Bike Sharing System Analytics (Washington D. C.)

-Diane Avila, Sukanya Ghosh, Suhas Nadiga, Jinwei Wang, Ming Zhu

Team4 | OPIM5894

Agenda

- Description of Business Problem
- Visualizations
- Data Transformation
- Modeling
- Recommendations

1 Description of Business Problem

1.1 Business Problem

1) Hourly forecast demand of bike sharing usage in Washington D.C.

2) Hourly forecast casual demand of bike sharing usage in Washington D.C.

1.2 Motivation

- **Analyzing the factors affect city ridership.**
- **Measure the level of “Go Green and Healthy” in the City.**
- **Temporal ridership change pattern exploration –potential market growth and inventory management.**

Source:

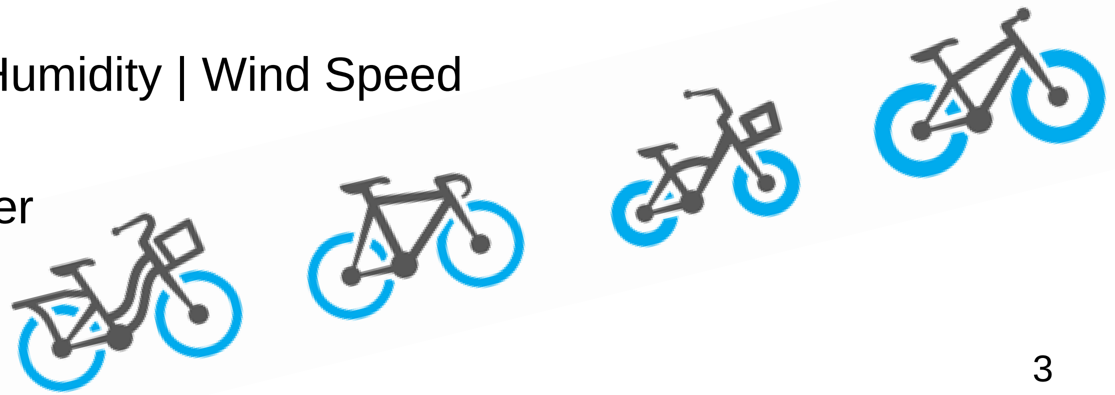
- *’Effects of Built Environment and Weather on Bike Sharing Demand: Station Level Analysis of Commercial Bike Sharing in Toronto’;*
- *’Demand Forecasting on Bay Area BikeShare’.*
- *‘Forecasting Bike Sharing Demand’*

1.3 Data Overview

- Washington DC, spanning over two years with hourly rental data
- Training set first 19 days of each month
- Test set is the 20th to the end of the month

<u>Data Features</u>

- Date Time
- Season
- Weather |Temperature | Humidity | Wind Speed
- Holiday | Working Day
- Casual or Registered User



1.3 Data Overview

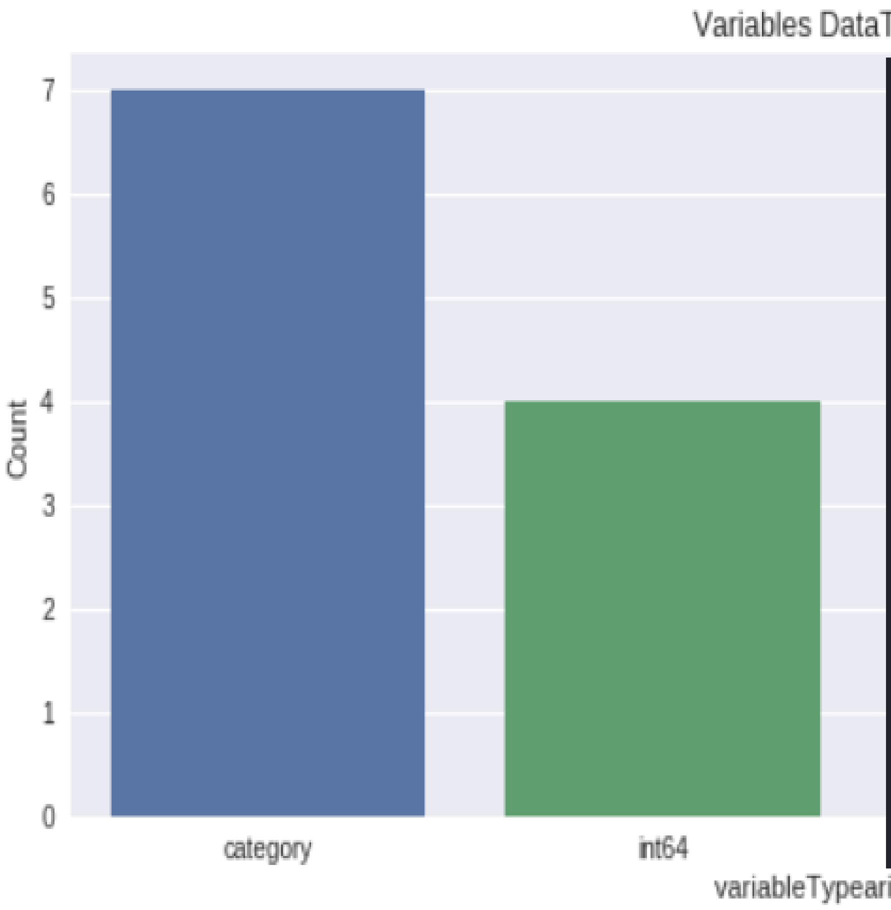
Independent Variables:
-Categorical:
weather,season,holiday,
workingday
-Continuous:
datetime,temp,atemp,
humidity,windspeed,
casual,registered

Dependent Variables:
-Hourly count of bike

Dataset	**Instances**
Train	10,886

```
In [63]: missing=sum(bikes.isnull().values.ravel())

In [64]: missing
Out[64]: 0
```

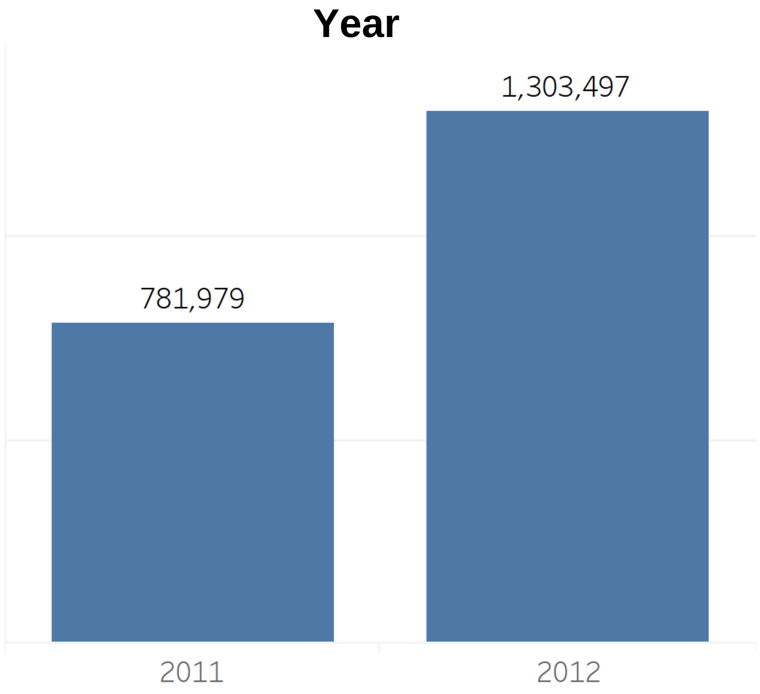
```
In [47]: bikes.describe()
Out[47]:
             season       holiday    workingday       weather         temp
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.00000
mean       2.506614      0.028569      0.680875      1.418427     20.23086
std        1.116174      0.166599      0.466159      0.633839      7.79159
min        1.000000      0.000000      0.000000      1.000000      0.82000
25%        2.000000      0.000000      0.000000      1.000000     13.94000
50%        3.000000      0.000000      1.000000      1.000000     20.50000
75%        4.000000      0.000000      1.000000      2.000000     26.24000
max        4.000000      1.000000      1.000000      4.000000     41.00000

              atemp      humidity     windspeed        casual    registered
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.000000
mean      23.655084     61.886460     12.799395     36.021955    155.552177
std        8.474601     19.245033      8.164537     49.960477    151.039033
min        0.760000      0.000000      0.000000      0.000000      0.000000
25%       16.665000     47.000000      7.001500      4.000000     36.000000
50%       24.240000     62.000000     12.998000     17.000000    118.000000
75%       31.060000     77.000000     16.997900     49.000000    222.000000
max       45.455000    100.000000     56.996900    367.000000    886.000000

              total     dayofweek          hour         month          year
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.000000
mean     191.574132      3.013963     11.541613      6.521495   2011.501929
std      181.144454      2.004585      6.915838      3.444373      0.500019
min        1.000000      0.000000      0.000000      1.000000   2011.000000
25%       42.000000      1.000000      6.000000      4.000000   2011.000000
50%      145.000000      3.000000     12.000000      7.000000   2012.000000
75%      284.000000      5.000000     18.000000     10.000000   2012.000000
max      977.000000      6.000000     23.000000     12.000000   2012.000000
```




```
datetime       object
season          int64
holiday         int64
workingday      int64
weather         int64
temp          float64
atemp         float64
humidity        int64
windspeed     float64
casual          int64
registered      int64
count           int64
dtype: object
```

2 Visualization

Data Visualization - Year



Record Count - Year

2011	49.8%
2012	50.2%

- Bike Sharing nearly doubled in a year
- A sign of exponential growth
- A contender for Time-Series Forecasting

Data Visualization – Holiday & Working Day

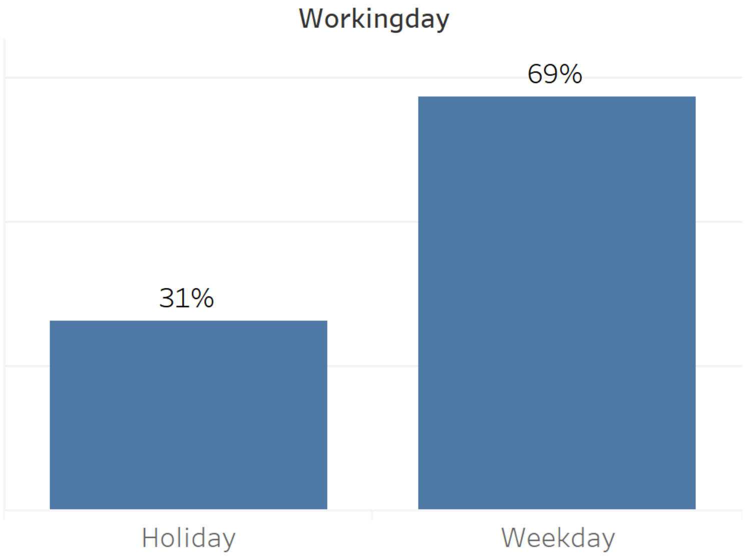
Record Count - Holiday

WorkingDay	97.1%
Holiday	2.9%

Record Count - Working Day

Holiday	31.9%
Weekday	68.1%

Bikes Rented



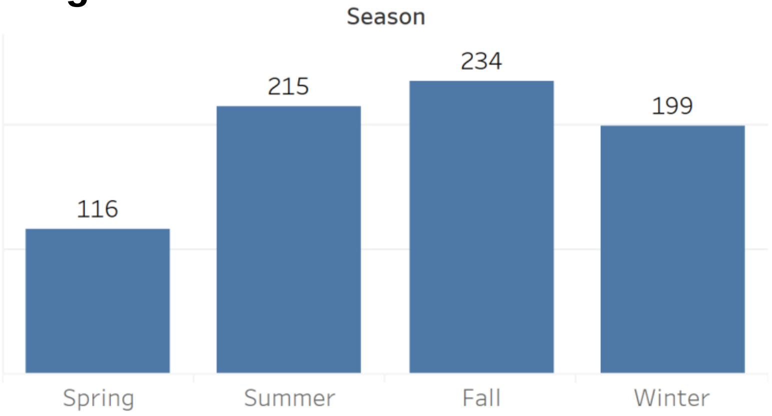
- Only 3% of Holiday records – not a strong predictor, biased towards Working Day

Data Visualization – Season & Weather

Record Count - Season

Spring	24.7%
Summer	25.1%
Fall	25.1%
Winter	25.1%

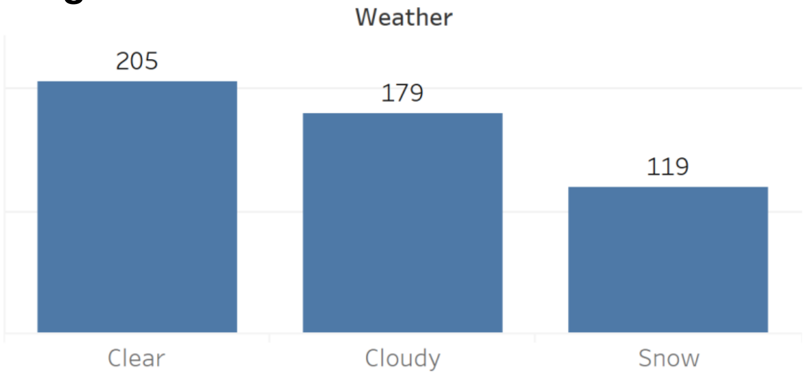
Avg. Bikes Rented



Record Count - Weather

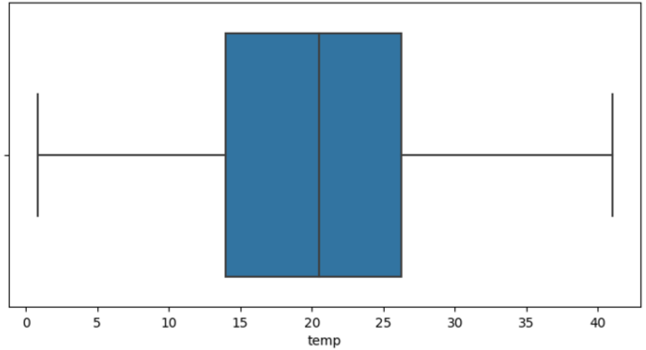
Clear	66.07%
Cloudy	26.03%
Snow	7.89%
Rain	0.01%

Avg. Bikes Rented

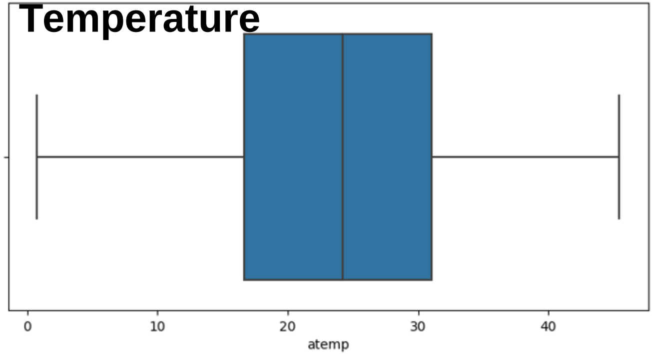


Data Visualization - Temperature

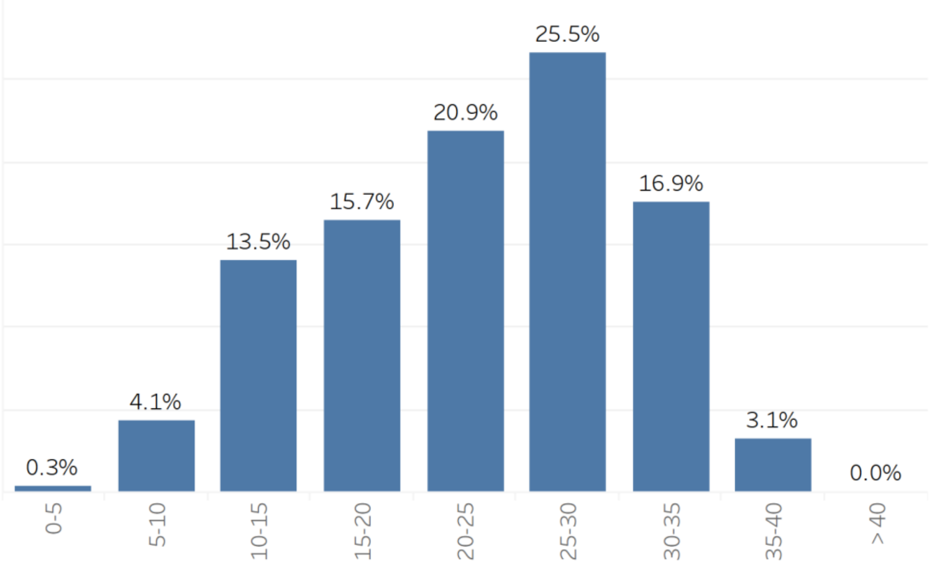
Box Plot - Temperature



Box Plot – Feels Like Temperature



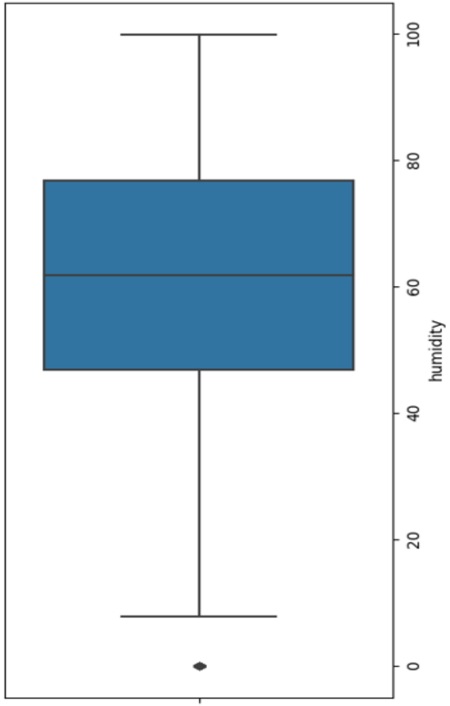
Histogram - Temperature



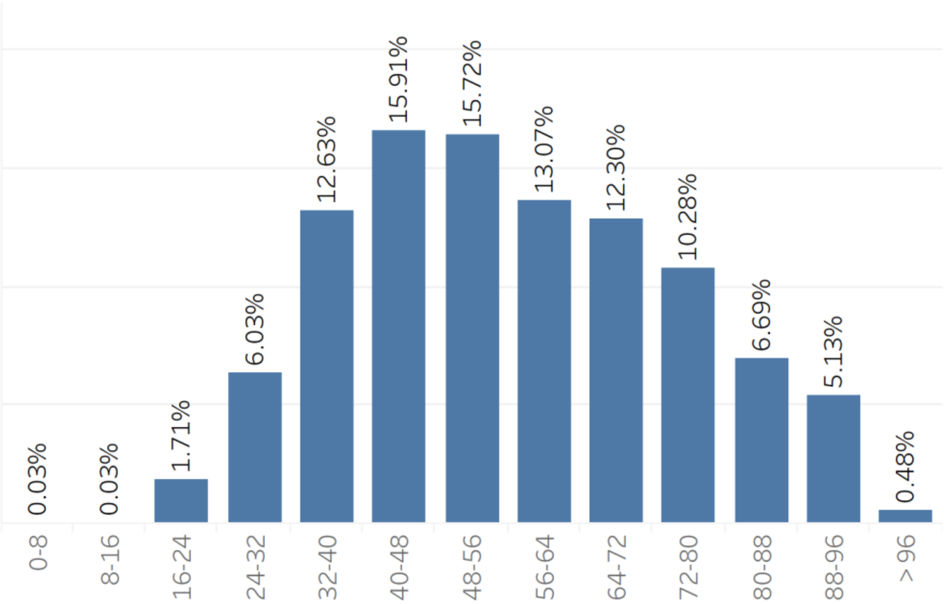
- Feels like temp is slightly higher than temp
- Feels like temp can be influenced by a number of factors like windspeed, humidity etc.

Data Visualization - Humidity

Box Plot - Humidity

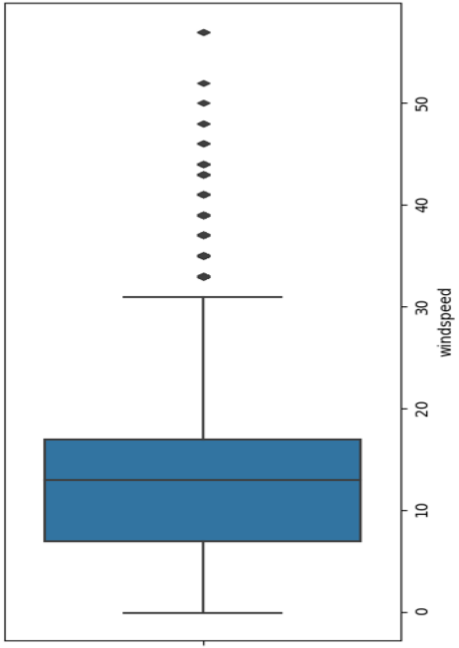


Histogram - Humidity

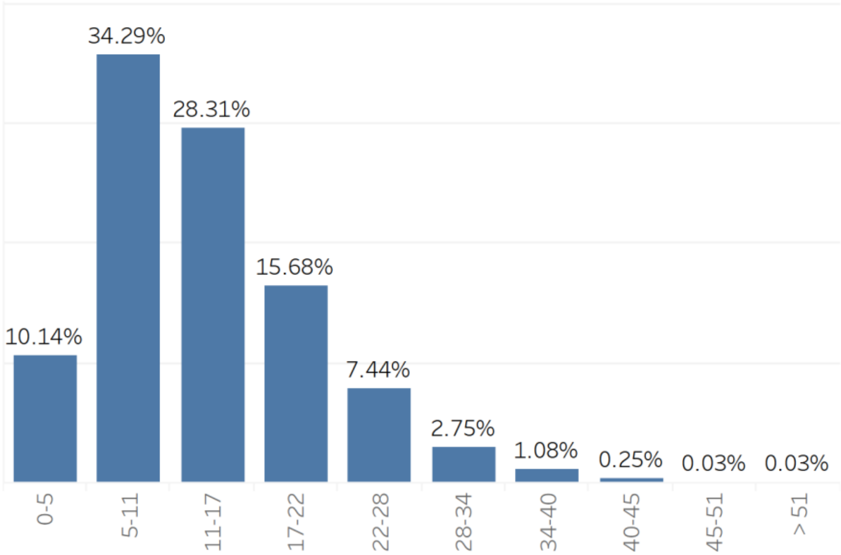


Data Visualization - Windspeed

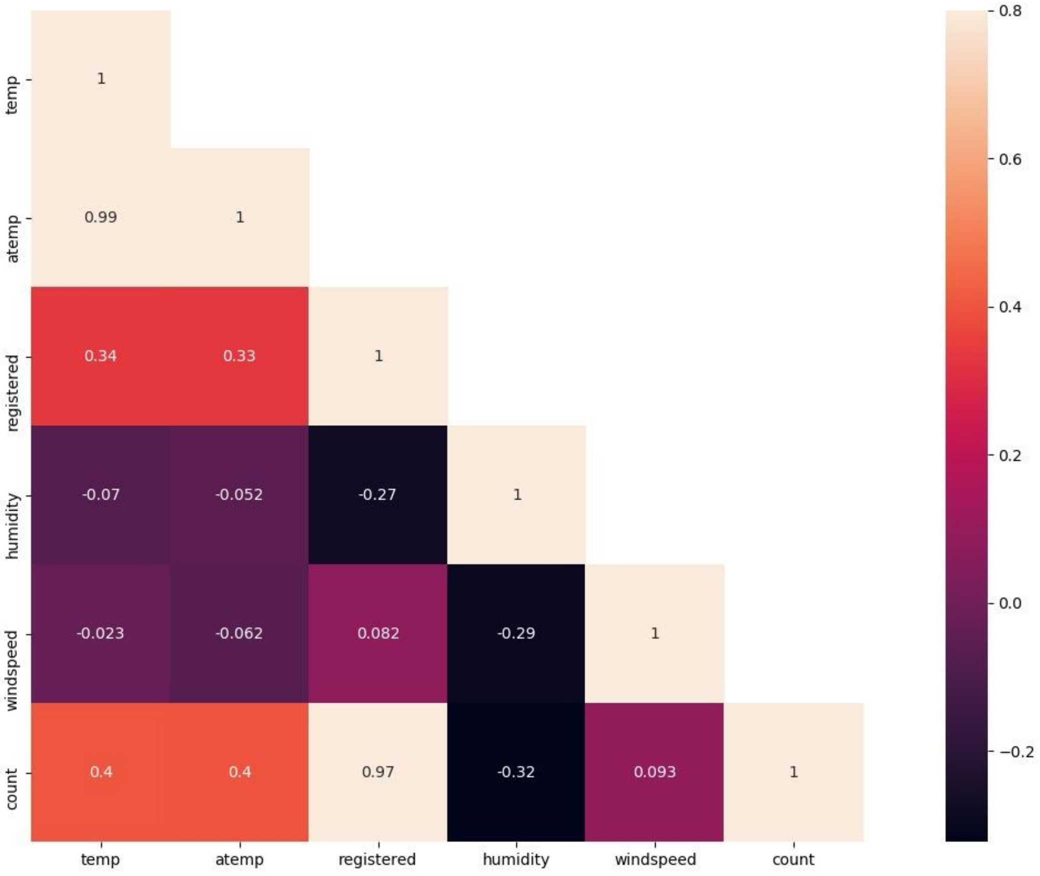
Box Plot - Windspeed



Histogram - Windspeed

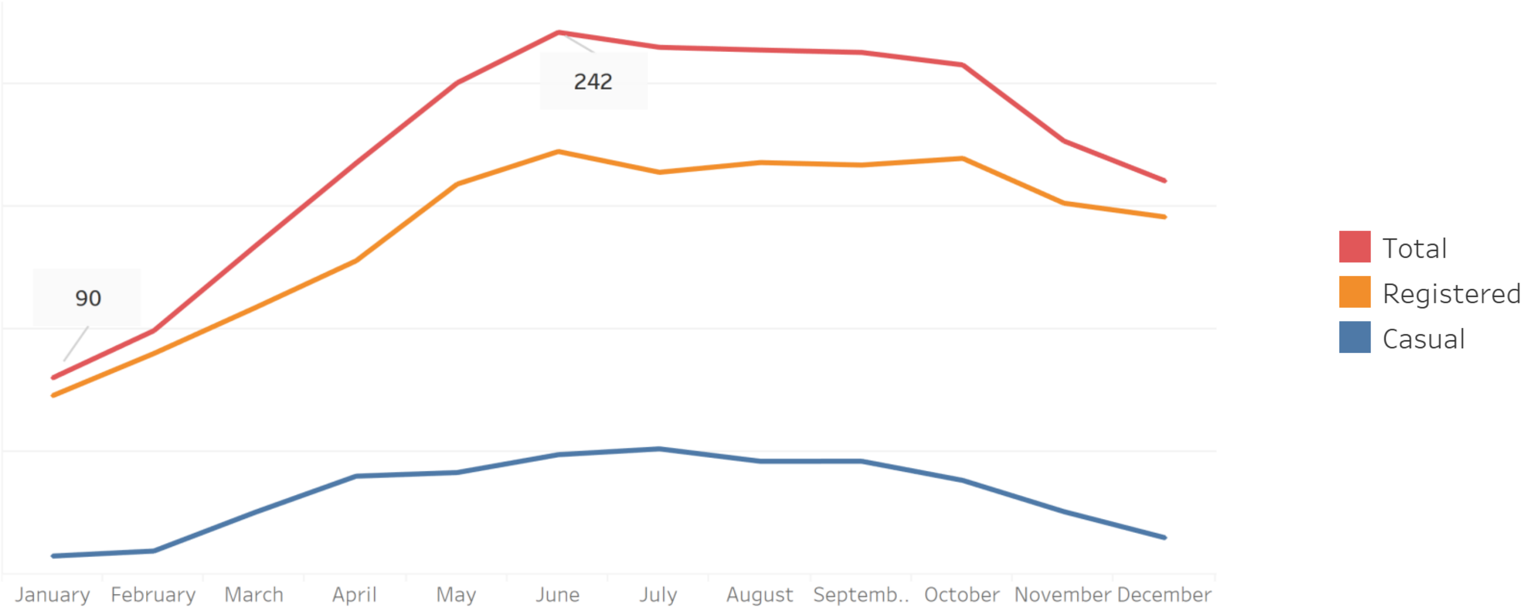


Correlation Analysis



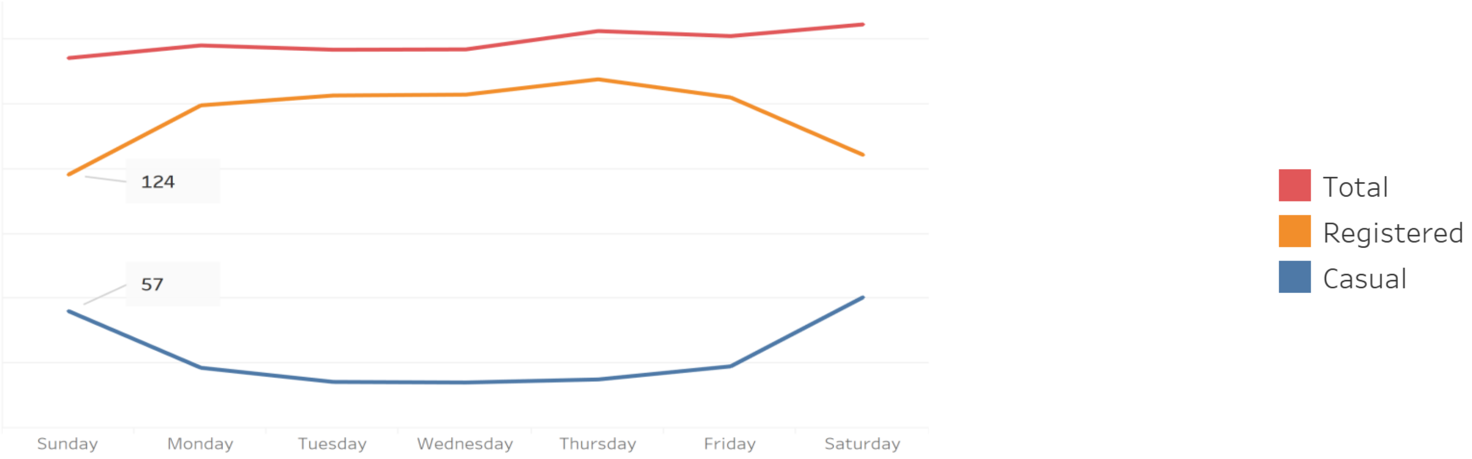
- Temperature has a +ve correlation with Count
- Humidity has a -ve correlation with Count
- Multicollinearity can be observed between Temp & Atemp
- No correlation between Windspeed and Count variable

Avg. Bike Rentals over the months

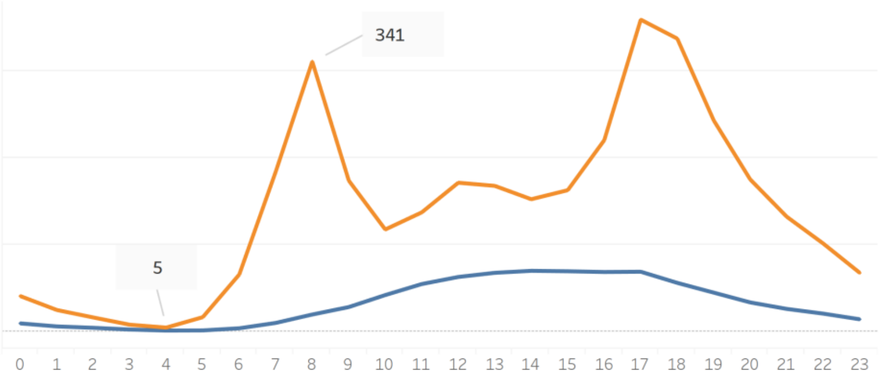


- Low renting of bikes from Dec to Feb because of winter.
- An average of over 200 bikes were rented every hour in June & July

Avg. Bike Rentals over the week



Avg. Bike Rental Spread over a day



- Peak sharing for registered users at 8 AM and 5 PM
- Casual users rented more bikes in the weekends
- A curious case for predicting casual users

3 Data Transformation(Pandas)

3.1 Feature Engineering

'datetime' to 'month', 'hour', 'year'

```
In [1]: from datetime import datetime
   ...: new_df=train_df
   ...: new_df['month']=new_df['datetime'].apply(lambda x:x.month)
   ...: new_df['hour']=new_df['datetime'].apply(lambda x:x.hour)
   ...: new_df['day']=new_df['datetime'].apply(lambda x:x.day)
   ...: new_df['year']=new_df['datetime'].apply(lambda x:x.year)
   ...: new_df.head()
```

Define Categorical and Numerical

```
...: categoricalFeatureNames = ["season","holiday","workingday","weather","month","year","hour","weekday"]
...: numericalFeatureNames = ["temp","humidity","windspeed","atemp",]
```

3.2 Drop Multicollinearity and Redundant Variables

```
final_df=new_df.drop(['datetime','atemp','casual','registered','day'], axis=1)
```

```
Index(['season', 'holiday', 'workingday', 'weather', 'temp', 'humidity',
       'windspeed', 'count', 'month', 'hour', 'year', 'w_2', 'w_3', 'w_4',
       'y_2012', 'm_2', 'm_3', 'm_4', 'm_5', 'm_6', 'm_7', 'm_8', 'm_9',
       'm_10', 'm_11', 'm_12', 'h_1', 'h_2', 'h_3', 'h_4', 'h_5', 'h_6', 'h_7',
       'h_8', 'h_9', 'h_10', 'h_11', 'h_12', 'h_13', 'h_14', 'h_15', 'h_16',
       'h_17', 'h_18', 'h_19', 'h_20', 'h_21', 'h_22', 'h_23', 's_2', 's_3',
       's_4'],
      dtype='object')
```

3.3 Clear Outliers for the “windspeed”

```
...: import numpy as np
...: import pandas as pd
...: train_df=pd.read_csv('hour2.csv')
...: train_df.datetime = pd.to_datetime(train_df.datetime)
...: #remove outliers of windspeed
...: from datetime import datetime
...: q1 = train_df['windspeed'].quantile(0.25)
...: q3 = train_df['windspeed'].quantile(0.75)
...: iqr = q3-q1 #Interquartile range
...: fence_low  = q1-1.5*iqr
...: fence_high = q3+1.5*iqr
...: mask = (train_df['windspeed'] > fence_low) & (train_df['windspeed'] < fence_high)
...: df_out = train_df.loc[mask]
...: df_out['casual2'] = np.log1p(df_out['casual'])
...: data = df_out
...:
```

4 Modeling

4.1 Literature References

In Common:
Models Applied:
Random Forest Regressor, Gradient Boost, Time Series Analysis etc.
Metrics Employed: RMSLE

RMSLE is suitable for our problem because RMSLE penalises an underprediction more than an over prediction. The bike sharing company would lose revenue if the number of bikes will be less than the demand for the bikes.

In Difference:

- Our modeling: Explored in deep learning : Tensorflow and Neural Network

Source:

- *'Effects of Built Environment and Weather on Bike Sharing Demand: Station Level Analysis of Commercial Bike Sharing in Toronto';*
- *'Demand Forecasting on Bay Area BikeShare'*.
- *'Forecasting Bike Sharing Demand'*

4.2 Model Fitting : For looping To Synthesize The Result:

```
In [7]: #modeling
   ...: n_folds = 6
   ...: model_br = BayesianRidge()
   ...: model_lr = LinearRegression()
   ...: model_etc = ElasticNet()
   ...: model_svr = SVR()
   ...: model_gbr = GradientBoostingRegressor()
   ...: model_rf =RandomForestRegressor()
   ...: model_names = ['BayesianRidge', 'LinearRegression', 'ElasticNet', 'SVR', 'GBR','RF']
   ...: model_dic = [model_br, model_lr, model_etc, model_svr, model_gbr, model_rf]
   ...: cv_score_list = []
   ...: pre_y_list = []
   ...: for model in model_dic:
   ...:     scores = cross_val_score(model, x_train, y_train, cv=n_folds)
   ...:     cv_score_list.append(scores)
   ...:     pre_y_list.append(model.fit(x_train, y_train).predict(x_train))
   ...: model_metrics_name = [rmsle, mean_absolute_error, mean_squared_error, r2_score]
   ...: model_metrics_list = []
   ...: for i in range(6):
   ...:     tmp_list = []
   ...:     for m in model_metrics_name:
   ...:         tmp_score = m(y_train, pre_y_list[i])
   ...:         tmp_list.append(tmp_score)
   ...:     model_metrics_list.append(tmp_list)
```

4.2 Model Fitting

'Sklearn' & ‘Numpy’ :'BayesianRidge', 'LinearRegression', 'ElasticNet', 'SVR', 'GBR','RF'

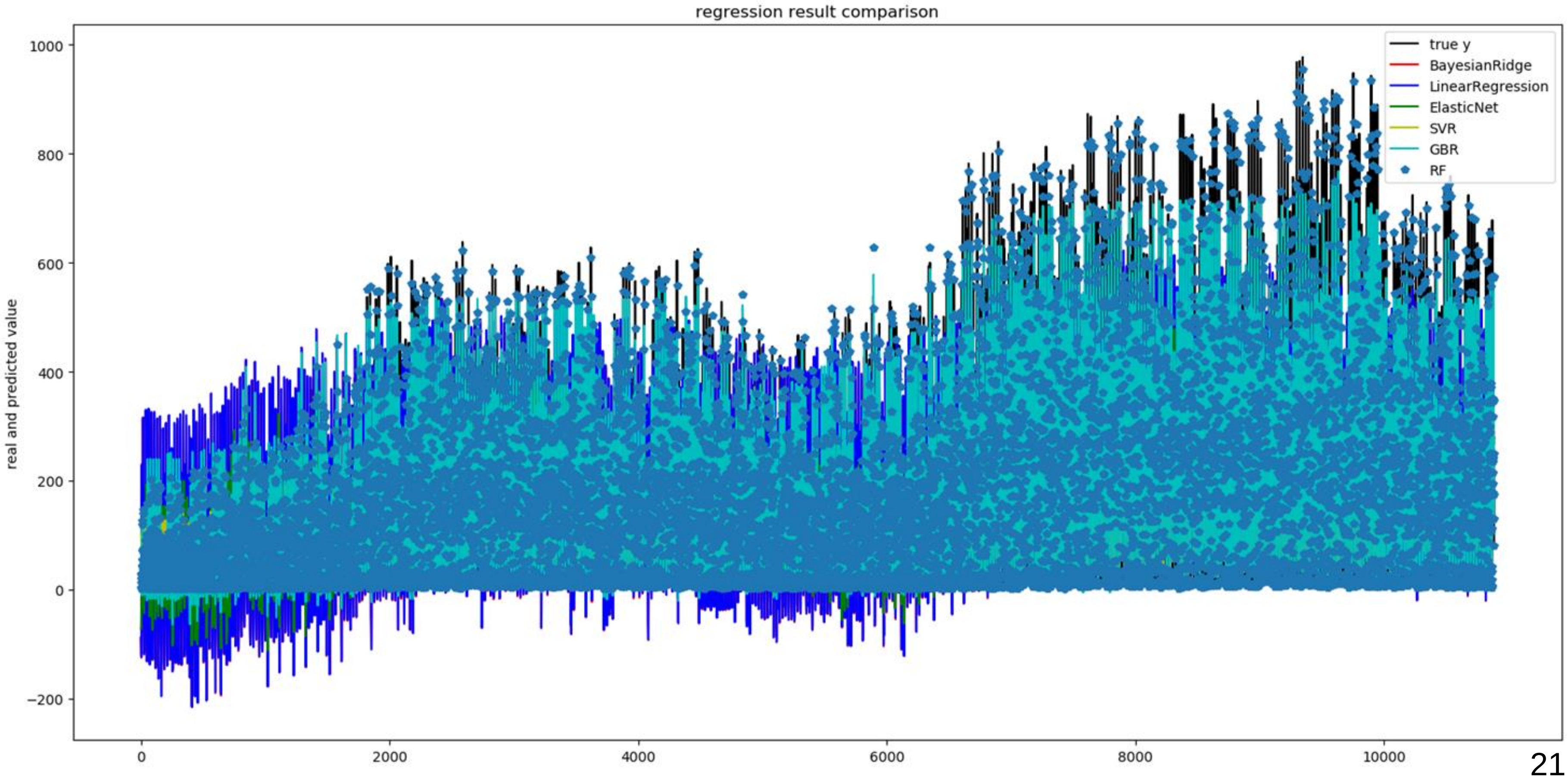
```
In [6]: import numpy as np
   ...: from sklearn.linear_model import BayesianRidge, LinearRegression, ElasticNet
   ...: from sklearn.svm import SVR
   ...: from sklearn.ensemble.gradient_boosting import GradientBoostingRegressor
   ...: from sklearn.ensemble import RandomForestRegressor
   ...: from sklearn.model_selection import cross_val_score
   ...: from sklearn.metrics import explained_variance_score, mean_absolute_error, mean_squared_error, r2_score
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: #X and Y assigned
   ...: x_train=final_df.iloc[:,final_df.columns!='count'].values
   ...: y_train=final_df.iloc[:,7].values
   ...:
```

```
In [1]: def rmsle(y, y_):
   ...:     log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
   ...:     log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
   ...:     calc = (log1 - log2) ** 2
   ...:     return np.sqrt(np.mean(calc))
```

```
-------------------------------------------------------------------
cross validation result:
                         0         1         2             3         4  \
BayesianRidge    -0.609746  0.693791  0.634383  6.755831e-01  0.640063
LinearRegression -0.619357  0.693387  0.632062 -3.434417e+16  0.640505
ElasticNet       -0.557119  0.284941  0.351370  3.212503e-01  0.338040
SVR              -0.548532  0.237292  0.276478  3.144154e-02 -0.193004
GBR              -0.214820  0.845135  0.808219  7.725227e-01  0.798142
RF               -0.182532  0.880132  0.858520  6.940084e-01  0.892200

                         5
BayesianRidge     0.642755
LinearRegression  0.643590
ElasticNet        0.248526
SVR              -0.169742
GBR               0.758468
RF                0.833851
-------------------------------------------------------------------
regression metrics:
                     rmlse         mae           mse        r2
BayesianRidge     1.115382   74.427630  10060.655381  0.693369
LinearRegression  1.105990   74.650377  10065.174720  0.693231
ElasticNet        1.227738  102.496689  19154.306019  0.416211
SVR               1.202922  108.086857  26062.076203  0.205674
GBR               0.706672   47.343728   4738.600931  0.855576
RF                0.175767   11.509406    385.864961  0.988240
-------------------------------------------------------------------
short name       full name
rmsle     root_mean_square_logarithmic_error
mae       mean_absolute_error
mse       mean_squared_error
r2        r2
-------------------------------------------------------------------
```

- **Cross-Validation:6 folds**
- **Validation Metrics: RMLSE, MAE, MSE, R2**

Outcome Prediction Vs. True Y Value



4.3 Statistical Transformation On The Y-Variables:

Y in Squared Root:

regression metrics:

	rmlse	mae	mse	r2
GBR	0.435684	45.793685	4850.908911	0.860817
RF	0.159998	12.557421	460.803853	0.986779

Y in Logarithm

regression metrics:

	rmlse	mae	mse	r2
GBR	0.407384	46.835255	5395.414881	0.835678
RF	0.134694	11.535057	441.566363	0.986552

4.4 Compare with Kaggle-Base Case (Top 10 Percentile)

Kaggle Case:

regression metrics:

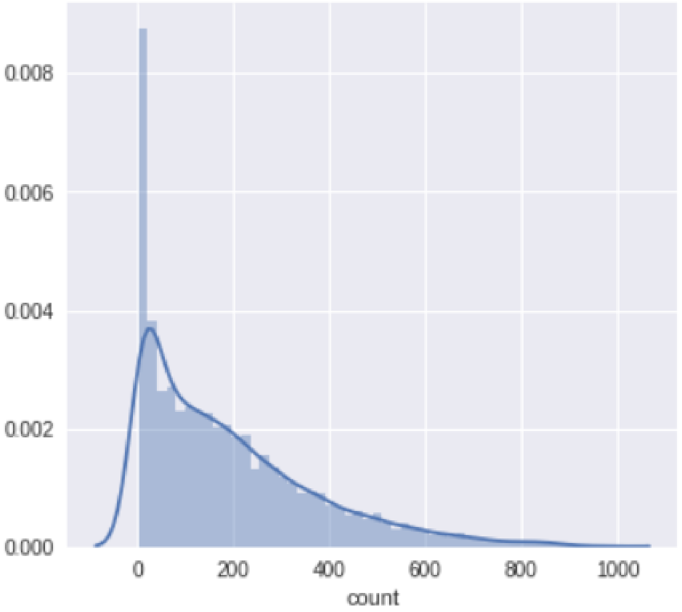
	rmlse	mae	mse	r2
BayesianRidge	0.978241	0.800255	1.032940	0.487290
LinearRegression	0.978240	0.800259	1.032912	0.487303
ElasticNet	1.013156	0.832484	1.110526	0.448779
SVR	0.409906	0.245153	0.202378	0.899548
GBR	0.353146	0.276657	0.137952	0.931526
RF	0.133898	0.090968	0.020325	0.989912

Ours(6-Folds):

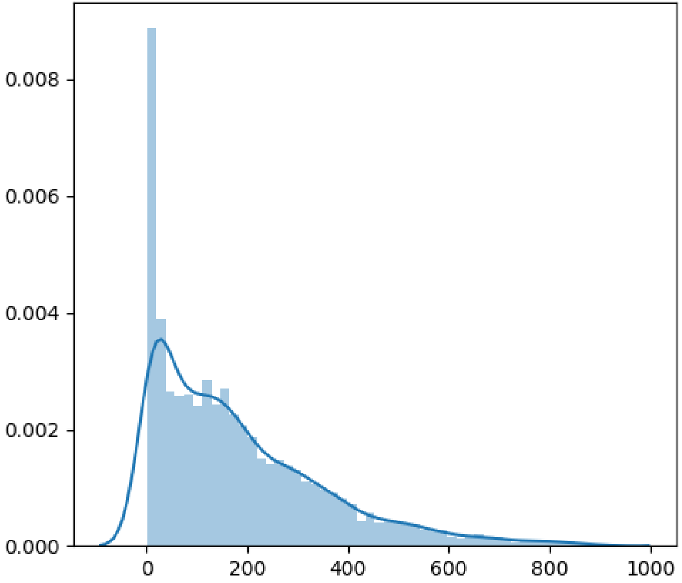
regression metrics:

	rmlse	mae	mse	r2
GBR	0.407384	46.835255	5395.414881	0.835678
RF	0.134694	11.535057	441.566363	0.986552

4.5 Prediction Using The Best Model:



Kaggle Base Case



Our Model

4.6.1 Fitting Model for 'Casual':

Raw Data

regression metrics:

	rmlse	mae	mse	r2
BayesianRidge	1.275335	23.452117	1178.754663	0.440934
LinearRegression	1.284229	23.456218	1178.777465	0.440923
ElasticNet	1.429756	27.456823	1720.158023	0.184155
SVR	0.646716	13.090897	733.301022	0.652206
KNN	0.594399	10.096663	298.858228	0.858256
GBR	0.838207	12.054156	405.884006	0.807495
RF	0.537747	8.751098	212.583055	0.899175

Y in Squared Root

regression metrics:

	rmlse	mae	mse	r2
BayesianRidge	1.334212	24.359849	1289.057044	0.425388
LinearRegression	1.308310	24.372991	1288.752373	0.425524
ElasticNet	1.464476	27.626707	1787.619402	0.203148
SVR	0.651753	13.241749	810.687243	0.638627
KNN	0.587349	9.627465	297.224038	0.867509
GBR	0.828001	12.068556	425.398057	0.810374
RF	0.563367	8.832952	222.268367	0.900921

4.6.1 Fitting Model for ‘Casual’:

Y in Log Shape

regression metrics:

	rmlse	mae	mse	r2
BayesianRidge	0.849416	21.037863	1666.115064	0.257310
LinearRegression	0.849410	21.042011	1667.613136	0.256643
ElasticNet	1.164606	27.704291	2439.721616	-0.087533
SVR	0.445502	9.015922	289.902731	0.870773
KNN	0.462232	9.634476	338.377503	0.849164
GBR	0.459998	10.509177	429.463586	0.808562
RF	0.437514	8.618932	234.340680	0.895540

Add Quadratic Function for continuous variables: Humidity,Windspeed,Temp

regression metrics:

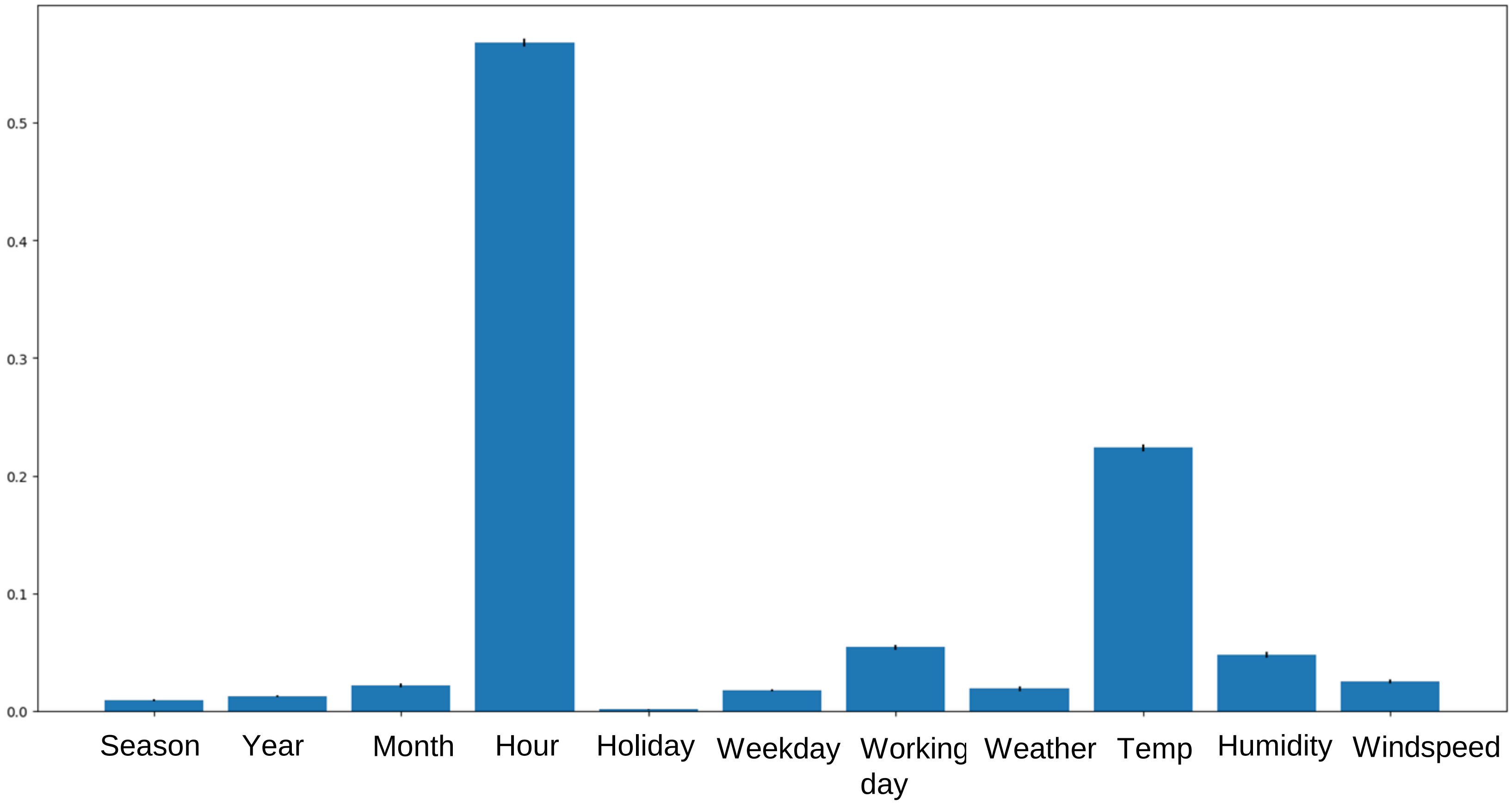
	rmlse	mae	mse	r2
BayesianRidge	1.053747	19.714371	1215.313931	0.458260
LinearRegression	1.053808	19.713401	1215.065198	0.458371
ElasticNet	1.353550	27.398011	2178.950674	0.028708
SVR	0.554822	9.832069	385.820828	0.828016
KNN	0.557049	9.568785	318.446308	0.858049
GBR	0.576497	10.332912	395.359313	0.823764
RF	0.541660	8.830216	251.073483	0.888081

4.6.2 Parameter Tuning:

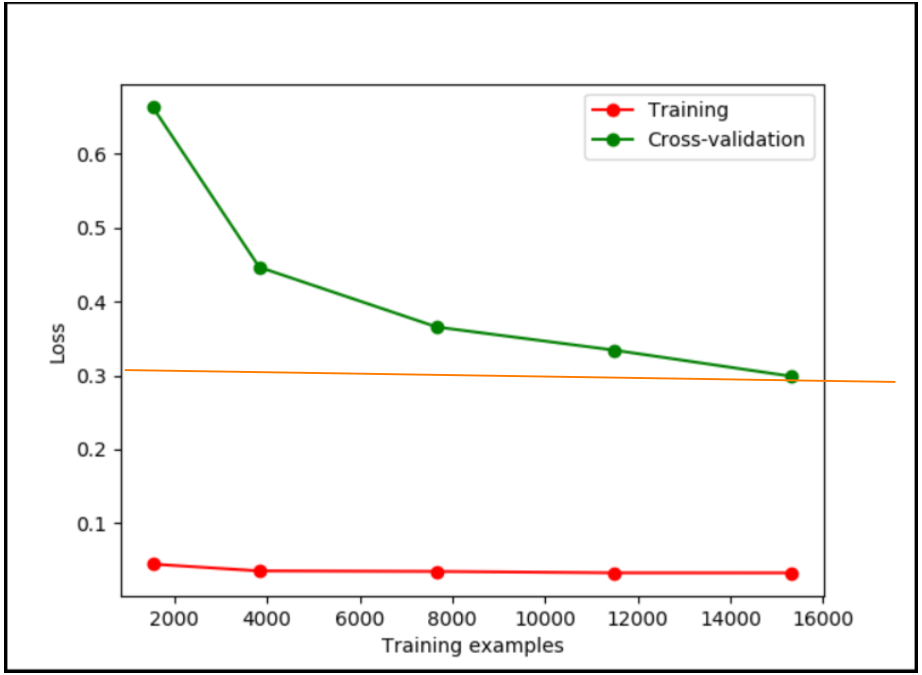
```
In [20]: from sklearn.grid_search import GridSearchCV
    ...:
    ...: param_test1 = {
    ...:     'n_estimators': [200,900]
    ...: }
    ...: gsearch1 = GridSearchCV(estimator = model_rf, param_grid = param_test1, scoring='neg_mean_squared_log_error')
    ...: gs = gsearch1.fit(x_test,y_test)
    ...:

In [21]:  gsearch1.best_params_
Out[21]: ({'n_estimators': 900}
```

```
In [22]: from sklearn.ensemble import RandomForestRegressor
    ...: rfModel = RandomForestRegressor(n_estimators = 900)
    ...: rfModel.fit(x_train,y_train)
    ...: preds = rfModel.predict(X= x_test)
    ...: print ("RMSLE Value For Random Forest: ",rmsle(np.exp(y_test),np.exp(preds)))
    ...:
RMSLE Value For Random Forest:  0.417360090269
```

Feature Significance Of The Parameters

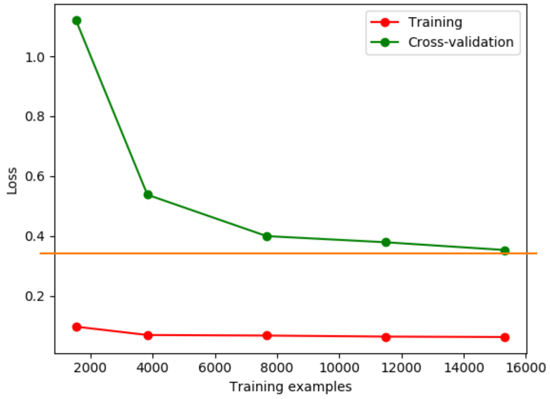


4.6.3 Learning Curve: Random Forest



Our Model

Conclusion:
-Logarithm combine with random forest improved the base model.
-Cross-Validation prevent overfitting.
-More data needed to avoid the variance.

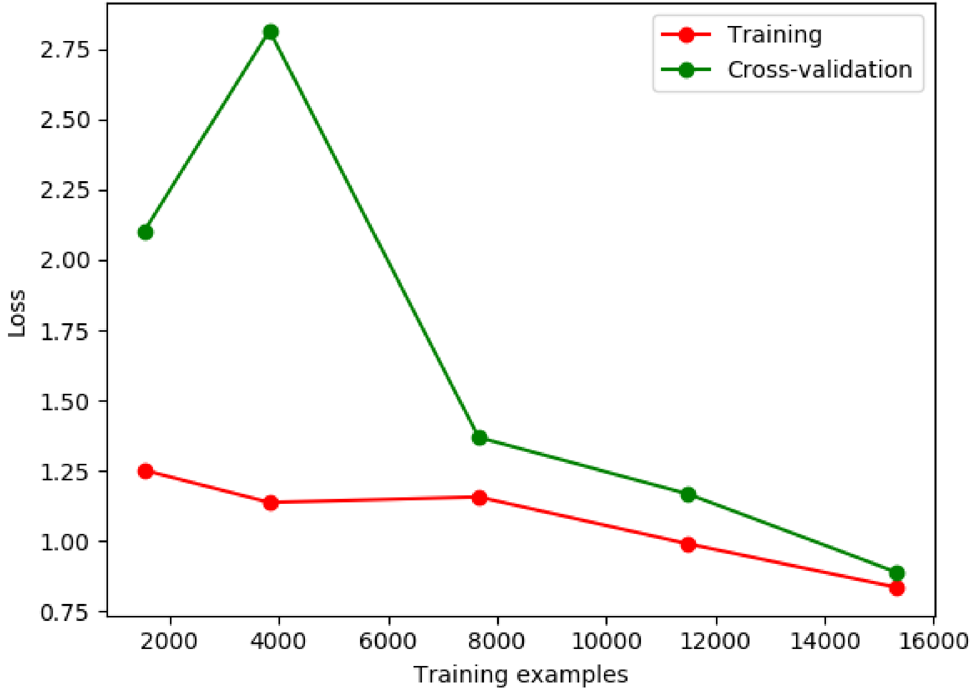


Base Case

4.7.1 Deep Learning： Neural Network

```
In [2]: # define base model
   ...: def baseline_model():
   ...:     # create model
   ...:     model = Sequential()
   ...:     model.add(Dense(11, input_dim=11, kernel_initializer='normal', activation='relu'))
   ...:     model.add(Dense(1, kernel_initializer='normal'))
   ...:     # Compile model
   ...:     model.compile(loss='mean_squared_error', optimizer='adam')
   ...:     return model
   ...:
   ...: # fix random seed for reproducibility
   ...: seed = 7
   ...: numpy.random.seed(seed)
   ...: # evaluate model with standardized dataset
   ...: estimator = KerasRegressor(build_fn=baseline_model, nb_epoch=100, batch_size=5, verbose=0)
   ...: kfold = KFold(n_splits=10, random_state=seed)
   ...: results = cross_val_score(estimator, x, y, cv=kfold)
   ...: print("Results: %.2f (%.2f) mse" % (results.mean(), results.std()))
```

4.7.2 Learning Curve: Neural Network (Metric: M.S.E)



Conclusion：
-Compare with Random Forest，Neural Network is more advanced in performance of accuracy and case generalization.
-the trend indicated that with more data, the performance may be even better.

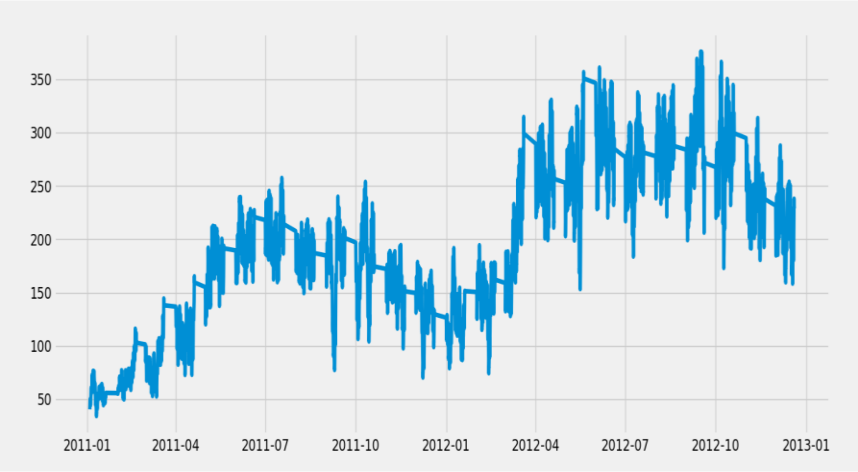
4.8 Time Series: What makes Time Series Special?

datetime	count
2011-01-01 00:00:00	16
2011-01-01 01:00:00	40
2011-01-01 02:00:00	32
2011-01-01 03:00:00	13
2011-01-01 04:00:00	1

1. **It is time dependent.**
2. **Seasonality trends is variations specific to a particular time frame**

4.8 Time Series: How to Check Stationarity of a Time Series?

```
plt.plot(ts)
```



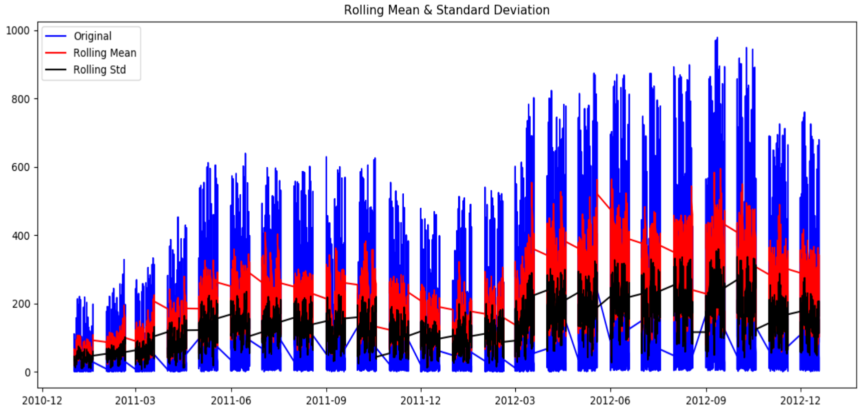
1. **Constant Mean**

1. **Constant Variance**

1. **An autocovariance that does not depend on time**

4.8 Time Series: How to Check Stationarity of a Time Series?

Plotting Rolling Statistics



- **Constant Mean**
- **Constant Variance**
- **An autocovariance that does not depend on time**

Dickey-Fuller Test

```
Test Statistic                 -6.419976e+00
p-value                         1.801620e-08
#Lags Used                      3.600000e+01
Number of Observations Used     1.084900e+04
Critical Value (1%)            -3.430953e+00
Critical Value (5%)            -2.861806e+00
Critical Value (10%)           -2.566912e+00
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary
```

4.8 Time Series: Estimating & Eliminating Trend

1. Estimating & Eliminating Trend: **Transformation**
2. Eliminating Trend and Seasonality

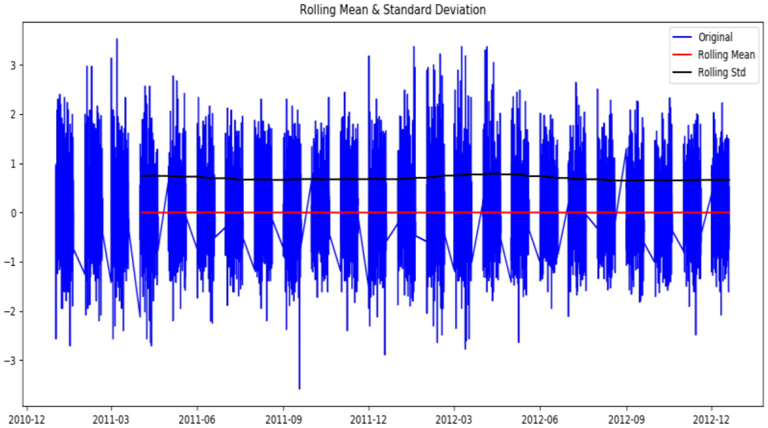
Differencing – taking the difference with a particular time lag
Decomposition – modeling both trend and seasonality and removing them from the model

```
ts_log_diff = ts_log - ts_log.shift()

plt.plot(ts_log_diff)
```

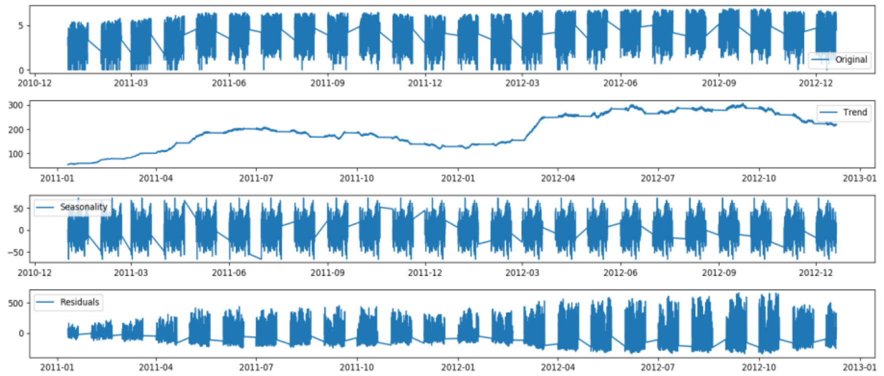
Differencing : Dickey-Fuller Test

```
In [54]: TestStationaryAdfuller(ts_log_diff)
Test Statistic                    -22.930693
p-value                             0.000000
#Lags Used                         39.000000
Number of Observations Used     10845.000000
Critical Value (1%)                -3.430953
Critical Value (5%)                -2.861807
Critical Value (10%)               -2.566912
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary
```

Differencing :Plotting Rolling Statistics



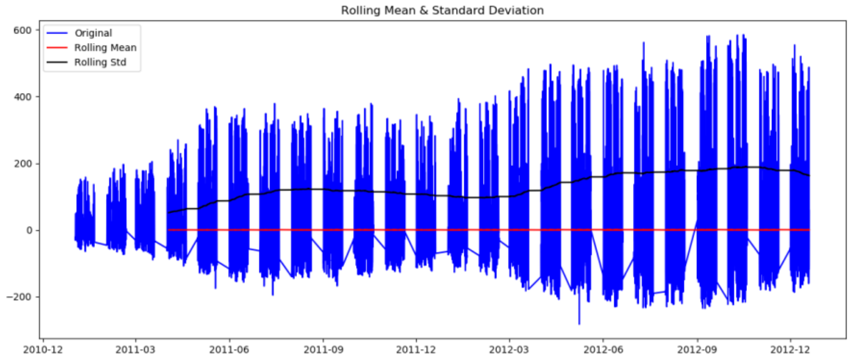
4.8 Time Series: Decomposing



Decomposing : Dickey-Fuller Test

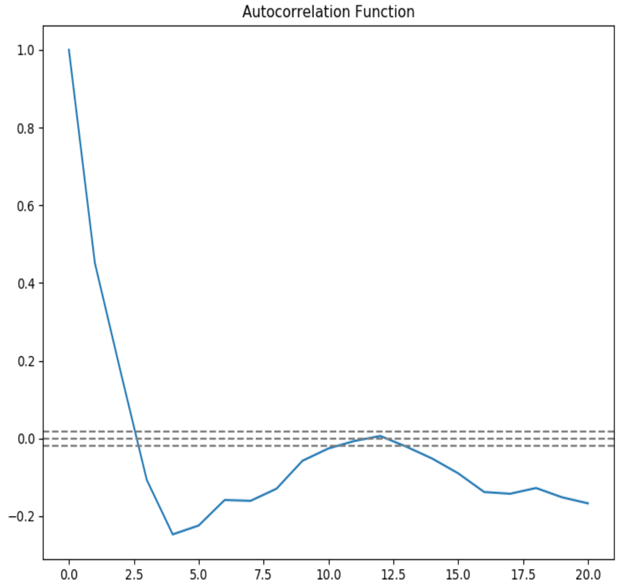
```
In [72]: TestStationaryAdfuller(ts_log_decompose)
Test Statistic                -30.955728
p-value                         0.000000
#Lags Used                     39.000000
Number of Observations Used 10832.000000
Critical Value (1%)            -3.430954
Critical Value (5%)            -2.861807
Critical Value (10%)           -2.566912
dtype: float64
Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root, hence it is stationary
```

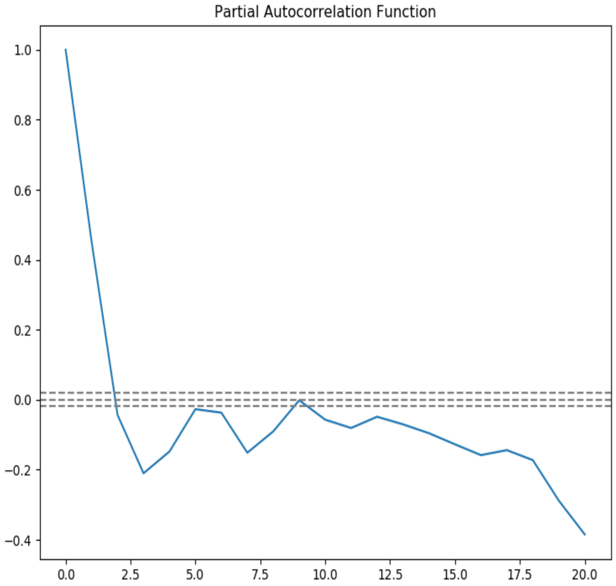
Decomposing :Plotting Rolling Statistics



4.8 Time Series: Forecasting a Time Series : ACF,PACF

ARIMA : Auto-Regressive Integrated Moving Averages.





In this plot, the two dotted lines on either sides of 0 are the confidence interevals. These can be used to determine the 'p' and 'q' values as:

1.p – The lag value where the **PACF** chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=2.

2.q – The lag value where the **ACF** chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=2.

4.8 Time Series: Forecasting a Time Series

```
In [52]: mod = sm.tsa.statespace.SARIMAX(ts_log, ^M
    ...:                                 order=(2,1,2), ^M
    ...:                                 seasonal_order=(2,1,2,12),   ^M
    ...:                                 enforce_stationarity=False,^M
    ...:                                 enforce_invertibility=False)
    ...:
    ...:

In [53]: results = mod.fit()^M
    ...: print(results.summary())
```

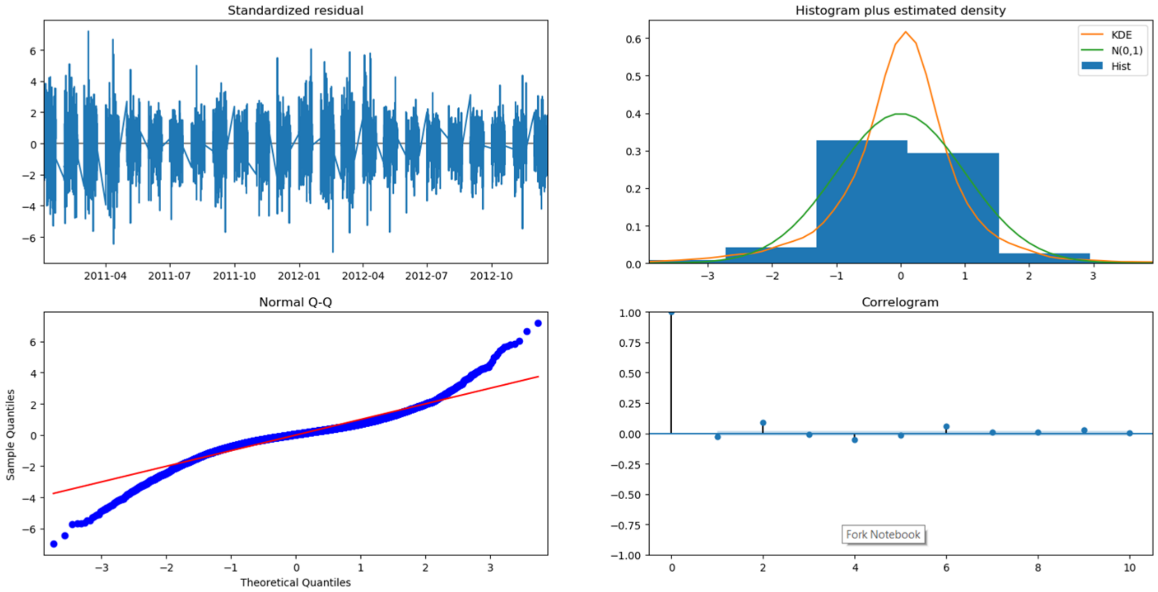
Validating

```
In [54]: pred = results.get_prediction(start = 10880,end = 10886, dynamic=False)^M
    ...: pred_ci = pred.conf_int()^M
    ...: pred_ci.head()
    ...:
Out[54]:
      lower count  upper count
10880     5.295345     7.056084
10881     5.124588     6.885327
10882     4.530270     6.291009
10883     4.203593     5.964332
10884     3.786981     5.547720
```

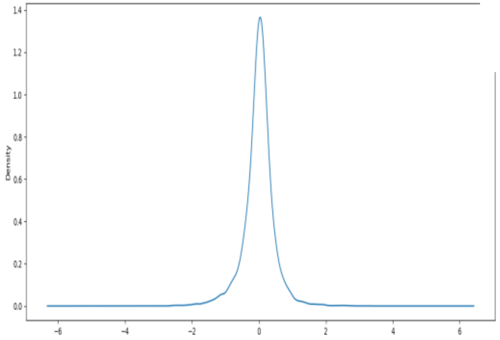
```
The Mean Squared Error (MSE) of the forecast is 4.09
```

```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                              count   No. Observations:                10886
Model:             SARIMAX(2, 1, 2)x(2, 1, 2, 12)   Log Likelihood               -6746.927
Date:                            Fri, 23 Feb 2018   AIC                          13511.854
Time:                                    19:53:09   BIC                          13577.511
Sample:                                01-01-2011   HQIC                         13533.985
                                     - 12-19-2012
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.4434      0.015     97.891      0.000       1.415       1.472
ar.L2         -0.6612      0.011    -61.377      0.000      -0.682      -0.640
ma.L1         -1.4969      0.017    -88.834      0.000      -1.530      -1.464
ma.L2          0.4977      0.017     29.515      0.000       0.465       0.531
ar.S.L12      -0.8602      0.013    -67.917      0.000      -0.885      -0.835
ar.S.L24       0.1289      0.012     10.362      0.000       0.105       0.153
ma.S.L12      -0.2311      0.005    -47.661      0.000      -0.241      -0.222
ma.S.L24      -0.9009      0.008   -115.338      0.000      -0.916      -0.886
sigma2         0.1757      0.002    104.293      0.000       0.172       0.179
===================================================================================
Ljung-Box (Q):                      818.05   Jarque-Bera (JB):             11526.05
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.54   Skew:                            -0.36
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.00
===================================================================================
```

4.8 Time Series: Forecasting a Time Series



Residual plot



5 Recommendations

- **Better distribution of bikes**
- **Improve bike sharing service & customer satisfaction**
- **Friendly to environment and personal health**
- **Data collection**

THANK YOU

Team4 | OPIM5894

Appendix

Modeling Fitting Results : LASSO

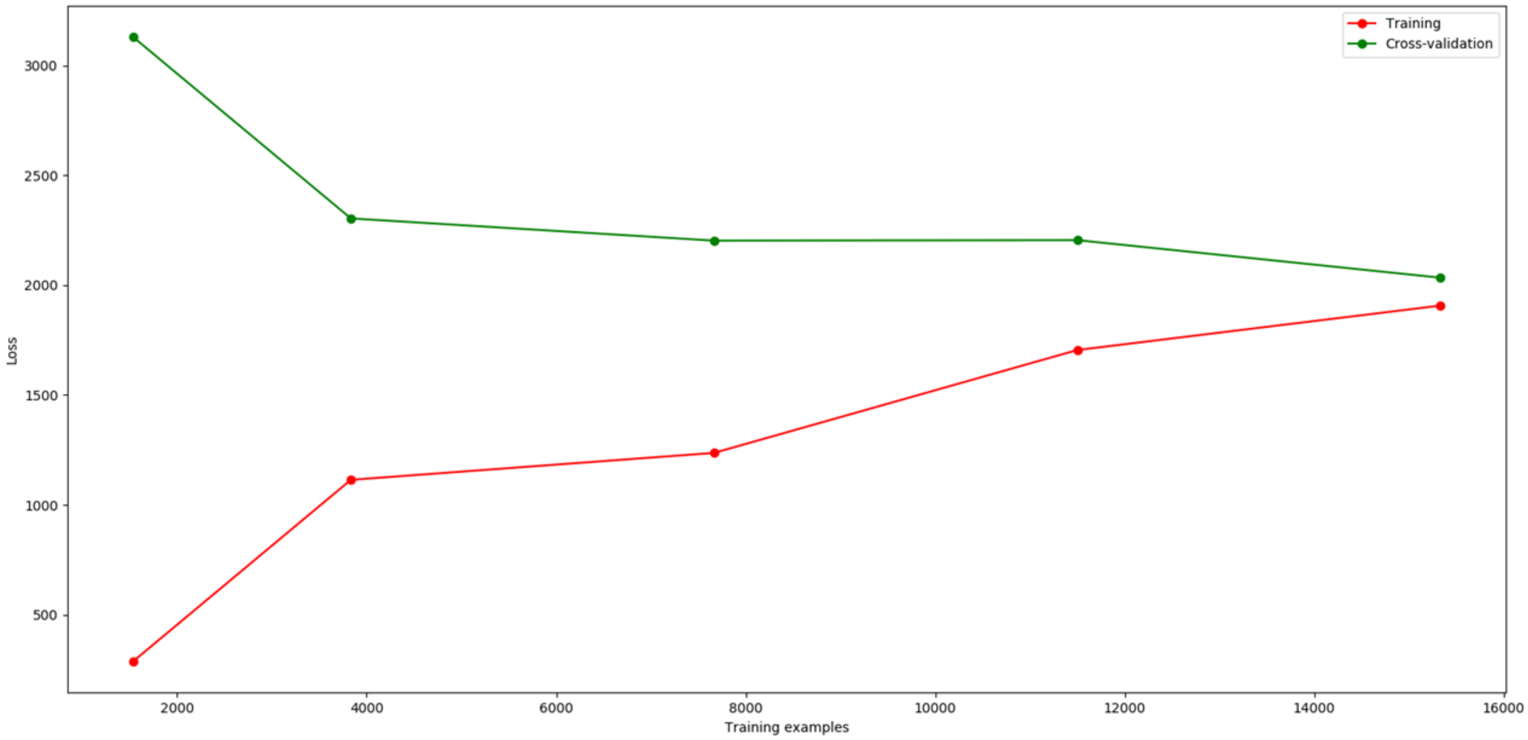
```
{'alpha': 0.0050000000000000001, 'max_iter': 3000}
RMSLE Value For Lasso Regression:  0.978133604313
```

2.4.2 Using Random Forest To Predict "Windspeed = 0"

```
   ...: from sklearn.ensemble import RandomForestRegressor
   ...:
   ...: dataWind0 = data[data["windspeed"]==0]
   ...: dataWindNot0 = data[data["windspeed"]!=0]
   ...: rfModel_wind = RandomForestRegressor()
   ...: windColumns = ["season","weather","humidity","month","temp","year","atemp"]
   ...: rfModel_wind.fit(dataWindNot0[windColumns], dataWindNot0["windspeed"])
   ...:
   ...: wind0Values = rfModel_wind.predict(X= dataWind0[windColumns])
   ...: dataWind0["windspeed"] = wind0Values
   ...: data = dataWindNot0.append(dataWind0)
   ...: data.reset_index(inplace=True)
   ...: data.drop('index',inplace=True,axis=1)
   ...:
```

2.4.2 Using Random Forest To Predict “Windspeed = 0”

```
...: from sklearn.ensemble import RandomForestRegressor
...:
...: dataWind0 = data[data["windspeed"]==0]
...: dataWindNot0 = data[data["windspeed"]!=0]
...: rfModel_wind = RandomForestRegressor()
...: windColumns = ["season","weather","humidity","month","temp","year","atemp"]
...: rfModel_wind.fit(dataWindNot0[windColumns], dataWindNot0["windspeed"])
...:
...: wind0Values = rfModel_wind.predict(X= dataWind0[windColumns])
...: dataWind0["windspeed"] = wind0Values
...: data = dataWindNot0.append(dataWind0)
...: data.reset_index(inplace=True)
...: data.drop('index',inplace=True,axis=1)
...:
```


Training
Cross-validation
3000
2500
2000
1500
1000
500
Loss
2000
4000
6000
8000
10000
12000
14000
16000
Training examples

```
In [30]: importance
Out[30]:
array([0.00934437, 0.01262281, 0.02177616, 0.56781821, 0.00119953,
       0.01741187, 0.05423668, 0.01897225, 0.22362681, 0.04782957,
       0.02516175])
```

```
In [115]: def rmsle(y, y_):
     ...:     log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))
     ...:     log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))
     ...:     calc = (log1 - log2) ** 2
     ...:     return np.sqrt(np.mean(calc))
     ...:

In [116]: rmsle(y_test,prediction)
C:\Users\Ming\Anaconda3\Scripts\ipython:3: RuntimeWarning: invalid value encountered in log
Out[116]: 2.1415675392742197
```

Evaluation Of Model

We will be evaluating our model on the basis of Root Mean Square Log Error (RMSLE). RMSLE is calculated as:

$$\varepsilon = \sqrt{(1/n) \sum_{i=1}^{n} (log(p_i + 1) - log(a_i + 1))^2}$$

Where, p_i is the predicted value, a_i is the actual value and n is the total number of samples.

RMSLE is suitable for our problem because RMSLE penalises an underprediction more than an over prediction. The bike sharing company would lose revenue if the number of bikes will be less than the demand for the bikes.

tensorflow

```
x = sm.add_constant(x)
x = tf.Variable(x, dtype=tf.float32, name= "x")
y = tf.Variable(y.reshape(-1,1),dtype=tf.float32, name=
"y")
xt = tf.transpose(x)
theta =
tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(xt,x)),xt)
,y)
init = tf.global_variables_initializer()
with tf.Session() as sess:
    init.run()
    theta_value = theta.eval()
theta_value
```

Keras Neural Network

```
# define base model
def baseline_model():
	# create model
	model = Sequential()
	model.add(Dense(11, input_dim=11, kernel_initializer='normal', activation='relu'))
	model.add(Dense(1, kernel_initializer='normal'))
	# Compile model
	model.compile(loss='mean_squared_error', optimizer='adam')
	return model

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# evaluate model with standardized dataset
estimator = KerasRegressor(build_fn=baseline_model, nb_epoch=100, batch_size=5, verbose=0)
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(estimator, x, y, cv=kfold)
print("Results: %.2f (%.2f) mse" % (results.mean(), results.std()))
```