TMDB Linear Regression

March 25, 2019

TMDB Box Office Prediction Using Linear Regression

```
In [4]: """
        Created on Fri Mar 15 14:42:28 2019

        @author: rafay
        """
        #Import the necessary Libraries
        import pandas as pd
        from sklearn.linear_model import LinearRegression
        import statsmodels.api as sm
        from sklearn.model_selection import train_test_split
        import numpy as np
        import os
```

In this competition the Training and Test set have already been split up for convenience. The revenue predictions are supposed to be made on a .csv file called "sample_submission" The predictions will fill the missing values or 1000000s in the sample_submission file under "Revenue" A preview of the sample_submission is shown below

```
In [12]: sample_submission = pd.read_csv("sample_submission.csv")
         sample_submission.head()
```

Out[12]:

	id	revenue
0	3001	1000000
1	3002	1000000
2	3003	1000000
3	3004	1000000
4	3005	1000000

First I'm going to import the test and train set which is provided by the competition

```
In [13]: #Import Train and Test files
         train = pd.read_csv("train.csv")
         test = pd.read_csv("test.csv")
```

For Linear Regression in this problem I will use the "Budget" column as my X and "Revenue" column as y. X is the variable used to make predictions on y. In the code below X and y are seperated from the training set:

```
In [14]: #Get x and y values from the dataset
         X = train["budget"].to_frame() #x is the budget(training)

         y = train["revenue"].to_frame() #y is the revenue(training)
```

Now I will split the training set further into a training and test set.I split the training set into two because we don't have revenue values in the original test set given by the competition. By splitting the training set into two I will make predictions on the test set and calculate my models performance from it.

```
In [15]: #Splitting the training set further into a training and test set
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

Next I'm going to import the Linear Regression object from the scikit-learn library and fit my regressor to the training values

```
In [16]: #Fit the data
         regressor = LinearRegression()
         model = regressor.fit(X_train,y_train)
```

Easy! Now I will make predictions for the revenue using the X_test object

```
In [21]: #Prediction of revenue
         y_pred = regressor.predict(X_test)
```

Now that I have the predictions of the revenue based off of the training data I will try to calculate the performance of my model using Root Mean Squared Logarithmic Error. Root Mean Squared Logarithmic Error (RMSLE) is the technique to find out the difference between the values predicted by your machine learning model and the actual values.

```
In [22]: #Root Mean Squared Logarithmic Error
         from sklearn.metrics import mean_squared_log_error
         from math import sqrt
         rmsle = sqrt(mean_squared_log_error(y_test, y_pred))
         print(rmsle)
```

```
2.4961637054675383
```

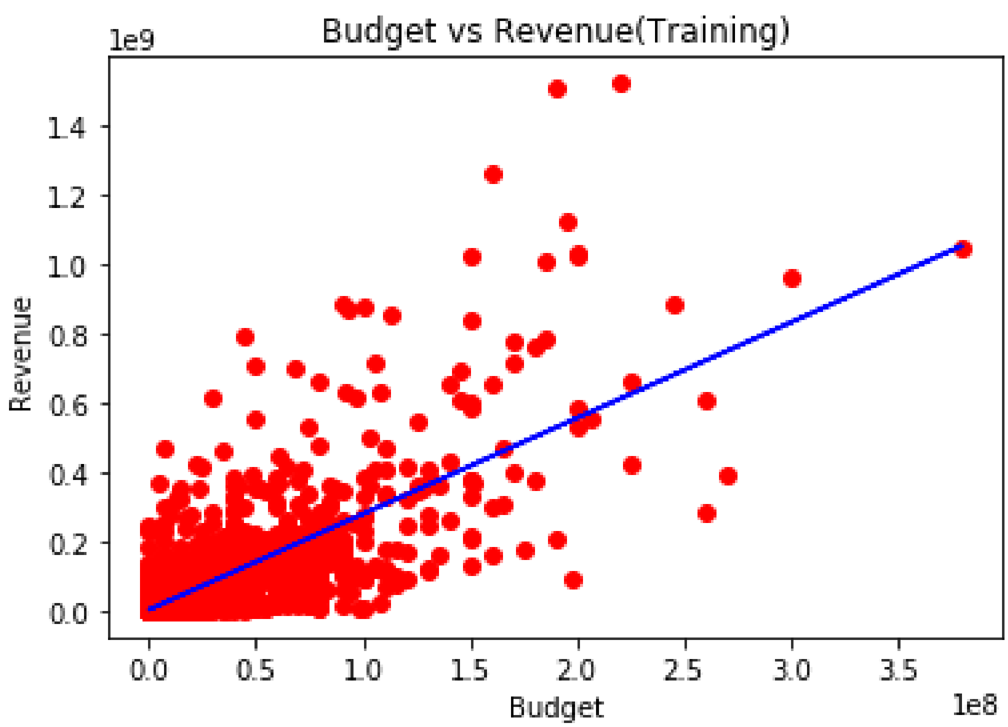
The score above is pretty decent. It means that my model has done well. However, I'm still not satisfied with the technique used to determine model efficiency. I will calculate the r-squared which is the statistical measure of how close the data are to the fitted regression line. 0 indicates that the model explains none of the variability of the response data around its mean. 1.0 indicates that the model explains all the variability of the response data around its mean. What we're aiming for here is a value closer to 1.0 but not exactly 1.0 Let's calculate r-squared below

```
In [26]: #R Squared
         r2 = model.score(X_train, y_train)
         r2
```

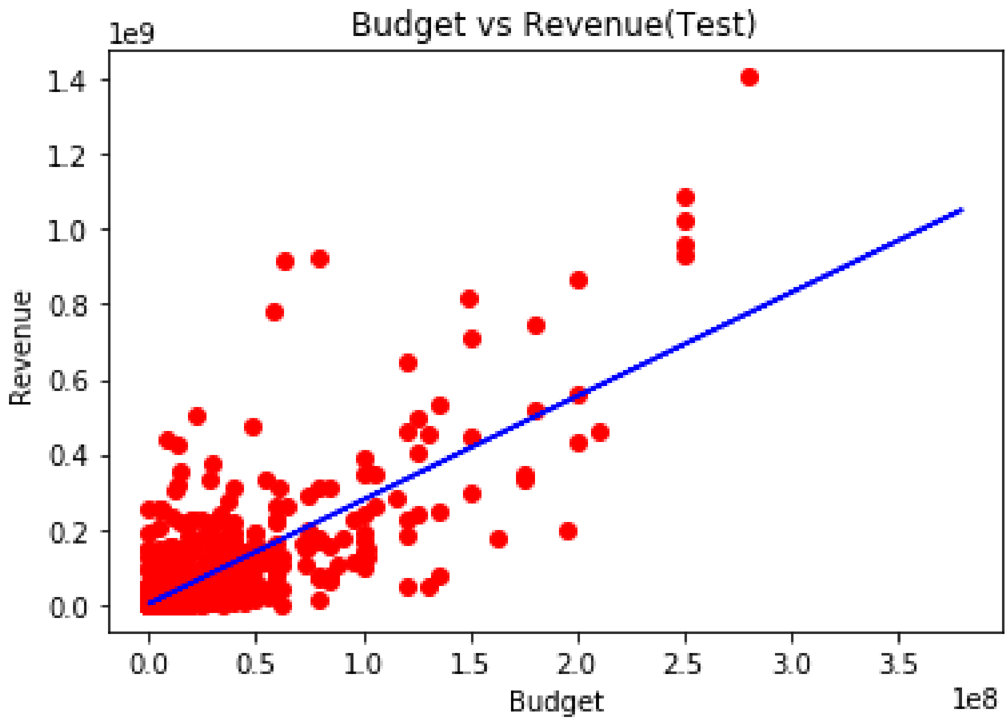
Out[26]: 0.5555010404265975

We have an r-squared score of 0.55 which is pretty decent. It's not close to 0 and it's not close to 1 so we will continue with out predictions. But first lets plot our results

```
In [27]: #Visualize the Training results
         import matplotlib.pyplot as plt
         plt.scatter(X_train, y_train, color = 'red')
         plt.plot(X_train, regressor.predict(X_train), color = 'blue')
         plt.title("Budget vs Revenue(Training)")
         plt.xlabel("Budget")
         plt.ylabel("Revenue")
         plt.show()
```




```
In [28]: #Visualize the Test results
         plt.scatter(X_test, y_test, color = 'red')
         plt.plot(X_train, regressor.predict(X_train), color = 'blue')
         plt.title("Budget vs Revenue(Test)")
         plt.xlabel("Budget")
         plt.ylabel("Revenue")
         plt.show()
```



Our visualizations seem to match our predictions for r-squared. However, you can not determine the variability by observing solely the plots. We'll use a statistical model summary to make sure everything is perfect

```
In [29]: #Model Summary
         X = sm.add_constant(X)

         my_model = sm.OLS(y, X)
         result = my_model.fit()

         print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                revenue   R-squared:                       0.567
Model:                            OLS   Adj. R-squared:                  0.567
Method:                 Least Squares   F-statistic:                     3925.
Date:                Mon, 25 Mar 2019   Prob (F-statistic):               0.00
Time:                        14:53:12   Log-Likelihood:                -59219.
No. Observations:                3000   AIC:                         1.184e+05
Df Residuals:                    2998   BIC:                         1.185e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
const        3.709e+06   1.93e+06      1.917      0.055   -8.47e+04     7.5e+06
budget          2.7969      0.045     62.650      0.000       2.709       2.884
==============================================================================
Omnibus:                     2148.524   Durbin-Watson:                   2.042
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            68600.148
Skew:                           3.006   Prob(JB):                         0.00
Kurtosis:                      25.642   Cond. No.                     5.07e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.07e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

The summary above is a bit overwhelming but we'll only look at a couple of things. P-value seems to be very low which indicates strong evidence against the null hypothesis(in simpler terms, our variables used and results are significant.) We can go on to look at another feature of this summary which is the R-squared. The R-squared is similar to the r-squared I calculated so I'm satisfied. Let's move on to making actual predictions on the Test set

In [30]:
```python
#Making Predictions on the Test set(Original Test set)
X_X = test["budget"].to_frame()
test_y_pred = regressor.predict(X_X)
```

In the code above the object X_X refers to the budget variable of the "test.csv" file. I am using the X_X object to make predictions for the revenue in the test file. I've used my regressor object to predict the revenues of the film

In [32]:
```python
#Export our results to a submission .csv file
submission_file = pd.read_csv('sample_submission.csv')
submission_file['revenue'] = test_y_pred
submission_file.to_csv('submission_file.csv', index=False)
submission_file.head()
```

Out[32]:

	id	revenue
0	3001	3.004280e+06
1	3002	3.246774e+06
2	3003	3.004280e+06
3	3004	2.174244e+07
4	3005	8.515504e+06

And here we go. We have successfully made predictions for movie revenue based on budget. The code above replaces the values in the "revenue" variable of the sample_submission file with our predicted values from test_y_pred