Data Profiling (Preprocessing)

```
import pandas
import seaborn as sb
import matplotlib.pyplot as plt

moviedata = pandas.read_csv("tmdb_5000_movies.csv")

# Gathering dataset information
moviedata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4803 non-null   int64  
 1   genres                4803 non-null   object 
 2   homepage              1712 non-null   object 
 3   id                    4803 non-null   int64  
 4   keywords              4803 non-null   object 
 5   original_language     4803 non-null   object 
 6   original_title        4803 non-null   object 
 7   overview              4800 non-null   object 
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object 
 10  production_countries  4803 non-null   object 
 11  release_date          4802 non-null   object 
 12  revenue               4803 non-null   int64  
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object 
 15  status                4803 non-null   object 
 16  tagline               3959 non-null   object 
 17  title                 4803 non-null   object 
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64  
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

```
# Checking any NaN value in every column
moviedata.isna().any()
```

```
budget                  False
genres                  False
homepage                 True
id                      False
keywords                False
original_language       False
original_title          False
overview                 True
popularity              False
production_companies    False
production_countries    False
release_date             True
```

```
revenue                False
runtime                 True
spoken_languages       False
status                 False
tagline                 True
title                  False
vote_average           False
vote_count             False
dtype: bool
```

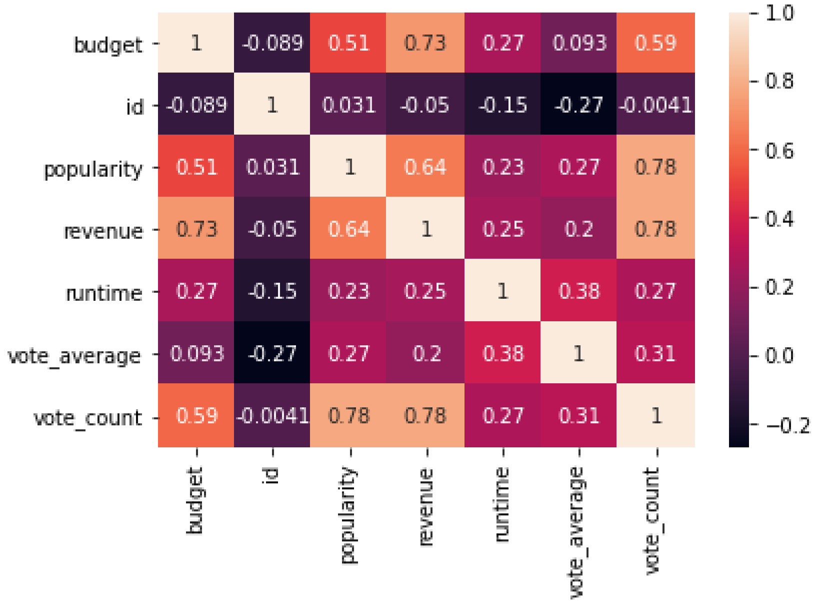
```
# NaN percentage in every column
print("NaN Percentage in every column\n")
for i in moviedata:
  percentage = (moviedata[i].isna().sum()/11466)*100
  print(i, ": %.2f %%" % percentage)
```

```
NaN Percentage in every column

budget : 0.00 %
genres : 0.00 %
homepage : 26.96 %
id : 0.00 %
keywords : 0.00 %
original_language : 0.00 %
original_title : 0.00 %
overview : 0.03 %
popularity : 0.00 %
production_companies : 0.00 %
production_countries : 0.00 %
release_date : 0.01 %
revenue : 0.00 %
runtime : 0.02 %
spoken_languages : 0.00 %
status : 0.00 %
tagline : 7.36 %
title : 0.00 %
vote_average : 0.00 %
vote_count : 0.00 %
```

```
# Describing Data
moviedata.describe()
```

```
# Data Correlation Matrix
correlation_matrix = moviedata.corr()
sb.heatmap(data = correlation_matrix, annot = True)
plt.show()
```



Data Cleaning and Data Distribution (Preprocessing)

```
# Removing unrelated columns
upd_moviedata = moviedata.drop(['genres', 'homepage', 'id', 'keywords', 'original_language
```

```
# Checking any NaN value in every column
upd_moviedata.isna().any()
```

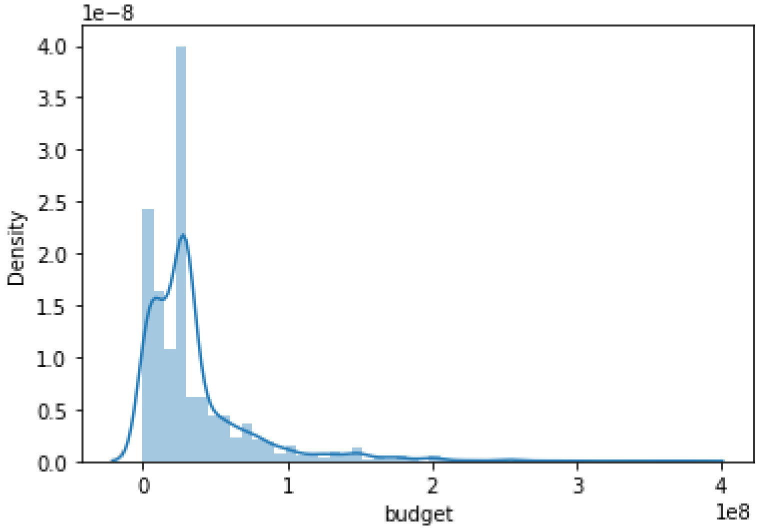
```
budget          False
popularity      False
revenue         False
runtime          True
vote_average    False
vote_count      False
dtype: bool
```

```
# Changing value 0 into mean for each column
upd_moviedata = upd_moviedata.replace(0,upd_moviedata.mean())
```

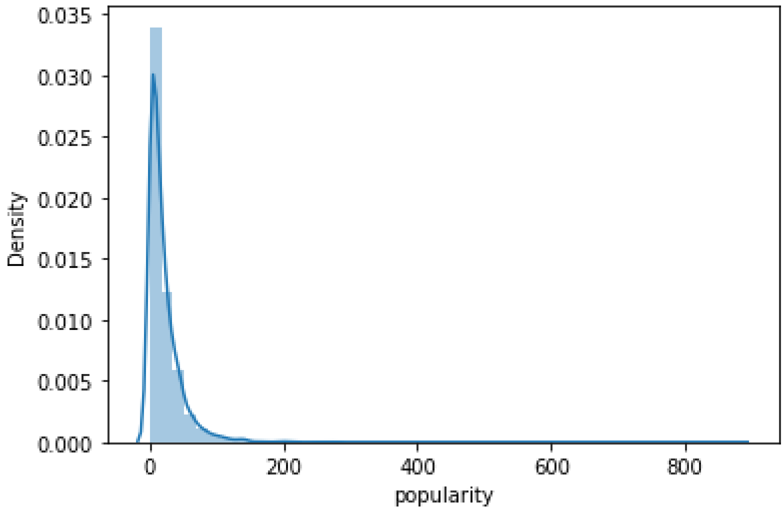
```
# Changing NaN value into mean for each column
upd_moviedata = upd_moviedata.fillna(upd_moviedata.mean())
```

```
# Data distribution and Probability Density Function
for i in upd_moviedata:
  sb.distplot(upd_moviedata[i])
  plt.show()
```

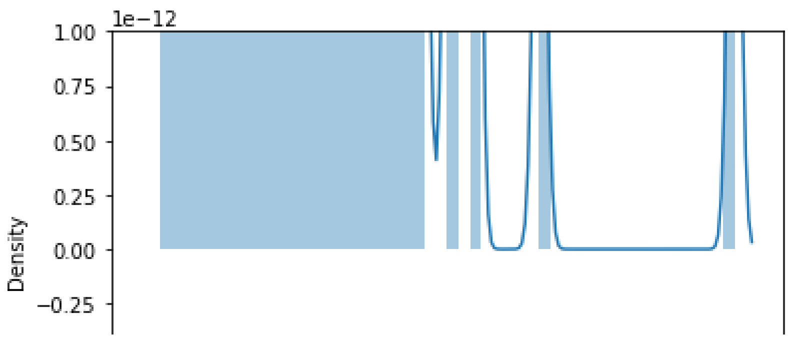
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
```




```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



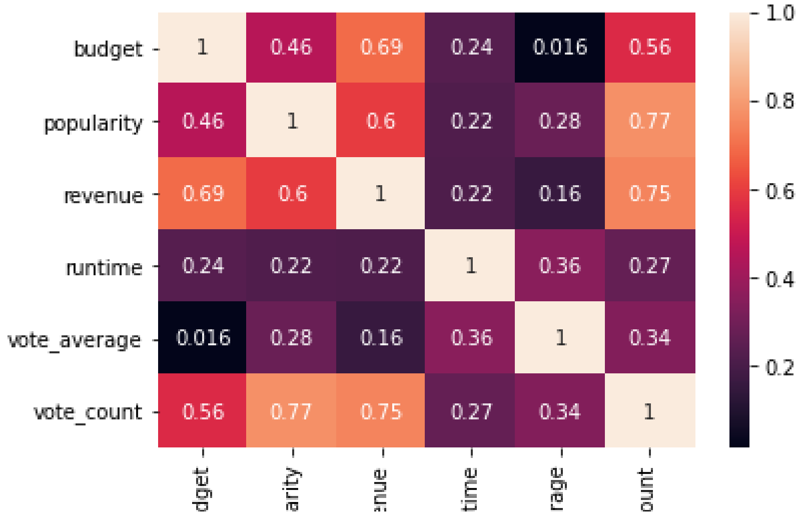

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



Choosing Dependant Variables (Feature Engineering)

0.0 0.5 1.0 1.5 2.0 2.5 3.0

```
corr_matrix = upd_moviedata.corr()
sb.heatmap(data = corr_matrix, annot = True)
plt.show()
```



I believe that *revenue* can be considered as the Y since it is dependant with *budget*, *popularity*, *vote_count* and has the suitable correlation values, which are 0.69, 0.6, and 0.75 consecutively (above or equal with 0.6). It also means that *budget*, *popularity*, *vote_count* will be the most suitable predictor for *revenue* since it has equal or higher correlation values than 0.6, which are 0.69, 0.6, and 0.75 consecutively. Meanwhile, *runtime* and *vote_average* are not included as the predictor of *revenue* since they have correlation values below 0.3 and close to 0.

Data Training

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# 1. Splitting data between train and test
x = upd_moviedata[["budget", "popularity", "vote_count"]]
y = upd_moviedata["revenue"]
trainx, testx, trainy, testy = train_test_split(x, y, test_size=0.2)

# 2. Fit data into Linear Model
model = LinearRegression().fit(trainx, trainy)

# 3. Data Test Prediction
test_pred = model.predict(testx)
for i in range(len(testy)):
  print(testx.values[i], test_pred[i])
```

```
[2.90450399e+07 1.86644300e+00 2.20000000e+01] 52044577.25166921
[6.2440870e+06 1.4970654e+01 2.5900000e+02] 32117095.42860692
[1.90000e+07 7.38652e-01 3.00000e+00] 35003823.47447349
[5.5000000e+07 1.5274137e+01 3.0300000e+02] 111537196.91581115
[2.90450399e+07 1.67544520e+01 3.22000000e+02] 72079683.85513324
[2.90450399e+07 1.20245000e+00 8.00000000e+00] 51110389.6978876
[6.00000000e+06 1.43659698e+02 5.89300000e+03] 404149146.8368335
[4.000000e+07 8.303696e+00 1.520000e+02] 77921501.14905898
[1.300000e+07 6.735922e+00 1.540000e+02] 35632168.72497571
[2.90450399e+07 2.14923006e+01 6.90217989e+02] 96325842.98557508
[2.90450399e+07 7.43631500e+00 1.42000000e+02] 60048804.22123356
```

```
[3.8000000e+07 1.4779041e+01 4.5200000e+02] 94603017.8547202
[2.90450399e+07 7.16764000e-01 1.30000000e+01] 51425615.329102226
[1.3000000e+07 1.6681567e+01 3.0800000e+02] 45975525.22358976
[2.90450399e+07 6.98835700e+00 7.10000000e+01] 55385506.44846419
[1.500000e+07 1.833058e+00 7.000000e+00] 29015196.771957505
[2.90450399e+07 1.03914900e+00 4.00000000e+00] 50844151.75113996
[6.0000000e+07 2.3700759e+01 3.7400000e+02] 124251966.40187955
[4.0000000e+07 2.8029015e+01 8.5200000e+02] 124287643.0092929
[3.0000000e+07 2.2527211e+01 1.2140000e+03] 132169107.1234017
[2.90450399e+07 4.54290000e-02 4.00000000e+00] 50818838.41218146
[5.0000000e+07 3.8616744e+01 8.5400000e+02] 140384844.033484
[ 1.        1.859965 42.      ] 7764381.222262649
[4.8000000e+07 8.8844777e+01 2.2000000e+03] 226714317.45529556
[3.500000e+07 3.503408e+00 5.300000e+01] 63464559.49133373
[9.2620000e+07 3.2351527e+01 6.5700000e+02] 194216201.94892082
[6.2000000e+07 1.8096884e+01 3.4000000e+02] 125020844.53662546
[1.7500000e+08 3.2852443e+01 1.9620000e+03] 409039368.1052439
[5.000000e+06 3.792015e+00 3.700000e+01] 15334226.42294484
[2.90450399e+07 1.08850100e+00 6.00000000e+00] 50976447.974767014
[2.90450399e+07 1.02003000e-01 3.00000000e+00] 50754760.00691542
[2.0000000e+05 3.8771062e+01 8.8300000e+02] 64120483.4014851
[2.90450399e+07 2.41472100e+00 3.10000000e+01] 52648219.499314725
[7.5000000e+06 2.5281197e+01 8.4100000e+02] 72483445.44286954
[8.400000e+07 4.434333e+01 1.526000e+03] 237927800.20580786
[9.00000e+06 7.17811e-01 1.40000e+01] 20027549.999320492
[8.5000000e+07 5.6257411e+01 2.5660000e+03] 307941250.7226883
[1.500000e+06 8.589355e+00 1.530000e+02] 17562936.21008442
[4.0000000e+07 2.4992057e+01 7.3600000e+02] 116610015.87621322
[8.00000000e+06 1.21463076e+02 8.42800000e+03] 572815031.4518367
[2.90450399e+07 3.89022300e+00 2.12000000e+02] 64544840.782407716
[4.9000000e+07 2.1746245e+01 4.3700000e+02] 111063805.81202644
[8.0000000e+07 1.4530946e+01 2.9200000e+02] 150038694.12906277
[3.0000000e+07 1.0756266e+01 2.0200000e+02] 65563495.40533486
[2.90450399e+07 1.27089630e+01 2.82000000e+02] 69355850.55895534
[2.90450399e+07 5.07901700e+00 1.40000000e+01] 51602255.89035786
[6.0000000e+07 2.7615108e+01 6.6900000e+02] 143679939.88555995
[1.200000e+07 4.580081e+00 7.500000e+01] 28831563.513523046
[2.300000e+07 2.733256e+01 5.460000e+02] 77536947.43363512
[2.90450399e+07 3.75423000e-01 2.00000000e+00] 50696205.387323216
[3.000000e+07 9.278361e+00 2.060000e+02] 65787926.400648765
[2.90450399e+07 1.08135000e-01 1.00000000e+00] 50623877.14449552
[6.0000000e+06 2.3431117e+01 3.8900000e+02] 40467020.52576992
[1.0000000e+07 1.8664624e+01 4.1700000e+02] 48458732.049058646
[2.90450399e+07 2.21730000e-02 2.00000000e+00] 50687206.94008749
[8.0000000e+07 1.5673154e+01 3.5900000e+02] 154457598.61870927
[2.90450399e+07 9.98300400e+00 2.44000000e+02] 66796669.126073256
[2.0000000e+06 2.6281839e+01 5.1900000e+02] 42778594.08776049
[3.50000e+06 2.55748e+01 5.03000e+02] 44066739.5613651
```

The first step that I have performed in the training process is splitting the data into train and test by utilizing sklearn.model_selection train_test_split with test size is 0.2 of the train size. The Xs are *budget*, *popularity*, *vote_count* and the Y is *revenue*. The second step that I have performed is fitting the train dataset into linear model by using .fit(). The third step is predict the Y value based on test dataset. In this stage, I use .predict() function. In addition, I also print all the Y predicted.

Evaluation

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# The learning model and evaluation results using evaluation metrics

# Dataset Test Result
print("R2 = ", r2_score(testy, test_pred))
print("Mean Absolute Error = ", mean_absolute_error(testy, test_pred))
print("Mean Squared Error = ", mean_squared_error(testy, test_pred))
```

```
R2 =  0.7060530973709802
Mean Absolute Error =  56202611.16459572
Mean Squared Error =  8702114743637363.0
```

To evaluate the learning model, **R2, Mean Absolute Error (MAE), and Mean Squared Error (MSE)** are utilized which can be obtained by using *sklearn.metrics* r2_score, mean_absolute_error, mean_squared_error. R2 shows the level of the variance in the dependent variable that explained by independent variable. R2 commonly measures the relation strength between your model and the reliant variable on 0 – 1 scale. From the dataset train result above, the **R2 score obtained is 0.7060530973709802**. Mean Absolute Error (MAE) measures the absolute average distance between the predicted and the real data. From the dataset train result above, the **MAE score obtained is 56202611.16459572**. Mean Squared Error (MSE) is an estimator measures the average of error squares such as the average squared difference between the estimated values and true value. From the dataset train result above, the **MSE score obtained is 8702114743637363**.

```
# Checking underfitting or overfitting possibility
# Data Training predicition
train_pred = model.predict(trainx)
for i in range(len(trainy)):
  print(trainx.values[i], train_pred[i])
```

```
[2.50000e+05 1.30169e-01 3.00000e+00] 5557465.69366238
[2.90450399e+07 1.34765960e+01 2.70000000e+02] 68589170.32477905
[1.500000e+07 6.216203e+00 2.520000e+02] 45179135.422979206
[3.5000000e+07 3.8100488e+01 9.2300000e+02] 121347854.85493566
[2.000000e+07 9.455596e+00 1.310000e+02] 45182018.7922151
[3.0000000e+06 1.1158167e+01 2.1600000e+02] 24110571.51438314
[4.0000000e+07 2.6072228e+01 9.0200000e+02] 127513773.78391697
[1.0000000e+07 1.0072199e+01 1.9600000e+02] 33760037.8752031
[4.700000e+07 6.067212e+00 7.400000e+01] 83741527.17940478
[4.300000e+07 5.418921e+00 2.080000e+02] 86226047.4233456
[8.000000e+06 4.932893e+00 8.100000e+01] 22955085.020912077
[1.7500000e+07 1.4304444e+01 1.7200000e+02] 44067721.47419163
[1.5000000e+07 4.3644978e+01 3.0450000e+03] 229128624.206643
[3.0000000e+06 1.3338539e+01 1.5500000e+02] 20169421.33386994
[1.500000e+06 4.680206e+00 1.830000e+02] 19428943.211955883
[2.0000000e+07 7.3567232e+01 3.0160000e+03] 235839005.38113794
[7.0000000e+06 2.1626288e+01 4.6300000e+02] 46839136.6952301
[1.5000000e+07 5.8553213e+01 1.4440000e+03] 124611614.97405098
[3.100000e+07 6.487344e+00 8.900000e+01] 59620690.41106211
```

```
[9.0000e+05 4.0971e+00 7.3000e+01] 11265153.562658973
[2.90450399e+07 1.58002700e+00 4.00000000e+00] 50857929.70483838
[2.0000000e+07 2.0495749e+01 2.9200000e+02] 56011892.76057032
[5.0000000e+07 2.1312186e+01 2.7400000e+02] 101942710.85782374
[1.3000000e+08 2.0678787e+01 4.5100000e+02] 239095192.8562332
[9.0000000e+07 1.6058284e+01 3.6900000e+02] 170819061.92101583
[1.6000000e+07 2.1380635e+01 4.0200000e+02] 56962999.32241112
[2.90450399e+07 2.24280900e+00 4.00000000e+01] 53233516.122856334
[5.5000000e+07 1.1329727e+01 1.3200000e+02] 100232879.69435589
[2.90450399e+07 1.96321900e+00 1.30000000e+01] 51457366.665405735
[2.90450399e+07 2.65398800e+00 2.30000000e+01] 52130158.16304191
[2.000000e+06 1.867004e+00 8.000000e+00] 8676186.346664248
[7.5000000e+07 3.1482872e+01 3.4620000e+03] 350319204.74979115
[2.90450399e+07 4.26237400e+00 4.80000000e+01] 53809117.38996108
[1.8000000e+07 3.4694216e+01 1.0980000e+03] 106043026.34313764
[1.5000000e+07 2.7165222e+01 6.8000000e+02] 73755136.16625977
[2.90450399e+07 7.70822700e+00 4.90000000e+01] 53962414.20869209
[2.90450399e+07 2.08825100e+00 1.50000000e+01] 51591590.70923605
[4.2000000e+07 2.8969151e+01 1.2620000e+03] 154313891.15160236
[2.90450399e+07 8.96224500e+00 9.00000000e+01] 56680659.02733443
[6.0000000e+07 1.8831023e+01 5.2600000e+02] 134086887.0226902
[5.0000000e+07 2.0415572e+01 2.7400000e+02] 101919871.13023756
[1.9000000e+07 2.1959742e+01 4.6500000e+02] 65814418.86805451
[1.2000000e+07 1.0297189e+01 1.3700000e+02] 33039408.193369254
[3.600000e+07 2.711589e+01 6.110000e+02] 102195592.41083369
[6.8000000e+07 2.2392544e+01 4.3700000e+02] 140903538.32763168
[2.800000e+07 2.439184e+00 2.600000e+01] 50680902.598246865
[6.000000e+06 1.181832e+01 1.670000e+02] 25625867.9431715
[1.6000000e+07 4.1083914e+01 1.6620000e+03] 140019517.88509786
[3.119200e+04 1.330379e+00 2.600000e+01] 6751537.215482621
[2.90450399e+07 3.81094300e+00 2.40000000e+01] 52225149.17036182
[1.1000000e+08 1.9625972e+01 3.8900000e+02] 203613248.28006285
[1.100000e+07 9.705589e+00 1.530000e+02] 32503004.956285417
[3.7000000e+07 2.4218358e+01 1.4680000e+03] 159841667.67210412
[3.500000e+07 1.884332e+01 4.140000e+02] 87507869.03595479
[7.5000000e+07 6.0442593e+01 3.2600000e+03] 337821959.1982658
[4.8000000e+07 1.8102572e+01 3.3900000e+02] 102980429.47752692
[3.600000e+07 1.068453e+00 1.400000e+01] 62416917.10577422
[4.800000e+06 3.883192e+01 8.870000e+02] 71604482.20915419
[9.4000000e+07 1.2516546e+01 1.8700000e+02] 165082870.20941356
```

```
# Dataset Train Result
print("R2 = ", r2_score(trainy, train_pred))
print("Mean Absolute Error = ", mean_absolute_error(trainy, train_pred))
print("Mean Squared Error = ", mean_squared_error(trainy, train_pred))
```

```
R2 =  0.6585761492504613
Mean Absolute Error =  53449454.55158375
Mean Squared Error =  7675263070504818.0
```

Regarding underfitting and overfitting, underfitting means that the model or the algorithm does not fit the data well enough. It usually happens when the data is not enough to build an accurate model. Meanwhile, overfitting occurs when a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set. Hence to check that whether my model is underfitting, overfitting, or not, the evaluation result with the dataset train result can be compared. The model is not underfit since the predictors which correlate with revenue are budget, popularity, vote_count (correlation value is above 0.6). On the other hand, the result of the dataset train result (R2 = 0.6585761492504613) is not exceed the dataset test result (R2 = 0.7060530973709802). Therefore, it can be concluded that this model is not overfitted.