Sri Lanka Institute of Information Technology

Year 4 | Semester 1 | 2022

ESTIMATING HOUSE PRICES USING DIFFERENT REGRESSION MODELS AND COMPARE THE PERFORMANCE OF THE MODELS

Machine Learning (IT4060)

Assignment 02

Submitted By:

Student ID	**Name**
IT19120126	Fernando W.P.S.
IT19073460	Vidhanahena B.L.O
IT19106502	Madhubashana I.K.

Table of Contents

1. Introduction

Property value estimation and prediction are crucial in the real estate industry. Potential homeowners, investors, appraisers, tax assessors, and other real estate sector participants, such as mortgage lenders and insurers, all benefit from a realistic house price estimate. A cost and a percentage of the cost are the usual methods for assessing the worth of a house. There are no agreed-upon criteria or methods for comparing selling prices. As a result, having a home is quite beneficial. The price estimation approach will fill a knowledge gap and improve the real estate market's performance.

Machine learning models, particularly in the field of artificial intelligence, are growing increasingly complex as technology improves. be used to predict property prices after being educated on previous market data In this project, the cost of housing will be calculated using machine learning, and an estimate model will be created.

The machine learning model will be trained using the dataset of KC House property prices. The dataset has around 30,000 records with 21 columns. This dataset contains the bulk of the qualities (features) that individuals examine when buying a property. Other factors, however, are less important when making a purchase. A house. During the data cleansing process, certain traits must be removed.

The column description of the dataset is as follows.

Column Name	Data Type	Description
date	object	Date of the home sale
price	float64	Price of the house
bedrooms	int64	Number of bedrooms
bathrooms	float64	Number of bathrooms
sqft_living	int64	Land area
sqft_lot	int64	apartment interior living space of Square foot
floors	int64	The number of floors in the house
waterfront	int64	A dummy variable that indicates whether the apartment has a view of the water or not.
view	int64	A scale of 0 to 4 indicating how nice the property's view was.
condition	int64	The apartment's condition is graded on a scale of 1 to 5.
grade	int64	An index ranging from 1 to 13, with 1-3 indicating poor building construction and design, 7 indicating average construction and design, and 11-13 indicating excellent quality construction and design.
sqft_above	int64	The square footage of the

		above-ground level internal dwelling space
sqft_basement	int64	Interior housing space below ground level square footage
yr_built	int64	The year the home was constructed
yr_renovated	int64	The year in which the home was last renovated
zipcode	int64	What zipcode area the house is in
lat	float64	The house's latitude
long	float64	The house's longitude
sqft_living15	int64	The interior living space for the nearest 15 neighbors in square feet
sqft_lot15	int64	The total square footage of the nearest 15 neighbors' property lots

The URL for the dataset is as follows
https://www.kaggle.com/datasets/shivachandel/kc-house-data

2. Methodology

2.1 Data Analyzing and Visualizing

Foremost it is better to do an analysis on the data that gathered.

- Use shape function to get the size of the data frame. As shown below there are 21,613 rows and 21 columns in the data frame this indicate that the dataset is large enough to train a machine learning model.

```
#Display number of rows and number of columns
```

```
df.shape
```

```
(21613, 21)
```

- Get how many unique values in each categorical column. High-cardinality columns (columns with almost all values are unique) like id are not suitable for regression so they will be removed.

```
print("Column Name   Unique Values")
print("-----------   -------------")
for column in df.columns:
    if(df[column].dtypes != df['floors'].dtypes and df[column].dtypes != df['price'].dtypes):
        print(column + ': \t' + str(df[column].nunique()))
```

```
Column Name   Unique Values
-----------   -------------
id:     21436
date:   372
bedrooms:       13
sqft_living:    1038
sqft_lot:       9782
waterfront:     2
view:   5
condition:      5
grade:  12
sqft_above:     946
sqft_basement:  306
yr_built:       116
yr_renovated:   70
zipcode:        70
sqft_living15:  777
sqft_lot15:     8689
```

View how many null values in the data frame.

- As the figure shows there are no null values in this data frame.

```
#Dispaly null values in each column
```

```
df.isnull().sum()
```

```
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
```

View the Correlation Matrix

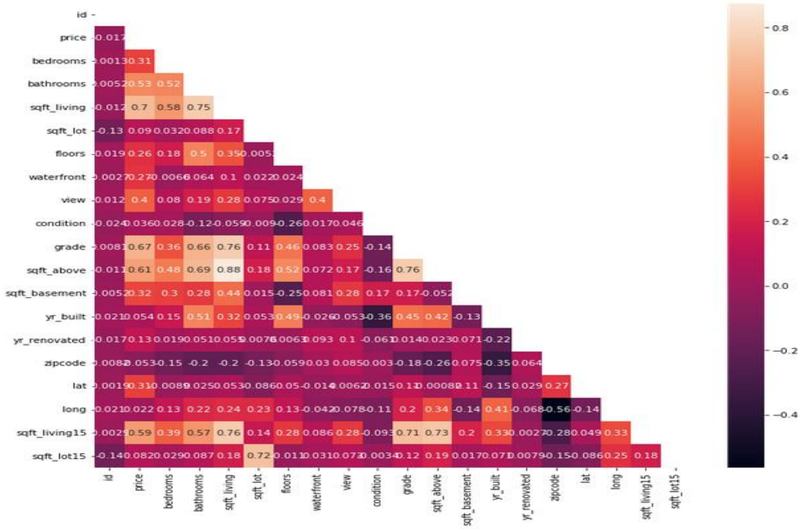
- By taking each pair of features as a couple the correlation matrix will shows each feature's linear correlations. Figure 2.5 displays the correlation matrix as a heats map. Considering the heat map, we can see the correlation values are in between -0.5 and +0.8. It concludes that in this data there are no strong co-related features. Remove features based on co-relations are not needed in this case.

```
# Display the Correlation Matrix
```

```
df.corr()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	co
id	1.000000	-0.016762	0.001286	0.005160	-0.012258	-0.132109	0.018525	-0.002721	0.011592	-0.0
price	-0.016762	1.000000	0.308350	0.525138	0.702035	0.089661	0.256794	0.266369	0.397293	0.0
bedrooms	0.001286	0.308350	1.000000	0.515884	0.576671	0.031703	0.175429	-0.006582	0.079532	0.0
bathrooms	0.005160	0.525138	0.515884	1.000000	0.754665	0.087740	0.500653	0.063744	0.187737	-0.1
sqft_living	-0.012258	0.702035	0.576671	0.754665	1.000000	0.172826	0.353949	0.103818	0.284611	-0.0
sqft_lot	-0.132109	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005201	0.021604	0.074710	-0.0
floors	0.018525	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000000	0.023698	0.029444	-0.2
waterfront	-0.002721	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023698	1.000000	0.401857	0.0
view	0.011592	0.397293	0.079532	0.187737	0.284611	0.074710	0.029444	0.401857	1.000000	0.0
condition	-0.023783	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.016653	0.045990	1.0
grade	0.008130	0.667434	0.356967	0.664983	0.762704	0.113621	0.458183	0.082775	0.251321	-0.1
sqft_above	-0.010842	0.605567	0.477600	0.685342	0.876597	0.183512	0.523885	0.072075	0.167649	-0.1
sqft_basement	-0.005151	0.323816	0.303093	0.283770	0.435043	0.015286	-0.245705	0.080588	0.276947	0.1
yr_built	0.021380	0.054012	0.154178	0.506019	0.318049	0.053080	0.489319	-0.026161	-0.053440	-0.3
yr_renovated	-0.016907	0.126434	0.018841	0.050739	0.055363	0.007644	0.006338	0.092885	0.103917	-0.0
zipcode	-0.008224	-0.053203	-0.152668	-0.203866	-0.199430	-0.129574	-0.059121	0.030285	0.084827	0.0
lat	-0.001891	0.307003	-0.008931	0.024573	0.052529	-0.085683	0.049614	-0.014274	0.006157	-0.0
long	0.020799	0.021626	0.129473	0.223042	0.240223	0.229521	0.125419	-0.041910	-0.078400	-0.1
sqft_living15	-0.002901	0.585379	0.391638	0.568634	0.756420	0.144608	0.279885	0.086463	0.280439	-0.0
sqft_lot15	-0.138798	0.082447	0.029244	0.087175	0.183286	0.718557	-0.011269	0.030703	0.072575	-0.0

Plot the Correlation Matrix



View Statistics of each column

```
#Dispaly statics of each column
```

```
df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	216
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	

Histogram of every numeric column

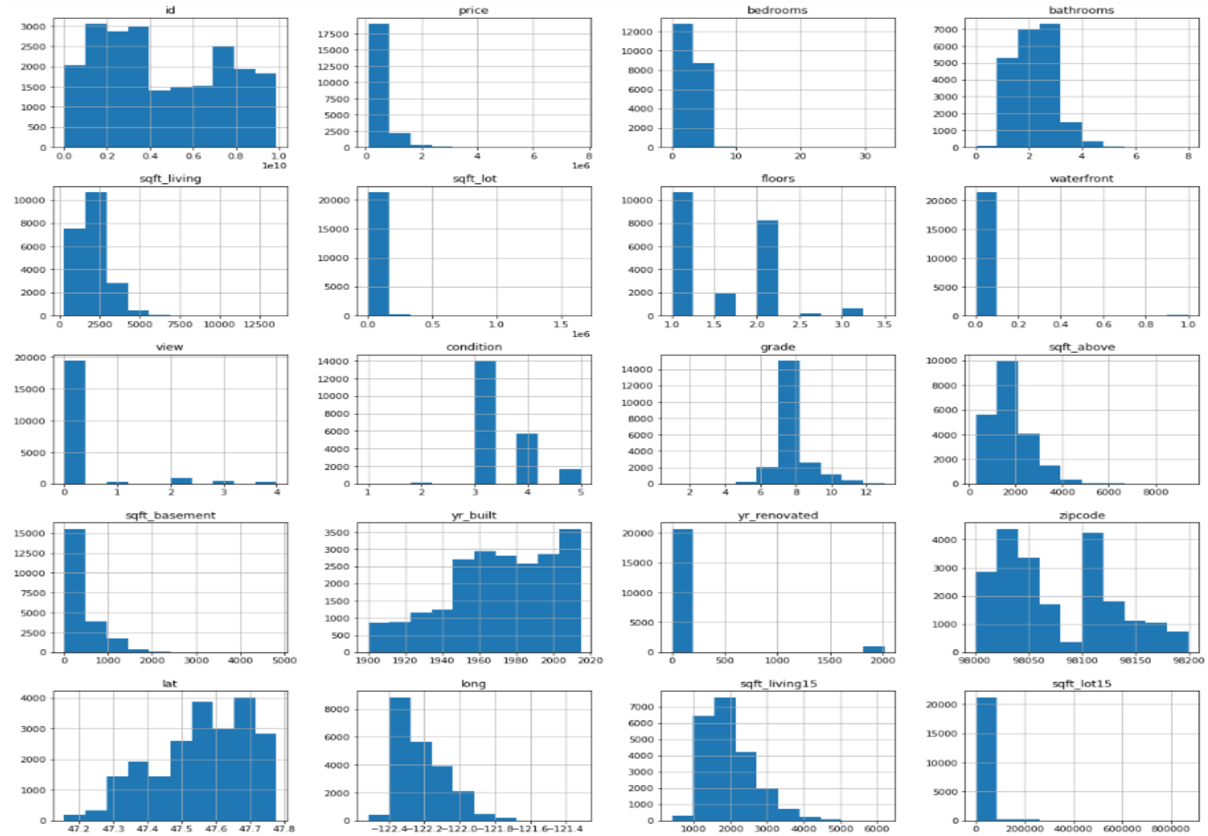
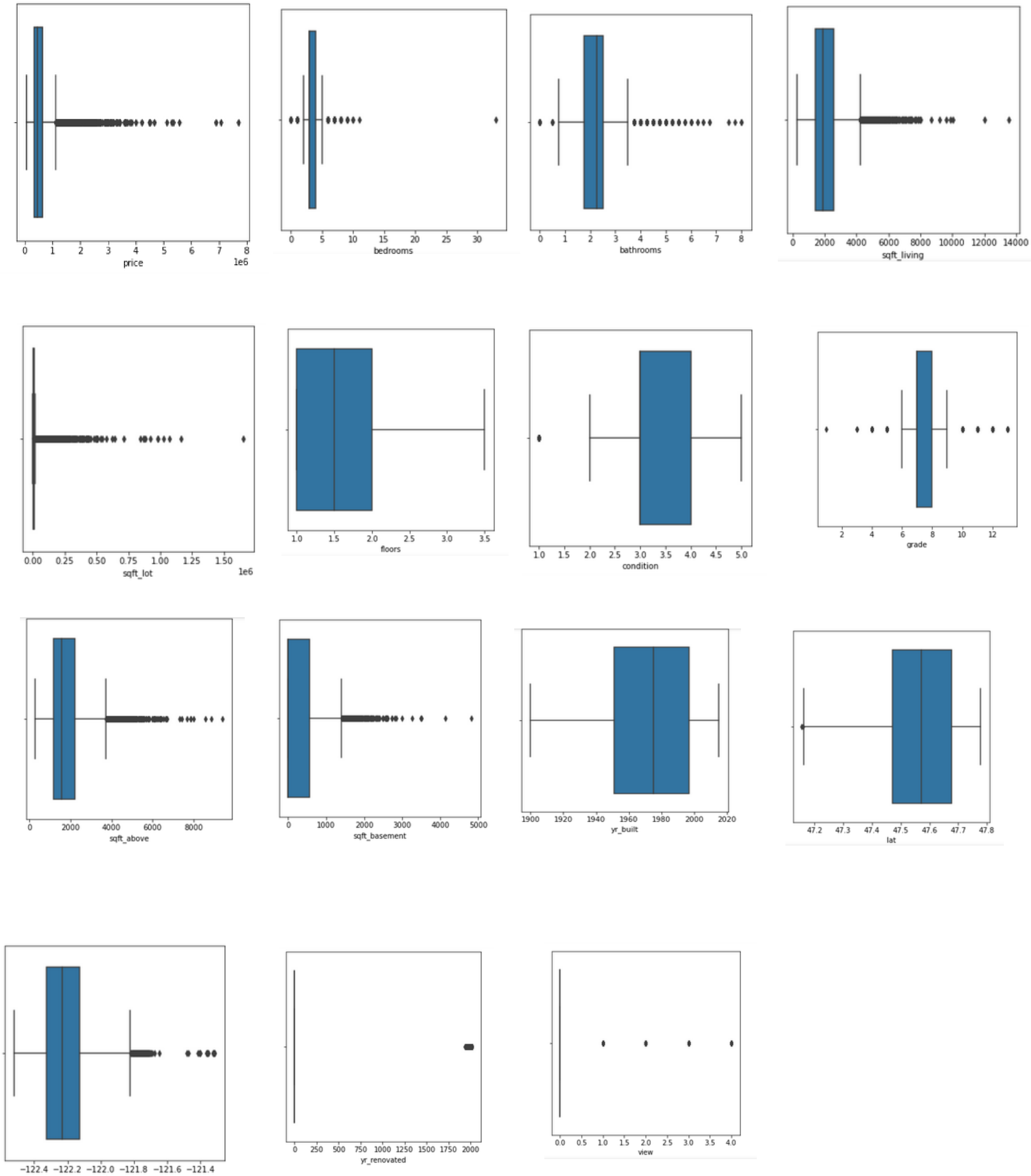


Figure 2.7

Boxplot for every numeric column



2.2 Data Cleaning

Data cleaning is the process of locating and updating, eliminating or replacing elements of data that are incorrect, incomplete, unreliable, obsolete, or unavailable. Model training on machine learning requires data cleansing, which is a critical component.
We did the updates to new data frames as a good practice when working with data frames.

2.2.1 Basic Data Cleaning

Dropping the non-essential features in the dataset.

```
In [13]: df2 = df1.drop([ 'zipcode', 'date', 'waterfront', 'view', 'yr_renovated'],
                    axis='columns')
         df2.head(3)
```

Out[13]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_basement	yr_built	lat	long	sqft_living15	sqft_lot1
0	221900.0	3	1.00	1180	5650	1.0	3	7	1180	0	1955	47.5112	-122.257	1340	565
1	538000.0	3	2.25	2570	7242	2.0	3	7	2170	400	1951	47.7210	-122.319	1690	763
2	180000.0	2	1.00	770	10000	1.0	3	6	770	0	1933	47.7379	-122.233	2720	806

Dealing with lost values. The ' floors' and 'yr_built' columns we identified in the data analysis have zero values. Therefore, they will be replaced with the mean value of the column.

```
In [14]: # Fill missing values of 'floors' and 'yr_built' columns
         df2['yr_built'] = df2['yr_built'].fillna(df2['yr_built'].median())
         df2['floors'] = df2['floors'].fillna(df2['floors'].median())

         df2.head(3)
```

Out[14]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_
0	221900.0	3	1.00	1180	5650	1.0	3	7	1180	
1	538000.0	3	2.25	2570	7242	2.0	3	7	2170	
2	180000.0	2	1.00	770	10000	1.0	3	6	770	

2.2.2 Detect and Remove Outliers

- Detect and Remove Outliers Using IQR

```
In [16]:
         # Remove outliers of all columns and assign to new data freame

         cols = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'condition', 'grade', 'sqft_above',
                 'sqft_basement','yr_built']
         Q1 = df1[cols].quantile(0.25)
         Q3 = df1[cols].quantile(0.75)
         IQR = Q3 - Q1

         df3 = df2[~((df1[cols] < (Q1 - 1.5 * IQR)) | (df1[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Abandonment of houses with more than 3 floors is unusual and can lead to errors. To do this, get the rows that meet the 'Number of floors <3' condition for further processing.

In [18]:
```
# See the houses there are floors more than 3, because it is uncommon and need to be removed from further processing
df4 = df3.copy()
df4 = df3[df3.floors < 3]
```

Leaving homes with +2 bathrooms rather than bedrooms is more common and can lead to errors. To do this, get rows that satisfy the condition 'number of bathrooms <number of bedrooms + 2'.

In [19]:
```
#See the houses there are more bathrooms than bedrooms count+2,because it is uncommon and need to be removed from further process

df4 = df3[df3.bathrooms < df3.bathrooms + 2]
```

2.3. Data Pre-Processing

Data pre-processing is the practice of processing raw data using a machine learning model. It is similar to the most important stages in data cleaning and building a machine learning model.

2.3.1. Data Standardization and Normalization

Convert floors, bathrooms and price column data types to integers.

In [22]:
```
#Convert bathrooms,floors and price into integers.
df5 = df4.copy()


df5['bathrooms'] = df5['bathrooms'].astype('int64')
df5['floors'] = df5['floors'].astype('int64')
df5['price'] = df5['price'].astype('int64')
```

Divide all the values by the maximum number in its column to get the values between the numbers 0 and 1 standard format.

In [24]:
```
df6 = df5.copy()

df6['price'] = df5['price'] / df5['price'].max()
df6['bedrooms'] = df5['bedrooms'] / df5['bedrooms'].max()
df6['bathrooms'] = df5['bathrooms'] / df5['bathrooms'].max()
df6['sqft_living'] = df5['sqft_living'] / df5['sqft_living'].max()
df6['sqft_lot'] = df5['sqft_lot'] / df2['sqft_lot'].max()
df6['floors'] = df2['floors'] / df5['floors'].max()
df6['condition'] = df5['condition'] / df5['condition'].max()
df6['grade'] = df5['grade'] / df5['grade'].max()
df6['sqft_above'] = df5['sqft_above'] / df5['sqft_above'].max()
df6['sqft_basement'] = df5['sqft_basement'] / df5['sqft_basement'].max()
df6['yr_built'] = df5['yr_built'] / df5['yr_built'].max()
df6['lat'] = df5['lat'] / df5['lat'].max()
df6['long'] = df5['long'] / df5['long'].max()
df6['sqft_living15'] = df5['sqft_living15'] / df5['sqft_living15'].max()
df6['sqft_lot15'] = df5['sqft_lot15'] / df5['sqft_lot15'].max()


df6
```

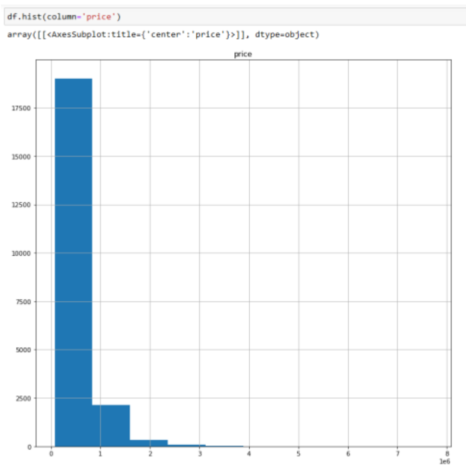
Out[24]:

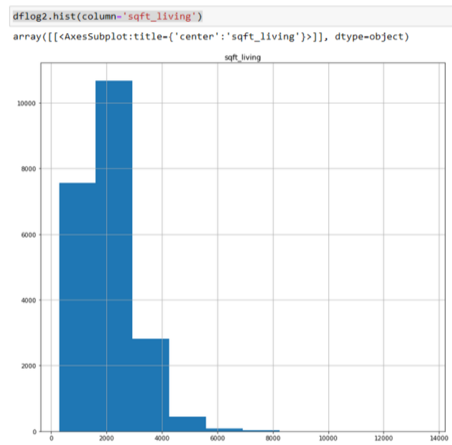
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqft_basement	yr_built	lat	long	sqft_livir
0	0.198125	0.6	0.333333	0.279621	0.003421	0.333333	0.6	0.777778	0.315508	0.000000	0.970223	0.994424	1.007732	0.27
1	0.480357	0.6	0.666667	0.609005	0.004385	0.666667	0.6	0.777778	0.580214	0.285714	0.968238	0.998815	1.008243	0.34
2	0.160714	0.4	0.333333	0.182464	0.006056	0.333333	0.6	0.666667	0.205882	0.000000	0.959305	0.999169	1.007534	0.54
3	0.539286	0.8	1.000000	0.464455	0.003028	0.333333	1.0	0.777778	0.280749	0.650000	0.975186	0.994625	1.008853	0.27
4	0.455357	0.6	0.666667	0.398104	0.004893	0.333333	0.6	0.888889	0.449198	0.000000	0.986104	0.996634	1.005984	0.36
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21608	0.321429	0.6	0.666667	0.362559	0.000685	1.000000	0.6	0.888889	0.409091	0.000000	0.997022	0.998361	1.008465	0.30
21609	0.357143	0.8	0.666667	0.547393	0.003520	0.666667	0.6	0.888889	0.617647	0.000000	0.999504	0.994414	1.008597	0.36
21610	0.359019	0.4	0.000000	0.241706	0.000818	0.666667	0.6	0.777778	0.272727	0.000000	0.997022	0.996166	1.008078	0.20
21611	0.357143	0.6	0.666667	0.379147	0.001446	0.666667	0.6	0.888889	0.427807	0.000000	0.994541	0.994912	1.006182	0.28
21612	0.290179	0.4	0.000000	0.241706	0.000652	0.666667	0.6	0.777778	0.272727	0.000000	0.996526	0.996159	1.008078	0.20

16823 rows × 15 columns

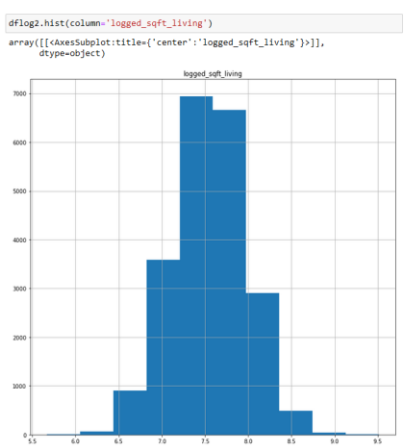
2.4. Log transformation

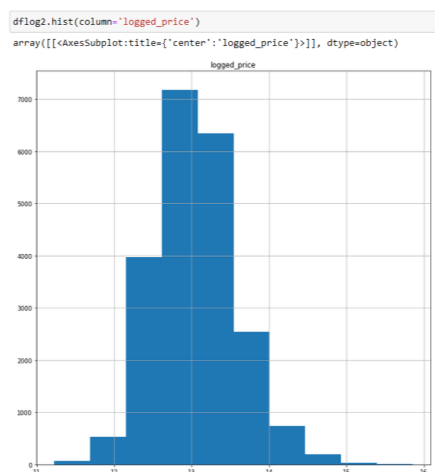
- According to the selected data frame, a certain number of variables have a skewed distribution.





- According to their range analysis which is greater than zero we determine to use log transformation on those variables. The Log transformation is one of the most popular transformation techniques used in feature engineering. The Log transformation is used to convert skewed distribution to a less-skewed distribution.





- Variable ‘price’ and variable ‘sqft_living’ are the variables that we used to do the Log transformation. In this transform, converting the selected variable’s column values to the log of the values then use these log values columns instead.

```
dflog2['logged_price'] =np.log(dflog2.price)
dflog2.head(5)
```

ondition	grade	sqft_above	sqft_basement	yr_built	lat	long	sqft_living15	sqft_lot15	logged_price
3	7	1180	0	1955	47	-122	1340	5650	12.309982
3	7	2170	400	1951	47	-122	1690	7639	13.195614
3	6	770	0	1933	47	-122	2720	8062	12.100712
5	7	1050	910	1965	47	-122	1360	5000	13.311329
3	8	1680	0	1987	47	-122	1800	7503	13.142166

```
dflog2['logged_sqft_living'] =np.log(dflog2.sqft_living)
dflog2.head(5)
```

ft_above	sqft_basement	yr_built	lat	long	sqft_living15	sqft_lot15	logged_price	logged_sqft_living
1180	0	1955	47	-122	1340	5650	12.309982	7.073270
2170	400	1951	47	-122	1690	7639	13.195614	7.851661
770	0	1933	47	-122	2720	8062	12.100712	6.646391
1050	910	1965	47	-122	1360	5000	13.311329	7.580700
1680	0	1987	47	-122	1800	7503	13.142166	7.426549

2.5. Model Training

2.5.1. Train-Test data splitting

The same dataset will be divided into two parts Training and test data. 80% (30,000 records) data will be training data and 20% will be test data

```
In [238]: # Split df into X and y
          df9=df6.copy()
          y = df9['price']
          X = df9.drop('price', axis=1)

In [239]: # Train-Test split

          X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)
```

2.5.2. Used Algorithms

2.5.1.1. Linear Regression (Multivariate)

The most often used supervised machine learning approach is linear regression. A regression analysis is performed. Regression models the intended predicted value based on independent variables. Its main goal is to predict and find connections between variables. To identify the value of a dependent variable (y) based on the value fluctuations of an independent variable, the linear regression technique was applied (x). A linear relationship exists between x (input) and y (output) as a result of this regression approach (output). It has been discovered (output). As a result, the term "linear regression" was developed.

Code snippet for training the ridge regression model

```
In [240]: # Linear Regression

          linearRegressionModel = LinearRegression()
          linearRegressionModel.fit(X_train, y_train)

Out[240]: LinearRegression()
```

2.5.1.2. Ridge Regression

Another Regression method that is used in machine learning. Ridge regression is a method of calculating the coefficients of multiple regression models in situations when linearly independent variables are heavily correlated. This strategy is typically used when the independent variables have a strong relationship.
We were successful in this circumstance since least-squared approaches offer unbiased values in the case of multi collinear discoveries. If the collinearity is really strong, there may be some bias.

Code snippet for training the ridge regression model

```
#train using ridg regeression
ridgeRegressionModel = Ridge()
ridgeRegressionModel.fit(X_train, y_train)

Ridge()
```

2.5.1.3. Decision Tree for Regression

In machine learning, decision tree regression may be used to conduct non-linear regression. The decision tree response algorithm's main goal is to split the sample into smaller groups. A subset of the dataset is constructed to plan the value of any data point linked with the issue statement. This algorithm creates a decision tree by dividing data into decision and sheet nodes. When there isn't enough variation in the data, ML specialists prefer this approach. Training the ridge regression model with a code snippet

Code snippet for training the ridge regression model

```
In [40]: # DecisionTree Regression

         decisionTreeRegressionModel = DecisionTreeRegressor()
         decisionTreeRegressionModel.fit(X_train, y_train)

Out[40]: DecisionTreeRegressor()
```

2.5.1.4. Multi-layer Perceptron for Regression

In artificial neural networks, the multi-layer perceptron is the most beneficial algorithm. The perceptron is a model of a single neuron that acts as a predecessor to a larger neural network.

Learner's class in science MLPRegressor activates a multi-layer perceptron (MLP) that has been trained via backpropagation without the output layer being activated or the detection function being used as a function. As a result, the output is a sequence of continuous numbers, and the cost function is a square error. The notion of

```
In [39]: # MLP Regression

         MLPRegressionModel = MLPRegressor()
         MLPRegressionModel.fit(X_train, y_train)

Out[39]: MLPRegressor()
```

regulation is used by MLPRegressor to avoid over-adjustment of the model.

3. Model Evaluating and Discussion

To get more accurate outcome, we used three model evaluation methods.

3.1. R squared score

The R-squared statistic indicates how close the data is to the fitted regression line. For multiple regression, it is also defined as the coefficient of determination. If the output value is higher, the better the model.

3.2. K-fold cross-validation

K-fold cross-validation is a popular cross-validation process. By dividing the dataset into k subsets (also known as folds) and use them k times to use it. You might be doing a 5-fold cross-validation by splitting the dataset into 5 subsets, for example. At least once, each subset must be used as the validation package. If the 5 output values are higher, the better the model.

Model / Algorithm	**R squared score**	**K-Fold Cross validation scores**
Linear Regression	0.8431620778270578	0.87609219, 0.87027479, 0.85976415, 0.86407451, 0.87370015
Ridge Regression	0.8221305459534729	0.86715737, 0.86262984, 0.85034475, 0.85293042, 0.86358758
Decision Tree for Regression	0.7172716727401374	0.70280898, 0.69441337, 0.71616371, 0.73280005, 0.65639982
Multi Player Perceptron for Regression	0.9977114221914203	0.99877787, 0.99782147, 0.99824817, 0.99819872, 0.9982088

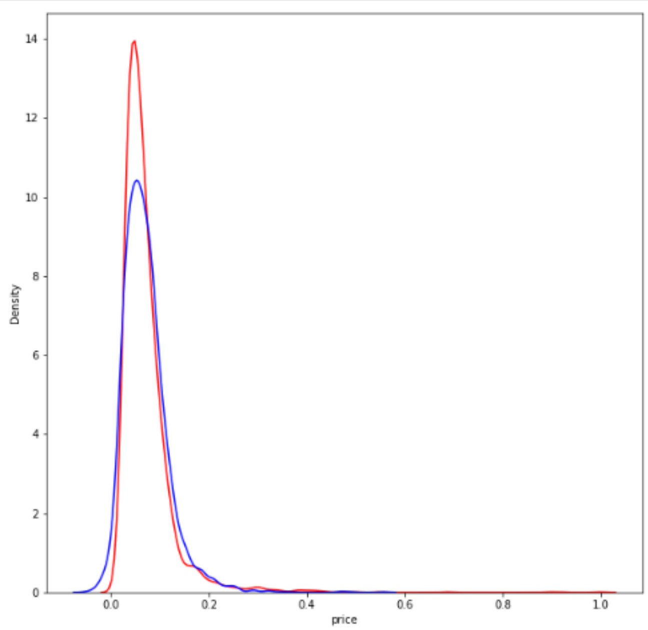
Based on the results of both R squared score, K-Fold Cross-validation approaches, we can say that MLP Regression, Linear Regression, and Ridge Regression give better prediction in this scenario.

3.3. Fitness of the models

Since above measurements says MLP, Linear and Ridge regression models are better we further tested the fitness of those three using graphs.

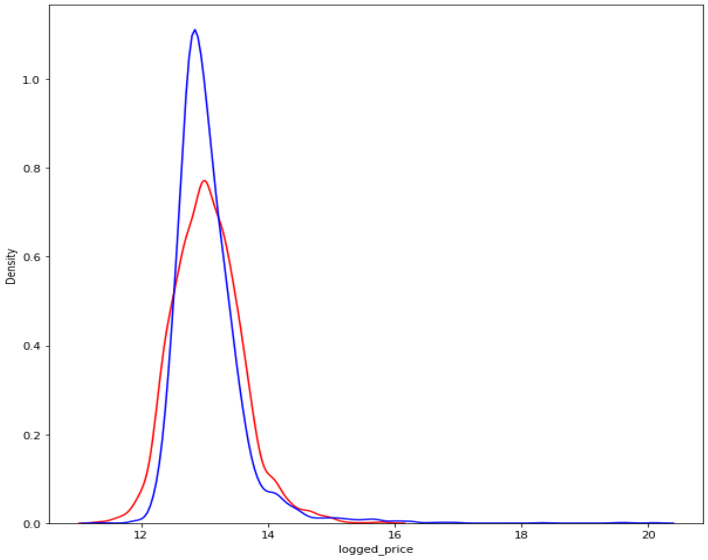
```
# Below function will draw the predicted prices graph and actual prices graph in same plot

#Red Plot - Actual Values
#Blue Plot - Predicted Values

def DistributionPlot(RedFunction, BlueFunction):
    plt.figure(figsize=(10, 10))

    ax1 = sns.distplot(RedFunction, hist=False, color="r")
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", ax=ax1)

    plt.show()
    plt.close()
```

i. Linear Regression model



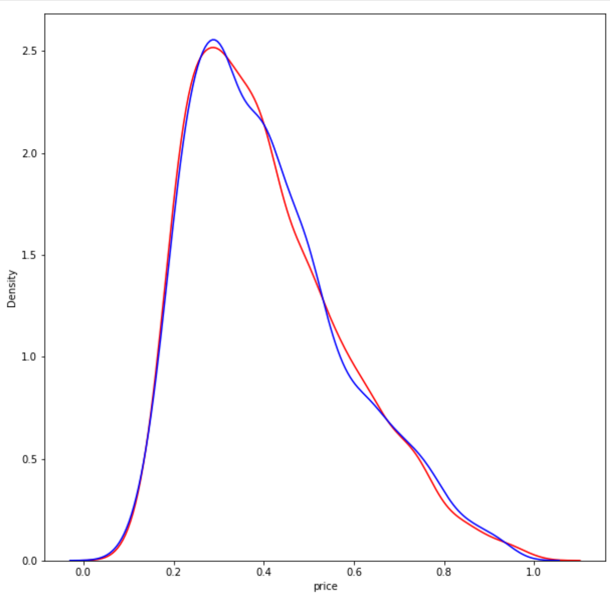
ii. Ridge Regression model

```
#predicted values of y  for X_Test values and store in 'yplot_test_ridge' var
yplot_test_ridge = ridgeRegressionModel.predict(X_test)

# Graph for values of y  for X_Test values
DistributionPlot(y_test,yplot_test_ridge)
```



iii. MLP Regression

```
In [43]: # For MLP Regression

         # get predicted y values for X_Test values and store in 'yhat_test' variable

         yhat_test_MLP = MLPRegressionModel.predict(X_test)

         # Draw the graphs
         DistributionPlot(y_test,yhat_test_MLP)
```



According to above graphs, we can say these 3 models have proper fitness with this dataset and scenario and those are capable of estimating accurate outputs.

4. References

Plotting a diagonal correlation matrix [online] https://seaborn.pydata.org/examples/many_pairwise_correlations.html [Accessed on 12.04.2021]

Correlation Matrix: Definition [online] https://www.statisticshowto.com/correlation-matrix/ [Accessed on 12.04.2021]

Categorical encoding using Label-Encoding and One-Hot-Encoder [online] https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder- 911ef77fb5bd [Accessed on 15.04.2021]

sklearn.linear_model.LinearRegression [online] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html [Accessed on 18.04.2021]

sklearn.linear_model.Ridge [online] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html [Accessed on 18.04.2021]

sklearn.linear_model.DecisionTreeRegressor [online] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html [Accessed on 18.04.2021]

sklearn.linear_model.MLPRegressor [online] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html [Accessed on 18.04.2021]

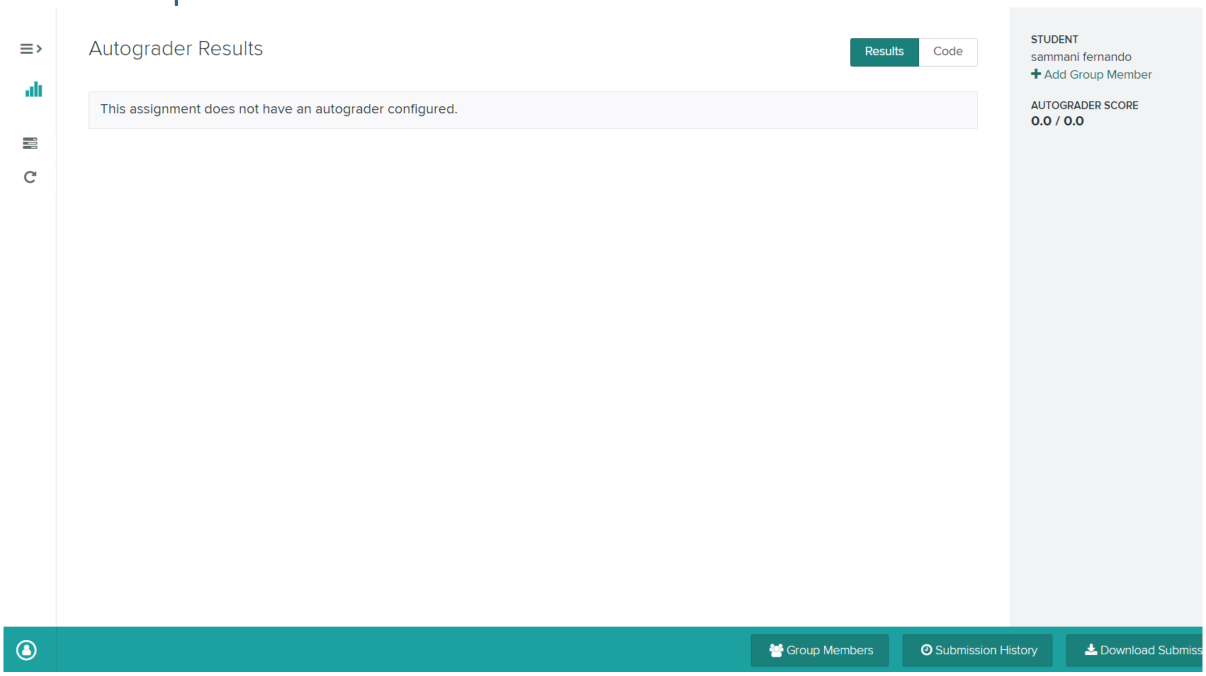
Validating your Machine Learning Model [online] https://towardsdatascience.com/validating-your- machine-learning-model-25b4c8643fb7 [Accessed on 21.04.2021]

A Gentle Introduction to k-fold Cross-Validation [online] https://machinelearningmastery.com/k-fold- cross-validation/ [Accessed on 21.04.2021]

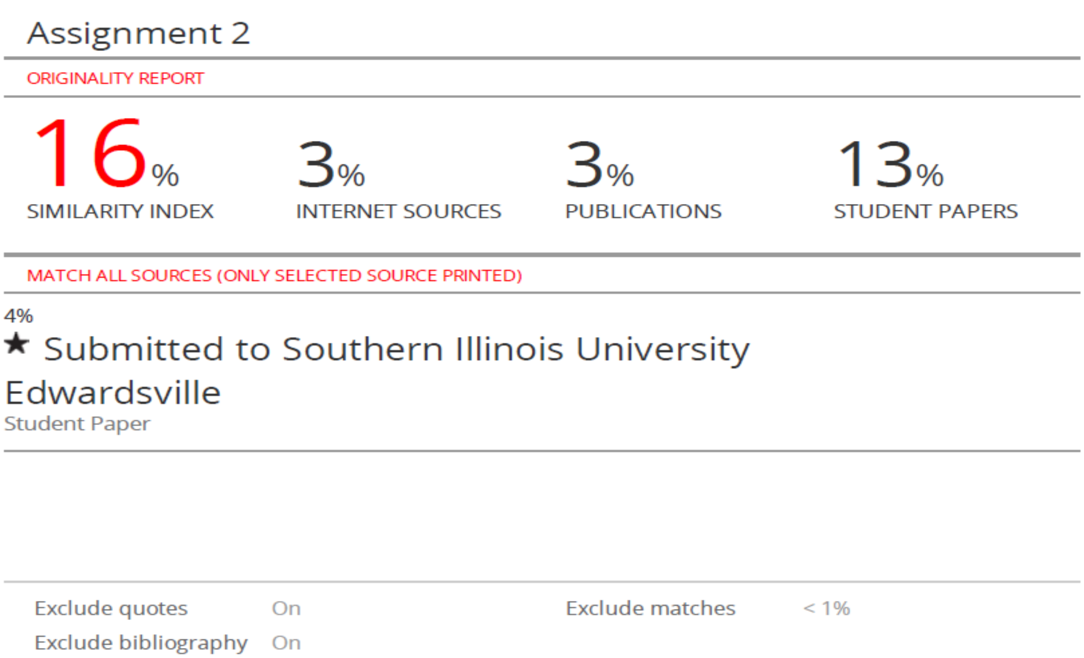
Coefficient of Determination (R Squared): Definition, Calculation [online] https://www.statisticshowto.com/probability-and-statistics/coefficient-of-determination-r-squared/ [Accessed on 21.04.2021]

5. Appendix

Gradescope Status



Turnitin Status



Assignment 2

ORIGINALITY REPORT

16%	3%	3%	13%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

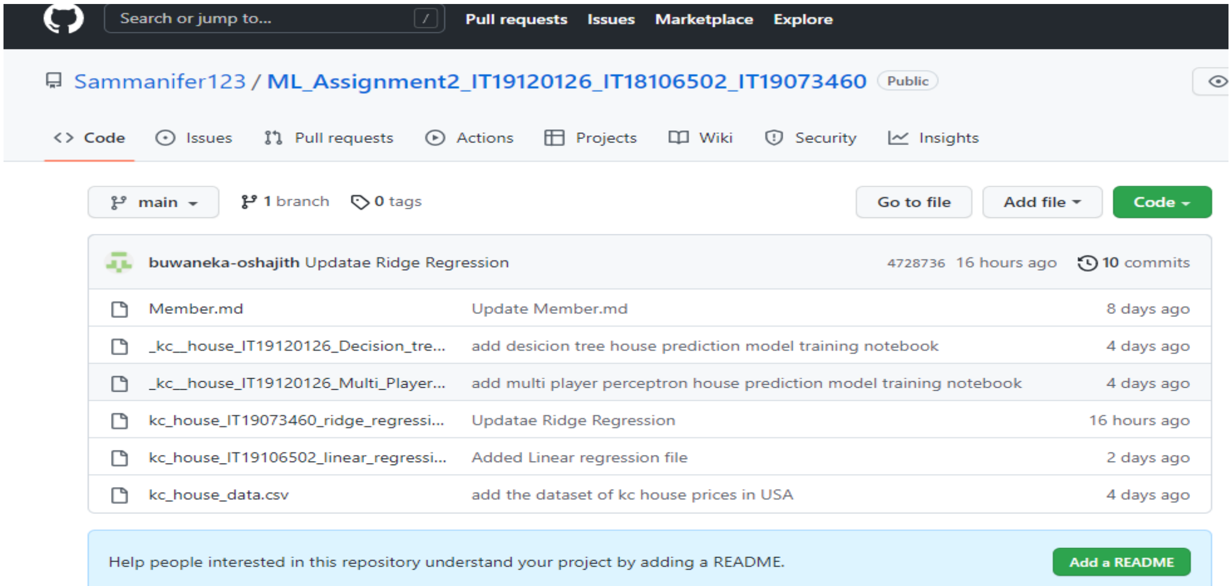
4%

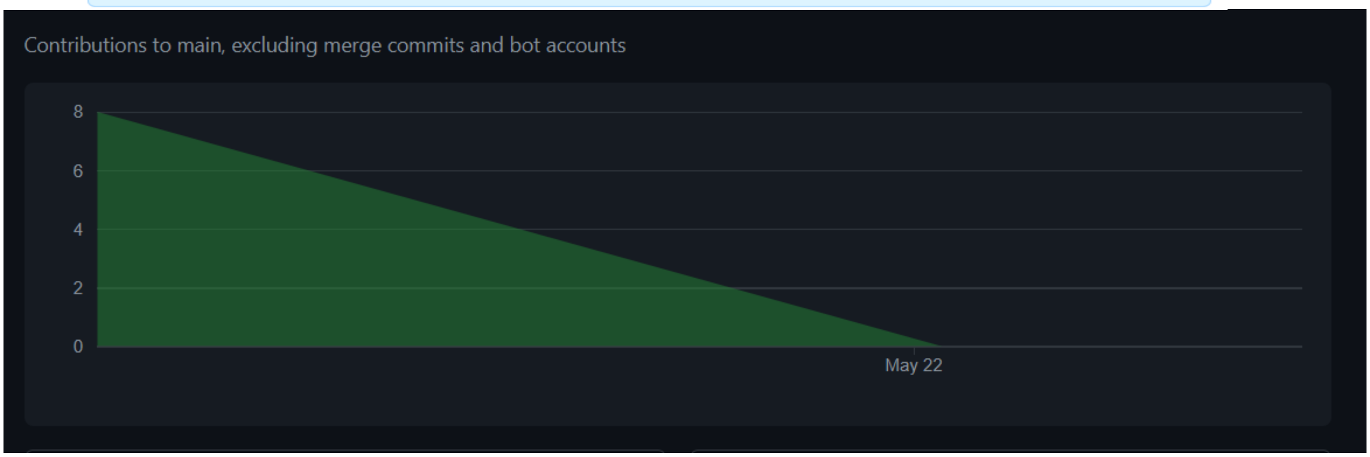
★ Submitted to Southern Illinois University Edwardsville

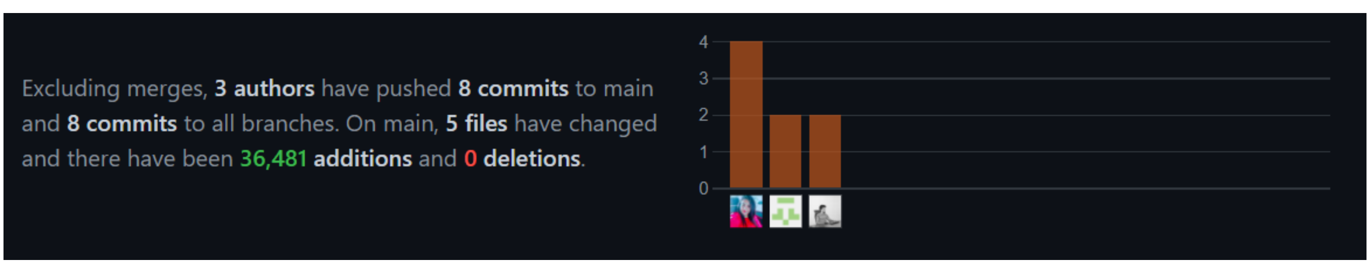
Student Paper

Exclude quotes	On	Exclude matches	< 1%
Exclude bibliography	On		

GitHub repository







Individual Contribution

1. IT19120126

Individual reference

[1]Medium. 2022. *How to Improve a Neural Network With Regularization*. [online] Available at: <https://towardsdatascience.com/how-to-improve-a-neural-network-with-regularization-8a18ecda9fe3> [Accessed 25 May 2022].

[2]Youtube.com. 2022. [online] Available at: <https://www.youtube.com/watch?v=RFSk6C9-9Wk> [Accessed 25 May 2022].

[3]Medium. 2022. *House Prices Prediction Using Deep Learning*. [online] Available at: <https://towardsdatascience.com/house-prices-prediction-using-deep-learning-dea265cc3154> [Accessed 25 May 2022].

[4]Medium. 2022. *Machine Learning Decision Tree Model Example_Melbourne House Price Prediction*. [online] Available at: <https://medium.com/@feng.cu/machine-learning-decision-tree-model-example-melbourne-house-price-prediction-83a22d16e50> [Accessed 25 May 2022].

[5]D. C. Hanselman, "Techniques for improving resolver-to-digital conversion accuracy," in IEEE Transactions on Industrial Electronics, vol. 38, no. 6, pp. 501-504, Dec. 1991, doi: 10.1109/41.107116.

Source code

Decision tree

```
# librires import
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
matplotlib.rcParams["figure.figsize"] = (12, 12)


import warnings
warnings.filterwarnings(action='ignore')

df1 = pd.read_csv('kc_house_data.csv')

# Print first five rows of the table
```

```
df1.head(5)

# display the number of rows and columns in the dataset
df1.shape

# Display the columns names in the dataset
df1.columns

# display the data types
df1.dtypes


# display unique values in the each Categorical Column
print("Column Name   Unique Values")
print("-----------   -------------")
for column in df1.columns:
    if(df1[column].dtypes != df1['floors'].dtypes and df1[column].dtypes != df1['price'].dtypes):
        print(column + ': \t' + str(df1[column].nunique()))


# display null values in cloumns
df1.isnull().sum()

df1.corr()

# Plot the Correlation Matrix

matrix = np.triu(df1.corr())
sns.heatmap(df1.corr(), annot = True, mask=matrix)

# Histogram of the each colounm
df1.hist(figsize=(20, 20))

#price Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['price'])

#bedrooms Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['bedrooms'])

#bathrooms Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['bathrooms'])

#sqft_living Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_living'])

#sqft_lot Boxplot
plt.figure(figsize=(5, 5))
```

```
sns.boxplot(x=df1['sqft_lot'])

#floors Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['floors'])

#waterfront Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['waterfront'])

#view Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['view'])

#condition Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['condition'])

#grade Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['grade'])

#sqft_above Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_above'])

#sqft_basement Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_basement'])

#yr_built Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['yr_built'])

#yr_renovated Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['yr_renovated'])

#zipcode Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['zipcode'])

#lat Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['lat'])

#long Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['long'])

#sqft_living15 Boxplot
```

```
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_living15'])

#sqft_lot15 Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_lot15'])

df1.describe()

df2 = df1.drop([ 'zipcode', 'date', 'waterfront', 'view', 'yr_renovated'],
            axis='columns')
df2.head(3)

# Fill missing values of 'floors' and 'yr_built' columns
df2['yr_built'] = df2['yr_built'].fillna(df2['yr_built'].median())
df2['floors'] = df2['floors'].fillna(df2['floors'].median())

df2.head(3)


# See again how many null values in each column. But now we deal with 'df2'

df2.isnull().sum()


# Remove outliers of all columns and assign to new data freame

cols = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'condition', 'grade', 'sqft_above',
      'sqft_basement','yr_built']
Q1 = df1[cols].quantile(0.25)
Q3 = df1[cols].quantile(0.75)
IQR = Q3 - Q1

df3 = df2[~((df1[cols] < (Q1 - 1.5 * IQR)) | (df1[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]


# See the houses there are floors more than 3, because it is uncommon and need to be removed from
further processing
df4 = df3.copy()
df4 = df3[df3.floors < 3]

#See the houses there are more bathrooms than bedrooms count+2,because it is uncommon and need
to be removed from further processing

df4 = df3[df3.bathrooms < df3.bathrooms + 2]


#Convert bathrooms,floors and price into integers.
df5 = df4.copy()
```

```
df5['bathrooms'] = df5['bathrooms'].astype('int64')
df5['floors'] = df5['floors'].astype('int64')
df5['price'] = df5['price'].astype('int64')

df6 = df5.copy()

df6['price'] = df5['price'] / df5['price'].max()
df6['bedrooms'] = df5['bedrooms'] / df5['bedrooms'].max()
df6['bathrooms'] = df5['bathrooms'] / df5['bathrooms'].max()
df6['sqft_living'] = df5['sqft_living'] / df5['sqft_living'].max()
df6['sqft_lot'] = df5['sqft_lot'] / df2['sqft_lot'].max()
df6['floors'] = df2['floors'] / df5['floors'].max()
df6['condition'] = df5['condition'] / df5['condition'].max()
df6['grade'] = df5['grade'] / df5['grade'].max()
df6['sqft_above'] = df5['sqft_above'] / df5['sqft_above'].max()
df6['sqft_basement'] = df5['sqft_basement'] / df5['sqft_basement'].max()
df6['yr_built'] = df5['yr_built'] / df5['yr_built'].max()
df6['lat'] = df5['lat'] / df5['lat'].max()
df6['long'] = df5['long'] / df5['long'].max()
df6['sqft_living15'] = df5['sqft_living15'] / df5['sqft_living15'].max()
df6['sqft_lot15'] = df5['sqft_lot15'] / df5['sqft_lot15'].max()


df6

y = df6['price']
X = df6.drop('price', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)


MLPRegressionModel = MLPRegressor()
MLPRegressionModel.fit(X_train, y_train)


df6.hist(column='price')

df6['logged_prcie'] =np.log(df6.price)
df6.head(5)

df6.hist(column='logged_prcie')

sns.displot(df6['sqft_living'])

df6['logged_sqft_living'] =np.log(df6.sqft_living)
df6.head(5)

sns.displot(df6['logged_sqft_living'])


y = df6['price']
```

```
X = df6.drop('price', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)

# DecisionTree Regression

decisionTreeRegressionModel = DecisionTreeRegressor()
decisionTreeRegressionModel.fit(X_train, y_train)

# For decisionTree Regression

decisionTreeRegressionModel.score(X_test, y_test)

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(DecisionTreeRegressor(), X, y, cv=cv)


def DistributionPlot(RedFunction, BlueFunction):
    plt.figure(figsize=(10, 10))

    ax1 = sns.distplot(RedFunction, hist=False, color="r")
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", ax=ax1)

    plt.show()
    plt.close()


# get predicted y values for X_Test values and store in 'yhat_test' variable

yhat_test_DT = decisionTreeRegressionModel.predict(X_test)

#Graph
DistributionPlot(y_test,yhat_test_DT)
```

Multi player Perceptron

```
# librires import
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
matplotlib.rcParams["figure.figsize"] = (12, 12)
```

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

df1 = pd.read_csv('kc_house_data.csv')

# Print first five rows of the table
df1.head(5)

# display the number of rows and columns in the dataset
df1.shape

# Display the columns names in the dataset
df1.columns

# display the data types
df1.dtypes


# display unique values in the each Categorical Column
print("Column Name   Unique Values")
print("-----------   -------------")
for column in df1.columns:
   if(df1[column].dtypes != df1['floors'].dtypes and df1[column].dtypes != df1['price'].dtypes):
      print(column + ': \t' + str(df1[column].nunique()))


# display null values in cloumns
df1.isnull().sum()

df1.corr()

# Plot the Correlation Matrix

matrix = np.triu(df1.corr())
sns.heatmap(df1.corr(), annot = True, mask=matrix)

# Histogram of the each colounm
df1.hist(figsize=(20, 20))

#price Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['price'])

#bedrooms Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['bedrooms'])

#bathrooms Boxplot
plt.figure(figsize=(5, 5))
```

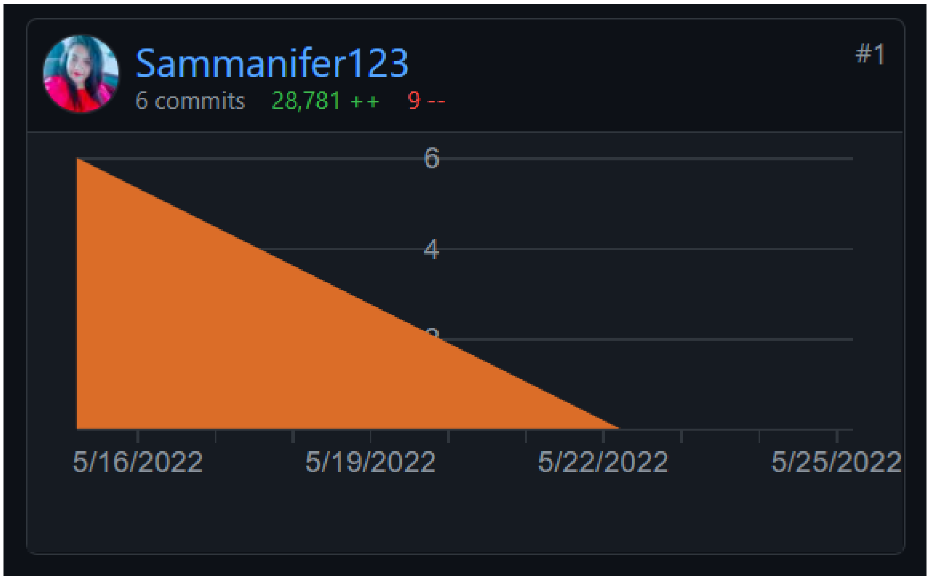
```
sns.boxplot(x=df1['bathrooms'])

#sqft_living Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_living'])

#sqft_lot Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_lot'])

#floors Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['floors'])

#waterfront Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['waterfront'])

#view Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['view'])

#condition Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['condition'])

#grade Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['grade'])

#sqft_above Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_above'])

#sqft_basement Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_basement'])

#yr_built Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['yr_built'])

#yr_renovated Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['yr_renovated'])

#zipcode Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['zipcode'])

#lat Boxplot
```

```
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['lat'])

#long Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['long'])

#sqft_living15 Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_living15'])

#sqft_lot15 Boxplot
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_lot15'])

df1.describe()

df2 = df1.drop([ 'zipcode', 'date', 'waterfront', 'view', 'yr_renovated'],
          axis='columns')
df2.head(3)

# Fill missing values of 'floors' and 'yr_built' columns
df2['yr_built'] = df2['yr_built'].fillna(df2['yr_built'].median())
df2['floors'] = df2['floors'].fillna(df2['floors'].median())

df2.head(3)


# See again how many null values in each column. But now we deal with 'df2'

df2.isnull().sum()


# Remove outliers of all columns and assign to new data freame

cols = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'condition', 'grade', 'sqft_above',
    'sqft_basement','yr_built']
Q1 = df1[cols].quantile(0.25)
Q3 = df1[cols].quantile(0.75)
IQR = Q3 - Q1

df3 = df2[~((df1[cols] < (Q1 - 1.5 * IQR)) | (df1[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]


# See the houses there are floors more than 3, because it is uncommon and need to be removed from
further processing
df4 = df3.copy()
df4 = df3[df3.floors < 3]

#See the houses there are more bathrooms than bedrooms count+2,because it is uncommon and need
to be removed from further processing
```

```
df4 = df3[df3.bathrooms < df3.bathrooms + 2]


#Convert bathrooms,floors and price into integers.
df5 = df4.copy()


df5['bathrooms'] = df5['bathrooms'].astype('int64')
df5['floors'] = df5['floors'].astype('int64')
df5['price'] = df5['price'].astype('int64')

df6 = df5.copy()

df6['price'] = df5['price'] / df5['price'].max()
df6['bedrooms'] = df5['bedrooms'] / df5['bedrooms'].max()
df6['bathrooms'] = df5['bathrooms'] / df5['bathrooms'].max()
df6['sqft_living'] = df5['sqft_living'] / df5['sqft_living'].max()
df6['sqft_lot'] = df5['sqft_lot'] / df2['sqft_lot'].max()
df6['floors'] = df2['floors'] / df5['floors'].max()
df6['condition'] = df5['condition'] / df5['condition'].max()
df6['grade'] = df5['grade'] / df5['grade'].max()
df6['sqft_above'] = df5['sqft_above'] / df5['sqft_above'].max()
df6['sqft_basement'] = df5['sqft_basement'] / df5['sqft_basement'].max()
df6['yr_built'] = df5['yr_built'] / df5['yr_built'].max()
df6['lat'] = df5['lat'] / df5['lat'].max()
df6['long'] = df5['long'] / df5['long'].max()
df6['sqft_living15'] = df5['sqft_living15'] / df5['sqft_living15'].max()
df6['sqft_lot15'] = df5['sqft_lot15'] / df5['sqft_lot15'].max()


df6

y = df6['price']
X = df6.drop('price', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)


MLPRegressionModel = MLPRegressor()
MLPRegressionModel.fit(X_train, y_train)


df6.hist(column='price')

df6['logged_prcie'] =np.log(df6.price)
df6.head(5)

df6.hist(column='logged_prcie')

sns.displot(df6['sqft_living'])
```

```
df6['logged_sqft_living'] =np.log(df6.sqft_living)
df6.head(5)

sns.displot(df6['logged_sqft_living'])


y = df6['price']
X = df6.drop('price', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)

# MLP Regression

MLPRegressionModel = MLPRegressor()
MLPRegressionModel.fit(X_train, y_train)

# For MLP Regression

MLPRegressionModel.score(X_test, y_test)

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(MLPRegressor(), X, y, cv=cv)

def DistributionPlot(RedFunction, BlueFunction):
   plt.figure(figsize=(10, 10))

   ax1 = sns.distplot(RedFunction, hist=False, color="r")
   ax2 = sns.distplot(BlueFunction, hist=False, color="b", ax=ax1)

   plt.show()
   plt.close()

# For MLP Regression

# get predicted y values for X_Test values and store in 'yhat_test' variable

yhat_test_MLP = MLPRegressionModel.predict(X_test)

# Draw the graphs
DistributionPlot(y_test,yhat_test_MLP)
```

GitHub status (Individual commits)



2. IT19073460

Individual reference

[1]C. R. Madhuri, G. Anuradha and M. V. Pujitha, "House Price Prediction Using Regression Techniques: A Comparative Study," *2019 International Conference on Smart Structures and Systems (ICSSS)*, 2019, pp. 1-5, doi: 10.1109/ICSSS.2019.8882834.

[2], Josh. “Regularization Part 1: Ridge (L2) Regression"}],“Accessibility”:{“AccessibilityData”:{“Label”:“Regularization Part 1: Ridge (L2) Regression de StatQuest with Josh Starmer Il Y a 3 Ans 20 Minutes 664 821 Vues”}}},“LongBylineText”:{“Runs”:[{“Text”:"StatQuest with Josh Starmer.”

[3]*YouTube*, 2022, www.youtube.com/results?search_query=ridge+regression. Accessed 25 May 2022.

Source code

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression, Ridge


from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

matplotlib.rcParams["figure.figsize"] = (12, 12)
import warnings
warnings.filterwarnings(action='ignore')


df = pd.read_csv('kc_house_data.csv')
```

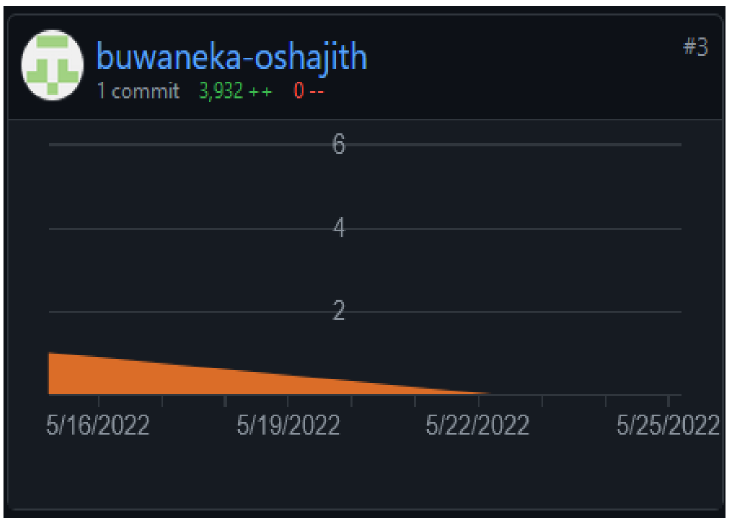
```
# Print first five rows of the table
df.head(5)

#Display number of rows and number of columns
df.shape
#display column namnes
df.dtypes
#Check for unique values
print("Column Name   Unique Values")
print("-----------   -------------")
for column in df.columns:
    if(df[column].dtypes != df['floors'].dtypes and df[column].dtypes != df['price'].dtypes):
        print(column + ': \t' + str(df[column].nunique()))
#Dispaly null values in each column
df.isnull().sum()
# Display the Correlation Matrix

df.corr()
#Dispaly statics of each column
df.describe()
#Drop unwanted features that do not required to build the model
#Name the new data frame as df1
#Name the new data frame as df1
df1 = df.drop(['id', 'zipcode', 'date', 'waterfront', 'view', 'yr_renovated'],
              axis='columns')
# Print first five rows of the new data frame
df1.head(5)

#Detect and remove outliers
cols = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'condition', 'lat', 'long', 'grade',
'sqft_above', 'sqft_basement',
        'yr_built']
Q1 = df1[cols].quantile(0.25)
Q3 = df1[cols].quantile(0.75)
IQR = Q3 - Q1
df2 = df1[~((df1[cols] < (Q1 - 1.5 * IQR)) | (df1[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]

#Convert floors,long,latitude, bathrooms and price into integers.
df2['floors'] = df2['floors'].astype('int64')
df2['bathrooms'] = df2['bathrooms'].astype('int64')
df2['price'] = df2['price'].astype('int64')
df2['lat'] = df2['lat'].astype('int64')
df2['long'] = df2['long'].astype('int64')
#Turn all numerical values of a range between 0 and 1

#by deviding values from Maximum of the column
df4 = df2.copy()

df4['price'] = df2['price'] / df2['price'].max()
```

```
df4['bedrooms'] = df2['bedrooms'] / df2['bedrooms'].max()
df4['bathrooms'] = df2['bathrooms'] / df2['bathrooms'].max()
df4['sqft_living'] = df2['sqft_living'] / df2['sqft_living'].max()
df4['sqft_lot'] = df2['sqft_lot'] / df2['sqft_lot'].max()
df4['floors'] = df2['floors'] / df2['floors'].max()
df4['condition'] = df2['condition'] / df2['condition'].max()
df4['grade'] = df2['grade'] / df2['grade'].max()
df4['sqft_above'] = df2['sqft_above'] / df2['sqft_above'].max()
df4['sqft_basement'] = df2['sqft_basement'] / df2['sqft_basement'].max()
df4['yr_built'] = df2['yr_built'] / df2['yr_built'].max()
#df4['lat'] = df2['lat'] / df2['lat'].max()
#df4['long'] = df2['long'] / df2['long'].max()
df4['sqft_living15'] = df2['sqft_living15'] / df2['sqft_living15'].max()
df4['sqft_lot15'] = df2['sqft_lot15'] / df2['sqft_lot15'].max()

df4

#split and Train data
y = df4['price']
X = df4.drop('price', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)
#ridge regression
ridgeRegressionModel = Ridge()
ridgeRegressionModel.fit(X_train, y_train)
#R-squared value
ridgeRegressionModel.score(X_test, y_test)

#k-fold values
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(Ridge(), X, y, cv=cv)

def DistributionPlot(RedFunction, BlueFunction):
   plt.figure(figsize=(10, 10))

   ax1 = sns.distplot(RedFunction, hist=False, color="r")
   ax2 = sns.distplot(BlueFunction, hist=False, color="b", ax=ax1)

   plt.show()
   plt.close()

#predicted values of y  for X_Test values and store in 'yplot_test_ridge' variable

yplot_test_ridge = ridgeRegressionModel.predict(X_test)

# Graph for values of y  for X_Test values
DistributionPlot(y_test,yplot_test_ridge)

#log Transformation
dflog = df.copy()
dflog2 = dflog.drop(['id', 'zipcode', 'date', 'waterfront', 'view', 'yr_renovated'],
```

```
            axis='columns')
dflog2['floors'] = dflog2['floors'].astype('int64')
dflog2['bathrooms'] = dflog2['bathrooms'].astype('int64')
dflog2['price'] = dflog2['price'].astype('int64')
dflog2['lat'] = dflog2['lat'].astype('int64')
dflog2['long'] = dflog2['long'].astype('int64')

# Price Histrogarm before Logged price
dflog2.hist(column='price')
#Turn price to logged price
dflog2['logged_price'] =np.log(dflog2.price)
dflog2.head(5)
# Price Histrogarm After  Logged price
dflog2.hist(column='logged_price')
#sqft_living Histrogarm before  Logged sqft_living
dflog2.hist(column='sqft_living')
sns.displot(df2['sqft_living'])
#Turn sqft_living to logged state
dflog2['logged_sqft_living'] =np.log(dflog2.sqft_living)
dflog2.head(5)
#sqft_living Histrogarm After  Logged sqft_living
dflog2.hist(column='logged_sqft_living')
sns.displot(dflog2['logged_sqft_living'])

# Applying ridge regression after log Transformation
dflog3 = dflog2.drop(['sqft_living'],
            axis='columns')
y = dflog3['logged_price']
X = dflog3.drop('logged_price', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)
#train using ridg regeression
ridgeRegressionModel = Ridge()
ridgeRegressionModel.fit(X_train, y_train)
#Test Ridge Regression Model R squared score
ridgeRegressionModel.score(X_test, y_test)
#Test the model
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(Ridge(), X, y, cv=cv)
#plot
def DistributionPlot(RedFunction, BlueFunction):
   plt.figure(figsize=(10, 10))

   ax1 = sns.distplot(RedFunction, hist=False, color="r")
   ax2 = sns.distplot(BlueFunction, hist=False, color="b", ax=ax1)

   plt.show()
   plt.close()

#predicted values of y  for X_Test values and store in 'yplot_test_ridge' variable
```

```
yplot_test_ridge = ridgeRegressionModel.predict(X_test)

# Graph for values of y  for X_Test values
DistributionPlot(y_test,yplot_test_ridge)
```

GitHub status (Individual commits)



3. IT19106502

Individual reference

[1] youtube, " Linear Regression Machine Learning," 05 03 2020. [Online]. [Accessed 2022].

[2] R. S. a. A. M1, "Linear Regression Algorithm Based Price Prediction of Car and Accuracy Comparison with Support Vecto
https://iopscience.iop.org/article/10.1149/10701.12953ecst/pdf. [Accessed 2022].

[3] "Statology," [Online]. Available: https://www.statology.org/predictions-regression/. [Accessed 2022].

[4] V. Valkov, "Predicting House Prices with Linear Regression," 02 04 2019. [Online]. [Accessed 2022].

Source code

```
# ESTIMATIMG KC-HOUSE PRICE USING LINEAR REGRESSION MODEL

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor

from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

matplotlib.rcParams["figure.figsize"] = (12, 12)
import warnings
warnings.filterwarnings(action='ignore')
df1 = pd.read_csv('kc_house_data.csv')
# Print first five rows of the table
df1.head(5)
# see number of rows, number of columns
df1.shape
# see columns names

df1.columns
# See data types of the Columns
df1.dtypes
# See how many unique values in the each 'Categorical Column' (Columns that have values other than
numbers)
print("Column Name   Unique Values")
print("-----------   -------------")
for column in df1.columns:
    if(df1[column].dtypes != df1['price'].dtypes and df1[column].dtypes != df1['bathrooms'].dtypes):
        print(column + ': \t' + str(df1[column].nunique()))
```

```
# See how many null values in each column
df1.isnull().sum()
# See the Correlation Matrix (Linear Correlation)
df1.corr()
# Plot the Correlation Matrix (Linear Correlation)
matrix = np.triu(df1.corr()) # in order to produce only the lower part of the matrix
sns.heatmap(df1.corr(), annot = True, mask=matrix)
# Histogram per each numerical column
df1.hist(figsize=(15, 15))
# Boxplot for each column
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['price'])
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['bedrooms'])
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['bathrooms'])
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_living'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_lot'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['floors'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['waterfront'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['view'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['condition'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['grade'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_above'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_basement'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['yr_built'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['yr_renovated'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['zipcode'])
```

```
plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['lat'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['long'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_living15'])

plt.figure(figsize=(5, 5))
sns.boxplot(x=df1['sqft_lot15'])

# Get the statistics per each columnGet the statistics per each column

df1.describe()

# Drop features that are not required to build our model and create new data frame 'df2'
df2 = df1.copy()
df2 = df2.drop([ 'date', 'zipcode', 'view', 'waterfront', 'yr_renovated'], axis='columns')
df2.head(6)

# See again how many null values in each column. But now we deal with 'df2'
df2.isnull().sum()

df2.shape

# Remove outliers of all columns and assign to new data freame called 'df3'
df3=df2.copy()
cols = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'lat', 'long', 'floors', 'condition', 'grade',
'sqft_above', 'sqft_basement', 'yr_built', 'sqft_living15', 'sqft_lot15']

Q1 = df1[cols].quantile(0.25)
Q3 = df1[cols].quantile(0.75)
IQR = Q3 - Q1

df3 = df1[~((df1[cols] < (Q1 - 1.5 * IQR)) | (df1[cols] > (Q3 + 1.5 * IQR))).any(axis=1)]
# See the houses there are bathrooms more than 3, because it is uncommon and need to be removed
from further processing

df4 = df3.copy()
df4 = df4[df4.bathrooms < 3]
# See the houses there are more bathrooms than bedrooms count + 2, because it is uncommon and ned
to be removed from further processing
df4 = df4[df4.bathrooms < df4.bedrooms + 2]
df2.shape
#Convert bathrooms and floors into integers.
df5 = df2.copy()
df5['price'] = df5['price'].astype('int64')
df5['bathrooms'] = df5['bathrooms'].astype('int64')
df5['floors'] = df5['floors'].astype('int64')
```

```
# In this dataset all the other distances are in Integer datatype.
# Therefore, it is better to convert 'NEAREST_SCH_DIST' column in to 'Meters' as per the consistency.
#df5['NEAREST_SCH_DIST'] = df5['NEAREST_SCH_DIST'].astype('int64')
df5.dtypes


# Get all numerical values to a range between 0 and 1
# divide all values from its column's maximum number
df6 = df5.copy()
df6['price'] = df5['price'] / df5['price'].max()
df6['bedrooms          '] = df5['bedrooms'] / df5['bedrooms'].max()
df6['bathrooms         '] = df5['bathrooms'] / df5['bathrooms'].max()
df6['sqft_living       '] = df5['sqft_living'] / df5['sqft_living'].max()
df6['sqft_lot          '] = df5['sqft_lot'] / df5['sqft_lot'].max()
df6['lat               '] = df5['lat'] / df5['lat'].max()
df6['long          '] = df5['long'] / df5['long'].max()
df6['floors            '] = df5['floors'] / df5['floors'].max()
df6['condition         '] = df5['condition'] / df5['condition'].max()
df6['grade                     '] = df5['grade'] / df5['grade'].max()
df6['sqft_above                '] = df5['sqft_above'] / df5['sqft_above'].max()
df6['sqft_basement             '] = df5['sqft_basement'] / df5['sqft_basement'].max()
df6['yr_built              '] = df5['yr_built'] / df5['yr_built'].max()
df6
# Split df into X and y
df9=df6.copy()
y = df9['price']
X = df9.drop('price', axis=1)
# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)
# Linear Regression
linearRegressionModel = LinearRegression()
linearRegressionModel.fit(X_train, y_train)
# For Linear Regression
linearRegressionModel.score(X_test, y_test)
# For Linear Regression
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
cross_val_score(LinearRegression(), X, y, cv=cv)
# For Linear Regression
# get predicted y values for X_Test values and store in 'yhat_test' variable
yhat_test_linear = linearRegressionModel.predict(X_test)
# Draw the graphs
DistributionPlot(y_test,yhat_test_linear)
# For Linear Regression

# get predicted y values for X_Test values and store in 'yhat_test' variable
yhat_test_linear = linearRegressionModel.predict(X_test)
# Draw the graphs
DistributionPlot(y_test,yhat_test_linear)
# Below function will draw the predicted prices graph and actual prices graph in same plot
#Red Plot - Actual Values
#Blue Plot - Predicted Values
```

```
def DistributionPlot(RedFunction, BlueFunction):
    plt.figure(figsize=(10, 10))

    ax1 = sns.distplot(RedFunction, hist=False, color="r")
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", ax=ax1)

    plt.show()
    plt.close()
df9.hist(column='price')

df9['logged_prcie'] =np.log(df9.price)
df9.head(5)
df9.hist(column='logged_prcie')
df9['logged_sqft_living'] =np.log(df9.sqft_living)
df9.head(5)
sns.displot(df9['logged_sqft_living'])
df9.head(5)
y = df9['price']
X = df9.drop('price', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, shuffle=True, random_state=1)
# Linear Regression

linearRegressionModel = LinearRegression()
linearRegressionModel.fit(X_train, y_train)
linearRegressionModel.score(X_test, y_test)
# For Linear Regression

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(LinearRegression(), X, y, cv=cv)
# For Linear Regression

linearRegressionModel.score(X_test, y_test)
# For Linear Regression

# get predicted y values for X_Test values and store in 'yhat_test' variable

yhat_test_linear = linearRegressionModel.predict(X_test)
# Draw the graphs
DistributionPlot(y_test,yhat_test_linear)
```

GitHub status (Individual commits)