<u>INDEX</u>

Experiment No.1

Reading and Writing Data

Up till now we've used data sets built into R or generated data by hand, but in the real world, you'll be working with data outside your R environment. Reading data into the R environment is often the first step in conducting data analysis. Data comes in many different forms. R is able to deal with most data formats, but in this lesson will focus on reading the common tabular data formats you are likely to encounter, such as comma separated values files (CSV) and Microsoft Excel files.

Before we can jump in and starting loading data, we need to learn a little bit about R's working directory and file paths. When you run R, it starts in a default location in your computer's file system called the working directory. You can check your working directory with the getwd() function.

```
getwd()
```

```
'/kaggle/working'
```

Since this guide is written in the Kaggle kernel environment, our working directory is a folder assigned to this kernel by Kaggle. If you are working in R on your local machine, the working directory will be a folder on your hard drive.

The working directory acts as your starting point for accessing other files on your computer. To load data into R, you either need to put the data file in your working directory, change your working directory to the folder containing the data or supply the data's file path to the data reading function.

You can change your working directory by supplying a new file path in quotes to the setwd().

```
setwd('/kaggle/') # Set a new working directory
```

```
getwd()         # Check the working directory again
```

```
'/kaggle'
```

*Note: in a local R environment, you can use forward slashes for your file path even in Windows which normally uses backslashes. If you want to use backslashes for file paths in Windows you should use double backslashes (\)

Instead of worrying about slashes in file paths, you can have R construct file paths for you using the file.path() function. It takes a comma separated sequence of character strings and then uses them to construct a file path string for you.

```
my_path <- file.path("/kaggle", "working")  # Construct path


setwd(my_path)                # Set the working directory to the path


getwd()                    # Check the working directory again
'/kaggle/working'
```

If you are working in a local R environment in R Studio, you can also change the working directory under the "Session" dropdown menu. Under session select "Set working directory", "Choose Directory", navigate to the folder you want to set as your working directory and click "Select folder."

You can list the files and folders in the current working directory using the list.files() function.

```
list.files("../input")
'draft2015' 'titanic'
```

There is only one file in the current working directory, but what about the Kaggle folder that we switched in and out of earlier?

*Note: For file paths, ".." means look one folder up from the current folder."

```
list.files("..")
'input' 'kaggle_bigquery.R' 'lib' 'src' 'working'
```

Here we see the Kagge folder holds the "working" folder that is our current working directory, as well as folders for "config", "input" and "lib." The "input" directory on Kaggle holds the data sets we've connected to our kernel.

```
list.files("../input")


list.files("../input/titanic")
'draft2015' 'titanic'
'gender_submission.csv' 'test.csv' 'train.csv'
```

Here we see the three data files for the Titanic competition.

Read CSV and TSV Files

Data is commonly stored in simple text files consisting of values delimited (separated) by a special character. For instance, CSV files use commas as the delimiter and tab separated value files (TSV) use tabs as the delimiter.

You can use the read.csv() function to read CSV files into R. This kernel is connected to the Titanic disaster data set, so we will use read.csv() to read it into our R session.

```
# Load the titanic data

titanic <- read.csv("../input/titanic/train.csv")

head(titanic)
```

```
A data.frame: 6 × 12
```

Experiment No.2

Using R as a calculator and descriptive statistics in R.

Calculating Descriptive Statistics in R

Here we will learn about how to perform the calculation of Descriptive Statistical metrics on a Data set and finally, we will create a data quality Report file. Let us start learning "calculating Descriptive Statistics in R".

We will follow each step as a Task for better understanding. It will also help us to complete all work in sequential tasks.

Task 1: Load and View data Set
It is better to confirm the working directory using getwd() and save your data in the working directory, or save the data in the required folder and then set the path of this folder (directory) in R using setwd() function.

```
getwd()
data <- read.csv("data.csv")
```

Task 2: Calculate Measure of Frequency Metrics
Before calculating the frequency metrics it is better to check the data structure and some other useful information about data, For example,

Note: here we are using mtcars data set.

```
data <- mtcars
str(data)
head(data)
length(data$cyl)
length(unique(data$cyl))
table(data$cyl)
freq <- table(data$cyl)
freq <- sort(freq, descreasing = T)
print(freq)
```

The above lines of code will tell you about the number of observations in the data set, frequency of cylinder variable, its unique category, and finally sorted frequency in order.

Experiment No.3

Reading and Writing Data

Up till now we've used data sets built into R or generated data by hand, but in the real world, you'll be working with data outside your R environment. Reading data into the R environment is often the first step in conducting data analysis. Data comes in many different forms. R is able to deal with most data formats, but in this lesson will focus on reading the common tabular data formats you are likely to encounter, such as comma separated values files (CSV) and Microsoft Excel files.

Before we can jump in and starting loading data, we need to learn a little bit about R's working directory and file paths. When you run R, it starts in a default location in your computer's file system called the working directory. You can check your working directory with the getwd() function.

```
getwd()
'/kaggle/working'
```

Since this guide is written in the Kaggle kernel environment, our working directory is a folder assigned to this kernel by Kaggle. If you are working in R on your local machine, the working directory will be a folder on your hard drive.

The working directory acts as your starting point for accessing other files on your computer. To load data into R, you either need to put the data file in your working directory, change your working directory to the folder containing the data or supply the data's file path to the data reading function.

You can change your working directory by supplying a new file path in quotes to the setwd().

```
setwd('/kaggle/') # Set a new working directory

getwd()        # Check the working directory again
'/kaggle'
```

*Note: in a local R environment, you can use forward slashes for your file path even in Windows which normally uses backslashes. If you want to use backslashes for file paths in Windows you should use double backslashes (\)

Instead of worrying about slashes in file paths, you can have R construct file paths for you using the file.path() function. It takes a comma separated sequence of character strings and then uses them to construct a file path string for you.

```
my_path <- file.path("/kaggle", "working")  # Construct path

setwd(my_path)              # Set the working directory to the path

getwd()                  # Check the working directory again
```

```
'/kaggle/working'
```

If you are working in a local R environment in R Studio, you can also change the working directory under the "Session" dropdown menu. Under session select "Set working directory", "Choose Directory", navigate to the folder you want to set as your working directory and click "Select folder."

You can list the files and folders in the current working directory using the list.files() function.

```
list.files("../input")
'draft2015' 'titanic'
```

There is only one file in the current working directory, but what about the Kaggle folder that we switched in and out of earlier?

*Note: For file paths, ".." means look one folder up from the current folder."

```
list.files("..")
'input' 'kaggle_bigquery.R' 'lib' 'src' 'working'
```

Here we see the Kagge folder holds the "working" folder that is our current working directory, as well as folders for "config", "input" and "lib." The "input" directory on Kaggle holds the data sets we've connected to our kernel.

```
list.files("../input")
```

```
list.files("../input/titanic")
'draft2015' 'titanic'
'gender_submission.csv' 'test.csv' 'train.csv'
```

Here we see the three data files for the Titanic competition.

Read CSV and TSV Files

Data is commonly stored in simple text files consisting of values delimited (separated) by a special character. For instance, CSV files use commas as the delimiter and tab separated value files (TSV) use tabs as the delimiter.

You can use the read.csv() function to read CSV files into R. This kernel is connected to the Titanic disaster data set, so we will use read.csv() to read it into our R session.

```
# Load the titanic data
titanic <- read.csv("../input/titanic/train.csv")
head(titanic)
A data.frame: 6 × 12
```

Experiment No.4

Different types of visualization and plotting:-

Experiment No.5

Covariance Syntax in R-

```
# Data vectors
x <- c(1, 3, 5, 10)

y <- c(2, 4, 6, 20)

# Print covariance using different methods
print(cov(x, y))
print(cov(x, y, method = "pearson"))
print(cov(x, y, method = "kendall"))
print(cov(x, y, method = "spearman"))
```

OUTPUT-

```
[1] 30.66667

[1] 30.66667

[1] 12

[1] 1.666667
```

Correlation in R-

```
  # Data vectors
  x <- c(1, 3, 5, 10)

  y <- c(2, 4, 6, 20)

  # Print correlation using different methods
  print(cor(x, y))

  print(cor(x, y, method = "pearson"))
  print(cor(x, y, method = "kendall"))
  print(cor(x, y, method = "spearman"))
```

Output:

```
[1] 0.9724702

[1] 0.9724702

[1] 1

[1] 1
```

Experiment No.6

Then use logistic regression to determine the relationship between variables that influence a student's admission to an institute based on his or her GRE score, GPA, and rank. REGRESSION MODEL Import data from web storage. Name the dataset, and rank.

Reading Data into R

I have already downloaded the dataset into my laptop.
You can access the dataset here https://stats.idre.ucla.edu/wp-content/uploads/2016/02/binary.sas7bdat

```
library(rio) #for data reading

data <- import("binary.sas7bdat")
```

Data Cleaning

Looking at the structure of the dataset

```
str(data)

## 'data.frame':    400 obs. of  4 variables:

##  $ ADMIT: num  0 1 1 1 0 1 1 0 1 0 ...

##  $ GRE  : num  380 660 800 640 520 760 560 400 540 700 ...

##  $ GPA  : num  3.61 3.67 4 3.19 2.93 ...

##  $ RANK : num  3 3 1 4 4 2 1 2 3 2 ...

##  - attr(*, "label")= chr "LOGIT"
```

Variables **ADMIT** and **RANK** are of type numeric but they should be factor variables since were are not going to perform any mathematical operations on them.

```
data$ADMIT <- as.factor(data$ADMIT)

data$RANK <- as.factor(data$RANK)

str(data)

## 'data.frame':    400 obs. of  4 variables:

##  $ ADMIT: Factor w/ 2 levels "0","1": 1 2 2 2 1 2 2 1 2 1 ...
## $ GRE  : num  380 660 800 640 520 760 560 400 540 700 ...

## $ GPA  : num  3.61 3.67 4 3.19 2.93 ...

## $ RANK : Factor w/ 4 levels "1","2","3","4": 3 3 1 4 4 2 1 2 3 2 ...

## - attr(*, "label")= chr "LOGIT"
```

Looking at the summary of the dataset

```
summary(data)
```

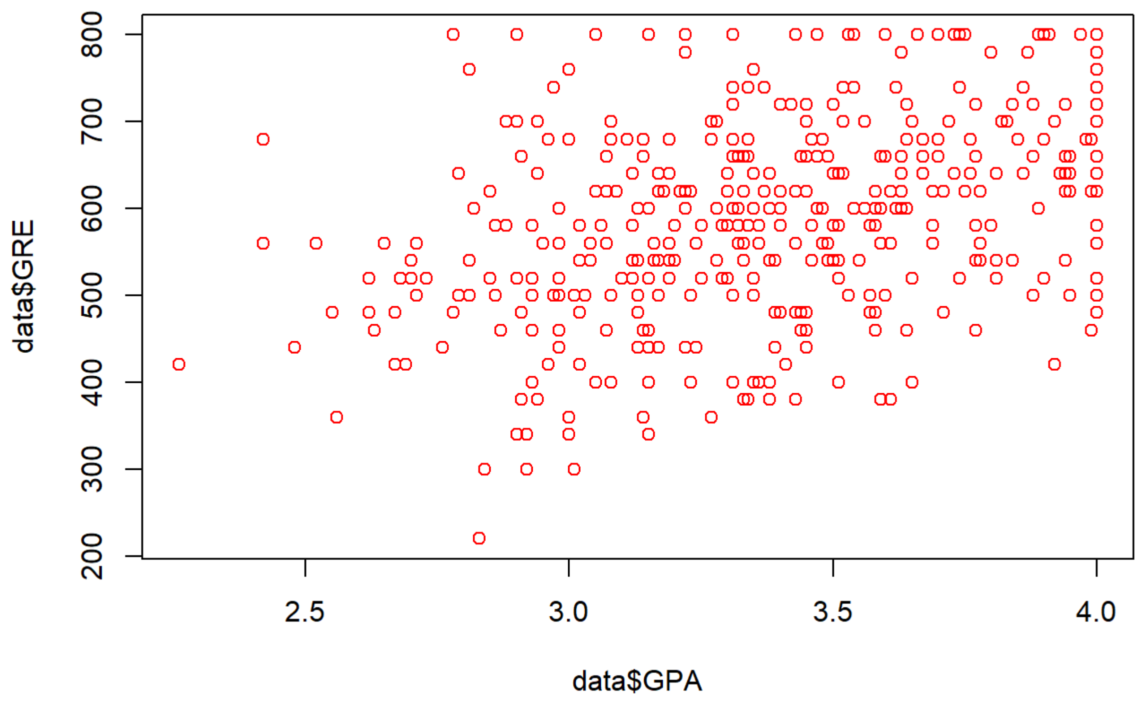
```
## ADMIT      GRE            GPA        RANK
## 0:273   Min.   :220.0   Min.   :2.260   1: 61
## 1:127   1st Qu.:520.0   1st Qu.:3.130   2:151
##         Median :580.0   Median :3.395   3:121
##         Mean   :587.7   Mean   :3.390   4: 67
##         3rd Qu.:660.0   3rd Qu.:3.670
##         Max.   :800.0   Max.   :4.000
```

From the summary statistics we observe

- Most of students did not get admitted
- There are no missing data values(NAs).

Checking for multicollineality

```
plot(data$GPA,data$GRE,col="red")
```



```
cor(data$GRE,data$GPA)
```

```
## [1] 0.3842659
```

From the plot we can infer that the two numeric variables are not correlated which is confirmed by low correlation value of **0.3842659**.

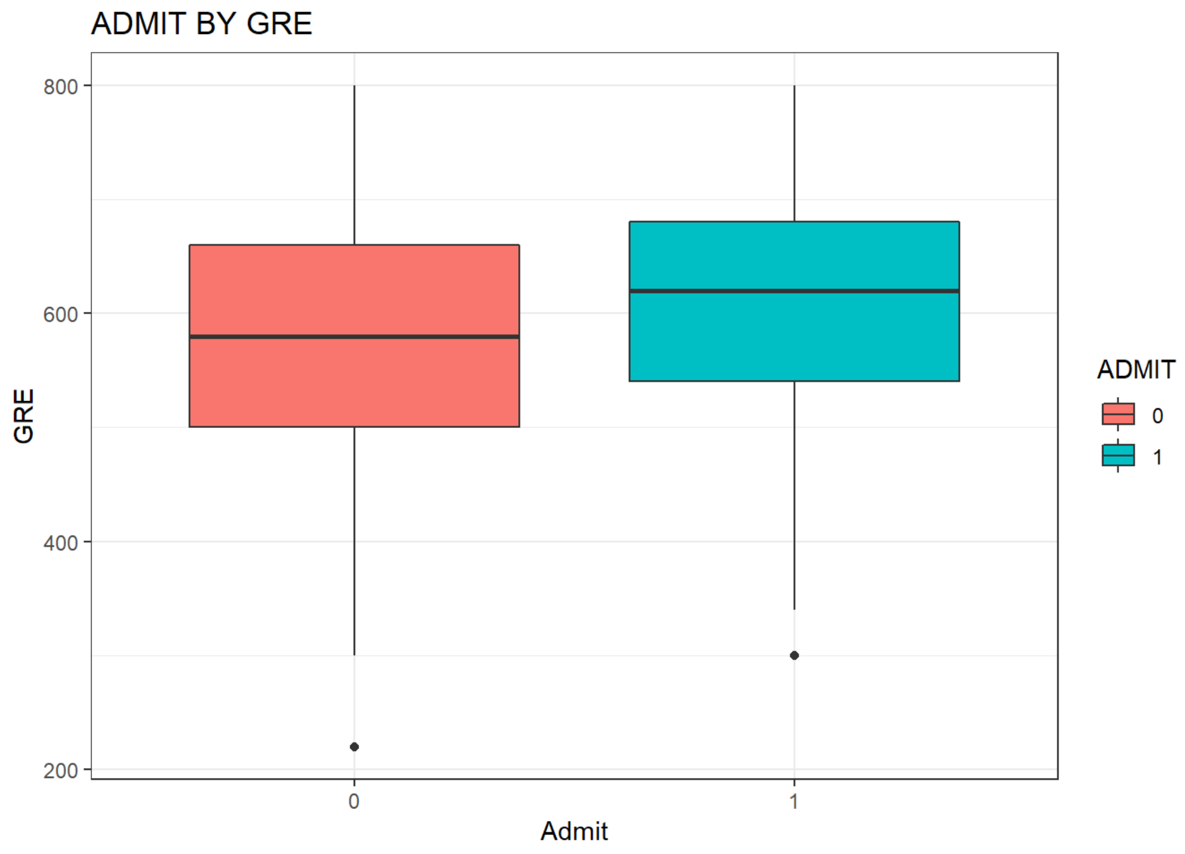
Exploratoty Data Analysis.

We will explore the relationship between dependent and independent variables by way of visualization.

GRE

Since GRE is numeric variable and dependent variable is factor variable, we plot a box plot.

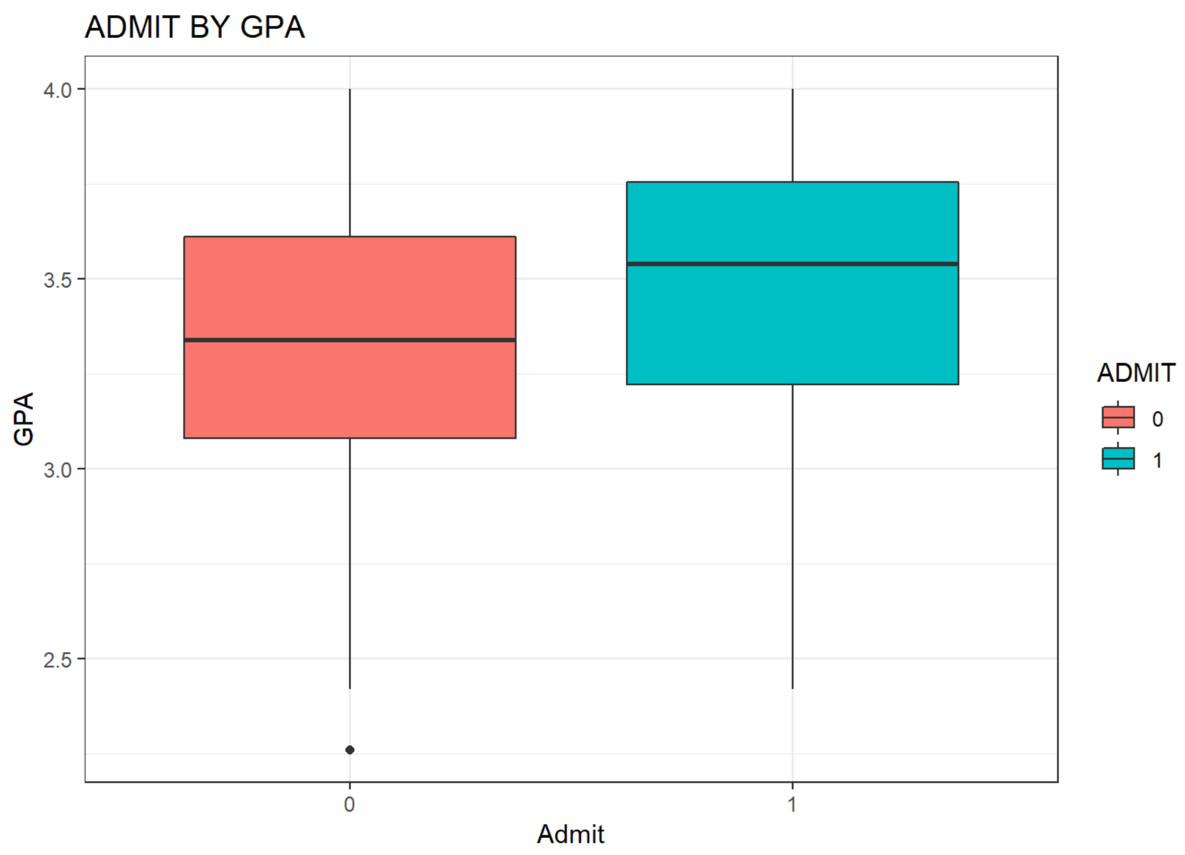
```
library(ggplot2) #For plotting
ggplot(data,aes(ADMIT,GRE,fill=ADMIT))+
  geom_boxplot()+
  theme_bw()+
  xlab("Admit")+
  ylab("GRE")+
 ggtitle("ADMIT BY GRE")
```



The two box plots are differents in terms of displacement, and hence *GRE* is significant variable.

GPA

```
ggplot(data,aes(ADMIT,GPA,fill=ADMIT))+
 geom_boxplot()+
 theme_bw()+
 xlab("Admit")+
 ylab("GPA")+
 ggtitle("ADMIT BY GPA")
```

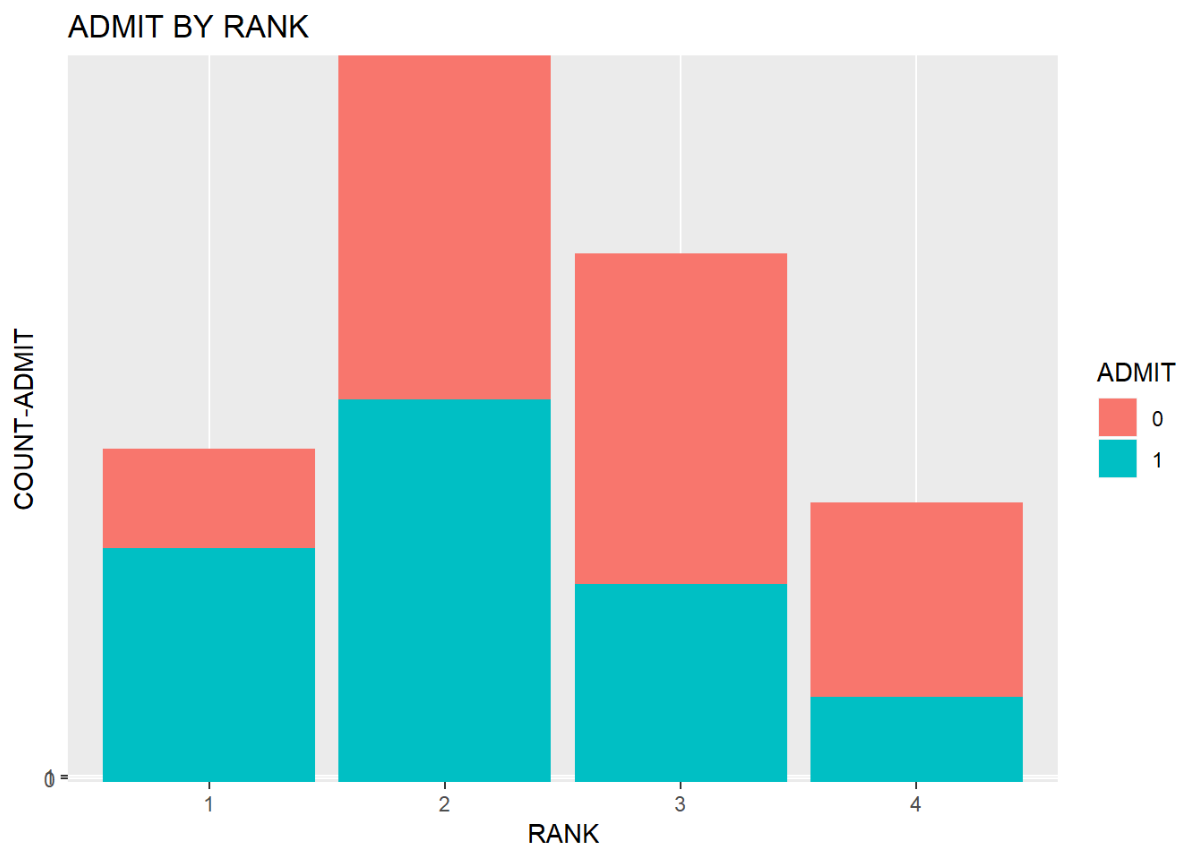


There is clear difference in displacement between the two box plots, hence *GPA* is an important predictor.

RANK

RANK is a factor variable and since the dependent variable is a factor variable we plot a bar plot.

```
ggplot(data,aes(RANK,ADMIT,fill=ADMIT))+
  geom_col()+
  xlab("RANK")+
  ylab("COUNT-ADMIT")+
  ggtitle("ADMIT BY RANK")
```



There is a clear pattern; as *RANK* increases the possibility of a student being admitted decreases.

Experiment No.7

Aim- MULTIPLE REGRESSION MODEL (Apply on IRIS Dataset)

Multivariate Linear Regression Model

In [10]:

```
mapping = {
    'Iris-setosa' : 1,
    'Iris-versicolor' : 2,
    'Iris-virginica' : 3
}

X = data.drop(['Id', 'Species'], axis=1).values # Input Feature Values
y = data.Species.replace(mapping).values.reshape(rows,1) # Output values

X = np.hstack(((np.ones((rows,1))), X))# Adding one more column for bias
```

In [11]:

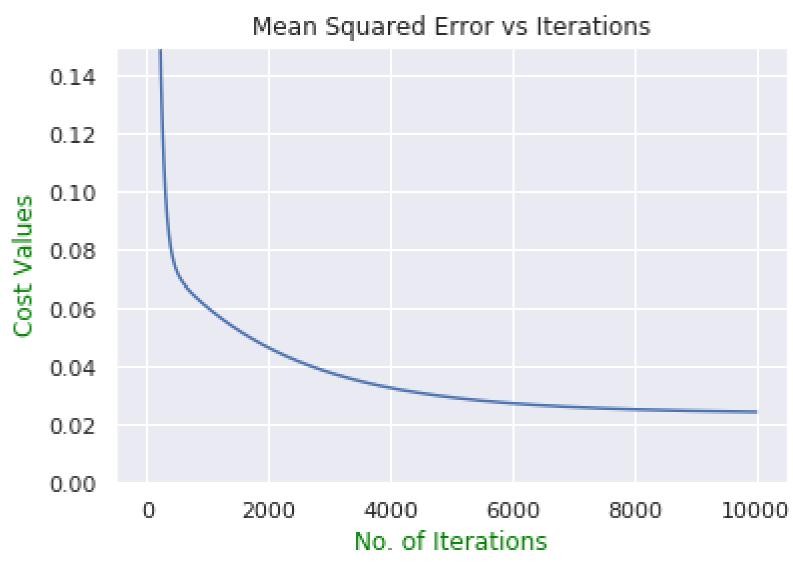
```
np.random.seed(0) # Let's set the zero for time being
theta = np.random.randn(1,5) # Setting values of theta randomly

print("Theta : %s" % (theta))
Theta : [[1.76405235 0.40015721 0.97873798 2.2408932  1.86755799]]
```

In [12]:

```
iteration = 10000
learning_rate = 0.003 # If you are going by formula, this is actually alpha.
J = np.zeros(iteration) # 1 x 10000 maxtix
```

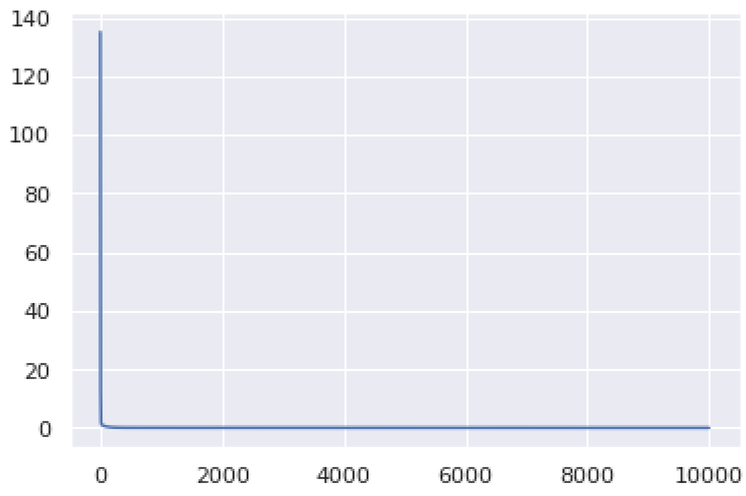
In [13]:

```
# Let's train our model to compute values of theta
for i in range(iteration):
    J[i] = (1/(2 * rows) * np.sum((np.dot(X, theta.T) - y) ** 2 ))
    theta -= ((learning_rate/rows) * np.dot((np.dot(X, theta.T) - y).reshape(1,rows), X))

prediction = np.round(np.dot(X, theta.T))

ax = plt.subplot(111)
ax.plot(np.arange(iteration), J)
ax.set_ylim([0,0.15])
plt.ylabel("Cost Values", color="Green")
plt.xlabel("No. of Iterations", color="Green")
plt.title("Mean Squared Error vs Iterations")
plt.show()
```



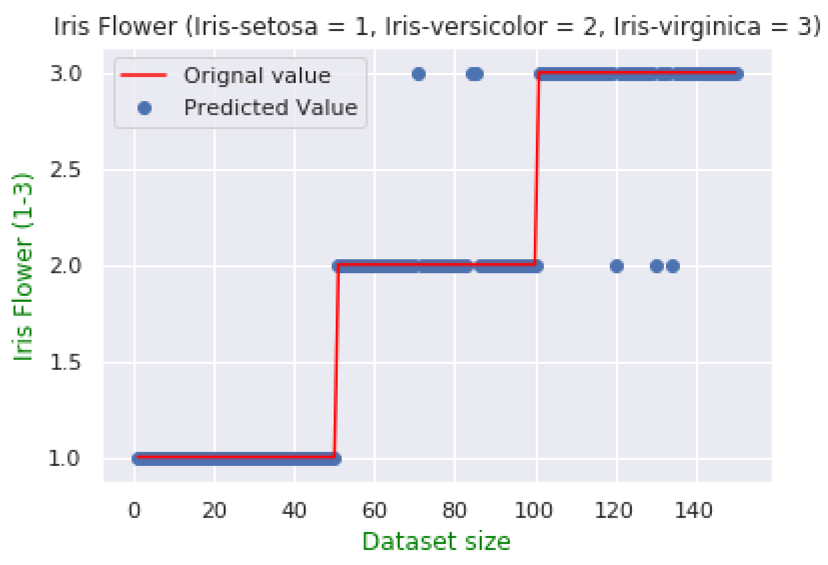
In [14]:

```
ax = sns.lineplot(x=np.arange(iteration), y=J)
plt.show()
```



In [15]:

```
ax = plt.subplot(111)

ax.plot(np.arange(1, 151, 1), y, label='Orignal value', color='red')
ax.scatter(np.arange(1, 151, 1), prediction, label='Predicted Value')

plt.xlabel("Dataset size", color="Green")
plt.ylabel("Iris Flower (1-3)", color="Green")
plt.title("Iris Flower (Iris-setosa = 1, Iris-versicolor = 2, Iris-virginica = 3)")

ax.legend()
plt.show()
```



In [16]:

```
accuracy = (sum(prediction == y)/float(len(y)) * 100)[0]
print("The model predicted values of Iris dataset with an overall accuracy of %s" % (accuracy))
The model predicted values of Iris dataset with an overall accuracy of 96.0
```

Experiment No.8

Aim- Choose a classifier for the classification problems and evaluate the classifier's performance:-

We will use Iris-Flower Classification Dataset which is perfect for beginners as I want this blog to be beginner friendly.

Our goal here is to correctly classify the flower type, based on its properties the flower can be of type setosa, versicolor, virginica i.e ***Species*** column in our dataset, also we will identify the algorithm which is best suited for this problem.

```
#Import Libraries and Read the dataimport pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm, tree
import xgboost
from sklearn.model_selection import train_test_splitdata  =  pd.read_csv("D://Blogs//Iris
Multiple Algo//Iris.csv")#Create Dependent and Independent Datasets based on our Dependent
#and Independent featuresX  = data[['SepalLengthCm','SepalWidthCm','PetalLengthCm']]
y= data['Species']#Split the Data into Training and Testing sets with test size as #30%X_train,
X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, shuffle=True)
```

Now, we will create an array of Classifiers and append different classification models to our array.

```
model1 = xgboost.XGBClassifier()
classifiers.append(model1)model2 = svm.SVC()
classifiers.append(model2)model3 = tree.DecisionTreeClassifier()
classifiers.append(model3)model4 = RandomForestClassifier()
classifiers.append(model4)
```

We will fit our algorithms in our classifiers array on Train dataset and check the accuracy and confusion matrix for our test dataset prediction given by different algorithms

```
for clf in classifiers:
    clf.fit(X_train, y_train)
    y_pred= clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("Accuracy of %s is %s"%(clf, acc))
    cm = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix of %s is %s"%(clf, cm)
```

OUTPUT-

```
Accuracy of XGBClassifier is 95.55
Confusion Matrix of XGBClassifier is [[ 9  0  0]
                                      [ 0 15  1]
                                      [ 0  1 19]]Accuracy of SVC is 95.55
Confusion Matrix of SVC is [[ 9  0  0]
                            [ 0 16  0]
                            [ 0  2 18]]Accuracy of DecisionTreeClassifier is  88.88
Confusion Matrix of DecisionTreeClassifier is [[ 9  0  0]
                                               [ 0 15  1]
                                               [ 0  4 16]]
Accuracy of RandomForestClassifier is  84.44
Confusion Matrix of RandomForestClassifier is [[ 9  0  0]
                                               [ 0 15  1]
                                               [ 0  6 14]]
```

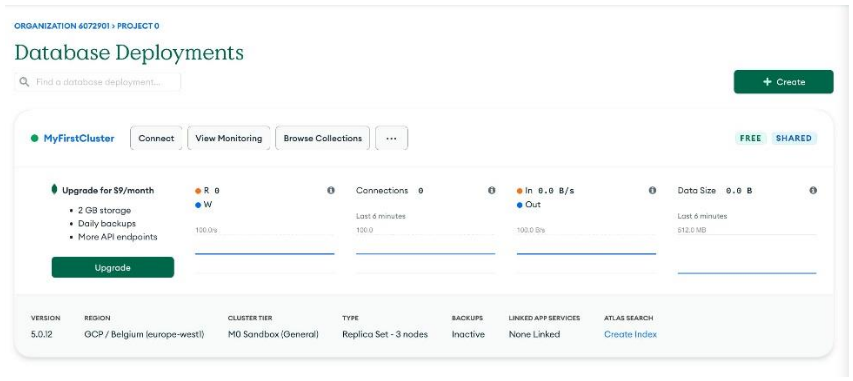
Experiment No.9

Aim- Create a Database on Mongodb Cloud: -

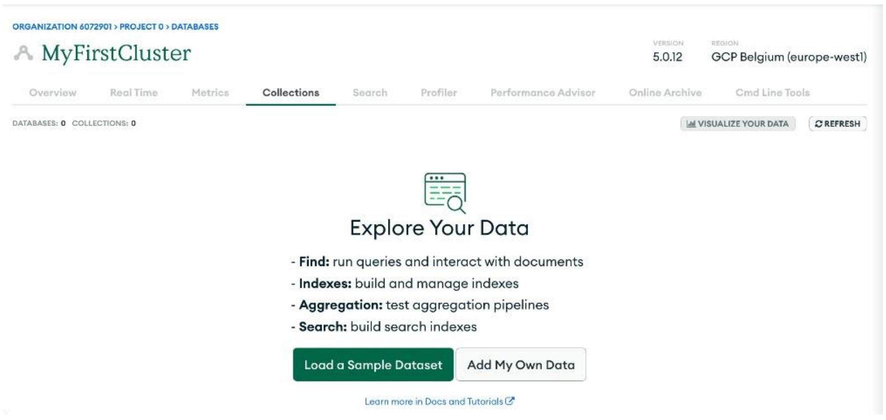
Register a Atlas account with your email address

Deploy your first cluster

From your cluster page, click on "Browse Collections."



If there are no databases in this cluster, you will be presented with the option to create your first database by clicking on the "Add My Own Data" button.


ORGANIZATION 6072901 > PROJECT 0 > DATABASES
MyFirstCluster
VERSION
5.0.12
REGION
GCP Belgium (europe-west1)
Overview
Real Time
Metrics
Collections
Search
Profiler
Performance Advisor
Online Archive
Cmd Line Tools
DATABASES: 0 COLLECTIONS: 0
VISUALIZE YOUR DATA
REFRESH
Explore Your Data
- Find: run queries and interact with documents
- Indexes: build and manage indexes
- Aggregation: test aggregation pipelines
- Search: build search indexes
Load a Sample Dataset
Add My Own Data
Learn more in Docs and Tutorials

This will open up a modal, asking you for a database name and collection name. Once these fields are filled, click on "Create" and your database will be created for you.

The database is now available to you. You can manually enter new documents, or connect to the database using any of the MongoDB drivers.

Experiment No.10

MongoDB Compass

MongoDB Compass is a GUI for MongoDB. It is also known as MongoDB GUI. MongoDB allows users to analyze the content of their stored data without any prior knowledge of MongoDB query syntax. When we explore exploring our data in the visual environment, we can use Compass GUI to optimize performance, manage indexes, and implement document-validation.

All the versions of MongoDB Compass are opensource (i.e., we can freely deploy and view the repositories of all MongoDB GUI versions). The source repositories of MongoDB compass can be found on the following link of GitHub

https://github.com/mongodb-js/compass/

Available Compass Edition

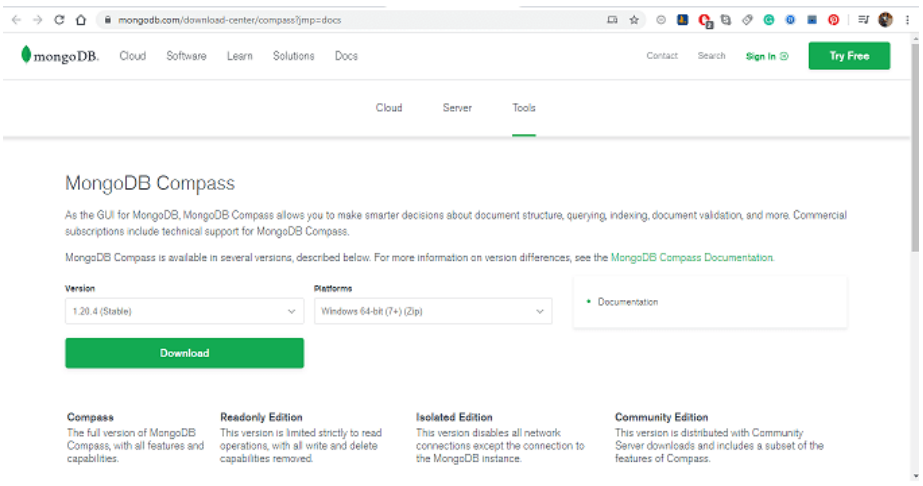
- Compass Community: This edition is designed for developing with MongoDB and includes a subset of the features of Compass.
- Compass: It is released as the full version of MongoDB Compass. It includes all the features and capabilities that MongoDB provides.
- Compass Randomly: It is limited to read operation only with all update and delete capabilities removed.
- Compass Isolated: The Isolated edition of MongoDB compass doesn't start any network requests except to the MongoDB server to which MongoDB GUI connects. It is designed to use in highly secure environments.
-

How to Download and Install MongoDB Compass

Step 1: To download MongoDB Compass, you can use your preferred web browser, and Open the https://www.mongodb.com/download-center/compass?jmp=docs page.

Step 2: You need to select the installer and version you prefer. GUI installer are available as a .exe or .msi package or a .zip archive.

Step 3: Finally, Click on the download button.



MongoDB Compass

As the GUI for MongoDB, MongoDB Compass allows you to make smarter decisions about document structure, querying, indexing, document validation, and more. Commercial subscriptions include technical support for MongoDB Compass.

MongoDB Compass is available in several versions, described below. For more information on version differences, see the MongoDB Compass Documentation.

Version: 1.20.4 (Stable)

Platforms: Windows 64-bit (7+) (Zip)

- Documentation

Compass
The full version of MongoDB Compass, with all features and capabilities.

Readonly Edition
This version is limited strictly to read operations, with all write and delete capabilities removed.

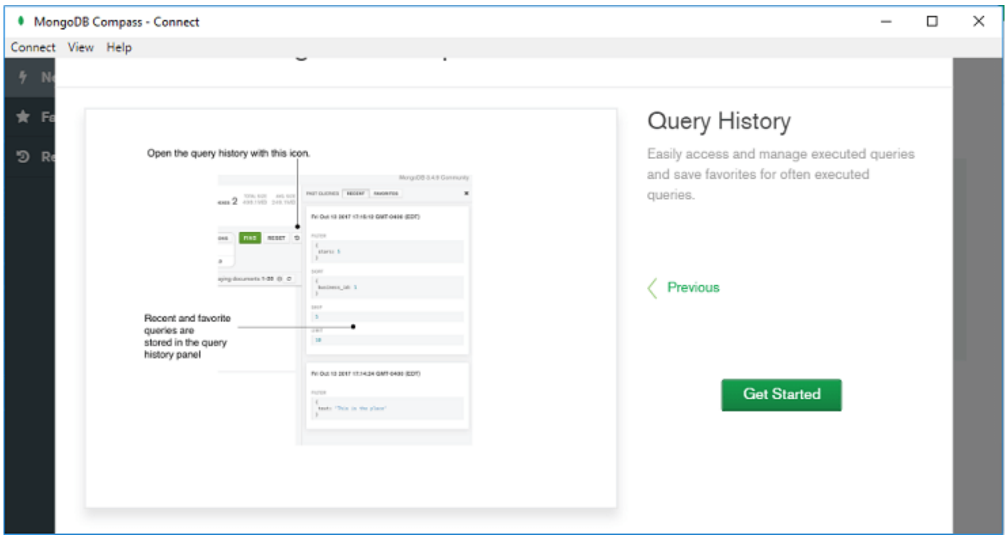
Isolated Edition
This version disables all network connections except the connection to the MongoDB instance.

Community Edition
This version is distributed with Community Server downloads and includes a subset of the features of Compass.

Step 4: Click on the installer file after the download is complete.

Step 5: Follow the pop-ups to install MongoDB Compass GUI.

Step 6: Once it is installed, it launches and ask you to configure privacy settings and specify update preference.


MongoDB Compass - Connect
Connect
View
Help
Open the query history with this icon.
Recent and favorite queries are stored in the query history panel
Previous
Get Started

Query History

Easily access and manage executed queries and save favorites for often executed queries.