Project Report

Bike Rental Prediction

Data Science

(Submitted By: Ankit Anand)

Abstract— In this report, the count of a bike rented is detected using machine learning algorithms. From the given data, a feature of different set of variables is extracted by analysing the correlation plot. Different variables which are not useful for predicting the target variable are dropped. Using this feature data, different models are built: Decision Tree, Linear Regression, Random Forest, KNN classifier, KFold cross validation. Final prediction of the bike rental count on daily based on the environmental and seasonal settings is made from the model. From the given model's best performance was with Random Forest model, which is then implemented on the given test data for the final bike rental count.

Contents

Chapter 1

Introduction

1.1 Problem Statement

The objective of this case is Predication of bike rental count on daily based on the environmental and seasonal settings.

Given is the set of data with different variables like 'season', 'year', 'month', holiday', 'weekday', 'workingday', 'weather', 'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered'. On the basic these conditions we have to predict the amount of Bike rental. The main objective is making company utilize their resources and manpower based on the count of the bike rented. So, with accurate prediction we can manipulate the amount of bike on the basic of the demand and different environment conditions .By this company can maximize their profit and in return helps in positive growth of the company.

Chapter 2

Business Understanding

There are two main tasks addressed in this stage:

- **Define objectives**: Work with our customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals that the data science techniques can target.
- **Identify data sources:** Find the relevant data that helps to answer the questions that define the objectives of the project.

2.1 Define objective

1. A central objective of this step is to identify the key business variables that the analysis needs to predict. These variables are referred as the *model targets*, and the metrics associated with them is used to determine the success of the project.
2. Define the project goals by asking and refining "sharp" questions that are relevant, specific, and unambiguous. Data science is a process that uses names and numbers to answer such questions. We typically use data science or machine learning to answer five types of questions:
    - How much or how many? (regression)
    - Which category? (classification)
    - Which group? (clustering)
    - Is this weird? (anomaly detection)
    - Which option should be taken? (recommendation)

   We determine which of these questions we're asking and how answering it achieves our business goals.

2.2 Identify data sources

Identify data sources that contain known examples of answers to our sharp questions. Look for the following data:

- Data that's relevant to the question. Do we have measures of the target and features that are related to the target?
- Data that's an accurate measure of our model target and the features of interest.

Here we are given with the set of data which includes different environment conditions . Based on the this we have to predict what will the expected count of bike which will be rented. By correctly predicting the count the company will be able to allocate its resources that when will the count expected to rise and when it will fall. And, so this will make the company to apply management skills for the workforce.

Chapter 3

Data Understanding

Our task is to build regression models which will predict the count of bike rented depending upon different environmental and seasonal settings .

The details of data attributes in the dataset are as follows –

instant: Record index
dteday: Date
season: Season (1:springer, 2:summer, 3:fall, 4:winter)
yr: Year (0: 2011, 1:2012)
mnth: Month (1 to 12)
hr: Hour (0 to 23)
holiday: weather day is holiday or not (extracted fromHoliday Schedule)
weekday: Day of the week
workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
weathersit: (extracted fromFreemeteo)
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: Normalized temperature in Celsius. The values are derived via
(t-t_min)/(t_max-t_min),
t_min=-8, t_max=+39 (only in hourly scale)
atemp: Normalized feeling temperature in Celsius. The values are derived via
(t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)
hum: Normalized humidity. The values are divided to 100 (max)
windspeed: Normalized wind speed. The values are divided to 67 (max)
casual: count of casual users
registered: count of registered users
cnt: count of total rental bikes including both casual and registered
Given below is a sample of the data set that we are using to predict the count of bike rented:

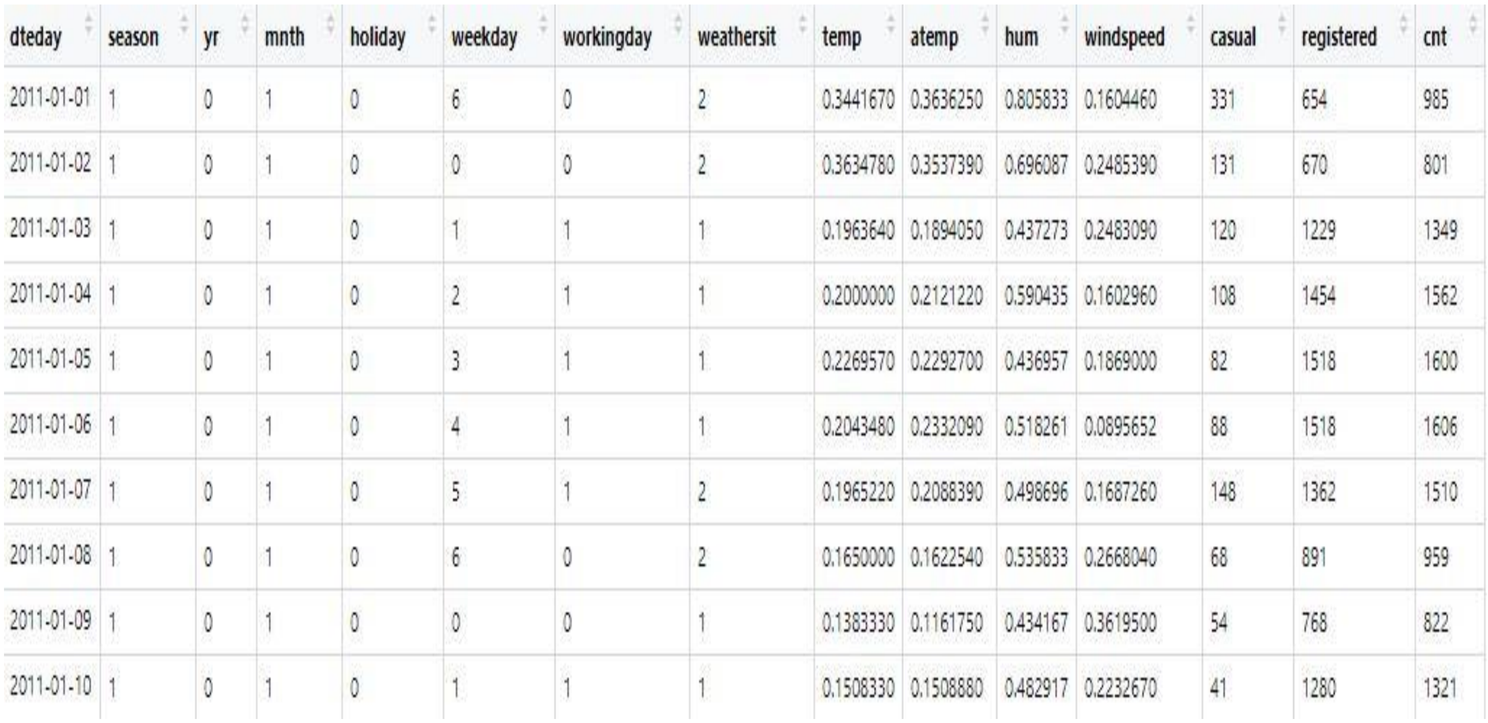
Missing values = No

1) **Day.csv data**: Count-731, Variables- 15

Here, **Target Variable** = count

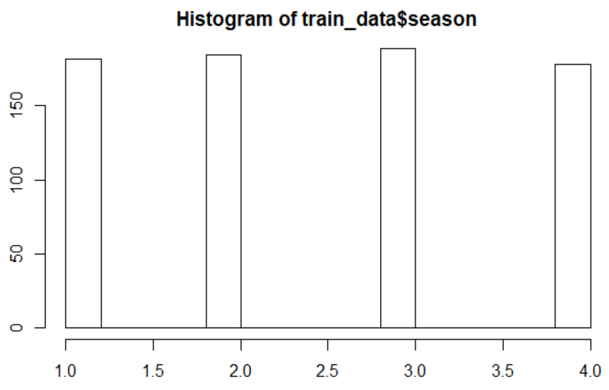
Independent Variables= dteday, season, yr, mnth, holiday, weekday, workingday, weathersit, temp, atemp, hum, windspeed, casual, register

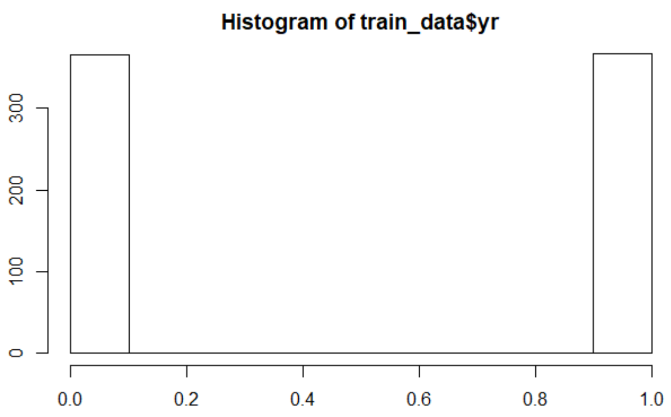
Data.head()

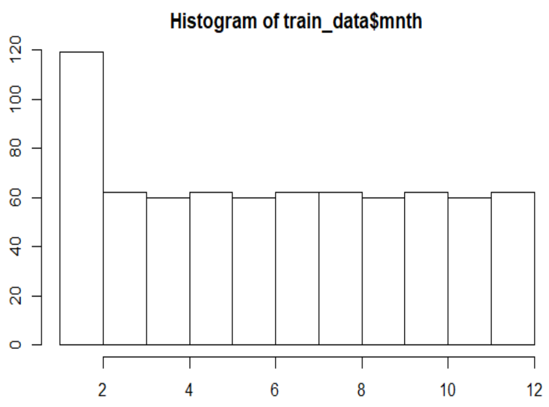


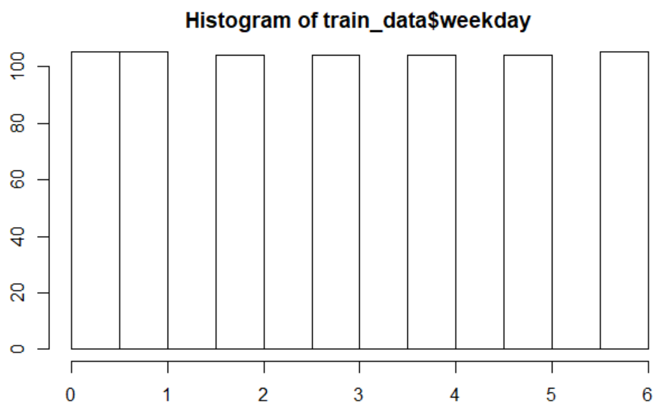
dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2011-01-01	1	0	1	0	6	0	2	0.3441670	0.3636250	0.805833	0.1604460	331	654	985
2011-01-02	1	0	1	0	0	0	2	0.3634780	0.3537390	0.696087	0.2485390	131	670	801
2011-01-03	1	0	1	0	1	1	1	0.1963640	0.1894050	0.437273	0.2483090	120	1229	1349
2011-01-04	1	0	1	0	2	1	1	0.2000000	0.2121220	0.590435	0.1602960	108	1454	1562
2011-01-05	1	0	1	0	3	1	1	0.2269570	0.2292700	0.436957	0.1869000	82	1518	1600
2011-01-06	1	0	1	0	4	1	1	0.2043480	0.2332090	0.518261	0.0895652	88	1518	1606
2011-01-07	1	0	1	0	5	1	2	0.1965220	0.2088390	0.498696	0.1687260	148	1362	1510
2011-01-08	1	0	1	0	6	0	2	0.1650000	0.1622540	0.535833	0.2668040	68	891	959
2011-01-09	1	0	1	0	0	0	1	0.1383330	0.1161750	0.434167	0.3619500	54	768	822
2011-01-10	1	0	1	0	1	1	1	0.1508330	0.1508880	0.482917	0.2232670	41	1280	1321

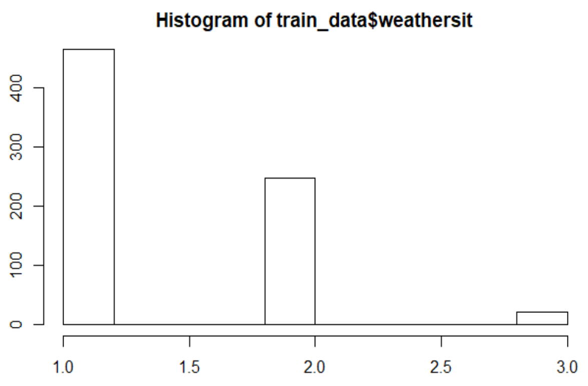
Visualization Plot of different variables:

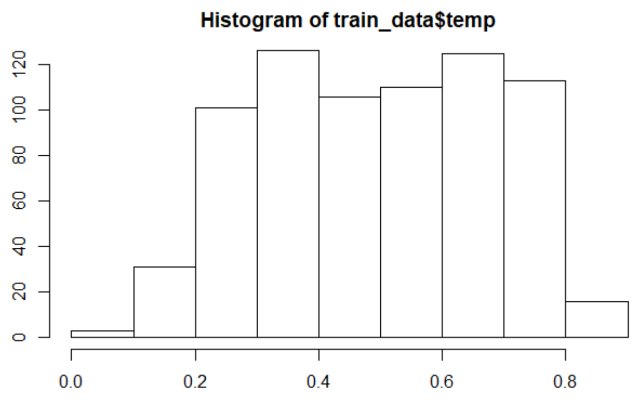


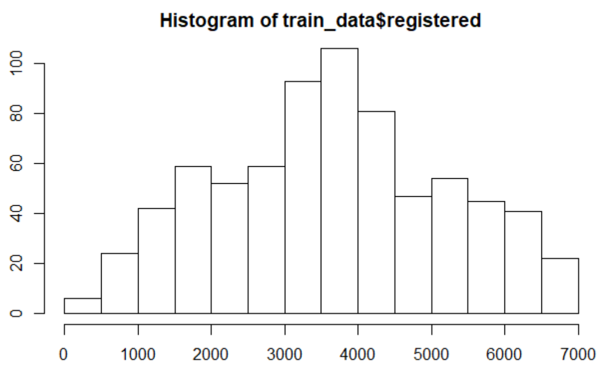


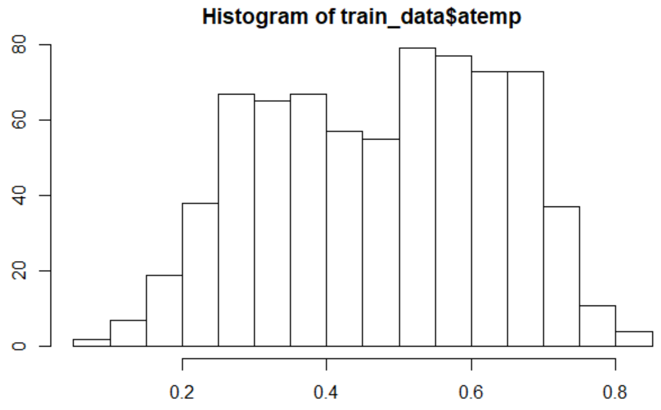


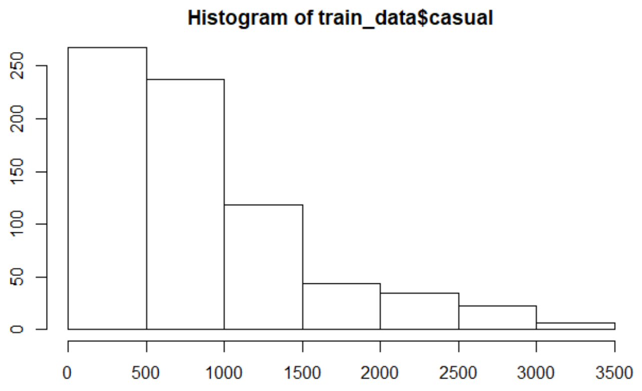


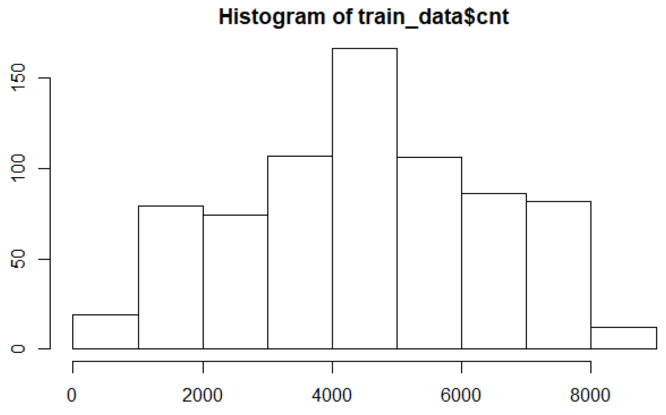

Histogram of train_data$weathersit
0
100
200
300
400
1.0
1.5
2.0
2.5
3.0


Histogram of train_data$temp
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8


Histogram of train_data$registered
0
20
40
60
80
100
0
1000
2000
3000
4000
5000
6000
7000


Histogram of train_data$atemp
0
20
40
60
80
0.2
0.4
0.6
0.8


Histogram of train_data$casual
0
50
100
150
200
250
0
500
1000
1500
2000
2500
3000
3500


Histogram of train_data$cnt
0
50
100
150
0
2000
4000
6000
8000

Chapter 4

Data Preparation

Goals

- Produce a clean, high-quality data set whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so that it is ready to model.

Before we train our models, we need to develop a sound understanding of the data. Real-world data sets are often noisy, are missing values, or have a host of other discrepancies. We can use data summarization and visualization to audit the quality of your data and provide the information we need to process the data before it's ready for modelling. This process is often iterative.

Missing Value analysis.

The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly, then we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from ones where the missing values are present.

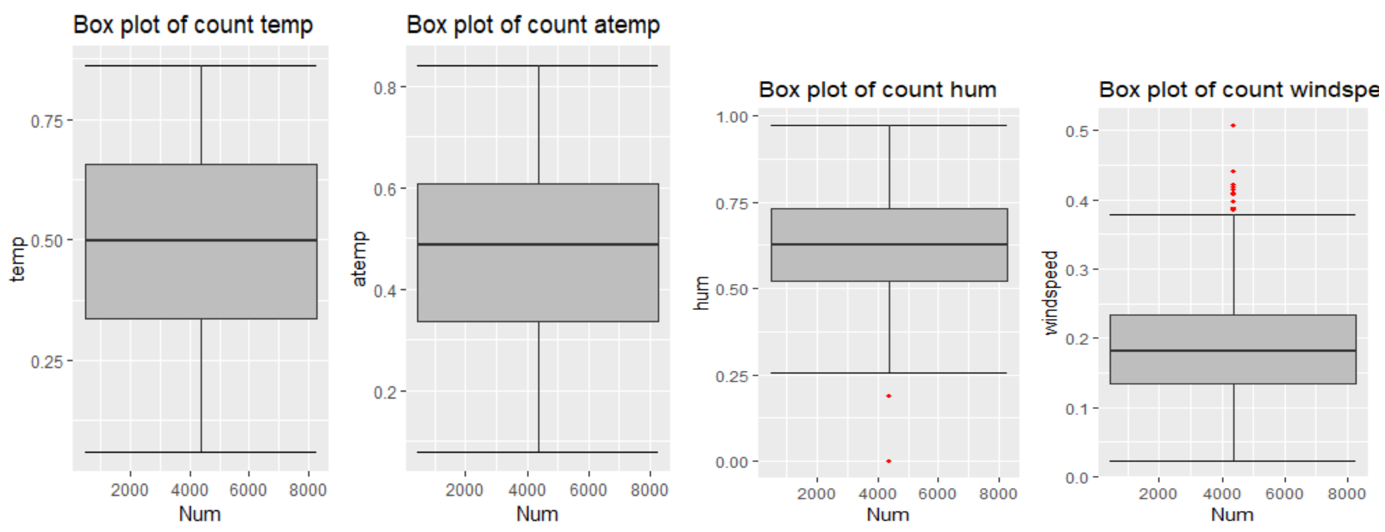
Percent of missing values in our data before imputation:

instant	0
dteday	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
casual	0
registered	0
cnt	0

Outlier Analysis:

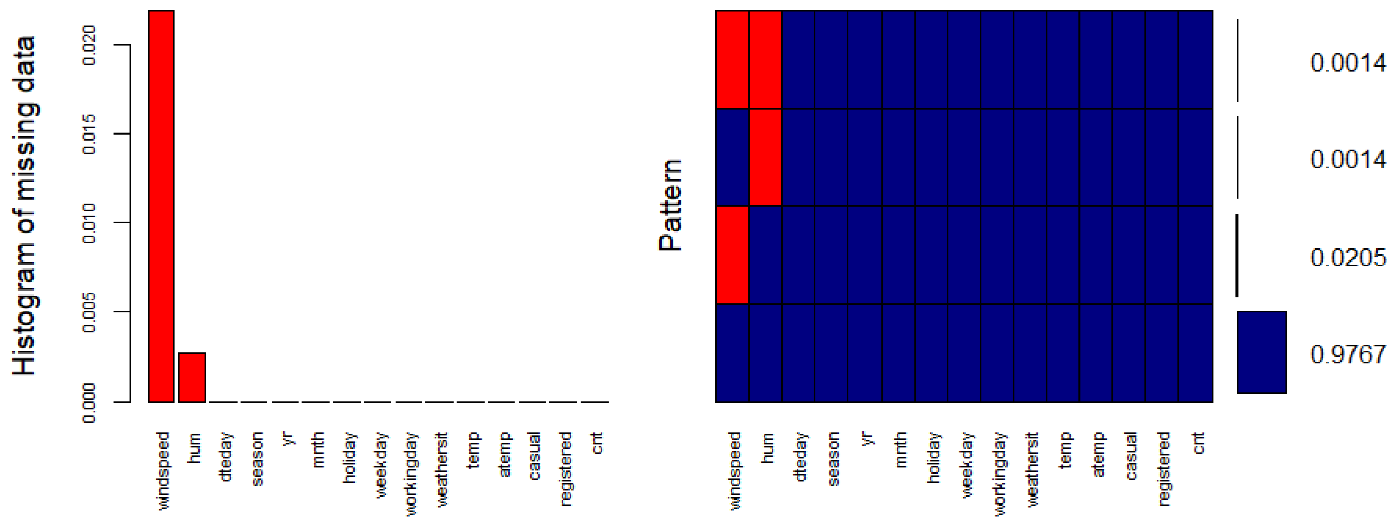
Observations inconsistent with rest of the dataset Global Outlier.
Causes of Outliers:
• Poor data quality / contamination
• Low quality measurements, malfunctioning equipment, manual error
• Correct but exceptional data
To detect the Outlier in the data – Graphical approach- Box Plot method.

The outliers in our data are.



So, in our variable there is outlier in only in variable- hum, windspeed.

After replacing the outlier with NA , number of missing terms in our data in 15. The missing values are visualizes as shown:



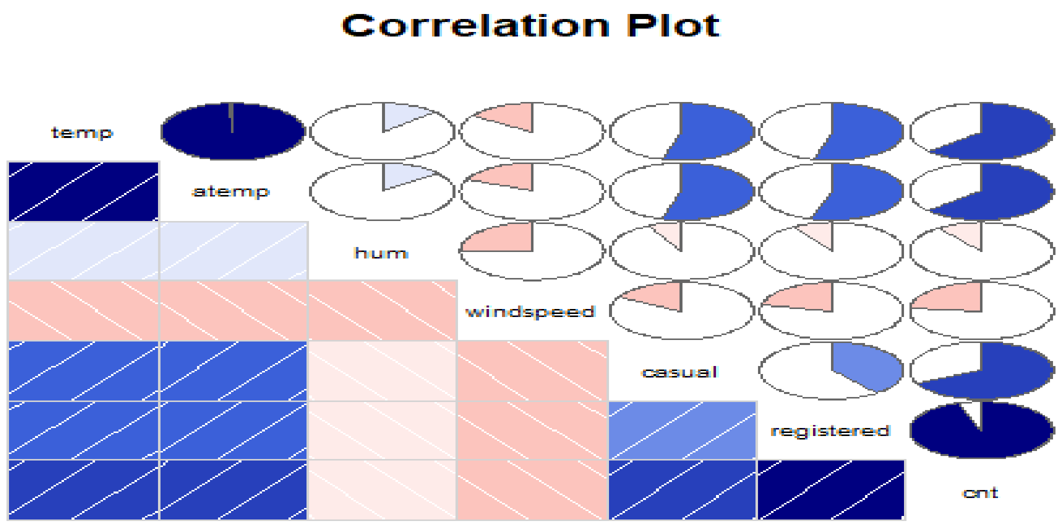
After the outlier are removed and filled with NA, then using the MICE method these NA are replaced. These are removed by using CART method under MICE . And then our data is clean, and no missing values are available.

Feature Selection:

Selecting a subset of relevant features (variables, predictors) for use in model construction. Subset of a learning algorithm's input variables upon which it should focus attention, while ignoring the rest.
Feature Selection is the process where we automatically or manually select those features which contribute most to our prediction variable or output in which we are interested in.

The main criterial of deciding the features between the variables is the correlation analysis. It is performed on the numerical variable. Whether to select a variable or remove it depends on how they are corelated. Ideally there should be no correlation between two independent variables but high correlation between the dependent and independent variable.

The correlation between the numeric variables is as shown.



After the analysis of the correlation plot , the results obtained are:

- Variable temp is highly correlated with variable atemp, so variable atemp is removed
- Target variable count is summation of the variable causal and registered, so these two variables are removed
- Variable hum is very less corelated with the target variable Count , so it is removed.
- Variable dteday is only displaying the date and not contributing to the model prediction, so it is also removed.
- Variable workingday is highly corelated with weekday , so it is also removed

From the correlation plot using feature selection , variables dteday, atemp , causal, registered, workingday are removed from the data as this are not important for the analysis.

Chapter 5

Modelling

In our data we have the knowledge of the output and also learning is in presence of the independent variables, so this modelling can be classified under Supervised Learning. Also, the variables are labelled, and we have to predict the values of the continuous variable i.e. count, so the modelling method to be used is of Regression. Different model under Supervised Learning for Regression problem should have to be checked. On the basic of the performance of the model, the best model will be chosen for prediction of the count of our test data.

1.Decision Tree.

A predictive model based on a branching series of Boolean tests. It Can be used for classification and regression. Decision tree is a rule. Each branch connects nodes with "and" and multiple branches are connected by "or". A decision tree is drawn upside down with its root at the top.

The train data is first split into train and test set of data in a ratio of 70:30.
In python from sklearn library, train_test_split and DecisionTreeRegressor are imported. Using the fit command, the X_train and y_train data are fitted and using predict X_test the final prediction is made.
Summary of the model developed using Tree in R:

```
Call:
rpart(formula = cnt ~ ., data = train_data, method = "anova")
  n= 731

          CP        nsplit   rel error     xerror       xstd
1  0.39050947     0    1.0000000  1.0030724   0.04040584
2  0.21194555     1    0.6094905  0.6119718   0.02696435
3  0.08018431     2    0.3975450  0.3996971   0.02560430
4  0.03971591     3    0.3173607  0.3212270   0.02276699
5  0.03364356     4    0.2776447  0.2926581   0.02297449
6  0.01418363     5    0.2440012  0.2587517   0.02288942
7  0.01184417     6    0.2298176  0.2497486   0.02080256
8  0.01158174     7    0.2179734  0.2383513   0.01996828
9  0.01100289     8    0.2063916  0.2346196   0.02013552
10 0.01024250     9    0.1953888  0.2271538   0.02004989
11 0.01000000    10    0.1851463  0.2235926   0.01992123

Variable importance
    temp     mnth    season      yr   windspeed   weathersit
     29       25        20       19       3            2
```

Performance tuning in Decision Tree in Python:

1. Using Default Value:

```
DecisionTreeRegressor(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=False, random_state=None, splitter='best')
```

Its Results:

```
R2 score : 0.82
Root Mean squared error: 822.47
Mean Absolute error: 572.52
MAPE: 18.21
Accuracy: 81.79
```

2. Using Randomized Search :

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=DecisionTreeRegressor(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=False,
                                                   random_state=None,
                                                   splitter='best'),
                   iid='warn', n_iter=10, n_job...e,
                   param_distributions={'criterion': ['mse', 'mae'],
                                        'max_depth': [6, None],
                                        'max_features': <scipy.stats._distn_infrast
ructure.rv_frozen object at 0x000002A0EB971BA8>,
                                        'min_samples_leaf': <scipy.stats._distn_inf
rastructure.rv_frozen object at 0x000002A0EB971D30>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

Its Result:

```
Tuned Decision Tree Parameters: {'criterion': 'mse',
                                 'max_depth': None,
                                 'max_features': 7,
                                 'min_samples_leaf': 6}
Best score is 0.7958916053204723
```

3. Using Grid Search :

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeRegressor(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=False, random_state=None,
                                             splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'criterion': ['mse', 'mae'],
                         'max_depth': [2, 6, 8, None],
                         'max_leaf_nodes': range(2, 9),
                         'min_samples_leaf': range(1, 9)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Its Results:

```
Tuned Decision Tree Parameters: {'criterion': 'mse', 'max_depth': 6,
                                 'max_leaf_node': 8,
                                  'min_samples_leaf': 7}

Best score is 0.7605731081722639
```

2.Random Forest:

Random forest is an ensemble that consists of many decision trees. The method combines Breiman's "bagging" idea and the random selection of features. Outputs the class that is the mode of the class's output by individual trees. Mean for regression.

Performance tuning in Random Forest in R:

1. Using ntree=200 and mtry=2

```
#Random Forest
 RF_model = randomForest(cnt ~.   , train_data, importance = TRUE, ntree=200, mtry=2)
 RF_Predictions = predict(RF_model, test[,-9])
 importance(RF_model, type = 1)
```

```
%IncMSE
season      18.5273522                       MAPE      : 12.75168
yr          79.6731257                       MAE       : 358.1391
mnth        15.5369242                       RMSE      : 486.2647
holiday      0.3391629                       R-squared: 0.9423453
weekday      0.3126971
weathersit  22.7915446
temp        28.6900501
windspeed    9.6499027
```

2. Using ntree= 500 , mtry = 7, nodesize =10

```
#Random Forest
> RF_model = randomForest(cnt ~.   , train_data, importance = TRUE, ntree=500, mtry=7,n
odesize=10)
> RF_Predictions = predict(RF_model, test[,-9])
> importance(RF_model, type = 1)
```

```
%IncMSE
season       47.325429                      MAPE       : 10.57839
yr          188.894926                      MAE        : 295.9641
mnth         27.791476                      RMSE       : 428.6273
holiday       4.646559                      R-squared  : 0.955203
weekday       8.811656
weathersit   51.864504
temp        103.432608
windspeed    14.114756
```

3.Removing less important variable i.e. holiday

```
#Random Forest
> RF_model = randomForest(cnt ~.   , df, importance = TRUE, ntree=500, mtry=7,nodesize=
10)
> RF_Predictions = predict(RF_model, test[,-8])
> importance(RF_model, type = 1)
```

```
%IncMSE                                        MAPE       : 10.35213
season       56.778685                         MAE        : 292.6396
yr          192.843333                         RMSE       : 422.5737
mnth         41.522479                         R-squared  : 0.9564594
weekday       9.101914
weathersit   55.133625
temp        118.017032
windspeed    14.423572
```

Performance tuning in Random Forest in Python:

1. Using default values:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', 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=21,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

Its Results:

```
R2 score : 0.87
Root Mean squared error: 707.02
Mean Absolute error: 497.15
MAPE: 14.70
Accuracy: 85.30
```

2. Using Grid Search :

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             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='warn', n_jobs=None,
                                             oob_score=False, random_state=10,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [3, 5, 8], 'n_estimators': range(1, 40)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Its Results:

```
Tuned Decision Tree Parameters: {'max_depth': 8, 'n_estimators': 35}
Best score is 0.8600502658396927
```

3. Using Randomized Search :

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   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='warn',
                                                   n_jobs=None, oob_score=False,
                                                   random_state=10, verbose=0,
                                                   warm_start=False),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': [3, 4, 5, 6, 7, 8],
                                        'n_estimators': range(1, 40)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

Its Results:

```
Tuned Decision Tree Parameters: {'n_estimators': 38, 'max_depth': 7}
Best score is 0.8612880776794894
```

3.Linear Regression:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering, and the number of independent variables being used. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

Linear regression model for R:

```
Call:
lm(formula = cnt ~ ., data = train_data)

Residuals:
    Min      1Q  Median      3Q     Max 
-4197.8  -375.8    70.2   485.4  2947.5 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   660.17     180.54   3.657 0.000275 ***
season2       851.57     184.10   4.626 4.44e-06 ***
season3       858.03     218.73   3.923 9.61e-05 ***
season4      1618.82     185.51   8.726  < 2e-16 ***
yr1          2064.87      59.02  34.985  < 2e-16 ***
mnth2         163.34     147.21   1.110 0.267564    
mnth3         628.60     168.84   3.723 0.000213 ***
mnth4         591.12     252.47   2.341 0.019494 *  
mnth5         837.53     273.97   3.057 0.002320 ** 
mnth6         766.74     284.70   2.693 0.007247 ** 
mnth7         259.38     318.24   0.815 0.415330    
mnth8         587.49     307.57   1.910 0.056528 .  
mnth9        1040.01     271.46   3.831 0.000139 ***
mnth10        514.18     247.86   2.074 0.038401 *  
mnth11       -127.83     236.80  -0.540 0.589490    
mnth12       -135.22     186.91  -0.723 0.469641    
holiday1     -573.12     184.79  -3.102 0.002002 ** 
weekday1      214.68     112.37   1.910 0.056480 .  
weekday2      328.68     109.92   2.990 0.002885 ** 
weekday3      397.92     110.23   3.610 0.000328 ***
weekday4      436.78     109.85   3.976 7.72e-05 ***
weekday5      480.67     109.66   4.383 1.35e-05 ***
weekday6      455.34     109.21   4.169 3.44e-05 ***
weathersit2  -695.44      63.66 -10.925  < 2e-16 ***
weathersit3 -2426.49     182.55 -13.292  < 2e-16 ***
temp         3892.90     408.00   9.541  < 2e-16 ***
windspeed   -2047.29     426.26  -4.803 1.91e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 789.7 on 704 degrees of freedom
Multiple R-squared:  0.8397,	Adjusted R-squared:  0.8338 
F-statistic: 141.9 on 26 and 704 DF,  p-value: < 2.2e-16
```

This table shows the results obtained by Linear Regression Model. The row of residual shows the error. The coefficient shows how much information each variable stores. Estimate, for example season2=851.57 show that 1-unit change in the season2 variable changes the cnt variable by 851.57.The column Std. Error measures the average amount that the coefficient estimates varying from actual average value of the targe t variable. The column t-value measures how much our std Error away from 0. R-squared and adjusted R-sq uare of 0.8397shows that our independent variable can only be able to show 83.97% of variance of the target variable.

Linear regression model for Python:

OLS Regression Results

Dep. Variable:	count	**R-squared:**	0.853
Model:	OLS	**Adj. R-squared:**	0.846
Method:	Least Squares	**F-statistic:**	119.2
Date:	Wed, 07 Aug 2019	**Prob (F-statistic):**	2.04e-211
Time:	15:08:26	**Log-Likelihood:**	-4687.7
No. Observations:	584	**AIC:**	9431.
Df Residuals:	556	**BIC:**	9554.
Df Model:	27		
Covariance Type:	nonrobust		

4.KNN

Stands for K-Nearest Neighbour. KNN is simple algorithm that stores all available cases and classifies new cases based on a similarity measure. It is also called as lazy learning algorithm because whenever a new test data come then it will calculate the distance between test case vs all the training cases. It does store any pattern and so calculation time is increased. This algorithm Pick a number of neighbours we want to use for classification or regression. It then Choose a method to measure distances.
In python it is imported through:
from sklearn.neighbors import KNeighborsRegressor. Then we have to choose a number of neighbours for which we want to impute the method.

KNN in python:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                    weights='uniform')
```
Its Result:
```
R2 score : 0.77
Root Mean squared error: 932.20
Mean Absolute error: 711.02
MAPE: 22.49
Accuracy: 77.51
```

5. k-fold Cross-Validation:

Cross-validation is a statistical method used to estimate the skill of machine learning models. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group:
    - Take the group as a hold out or test data set
    - Take the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

The results for K fold cross for different model R is as shown:

1. For Random Forest:

```
Aggregating results
Selecting tuning parameters
Fitting mtry = 14 on full training set
> print(model)
Random Forest

731 samples
8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 658, 658, 659, 655, 659, 658, ...
Resampling results across tuning parameters:

mtry  RMSE       Rsquared   MAE
2    1098.5945  0.7944355  891.3108
14    766.1160  0.8436846  547.8034
26    780.7158  0.8376702  554.9368

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 14.
```

2. For K- Nearest Neighbors Aggregating results

```
Selecting tuning parameters
Fitting k = 9 on full training set
> print(model)
k-Nearest Neighbors
731 samples
  8 predictor
No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 658, 658, 657, 658, 657, 658, ...
Resampling results across tuning parameters:

  k  RMSE      Rsquared   MAE
  5  1088.857  0.6873566  803.1541
  7  1076.702  0.6960838  816.2062
  9  1073.414  0.7004565  827.6891
RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 9.
```

3. For Linear Regression

```
Aggregating results
Fitting final model on full training set
> print(model)
Linear Regression

731 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 658, 658, 659, 658, 658, 657, ...
Resampling results:

  RMSE      Rsquared   MAE
  814.5958  0.8236876  599.7303

Tuning parameter 'intercept' was held constant at a value of TRUE
```

The results for K fold cross for different model Python is as shown:

```
KFold(n_splits=11, random_state=10, shuffle=True)
```

For dtree:

- R2 score : 0.76
- Root Mean squared error: 911.94
- Mean Absolute error: 614.12
- MAPE: 20.30
- Accuracy: 79.70

For KNN :

- R2 score : 0.78
- Root Mean squared error: 877.07
- Mean Absolute error: 602.38
- MAPE: 21.30
- Accuracy: 78.70

For Rf

- R2 score : 0.83
- Root Mean squared error: 768.19
- Mean Absolute error: 464.54
- MAPE: 17.05
- Accuracy: 82.95

MAPE Result :

For Decision Tree :
[19.285849628437603, 18.397309470622627, 17.189387058140177, 359.9034619875289, 16.008911443827433, 31.08657487183032, 20.591526828543703, 16.382059135179876, 20.311482677739168, 22.799904003907027, 18.72375135797008]

For KNN :
[23.761075431268935, 22.199161158387227, 19.098711822400073, 390.8605970709013, 16.66052069432602, 35.3120440775067, 19.77722752932658, 17.060542258602947, 19.41034645300673, 18.857342337146086, 21.300739056286083]

For Random Forest :
[15.639294013295402, 15.57613388700085, 15.720174399435685, 296.49058133122156, 12.281790136995738, 25.271389407011846, 15.073681250525265, 10.47286258912342, 19.532064854473816, 16.47230061497855, 19.423824334725946]

Chapter 6

Model Evaluation

Once we have developed different models, the main task is to detect the performance of the model and also how it can accomplish the need of the problem statement. Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

Error matrix

One of the best tools for the evaluation is the Error matrix. It helps to evaluate our analytical model. It also helps us to make trade-off between multiple matrixes.
Choice of the matrix depends upon :

- Type of model
- Implementation plan of model

Also, the matrix selection depends on the types of problem we are dealing with like: Classification, Regression , Optimization, Unsupervised.

Since, in our problem the target variable is continuous in nature we will go for **Regression Matrix evaluation**.

Regression Evaluation measure of error:

Mean Absolute Error:
The mean absolute error(MAE) has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. It is similar in magnitude to RMSE, but slightly smaller.

Mean Absolute Percentage Error: It measures accuracy as a percentage of error. It is obtained by multiplying MAE by 100.

Root Mean Square Error: It is the square root of the square of difference of the given value and true value.

Accuracy: This signifies how accurately our model is able to predict the target variable. In is obtained in percentage and calculated by subtracting Mean Absolute Percentage Error form 100.

Accuracy = 100 – MAPE

Model Evaluation:

Comparison of parameters for different models in R:

Model	MAPE	MAE	RMSE	R-squared	Accuracy
Decision Tree	22.79342	642.6354	851.9674	0.8230155	77.20
Random Forest	10.57839	292.6396	422.5737	0.9564594	89.4216
KNN	33.04551	1052.705	1421.543	0.5072696	66.95449

Also, for :

Linear Regression Model: MAPE =19.68449 Accuracy = 80.31

K fold cross validation_Random Forest

mtry	RMSE	Rsquared	MAE
2	1098.7640	0.7945031	891.6332
14	7554.7969	0. 8473857	537.5845
26	771.3941	0.8398626	544.1769

K fold cross validation_KNN

K	RMSE	Rsquared	MAE
5	1056.833	0.7029138	803.3087
7	1065.646	0.6983565	819.7315
9	1064.629	0.7037592	830.2544

K fold cross validation_Linear Regression

Resampling results:

RMSE	Rsquared	MAE
793.0179	0.8342588	588.9346

Comparison of parameters for different models in Python:

Model	**MAPE**	**MAE**	**RMSE**	**R-squared**	**Accuracy**
Decision Tree	18.21	572.52	822.47	0.82	81.79
Random Forest	14.70	497.15	707.02	0.87	85.30
KNN	22.49	711.02	932.20	0.77	77.51
Decision Tree_KFfold	20.30	614.12	911.94	0.76	79.70
Random Forest_KFold	17.05	464.54	768.19	0.83	82.95
KNN_KFold	21.30	602.38	877.07	0.78	78.70

Also, for :

Linear Regression Model: MAPE =16.45
Accuracy = 83.55

Chapter 7

Model Selection

Selecting Model

After preforming different algorithms and from the table above , the best model is extracted and selected for the final bike rental count.

For R:

Model	MAPE	MAE	RMSE	R-squared	Accuracy
Random Forest	10.57839	292.6396	422.5737	0.9564594	89.4216

For Python:

Model	MAPE	MAE	RMSE	R-squared	Accuracy
Random Forest	14.70	497.15	707.02	0.87	85.30

Extra figures:

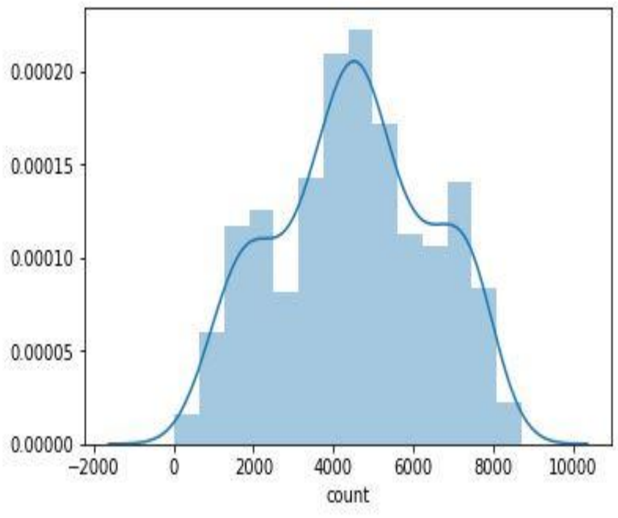


Fig: Distribution of Count

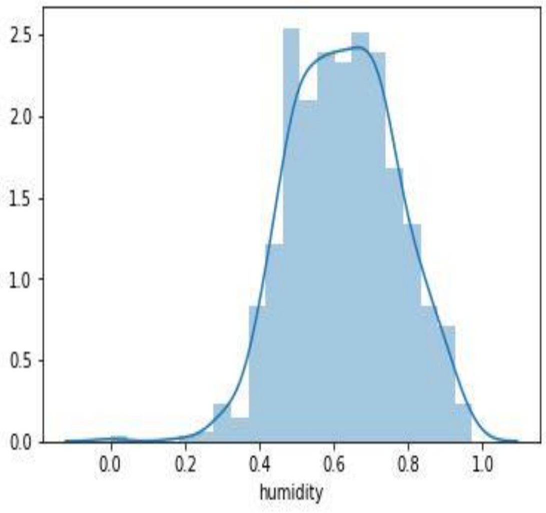


Fig: Distribution of Humidity

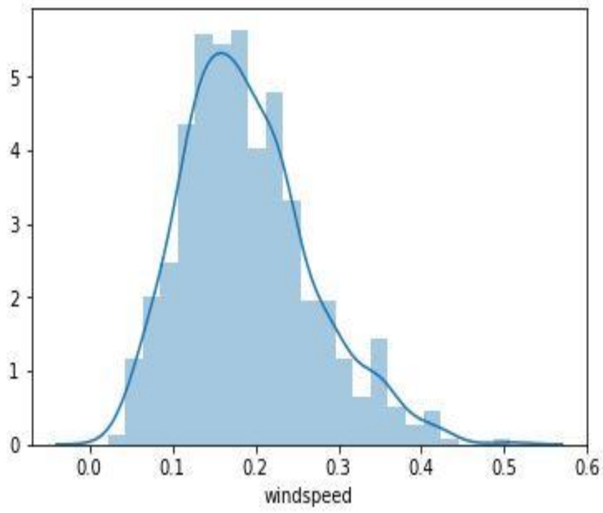


Fig: Distribution of windspeed

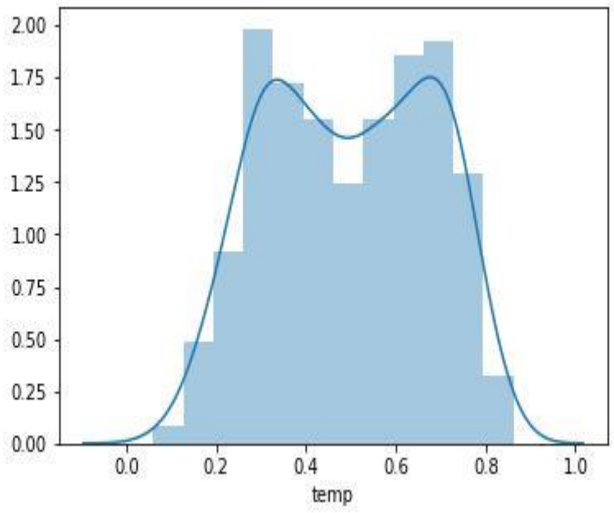


Fig: Distribution of temp

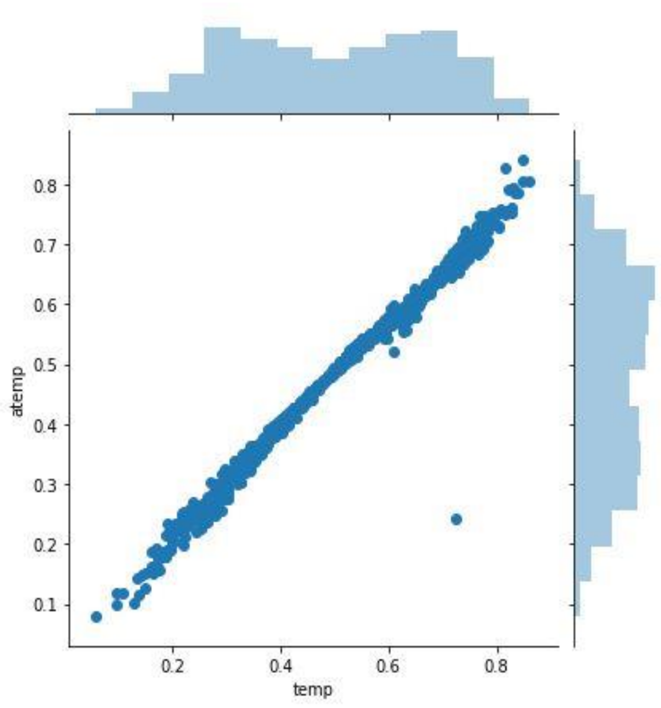


Fig: Distribution of temp vs atemp

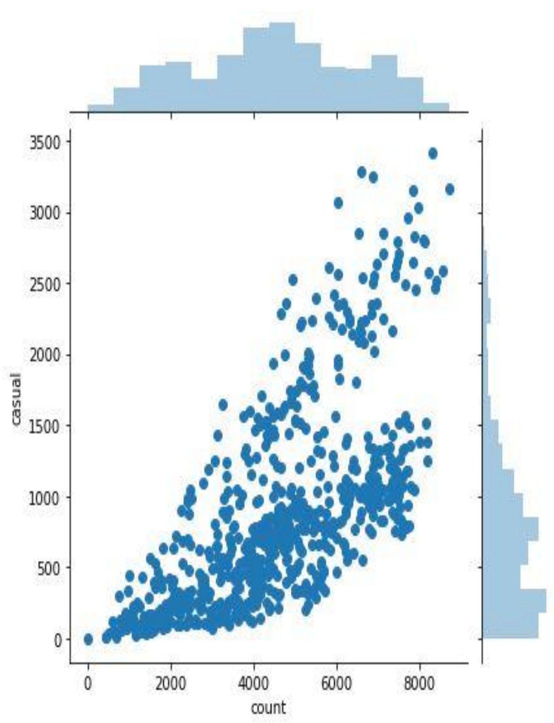


Fig: Distribution of causal vs count

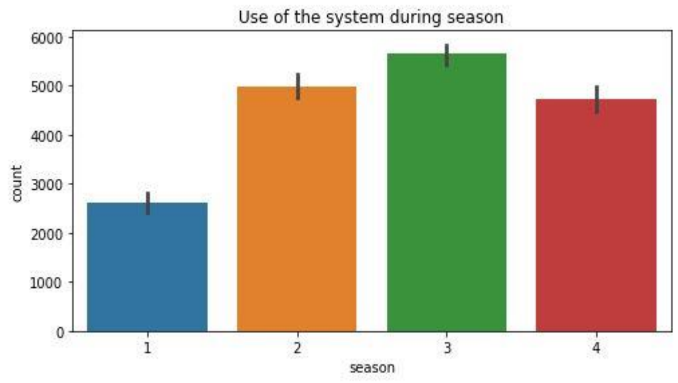


Fig: Distribution of count vs season

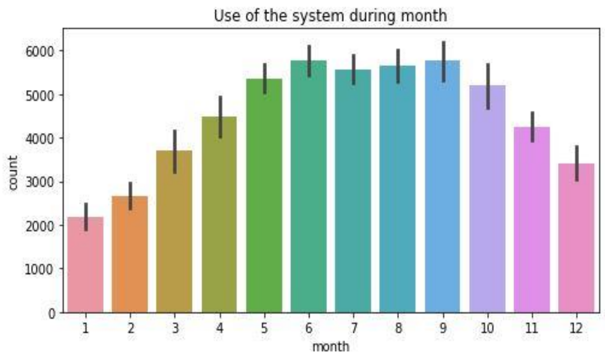


Fig: Distribution of count vs month

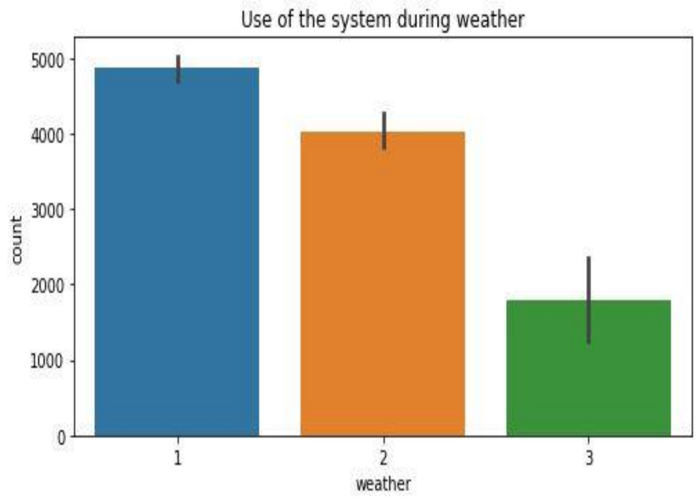


Fig: Distribution of count vs weather

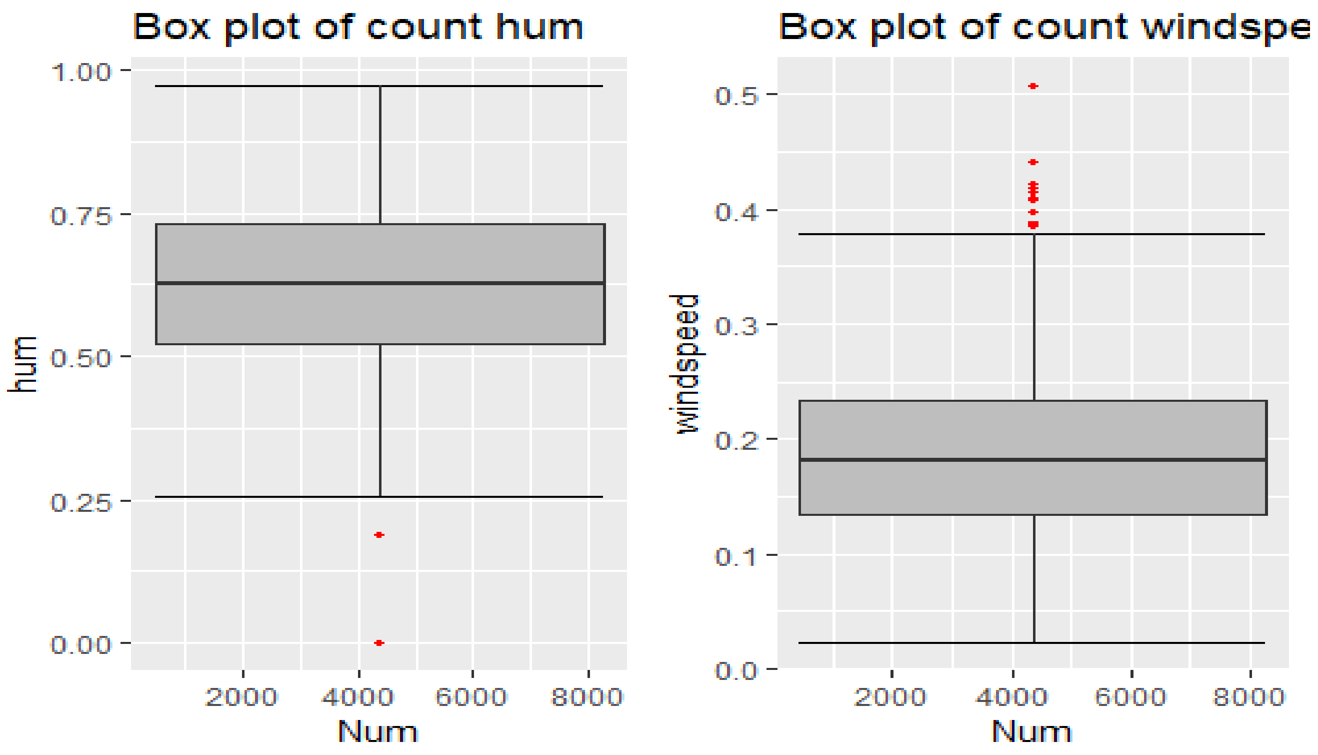


Fig: Outlier in Data

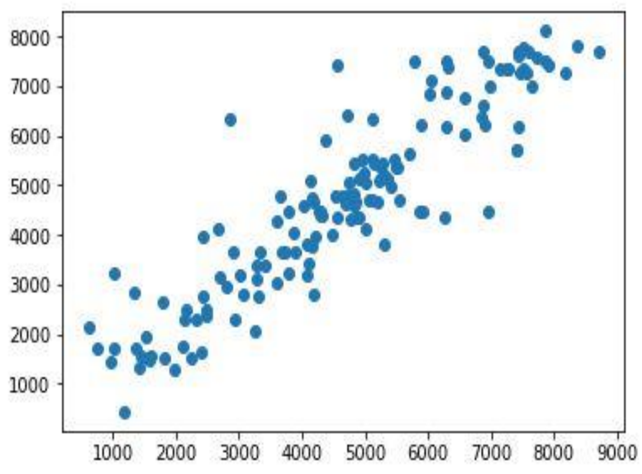


Fig: Scatter plot of Predicted vs Actual Decision Tree

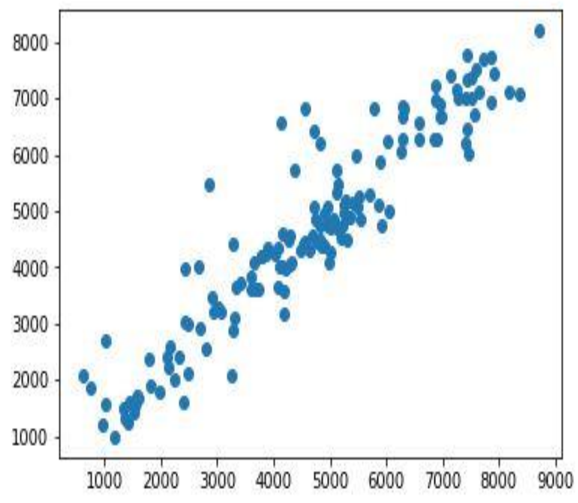


Fig: Scatter plot of Predicted vs Actual Random Forest

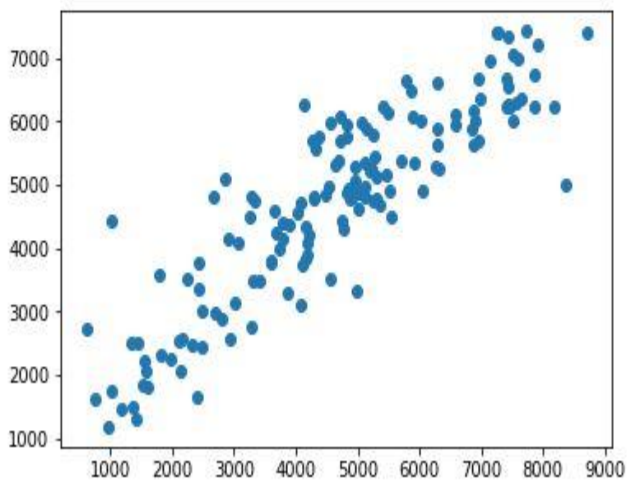


Fig: Scatter plot of Predicted vs Actual KNN

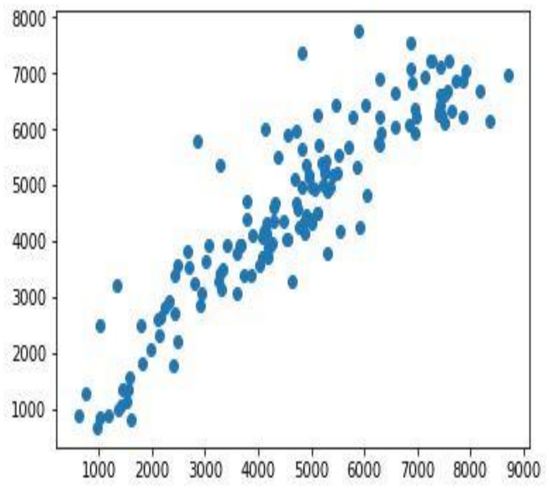


Fig: Scatter plot of Predicted vs Actual Linear Regression

Python Code

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
# In[2]:
train = pd.read_csv("D:\\Data Science\\Project2\\day.csv")
# In[3]:
train.head()
# In[4]:
train.describe()
# In[5]:
train.shape
# In[6]:
train.info()
# In[7]:
train.isna().sum()
In[9]:
#plt.figure(figsize=(14,5))
#sns.pairplot(train[])
train.rename(columns={'weathersit':'weather',
                     'mnth':'month',
                     'hr':'hour',
                     'hum': 'humidity',
                     'cnt':'count',
                     'yr':'year'},inplace=True)

train = train.drop(['instant','dteday'], axis=1)
# In[10]:
train.columns
# In[11]:
train.head()
# In[12]:
train.dtypes
# In[13]:
train['season'] = train.season.astype('category')
train['month'] = train.month.astype('category')
train['holiday'] = train.holiday.astype('category')
train['weekday'] = train.weekday.astype('category')
train['workingday'] = train.workingday.astype('category')
train['weather'] = train.weather.astype('category')
train['year'] = train.year.astype('category')
train.dtypes
# In[18]:
fig, ax = plt.subplots(figsize=(8,4))
```

```
sns.barplot(data=train[['weather',
                   'count']],
          y='count', x='weather',
          ax=ax)
ax.set(title="Use of the system during weather")
 [19]:
names1=['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
      'weather', 'temp', 'atemp', 'humidity', 'windspeed', 'casual',
      'registered', 'count']

df_corr = train.loc[:,cnames1]
# In[21]:
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(16, 6))
#Generate correlation matrix
corr = df_corr.corr()

#Plot using seaborn library
sns.heatmap(corr,                    mask=np.zeros_like(corr,                    dtype=np.bool),
cmap='viridis',linewidths=1,linecolor='black',annot=True)
# In[22]:
#sns.distplot(train['count'])
# In[26]:
train = train.drop(['atemp', 'casual', 'registered','workingday'], axis=1)
# In[28]:
#sns.boxplot(x=train['season'], y=train['count'])
# In[29]:
#sns.boxplot(x=train['weather'],y=train['count'])
# In[30]:
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error
# In[31]:
train.columns
# In[32]:
X = train[['season', 'year', 'month', 'holiday', 'weekday','weather', 'temp', 'humidity', 'windspeed']]
# In[33]:
y= train['count']
# In[74]:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
# In[41]:
dtree = DecisionTreeRegressor()
# In[42]:
dtree.fit(X_train,y_train)
# In[43]:
predictions = dtree.predict(X_test)
# In[44]:
```

```
#Calculate MAPE
def MAPE(y_true, y_pred):
   mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
   return mape
# In[45]:
print("R2 score : %.2f" % r2_score(y_test, predictions))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions))
print("MAPE: %.2f" % MAPE(y_test, predictions))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions)))
# In[46]:
# Necessary imports
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

# Creating the hyperparameter grid
param_dist = {"max_depth": [6, None],
"max_features": randint(1, 9),
"min_samples_leaf": randint(1, 9),
"criterion": ["mse", "mae"]}

# Instantiating Decision Tree classifier
tree = DecisionTreeRegressor()

# Instantiating RandomizedSearchCV object
tree_cv = RandomizedSearchCV(tree, param_dist, cv = 5)

tree_cv.fit(X_train, y_train)
predictions = dtree.predict(X_test)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
# In[48]:
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint

param_grid = {"criterion": ["mse","mae"],

          "max_depth": [2, 6, 8, None],
          "min_samples_leaf": range(1,9),
          "max_leaf_nodes": range(2,9),

          }

dtree = DecisionTreeRegressor()
```

```
grid_cv_dtm = GridSearchCV(dtree, param_grid, cv=5)

grid_cv_dtm.fit(X_train,y_train)
# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(grid_cv_dtm.best_params_))
print("Best score is {}".format(grid_cv_dtm.best_score_))
# In[50]:
plt.scatter(y_test,predictions)
# In[51]:
#Random Forest
from sklearn.ensemble import RandomForestRegressor
# In[75]:
# function to increment n_estimators one by one and calculate error rate..

error_rate=[]
for i in range(1,60):
    ran =RandomForestRegressor(n_estimators=i)
    ran.fit(X_train,y_train)
    pred_i=ran.predict(X_test)
    error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)


# In[76]:
# plot of error rate by the function

plt.figure(figsize=(8,4))
plt.plot(range(1,60),error_rate,marker='o')
# In[85]:
RF_model = RandomForestRegressor(n_estimators =6).fit(X_train,y_train)
# In[87]:
predictions = RF_model.predict(X_test)
# In[88]:
print("R2 score : %.2f" % r2_score(y_test, predictions))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions))
print("MAPE: %.2f" % MAPE(y_test, predictions))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions)))


# In[63]:
plt.scatter(y_test,predictions)
# In[89]:
# parameters for GridSearchCV
param_grid2 = {"n_estimators": range(1,40),
          "max_depth": [3, 5, 8 ],

          }
clf = RandomForestRegressor(random_state = 10)
```

```
grid_search = GridSearchCV(clf, param_grid=param_grid2)
grid_search.fit(X_train, y_train)
# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(grid_search.best_params_))
print("Best score is {}".format(grid_search.best_score_))
# In[90]:
# Necessary imports
from scipy.stats import randint

from sklearn.model_selection import RandomizedSearchCV

# Creating the hyperparameter grid
param_dist = {"n_estimators": range(1,40),
            "max_depth": [3,4,5,6,7,8 ],
            }

# Instantiating Decision Tree classifier
clf = RandomForestRegressor(random_state = 10)

# Instantiating RandomizedSearchCV object
forest_cv = RandomizedSearchCV(clf, param_dist, cv = 5)

forest_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(forest_cv.best_params_))
print("Best score is {}".format(forest_cv.best_score_))
# In[92]:
from sklearn.neighbors import KNeighborsRegressor
# In[93]:

# function to increment n_neighbors one by one and calculate error rate..

error_rate=[]
for i in range(1,40):
  knn =KNeighborsRegressor(n_neighbors=i)
  knn.fit(X_train,y_train)
  pred_i=knn.predict(X_test)
  error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)


# In[94]:
# plot of error rate by the function
plt.figure(figsize=(8,4))
plt.plot(range(1,40),error_rate,marker='o')
# In[95]:
KNN_model = KNeighborsRegressor(n_neighbors = 6)
# In[96]:
```

```
KNN_model.fit(X_train, y_train)
# In[97]:

predictions = KNN_model.predict(X_test)
# In[98]:
print("R2 score : %.2f" % r2_score(y_test, predictions))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions))
print("MAPE: %.2f" % MAPE(y_test, predictions))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions)))
# In[99]:
plt.scatter(y_test,predictions)
# In[100]:
linear = train.iloc[:,6:9]
# In[102]:
cat_names = ['season', 'year', 'month', 'holiday', 'weekday','weather']
for i in cat_names:
   temp = pd.get_dummies(train[i], prefix = i)
   linear = linear.join(temp)



# In[103]:
linear= linear.join(train['count'])
# In[104]:
linear.columns
# In[105]:
train1, test1 = train_test_split(linear, test_size=0.2,random_state=10)
# In[107]:
#Import libraries for LR
import statsmodels.api as sm
# In[108]:
# Train the model using the training sets
model = sm.OLS(train1.iloc[:, 33 ], train1.iloc[:, 0:33 ]).fit()
# In[109]:
# Print out the statistics
model.summary()
# In[239]:
# make the predictions by the model
predictions_LR = model.predict(test1.iloc[:,0:33])
# In[240]:
#Calculate MAPE
MAPE(test1.iloc[:,33], predictions_LR)
# In[241]:
plt.scatter(test1.iloc[:,33],predictions_LR)
# In[133]:
X = train[['season', 'year', 'month', 'holiday', 'weekday', 'weather', 'temp', 'humidity', 'windspeed']]
# In[134]:
y= train['count']
```

```
# In[135]:
def get_score(model, X_train, X_test, y_train, y_test):
   model.fit(X_train, y_train)
   predictions = model.predict(X_test)

   return model.score(X_test, y_test),MAPE(y_test,predictions)
# In[136]:
from sklearn.model_selection import KFold # import KFold
kf = KFold(n_splits=11, random_state=10, shuffle=True) # Define the split - into 2 folds
kf.get_n_splits(X) # returns the number of splitting iterations in the cross-validator
print(kf)
scores_dtree = 0
scores_knn = 0
scores_rf = 0

MAPE_dtree =[]
MAPE_knn = []
MAPE_rf = []

for train_index, test_index in kf.split(X):
   #print('train', train_index, 'Test', test_index)
   X_train, X_test = X.iloc[train_index], X.iloc[test_index]
   y_train, y_test = y.iloc[train_index], y.iloc[test_index]



   scores_dtree= scores_dtree+get_score(DecisionTreeRegressor(), X_train, X_test, y_train, y_test)[0]
   MAPE_dtree.append(get_score(DecisionTreeRegressor(), X_train, X_test, y_train, y_test)[1])
   scores_knn   = scores_knn+get_score(KNeighborsRegressor(n_neighbors = 4), X_train, X_test, y_train,
y_test)[0]
   MAPE_knn.append(get_score(KNeighborsRegressor(n_neighbors = 4), X_train, X_test, y_train,
y_test)[1])
   scores_rf   = scores_rf +get_score(RandomForestRegressor(n_estimators=12), X_train, X_test, y_train,
y_test)[0]
   MAPE_rf.append(get_score(RandomForestRegressor(n_estimators=12), X_train, X_test, y_train,
y_test)[1])


# In[137]:
RF_model = RandomForestRegressor(n_estimators =15).fit(X_train,y_train)
# In[138]:
predictions = RF_model.predict(X_test)
# In[139]:
print("R2 score : %.2f" % r2_score(y_test, predictions))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions))
print("MAPE: %.2f" % MAPE(y_test, predictions))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions)))
```

```
# In[140]:
dtree = DecisionTreeRegressor()

# In[141]:
dtree.fit(X_train,y_train)
# In[142]:
predictions = dtree.predict(X_test)
# In[143]:
print("R2 score : %.2f" % r2_score(y_test, predictions))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions))
print("MAPE: %.2f" % MAPE(y_test, predictions))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions)))
# In[144]:
KNN_model = KNeighborsRegressor(n_neighbors = 4)
# In[145]:
KNN_model.fit(X_train, y_train)
# In[146]:
predictions = KNN_model.predict(X_test)
# In[147]:
print("R2 score : %.2f" % r2_score(y_test, predictions))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions))
print("MAPE: %.2f" % MAPE(y_test, predictions))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions)))
# In[129]:
print('Score_dtree : ',(scores_dtree/10)*100)

print('Score_KNN   : ',(scores_knn/10)*100)

print('Score_Rf    : ',(scores_rf/10)*100)



# In[130]:
print(MAPE_dtree ,'\n''\n''\n',MAPE_knn,'\n''\n''\n', MAPE_rf)
# In[116]:

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

# In[117]:



model = RandomForestRegressor(n_estimators=12)
kfold = KFold(n_splits=11, random_state=10, shuffle=True)
result = cross_val_score(model, X, y, cv=kfold,scoring='r2')
print(result.mean()*100)
```

R Code

```
rm(list = ls())

setwd("D:/Data Science/Project2")
getwd()
#Load Libraries
x = c("ggplot2","dplyr", "corrgram", "DMwR",  "randomForest", "unbalanced",  "e1071",
    "MASS", "rpart", "gbm", 'DataCombine', 'inTrees','caret','C50','mice','class')
lapply(x, require, character.only = TRUE)
rm(x)

# load the data
train_data = read.csv("day.csv", header = T, na.strings = c(" ", "", "NA"))[,-1]
summary(train_data)

str(train_data)
sum(is.na(train_data))
View(train_data)

par(mfrow=c(1,1))
par(mar = rep(2,4))
hist(train_data$dteday)
hist(train_data$season)
hist(train_data$yr)
hist(train_data$mnth)
hist(train_data$holiday)
hist(train_data$weekday)
hist(train_data$workingday)
hist(train_data$weathersit)
hist(train_data$temp)
hist(train_data$atemp)
hist(train_data$hum)
hist(train_data$windspeed)
hist(train_data$casual)
hist(train_data$registered)
hist(train_data$cnt)

train_data[,1:8]= lapply (train_data[, 1:8], as.factor)
train_data[,9:15]= lapply (train_data[, 9:15], as.numeric)

str(train_data)
```

```
cnames = colnames(train_data[,c(9:12)])
#
for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "cnt", group = 1), data = train_data)+
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                        outlier.size=1, notch=FALSE) +
           theme(legend.position="bottom")+
           labs(y=cnames[i],x="Num")+
           ggtitle(paste("Box plot of count",cnames[i])))
}
#
# ## Plotting plots together
gridExtra::grid.arrange(gn1,gn2,ncol=2)
gridExtra::grid.arrange(gn3,gn4, ncol=2 )


# replacing outlier with NA
for(i in cnames){
  val = train_data[,i][train_data[,i] %in% boxplot.stats(train_data[,i])$out]
  #print(length(val))
  train_data[,i][train_data[,i] %in% val] = NA
}
######################
init = mice(train_data, maxit=0)
meth = init$method
predM = init$predictorMatrix
set.seed(103)
train_data = mice(train_data, method='cart', predictorMatrix=predM, m=1)
train_data = complete(train_data)


sub=data.frame(train_data$registered,train_data$casual,train_data$cnt,train_data$temp,train_data$hu
m,train_data$atemp,train_data$windspeed)
cor(sub)
## Correlation Plot
corrgram(train_data[,9:15], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot"
)

#to check the degree of association of categorical variable
## Chi-squared Test of Independence
factor_index = sapply(train_data,is.factor)
factor_data = train_data[,factor_index]
library(GoodmanKruskal)
plot(GKtauDataframe( factor_data), corrColors = 'blue')
```

```
##################################################################3

train_data = subset(train_data,select=-c(dteday,atemp,hum,casual,registered,workingday ))

df = train_data
#train_data = df
#################################


set.seed(123)
train_index = sample(1:nrow(train_data), 0.7 * nrow(train_data))
colnames(train_data)

train = train_data[train_index,]
test = train_data[-train_index,]


#Decision Tree
fit = rpart(cnt  ~. , data = train_data, method = "anova")
summary(fit)
predictions_DT = predict(fit, test[,-9])

# Define MAPE
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y*100))
}


MAPE(test[,9], predictions_DT)
#[1] 22.79342

# R native funcitons

fmae = MAE(predictions_DT, test[,9])
# caret package functions
frmse = RMSE(predictions_DT, test[,9])
fr2 = R2(predictions_DT, test[,9], form = "traditional")
cat( "MAE:", fmae, "\n",
    "RMSE:", frmse, "\n",
    "R-squared:", fr2
    )
#MAE: 642.6354
#RMSE: 851.9674
#R-squared: 0.8230155

#Random Forest
```

```
RF_model = randomForest(cnt ~.  , train_data, importance = TRUE, ntree=200, mtry=2)
RF_Predictions = predict(RF_model, test[,-9])
importance(RF_model, type = 1)
MAPE(test[,9], RF_Predictions)
#[1] 12.75168
# R native funcitons
fmae = MAE(RF_Predictions, test[,9])
# caret package functions
frmse = RMSE(RF_Predictions, test[,9])
fr2 = R2(RF_Predictions, test[,9], form = "traditional")
cat( "MAE:", fmae, "\n",
    "RMSE:", frmse, "\n",
    "R-squared:", fr2
)
#MAE: 358.1391
#RMSE: 486.2647
#R-squared: 0.9423453




#Random Forest
RF_model = randomForest(cnt ~.  , train_data, importance = TRUE, ntree=500, mtry=7,nodesize=10)
RF_Predictions = predict(RF_model, test[,-9])
importance(RF_model, type = 1)

MAPE(test[,9], RF_Predictions)
# [1] 10.57839
# R native funcitons

fmae = MAE(RF_Predictions, test[,9])
# caret package functions
frmse = RMSE(RF_Predictions, test[,9])
fr2 = R2(RF_Predictions, test[,9], form = "traditional")
cat( "MAE:", fmae, "\n",
    "RMSE:", frmse, "\n",
    "R-squared:", fr2
)
#MAE: 295.9641
#RMSE: 428.6273
#R-squared: 0.955203
##################################
df = subset(train_data, select = -c(holiday) )
set.seed(123)
train_index = sample(1:nrow(df), 0.7 * nrow(df))
colnames(df)

train = df[train_index,]
```

```
test = df[-train_index,]

#Random Forest_df
RF_model = randomForest(cnt ~.  , df, importance = TRUE, ntree=500, mtry=7,nodesize=10)
RF_Predictions = predict(RF_model, test[,-8])
importance(RF_model, type = 1)
MAPE(test[,8], RF_Predictions)
#[1] 10.35213
# R native funcitons

fmae = MAE(RF_Predictions, test[,8])
# caret package functions
frmse = RMSE(RF_Predictions, test[,8])
fr2 = R2(RF_Predictions, test[,8], form = "traditional")
cat( "MAE:", fmae, "\n",
    "RMSE:", frmse, "\n",
    "R-squared:", fr2
)

#MAE: 292.6396
#RMSE: 422.5737
#R-squared: 0.9564594
################################################
#Linear Regression
set.seed(123)

train_index = sample(1:nrow(train_data), 0.7 * nrow(train_data))
colnames(train_data)

train = train_data[train_index,]
test = train_data[-train_index,]

lm_model = lm(cnt ~. , data = train_data)
summary(lm_model)
predictions_LR = predict(lm_model, test[,-9])
MAPE(test[,9], predictions_LR)
#[1] 19.68449


##################################################################


####################################################
##KNN Implementation
library(class)
#Predict test data
KNN_Predictions = knn(train[, 1:8], test[, 1:8], train$cnt, k = 5)
#convert the values into numeric
```

```
KNN_Predictions=as.numeric(as.character((KNN_Predictions)))
#Calculate MAPE
MAPE(test[,9], KNN_Predictions)

# R native funcitons

fmae = MAE(KNN_Predictions, test[,9])
# caret package functions
frmse = RMSE(KNN_Predictions, test[,9])
fr2 = R2(KNN_Predictions, test[,9], form = "traditional")
cat( "MAE:", fmae, "\n",
     "RMSE:", frmse, "\n",
     "R-squared:", fr2
)
#MAE: 1052.705
#RMSE: 1421.543
#R-squared: 0.5072696
#[1] 33.04551




#######################
#K fold cross validation_Random Forest
model <- train(
  cnt ~ .,
  train_data,
  method = "rf",
  trControl = trainControl(
    method = "cv",
    number = 7,
    verboseIter = TRUE
  )
)
print(model)
################################################3
#K fold cross validation_KNN
model <- train(
  cnt ~ .,
  train_data,
  method = "knn",
  trControl = trainControl(
    method = "cv",
    number = 10,
    verboseIter = TRUE
  )
)
print(model)
```

```
############################################################

#K fold cross validation_Linear Regression
model <- train(
  cnt ~ .,
  train_data,
  method = "lm",
  trControl = trainControl(
    method = "cv",
    number = 10,
    verboseIter = TRUE
  )
)
print(model)
```

References:

https://edwisor.com/
https://stackoverflow.com
https://www.geeksforgeeks.org/k-nearest-neighbours/
https://www.udemy.com/

Book:
ISLR: An Introduction to Statistical Learning with Applications in R by Gareth James • Daniela Witten • Trevor Hastie Robert Tibshirani