In [1]:
```
import pandas as pd
df = pd.read_csv(r"R:\datasets for analysis\kaggle data\beginner datasets\beginner_datasets\bike.csv")
```

In [2]:
```
df.head(15)
```

Out[2]:
```
    instant    dteday  season  yr  mnth  hr  holiday  weekday  workingday  weathersit  temp   atemp   hum  windspeed  cnt
 0        1  1/1/2011       1   0     1   0        0        6           0           1  0.24  0.2879  0.81     0.0000   16
 1        2  1/1/2011       1   0     1   1        0        6           0           1  0.22  0.2727  0.80     0.0000   40
 2        3  1/1/2011       1   0     1   2        0        6           0           1  0.22  0.2727  0.80     0.0000   32
 3        4  1/1/2011       1   0     1   3        0        6           0           1  0.24  0.2879  0.75     0.0000   13
 4        5  1/1/2011       1   0     1   4        0        6           0           1  0.24  0.2879  0.75     0.0000    1
 5        6  1/1/2011       1   0     1   5        0        6           0           2  0.24  0.2576  0.75     0.0896    1
 6        7  1/1/2011       1   0     1   6        0        6           0           1  0.22  0.2727  0.80     0.0000    2
 7        8  1/1/2011       1   0     1   7        0        6           0           1  0.20  0.2576  0.86     0.0000    3
 8        9  1/1/2011       1   0     1   8        0        6           0           1  0.24  0.2879  0.75     0.0000    8
 9       10  1/1/2011       1   0     1   9        0        6           0           1  0.32  0.3485  0.76     0.0000   14
10       11  1/1/2011       1   0     1  10        0        6           0           1  0.38  0.3939  0.76     0.2537   36
11       12  1/1/2011       1   0     1  11        0        6           0           1  0.36  0.3333  0.81     0.2836   56
12       13  1/1/2011       1   0     1  12        0        6           0           1  0.42  0.4242  0.77     0.2836   84
13       14  1/1/2011       1   0     1  13        0        6           0           2  0.46  0.4545  0.72     0.2985   94
14       15  1/1/2011       1   0     1  14        0        6           0           2  0.46  0.4545  0.72     0.2836  106
```

In [3]:
```
import datetime as dt
```

In [4]:

```
df['dteday'] = pd.to_datetime(df['dteday'])
# day
df['Day'] = df['dteday'].dt.day
# month
df['Month'] = df['dteday'].dt.month
# year
df['Year'] = df['dteday'].dt.year
```

In [5]:

```
df.head()
```

Out[5]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt	Day	Month
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	16	1	1
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	40	1	1
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	32	1	1
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	13	1	1
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	1	1	1

In [6]:

```
data = df.copy()
```

In [7]:

```
X=data.drop(['dteday','instant','mnth','cnt'],axis = 1)
```

In [8]:

```
y = data.cnt
```

In [9]:

```
from sklearn.model_selection import train_test_split
```

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
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
cnt           0
Day           0
Month         0
Year          0
dtype: int64
```

In [11]:
```
data.describe()
```

Out[11]:

	instant	season	yr	mnth	hr	holiday	weekday	workingday	weathersi
count	17379.0000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.00000
mean	8690.0000	2.501640	0.502561	6.537775	11.546752	0.028770	3.003683	0.682721	1.42528
std	5017.0295	1.106918	0.500008	3.438776	6.914405	0.167165	2.005771	0.465431	0.63935
min	1.0000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.00000
25%	4345.5000	2.000000	0.000000	4.000000	6.000000	0.000000	1.000000	0.000000	1.00000
50%	8690.0000	3.000000	1.000000	7.000000	12.000000	0.000000	3.000000	1.000000	1.00000
75%	13034.5000	3.000000	1.000000	10.000000	18.000000	0.000000	5.000000	1.000000	2.00000
max	17379.0000	4.000000	1.000000	12.000000	23.000000	1.000000	6.000000	1.000000	4.00000

In [12]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
y = (y - y.mean())/ y.std()
```

In [13]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

In [14]:

```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors = 10, algorithm = 'brute')
knn.fit(X_train, y_train)
```

Out[14]:

```
KNeighborsRegressor(algorithm='brute', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                    weights='uniform')
```

In [15]:

```
print("Accuracy on training set: {:.2f}".format(knn.score(X_train,y_train)))
print("Accuracy on testing set: {:.2f}".format(knn.score(X_test,y_test)))
```

```
Accuracy on training set: 0.65
Accuracy on testing set: 0.58
```

In [16]:

```
for i in range(1,30,1):
    print(f"with {i*2} neighbors and {2*i} leaf size : ")
    knn = KNeighborsRegressor(n_neighbors = i*2, leaf_size = 2*i,
                              weights ='distance', algorithm = 'brute').fit(X_train,y_train)
    print(f"accuracy:{ knn.score(X_test,y_test)*100: 2f}%")
```

```
with 2 neighbors and 2 leaf size :
accuracy: 58.854180%
with 4 neighbors and 4 leaf size :
accuracy: 62.179988%
with 6 neighbors and 6 leaf size :
accuracy: 62.751724%
with 8 neighbors and 8 leaf size :
accuracy: 63.429995%
with 10 neighbors and 10 leaf size :
accuracy: 63.334330%
with 12 neighbors and 12 leaf size :
accuracy: 63.276748%
with 14 neighbors and 14 leaf size :
accuracy: 62.603158%
with 16 neighbors and 16 leaf size :
accuracy: 62.278956%
with 18 neighbors and 18 leaf size :
accuracy: 61.817685%
with 20 neighbors and 20 leaf size :
accuracy: 61.257682%
with 22 neighbors and 22 leaf size :
accuracy: 61.049969%
with 24 neighbors and 24 leaf size :
accuracy: 60.569107%
with 26 neighbors and 26 leaf size :
accuracy: 60.111930%
with 28 neighbors and 28 leaf size :
accuracy: 59.632712%
with 30 neighbors and 30 leaf size :
accuracy: 59.329908%
with 32 neighbors and 32 leaf size :
accuracy: 58.943147%
with 34 neighbors and 34 leaf size :
accuracy: 58.656539%
with 36 neighbors and 36 leaf size :
accuracy: 58.287103%
with 38 neighbors and 38 leaf size :
accuracy: 57.977584%
with 40 neighbors and 40 leaf size :
accuracy: 57.782101%
with 42 neighbors and 42 leaf size :
accuracy: 57.488333%
with 44 neighbors and 44 leaf size :
```

```
accuracy: 57.251755%
with 46 neighbors and 46 leaf size :
accuracy: 57.033172%
with 48 neighbors and 48 leaf size :
accuracy: 56.751186%
with 50 neighbors and 50 leaf size :
accuracy: 56.544418%
with 52 neighbors and 52 leaf size :
accuracy: 56.304261%
with 54 neighbors and 54 leaf size :
accuracy: 56.181814%
with 56 neighbors and 56 leaf size :
accuracy: 55.992454%
with 58 neighbors and 58 leaf size :
accuracy: 55.797652%
```

With KNeighborsRegressor with 8 neighbors and 8 leaf size the accuracy is 63.43% (highest)

In [17]:

```
from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators = 1000,learning_rate = 0.002, max_depth = 10)
xgb.fit(X_train, y_train)
```

Out[17]:

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.002, max_delta_step=0, max_depth=10,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

In [18]:

```
print("Accuracy on training set: {:.2f}".format(xgb.score(X_train,y_train)))
print("Accuracy on testing set: {:.2f}".format(xgb.score(X_test,y_test)))
```

```
Accuracy on training set: 0.93
Accuracy on testing set: 0.91
```

With XGBoost the accuracy is 91%