Table of Contents

Chapter 1

1.1 Introduction

Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. Scheduling is one of important issues for improving the efficiency of all cloud-based services. In cloud computing Task scheduling is used to allocate the task to best suitable resource for execution. There are different types of task scheduling algorithms. Some issues like input delay, output delay, processing delay etc. in task scheduling have been considered in our project. Here we use different task scheduling algorithms to see how the performance of cloud computing changes.

1.2 Objective

There are some clear objectives for the project. To help you understand the project better, the following key goals are listed:

- Get a general idea of cloud computing operation
- Apply different scheduling algorithm for cloud computing
- Make the dataset for our project
- Plot the graph for understand better result
- Able to use different python libraries

Chapter 2

2.1 Description

In this project we use Frist Come Frist Serve (FCFS), Shortest Job Scheduling (SJF) and Priority Scheduling algorithms to could computer to see which performs better. We use process delay, input delay and output delay to calculate which performs better. Process delay is calculated by task size and size of the packet. When data is sent to the server via internet how long it takes the data to store in the cloud. Input delay and output delay is calculated by size of task and speed of the internet. Input delay indicates how to it takes to reach the server and output delay indicates how long it takes to come back to the local device.

2.2 Component Description

- **Jupyter Notebook:** Jupyter is an open-source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Jupyter integrates with a number of prominent packages in the scientific Python stack, including NumPy, Scikit Learn, Matplotlib, pandas as well as other open-source software. It is released under the MIT license.
- **NumPy**: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Matplotlib**: Matplotlib is a plotting library for python programming language which is an extension of numpy of numerical mathematics. It is an object-oriented API for embedding plots into applications using general purpose GUI. There is also a procedural “pylab” interface designed closely to reassemble that of MATLAB.
- **Pandas**: Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself.

Chapter 3

3.1 Methodology

- First, we create a dataset which contains input data size, output data size and priority value of the task. In this dataset we use one as highest priority and five as the lowest priority.
- Then we calculate the value of process delay, input delay and output delay.
- Generate final delay using, final delay = process delay+ input delay + output delay.
- Apply FCFS, SJF and Priority Scheduling algorithm to this final delay.
- Plot the graph after applied the algorithm.
- Compare the graph to understand which algorithm is better for the server to follow.

- Formula for calculating Process delay

$$ProccesDelay = \frac{SizeofInputTask * WorkLoad}{DutyCycle}$$

$$ProccesDelay = \frac{(SizeofInputTask * 10^3)cycles/bit}{32 * 10^6}$$

- Formula for calculating Input delay (For this project we set the net speed to 75 Mbps)

$$InputDelay = \frac{SizeofTask}{SpeedofInternet}$$

$$InputDelay = \frac{SizeofTask}{75 * 10^6}$$

- Formula for calculating Output delay (For this project we set the net speed to 75 Mbps)

$$OutputDelay = \frac{SizeofTask}{SpeedofInternet}$$

$$OutputDelay = \frac{SizeofTask}{75 * 10^6}$$

Chapter 4

4.1 Code Explanation

- Importing necessary libraries and upload dataset file. Here we use Pandas read.csv to upload the dataset file.

```
In [175]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt

In [176]: df = pd.read_csv(r'C:\Users\myaha\OneDrive\Desktop\Jupyter Notebook files\CN Project\task.csv')

In [177]: df.head()
```

Out[177]:

	Input Data	Output Data	Priority Number
0	60	60	4
1	38	38	2
2	28	28	3
3	44	44	1
4	75	75	3

- Calculating process delay, input delay and output delay. Then by adding all of them we calculate the final delay.

```
In [178]: process_delay = (indata*10**6)/(32*10**6)
          process_delay.head()

Out[178]: 0    1.87500
          1    1.18750
          2    0.87500
          3    1.37500
          4    2.34375
          Name: Input Data, dtype: float64

In [179]: input_delay = (indata*10**3)/(75*10**6)
          output_delay = (outdata*10**3)/(75*10**6)

In [182]: final_delay = input_delay + output_delay + process_delay
          final_delay

Out[182]: 0     1.876600
          1     1.188513
          2     0.875747
          3     1.376173
          4     2.345750
```

- Then we apply FCFS algorithm by using the final delay. Here we use Numpy array to calculate the value and store the value for further operation.

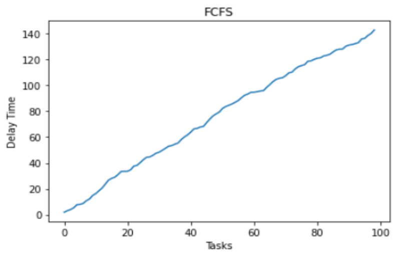
```
In [183]: def FCFS(x):
              sum=0
              fcfs = np.empty((0))
              for i in range(len(x)):
                  sum=sum + final_delay[i]
                  fcfs = np.append(fcfs,sum)
              return fcfs

In [184]: fcfs = FCFS(final_delay)
          fcfs

Out[184]: array([ 1.8766    ,  3.06511333,  3.94086   ,  5.31703333,
                  7.66278333,  7.91299667,  8.72619   , 10.79045   ,
                 12.16662333, 14.88769333, 16.35769667, 18.54706333,
                 20.67387667, 23.73899   , 26.74155   , 28.11772333,
                 29.02474667, 31.05773   , 33.46603333, 33.52858667,
                 33.55986333, 34.77965333, 37.56327667, 38.15753333,
```

- Plot the FCFS using matplotlib library. Here we use plt.plot() for plotting and xlabel, ylabel and plt.title() is for labeling and give title to the graph.

```
In [186]: # FCFS Display
plt.plot(task_number,fcfs)
plt.xlabel("Tasks")
plt.ylabel("Delay Time")
plt.title("FCFS")
plt.show()
```



- Final Delay is transferred to Numpy array and sorted for SJF algorithm.

```
In [187]: # SJF Start
temp = final_delay.to_numpy()
temp = np.sort(temp)
temp
```

```
Out[187]: array([0.03127667, 0.06255333, 0.09383    , 0.12510667, 0.25021333,
           0.28149   , 0.31276667, 0.34404333, 0.34404333, 0.34404333,
           0.40659667, 0.40659667, 0.43787333, 0.43787333, 0.50042667,
           0.50042667, 0.53170333, 0.53170333, 0.56298   , 0.56298   ,
           0.59425667, 0.71936333, 0.71936333, 0.78191667, 0.81319333,
           0.81319333, 0.84447   , 0.84447   , 0.87574667, 0.87574667,
           0.90702333, 0.90702333, 0.90702333, 0.90702333, 0.96957667,
           0.96957667, 1.00085333, 1.09468333, 1.12596   , 1.12596   ,
```

- Apply SJF algorithm and calculate the execution value to sjf numpy array.

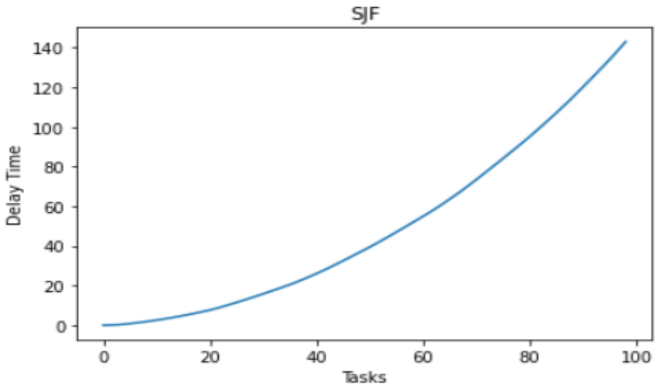
```
]: sum=0
sjf = np.empty((0))
for i in range(len(temp)):
    sum=sum + temp[i]
    sjf = np.append(sjf,sum)
sjf
```

```
]: array([3.12766667e-02, 9.38300000e-02, 1.87660000e-01, 3.12766667e-01,
          5.62980000e-01, 8.44470000e-01, 1.15723667e+00, 1.50128000e+00,
          1.84532333e+00, 2.18936667e+00, 2.59596333e+00, 3.00256000e+00,
          3.44043333e+00, 3.87830667e+00, 4.37873333e+00, 4.87916000e+00,
          5.41086333e+00, 5.94256667e+00, 6.50554667e+00, 7.06852667e+00,
          7.66278333e+00, 8.38214667e+00, 9.10151000e+00, 9.88342667e+00,
          1.06966200e+01, 1.15098133e+01, 1.23542833e+01, 1.31987533e+01,
          1.40745000e+01, 1.49502467e+01, 1.58572700e+01, 1.67642933e+01,
```

- Plot the SJF using matplotlib library. Here we use plt.plot() for plotting and xlabel, ylabel and plt.title() is for labeling and give title to the graph.

```
plt.plot(task_number,sjf,label = "SJF")
plt.xlabel("Tasks")
plt.ylabel("Delay Time")
plt.title("SJF")
```

```
Text(0.5, 1.0, 'SJF')
```



- For Priority scheduling algorithm we sort the dataset according to the priority number for the tasks.

```
90]: # Priority Scheduling
p = df
p = t.sort_values("Priority Number")
p.head()
```

90]:

	Input Data	Output Data	Priority Number
88	4	4	1
47	77	77	1
5	8	8	1
64	80	80	1
3	44	44	1

- Calculate input delay, output delay, process delay and final delay by adding them.

```
# input output and proccess delay calculation
p_indata = t["Input Data"]
p_outdata = t["Output Data"]
p_process_delay = (p_indata*10**6)/(32*10**6)
p_input_delay = (p_indata*10**3)/(75*10**6)
p_output_delay = (p_outdata*10**3)/(75*10**6)
p_final_delay = p_input_delay + p_output_delay + p_process_delay
p_final_delay
```

```
88    0.125107
78    0.344043
56    2.158090
66    2.345750
83    0.562980
```

- Apply priority scheduling algorithm upon t_pr.

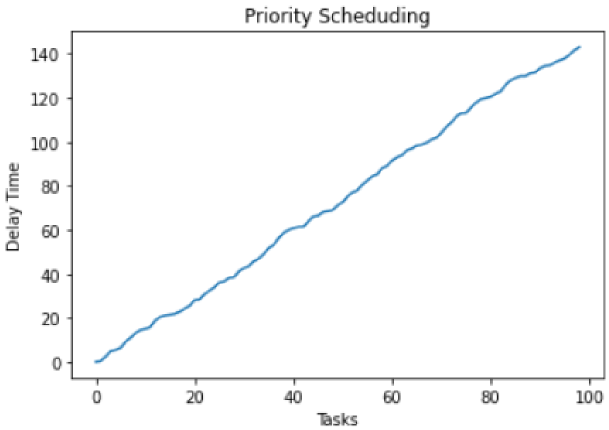
```
3]: # Apply Priority Sceduling
    sum=0
    pr = np.empty((0))
    for i in range(len(t_pr)):
        sum=sum + t_pr[i]
        pr = np.append(pr,sum)
    pr
```

```
3]: array([1.25106667e-01, 4.69150000e-01, 2.62724000e+00, 4.97299000e+00,
           5.53597000e+00, 6.44299333e+00, 9.22661667e+00, 1.10406633e+01,
           1.30423700e+01, 1.45749267e+01, 1.50753533e+01, 1.57947167e+01,
           1.88598300e+01, 2.04549400e+01, 2.11743033e+01, 2.14870700e+01,
           2.19249433e+01, 2.28945200e+01, 2.42706933e+01, 2.56468667e+01,
           2.81490000e+01, 2.83992133e+01, 3.08075167e+01, 3.23713500e+01,
           3.39664600e+01, 3.60932733e+01, 3.66249767e+01, 3.82200867e+01,
```

- Plot the Priority scheduling using matplotlib library. Here we use plt.plot() for plotting and xlabel, ylabel and plt.title() is for labeling and give title to the graph.

```
]: plt.plot(task_number,pr,label = "Priority Sceduling")
   plt.xlabel("Tasks")
   plt.ylabel("Delay Time")
   plt.title("Priority Scheduding")
```

```
]: Text(0.5, 1.0, 'Priority Scheduding')
```

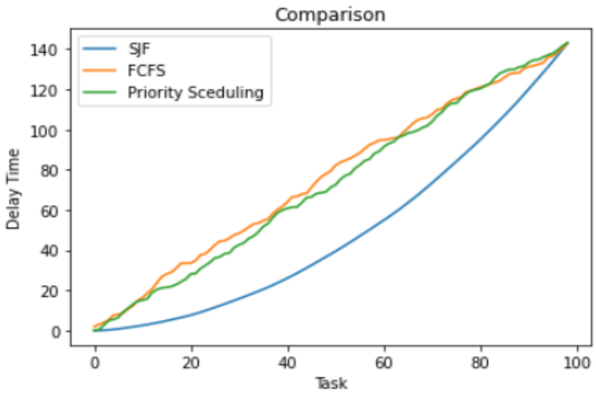


4.2 Comparison

After applying FCFS, SJF and Priority Scheduling algorithm we plotted the graph according to their task and delay time. As no idle time was there all of them finish at the same time but SJF was better since it’s delay time for most of the process is faster than the other algorithms. Priority scheduling algorithm can be faster than FCFS if the priority are set perfectly. FCFS have the largest delay time as it takes whichever task comes first. We find that SJF performs best in this situation. Here is the graphical comparison of the dataset.

```
196]: plt.plot(task_number,sjf,label = "SJF")
      plt.plot(task_number,fcfs,label = "FCFS")
      plt.plot(task_number,pr,label = "Priority Sceduling")
      plt.xlabel("Task")
      plt.ylabel("Delay Time")
      plt.title("Comparison")
      plt.legend()

196]: <matplotlib.legend.Legend at 0x2fc4d57ba30>
```



Chapter 5

5.1 Apply Random Forest in SJF

Recalculating the value of the data and set the target value for random forest. Drop the unnecessary columns.

```
In [180]: # creating targets for random forest for SJF
          #df
          sort_input_delay = input_delay.to_numpy()
          sort_input_delay = np.sort(sort_input_delay)
          sort_final_delay = temp

          target_sjf = df
          target_sjf = target_sjf.drop('Priority Number', axis=1)
          target_sjf = target_sjf.drop('Input Data', axis=1)
          target_sjf['Input Delay'] =sort_input_delay
          target_sjf = target_sjf.drop('Output Data', axis=1)
          target_sjf['Output Delay'] = sort_input_delay
          #target_sjf = target_sjf.drop('Unnamed: 3', axis=1)
          target_sjf['target'] = sort_final_delay
          target_sjf
```

Out[180]:

	Input Delay	Output Delay	target
0	0.000013	0.000013	0.031290
1	0.000013	0.000013	0.031650
2	0.000013	0.000013	0.031810
3	0.000013	0.000013	0.031850
4	0.000013	0.000013	0.032090

```
In [181]: X = target_sjf.drop('target',axis='columns')
          #X
          y = target_sjf.target
          #y
```

Applying Random Forest to the updated dataset.

```
#Apply Random Forest
from sklearn.model_selection import train_test_split
X_train_SJF, X_test_SJF, y_train_SJF, y_test_SJF = train_test_split(X,y,test_size=0.2)
#X_train
```

```
In [210]: from sklearn.ensemble import RandomForestRegressor
          model = RandomForestRegressor(n_estimators=100,oob_score=True)
          model.fit(X_train_SJF, y_train_SJF)

Out[210]: RandomForestRegressor(oob_score=True)
```

The out-of-bag (OOB) error is the average error for each calculated using predictions from the trees that do not contain in their respective bootstrap sample.

Calculate the model score value.

```
In [219]: model.score(X_test_SJF, y_test_SJF)

Out[219]: 0.9999974205907112
```

Apply mean square error to the SJF data to get the error.

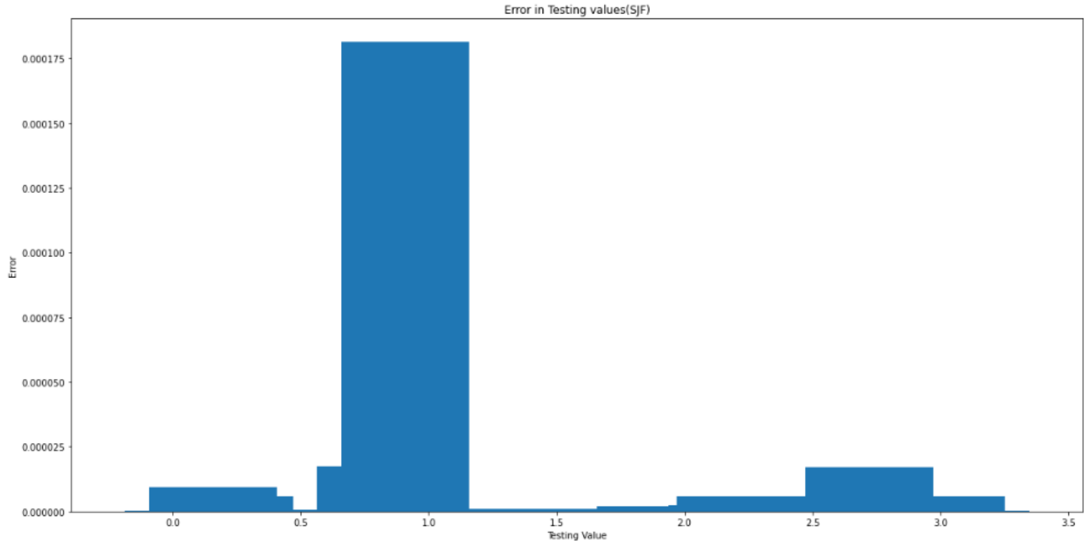
```
In [222]: mse_SJF = ((y_test_SJF - y_predicted_SJF) ** 2)

In [223]: mse_SJF

Out[223]: 97     1.996398e-07
          113    2.504493e-07
          343    3.840875e-07
          124    5.998334e-07
          406    8.513384e-08
                     ...
          227    1.079460e-07
          278    2.245722e-07
          583    6.990138e-08
          106    1.163568e-11
          281    8.110031e-07
          Name: target, Length: 140, dtype: float64
```

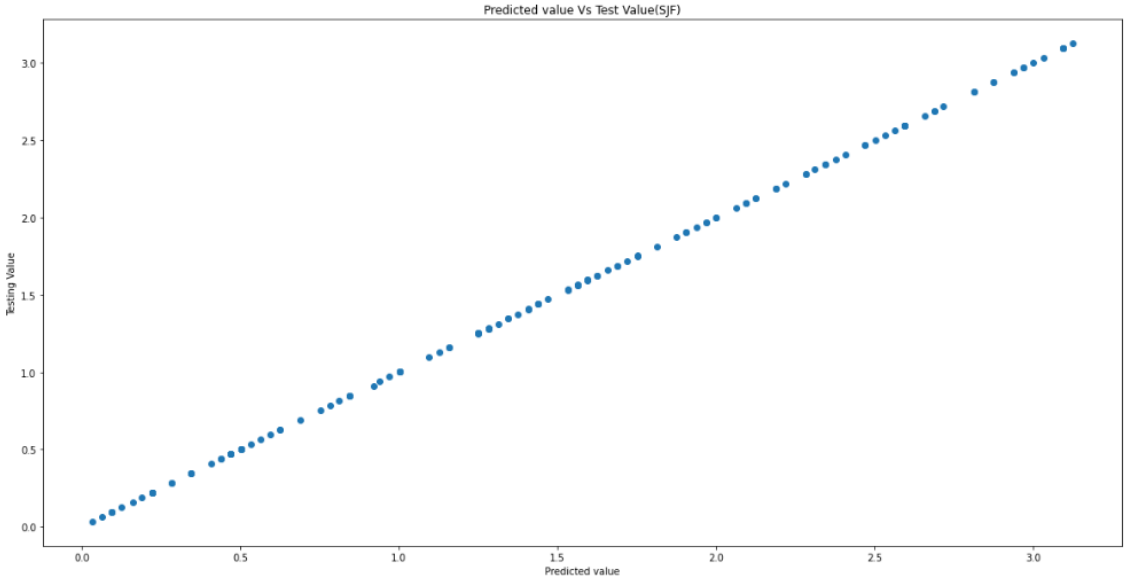
Plotting error bar chart for the SJF dataset.

```
In [224]: plt.bar(y_test_SJF,mse_SJF,width = .5)
          plt.title("Error in Testing values(SJF)")
          plt.xlabel("Testing Value")
          plt.ylabel("Error")
          plt.show()
```



Compare predicated vs tested for the SJF.

```
[225]: plt.scatter(y_predicted_SJF,y_test_SJF)
       plt.title("Predicted value Vs Test Value(SJF)")
       plt.ylabel("Testing Value")
       plt.xlabel("Predicted value")
       plt.show()
```



5.2 Apply Random Forest in FCFS

Recalculating the value of the data and set the target value for random forest. Drop the unnecessary columns.

```
In [227]: #Apply Random Forest for FCFS
          target_FCFS = pd.DataFrame()
          target_FCFS['Input Delay'] = input_delay
          target_FCFS['Output Delay'] = output_delay
          target_FCFS['target'] = final_delay
          target_FCFS
Out[227]:
```

	Input Delay	Output Delay	target
0	0.000480	0.000307	1.125787
1	0.001320	0.000133	3.095203
2	0.000013	0.001093	0.032357
3	0.001040	0.000867	2.439407
4	0.001320	0.000920	3.095990

```
In [228]: X = target_FCFS.drop('target',axis='columns')
          #X
          y = target_FCFS.target
          #y
```

Applying Random Forest to the updated dataset.

```
]: #Apply Random Forest
   from sklearn.model_selection import train_test_split
   X_train_FCFS, X_test_FCFS, y_train_FCFS, y_test_FCFS = train_test_split(X,y,test_size=0.2)
   #X_train
```

```
:n [231]: from sklearn.ensemble import RandomForestRegressor
          model = RandomForestRegressor(n_estimators=20)
          model.fit(X_train_FCFS, y_train_FCFS)
)ut[231]: RandomForestRegressor(n_estimators=20)
```

The out-of-bag (OOB) error is the average error for each calculated using predictions from the trees that do not contain in their respective bootstrap sample.

Calculate the model score value.

```
In [233]: #FCFS Acuracy
          model.score(X_test_FCFS, y_test_FCFS)

Out[233]: 0.9999753531158276
```

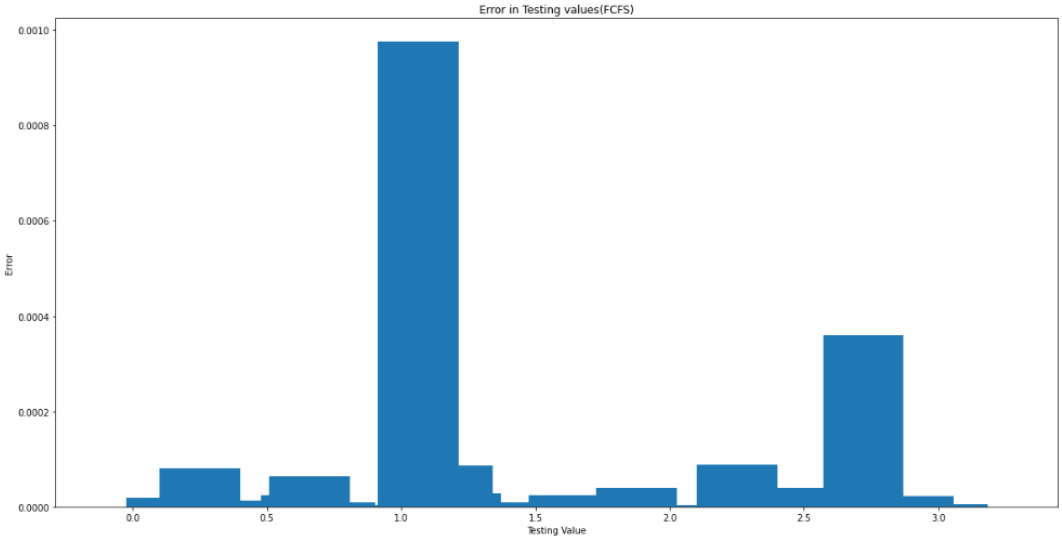
Apply mean square error to the FCFS data to get the error.

```
In [235]: mse_FCFS = ((y_test_FCFS - y_predicted_FCFS) ** 2)
          mse_FCFS

Out[235]: 31     3.326368e-06
          236    3.073593e-06
          621    2.730756e-06
          518    2.647671e-06
          570    6.183511e-08
                     ...
          549    2.567111e-09
          467    3.570840e-04
          22     3.441778e-09
          166    1.672280e-06
          472    4.694444e-08
          Name: target, Length: 140, dtype: float64
```

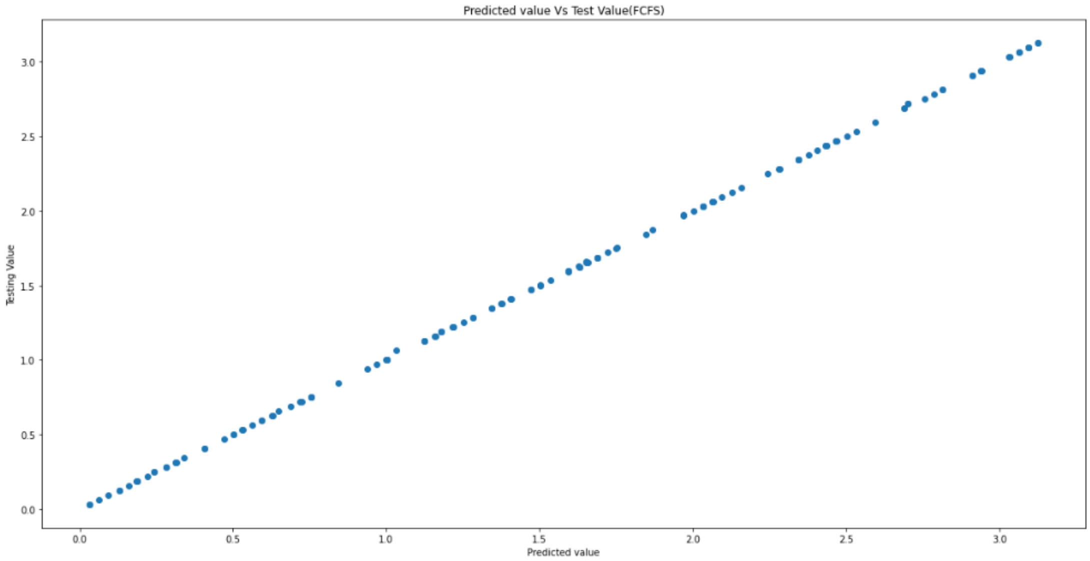
Plotting error bar chart for the FCFS dataset.

```
In [236]: plt.bar(y_test_FCFS,mse_FCFS,width = .3)
          plt.title("Error in Testing values(FCFS)")
          plt.xlabel("Testing Value")
          plt.ylabel("Error")
          plt.show()
```



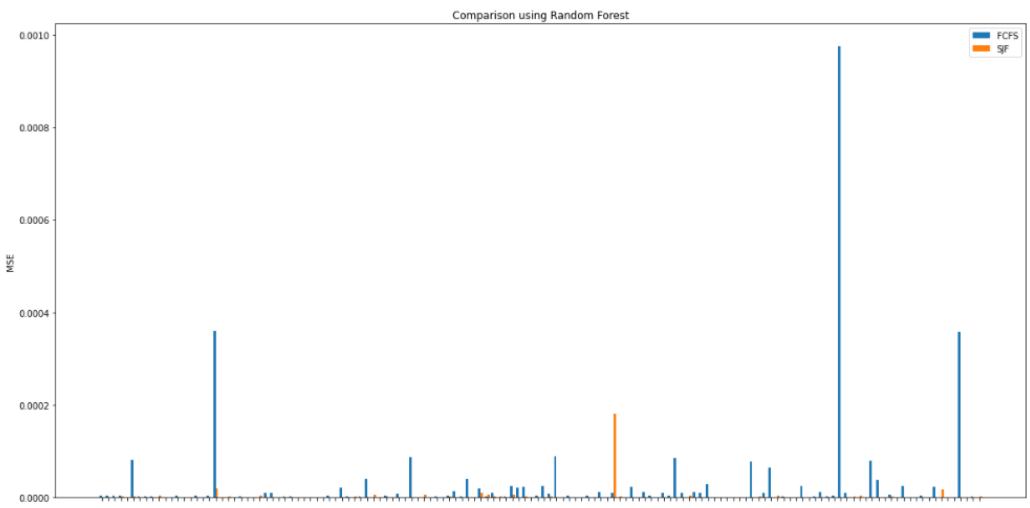
Compare predicated vs tested for the FCFS.

In [237]:
```
plt.scatter(y_predicted_FCFS,y_test_FCFS)
plt.title("Predicted value Vs Test Value(FCFS)")
plt.ylabel("Testing Value")
plt.xlabel("Predicted value")
plt.show()
```



5.3 Comparing between FCFS and SJF in Random Forest

```
# Comparing the error in FCFS and SJF using Random Forest
plt.rcParams["figure.figsize"] = (20,10)
X_axis = np.arange(len(y_test_FCFS))
plt.bar(X_axis-.2,mse_FCFS,0.4,label="FCFS")
plt.bar(X_axis+.2,mse_SJF,0.4,label="SJF")
plt.xticks(X_axis, y_test_FCFS)
plt.xlabel("Test Values")
plt.ylabel("MSE")
plt.title("Comparison using Random Forest")
plt.legend()
plt.show()
```



So after applying the random forest in the model we conclude that the FCFS performs weak in response to SJF in almost every value.

5.4 Apply Decision Tree in FCFS

Recalculating the value of the data and set the target value for Decision Tree. Drop the unnecessary columns.

FCFS

```
In [54]: d_FCFS = target_FCFS
```

```
In [55]: X = d_FCFS.drop('target',axis='columns')
         #X
         y = d_FCFS.target
```

Applying Random Forest to the updated dataset.

```
In [57]: from sklearn import tree
         model = tree.DecisionTreeRegressor()
```

```
In [58]: model.fit(X_train_FCFS, y_train_FCFS)
Out[58]: DecisionTreeRegressor()
```

Calculate model score and predicated value.

```
In [59]: model.score(X_test_FCFS, y_test_FCFS)
Out[59]: 0.9999999575341306
```

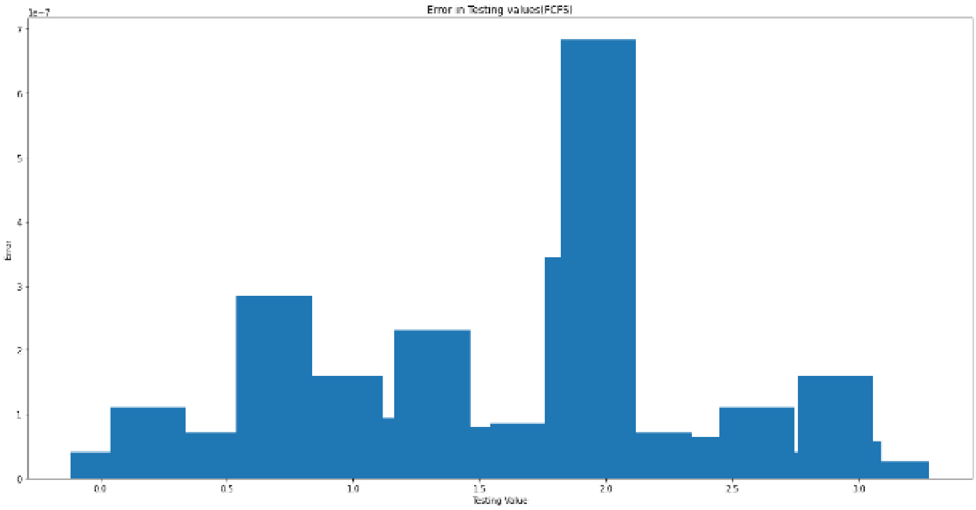
```
In [60]: y_predicted_FCFS = model.predict(X_test_FCFS)
         y_predicted_FCFS
```

Calculate MSE for the FCFS after applying decision tree.

```
In [61]: mse_FCFS = ((y_test_FCFS - y_predicted_FCFS) ** 2)
         mse_FCFS
Out[61]: 87     8.711111e-09
         421    1.777778e-10
         89     1.777778e-10
         367    7.111111e-10
         483    1.777778e-10
                    ...
```

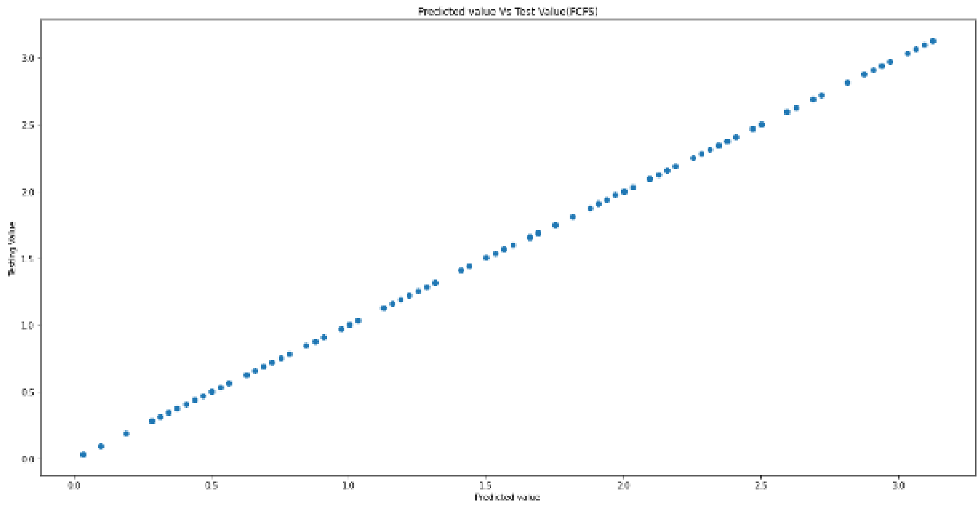
Plotting error bar chart for the SJF dataset.

```
In [62]: plt.bar(y_test_FCFS,mse_FCFS,width = .3)
         plt.title("Error in Testing values(FCFS)")
         plt.xlabel("Testing Value")
         plt.ylabel("Error")
         plt.show()
```



Compare predicated vs tested for the FCFS.

```
In [63]: plt.scatter(y_predicted_FCFS,y_test_FCFS)
         plt.title("Predicted value Vs Test Value(FCFS)")
         plt.ylabel("Testing Value")
         plt.xlabel("Predicted value")
         plt.show()
```



5.5 Apply Decision Tree in SJF

Recalculating the value of the data and set the target value for Decision Tree. Drop the unnecessary columns.

```
In [180]: # creating targets for random forest for SJF
#df
sort_input_delay = input_delay.to_numpy()
sort_input_delay = np.sort(sort_input_delay)
sort_final_delay = temp

target_sjf = df
target_sjf = target_sjf.drop('Priority Number', axis=1)
target_sjf = target_sjf.drop('Input Data', axis=1)
target_sjf['Input Delay'] =sort_input_delay
target_sjf = target_sjf.drop('Output Data', axis=1)
target_sjf['Output Delay'] = sort_input_delay
#target_sjf = target_sjf.drop('Unnamed: 3', axis=1)
target_sjf['target'] = sort_final_delay
target_sjf
```

Out[180]:

	Input Delay	Output Delay	target
0	0.000013	0.000013	0.031290
1	0.000013	0.000013	0.031650
2	0.000013	0.000013	0.031810
3	0.000013	0.000013	0.031850
4	0.000013	0.000013	0.032090

```
In [181]: X = target_sjf.drop('target',axis='columns')
#X
y = target_sjf.target
#y
```

Applying Decision Tree to the updated dataset.

```
#Apply Random Forest
from sklearn.model_selection import train_test_split
X_train_SJF, X_test_SJF, y_train_SJF, y_test_SJF = train_test_split(X,y,test_size=0.2)
#X_train
```

```
In [66]: from sklearn import tree
model = tree.DecisionTreeRegressor()
```

```
In [67]: model.fit(X_train_SJF, y_train_SJF)
Out[67]: DecisionTreeRegressor()
```

Calculate the model score value.

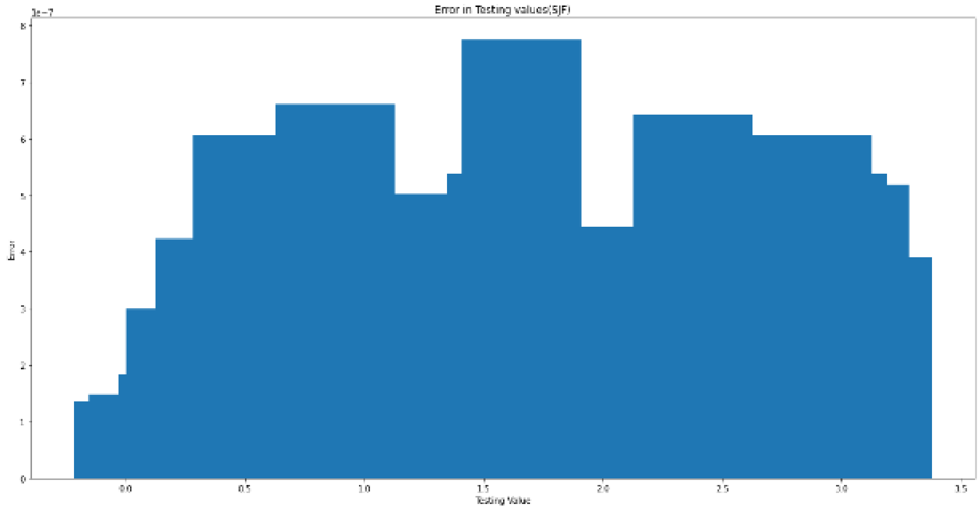
```
In [68]: model.score(X_test_SJF, y_test_SJF)
Out[68]: 0.9999998000845888
```

Apply mean square error to the SJF data to get the error.

```
In [70]: mse_SJF = ((y_test_SJF - y_predicted_SJF) ** 2)
mse_SJF
Out[70]: 196    8.100000e-09
550    3.849070e-08
31     7.840000e-08
604    1.109511e-08
180    2.249469e-07
```

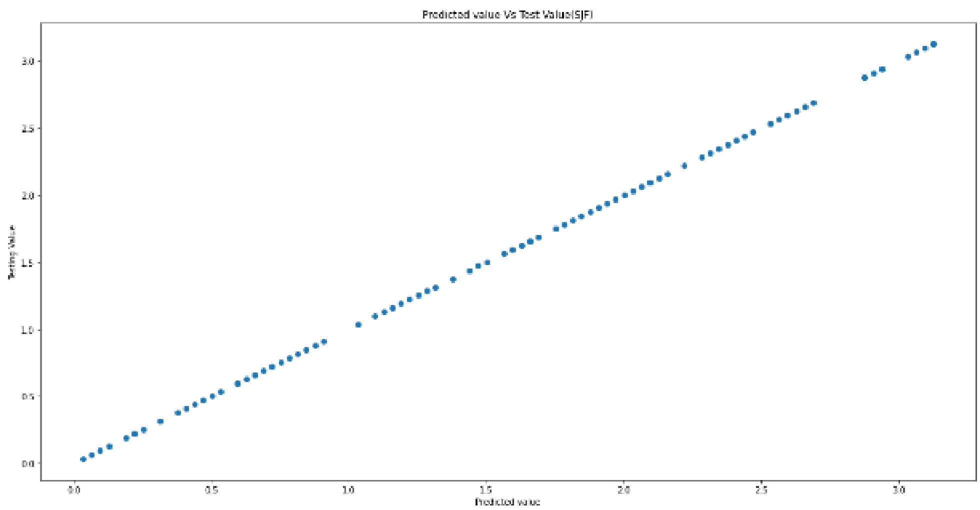
Plotting error bar chart for the SJF dataset.

```
In [71]: plt.bar(y_test_SJF,mse_SJF,width = .5)
         plt.title("Error in Testing values(SJF)")
         plt.xlabel("Testing Value")
         plt.ylabel("Error")
         plt.show()
```



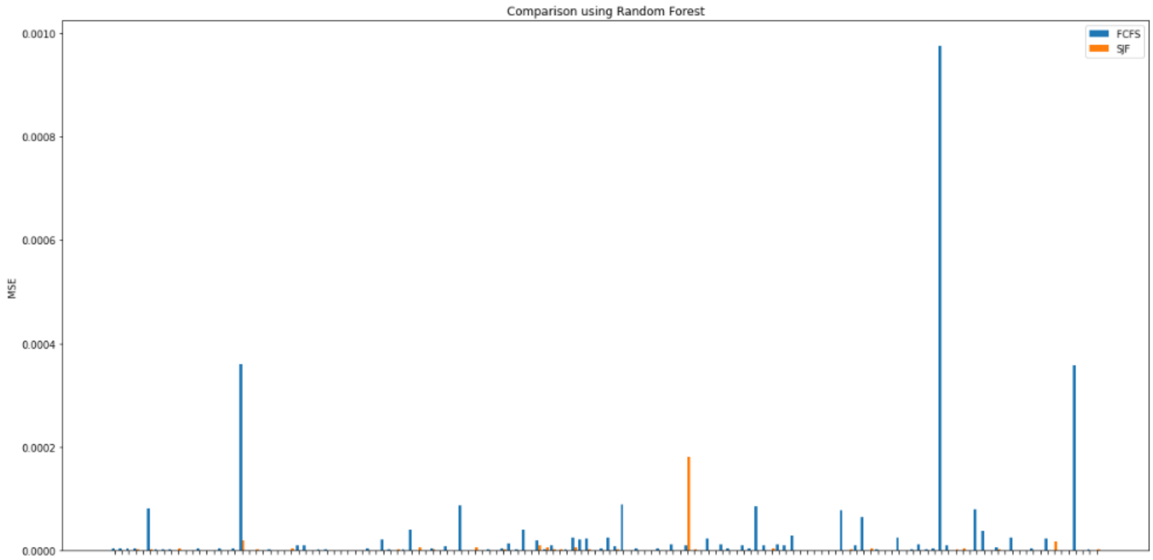
Compare predicated vs tested for the SJF.

```
In [72]: plt.scatter(y_predicted_SJF,y_test_SJF)
         plt.title("Predicted value Vs Test Value(SJF)")
         plt.ylabel("Testing Value")
         plt.xlabel("Predicted value")
         plt.show()
```



5.6 Comparing between FCFS and SJF in Decision Tree

```
]: # Comparing the error in FCFS and SJF using Random Forest
plt.rcParams["figure.figsize"] = (20,10)
X_axis = np.arange(len(y_test_FCFS))
plt.bar(X_axis-.2,mse_FCFS,0.4,label="FCFS")
plt.bar(X_axis+.2,mse_SJF,0.4,label="SJF")
plt.xticks(X_axis, y_test_FCFS)
plt.xlabel("Test Values")
plt.ylabel("MSE")
plt.title("Comparison using Random Forest")
plt.legend()
plt.show()
```



5.7 Apply Linear Regression in FCFS

Recalculating the value of the data and set the target value for Decision Tree. Drop the unnecessary columns.

FCFS

```
In [54]: d_FCFS = target_FCFS
```

```
In [55]: X = d_FCFS.drop('target',axis='columns')
         #X
         y = d_FCFS.target
```

Applying Random Forest to the updated dataset.

```
In [74]: from sklearn import linear_model
         model =linear_model.LinearRegression()
```

```
In [78]: model.fit(X_train_FCFS, y_train_FCFS)

Out[78]: LinearRegression()
```

Calculate model score and predicated value.

```
In [79]: model.score(X_test_FCFS, y_test_FCFS)
Out[79]: 1.0
```

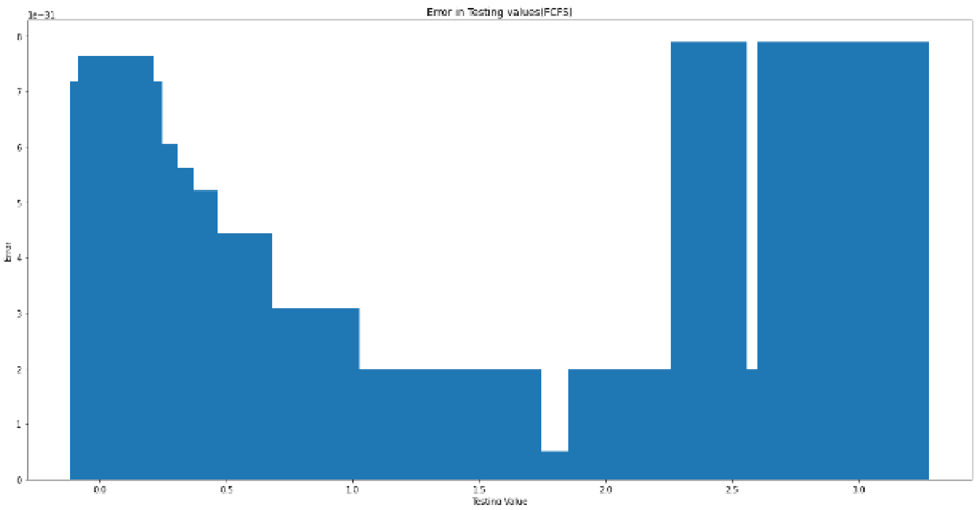
```
In [80]: y_predicted_FCFS = model.predict(X_test_FCFS)
         y_predicted_FCFS
Out[80]: array([1.8762    , 2.12592   , 0.09489667, 1.00105333, 0.65715667,
                0.96936333, 2.28242333, 1.0012    , 1.43927333, 1.62621333,
                0.6887    , 1.15695667, 1.18807333, 1.31308667, 0.28146333,
                1.59545667, 0.84447   , 0.15737   , 0.09463   , 1.34537667,
```

Calculate MSE for the FCFS after applying decision tree.

```
In [81]: mse_FCFS = ((y_test_FCFS - y_predicted_FCFS) ** 2)
         mse_FCFS
Out[81]: 118    4.930381e-32
         193    0.000000e+00
         353    6.704162e-31
         445    1.972152e-31
         400    3.081488e-31
                    ...
```

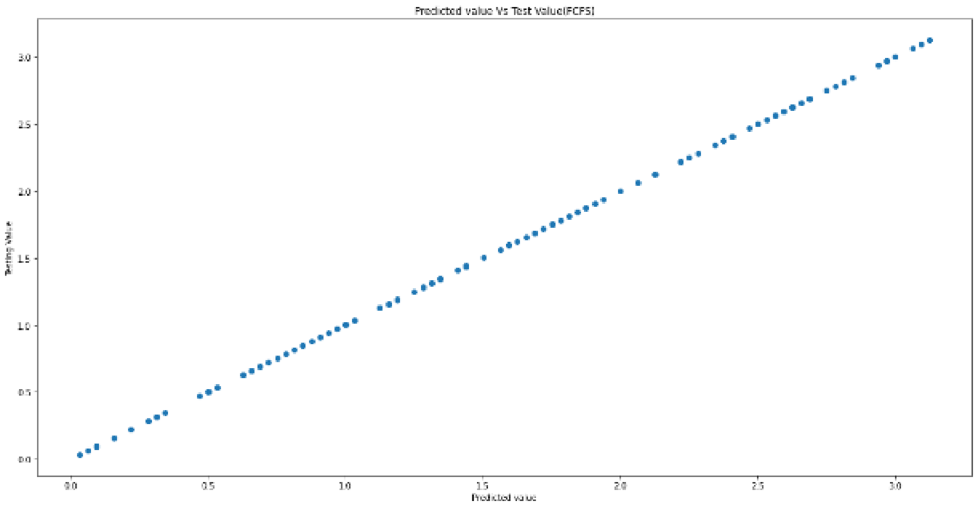
Plotting error bar chart for the SJF dataset.

```
In [82]: plt.bar(y_test_FCFS,mse_FCFS,width = .3)
         plt.title("Error in Testing values(FCFS)")
         plt.xlabel("Testing Value")
         plt.ylabel("Error")
         plt.show()
```



Compare predicated vs tested for the FCFS.

```
In [83]: plt.scatter(y_predicted_FCFS,y_test_FCFS)
         plt.title("Predicted value Vs Test Value(FCFS)")
         plt.ylabel("Testing Value")
         plt.xlabel("Predicted value")
         plt.show()
```



5.8 Apply Linear Regression in SJF

Recalculating the value of the data and set the target value for Linear Regression. Drop the unnecessary columns.

```
In [180]: # creating targets for random forest for SJF
          #df
          sort_input_delay = input_delay.to_numpy()
          sort_input_delay = np.sort(sort_input_delay)
          sort_final_delay = temp

          target_sjf = df
          target_sjf = target_sjf.drop('Priority Number', axis=1)
          target_sjf = target_sjf.drop('Input Data', axis=1)
          target_sjf['Input Delay'] =sort_input_delay
          target_sjf = target_sjf.drop('Output Data', axis=1)
          target_sjf['Output Delay'] = sort_input_delay
          #target_sjf = target_sjf.drop('Unnamed: 3', axis=1)
          target_sjf['target'] = sort_final_delay
          target_sjf
```

Out[180]:

	Input Delay	Output Delay	target
0	0.000013	0.000013	0.031290
1	0.000013	0.000013	0.031650
2	0.000013	0.000013	0.031810
3	0.000013	0.000013	0.031850
4	0.000013	0.000013	0.032090

```
In [181]: X = target_sjf.drop('target',axis='columns')
          #X
          y = target_sjf.target
          #y
```

Applying Linear Regression to the updated dataset.

```
In [86]: model.fit(X_train_SJF, y_train_SJF)

Out[86]: LinearRegression()
```

Calculate the model score value.

```
In [87]: model.score(X_test_SJF, y_test_SJF)

Out[87]: 0.9999998467395071
```

Apply mean square error to the SJF data to get the error.

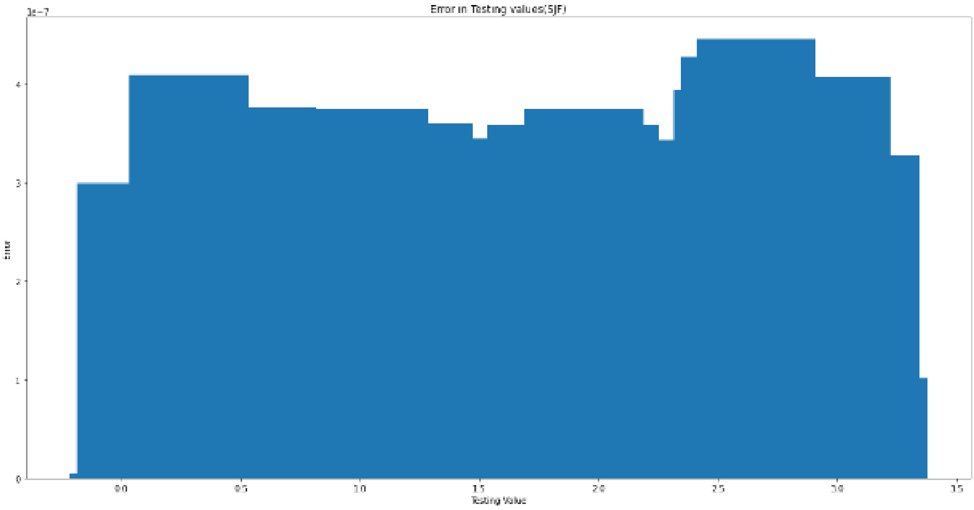
```
 [88]: y_predicted_SJF = model.predict(X_test_SJF)

 [89]: mse_SJF = ((y_test_SJF - y_predicted_SJF) ** 2)
       mse_SJF

t[89]: 220    7.084534e-08
       489    1.290235e-07
       595    1.675472e-09
       44     2.875034e-09
       568    7.898470e-08
                  ...
```

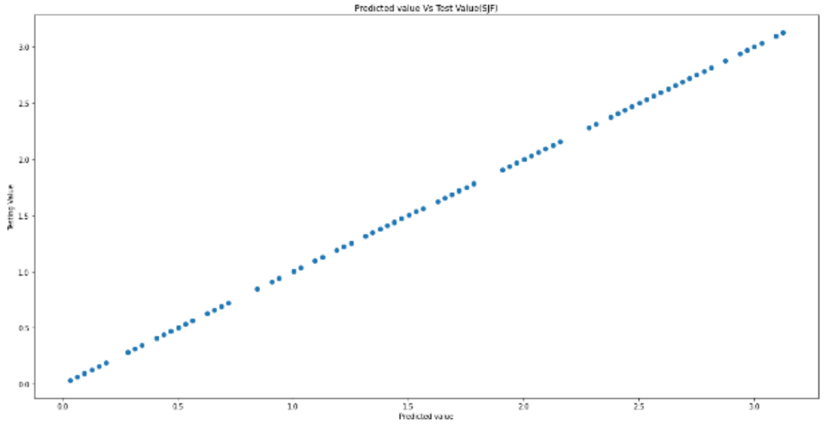
Plotting error bar chart for the SJF dataset.

```
n [90]: plt.bar(y_test_SJF,mse_SJF,width = .5)
        plt.title("Error in Testing values(SJF)")
        plt.xlabel("Testing Value")
        plt.ylabel("Error")
        plt.show()
```



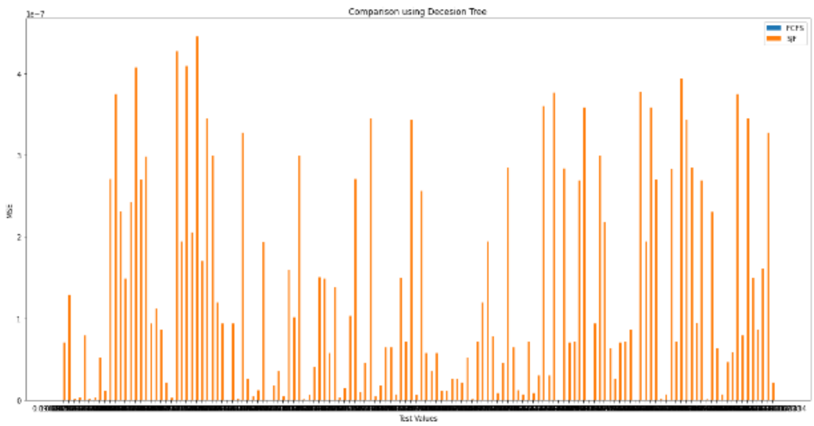
Compare predicated vs tested for the SJF.

```
In [91]: plt.scatter(y_predicted_SJF,y_test_SJF)
         plt.title("Predicted value Vs Test Value(SJF)")
         plt.ylabel("Testing Value")
         plt.xlabel("Predicted value")
         plt.show()
```



5.9 Comparing between FCFS and SJF in Linear Regression

```
In [92]: # Comparing the error in FCFS and SJF using Decesion Tree
         plt.rcParams["figure.figsize"] = (20,10)
         X_axis = np.arange(len(y_test_FCFS))
         plt.bar(X_axis-.2,mse_FCFS,0.4,label="FCFS")
         plt.bar(X_axis+.2,mse_SJF,0.4,label="SJF")
         plt.xticks(X_axis, y_test_FCFS)
         plt.xlabel("Test Values")
         plt.ylabel("MSE")
         plt.title("Comparison using Decesion Tree")
         plt.legend()
         plt.show()
```



Finally, we can say that Linear Regression gives the best result for the FCFS and Decision Tree gives the best result for the SJF. Random Forest gives a average accuracy for both the algorithm.

Chapter 6

6.1 Conclusion

In this project, we used Numpy, Matplotlib and Pandas as python libraires. We preprocessed the dataset and important features were selected using the feature extraction technique. Performance evaluation is done here by FCFS, SJF and Priority Scheduling algorithm. In cloud computing the SJF gives the lowest delay time per process and FCFS gives the highest. Priority Scheduling will give best performance if we give the right priority to the right process. From the graph from 20 to 80 task best performance without any doubt is SJF.