Capstone Project-2

Seoul Bike Sharing Demand Prediction

The Bike Sharing Analysis follows :

- Problem Statements
- Data Information
- Analysis of Data
- Data Cleaning/ Imputation
- Data Preparation
- Model Training
- Evaluation Metrics
- Challenges
- Conclusion

Problem Statements

- What can we learn from predictions? (ex: Days, Temperature, seasons,etc).
- Prediction of bike count required at each hour for the stable supply of rental bikes.
- Highest Booking counts in Season, Month and Week.
- Finding Variations in data
- Finding the best estimating algorithm

Data summary

Dataset file : Seoulbikedata.CSV file from Dec2017 to Jan2018

Shape:

- **Columns:14**
- **Rows:8760**

Important Columns and Units

Date
Rented Bike Count
Hour 24units
Temperature (°C)
Humidity (%)
Wind speed (m/s)
Visibility (10m)
Solar Radiation (MJ/m2)
Seasons
Holidays
Functioning Day

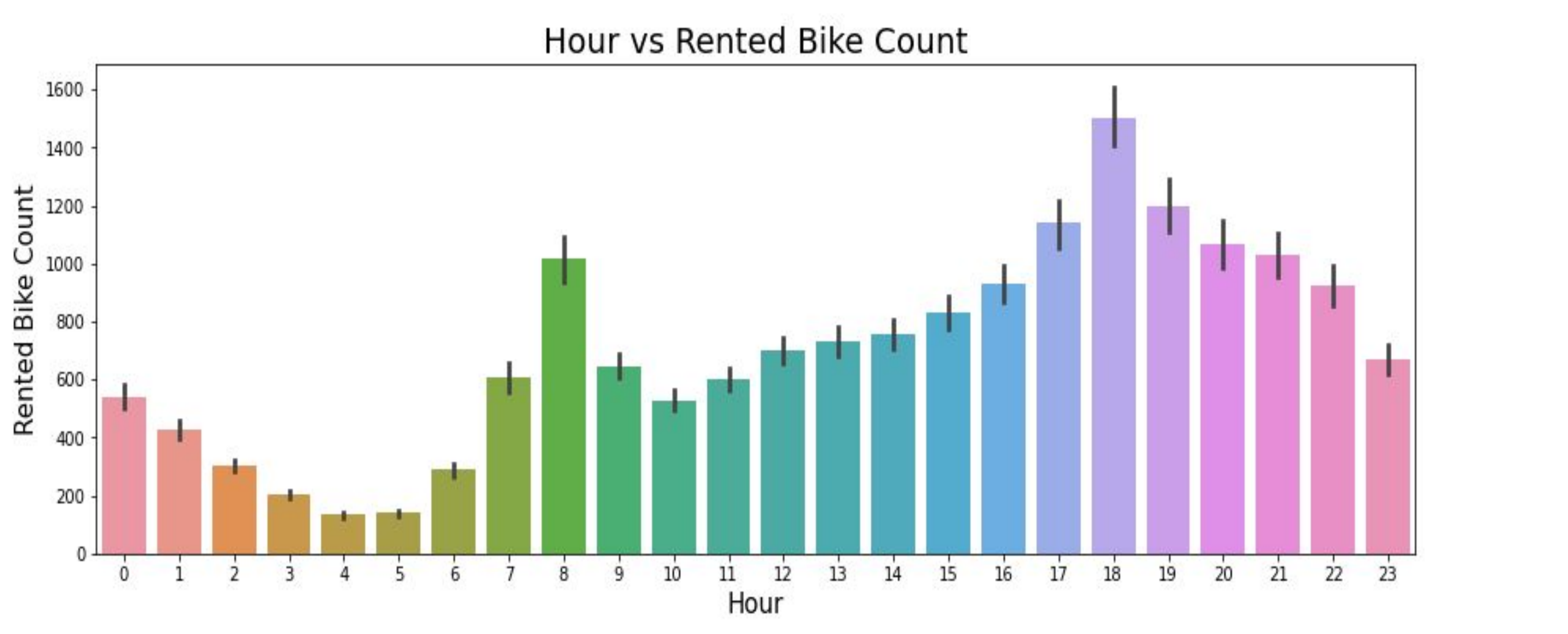
Data Cleaning and imputation

- Checking for Duplication in Data frame columns.
- Checking for Nan/Null Values.


```
# checking for null values
df.isnull().sum()
```

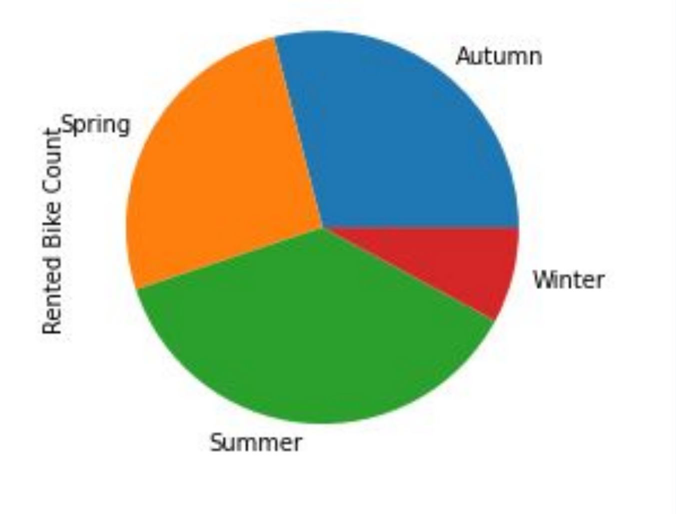
```
Date                         0
Rented Bike Count            0
Hour                         0
Temperature(°C)              0
Humidity(%)                  0
Wind speed (m/s)             0
Visibility (10m)             0
Dew point temperature(°C)    0
Solar Radiation (MJ/m2)      0
Rainfall(mm)                 0
Snowfall (cm)                0
Seasons                      0
Holiday                      0
Functioning Day              0
dtype: int64
```

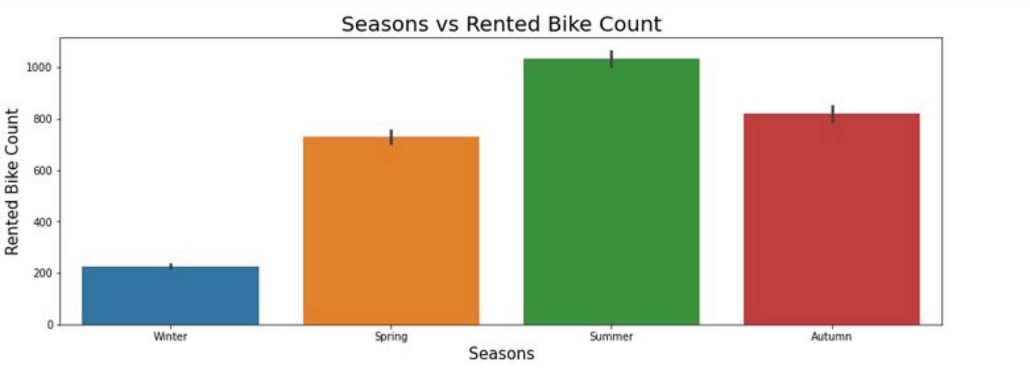
What time in a day is highest bike rented?



Which Season has most bike Rents?

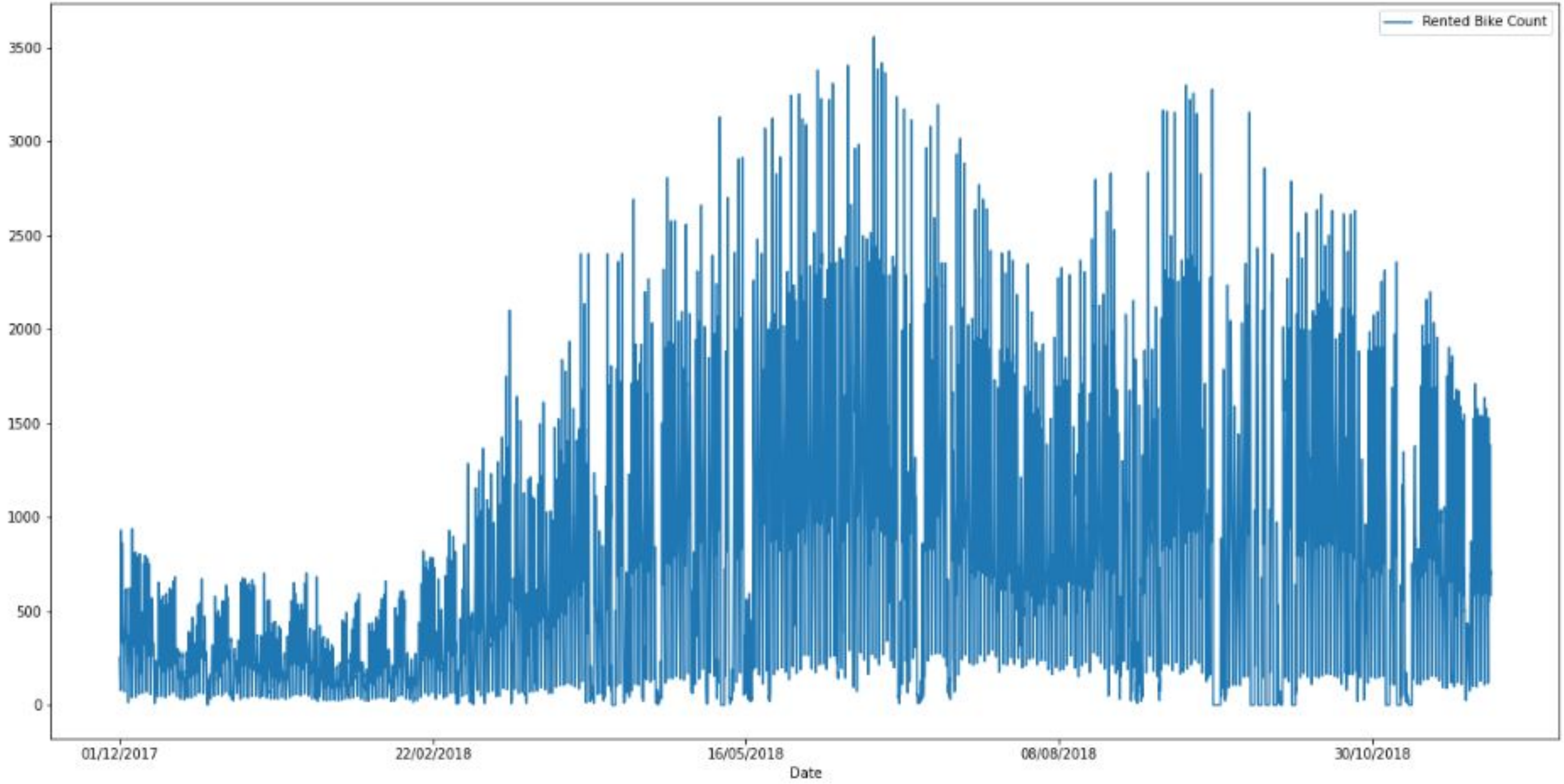
Seasons	Rented Bike Count
Summer	2283234
Autumn	1790002
Spring	1611909
Winter	487169



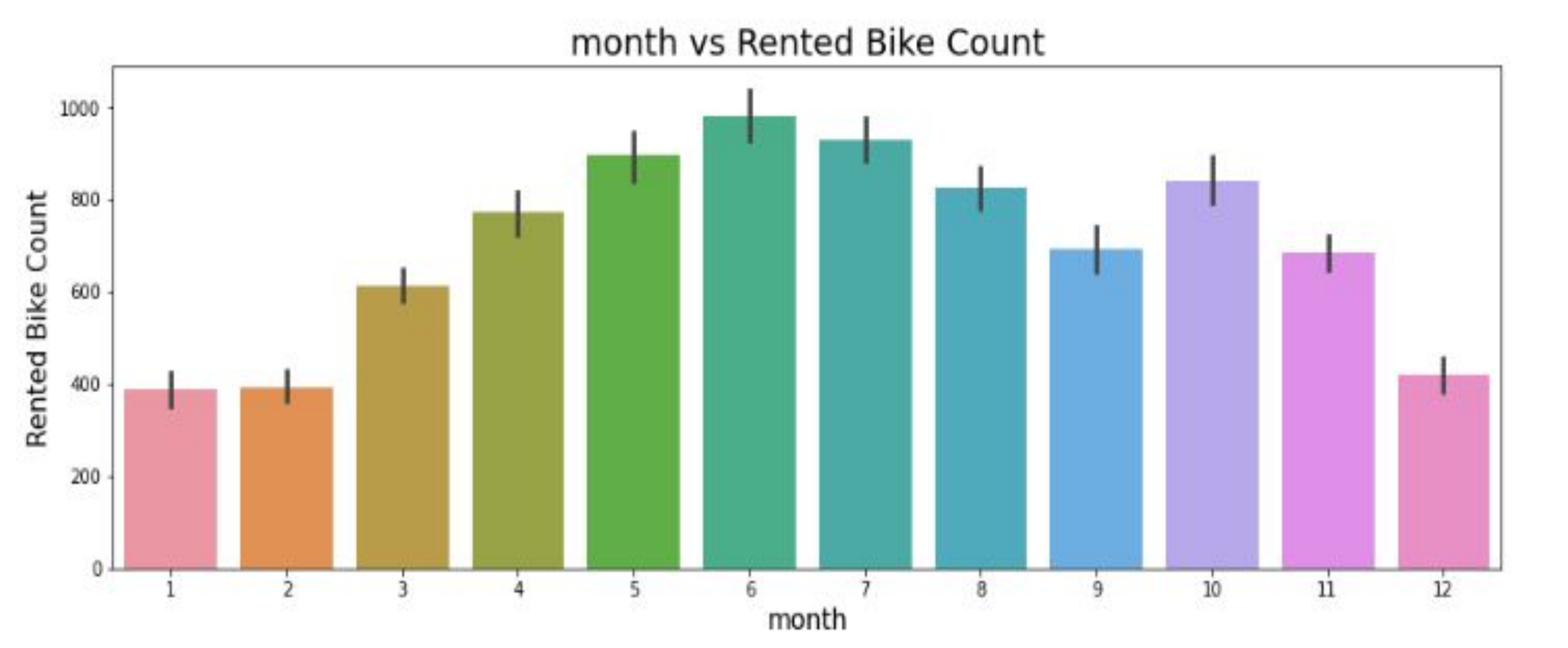


Which Date in a month has highest booking count?

AI

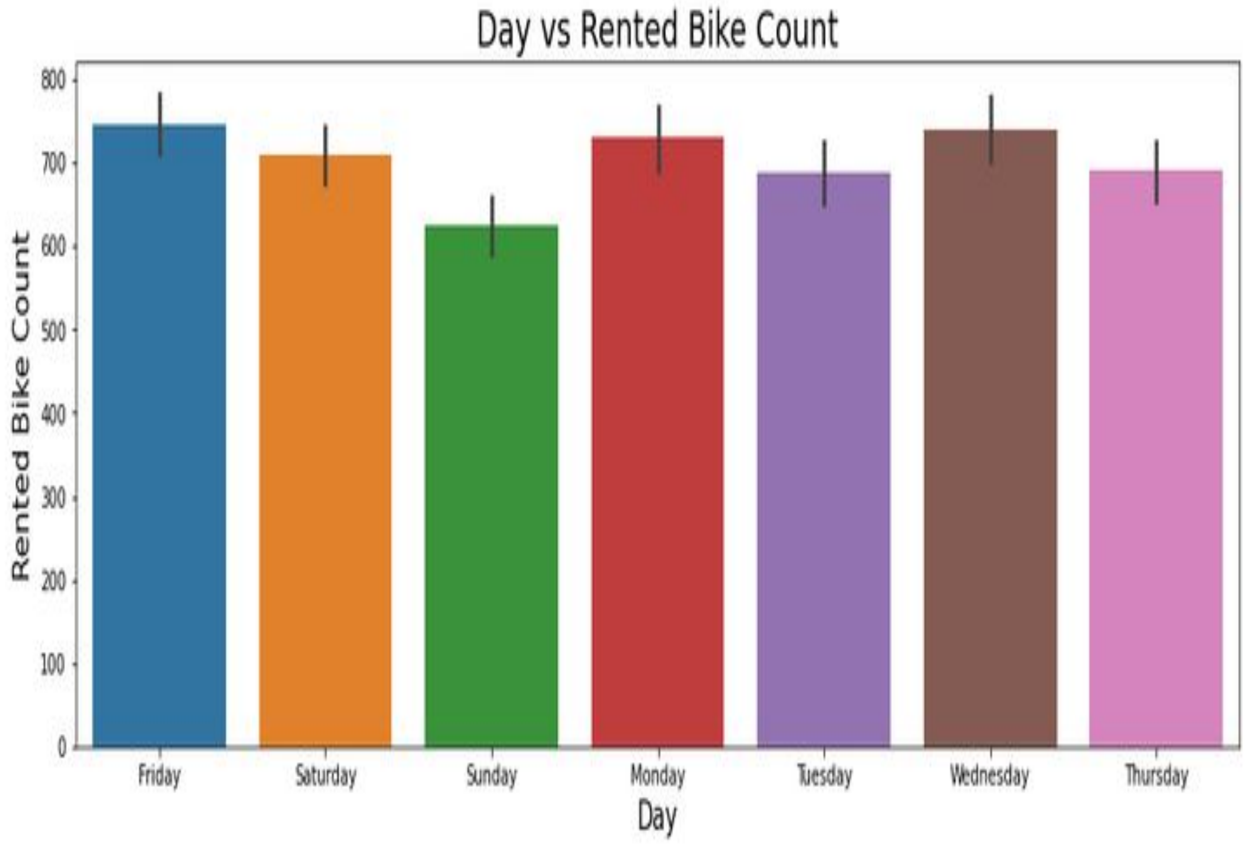


Which Month has highest booking count?

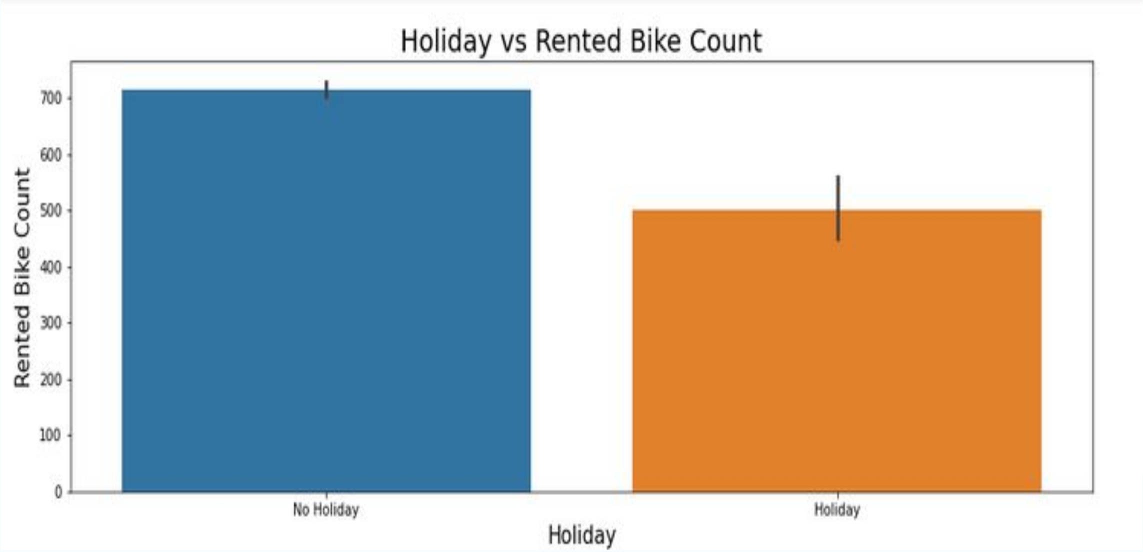


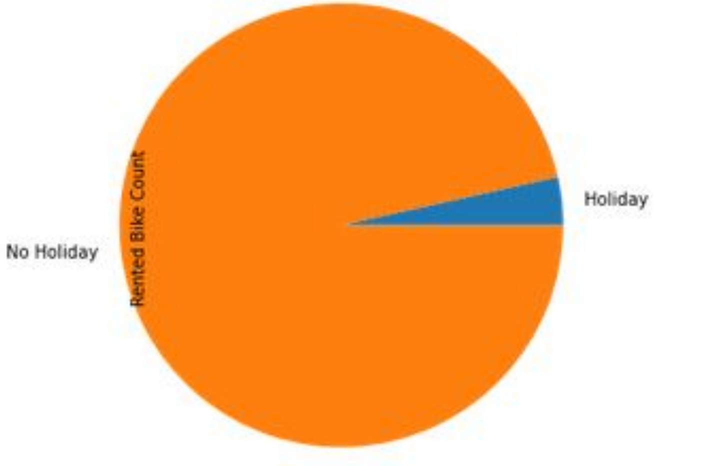
Which day in a week has highest booking?

Day	Rented Bike Count
Friday	950334
Wednesday	923956
Monday	911743
Saturday	885492
Thursday	861999
Tuesday	858596
Sunday	780194

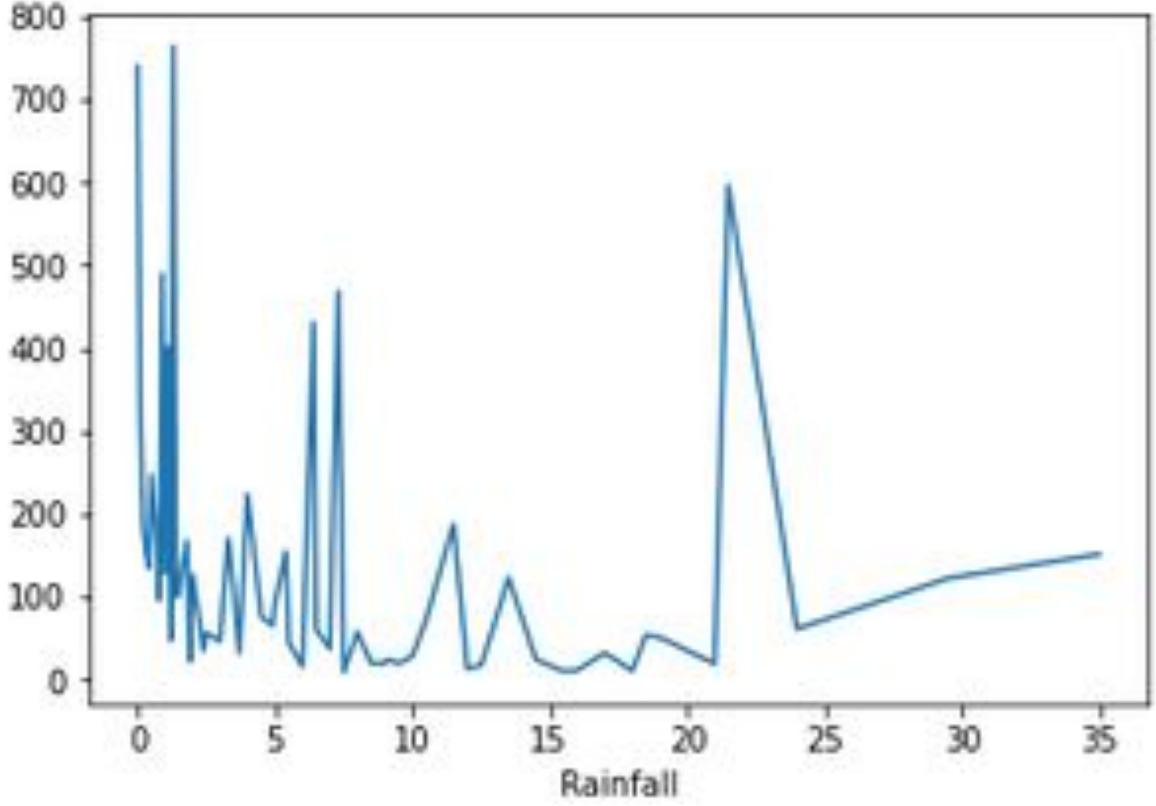


Is there any bookings on Holiday?

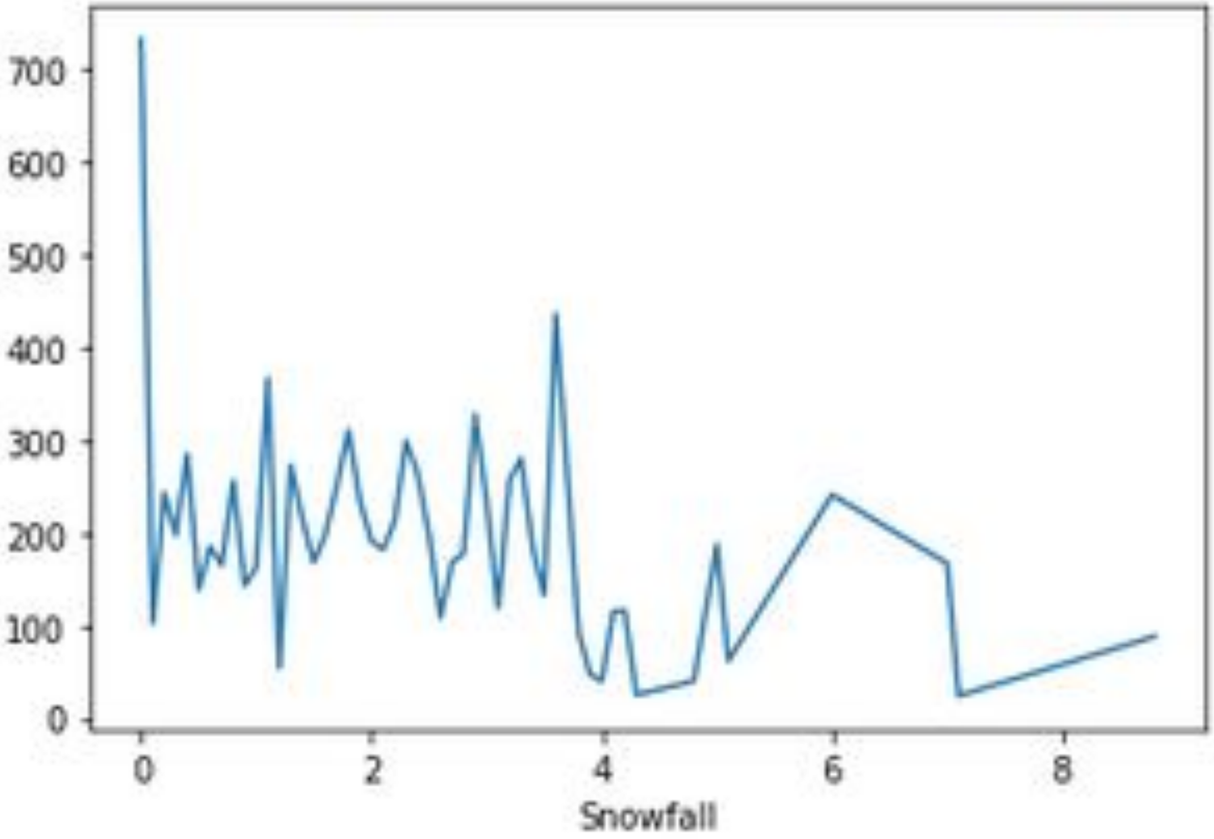




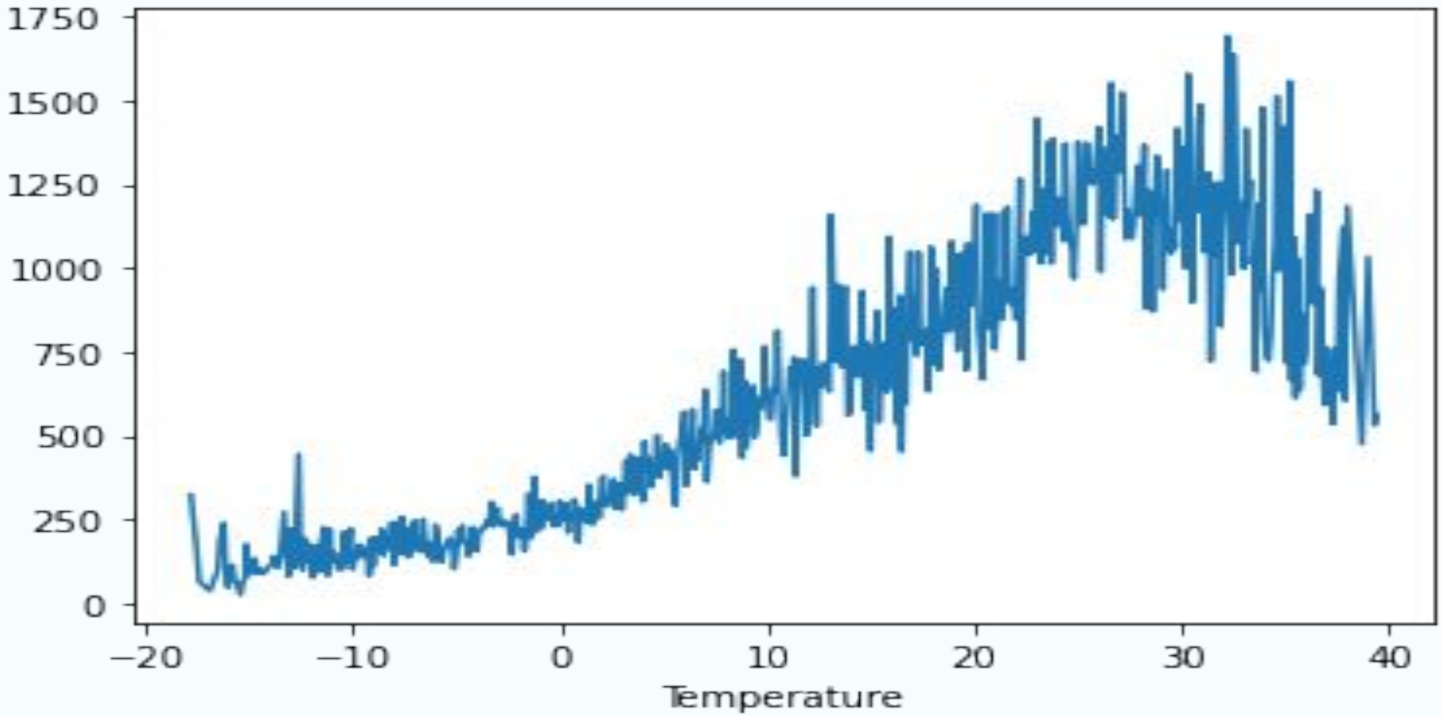
Is there any Variations in Rainfall(mm) Data?



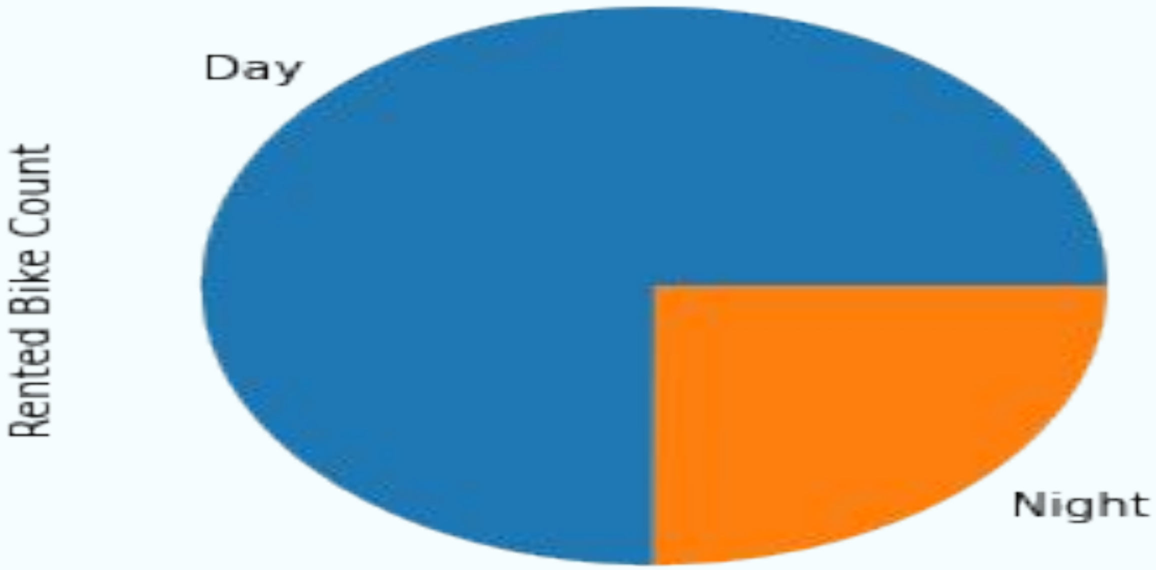
Is there any Variations in Snowfall(cm) Data?



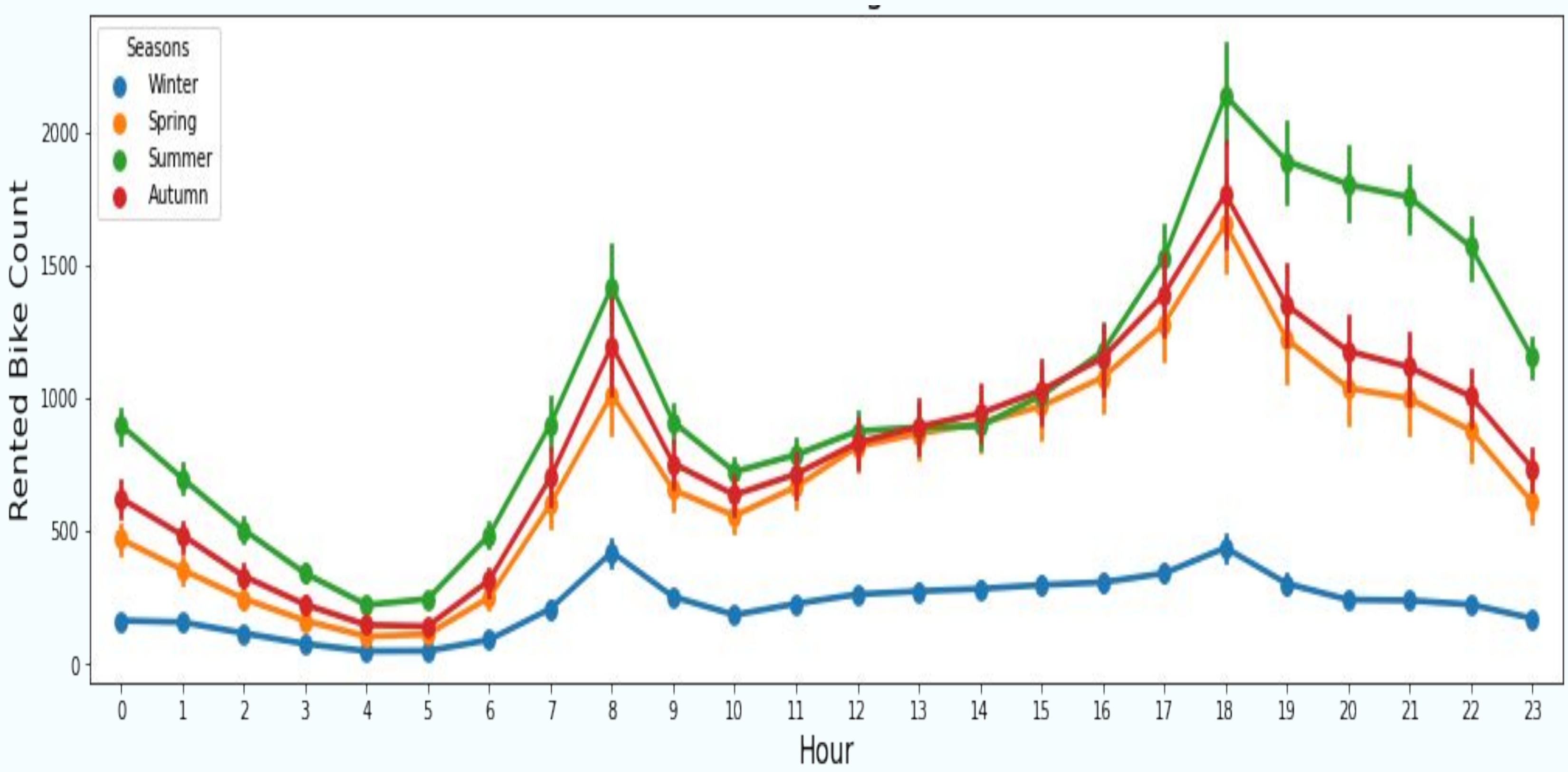
Line plot of Temperature vs Rented bike count



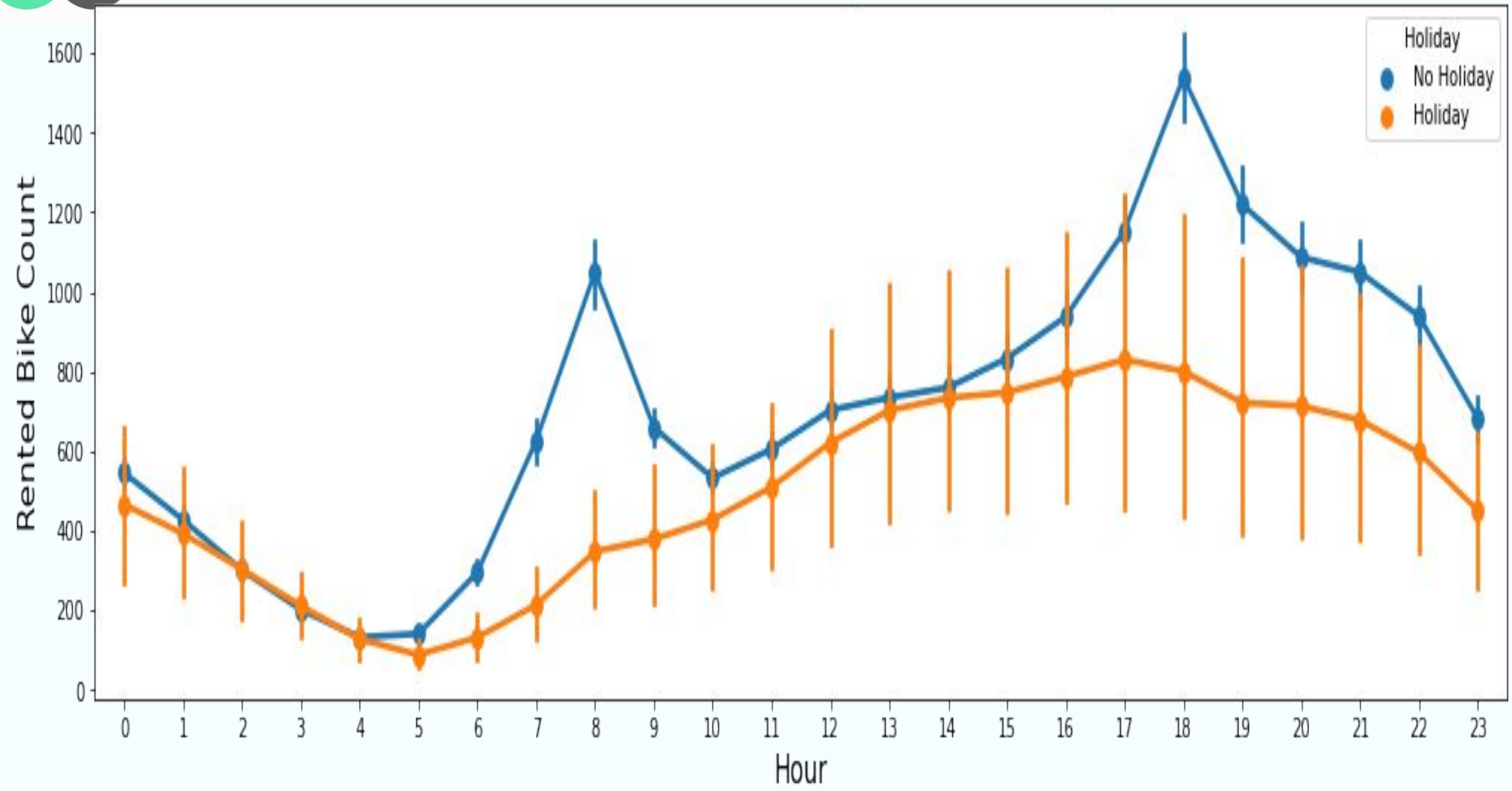
Pie chart of Rental bikes count during Day and Night



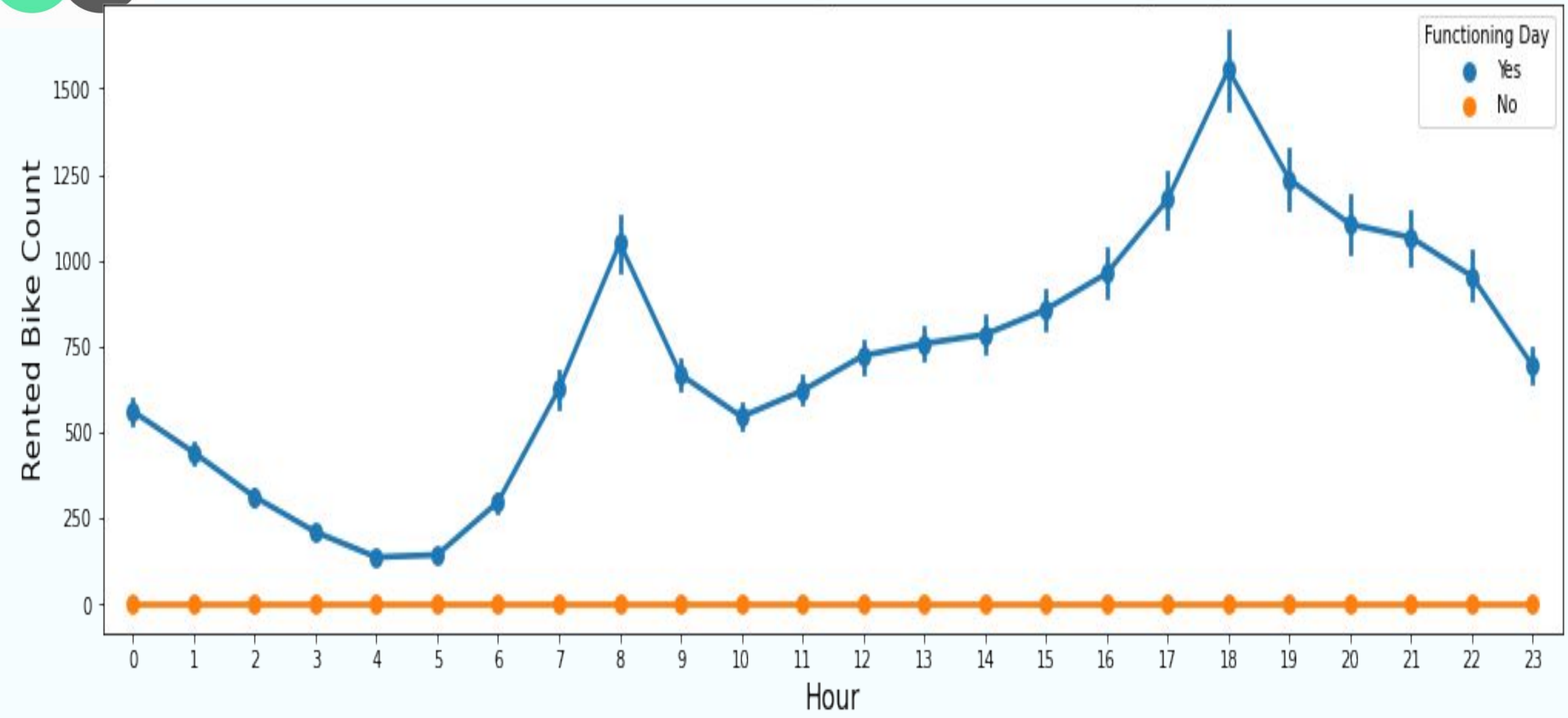
Rented bike Count During Different Seasons

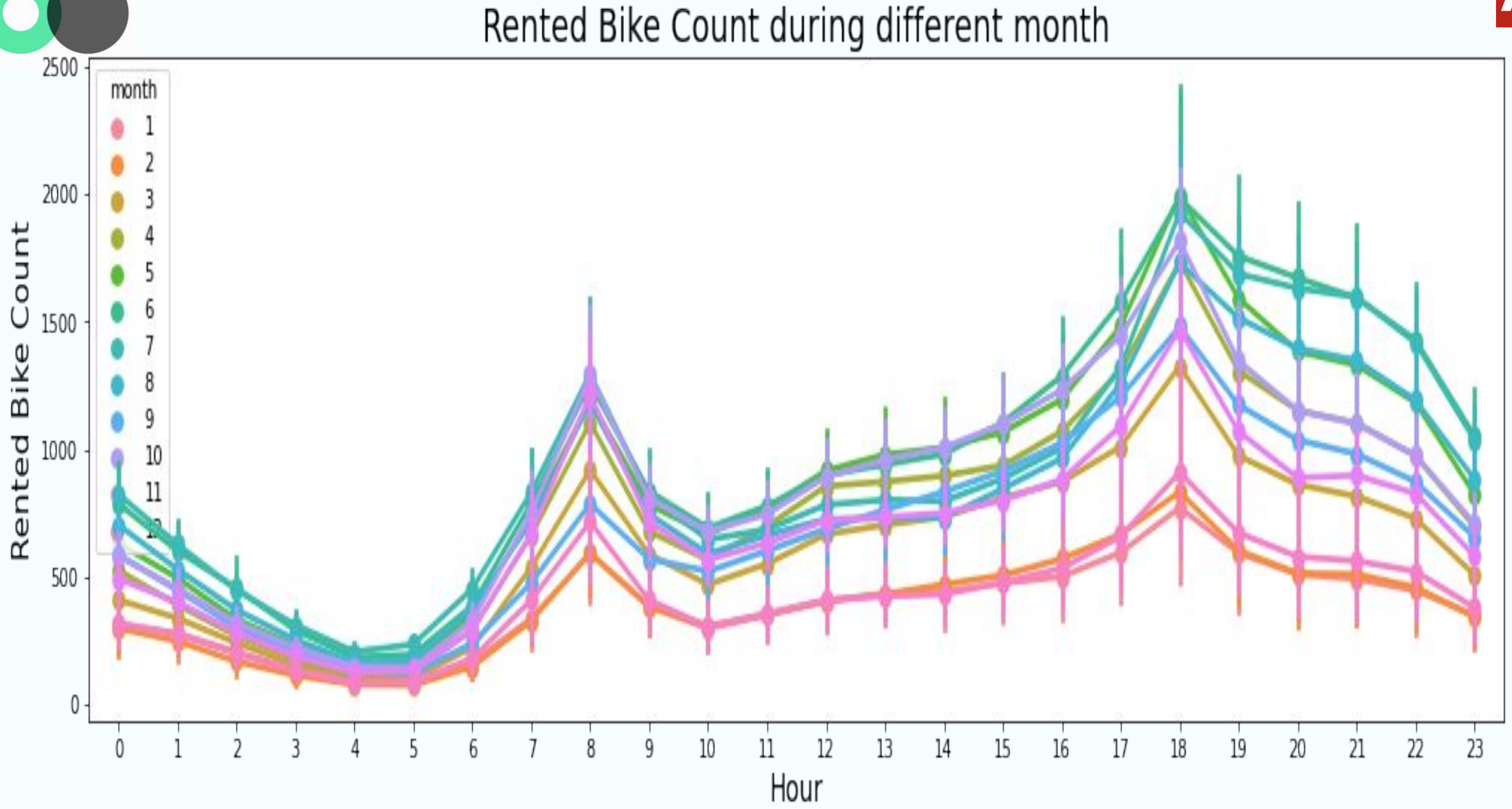


Rented Bike Count during different Holiday

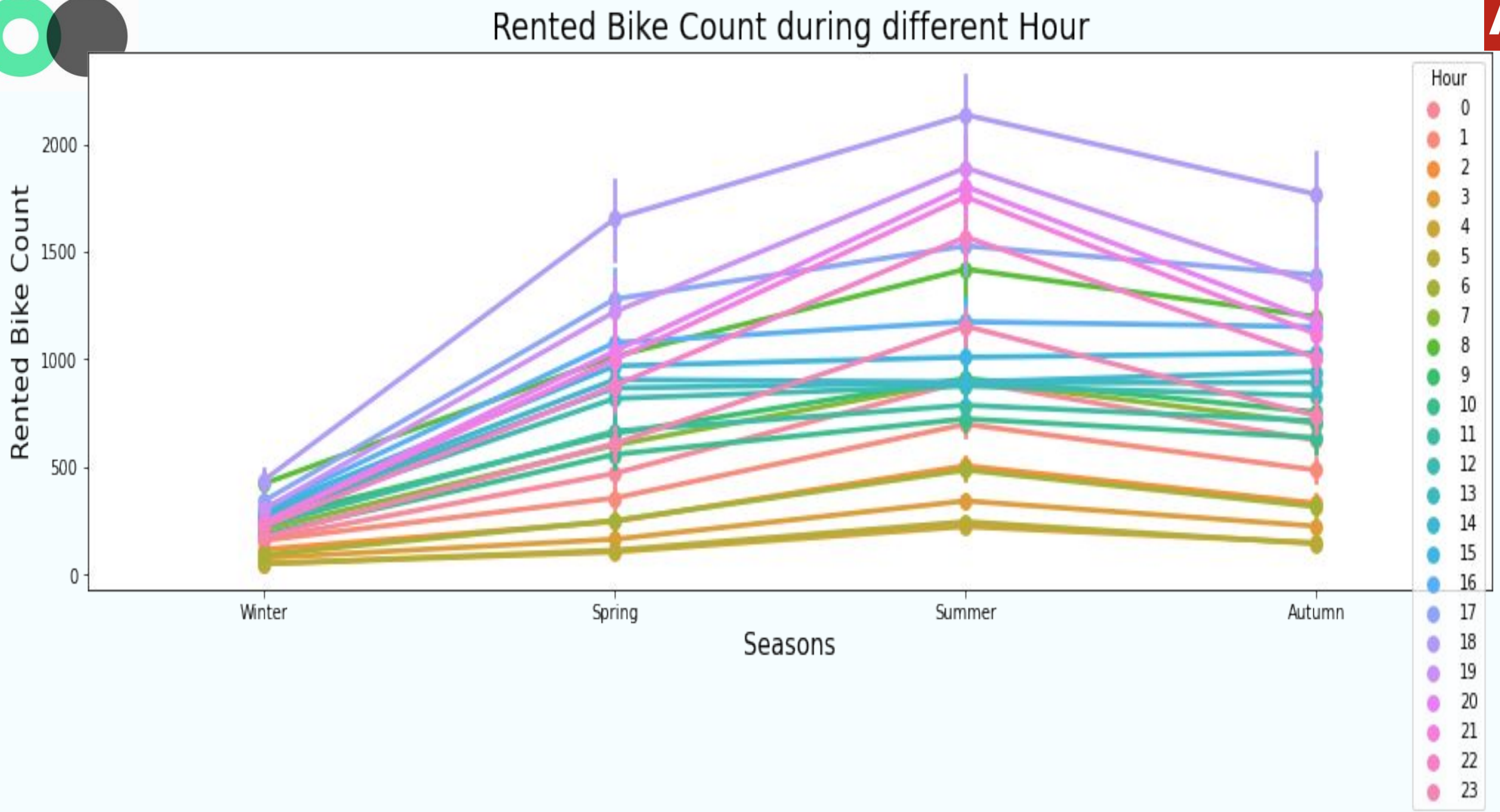


Rented Bike Count during different Functioning Day



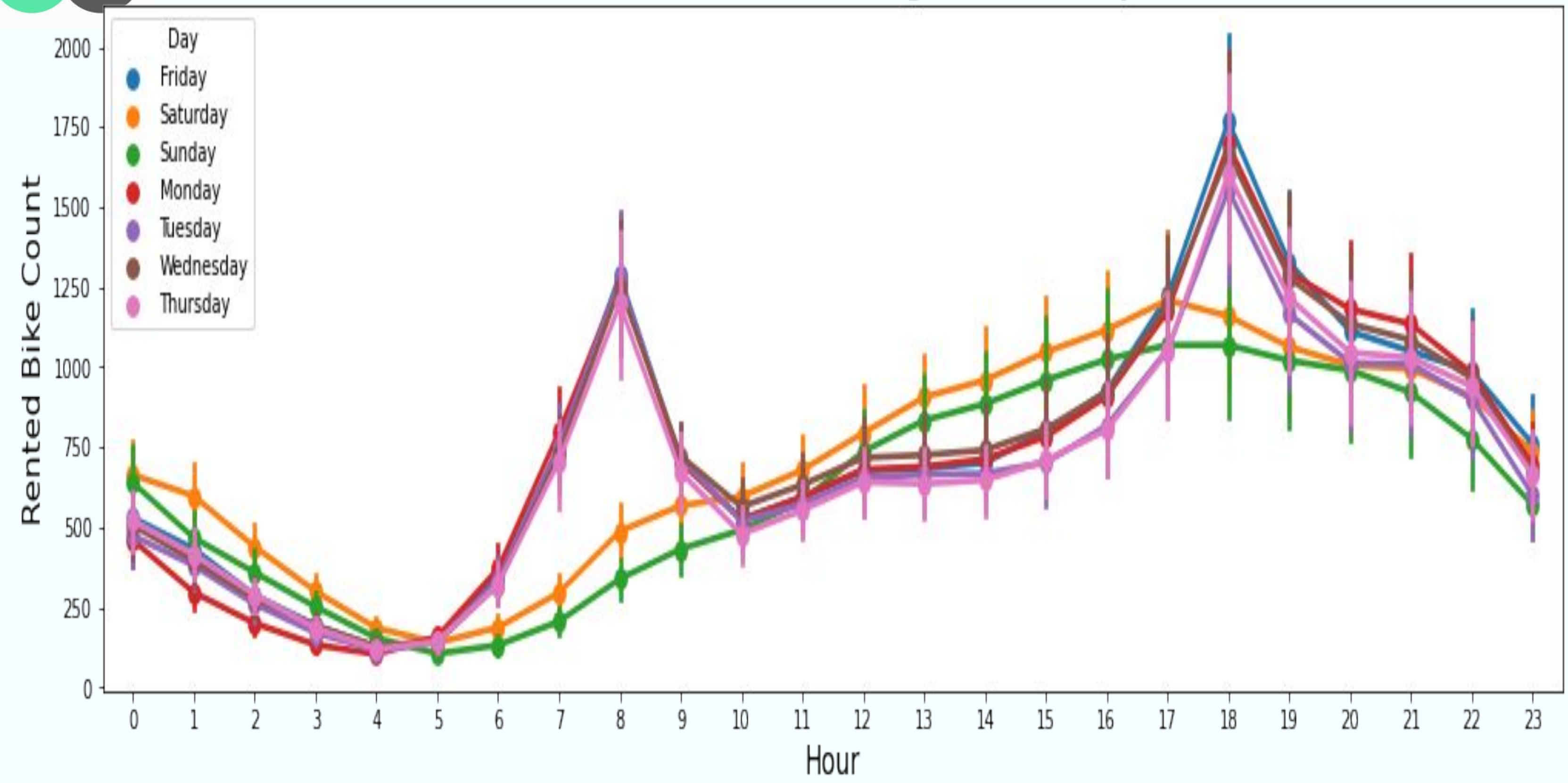


Rented Bike Count during different month
month
1
2
3
4
5
6
7
8
9
10
11
12
Rented Bike Count
0
500
1000
1500
2000
2500
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Hour

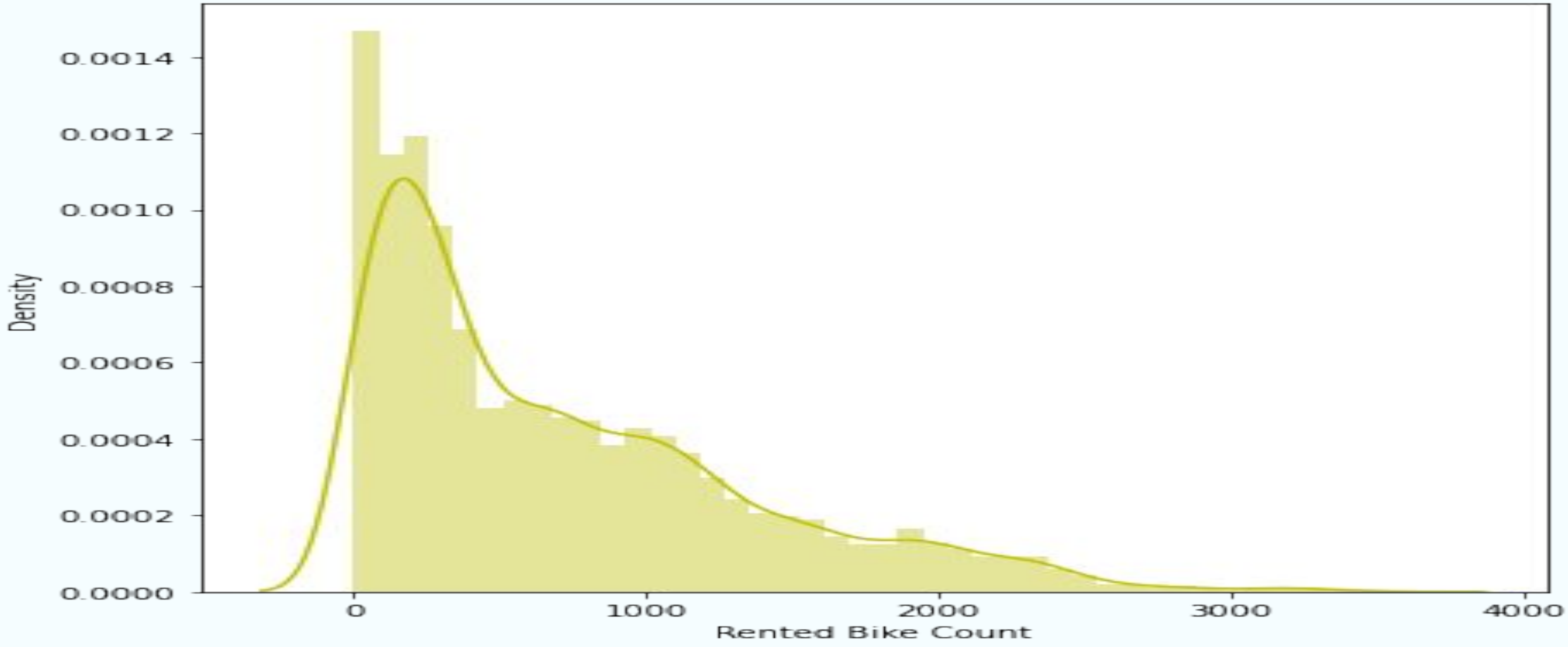


Rented Bike Count during different Hour
Rented Bike Count
0
500
1000
1500
2000
Winter
Spring
Summer
Autumn
Seasons
Hour
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

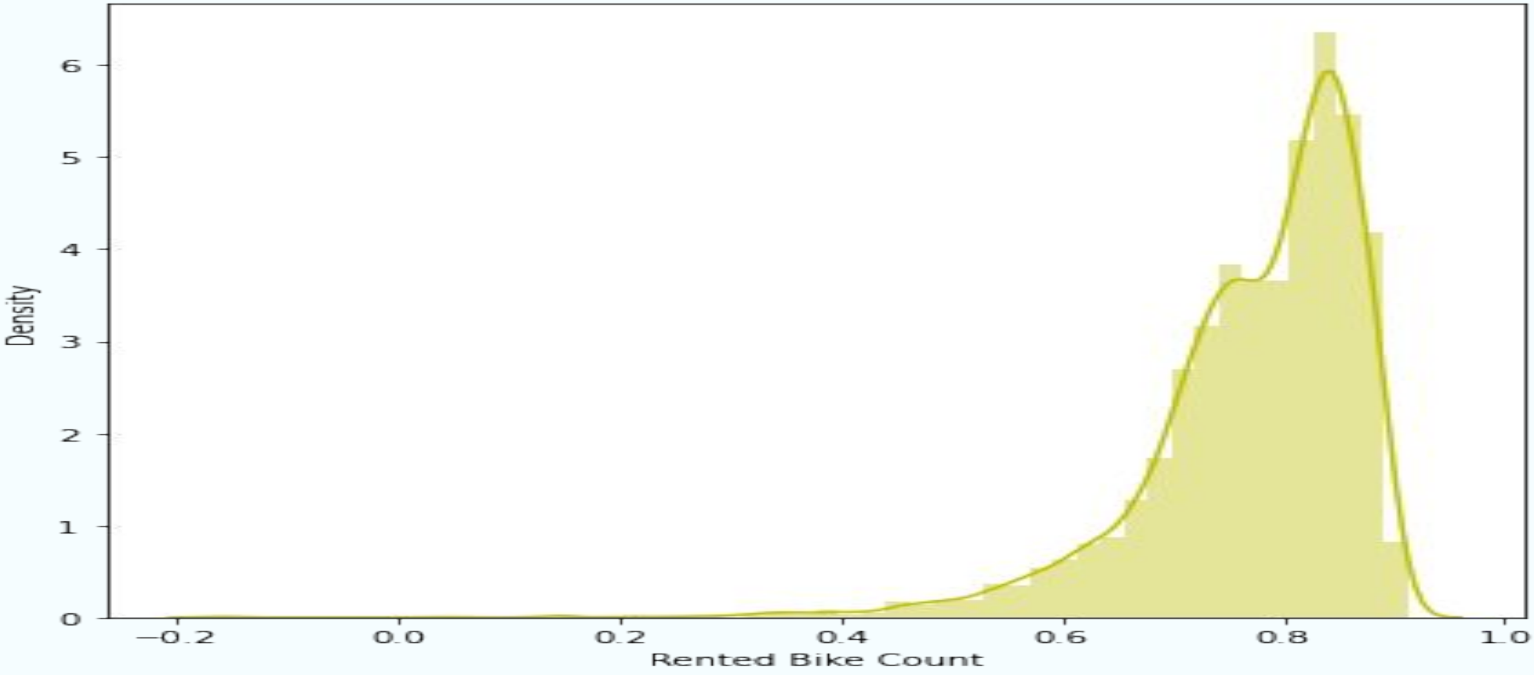
Rented Bike Count during different Day



Distribution Plot of Bikes taken for rental



Distribution of Bikes taken for rental after applying log transformation



Heatmap

Rented Bike Count
Temperature
Humidity
Wind speed
Visibility
Solar Radiation
Rainfall
Snowfall
1 0.56 0.28 0.1 0.25 0.35 0.26 0.19
0.56 1 0.16 0.036 0.035 0.35 0.05 0.22
0.28 0.16 1 0.34 0.54 0.46 0.24 0.11
0.1 0.036 0.34 1 0.17 0.33 0.02 0.0036
0.25 0.035 0.54 0.17 1 0.15 0.17 0.12
0.35 0.35 0.46 0.33 0.15 1 0.074 0.072
0.26 0.05 0.24 0.02 0.17 0.074 1 0.0085
0.19 0.22 0.11 0.0036 0.12 0.072 0.0085 1
1.0
0.8
0.6
0.4
0.2

Convert The Dataset Into The Dependent And The Independent Variable

Independent Variable (X): Temperature , Humidity , Wind speed , Visibility , Solar, Holiday , Functioning Day , hour_1 , hour_2 , hour_3 , hour_4 , hour_5 , hour_6 , hour_7 , hour_8 , hour_9 , hour_10 ,hour_11 , hour_12 , hour_13 , hour_14 , hour_15 , hour_16 , hour_17 , hour_18 , hour_19 , hour_20 , hour_21 , hour_22 , hour_23 , season_Spring , season Summer ,season Winter , month_2 , month_3 , month_4 , month_5 , month_6 , month_7 , monthweekDay_4 , weekDay_5 , month_8 , month_9 , month_10 , month_11 , month_12 , weekDay_2 , weekDay_3 , weekDay_4 ,weekDay_5 , weekDay_6, weekDay_7

Dependent Variable (Y): Rented Bike Count'

Split The Data Into Training Set And The Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X,Y, test_size = 0.2, random_state = 0)
print(X_train.shape)
print(X_test.shape)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = reg.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error

MSE  = mean_squared_error((y_test),(y_pred))
print("MSE :" , MSE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)
```

```
MSE : 0.5064503030007469
RMSE : 0.7116532182184993
```

```
from sklearn.metrics import r2_score
r2 = r2_score((y_test),(y_pred))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test),(y_pred)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
R2 : 0.6349747136136711
Adjusted R2 :  0.6238594491073882
```

```python
from sklearn.linear_model import LinearRegression, Ridge, HuberRegressor, ElasticNetCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, ExtraTreesRegressor

models = [LinearRegression(),
          Ridge(),
          ElasticNetCV(),
          DecisionTreeRegressor(),
          RandomForestRegressor(),
          ExtraTreesRegressor(),
          GradientBoostingRegressor()]
```

```
[ ]
```

```python
from sklearn import model_selection
def train(model):
    kfold = model_selection.KFold(n_splits=5, random_state=42)
    pred = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring='neg_mean_squared_error')
    cv_score = pred.mean()
    print('Model:',model)
    print('CV score:', abs(cv_score))
```

```python
for model in models:
    train(model)
```

```
Model: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
CV score: 0.561815448863636
Model: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Code + Text

```
Model: HuberRegressor(alpha=0.0001, epsilon=1.35, fit_intercept=True, max_iter=100,
               tol=1e-05, warm_start=False)
CV score: 1.269821534132796
Model: ElasticNetCV(alphas=None, copy_X=True, cv=None, eps=0.001, fit_intercept=True,
             l1_ratio=0.5, max_iter=1000, n_alphas=100, n_jobs=None,
             normalize=False, positive=False, precompute='auto',
             random_state=None, selection='cyclic', tol=0.0001, verbose=0)
CV score: 0.8465942024148123
Model: DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
CV score: 0.8330128063441314
Model: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
CV score: 0.4909116591048976
Model: ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=100, n_jobs=None, oob_score=False,
                    random_state=None, verbose=0, warm_start=False)
CV score: 0.4409953642506461
Model: GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
CV score: 0.4155874268064301
```

Gradient Boosting Algorithm:

```
grad_bos=GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

```
[ ] grad_bos.fit(X_train, y_train)
    y_pred_gradboosting = grad_bos.predict(X_test)
```

```
[ ] MSE  = mean_squared_error((y_test),(y_pred_gradboosting))
    print("MSE :" , MSE)

    RMSE = np.sqrt(MSE)
    print("RMSE :" ,RMSE)

    r2 = r2_score((y_test),(y_pred_gradboosting))
    print("R2 :" ,r2)
    print("Adjusted R2 : ",1-(1-r2_score((y_test),(y_pred_gradboosting)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
MSE : 0.31484789811233843
RMSE : 0.561113088523462
R2 : 0.7730726124643654
Adjusted R2 :  0.7661625214919039
```

Let's Apply Random Forest

```
from sklearn.ensemble import RandomForestRegressor
rf_exp = RandomForestRegressor(n_estimators= 1000, random_state=100)
rf_exp.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=1000, n_jobs=None, oob_score=False,
                      random_state=100, verbose=0, warm_start=False)
```

```
[ ] predictions = rf_exp.predict(X_test)
    # Performance metrics
    errors = abs(predictions - y_test)
```

```
[ ] print('Metrics for Random Forest Trained on Expanded Data')
    print('Average absolute error:', round(np.mean(errors), 2), 'degrees.')
```

```
Metrics for Random Forest Trained on Expanded Data
Average absolute error: 0.27 degrees.
```

```
[ ] mape = np.mean(100 * (errors / y_test))
```

```
[ ] accuracy = 100 - np.mean(mape)
```

Accuracy: 94.0% MSE : 0.2271
RMSE: 0.4765
R2: : 0.8362 Adjusted R2: 0.8313

```
[ ] accuracy = 100 - np.mean(mape)
    print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 94.0 %.
```

```
MSE  = mean_squared_error((y_test),(predictions))
print("MSE :" , MSE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test),(predictions))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test),(predictions)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
MSE : 0.22713231786046767
RMSE : 0.47658400923705746
R2 : 0.8362938300493313
Adjusted R2 :  0.8313088675051574
```

```
[ ] features = X_train.columns
    importances = rf_exp.feature_importances_
    indices = np.argsort(importances)
```

Random Forest with some Parameter

AI

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

from pylab import rcParams
rcParams['figure.figsize'] = 8, 8
```

```
randomForestAlgo = RandomForestRegressor()


param = {'n_estimators' : [int(x) for x in np.linspace(start=10,stop=100, num=10)],
         'max_depth' : [60,70,80,90,100],
         'min_samples_split':[2,4,6,8],
         'min_samples_leaf':[1,2,3,4],
         'bootstrap' : [True,False]
        }

gridSearch_RandomForest=GridSearchCV(randomForestAlgo,param,scoring='r2',cv=5,verbose=2,n_jobs=-1)
best_mode_try=gridSearch_RandomForest.fit(X_train,y_train)
```

```
[89] best_mode_try.best_params_
```

```
{'bootstrap': True,
 'max_depth': 70,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 90}
```

MSE: 0.2326 R2: 0.8323
RMSE: 0.4823 Adjusted R2: 0.82722

```
[67] randomForestAlgo = RandomForestRegressor()
     param = {'bootstrap': [True],
              'max_depth': [70],
              'min_samples_leaf': [1],
             'min_samples_split': [2],
              'n_estimators': [90]}

     gridSearch_RandomForest=GridSearchCV(randomForestAlgo,param,scoring='r2',cv=5,verbose=2,n_jobs=-1)
     best_mode_try=gridSearch_RandomForest.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   16.3s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   16.3s finished
```

```
[68] y_random_pred=best_mode_try.predict(X_test)
```

```
MSE  = mean_squared_error((y_test),(y_random_pred))
print("MSE :" , MSE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test),(y_random_pred))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test),(y_random_pred)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

```
MSE : 0.23262969000795694
RMSE : 0.48231700157464585
R2 : 0.8323315857173212
Adjusted R2 :  0.8272259701788718
```

Accuracy: 93.93 %

AI

```
[69] MSE  = mean_squared_error((y_test),(y_random_pred))
     print("MSE :" , MSE)

     RMSE = np.sqrt(MSE)
     print("RMSE :" ,RMSE)

     r2 = r2_score((y_test),(y_random_pred))
     print("R2 :" ,r2)
     print("Adjusted R2 : ",1-(1-r2_score((y_test),(y_random_pred)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))

MSE : 0.23262969000795694
RMSE : 0.48231700157464585
R2 : 0.8323315857173212
Adjusted R2 :  0.8272259701788718
```

```
[70] errors = abs(y_random_pred - y_test)
```

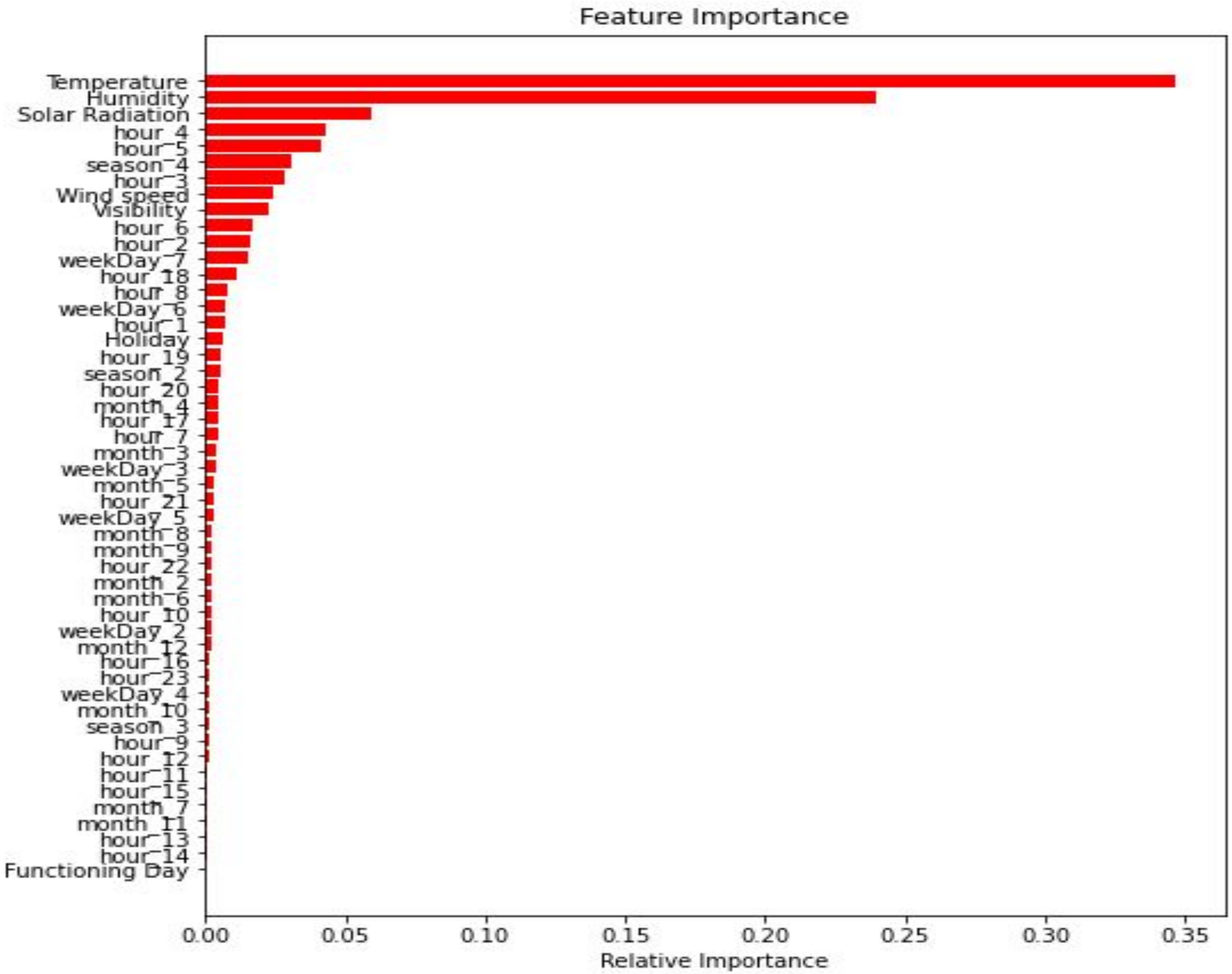
```
print('Metrics for Random Forest Trained on Expanded Data')
print('Average absolute error:', round(np.mean(errors), 2), 'degrees.')

Metrics for Random Forest Trained on Expanded Data
Average absolute error: 0.28 degrees.
```

```
[74] mape = np.mean(100 * (errors / y_test))
```

```
[75] accuracy = 100 - np.mean(mape)
     print('Accuracy:', round(accuracy, 2), '%.')

Accuracy: 93.93 %.
```


Feature Importance
Temperature
Humidity
Solar Radiation
hour_4
hour_5
season_4
hour_3
Wind speed
Visibility
hour_6
hour_2
weekDay_7
hour_18
hour_8
weekDay_6
hour_1
Holiday
hour_19
season_2
hour_20
month_4
hour_17
hour_7
month_3
weekDay_3
month_5
hour_21
weekDay_5
month_8
month_9
hour_22
month_2
month_6
hour_10
weekDay_2
month_12
hour_16
hour_23
weekDay_4
month_10
season_3
hour_9
hour_12
hour_11
hour_15
month_7
month_11
hour_13
hour_14
Functioning Day
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Relative Importance

Conclusion

- People like to ride bikes when it is pretty hot around 25°C in average
- In morning hours(8-9) and in evening hours (5-8),the bikes taken for rental are more .
- So let's focus on the seasons where we have the most rents because at the month of may (5) to july (7) bikes have the most rents.
- Bikes taken for rental are more in Summer and less in Winter
- Here we see at the weekend Bike goes to be rented less compare to the working days.
- During No Holidays , the bikes taken for rental are more than during holidays.
- Bikes for rental are very high during functioning days.
- Number of Bike Rented in day is high as compare to the night
- During Summer ,rented bikes are more in each hour than other seasons
- During Winter ,rented bikes are less in each hour compared to other seasons.
- We see the Rainfall so most of the value is 0.0 and but some of the value we can say that people enjoyed ride with bike during rainfall.
- When snowfall more than 4 cm of snow, the bike rents is much lower

Conclusion

- At Saturday and Sunday we see the Bike rented is less but at the evening time it goes bit up.
- Monday to friday all the hours seems like same for the Rented Bike count.
- Used many of the algorithm t check for the best predicted results but Random Forest model looks better as compare to the other model.


Q&A

THANK YOU