# BIKE RENTAL PREDICTION

**Predication of bike rental count on daily based on the environmental and seasonal settings**

APRIL 16

**Edwisor**
**Authored by: Saurav Joshi**

# Contents

Chapter 1

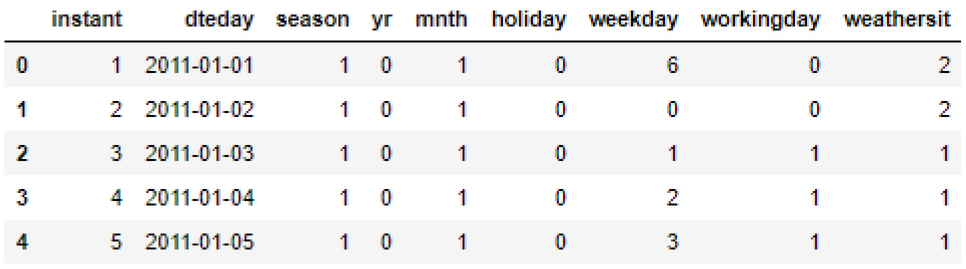
# Introduction

## 1.1 Problem Statement

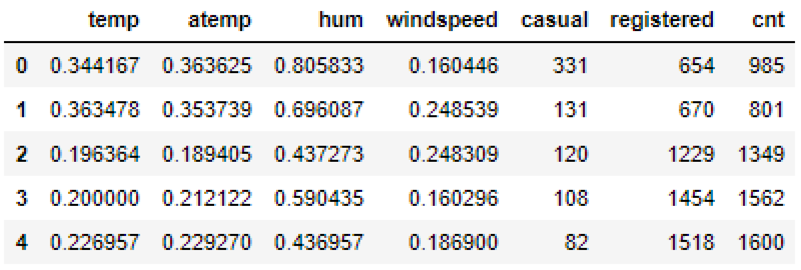
The objective of this case is to make Prediction of bike rental count based on the environmental and seasonal settings.

## 1.2 Data

The main objective deals with constructing a model that predicts the rental count by understanding the user behavior and usage pattern. The data given to us is a historical data for the year 2011 and 2012. A quick sneak-peak into the data we have is shown below:



| | instant | dteday | season | yr | mnth | holiday | weekday | workingday | weathersit |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2011-01-01 | 1 | 0 | 1 | 0 | 6 | 0 | 2 |
| 1 | 2 | 2011-01-02 | 1 | 0 | 1 | 0 | 0 | 0 | 2 |
| 2 | 3 | 2011-01-03 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 3 | 4 | 2011-01-04 | 1 | 0 | 1 | 0 | 2 | 1 | 1 |
| 4 | 5 | 2011-01-05 | 1 | 0 | 1 | 0 | 3 | 1 | 1 |

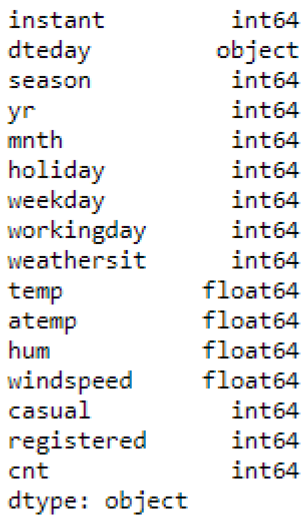


| | temp | atemp | hum | windspeed | casual | registered | cnt |
|---|---|---|---|---|---|---|---|
| 0 | 0.344167 | 0.363625 | 0.805833 | 0.160446 | 331 | 654 | 985 |
| 1 | 0.363478 | 0.353739 | 0.696087 | 0.248539 | 131 | 670 | 801 |
| 2 | 0.196364 | 0.189405 | 0.437273 | 0.248309 | 120 | 1229 | 1349 |
| 3 | 0.200000 | 0.212122 | 0.590435 | 0.160296 | 108 | 1454 | 1562 |
| 4 | 0.226957 | 0.229270 | 0.436957 | 0.186900 | 82 | 1518 | 1600 |

Table 1 – Head of the dataframe

## 1.3 Data Structure and Size

The data provided has a shape (731, 16). The data has total 16 attributes and 731 observations with missing values spread. The variables qualifying as numerical variables from the dataset are normalized.
The overall variables received have the following data-type when loaded.



```
instant          int64
dteday          object
season           int64
yr               int64
mnth             int64
holiday          int64
weekday          int64
workingday       int64
weathersit       int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
dtype: object
```

**Table 2 – Data-types when loaded**

Chapter 2

# Exploratory Data Analysis

As stated by Wikipedia –

*"In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods"*

Exploratory data analysis is for seeing what the data can tell us beyond the formal modelling or hypothesis testing task. It deals with performing initial investigations on data so as to discover patterns, to spot anomalies.

The initial insights that were seen could be figured out as follows

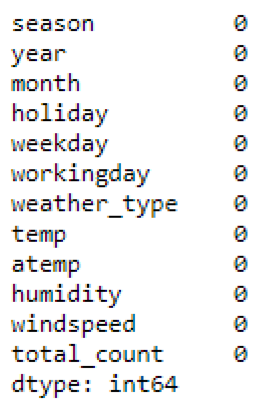
1. The feature ***dteday*** is providing us the information related to time-dimension, however the juice of the variable is extracted and can be contained from features such as year, month, weekday. Henceforth dteday is not needed.
2. The feature ***instant*** provides us sequencing and doesn't provide any information helpful. This too shall be dropped.
3. Lastly the feature variables ***casual*** and ***registered*** are leaky variables because the total_count is generated from these two. Hence they are also not needed.

Exploratory data analysis further includes cleaning the data, handling outliers, visualizing the data to get further insights.

## 2.1 Data-Preprocessing

The realization of missing values is important to understand, when we are trying to successfully manage the data. These can lead to inaccurate inference if not handled properly. Handling missing values often includes checking for the relative percentage of the missing values with the total observations. Any variable possessing missing percentage more than a threshold percentage is dropped as it shall not be providing any deterministic inference to the model else includes imputing the missing values through statistical or any other suitable technique.

The missing value analysis for the data provided gives us the following:



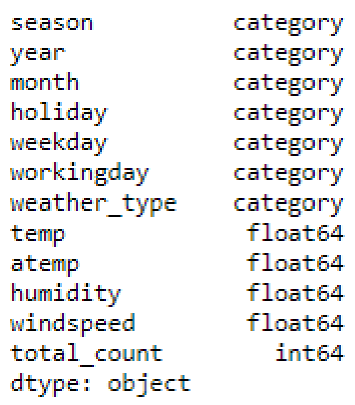
```
season          0
year            0
month           0
holiday         0
weekday         0
workingday      0
weather_type    0
temp            0
atemp           0
humidity        0
windspeed       0
total_count     0
dtype: int64
```

**Table 3 – Missing Value Percentage**

As observed from the data above, our dataset isn't missing any values, we're good in this handling.

## 2.2 Data type Handling

It is an important pre-processing step to transform the data into their correct data-type. This typically requires domain knowledge and understanding of the problem statement. Data type handling helps in setting relationships between measures and dimensions. Also this helps in better training of the models as helps in deciding which features can be taken as running data and which can be taken as categories.
Below are the data-types after processing.



```
season          category
year            category
month           category
holiday         category
weekday         category
workingday      category
weather_type    category
temp             float64
atemp            float64
humidity         float64
windspeed        float64
total_count        int64
dtype: object
```

**Table 4 – Data-type after processing**

## 2.3 Univariate Data Analysis

We try by plotting the density plots so as to check the distribution of the data, check for the skewness. This shall help us acknowledge the spread of data and the affected data under the presence of outliers if any.

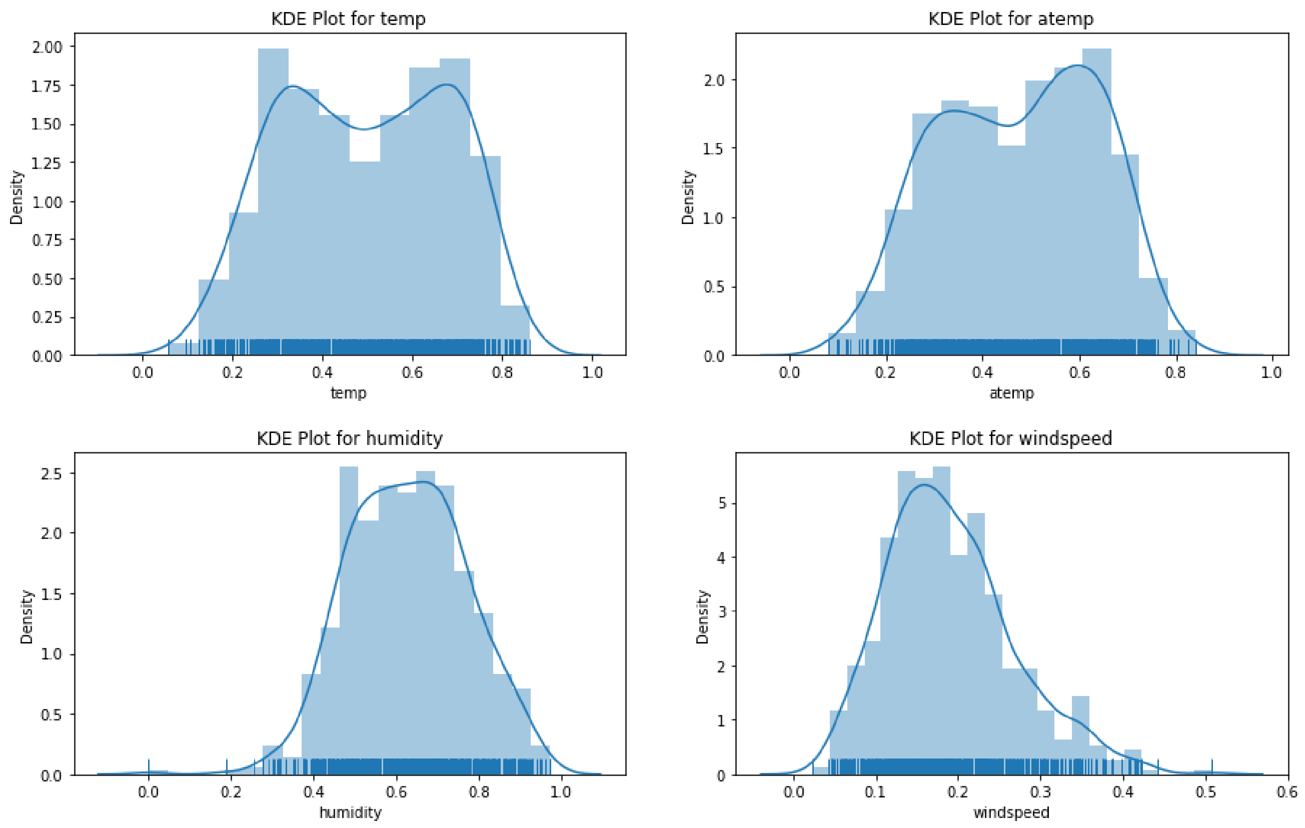


**Figure 1 – KDE Distributions**

It can be seen from the density plots above for the continuous variable that feature humidity is slightly left skewed and the feature windspeed is slightly right skewed.
The major concentration for humidity lies between 0.4 and 0.8 for the normalized values and 0.1 and 0.3 for the windspeed's normalized values.
The features temp and atemp doesn't possess as such skewness.

We shall now see the relation of every numerical variable with the target variable total_count by plotting a regression plot, to see if we can figure out any sort of trend at the primary stage.

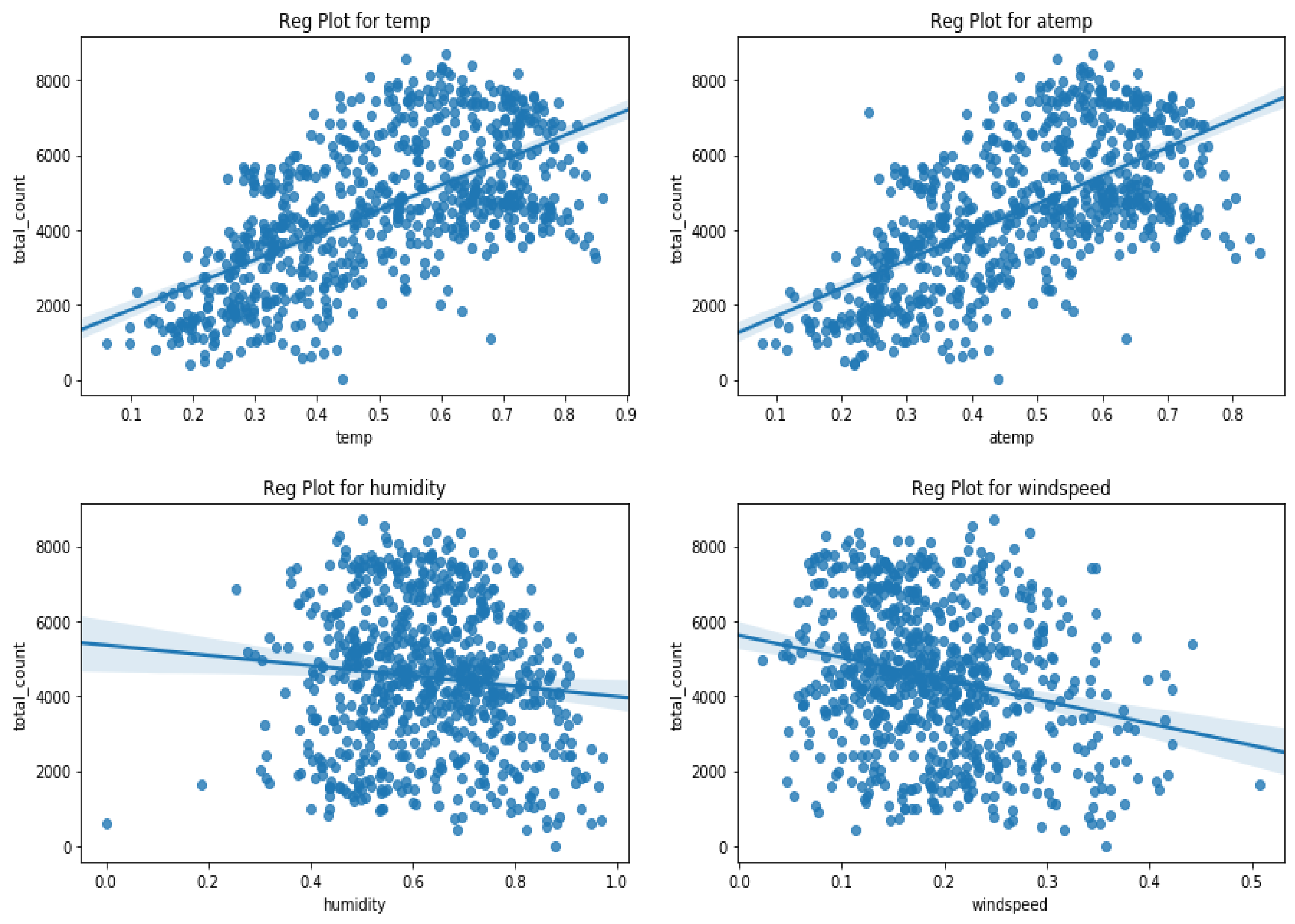


**Figure 2 – Reg Plot for Numerical Variables**

Quick notable insights from the Reg Plot for Numerical Variables

- Temp and atemp both show a linear positive trend with the total_count target variable.
- Windspeed show a slight negative linear trend with total_count
- Humidity is almost flat with total_count

Coming to categorical variables the following chart shows population with-hold overview by category and occurrence. This is useful in getting inference for stating a variable behavior given a new observation i.e. to which particular category the observation shall fall.
We'll be plotting both countplot and categorical bar plot for the categorical variables. The count plot shall shows us the frequency of data and bar plot shall help us know spread of data.

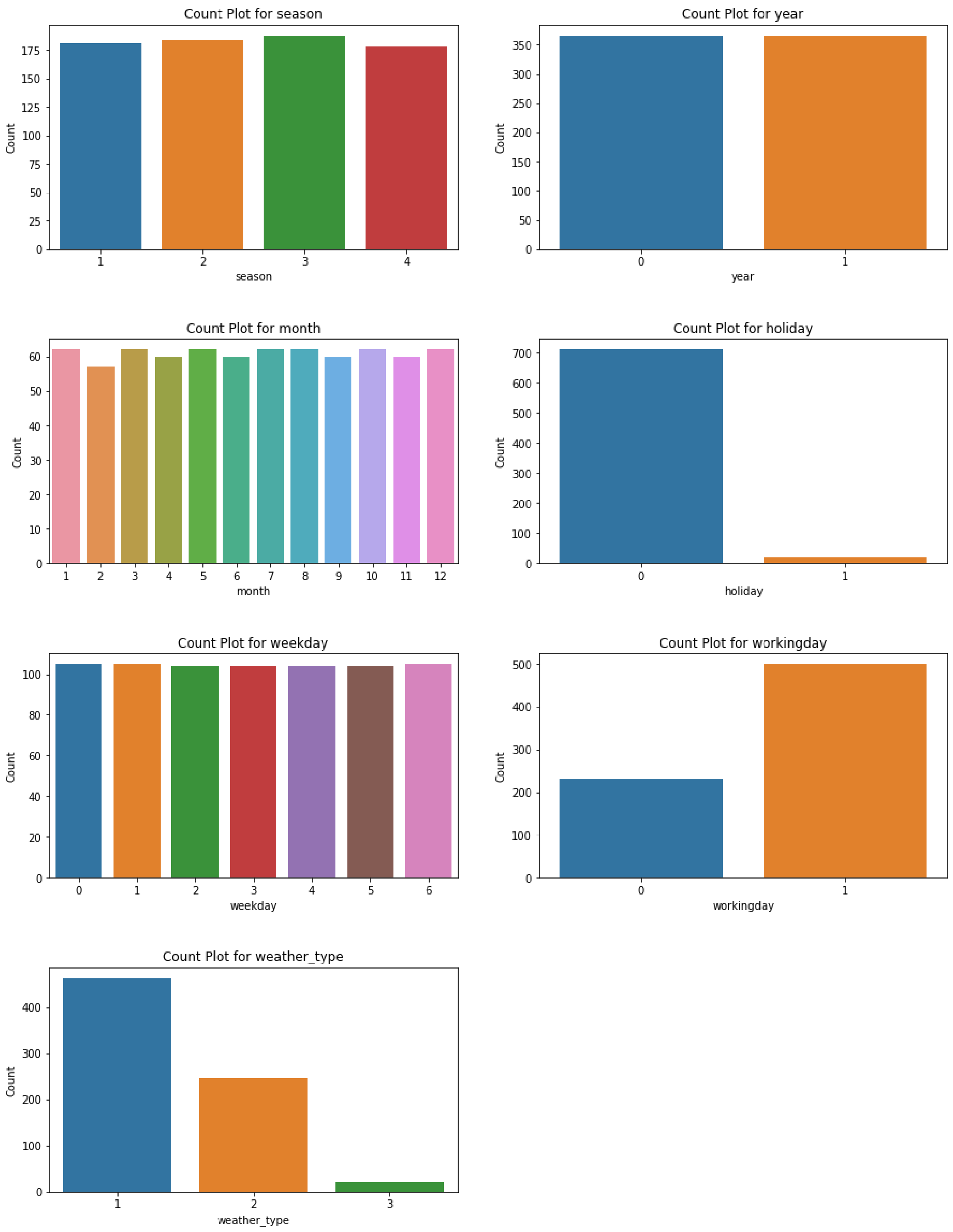


**Figure 3 – Count-plot for categorical Variables**

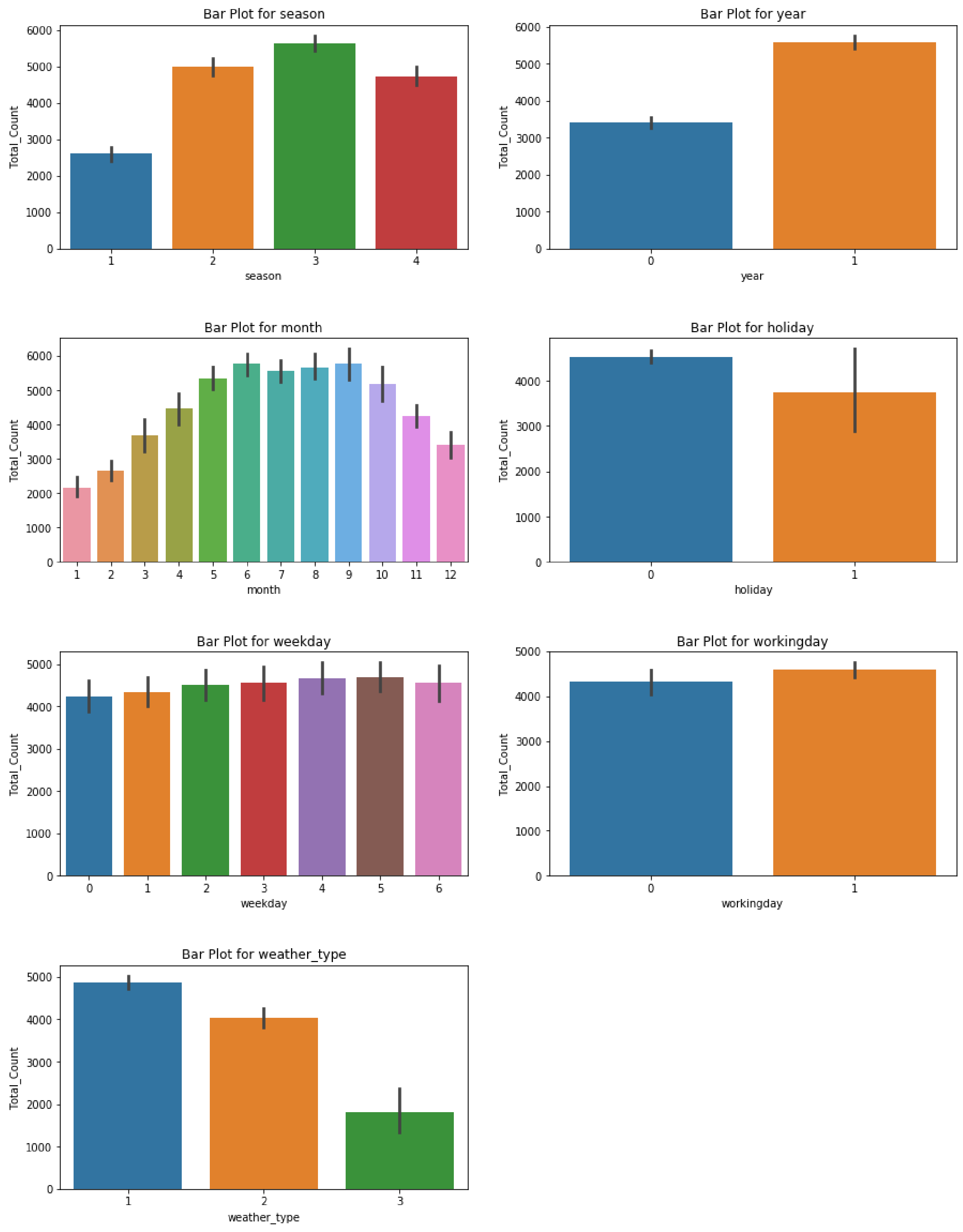


**Figure 4 – Bar-plot for categorical Variables**

Quick notable insights that we get from the count plot and bar plot above are:

- Workingday: The count of working days is more than holidays as from Fig3. Whereas the total_count for working days is just slightly higher than the non-working days from Fig4, as holiday factor play their part in thrusting count despite of low frequency.
- Weekday: All weekdays are equally spread as from Fig3. The bar plot also has slight variation present.
- Holiday: Holiday bar from countplot has very low number than the working days. The bar plot shows that the weekdays counts are greater than holiday count slightly.
- Month: The count plot shows equal presence in the dataset. Whereas the barplot shows months from June-September dominating the other months.
- Year: The count plot shows equal presence in the dataset. Whereas the barplot shows greater count in 2011 than 2012.
- Season: Countplot shows almost similar presence in the dataset, whereas the barplot shows the dominance in the season of summer and fall. However the spring season contributes in lower total_count as compared to the other three seasons.

## 2.4 Outlier Handling

The KDE plots significantly shows that the variables (*humidity and windspeed*) have been affected by the presence of outliers. These outliers are accountable for the skewness in the data.
A method for the detection of the outliers in the data is through checking the fall of data point more than 1.5 interquartile range below the first quartile and above the third quartile.

Following were the outliers detected from the data.

```
windspeed       13
humidity         2
total_count      0
atemp            0
temp             0
weather_type     0
workingday       0
weekday          0
holiday          0
month            0
year             0
season           0
dtype: int64
```

**Table 5 – Outlier presence in Variables**

The outliers can be processed by removing the values and imputing them back through suitable technique.

1. The first chart shows the outliers as seen through boxplots. With the dots describing the values outside the minimum and maximum boundaries if any.
2. The second chart show the boxplots after outlier handling.
3. The third chart shows the KDE plot after outlier handling.

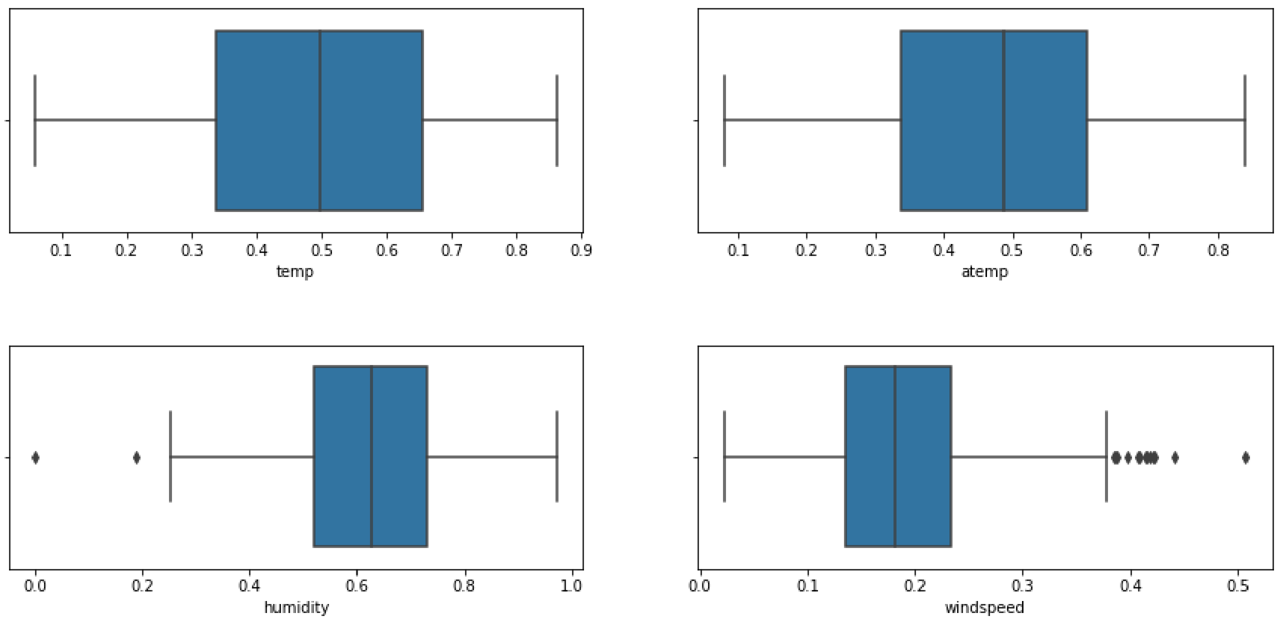


**Figure 5 – Boxplot before Outlier handling**

The outliers were removed using inter-quartile method.

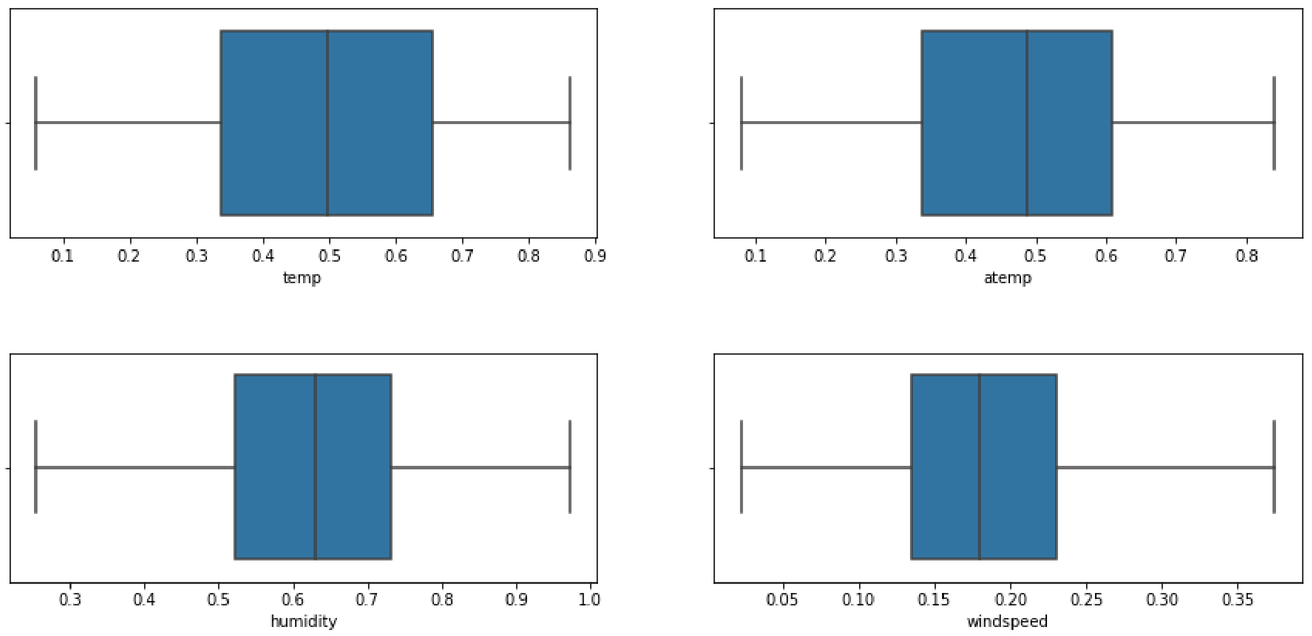


**Figure 6 – Boxplot after Outlier handling**

KDE plot after outlier handling show smoother density distribution. Also the rugs on the X-axis shows the outlier values missing. We can conclude that the skewness is handled by a far-extent by handling the outliers.

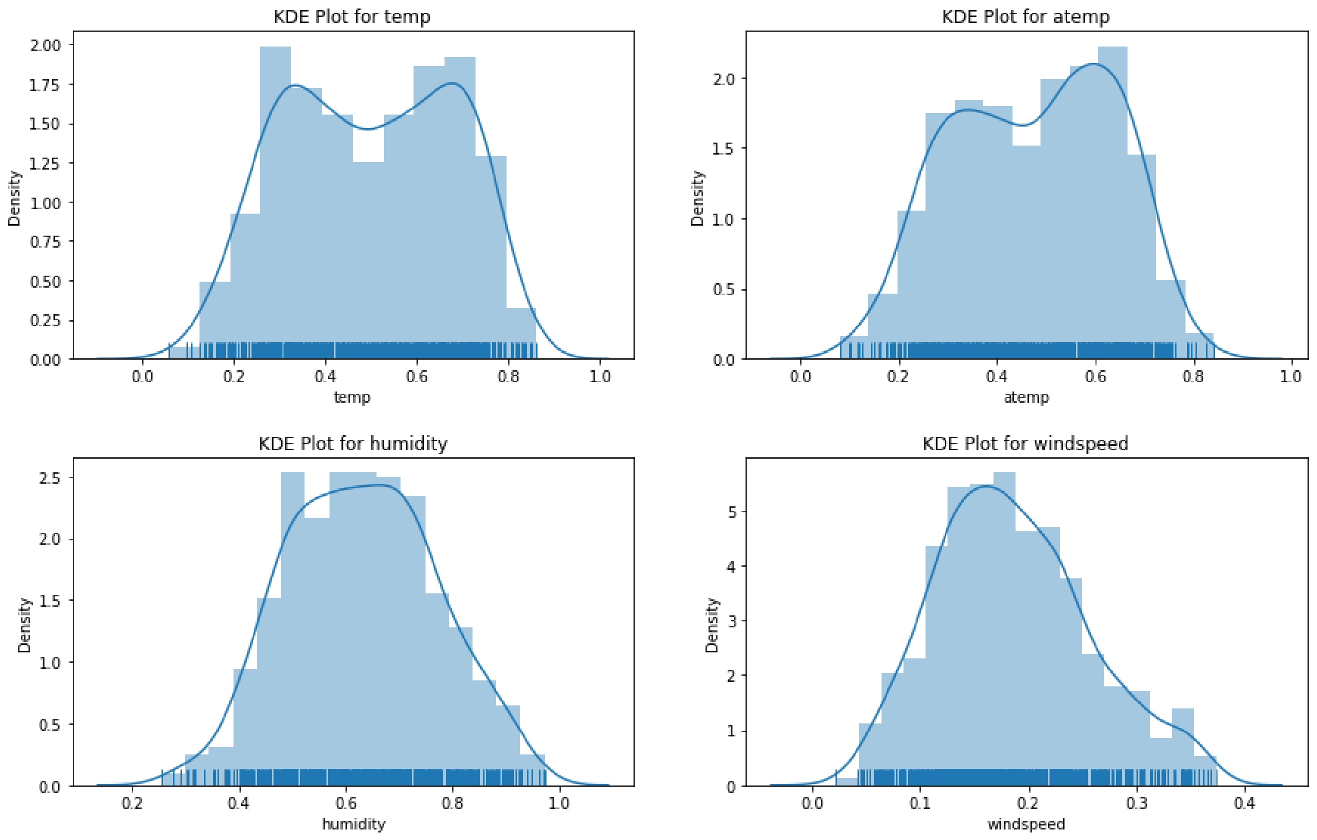


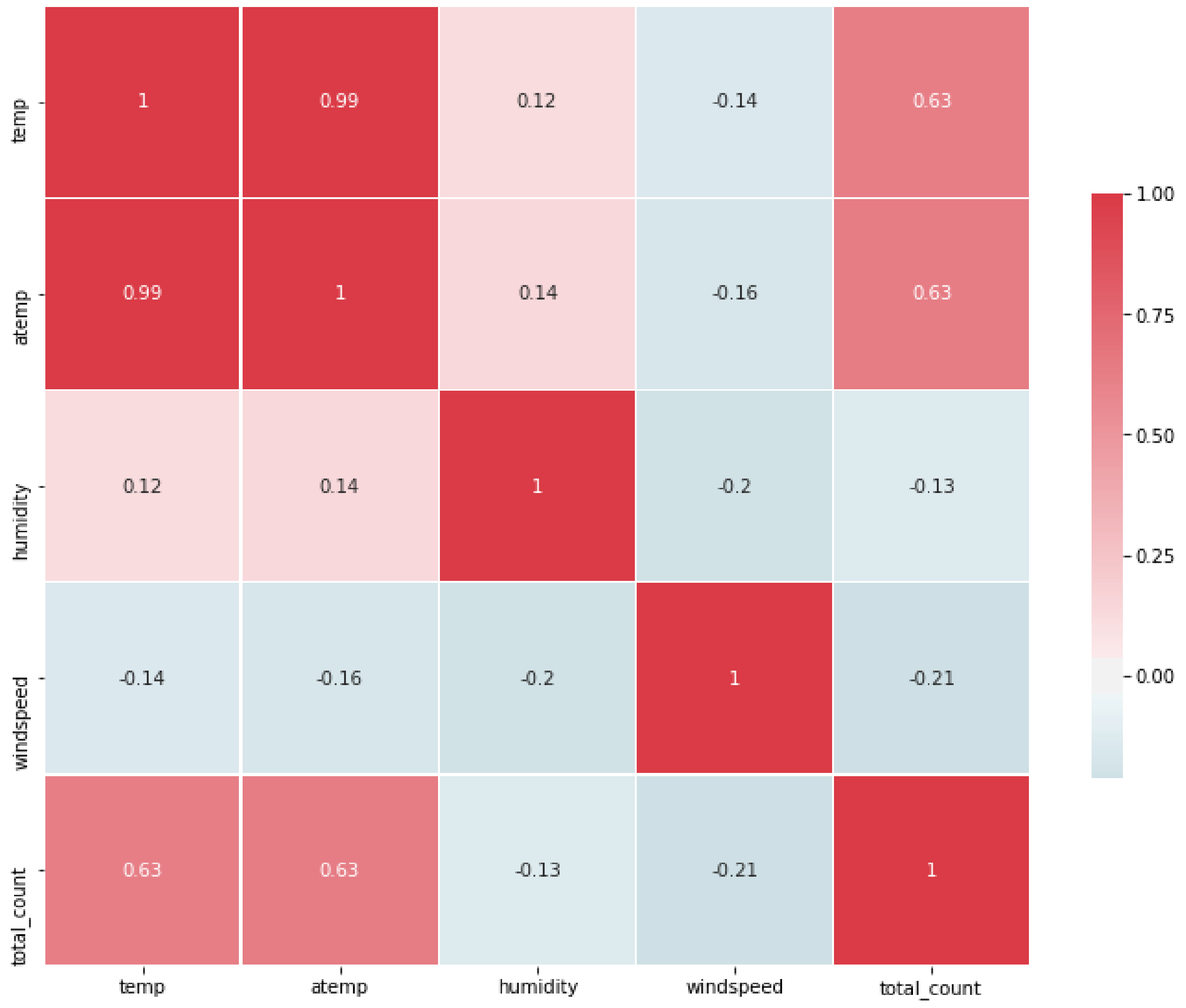
**Figure 7 – KDE after Outlier Handling**

## 2.5 Feature Selection

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.

**Pearson's Correlation Coefficient:**

Pearson's correlation coefficient is the test statistics that measures the statistical relationship, or association, between two continuous variables. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.



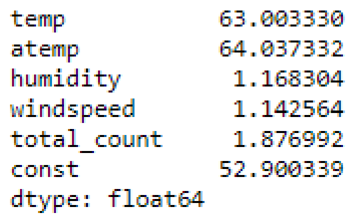
| | temp | atemp | humidity | windspeed | total_count |
|---|---|---|---|---|---|
| temp | 1 | 0.99 | 0.12 | -0.14 | 0.63 |
| atemp | 0.99 | 1 | 0.14 | -0.16 | 0.63 |
| humidity | 0.12 | 0.14 | 1 | -0.2 | -0.13 |
| windspeed | -0.14 | -0.16 | -0.2 | 1 | -0.21 |
| total_count | 0.63 | 0.63 | -0.13 | -0.21 | 1 |

**Figure 8 – Heat Map for Pearson's Correlation Coefficient**

The heat-map for the Pearson's Correlation Coefficient shows +0.99 for temp and atemp.

**Mulitcollinearity:**

Multicollinearity occurs when independent variables are correlated. Ideally the predictor variables should be independent. We use Variance Inflation Factor (VIF) values for checking for multicollinearity. We drop the variables possessing high values for VIF. Based on this temp shall be dropped.



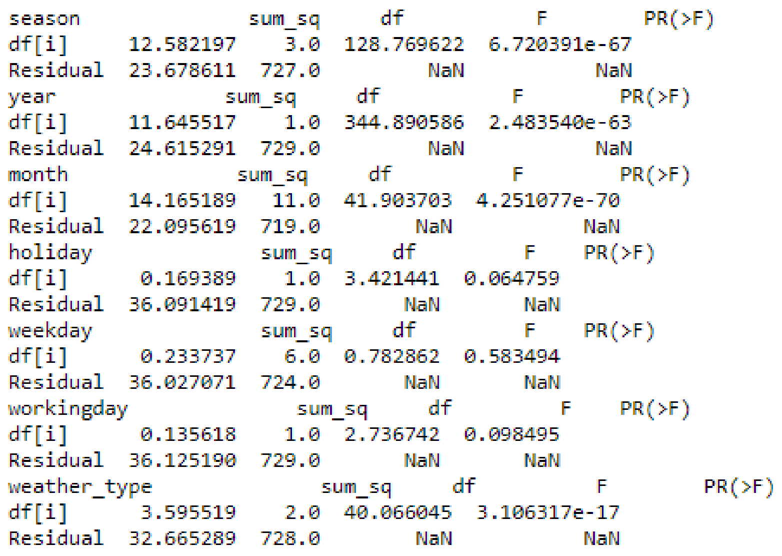
```
temp           63.003330
atemp          64.037332
humidity        1.168304
windspeed       1.142564
total_count     1.876992
const          52.900339
dtype: float64
```

**Table 6 – VIF Factor table for numerical features**

**ANOVA Test:**

Analysis of variance or ANOVA test is being used as we're having categorical variables as independent variables and a dependent numerical variable.

The test results in following p-values against the categorical variables



```
season               sum_sq     df           F        PR(>F)
df[i]     12.582197    3.0  128.769622  6.720391e-67
Residual  23.678611  727.0         NaN           NaN
year               sum_sq     df           F        PR(>F)
df[i]     11.645517    1.0  344.890586  2.483540e-63
Residual  24.615291  729.0         NaN           NaN
month               sum_sq     df          F        PR(>F)
df[i]     14.165189   11.0  41.903703  4.251077e-70
Residual  22.095619  719.0        NaN           NaN
holiday               sum_sq     df         F    PR(>F)
df[i]      0.169389    1.0  3.421441  0.064759
Residual  36.091419  729.0       NaN       NaN
weekday               sum_sq     df         F    PR(>F)
df[i]      0.233737    6.0  0.782862  0.583494
Residual  36.027071  724.0       NaN       NaN
workingday               sum_sq     df         F    PR(>F)
df[i]      0.135618    1.0  2.736742  0.098495
Residual  36.125190  729.0       NaN       NaN
weather_type               sum_sq     df          F        PR(>F)
df[i]      3.595519    2.0  40.066045  3.106317e-17
Residual  32.665289  728.0        NaN           NaN
```

**Table 7 – ANOVA Result**

The test shows variables weekday with p-value = 0.590 >> 0.05. Hence this shall be dropped.

Chapter 3

# Modelling

## 3.1 Data Modelling

The pre-processing has been done. Outliers have been handled, and the exploratory data analysis gave us some initial insights. We now create models for this processed data and validate the same.

For validation we split a portion out of the master data, apart from the one used for training the model. Since the problem is a regression problem, we come up with various models for making the prediction.

1. Gradient Boosting: Gradient boosted model was fitted onto the training set and provided the following feature importance.

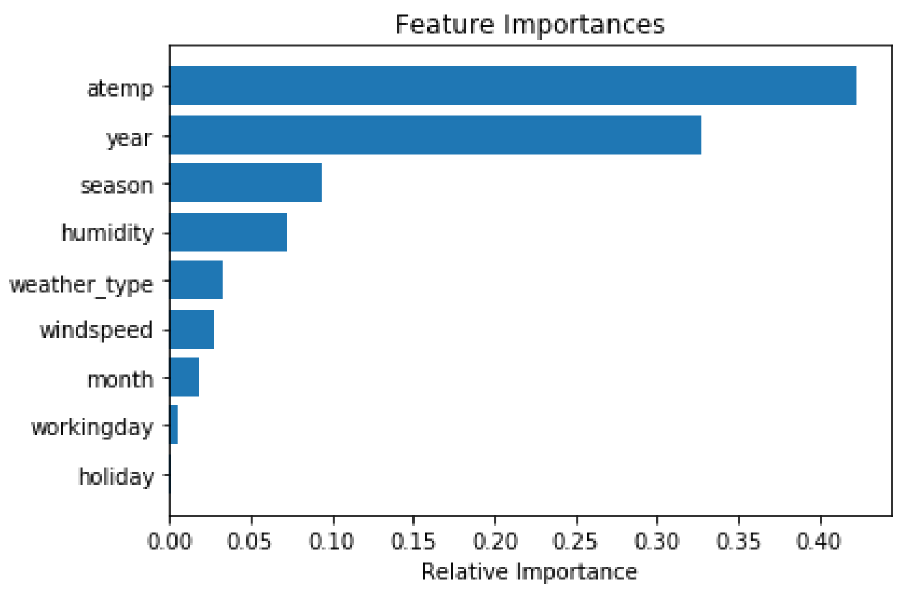


**Figure 9 – Gradient Boosting Feature Importance**

2. Random Forest: Random forest model was fitted onto the training set and provided the following feature importance.

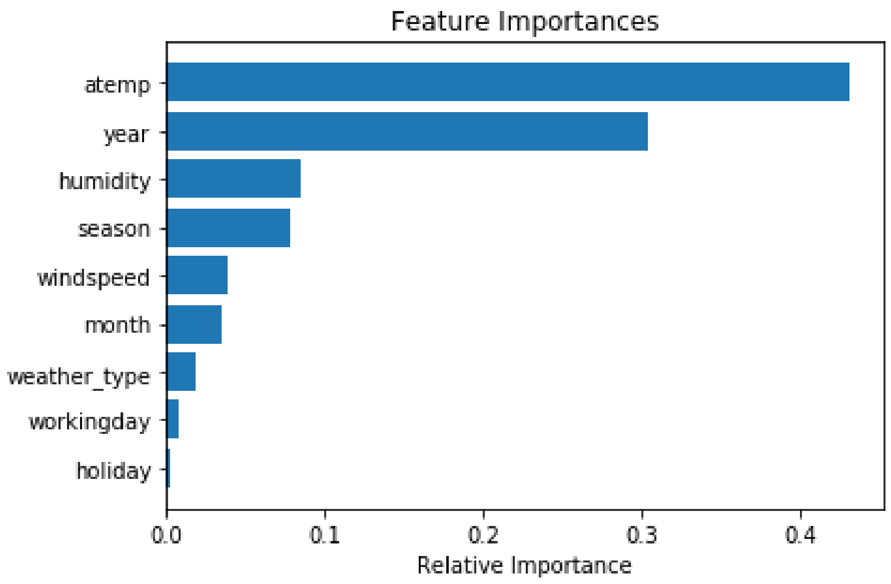


**Figure 10 – Random Forest Feature Importance**

3. Decision Tree: Decision tree model was fitted onto the training set and provided the following feature importance.

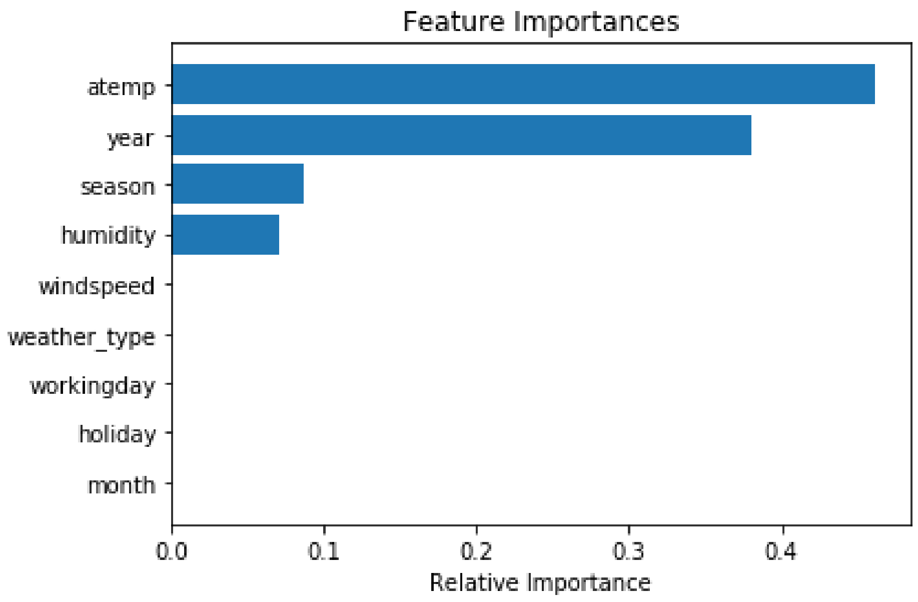


**Figure 11 – Decision Tree Feature Importance**

## Chapter 4

# Evaluation

We created various models against our dataset. Now comparing all of them with the standard performance matrices such as R Square, RMSE (Root Mean Squared Error) and MAPE (Mean Absolute Percentage Error).

1. Linear Regression Model:
    a. R Squared: 0.7763
    b. RMSE: 0.0910
    c. MAPE: 17.6219
2. Decision Tree Model:
    a. R Squared: 0.7770
    b. RMSE: 0.1046
    c. MAPE: 24.3580
3. Random Forest Model:
    a. R Squared: 0.9075
    b. RMSE: 0.0679
    c. MAPE: 14.7493
4. Gradient Boosting Model:
    a. R Squared: 0.9074
    b. RMSE: 0.0672
    c. MAPE: 13.4998

Based on the results above, we can conclude that the gradient boosting algorithm performed best taking into consideration all three performance matrices and accepted as solution for the problem statement.

## Python Code

```
#!/usr/bin/env python
# coding: utf-8


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from fancyimpute import KNN

#Sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression , RandomizedLasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import RFE
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
#from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import preprocessing

#Chi-square test
from scipy.stats import chi2_contingency
from statsmodels.stats.outliers_influence import variance_inflation_factor



bike_rental = pd.read_csv('day.csv', header=[0])



df = bike_rental.copy()



df.head().iloc[:,0:9]
df.head().iloc[:,9:]



df.shape



df.dtypes



df.drop(columns=['instant', 'dteday','casual','registered'], inplace=True, axis=1)
df = df.rename(columns={'dteday':'date', 'yr':'year', 'mnth':'month',
'weathersit' : 'weather_type', 'hum' : 'humidity', 'cnt':'total_count'})
df.dtypes



categorical_variable = ['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
'weather_type']
#df['date'] = pd.to_datetime(df['date'])
df[categorical_variable] = df[categorical_variable].apply(pd.Series.astype,
dtype='category')
# df['day'] = df['date'].apply(lambda x: x.day)
# df['day'] = df['day'].astype('category')
#categorical_variable.append('day')
numerical_varible = [var for var in df.columns if var not in categorical_variable and
var not in ['total_count']]



# col = df.columns.tolist()
# col.insert(2, col.pop(col.index('day')))
# df = df[col]
df.dtypes



df.isnull().sum()



fig = plt.figure()
fig.subplots_adjust(hspace=0.3, wspace=0.2)
fig.set_figheight(20)
fig.set_figwidth(15)
```

```python
for pos, i in enumerate(numerical_varible):
    ax = fig.add_subplot(4, 2, pos+1)
    ax = sns.distplot(df[i], rug=True, hist=True)
    ax.set_xlabel(i)
    ax.set_ylabel('Density')
    ax.set_title('KDE Plot for {}'.format(i))
fig.savefig('Numerical_DistPlot', bbox_inches='tight')
# ax1 = fig.add_subplot(2, 2, 1)
# ax1 = sns.distplot(df['atemp'], rug=True, hist=True)
# ax1.set_xlabel('ATemp')
# ax1.set_ylabel('Probability density ')
# ax1.set_title("KDE Plot for ATemp")

# ax1 = fig.add_subplot(2, 2, 2)
# ax1 = sns.distplot(df['temp'], rug=True, hist=True)
# ax1.set_xlabel('Temp')
# ax1.set_ylabel('Probability density ')
# ax1.set_title("KDE Plot for Temp")



fig = plt.figure()
fig.subplots_adjust(hspace=0.3, wspace=0.2)
fig.set_figheight(20)
fig.set_figwidth(15)

for pos, i in enumerate(numerical_varible):
    ax = fig.add_subplot(4, 2, pos+1)
    ax = sns.regplot(df[i],df['total_count'])
    ax.set_xlabel(i)
    ax.set_ylabel('total_count')
    ax.set_title('Reg Plot for {}'.format(i))
fig.savefig('Numerical_RegPlot', bbox_inches='tight')
# ax1 = fig.add_subplot(2, 2, 1)
# ax1 = sns.distplot(df['atemp'], rug=True, hist=True)
# ax1.set_xlabel('ATemp')
# ax1.set_ylabel('Probability density ')
# ax1.set_title("KDE Plot for ATemp")

# ax1 = fig.add_subplot(2, 2, 2)
# ax1 = sns.distplot(df['temp'], rug=True, hist=True)
# ax1.set_xlabel('Temp')
# ax1.set_ylabel('Probability density ')
# ax1.set_title("KDE Plot for Temp")
```

```
fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.2)
fig.set_figheight(20)
fig.set_figwidth(15)

for pos, i in enumerate(categorical_variable):
    ax = fig.add_subplot(4, 2, pos+1)
    ax = sns.countplot(df[i])
    ax.set_xlabel(i)
    ax.set_ylabel('Count')
    ax.set_title('Count Plot for {}'.format(i))
fig.savefig('Categorical_CountPlot', bbox_inches='tight')



fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.2)
fig.set_figheight(20)
fig.set_figwidth(15)

for pos, i in enumerate(categorical_variable):
    ax = fig.add_subplot(4, 2, pos+1)
    ax = sns.barplot(df[i], df['total_count'])
    ax.set_xlabel(i)
    ax.set_ylabel('Total_Count')
    ax.set_title('Bar Plot for {}'.format(i))
fig.savefig('Categorical_BarPlot', bbox_inches='tight')



fig = plt.figure()
fig.subplots_adjust(hspace=0.5, wspace=0.2)
fig.set_figheight(15)
fig.set_figwidth(15)

for pos,i in enumerate(numerical_varible):
    ax1 = fig.add_subplot(4, 2, pos+1)
    ax1 = sns.boxplot(df[i], data=df)
    #plt.savefig('Outliers_Visualization_{}'.format(pos))
    #plt.close()
fig.savefig('Outlier1', bbox_inches='tight')
```

```
for i in numerical_varible:
    q75, q25 = np.percentile(df.loc[:, i], [75, 25])
    iqr = q75 -q25
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)
    df.loc[df[i] < minimum,i] = np.nan
    df.loc[df[i] > maximum, i] = np.nan
print(df.isna().sum().sort_values(ascending=False))



# #date_value = df.drop(columns=['date'], axis=1)
# date_value = df['date']
# df = df.drop(columns=['date'],axis=1)



df = pd.DataFrame(KNN(k=3).fit_transform(df), columns=df.columns)
df[categorical_variable] = df[categorical_variable].apply(pd.Series.astype,
dtype='category')



# df['date'] = date_value



# col = df.columns.tolist()
# col.insert(2, col.pop(col.index('date')))
# df = df[col]



# df['total_count'] = df['casual'] + df['registered']



#Getting the Pearson Correlation Matrix
correlation_data = df.copy()
corr = correlation_data.corr()
#Masking to get the Lower Triangular Matrix and masking the Upper Triangular Matrix
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
#Color map for plot
cmap = sns.diverging_palette(220, 10, as_cmap=True)
```

```
f, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(corr, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.6, cbar_kws={"shrink": .6}, annot=True)
f.savefig('Correlation', bbox_inches='tight')



numerical_varible.append('total_count')



X = df[numerical_varible].assign(const = 1)
pd.Series([variance_inflation_factor(X.values,i) for i in
range(X.shape[1])],index=X.columns)


# Will Drop temperature as it is highly correlated with temp.
# Also dropping the registered and casual as they are already described by the
total_count variable.


fig = plt.figure()
fig,(ax1,ax2,ax3) = plt.subplots(ncols=3)
fig.subplots_adjust(hspace=0.2, wspace=0.4)
fig.set_size_inches(12, 5)
sns.regplot(x="atemp", y="total_count", data=df,ax=ax1)
sns.regplot(x="windspeed", y="total_count", data=df,ax=ax2)
sns.regplot(x="humidity", y="total_count", data=df,ax=ax3)
f.savefig('Correlation', bbox_inches='tight')



#Finding correlation between categorical variables
category = df.loc[:, categorical_variable]

#Factorinzing the columns
factors_paired = [(i,j) for i in category.columns.values for j in
category.columns.values]

chi2, p_values =[], []

for f in factors_paired:
    #print(f[0], f[1])
    if f[0] != f[1]:
        chitest = chi2_contingency(pd.crosstab(category[f[0]], category[f[1]]))
        chi2.append(chitest[0])
        p_values.append(chitest[1])
    else:      #for same factor pair
        chi2.append(0)
        p_values.append(0)
chi2 = np.array(p_values).reshape((7,7)) # shape it as a matrix
chi2 = pd.DataFrame(chi2, index = category.columns.values,
columns=category.columns.values)



chi2



df['total_count'] = (df['total_count']-
min(df['total_count']))/(max(df['total_count'])-min(df['total_count']))



## anova test
from scipy import stats
for i in categorical_variable:
    print(i)
    print(stats.f_oneway(df[i],df['total_count']))



import statsmodels.api as sm
from statsmodels.formula.api import ols

for i in categorical_variable:
    mod = ols('total_count ~ df[i]',
                    data=df).fit()

    aov_table = sm.stats.anova_lm(mod, typ=2)
    print ( aov_table)



fig = plt.figure()
fig.subplots_adjust(hspace=0.5, wspace=0.2)
fig.set_figheight(15)
fig.set_figwidth(15)

for pos, i in enumerate(categorical_variable):
    value = pd.DataFrame(df.groupby([i])['total_count'].mean())
    value.reset_index(inplace=True)
    ax = fig.add_subplot(4, 2, pos+1)
    value.plot(kind='line',ax=ax)
    ax.set_xlabel(i)  #X-axis label
    ax.set_ylabel('Mean Count of rented bikes') #Y-axis label
    ax.set_title("Mean count based on {}".format(i)) #Chart title
f.savefig('Mean VS Category', bbox_inches='tight')



df.columns



df = df.drop(['temp', 'weekday'], axis=1)



df2 = df.copy()



# scaler = preprocessing.MinMaxScaler(feature_range = (0,1))
# scaled_data = scaler.fit_transform(df[['total_count']])



# scaled_data = pd.DataFrame(scaled_data)
# scaled_data = scaled_data.rename(columns = {0 : 'total_count'})



#df2['total_count'] = (df2['total_count']-
min(df2['total_count']))/(max(df2['total_count'])-min(df2['total_count']))



# col = df2.columns.tolist()
# col.insert(len(col)-1, col.pop(col.index('total_count')))
# df2 = df2[col]



X = df2.iloc[:, 0:len(df2.columns.tolist())-1]
Y = df2.iloc[:,len(df2.columns.tolist())-1]



x_train, x_cv, y_train, y_cv = train_test_split(X, Y , test_size = 0.2,
random_state=1)
```

```python
# x_train  = x_train.drop(dropFeatures,axis=1)
# x_cv  = x_cv.drop(dropFeatures,axis=1)



gbm = GradientBoostingRegressor()

gbm.fit(x_train, y_train)
preds_gbm = gbm.predict(x_cv)

#R -square
score = gbm.score(x_cv, y_cv)
print ('GBM R square: %.4f' % score)

#RMSE
mse = mean_squared_error(preds_gbm, y_cv)
rmse = np.sqrt(mse)
print('Gradient Boosting RMSE: %.4f' % rmse)



features = df2.columns
importances = gbm.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices])
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.savefig('FI_gbm', bbox_inches='tight')
plt.show()



from sklearn.ensemble import RandomForestRegressor
rfModel = RandomForestRegressor(n_estimators=500)
#yLabelsLog = np.log1p(yLabels)
rfModel.fit(x_train, y_train)
preds_rf = rfModel.predict(x_cv)


#R -square
score = rfModel.score(x_cv, y_cv)
print ('Random Forest R square: %.4f' % score)
```

```python
#RMSE
mse = mean_squared_error(preds_rf, y_cv)
rmse = np.sqrt(mse)
print('Random Forest RMSE: %.4f' % rmse)



features = df2.columns
importances = rfModel.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices])
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.savefig('FI_rf', bbox_inches='tight')
plt.show()



dtmodel = DecisionTreeRegressor(max_depth=4,
                           min_samples_split=5,
                           max_leaf_nodes=10)

#yLabelsLog = np.log1p(yLabels)
#yLabelsLogtest = np.log1p(ytestLabels)

dtmodel.fit(x_train, y_train)

pred_dt = dtmodel.predict(x_cv)
dtmodel
#print ("RMSLE Value Decision tree",
rmsle(np.exp(yLabelsLogtest),np.exp(preds),False))

#print ("RMSLE Value For Decision Tree:
",rmsle(np.exp(yLabelsLogtest),np.exp(predictions),False))
#R -square
score = dtmodel.score(x_cv, y_cv)
print ('Decsion Tree R square: %.4f' % score)


#RMSE
mse = mean_squared_error(pred_dt, y_cv)
rmse = np.sqrt(mse)
print('Decision tree RMSE: %.4f' % rmse)
print(dict(zip(df2.columns, dtmodel.feature_importances_)))



features = df2.columns
importances = dtmodel.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices])
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.savefig('FI_df', bbox_inches='tight')
plt.show()



from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
from sklearn.feature_extraction import DictVectorizer



dot_data = StringIO()
export_graphviz(dtmodel, out_file=dot_data,filled=True,
rounded=True,special_characters=True,feature_names=x_train.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())



from sklearn.linear_model import LinearRegression
mlr = LinearRegression()
mlr.fit(x_train, y_train)
pred_mlr = mlr.predict(x_cv)
score = dtmodel.score(x_cv, y_cv)
print ('MLR square: %.4f' % score)
#RMSE
mse = mean_squared_error(pred_mlr, y_cv)
rmse = np.sqrt(mse)
print('MLR RMSE: %.4f' % rmse)



def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```
y_cv[y_cv == 0] = 0.1
mape = mean_absolute_percentage_error(y_cv.values, pred_dt)
print("Mean Absolute Percentage error for Decision Tree Model: %.4f" % mape)



mape = mean_absolute_percentage_error(y_cv.values, pred_mlr)
print("Mean Absolute Percentage error for Multiple Linear Regression Model: %.4f" %
mape)



mape = mean_absolute_percentage_error(y_cv.values, preds_rf)
print("Mean Absolute Percentage error for Random Forest Model: %.4f" % mape)


mape = mean_absolute_percentage_error(y_cv.values, preds_gbm)
print("Mean Absolute Percentage error for Gradient Boosting Model: %.4f" % mape)
```

## References

https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0

https://towardsdatascience.com/statistical-tests-when-to-use-which-704557554740

https://towardsdatascience.com/random-forest-in-python-24d0893d51c0

http://benalexkeen.com/gradient-boosting-in-python-using-scikit-learn/