Bike Rental Count Prediction

By

Ayushman Mishra

Contents

Chapters **Page No.**

1.Introduction

Whether it's to boost your fitness, health or bank balance, or as an environmental choice, taking up bicycle riding could be one of the best decisions you ever make. Remember the days of the bicycle built for two, when tourists rented bikes to explore island areas where cars either didn't exist or were blessedly limited? Those days are still here, but the majority of bicycle rental businesses are clustered around heavily trafficked tourist spots. However, with increased rails-to-trails projects and traffic congestion there are many more bicycle paths away from resort areas, office space, residential area that are creating excellent new rental opportunities. Many bicycle rental shops are now featuring inline skate rentals as well, especially in places like USA, UK. The bike rental service has a great potential as a business opportunity.

1.1 Problem Statement:

The objective of this case study is the prediction of bike rental count on daily based on the environmental and seasonal settings. The dataset contains 731 observations, 15 predictors and 1 target variable. The predictors are describing various environment factors and settings like season, humidity etc. We need to build a prediction model to predict estimated count or demand of bikes on a particular day based on the environmental factors.

1.2 Data:

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 from Holiday Schedule)

- **weekday:** Day of the week
- **workingday:** If day is neither weekend nor holiday is 1, otherwise is 0.
- **weathersit:** (extracted from Freemeteo)
  a) Clear, Few clouds, Partly cloudy, Partly cloudy
  b) Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  c) Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  d) 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

Here ***cnt*** is our target variable which equal to sum of ***casual*** and ***registered*** variable. ***Instant*** variable is just the index number. After reading the data while dropping ***instant*** column and making ***dteday*** as a new index column, our data will look like in a following manner:

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	0.344167	0.363625	0.805833	0.160446	331	654	985
2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600
2011-01-06	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.089565	88	1518	1606
2011-01-07	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
2011-01-08	1	0	1	0	6	0	2	0.165000	0.162254	0.535833	0.266804	68	891	959
2011-01-09	1	0	1	0	0	0	1	0.138333	0.116175	0.434167	0.361950	54	768	822
2011-01-10	1	0	1	0	1	1	1	0.150833	0.150888	0.482917	0.223267	41	1280	1321

Fig. 1.2: Data

2.Methodology

2.1 Pre-Processing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Before Outliers analysis our data will undergo under following process:

i. Type conversion of categorical variables

```
#Converting to categorical variables
categorical_variables = ['season','yr','mnth','holiday','weekday','workingday','weathersit']

df_data[categorical_variables] = df_data[categorical_variables].apply(lambda x: x.astype('category'))

df_data.info()
```

Fig.2.1.1 Type Conversion of categorical variables.

ii. Missing Value Analysis

```
# check for missing values
df_data.isnull().sum()
```

```
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
dtype: int64
```

Fig.2.1.2 Checking Missing Values

Visualize the Missing Values in each column

```
mn.matrix(df_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x23eb9c1cfd0>
```

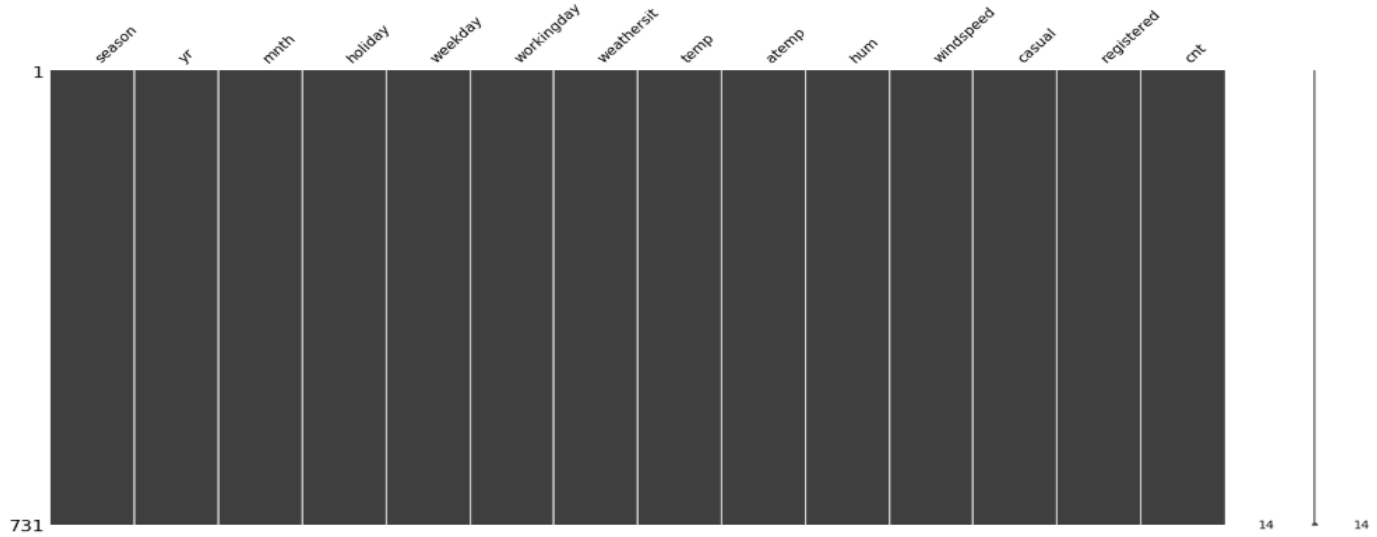


Fig.2.1.3 Visualization of Missing Values.

2.1.1 Data Visualization

First we check the Data Continuity of the Continuous Variables which includes our target variable “cnt” also.

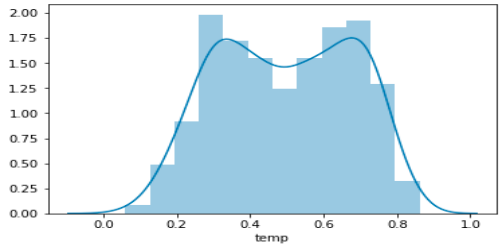
```
continuous_variables = ['temp','atemp','hum','windspeed','cnt']

#check the normal distribution of continuous variable
sns.distplot(df_data['temp'])
```

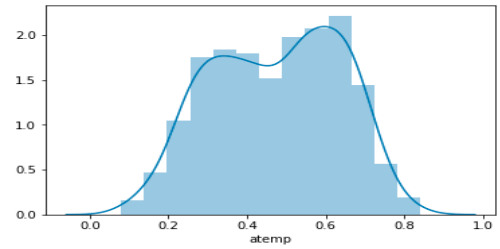
Fig.2.1.1.1 Checking Distribution of Continuous Variables.

Fig.2.1.1.2 shows the histograms of the normal distribution of all continuous variable and Fig.2.1.1.3 shows the relationship of all continuous variables with target variable.

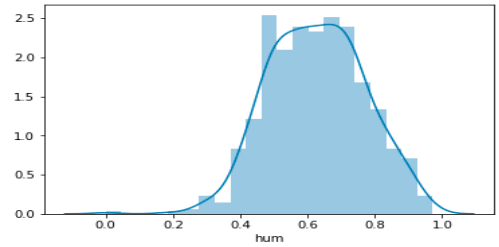
```
<matplotlib.axes._subplots.AxesSubplot at 0x23ebb53ccc0>
```



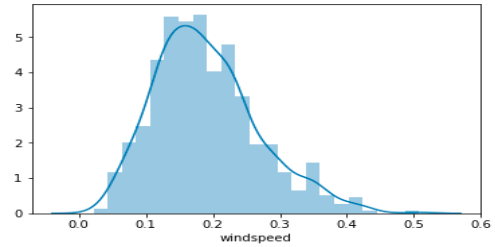

```
sns.distplot(df_data['atemp'])
<matplotlib.axes._subplots.AxesSubplot at 0x23eb7a0f470>
```




```
sns.distplot(df_data['hum'])
<matplotlib.axes._subplots.AxesSubplot at 0x23ebb4ec7f0>
```




```
sns.distplot(df_data['windspeed'])
<matplotlib.axes._subplots.AxesSubplot at 0x23eb9b18e80>
```




```
sns.distplot(df_data['cnt'])
<matplotlib.axes._subplots.AxesSubplot at 0x23ebc04f080>
```

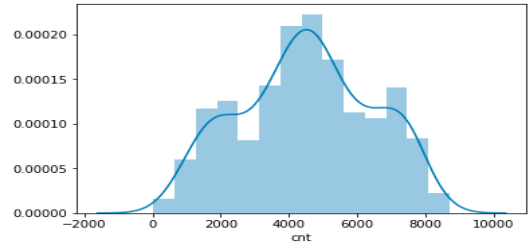
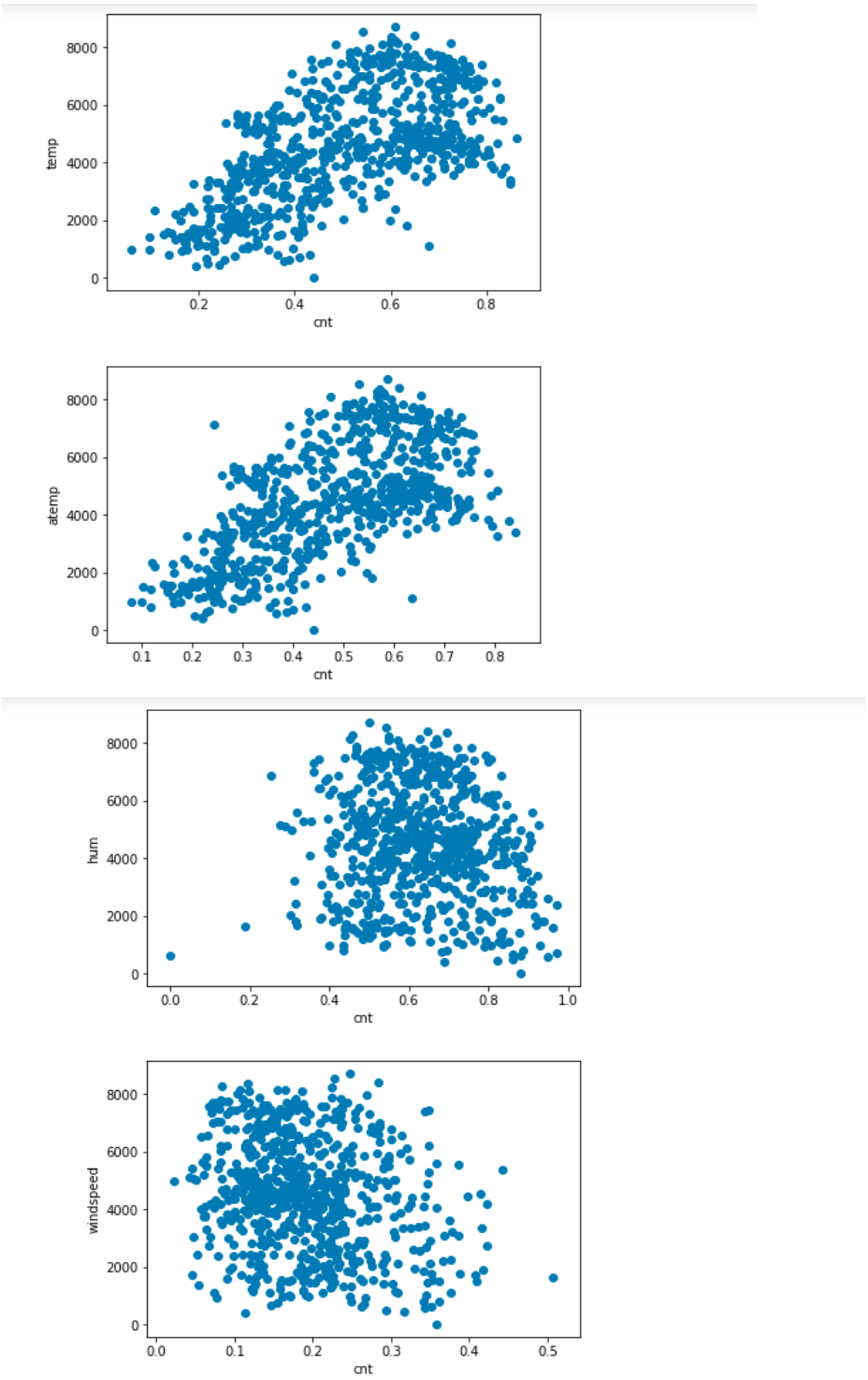


Fig. 2.1.1.2 Histograms of all continuous variables




```
#check Relationship of target variable with other continuous variables
for var in continuous_variables:
    plt.scatter(x=df_data[var],y=df_data['cnt'])
    plt.xlabel('cnt')
    plt.ylabel(var)
    plt.show()
```

Fig.2.1.1.3 Relationship b/w all continuous variables and target variable

2.1.2 Outlier Analysis

An ***outlier*** is an observation that lies an abnormal distance from other values in a random sample from a population. In a sense, this definition leaves it up to the analyst (or a consensus process) to decide what will be considered abnormal. Before abnormal observations can be singled out, it is necessary to characterize normal observations.

Now we plot boxplot for outlier's detection

```
#Boxplots of all continuous variables
for var in continuous_variables:
    sns.boxplot(y=var,data=df)
    plt.title('Boxplot of '+var)
    plt.show()
```

Fig.2.1.2.1 plotting Boxplot of all continuous variables

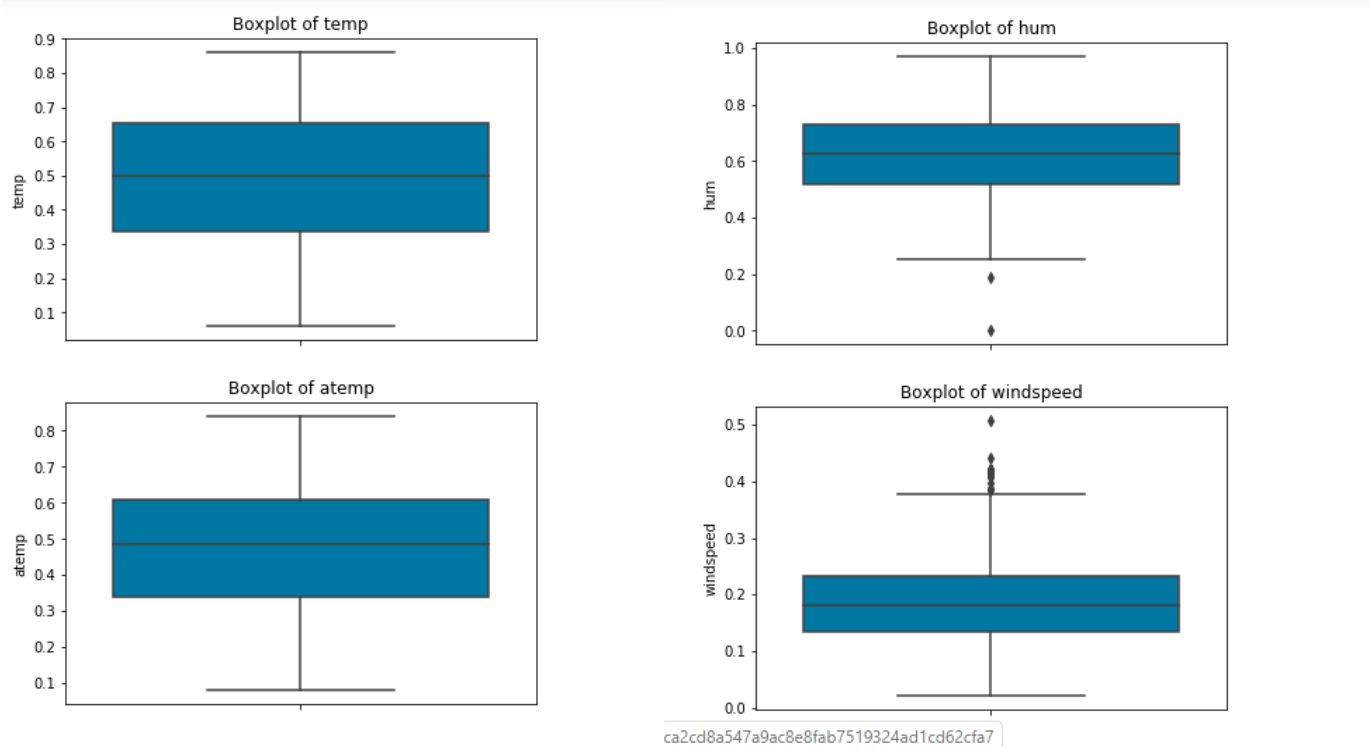


Fig.2.1.2.2 Boxplot of all continuous variables.

In above plots we can see that variables "windspeed" and "hum" have outliers.

Outlier's Removal

We use KNN Imputation to remove outliers.

```
#Dealing with outliers
names = ['windspeed','hum'] #columns names having outliers
for i in names:
    q75, q25 = np.percentile(df[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

#Calculate missing value
missing_val = pd.DataFrame(df.isnull().sum())
```

```
missing_val
```

	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	2
windspeed	13
cnt	0

```
#Impute with KNN
df = pd.DataFrame(KNN(k = 3).fit_transform(df), columns=df.columns,index=df_data.index)
```

```
Imputing row 1/731 with 0 missing, elapsed time: 0.108
Imputing row 101/731 with 0 missing, elapsed time: 0.109
Imputing row 201/731 with 0 missing, elapsed time: 0.110
Imputing row 301/731 with 0 missing, elapsed time: 0.110
Imputing row 401/731 with 0 missing, elapsed time: 0.111
Imputing row 501/731 with 0 missing, elapsed time: 0.111
Imputing row 601/731 with 0 missing, elapsed time: 0.111
Imputing row 701/731 with 0 missing, elapsed time: 0.112
```

Fig.2.1.2.3 Outlier Removal Using KNN-Imputation.

2.1.3 Feature Selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in.

Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

We perform two test for our feature selection process:

i. Plotting heat map to check correlation between continuous variables.
ii. Chi-square Test of Independence for categorical variables.

First we plot heat map to check correlation.

```
df_corr = df.loc[:,continuous_variables]
f, ax = plt.subplots(figsize=(10, 10))
corr = df_corr.corr()
sns.heatmap(data=corr,annot=True,square=True,linewidths=0.5,linecolor='w')
plt.title("correlation between all continuous variables \n")
plt.show()
```

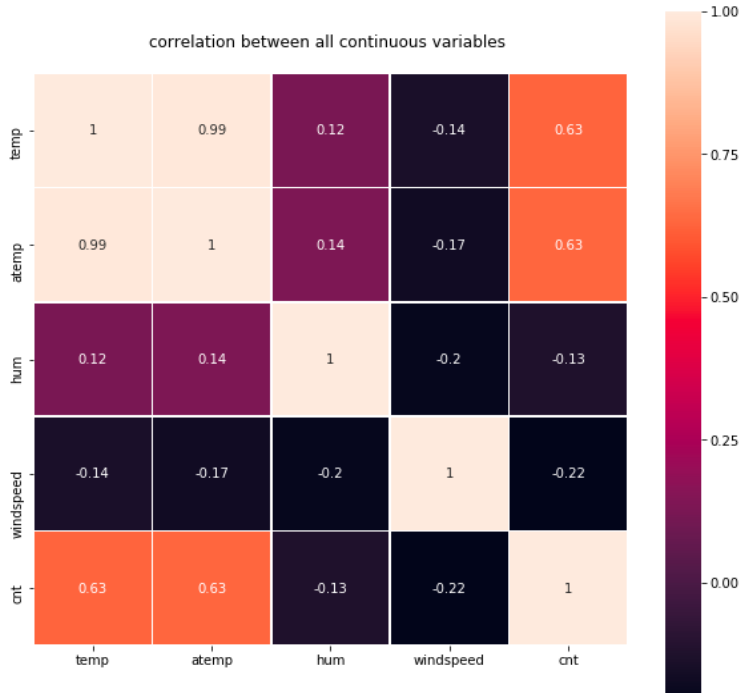


Fig.2.1.3.1 Heat Map for correlation.

Each square shows the correlation between the variables on each axis. Correlation ranges from -1 to +1. Values closer to zero means there is no linear trend between the two variables. The close

to 1 the correlation is the more positively correlated they are; that is as one increases so does the other and the closer to 1 the stronger this relationship is. A correlation closer to -1 is similar, but instead of both increasing one variable will decrease as the other increases. The number and darker the color the higher the correlation between the two variables. The plot is also symmetrical about the diagonal since the same two variables are being paired together in those squares.

Secondly, we perform chi-square test of independency.

The ***Chi Square*** statistic is commonly used for testing relationships between categorical variables. The null hypothesis of the Chi-Square test is that no relationship exists on the categorical variables in the population; they are independent.

```
dependent_variables = []
independent_variables = []
for i in categorical_variables:
    for j in categorical_variables:
        if(i == j):
            continue
        else:
            chi2, p, dof, ex = chi2_contingency(pd.crosstab(df[i], df[j]))
            if(p < 0.5):
                dependent_variables.append((i,j,p))
            else:
                independent_variables.append((i,j,p))
```

```
dependent_variables
```

```
[('season', 'mnth', 0.0),
 ('season', 'weathersit', 0.021179301044733697),
 ('yr', 'weathersit', 0.12737941480418666),
 ('mnth', 'season', 0.0),
 ('mnth', 'weathersit', 0.014637111771019196),
 ('holiday', 'weekday', 8.567055344615667e-11),
 ('holiday', 'workingday', 4.033370935452143e-11),
 ('weekday', 'holiday', 8.567055344615637e-11),
 ('weekday', 'workingday', 6.775030505809736e-136),
 ('weekday', 'weathersit', 0.2784593307450517),
 ('workingday', 'holiday', 4.033370935452143e-11),
 ('workingday', 'weekday', 6.775030505809736e-136),
 ('workingday', 'weathersit', 0.2537639982644043),
 ('weathersit', 'season', 0.02117930104473366),
 ('weathersit', 'yr', 0.12737941480418666),
 ('weathersit', 'mnth', 0.014637111771019196),
 ('weathersit', 'weekday', 0.27845933074505175),
 ('weathersit', 'workingday', 0.2537639982644043)]
```

Fig. 2.1.3.2 Chi-Square Test.

Hypothesis testing:-

(i) Null Hypothesis: 2 variables are independent.

(ii) Alternate Hypothesis: 2 variables are not independent.

- "p" is the probability the variables are independent.
- If p-value is less than 0.05 then we reject the null hypothesis saying that 2 variables are dependent.
- And if p-value is greater than 0.05 then we accept the null hypothesis saying that 2 variables are independent.

Conclusion:

Above tests leads to following Conclusion.

(I) season have high dependency with mnth and weathersit with p value ~ 0 so we will keep season and discard mnth and weathersit

(II) holiday have high dependency with weekday and workingday so we will keep holiday and discard other two

(II) yr have dependency with weathersit and we already discarded the weathersit so we will keep yr.

(IV) we can see that the variables 'temp' and 'atemp' are very highly correlated with each other, so we will keep only temp variable.

2.1.4 Feature Scaling

Data scaling methods are used when we want our variables in data to scale on common ground. It is performed only on continuous variables.

- **Normalization:** Normalization refer to the dividing of a vector by its length. Normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN.

Normalizing the data improves convergence of such algorithms. Normalization of data scales the data to a very small interval, where outliers can be loosed.

- **Standardization:** Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.

High variance will affect the accuracy of the model. So, we want to normalise that variance.

```
#quickly check the variance of continuos variables
vars = ['temp','hum','windspeed','cnt']
df[vars].var()
```

```
temp         3.350767e-02
hum          1.957466e-02
windspeed    5.146815e-03
cnt          3.752788e+06
dtype: float64
```

```
#Normalisation
for i in vars:
    print(i)
    df[i] = (df[i] - min(df[i]))/(max(df[i]) - min(df[i]))
```

```
temp
hum
windspeed
cnt
```

```
df.head()
```

dteday	temp	hum	windspeed	cnt	season_1.0	season_2.0	season_3.0	season_4.0	yr_0.0	yr_1.0	holiday_0.0	holiday_1.0
2011-01-01	0.355170	0.767981	0.388102	0.110792	1	0	0	0	1	0	1	0
2011-01-02	0.379232	0.615202	0.635752	0.089623	1	0	0	0	1	0	1	0
2011-01-03	0.171000	0.254904	0.635105	0.152669	1	0	0	0	1	0	1	0
2011-01-04	0.175530	0.468123	0.387681	0.177174	1	0	0	0	1	0	1	0
2011-01-05	0.209120	0.254464	0.462471	0.181546	1	0	0	0	1	0	1	0

Fig.2.1.4.1 Normalization of continuous variables.

2.2 Model Selection

Grid search

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified. Here for every algorithm we used Grid Search to get the best Parameters.

If you work with ML, you know what a nightmare it is to stipulate values for hyper parameters. There are libraries that have been implemented, such as GridSearchCV of the sklearn library, in order to automate this process and make life a little bit easier for ML enthusiasts.

2.2.1 Decision Tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

```
features =df.copy().drop(columns=['cnt'])
target = df.copy().drop(columns=['temp', 'hum', 'windspeed', 'season_1.0', 'season_2.0',
       'season_3.0', 'season_4.0', 'yr_0.0', 'yr_1.0', 'holiday_0.0',
       'holiday_1.0'])
d_tree = DecisionTreeRegressor()
# Applying Grid Search to find the best parameters
parameters = {"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_search = GridSearchCV(estimator=d_tree, param_grid=parameters,cv=5,n_jobs=-1)

grid_search.fit(features,target)
```

```
grid_search.best_params_
```

```
{'criterion': 'mse',
 'max_depth': 8,
 'max_leaf_nodes': 100,
 'min_samples_leaf': 20,
 'min_samples_split': 10}
```

```
tree = DecisionTreeRegressor(criterion='mse',max_depth=8,max_leaf_nodes=100,min_samples_leaf=10,min_samples_split=10)
```

```
tree.fit(train_feature_variables,train_target_variable)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=8,
                      max_features=None, max_leaf_nodes=100,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=10, min_samples_split=10,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

```
prediction = tree.predict(test_feature_variables)
prediction = pd.DataFrame(prediction)
prediction = prediction.set_index(test_target_variable.index).rename(columns = {0:'cnt'})
```

```
#calculate RMSE on test data
print("R2 Score:",r2_score(test_target_variable,prediction))
print("Mean Absolute Error:",mean_absolute_error(test_target_variable,prediction))
print("Mean Squared Error:",mean_squared_error(test_target_variable,prediction))
print("Root Mean Squared Error:",np.sqrt(mean_squared_error(test_target_variable,prediction)))
print("MAPE:",mean_absolute_percentage_error(test_target_variable,prediction))
print("Accuracy:",(100 - mean_absolute_percentage_error(test_target_variable,prediction)))
```

```
R2 Score: 0.8538822024169046
Mean Absolute Error: 0.06448892461923823
Mean Squared Error: 0.00797092699887646
Root Mean Squared Error: 0.08928004815677723
MAPE: cnt    21.600067
dtype: float64
Accuracy: cnt    78.399933
dtype: float64
```

Fig. 2.2.1.1 Decision Tree

2.2.2 Linear Regression

Linear regression is used for finding linear relationship between target and one or more predictors.

```
#Grid Search
parameters =  {'copy_X':[True, False],
          'fit_intercept':[True,False]}
model_grid = GridSearchCV(estimator=model,param_grid=parameters,cv=5,scoring='r2')
model_grid.fit(features,target)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='deprecated', n_jobs=None,
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

```
print("Best score:",model_grid.best_score_,'\n Best Parameters :',model_grid.best_params_)
```

```
Best score: 0.32099064163348145
 Best Parameters : {'copy_X': True, 'fit_intercept': False}
```

```
model = LinearRegression(copy_X= True, fit_intercept=False).fit(train_feature_variables,train_target_variable)
prediction = model.predict(test_feature_variables)
prediction = pd.DataFrame(prediction)
prediction = prediction.set_index(test_target_variable.index).rename(columns = {0:'cnt'})
```

```
#calculate RMSE on test data
print("R2 Score:",r2_score(test_target_variable,prediction))
print("Mean Absolute Error:",mean_absolute_error(test_target_variable,prediction))
print("Mean Squared Error:",mean_squared_error(test_target_variable,prediction))
print("Root Mean Squared Error:",np.sqrt(mean_squared_error(test_target_variable,prediction)))
print("MAPE:",mean_absolute_percentage_error(test_target_variable,prediction))
print("Accuracy:",(100 - mean_absolute_percentage_error(test_target_variable,prediction)))
```

```
R2 Score: 0.8191423791114653
Mean Absolute Error: 0.07372336877405293
Mean Squared Error: 0.009866032181830303
Root Mean Squared Error: 0.0993279023327801
MAPE: cnt    20.09239
```

Fig.2.2.2.1 Linear Search

2.2.3 Random Forest

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

```
# Create the random grid
parameters = {'n_estimators': list(range(100,700,100)),
              'max_depth': list(range(10,20,1)),
              'min_samples_split':range(2,5,1)}

Forest = RandomForestRegressor()

# Instantiate the gridSearchCV object: Forest
Forest_grid = GridSearchCV(estimator=Forest, param_grid=parameters, cv=5,n_jobs=-1)
Forest_grid.fit(features,target.values.ravel())
print("Best score:",Forest_grid.best_score_,'\n Best Parameters :',Forest_grid.best_params_)
```

```
Best score: 0.30408577443144286
 Best Parameters : {'max_depth': 11, 'min_samples_split': 4, 'n_estimators': 100}
```

```
Forest = RandomForestRegressor(max_depth= 11, min_samples_split=4,n_estimators=100)
Forest.fit(train_feature_variables,train_target_variable.values.ravel())
prediction = Forest.predict(test_feature_variables)
prediction = pd.DataFrame(prediction)
prediction = prediction.set_index(test_target_variable.index).rename(columns = {0:'cnt'})
```

```
#calculate RMSE on test data
print("R2 Score:",r2_score(test_target_variable,prediction))
print("Mean Absolute Error:",mean_absolute_error(test_target_variable,prediction))
print("Mean Squared Error:",mean_squared_error(test_target_variable,prediction))
print("Root Mean Squared Error:",np.sqrt(mean_squared_error(test_target_variable,prediction)))
print("MAPE:",mean_absolute_percentage_error(test_target_variable,prediction))
print("Accuracy:",(100 - mean_absolute_percentage_error(test_target_variable,prediction)))
```

```
R2 Score: 0.9091978063123916
Mean Absolute Error: 0.053799024998306445
Mean Squared Error: 0.004953384660825899
Root Mean Squared Error: 0.07038028602404156
MAPE: cnt    16.624133
dtype: float64
Accuracy: cnt    83.375867
dtype: float64
```

Fig.2.2.3.1 Random Forest

2.2.4 Gradient Boosting

A Gradient Boosting Machine or GBM combines the predictions from multiple decision trees to generate the final predictions. Keep in mind that all the weak learners in a gradient boosting machine are decision trees.

```
GB = GradientBoostingRegressor().fit(train_feature_variables, train_target_variable.values.ravel())
# predict on train data
prediction = GB.predict(test_feature_variables)
prediction = pd.DataFrame(prediction)
prediction = prediction.set_index(test_target_variable.index).rename(columns = {0:'cnt'})
```

```
#calculate RMSE on test data
print("R2 Score:",r2_score(test_target_variable,prediction))
print("Mean Absolute Error:",mean_absolute_error(test_target_variable,prediction))
print("Mean Squared Error:",mean_squared_error(test_target_variable,prediction))
print("Root Mean Squared Error:",np.sqrt(mean_squared_error(test_target_variable,prediction)))
print("MAPE:",mean_absolute_percentage_error(test_target_variable,prediction))
print("Accuracy:",(100 - mean_absolute_percentage_error(test_target_variable,prediction)))
```

```
R2 Score: 0.9097211561096751
Mean Absolute Error: 0.05159056187627023
Mean Squared Error: 0.004924835208958808
Root Mean Squared Error: 0.07017717014071463
MAPE: cnt    14.720545
dtype: float64
Accuracy: cnt    85.279455
dtype: float64
```

Fig.2.2.4.1 Gradient Boosting

3.Conclusion

3.1 Model Evaluation

The dependent variable in our model is a continuous variable i.e., Count of bike rentals. Hence the models that we choose are Linear Regression, Decision Tree, Random Forest and Gradient Boosting.

The Following error metrics are chosen chosen for the problem statement:

i. **Root Mean Squared Error:**
RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

ii. **Mean Absolute Error:**
MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

iii. R – Square:

R-squared (R^2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. Whereas correlation explains the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of one variable explains the variance of the second variable. So, if the R^2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs.

$$R^2 = 1 - \frac{\text{Unexplained Variation}}{\text{Total Variation}}$$

iv. Mean Absolute Percentage Error (MAPE):

The mean absolute percentage error (MAPE) is a statistical measure of how accurate a forecast system is. It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values. Where A_t is the actual value and F_t is the forecast value, this is given by:

$$M = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

The mean absolute percentage error (MAPE) is the most common measure used to forecast error, and works best if there are no extremes to the data (and no zeros).

v. Accuracy:

Accuracy = 1 – MAPE or 100 – MAPE (if it is in percentage).

3.2 Model Selection

We will select that model which has low MAPE or high accuracy and whose r-squared value is high.

In our case,

Gradient Boosting Model is providing best results having R-squared = 0.90 and Accuracy = 85.27%

Appendix – R Code

```
# loading dataset
df = read.csv('day.csv',header = TRUE,na.strings = c('',' ','NA'),row.names="dteday")[, -c(1)]
#dropping instatnt because it is unstatiscally
```

Fig.A.1: Reading data

```
# converting variables to their proper datatypes
categorical_variables = c("season","yr","mnth","holiday","weekday","workingday","weathersit")

for(i in categorical_variables){
  df[,i] = as.factor(df[,i])
}
```

Fig.A.2: Conversion of Categorical Variables Type

```
# check for missing values
apply(df,2, function(x){ sum(is.na(x))})
#we found 2 and 13 outliers in hum and windspeed respectively
```

Fig.A.3: Check for Missing Values.

```
#function to vizualize the continuity of continuous variables
continuos_vars_display <- function(cntnus){
  ggplot(df) +
    geom_histogram(aes(x = cntnus, y = ..density..),fill='green',colour='black') +
    geom_density(aes(x = cntnus, y = ..density..))

}
continuos_vars_display(df$temp)
continuos_vars_display(df$atemp)
continuos_vars_display(df$hum)
continuos_vars_display(df$windspeed)
continuos_vars_display(df$cnt)
```

Fig.A.4: Data Visualization

```
##Plotting Box Plots
for(i in 1:length(continuous_variables)){
  assign(paste0("gn",i), ggplot(data = df, aes_string(x = "cnt", y = continuous_variables[i]))+
           stat_boxplot(geom = "errorbar",width = 0.5)+
           geom_boxplot(outlier.colour = 'red',fill='grey',outlier.shape = 18,outlier.size = 4)+
           labs(y=continuous_variables[i],x='cnt')+
           ggtitle(paste("Box plot of",continuous_variables[i])))

}

gridExtra::grid.arrange(gn1,gn2,ncol=2)
gridExtra::grid.arrange(gn3,gn4,ncol=2)
```

Fig.A.5: Plotting BoxPlots for Outlier's detection.

```
#as we can see hum and windspeed has outliers and now we will remove them
#make a copy of original data
df_copy = df

outliers_vars = c('hum','windspeed')

for(i in outliers_vars){
  temp = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  df[,i][df[,i] %in% temp] = NA
}

# check for missing values
apply(df,2, function(x){ sum(is.na(x))})
#we found 2 and 13 outliers in hum and windspeed respectively

#outliers removal
df = knnImputation(data = df, k = 3)
```

Fig.A.6: Outlier's Removal using KNN-Imputation.

```
#check relationship b/w categorical and target variable
for(i in 1:length(categorical_variables)){
  assign(paste0("s",i),ggplot(data = df, aes_string(fill = "cnt", x = categorical_variables[i])) +
    geom_bar(position = "dodge") + ggtitle(paste("cnt vs ",categorical_variables[i])))
}
gridExtra::grid.arrange(s1,s2,s3,ncol=3)
gridExtra::grid.arrange(s4,s5,s6,ncol=3)
gridExtra::grid.arrange(s7,ncol=1)
```

Fig.A.7: Visualizing Relationship between all categorical variables and target variable.

```
###################### Feature Extraction ################################
# correlation plot
corrgram(df[,continuous_variables],order = FALSE,
         upper.panel=panel.pie, text.panel=panel.txt, main='Correlation Plot')
# as we can see temp and atemp are highly positively correlated with each other


## Chi-squared Test of Independence
for(i in categorical_variables){
  for(j in categorical_variables){
    print(i)
    print(j)
    print(chisq.test(table(df[,i], df[,j]))$p.value)
  }
}
# check VIF
vif(df[,8:11])
vifcor(df[,8:11])

# ANOVA test

for (i in categorical_variables) {
  print(i)
  print(summary(aov(df$cnt ~df[,i], df)))
}
```

Fig.A.8: Feature selection Tests i.e., correlation plot, chi-square test, VIF, ANOVA test.

```
#Taking correlation plot and vif Results into cosideration, Removing variables atemp beacuse it is highly correlated with temp,
#Taking Anova Test Results into consideration, Removing weekday, holiday because they don't contribute much to the independent cariable
## Dimension Reduction
df=subset(df,select=-c(atemp,holiday,weekday))
```

Fig.A.9: Dimension Reduction.

```
#Normality check for target variable
qqnorm(df$cnt); qqline(df$cnt)


# Normalization of cnt
df$cnt = (df$cnt - min(df$cnt)) / (max(df$cnt) - min(df$cnt))
#no changes has been seen in qqnorm before and after normalisation of cnt variable
```

Fig.A.10: Normalization.

```
############ Splitting df into train and test ###################
set.seed(12345)
train_index = createDataPartition(df$cnt, p = 0.8, list=FALSE)
train = df[train_index,]
test = df[-train_index,]
```

Fig.A.11: Splitting data into train, test.

```
# Linear Regression
linear_regressor = lm(cnt ~.,data = train)
summary(linear_regressor)

pred = predict(linear_regressor,test[,-27])

regr.eval(test[,27],preds = pred)
#       mae           mse          rmse          mape
# 0.069017507   0.009433239  0.097124863  0.206684169

#calculate R-Squared value
rsq(fitObj = linear_regressor,adj = TRUE,data = train)
# 0.8418282
```

Fig.A.12: Linear Regression Model.

```
############## Decision Tree Model ###############
tree = rpart(cnt ~ ., data=train, method = "anova")
summary(tree)

pred_dt = predict(tree, test[,-27])

regr.eval(test[,27],preds = pred_dt)
#       mae          mse          rmse          mape
# 0.07537033   0.01037191   0.10184259    0.21901332

# calculate R-Square value
rss_dt = sum((pred_dt - test$cnt) ^ 2)
tss_dt = sum((test$cnt - mean(test$cnt)) ^ 2)
rsq_dt = 1 - rss_dt/tss_dt
# 0.7605108
```

Fig.A.13: Decision Tree Model.

```
#############  Random forest Model ####################
rf_model = randomForest(cnt ~.,data=train, importance = TRUE, ntree=500)
summary(rf_model)

pred_rm = predict(rf_model,test[-27])

regr.eval(test[,27],preds = pred_rm)
#     mae             mse        rmse        mape
# 0.056957955  0.006789371  0.082397642 0.161405671

# calculate R-Square value
rss_rf = sum((pred_rm - test$cnt) ^ 2)
tss_rf = sum((test$cnt - mean(test$cnt)) ^ 2)
rsq_rf = 1 - rss_rf/tss_rf
# 0.8432323
```

Fig.A.14: Random Forest Model.

```
############  XGBOOST Model ##########################
train_data_matrix = as.matrix(sapply(train[-27],as.numeric))
test_data_matrix = as.matrix(sapply(test[-27],as.numeric))

xgb = xgboost(data = train_data_matrix,label = train$cnt, nrounds = 13,verbose = TRUE)

pred_xgb = predict(xgb,test_data_matrix)

regr.eval(test[,27],preds = pred_xgb)
#  mae           mse          rmse       mape
# 0.060146910 0.006860217 0.082826426 0.164038736


#calculate R-Square value
rss_xgb = sum((pred_xgb - test$cnt) ^ 2)
tss_xgb = sum((test$cnt - mean(test$cnt)) ^ 2)
rsq_xgb = 1 - rss_xgb/tss_xgb
#0.8415965


#########################################################################################################################################
# from the above models we can conclude that XGBOOST Model is the best fit for this problem.
```

Fig.A.15: XGboost Model.

References

1) https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/
2) https://www.r-bloggers.com/great-circle-distance-calculations-in-r/
3) https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/
4) http://benalexkeen.com/gradient-boosting-in-python-using-scikit-learn/
5) https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
6) https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74 .
7) https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f
8) https://campus.datacamp.com/courses/supervised-learning-with-scikit-learn/fine-tuning-your-model?ex=10.
9) https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
10) https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

Thank You