Manarat International University

Department of Computer Science and Engineering

Artificial Intelligence (CSE – 411)

Project Report of the contest

Title: House price prediction .

Team name :AH

Contests name :

Ariful Hoque ID:1640CSE00491

Abul Hasanat ID:1640CSE00530

Shazedul Karim Zihad ID:1640CSE00479

Kaggle

git

Introduction:-

1 project Goal: In this project, we will predicted of house price for a buyer . so that buyer can buy a house minimum prices and better houses . House price are related with there location ,room length weight ,numbers of room ,fontal areas, bad rooms ,dinning rooms etc .

1.1 problem statement: To predict house prices from test.cvs and train.csv file ,using advance regression technique and linear regression algorithm .firstly I look at train data after I have trained a model .

Data processing :

we plotting and allow to visualize the distribution of data .so import functionality into the environment as well .check outlier ,and see other pattern that we might miss otherwise .using matplotlib .

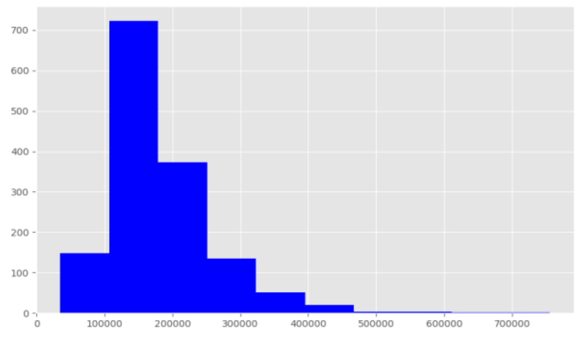
```
plt.style.use(style='ggplot')
plt.rcParams['figure.figsize'] = (10, 6)
```

To get more information about sales prices like count , mean,std,etc we use series.describe() method.

```
print (train.SalePrice.describe())
```

check skewness which is measure of the shape that distribute of values .

```
print ("Skew is:", train.SalePrice.skew())

plt.hist(train.SalePrice, color='blue')

plt.show()
```



Note that We see this graph that longer tail r right side so the distribute positively skewed .skew is :1,8828757597682129

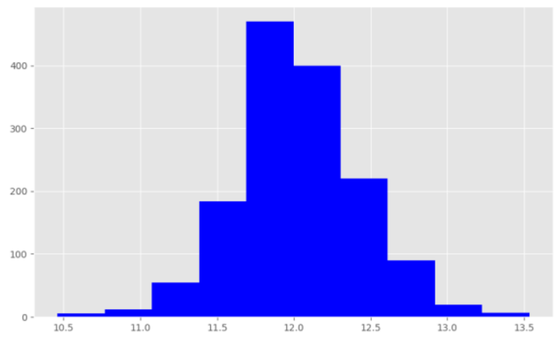
When performing regression sometimes it make s sense to log transform he target variable, one reason for this is to improve the linearity of the data .

Importantly the predictions generated by the final model is also be log-transformed.

So we was converting these predictions back to their original form later .np.log() will transform the variable ,and np.exp() will reverse transformation.

```
target = np.log(train.SalePrice)

print ("\n Skew is:", target.skew())

plt.hist(target, color='blue')
plt.show()
```

A value closer to 0 means that we have improved the skewness of the data ,now we can see there is a normal distribution .



.. figure normal distribution

Feature engineering :feature engineering is the process of making features of the data suitable for use in machine learning and modeling.

The DataFrame.crr() method displays the correlation between the columns. Then I can examine the correlations between the feature and the target.

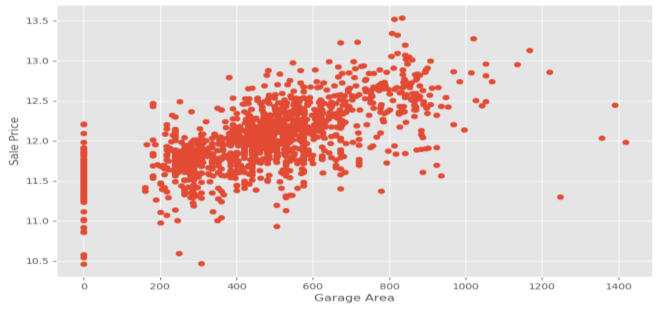
```
numeric_features = train.select_dtypes(include=[np.number])

corr = numeric_features.corr()

print (corr['SalePrice'].sort_values(ascending=False)[:5], '\n')

print (corr['SalePrice'].sort_values(ascending=False)[-5:])
```

The first five Features are the most positively correlated with sale price ,while next five are negatively correlated.

Handling outliers:lets use ptl.scatter() to generate some scatter plots and visualize the relationship between the Garege Area 'garageArea' and Saleprice.

```
plt.scatter(x=train['GarageArea'], y=target)

plt.ylabel('Sale Price')

plt.xlabel('Garage Area')

plt.show()
```

As we seen that there are many homes with 0 for garageArea, indicating that they don’t have a garge. There are a few outliers as well .Outliers can affect a regression model by pulling the estimated regression line further away from the true population regression line ,so we has removed those observations of data .

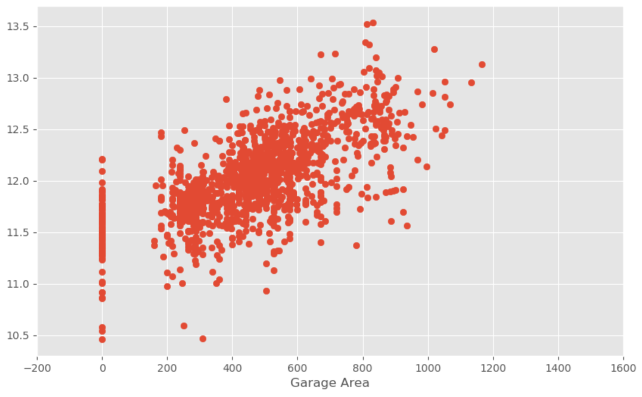


we are crate a new data frame with some outliers removed.

```
train = train[train['GarageArea'] < 1200]
```

Display the previous graph again without outliers

```
plt.scatter(x=train['GarageArea'], y=np.log(train.SalePrice))

plt.xlim(-200,1600)

plt.ylabel('Sale Price')

plt.xlabel('Garage Area')

plt.show()
```



where we see that outliers removed .

Handling missing values: we will create a Data frame to view the top missing columns. Chaining together the train.isnull().sum()

methodes, we return a series of the counts of the null values in each column.

```
nulls = pd.DataFrame(train.isnull().sum().sort_values(ascending=False)[:25])

nulls.columns = ['Null Count']

nulls.index.name = 'Feature'

#nulls

print(nulls)
```

wrangling the non-numeric feature:-view non-numaric futures

```
categoricals = train.select_dtypes(exclude=[np.number])

#categoricals.describe()

print(categoricals.describe())
```

The count column indicates the count of non-null observations ,while unique counts the number of unique values.Top is the most commonly occurring value,with frequency of the top value show by freq.

Transforming features:for many of those features we might want to use one-hot encoding to make use of the information for modeling.

One-hot encoding is a technique which will transform categorical data into numbers so the model can understand whether or not aa particular observation falls into one category or another .

When transformation featers, it’s important to remember that any transformation that’s we applied the training data before fitting the model must be applied to the test data .

e.g:

i)consider the street data ,which indicates wheter there is graval or paved road access to the property

```
print ("Original: \n")

print (train.Street.value_counts(), "\n")
```

In the street column , the unique values are Pave and Grvl, which describe the type of road access to the property ,the model needs numerical data ,so, I have used one-hot encoding to transform the data into a Boolean column .

Create a new column called enc-street . the pd.get_dummies() method will be used for this.

As mentioned earlier we need to this onboth .the train and test data.

```
train['enc_street'] = pd.get_dummies(train.Street, drop_first=True)

test['enc_street'] = pd.get_dummies(test.Street, drop_first=True)
```

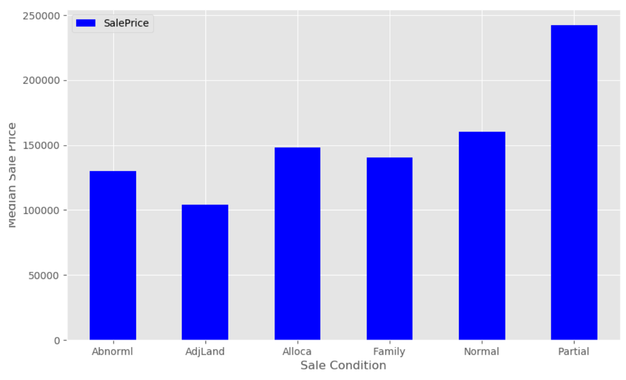
now we see encode value :

```
print ('Encoded: \n')

print (train.enc_street.value_counts())
```

pave and Grv value converted into 1 and 0.

ii)look at SaleCondition by constructing and plotting a pivot table, as we did above for OverallQual

```
condition_pivot = train.pivot_table(index='SaleCondition', values='SalePrice', aggfunc=np.median)

condition_pivot.plot(kind='bar', color='blue')

plt.xlabel('Sale Condition')

plt.ylabel('Median Sale Price')

plt.xticks(rotation=0)

plt.show()
```

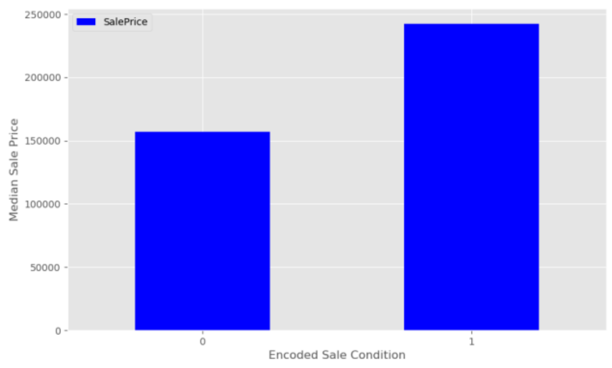


Note that partial has a significantly higher median sale price than the others we will encode this as a new feature, we select all houses where SeleCondition and assign the value 1, otherwise assign 0.

```
 def encode(x): return 1 if x == 'Partial' else 0

train['enc_condition'] = train.SaleCondition.apply(encode)

test['enc_condition'] = test.SaleCondition.apply(encode)

print("\n")
```

explore this newly modified feature as a plot.

```
condition_pivot = train.pivot_table(index='enc_condition', values='SalePrice', aggfunc=np.median)

condition_pivot.plot(kind='bar', color='blue')

plt.xlabel('Encoded Sale Condition')

plt.ylabel('Median Sale Price')

plt.xticks(rotation=0)

plt.show()
```



Dealing with missing values:

We'll fill the missing values with an average value and then assign the results to data This is a method of interpolation

The DataFrame interpolate() methods makes it simple

```
data = train.select_dtypes(include=[np.number]).interpolate().dropna()
```

Check if the all of the columns have 0 null values.

```
# sum(data.isnull().sum() != 0)
```

```
print(sum(data.isnull().sum() != 0
```

AS YOU SE THERE IS NO NULL

Build a linear model:

separate the features and the target variable for modeling.We will assign the features to X and the target variable(Sales Price)to y.

I use np.log() as explained above to transform the Y variable for the model Data.Drop([features],axis=1) tells pandas which columns we want to exclude.

Iwon't include SalePrice since it is the target variable , and ID is just and index with no relationship to SalePrice.

```
y = np.log(train.SalePrice)
X = data.drop(['SalePrice', 'Id'], axis=1)
```

partition the data:

Partitioning the data in this way allows us to evaluate how our model might perform on data that it has never seen before.If we train the model on all of the test data, it will be difficult to tell if overfitting has taken place. also state how many percentage from train data set, we want to take as test data set.

In this example, about 33% of the data is devoted to the hold-out set.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=.33)
```

create a Linear Regression model:

First, we instantiate the model.

```
lr = linear_model.LinearRegression()
```

then I need to fit the model (model fitting).this is used to estimate the relationship between our predictors and the target variable so we can make accurate predictions new data.i have fited the model using x_train aand y_train.and I have scored x_test and y_test . the ir.fit() method will fit the linear regression on the features and terget variable that we pass.

```
model = lr.fit(X_train, y_train)
```

Evaluate the performance:

The r-squared value is a measure of how close the data are to the fitted regression line a higher r-squared value means a better fit(very close to value 1)

```
print("R^2 is: \n", model.score(X_test, y_test))
```

use the model we have built to make predictions on the test data set.

```
predictions = model.predict(X_test)
```

calculates the rmse

```
print('RMSE is: \n', mean_squared_error(y_test, predictions))
```

view this relationship between predictions and actual_values graphically with a scatter plot.

```
actual_values = y_test

plt.scatter(predictions, actual_values, alpha=.75,

            color='b')  # alpha helps to show overlapping data

plt.xlabel('Predicted Price')

plt.ylabel('Actual Price')

plt.title('Linear Regression Model')

plt.show()
```

improve the model:

try using Ridge Regularization to decrease the influence of less important features . experiment by looping through a few different values of alpha, and see how this changes our results.

```
for i in range (-2, 3):

  alpha = 10**i

  rm = linear_model.Ridge(alpha=alpha)
```

```
ridge_model = rm.fit(X_train, y_train)
preds_ridge = ridge_model.predict(X_test)

plt.scatter(preds_ridge, actual_values, alpha=.75, color='b')
plt.xlabel('Predicted Price')
plt.ylabel('Actual Price')
plt.title('Ridge Regularization with alpha = {}'.format(alpha))
overlay = 'R^2 is: {}\nRMSE is: {}'.format(
            ridge_model.score(X_test, y_test),
            mean_squared_error(y_test, preds_ridge))
plt.annotate(s=overlay,xy=(12.1,10.6),size='x-large')
plt.show()
```

over all complete .

result: position 1686

score :0.127182