DATA MINING FINAL TERM PROJECT (CS-634)

OPTION1
SUPERVISED DATA MINING (CLASSIFICATION)

Professor
DR. JASON WANG, NJIT

Sai Sarvani Ippagunta
UCID: si89
email: si89@njit.edu

OPTION1:

To implement two algorithms (Random Forest and Decision Tree) on same dataset and compare their accuracy.

PREDICTION OF BIKE RENTALS – DECISION TREES AND RANDOM FOREST

Bike Sharing Demand. Below is the URL of dataset:

https://www.kaggle.com/c/bike-sharing-demand

Description of the Data Problem:

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. In this competition, participants are asked to combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

This dataset was provided by Hadi Fanaee Tork using data from Capital Bikeshare.

GOAL:

To predict the total count of bikes rented(class label) during each hour covered by the test set from 20th to the end of the month using the training set from 1st to 19th of the same month with both Random Forest and Decision Tree algorithms and compare their accuracies.

Description of dataset:

We are provided with hourly rental data spanning two years of bikes.
The training set is comprised of the first 19 days of each month of rental data,
while the test set is the 20th to the end of the month.

Total number of instances in Training Dataset: 10887
Total number of instances in Test Dataset: 6494

Class Labels/ Dependent variables:

casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

Features/predictors for the dataset:

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed.

EXPLORATORY DATA ANALYSIS AND DATA PREPARATION:

Exploring the dataset and understanding training dataset using initial analysis with box plots.
Screen shots are below in order –

Exploring the structure of the data :

In [3]:
```
# Exploring dataset and analyzing data
import pandas as pd
import numpy as np
train_dataframe = pd.read_csv('train.csv')
```

In [4]:
```
train_dataframe.head(2)
```

Out[4]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40

1. Creating a new features in dataset for splitting up datetime like year, month, dayofweek and hour. As they might be interesting features to predict count on.

In [5]:
```
###Defining function to create new features to be used in exploration ###########
def new_date_features(dataframe: pd)-> None:
    '''Creating new features :
        Creating new columns on datetime timestamp
    New Features are:
    year
    month
    day_of_week
    hour
    :param df: train dataset
        '''
train_dataframe['year'] = train_dataframe.datetime.map(
    lambda x: pd.to_datetime(x).year ).astype(int)
train_dataframe['month'] = train_dataframe.datetime.map(
    lambda x: pd.to_datetime(x).month ).astype(int)
train_dataframe['dayofweek'] = train_dataframe.datetime.map(
    lambda x: pd.to_datetime(x).dayofweek ).astype(int)
train_dataframe['hour'] = train_dataframe.datetime.map(
    lambda x: pd.to_datetime(x).hour ).astype(int)
```

In [6]:
```
#new dataframe with new features for dates
new_date_features(train_dataframe)
```

Checking the structure of the data to confirm after adding the additional features

In [7]:
```
train_dataframe.head(2)
```

Out[7]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	dayofweek	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011	1	5	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011	1	5	1

We can see the four additional columns added to the data frame for further analysis.

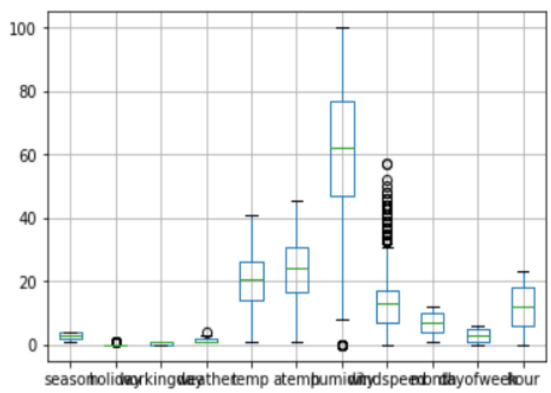
These additional columns are added to closely monitor the correlation between label and them .

2. Plotting various box plots to understand the count under various conditions.

1. Box Plot for all the features

In [8]:
```
##########################################################################################################
#Analyzing data sets using box plots
##########################################################################################################
train_boxplot_df= train_dataframe.drop(['year','casual','registered','count'],axis=1)
```

In [10]:
```
#plotting the graph
import matplotlib as plt
plot1 = train_boxplot_df.boxplot()
```



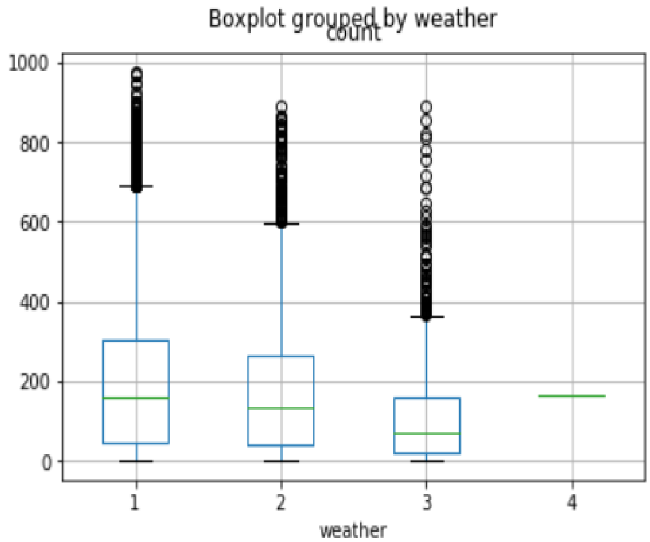
Observation :

Season, holiday, workingday,weather,temp,atemp,humidity,windspeed,month,dayofweek and hour are the predictors for which count is plotted

2. Box Plot for count vs weather :

```
In [11]: ## correlation between count vs weather data
         train_dataframe.boxplot(column='count', by='weather')
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1ba2fc0fc18>
```

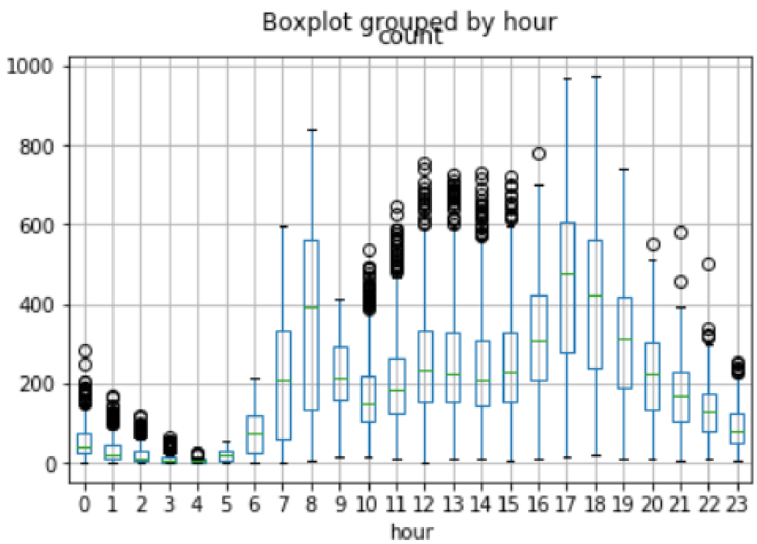


Observation: we can see most of the counts aboce 700 upto 1000, max at 1000 approx, min-0, median – 180 and outliers from the box plot. For season 1

3. Box plot grouped by hour vs count

```
In [12]: #count based on hour analysis
         train_dataframe.boxplot(column='count', by='hour')
```

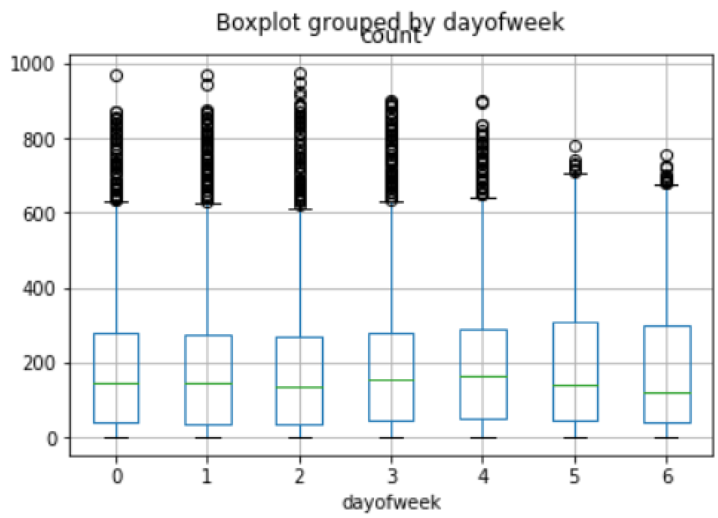
```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1ba2fcc3898>
```



4. Boxplots for count vs day of week

In [13]:
```
#count based on days of week  analysis
train_dataframe.boxplot(column='count',by='dayofweek')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1ba30eefa58>



Mapping name of day of week to the respective name of week

In [13]:
```
#mapping days of week to name of week
days={
    0:'Sunday',
    1:'Monday',
    2:'Tuesday',
    3:'Wednesday',
    4:'Thursday',
    5:'Friday',
    6:'Satruday'
}
```

In [14]:
```
train_dataframe['namedayofweek']= train_dataframe['dayofweek'].map(days)
```

Observations from boxplots :
Count vs hour : bike rentals are high during the peak hours from 7 am to 7 pm .

Count vs dayofweek : bike rentals are almost similar on all days . Median bike rentals are around 180 on every day of week. Outliers are found from 600 to 1000.

3. CORRELATION MATRIX :

Correlation Matrix determines which features that are positively correlated, negatively correlated and highly correlated. This helps us to identify important features in predicting class label and identify duplicate features (positively correlated), so that these one of the feature can be removed from the model.

In [15]:
```
##########################################################################################################
#Correlation matrix
#Matrix for us to know the correlation between various factors/attributes/columns in predicting target label i.e count
#Also to identify positive or negative correlation between them and label
#WE could remove correlated features
##########################################################################################################
print(train_dataframe[['weather','temp','atemp','humidity','windspeed','casual','registered','count']].corr())
```

```
             weather      temp     atemp  humidity  windspeed    casual  \
weather     1.000000 -0.055035 -0.055376  0.406244   0.007261 -0.135918
temp       -0.055035  1.000000  0.984948 -0.064949  -0.017852  0.467097
atemp      -0.055376  0.984948  1.000000 -0.043536  -0.057473  0.462067
humidity    0.406244 -0.064949 -0.043536  1.000000  -0.318607 -0.348187
windspeed   0.007261 -0.017852 -0.057473 -0.318607   1.000000  0.092276
casual     -0.135918  0.467097  0.462067 -0.348187   0.092276  1.000000
registered -0.109340  0.318571  0.314635 -0.265458   0.091052  0.497250
count      -0.128655  0.394454  0.389784 -0.317371   0.101369  0.690414

            registered     count
weather      -0.109340 -0.128655
temp          0.318571  0.394454
atemp         0.314635  0.389784
humidity     -0.265458 -0.317371
windspeed     0.091052  0.101369
casual        0.497250  0.690414
registered    1.000000  0.970948
count         0.970948  1.000000
```

Observation:

- Temp is positively correlated to count
- Temp and atemp are highly correlated , atemp can be eliminated from the dataframe for further analysis since it doest make any change in the correlation . Only one of them can be considered.
- Humidity and count are negatively correlated , so we can assume that more the humid the weather is , the less bike rentals are taken.
- Windspeed is also negatively correlated, hence we may assume that more windspeed less number of rentals.

5. Preparing Test data set:

Creating new features for test dataset that are created for training dataset.

In [17]:
```
##########################################################################################################
#Preparing the test data with the new features to test our model
test_df = pd.read_csv('test.csv',header=0)
test_df.head(1)
```

Out[17]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027

```
In [18]: # function call to create additional columns like train data
         new_date_features1(test_df)
         test_df.head(5)
```

Out[18]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	dayofweek	hour
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027	2011	1	3	0
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	3	1
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	3	2
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	3	3
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	3	4

```
In [19]: test_data_timestamp=test_df['datetime'] #column to store timestamps of test data
         test_df=test_df.drop(['datetime'],axis=1) # dropping the unnecessary column- timestamp
         print("test_df= ", list(test_df))

         test_df=  ['season', 'holiday', 'workingday', 'weather', 'temp', 'atemp', 'humidity', 'windspeed', 'year', 'month', 'dayofwee
         k', 'hour']
```

Now the test data set has also the same predictors/independent variables to send to the ML classifier to predict the labels.

Training your Model:

Training model with Random Forest and Decision tree algorithms on training dataset.

- RandomForestClassifier module is being used from sklearn.ensemble in python with 120 trees. n_estimators are selected with trail and error method.
- DecisionTreeClassifier module is being used from sklearn.tree in python.

Importing Libraries:

```
In [*]: ##########################################################################################################
        #Testing the model with classifiers - Random Forest and Decision Tree
        #Random forest is used from the sklearn.ensemble in python with 120 trees
        #####n_estimators are selected with trail and error method
        #Decision Tree has been used from sklearn.tree in python
        ## To predict 'count' label
        from sklearn.preprocessing import LabelEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
        from sklearn.tree import DecisionTreeClassifier #Import Decision Tree Classifier
        from sklearn.ensemble import RandomForestClassifier #Import Random Forest Classifier
```

Training the classifiers with training data_set

```
In [21]: target_dataframe = train_dataframe['count'] #y
         predictors_dataframe_train = train_dataframe.drop(['datetime','casual','registered','count','namedayofweek'],axis=1)

         #printing predictors
         predictors= list(predictors_dataframe_train)

         #numpy array to give an input to the decision tree or random forest classifier
         target_dataframe= target_dataframe.values
         predictors_dataframe_train=predictors_dataframe_train.values
         predictors_dataframe_test = test_df.values

         ##########################################################################################################
         #Random Forest Classifier
         ##########################################################################################################
         random_forest = RandomForestClassifier(n_estimators = 120)
         #fit data set in random forest
         random_forest = random_forest.fit (predictors_dataframe_train,target_dataframe)

         ##########################################################################################################
         #Decision Tree Classifier
         ##########################################################################################################
         forest_decisiontree = DecisionTreeClassifier()
         forest_decisiontree = forest_decisiontree.fit(predictors_dataframe_train,target_dataframe)
```

Predicting the count to test data set and saving it to a file:

Once the model is trained, applying the model on test data set to predict the count with both the algorithms and pass those values to two corresponding predictor files and saving it.

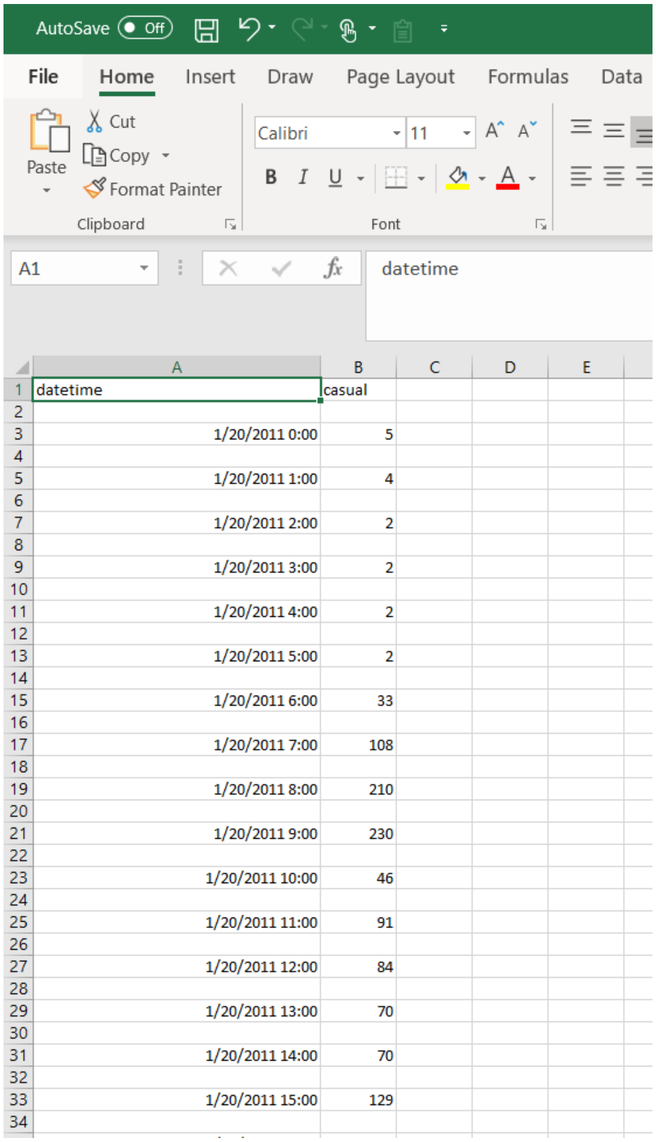
In [23]:
```
#########################################################################################################
#Predicting the count and saving it into a file
import csv
count_pred_random_forest = random_forest.predict(predictors_dataframe_test).astype(int)
count_pred_decision_tree = forest_decisiontree.predict(predictors_dataframe_test).astype(int)

def write_to_file(
            filename,
            test_data_timestamp,
            predicted_dataframe
):
        with open(filename, 'w') as fobj:
            fobj_write = csv.writer(fobj)
            fobj_write.writerow(["datetime","casual"])
            fobj_write.writerows(zip(test_data_timestamp, predicted_dataframe))

write_to_file(
        "RANDOM_FOREST_FILE.csv",
        test_data_timestamp,
        count_pred_random_forest,
)

write_to_file(
        "DECISION_TREE_FILE.csv",
        test_data_timestamp,
        count_pred_decision_tree,
)
```

Printing important features extracted from the training data using Randomforest:

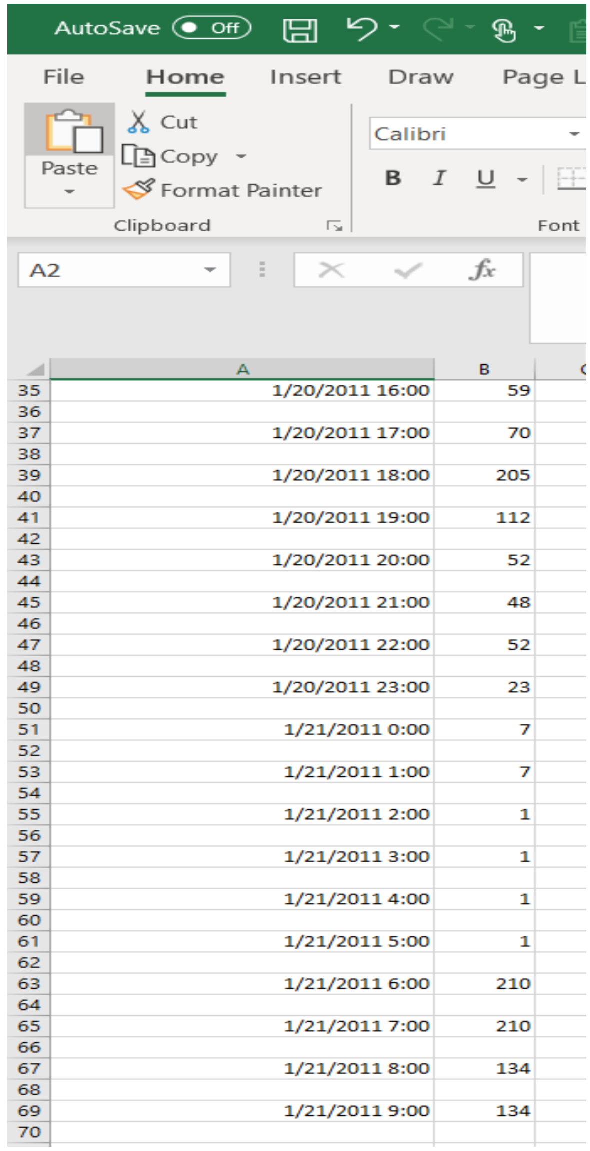
In [24]:
```
#################################################################################
#Printing important features using random forest
#########################################################################
print(pd.DataFrame(
                random_forest.feature_importances_,
                columns =['Importance'],
                index = predictors).sort_values(['Importance'],ascending =False))
```

```
            Importance
hour          0.187659
windspeed     0.154606
humidity      0.152476
atemp         0.120639
temp          0.118234
dayofweek     0.085297
month         0.072773
weather       0.043506
season        0.024766
year          0.021290
workingday    0.014407
holiday       0.004347
```

Radom Forest Count predicted: Total records with two columns . A sample of the random forest predicted values data set is shown below :

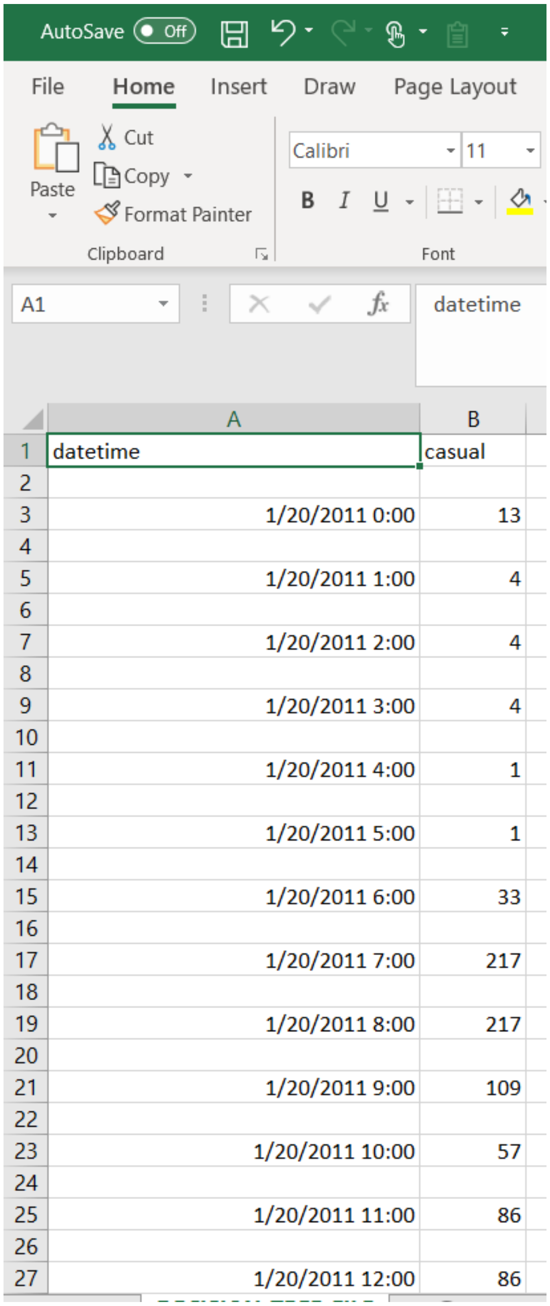


	A	B
1	datetime	casual
2		
3	1/20/2011 0:00	5
4		
5	1/20/2011 1:00	4
6		
7	1/20/2011 2:00	2
8		
9	1/20/2011 3:00	2
10		
11	1/20/2011 4:00	2
12		
13	1/20/2011 5:00	2
14		
15	1/20/2011 6:00	33
16		
17	1/20/2011 7:00	108
18		
19	1/20/2011 8:00	210
20		
21	1/20/2011 9:00	230
22		
23	1/20/2011 10:00	46
24		
25	1/20/2011 11:00	91
26		
27	1/20/2011 12:00	84
28		
29	1/20/2011 13:00	70
30		
31	1/20/2011 14:00	70
32		
33	1/20/2011 15:00	129
34		

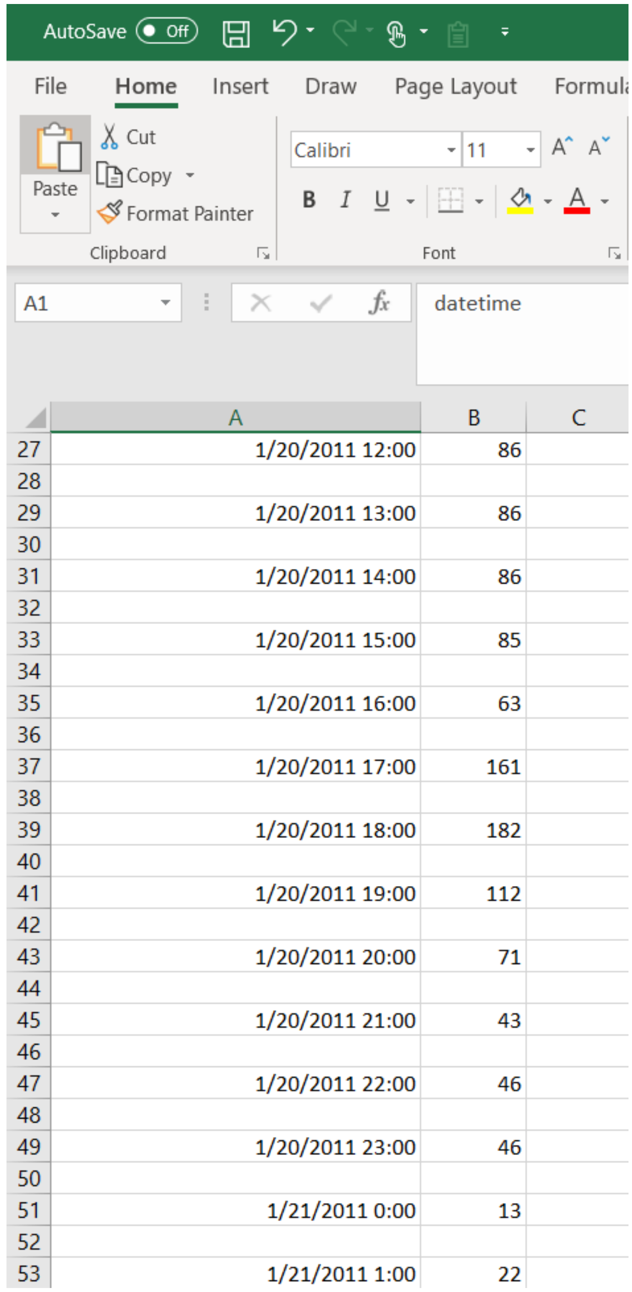


	A	B
35	1/20/2011 16:00	59
36		
37	1/20/2011 17:00	70
38		
39	1/20/2011 18:00	205
40		
41	1/20/2011 19:00	112
42		
43	1/20/2011 20:00	52
44		
45	1/20/2011 21:00	48
46		
47	1/20/2011 22:00	52
48		
49	1/20/2011 23:00	23
50		
51	1/21/2011 0:00	7
52		
53	1/21/2011 1:00	7
54		
55	1/21/2011 2:00	1
56		
57	1/21/2011 3:00	1
58		
59	1/21/2011 4:00	1
60		
61	1/21/2011 5:00	1
62		
63	1/21/2011 6:00	210
64		
65	1/21/2011 7:00	210
66		
67	1/21/2011 8:00	134
68		
69	1/21/2011 9:00	134
70		

Decision Tree Count predicted:



	A	B
1	datetime	casual
2		
3	1/20/2011 0:00	13
4		
5	1/20/2011 1:00	4
6		
7	1/20/2011 2:00	4
8		
9	1/20/2011 3:00	4
10		
11	1/20/2011 4:00	1
12		
13	1/20/2011 5:00	1
14		
15	1/20/2011 6:00	33
16		
17	1/20/2011 7:00	217
18		
19	1/20/2011 8:00	217
20		
21	1/20/2011 9:00	109
22		
23	1/20/2011 10:00	57
24		
25	1/20/2011 11:00	86
26		
27	1/20/2011 12:00	86



	A	B	C
27	1/20/2011 12:00	86	
28			
29	1/20/2011 13:00	86	
30			
31	1/20/2011 14:00	86	
32			
33	1/20/2011 15:00	85	
34			
35	1/20/2011 16:00	63	
36			
37	1/20/2011 17:00	161	
38			
39	1/20/2011 18:00	182	
40			
41	1/20/2011 19:00	112	
42			
43	1/20/2011 20:00	71	
44			
45	1/20/2011 21:00	43	
46			
47	1/20/2011 22:00	46	
48			
49	1/20/2011 23:00	46	
50			
51	1/21/2011 0:00	13	
52			
53	1/21/2011 1:00	22	

Total 6493 counts have been predicted. Full data set could be found in the github link in the summary at the ending.

Comparison of Random Forest and Decision Trees using Confusion Matrix (validation sets):

Below snippet of code creates validation dataset on splitting training into 80% train and 20% validation sets. Now, train your model using new train datasets and apply it on validation dataset. Then obtain a confusion matrix for both Random Forest and Decision Tree algorithms.

In [37]:
```
#Preparing Training Set
training_set_target_value = train["count"]
training_set = train.drop(['datetime','casual','registered','count','namedayofweek'],axis=1)
```

In [38]:
```
#preparing validation dataset
validation_count = validation["count"]
validation_set=  validation.drop(['datetime','casual','registered','count','namedayofweek'],axis=1)
```

In [39]:
```
#training your model on training datset
#numpy array to give input to random forest or decision tree classifiers
training_set_target_value = training_set_target_value.values
training_set = training_set.values
validation_set = validation_set.values
```

In [40]:
```
#Random Forest Classifier
random_forest = RandomForestClassifier(n_estimators=120)
#fit dataset in random forest
random_forest = random_forest.fit(training_set, training_set_target_value)
```

In [41]:
```
# Decision Tree Classifier
forest_decisiontree = DecisionTreeClassifier()
forest_decisiontree = forest_decisiontree.fit(training_set, training_set_target_value)
```

In [42]:
```
#predicting count and saving it to file
validation_pred_random_forest = random_forest.predict(validation_set).astype(int)
validation_pred_decision_tree = forest_decisiontree.predict(validation_set).astype(int)
```

Saving the predicted counts from validation data set

In [44]:
```
write_to_file(
        "Validate_RANDOM_FOREST_FILE.csv",
        vaidate_data_timestamp,
        validation_pred_random_forest,
)

write_to_file(
        "Validate_DECISION_TREE_FILE.csv",
        vaidate_data_timestamp,
        validation_pred_decision_tree,
)
```

And obtaining the confusion matrix for predicted counts and actual counts of the validaton data sets

Confusion matrix output for RandomForest:

```
conf_mat_RF = confusion_matrix(validation_count, validation_pred_random_forest)
print(conf_mat_RF)
```

```
[[8 5 5 ... 0 0 0]
 [5 4 3 ... 0 0 0]
 [3 9 4 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Confusion matrix output for Decision Tree:

```
In [52]: conf_mat_DT = confusion_matrix(validation_count, validation_pred_decision_tree)
         print(conf_mat_DT)
```

```
[[7 6 2 ... 0 0 0]
 [3 2 4 ... 0 0 0]
 [8 5 6 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Conclusions:

Applied both RandomForest and Decision Tree classifiers on bike share dataset to predict count. Used validation technique to compare both the models. From the result we have seen Random forest gave us more information compared to decision trees like identifying important features in a dataset.

Tools used : Jupyternotebooks, MS word and MS Excel.
Programming Language and libraries : Python 3.6 with Anaconda Package – Numpy, pandas, matplotlib and scikitlearn.

References:
Data Set and other references :
1. https://www.kaggle.com/c/bike-sharing-demand/data
2. Stackoverflow
3.Python documentation for numpy and pandas.
Source Code for ML algorithms:
Random Forest : sklearn.ensemble
https://github.com/scikit-learn/scikit-learn/blob/7b136e9/sklearn/ensemble/forest.py#L753
Decision Tree : sklearn.tree
https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/tree/tree.py