Week6_4_DTree_Rforest_boosting_models

May 13, 2021

Decision Tree, Random Forest, and Boosting models

Importing Python Libraries we need

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor,RandomForestRegressor,␣
↪GradientBoostingRegressor, AdaBoostRegressor, StackingRegressor
from xgboost import XGBRegressor
from sklearn import metrics
from sklearn.model_selection import GridSearchCV, train_test_split

import warnings
warnings.filterwarnings("ignore")
```

Load and overview the dataset

[2]:
```
data=pd.read_csv("/home/jayanthikishore/Downloads/ML_classwork/DT_RF_Ensemble/
↪hour.csv")
data.head()
```

[2]:
```
   instant      dteday  season  yr  mnth  hr  holiday  weekday  workingday  \
0        1  2011-01-01       1   0     1   0        0        6           0
1        2  2011-01-01       1   0     1   1        0        6           0
2        3  2011-01-01       1   0     1   2        0        6           0
3        4  2011-01-01       1   0     1   3        0        6           0
4        5  2011-01-01       1   0     1   4        0        6           0

   weathersit  temp   atemp   hum  windspeed  casual  registered  cnt
0           1  0.24  0.2879  0.81        0.0       3          13   16
1           1  0.22  0.2727  0.80        0.0       8          32   40
2           1  0.22  0.2727  0.80        0.0       5          27   32
3           1  0.24  0.2879  0.75        0.0       3          10   13
4           1  0.24  0.2879  0.75        0.0       0           1    1
```

Dataset description

Bike-sharing rental process is highly correlated to the environmental and seasonal settings. For instance, weather conditions, precipitation, day of week, season, hour of the day, etc. can affect the rental behaviors.

- instant: record index
- dteday : date
- season : season (1:spring, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mnth : month (1 to 12)
- hr : hour (0 to 23)
- holiday : whether day is holiday or not
- weekday : day of the week
- workingday : if day is neither weekend nor holiday then 1, otherwise is 0.
- weathersit :
  - 1: Clear, Few clouds, 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 : Normalized temperature in Celsius. The values are divided to 41 (max)
- atemp: Normalized feeling temperature in Celsius. The values are divided to 50 (max). The "feel like" temperature relies on environmental data including the ambient air temperature, relative humidity, and wind speed to determine how weather conditions feel to bare skin.
- 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
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

Check data types and number of non-null values for each column

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instant     17379 non-null  int64
 1   dteday      17379 non-null  object
 2   season      17379 non-null  int64
 3   yr          17379 non-null  int64
 4   mnth        17379 non-null  int64
 5   hr          17379 non-null  int64
 6   holiday     17379 non-null  int64
 7   weekday     17379 non-null  int64
 8   workingday  17379 non-null  int64
 9   weathersit  17379 non-null  int64
 10  temp        17379 non-null  float64
```

```
 11  atemp       17379 non-null  float64
 12  hum         17379 non-null  float64
 13  windspeed   17379 non-null  float64
 14  casual      17379 non-null  int64
 15  registered  17379 non-null  int64
 16  cnt         17379 non-null  int64
dtypes: float64(4), int64(12), object(1)
memory usage: 2.3+ MB
```

Data shape

[4]:
```
data.shape
```

[4]:
```
(17379, 17)
```

- We can see that there are total 17,379 number of rows and 17 columns in the dataset.

Check NULL values at variable

[5]:
```
data.isna().sum()
```

[5]:
```
instant       0
dteday        0
season        0
yr            0
mnth          0
hr            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
```

- There are no NULL values are found in the dataset

Statistical summary for specified variables

[6]:
```
data[['temp','atemp','hum','windspeed','cnt']].describe().T
```

[6]:
```
         count      mean       std   min     25%     50%     75%  \
temp   17379.0  0.496987  0.192556  0.02  0.3400  0.5000  0.6600
atemp  17379.0  0.475775  0.171850  0.00  0.3333  0.4848  0.6212
hum    17379.0  0.627229  0.192930  0.00  0.4800  0.6300  0.7800
```

```
windspeed  17379.0    0.190098    0.122340  0.00   0.1045    0.1940    0.2537
cnt        17379.0  189.463088  181.387599  1.00  40.0000  142.0000  281.0000

                max
temp         1.0000
atemp        1.0000
hum          1.0000
windspeed    0.8507
cnt        977.0000
```

Number of unique values in each column

[7]:
```python
data.nunique()
```

[7]:
```
instant       17379
dteday          731
season            4
yr                2
mnth             12
hr               24
holiday           2
weekday           7
workingday        2
weathersit        4
temp             50
atemp            65
hum              89
windspeed        30
casual          322
registered      776
cnt             869
dtype: int64
```

Drop some columns from the data table

[8]:
```python
data.drop(columns=['instant','dteday'],inplace=True)
```

Number of observations in each category

[9]:
```python
catog_cols = ['season','yr','holiday','workingday','weathersit']
for column in catog_cols:
    print(data[column].value_counts())
    print('-'*30)
```

```
3    4496
2    4409
1    4242
4    4232
```

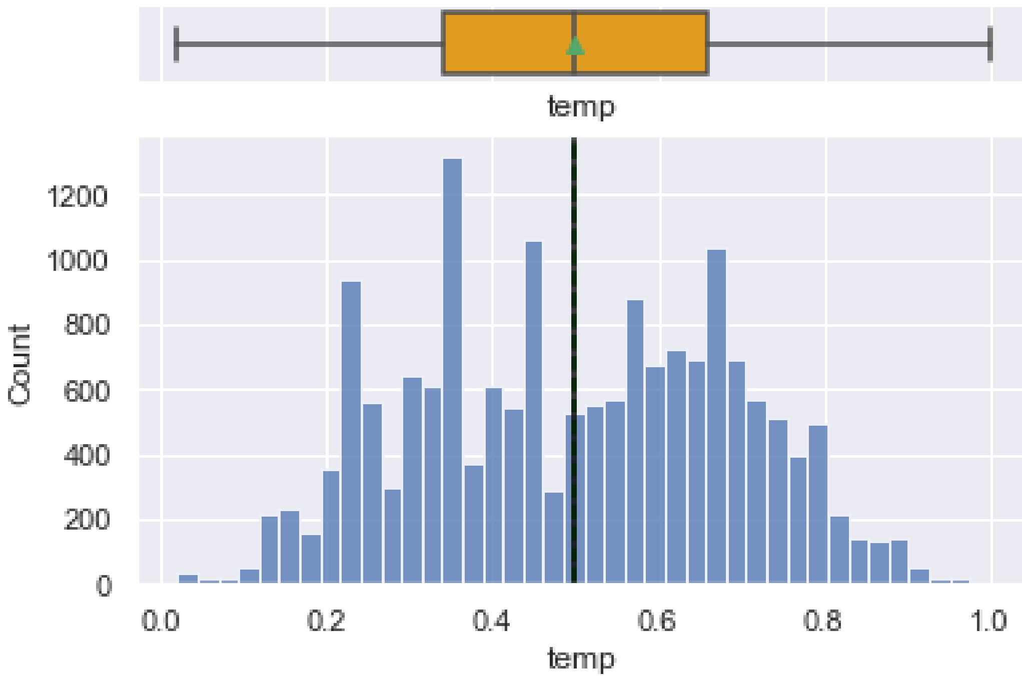
```
Name: season, dtype: int64
------------------------------
1    8734
0    8645
Name: yr, dtype: int64
------------------------------
0    16879
1      500
Name: holiday, dtype: int64
------------------------------
1    11865
0     5514
Name: workingday, dtype: int64
------------------------------
1    11413
2     4544
3     1419
4        3
Name: weathersit, dtype: int64
------------------------------
```

Exploratory Data Analysis (EDA)

- Univariate Analysis - Temperature observations

[10]:
```python
# set a grey background (use sns.set_theme() if seaborn version 0.11.0 or␣
↪above)
sns.set(style="darkgrid")

# creating a figure composed of two matplotlib.Axes objects (ax_box and ax_hist)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,␣
↪gridspec_kw={"height_ratios": (.15, .85)})

# assigning a graph to each ax
sns.boxplot(data["temp"], ax=ax_box, showmeans=True,color='orange')
sns.histplot(data=data, x="temp", ax=ax_hist)
ax_hist.axvline(np.mean(data['temp']), color='green', linestyle='--') # Add␣
↪mean to the histogram
ax_hist.axvline(np.median(data['temp']), color='black', linestyle='-') # Add␣
↪median to the histogram
```

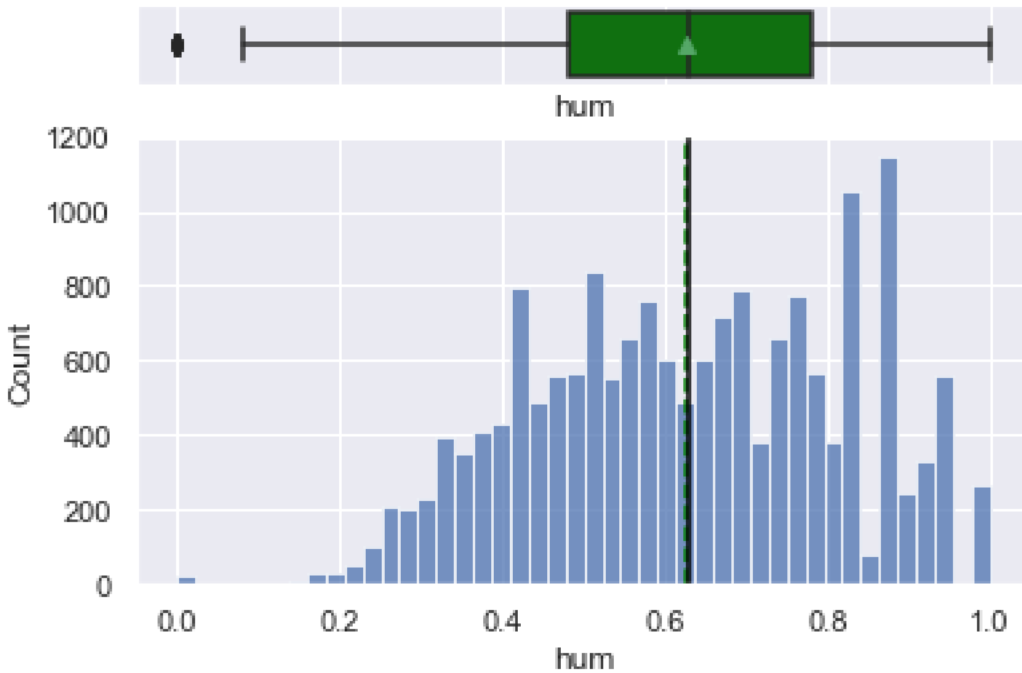
[10]:
```
<matplotlib.lines.Line2D at 0x7ffa4e49ed30>
```



- Temperature data looks like symmetric and mean and median value nearly 0.5
- Humidity observations

[11]:
```
# set a grey background (use sns.set_theme() if seaborn version 0.11.0 or␣
↪above)
sns.set(style="darkgrid")

# creating a figure composed of two matplotlib.Axes objects (ax_box and ax_hist)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,␣
↪gridspec_kw={"height_ratios": (.15, .85)})

# assigning a graph to each ax
sns.boxplot(data["hum"], ax=ax_box, showmeans=True,color='green')
sns.histplot(data=data, x="hum", ax=ax_hist)
ax_hist.axvline(np.mean(data['hum']), color='green', linestyle='--') # Add mean␣
↪to the histogram
ax_hist.axvline(np.median(data['hum']), color='black', linestyle='-') # Add␣
↪median to the histogram
```

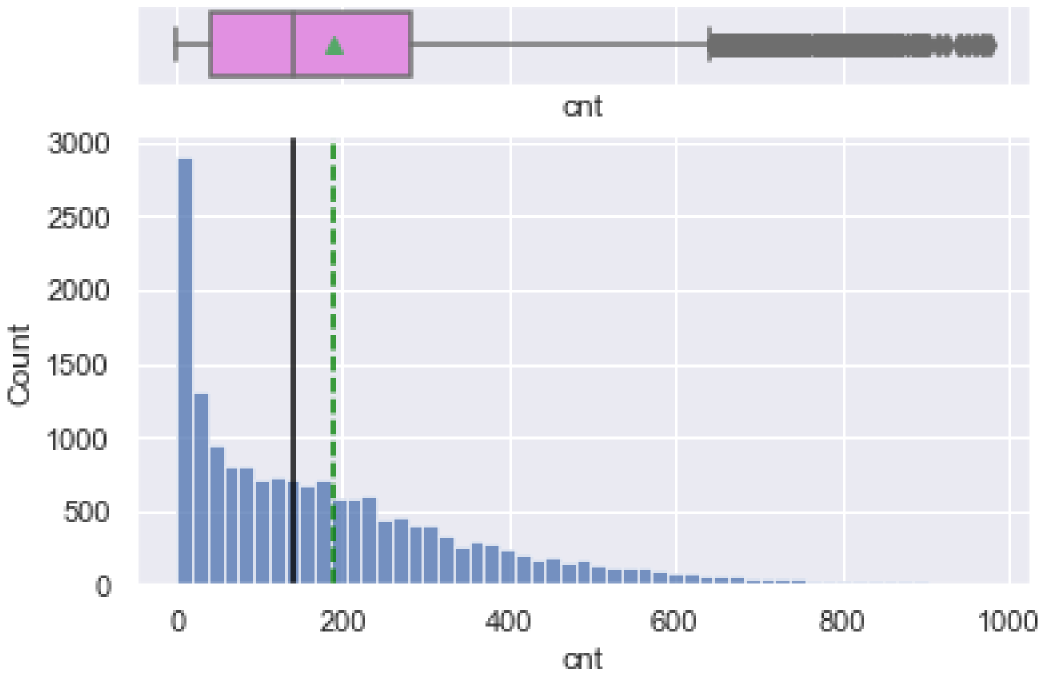
[11]:
```
<matplotlib.lines.Line2D at 0x7ffa4e323c40>
```



- Humidity observations lies between 0.4 to 0.9

[12]:
```
# set a grey background (use sns.set_theme() if seaborn version 0.11.0 or␣
↪above)
sns.set(style="darkgrid")

# creating a figure composed of two matplotlib.Axes objects (ax_box and ax_hist)
f, (ax_box, ax_hist) = plt.subplots(2, sharex=True,␣
↪gridspec_kw={"height_ratios": (.15, .85)})

# assigning a graph to each ax
sns.boxplot(data["cnt"], ax=ax_box, showmeans=True,color='violet')
sns.histplot(data=data, x="cnt", ax=ax_hist)
ax_hist.axvline(np.mean(data['cnt']), color='green', linestyle='--') # Add mean␣
↪to the histogram
ax_hist.axvline(np.median(data['cnt']), color='black', linestyle='-') # Add␣
↪median to the histogram
```

[12]: <matplotlib.lines.Line2D at 0x7ffa4e191370>



Counting top 5 highest values

[13]:
```
data['cnt'].nlargest()
```

[13]:
```
14773    977
14964    976
14748    970
14725    968
15084    967
Name: cnt, dtype: int64
```

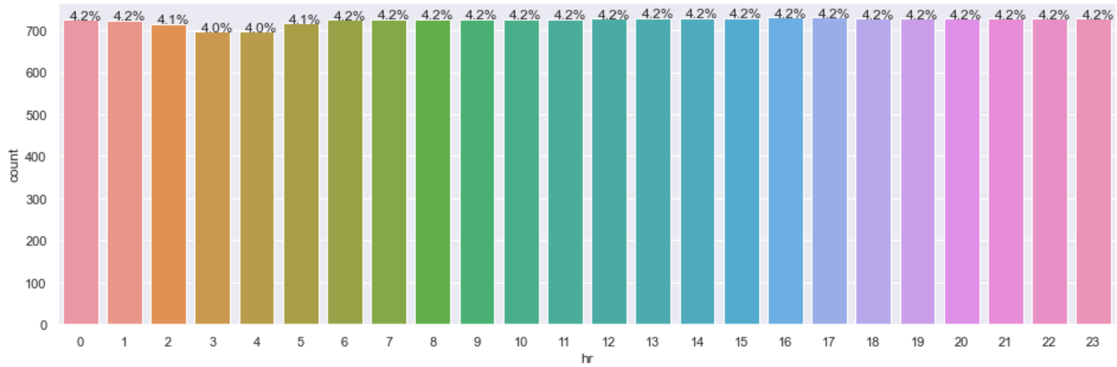
Function to create barplots that indicate percentage for each category

[14]:
```
def percentage_catgry(feature):
    #Creating a countplot for the feature
    sns.set(rc={'figure.figsize':(16,5)})
    ax=sns.countplot(x=feature, data=data)

    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percentage␣
 ↪of each class of the category
        x = p.get_x() + p.get_width() / 2 - 0.25 # width of the plot
        y = p.get_y() + p.get_height()           # hieght of the plot
        ax.annotate(percentage, (x, y), size = 12) # annotate the percantage
```

```
plt.show() # show the plot
```

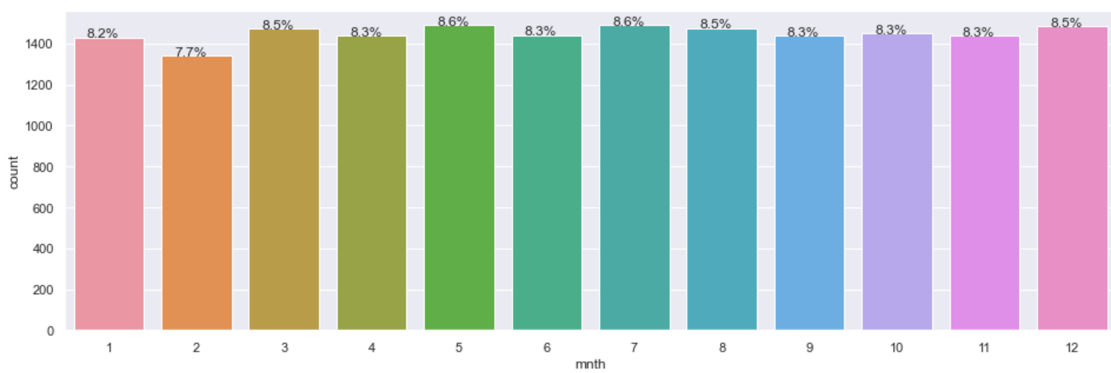
Each hour observation

[15]:
```
percentage_catgry(data['hr'])
```



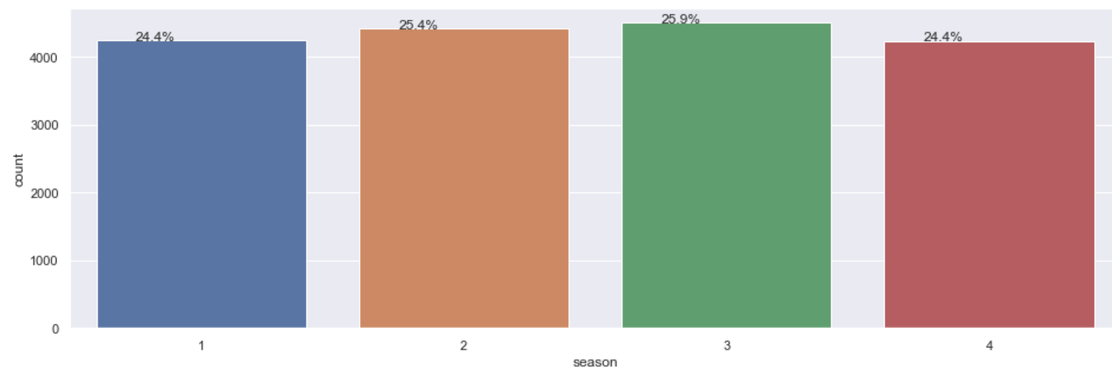
Monthly percentage of observation

[16]:
```
percentage_catgry(data['mnth'])
```



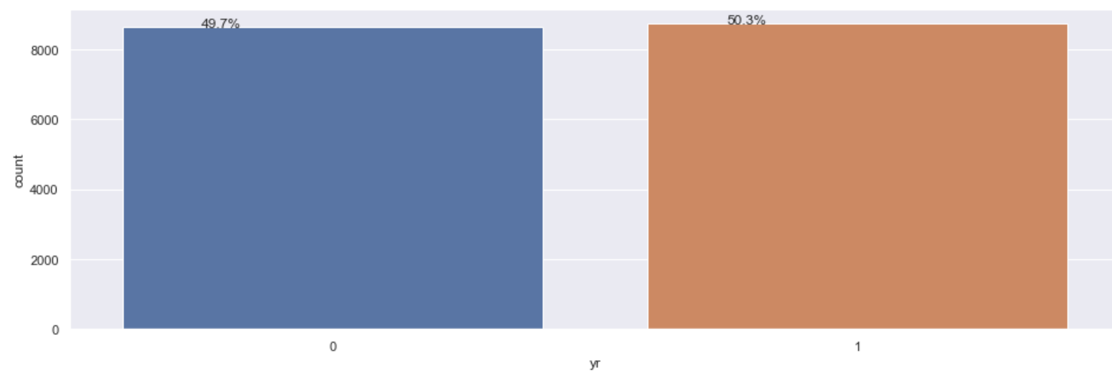
Season wise percentage of observation

[17]:
```
percentage_catgry(data['season'])
```

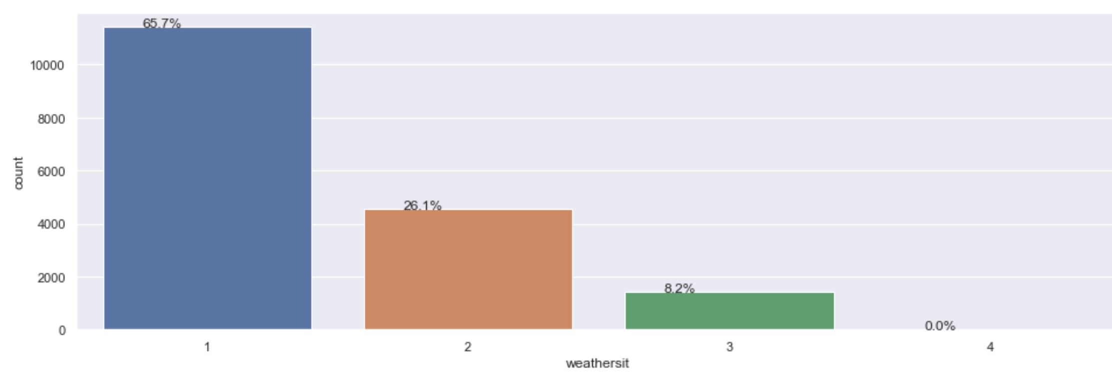


Each year percentage of observation

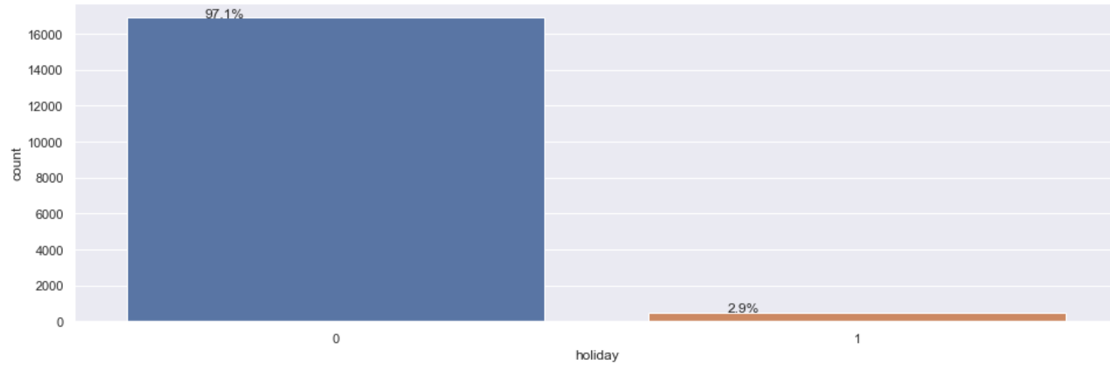
[18]:
```
percentage_catgry(data['yr'])
```



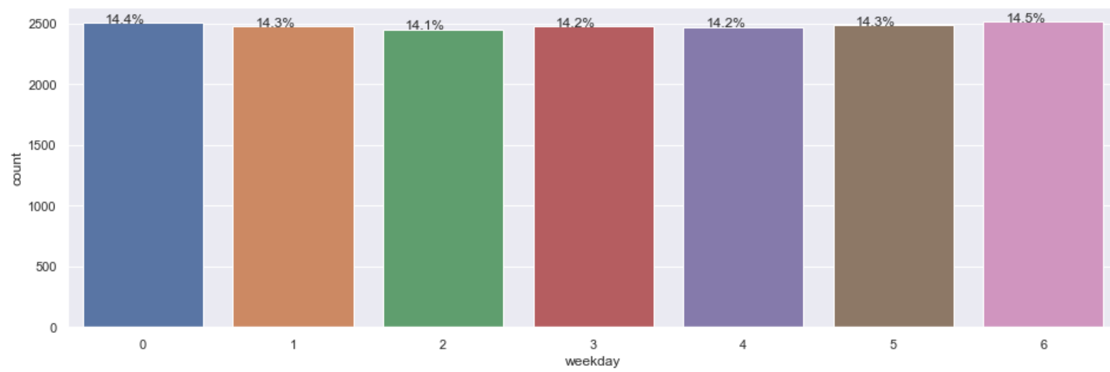
[19]:
```
percentage_catgry(data['weathersit'])
```



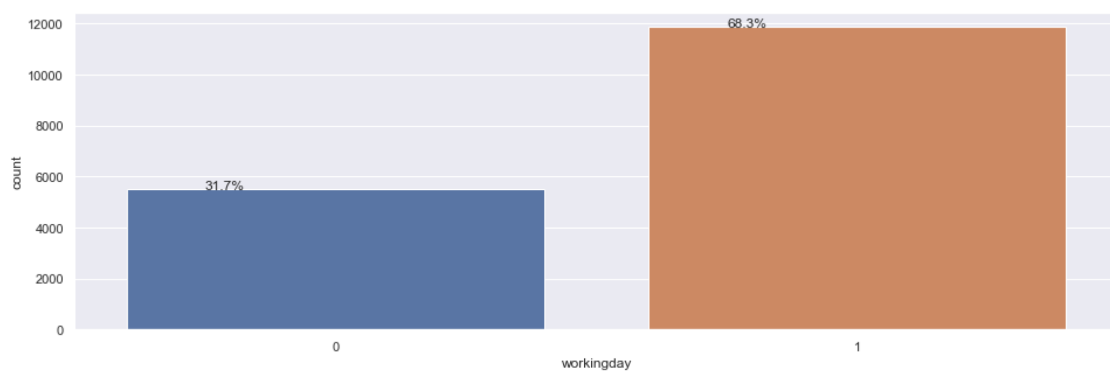
[20]:
```
percentage_catgry(data['holiday'])
```



[21]:
```
percentage_catgry(data['weekday'])
```

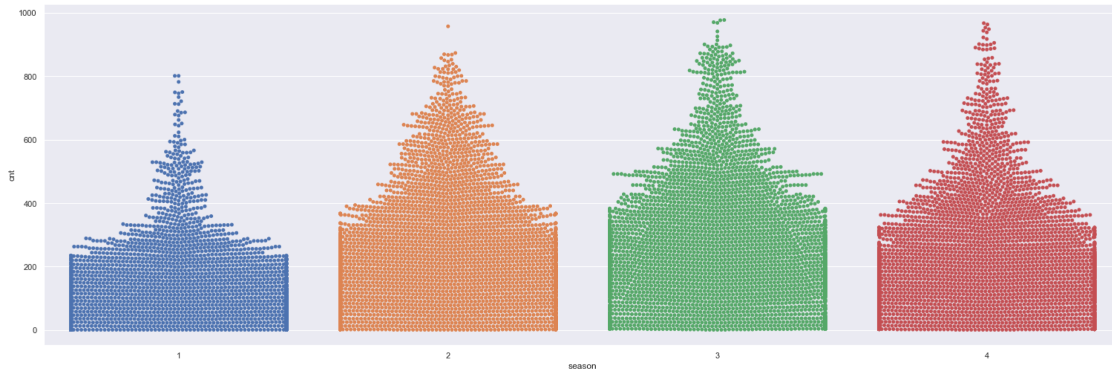


[22]:
```
percentage_catgry(data['workingday'])
```



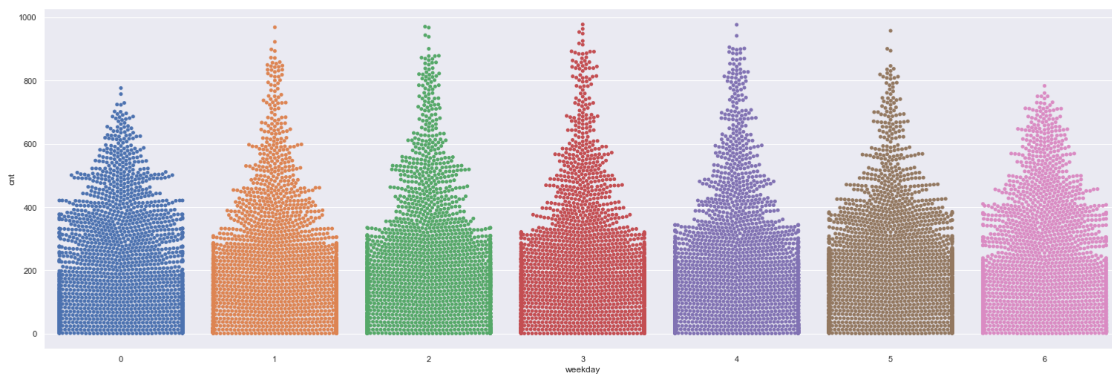
Bivariate Analysis

[23]:
```
sns.set(rc={'figure.figsize':(21,7)})
sns.catplot(x="season", y="cnt", kind="swarm", data=data, height=7, aspect=3);
```



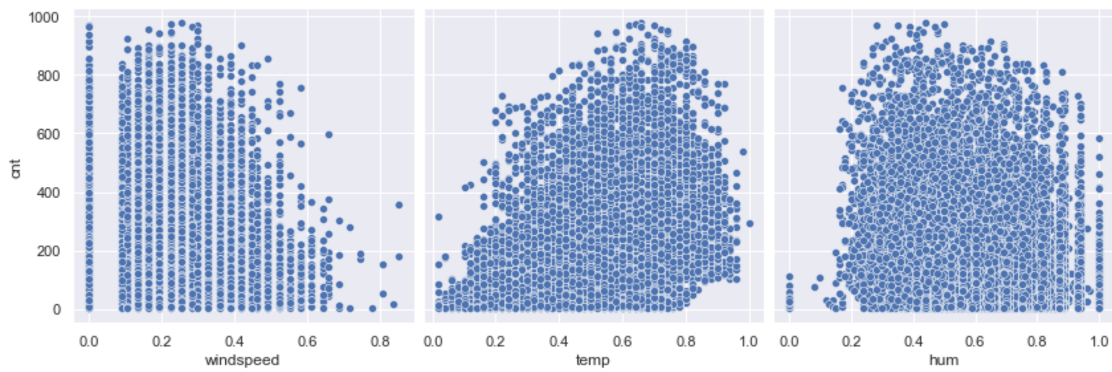
- The lowest number of bikes are rented in first season
- Highest number of bikes are shared in 3rd season

[24]:
```
sns.set(rc={'figure.figsize':(21,7)})
sns.catplot(x="weekday", y="cnt", kind="swarm", data=data, height=7, aspect=3);
```

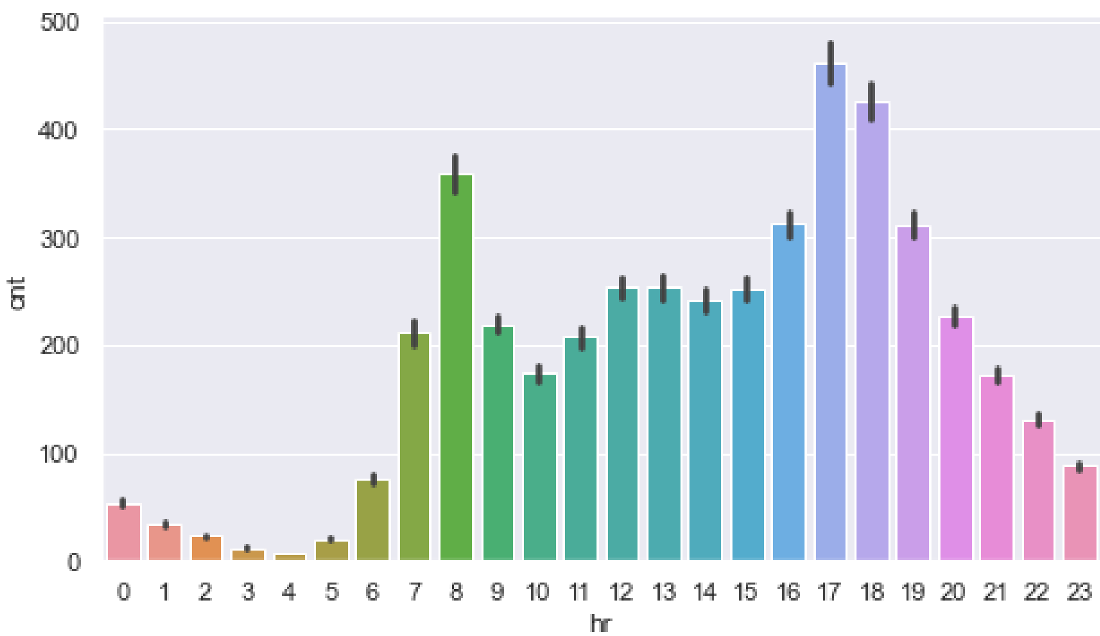


- Weekends i.e. weekday=0 and weekday=6 have low count of bikes rented and it is less varying.

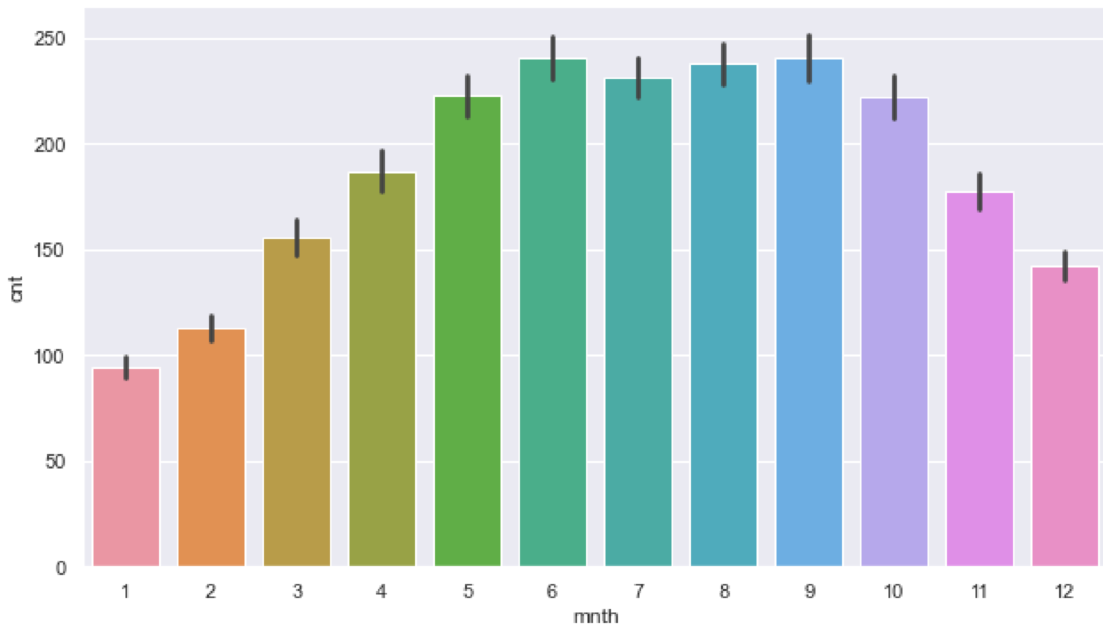
[25]:
```
sns.pairplot(
    data,
    x_vars=["windspeed", "temp", "hum"],
    y_vars=["cnt"],
    height=4,
    aspect=1
    );
```



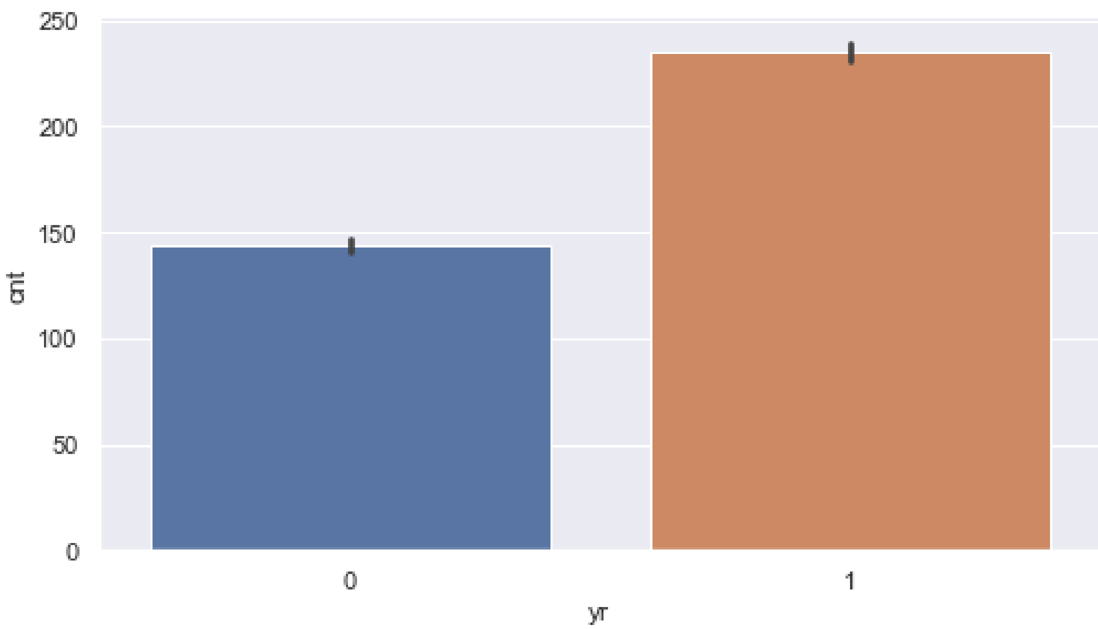
[26]:
```
sns.catplot(x="hr", y="cnt", data=data, kind='bar', height=4, aspect=1.72,␣
↪estimator=np.mean);
```



[27]:
```
sns.catplot(x="mnth", y="cnt", data=data, kind='bar', height=5, aspect=1.75,␣
↪estimator=np.mean);
```

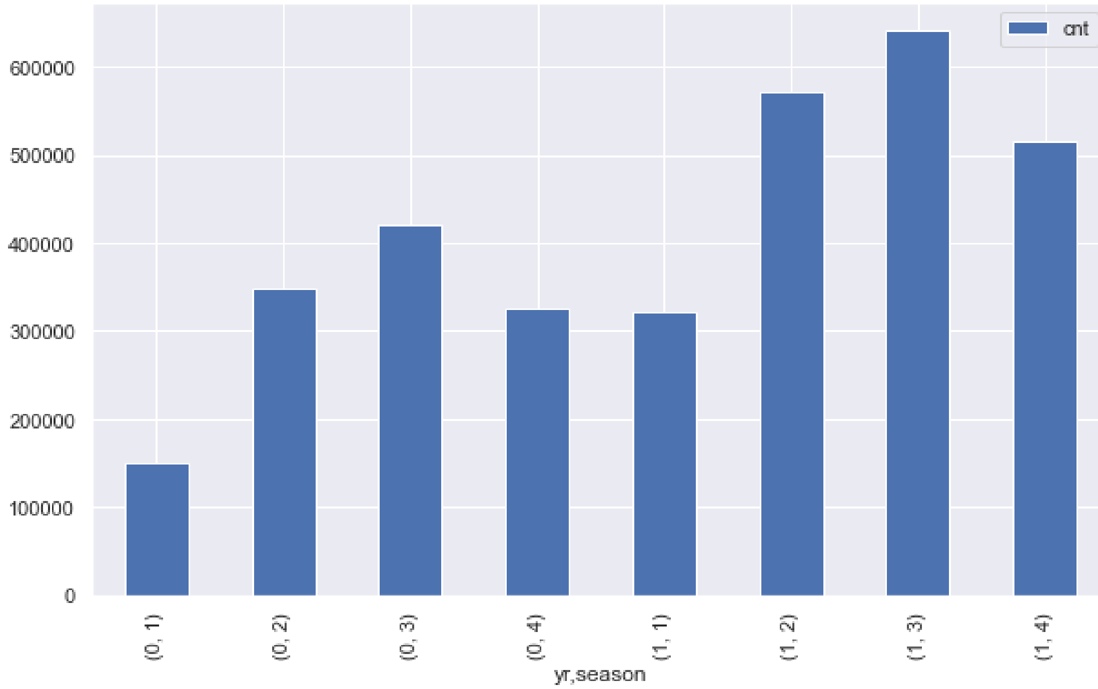


[28]:
```
sns.catplot(x="yr", y="cnt", data=data, kind='bar', size=4, aspect=1.75,␣
↪estimator=np.mean);
```



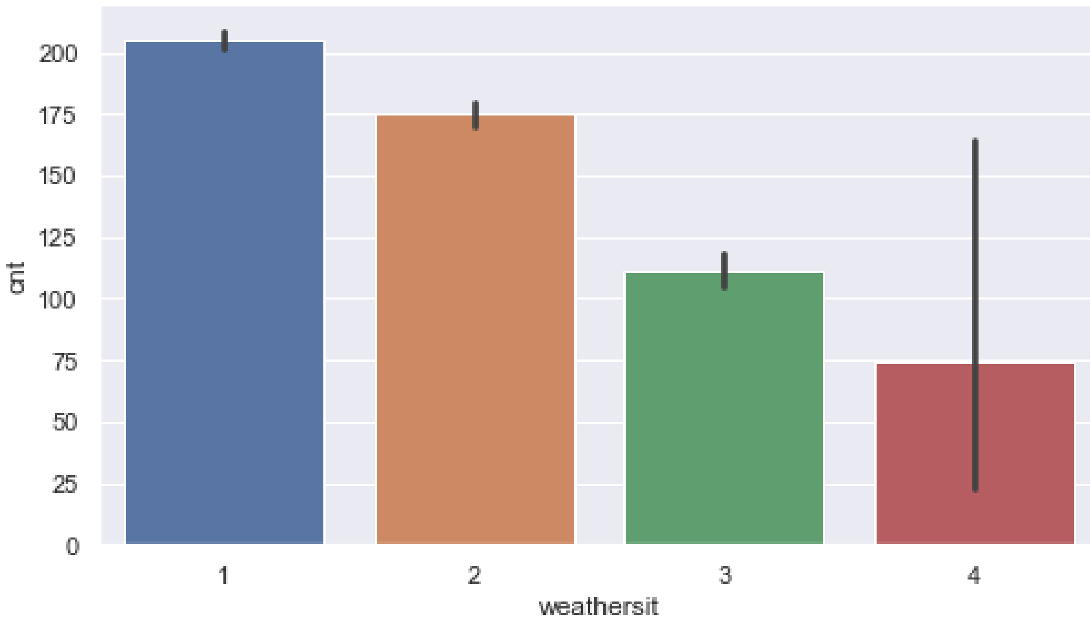
- Average count of bikes rented is high for year 2012 as compared to 2011.

[29]:
```
sns.set(rc={'figure.figsize':(10,6)})
pd.pivot_table(data=data, index=['yr', 'season'], values='cnt', aggfunc=np.sum).
↪plot(kind='bar');
```

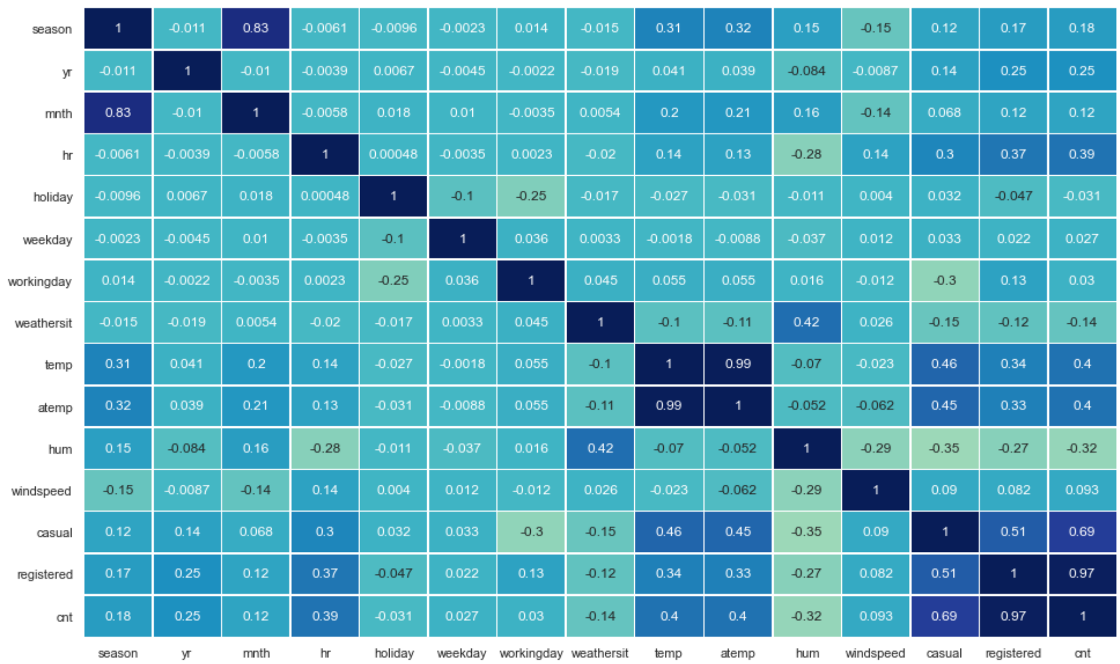


- We can see that number of bikes rented is higher in year 2012 for each season as compared to seasons in 2011.

[30]:
```
sns.catplot(x="weathersit", y='cnt', kind='bar', data=data, size=4, aspect=1.
↪75, estimator=np.mean);
```



[31]:
```
sns.set(rc={'figure.figsize':(16,10)})
sns.heatmap(data.corr(),
            annot=True,
            linewidths=.5,
            center=0,
            cbar=False,
            cmap="YlGnBu")
plt.show()
```



[32]:
```
#Dropping columns - casual and registered
data.drop(columns=['casual','registered'], inplace=True)
```

Split the datasets

[33]:
```
# Separating features and the target column
X = data.drop('cnt', axis=1)
y = data['cnt']
```

[34]:
```
# Splitting the data into train and test sets in 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=1, shuffle=True)
```

[35]:
```
X_train.shape, X_test.shape
```

[35]: ((12165, 12), (5214, 12))

Building Models

[36]:
```
##  Function to calculate r2_score and RMSE on train and test data
def get_model_score(model, flag=True):

    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(X_train)
```

```
    pred_test = model.predict(X_test)

    train_r2=metrics.r2_score(y_train,pred_train)
    test_r2=metrics.r2_score(y_test,pred_test)
    train_rmse=np.sqrt(metrics.mean_squared_error(y_train,pred_train))
    test_rmse=np.sqrt(metrics.mean_squared_error(y_test,pred_test))

    #Adding all scores in the list
    score_list.extend((train_r2,test_r2,train_rmse,test_rmse))

    # If the flag is set to True then only the following print statements will␣
↪be dispayed, the default value is True
    if flag==True:
        print("R-sqaure on training set : ",metrics.
↪r2_score(y_train,pred_train))
        print("R-square on test set : ",metrics.r2_score(y_test,pred_test))
        print("RMSE on training set : ",np.sqrt(metrics.
↪mean_squared_error(y_train,pred_train)))
        print("RMSE on test set : ",np.sqrt(metrics.
↪mean_squared_error(y_test,pred_test)))

    # returning the list with train and test scores
    return score_list
```

Decision Tree Model

[37]:
```
dtree=DecisionTreeRegressor(random_state=1)
dtree.fit(X_train,y_train)
```

[37]:
```
DecisionTreeRegressor(random_state=1)
```

[38]:
```
dtree_score=get_model_score(dtree)
```

```
R-sqaure on training set :  0.9999939364265239
R-square on test set :  0.8922273749987013
RMSE on training set :  0.4424086819235673
RMSE on test set :  60.82783340261016
```

- Decision tree model with default parameters is overfitting the train data.
- Let's see if we can reduce overfitting and improve performance on test data by tuning hyperparameters.

Hyper Parameter Tuning

[39]:
```
# Choose the type of classifier.
dtree_tuned = DecisionTreeRegressor(random_state=1)
```

```python
# Grid of parameters to choose from
parameters = {'max_depth': list(np.arange(2,20)) + [None],
              'min_samples_leaf': [1, 3, 5, 7, 10],
              'max_leaf_nodes' : [2, 3, 5, 10, 15] + [None],
              'min_impurity_decrease': [0.001, 0.01, 0.1, 0.0]
             }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.r2_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
dtree_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_tuned.fit(X_train, y_train)
```

[39]:
```
DecisionTreeRegressor(max_depth=14, min_impurity_decrease=0.1,
                      min_samples_leaf=5, random_state=1)
```

[40]:
```python
dtree_tuned_score=get_model_score(dtree_tuned)
```

```
R-sqaure on training set :  0.9588561760392927
R-square on test set :  0.9119854311073714
RMSE on training set :  36.44279225816553
RMSE on test set :  54.96995739195947
```

- The overfitting is reduced after hyperparameter tuning and test score has increased by approx 2%.
- RMSE is also reduced on test data and the model is generalizing better than the decision tree model with default parameters.

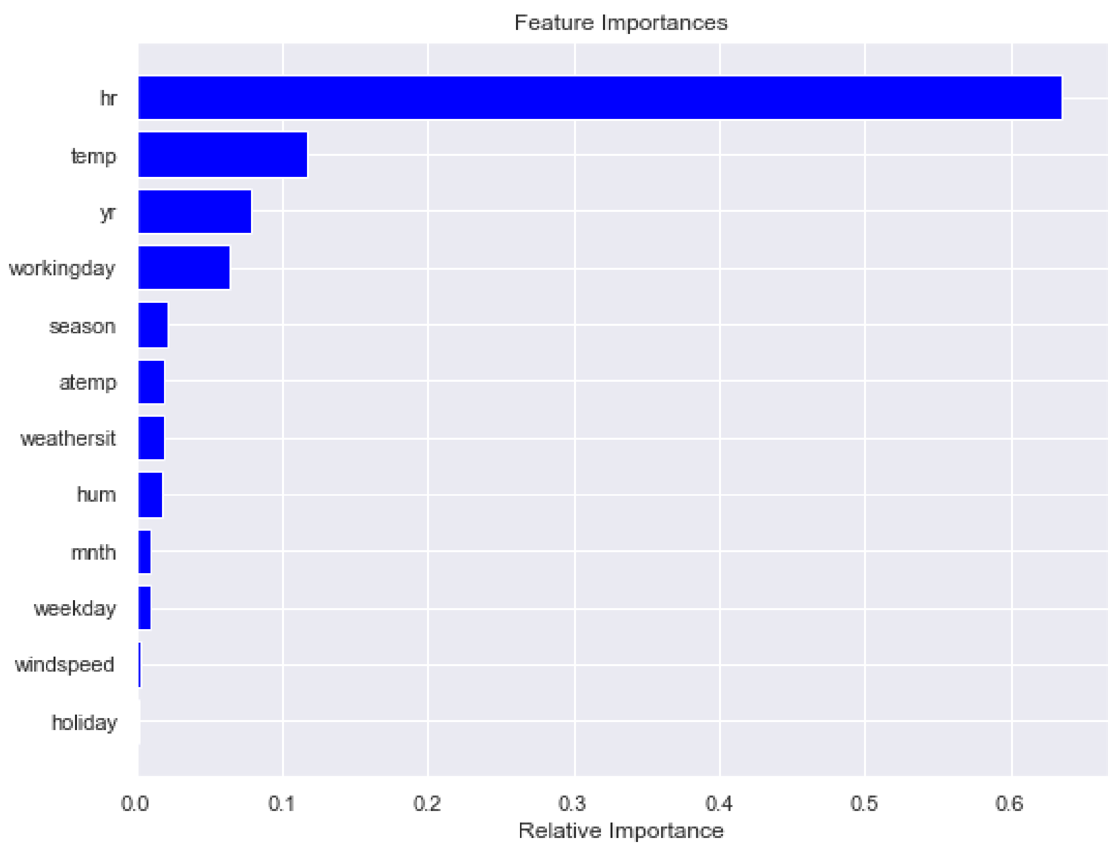
Plotting the feature importance of each variable

[41]:
```python
# importance of features in the tree building ( The importance of a feature is␣
↪computed as the
#(normalized) total reduction of the criterion brought by that feature. It is␣
↪also known as the Gini importance )

print(pd.DataFrame(dtree_tuned.feature_importances_, columns = ["Imp"], index =␣
↪X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
              Imp
hr       0.634942
temp     0.117441
yr       0.079271
```

```
workingday   0.063920
season       0.022164
atemp        0.019463
weathersit   0.019460
hum          0.017227
mnth         0.010498
weekday      0.010009
windspeed    0.003882
holiday      0.001724
```

[42]:
```python
feature_names = X_train.columns
importances = dtree_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(9,7))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='blue',␣
 ↪align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Random Forest Model

[43]:
```
rf_estimator=RandomForestRegressor(random_state=1)
rf_estimator.fit(X_train,y_train)
```

[43]:
```
RandomForestRegressor(random_state=1)
```

[44]:
```
rf_estimator_score=get_model_score(rf_estimator)
```

```
R-sqaure on training set :  0.9919022783128177
R-square on test set :  0.9421589674961293
RMSE on training set :  16.167420793032345
RMSE on test set :  44.56214995223083
```

- Random forest is giving good r2 score of 94% on the test data but it is slightly overfitting the train data.
- Let's try to reduce this overfitting by hyperparameter tuning.

Hyperparameter Tuning

[45]:
```
# Choose the type of classifier.
rf_tuned = RandomForestRegressor(random_state=1)

# Grid of parameters to choose from
parameters = {
                'max_depth':[4, 6, 8, 10, None],
                'max_features': ['sqrt','log2',None],
                'n_estimators': [80, 90, 100, 110, 120]
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.r2_score)

# Run the grid search
grid_obj = GridSearchCV(rf_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
rf_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_tuned.fit(X_train, y_train)
```

[45]:
```
RandomForestRegressor(max_features=None, n_estimators=120, random_state=1)
```

[46]:
```
rf_tuned_score=get_model_score(rf_tuned)
```

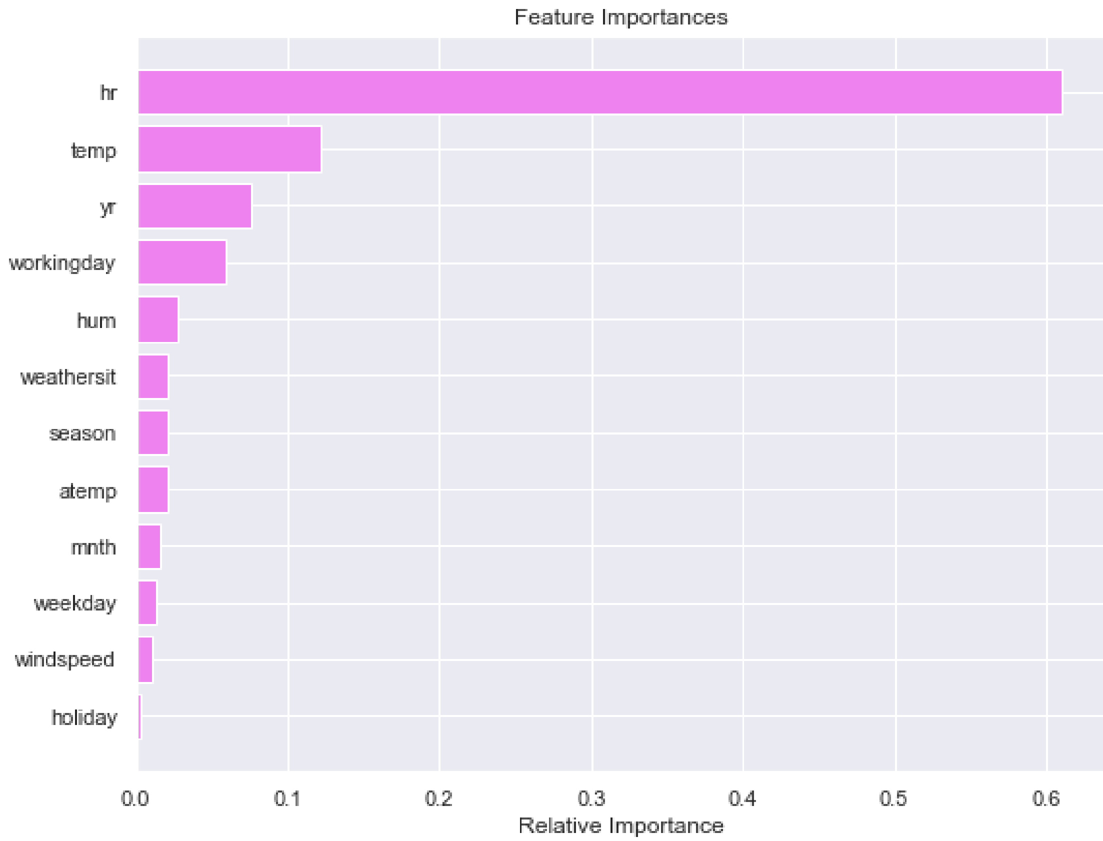
```
R-sqaure on training set :  0.9919096176193417
R-square on test set :  0.9421110621417824
RMSE on training set :  16.16009252467567
RMSE on test set :  44.58059986277221
```

- No significant change in the result. The result is almost same before or after the hyperparameter tuning.

[47]:
```
# importance of features in the tree building ( The importance of a feature is␣
↪computed as the
#(normalized) total reduction of the criterion brought by that feature. It is␣
↪also known as the Gini importance )

print(pd.DataFrame(rf_tuned.feature_importances_, columns = ["Imp"], index =␣
↪X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
                 Imp
hr          0.610116
temp        0.121773
yr          0.076295
workingday  0.059489
hum         0.026844
weathersit  0.020962
season      0.020876
atemp       0.020670
mnth        0.016153
weekday     0.013539
windspeed   0.010510
holiday     0.002774
```

[48]:
```
feature_names = X_train.columns
importances = rf_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(9,7))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet',␣
↪align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Boosting Models

AdaBoost Regressor

[49]:
```
ab_regressor=AdaBoostRegressor(random_state=1)
ab_regressor.fit(X_train,y_train)
```

[49]:
```
AdaBoostRegressor(random_state=1)
```

[50]:
```
ab_regressor_score=get_model_score(ab_regressor)
```

```
R-sqaure on training set :  0.6620671450557589
R-square on test set :  0.6763209472188219
RMSE on training set :  104.44184314767001
RMSE on test set :  105.41572853656164
```

- AdaBoost is generalizing well but it is giving poor performance, in terms of r2 score as well as RMSE, as compared to decision tree and random forest model.

Hyperparameter Tuning

[51]:
```python
# Choose the type of classifier.
ab_tuned = AdaBoostRegressor(random_state=1)

# Grid of parameters to choose from
parameters = {'n_estimators': np.arange(10,100,10),
              'learning_rate': [1, 0.1, 0.5, 0.01],
              }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.r2_score)

# Run the grid search
grid_obj = GridSearchCV(ab_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
ab_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
ab_tuned.fit(X_train, y_train)
```

[51]:
```
AdaBoostRegressor(learning_rate=1, n_estimators=30, random_state=1)
```

[52]:
```python
ab_tuned_score=get_model_score(ab_tuned)
```

```
R-sqaure on training set :  0.669247006578227
R-square on test set :  0.6823432190426465
RMSE on training set :  103.32637908778995
RMSE on test set :  104.43045797379186
```

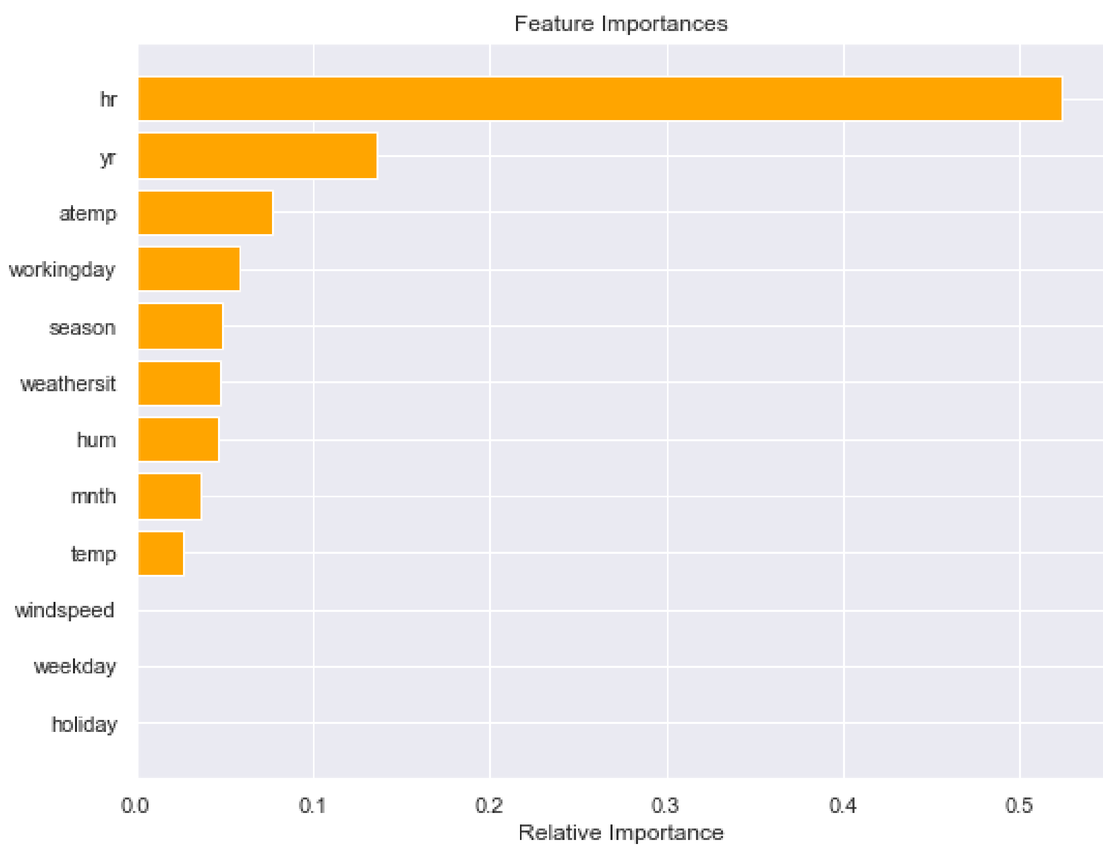
- We can see that there is no significant improvement in the model after hyperparameter tuning.

[53]:
```python
# importance of features in the tree building

print(pd.DataFrame(ab_tuned.feature_importances_, columns = ["Imp"], index =␣
↪X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
                 Imp
hr          0.523674
yr          0.135837
atemp       0.076841
workingday  0.058274
season      0.049087
weathersit  0.047040
hum         0.046359
mnth        0.036370
temp        0.026518
holiday     0.000000
```

```
weekday      0.000000
windspeed    0.000000
```

[54]:
```
feature_names = X_train.columns
importances = ab_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(9,7))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='orange',␣
 ↪align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Gradient Boosting Regressor

[55]:
```
gb_estimator=GradientBoostingRegressor(random_state=1)
gb_estimator.fit(X_train,y_train)
```

[55]: GradientBoostingRegressor(random_state=1)

[56]:
```
gb_estimator_score=get_model_score(gb_estimator)
```

```
R-sqaure on training set :  0.8399586813152043
R-square on test set :  0.8397497482833491
RMSE on training set :  71.8745898418818
RMSE on test set :  74.17327883963448
```

- Gradient boosting is generalizing well and giving decent results but not as good as random forest.

Hyperparameter Tuning

[57]:
```
# Choose the type of classifier.
gb_tuned = GradientBoostingRegressor(random_state=1)

# Grid of parameters to choose from
parameters = {'n_estimators': np.arange(50,200,25),
              'subsample':[0.7,0.8,0.9,1],
              'max_features':[0.7,0.8,0.9,1],
              'max_depth':[3,5,7,10]
              }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.r2_score)

# Run the grid search
grid_obj = GridSearchCV(gb_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
gb_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
gb_tuned.fit(X_train, y_train)
```

[57]:
```
GradientBoostingRegressor(max_depth=7, max_features=0.9, n_estimators=175,
                          random_state=1, subsample=0.7)
```

[58]:
```
gb_tuned_score=get_model_score(gb_tuned)
```

```
R-sqaure on training set :  0.9846321402838036
R-square on test set :  0.955523582334073
RMSE on training set :  22.272348450912293
RMSE on test set :  39.07626239961316
```

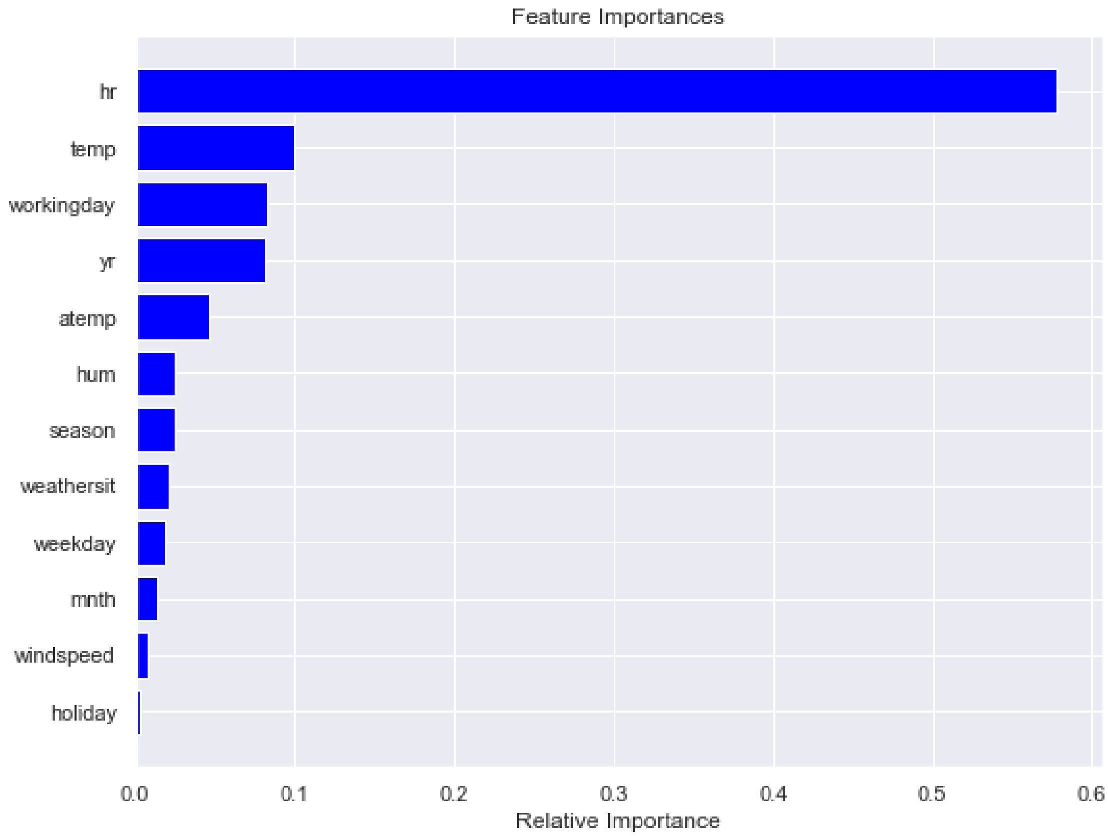
- We can see that the model has improved significantly in terms of r2 score and RMSE.
- The r2 score has increase by approx 12% on the test data.

- RMSE has decreased by more than 30 for the test data.

[59]:
```
# importance of features in the tree building ( The importance of a feature is␣
↪computed as the
#(normalized) total reduction of the criterion brought by that feature. It is␣
↪also known as the Gini importance )

print(pd.DataFrame(gb_tuned.feature_importances_, columns = ["Imp"], index =␣
↪X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
                 Imp
hr          0.577712
temp        0.099255
workingday  0.083006
yr          0.081833
atemp       0.046505
hum         0.024214
season      0.024071
weathersit  0.020767
weekday     0.018315
mnth        0.013716
windspeed   0.007647
holiday     0.002959
```

[60]:
```
feature_names = X_train.columns
importances = gb_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(9,7))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='blue',␣
↪align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



XGBoost Regressor

[61]:
```
xgb_estimator=XGBRegressor(random_state=1)
xgb_estimator.fit(X_train,y_train)
```

[61]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=1,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[62]:
```
xgb_estimator_score=get_model_score(xgb_estimator)
```

```
R-sqaure on training set :  0.9770545660089234
R-square on test set :  0.9501815995087658
RMSE on training set :  27.21494474999754
RMSE on test set :  41.356426656666834
```

Hyperparameter Tuning

[63]:
```
# Choose the type of classifier.
xgb_tuned = XGBRegressor(random_state=1)

# Grid of parameters to choose from
parameters = {'n_estimators': [75,100,125,150],
              'subsample':[0.7, 0.8, 0.9, 1],
              'gamma':[0, 1, 3, 5],
              'colsample_bytree':[0.7, 0.8, 0.9, 1],
              'colsample_bylevel':[0.7, 0.8, 0.9, 1]
              }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.r2_score)

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
xgb_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb_tuned.fit(X_train, y_train)
```

[63]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
             colsample_bynode=1, colsample_bytree=0.8, gamma=3, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=150, n_jobs=8, num_parallel_tree=1, random_state=1,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.9,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[64]:
```
xgb_tuned_score=get_model_score(xgb_tuned)
```

```
R-sqaure on training set :  0.9830195356300606
R-square on test set :  0.9483412326720347
RMSE on training set :  23.411761984705524
RMSE on test set :  42.113383651912415
```

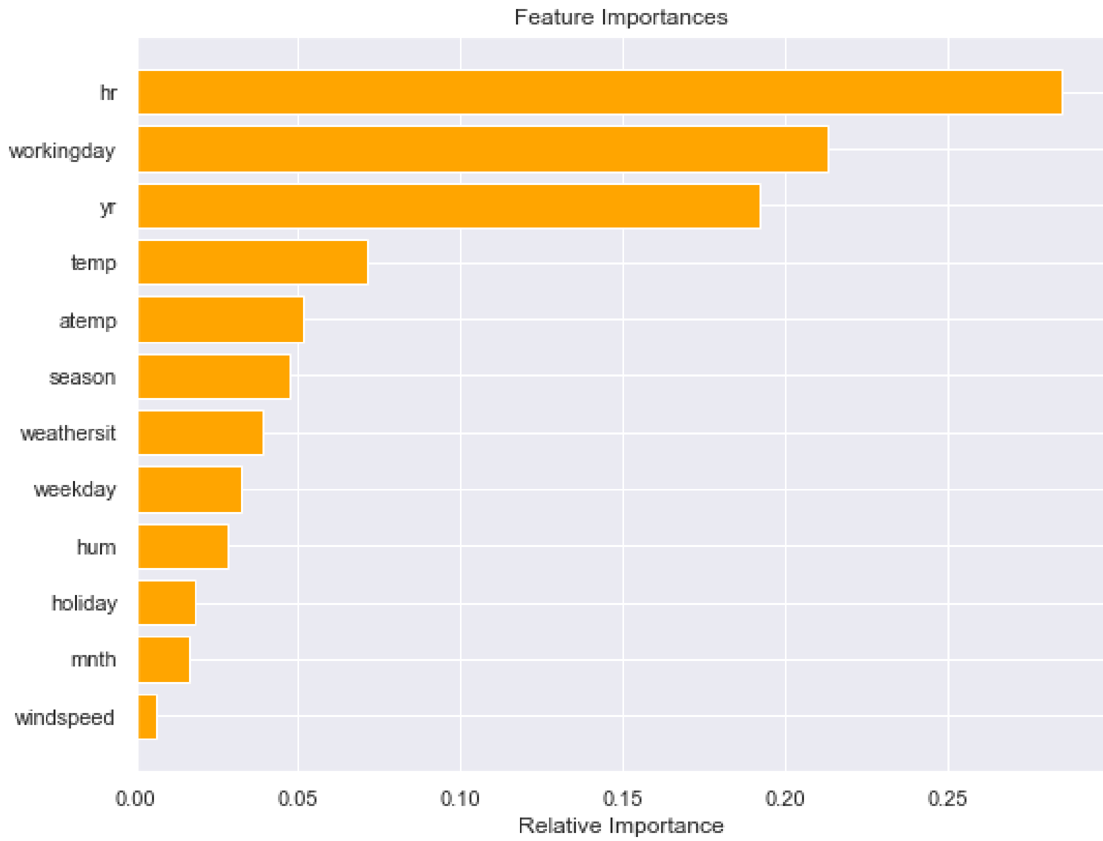
[65]:
```
# importance of features in the tree building ( The importance of a feature is␣
↪computed as the
#(normalized) total reduction of the criterion brought by that feature. It is␣
↪also known as the Gini importance )
```

```python
print(pd.DataFrame(xgb_tuned.feature_importances_, columns = ["Imp"], index =
    X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
                 Imp
hr          0.284964
workingday  0.212911
yr          0.191848
temp        0.070928
atemp       0.051288
season      0.047408
weathersit  0.039195
weekday     0.032438
hum         0.028440
holiday     0.017976
mnth        0.016521
windspeed   0.006084
```

[66]:

```python
feature_names = X_train.columns
importances = xgb_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(9,7))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='orange',
    align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Stacking Model Now, let's build a stacking model with the tuned models - decision tree, random forest and gradient boosting, then use XGBoost to get the final prediction.

[67]:
```
estimators=[('Decision Tree', dtree_tuned),('Random Forest', rf_tuned),
            ('Gradient Boosting', gb_tuned)]
final_estimator=XGBRegressor(random_state=1)
```

[68]:
```
stacking_estimator=StackingRegressor(estimators=estimators,␣
 ↪final_estimator=final_estimator,cv=5)
stacking_estimator.fit(X_train,y_train)
```

[68]:
```
StackingRegressor(cv=5,
                  estimators=[('Decision Tree',
                               DecisionTreeRegressor(max_depth=14,
                                                     min_impurity_decrease=0.1,
                                                     min_samples_leaf=5,
                                                     random_state=1)),
                              ('Random Forest',
                               RandomForestRegressor(max_features=None,
```

```
                                            n_estimators=120,
                                            random_state=1)),
                               ('Gradient Boosting',
                                GradientBoostingRegressor(max_depth=7,
                                                          max_features=0.9,
                                                          n_estimators=175,
                                                          random_state=1,
                                                          subsa…
                                                  importance_type='gain',
                                                  interaction_constraints=None,
                                                  learning_rate=None,
                                                  max_delta_step=None,
                                                  max_depth=None,
                                                  min_child_weight=None,
                                                  missing=nan,
                                                  monotone_constraints=None,
                                                  n_estimators=100, n_jobs=None,
                                                  num_parallel_tree=None,
                                                  random_state=1, reg_alpha=None,
                                                  reg_lambda=None,
                                                  scale_pos_weight=None,
                                                  subsample=None, tree_method=None,
                                                  validate_parameters=None,
                                                  verbosity=None))
```

[69]:
```python
stacking_estimator_score=get_model_score(stacking_estimator)
```

```
R-sqaure on training set :  0.9832203807316542
R-square on test set :  0.9500427975861883
RMSE on training set :  23.272892849941474
RMSE on test set :  41.41399934721364
```

Comparing all Models

[70]:
```python
# defining list of models
models = [dtree, dtree_tuned, rf_estimator, rf_tuned, ab_regressor, ab_tuned,␣
 ↪gb_estimator, gb_tuned, xgb_estimator,
         xgb_tuned, stacking_estimator]

# defining empty lists to add train and test results
r2_train = []
r2_test = []
rmse_train= []
rmse_test= []

# looping through all the models to get the rmse and r2 scores
for model in models:
```

```
    # accuracy score
    j = get_model_score(model,False)
    r2_train.append(j[0])
    r2_test.append(j[1])
    rmse_train.append(j[2])
    rmse_test.append(j[3])
```

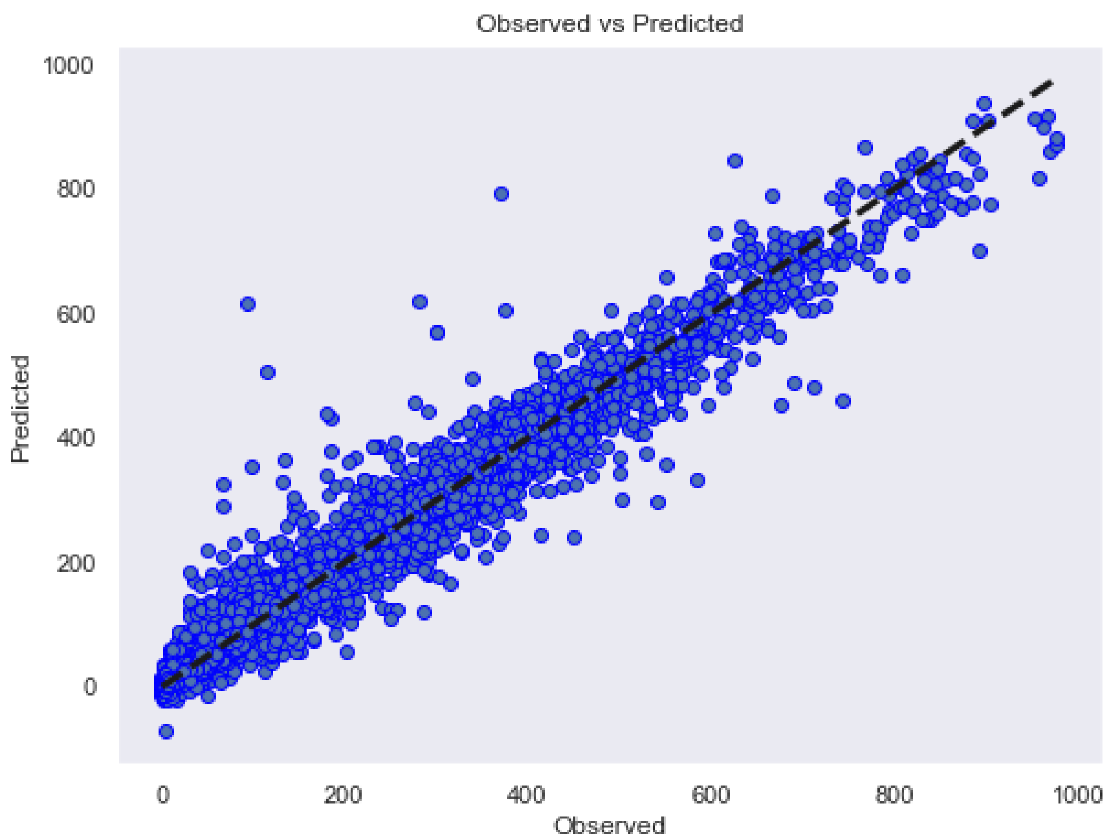
[71]:
```
comparison_frame = pd.DataFrame({'Model':['Decision Tree','Tuned Decision␣
↪Tree','Random Forest','Tuned Random Forest',
                                          'AdaBoost Regressor', 'Tuned AdaBoost␣
↪Regressor',
                                          'Gradient Boosting Regressor', 'Tuned␣
↪Gradient Boosting Regressor',
                                          'XGBoost Regressor',  'Tuned XGBoost␣
↪Regressor','Stacking Regressor'],
                                          'Train_r2': r2_train,'Test_r2':␣
↪r2_test,
                                          'Train_RMSE':rmse_train,'Test_RMSE':
↪rmse_test})
comparison_frame
```

[71]:

	Model	Train_r2	Test_r2	Train_RMSE	Test_RMSE
0	Decision Tree	0.999994	0.892227	0.442409	60.827833
1	Tuned Decision Tree	0.958856	0.911985	36.442792	54.969957
2	Random Forest	0.991902	0.942159	16.167421	44.562150
3	Tuned Random Forest	0.991910	0.942111	16.160093	44.580600
4	AdaBoost Regressor	0.662067	0.676321	104.441843	105.415729
5	Tuned AdaBoost Regressor	0.669247	0.682343	103.326379	104.430458
6	Gradient Boosting Regressor	0.839959	0.839750	71.874590	74.173279
7	Tuned Gradient Boosting Regressor	0.984632	0.955524	22.272348	39.076262
8	XGBoost Regressor	0.977055	0.950182	27.214945	41.356427
9	Tuned XGBoost Regressor	0.983020	0.948341	23.411762	42.113384
10	Stacking Regressor	0.983220	0.950043	23.272893	41.413999

- Tuned gradient boosting model is the best model here. It has highest r2 score of approx 95.5% and lowest RMSE of approx 39 on the test data.
- Gradient boosting, xgboost and stacking regressor are the top 3 models. They are all giving similar performance.

[72]:
```
# So plot observed and predicted values of the test data for the best model i.e.
↪ tuned gradient boosting model
fig, ax = plt.subplots(figsize=(8, 6))
y_pred=gb_tuned.predict(X_test)
ax.scatter(y_test, y_pred, edgecolors=(0, 0, 1))
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
ax.set_xlabel('Observed')
ax.set_ylabel('Predicted')
ax.set_title("Observed vs Predicted")
plt.grid()
plt.show()
```



Saving the model results

[79]:
```
comparison_frame.to_csv('/home/jayanthikishore/Downloads/ML_classwork/
↪DT_RF_Ensemble/dt_rf_modelsres.csv')
```

Highlighting the results with different colors

- The above 0.7 r2 values are highlighting as a green
- The above 70 rmse values are highlighting as a red

[80]:
```
def r2_highlight(val):

    if val < 0.4:
        color = 'red'
    elif val > 0.7:
        color = 'green'
    else:
        color = 'black'

    return 'color: %s' %color
```

[81]:
```
def rmse_color(valu):

    if valu > 80.:
        color = 'red'
    else:
        color = 'black'
    return 'color: %s' %color
```

[82]:
```
dff1 = comparison_frame.style.applymap(r2_highlight,␣
↪subset=['Train_r2','Test_r2'])
dff2 = dff1.applymap(rmse_color, subset=['Train_RMSE','Test_RMSE'])
dff2
```

[82]:
```
<pandas.io.formats.style.Styler at 0x7ffa3a3e4c70>
```

[ ]:
```

```