Bike-Sharing Market Analysis

With Pyspark

Lei Chen

Department of Computer Science
The University of Auckland New Zealand

Depository

Codes and Dataset for this report is available on my GitHub Depository:
https://github.com/Litzenz/INFOSYS722
Please see below invitation links if it is unavailable:
https://github.com/Litzenz/INFOSYS722/invitations

Background

The bike-sharing system is a new generation of traditional bike rentals with automatically process from membership, rental to return. These systems make it easy for users to rent a bike from a specific location and then return to another location. Currently, there are more than 500 bike-sharing projects around the world, consisting of more than half a million bikes. Today, there is great interest in these systems because of their important role in transport, environment and health issues.

In addition, the data generated by these systems make them attractive for research. Unlike other transportation services, such as buses or subways, the travel, departure, and arrival locations of these systems are clearly documented. This feature makes bike-sharing systems as a virtual sensor networks that can be used to detect mobility in cities.

1. Business Understanding

1.1. Business Objectives

However, in China, the pursuit of rapid expansion in the market competition, resulted in a large number of bicycle disorderly release in most cities, which has been a huge waste of resources and disrupting the urban order. Thus, a prediction to the demand of sharing-bikes is necessary, not only for the companies concerning their profits, but also for the government, which are able to dynamically manage the overall amount of shared bikes in the market.

As we all know, Bike-sharing rental process is highly correlated to the commuting traffic, and weather settings. For instance, weather conditions, day of week, season, hour of the day, etc. can affect the rental behaviors. Meanwhile, count of rented bikes are also correlated to some events in the town.

Thus, this project will focus on below objectives:

- Predication of bike rental count based on various variables.

Tentatively, it will be judged a success with below Success Criteria:

- Reduce resource wasted

1.2. Assessing the Situation

- Personnel

  I will work on Pysark in AWS EC2 environment.

  I will use git for to synchronize my works and create different branch for steps iteration.

  Also resources including online learning materials and guide from tutors are available.

- Data Sources

  The bike-sharing systems record almost all information of user's rental behavior, and some of them have released as open data sets that can support my data-mining project.

  So I can search for the appropriate data from authoritative dataset websites, such as Kaggle: https://www.kaggle.com/

Yelp: https://www.yelp.com/dataset
NZ Government Data Portal: https://catalogue.data.govt.nz/dataset
KD Nuggets: https://www.kdnuggets.com/datasets/index.html

- Requirements
  There are not deployment requirements in this project.
- Assumption
  I assume the data set has all features which are relevant to the predictor, and no other factors would influence the project except the data quality.
- Constraints
  Kaggle is an open source platform and all data sets provided by Kaggle is free to access. So there is no constraints for data access, financially and legally.
- Risks
  There should be no real-world risk as it is just a predicting project with historical data. But in cloud environment, synchronization issue may happen due to network failure.
- Contingency
  For synchronization risk, I will use Github to avoid my works unexpected losing or overwriting

1.3. Data Mining Goals

We can translate the business objectives into data mining terms.
The goal for this project to be completed is:

- use the dataset with historical records of previous rentals to generate an efficient model to predict bike-sharing demand trend.

The success criterion is:

- R2 (coefficient of determination) score of model is greater than 0.85.

1.4. Project Plan

The overview plan for the study is as shown in the table below.

Phase	Time	Resources	Comments
Business Understanding	0.5 week	Google	Case study for bike-sharing
Data Understanding	0.5 week	Kaggle, Pyspark	Learn data visualization
Data Preparation	0.5 week	Pyspark	Learn data operation
Data Transformation	0.5 week	Pyspark	Learn data manipulation
Data Mining	0.5 week	Pyspark	Learn data mining algorithms
Interpretation	0.5 week	Pyspark	Pattern study and visualization

2. Data Understanding

2.1. Data Collection

This project uses bike-sharing data set downloaded from Kaggle:
https://www.kaggle.com/marklvl/bike-sharing-dataset,
The original data source are available in below website:
http://capitalbikeshare.com/system-data
And the corresponding weather and holiday schedule information are extracted from:
http://www.freemeteo.com and http://dchr.dc.gov/page/holiday-schedule respectively.
Kaggle is a popular open data website where we can collect and download authoritative and verified datasets. So the dataset in this project is reliable and credible.

2.2. Data Description

These data contain the two-year historical log, corresponding to years 2011 and 2012, from Capital Bikeshare system, Washington D.C., USA, stored in a CSV files.
Firstly, I import the data and check its overall information, see Figure 1.

```
import findspark
findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('iteration4').getOrCreate()
df = spark.read.csv('dataset.csv', header=True)
df.show(5) #df.head()
# check features and data type
df.printSchema() #df.columns & df.dtypes
# check number of records
print('Number of Rows: ', df.count())
```

```
+-------+----------+------+---+----+---+-------+-------+----------+----------+----+------+----+---------+------+----------+---+
|instant|    dteday|season| yr|mnth| hr|holiday|weekday|workingday|weathersit|temp| atemp| hum|windspeed|casual|registered|cnt|
+-------+----------+------+---+----+---+-------+-------+----------+----------+----+------+----+---------+------+----------+---+
|   null|01/01/2011|     1|  0|   1|  0|      0|      6|         0|         1|0.24|0.2879|0.81|        0|     3|        13| 16|
|      2|01/01/2011|     1|  0|   1|  1|      0|      6|         0|         1|0.22|  null| 0.8|        0|     8|        32| 40|
|      3|01/01/2011|     1|  0|   1|  2|      0|      6|         0|         1|0.22|0.2727|null|        0|     5|        27| 32|
|      4|01/01/2011|     1|  0|   1|  3|      0|      6|         0|         1|0.24|0.2879|0.75|        0|     3|        10| 13|
|      5|01/01/2011|     1|  0|   1|  4|      0|      6|         0|         1|0.24|0.2879|0.75|        0|     0|         1|  1|
+-------+----------+------+---+----+---+-------+-------+----------+----------+----+------+----+---------+------+----------+---+
only showing top 5 rows

root
 |-- instant: string (nullable = true)
 |-- dteday: string (nullable = true)
 |-- season: string (nullable = true)
 |-- yr: string (nullable = true)
 |-- mnth: string (nullable = true)
 |-- hr: string (nullable = true)
 |-- holiday: string (nullable = true)
 |-- weekday: string (nullable = true)
 |-- workingday: string (nullable = true)
 |-- weathersit: string (nullable = true)
 |-- temp: string (nullable = true)
 |-- atemp: string (nullable = true)
 |-- hum: string (nullable = true)
 |-- windspeed: string (nullable = true)
 |-- casual: string (nullable = true)
 |-- registered: string (nullable = true)
 |-- cnt: string (nullable = true)

Number of Rows:  17379
```

Figure 1 Data Information

There are 17379 rental records in total, with 17 columns of fields, including season, month, hours in a day, working day or not, weather situations and so on.

Fields ‘dteday’ can be considered as the IDs of records, and fields ‘registered’ and ‘casual’ are just components of the overall rental count, so we can ignore them in the data mining process. Also I noticed that all the data types imported were ‘string’, so it it necessary to specify data type before further exploration.

Table 1 lists the detailed attributes of the data and explanation of them according to data source website.

Field	Explaination
instant	record index
dteday	date (yyyy-mm-dd)
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	holiday is 1, otherwise is 0
weekday	day of the week (0 to 6)
workingday	working day is 1, otherwise is 0
weathersit	weather conditions (1 to 4)
temp	Normalized temperature in Celsius.
atemp	Normalized feeling temperature in Celsius.
hum	Normalized humidity.
windspeed	Normalized wind speed.
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes

Table 1 Fields Detail

According to Table 1, I specify the data type for each feature and reloaded the data again.

```
# data format
from pyspark.sql.types import (StructField, StructType,
  TimestampType, IntegerType, FloatType)
# define data schema (or use inferSchema =True when loading dataframe)
data_schema = [StructField('instant', IntegerType(), True),
                        StructField('dteday', TimestampType(), True),
                        StructField('season', IntegerType(), True),
                        StructField('yr', IntegerType(), True),
                        StructField('mnth', IntegerType(), True),
                        StructField('hr', IntegerType(), True),
                        StructField('holiday', IntegerType(), True),
                        StructField('weekday', IntegerType(), True),
                        StructField('workingday', IntegerType(), True),
                        StructField('weathersit', IntegerType(), True),
                        StructField('temp', FloatType(), True),
                        StructField('atemp', FloatType(), True),
                        StructField('hum', FloatType(), True),
                        StructField('windspeed', FloatType(), True),
```

```
                    StructField('casual', IntegerType(), True),
                    StructField('registered', IntegerType(), True),
                    StructField('cnt', IntegerType(), True)]
final_struct = StructType(fields = data_schema)
df = spark.read.csv('dataset.csv', schema=final_struct, header=True)
```

2.3. Data Exploration

To predict the rental demand, we can start with checking the correlation of 'cnt' (overall count of rental bikes) against other fields. Basically, these fields can be divided into 2 parts: the weather features (including temperature, windspeed and humidity), and time features such as year, month, and season.

For better data understanding, I built some plots to demonstrate the relationships of some relevant fields and the prediction goal – total rental counts. As Spark style data is hard to visualize with popular Python visualization packages, I use 'toPandas' method to transfer data into Pandas style before plotting.

```
# Import the relevant Python libraries.
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn

data = df.toPandas()

sn.barplot(data['weathersit'], data['cnt'])
plt.title('the influnce of weather')

sn.boxplot(data['yr'], data['cnt'])
plt.title('the influnce of year')

sn.pointplot(data['mnth'], data['cnt'])
plt.title('the influnce of month')

sn.boxplot(data['season'], data['cnt'])
plt.title('the influnce of season')

sn.barplot(data['hr'], data['cnt'])
plt.title('the influnce of hours in a day')
```

Figure 2 shows that the effect of weather conditions on rental behavior is obvious.

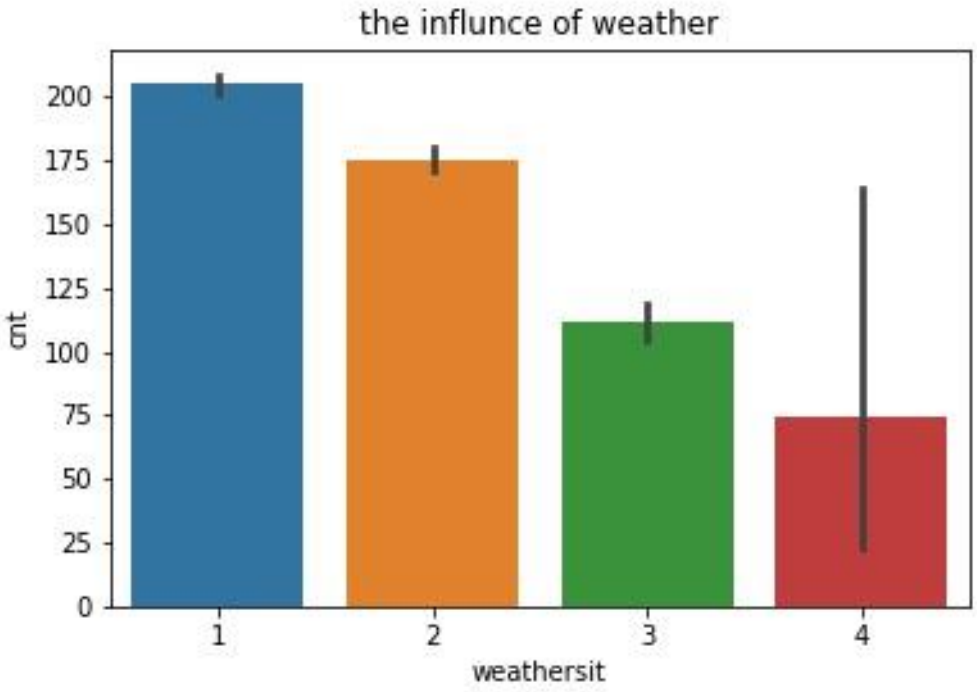

Figure 2

Figure 3 demonstrates that the number of rentals in 2012 is significantly higher than in 2011, showing that shared bikes have become more and more popular.

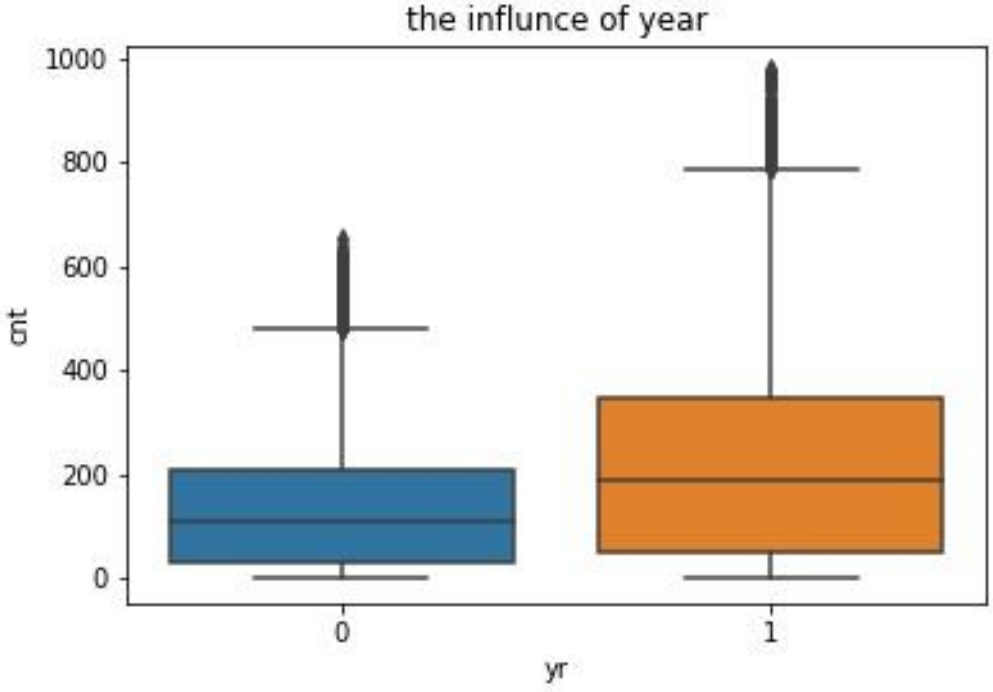

Figure 3

Meanwhile, the impact of months on rental counts is clearly the same in 2011 and 2012, with a rapid increase in monthly rentals from January to peak in June, but a sharp decline after

October, which is a clearly seasonal trend, see Figure 4.

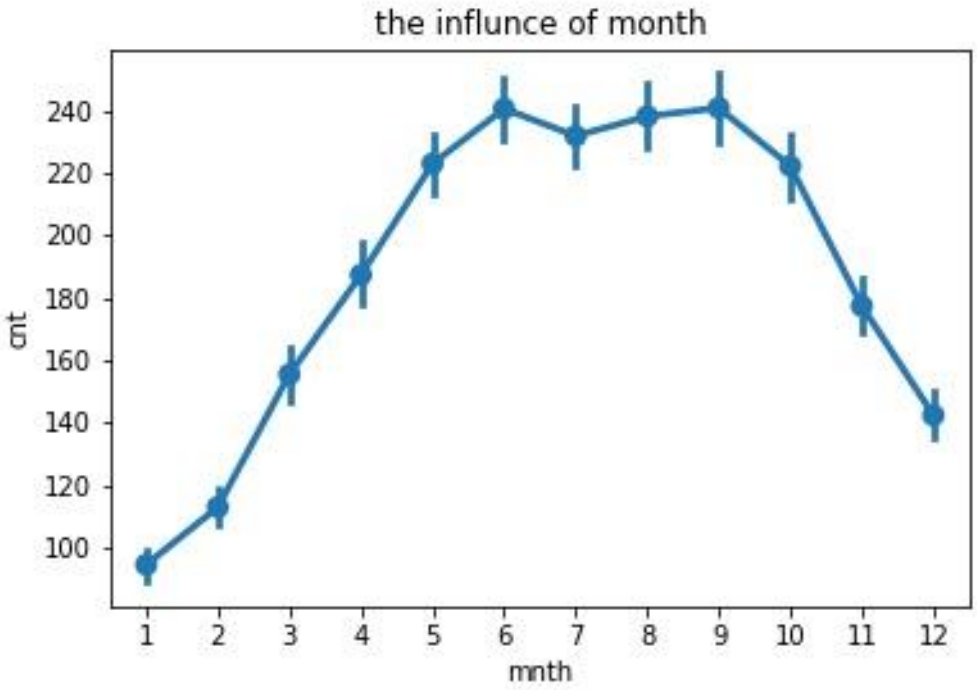

Figure 4

In terms of hours within a day, Figure 5 shows two peaks every day, around 8 a.m. and 17 p.m., just in time for the rush hour of working days.

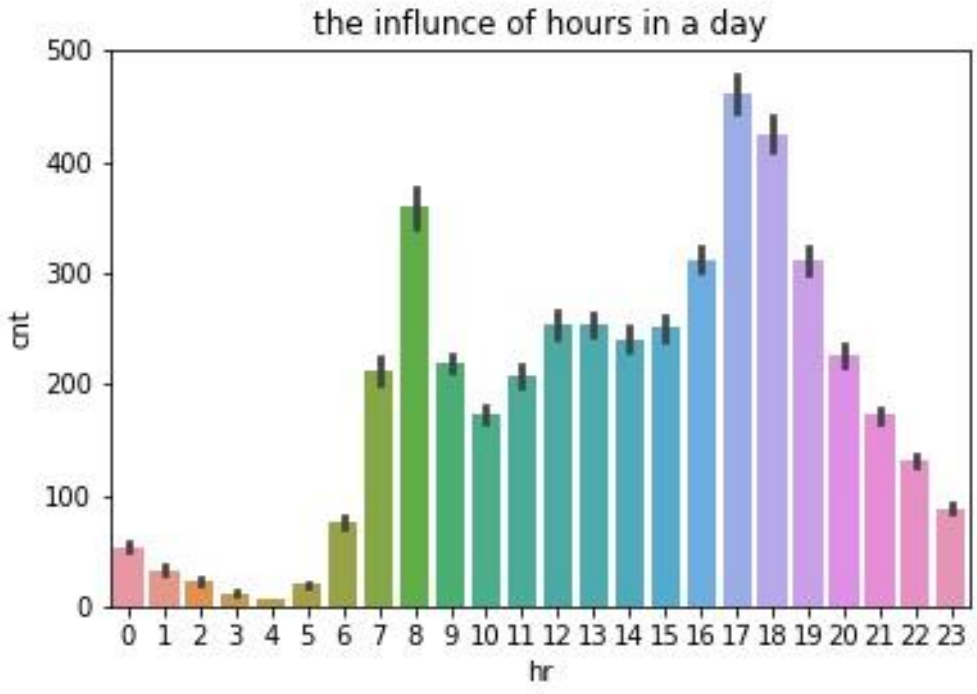

Figure 5

2.4. Data Quality Audit

I used ‘dataframe.describe().show()’ to check the overall data and its quality is pretty good.

Most fields are recorded well without null value, except 5 fields with missing values (3968 missing values for ‘instant’, and 500, 724, 715 and 7 for temp, atemp, hum, and cnt respectively).

```
# data audit
df.describe('instant', 'temp', 'atemp', 'hum', 'cnt').show(1)
```

```
+-------+-------+-----+-----+-----+-----+
|summary|instant| temp|atemp|  hum|  cnt|
+-------+-------+-----+-----+-----+-----+
|  count|  13411|16879|16655|16664|17372|
+-------+-------+-----+-----+-----+-----+
```

So according to data exploration and data audit result, the following steps will:

- Remove fields with more than 20% missing values
- Remove row with missing ‘cnt’ values as it acts as role of target
- Fill null values for other fields
- Ignore ‘casual’, ‘registered’, and ‘dteday’ fields as they do not make sense for the mining goal.

3. Data Preparation

According to the above exploration of the data, I will start to conduct the work of data preparation in this chapter, including data select, data clean, data construct and data format, to deal with the data quality issues mentioned in chapter 2.4.

3.1. Select the data

In general, there are two ways to select data: selecting rows and selecting fields.

For the fields selection, as analyzed in the previous section, 'dteday', 'registered', 'causal' do not make sense for the data mining goals, because this project focuses only on the prediction of the total rental numbers. So we can just disposal these 3 features in the following data manipulation.

Besides, we have found that the impact of the season on the total rentals, is pretty similar to that of the month, see Figure 5. As the month is more detailed, we can also ignore the 'season' field in following process.

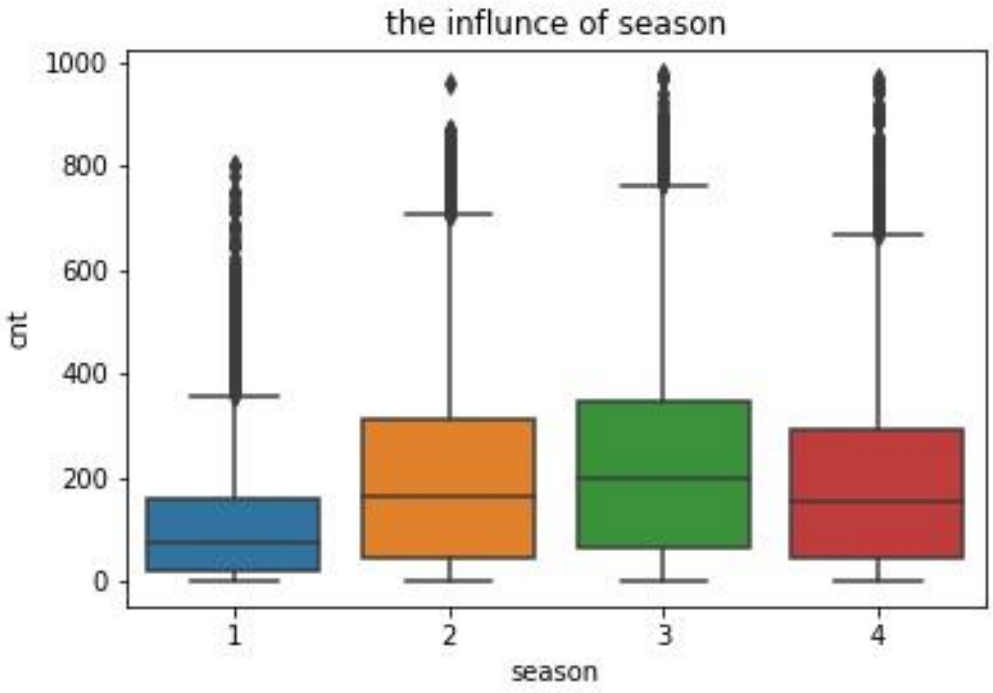

Figure 5

Furthermore, fields 'holiday', 'weekday' and 'workingday' are all time features about working or not, maybe we should check the impact of working. When further exploring the data, I find that the rental counts are almost the same each day in a week, see Figure 6. So it is reasonable to ignore this feature.

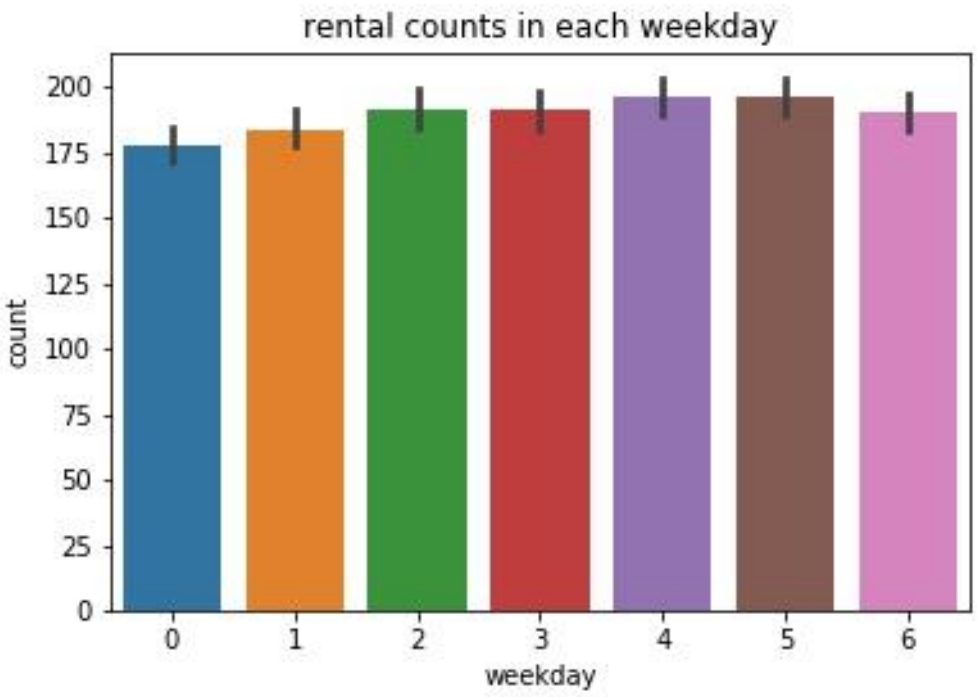

Figure 6 Rental counts in each weekday

Finally, I select data shown in Figure 7:

```
df_selected = df.drop('dteday', 'registered', 'casual', 'season', 'weekday')
df_selected.printSchema()
```

```
root
 |-- instant: integer (nullable = true)
 |-- yr: integer (nullable = true)
 |-- mnth: integer (nullable = true)
 |-- hr: integer (nullable = true)
 |-- holiday: integer (nullable = true)
 |-- workingday: integer (nullable = true)
 |-- weathersit: integer (nullable = true)
 |-- temp: double (nullable = true)
 |-- atemp: double (nullable = true)
 |-- hum: double (nullable = true)
 |-- windspeed: double (nullable = true)
 |-- cnt: integer (nullable = true)
```

Figure 7 Data Selected

3.2. Clean the data

As the result of Data Audit, there is some Null data in ‘instant’, ‘temp’, ‘atemp’ , ‘hum’ and ‘cnt’ fields. In the process of data cleaning, I need to remove ‘instant’ fields as it has more than 20% missing values, and rows where the ‘cnt’ values are null.

As there are only 500, 724, 715 missing values for temp, atemp, and hum features, it is better to impute them with mean value of these fields.

```
# drop feature 'instant' and remove rows where 'cnt' is null
df_cleaned = df_selected.drop('instant')
```

```
df_cleaned = df_cleaned.na.drop(subset='cnt')
df_cleaned.describe('cnt').show(1)
df_cleaned.count()

# fill null values with mean of values in 'temp', 'atemp', 'hum'
from pyspark.sql.functions import mean
mean_temp = df.select(mean(df.temp)).collect()[0][0]
mean_atemp = df.select(mean(df.atemp)).collect()[0][0]
mean_hum = df.select(mean(df.hum)).collect()[0][0]
mean = {'temp': mean_temp, 'atemp': mean_atemp, 'hum': mean_hum}
df_cleaned = df_cleaned.na.fill(mean)

df_cleaned.describe('temp', 'atemp', 'hum', 'cnt').show(1)
```

```
+-------+-----+-----+-----+-----+
|summary| temp|atemp|  hum|  cnt|
+-------+-----+-----+-----+-----+
|  count|17372|17372|17372|17372|
+-------+-----+-----+-----+-----+
```

After data cleaning, there are total 17372 records without null values. Now we have dealt with data quality issues mentioned by Section 2.4.

3.3. Construct the data

In summary, I will extract 10 features including year, month, hour, workingday, holiday, weather, temp, atemp, humidity, and windspeed for further data mining process, where temp, atemp, humidity, and windspeed are continuous variables, while year, month, hour, workingday, holiday, and weather are categorical variables.

Since workingday and holiday are already binary values, the remaining discrete variables require transformation by a one-hot encoder, which converts the various categories into a single vector, and makes it easier to process in the following data mining process.

```
from pyspark.ml.feature import OneHotEncoder
# one hot encode: convert numbers into a vector
mnthEncoder = OneHotEncoder(inputCol='mnth', outputCol='mnthVec')
hrEncoder = OneHotEncoder(inputCol='hr', outputCol='hrVec')
weatherEncoder = OneHotEncoder(inputCol='weathersit', outputCol='weatherVec')

from pyspark.ml import Pipeline
pipeline = Pipeline(stages = [hrEncoder, mnthEncoder, weatherEncoder])
df_constructed = pipeline.fit(df_cleaned).transform(df_cleaned)

df_constructed.printSchema()
```

Figure 8 shows the data information after cleaning and constructing.

```
root
 |-- yr: integer (nullable = true)
 |-- mnth: integer (nullable = true)
 |-- hr: integer (nullable = true)
 |-- holiday: integer (nullable = true)
 |-- workingday: integer (nullable = true)
 |-- weathersit: integer (nullable = true)
 |-- temp: double (nullable = false)
 |-- atemp: double (nullable = false)
 |-- hum: double (nullable = false)
 |-- windspeed: double (nullable = true)
 |-- cnt: integer (nullable = true)
 |-- hrVec: vector (nullable = true)
 |-- mnthVec: vector (nullable = true)
 |-- weatherVec: vector (nullable = true)
```

Figure 8 Data Cleaned and Constructed

3.4. Integrate various data sources

As mentioned before, the original source of this data has been combined with corresponding weather and holiday schedule information into one single csv file. Therefore, I don't need to integrate any other data source.

3.5. Format the data

In this project, I should use the regression model where all data needed should be numeric type. Before data exploration, I have checked the type of all features imported and formatted them according to Table 1 like Figure 9. So no further format need to be done at this section.

```
root
 |-- instant: integer (nullable = true)
 |-- dteday: timestamp (nullable = true)
 |-- season: integer (nullable = true)
 |-- yr: integer (nullable = true)
 |-- mnth: integer (nullable = true)
 |-- hr: integer (nullable = true)
 |-- holiday: integer (nullable = true)
 |-- weekday: integer (nullable = true)
 |-- workingday: integer (nullable = true)
 |-- weathersit: integer (nullable = true)
 |-- temp: float (nullable = true)
 |-- atemp: float (nullable = true)
 |-- hum: float (nullable = true)
 |-- windspeed: float (nullable = true)
 |-- casual: integer (nullable = true)
 |-- registered: integer (nullable = true)
 |-- cnt: integer (nullable = true)
```

Figure 9 Data Formatted in Section 2.2

4. Data Transformation

4.1. Reduce the data

After data preparation of Chapter 3, I have encoded three categorical features (‘mnth’, ‘hr’, ‘weathersit’) with One-Hot method, so they are redundant and useless for further data mining progress, then can be removed in this section, see Figure 10

```
# Reduce the data
df_reduced = df_constructed.drop('mnth', 'hr', 'weathersit')
```

```
root
 |-- yr: integer (nullable = true)
 |-- holiday: integer (nullable = true)
 |-- workingday: integer (nullable = true)
 |-- temp: double (nullable = false)
 |-- atemp: double (nullable = false)
 |-- hum: double (nullable = false)
 |-- windspeed: double (nullable = true)
 |-- cnt: integer (nullable = true)
 |-- hrVec: vector (nullable = true)
 |-- mnthVec: vector (nullable = true)
 |-- weatherVec: vector (nullable = true)
```

Figure 10 Data Reduced

4.2. Project the data

Before Spark can accept the data for machine learning. The data needs to be transformed with two columns: label and features. Therefore, in this section, I will combine all of the features into a single vector with Vector Assembler method and just select ‘cnt’ and ‘features’ as the final data for data mining, see Figure 11.

```
from pyspark.ml.feature import VectorAssembler
featuresCol = df_reduced.drop('cnt').columns
assembler = VectorAssembler(inputCols = featuresCol, outputCol = 'features')
df_projected = assembler.transform(df_reduced)

final_data = df_projected.select('cnt', 'features')
final_data.show(5)
```

```
+---+--------------------+
|cnt|            features|
+---+--------------------+
| 16|(46,[3,4,5,7,31,4...|
| 40|(46,[3,4,5,8,31,4...|
| 32|(46,[3,4,5,9,31,4...|
| 13|(46,[3,4,5,10,31,...|
|  1|(46,[3,4,5,11,31,...|
+---+--------------------+
only showing top 5 rows
```

Figure 11 Project the data

Finally, the data is transformed with predictor ‘cnt’ and one field ‘features’ including **46 features (will be studied later)** and it is ready for modeling now.

5. Data Mining Method Select

5.1. Match the goals to methods of data mining

Generally, there are three main types of modelling methods:

- Classification predicts discrete number of values. The data is categorized under different labels according to some parameters.
- Regression predicts the numeric data instead of labels. It can also identify the distribution trends based on the available data or historic data.
- Clustering is the task of partitioning the unlabeled dataset into groups where points within single cluster are very similar and points in different clusters are different.

The data mining goal for this project is using historical records about previous rentals to predict future bike-sharing demand trend.

5.2. Select the appropriate data-mining methods

In our dataset, the predictor 'cnt' is labeled, so Clustering is not an option. On the other hand, the type of 'cnt' variable is the rental counts of bikes, which is a continuous value. Besides, it needs existing historical data to make the prediction. Therefore, according to data mining goals, the best data mining method is Regression. I will choose the appropriate regression data-mining algorithms in the following data mining steps.

6. Data Mining Algorithm Select

6.1. Exploratory of algorithms

The data-mining goals of this project are predicting the possible bike-rental demands. As discussed in Chapter 5, there are a variety of regression algorithms for different data mining objectives. Representative algorithms are Random Forest Regression, Decision Tree Regression and Gradient Boosted Tree Regression.

Decision Tree algorithm can predict or classify future observations based on a set of decision rules. Its reasoning process is clear to understand when searching for patterns.

Random Forest and Gradient-boosted Trees are popular algorithms using ensembles of decision trees. Both of them are based on decision tree and they perform better than single decision tree algorithm.

6.2. Select algorithm

All of 3 regression algorithms mentioned in Section 6.1 can be used to reach the data-mining goal, while Decision Tree may have better efficiency and the other two (ensembles of decision trees) may gain better performance.

To find the best algorithm, I test them with data extracted from Chapter 4.2, and will use R2 (coefficient of determination) to access their performances.

```
# select algorithm
from pyspark.ml.regression import (RandomForestRegressor,
                                   GBTRegressor, DecisionTreeRegressor)
# create evaluator with R2
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(labelCol='cnt', predictionCol='prediction',
                                metricName='r2')
# for performance visualization
import numpy as np
import matplotlib.pyplot as plt
```

6.3. Build Models

To test models created more efficiently, I subset a sample containing only 10% of the data for the following model creating and accessing.

```
#create a sample for model test
sample, x = final_data.randomSplit([0.1, 0.9])
```

For each model, I will test different parameters as below:

Decision Tree Regression model with maxDepth: 3, 6, 9, ..., 30

```
r2_dtr = np.zeros(10)
for i in np.arange(10):
    dtr = DecisionTreeRegressor(labelCol='cnt', maxDepth= (i+1)*3)
    dtrModel = dtr.fit(sample)
    prediction_dtr = dtrModel.transform(sample)
    r2_dtr[i] = evaluator.evaluate(prediction_dtr)
plt.plot(np.arange(3, 33, 3), r2_dtr)
```

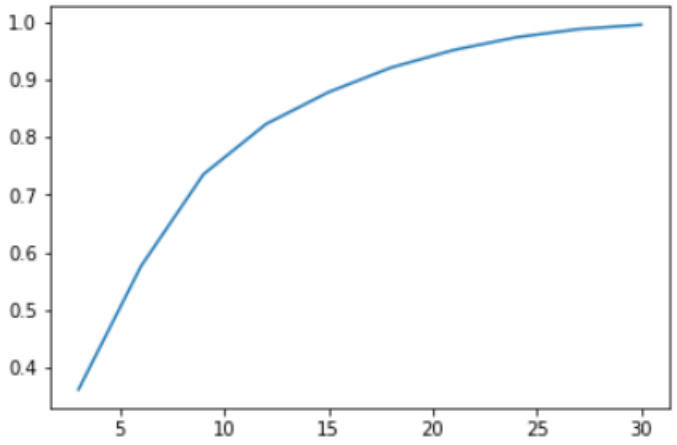

Figure 12 Decision Tree Regression with different Depth

Random Forest Regression model with maxDepth: 3, 6, 9, ..., 30

```
r2_rfr = np.zeros(10)
for i in np.arange(10):
    rfr = RandomForestRegressor(labelCol='cnt', maxDepth=(i+1)*3)
    rfrModel = rfr.fit(sample)
    prediction_rfr = rfrModel.transform(sample)
    r2_rfr[i] = evaluator.evaluate(prediction_rfr)
plt.plot(np.arange(3, 33, 3), r2_rfr)
```

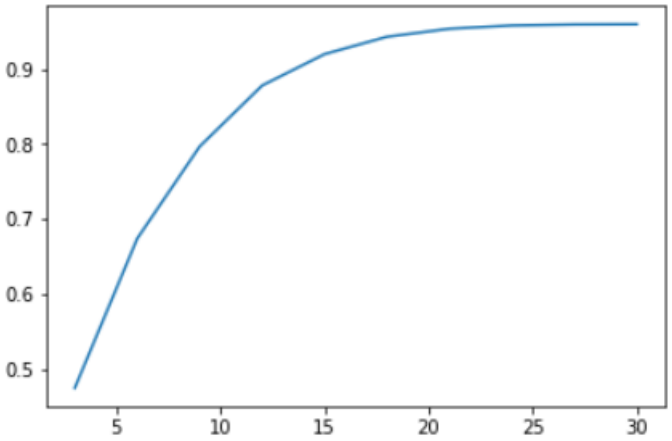

Figure 13 Random Forest Regression with different Depth

From Figure 12 and Figure 13, these 2 modesl can meet the data mining goal (R2 score greater than 0.85) when max depth reaches 15, and will rise slower after 20, so I will choose 20 as their maxDepth parameter.

Gradient Boosted Trees Regression model with maxIter: 10, 20, 30, ..., 100

```
r2_gbt = np.zeros(10)
for i in np.arange(10):
    gbt = GBTRegressor(labelCol='cnt', maxIter = (i+1)*10)
    gbtModel = gbt.fit(sample)
    prediction_gbt = gbtModel.transform(sample)
```

```
        r2_gbt[i] = evaluator.evaluate(prediction_gbt)
plt.plot(np.arange(10, 105, 10), r2_gbt)
```

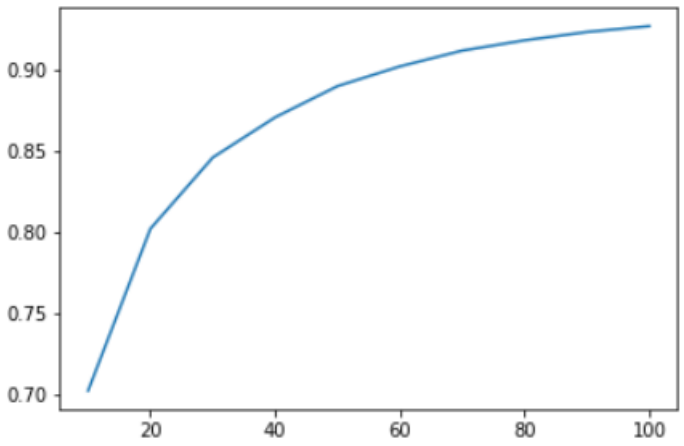

Figure 14 Gradient Boosted Trees with different maxIter

Figure 14 shows that Gradient Boosted Tree Regression can meet the data mining goal when max iteration reaches 50, and will rise slower after 80, therefore, I will choose 80 as its maxIter parameter.

As the algorithms tested above, all the models meet the success criteria with high correlation of rental counts at the chosen parameter. Therefore, I build Models as below for further data mining.

Decision Tree Regression model with maxDepth 20:

```
DTR = DecisionTreeRegressor(labelCol='cnt', maxDepth=25)
```

Gradient Boosted Trees Regression model with maxIter 80:

```
GBT = GBTRegressor(labelCol='cnt', maxIter = 80)
```

Random Forest Regression model with maxDepth20:

```
RFR = RandomForestRegressor(labelCol='cnt', maxDepth=20)
```

7. Data Mining

7.1. Create and justify test designs

In Section 6.1 I train and test 3 models based on the same sample data, which is considerably too optimal as a model make predictions better based on the data where it was fitted.

Therefore, we need split data randomly into train and test sets, which is important to evaluate models. Usually we use most of the data for training, and split them randomly makes sure the similarity between train and test sets.

Normally, we can partition the data set into two parts: 70% as training set and 30% as test set. I also created test design with 80% for training and 20% for testing in the iteration of this step and the output shows they can gain almost the same results. As fitting model takes more times than testing, it is better to use smaller training set to balance the output and the efficiency.

```
# split data into train and test
train, test = final_data.randomSplit([0.7, 0.3])
```

7.2. Conduct data mining

In this section, with the decision tree regression model and a gradient boosted trees regression model created in Section 6.3, I train models on training set and output the predictions based on test set. From the summary of below, these models' predictions are pretty fitted.

Decision Tree Regression model and prediction summary:

```
DTR = DecisionTreeRegressor(labelCol='cnt', maxDepth=25)
DTRmodel = DTR.fit(train)
prediction_DTR = DTRmodel.transform(test)
# show the summary
prediction_DTR.describe().show()
```

```
+-------+------------------+------------------+
|summary|               cnt|        prediction|
+-------+------------------+------------------+
|  count|              5215|              5215|
|   mean| 190.4999041227229|190.82029936999675|
| stddev|181.10015716607145|176.26218216968576|
|    min|                 1|               1.0|
|    max|               968|             977.0|
+-------+------------------+------------------+
```

Gradient Boosted Trees Regression model and prediction summary:

```
GBT = GBTRegressor(labelCol='cnt', maxIter = 80)
GBTmodel = GBT.fit(train)
prediction_GBT = GBTmodel.transform(test)
```

```
+-------+------------------+------------------+
|summary|               cnt|        prediction|
+-------+------------------+------------------+
|  count|              5215|              5215|
|   mean| 190.4999041227229| 189.9079751686032|
| stddev|181.10015716607145|167.89031165426218|
|    min|                 1|-77.10967793918589|
|    max|               968| 906.5719648570255|
+-------+------------------+------------------+
```

Random Forest Regression model and prediction summary:

```
RFR = RandomForestRegressor(labelCol='cnt', maxDepth=20)
RFRmodel = RFR.fit(train)
prediction_RFR = RFRmodel.transform(test)
```

```
+-------+------------------+------------------+
|summary|               cnt|        prediction|
+-------+------------------+------------------+
|  count|              5288|              5288|
|   mean|191.91509077155825| 192.5416922123185|
| stddev|184.23537222941195| 164.7192362503229|
|    min|                 1|2.6356823510388874|
|    max|               977| 923.4961038961037|
+-------+------------------+------------------+
```

7.3. Search for patterns

Firstly, I check the contributions of all the 46 features to the predictions that models made in below list, where they show the similar pattern. Importance of feature 4 is highest with importance 0.19, 0.13, 0.12 in three models respectively, followed by feature 0, 2, 3, 5, all of which are greater than 0.05. Features 15, 24, 25 in Decision Tree and Random Forest models also have high importance greater than 0.05, but in Gradient Boosted Tree their evaluations are pretty low.

Decision Tree Regression model:

```
# check the importance of each feature
DTRmodel.featureImportances
```

```
SparseVector(46, {0: 0.0862, 1: 0.0021, 2: 0.1025, 3: 0.0566, 4: 0.1887, 5: 0.0696, 6: 0.0143, 7: 0.0159, 8: 0.0099, 9: 0.0141, 10: 0.017, 11: 0.0152, 12: 0.0148, 13: 0.0085, 14: 0.0208, 15: 0.0608, 16: 0.0127, 17: 0.0023, 18: 0.0023, 19: 0.0037, 20: 0.0021, 21: 0.001, 22: 0.0008, 23: 0.0141, 24: 0.0938, 25: 0.0848, 26: 0.0351, 27: 0.0067, 28: 0.0024, 29: 0.0025, 31: 0.0022, 32: 0.0028, 33: 0.002, 34: 0.0018, 35: 0.0017, 36: 0.0009, 37: 0.003, 38: 0.0005, 39: 0.0029, 40: 0.005, 41: 0.0038, 43: 0.0028, 44: 0.0012, 45: 0.0078})
```

Random Forest Regression model:

```
# check the importance of each feature
RFRmodel.featureImportances
```

```
SparseVector(46, {0: 0.0705, 1: 0.0022, 2: 0.0842, 3: 0.1028, 4: 0.1304, 5: 0.0917, 6: 0.0185, 7: 0.0212, 8: 0.0162, 9: 0.0185, 10: 0.0294, 11: 0.0269, 12: 0.0214, 13: 0.0067, 14: 0.011, 15: 0.061, 16: 0.0075, 17: 0.0024, 18: 0.0026, 19: 0.0033, 20: 0.0032, 21: 0.0026, 22: 0.0032, 23: 0.0112, 24: 0.099, 25: 0.0692, 26: 0.0233, 27: 0.0063, 28: 0.0018, 29: 0.0036, 31: 0.0029, 32: 0.0023, 33: 0.0029, 34: 0.0023, 35: 0.0019, 36: 0.0017, 37: 0.0037, 38: 0.0015, 39: 0.0041, 40: 0.0042, 41: 0.0033, 43: 0.0044, 44: 0.0034, 45: 0.0094})
```

Gradient Boosted Trees model:

```
# check the importance of each feature
```

```
GBTmodel.featureImportances
```

```
SparseVector(46, {0: 0.0747, 1: 0.0109, 2: 0.2143, 3: 0.0835, 4: 0.1224, 5: 0.0721, 6: 0.0306, 7: 0.0124, 8: 0.0122, 9: 0.0152, 10: 0.0153, 11: 0.0161, 12: 0.0125, 13: 0.0071, 14: 0.0124, 15: 0.0334, 16: 0.0117, 17: 0.0107, 18: 0.0089, 19: 0.0129, 20: 0.0112, 21: 0.0111, 22: 0.0123, 23: 0.0182, 24: 0.0171, 25: 0.023, 26: 0.0196, 27: 0.0112, 28: 0.0074, 29: 0.0029, 31: 0.0079, 32: 0.0089, 33: 0.0081, 34: 0.0046, 35: 0.0016, 36: 0.0005, 37: 0.002, 38: 0.0039, 39: 0.0061, 40: 0.0076, 41: 0.0059, 43: 0.0034, 44: 0.0006, 45: 0.0156})
```

Besides, I check the procedure of tree decision-making with Decision Tree as it is easier to understanding. From below structure, the decision is made started from feature 4, followed by feature 15, 24, 25, 3, 0, 2, which matches what I discussed above.

```
# search for patterns
# check the decision tree
print(DTRmodel.toDebugString)
```

```
DecisionTreeRegressionModel (uid=DecisionTreeRegressor_4afb8dc024bfa2e13546) of depth 20 with 10419 nodes
  If (feature 4 <= 0.5909)
   If (feature 15 in {0.0})
    If (feature 24 in {0.0})
     If (feature 25 in {0.0})
      If (feature 3 <= 0.28)
       If (feature 16 in {0.0})
        If (feature 14 in {0.0})
         If (feature 26 in {0.0})
          If (feature 17 in {0.0})
           If (feature 19 in {0.0})
            If (feature 5 <= 0.45)
             If (feature 0 <= 0.0)
              If (feature 8 in {1.0})
               If (feature 2 <= 0.0)
                If (feature 3 <= 0.16)
                 Predict: 12.0
                Else (feature 3 > 0.16)
                 Predict: 16.0
```

We can check all the contribution of all the 46 features to the predictions that the model makes in above list, where it matches what I discussed above, importance of feature 4 is highest, followed by feature 2, 24, 25, 0, 3, 15.

Yet the pattern found is hard to understand with features only labelled with number index. Therefore, I will try to figure out what these features are and what can we conclude from the pattern we found.

Before modelling, I encoded the data with One-Hot method, where it contains 3 features ‘hrVec’ with 23 binary elements, ‘mnthVec’ with 12 elements, and ‘weatherVec’ with 4 elements, see below data structure after One-Hot encoding.

```
# before vector assemble
df_reduced.show(5)
```

```
+---+-------+----------+----+------------------+------------------+---------+---+-------------+-------------+-------------+
| yr|holiday|workingday|temp|             atemp|               hum|windspeed|cnt|         hrVec|       mnthVec|   weatherVec|
+---+-------+----------+----+------------------+------------------+---------+---+-------------+-------------+-------------+
|  0|      0|         0|0.24|            0.2879|              0.81|      0.0| 16|(23,[0],[1.0])|(12,[1],[1.0])|(4,[1],[1.0])|
|  0|      0|         0|0.22|0.4771872830981645|               0.8|      0.0| 40|(23,[1],[1.0])|(12,[1],[1.0])|(4,[1],[1.0])|
|  0|      0|         0|0.22|            0.2727|0.6233845415266452|      0.0| 32|(23,[2],[1.0])|(12,[1],[1.0])|(4,[1],[1.0])|
|  0|      0|         0|0.24|            0.2879|              0.75|      0.0| 13|(23,[3],[1.0])|(12,[1],[1.0])|(4,[1],[1.0])|
|  0|      0|         0|0.24|            0.2879|              0.75|      0.0|  1|(23,[4],[1.0])|(12,[1],[1.0])|(4,[1],[1.0])|
+---+-------+----------+----+------------------+------------------+---------+---+-------------+-------------+-------------+
```

Then I transformed data with vector assembler which convert all features into one vector containing all the 46 features, which are ‘yr’, ‘holiday’, ‘workingday’, ‘temp, ‘atemp’, ‘hum’,

'windspeed', 'hr' (23 binary features), 'mnth' (12 binary features) and 'weathersit' (4 binary features. Below I extract 'features' column from vector-assembled data and compared with the data before vector assembling.

```
# after vector assemble
df_projected.select('features').collect()[1:5]
```

```
[Row(features=SparseVector(46, {3: 0.24, 4: 0.2879, 5: 0.81, 7: 1.0, 31: 1.0, 43: 1.0})),
 Row(features=SparseVector(46, {3: 0.22, 4: 0.4772, 5: 0.8, 8: 1.0, 31: 1.0, 43: 1.0})),
 Row(features=SparseVector(46, {3: 0.22, 4: 0.2727, 5: 0.6234, 9: 1.0, 31: 1.0, 43: 1.0})),
 Row(features=SparseVector(46, {3: 0.24, 4: 0.2879, 5: 0.75, 10: 1.0, 31: 1.0, 43: 1.0})),
 Row(features=SparseVector(46, {3: 0.24, 4: 0.2879, 5: 0.75, 11: 1.0, 31: 1.0, 43: 1.0}))]
```

Therefore, I can confirm that the index in feature importance can be labelled with the actual features like below:

Index	0	1	2	3	4	5	6	7-29	31-42	43-46
Feature	Year	Holiday	Work	Temp	Atemp	Hum	Wind	Hour	Mnth	weather

After the features are labelled, we can search for patterns much easier. As we discussed before, feature 4 'atemp' (apparent temperature) plays the most important role for the prediction, , followed by feature 0, 3, 5 (year, temperature, humidity respectively). And features 15, 24, 25 (hour features) also contribute much, which matches the discussion we have done in Chapter 2, where we have checked the rental counts based on year, hours, and weather situations.
However, I also notice that all these 3 models consider feature 2 'workingday' as another important variable impacting the rental behaviors, which we have not investigated before. Thus, in the following chapter, I will explore the data to check more about the counts based on working day or not.

8. Interpretation

8.1. Study the mined patterns

The pattern we searched in section 7.3 suggested that both weather and time factors play important roles impacting the total rental demand. In Chapter 2, we have checked the rental counts based on years, hours in a day, and weather situations.

In this section, I will study more about the working day and try to confirm its importance for rental behaviors. Besides although the feature 'holiday' and 'weekday' are considered less important in models, they actually are some aspects of the binary feature working day (work or not), Therefore, I will also take holidays or week days into consideration, which we did not check before.

I build a plot to demonstrate the difference between working days and non-working days and we can see that 'workingday', 'holiday' and 'weekday' have the same pattern against rental counts. Figure 15 shows that working days have high rush-hour rental counts, and low counts in the rest of a day, while the rental counts are high in noon and afternoon of non-working days (including holidays and weekends), which meets the actual people's rental behavior.

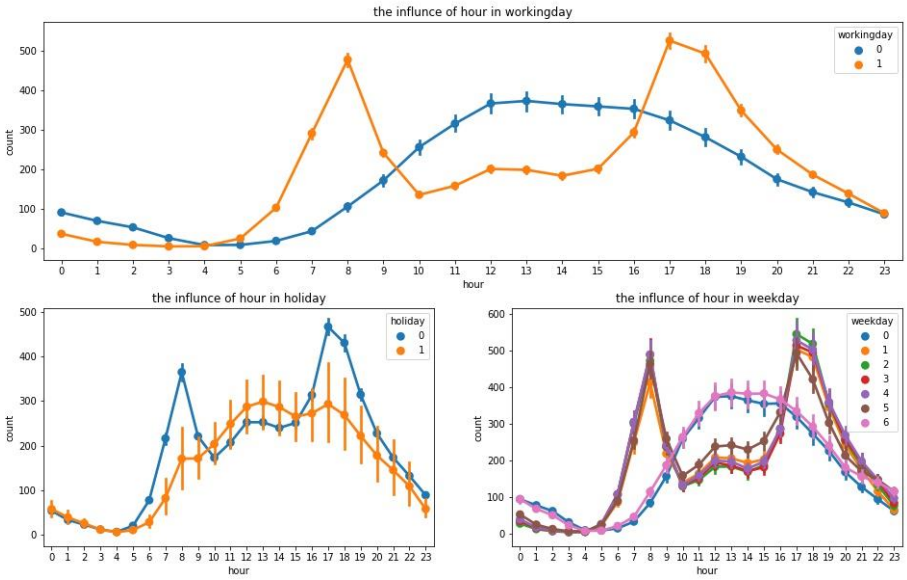

Figure 15 Working or Not

8.2. Visualize the data, results, models and patterns

Figure 16, Figure 17 and Figure 18 show the importance of how each feature on the total sharing bike rental counts predicted by Decision Tree Regression, Random Forest Regression and Gradient Boosted Trees respectively.

```
plt.plot(DTRmodel.featureImportances.values)
plt.plot(RFRmodel.featureImportances.values)
plt.plot(GBTmodel.featureImportances.values)
```

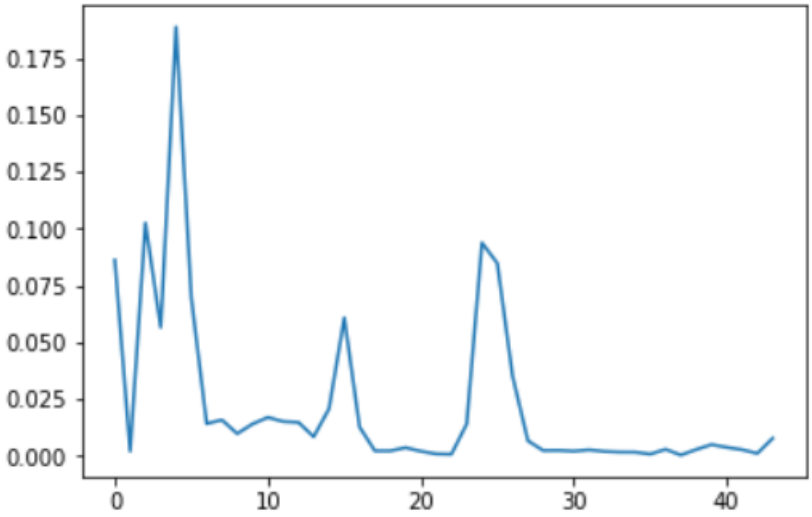

Figure 16 Feature Importance of DTR model

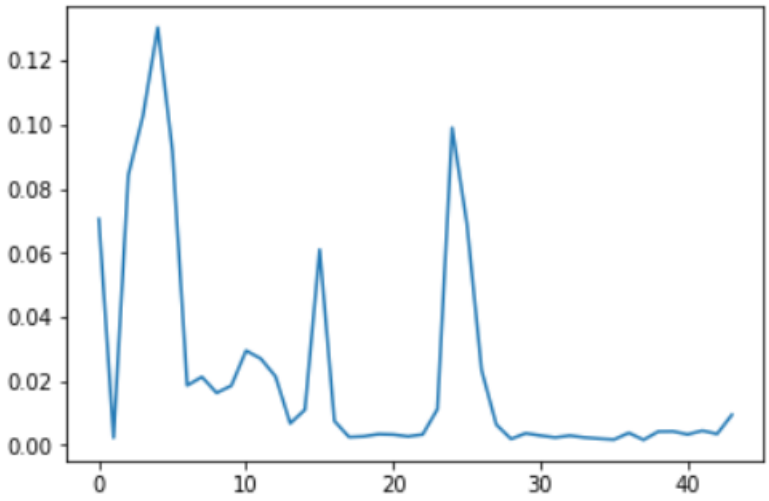

Figure 17 Feature Importance of RFR model

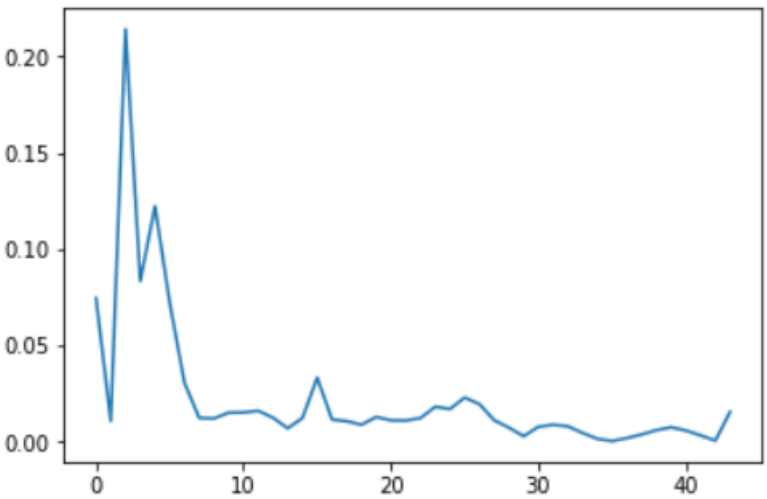

Figure 18 Feature Importance of GBT model

These raw visualizations are hard to understand with features only labelled with number index.

Therefore, I need label the index of each importance element with its actual feature as we discussed in Section 7.3 like below:

Index	0	1	2	3	4	5	6	7-29	31-42	43-46
Feature	Year	Holiday	Work	Temp	Atemp	Hum	Wind	Hour	Mnth	weather

Then for the features which are encoded by One-Hot method before, I can group them together and draw some figures which clearly demonstrate the contributions of all the actual features to the prediction of our data mining goal: the future rental demand of sharing bikes.

```
# label the features and visualization:
features = ('year', 'hday', 'wday', 'temp', 'atemp', 'hum', 'wind',
                    'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr',
                    'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr', 'hr',
                    'mnth', 'mnth', 'mnth', 'mnth', 'mnth', 'mnth', 'mnth','mnth',
                    'mnth', 'mnth', 'mnth', 'mnth', 'wsit', 'wsit', 'wsit', 'wsit')
plt.bar(features ,DTRmodel.featureImportances)
plt.bar(features ,RFRmodel.featureImportances)
plt.bar(features ,GBTmodel.featureImportances)
```

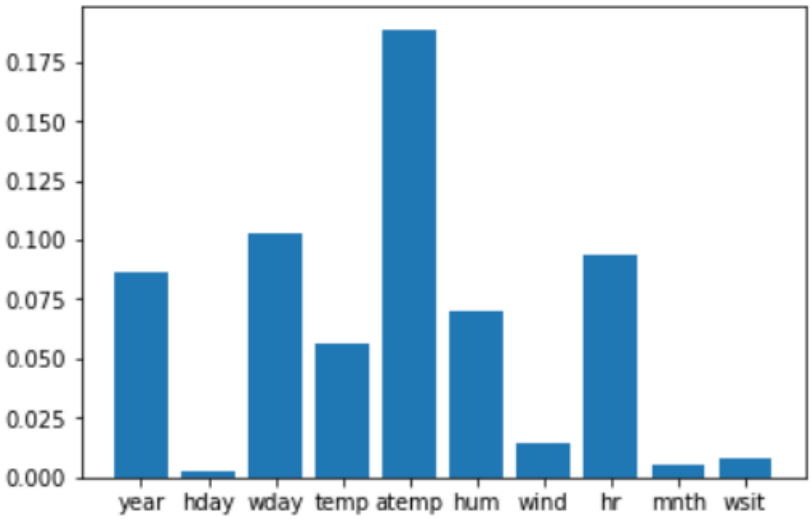

Figure 19 Feature Importance of DTR model

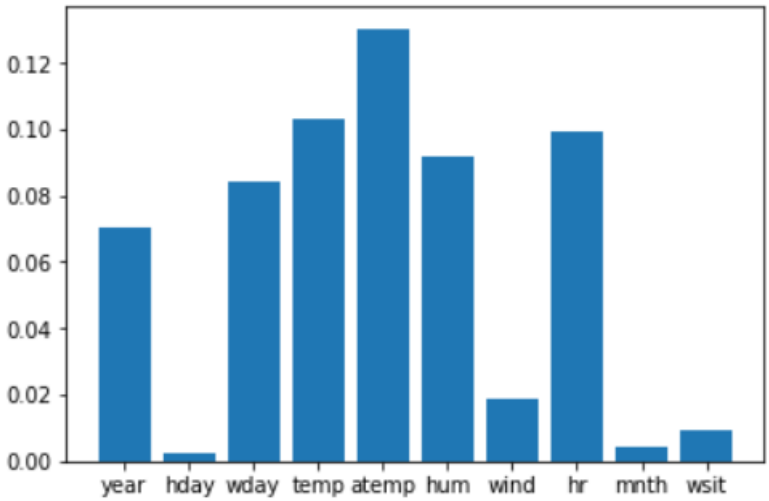

Figure 20 Feature Importance of RFR model

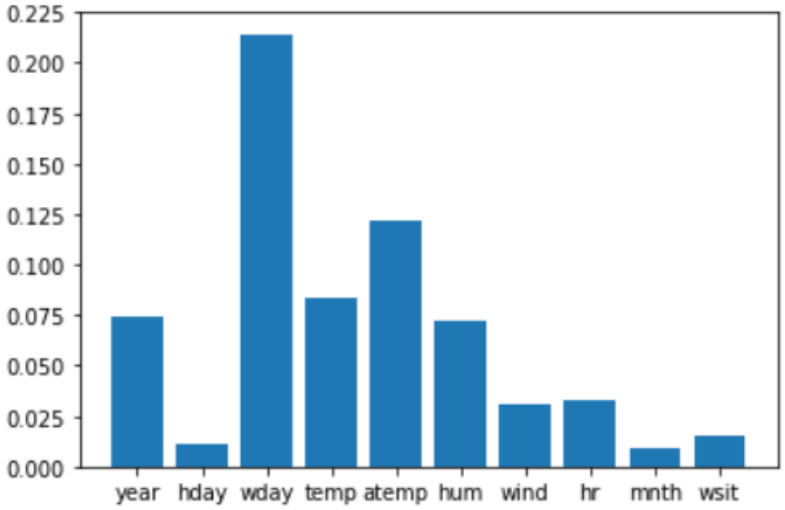

Figure 21 Feature Importance of GBT model

8.3. Interpret the data, results, models and patterns

In previous steps we have built some significant plots to visualize and interpret the relationships between different features and the bike rental counts in our data set.

Actually, if we further check the casual and registered rental counts respectively, the familiar pattern happens, see Figure 22. It may indicate that registered users are mainly made up of the commuters, whose behavior is more routine and regular. The unregistered users' behavior is more casual, but still has clear pattern to be identified and predicted.

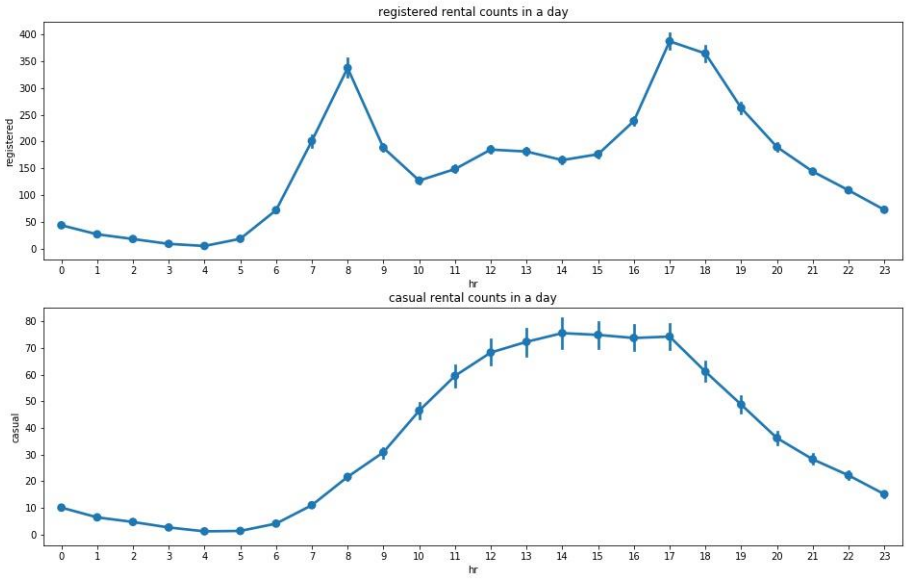

Figure 22 Registered or Casual

This model is generated based on total rental counts prediction where registered users take a large proportion than casual scenarios. However, the pattern found about the rental behavior in non-working days, or that of unregistered users may also be useful, as it provides a clue to

detect some abnormal fluctuation of rental counts, like during some specific holidays, celebrations or important events. Such prediction is necessary and useful to improve the flexibility of the bike-sharing market management.

Once we identify the behavior pattern of rental users, we can release the sharing-bikes dynamically to reach a better input-output ratio. What's more, government then is able to manage and the bike-sharing market more efficiently, regulate the market operation mode, and improve the market mechanism.

8.4. Assess and evaluate results, models and patterns

With the fitted model based on training set, we can check the coefficient of determination of predicted total rental counts with test set.

```
#evaluate model
r2_GBT = evaluator.evaluate(prediction_GBT)
r2_DTR = evaluator.evaluate(prediction_DTR)
r2_RFR = evaluator.evaluate(prediction_RFR)
print('R2 Score of GBT Regression: ', r2_GBT)
print('R2 Score of Decision Tree Regression: ', r2_DTR)
print('R2 Score of Random Forest Regression: ', r2_RFR)
```

```
R2 Score of GBT Regression:  0.891504171558418
R2 Score of Decision Tree Regression:  0.8082922886030754
R2 Score of Random Forest Regression:  0.8788998534479309
```

We can see that all models' R2 score drop, especially the decision tree regression model, which is 0.808, less than 0.85 and cannot meet the success criteria anymore. Although Random Forest Regression Model get R2 score 0.879, its performance is not the best. So finally, we will choose Gradient Boosted Trees Regression model (scored 0.892) for this project.

8.5. Iterate prior steps

During the whole steps of this project, I made several iterations to make sure the data mining process is more effective and has better performance. They are documented as below:

- Iteration 1. Business Understanding:
  First of all, it is always the most important part to find a meaningful topic for the project. I studied UN 17 Sustainable Goals and explored several open data source websites to find an interesting topic. As I noticed sharing economy has become so popular but bike sharing in China met a serious problem due to rapid expansion with competition, bring a huge waste of resources and disrupting the urban order.
  So finally I choose this topic and set the business goal and data mining goal accordingly.
- Iteration 2. Data Understanding:
  After the data set was imported, I found all data types were 'string' type so I re-imported data with structured types according to the data description from the data source website. When studying patterns, I also re-explored the data to find impacting patterns of working or holidays, registered rental or casual rental.
- Iteration 3. Data Preparation:
  When selecting data, I went back to data exploration to check the impact of 'season' and 'weekday' features. The impact of the season on the total rentals is pretty similar to that

of the month and the rental counts are almost the same each day in a week. Therefore, I ignored them in this chapter.

To process the categorical features effectively, I converted the various categories into a single vector with one-hot encode at data constructing section.

- Iteration 4. Data Transformation:

  The smaller data will make models running more efficiently, so I removed some redundant features when reducing the data.

  For machine learning models, Spark requires the data containing with two columns: label and features. Therefore, before conducting data mining, I combined all of the features into a single vector in this chapter.

- Iteration 5. Data Mining Method Select:

  After exploring 3 main types of data mining method, I selected regression method for following data mining process in this chapter, as the data mining goal is using historical records to predict future trend (a continuous value). Classification methods are used for discrete predictors and Clustering methods are used for unlabeled predictors.

- Iteration 6. Data Mining Method Select:

  I explored 3 popular regression algorithms, tried several values for max Depth and max Iteration parameters, and visualized their performances based on selected parameters, to find the best balance between accuracy and efficiency.

- Iteration 7. Data Mining:

  When creating test designs, I changed training set to 80% and test set to 20% and conducted data mining.

```
# split data into train and test
train, test = final_data.randomSplit([0.8, 0.2])
# train the model
GBTmodel = GBT.fit(train)
prediction_GBT = GBTmodel.transform(test)
DTRmodel = DTR.fit(train)
prediction_DTR = DTRmodel.transform(test)
RFRmodel = RFR.fit(train)
prediction_RFR = RFRmodel.transform(test)
#evaluate model
r2_GBT = evaluator.evaluate(prediction_GBT)
r2_DTR = evaluator.evaluate(prediction_DTR)
r2_RFR = evaluator.evaluate(prediction_RFR)
```

```
R2 Score of GBT Regression:  0.8918535668709439
R2 Score of Decision Tree Regression:  0.8108998257884252
R2 Score of Random Forest Regression:  0.876682946194543
```

  It shows that the score of three models almost remain the same as previous test plan.

- Iteration 8. Interpretation:

  According to the pattern found in data mining, I re-explored the data to find impacting patterns of working or not (including weekdays and holidays). I also further studied the impacting pattern of registered rental or casual rental.

When visualized the models, I studied models deeply and group features to make my visualization more meaningful and clearer.

Disclaimer

I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright. (See: https://www.auckland.ac.nz/en/students/forms-policies-and-guidelines/student-policies-and-guidelines/academic-integrity-copyright.html).

I also acknowledge that I have appropriate permission to use the data that I have utilised in this project. (For example, if the data belongs to an organisation and the data has not been published in the public domain then the data must be approved by the rights holder.) This includes permission to upload the data file to Canvas. The University of Auckland bears no responsibility for the student's misuse of data.