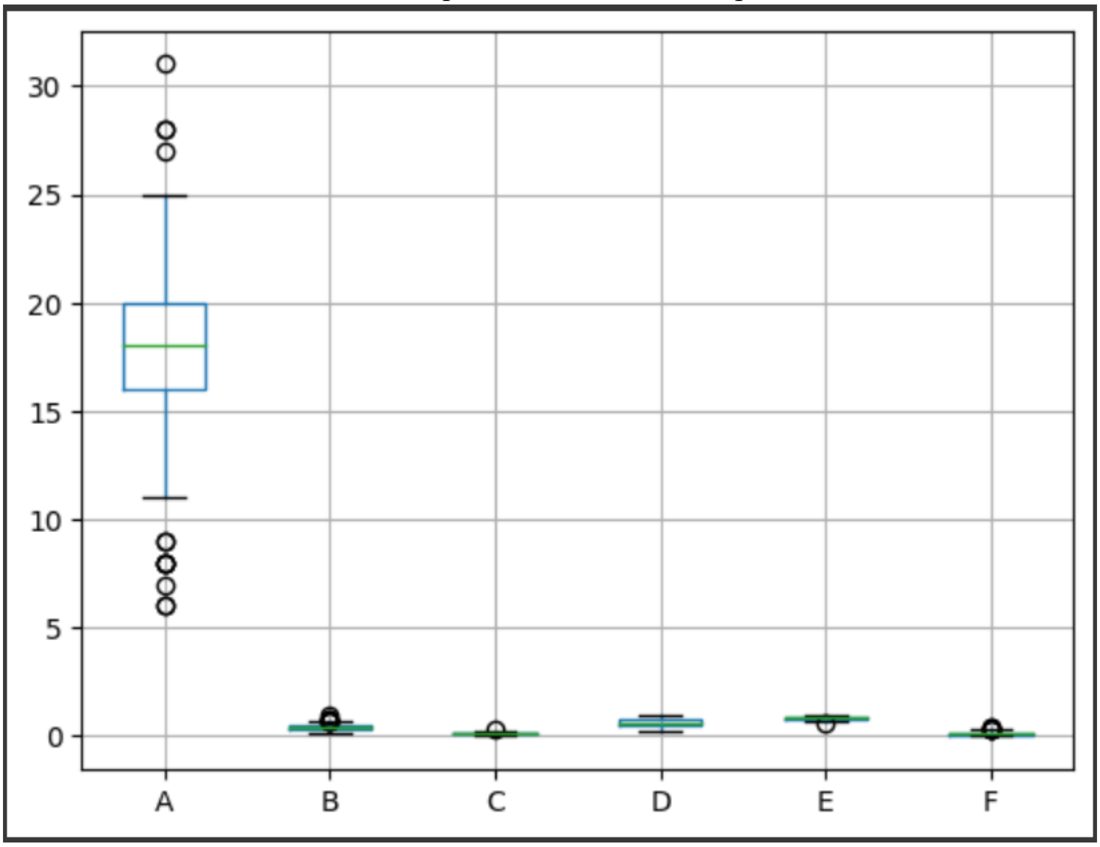
In-Lab

Task 1

Writing Code to visualize data using box plot:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn import linear_model
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
import pandas as pd
import csv

import matplotlib.pyplot as plt

from google.colab import files
u = files.upload()

np.random.seed(7)
df = pd.read_csv("Alumni Giving Regression
(Edited).csv",delimiter=",")
dd_df_1 = df.head()

import seaborn as sns
boxplot = pd.DataFrame(df).boxplot()
```

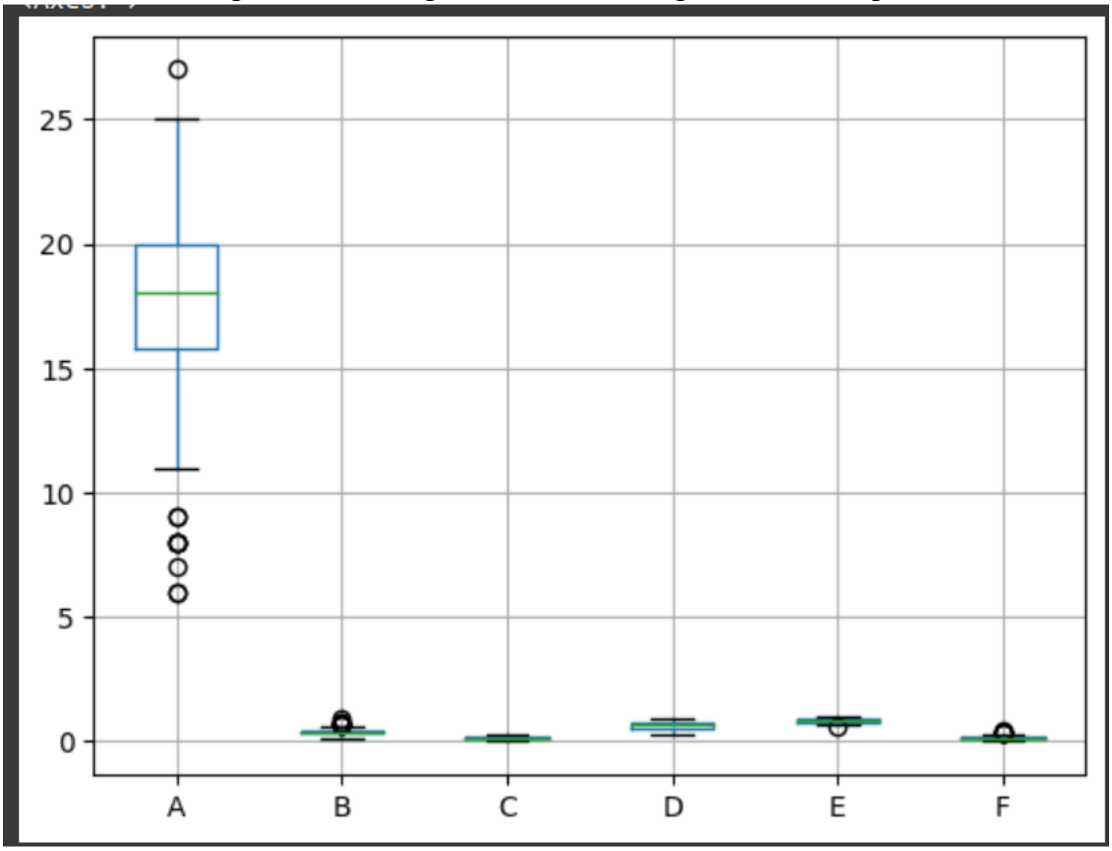
It print the boxplot of dataset which is basically distribution of the data based on five number summaries. Minimum, first quartile, median, third quartile and maximum.



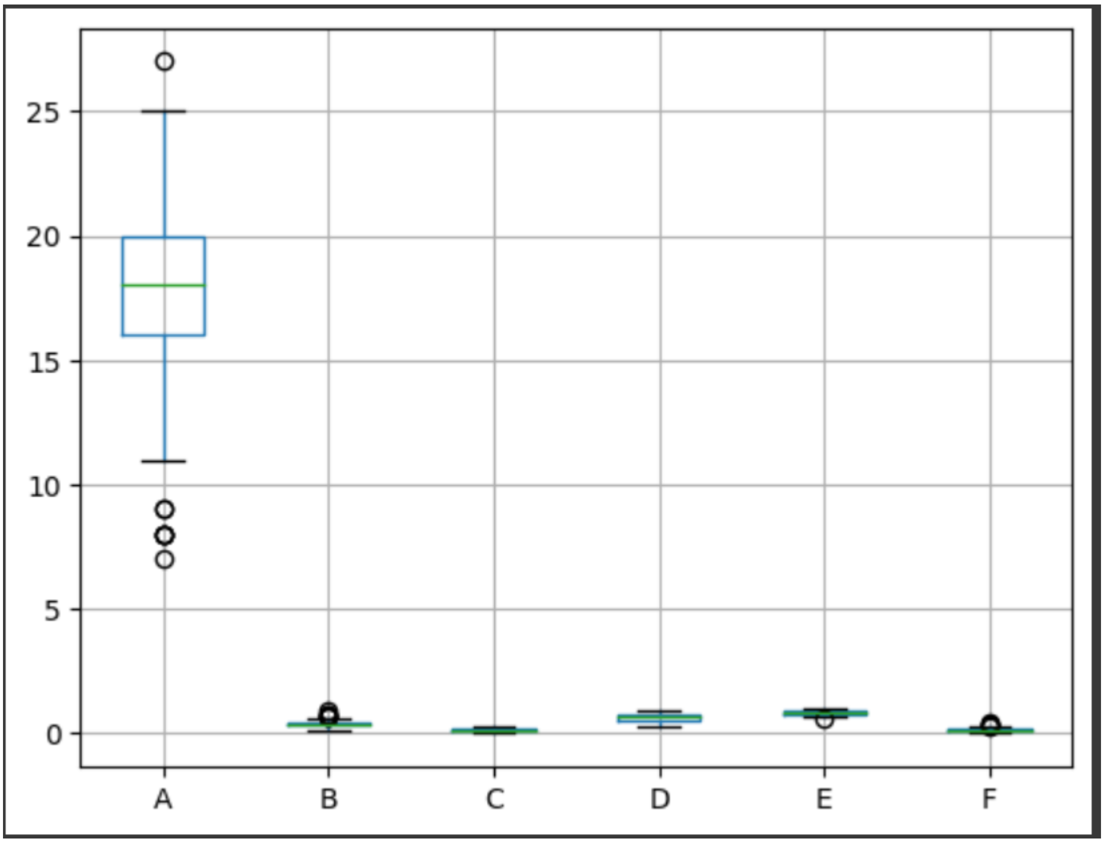
Task 2

```
quantile99 = df.iloc[:,0].quantile(0.99)
df1 = df[df.iloc[:,0]< quantile99]
df1.boxplot()

quantile1 = df.iloc[:,0].quantile(0.01)
quantile99 = df.iloc[:,0].quantile(0.99)
df2  = df[(df.iloc[:,0] > quantile1) & (df.iloc[:,0] < quantile99)]
df2.boxplot()
```

Code 1 finds the 99th percentile of the first column, creates a subset df1 and store the values containing less than this percentile in it and generates a boxplot for it.



Code 2 finds the 1st and 99th percentile of the first column and select the values which are greater than the 1st percentile and less than the 99th percentile and store them in df2 subset and generates a boxplot for it.



Task 3

```
df.dropna()
Y_POSITION = 5
model_2_features = [i for i in range(0,Y_POSITION)]
X = df.iloc[:,model_2_features]
Y = df.iloc[:,Y_POSITION]
X_train, X_test, y_train, y_test =
train_test_split(X,Y,test_size=0.20,random_state=2020)
model3 = RandomForestRegressor()
model3.fit(X_train,y_train)
RF = model3
importances = RF.feature_importances_
std = np.std([tree.feature_importances_ for tree in RF.estimators_],
axis=0)
indices = np.argsort(importances)[::-1]
print("Feature ranking: ")
for f in range(X.shape[1]):
  print("%d. feature (Column index) %s (%f)" % (f + 1, indices[f],
importances[indices[f]]))
```

```
Feature ranking:
1. feature (Column index) 3 (0.381108)
2. feature (Column index) 0 (0.212209)
3. feature (Column index) 1 (0.176601)
4. feature (Column index) 4 (0.137474)
5. feature (Column index) 2 (0.092607)
```

Figure 1

Task 4

```
indices_top3 = indices[:3]
print(indices_top3)
dataset = df
df = pd.DataFrame(df)
Y_POSITION = 5
TOP_N_FEATURE = 3
X = df.iloc[:,indices_top3]
Y = df.iloc[:,Y_POSITION]
X_train, X_test, y_train, y_test =
train_test_split(X,Y,test_size=0.20,random_state=2020)

model1 = linear_model.LinearRegression()
model1.fit(X_train,y_train)
y_pred_train1 = model1.predict(X_train)
print("Regression")
print("==============================")
RMSE_train1 = mean_squared_error(y_train,y_pred_train1)
print("Regression Train set: RMSE {}".format(RMSE_train1))
print("===========================")
y_pred1 = model1.predict(X_test)
RMSE_test1 = mean_squared_error(y_test,y_pred1)
print("Regression Test set: RMSE {}".format(RMSE_test1))
print("==================")
```

```
[3 0 1]
Regression
==============================
Regression Train set: RMSE 0.0027952079052752685
===========================
Regression Test set: RMSE 0.004341758028139646
==================
```

Figure 2

There is no major difference between test scores of both previous lab without selecting top 3 features and this lab by selecting top 3 features.