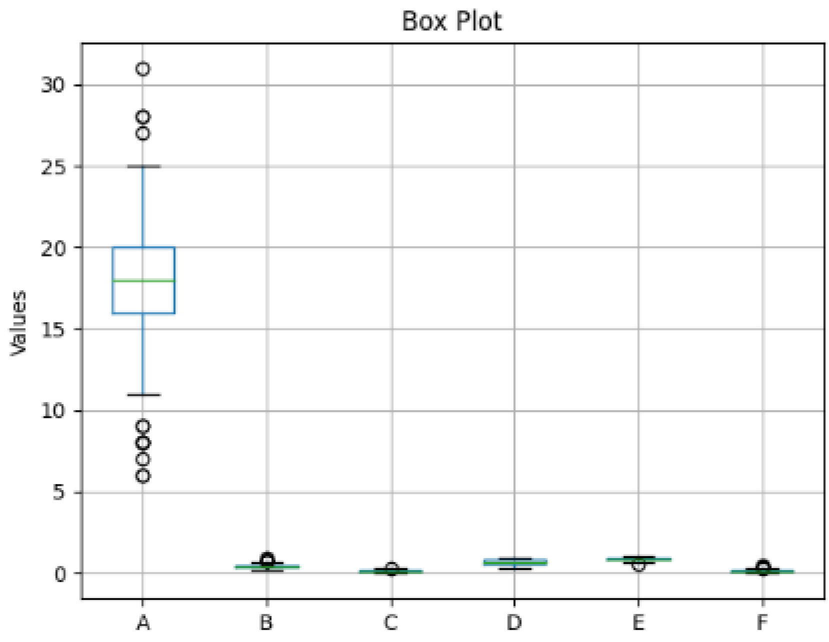
In-Lab

Task 1:

```
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

import seaborn as sns
import pandas as pd
boxplot = pd.DataFrame(df).boxplot()
plt.title("Box Plot")
plt.ylabel('Values')
plt.xlabel('Columns')
```

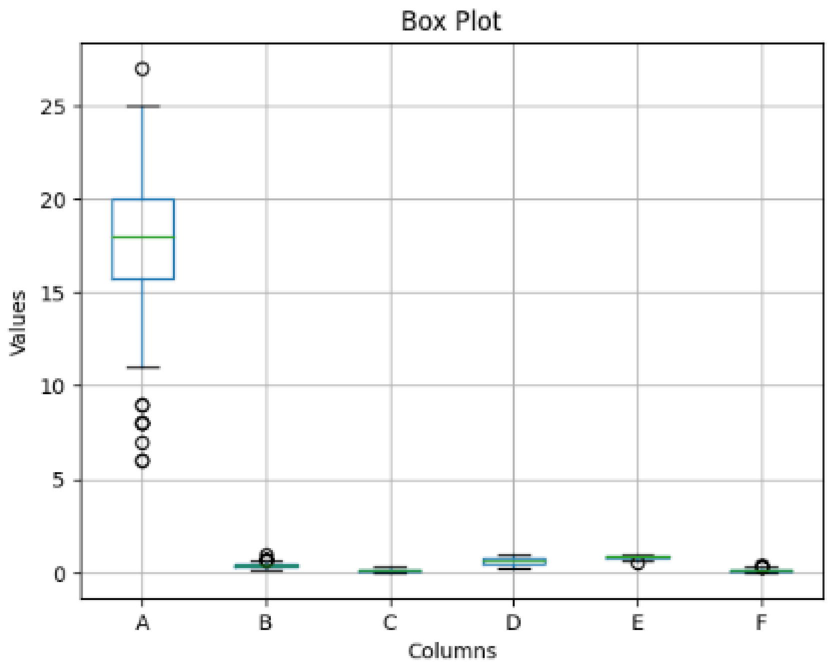
Output:



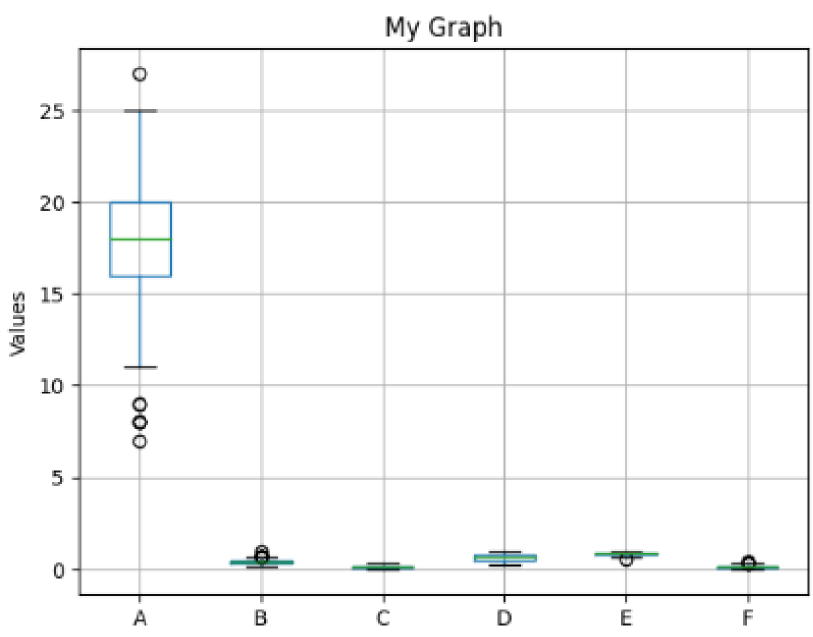
Task 2:

Load and Preprocess Data:

```
# %%
quantile99 = df.iloc[:,0].quantile(0.99)
df1 = df[df.iloc[:,0] < quantile99]
df1.boxplot()
plt.title("Box Plot")
plt.xlabel('Columns')
plt.ylabel('Values')

# %%%
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()
plt.title("Box Plot")
plt.ylabel('Values')
plt.xlabel('Columns')
```



In the first section (df1), we are removing values that are greater than the 99th percentile, effectively trimming the upper tail of the distribution.



In the second section (df2), you are removing values outside the range defined by the 1st and 99th percentiles. This involves trimming both the lower and upper tails of the distribution.

```
# Feature ranking
model3 = RandomForestRegressor()
X = df[['A', 'B', 'C', 'D', 'F']]
y = df['E']

# Train the model on the data
model3.fit(X, y)

# %%
df.dropna()
# %%
#
#Feature Ranking
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 the feature ranking
print("Feature ranking:")
for f in range(X.shape[1]):
  print("%d. feature (Column index) %s (%f)" % (f + 1, indices[f],
importances[indices[f]]))
```

Output:

```
Feature ranking:
1. feature (Column index) 3 (0.892467)
2. feature (Column index) 2 (0.037888)
3. feature (Column index) 1 (0.030620)
4. feature (Column index) 4 (0.021524)
5. feature (Column index) 0 (0.017500)
```

Task 3:

```
# %%
indices_top3= indices[:3]
print(indices_top3)
dataset=df
df = pd.DataFrame(df)
Y_position= 5
TOP_N_FEATURE = 3
X = dataset.iloc[:, indices_top3]
Y = dataset.iloc[:,Y_position]
# create model
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.20,random_state=2020)
#Model 1: linear regression
model1=linear_model.LinearRegression()
model1.fit(X_train, y_train)
y_pred_train1 = model1.predict(X_train)
print("Regression")

RMSE_train1 = mean_squared_error(y_train, y_pred_train1)
print("Regression TrainSet: RMSE {}".format(RMSE_train1))
y_pred1= model1.predict(X_test)
print("======")
RMSE_test1 = mean_squared_error(y_test,y_pred1)
print("Regression Testset: RMSE {}".format(RMSE_test1))
print("======")
```

Output:

```
[3 2 1]
Regression
Regression TrainSet: RMSE 0.002796386754276771
======
Regression Testset: RMSE 0.004386394878107668
======
```

Result :

As we can see the RMSE values of Train set are approximately the same while the values of test set this time is slightly worse than last time.