Bike Rental Project

*SUSHANT THAKUR*

*24 November 2019*

**Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **Introduction** | | | **2** |
|  | 1.1 | Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 2 |
|  | 1.2 | Data . | ................................................. | 2 |
| **2** | **Methodology** | | | **4** |
|  | 2.1 | Pre Processing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 4 |
|  |  | 2.1.1 | Outlier Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
|  |  | 2.1.2 | Feature Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 10 |
|  | 2.2 | Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 12 |
|  |  | 2.2.1 | Model Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
|  |  | 2.2.2 | Multiple Linear Regression . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
|  |  | 2.2.3 | Regression Trees . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
|  |  | 2.2.4 | Classification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
| **3** | **Conclusion** | |  | **15** |
|  | 3.1 | Model Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 15 |
|  |  | 3.1.1 | Mean Absolute Error (MAE) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
|  |  | 3.1.2 | Mean Squared Error (MSE) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
|  | 3.2 | Model Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 16 |
| **Appendix A - Extra Figures**  **Appendix B - Python Code** | | | | **17** |
|  | | | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  |  |  | |  |
|  | | |  |  |

1

**Chapter 1**

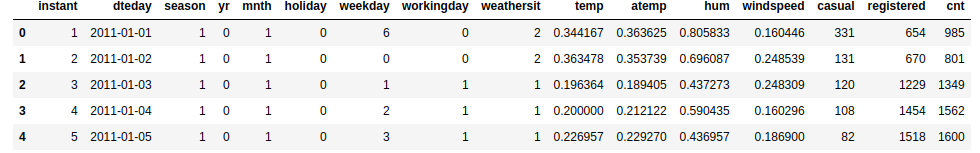
**Introduction**

**1.1 Problem Statement**

The Bike Rental Data contains the daily count of rental bikes between the year 2011 and 2012 with corresponding weather and seasonal information. We would like to predict the daily count of rental count in order to automate the system.

1. **Data**

Our task is to build Regression model which will give the daily count of rental bikes based on weather and season Given below is a sample of the data set that we are using to predict the count:

****

Below are the variables we used to predict the count of bike rentals

instant 731 non-null int64

dteday 731 non-null object

season 731 non-null int64

yr 731 non-null int64

mnth 731 non-null int64

holiday 731 non-null int64

weekday 731 non-null int64

workingday 731 non-null int64

weathersit 731 non-null int64

temp 731 non-null float64

atemp 731 non-null float64

hum 731 non-null float64

windspeed 731 non-null float64

casual 731 non-null int64

registered 731 non-null int64

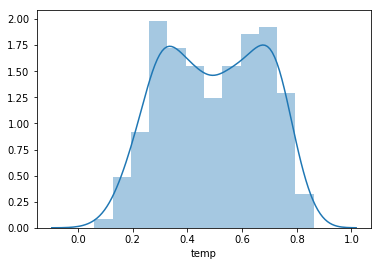
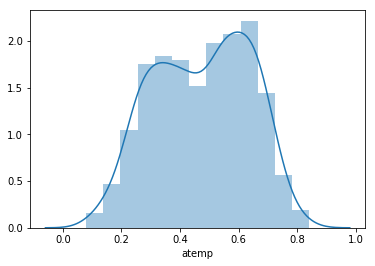
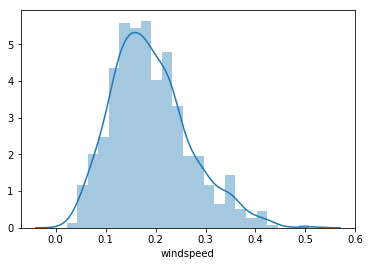
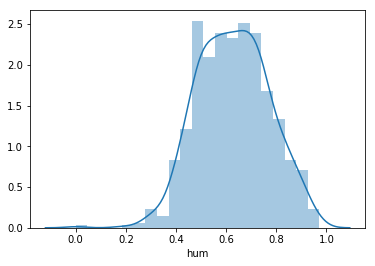
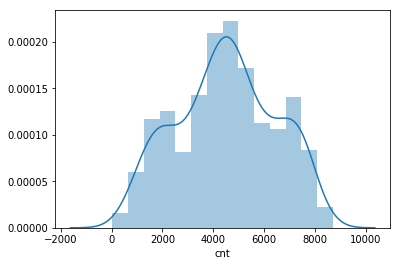
**Chapter 2**

**Methodology**

**2.1** **Pre Processing**

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms *looking at data* refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. To start this process we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

In Figure 2.1 we have plotted the probability density functions of some the predictor variables we have available in the data as well as the dependent cnt variable.

****

**2.1.1**

**Multicollinearity analysis**

As we don’t want our model to become complex or also remove redundancy so we could select appropriate variables for our model with less redundancy.

variables:temp

VIF Factor features

0 8.3 Intercept

1 1.0 temp

Variables:temp+hum

VIF Factor features

0 25.2 Intercept

1 1.0 temp

2 1.0 hum

variables:temp+hum+atemp

VIF Factor features

0 26.9 Intercept

1 61.0 temp

2 1.0 hum

3 61.2 atemp

variables:temp+hum+atemp+windspeed

VIF Factor features

0 45.6 Intercept

1 63.0 temp

2 1.1 hum

3 63.6 atemp

4 1.1 windspeed

**2.2.1 Missing Value Analysis**

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

Below table illustrate no missing value present in the data.

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

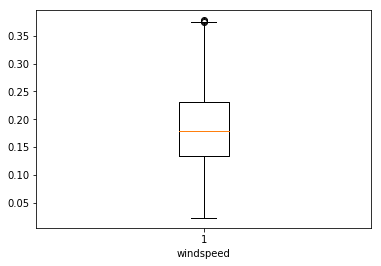
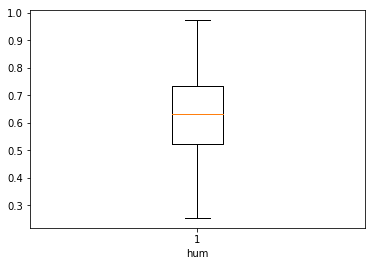
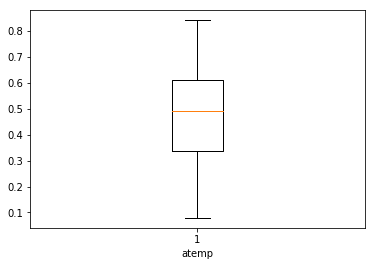
cnt 0

**2.2.2 Outlier Analysis**

The Other steps of Preprocessing Technique is Outliers analysis , an outlier is an observation point that is distant from other observations. Outliers in data can distort predictions and affect the accuracy, if you don’t detect and handle them appropriately especially in regression models..

As we are observed in histograms that are drawn above, the data is skewed so, there is chance of outlier in independent variable , one of the best method to detect outliers is Boxplot

Below figures shows presence of Outliers in variable but they are very close.



.

**2.2.3 Features Selections**

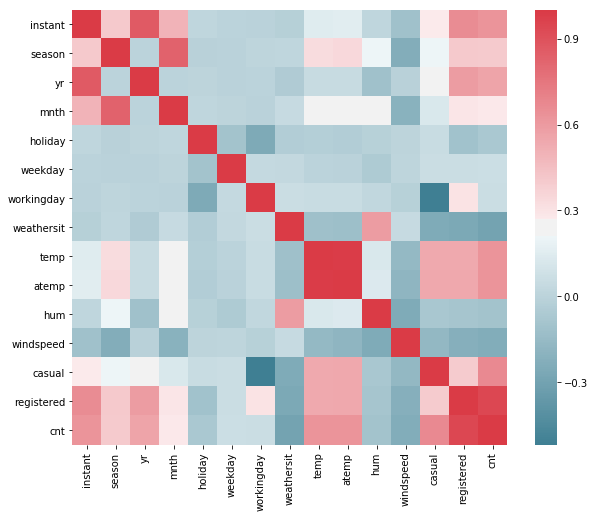
Machine learning works on a simple rule – garbage in and garbage(noise) out

This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. I have myself witnessed feature subsets giving better results than complete set of feature for the same algorithm or – “Sometimes, less is better!”.

We should consider the selection of feature for model based on below criteria

1. The relationship between two independent variable should be less and
2. The relationship between Independent and Target variables should be high.

Below fig illustrates that relationship between all numeric variables using correlation plot .



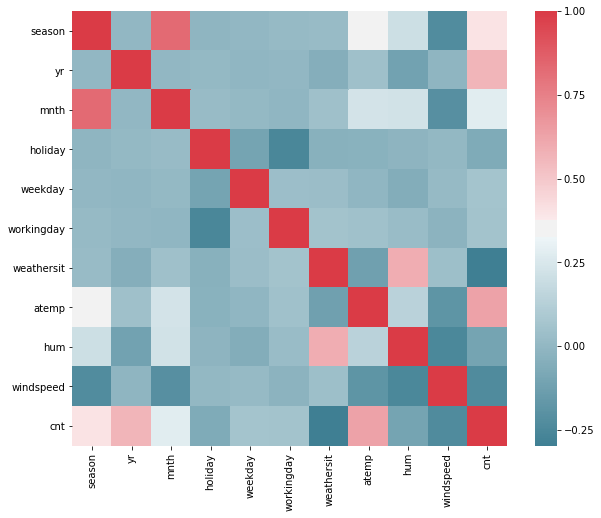
|  |  |  |
| --- | --- | --- |
| **VIF Factor** | **features** |  |
|  | **45.6** | **Intercept** |
|  | **63.0** | **temp** |
|  | **1.1** | **hum** |
|  | **63.6** | **atemp** |
|  | **1.1** | **Windspeed** |

**As from the above table we can say thet temp and atemp has very high value of VIF so they both are explaining the same variance of data so we can drop one of the other**

**2.4.1 Dimensionality Reduction for numeric variables**

there is strong relationship between independent variables ‘temp’ and ‘atemp’ so considering any one feature enough to predict the better.

After removing some variable which are not useful for predicting the target variable or have strong collinearity with above data.



**2.2.2 Features Scaling**

The word “normalization” is used informally in statistics, and so the term normalized data can have multiple meanings. In most cases, when you normalize data you eliminate the units of measurement for data, enabling you to more easily compare data from different places. Some of the more common ways to normalize data include:

Transforming data using a [z-score](http://www.statisticshowto.com/probability-and-statistics/z-score/) or [t-score](http://www.statisticshowto.com/probability-and-statistics/t-distribution/t-score-formula/). This is usually called standardization. In the vast majority of cases, if a statistics textbook is talking about normalizing data, then this is the definition of “normalization” they are probably using.

[Rescaling data](http://www.statisticshowto.com/what-is-rescaling-data/) to have values between 0 and 1. This is usually called feature scaling. One possible formula to achieve this is.

We already have our continuous variable between 0 to 1.so there is no need of feature scaling.

**Chapter 3**

**Modelling**

**3.1 Model Selection**

In out earlier stage of analysis we have come to understand that few variables like ‘temp’ ,’casual,’registered ‘ are going to play key role in model development , for model development dependent variable may fall under below categories

1. Nominal
2. Ordinal
3. Interval
4. Ratio

In our case dependent variable is interval so, the predictive analysis that we can perform is **Regression** Analysis

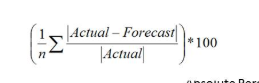
We will start our model building from Decision Tree .

**3.1.1 Evaluating Regression Model**

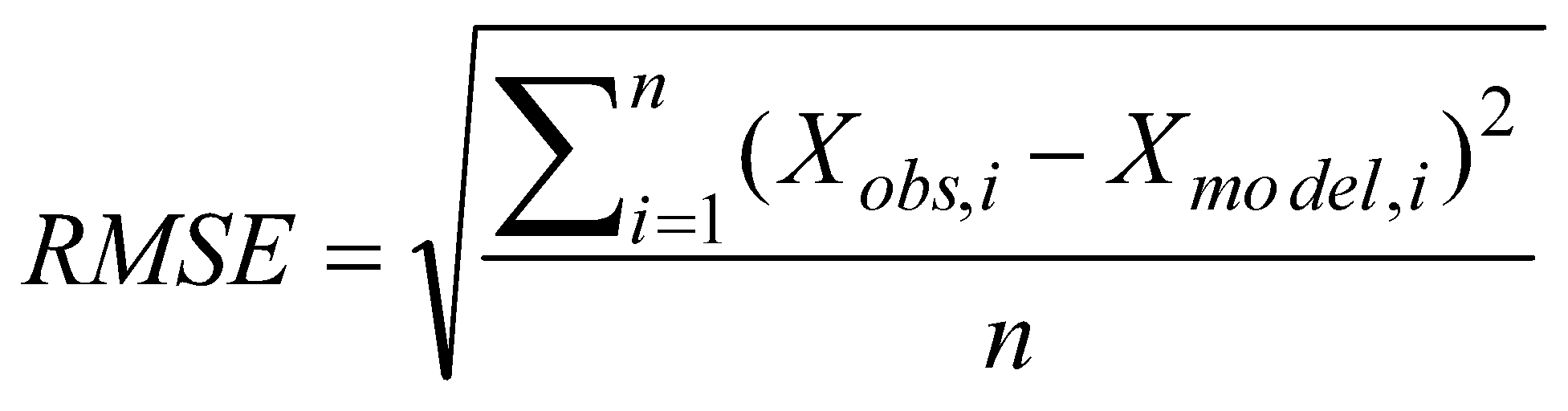
The main concept of looking at what is called **residuals** or difference between our predictions f(x[I,]) and actual outcomes y[i].

We are using two methods to evaluating performance of model

1. **MAPE** : (Mean Absolute Percent Error) measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.



1. **RMSE :** (Root Mean Square Error) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.



**3.1 Linear Regression**

[Multiple linear regression](http://www.statisticssolutions.com/academic-solutions/membership-resources/member-profile/data-analysis-plan-templates/data-analysis-plan-multiple-linear-regression/) is the most common form of linear regression analysis.  As a predictive analysis, the multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables.  The independent variables can be continuous or categorical.

**3.2 Decision Tree**

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

**Hyper parameter tunning:**

As we will improve our model by taking changing parameters.For this Project we will use grid search cv for finding the best parameter for our model the configuration that we will use will be

param\_grid = {"criterion": ["mse", "mae"],

"min\_samples\_split": [10, 20, 40],

"max\_depth": [2, 6, 8],

"min\_samples\_leaf": [20, 40, 100],

"max\_leaf\_nodes": [5, 20, 100],

}

**3.3 Random Forest**

Random forests or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks, that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set).

Random forest functions in below way

1. Draws a bootstrap sample from training data.
2. For each sample grow a decision tree and at each node of the tree
3. Ramdomly draws a subset of mtry variable and p total of features that are available
4. Picks the best variable and best split from the subset of mtry variable
5. Continues until the tree is fully grown.

**Hyper parameter tunning:**

As we will improve our model by taking changing parameters.For this Project we will use grid search cv for finding the best parameter for our model the configuration that we will use will be

param\_grid = {

'bootstrap': [True],

'max\_depth': [80, 90, 100, 110],

'n\_estimators': [100,200]

}

**3.4 knn(k -Nearest Neighbour)**

Knn is one of the simplest algorithm in logical way. It is used for both classification and regression

It place the point around the points which have similar features.It is instance based method.In regression we take mean of all the points around the query point to compute its value.

**Hyper parameter tunning:**

As one of the important parameter of the knn is k value which is number of neighbours.

We will be using for loop and will train the model on different k value and find the best one for our models

def knn(x\_train,y\_train,x\_test,y\_test):

diff\_neighbours = [1,2,3,4,5,6,7,8,9]

for i in range(0,len(diff\_neighbours)):

rmse\_errors = knn\_model(x\_train,y\_train,x\_test,y\_test,diff\_neighbours[i])

print("rmse of knn for train data for dpth {} is {}".format(diff\_neighbours[i],rmse\_errors['rmse\_train']))

print("rmse of knn for test data for dpth {} is {}".format(diff\_neighbours[i],rmse\_errors['rmse\_test']))

knn(X\_train,Y\_train,X\_test,Y\_test)

**Model Selection**

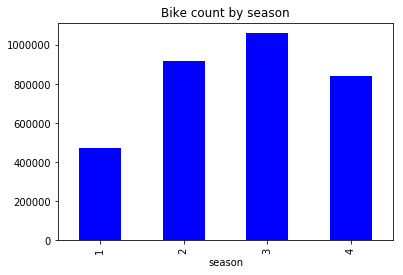
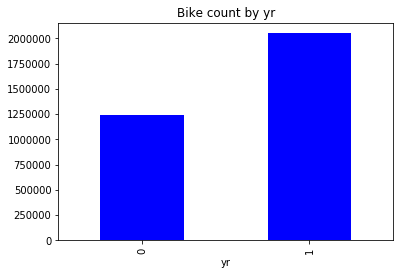
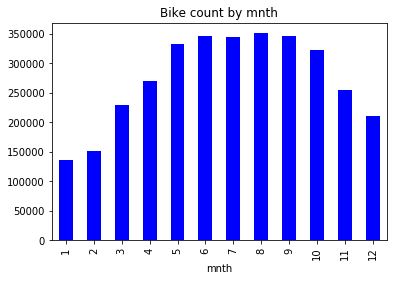
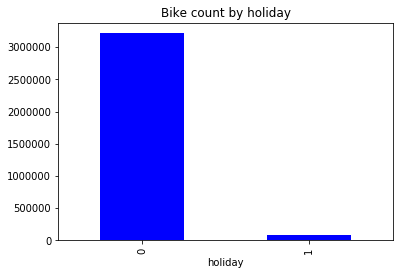
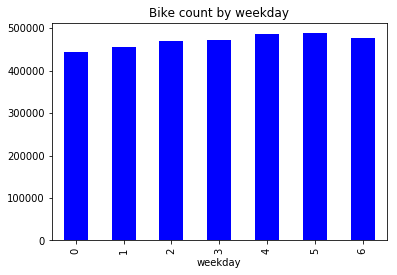
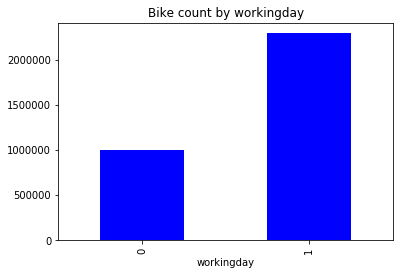
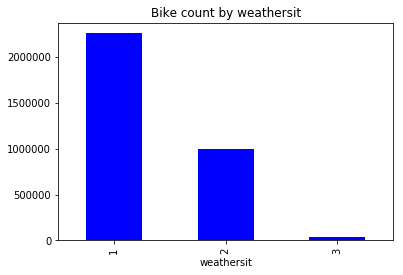
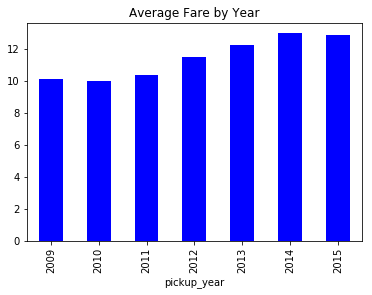
As we predicted counts for Bike Rental using three Models Linear regression,Decision tree,Knn,Random Forest.We have also done hyperparameter

**Conclusion**: - For the Bike Rental Data Random forest Model is best model to predict the count.

6

**Appendix A**

Relationship between diff predictor variables and target variables.



**Appendix B - Python Code**

# coding: utf-8

# <h3>Problem statement</h3>

# <p>

# The objective of this Case is to Predication of bike rental count on daily based on the

# environmental and seasonal settings.

# </p>

#

#

# In[1]:

#importing all the libraries

#pandas => for dataframe manipulation

#seaborn => wriiten on top of matplotlib for data visualization

#random => for random number generator

#sklearn => for machine learning algorithms

#statsmodel => for performing statistical tests like VIF,Rsquare,Adjusted Rsquare

import pandas as pd

import seaborn as sns

import numpy as np

import random

from math import radians, cos, sin, asin, sqrt

from sklearn.model\_selection import train\_test\_split,GridSearchCV

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

import warnings

warnings.filterwarnings('ignore')

from patsy import dmatrices

import statsmodels.formula.api as sm

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

import matplotlib.pyplot as plt

random.seed(113)

# In[2]:

df = pd.read\_csv("day.csv");

# In[3]:

#Let us check the shape of dataset

print(df.shape)

# In[4]:

"""

We have 731 rows and 16 columns in dataset

"""

#Let us have some information about the dataframe like null values and types

df.info()

# In[5]:

df.head()

# In[6]:

df.isnull().sum()

# In[7]:

"""

For better understanding of data mapping numerical categories in to their respective strings

"""

# d = {1 : 'springer', 2:'summer' ,3:'fall',4:'winter'}

# df['season'] = df['season'].map(d)

# In[8]:

df.head()

# In[9]:

df['season'].value\_counts()

# In[10]:

df['yr'].value\_counts()

"""

0 here means 2011

1 here means 2012

you can map as above for better understanding

"""

# In[11]:

"""

For better understanding of data mapping numerical categories in to their respective strings

"""

# d = {1:'jan',

# 2:'feb',

# 3:'march',

# 4:'april',

# 5:'may',

# 6:'june',

# 7:'july',

# 8:'aug',

# 9:'sep',

# 10:'oct',

# 11:'nov',

# 12:'dec'

# }

# df['mnth'] = df['mnth'].map(d)

# In[12]:

df['mnth'].value\_counts()

# In[13]:

df['holiday'].value\_counts()

"""

0 here means No holiday

1 here means holiday

"""

# In[14]:

"""

For better understanding of data mapping numerical categories in to their respective strings

"""

# d = {

# 0:'sun',

# 1:'mon',

# 2:'tue',

# 3:'wed',

# 4:'thu',

# 5:'fri',

# 6:'sat',

# }

# df['weekday'] = df['weekday'].map(d)

# In[15]:

"""

For better understanding of data mapping numerical categories in to their respective strings

"""

# d = {

# 1:'clear\_or\_few',

# 2:'mist\_and\_cloudy',

# 3:'lightrain\_or\_snow',

# 4:'heavy\_rain'

# }

# df['weathersit'] = df['weathersit'].map(d)

# In[16]:

df['weathersit'].value\_counts()

# In[17]:

df.head()

# In[18]:

ax = sns.distplot(df['temp'])

plt.show()

# In[19]:

ax = sns.distplot(df['atemp'])

plt.show()

# In[20]:

ax = sns.distplot(df['windspeed'])

plt.show()

# In[21]:

ax = sns.distplot(df['hum'])

plt.show()

# In[22]:

ax = sns.distplot(df['cnt'])

plt.show()

# In[23]:

# sns.set(style="ticks", color\_codes=True)

# g = sns.pairplot(df, vars=["temp", "atemp","windspeed",'hum','cnt'])

# plt.show()

# In[24]:

"""

visualizing all columns one by one to take insights from the data with respect to target variable count

"""

# cols = ['season','yr','mnth','holiday','weekday','workingday','weathersit']

df.groupby('season')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by season')

plt.show()

# In[25]:

##As we can clearly see in season 2 and 3 bike rental count increases as compared to 1 and 4

# In[26]:

df.groupby('yr')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by yr');

plt.show()

# In[27]:

##As from the graph we can say that the growth of the requirements of bike increases in 2012 as compared to 2011

# In[28]:

df.groupby('mnth')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by mnth')

plt.show()

# In[29]:

df.groupby('holiday')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by holiday');

plt.show()

# In[30]:

## As expected the requirement of bike is more on workdays rather than on holidays

# In[31]:

df.groupby('weekday')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by weekday')

plt.show()

# In[32]:

df.groupby('workingday')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by workingday')

plt.show()

# In[33]:

df.groupby('weathersit')['cnt'].sum().plot.bar(color = 'b');

plt.title('Bike count by weathersit');

plt.show()

# In[34]:

##Let us do some testing on numerical variables for multicollinearity we will only use temp,hum,atemp,

#windspeed as these are only numerical variable

"""

We will use variance inflation factor to detect the multicollinearity

"""

df\_subset = df[['cnt','temp','hum','atemp','windspeed']].dropna()

#subset the dataframe

# In[35]:

features = ''

for column in df\_subset.columns:

if(column != 'cnt'):

features += '+'+column

print(features)

# In[36]:

# %%capture

#gather features

y, X = dmatrices('cnt ~'+features , df, return\_type='dataframe')

# In[37]:

# For each X, calculate VIF and save in dataframe

vif = pd.DataFrame()

vif["VIF Factor"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

vif["features"] = X.columns

# In[38]:

"""

tabulating the vif value of differet columns in a variable

"""

vif.round(1)

# <p>As from the above table we can say thet temp and atemp has very high value of VIF so they both are

# explaining the same variance of data so we can drop one of the other</p>

# In[39]:

"""

Also displaying correlation plot to detect the collinearity in data

"""

f, ax = plt.subplots(figsize=(10, 8))

corr =df.corr()

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True),

square=True, ax=ax)

plt.show()

# <h4>As from the correlation plot we can see there is high relation between</h4>

# <ul>

# <li>1.temp and atemp </li>

# <li> casual ,registered,cnt</li>

# <ul>

# So we can remove on of the two variable from temp and atemp

#

# and also cnt is summation of registered and casual so we can just keep cnt as it contains summation value of two variables and are highly correlated

# In[40]:

df = df.drop(['instant','registered','casual','temp','dteday'],axis=1)

# In[41]:

f, ax = plt.subplots(figsize=(10, 8))

corr =df.corr()

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True),

square=True, ax=ax)

plt.show()

# In[42]:

df.head()

# In[43]:

corrs = df.corr()

corrs['cnt'].plot.bar(color = 'b');

plt.title('cnt');

plt.show()

# In[44]:

#As we cannot feed categorical data in some regression models like linear regression

#so we will convert the data in to numerical one.

#converting season,weathersit,yr,mnth

#We are not required to convert working day or holiday as they cotain value only zero or one

cat\_columns = ['season','weathersit','yr','mnth']

df = pd.get\_dummies(df, prefix\_sep="\_\_",

columns=cat\_columns)

# In[71]:

plt.boxplot(df['windspeed'])

plt.xlabel('windspeed')

plt.show()

# In[72]:

plt.boxplot(df['hum'])

plt.xlabel('hum')

plt.show()

# In[73]:

plt.boxplot(df['atemp'])

plt.xlabel('atemp')

plt.show()

# In[48]:

#Detect & Delete Outliers

cnames = ["atemp","hum","windspeed"]

for i in cnames :

print (i)

q75,q25 = np.percentile(df.loc[:,i],[75,25])

iqr = q75-q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

print (min)

print (max)

df =df.drop(df[df.loc[:,i] < min].index)

df = df.drop(df[df.loc[:,i] > max].index)

# In[49]:

plt.boxplot(df['windspeed'])

plt.show()

# In[50]:

plt.boxplot(df['hum'])

plt.show()

# In[51]:

plt.boxplot(df['atemp'])

plt.show()

# In[52]:

df.head()

# In[53]:

"""

We can split our data in 80:20

where 80% data will be our training data and 20% data will be our test data

"""

train, test = train\_test\_split(df, test\_size=0.2)

# In[54]:

train.head()

# In[55]:

X\_train = train.drop(['cnt'],axis = 1)

Y\_train = train[['cnt']]

X\_test = test.drop(['cnt'],axis = 1)

Y\_test = test[['cnt']]

# In[56]:

X\_train.head()

# In[57]:

Y\_train.head()

# In[58]:

#Calculate MAPE

def MAPE(y\_true, y\_pred):

mape = np.mean(np.abs((y\_true - y\_pred) / y\_true))\*100

return mape

# In[59]:

def linear\_regression(x\_train,y\_train,x\_test,y\_test):

regressor = LinearRegression()

regressor = regressor.fit(x\_train, y\_train) #training the algorithm

y\_pred = regressor.predict(x\_train)

rmse\_train = sqrt(mean\_squared\_error(y\_train, y\_pred))

# print("rmse for simple linear regression for train data",rmse\_train)

y\_pred\_test = regressor.predict(x\_test)

rmse\_test = sqrt(mean\_squared\_error(y\_test, y\_pred\_test))

mape\_test = MAPE(y\_test, y\_pred\_test)

mape\_train = MAPE(y\_train, y\_pred)

SS\_Residual = sum((y\_test.values-y\_pred\_test)\*\*2)

SS\_Total = sum((y\_test.values-np.mean(y\_test.values))\*\*2)

r\_squared = 1 - (float(SS\_Residual))/SS\_Total

adjusted\_r\_squared = 1 - (1-r\_squared)\*(len(y\_test)-1)/(len(y\_test)-x\_test.shape[1]-1)

print("R squared",r\_squared)

print("adjusted r square",adjusted\_r\_squared)

#

# print("rmse for simple linear regression for test data",rmse\_test)

return {'rmse\_train':rmse\_train,'rmse\_test':rmse\_test,'mape\_test':mape\_test,'mape\_train':mape\_train}

# In[60]:

linear\_regression(X\_train,Y\_train,X\_test,Y\_test)

# In[61]:

def decision\_trees(x\_train,y\_train,x\_test,y\_test):

clf = DecisionTreeRegressor(max\_depth=4,

min\_samples\_split=5,

max\_leaf\_nodes=10)

"""

We will be using GridSearchCV for hyperperameter tunning

"""

param\_grid = {"criterion": ["mse", "mae"],

"min\_samples\_split": [10, 20, 40],

"max\_depth": [2, 6, 8],

"min\_samples\_leaf": [20, 40, 100],

"max\_leaf\_nodes": [5, 20, 100],

}

grid\_cv\_dtm = GridSearchCV(clf, param\_grid, cv=5)

grid\_cv\_dtm = grid\_cv\_dtm.fit(x\_train,y\_train)

# clf = clf.fit(x\_train, y\_train)

y\_pred\_tree = grid\_cv\_dtm.predict(x\_train)

rmse\_train = sqrt(mean\_squared\_error(y\_train, y\_pred\_tree))

# print("rmse for decision trees for train data",rmse)

y\_pred\_tree\_test = grid\_cv\_dtm.predict(x\_test)

rmse\_test = sqrt(mean\_squared\_error(y\_test,y\_pred\_tree\_test))

y\_test = np.reshape(y\_test,(y\_test.shape[0],1))

y\_pred\_tree\_test = np.reshape( y\_pred\_tree\_test,( y\_pred\_tree\_test.shape[0],1))

mape\_test = MAPE(Y\_test, y\_pred\_tree\_test)

print("R-Squared::{}".format(grid\_cv\_dtm.best\_score\_)

)

# #for rsquare and adjusted

# SS\_Residual = sum((y\_test.values-y\_pred\_tree\_test)\*\*2)

# SS\_Total = sum((y\_test.values-np.mean(y\_test.values))\*\*2)

# r\_squared = 1 - (float(SS\_Residual))/SS\_Total

# adjusted\_r\_squared = 1 - (1-r\_squared)\*(len(y\_test)-1)/(len(y\_test)-x\_test.shape[1]-1)

# print("R squared",r\_squared)

# print("adjusted r square",adjusted\_r\_squared)

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

# print("rmse for decision trees for test data",rmse)

print(mape\_test)

return {'rmse\_train':rmse\_train,'rmse\_test':rmse\_test}

# In[62]:

decision\_trees\_rmse = decision\_trees(X\_train,Y\_train,X\_test,Y\_test)

print("decison trees",decision\_trees\_rmse)

# In[63]:

def knn\_model(x\_train,y\_train,x\_test,y\_test,n):

neigh = KNeighborsRegressor(n\_neighbors=n)

neigh = neigh.fit(X\_train,Y\_train)

y\_pred\_tree\_knn = neigh.predict(X\_train)

rmse\_train = sqrt(mean\_squared\_error(Y\_train,y\_pred\_tree\_knn))

# print("rmse of knn for train data for {} neighbour is {}".format(diff\_neighbours[i],rmse))

y\_pred\_knn\_test = neigh.predict(X\_test)

rmse\_test = sqrt(mean\_squared\_error(Y\_test,y\_pred\_knn\_test))

#Reshaping the array as the value which we get from knn is not shaped that we can get the mape

mape\_test = MAPE(Y\_test,y\_pred\_knn\_test)

#for rsquare and adjusted

# SS\_Residual = sum((y\_test.values-y\_pred\_knn\_test)\*\*2)

# SS\_Total = sum((y\_test.values-np.mean(y\_test.values))\*\*2)

# r\_squared = 1 - (float(SS\_Residual))/SS\_Total

# adjusted\_r\_squared = 1 - (1-r\_squared)\*(len(y\_test)-1)/(len(y\_test)-x\_test.shape[1]-1)

# print("R squared",r\_squared)

# print("adjusted r square",adjusted\_r\_squared)

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

print(mape\_test)

return {'rmse\_train':rmse\_train,'rmse\_test':rmse\_test,'mape\_test':mape\_test}

# print("rmse of knn for test data for {} neighbour is {}".format(diff\_neighbours[i],rmse))

#tunning one of the hyperperameters

def knn(x\_train,y\_train,x\_test,y\_test):

diff\_neighbours = [1,2,3,4,5,6,7,8,9]

for i in range(0,len(diff\_neighbours)):

rmse\_errors = knn\_model(x\_train,y\_train,x\_test,y\_test,diff\_neighbours[i])

print("rmse of knn for train data for dpth {} is {}".format(diff\_neighbours[i],rmse\_errors['rmse\_train']))

print("rmse of knn for test data for dpth {} is {}".format(diff\_neighbours[i],rmse\_errors['rmse\_test']))

knn(X\_train,Y\_train,X\_test,Y\_test)

# In[64]:

#creating

def random\_forest\_model(x\_train,y\_train,x\_test,y\_test):

# print(depth)

param\_grid = {

'bootstrap': [True],

'max\_depth': [80, 90, 100, 110],

'n\_estimators': [100,200]

}

# Create a based model

regr = RandomForestRegressor()

# Instantiate the grid search model

grid\_search = GridSearchCV(regr, param\_grid = param\_grid,

cv = 3, n\_jobs = 10, verbose = 2)

# regr = RandomForestRegressor(max\_depth=depth, random\_state=0, n\_estimators=100)

regr = grid\_search.fit(x\_train,y\_train)

# feature\_importances = pd.DataFrame({'feature': X\_train.columns,

# 'importance': regr.feature\_importances\_}).\

# sort\_values('importance', ascending = False).set\_index('feature')

y\_pred\_tree\_random\_forest = regr.predict(X\_train)

# print(y\_pred\_tree\_random\_forest)

rmse\_train = sqrt(mean\_squared\_error(Y\_train, y\_pred\_tree\_random\_forest))

# print("rmse of Random trees for train data for dpth {} is {}".format(diff\_depths[i],rmse))

y\_pred\_tree\_test\_random = regr.predict(X\_test)

rmse\_test = sqrt(mean\_squared\_error(Y\_test,y\_pred\_tree\_test\_random))

# y\_train = np.reshape(y\_train,(y\_train.shape[0],1))

# y\_pred\_tree\_random\_forest = np.reshape( y\_pred\_tree\_random\_forest,( y\_pred\_tree\_random\_forest[0],1))

# mape\_train = MAPE(y\_train,y\_pred\_tree\_random\_forest)

# errors = abs(y\_pred\_tree\_test\_random - Y\_test)

y\_test = np.reshape(y\_test,(y\_test.shape[0],1))

y\_pred\_tree\_test\_random = np.reshape(y\_pred\_tree\_test\_random,(y\_pred\_tree\_test\_random.shape[0],1))

mape\_test = MAPE(y\_test,y\_pred\_tree\_test\_random)

# Calculate mean absolute percentage error (MAPE)

# mape = 100 \* (errors / y\_test)

# mape\_test = MAPE(Y\_test,y\_pred\_tree\_test\_random)

# mape\_train = MAPE(Y\_train, y\_pred\_tree\_random\_forest)

#for rsquare and adjusted

SS\_Residual = sum((y\_test.values-y\_pred\_tree\_test\_random)\*\*2)

SS\_Total = sum((y\_test.values-np.mean(y\_test.values))\*\*2)

r\_squared = 1 - (float(SS\_Residual))/SS\_Total

adjusted\_r\_squared = 1 - (1-r\_squared)\*(len(y\_test)-1)/(len(y\_test)-x\_test.shape[1]-1)

print("R squared",r\_squared)

print("adjusted r square",adjusted\_r\_squared)

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

# print("rmse of Random trees for test data for depth {} is {}".format(diff\_depths[i],rmse))

# print(feature\_importances.head(10))

print("mape\_test",mape\_test)

return {'rmse\_train':rmse\_train,'rmse\_test':rmse\_test,'mape\_test': mape\_test}

# In[65]:

#for hyperperameter tunning

# def random\_forest(x\_train,y\_train,x\_test,y\_test):

# diff\_depths = [1,2,3,4,5,6,7,8,9,10,11,12,13]

# for i in range(0,len(diff\_depths)):

# # print(diff\_depths[i])

# rmse\_errors = random\_forest\_model(x\_train,y\_train,x\_test,y\_test,diff\_depths[i])

# print("rmse of Random trees for train data for dpth {} is {}".format(diff\_depths[i],rmse\_errors['rmse\_train']))

# print("rmse of Random trees for test data for dpth {} is {}".format(diff\_depths[i],rmse\_errors['rmse\_test']))

# return

random\_forest\_model(X\_train,Y\_train,X\_test,Y\_test)

# <h6>Comparing all the models that we have built random forest has performed best </h6>

# <p>hence we will pick random forest for our production deployment</p>

# In[66]:

#As accuracy is 100 - mape

#mape for random forest is 14.09

accuracy = 100 - 14.09

print(accuracy)

# In[67]:

#accuracy for this model has 85.91

#mape for this model is 14.09

#R-square is 86%

#Adjusted R square 83%