**Python code:**

In [1]:

**import** **datetime**

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **matplotlib.pyplot** **as** **plt**

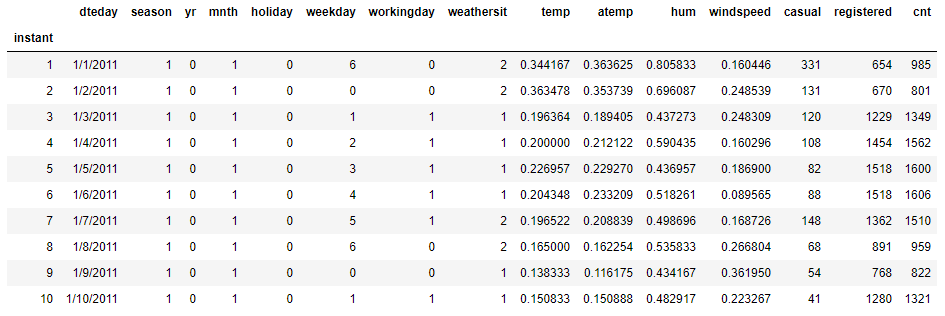
**import** **seaborn** **as** **sns**

**from** **random** **import** randrange, uniform

In [2]: df = pd.read\_csv("Day.csv",index\_col = 0)

In [3]: df.head(10)

Out[3]:



In [4]:

df.columns

Out[4]:

Index(['dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday',

'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt'],dtype='object')

**Exploratory Data Analysis**

In [5]: df.dtypes

Out[5]:

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

In [6]:

*#Converting variables datatype to required datatypes*

*#Categorical variables*

df['dteday'] = pd.to\_datetime(df['dteday'],yearfirst = **True**)

df['season'] = df['season'].astype('category')

df['yr'] = df['yr'].astype('category')

df['mnth'] = df['mnth'].astype('category')

df['holiday']= df['holiday'].astype('category')

df['weekday']= df['weekday'].astype('category')

df['workingday']= df['workingday'].astype('category')

df['weathersit']= df['weathersit'].astype('category')

*#Continuous variables*

df['temp'] = df['temp'].astype('float')

df['atemp']= df['atemp'].astype('float')

df['hum'] = df['hum'].astype('float')

df['windspeed'] = df['windspeed'].astype('float')

df['casual'] = df['casual'].astype('float')

df['registered'] = df['registered'].astype('float')

df['cnt'] = df['cnt'].astype('float')

In [7]: df.dtypes

Out[7]:

dteday datetime64[ns]

season category

yr category

mnth category

holiday category

weekday category

workingday category

weathersit category

temp float64

atemp float64

hum float64

windspeed float64

casual float64

registered float64

cnt float64

dtype: object

In [8]: ordered\_data = df.copy()

**Missing Value Analysis**

In [9]:

missing\_val = pd.DataFrame(df.isnull().sum())

missing\_val = missing\_val.reset\_index()

missing\_val = missing\_val.rename(columns={'index':'variable',0:'Missing\_values'})

In [10]:

missing\_val

Out[10]:

****

**\*\*Distribution of Data by Visualizations**

In [11]:

*#Creating new variables from existing variables for visualizations (Feature Engineering)*

df['actual\_temp'] = df['temp'] \* 39

df['actual\_atemp'] = df['atemp'] \* 50

df['actual\_windspeed'] = df['windspeed'] \* 67

df['actual\_hum'] = df['hum'] \* 100

In [12]:

df.columns

Out[12]:

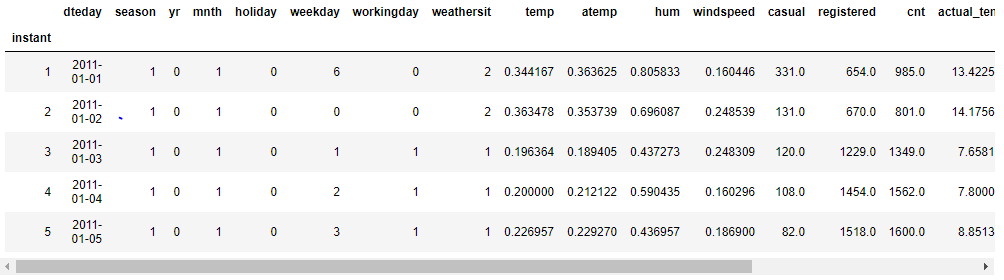
Index(['dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday',

'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual',

'registered', 'cnt', 'actual\_temp', 'actual\_atemp', 'actual\_windspeed', 'actual\_hum'], dtype='object')

In [13]: df.head()

Out[13]:



In [14]:

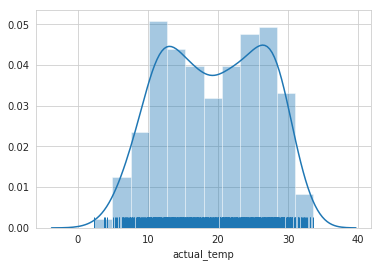
*#Cheking the Distribution of data using Histograms*

sns.set\_style("whitegrid")

sns.distplot(df['actual\_temp'],rug=**True**)

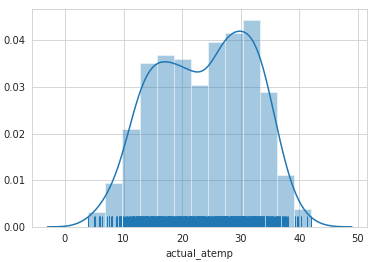
Out[14]

:



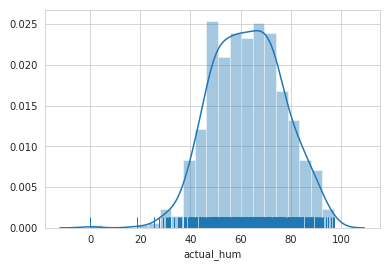
In [15]: sns.distplot(df['actual\_atemp'], rug=**True**)

Out[15]:



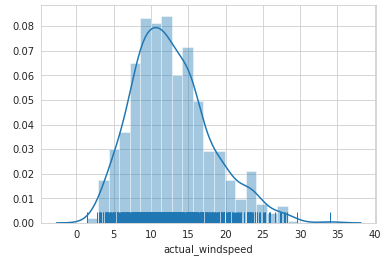
In [16]: sns.distplot(df['actual\_hum'], rug=**True**)

Out[16]:



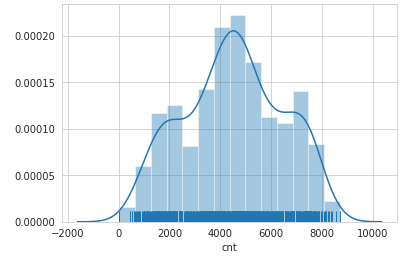
In [17]: sns.distplot(df['actual\_windspeed'],rug=**True**)

Out[17]:



In [18]: sns.distplot(df['cnt'],rug=**True**)

Out[18]:



In [19]:

continuous\_variables = ['actual\_temp','actual\_atemp','actual\_windspeed','actual\_hum','cnt']

**for** i **in** continuous\_variables:

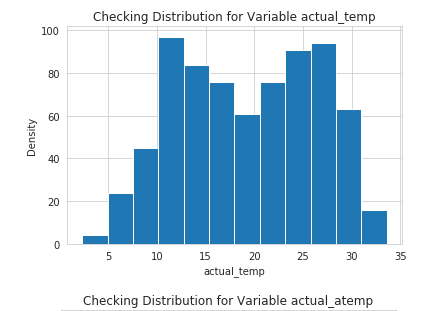
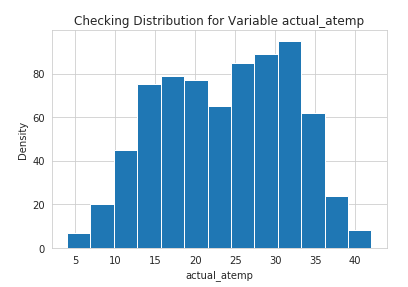
plt.hist(df[i],bins='auto')

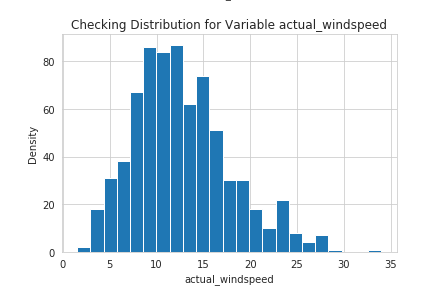
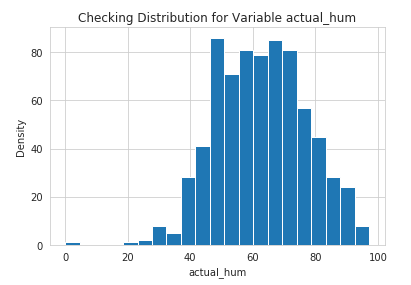
plt.title("Checking Distribution for Variable "+str(i))

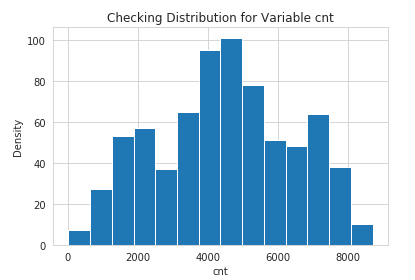
plt.ylabel("Density")

plt.xlabel(i)

plt.show()



In [20]: *#Bike Rentals Per Monthly*

monthly\_sales = df.groupby('mnth').size()

print(monthly\_sales)

*#Plotting the Graph*

plot = monthly\_sales.plot(title='Monthly Sales',xticks=(1,2,3,4,5,6,7,8,9,10,11,12))

plot.set\_xlabel('Months')

plot.set\_ylabel('Total Number of Bikes')

mnth

1 62

2 57

3 62

4 60

5 62

6 60

7 62

8 62

9 60

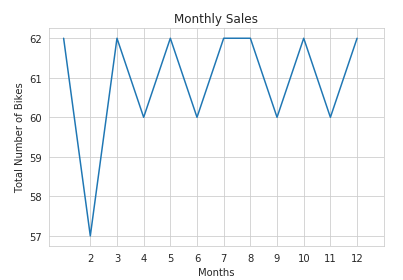
10 62

11 60

12 62

dtype: int64

Out[20]:



In [21]:

*#Checking the distribution categorical Data using factorplot*

sns.set\_style("whitegrid")

sns.factorplot(data=df, x='dteday', kind= 'count', size=4,aspect=2)

sns.factorplot(data=df, x='yr', kind= 'count', size=4,aspect=2)

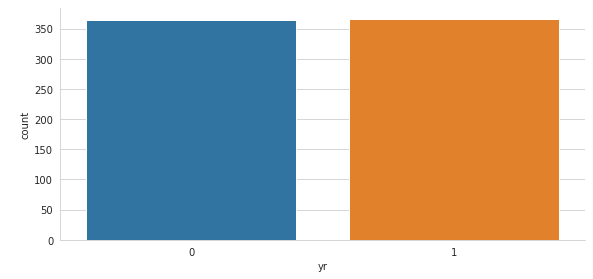
sns.factorplot(data=df, x='mnth', kind= 'count', size=4,aspect=2)

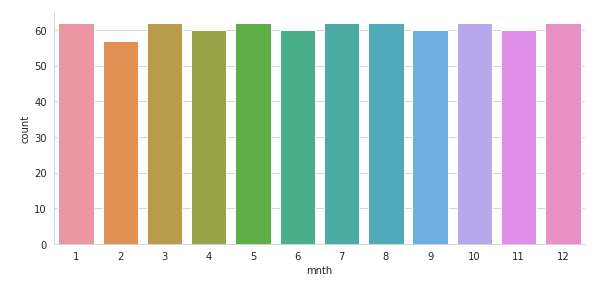
sns.factorplot(data=df, x='season', kind= 'count', size=4,aspect=2)

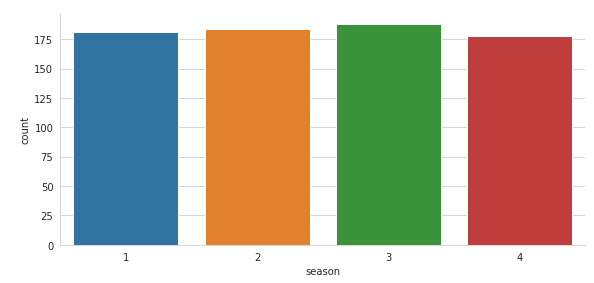
sns.factorplot(data=df, x='weathersit', kind= 'count', size=4,aspect=2)

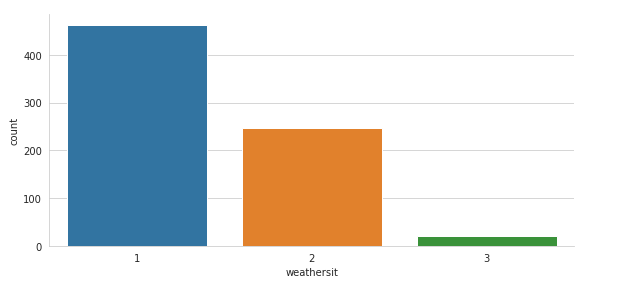
sns.factorplot(data=df, x='workingday', kind= 'count', size=4,aspect=2)

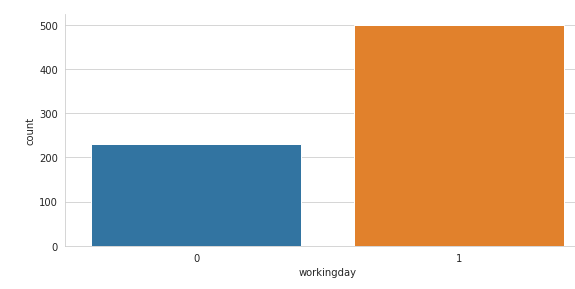
Out[21]:







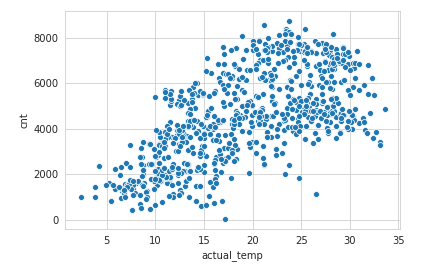




In [22]:*#Scatter plot for temprature against bike rentals*

sns.scatterplot(data=df,x='actual\_temp', y='cnt')

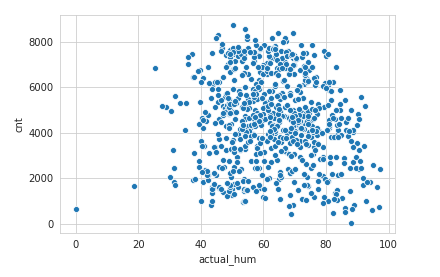
Out[22]:



In [23]:*#Scatter plot for humidity against bike rentals*

sns.scatterplot(data=df,x='actual\_hum',y='cnt')

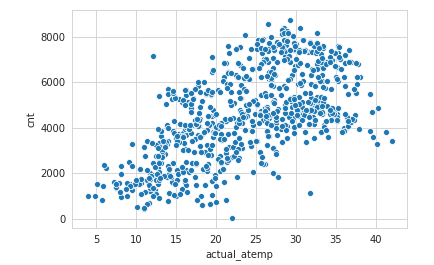
Out[23]:



In [24]:*#Scatter plot for atemp(feeled\_temparature) against bike rentals*

sns.scatterplot(data=df,x='actual\_atemp',y='cnt')

Out[24]:

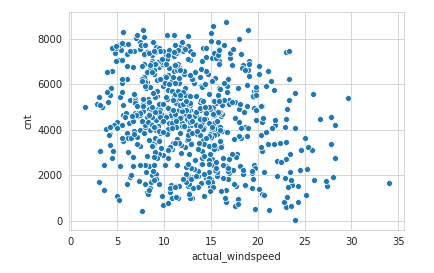


In [25]:

*#Scatter plot for windspeed against bike rentals*

sns.scatterplot(data=df,x='actual\_windspeed',y='cnt')

Out[25]:



**Outlier Analysis**

In [26]: df.columns

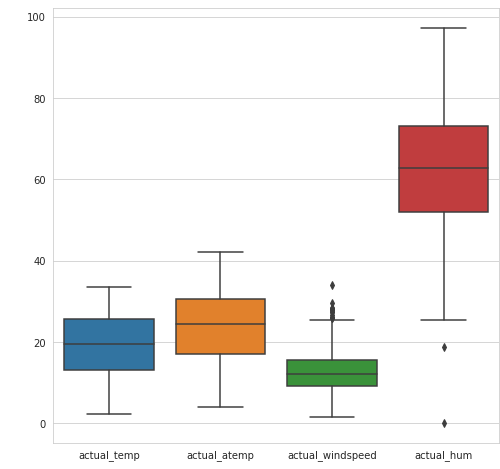
Out[26]: Index(['dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt', 'actual\_temp', 'actual\_atemp', 'actual\_windspeed', 'actual\_hum'],dtype='object')

In [27]: *#Checking Outliers in data using boxplot*

sns.boxplot(data=df[['actual\_temp','actual\_atemp','actual\_windspeed','actual\_hum']])

fig=plt.gcf()

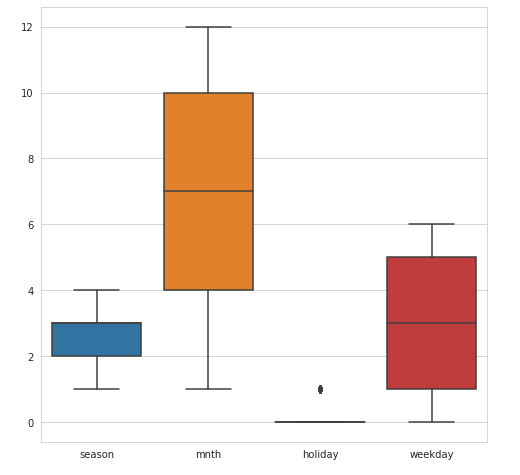
fig.set\_size\_inches(8,8)



In [28]: sns.boxplot(data=df[['season','mnth','holiday','weekday']])

fig=plt.gcf()

fig.set\_size\_inches(8,8)

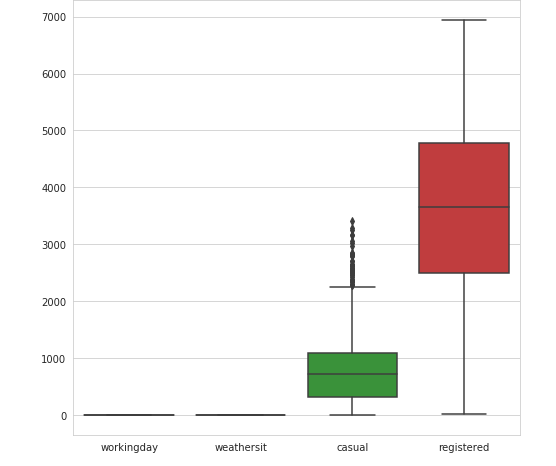


In [29]:

sns.boxplot(data=df[['workingday','weathersit','casual','registered']])

fig=plt.gcf()

fig.set\_size\_inches(8,8)



In [30]: *#Variables that are used to remove outliers*

*#Not considering casual because this is not predictor variable*

*#Not considering holiday because workingday variable includes holiday, so there is no usefulness of considering holiday variables.*

out\_names = ['actual\_windspeed', 'actual\_hum']

In [31]:*#Detecting and Removing Outliers*

**for** i **in** out\_names :

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)

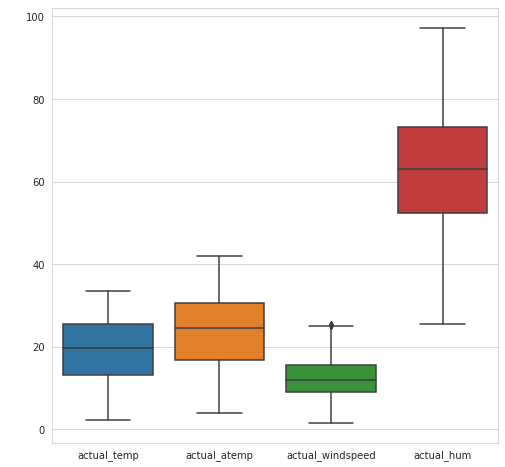
**Feature Selection**

In [32]: *#Checking Outliers in data after outliers removal using boxplot*

sns.boxplot(data=df[['actual\_temp','actual\_atemp','actual\_windspeed','actual\_hum']])

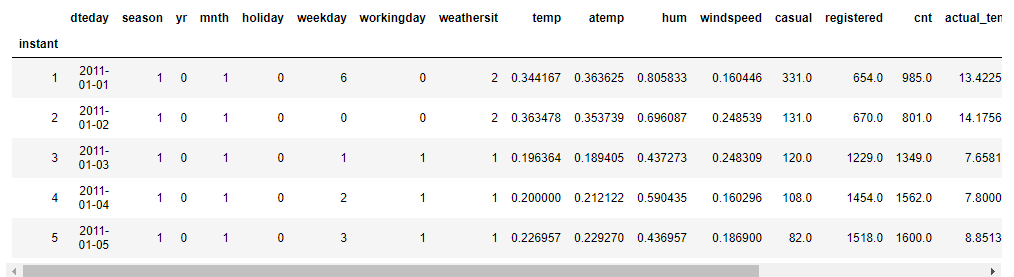
fig=plt.gcf()

fig.set\_size\_inches(8,8)



In [33]: df.head()

Out[33]:



In [34]: continuous\_variables = [ 'temp','atemp', 'hum', 'windspeed', 'casual','registered', 'cnt', 'actual\_temp', 'actual\_atemp', 'actual\_windspeed', 'actual\_hum']

In [35]: *#Feature selection on the basis of Correlation, multcollinearity and variable importance*

*#cnames = ["actual\_temp","actual\_atemp","actual\_hum","acttual\_windspeed"]*

*#cnames = ["temp","atemp","hum","windspeed"]*

df\_cor = df.loc[:,continuous\_variables]

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

*#Generate correlation matrix*

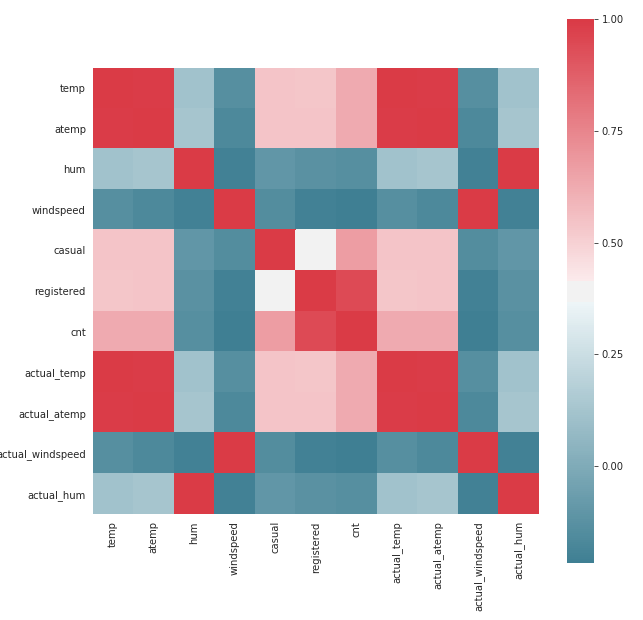
cor\_mat = df\_cor.corr()

*#Plot using seaborn library*

sns.heatmap(cor\_mat, mask=np.zeros\_like(cor\_mat, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=**True**),square=**True**, ax=ax)

plt.plot()

Out[35]:



**Hypothesis Testing**

**Null Hypothesis**  
Two variables are independent

**Alternate Hypothesis**  
Two variables are not independant\*\*

If p-value is less than 0.05 then reject null hypothesis, that means two variables are dependant but in our case most of the p-value are greater than 0.05,hence we need to accept that we failed to reject null hypothesis.

In [36]:

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

*# making every combination from cat\_columns*

factors\_paired = [(i,j) **for** i **in** cat\_columns **for** j **in** cat\_columns]

factors\_paired

p\_values = []

**from** **scipy.stats** **import** chi2\_contingency

**for** factor **in** factors\_paired:

**if** factor[0] != factor[1]:

chi2, p, dof, ex = chi2\_contingency(pd.crosstab(df[factor[0]], df[factor[1]]))

p\_values.append(p.round(3))

**else**:

p\_values.append('-')

p\_values = np.array(p\_values).reshape((7,7))

p\_values = pd.DataFrame(p\_values, index=cat\_columns, columns=cat\_columns)

print(p\_values)

**season yr mnth holiday weekday workingday weathersit**

**season - 0.999 0.0 0.641 1.0 0.946 0.013**

**yr 0.999 - 1.0 0.995 1.0 0.956 0.183**

**mnth 0.0 1.0 - 0.571 1.0 0.993 0.01**

**holiday 0.641 0.995 0.571 - 0.0 0.0 0.599**

**weekday 1.0 1.0 1.0 0.0 - 0.0 0.249**

**workingday 0.946 0.956 0.993 0.0 0.0 - 0.294**

**weathersit 0.013 0.183 0.01 0.599 0.249 0.294 -**

In [37]:

*# checking vif of numerical column without dropping multicollinear column*

**from** **statsmodels.stats.outliers\_influence** **import** variance\_inflation\_factor **as** vf

**from** **statsmodels.tools.tools** **import** add\_constant

continuous = add\_constant(df[['temp', 'atemp', 'hum', 'windspeed']])

vif = pd.Series([vf(continuous.values, i) **for** i **in** range(continuous.shape[1])], index = continuous.columns)

print(vif.round(1))

*# Checking VIF values of numeric columns after dropping column atemp*

**from** **statsmodels.stats.outliers\_influence** **import** variance\_inflation\_factor **as** vf

**from** **statsmodels.tools.tools** **import** add\_constant

continuous = add\_constant(df[['temp', 'hum', 'windspeed']])

vif = pd.Series([vf(continuous.values, i) **for** i **in** range(continuous.shape[1])], index = continuous.columns)

vif.round(1)

**const 46.4**

**temp 63.3**

**atemp 63.9**

**hum 1.1**

**windspeed 1.1**

**dtype: float64**

Out[37]:

**const 41.6**

**temp 1.0**

**hum 1.1**

**windspeed 1.1**

**dtype: float64**

In [38]:

*#Removing variables atemp beacuse it is highly correlated with temp,*

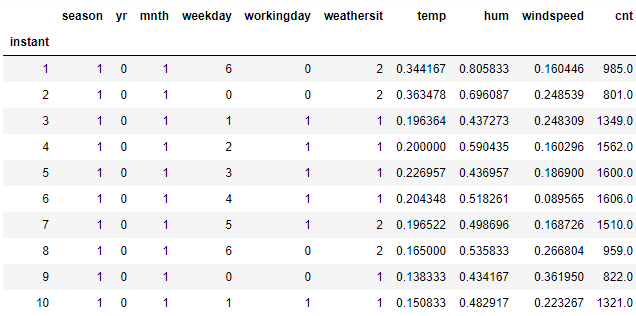
*#Removing weekday, holiday because they don't contribute much in predicting target variable*

*#Removing Causal and registered because that is what we need to predict.*

df = df.drop(columns=['holiday','dteday','atemp','casual','registered','actual\_temp','actual\_atemp','actual\_windspeed', 'actual\_hum'])

In [39]: df.head(10)

Out[39]:



In [40]:

df.columns

Out[40]:

Index(['season', 'yr', 'mnth', 'weekday', 'workingday', 'weathersit', 'temp',

'hum', 'windspeed', 'cnt'] dtype='object')

In [41]: df2 = df.copy()

In [42]: categorical\_var = ['season', 'yr', 'mnth', 'weekday', 'workingday', 'weathersit']

In [43]: df2.columns

Out[43]:

Index(['season', 'yr', 'mnth', 'weekday', 'workingday', 'weathersit', 'temp', 'hum', 'windspeed', 'cnt'], dtype='object')

In [44]: *#Dummy Variable creation for categorical variables*

df2 = pd.get\_dummies(data = df2,columns=categorical\_var)

In [45]:

df2.columns

Out[45]:

Index(['temp', 'hum', 'windspeed', 'cnt', 'season\_1', 'season\_2', 'season\_3',

'season\_4', 'yr\_0', 'yr\_1', 'mnth\_1', 'mnth\_2', 'mnth\_3', 'mnth\_4',

'mnth\_5', 'mnth\_6', 'mnth\_7', 'mnth\_8', 'mnth\_9', 'mnth\_10','mnth\_11',

'mnth\_12', 'weekday\_0', 'weekday\_1', 'weekday\_2', 'weekday\_3',

'weekday\_4', 'weekday\_5', 'weekday\_6', 'workingday\_0', 'workingday\_1',

'weathersit\_1', 'weathersit\_2', 'weathersit\_3'], dtype='object')

In [46]:

df2['count'] = df2['cnt']

df2 = df2.drop('cnt',axis=1)

df2.columns

Out[46]:

Index(['temp', 'hum', 'windspeed', 'season\_1', 'season\_2', 'season\_3',

'season\_4', 'yr\_0', 'yr\_1', 'mnth\_1', 'mnth\_2', 'mnth\_3', 'mnth\_4',

'mnth\_5', 'mnth\_6', 'mnth\_7', 'mnth\_8', 'mnth\_9', 'mnth\_10','mnth\_11',

'mnth\_12', 'weekday\_0', 'weekday\_1', 'weekday\_2', 'weekday\_3',

'weekday\_4', 'weekday\_5', 'weekday\_6', 'workingday\_0', 'workingday\_1',

'weathersit\_1', 'weathersit\_2', 'weathersit\_3', 'count' dtype='object')

In [47]:

df\_plt\_tree = df2.drop('count', axis=1)

df2.shape

Out[47]: (717, 34)

**Model Development**

In [48]: *#Import Libraries for decision tree*

**from** **sklearn.tree** **import** DecisionTreeRegressor

**from** **sklearn.metrics** **import** accuracy\_score,r2\_score,mean\_squared\_error

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** tree

In [49]:

*#Splitting data into train and test data*

train,test = train\_test\_split(df2,test\_size = 0.2, random\_state = 123)

In [50]:

*#Function for Performing all the tasks such as Error metrix rmse,mape,r-squared,accuracy,predictions*

**def** evaluate(model, test\_features, test\_actual):

predictions = model.predict(test\_features)

*#Creating new data frame for comparing actual and predicted values*

df\_Dt = pd.DataFrame({'actual':test\_actual,'predicted':predictions})

errors = abs(predictions - test\_actual)

mape = 100 \* np.mean(errors / test\_actual)

Accuracy = 100 - mape

rmse = np.sqrt(mean\_squared\_error(test\_actual,predictions))

rsquared = r2\_score(test\_actual, predictions)

print('<---Model Performance--->')

print('R-Squared Value = **{:0.2f}**'.format(rsquared))

print('RMSE = **{:0.2f}**'.format(rmse))

print('MAPE = **{:0.2f}**'.format(mape))

print('Accuracy = **{:0.2f}**%.'.format(accuracy))

**return**

**Decision Tree**

In [51]:

*#Decision Tree model development*

*#Training the model with train data*

model = DecisionTreeRegressor(random\_state = 123).fit(train.iloc[:,0:33],train.iloc[:,33])

*#Function for predictions, Error metrix rmse,mape,r-squared,accuracy*

evaluate(model, test.iloc[:,0:33], test.iloc[:,33])

**<---Model Performance--->**

**R-Squared Value = 0.73**

**RMSE = 962.42**

**MAPE = 17.75**

**Accuracy = 82.25%.**

**Linear Regression**

In [52]:

*#import libraries for Linear regression*

**from** **sklearn.linear\_model** **import** LinearRegression

*#Create model Linear Regression using LinearRegression*

model = LinearRegression().fit(train.iloc[:,0:33],train.iloc[:,33])

*#Function for predictions, Error metrix rmse,mape,r-squared,accuracy*

evaluate(model, test.iloc[:,0:33], test.iloc[:,33])

**<---Model Performance--->**

**R-Squared Value = 0.78**

**RMSE = 872.05**

**MAPE = 17.18**

**Accuracy = 82.82%.**

**Random forest**

In [53]:

*#Import the libraries for Random Forest*

**from** **sklearn.ensemble** **import** RandomForestRegressor

*#Train the model*

Rf\_model = RandomForestRegressor(n\_estimators=500,random\_state=123).fit(train.iloc[:,0:33], train.iloc[:,33])

*#Function for predictions, Error metrix rmse,mape,r-squared,accuracy*

evaluate(Rf\_model, test.iloc[:,0:33], test.iloc[:,33])

**<---Model Performance--->**

**R-Squared Value = 0.87**

**RMSE = 669.23**

**MAPE = 12.89**

**Accuracy = 87.11%.**