259222
```

	y	y_predicted
9643	550000.0	593768.317167
6904	349900.0	509862.854890
10239	257000.0	423815.793541
3869	887000.0	730099.961913
18260	493000.0	553112.645903
...	...	...
10467	359000.0	564024.130144
17460	245000.0	297004.647059
1580	380000.0	521816.627734
11433	245000.0	350533.639788
11589	887000.0	767790.714935

2162 rows × 2 columns



KNN Regressor:

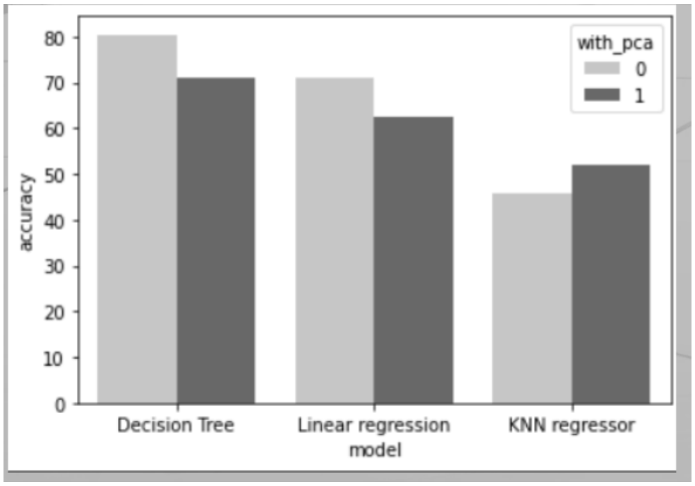


	y	y_predicted
9643	550000.0	477110.000000
6904	349900.0	570560.370370
10239	257000.0	345620.370370
3869	887000.0	582686.666667
18260	493000.0	695218.518519
...	...	...
10467	359000.0	425788.518519
17460	245000.0	333857.407407
1580	380000.0	409914.814815
11433	245000.0	298508.148148
11589	887000.0	575267.407407

```
neigh1= KNeighborsRegressor(n_neighbors=3)
num2=90
model3= BaggingRegressor(base_estimator=neigh1, n_estimators=num2)
model3.fit(x_train2, y_train2)
y_preds5= model3.predict(x_test2)
print("the accuracy of this model is: ",r2_score(y_test2,y_preds5)*100)
print("The mean squared error of KNN regressor without pca is: ", mean_squared_error(y_test2,y_preds5))
model.append('KNN regressor')
with_pca.append(1)
mse.append(mean_squared_error(y_test2,y_preds5))
accuracy.append(r2_score(y_test2,y_preds5)*100)
```

```
the accuracy of this model is:  52.131426279732416
The mean squared error of KNN regressor without pca is:  21123424881.67568
```

	model	with_pca	accuracy	mean_squared_error
0	Decision Tree	0	80.418729	8.447307e+09
1	Linear regression	0	70.873830	1.256495e+10
2	KNN regressor	0	45.638323	2.345148e+10
3	Decision Tree	1	71.089838	1.275746e+10
4	Linear regression	1	62.573156	1.651570e+10
5	KNN regressor	1	52.131426	2.112342e+10



BOOSTING:

ADA Boosting

Without PCA

```
x_train3, x_test3, y_train3, y_test3 = train_test_split(x,y,test_size=0.10)
```

```
a = AdaBoostRegressor()
a.fit(x_train3,y_train3)
y_preds6=a.predict(x_test3)
print("the accuracy of ada boosting regressor is: ",r2_score(y_test3,y_preds6)*100)
print("The mean squared error of ada boosting regressor without pca is: ", mean_squared_error(y_test3,y_preds6))
```

```
the accuracy of ada boosting regressor is:  53.57170005959174
The mean squared error of ada boosting regressor without pca is:  20525941749.636974
```

With PCA

```
a1= AdaBoostRegressor()
a1.fit(x_train2,y_train2)
y_preds7=a1.predict(x_test2)
print("the accuracy of ada boosting regressor is: ",r2_score(y_test2,y_preds7)*100)
print("The mean squared error of ada boosting regressor with pca is: ", mean_squared_error(y_test2,y_preds7))
```

```
the accuracy of ada boosting regressor is:  56.3488473788736
The mean squared error of ada boosting regressor with pca is:  19262363002.065434
```

Gradient Boost

Without PCA

```
g= AdaBoostRegressor()
g.fit(x_train3,y_train3)
y_preds8=g.predict(x_test3)
print("the accuracy of this model is: ",r2_score(y_test3,y_preds8)*100)
print("The mean squared error of gradient boosting regressor without pca is: ", mean_squared_error(y_test3,y_preds8))
```

```
the accuracy of this model is:  53.78762876286631
The mean squared error of gradient boosting regressor without pca is:  20430479714.818188
```

With PCA

```
g1= AdaBoostRegressor()
g1.fit(x_train2,y_train2)
y_preds9=g1.predict(x_test2)
print("the accuracy of this model is: ",r2_score(y_test2,y_preds9)*100)
print("The mean squared error of Gradient boosting regressor with pca is: ", mean_squared_error(y_test2,y_preds9))
```

```
the accuracy of this model is:  54.23019039181154
The mean squared error of Gradient boosting regressor with pca is:  20197283101.790363
```

ADA Boosting has higher accuracy compared to Gradient Boosting which is 56.35