The Data Science Pipeline

May 16, 2021

[31]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
from sklearn.model_selection import train_test_split
```

1 Introduction

We will provide a walk-through tutorial of the "Data Science Pipeline" that can be used as a guide for Data Science Projects. We will consider the following phases:

1. Data Collection/Curation
2. Data Management/Representation
3. Exploratory Data Analysis
4. Hypothesis Testing and Machine Learning
5. Communication of insights attened

For this project we will consider a supervised machine learning problem, and more particularly a regression model.

The Regression models involve the following components:

- The unknown parameters, often denoted as a scalar or vector .
- The independent variables, which are observed in data and are often denoted as a vector X_i.
- The dependent variable, which are observed in data and often denoted using the scalar Y_i. The error terms, which are not directly observed in data and are often denoted using the scalar e_i.

This tutorial is based on Python programming language and we will work with different libraries like `pandas`, `numpy`, `matplotlib`, `scikit-learn` and so on. Finally, in this tutorial we provide references and resources in the form of hyperlinks.

2 Data Collection/Curation

The UC Irvine Machine Learning Repository is a Machine Learning Repository which maintains *585 data sets* as a service to the machine learning community. You may view all data sets through our searchable interface. For a general overview of the Repository, please visit our About page.

For information about citing data sets in publication. For our project, we chose to work with the Bike Sharing Dataset Data Set.

2.0.1 Bike Sharing Dataset

This dataset contains the **hourly** count of rental bikes between years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information. Our goal is to build a Machine Learning model which will be able to predict the count of rental bikes.

3 Data Management/Representation

The fields of our dataset are the following:

```
- instant: record index
- dteday : date
- season : season (1:springer, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mnth : month ( 1 to 12)
- hr : hour (0 to 23)
- holiday : weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-sche
- weekday : day of the week
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
+ weathersit :
    - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clou
    - 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)
- 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
```

Let's start the analysis by loading the data.

[32]:
```
df = pd.read_csv("hour.csv")
```

[33]:
```
# get the first rows
df.head()
```

[33]:
```
   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

3.0.1 Feature Leakage

If we look carefully at our data, we will see that the addition of the `casual` and `registered` columns yield to the `cnt` column. This is what we call [leakage](https://en.wikipedia.org/wiki/Leakage_(machine_learning) and for that reason we will remove them from our dataset. The reason for that is when we want to predict the total Bike Rentals `cnt`, we will have as "known" indpendent variables the "casual" and the "registered" which is not true, since by the time of prediction we will lack this info.

[34]:
```
# drop the 'casual' and 'registered' columns
df.drop(['casual', 'registered'], axis=1, inplace=True)
```

3.0.2 Remove the instant column

We will also remove the `instant` from our model since is not an explanatory variable.

[35]:
```
df.drop(['instant'], axis=1, inplace=True)
```

3.0.3 Transform the Columns to the Right Data Type.

We will change the Data Type of the following columns:

- `dteday`: Conver it to Date
- `season`: Convert it to Categorical
- `weekday`: Convert it to Categorical
- `mnth`: Conver it to Categorical

[36]:
```
# let's convert them
df['dteday'] = pd.to_datetime(df['dteday'])
df['season'] = df['season'].astype("category")
df['weekday'] = df['weekday'].astype("category")
df['mnth'] = df['mnth'].astype("category")

# check the data types
df.dtypes
```

[36]:
```
dteday        datetime64[ns]
season              category
yr                     int64
mnth                category
hr                     int64
holiday                int64
```

```
weekday            category
workingday            int64
weathersit            int64
temp                float64
atemp               float64
hum                 float64
windspeed           float64
cnt                   int64
dtype: object
```

3.0.4 Check for Missing Values

At this point we will check for any missing values in our data

[37]:
```
df.isna().sum()
```

[37]:
```
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
dtype: int64
```

As we can see there is no missing value in any field.

3.0.5 Check for Duplicated Values

At this point we will check if there are are duplicated values, where as we can see below, there are not duplicated values. So, we are ok to proceed

[38]:
```
df.duplicated().sum()
```

[38]:
```
0
```

3.0.6 Description of the Dataset

Let's see a summary of our data fields for the continuous variables by showing the mean, std, min, max, and the Q1,Q2,Q3

[39]:
```
df.describe()
```

[39]:

	yr	hr	holiday	workingday	weathersit
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000
mean	0.502561	11.546752	0.028770	0.682721	1.425283
std	0.500008	6.914405	0.167165	0.465431	0.639357
min	0.000000	0.000000	0.000000	0.000000	1.000000
25%	0.000000	6.000000	0.000000	0.000000	1.000000
50%	1.000000	12.000000	0.000000	1.000000	1.000000
75%	1.000000	18.000000	0.000000	1.000000	2.000000
max	1.000000	23.000000	1.000000	1.000000	4.000000

	temp	atemp	hum	windspeed	cnt
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000
mean	0.496987	0.475775	0.627229	0.190098	189.463088
std	0.192556	0.171850	0.192930	0.122340	181.387599
min	0.020000	0.000000	0.000000	0.000000	1.000000
25%	0.340000	0.333300	0.480000	0.104500	40.000000
50%	0.500000	0.484800	0.630000	0.194000	142.000000
75%	0.660000	0.621200	0.780000	0.253700	281.000000
max	1.000000	1.000000	1.000000	0.850700	977.000000

Finally, let's the number of rows and colums of our dataset so far.

[40]:
```
df.shape
```

[40]: (17379, 14)

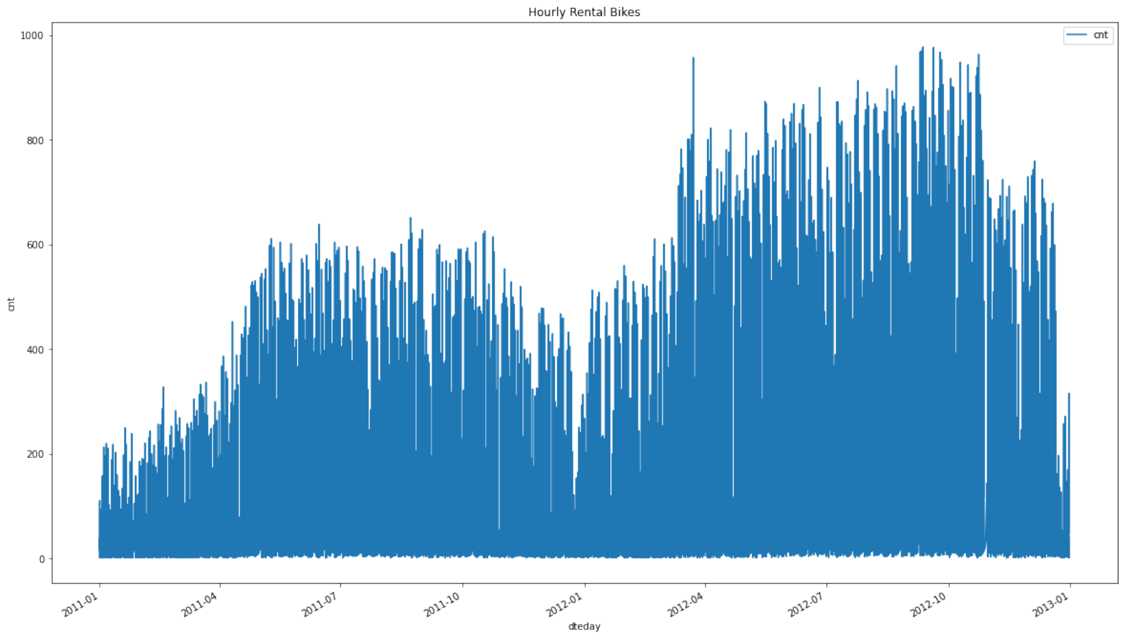
4 Exploratory Data Analysis

At this point we run an EDA. Let's have a look at the Bike Rentals across time.

4.0.1 Time Series Plot of Hourly Rental Bikes

[41]:
```
df.plot(x='dteday', y='cnt', figsize=(20,12), title = 'Hourly Rental Bikes')
plt.ylabel('cnt')
```

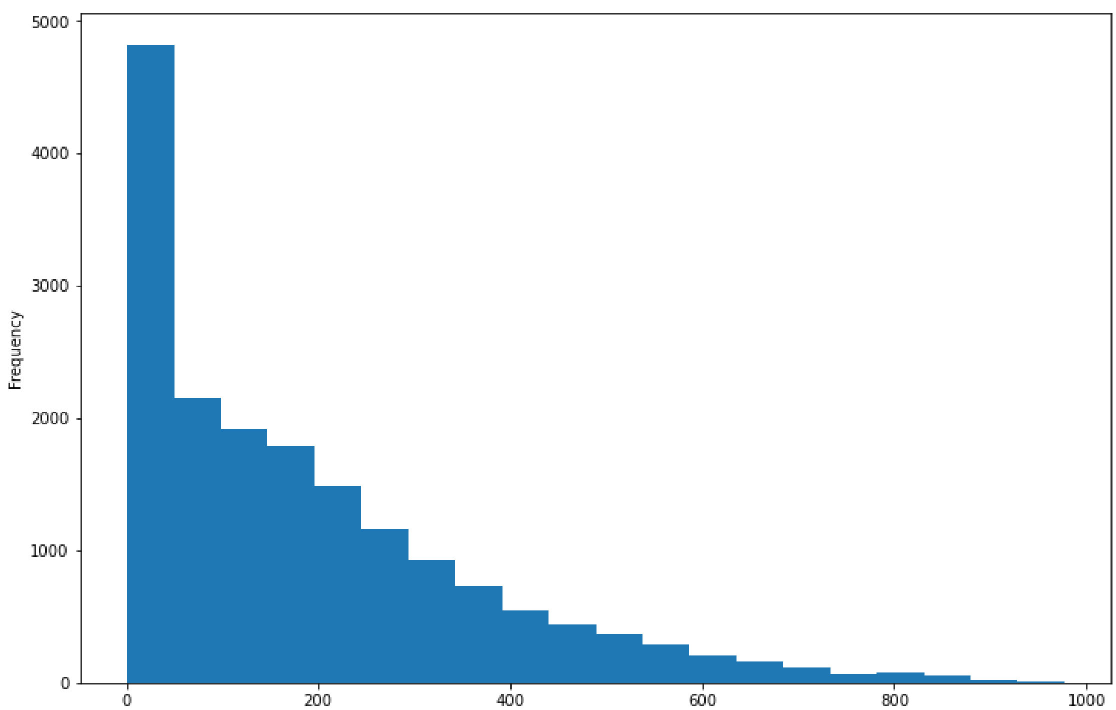
[41]: Text(0, 0.5, 'cnt')



4.0.2 Distribution of the Rental Bikes

[42]:
```
df['cnt'].plot.hist(bins=20, figsize=(12,8))
```

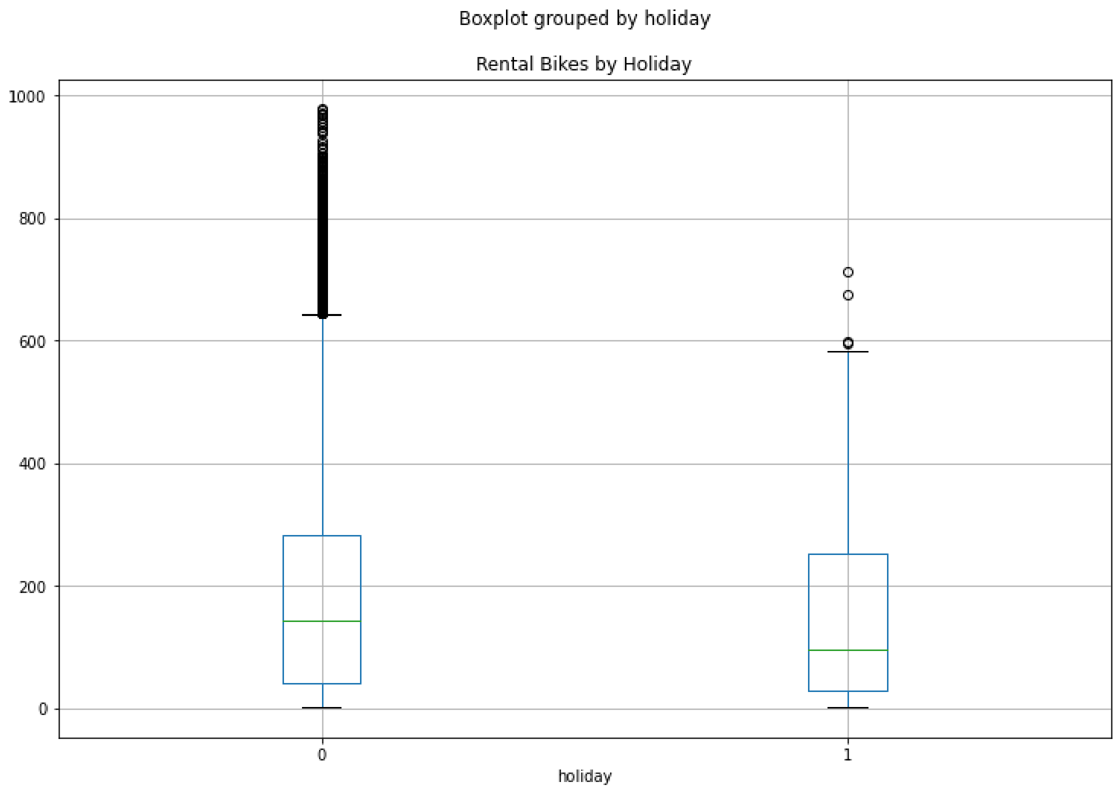
[42]: <AxesSubplot:ylabel='Frequency'>



4.0.3 Box Plots of Bike Rentals

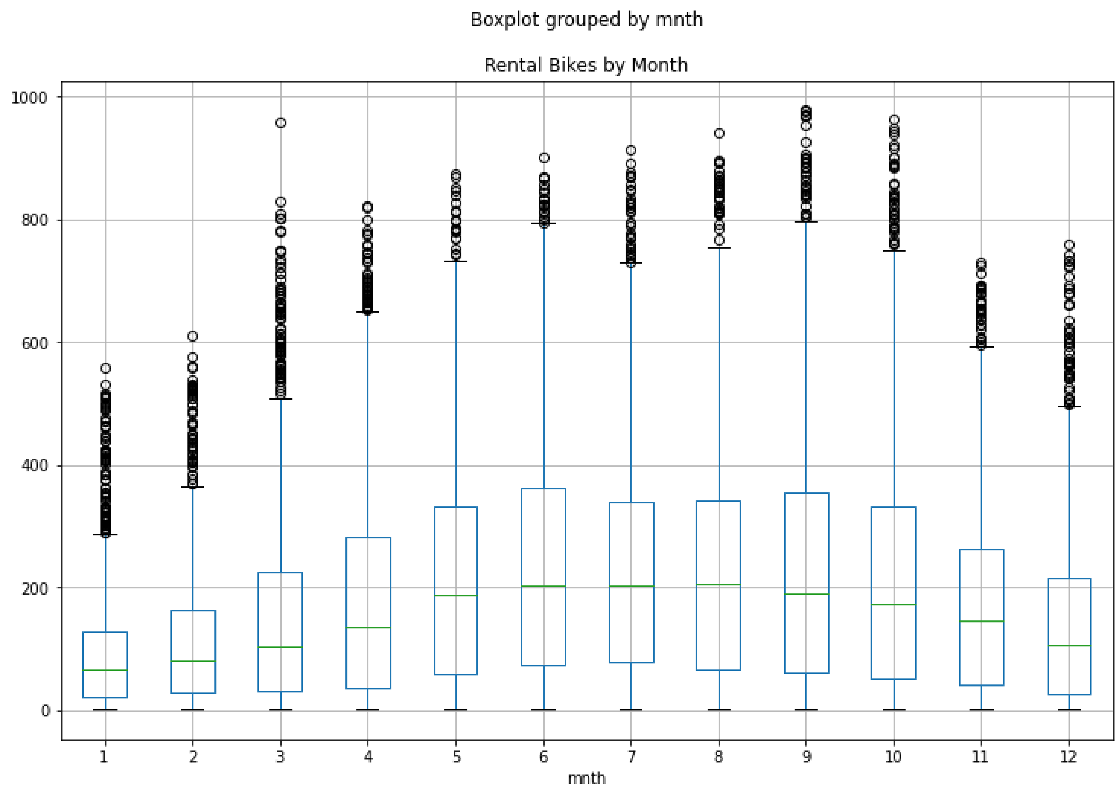
[43]:
```
df.boxplot(by='holiday', column='cnt', figsize=(12,8))
plt.title("Rental Bikes by Holiday")
```

[43]: Text(0.5, 1.0, 'Rental Bikes by Holiday')



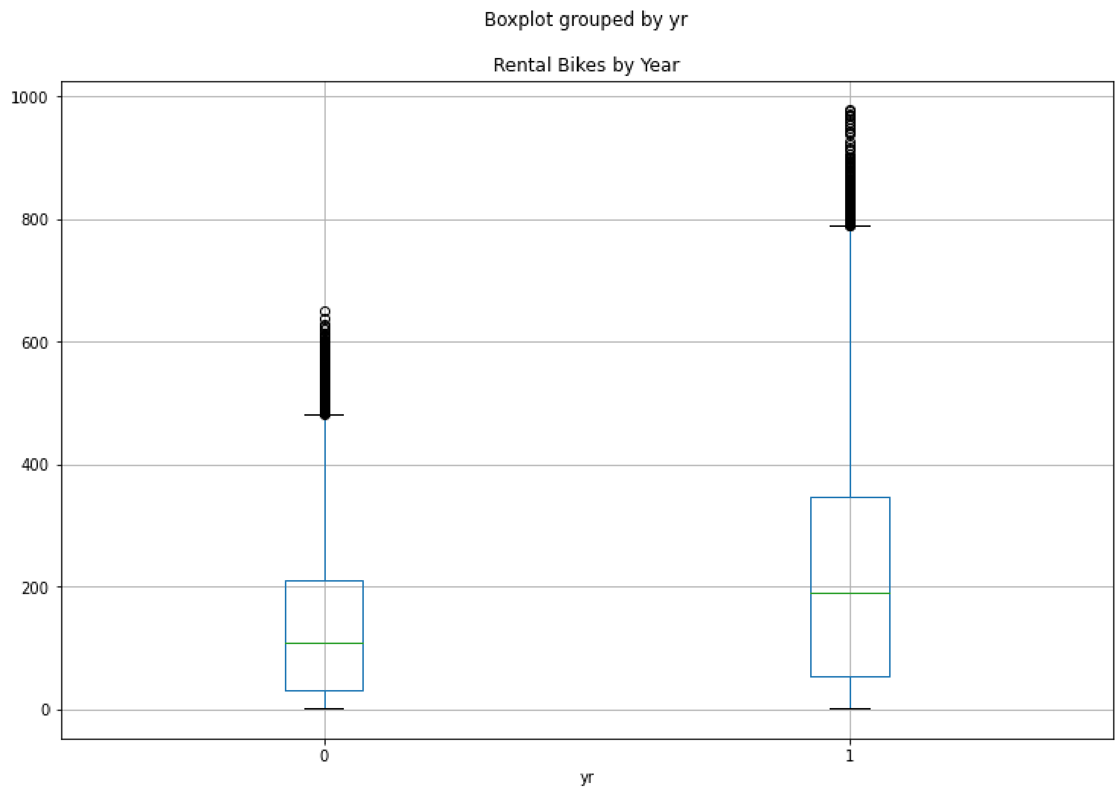
[44]:
```
df.boxplot(by='mnth', column='cnt', figsize=(12,8))
plt.title("Rental Bikes by Month")
```

[44]: Text(0.5, 1.0, 'Rental Bikes by Month')



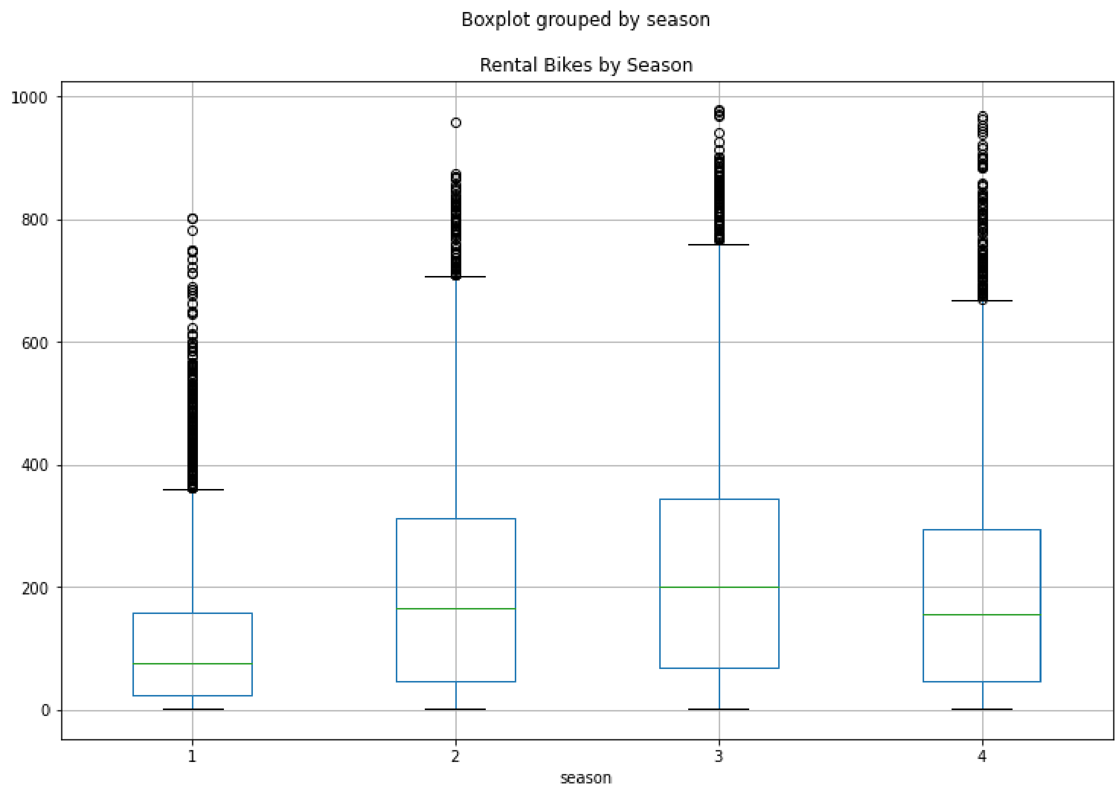
[45]:
```
df.boxplot(by='yr', column='cnt', figsize=(12,8))
plt.title("Rental Bikes by Year")
```

[45]: Text(0.5, 1.0, 'Rental Bikes by Year')



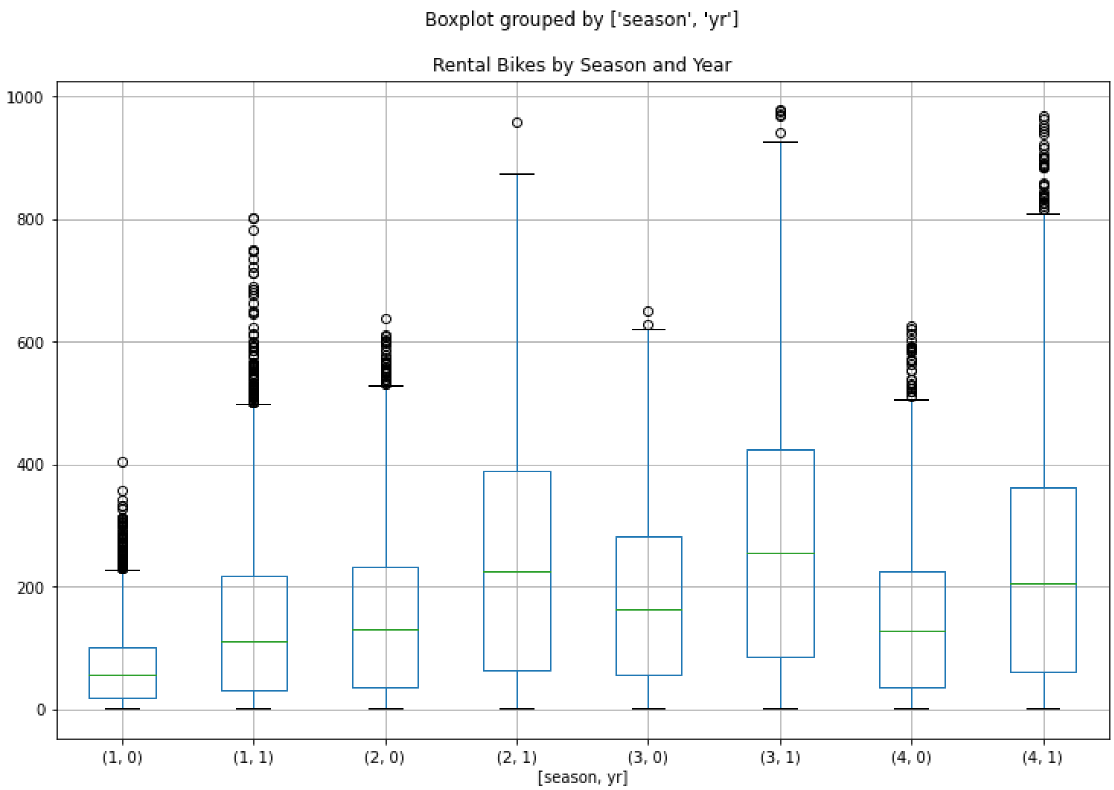

```
[46]: df.boxplot(by='season', column='cnt', figsize=(12,8))
      plt.title("Rental Bikes by Season")
```

```
[46]: Text(0.5, 1.0, 'Rental Bikes by Season')
```



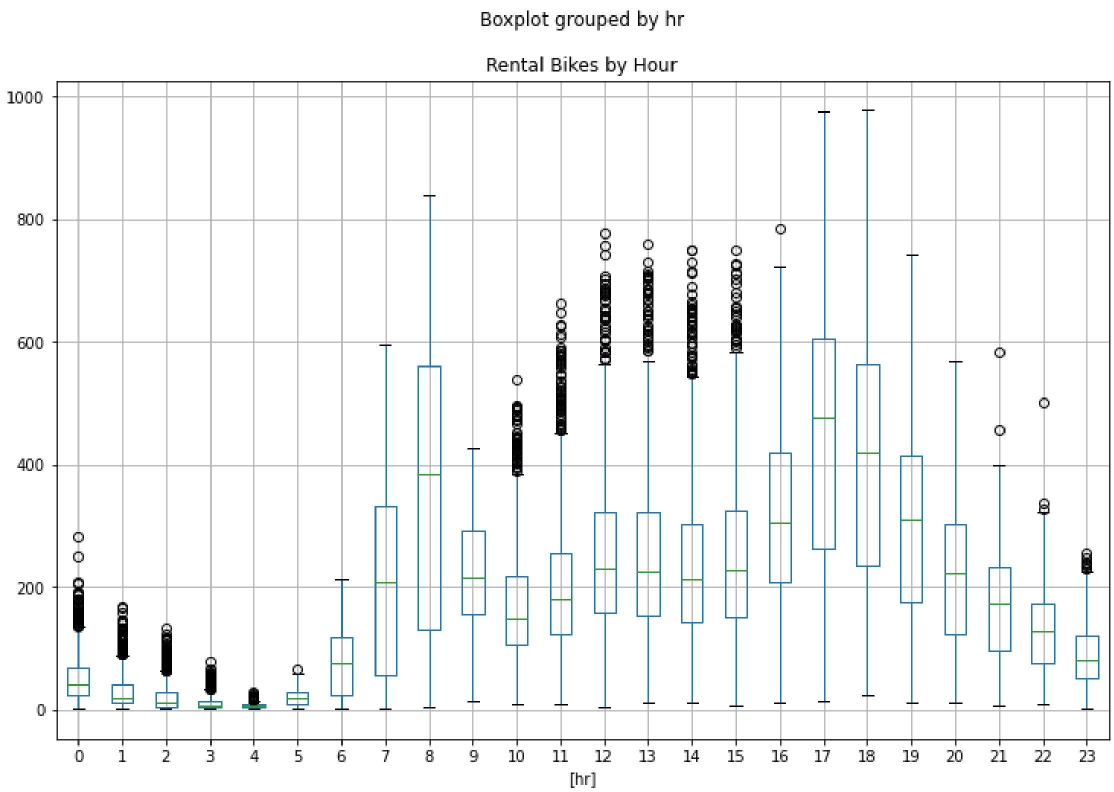

```
[47]: df.boxplot(by=['season','yr'], column='cnt', figsize=(12,8))
      plt.title("Rental Bikes by Season and Year")
```

```
[47]: Text(0.5, 1.0, 'Rental Bikes by Season and Year')
```



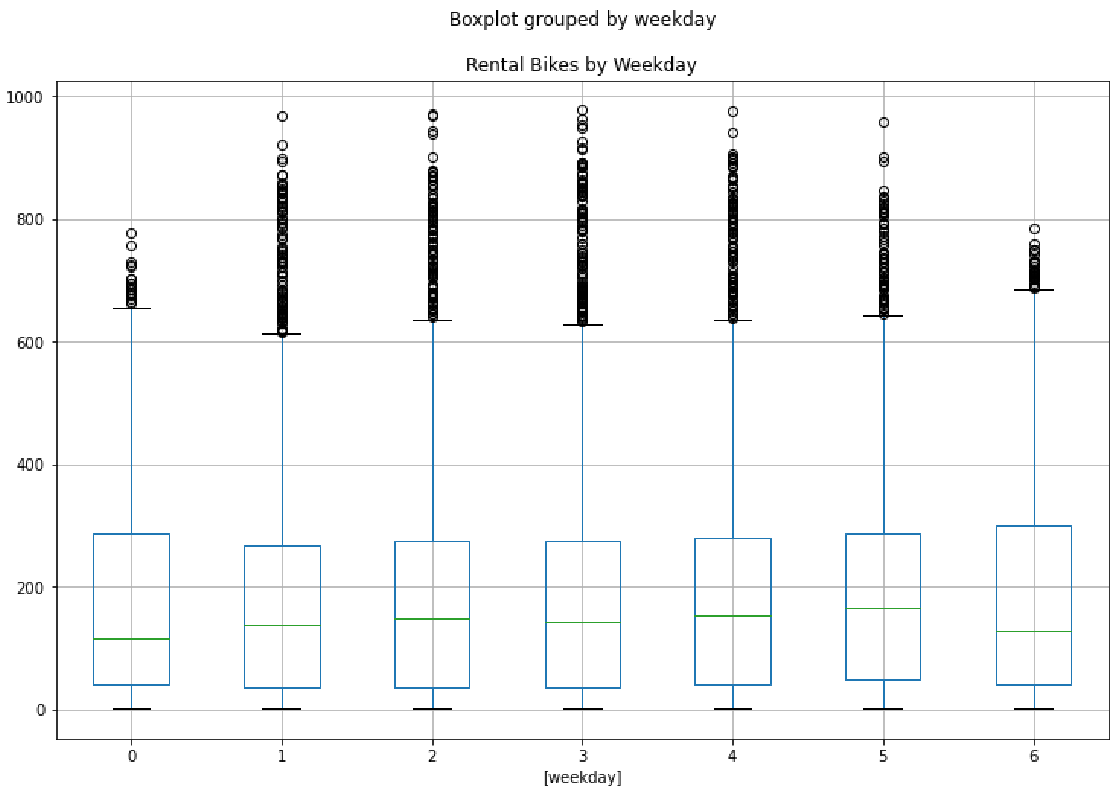
[48]:
```
df.boxplot(by=['hr'], column='cnt', figsize=(12,8))
plt.title("Rental Bikes by Hour")
```

[48]: Text(0.5, 1.0, 'Rental Bikes by Hour')



[49]:
```
df.boxplot(by=['weekday'], column='cnt', figsize=(12,8))
plt.title("Rental Bikes by Weekday")
```

[49]: Text(0.5, 1.0, 'Rental Bikes by Weekday')



4.0.4 Correlation and Correlation Heat Map

We will return the correlation Pearson coefficient of the numeric variables.

[50]:
```
# get the correlation

df.corr()
```

[50]:

	yr	hr	holiday	workingday	weathersit	temp
yr	1.000000	-0.003867	0.006692	-0.002196	-0.019157	0.040913
hr	-0.003867	1.000000	0.000479	0.002285	-0.020203	0.137603
holiday	0.006692	0.000479	1.000000	-0.252471	-0.017036	-0.027340
workingday	-0.002196	0.002285	-0.252471	1.000000	0.044672	0.055390
weathersit	-0.019157	-0.020203	-0.017036	0.044672	1.000000	-0.102640
temp	0.040913	0.137603	-0.027340	0.055390	-0.102640	1.000000
atemp	0.039222	0.133750	-0.030973	0.054667	-0.105563	0.987672
hum	-0.083546	-0.276498	-0.010588	0.015688	0.418130	-0.069881
windspeed	-0.008740	0.137252	0.003988	-0.011830	0.026226	-0.023125
cnt	0.250495	0.394071	-0.030927	0.030284	-0.142426	0.404772

\

	atemp	hum	windspeed	cnt
yr	0.039222	-0.083546	-0.008740	0.250495

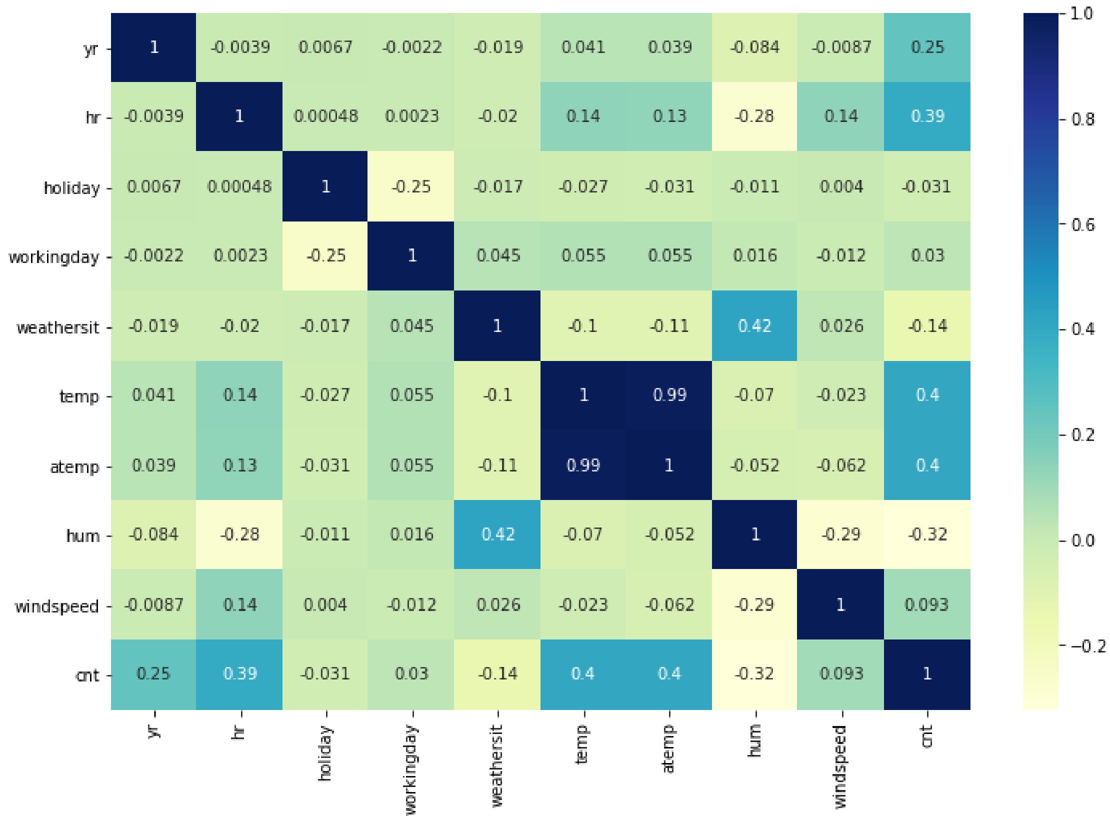
```
hr            0.133750 -0.276498   0.137252  0.394071
holiday      -0.030973 -0.010588   0.003988 -0.030927
workingday    0.054667  0.015688  -0.011830  0.030284
weathersit   -0.105563  0.418130   0.026226 -0.142426
temp          0.987672 -0.069881  -0.023125  0.404772
atemp         1.000000 -0.051918  -0.062336  0.400929
hum          -0.051918  1.000000  -0.290105 -0.322911
windspeed    -0.062336 -0.290105   1.000000  0.093234
cnt           0.400929 -0.322911   0.093234  1.000000
```

[51]:
```
# correlation of Rental Bikes vs the rest variables
df.drop('cnt', axis=1).corrwith(df.cnt)
```

[51]:
```
yr            0.250495
hr            0.394071
holiday      -0.030927
workingday    0.030284
weathersit   -0.142426
temp          0.404772
atemp         0.400929
hum          -0.322911
windspeed     0.093234
dtype: float64
```

[52]:
```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, cmap="YlGnBu")
```

[52]: <AxesSubplot:>



4.0.5 Multi-Collinearity

As expected the `temp` and `atemp` are strongly correlated causing a problem of muticollinearity and that is why we will keep only one. We will remove the `temp`.

[53]:
```
df.drop('temp', axis=1, inplace=True)
```

[54]:
```
# remove also the date column since we will not use it for the machine␣
↪modelling
df.drop('dteday', axis=1, inplace=True)
```

5 Hypothesis Testing and Machine Learning

Before we start analysing our models, we will need to apply one-hot encoding to the categorical variables. We will do that by applying the get_dummies function.

5.0.1 One-Hot Encoding

[55]:
```
df = pd.get_dummies(df)
df.head()
```

[55]:
```
   yr  hr  holiday  workingday  weathersit   atemp   hum  windspeed  cnt  \
0   0   0        0           0           1  0.2879  0.81        0.0   16
1   0   1        0           0           1  0.2727  0.80        0.0   40
2   0   2        0           0           1  0.2727  0.80        0.0   32
3   0   3        0           0           1  0.2879  0.75        0.0   13
4   0   4        0           0           1  0.2879  0.75        0.0    1

   season_1  …  mnth_10  mnth_11  mnth_12  weekday_0  weekday_1  weekday_2  \
0         1  …        0        0        0          0          0          0
1         1  …        0        0        0          0          0          0
2         1  …        0        0        0          0          0          0
3         1  …        0        0        0          0          0          0
4         1  …        0        0        0          0          0          0

   weekday_3  weekday_4  weekday_5  weekday_6
0          0          0          0          1
1          0          0          0          1
2          0          0          0          1
3          0          0          0          1
4          0          0          0          1

[5 rows x 32 columns]
```

5.0.2 Train-Test Dataset

For our analysis we split the dataset into train and test (75% -25%) so that to build the models on the train dataset and to evaluate them on the test dataset.

[57]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('cnt', axis=1), df.
 ↪cnt, test_size=0.25, random_state=5)
```

5.0.3 Machine Learning Models

We will try different machine learning models

- Linear Regression
- Random Forest
- Gradient Boost

and we will choose the one with the lowest RMSE.

Linear Regression

[58]:
```
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
```

```python
reg = LinearRegression().fit(X_train, y_train)

# Get the RMSE for the train dataset
print(np.sqrt(mean_squared_error(y_train, reg.predict(X_train))))

# Get the RMSE for the test dataset
print(np.sqrt(mean_squared_error(y_test, reg.predict(X_test))))
```

```
139.63173269641877
141.72019878710364
```

Random Forest

[59]:
```python
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor().fit(X_train, y_train)

# Get the RMSE for the train dataset
print(np.sqrt(mean_squared_error(y_train, rf.predict(X_train))))

# Get the RMSE for the test dataset
print(np.sqrt(mean_squared_error(y_test, rf.predict(X_test))))
```

```
16.405429061178168
43.00939768897331
```

Gradient Boost

[60]:
```python
from sklearn.ensemble import GradientBoostingRegressor

gb = GradientBoostingRegressor().fit(X_train, y_train)

# Get the RMSE for the train dataset
print(np.sqrt(mean_squared_error(y_train, gb.predict(X_train))))

# Get the RMSE for the test dataset
print(np.sqrt(mean_squared_error(y_test, gb.predict(X_test))))
```

```
69.83046975214577
69.94424353189754
```

5.0.4 Choose the Best Model

Based on the **RMSE** on both train and test dataset, the best model is the **Random Forest**.

5.0.5 Statistical Analysis

Based on the statistical analysis and the Gini, we will define the most important variables of the Random Forest model.

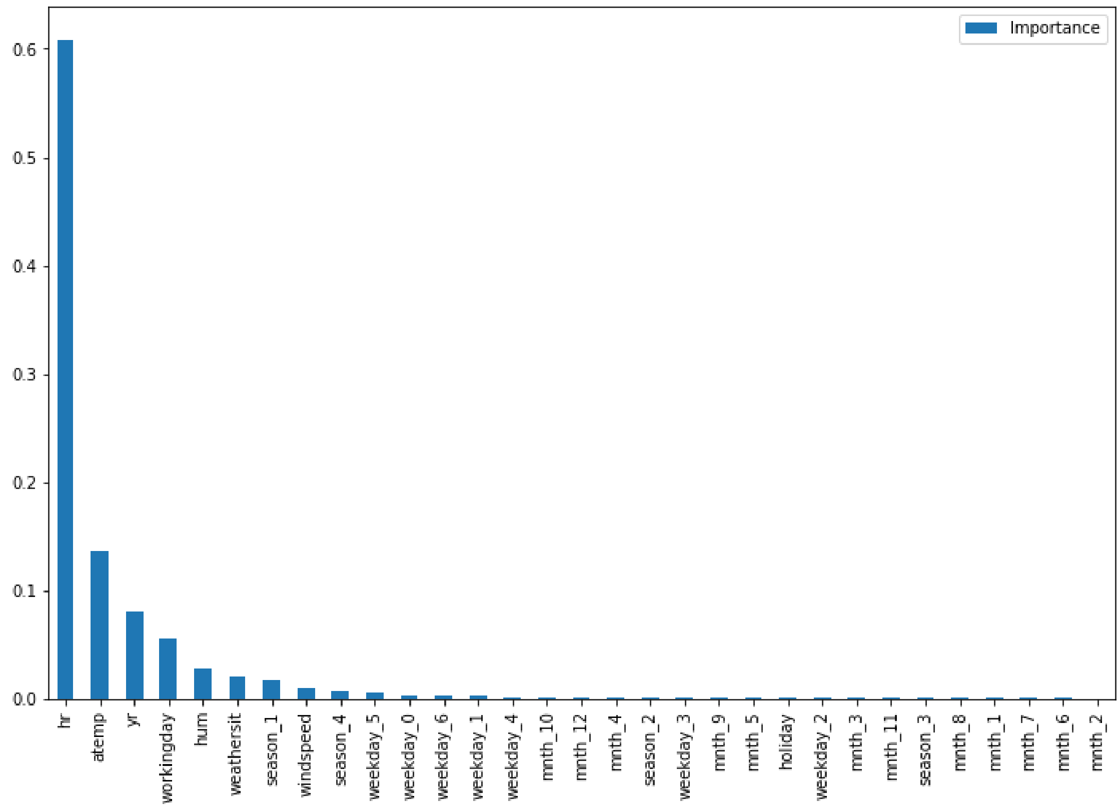
[61]:
```
feat_importances = pd.DataFrame(rf.feature_importances_, index=X_train.columns,
↪columns=["Importance"])
feat_importances.sort_values(by='Importance', ascending=False, inplace=True)
feat_importances
```

[61]:

	Importance
hr	0.608568
atemp	0.136450
yr	0.080672
workingday	0.055759
hum	0.027176
weathersit	0.019707
season_1	0.017859
windspeed	0.010408
season_4	0.006942
weekday_5	0.005842
weekday_0	0.002753
weekday_6	0.002346
weekday_1	0.001960
weekday_4	0.001871
mnth_10	0.001825
mnth_12	0.001820
mnth_4	0.001762
season_2	0.001725
weekday_3	0.001672
mnth_9	0.001587
mnth_5	0.001550
holiday	0.001513
weekday_2	0.001343
mnth_3	0.001177
mnth_11	0.001079
season_3	0.000910
mnth_8	0.000894
mnth_1	0.000857
mnth_7	0.000815
mnth_6	0.000745
mnth_2	0.000415

[62]:
```
feat_importances.plot(kind='bar', figsize=(12,8))
```

[62]: <AxesSubplot:>



As we can see the most important variables are:

- The Hour with an importance of 60%
- The Temperature with an importance of 14%
- The Year with an importance of 8%

6 Insights

We found that the number of Bike Rentals depend on the `hour` and the `temperature`. Also, it seems that there is an interaction between variables, like `hour and day of week`, or `month and year` etc and for that reason the tree based models like Gradient Boost and Random Forest performed much better than the linear regression. Moreover, the tree based models are able to capture non linear relationships, so for example, the hours and the temperature do not have necessary linear relationship, so for example if it is extremely hot or cold then the bike rentals can drop. Our model has an RMSE of 42 in the test dataset which seems to be promising.

6.0.1 Further Analysis

There is always a room of improvement when we build Machine Learning models. For instance we could try the following:

- Transform the `cnt` column to the logarith of `cnt`
- Try different models using Grid Search

- Fine tuning of the Hyperparameters of the model

[ ]:
```

```