In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the Libraries-

numpy: NumPy is a open source project and python library which is used for working with arrays which also has a functions for working in domain of linear algebra and matrices. Numpy stands for numerical python.

pandas: It is made for data manipulation. using pandas you can directly load the csv, html, json, txt and other formats into python and handle it.

matplotlib.pyplot: It is a collection of command style functions that make matplotlib work like MATLAB.

seaborn: It is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Know your dataset

https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset#

In [2]:
```
data = pd.read_csv('./Dataset/day.csv', usecols=['season','holiday','weekday','workingd
data.tail()
```

Out[2]:

	season	holiday	weekday	workingday	cnt
726	1	0	4	1	2114
727	1	0	5	1	3095
728	1	0	6	0	1341
729	1	0	0	0	1796
730	1	0	1	1	2729

While importing the dataset, a function read_csv of Pandas library is called in which ./Dataset/day.csv file passed in the form of parameter to load the file ./Dataset/day.csv.

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

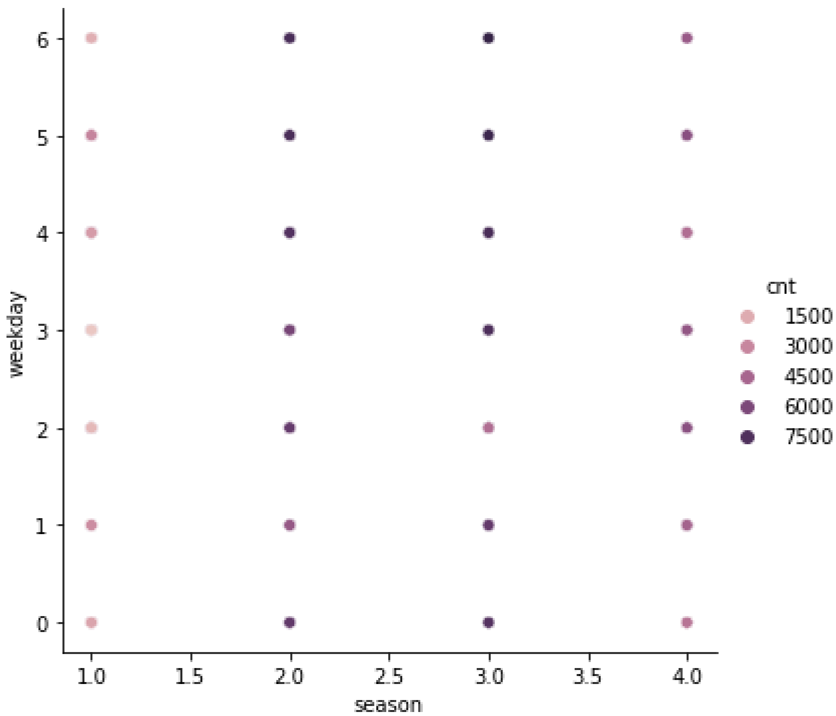
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
```

```
 0   season      731 non-null    int64
 1   holiday     731 non-null    int64
 2   weekday     731 non-null    int64
 3   workingday  731 non-null    int64
 4   cnt         731 non-null    int64
dtypes: int64(5)
memory usage: 28.7 KB
```

This line shows list of all the columns in your dataset and the type of data each column contains. This show the data types int64. Pandas uses the NumPy library to work with these types.

In [4]:

```
sns.relplot(
    x='season', y='weekday', hue='cnt', data=data)
plt.show()
```



It Shows relationship between season and weekday with graphical mapping.Assigning x and y and any semantic mapping variables will draw a single plot.While the points are plotted in two dimensions, another dimension can be added to the plot by coloring the points according to a third variable. In seaborn, this is referred to as using a “hue semantic”, because the color of the point gains the above meaning.

In [5]:

```
features = data.iloc[:,:-1]
features.head()
```

Out[5]:

	season	holiday	weekday	workingday
0	1	0	6	0
1	1	0	0	0
2	1	0	1	1
3	1	0	2	1
4	1	0	3	1

from this line we can see that the first three columns- season, holiday, weekday are independent variables which we must take in the feature matrix which is denoted as features. Features X is assigned as the second last column fromm the starting column of the file ./Dataset/day.csv i.e. season, holiday, and weekday.

In [6]:

```
target = data.iloc[:,-1]
target.head()
```

Out[6]:

```
0     985
1     801
2    1349
3    1562
4    1600
Name: cnt, dtype: int64
```

From this line we can see workinday column as dependent variable matrix which is denoted as target. Target is assigned as the last column called workingday.

In [7]:

```
from sklearn.model_selection import train_test_split
training_features, testing_features, training_target, testing_target = train_test_split
```

Here, train_test_split class is imported from sklearn library to split features and target into train and test model. Now, variables training_features, training_target, training_target, testing_target are created and using train_test_split class 80% data from features is splitted to training_features model and rest 20% to training_target model and same to the testing_features and testing_target by using test_size=0.1.

In [8]:

```
print(training_features.shape, training_target.shape, '\n',testing_features.shape, test
```

```
(657, 4) (657,)
 (74, 4) (74,)
```

This line shows that 657 examples (65 percent) were allocated to the features set and 74 examples (74 percent) were allocated to the target set, as we specified.

In [9]:

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(criterion='mse', min_samples_leaf=5)
```

Creating a DecisionTreeRegressor with parameter 'criterion' which measures the quality of the splitwith criteria 'mse' which provides information gain.

In [10]:

```
dtr.fit(training_features, training_target)
```

Out[10]:

```
DecisionTreeRegressor(min_samples_leaf=5)
```

This line is used for building model using training and testing set.

In [11]:

```
model_pred = dtr.predict(testing_features)
model_pred
```

Out[11]:
```
array([5483.26086957, 5682.7826087 , 2359.59090909, 3012.63636364,
       2767.2       , 4795.85714286, 5088.28571429, 4318.85      ,
       4942.52173913, 2997.27272727, 2767.2       , 3012.63636364,
       2997.27272727, 4824.64      , 5991.54545455, 4527.85714286,
       5516.36363636, 4570.48148148, 2579.20833333, 4527.85714286,
       5682.7826087 , 2359.59090909, 4318.85      , 5158.76      ,
       2358.125     , 5303.54166667, 5841.53846154, 5991.54545455,
       4505.07407407, 4942.52173913, 2358.125     , 5682.7826087 ,
       4659.41666667, 5483.26086957, 2997.27272727, 2359.59090909,
       5682.7826087 , 5483.26086957, 2767.2       , 4659.41666667,
       4942.52173913, 2358.125     , 5088.28571429, 2997.27272727,
       5991.54545455, 4318.85      , 1644.        , 5516.36363636,
       2359.59090909, 5158.76      , 5158.76      , 5516.36363636,
       4527.85714286, 4735.61538462, 5682.7826087 , 4505.07407407,
       4795.85714286, 4795.85714286, 5516.36363636, 2767.2       ,
       5224.        , 2359.59090909, 5717.        , 5483.26086957,
       5047.54166667, 5991.54545455, 4952.04347826, 2767.2       ,
       4942.52173913, 4795.85714286, 5047.54166667, 5991.54545455,
       5158.76      , 5303.54166667])
```

It is used to scale them to get a more accurate prediction from the model that predicts the class or regression target for the test_features.

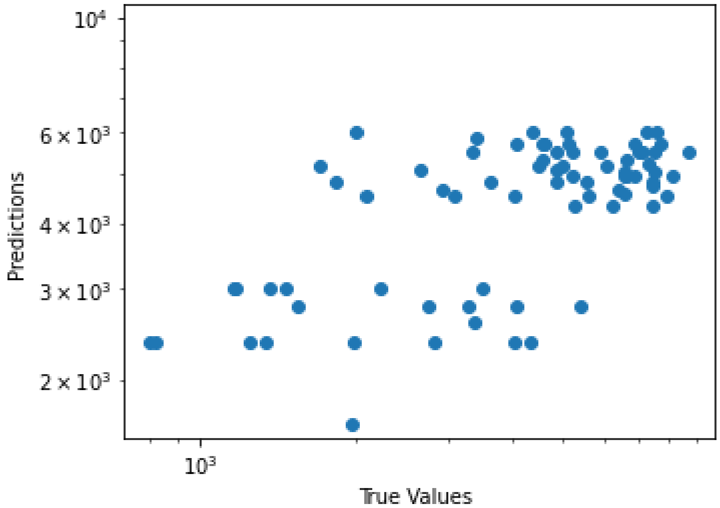
In [12]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
mse = mean_squared_error(testing_target, model_pred)
mae = mean_absolute_error(testing_target, model_pred)
rmse =np.sqrt(mean_squared_error(testing_target, model_pred))
r_square = r2_score(testing_target, model_pred)
print(f'mean square error:{mse}\nmean absolute error:{mae}\nroot mean square error:{rms
```

```
mean square error:3061917.693020366
mean absolute error:1501.307853090027
root mean square error:1749.833618667891
r_square: 0.3493714142575529
```

The above lines of codes are used in testing errors.

In [13]:
```
#evaluating loss functions
g=plt.scatter(testing_target, model_pred )
g.axes.set_yscale('log')
g.axes.set_xscale('log')
g.axes.set_xlabel('True Values ')
g.axes.set_ylabel('Predictions ')
g.axes.axis('equal')
g.axes.axis('square')
```

Out[13]:
```
(710.8914060902362, 9818.537599699175, 1541.0614653928699, 10648.70765900181)
```



This function is used to print the output

g=plt.scatter(testing_target, model_pred) this is used to plot the target and predicated points.

g.axes.set_yscale('log') this is used to plot on the vertical scale. g.axes.set_xscale('log') this is used to plot on the horizontal scale.

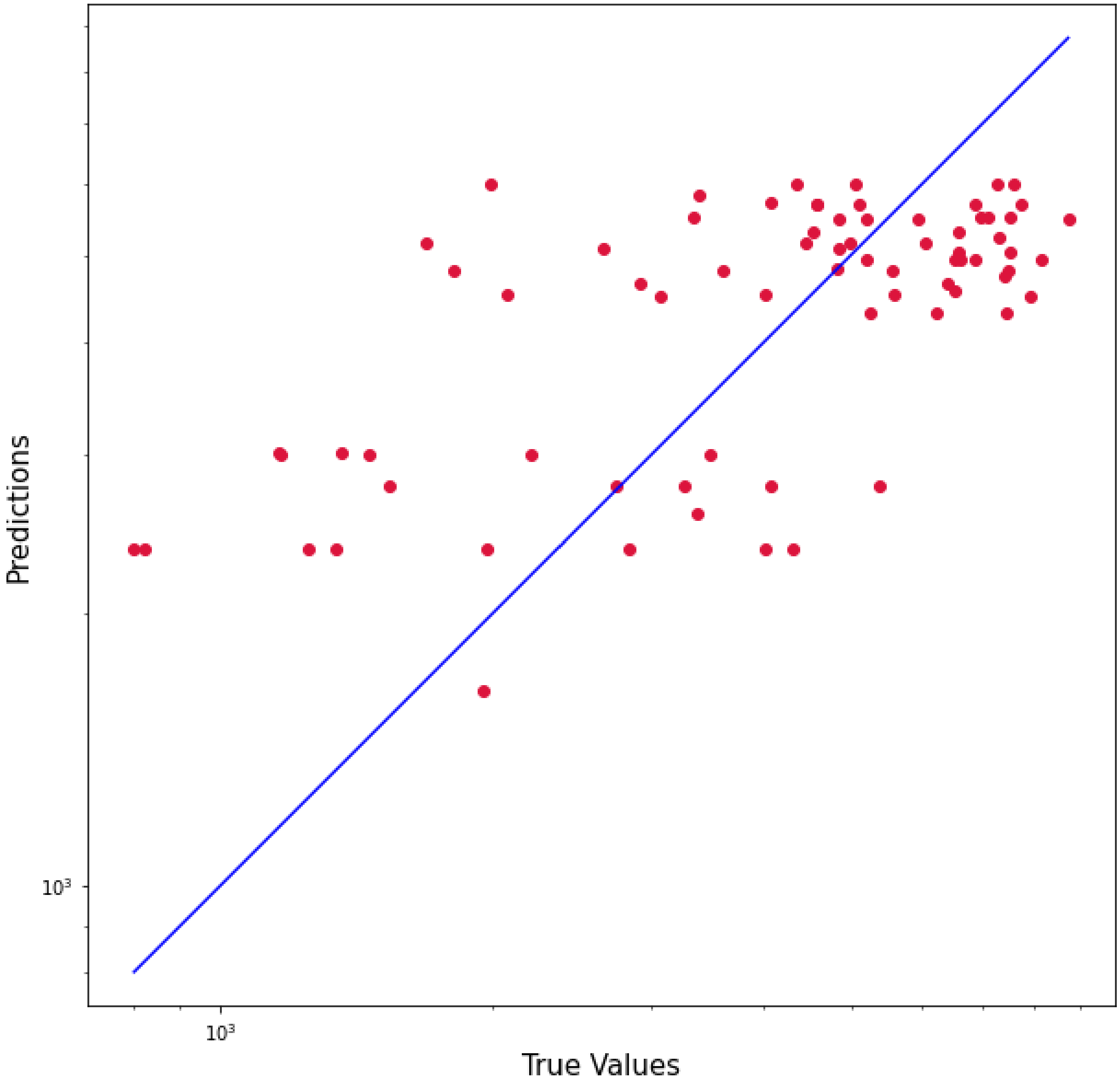
g.axes.set_xlabel('True Values ') this is used to plot on the vertical scale of the actual values in a graph. g.axes.set_ylabel('Predictions ') this is used to plot on the horizontal scale of the predicted values in a graph.

g.axes.axis('equal') g.axes.axis('square') This function is used to print the scatter line either in equal or squared form.

In [14]:

```
plt.figure(figsize=(10,10))
plt.scatter(testing_target, model_pred, c='crimson')
plt.yscale('log')
plt.xscale('log')

p1 = max(max(model_pred), max(testing_target))
p2 = min(min(model_pred), min(testing_target))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



plt.figure(figsize=(10,10)) this is used to define the size of the figure.

plt.scatter(testing_target, model_pred, c='crimson') This is used to print the plot in graph in scatter form where, testing_target shows output in y scale, model_pred shows outputs in x scale and

c='crimson' crimson converts non-standard bioinformatics tool outputs to JSON or YAML

plt.yscale('log') this is used to plot on the vertical scale. plt.xscale('log') this is used to plot on the horizontal scale.

p1 = max(max(model_pred), max(testing_target)) p1 denotes to print the maximum limitations or points of model_prediction and testiong_target. p2 = min(min(model_pred), min(testing_target)) p1 denotes to print the minimum limitations or points of model_prediction and testiong_target.

plt.plot([p1, p2], [p1, p2], 'b-') this aboce line of codes defines plotting lines in a graph while connecting points.

plt.xlabel('True Values', fontsize=15) plt.ylabel('Predictions', fontsize=15) the above lines of codes are meant to define the character size to be printed in output.

plt.axis('equal')

plt.show() this means to print the output of the above statements.

In [15]:
```
import pydotplus
import graphviz
```

Importing the Libraries-

pydotplus: pydotplus is an improved version of the old pydot project that provides a Python Interface to Graphviz's Dot language.

graphviz: Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks.

In [16]:
```
import os
os.environ['PATH'] = os.environ['PATH']+';'+os.environ['CONDA_PREFIX']+r"\Library\bin\g
```

This line add the GraphViz's path to the system.

In [17]:
```
features_column = data.columns[:-1]
features_column
```

Out[17]:
```
Index(['season', 'holiday', 'weekday', 'workingday'], dtype='object')
```

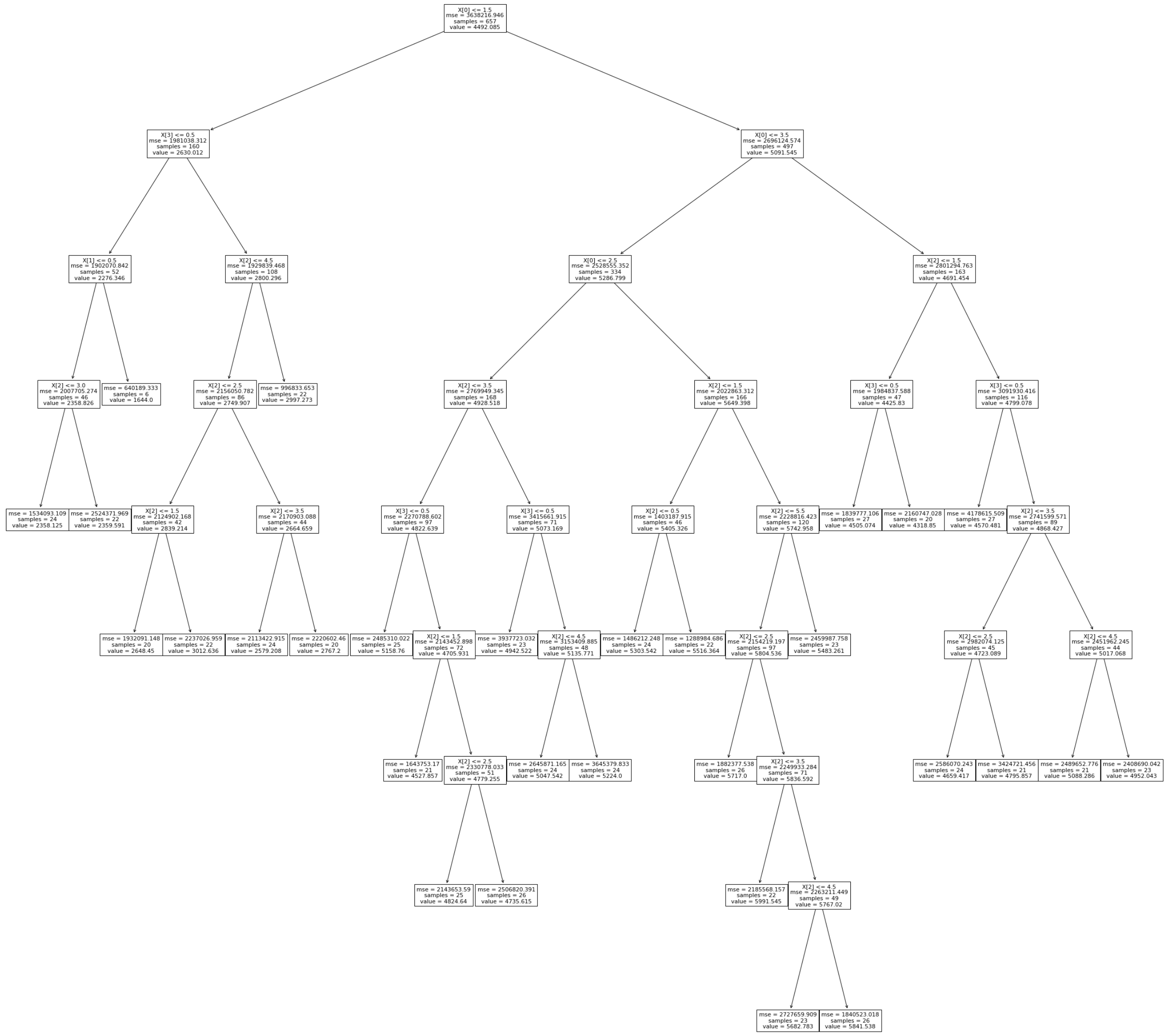
Headings for the 4 columns (season, holiday, weekday and workingday) in features_column.

In [18]:
```
from sklearn import tree
dot_data = tree.export_graphviz(
    dtr, out_file=None, feature_names=features_column, filled=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.set_size('"5,5!"')
graph.write_png('regression_tree_using_graphviz.png')
#If the diagram is not shown in the notebook, please check on the directory where ipynb
```

Out[18]:
```
True
```

tree.export_graphiz(), this function is used to export the regression decision tree in dot format.

In [19]:
```
#using matplotlib to make a regression tree diagram
fig = plt.figure(figsize=(40,40))
tree.plot_tree(dtr)
fig.savefig("regression_decistion_tree_using_matplotlib.png")
```



tree.export_matplotlib(), this function is used to export the regression decision tree in dot format.