Lab 2 - Security of Cyber Physical Systems

Student Name: Orkun Ratip

Student ID: 32899904

Problem definition

There are 5 users, and each have 10 tasks to perform. We are given two datasets: training and testing. Training datasets contain hourly price curve of a day and a classification as abnormal/normal. First task is to predict the price curves of the testing data as normal/abnormal by using a classification algorithm. We are also given an abnormal hourly price curve which we can use to schedule the predicted abnormal entries in our testing data. There are multiple constraints when it comes to the scheduling problem. In our case the constraints are energy demand and operation hours and maximum allowed energy. We will have to generate a linear programming solution to schedule tasks while minimising the energy cost.

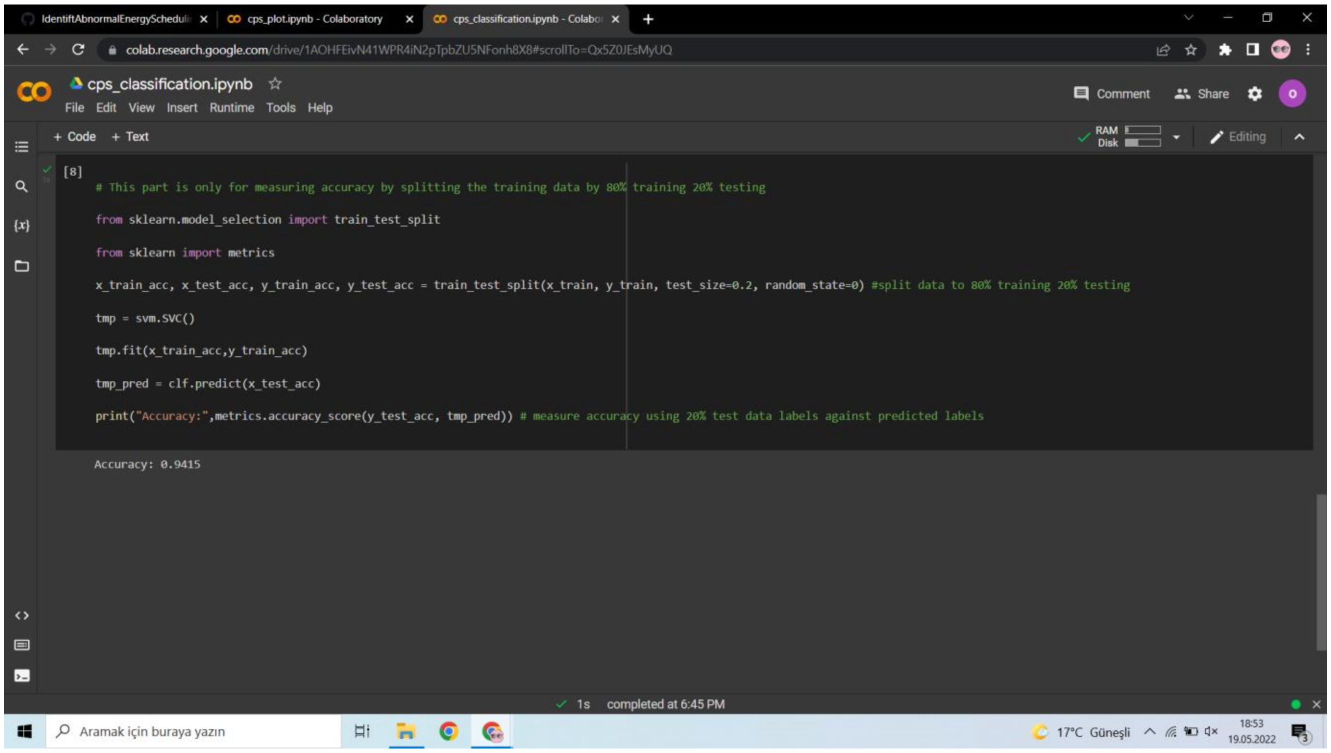
SVM Classifier

How does SVM work?

SVM is a supervised machine learning algorithm that uses hyperplanes to separate data. The hyperplane that gives the maximum margin is selected to be the best solution. The hyperplane depends on the dimension of the data. In this coursework there are 24 columns of price values which suggests that we have a 24-dimensional problem. SVM is effective since a model can be built flexibly by hyperparameter tuning to adapt to the given problem better.

My SVM model

I used python to implement my SVM model by using google colaboratory as my preferred IDE. I used sklearn.svm.svc library for building the model. Since the training and testing data were initially separated datasets, I used 100% of the training data for the model. I used various hyperparameter combinations which involved kernel, gamma and C. I found that for this specific version a 'linear' kernel was more effective with 0.9415 accuracy, whereas a polynomial kernel gave 0.78 accuracy. For C I used a wide range of values from 0.1 to 50. However, the C value had a negligible effect on accuracy, slightly changing it less than 0.01. Moreover, gamma value did not have any visual effect on the accuracy. Therefore, I chose to go with default C and gamma values. I used different preprocessing techniques such as sklearn.preprocessing.Normalizer and sklearn.preprocessingMinMaxScaler. MinMaxScaler() gave the best accuracy. Since the testing data is not labelled, the performance cannot be measured. I did a 20% train/test split on the training data just for performance measure's sake. My final accuracy was 0.9415.

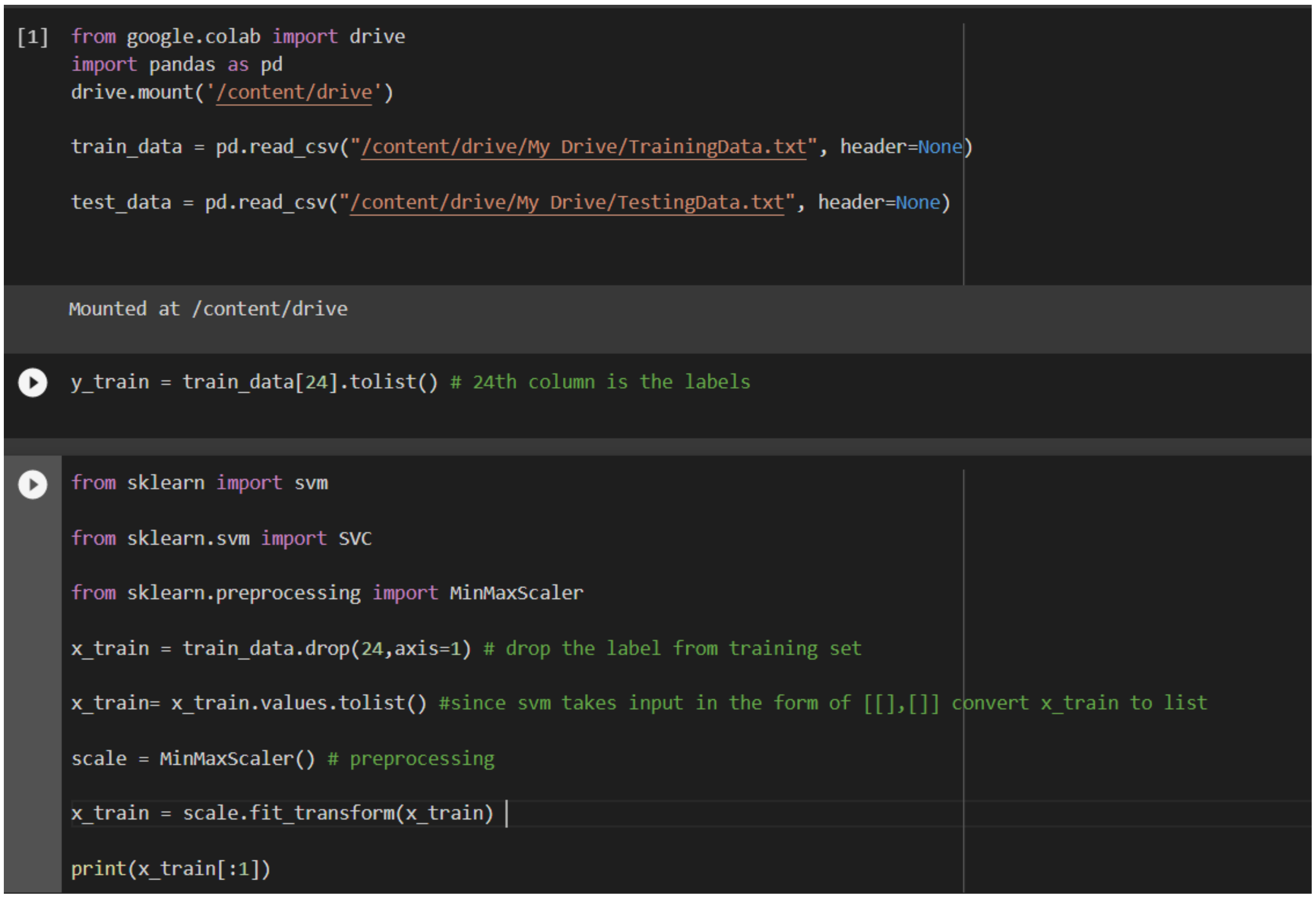


```
[8]
# This part is only for measuring accuracy by splitting the training data by 80% training 20% testing

from sklearn.model_selection import train_test_split

from sklearn import metrics

x_train_acc, x_test_acc, y_train_acc, y_test_acc = train_test_split(x_train, y_train, test_size=0.2, random_state=0) #split data to 80% training 20% testing

tmp = svm.SVC()

tmp.fit(x_train_acc,y_train_acc)

tmp_pred = clf.predict(x_test_acc)

print("Accuracy:",metrics.accuracy_score(y_test_acc, tmp_pred)) # measure accuracy using 20% test data labels against predicted labels
```

```
Accuracy: 0.9415
```

Code and explanation

I first read the training and testing datasets from my google drive. I separated the training data into two parts: x_train and y_train which are training feature space and labels respectively. Then I used a scaler for preprocessing.



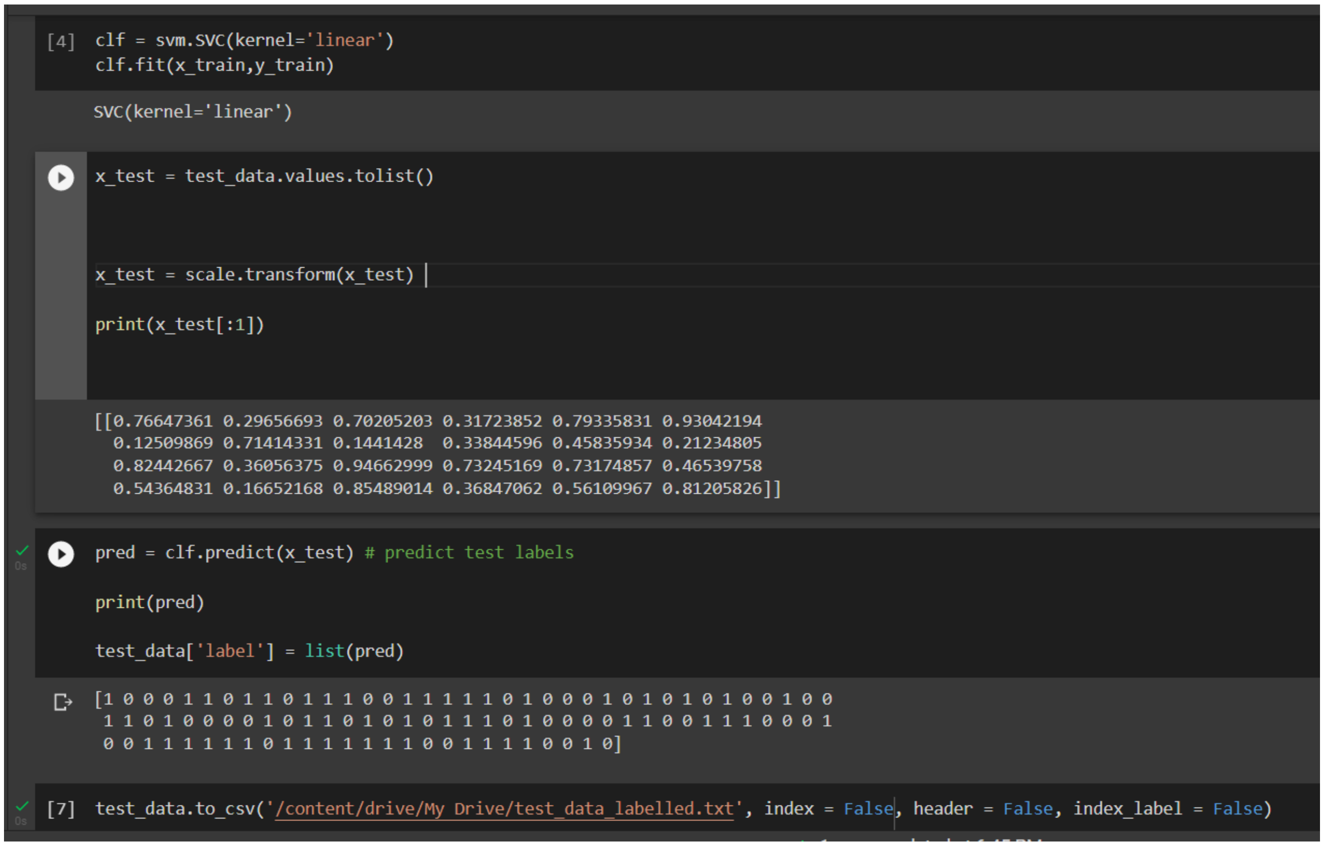
```
[1] from google.colab import drive
    import pandas as pd
    drive.mount('/content/drive')

    train_data = pd.read_csv("/content/drive/My Drive/TrainingData.txt", header=None)

    test_data = pd.read_csv("/content/drive/My Drive/TestingData.txt", header=None)
```

```
Mounted at /content/drive
```

```
y_train = train_data[24].tolist() # 24th column is the labels
```

```
from sklearn import svm

from sklearn.svm import SVC

from sklearn.preprocessing import MinMaxScaler

x_train = train_data.drop(24,axis=1) # drop the label from training set

x_train= x_train.values.tolist() #since svm takes input in the form of [[],[]] convert x_train to list

scale = MinMaxScaler() # preprocessing

x_train = scale.fit_transform(x_train)

print(x_train[:1])
```

I built the svm classifier to train it with x_train (feature space) and y_train (labels). I used a scaler on the test data as well and with the .predict() function of sklearn svm library I predicted the labels for the test data. I added the predicted labels as a column named 'label' and saved the file in the form of csv.



```
[4] clf = svm.SVC(kernel='linear')
    clf.fit(x_train,y_train)

    SVC(kernel='linear')

    x_test = test_data.values.tolist()


    x_test = scale.transform(x_test)

    print(x_test[:1])

    [[0.76647361 0.29656693 0.70205203 0.31723852 0.79335831 0.93042194
      0.12509869 0.71414331 0.1441428  0.33844596 0.45835934 0.21234805
      0.82442667 0.36056375 0.94662999 0.73245169 0.73174857 0.46539758
      0.54364831 0.16652168 0.85489014 0.36847062 0.56109967 0.81205826]]

    pred = clf.predict(x_test) # predict test labels

    print(pred)

    test_data['label'] = list(pred)

    [1 0 0 0 1 1 0 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0
     1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 0 0 1 1 1 0 0 0 1
     0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0]

[7] test_data.to_csv('/content/drive/My Drive/test_data_labelled.txt', index = False, header = False, index_label = False)
```

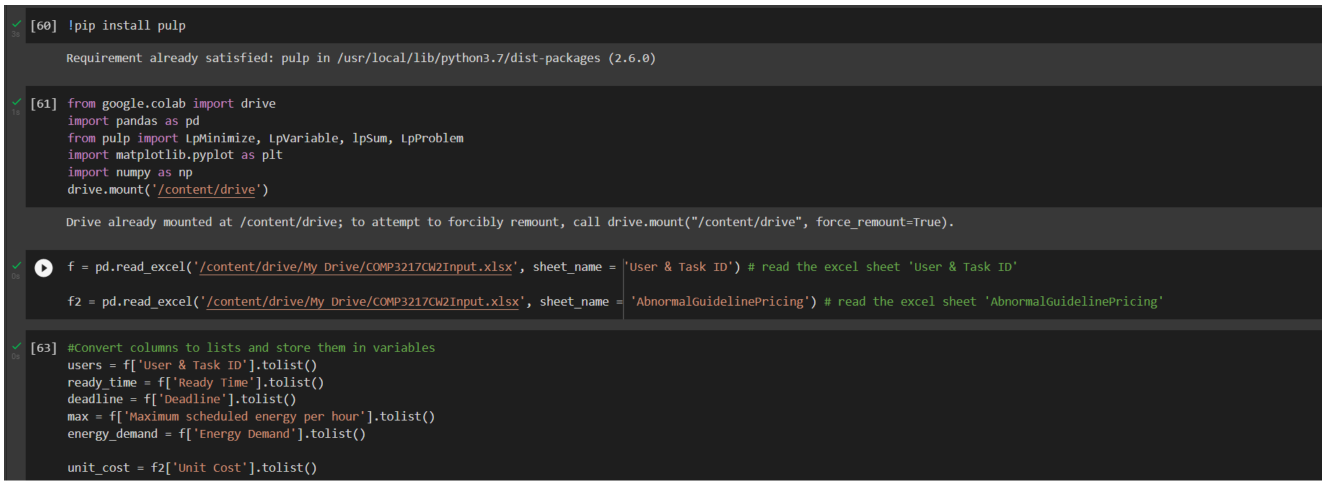
Linear Programming solution

Pulp Lp explanation

A linear problem consists of several linear equations and will most likely have multiple solutions. In this task we aim to minimise energy cost by properly generating a schedule for various tasks to be performed. I implemented a solution using google colab and python. My preferred library for Lp is pulp since it is less error prone compared to scipy.

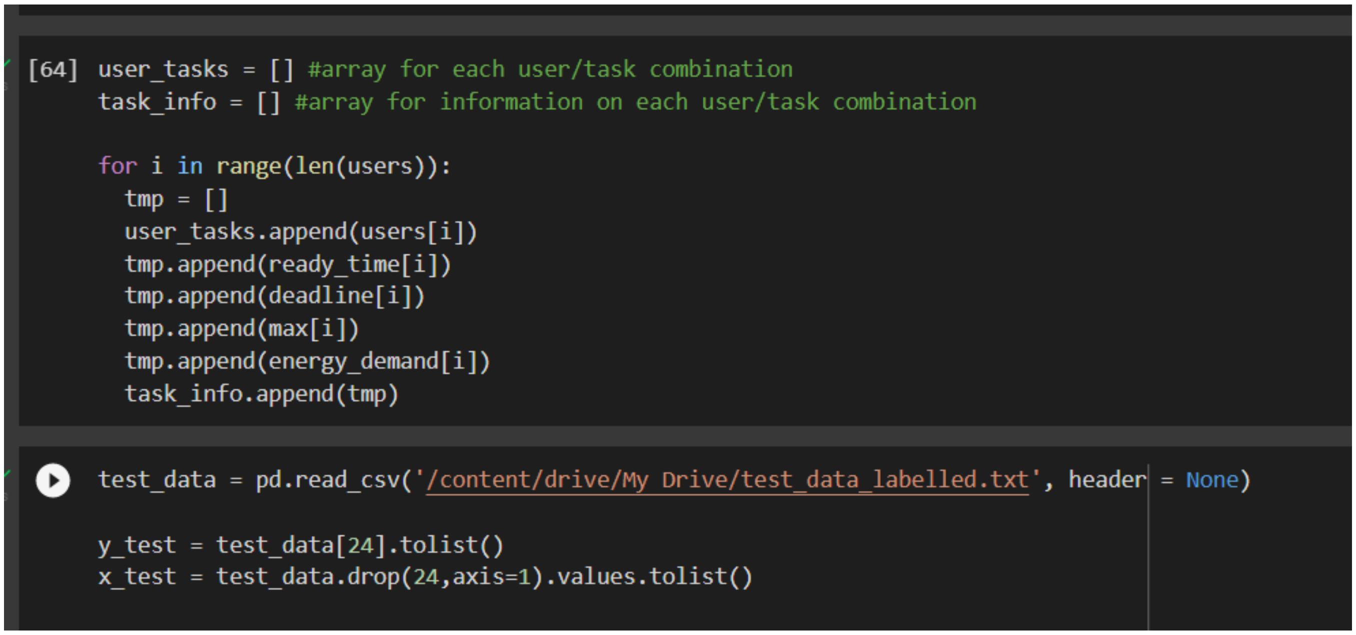
Code and explanation

I first read the excel sheets User & Task ID and AbnormalGuidelinePricing. I converted each column to an array.



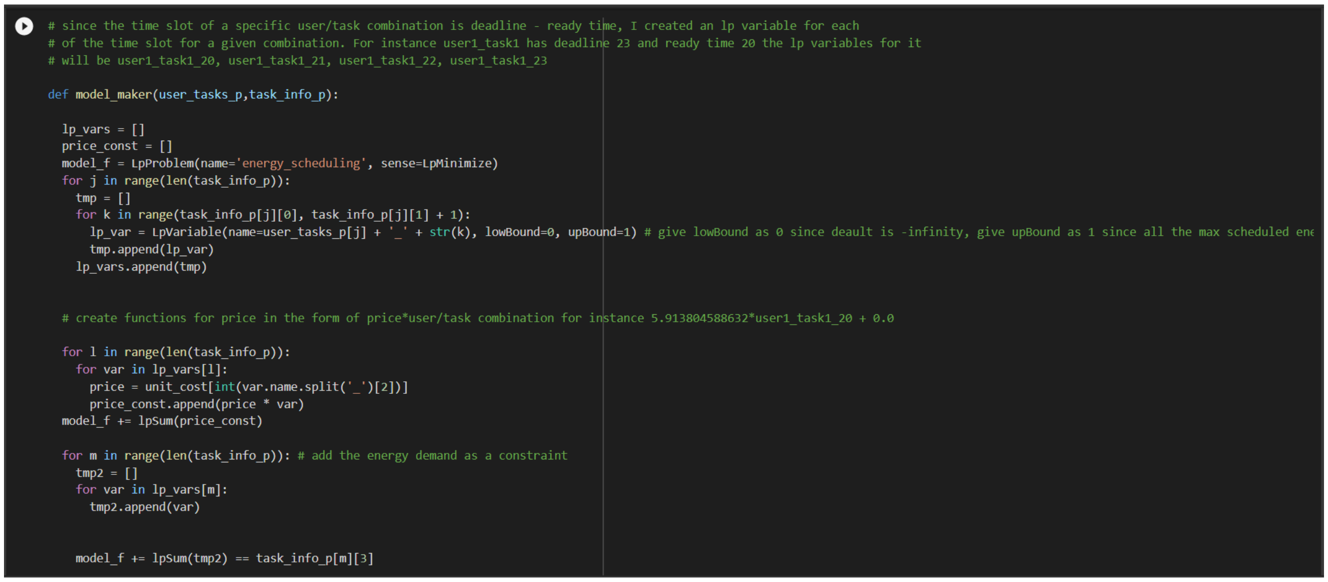
```
[60] !pip install pulp

     Requirement already satisfied: pulp in /usr/local/lib/python3.7/dist-packages (2.6.0)

[61] from google.colab import drive
     import pandas as pd
     from pulp import LpMinimize, LpVariable, lpSum, LpProblem
     import matplotlib.pyplot as plt
     import numpy as np
     drive.mount('/content/drive')

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

     f = pd.read_excel('/content/drive/My Drive/COMP3217CW2Input.xlsx', sheet_name = 'User & Task ID') # read the excel sheet 'User & Task ID'

     f2 = pd.read_excel('/content/drive/My Drive/COMP3217CW2Input.xlsx', sheet_name = 'AbnormalGuidelinePricing') # read the excel sheet 'AbnormalGuidelinePricing'

[63] #Convert columns to lists and store them in variables
     users = f['User & Task ID'].tolist()
     ready_time = f['Ready Time'].tolist()
     deadline = f['Deadline'].tolist()
     max = f['Maximum scheduled energy per hour'].tolist()
     energy_demand = f['Energy Demand'].tolist()

     unit_cost = f2['Unit Cost'].tolist()
```

I created an array of strings and an array of arrays to store user/task combinations (e.g. user1_task1) and task information (e.g. deadline: 23) respectively. I separated the data frame into two parts: curves and labels.

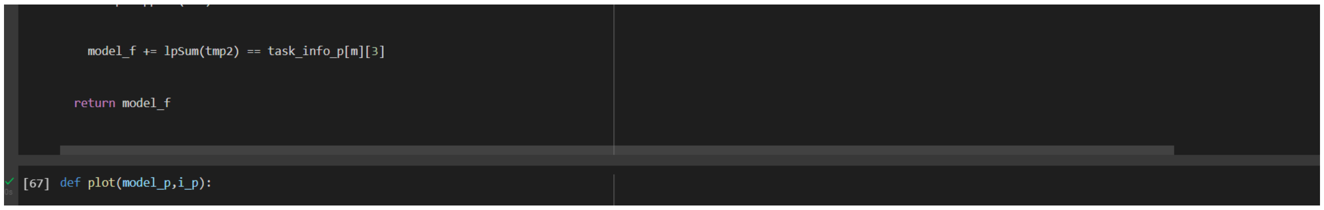


```
[64] user_tasks = [] #array for each user/task combination
     task_info = [] #array for information on each user/task combination

     for i in range(len(users)):
       tmp = []
       user_tasks.append(users[i])
       tmp.append(ready_time[i])
       tmp.append(deadline[i])
       tmp.append(max[i])
       tmp.append(energy_demand[i])
       task_info.append(tmp)

     test_data = pd.read_csv('/content/drive/My Drive/test_data_labelled.txt', header = None)

     y_test = test_data[24].tolist()
     x_test = test_data.drop(24,axis=1).values.tolist()
```

I implemented a function to create an LP model. First for loop will create LP variables with user/task/hour combination such as user1_task1_20. Each variable will have a lower bound of 0 (default lower bound is -inf) and an upper bound of 1(max hourly energy). The next for loop will combine the abnormal hourly price curve with the LP variables created. The price will be pinpointed by splitting variable.name (user_task_hour) from the last underscore, which will give the hour. The index of the price array that corresponds to this hour gives the price. Then the price will be multiplied by the LP variable to create a linear expression in the form of price*variable (e.g., 5.913804588632*user1_task1_20 + 0.0). Then LP sum function will be used to sum up all the linear expressions to form the formula. The last for loop will add the energy demand as an expression and it will also be added using LP sum.



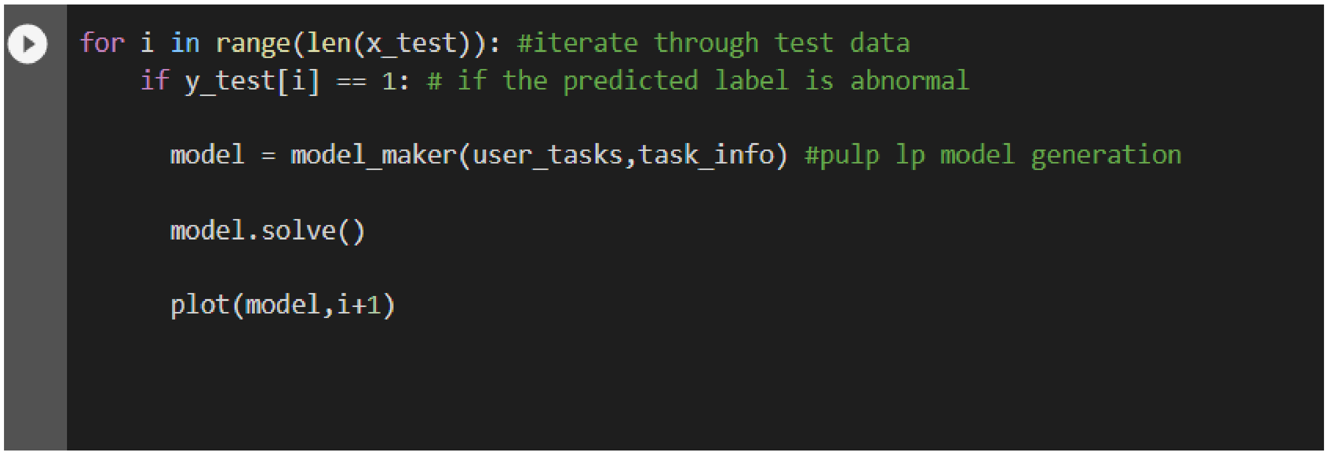
```
# since the time slot of a specific user/task combination is deadline - ready time, I created an lp variable for each
# of the time slot for a given combination. For instance user1_task1 has deadline 23 and ready time 20 the lp variables for it
# will be user1_task1_20, user1_task1_21, user1_task1_22, user1_task1_23

def model_maker(user_tasks_p,task_info_p):

  lp_vars = []
  price_const = []
  model_f = LpProblem(name='energy_scheduling', sense=LpMinimize)
  for j in range(len(task_info_p)):
    tmp = []
    for k in range(task_info_p[j][0], task_info_p[j][1] + 1):
      lp_var = LpVariable(name=user_tasks_p[j] + '_' + str(k), lowBound=0, upBound=1) # give lowBound as 0 since deault is -infinity, give upBound as 1 since all the max scheduled ene
      tmp.append(lp_var)
    lp_vars.append(tmp)


  # create functions for price in the form of price*user/task combination for instance 5.913804588632*user1_task1_20 + 0.0

  for l in range(len(task_info_p)):
    for var in lp_vars[l]:
      price = unit_cost[int(var.name.split('_')[2])]
      price_const.append(price * var)
  model_f += lpSum(price_const)

  for m in range(len(task_info_p)): # add the energy demand as a constraint
    tmp2 = []
    for var in lp_vars[m]:
      tmp2.append(var)


    model_f += lpSum(tmp2) == task_info_p[m][3]
```



```
    model_f += lpSum(tmp2) == task_info_p[m][3]


  return model_f
```

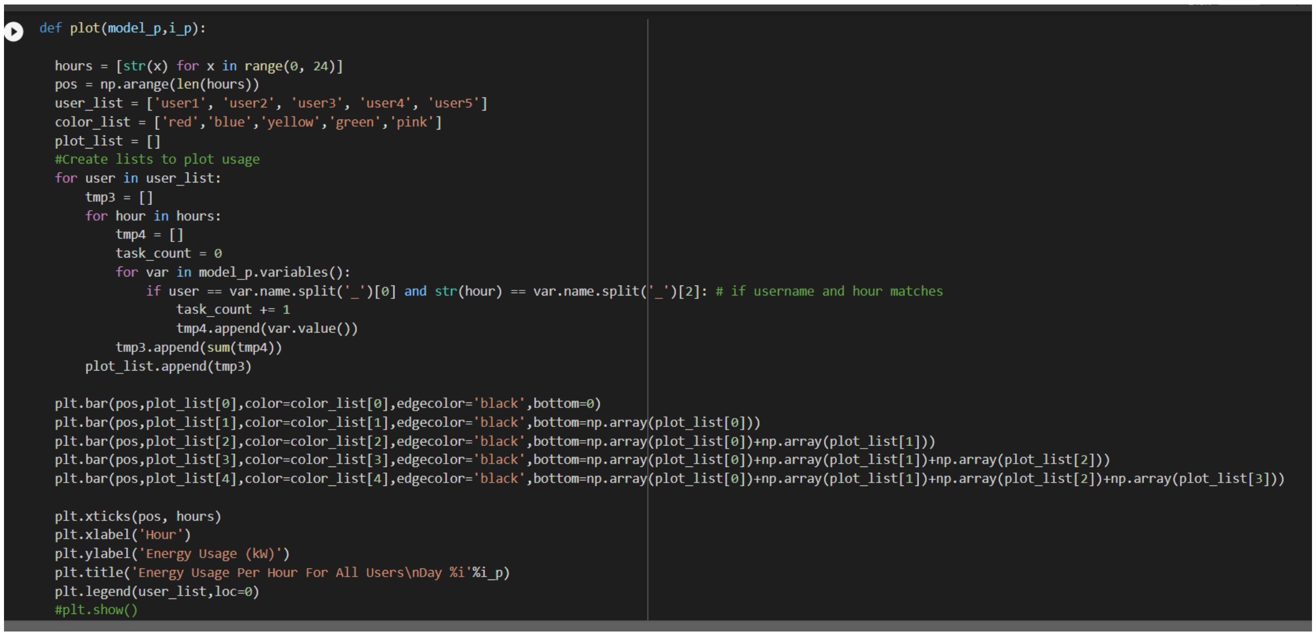
```
[67] def plot(model_p,i_p):
```

After creating the model with appropriate expressions, I used solve() function.

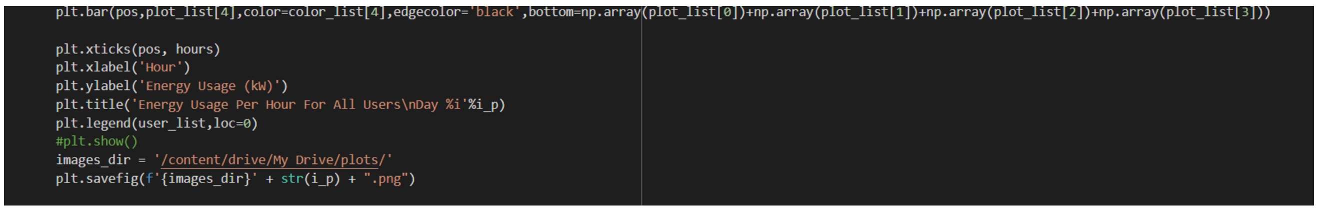


```
for i in range(len(x_test)): #iterate through test data
    if y_test[i] == 1: # if the predicted label is abnormal

      model = model_maker(user_tasks,task_info) #pulp lp model generation

      model.solve()

      plot(model,i+1)
```

Lastly, I used a plot function to plot the graphs of the scheduled task solutions for each predicted abnormal curve. The plot firstly function will match the user and the operation hours of that user's tasks. Then it will sum up each matched LP variables values using the .value() function. Lastly, it will plot the schedules in a bar plot using matplotlib.pyplot.



```
def plot(model_p,i_p):

  hours = [str(x) for x in range(0, 24)]
  pos = np.arange(len(hours))
  user_list = ['user1', 'user2', 'user3', 'user4', 'user5']
  color_list = ['red','blue','yellow','green','pink']
  plot_list = []
  #Create lists to plot usage
  for user in user_list:
      tmp3 = []
      for hour in hours:
          tmp4 = []
          task_count = 0
          for var in model_p.variables():
              if user == var.name.split('_')[0] and str(hour) == var.name.split('_')[2]: # if username and hour matches
                  task_count += 1
                  tmp4.append(var.value())
          tmp3.append(sum(tmp4))
      plot_list.append(tmp3)

  plt.bar(pos,plot_list[0],color=color_list[0],edgecolor='black',bottom=0)
  plt.bar(pos,plot_list[1],color=color_list[1],edgecolor='black',bottom=np.array(plot_list[0]))
  plt.bar(pos,plot_list[2],color=color_list[2],edgecolor='black',bottom=np.array(plot_list[0])+np.array(plot_list[1]))
  plt.bar(pos,plot_list[3],color=color_list[3],edgecolor='black',bottom=np.array(plot_list[0])+np.array(plot_list[1])+np.array(plot_list[2]))
  plt.bar(pos,plot_list[4],color=color_list[4],edgecolor='black',bottom=np.array(plot_list[0])+np.array(plot_list[1])+np.array(plot_list[2])+np.array(plot_list[3]))

  plt.xticks(pos, hours)
  plt.xlabel('Hour')
  plt.ylabel('Energy Usage (kW)')
  plt.title('Energy Usage Per Hour For All Users\nDay %i'%i_p)
  plt.legend(user_list,loc=0)
  #plt.show()
```



```
  plt.bar(pos,plot_list[4],color=color_list[4],edgecolor='black',bottom=np.array(plot_list[0])+np.array(plot_list[1])+np.array(plot_list[2])+np.array(plot_list[3]))

  plt.xticks(pos, hours)
  plt.xlabel('Hour')
  plt.ylabel('Energy Usage (kW)')
  plt.title('Energy Usage Per Hour For All Users\nDay %i'%i_p)
  plt.legend(user_list,loc=0)
  #plt.show()
  images_dir = '/content/drive/My Drive/plots/'
  plt.savefig(f'{images_dir}' + str(i_p) + ".png")
```

Plots

See the plots folder in my submission zip file. I did not include them here since it would make the pdf extremely long.