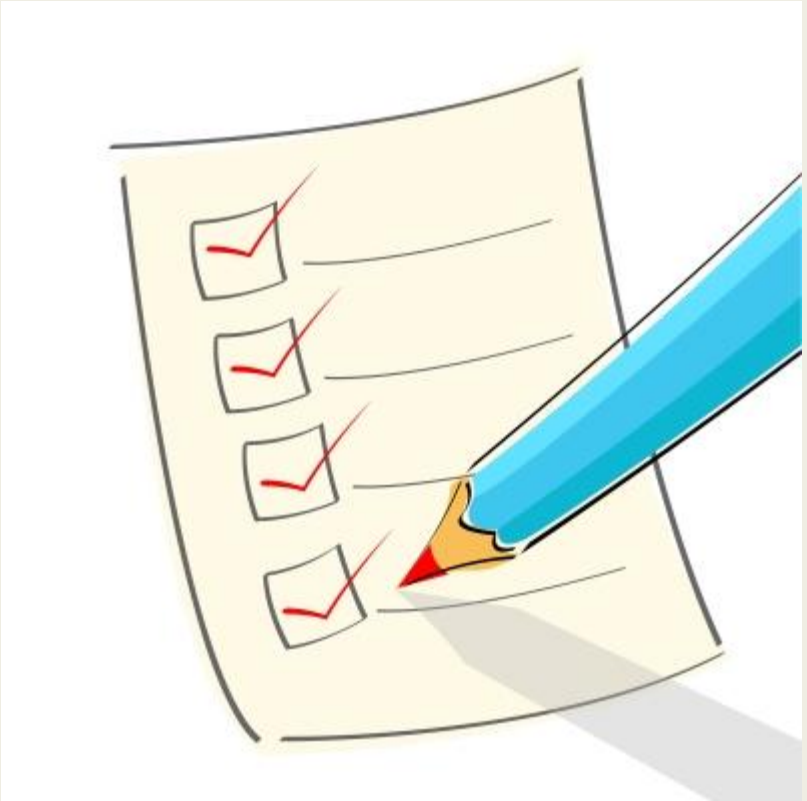
LONDON BIKE SHARING

Capstone Project

Shaakira Kamal, Clement Leo

CONTENTS:

- Intro
- Dataset
- Workflow
- Approach
- Algorithms
- EDA
- Machine Learning
- Challenges



INTRO:

Bike sharing has been around for many years since the 60s.

Over the years of modern advancements and business knowledge, it has evolved drastically.

Allows the masses to overcome high costs and barriers to ownership and other means of travel

Problem Statement:

- To maximize profits by predicting performance factors for the company to focus on.

DATASET:

Data was collated in London over a period of 2 years from January 2015 to January 2017 with hourly record

Has 17414 entries and 10 columns

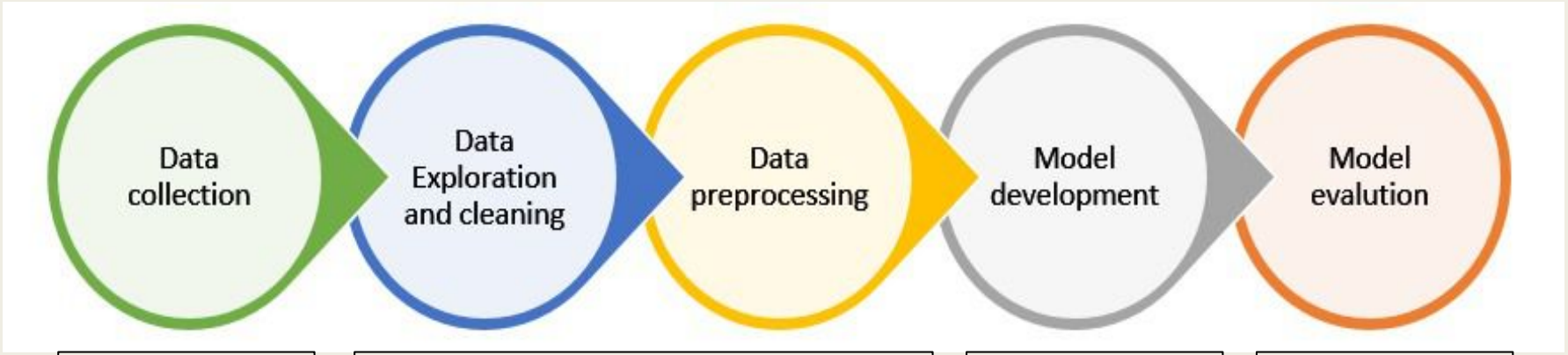
Clean set, no null values

Some of its features are True Temperature, Humidity, Wind Speed, Weekend, Holiday and Season

Source:
https://www.kaggle.com/datasets/hmavrodiev/london-bike-sharing-dataset?select=london_merged.csv

	timestamp	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season
0	2015-01-04 00:00:00	182	3.0	2.0	93.0	6.0	3.0	0.0	1.0	3.0
1	2015-01-04 01:00:00	138	3.0	2.5	93.0	5.0	1.0	0.0	1.0	3.0
2	2015-01-04 02:00:00	134	2.5	2.5	96.5	0.0	1.0	0.0	1.0	3.0
3	2015-01-04 03:00:00	72	2.0	2.0	100.0	0.0	1.0	0.0	1.0	3.0
4	2015-01-04 04:00:00	47	2.0	0.0	93.0	6.5	1.0	0.0	1.0	3.0

WORKFLOW:



Bike Sharing Dataset	Cleaning up, Preprocess & Insights	Use different algorithms	Compare the models

APPROACH:

Time Series Forecasting

- Components of the Dataset:

  - User trends

  The general tendency of the data to increase or decrease during a long period of time.

  - Seasonality

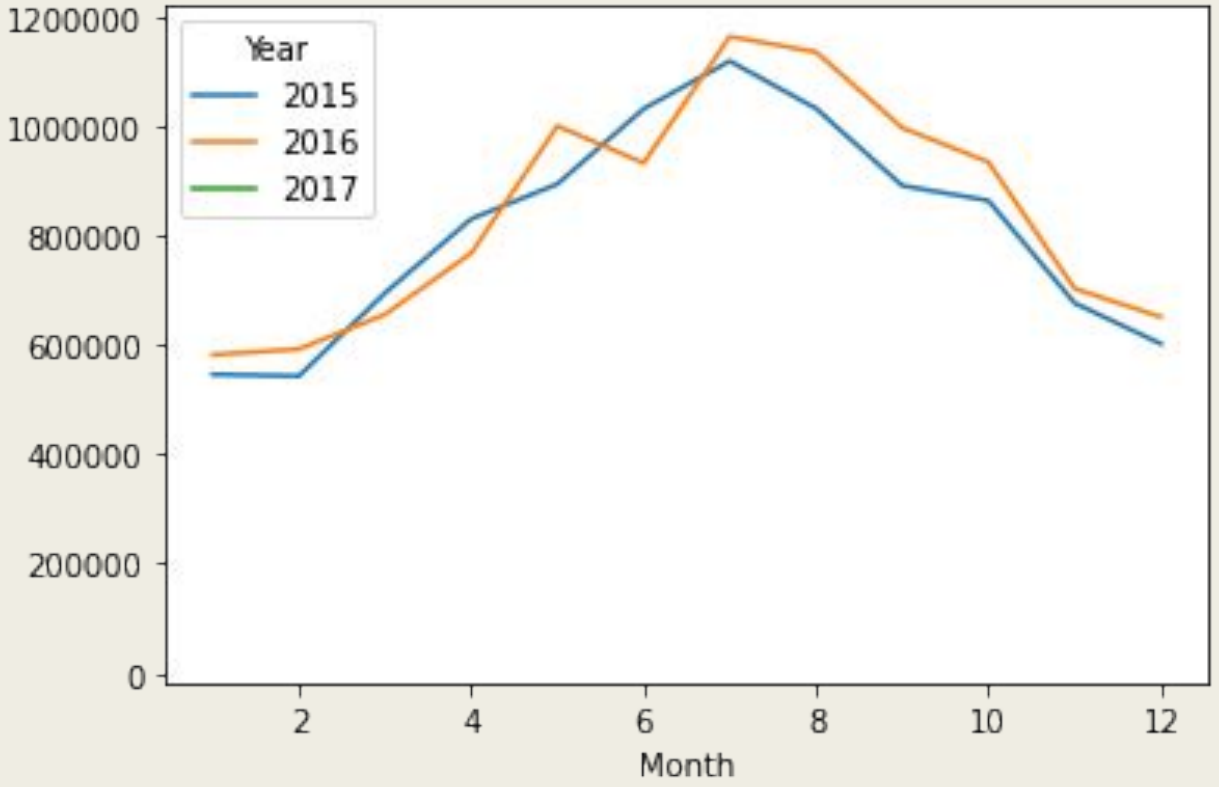
  Repeating pattern within a fixed time period
  [eg: Higher usage during Summer]

- Predicts future values based on previously observed values over a period of time

ALGORITHMS:

LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
XGBOOST	eXtreme Gradient Boosting

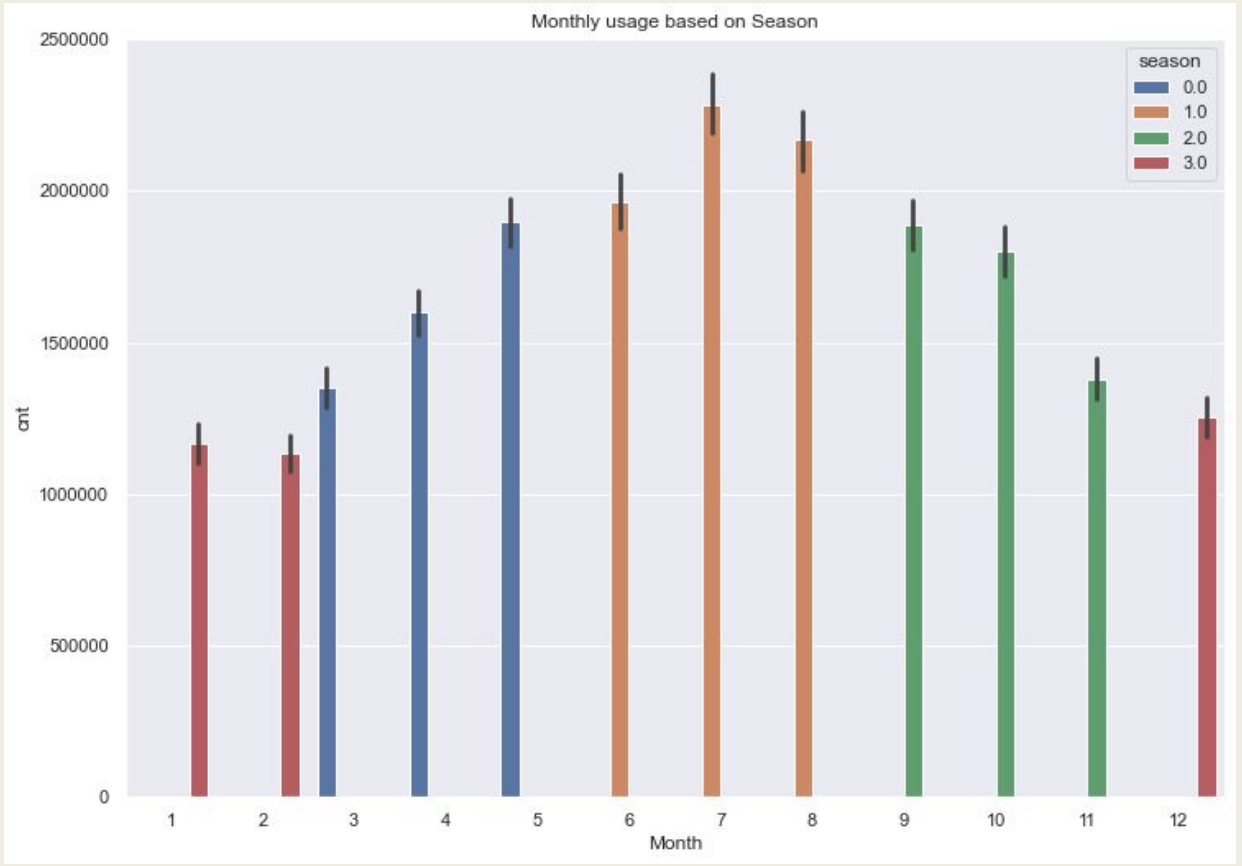
EDA: Yearly Trend



2017 isn’t visible as there were only a few months in it

Similar triangular shape with peaks around the middle of the year

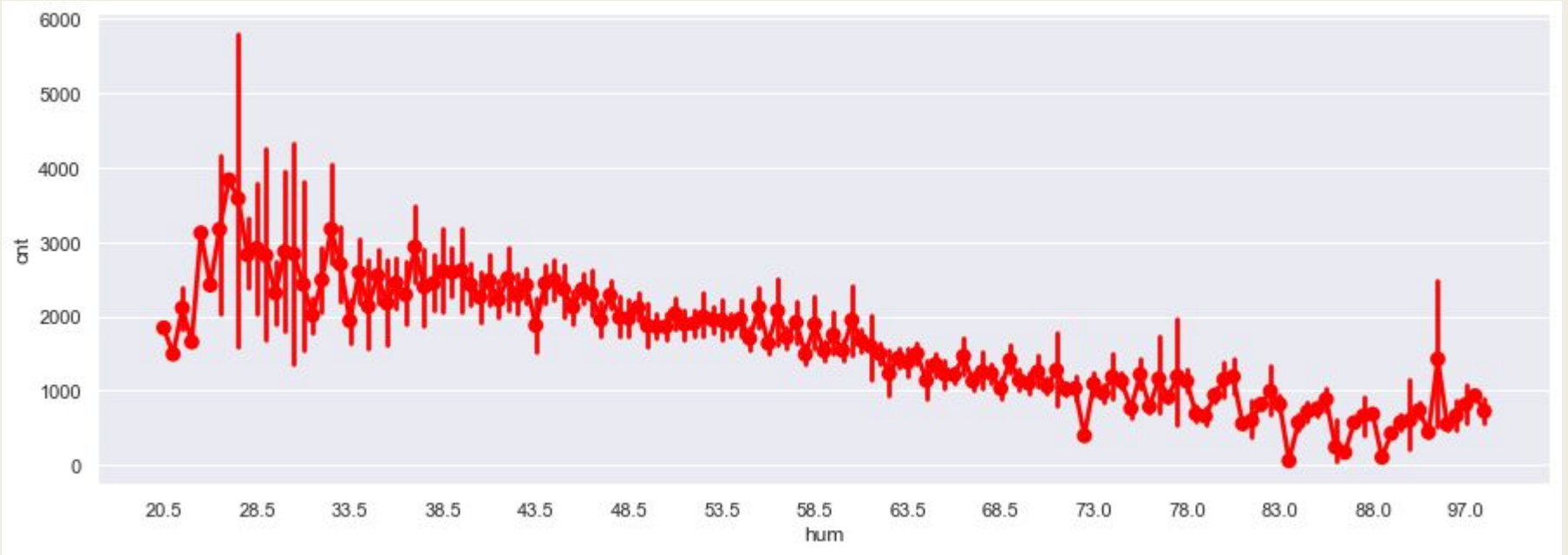
EDA: Seasonal Trends



0 : Spring - 1 : Summer - 2 : Autumn - 3 :Winter

Usage increase towards a peak in Spring and Summer where the humidity and weather the are best, usage decrease during Winter

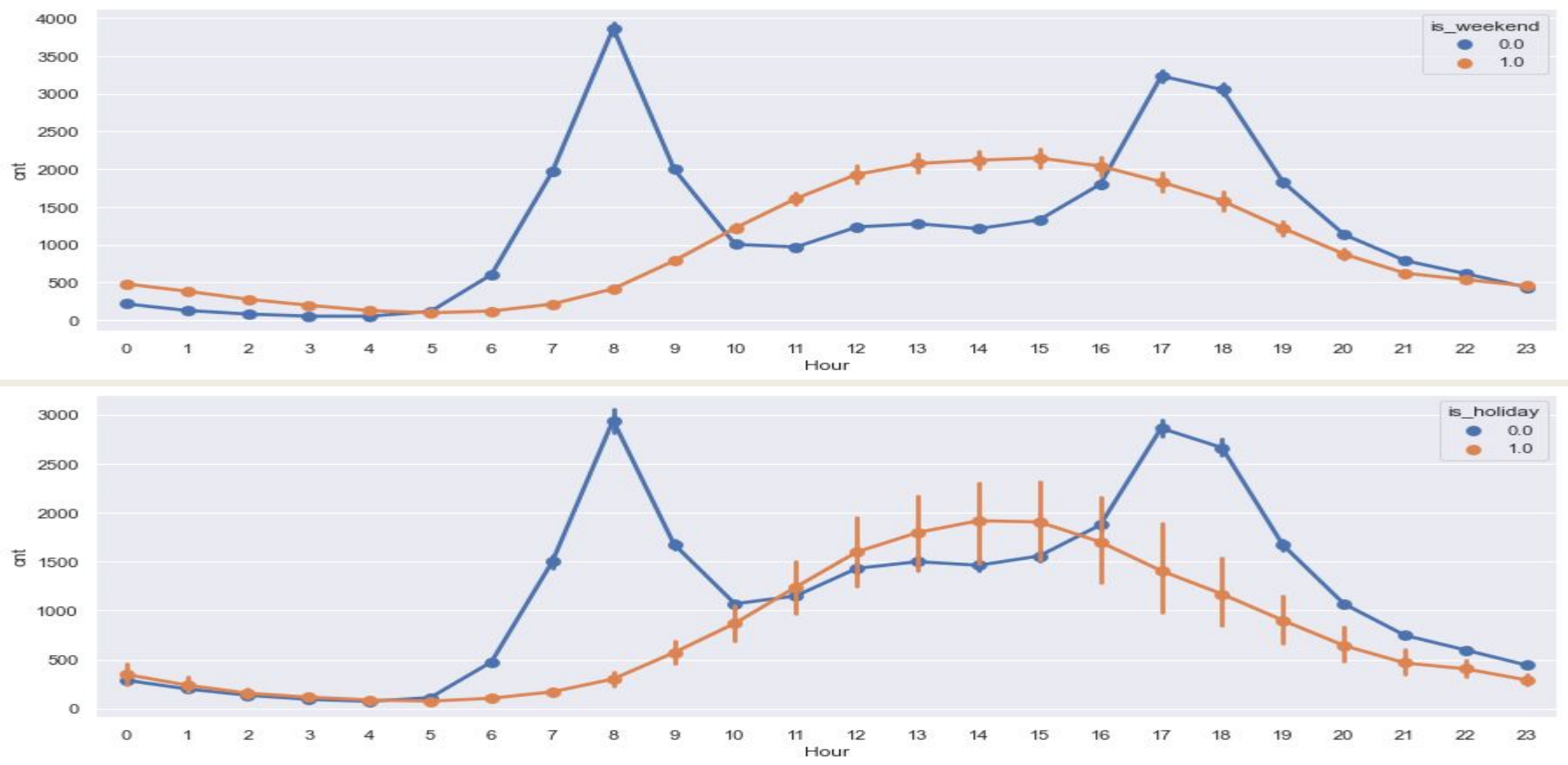
EDA: Humidity



Rentals decline as humidity goes up

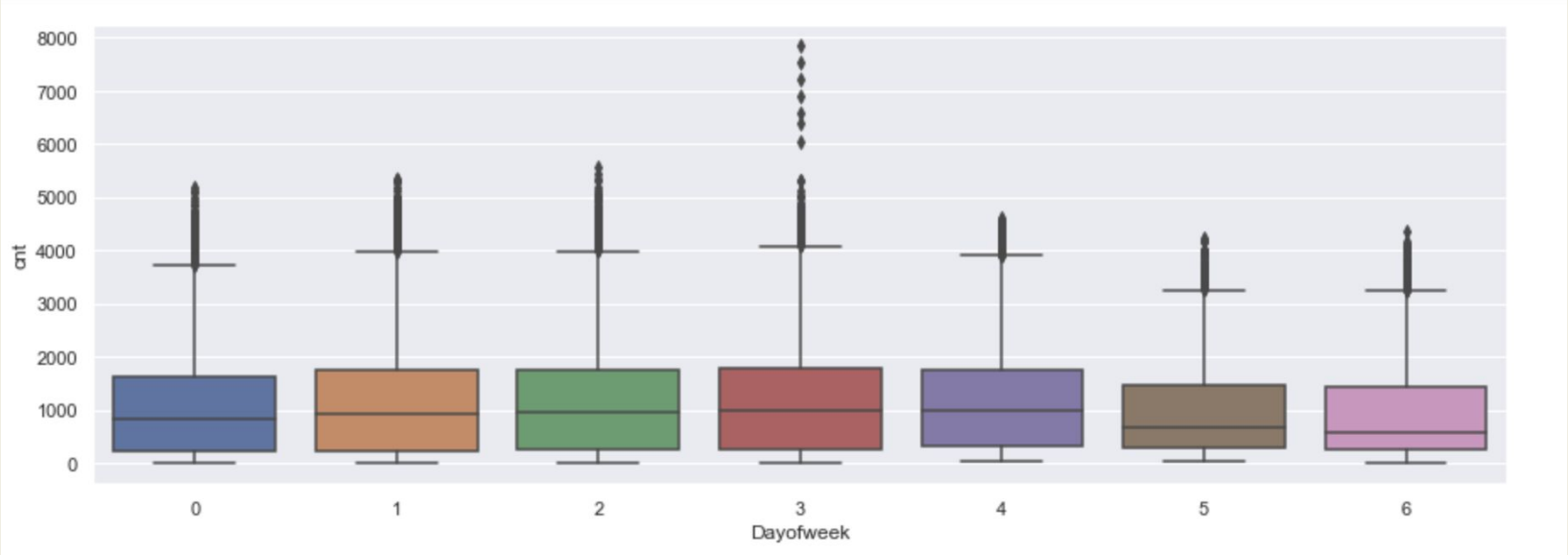
Optimal humidity is estimated between 25-40

EDA: Weekends & Holidays



Weekends & holidays have similar trends as opposed to working days
Peak hours is at 0800 and 1700 - where people start their day and end respectively
As for weekend , people sleep in and have a nice slow morning

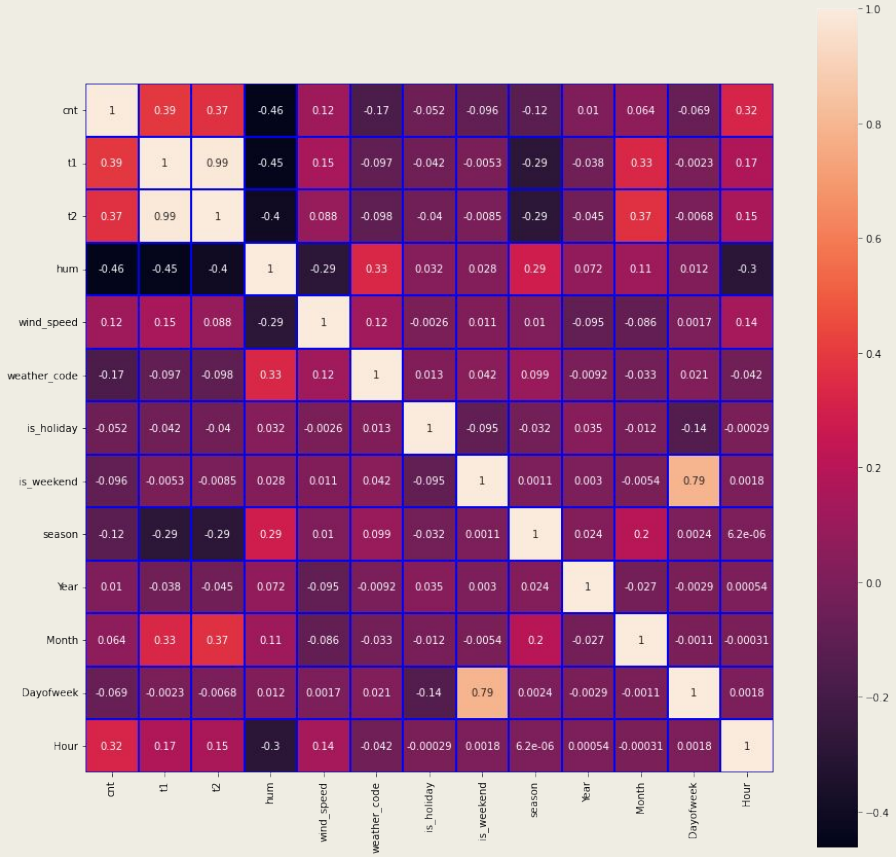
EDA: Weekdays



Mon to Wed has the highest usage with peaks on Wed

Opposite trend as compared to SG

EDA: Correlation Heatmap

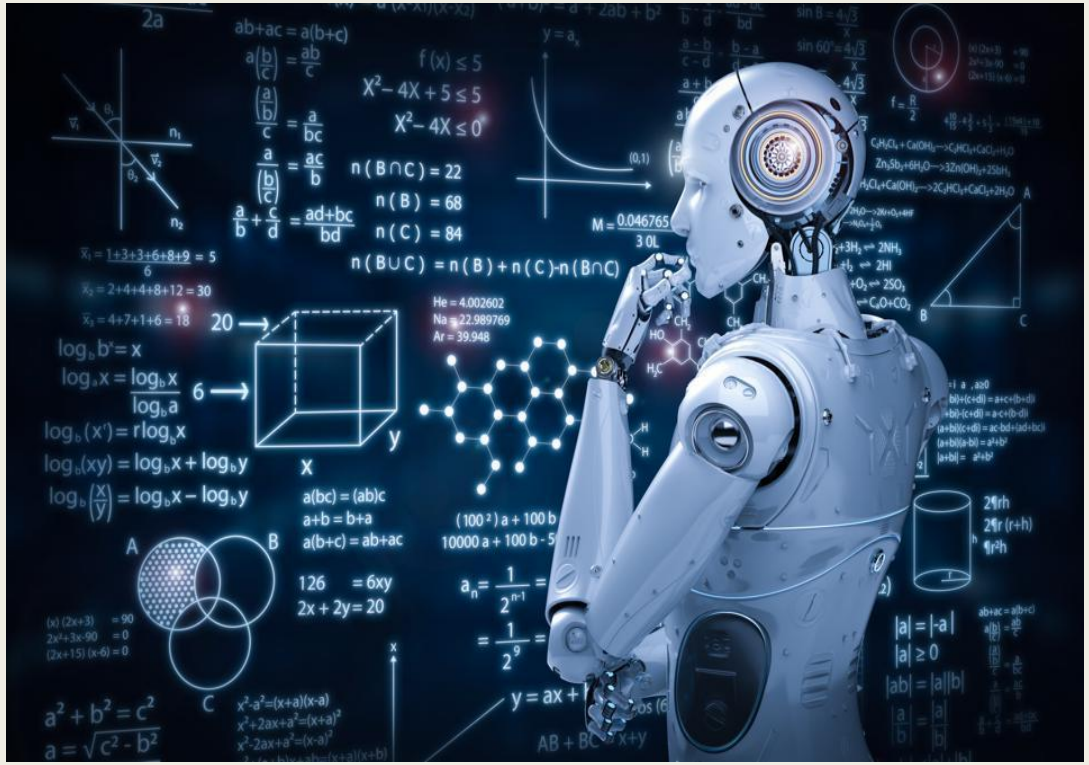


High Correlation: Wind speed, Temperature & Hour

Low Correlation: Humidity & Count

MACHINE LEARNING:

- TRAIN – TEST SPLIT: (90 : 10)
  - Decided on 90:10
  - Tried with 80:20 at first but the result was unsatisfactory
- SCALER :
  - Used Min-Max Scaler but results weren't promising
  - Used Robust Scaler instead as it reduces the effects of outliers.



ML: Preprocessing

- Split into Dates & Times

```
#df[['Date','Time']] = df['timestamp'].str.split(' ', expand=Tr

df["Year"] = df["timestamp"].dt.year
df['Month'] = df["timestamp"].dt.month
df["Dayofweek"] = df["timestamp"].dt.dayofweek
df["Hour"] = df["timestamp"].dt.hour
df.head()
df
```

- Categorize and Encode labels

```
from sklearn.preprocessing import LabelEncoder

# Label Encoding

le = LabelEncoder()

columns = ['weather_code', 'is_holiday', 'is_weekend', 'season',

for col in columns:
    le.fit(df[col])
    df[col] = le.transform(df[col])

df.head()
```

- Train Test Split

```
from sklearn.model_selection import train_test_split

train_size = int(len(df) * 0.9)
test_size = len(df) - train_size
train, test = df.iloc[0:train_size], df.iloc[train_size:len(df)]
print(len(train), len(test))
```

- Scaler

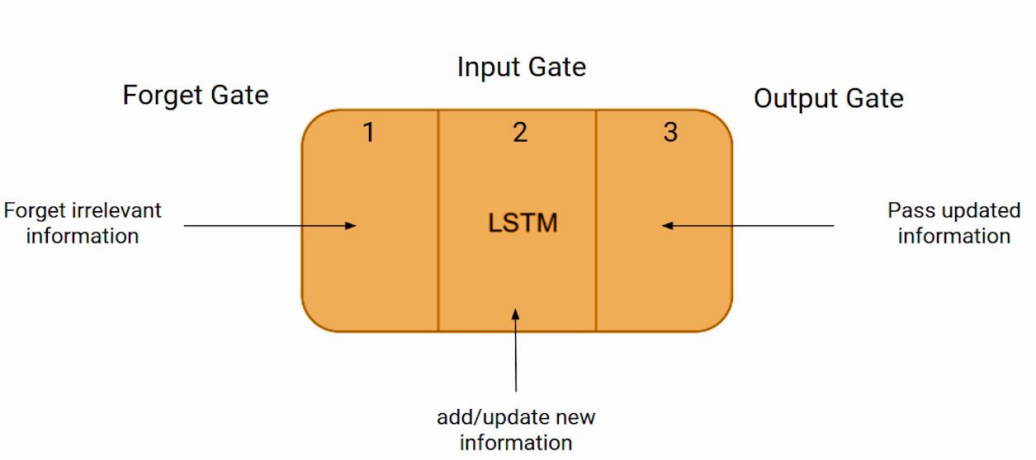
```
from sklearn.preprocessing import RobustScaler
scaler  = RobustScaler()

num_colu = ['t1', 't2', 'hum', 'wind_speed']
trans_1 = scaler.fit(train[num_colu].to_numpy())
train.loc[:,num_colu] = trans_1.transform(train[num_colu].to_num
test.loc[:,num_colu] = trans_1.transform(test[num_colu].to_numpy

scaler2 = RobustScaler()
scaler2 = scaler2.fit(train[["cnt"]])
train["cnt"] = scaler2.transform(train[["cnt"]])
test["cnt"] = scaler2.transform(test[["cnt"]])
```

ML: RNN

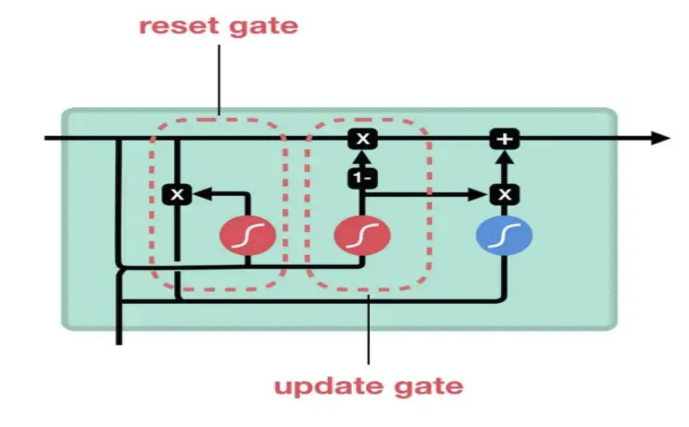
Traditionally, time series forecasting has been dominated by linear methods because they are well understood and effective on many simpler forecasting problems.
Deep neural networks have proved to be powerful and are achieving high accuracy in many application fields. For these reasons, they are one of the most widely used methods of machine learning to solve problems dealing with big data nowadays

Traditional RNN are good for processing sequence data for predictions but suffers from short-term memory hence LSTM and GRU are introduced.

- LSTM



- GRU



LSTM	GRU
Seperate hidden and cell state	Combined hidden and cell state

We are deploying the same method in both LSTM and GRU. There is no concrete evidence of which is better except that it is said LSTM performs better with larger dataset hence slower and uses larger memory.

- Units are set to 128 , Dropout is set to 0.2
- Since the dataset is quite big we have set 50 epochs and a batch of 32 for both

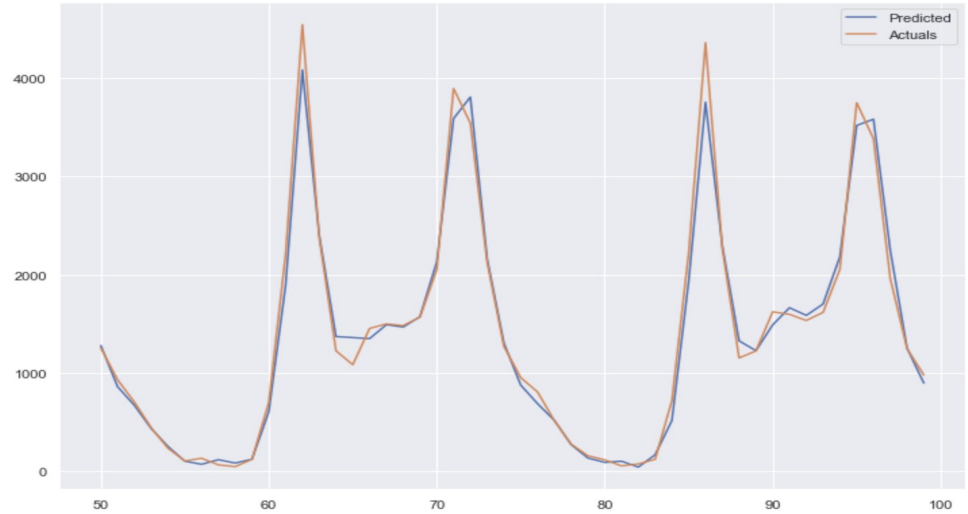
Epoch	Batch
Epoch is the complete passing through of all the datasets exactly at once.	The batch is the dataset that has been divided into smaller parts to be fed into the algorithm.

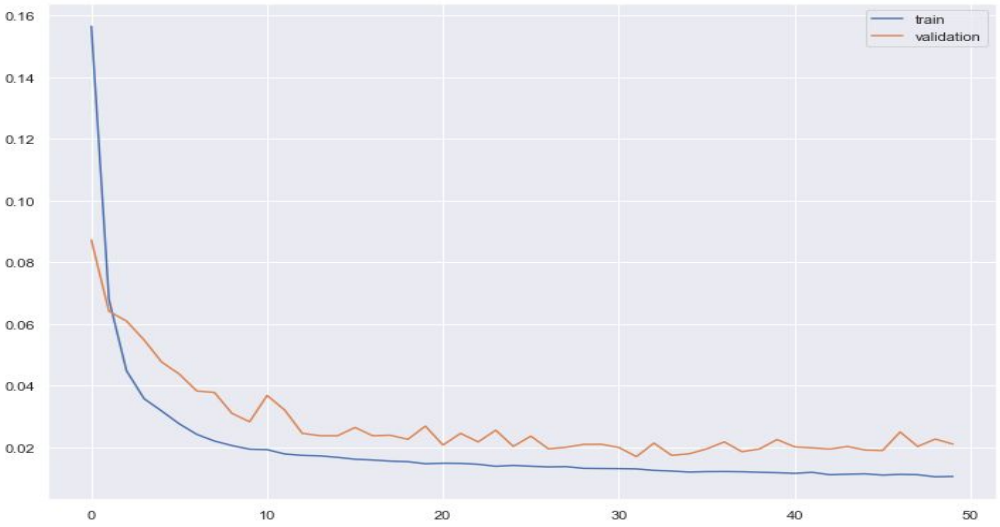
- After running the Machine we will compare the predicted Values and the actual Values looking at the Explained Variance Score (Similar to r2) Mean squared error (MSE) - The larger the number the larger the error, and RMSE
- We would also be looking at the loss , A low value for the loss means our model performed very well.

ML: LSTM

```
plt.plot(train_results['Train Predictions'][50:100])
plt.plot(train_results['Actuals'][50:100])
plt.legend(['Predicted', 'Actuals'])
```

```
<matplotlib.legend.Legend at 0x7fef30199e70>
```






```
lstm_train_pred = model1.predict(x_train)
lstm_test_pred = model1.predict(x_test)

#calculate root mean squared error
print('Train RMSE:', np.sqrt(mean_squared_error(y_train, lstm_train_pred)))
print('Test RMSE:', np.sqrt(mean_squared_error(y_test, lstm_test_pred)))
print('MSE:', mean_squared_error(y_test, train_predictions))
print('Explained variance score:', explained_variance_score(y_test, train_predictions))
```

✓ 1.4s

```
490/490 [==============================] - 1s 2ms/step
55/55 [==============================] - 0s 2ms/step
Train RMSE: 0.11969177346313367
Test RMSE: 0.15634960276049403
MSE: 0.02444519828336428
Explained variance score: 0.940909605485145
```

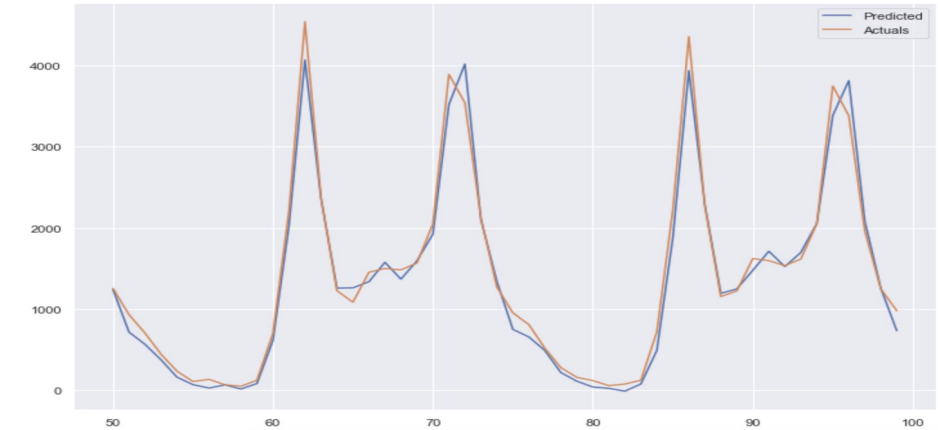
As we can see there is not much difference between The Predicted and The Actuals , We got 0.94 for the explained variance score given the huge param (72704)

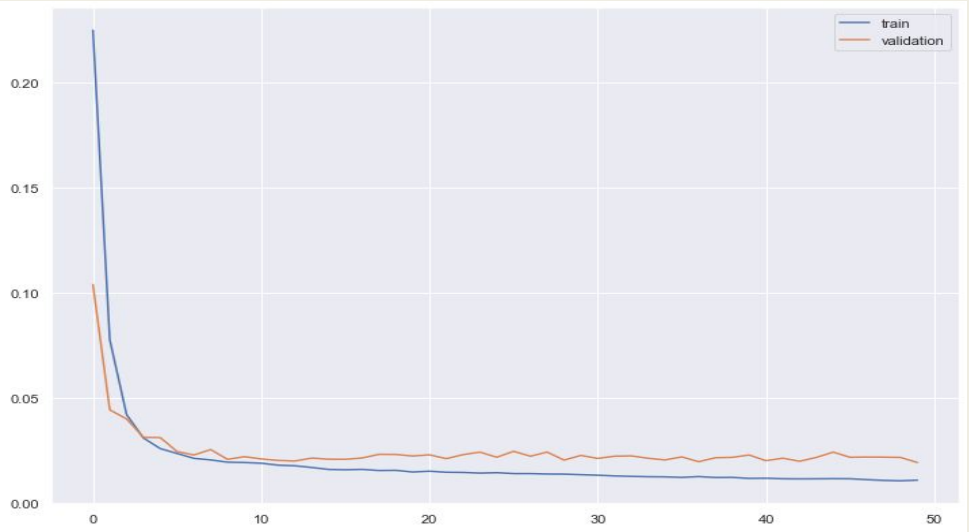
ML: GRU

```
1  plt.plot(train_results['Train Predictions'][50:100])
2  plt.plot(train_results['Actuals'][50:100])
3  plt.legend(['Predicted', 'Actuals'])
4
```

✓ 0.1s

```
<matplotlib.legend.Legend at 0x7fb371f23910>
```






```
1  gru_train_pred = model2.predict(x_train)
2  gru_test_pred = model2.predict(x_test)
3  print('Train RMSE:', np.sqrt(mean_squared_error(y_train, gru_train_pred)))
4  print('Test RMSE:', np.sqrt(mean_squared_error(y_test, gru_test_pred)))
5  print('MSE:', mean_squared_error(y_test, train_predictions2))
6  print('Explained variance score:', explained_variance_score(y_test, train_predictions2))
```

✓ 1.2s

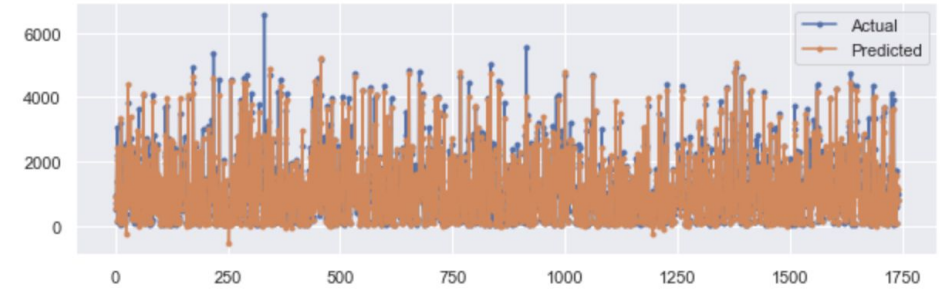
```
490/490 [==============================] - 1s 2ms/step
55/55 [==============================] - 0s 2ms/step
Train RMSE: 0.11052654952868815
Test RMSE: 0.16179669451030615
MSE: 0.026178170354461332
Explained variance score: 0.9373262292981961
```

As we can see it has similar performance to LSTM, but LSTM performed slightly better. It is still not bad if we are looking for less memory consumption and faster operation

ML: XGBOOST

```
1  plt.figure(figsize=(10,3))
2  plt.plot(y_test.values,marker=".",label="actual")
3  plt.plot(train_predictions4,marker=".",label="prediction")
4  plt.legend(['Actual', 'Predicted'])
5  plt.show()
```

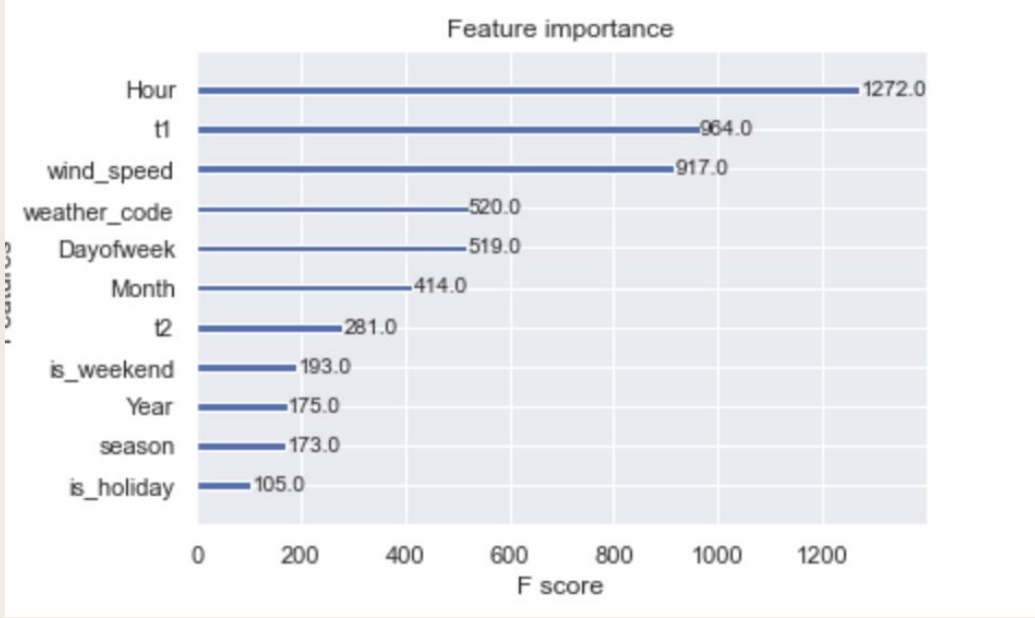
✓ 0.1s



R2: 0.9428885808154956

The performance is similar to LSTM

Hour and Temperature are most important features based on feature importance plot. These 2 features in corrplot also has high correlations with count of Bicycle



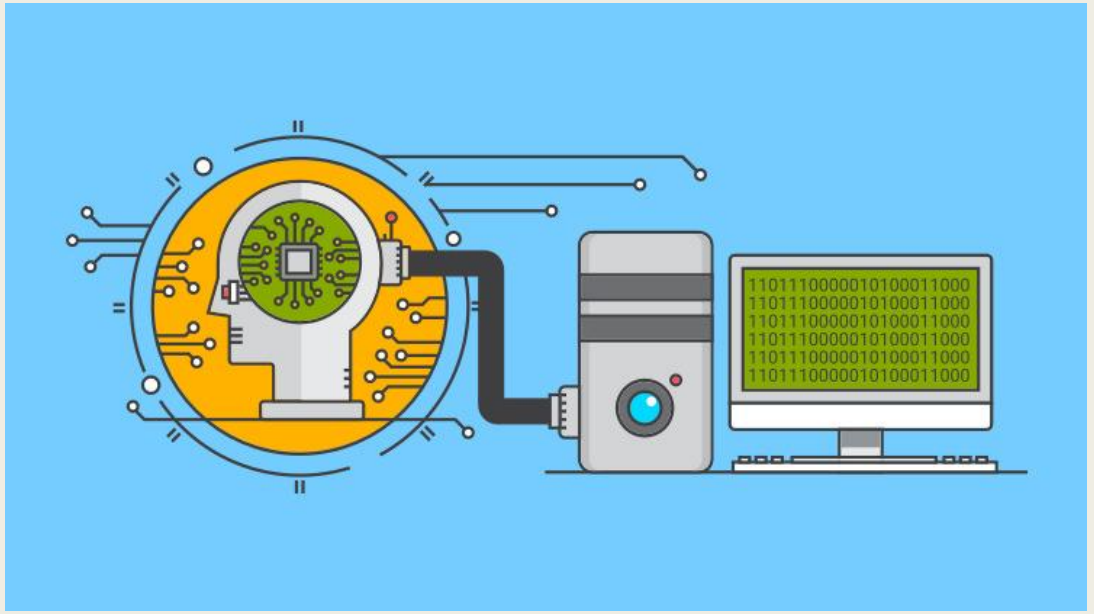
CHALLENGES:

- Understanding the concept
  - Don’t fully understand the maths that is going on behind the scenes

- Run the machine models first
  - To save time and try to understand along the way

- Trial and Errors
  - Tried different inputs many times to achieve a proper working code with satisfactory outcome

- LSTM requires a lot of parameters
  - Larger number of parameters means longer processing and waiting times



The End

https://github.com/Clemykeileo/ML-Project.git

Shaakira Github