In [83]:

```
import os
import numpy as np
import pandas as pd
import scipy.stats as stats
from scipy.stats import chi2_contingency
import matplotlib as plt
import matplotlib.pyplot as plt2
import seaborn as sns
import sys
```

In [84]:

```
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [85]:

```
os.chdir("G:/Project/Bike-Sharing-Dataset")
```

In [86]:

```
bike_sharing_train = pd.read_csv("day.csv")
```

In [87]:

```
print('Shape of our dataset:')
print(bike_sharing_train.shape,'\n')
```

```
Shape of our dataset:
(731, 16)
```

Exploratory Data Analysis

In [88]:

```
print('*'*25,'Exploratory Data Analysis: ','*'*25,'\n')
```

```
************************* Exploratory Data Analysis:  *******
******************
```

In [89]:

```
print('Column / Variable Names:')
print(bike_sharing_train.columns)
```

```
Column / Variable Names:
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holida
y', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'w
indspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
```

In [90]:

```
# Showing 1st few rows of our dataset
print('Showing 1st few rows of our dataset: \n')
print(bike_sharing_train.head(5))
```

```
Showing 1st few rows of our dataset:

   instant      dteday  season  yr  mnth  holiday  weekday  w
orkingday  \
0        1  2011-01-01       1   0     1        0        6
0
1        2  2011-01-02       1   0     1        0        0
0
2        3  2011-01-03       1   0     1        0        1
1
3        4  2011-01-04       1   0     1        0        2
1
4        5  2011-01-05       1   0     1        0        3
1

   weathersit      temp     atemp       hum  windspeed  casua
l  registered  \
0           2  0.344167  0.363625  0.805833   0.160446     33
1         654
1           2  0.363478  0.353739  0.696087   0.248539     13
1         670
2           1  0.196364  0.189405  0.437273   0.248309     12
0        1229
3           1  0.200000  0.212122  0.590435   0.160296     10
8        1454
4           1  0.226957  0.229270  0.436957   0.186900      8
2        1518

    cnt
0   985
1   801
2  1349
3  1562
4  1600
```

In [91]:

```
print("Basic info about dataset:\n")
print(bike_sharing_train.info())
```

```
Basic info about dataset:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
dteday        731 non-null object
season        731 non-null int64
yr            731 non-null int64
mnth          731 non-null int64
holiday       731 non-null int64
weekday       731 non-null int64
workingday    731 non-null int64
weathersit    731 non-null int64
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
casual        731 non-null int64
registered    731 non-null int64
cnt           731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
None
```

In [92]:

```
print("Checking the data types of the variables:\n")
print(bike_sharing_train.dtypes,'\n')
```

```
Checking the data types of the variables:

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
```

In [93]:

```
print("Converting the varibales to it's proper data type: \n\nAfter Convert
bike_sharing_train['season'] = bike_sharing_train['season'].astype('categor
bike_sharing_train['yr'] = bike_sharing_train['yr'].astype('category')
bike_sharing_train['mnth'] = bike_sharing_train['mnth'].astype('category')
bike_sharing_train['weekday'] = bike_sharing_train['weekday'].astype('categ
bike_sharing_train['workingday'] = bike_sharing_train['workingday'].astype(
bike_sharing_train['weathersit'] = bike_sharing_train['weathersit'].astype(
bike_sharing_train['holiday'] = bike_sharing_train['holiday'].astype('categ

bike_sharing_train['temp'] = bike_sharing_train['temp'].astype('float')
bike_sharing_train['atemp'] = bike_sharing_train['atemp'].astype('float')
bike_sharing_train['hum'] = bike_sharing_train['hum'].astype('float')
bike_sharing_train['windspeed'] = bike_sharing_train['windspeed'].astype('f
bike_sharing_train['cnt'] = bike_sharing_train['cnt'].astype('float')

print(bike_sharing_train.dtypes,'\n')
```

```
Converting the varibales to it's proper data type:

After Convertion:

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

In [94]:

```
categorical = ['season','yr','mnth','holiday','weekday','workingday','weath
print('Count of each categorical variable in our data is as follows:\n')
[print(bike_sharing_train[i].value_counts(),'\n\n') for i in categorical]
```

```
Count of each categorical variable in our data is as follo
ws:

3    188
2    184
1    181
4    178
Name: season, dtype: int64


1    366
0    365
Name: yr, dtype: int64


12    62
10    62
8     62
7     62
5     62
3     62
1     62
11    60
9     60
6     60
4     60
2     57
Name: mnth, dtype: int64


0    710
1     21
Name: holiday, dtype: int64


6    105
1    105
0    105
5    104
4    104
3    104
2    104
Name: weekday, dtype: int64
```

```
1    500
0    231
Name: workingday, dtype: int64


1    463
2    247
3     21
Name: weathersit, dtype: int64
```

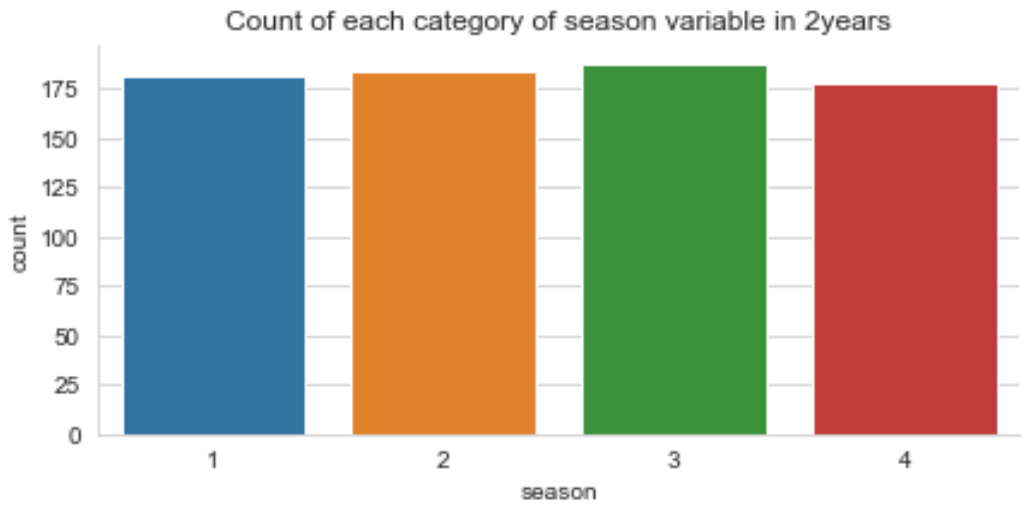
Out[94]:

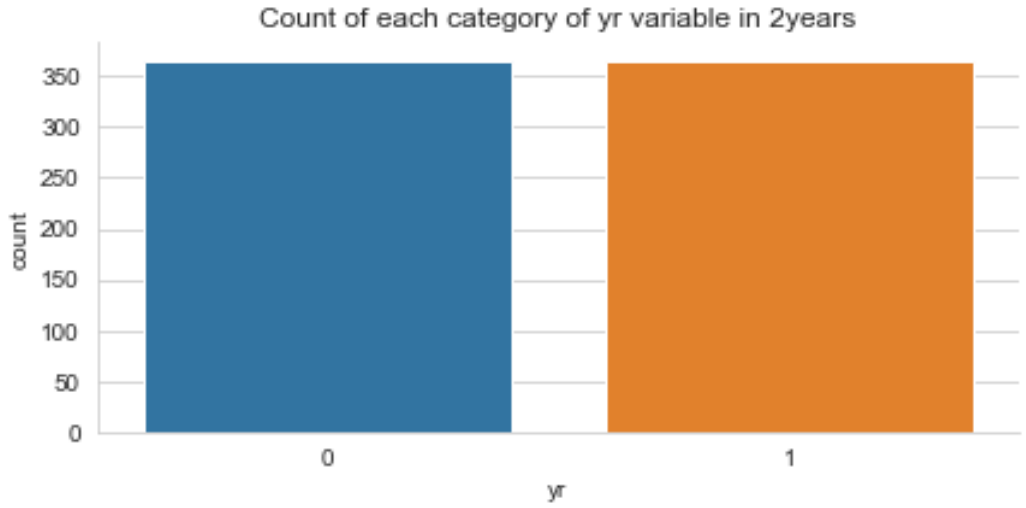
```
[None, None, None, None, None, None, None]
```

In [95]:

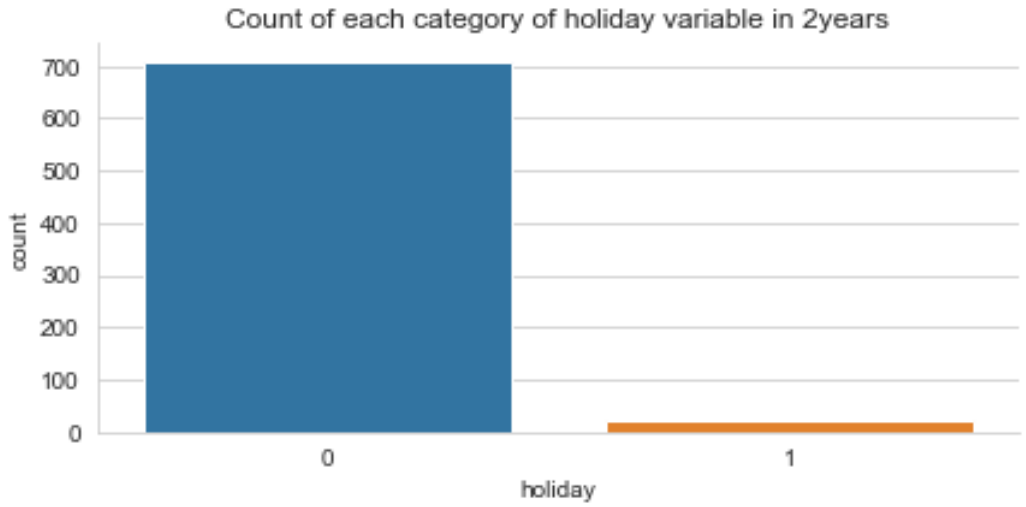
```
# Count of each category of a categorical variable
print("Checking count of each category of categorical variables in dataset\n
sns.set_style("whitegrid")

def check_count_of_category(categorical_var):
    ax = sns.factorplot(data=bike_sharing_train, x=categorical_var, kind= '
    title = "Count of each category of "+categorical_var+" variable in 2yea
    plt2.title(title)
    plt2.show()
[check_count_of_category(i) for i in categorical]
```

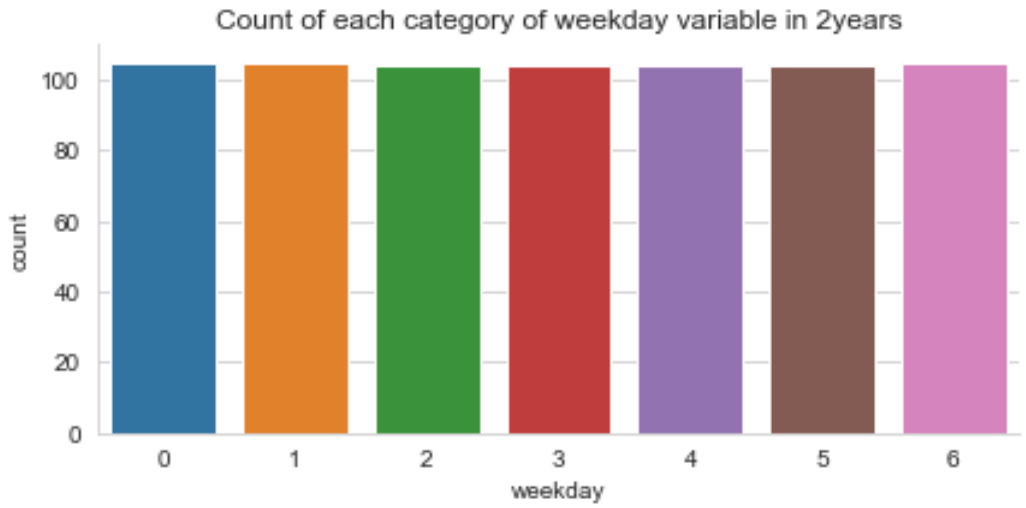
```
Checking count of each category of categorical variables in d
ataset
```

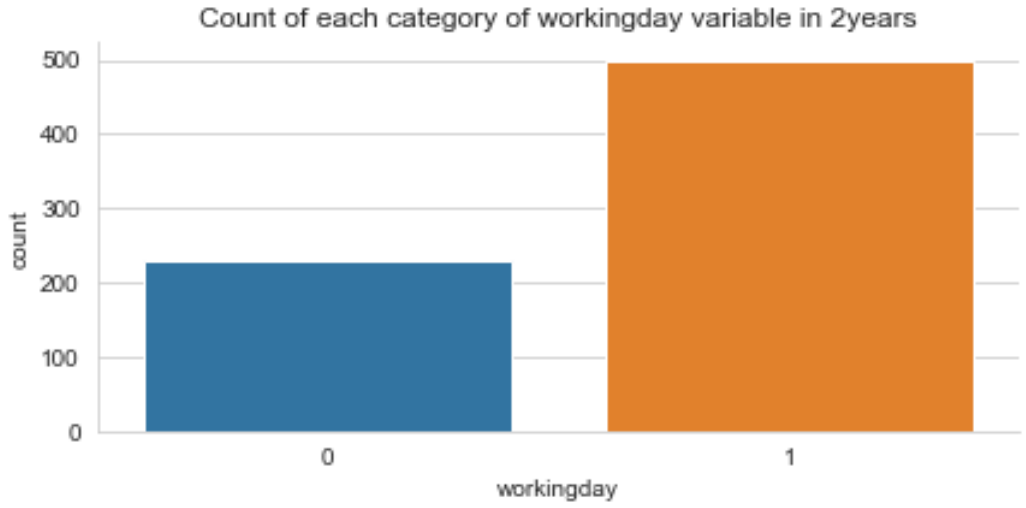


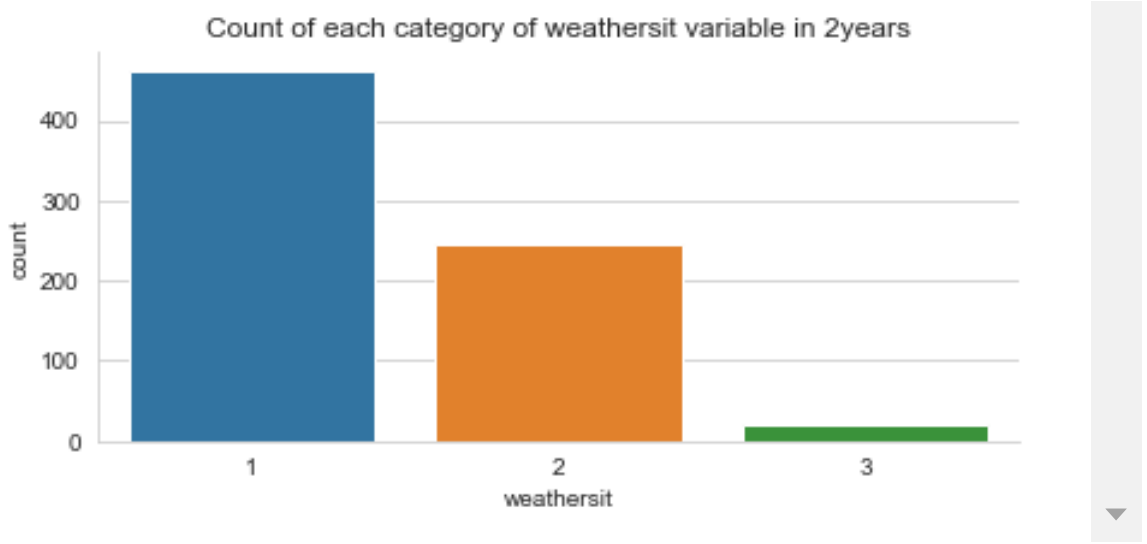



Count of each category of mnth variable in 2years
count
60
50
40
30


Count of each category of holiday variable in 2years
count
700
600
500
400
300
200
100
0
0
1
holiday


Count of each category of weekday variable in 2years
count
100
80
60
40
20
0
0
1
2
3
4
5
6
weekday


Count of each category of workingday variable in 2years
count
500
400
300
200
100
0
0
1
workingday



Out[95]:

```
[None, None, None, None, None, None, None]
```

Univariate & Bivariate analysis

In [96]:

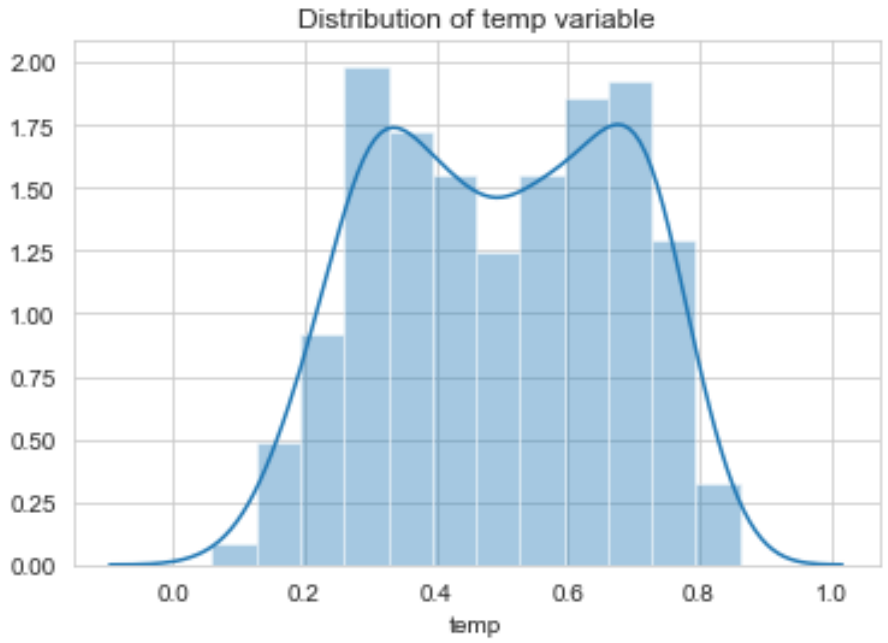
```
numeric = ['temp','atemp','hum','windspeed','casual','registered','cnt']
print("Descriptive statistics about the numeric columns:")
print(bike_sharing_train[numeric].describe(),'\n')
```

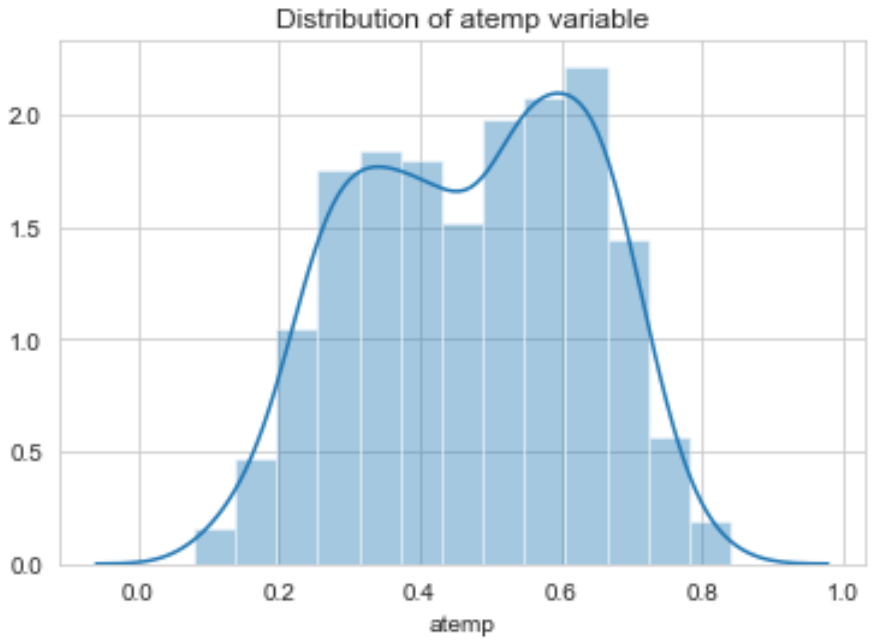
```
Descriptive statistics about the numeric columns:
             temp       atemp         hum   windspeed       c
asual  \
count  731.000000  731.000000  731.000000  731.000000   731.0
00000
mean     0.495385    0.474354    0.627894    0.190486   848.1
76471
std      0.183051    0.162961    0.142429    0.077498   686.6
22488
min      0.059130    0.079070    0.000000    0.022392     2.0
00000
25%      0.337083    0.337842    0.520000    0.134950   315.5
00000
50%      0.498333    0.486733    0.626667    0.180975   713.0
00000
75%      0.655417    0.608602    0.730209    0.233214  1096.0
00000
max      0.861667    0.840896    0.972500    0.507463  3410.0
00000

        registered          cnt
count   731.000000   731.000000
mean   3656.172367  4504.348837
std    1560.256377  1937.211452
min      20.000000    22.000000
25%    2497.000000  3152.000000
50%    3662.000000  4548.000000
75%    4776.500000  5956.000000
max    6946.000000  8714.000000
```

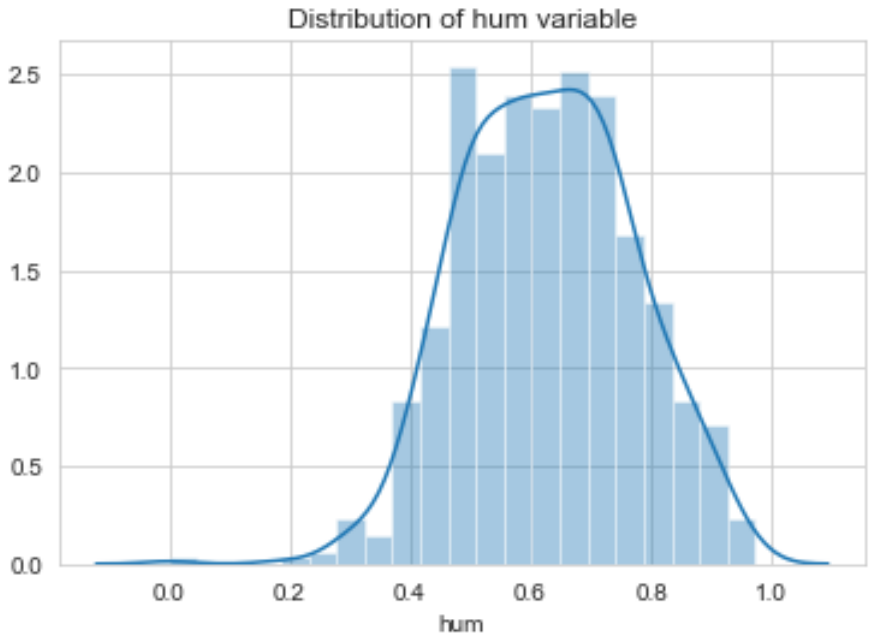
In [97]:

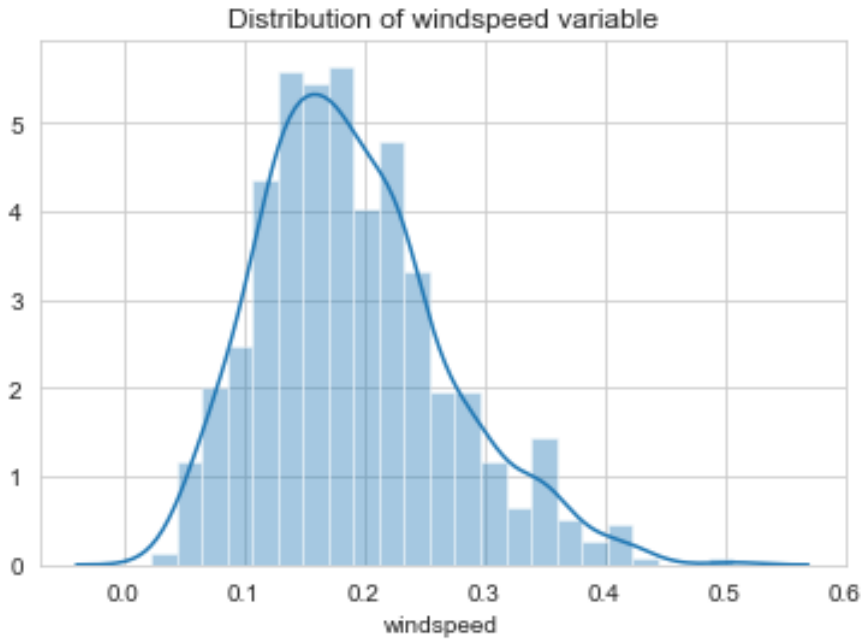
```
print("Univariate analysis of numerical variables")
def dist_plot(i):
    sns.distplot(bike_sharing_train[i])
    title = "Distribution of "+ i + " variable"
    plt2.title(title)
    plt2.show()

num = ['temp','atemp','hum','windspeed']

[dist_plot(i) for i in num]
```

```
Univariate analysis of numerical variables
```




Distribution of atemp variable
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
atemp


Distribution of hum variable
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
hum



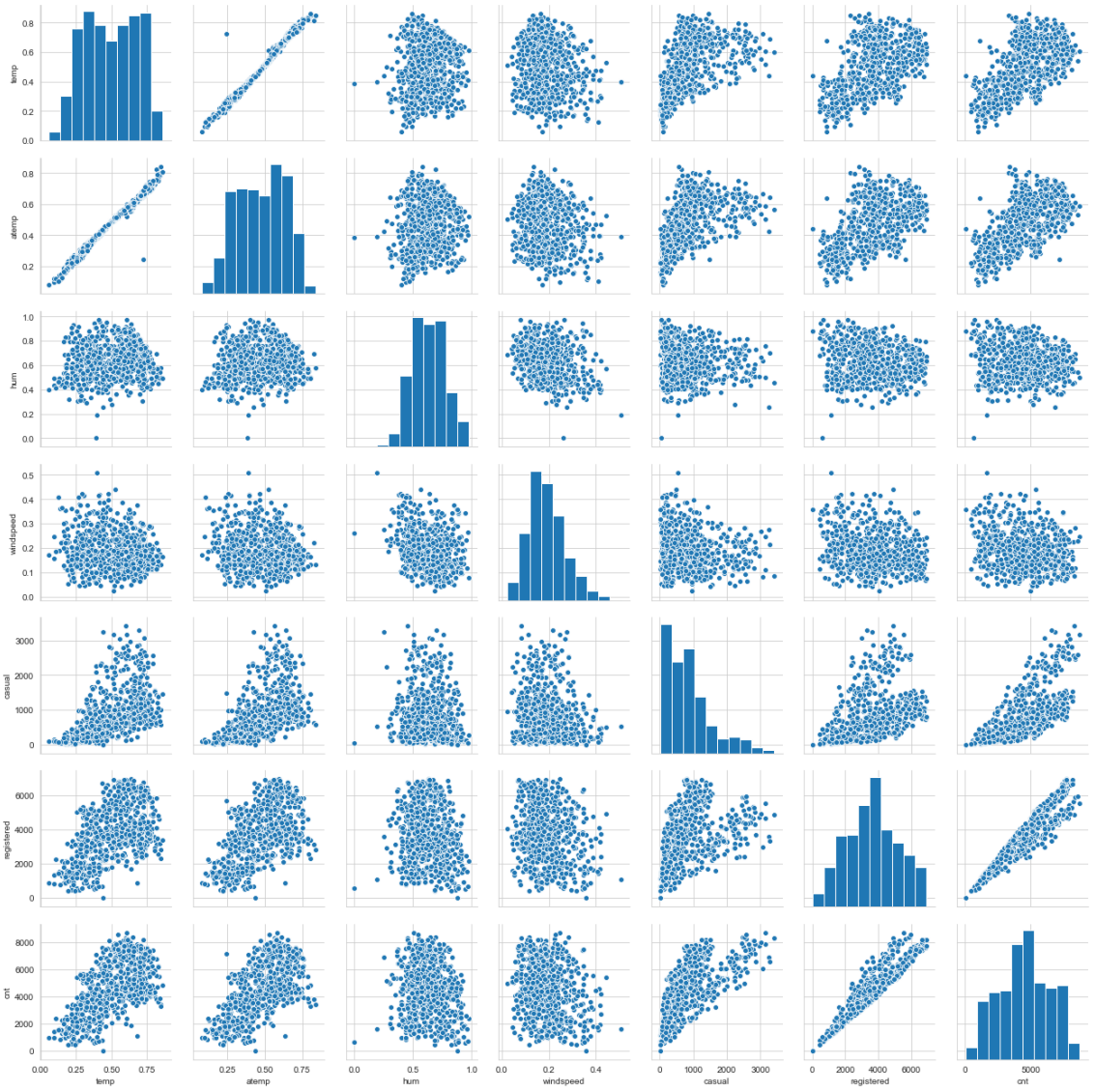
Out[97]:

```
[None, None, None, None]
```

In [98]:

```
print("Bivariate analysis of numerical variables")
sns.pairplot(bike_sharing_train[numeric])
plt2.show()
```

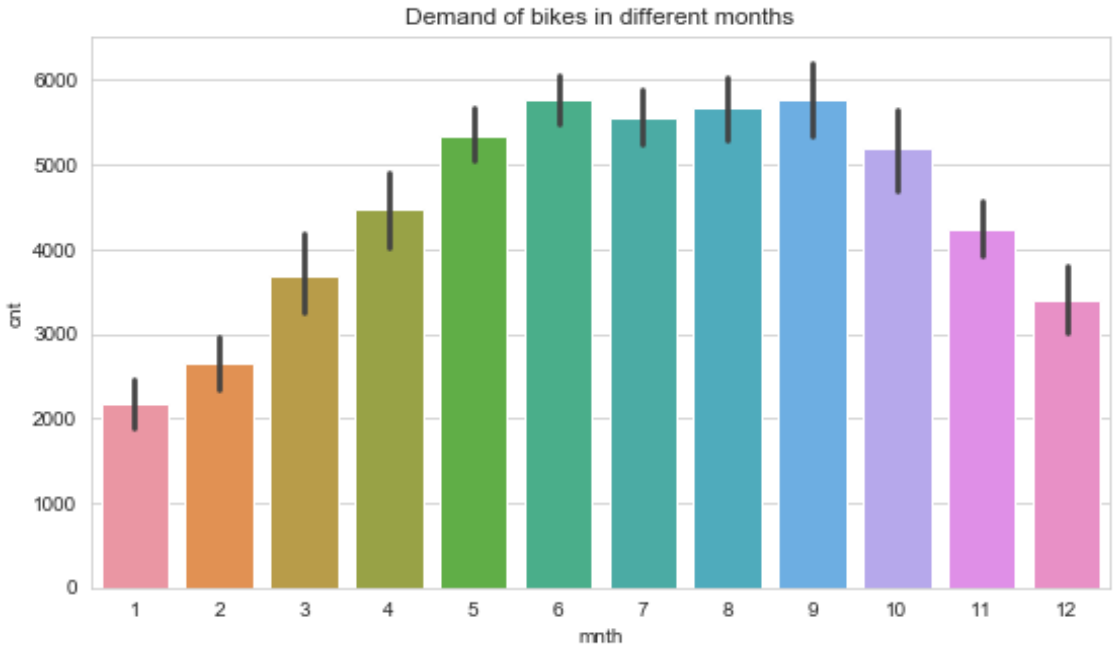
```
Bivariate analysis of numerical variables
```



For Month

In [99]:

```
# Month
fig, ax = plt2.subplots(nrows = 1, ncols = 1, figsize= (9,5), squeeze=False

x1 = 'mnth'
y1='cnt'

sns.barplot(x= x1, y = y1, data = bike_sharing_train, ax=ax[0][0])
title = "Demand of bikes in different months"
plt2.title(title)
plt2.show()
```

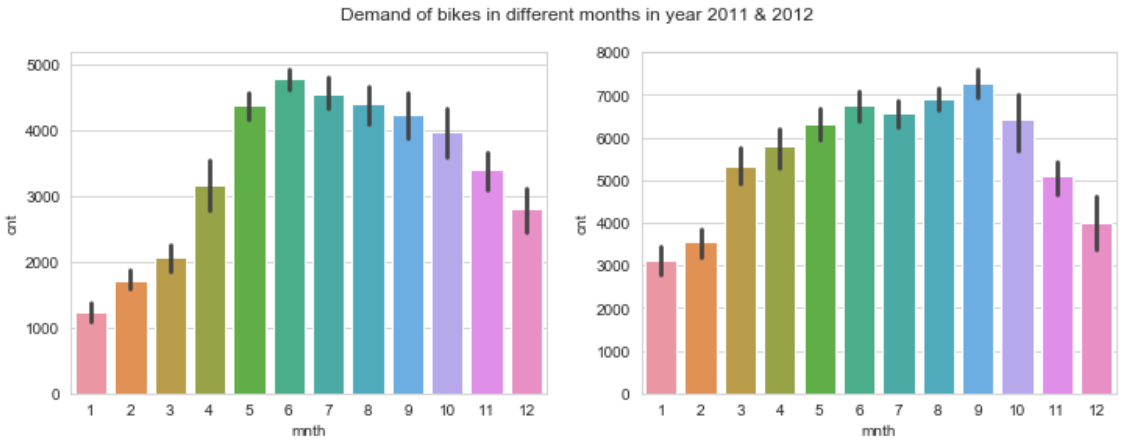


In [100]:

```
yr_0 = bike_sharing_train.loc[bike_sharing_train['yr'] == 0]
yr_1 = bike_sharing_train.loc[bike_sharing_train['yr'] == 1]
```

In [101]:

```
fig, ax = plt2.subplots(nrows = 1, ncols = 2, figsize= (12,4), squeeze=Fals
fig.suptitle("Demand of bikes in different months in year 2011 & 2012")
x1 = 'mnth'
y1='cnt'
sns.barplot(x= x1, y = y1, data = yr_0, ax=ax[0][0])
sns.barplot(x= x1, y = y1, data = yr_1, ax=ax[0][1])
plt2.show()
```



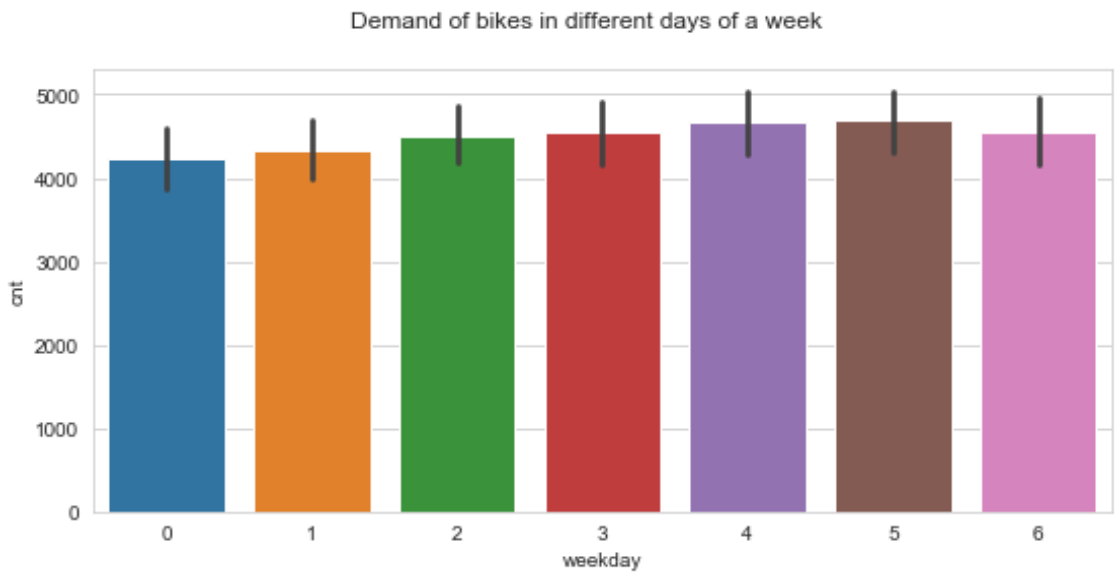
For Weekday

In [102]:

```
fig, ax = plt2.subplots(nrows = 1, ncols = 1, figsize= (9,4), squeeze=False
fig.suptitle("Demand of bikes in different days of a week")
x1 = 'weekday'
y1='cnt'

sns.barplot(x= x1, y = y1, data = bike_sharing_train, ax=ax[0][0])
```

Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x209c4cc0400>
```



In [103]:

```
print("From figures we can categorize 5-10th month as one category and rest
print("Similarly,in weekday variables; workindays can be categorized as one
```

```
From figures we can categorize 5-10th month as one category a
nd rest months as another category.

Similarly,in weekday variables; workindays can be categorized
as one and weekends as another category. As in working days d
emand of bikes found high than weekends.
```

In [104]:

```
# Keep on adding the unwanted variables (that we will get by applying diffe
# will finally we will remove from our dataset
remove = ['instant','dteday']
```

Missing Value Analysis

In [105]:

```
print('*'*25,'Missing Value Analysis: ','*'*25,'\n')
```

```
************************* Missing Value Analysis:  **********
***************
```

In [106]:

```
missing_val = pd.DataFrame(bike_sharing_train.isnull().sum())
```

In [107]:

```
print('Missing values in our dataset: \n')
print(missing_val)
```

```
Missing values in our dataset:

            0
instant     0
dteday      0
season      0
yr          0
mnth        0
holiday     0
weekday     0
workingday  0
weathersit  0
temp        0
atemp       0
hum         0
windspeed   0
casual      0
registered  0
cnt         0
```

In [108]:

```
print("No missing values present in our dataset")
```

```
No missing values present in our dataset
```

Outlier Analysis

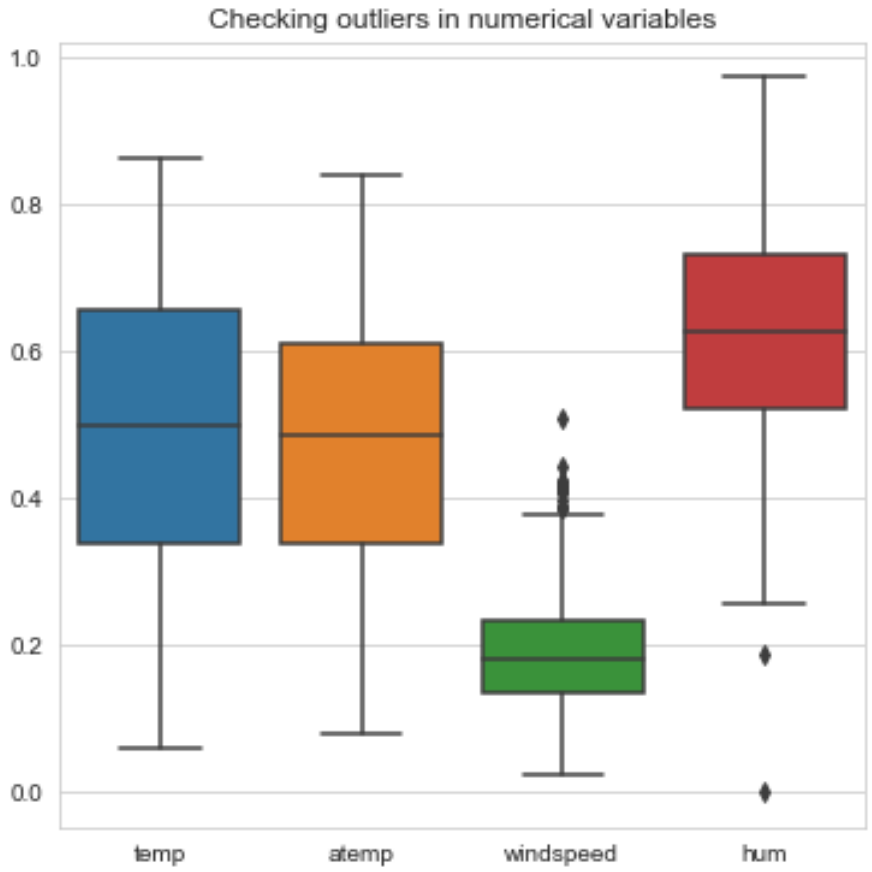
In [109]:

```
print('*'*25,'Outlier Analysis','*'*25,'\n')
```

```
************************* Outlier Analysis ******************
*******
```

In [110]:

```
# Check for outliers in data using boxplot
sns.boxplot(data=bike_sharing_train[['temp','atemp','windspeed','hum']])
fig=plt2.gcf()
title = "Checking outliers in numerical variables"
plt2.title(title)
fig.set_size_inches(6,6)
```



In [111]:

```
print("Outliers found in windspeed and humidity variable")
```

```
Outliers found in windspeed and humidity variable
```

In [112]:

```
# Numeric Variables
num = ['temp','atemp','hum','windspeed']
```

In [113]:

```
# Removing the outliers
for i in num:
    q75, q25 = np.percentile(bike_sharing_train[i], [75, 25])
    iqr = q75 - q25
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)

    bike_sharing_train = bike_sharing_train.drop(bike_sharing_train[bike_sh
    bike_sharing_train = bike_sharing_train.drop(bike_sharing_train[bike_sh
print("Outliers removed")
```

```
Outliers removed
```

Feature Engineering

In [114]:

```
bike_sharing_train.head()
```

Out[114]:

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

In [115]:

```
categorical = ['season','yr','mnth','holiday','weekday','workingday','weath
```

In [116]:

```
# Creating new variables  through binning
def binned_month(row):
    if row['mnth'] <= 4 or row['mnth'] >=11:
        return(0)
    else:
        return(1)

def binned_weekday(row):
    if row['weekday'] < 2:
        return(0)
    else:
        return(1)
```

In [117]:

```
bike_sharing_train['month_binned'] = bike_sharing_train.apply(lambda row : 
bike_sharing_train = bike_sharing_train.drop(columns=['mnth'])
bike_sharing_train['weekday_binned'] = bike_sharing_train.apply(lambda row
bike_sharing_train = bike_sharing_train.drop(columns=['weekday'])
```

In [118]:

```
categorical.remove('mnth')
categorical.remove('weekday')
categorical.append('month_binned')
categorical.append('weekday_binned')
```

Feature Selection

In [119]:

```
bike_sharing_train.columns
```

Out[119]:

```
Index(['instant', 'dteday', 'season', 'yr', 'holiday', 'worki
ngday',
       'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'ca
sual',
       'registered', 'cnt', 'month_binned', 'weekday_binne
d'],
      dtype='object')
```

Correlation Analysis

In [120]:

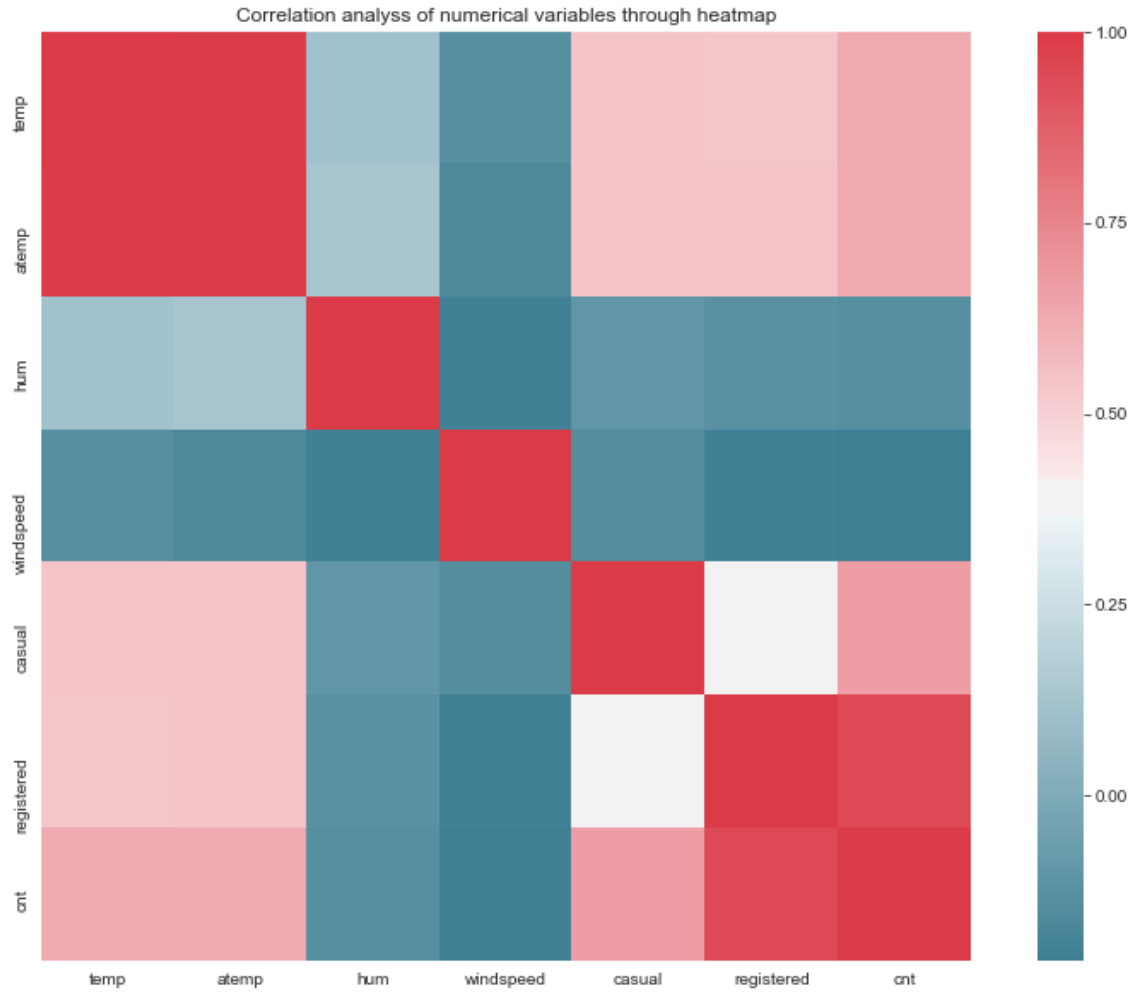
```
df_corr = bike_sharing_train[numeric]
```

In [121]:

```
# Correlation Analysis
f, ax = plt2.subplots(figsize=(14, 10))
corr = df_corr.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.divergi
            square=True, ax=ax)
plt2.title("Correlation analyss of numerical variables through heatmap")
```

Out[121]:

```
Text(0.5, 1.0, 'Correlation analyss of numerical variables th
rough heatmap')
```



In [122]:

```
print("From correlation analysis we found,\n    1.temp and atemp are highly
```

```
From correlation analysis we found,
    1.temp and atemp are highly correlated.
    2.registered and cnt also showing high correlation.
```

In [123]:

```
remove.extend(['atemp','casual','registered'])
```

Chi-square test

In [124]:

```
print("Chi-square Test\n1. Null Hypothesis: Two variables are independent\n
print("That means p < 0.05 means two categorical variables are dependent, s
```

```
Chi-square Test
1. Null Hypothesis: Two variables are independent
2. Alternate Hypothesis: Two variables are not independent
3. p-value < 0.05 , can not accept null hypothesis

That means p < 0.05 means two categorical variables are depen
dent, so we will remove one of variable from that pair to avo
id sending the same information to our model through 2 variab
les
```

In [125]:

```
# Create all combinations
factors_paired = [(i,j) for i in categorical for j in categorical]
```

In [126]:

```
# Calculating p-values for each pair
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(bike_sharing_train[
                                                        bike_sharing_train[fact
        if(p<0.05):
            p_values.append({factor:p.round(3)})
    else:
        p_values.append('-')

[print(i,'\n') for i in p_values if i != '-']
```

```
{('season', 'weathersit'): 0.013}

{('season', 'month_binned'): 0.0}

{('holiday', 'workingday'): 0.0}

{('holiday', 'weekday_binned'): 0.0}

{('workingday', 'holiday'): 0.0}

{('workingday', 'weekday_binned'): 0.0}

{('weathersit', 'season'): 0.013}

{('month_binned', 'season'): 0.0}

{('weekday_binned', 'holiday'): 0.0}

{('weekday_binned', 'workingday'): 0.0}
```

Out[126]:

```
[None, None, None, None, None, None, None, None, None, None]
```

In [127]:

```
print("Season with Weathersit-Month\nHoliday with Worikingday-Weekday\nWork
```

```
Season with Weathersit-Month
Holiday with Worikingday-Weekday
Workingday with Weekday-Holiday
```

In [128]:

```
bike_sharing_train.columns
```

Out[128]:

```
Index(['instant', 'dteday', 'season', 'yr', 'holiday', 'worki
ngday',
       'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'ca
sual',
       'registered', 'cnt', 'month_binned', 'weekday_binne
d'],
      dtype='object')
```

Importance of Features

In [129]:

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0, n_jobs=-1)
X = bike_sharing_train.drop(columns=['cnt','casual','registered','instant',
y = bike_sharing_train['cnt']
model = clf.fit(X, y)
importances = model.feature_importances_
```

In [130]:

```
X.columns
```

Out[130]:

```
Index(['season', 'yr', 'holiday', 'workingday', 'weathersit',
'temp', 'atemp',
       'hum', 'windspeed', 'month_binned', 'weekday_binned'],
      dtype='object')
```

In [131]:

```
print("Checking feature importance: \n")
l = list(zip(X,importances))
l.sort(key = lambda x: x[1])
[print(i[0]," : ",i[1].round(3)) for i in l]
```

```
Checking feature importance:

holiday  :  0.008
month_binned  :  0.024
weekday_binned  :  0.031
workingday  :  0.033
yr  :  0.037
weathersit  :  0.04
season  :  0.045
windspeed  :  0.188
hum  :  0.191
atemp  :  0.198
temp  :  0.202
```

Out[131]:

```
[None, None, None, None, None, None, None, None, None, None,
None]
```

In [132]:

```
remove.append('holiday')
```

Multi-colinearity test

In [133]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
numeric_df = add_constant(bike_sharing_train[['temp', 'atemp', 'hum', 'wind
vif = pd.Series([vf(numeric_df.values, j) for j in range(numeric_df.shape[1
vif.round(3)
```

Out[133]:

```
const        46.436
temp         63.326
atemp        63.933
hum           1.057
windspeed     1.102
dtype: float64
```

In [134]:

```
print("After removing atemp variable , VIF:\n")
numeric_df = add_constant(bike_sharing_train[['temp', 'hum', 'windspeed']])
vif = pd.Series([vf(numeric_df.values, i) for i in range(numeric_df.shape[1
                 index = numeric_df.columns)
print(vif.round(3))
```

```
After removing atemp variable , VIF:

const        41.644
temp          1.028
hum           1.052
windspeed     1.059
dtype: float64
```

Dummy for categorical

In [135]:

```
season_dm = pd.get_dummies(bike_sharing_train['season'], drop_first=True, p
bike_sharing_train = pd.concat([bike_sharing_train, season_dm],axis=1)
bike_sharing_train = bike_sharing_train.drop(columns = ['season'])
weather_dm = pd.get_dummies(bike_sharing_train['weathersit'], prefix= 'weat
bike_sharing_train = pd.concat([bike_sharing_train, weather_dm],axis=1)
bike_sharing_train = bike_sharing_train.drop(columns= ['weathersit'])
```

In [136]:

```
remove
```

Out[136]:

```
['instant', 'dteday', 'atemp', 'casual', 'registered', 'holid
ay']
```

In [137]:

```
# Removing unwanted variables
bike_sharing_train.drop(columns=remove, inplace=True)
```

In [138]:

```
# Reshaping
cnt = bike_sharing_train['cnt']
bike_sharing_train = bike_sharing_train.drop(columns=['cnt'])
bike_sharing_train['cnt'] = cnt
```

In [145]:

```
bike_sharing_train.shape
```

Out[145]:

```
(717, 13)
```

In [141]:

```
print(bike_sharing_train.head(5), '\n')
print('shape of dataset after all pre-processing\n',bike_sharing_train.shap
```

```
  yr workingday      temp       hum  windspeed  month_binned
weekday_binned  \
0  0          0  0.344167  0.805833   0.160446             0
1
1  0          0  0.363478  0.696087   0.248539             0
0
2  0          1  0.196364  0.437273   0.248309             0
0
3  0          1  0.200000  0.590435   0.160296             0
1
4  0          1  0.226957  0.436957   0.186900             0
1

   season_2  season_3  season_4  weather_2  weather_3     cnt
0         0         0         0          1          0   985.0
1         0         0         0          1          0   801.0
2         0         0         0          0          0  1349.0
3         0         0         0          0          0  1562.0
4         0         0         0          0          0  1600.0

shape of dataset after all pre-processing
 (717, 13)
```

Model Development

In [70]:

```
# Modularizing
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score

def fit_N_predict(model, X_train, y_train, X_test, y_test,model_code=''):

    if(model_code == 'OLS'):
        model = model.OLS(y_train,X_train.astype('float')).fit()
        print(model.summary())
        y_pred = model.predict(X_test.astype('float'))
        print("\n================================")
        print('Score on testing data: ',(r2_score(y_test,y_pred)*100).round
        print("================================")
        return

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("================================")
    print("Score on training data: ",(model.score(X_train, y_train)*100.0).
    print("================================")
    print("Score on testing data: ", (model.score(X_test, y_test)*100.0).ro
    print("================================")

    if(model_code == "DT"):
        from sklearn import tree
        dotfile = open("pt.dot","w")
        df = tree.export_graphviz(model, out_file=dotfile, feature_names = )
```

In [71]:

```
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score
from statistics import mean
kf = KFold(n_splits=10, shuffle=True, random_state=42)


def cross_validation(model,X,y):
    l = []
    for train_index, test_index in kf.split(X,y):
        X_train, X_test = X.iloc[train_index,], X.iloc[test_index,]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)
        l.append(r2_score(y_test,y_pred))
    print("Mean of 10 cross validation scores = ",(mean(l)*100).round(3))
```

In [72]:

```
# Partitioning of dataset
X = bike_sharing_train.drop(columns=['cnt'])
y = bike_sharing_train[['cnt']]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

In [73]:

```
print('######################### LINEAR REGRESSION ########################
from sklearn.linear_model import LinearRegression

model = LinearRegression()
fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="SK_LR")
cross_validation(model,X,y)
```

```
######################### LINEAR REGRESSION #################
########
================================
Score on training data:  81.263
================================
Score on testing data:  87.42
================================
Mean of 10 cross validation scores =  81.57
```

In [74]:

```
import statsmodels.api as sm
model = sm
fit_N_predict(model,X_train, y_train, X_test, y_test,model_code="OLS")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared (uncentered):                   0.970
Model:                            OLS   Adj. R-squared (uncentered):              0.970
Method:                 Least Squares   F-statistic:                              1533.
Date:                Tue, 07 Jan 2020   Prob (F-statistic):                        0.00
Time:                        15:29:18   Log-Likelihood:                         -4671.9
No. Observations:                 573   AIC:                                      9368.
Df Residuals:                     561   BIC:                                      9420.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
yr            2132.5260     70.874     30.089      0.000    1993.315    2271.737
workingday     111.1339     81.672      1.361      0.174     -49.287     271.555
temp          4724.7662    404.637     11.677      0.000    3929.977    5519.555
hum            244.8524    249.110      0.983      0.326    -244.450     734.155
windspeed     -778.5416    434.626     -1.791      0.074   -1632.234      75.151
month_binned   310.6609    120.701      2.574      0.010      73.580     547.741
weekday_binned 368.0815     83.653      4.400      0.000     203.771     532.392
season_2      1104.2190    131.957      8.368      0.000     845.028    1363.410
season_3       736.0110    174.951      4.207      0.000     392.372    1079.650
season_4      1489.8955    114.025     13.066      0.000
```

```
1265.927    1713.864
weather_2       -588.5358     91.709     -6.417      0.000
-768.671    -408.401
weather_3      -2465.6840    245.016    -10.063      0.000
-2946.945   -1984.423
==============================================================
====================
Omnibus:                        85.095   Durbin-Watson:
1.812
Prob(Omnibus):                   0.000   Jarque-Bera (JB):
225.228
Skew:                           -0.748   Prob(JB):
1.24e-49
Kurtosis:                        5.683   Cond. No.
22.5
==============================================================
====================

Warnings:
[1] Standard Errors assume that the covariance matrix of t
he errors is correctly specified.

================================
Score on testing data:  86.747
================================
```

In [279]:

```
print('########################## K NEIGHBOURS REGRESSOR ###################
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=5)
fit_N_predict(model, X_train, y_train, X_test, y_test,model_code="KNN")
cross_validation(model,X,y)
```

```
########################## K NEIGHBOURS REGRESSOR ############
############
================================
Score on training data:  85.281
================================
Score on testing data:  87.095
================================
Mean of 10 cross validation scores =  79.448
```

In [282]:

```
print('########################## SVR ##########################')
from sklearn.svm import SVR
model = SVR(kernel = 'linear')
fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="SVR")
cross_validation(model,X,y)
```

```
########################## SVR ##########################
================================
Score on training data:  9.268
================================
Score on testing data:  9.741
================================
Mean of 10 cross validation scores =  9.232
```

In [75]:

```
print('########################## Decision Tree ##########################')

from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(random_state=1)
fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="DT")
cross_validation(model,X,y)
```

```
########################## Decision Tree ####################
####
================================
Score on training data:  100.0
================================
Score on testing data:  82.819
================================
Mean of 10 cross validation scores =  74.95
```

In [141]:

```
print('######################### Random Forest ##########################')

from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(random_state=1)
fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="RF")
cross_validation(model,X,y)
```

```
######################### Random Forest ####################
####
================================
Score on training data:  96.95
================================
Score on testing data:  91.274
================================
Mean of 10 cross validation scores =  84.933
```

In [144]:

```
# Pre-processing of data for XGBoost
bike = bike_sharing_train.copy()
yr_dm = pd.get_dummies(bike['yr'], prefix= 'yr',drop_first=True)
bike = pd.concat([bike, yr_dm],axis=1)
bike = bike.drop(columns= ['yr'])

workingday_dm = pd.get_dummies(bike['workingday'], prefix= 'workingday',dro
bike = pd.concat([bike, workingday_dm],axis=1)
bike = bike.drop(columns= ['workingday'])

bike['yr_1'] = bike['yr_1'].astype('int')
bike['workingday_1'] = bike['workingday_1'].astype('int')

X1 = bike.drop(columns=['cnt'])
y1 = bike['cnt']
from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size =
```

In [145]:

```
print('######################### XGB Regressor #########################')

from xgboost import XGBRegressor
model = XGBRegressor(random_state=1)
fit_N_predict(model, X_train1, y_train1, X_test1, y_test1, model_code="XGB"
cross_validation(model,X1,y1)
```

```
######################### XGB Regressor #####################
####
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
================================
Score on training data:  94.004
================================
Score on testing data:  92.043
================================
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:18:22] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
Mean of 10 cross validation scores =  87.5
```

Hyper-parameter tuning for the best models

In [147]:

```
print('########################## Tuning Random Forest ######################

from sklearn.model_selection import GridSearchCV
model = RandomForestRegressor(random_state=1)
params = [{'n_estimators' : [400, 500, 600],
           'max_features': ['auto', 'sqrt', 'log2'],
           'min_samples_split': [2,4,6],
           'max_depth': [10, 12, 14],
           'min_samples_leaf': [2,3,5],
           'random_state' : [1]
          }]
grid_search = GridSearchCV(estimator=model, param_grid=params,cv = 5,
                           scoring = 'explained_variance', n_jobs=-1)
grid_search = grid_search.fit(X_train, y_train)
print('Best parameters for Random Forest',grid_search.best_params_)
```

```
########################## Tuning Random Forest ##############
###########
Best parameters for Random Forest {'max_depth': 14, 'max_feat
ures': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2,
'n_estimators': 500, 'random_state': 1}
```

In [148]:

```
# Developing Random Forest model with best params
model = RandomForestRegressor(random_state=1, max_depth=14, n_estimators=50
                                  max_features='auto', min_samples_leaf=2,m
fit_N_predict(model, X_train, y_train, X_test, y_test, model_code="RF")
cross_validation(model,X,y)
```

```
================================
Score on training data:  96.263
================================
Score on testing data:  91.706
================================
Mean of 10 cross validation scores =  86.835
```

In [149]:

```
print('######################### Tuning XGBoost ##########################')
model = XGBRegressor(random_state=1)
params = [{'n_estimators' : [200, 250, 300,350, 400,450],
           'max_depth':[2, 3, 5],
           'learning_rate':[0.01, 0.045, 0.05, 0.055, 0.1, 0.3],
           'gamma':[0, 0.001, 0.01, 0.03],
           'subsample':[1, 0.7, 0.8, 0.9],
           'random_state' :[1]
          }]
grid_search = GridSearchCV(estimator=model, param_grid=params,cv = 5,
                           scoring = 'explained_variance', n_jobs=-1)
grid_search = grid_search.fit(X_train1, y_train1)
print('Best parameters for XGBoost',grid_search.best_params_)
```

```
######################### Tuning XGBoost ####################
#####
[22:40:14] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
Best parameters for XGBoost {'gamma': 0, 'learning_rate': 0.0
45, 'max_depth': 3, 'n_estimators': 300, 'random_state': 1,
'subsample': 0.7}
```

In [150]:

```
# Developing XGBoost model with best params
model = XGBRegressor(random_state=1, learning_rate=0.045, max_depth=3, n_es
                         gamma = 0, subsample=0.7)
fit_N_predict(model, X_train1, y_train1, X_test1, y_test1, model_code="XGB"
cross_validation(model,X1,y1)
```

```
[22:40:54] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
================================
Score on training data:  95.519
================================
Score on testing data:  92.928
================================
[22:40:54] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:54] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:54] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:54] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:54] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:55] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:55] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:55] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:55] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
[22:40:55] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
Mean of 10 cross validation scores =  88.34
```

In [151]:

```
# Scatterplot showing the prediction vs actual values for the best model fo

model = XGBRegressor(random_state=1, learning_rate=0.045, max_depth=3, n_es
                     gamma = 0, subsample=0.7)
model.fit(X_train1,y_train1)
y_pred = model.predict(X_test1)
fig, ax = plt2.subplots(figsize=(7,5))
ax.scatter(y_test1, y_pred)
ax.plot([0,8000],[0,8000], 'r--', label='Perfect Prediction')
ax.legend()
plt2.title("Scatter plot between y_test and y_pred")
plt2.xlabel("y_test")
plt2.ylabel("y_pred")
plt2.tight_layout()
plt2.show()
```

```
[22:40:59] WARNING: src/objective/regression_obj.cu:152: reg:
linear is now deprecated in favor of reg:squarederror.
```