Assignment 2: Estimate the Ultimate Price of Each Residential Property

Samir Khanal (C0869664)

Computer Department, Lambton College

AML 1413 – Introduction to Artificial Intelligence

Mohammad Islam

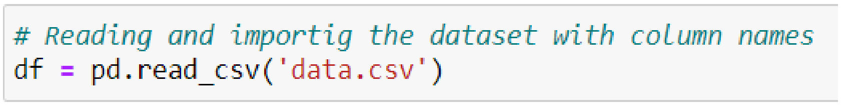
December 2022

Problem Statement
We have to estimate the ultimate price of each residential property (SalePrice in the dataset) in Toronto, given the 79 explanatory factors that describe (nearly) every feature of residential properties there.

Tools Used:
Python was used as the base language. Other Different tools that were used in this project are
1. Pandas
2. Matplotlib
3. NumPy
4. Jupyter Notebook
5. Scikit Learn
6. Seaborn

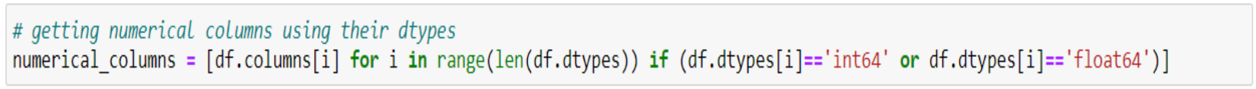
Process Steps:

A. Import Dataset
The given dataset 'data.csv' is read using the panda's method read_csv.



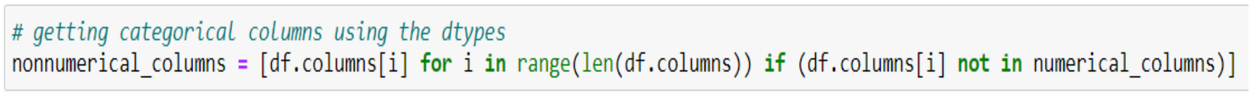
```
# Reading and importig the dataset with column names
df = pd.read_csv('data.csv')
```

B. Splitting of the dataset according to categorical and numerical features
As per the instruction, the numerical columns and categorical columns are separated on the basis of their datatypes.
The numerical columns are separated using 'int64' and 'float64' datatypes as present in their columns.



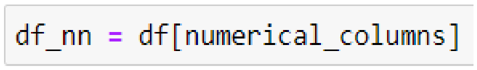
```
# getting numerical columns using their dtypes
numerical_columns = [df.columns[i] for i in range(len(df.dtypes)) if (df.dtypes[i]=='int64' or df.dtypes[i]=='float64')]
```

The categorical columns are the columns except for the numerical columns in the dataset.



```
# getting categorical columns using the dtypes
nonnumerical_columns = [df.columns[i] for i in range(len(df.columns)) if (df.columns[i] not in numerical_columns)]
```

C. Using numerical attributes only
Taking only numerical columns from the dataset into df_nn using the numerical_columns from before.



```
df_nn = df[numerical_columns]
```

1. Splitting into training and testing datasets

For splitting the dataset into training and testing datasets, the dataset needs to be shuffled first.

```
# Shuffling of data
df_nn = df_nn.sample(frac=1)
df_nn
```

Then the shuffled dataset is split into train data and test data in a 70 to 30 ratio.

```
# splitting of training and testing data
# 70% => training data and 30% => testing data
separating_index = int(len(df_nn)*0.7)
train_data_nn = df_nn[:separating_index]
test_data_nn = df_nn[separating_index:]
```

2. For Train data

i. Data Preprocessing

The separated train data might contain missing values, different unique symbols, and duplicate values. The null values or the missing values are imputed(replaced) with the mean values of each feature column. The duplicate values are deleted.

```
# checking for null or NaN values
train_data_nn.isnull().sum()
```

```
# imputation with mean values of each feature column
for f in train_data_nn.columns[1:]:
    # getting the mean value of the feature column
    replace_mean = train_data_nn[f].mean()
    # replacing the null values with mean value
    train_data_nn[f] = train_data_nn[f].fillna(replace_mean)
```

```
# Checking and deleting duplicate values
if train_data_nn.duplicated().sum() > 0:
    # getting the indexes of the duplicate values
    droplist=train_data_nn.loc[train_data_nn.duplicated()==True].index.tolist()
    # Deleting the duplicate values
    train_data_nn.drop(droplist,axis=0,inplace=True)
```

ii. Feature Selection and Feature Pruning

The feature selection and pruning are done using the correlation values. At first, the correlation between the features is calculated. Then, a 70% correlation is used as a threshold value to filter out the best features related to the target or dependent variable.

```
# calculating correlation between the features
cormat = train_data_nn.corr()
cormat = round(cormat, 2)
```

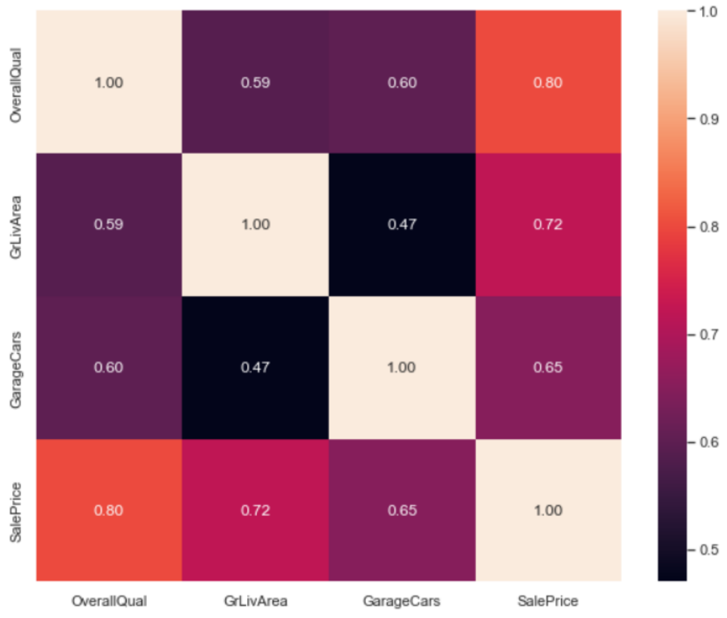
```
# taking 70% correlation as our threshold and selecting the features
features = []

for index, ec in enumerate(cormat.SalePrice):
    if ec >= 0.65:
        features.append(train_data_nn.columns[index])
```

```
features
```

```
['OverallQual', 'GrLivArea', 'GarageCars', 'SalePrice']
```

Then, the correlation of the filtered features is calculated again and visualized through the HeatMap.



iii. Data separation into dependent and independent variables

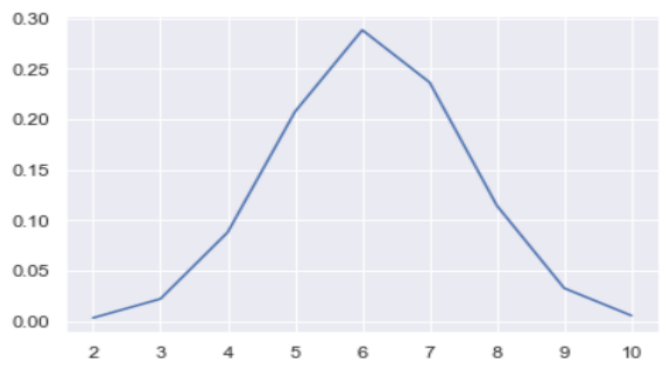
The pre-processed train data is separated into the target or dependent variable and independent variable using a simple python list-slicing method.

```
# separating the dependent and independent variable as target and dataset
y_train_nn = train_data_nn['SalePrice']
X_train_nn = train_data_nn[features[: -1]]
```

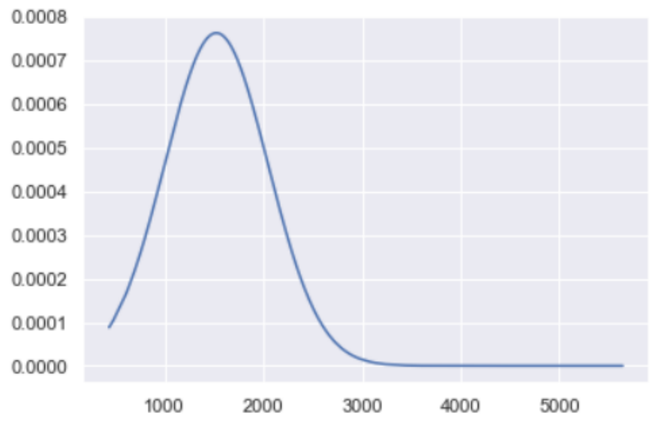
iv. The measure of Central Tendency For Each Feature

The visualization of the Central Tendency for each feature is done by using the Normal distribution method. It is also used to determine the outliers, but the outliers are not removed in this process for more generalization.

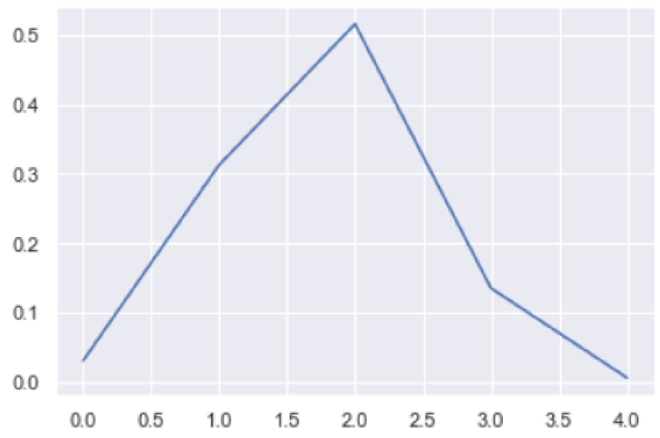
```
For Feature  OverallQual
```



```
For Feature  GrLivArea
```



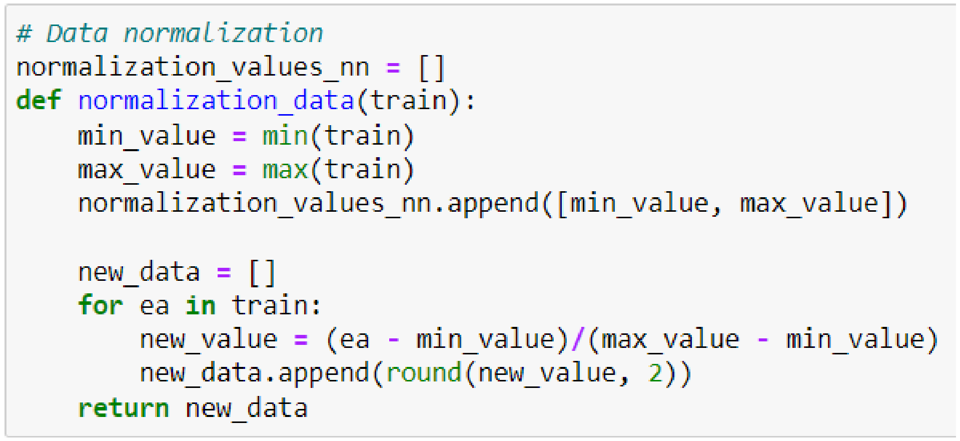
```
For Feature  GarageCars
```



v. Data Normalization

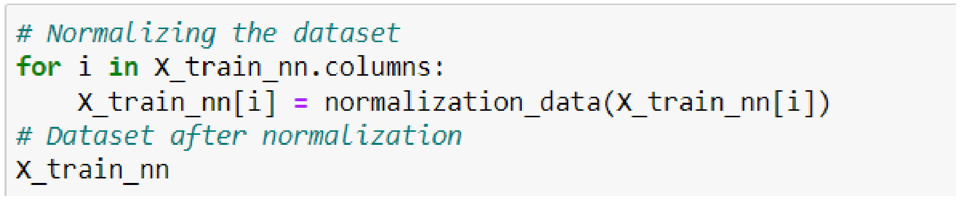
To normalize the data, the scaling data normalization method was applied to the feature columns of the dataset using the formula in the given equation.

$$\text{Normalized value(x)} = \frac{x - minimum\ value}{maximum\ value - minimum\ value}$$



```
# Data normalization
normalization_values_nn = []
def normalization_data(train):
    min_value = min(train)
    max_value = max(train)
    normalization_values_nn.append([min_value, max_value])

    new_data = []
    for ea in train:
        new_value = (ea - min_value)/(max_value - min_value)
        new_data.append(round(new_value, 2))
    return new_data
```



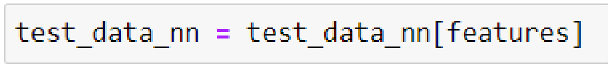
```
# Normalizing the dataset
for i in X_train_nn.columns:
    X_train_nn[i] = normalization_data(X_train_nn[i])
# Dataset after normalization
X_train_nn
```

3. For Test data

4.

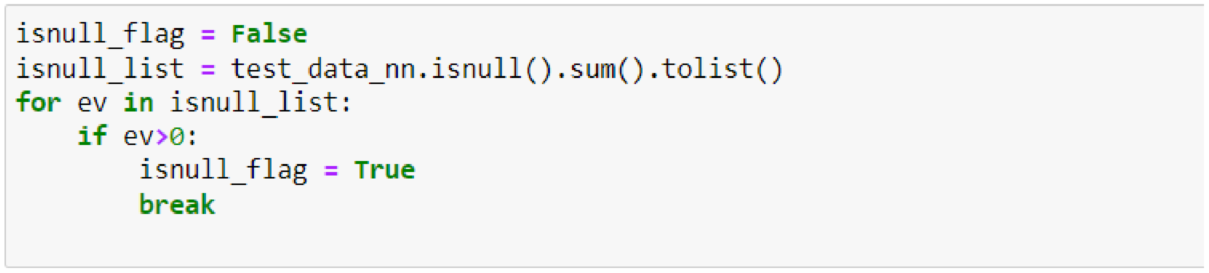
i. Feature Selection and Data Preprocessing

Already selected features in the training set were selected for test data.

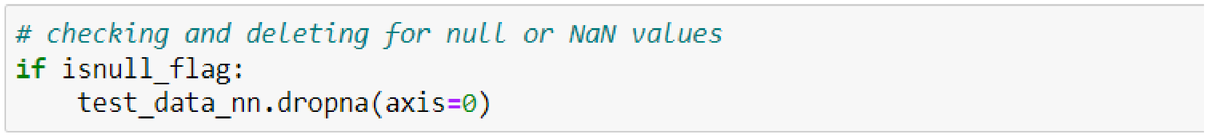


```
test_data_nn = test_data_nn[features]
```

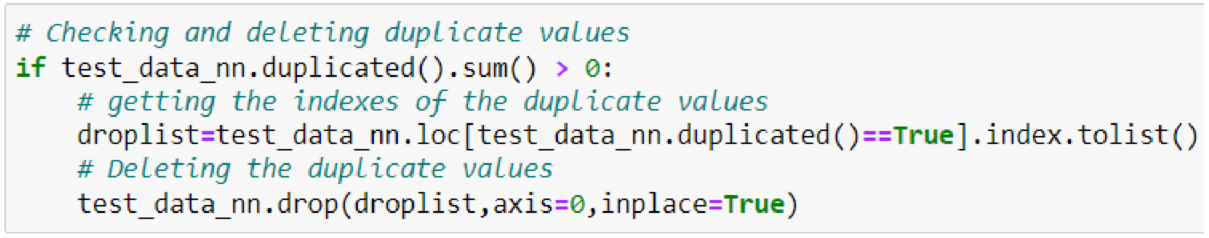
The missing values and the duplicate values all were deleted.



```python
isnull_flag = False
isnull_list = test_data_nn.isnull().sum().tolist()
for ev in isnull_list:
    if ev>0:
        isnull_flag = True
        break
```



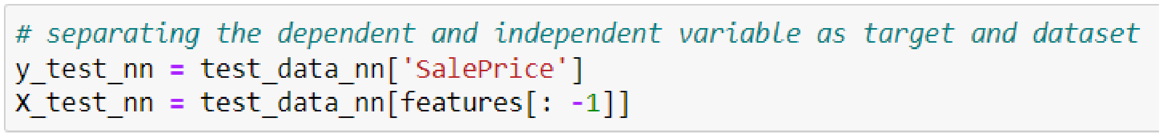
```python
# checking and deleting for null or NaN values
if isnull_flag:
    test_data_nn.dropna(axis=0)
```



```python
# Checking and deleting duplicate values
if test_data_nn.duplicated().sum() > 0:
    # getting the indexes of the duplicate values
    droplist=test_data_nn.loc[test_data_nn.duplicated()==True].index.tolist()
    # Deleting the duplicate values
    test_data_nn.drop(droplist,axis=0,inplace=True)
```

ii. Data separation into dependent and independent variables

The test data was separated into dependent or target variables and independent variables using a simple python slicing method.



```python
# separating the dependent and independent variable as target and dataset
y_test_nn = test_data_nn['SalePrice']
X_test_nn = test_data_nn[features[: -1]]
```

iii. Data Normalization

The minimum and maximum values from train data that were saved were used for the normalization of test data.

```
def normalization_data2(test, values):
    min_value = values[0]
    max_value = values[1]

    new_data = []
    for ea in test:
        new_value = (ea - min_value)/(max_value - min_value)
        new_data.append(round(new_value, 2))
    return new_data
```

```
count = 0
# Normalizing the dataset
for i in X_test_nn.columns:
    X_test_nn[i] = normalization_data2(X_test_nn[i], normalization_values_nn[count])
    count += 1
# Dataset after normalization
X_test_nn
```

5. Data Modeling and performance

As the given data is continuous data, Linear Regression and Support Vector Machine models are used.

i. Linear Regression

The Linear Regression model from scikit learn library is used to fit the data.

```
# create linear regression object
reg = linear_model.LinearRegression()
```

```
# train the model using the training sets
reg.fit(X_train_nn, y_train_nn)
```

```
▾ LinearRegression
LinearRegression()
```

```
# regression coefficients
print('Coefficients: ', reg.coef_)
```

```
Coefficients:  [221361.55526829 275298.7078862   88479.94484284]
```

Three coefficients of values 221361.555, 275298.707, and 88479.9448 were calculated by the model.

```
# predicted output of the training dataset
y_train_pred_nn = reg.predict(X_train_nn)
# predicted output of the test dataset
y_test_pred_nn = reg.predict(X_test_nn)
```

```
# Mean Squared Error
print("Mean Squared Error:",mean_squared_error(y_test_nn,y_test_pred_nn))
```

```
Mean Squared Error: 1773700461.308778
```

```
print("RMSE",np.sqrt(mean_squared_error(y_test_nn,y_test_pred_nn)))
```

```
RMSE 42115.32335514923
```

```
# Mean Absolute Error
print("Mean Absolute Error:",mean_absolute_error(y_test_nn,y_test_pred_nn))
```

```
Mean Absolute Error: 28198.40237195548
```

The model has a Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error of 1773700461.308, 42115.323355, and 28198.40237 respectively.

ii. Support Vector Machine

The Support Vector Machine model from scikit learn library is used to fit the data.

```
# creating SVM classifier
clf = svm.SVC()
```

```
# fitting data to classifier
clf.fit(X_train_nn, y_train_nn)
```

The training accuracy and testing accuracy were 3.7% and 1.13% respectively.

```
# train accuracy score
accuracy_score(y_train_nn, y_train_pred_nn)
```

```
0.03721841332027424
```

```
# test accuracy score
accuracy_score(y_test_nn, y_test_pred_nn)
```

```
0.011389521640091117
```

```
# Mean Squared Error
print("Mean Squared Error:",mean_squared_error(y_test_nn,y_test_pred_nn))
```

```
Mean Squared Error: 2777757523.3234625
```

```
print("RMSE",np.sqrt(mean_squared_error(y_test_nn,y_test_pred_nn)))
```

```
RMSE 52704.435518497514
```

```
# Mean Absolute Error
print("Mean Absolute Error:",mean_absolute_error(y_test_nn,y_test_pred_nn))
```

```
Mean Absolute Error: 35019.83826879271
```

The model has a Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error of 2777757523.323, 52704.435, and 35017.83826 respectively.

D. Using non-numerical attributes only

Taking only non-numerical columns from the dataset into df_nn using the nonnumerical_columns from before.

```
nonnumerical_columns.append('SalePrice')
```

```
df_nn = df[nonnumerical_columns]
```

1. Splitting into training and testing datasets

For splitting the dataset into training and testing datasets, the dataset needs to be shuffled first as before.

```
# Shuffling of data
df_nn = df_nn.sample(frac=1)
df_nn
```

Then the shuffled dataset is split into train data and test data in a 70 to 30 ratio.

```
# splitting of training and testing data
# 70% => training data and 30% => testing data
separating_index = int(len(df_nn)*0.7)
train_data_nn = df_nn[:separating_index]
test_data_nn = df_nn[separating_index:]
```

2. For Train data

i. Data Preprocessing

The separated train data might contain missing values, different unique symbols, and duplicate values. The null values or the missing value columns are deleted, and the duplicate row values are deleted.

```
# deletion of missing values columns
isnull_flag = False
isnull_list = test_data_nn.isnull().sum().tolist()
for ev in isnull_list:
    if ev>0:
        isnull_flag = True
        break

if isnull_flag:
    train_data_nn.dropna(axis=1, inplace=True)
```

```
# Checking and deleting duplicate values
if train_data_nn.duplicated().sum() > 0:
    # getting the indexes of the duplicate values
    droplist=train_data_nn.loc[train_data_nn.duplicated()==True].index.tolist()
    # Deleting the duplicate values
    train_data_nn.drop(droplist,axis=0,inplace=True)
```

ii. Converting categorical values to numerical values

The categorical values cannot be used for mathematical calculations, and they need to be converted into numerical values. So, the label encoder from preprocessing method of scikit learn library is used for encoding the categorical values.

```
# setting encoder for categorical columns
label_encoder = preprocessing.LabelEncoder()
```

```
# categorical to numerical converter
for cl in train_data_nn.columns:
    label_encoder.fit(train_data_nn[cl])
    train_data_nn[cl] = label_encoder.transform(train_data_nn[cl])
```

iii. Feature Selection and Feature Pruning

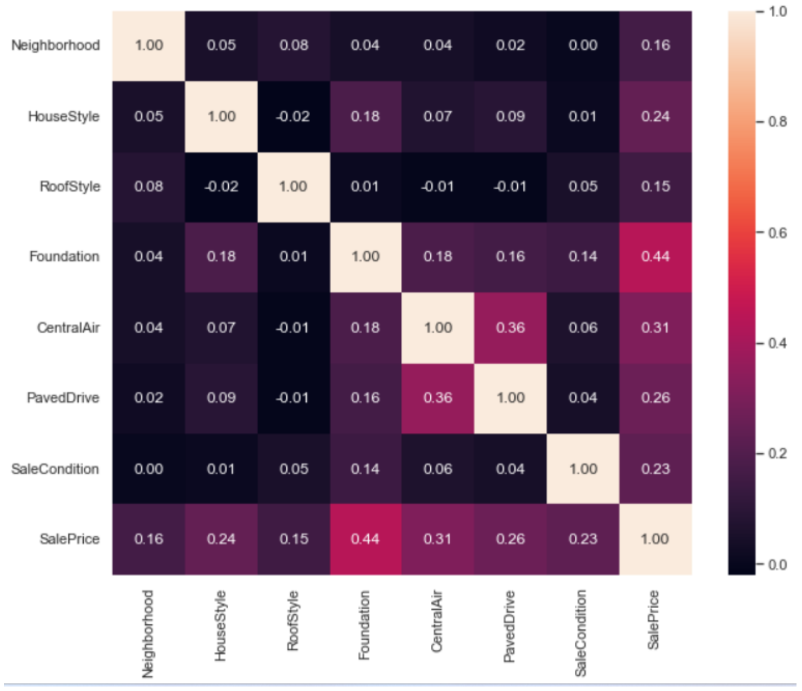
The feature selection and pruning are done using the correlation values. At first, the correlation between the features is calculated. Then, a 15% correlation is used as a threshold value to filter out the best features related to the target or dependent variable.

```python
# calculating correlation between the features
cormat = train_data_nn.corr()
cormat = round(cormat, 2)
```

```python
# taking 15% correlation as our threshold and selecting the features
features = []

for index, ec in enumerate(cormat.SalePrice):
    if ec >= 0.15:
        features.append(train_data_nn.columns[index])
```

Then, the correlation of the filtered features is calculated again and visualized through the HeatMap.



iv. Data separation into dependent and independent variables

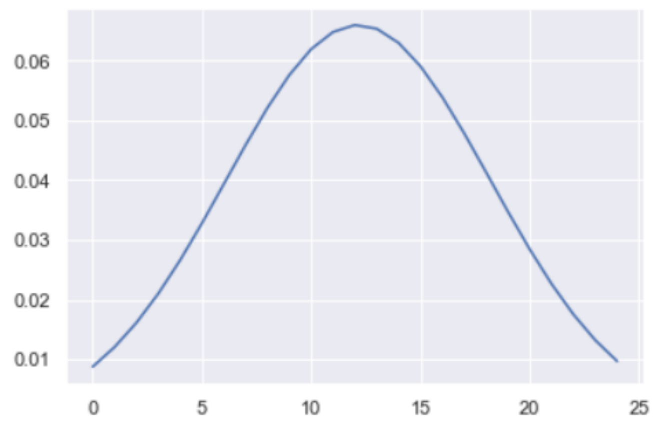
The pre-processed train data is separated into the target or dependent variable and independent variable using a simple python list-slicing method as before.

```python
# separating the dependent and independent variable as target and dataset
y_train_nn = train_data_nn['SalePrice']
X_train_nn = train_data_nn[features[: -1]]
```

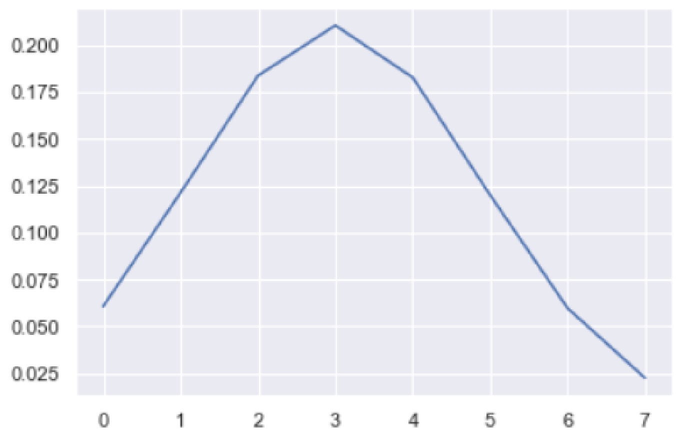
v. Measure of Central Tendency For Each Feature

The visualization of the Central Tendency for each feature is done by using the Normal distribution method. Some of the figures are shown below. It is also used to determine the outliers, but the outliers are not removed in this process for more generalization.

```
For Feature  Neighborhood
```



```
For Feature  HouseStyle
```



vi. Data Normalization

To normalize the data, the scaling data normalization method was applied to the feature columns of the dataset as before, using the same function.

```python
# Normalizing the dataset
for i in X_train_nn.columns:
    X_train_nn[i] = normalization_data(X_train_nn[i])
# Dataset after normalization
X_train_nn
```

3. For Test data

i. Feature Selection and Data Preprocessing

Already selected features in the training set were selected for test data.

```
test_data_nn = test_data_nn[features]
```

The missing value columns and the duplicate values all were deleted.

```
# deletion of missing values columns
isnull_flag = False
isnull_list = test_data_nn.isnull().sum().tolist()
for ev in isnull_list:
    if ev>0:
        isnull_flag = True
        break

if isnull_flag:
    test_data_nn.dropna(axis=1, inplace=True)
```

```
# Checking and deleting duplicate values
if test_data_nn.duplicated().sum() > 0:
    # getting the indexes of the duplicate values
    droplist=test_data_nn.loc[test_data_nn.duplicated()==True].index.tolist()
    # Deleting the duplicate values
    test_data_nn.drop(droplist,axis=0,inplace=True)
```

ii. Converting categorical values to numerical values

The categorical values were converted into numerical values using a label encoder from scikit learn library as before.

```
# categorical to numerical converter
for cl in features[: -1]:
    label_encoder.fit(test_data_nn[cl])
    test_data_nn[cl] = label_encoder.transform(test_data_nn[cl])
```

iii. Data separation into dependent and independent variables

The test data was separated into dependent or target variables and independent variables using a simple python slicing method as before.

```
# separating the dependent and independent variable as target and dataset
y_test_nn = test_data_nn['SalePrice']
X_test_nn = test_data_nn[features[: -1]]
```

iv. Data Normalization

The minimum and maximum values from train data that were saved were used for the normalization of test data.

```
count = 0
# Normalizing the dataset
for i in X_test_nn.columns:
    X_test_nn[i] = normalization_data2(X_test_nn[i], normalization_values_nn[count])
    count += 1
# Dataset after normalization
X_test_nn
```

4. Data Modeling and performance

As the given data is continuous data, Linear Regression and Support Vector Machine models are used like in the numerical data.

i. Linear Regression

The Linear Regression model from scikit learn library is used to fit the data.

```
# create linear regression object
reg = linear_model.LinearRegression()
```

```
# train the model using the training sets
reg.fit(X_train_nn, y_train_nn)
```

```
▾ LinearRegression
LinearRegression()
```

```
# regression coefficients
print('Coefficients: ', reg.coef_)
```

```
Coefficients:  [ 62.18645892  76.23345515 108.99251913 304.30857712  93.48804097
  67.49440167  99.51264831]
```

Seven coefficients of values 62.186, 76.2334, 108.9925, 304.3085, 93.488, 67.4944, and 99.5126 were calculated by the model.

```
# Mean Squared Error
print("Mean Squared Error:",mean_squared_error(y_test_nn,y_test_pred_nn))
```

```
Mean Squared Error: 43467795872.65857
```

```
print("RMSE",np.sqrt(mean_squared_error(y_test_nn,y_test_pred_nn)))
```

```
RMSE 208489.31836585436
```

```
# Mean Absolute Error
print("Mean Absolute Error:",mean_absolute_error(y_test_nn,y_test_pred_nn))
```

```
Mean Absolute Error: 188715.48548332567
```

The model has a Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error of 43467795872.65857, 208489.31836, and 188715.48548 respectively.

ii. Support Vector Machine

The Support Vector Machine model from scikit learn library is used to fit the data.

```
# creating SVM classifier
clf = svm.SVC()
```

```
# fitting data to classifier
clf.fit(X_train_nn, y_train_nn)
```

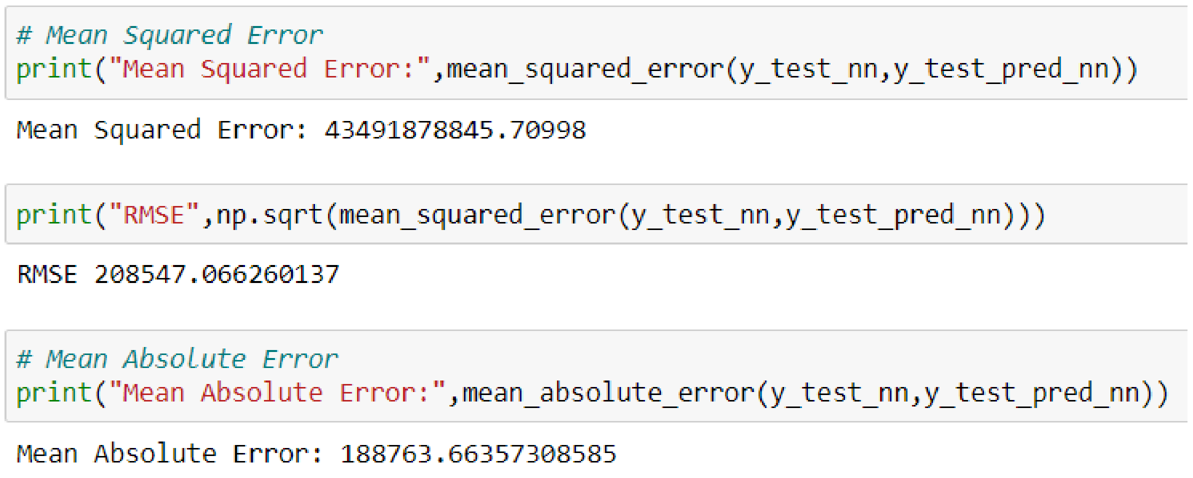
The training accuracy and testing accuracy were 3.6% and 0% respectively.

```
# train accuracy score
accuracy_score(y_train_nn, y_train_pred_nn)
```

```
0.036274509803921570
```

```
# test accuracy score
accuracy_score(y_test_nn, y_test_pred_nn)
```

```
0.0
```



```
# Mean Squared Error
print("Mean Squared Error:",mean_squared_error(y_test_nn,y_test_pred_nn))
```

```
Mean Squared Error: 43491878845.70998
```

```
print("RMSE",np.sqrt(mean_squared_error(y_test_nn,y_test_pred_nn)))
```

```
RMSE 208547.066260137
```

```
# Mean Absolute Error
print("Mean Absolute Error:",mean_absolute_error(y_test_nn,y_test_pred_nn))
```

```
Mean Absolute Error: 188763.66357308585
```

The model has a Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error of 43491878845.70998, 208547.066, and 188763.66357 respectively.

E. Comparison between models and Conclusion

The comparison between models for both numerical attributes and non-numerical attributes is done as in the Model performance table, Model comparison table-1, and Model comparison table-2.

Table-1: Model Performance Table

Models / Evaluation Metrics	**Numerical Attributes**		**Non-Numerical Attributes**	
	Linear Regression (LR1)	**Support Vector Machine (SVM1)**	**Linear Regression (LR2)**	**Support Vector Machine (SVM2)**
Mean Squared Error	1773700461.308	2777757523.323	43467795872.65857	43491878845.70998
Root Mean Squared Error	42115.323355	52704.435	208489.31836	208547.066
Mean Absolute Error	28198.40237	35017.83826	188715.48548	188763.66357

Table-2: Model Comparison table-1

Evaluation Metrics	**Models**	
	Linear Regression	**Support Vector Machine**
Mean Squared Error	LR2 > LR1	SVM2 > SVM1
Root Mean Squared Error	LR2 > LR1	SVM2 > SVM1
Mean Absolute Error	LR2 > LR1	SVM2 > SVM1

From the table above, it can be concluded that the models from numerical attributes are more accurate than the models from categorical data.

Table-3: Model Comparison table-2

Evaluation Metrics	**Models for numerical attributes**
	Linear Regression Vs Support Vector Machine
Mean Squared Error	LR1 < SVM1
Root Mean Squared Error	LR1 < SVM1
Mean Absolute Error	LR1 < SVM1

From this table, as all evaluation metrics for Linear Regression model, are low, Linear Regression model is more accurate than the SVM for this case. So, the Linear Regression model with numerical attributes is the best among the process we have performed.