➢ Team Members Name:

Pruthvi Prajapati
Kishan Shah
Amar shah

➢ Project Title:

Bike Sharing Demand

➢ Project Description:

We have hourly rental data spanning two years. For this competition, the training set is comprised of the first 19 days of each month, while the test set is the 20th to the end of the month. Our goal is to predict the total count of bikes rented during each hour covered by the test set based on time, weather, holidays, working days, temperature, using only information available prior to the rental period.

To get best and accurate future prediction for result, we developed a machine learning algorithm that can automatically categorize listings with best prediction. In our input data set we have Un-normalized the data like some of that are in hours, 0 and 1 binary number, decimal number. We use preprocessing.scale() function to make my data in normalized form. Finally, we use a Linear Regression and Random forest to classify the features and get output of a predicted data.

All data processing and machine learning was executed using algorithms in scikit-learn. Plots and curve were generated by importing matplotlib.

➢ Data Fields:

time - date and hourly time
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

➢ Project method and algorithms:

Initially, Read the dataset file “train.csv” and “test.csv” and assign it to a Pandas DataFrame. After that generate the feature matrix and label vector. As seen in Data fields above, the date and time of each sample was given as one feature which made the feature difficult to interpret and incorporate in a model. After performing feature matrix, all datatime were converted into feature vectors based on only hour, which formed the rows of the feature matrix hour.

```
dt = pd.to_datetime(train["datetime"])
train["hour"] = dt.map(lambda x: x.hour)
train.head()
```

This made the total number of features 13 - making it easier to integrate the features into the model. In both data set Hours were in 24 hours to make hour into categorical to numeric form we make hour in 0 to 3. 0- Morning time, 1- Afternoon time, 2- Evening time 3- Night time.

```
def categorical_to_numeric(x):
 if 0 <=  x < 6:
        return 0
elif 6 <= x < 13:
        return 1
elif 13 <= x < 19:
        return 2
 elif 19 <= x < 24:
        return 3
train['hour'] = train['hour'].apply(categorical_to_numeric)
train.head()
```

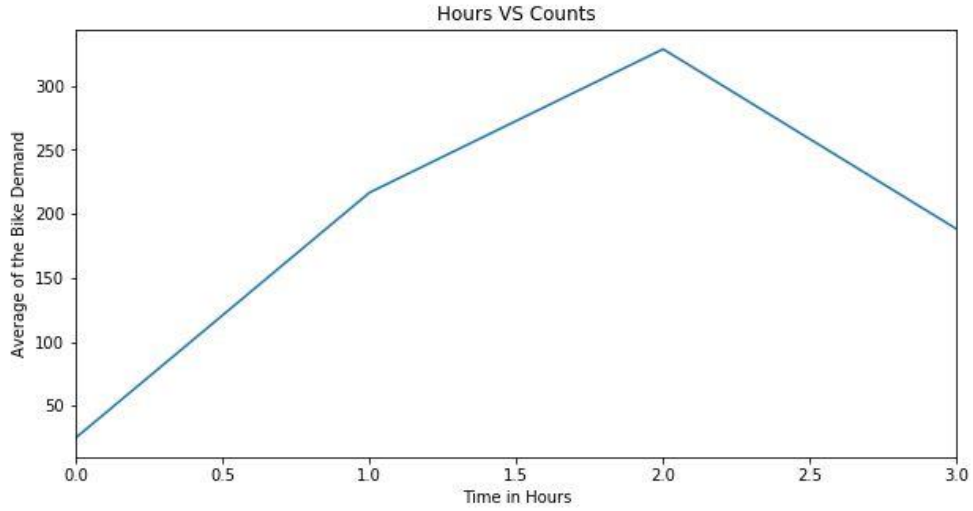
➢ **Table 1: Output After set hour in 0 to 3**

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	hour
0	1/1/2011 0:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16	0
10	1/1/2011 10:00	1	0	0	1	15.58	19.695	76	16.9979	12	24	36	1
20	1/1/2011 20:00	1	0	0	2	16.40	20.455	87	16.9979	11	25	36	3
30	1/2/2011 7:00	1	0	0	2	16.40	20.455	76	12.9980	0	1	1	1
40	1/2/2011 17:00	1	0	0	1	13.94	16.665	57	12.9980	7	58	65	2
50	1/3/2011 5:00	1	0	1	1	6.56	6.820	47	19.0012	0	3	3	0
60	1/3/2011 15:00	1	0	1	1	10.66	12.120	30	16.9979	14	58	72	2

By importing matplotlib function and using line graph to represent the data for Aggregations (reductions) of finding mean based on Hour and count.

```
figure,axes = plt.subplots(figsize = (10, 5))
hours = train.groupby(["hour"]).agg("mean")["count"]
hours.plot(kind="line", ax=axes)
plt.title('Hours VS Counts')
axes.set_xlabel('Time in Hours')
axes.set_ylabel('Average of the Bike Demand')
plt.show()
```

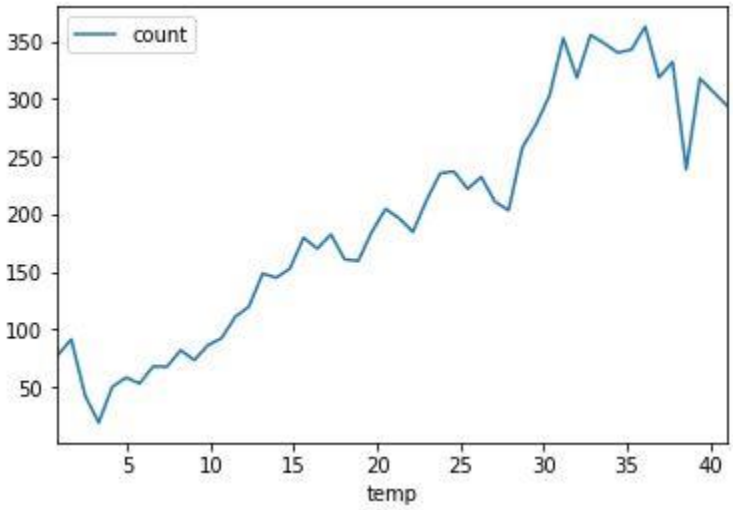
- **Representing data in Hours VS Counts**



In next we group the data by temperature and count for mean() function.

```
a = train.groupby('temp')[['count']].mean()
a.plot()
plt.show()
```

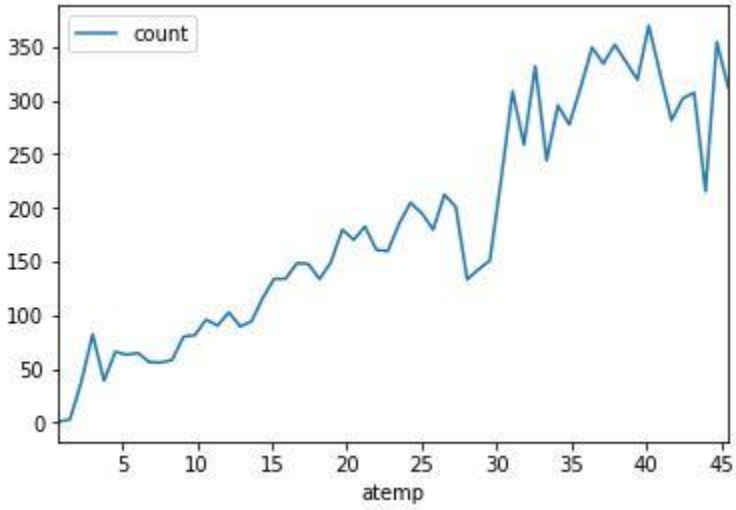
- **Group the data by temperature and count**



In next we group the data by atemp and count for mean() function.

```
a = train.groupby('atemp')[['count']].mean()
a.plot()
plt.show()
```

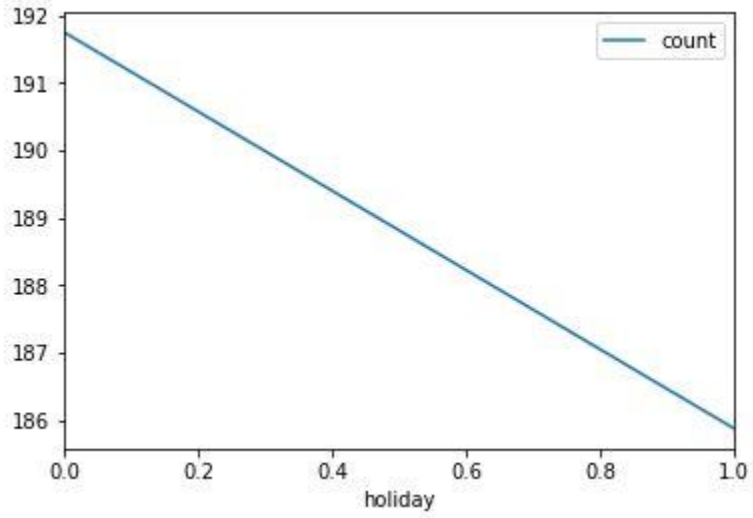
- **Group the data by atemp and count**



In next we group the data by holiday and count for mean() function.

```
a = train.groupby('holiday')[['count']].mean()
a.plot()
plt.show()
```

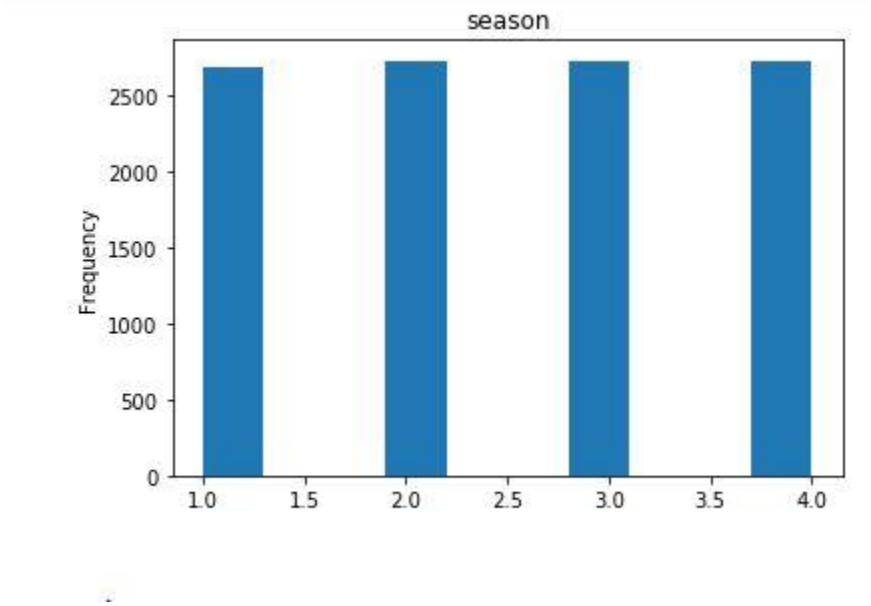
- **Group the data by holiday and count**

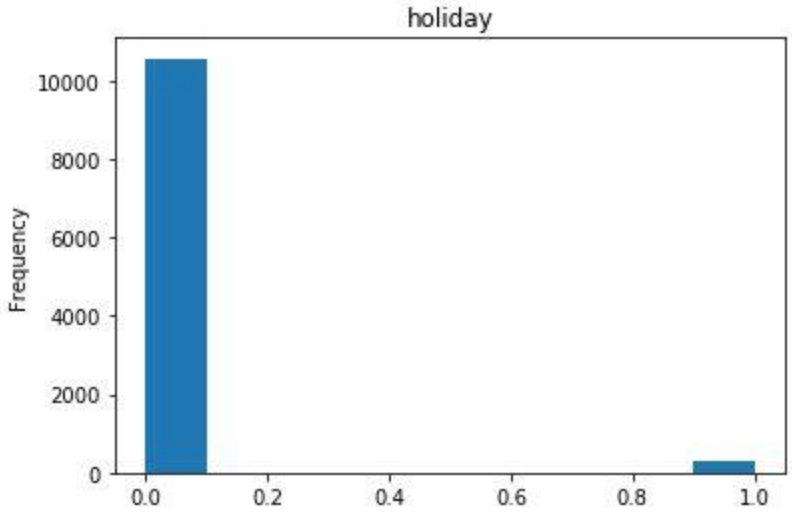


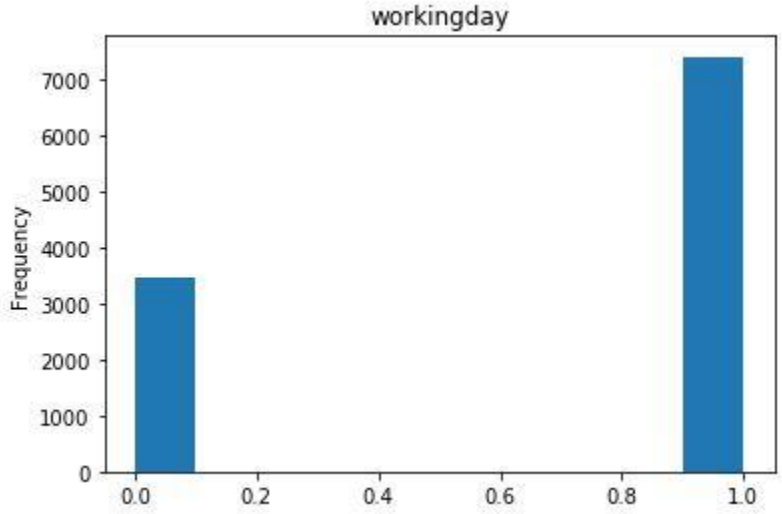
To represent the each of the separate feature matrix list into Histogram plot graph by set index count.

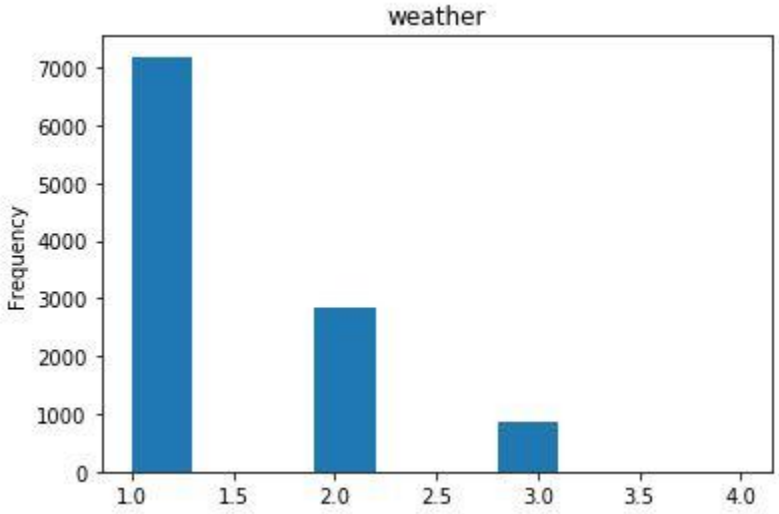
```
time = train.set_index('count')
for f in feature_cols:
        plt.figure()
        plt.title(f)
        time[f].plot(kind='hist')
 plt.show()
```

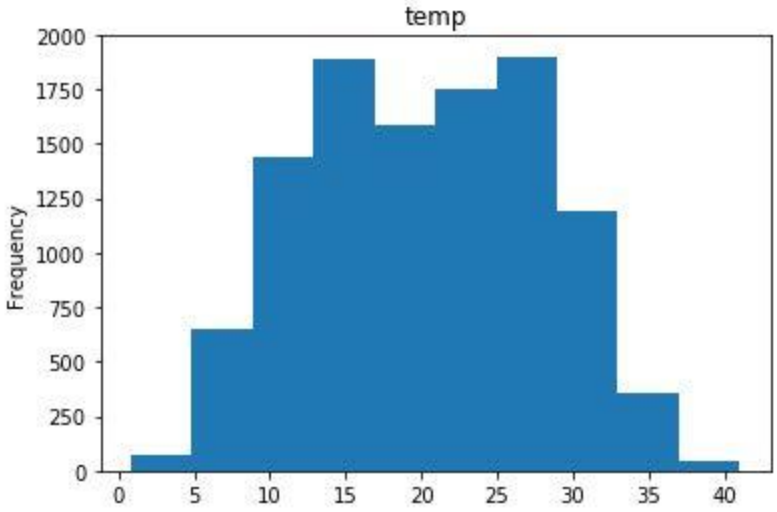
- **Represents a Table 1 into Histogram plot graph**

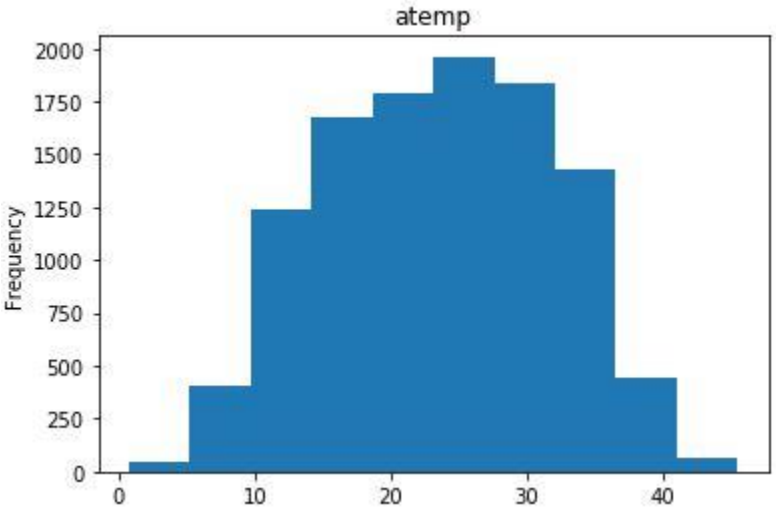


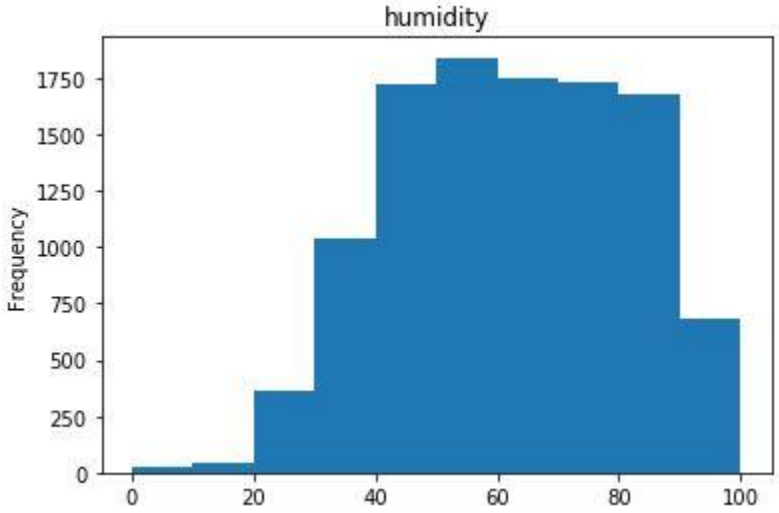

holiday
Frequency
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0

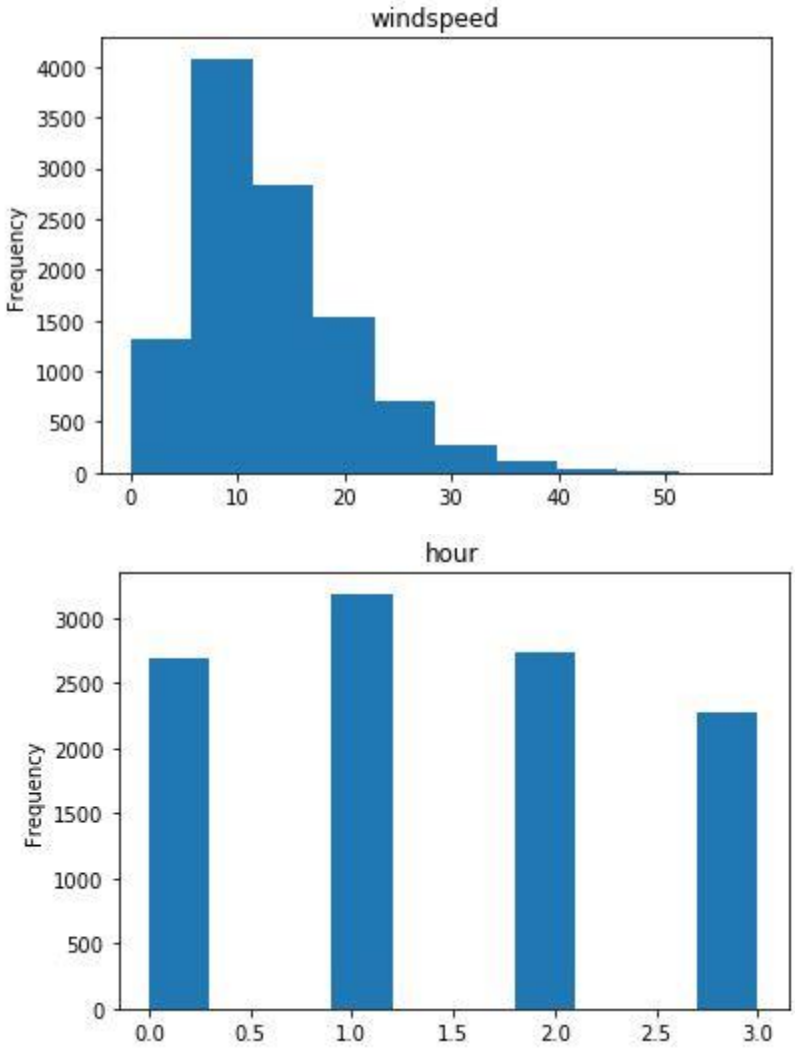

workingday
Frequency
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0


weather
Frequency
7000
6000
5000
4000
3000
2000
1000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0


temp
Frequency
2000
1750
1500
1250
1000
750
500
250
0
0
5
10
15
20
25
30
35
40


atemp
Frequency
2000
1750
1500
1250
1000
750
500
250
0
0
10
20
30
40


humidity
Frequency
1750
1500
1250
1000
750
500
250
0
0
20
40
60
80
100



To scale the feature matrix using preprocessing.scale() function for make our data into same normalized form.

```
X_minmax=scale(X)
X=pd.DataFrame(X_minmax,columns=X.columns)
print(X.head())
```

Before predict out data, we Split our train dataset into testing and training sets with the following parameters: test_size=0.2, random_state=3.

We first applied simple linear regression to the training set. Linear regression was used to attempt to fit the data without any additional feature processing. It was used to minimize the following cost function for which hθ(x) = θ^T x, where hθ(x) is used to generate new predictions.

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

In second we applied Random Forest to the train dataset.
my_RandomForest =RandomForestClassifier(n_estimators = 19, bootstrap = True, random_state=2)

Then we checked co-efficincy for linear regression and as a result we got 192.099213021. We make predictions on the testing set for train data base on linear regression and Random Forest.

- **Output After predict the data for Linear Regression**

```
# make predictions on the testing set for train data
y_prediction = my_linreg.predict(X_test)

print(y_prediction)
```

```
[ 101.73096012  122.4892021   174.17910433 ...,  214.02880076  237.34356929
   33.2184606 ]
```

- **Output After predict the data for Random Forest**

```
y_predict_rf= my_RandomForest.predict(X_test)

print(y_predict_rf)
```

```
[ 53   7  34 ..., 454 125   7]
```

Comparing with my train data set, we also predict our test data set to get a result.

```
# make predictions on the testing set for test data
y_prediction_test =my_RandomForest.predict(X_testData)

print(y_prediction_test)
```

```
[ 3 16 16 ..., 71 97 77]
```

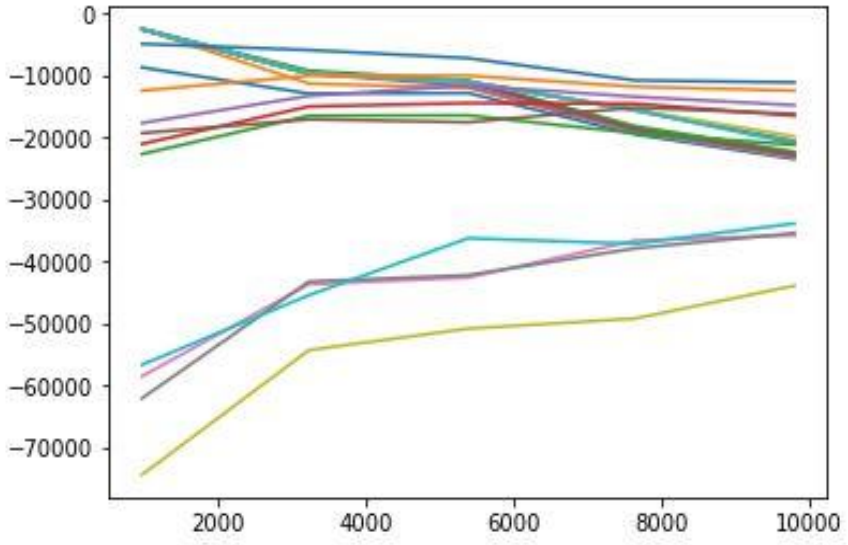
To evaluate our model, we calculated RMSE on a test set. The random predictions result in a test **root mean square error (RMSE) of Linear Regression: 146.854010961** and **Random Forest: 153.398471013.** After calculating the average RMSE as **final result of cross validation: 151.078097048**

- **Plot Learning Curve for Cross validation:**

from sklearn.model_selection import learning_curve

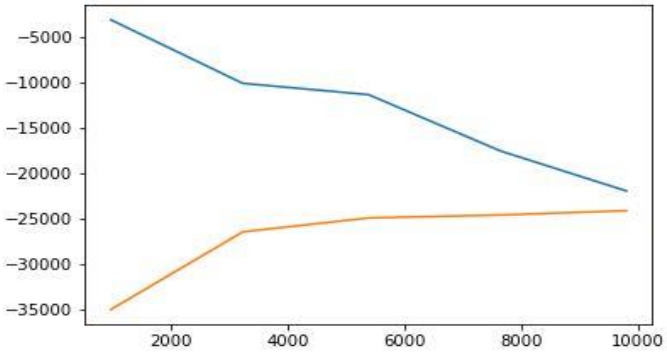
```
train_sizes_abs, train_scores, test_scores = learning_curve(my_linreg ,X, y, n_jobs=-1,cv=10, verbose=0, scoring='neg_mean_squared_error',train_sizes=np.array([ 0.1, 0.33, 0.55, 0.78, 1. ]))
plt.plot(train_sizes_abs, train_scores)
plt.plot(train_sizes_abs, test_scores)
plt.show()
```

- **Output for Plot Learning Curve for Cross validation:**



- **Mean Learning Curves for different Cross Validation Folds**

```
train_score_mean = np.mean(train_scores,axis=1)
test_score_mean = np.mean(test_scores,axis=1)
plt.plot(train_sizes_abs,train_score_mean)
plt.plot(train_sizes_abs, test_score_mean)
plt.show()
```



➢ Results:

	datetime	count
0	1/20/2011 0:00	3
1	1/20/2011 1:00	16
2	1/20/2011 2:00	16
3	1/20/2011 3:00	2
4	1/20/2011 4:00	2
5	1/20/2011 5:00	15
6	1/20/2011 6:00	230
7	1/20/2011 7:00	97
8	1/20/2011 8:00	68
9	1/20/2011 9:00	86

	datetime	count
6483	12/31/2012 14:00	91
6484	12/31/2012 15:00	62
6485	12/31/2012 16:00	82
6486	12/31/2012 17:00	67
6487	12/31/2012 18:00	82
6488	12/31/2012 19:00	71
6489	12/31/2012 20:00	71
6490	12/31/2012 21:00	71
6491	12/31/2012 22:00	97
6492	12/31/2012 23:00	77