DETECTION OF MANIPULATED PRICING IN SMART ENERGY CPS SCHEDULING

INTRODUCTION

According to the problem statement of the coursework, we are provided with the data of 5 users who own 10 smart home appliances each. This data indicates that each user is given 10 tasks to complete with the smart appliances for which proper scheduling is necessary. We are also provided with 10000 predictive guideline price curves as training data that gives the pricing details for each hour of the day and the corresponding label indicating whether the pricing curve is normal or abnormal. By using this data, a suitable model needs to be designed for classifying the data to be scheduled into normal or abnormal, which is done to effectively predict a pricing attack. A set of 100 pricing curves is provided to be used as testing data for which labelling (normal or abnormal) must be determined using the model designed previously. For the price curves that are classified to be abnormal, we must minimize the cost by developing a Linear Programming energy solution based on the abnormal predictive guideline price curve and plot the obtained scheduling results displaying the hourly energy usage of the 5 users. The minimized schedule obtained as a result of our solution will be the corresponding normal scheduling for each abnormal price curve. GitHub repository at https://github.com/777jomin/Detection-of-Manipulated-Pricing-in-Smart-Energy-CPS-Scheduling is used to maintain the code for this coursework.

TO RUN THE PROGRAM

1. Clone the git repository contents to a single folder.
2. This can be done by using the command "git clone https://github.com/777jomin/Detection-of-Manipulated-Pricing-in-Smart-Energy-CPS-Scheduling".
3. The contents of the repository Detection-of-Manipulated-Pricing-in-Smart-Energy-CPS-Scheduling will now be available on your local machine.
4. Execute the 'run.sh' file to perform complete execution of code and obtain the bar chart results in '/Plots', the Testing results in TestingResults.txt and prediction results in PredictionsOnly.txt.

Please note:

If you get an error saying "./run.sh: line 2: $'\r': command not found" or "command not found" or ": not found"
please run the following command before executing 'run.sh'

```
sed -i -e 's/\r$//' run.sh
```

SELECTION OF CLASSIFICATION MODEL

We need to make use of a suitable model that is trained based on the provided training pricing curves to classify the testing data into normal or abnormal categories. To achieve this objective, different classification models can be employed and here we have considered three models to try which are Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), and K-Nearest Neighbors

(KNN). We chose Linear Discriminant Analysis from the tried three models as it gave the highest accuracy along with consistent and stable results.

LINEAR DISCRIMINANT ANALYSIS (LDA)

Linear Discriminant Analysis is a linear model commonly used for classification and dimensional reduction. The Bayes Theorem is used in Linear Discriminant Analysis to estimate the likelihood of a data point being in each accessible class label. The winner and output will be the class label with the highest probability.

Using the likelihood of each class and the probability of the data belonging to each class, Bayes' Theorem may be used to estimate the probability of the output class (k) given the input (x):

$$P(Y=x|X=x) = (PIk * fk(x)) / sum(PIl * fl(x))$$

Here PIk indicates the base probability of each class k which can be observed in the training data, for example, 0.2 indicates an 80-20 split in a two-class problem. This is referred to as prior probability in Bayes’ Theorem.

$$PIk = nk/n$$

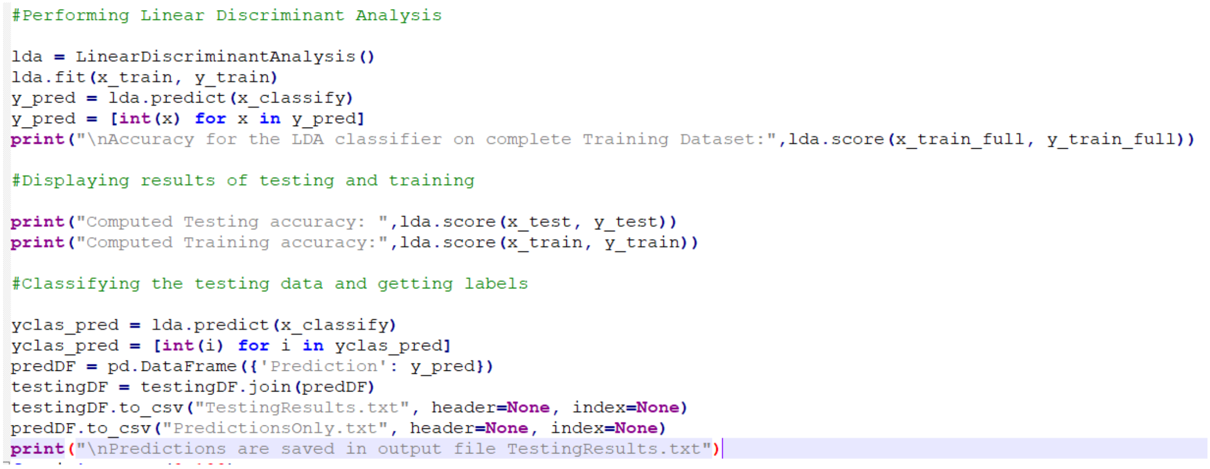
The function f(x) denotes the estimated probability of x belonging to a class and uses the Gaussian distribution function. Including the Gaussian element in the above equation and after performing required simplifications we obtain the equation shown below. The equation is called the discriminate function

$$Dk(x) = x * (muk/sigma^2) – (muk^2/(2*sigma^2)) + ln(PIk)$$

Here Dk(x) is the discriminate function for a class k, where the input is x. The other components of the equation muk, sigma^2 and PIk are estimated from the data we provide. [1]

LDA IMPLEMENTATION

We implement the LDA algorithm by making use of the existing model from the sklearn python library. The following code helps us achieve the implementation of LDA.

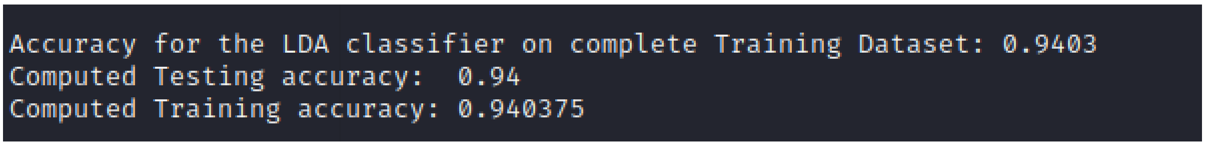


```
#Performing Linear Discriminant Analysis

lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
y_pred = lda.predict(x_classify)
y_pred = [int(x) for x in y_pred]
print("\nAccuracy for the LDA classifier on complete Training Dataset:",lda.score(x_train_full, y_train_full))

#Displaying results of testing and training

print("Computed Testing accuracy: ",lda.score(x_test, y_test))
print("Computed Training accuracy:",lda.score(x_train, y_train))

#Classifying the testing data and getting labels

yclas_pred = lda.predict(x_classify)
yclas_pred = [int(i) for i in yclas_pred]
predDF = pd.DataFrame({'Prediction': y_pred})
testingDF = testingDF.join(predDF)
testingDF.to_csv("TestingResults.txt", header=None, index=None)
predDF.to_csv("PredictionsOnly.txt", header=None, index=None)
print("\nPredictions are saved in output file TestingResults.txt")
```

For the local testing phase, we split the 10000 records from training data into 80% of training data and 20% of testing data. This testing data can be used to verify the accuracy of the prediction by LDA using the training data.

The LDA model is created with the sklearn library's LinearDiscrimnantAnalysis() function. The LDA object's fit function is used to train the model with the labelled data that is provided. After the model has been trained, it may be used to predict the labels in the testing data. The predict function of the LDA object, which is contained in y pred, is used to do this. This is what the previously separated local testing results predicted. The score function may be used to determine the accuracy of these outcomes, and the associated training and testing accuracy are shown in the image above. The provided unlabeled testing data is now utilised to forecast abnormal and normal instances. These projected outcomes are saved in /TestingResults.txt, while the predictions are saved in /PredictionsOnly.txt.

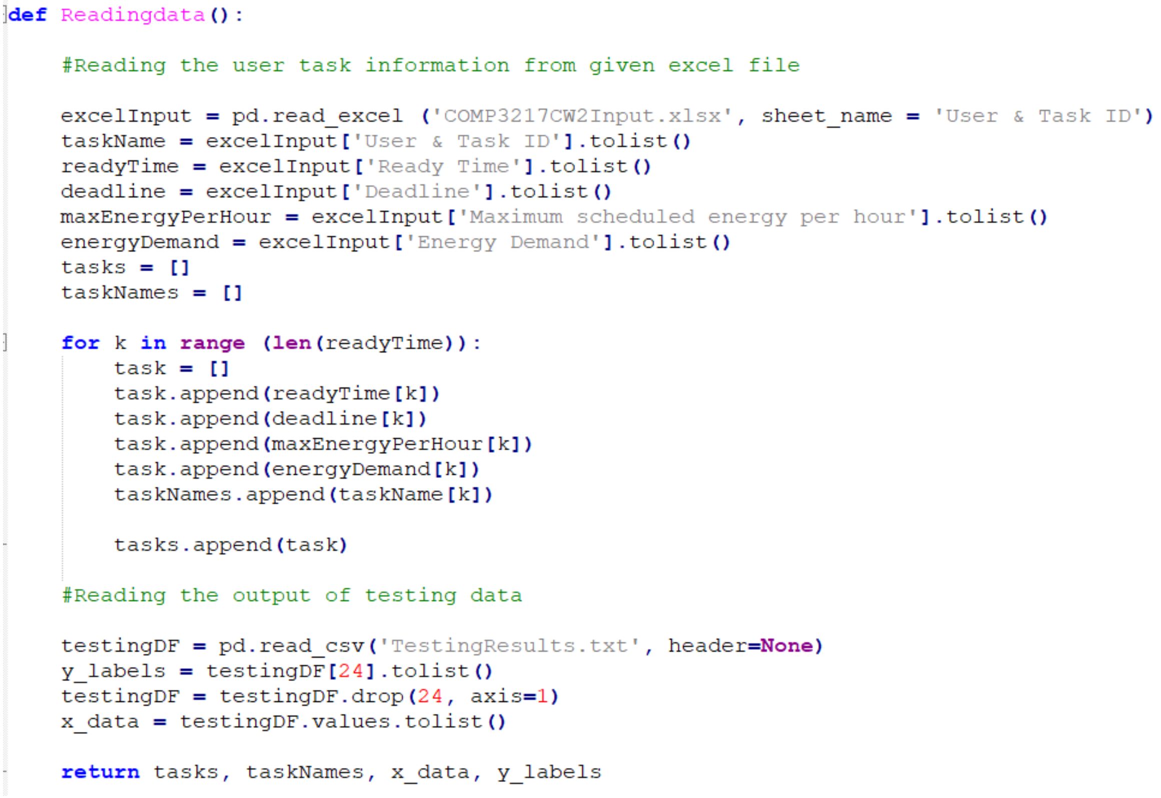
The computed training accuracy and testing accuracy using the LDA model is shown below:



```
Accuracy for the LDA classifier on complete Training Dataset: 0.9403
Computed Testing accuracy:  0.94
Computed Training accuracy: 0.940375
```

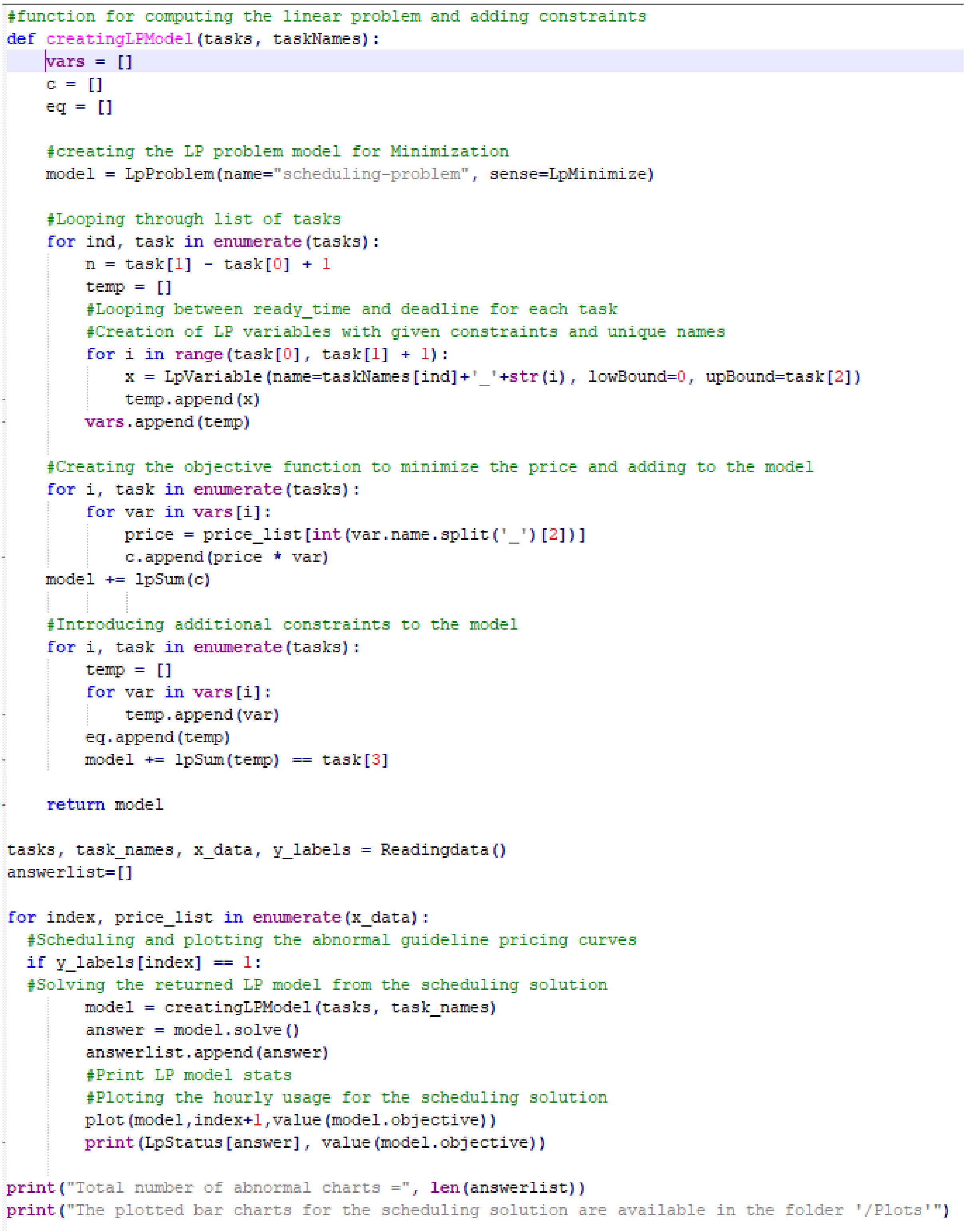
LINEAR PROGRAMMING AND SCHEDULING

For each testing data price curve that was predicted to be abnormal, we need to do the scheduling using a linear programming solution. The data obtained from the excel file COMP3217CW2input.xlsx shows that each of the 5 users is assigned 10 tasks which need to be completed using the 10 smart home appliances they own. The User and Task ID sheet in the excel file provides user data attributes such as user and task id, start time, deadline, max energy per hour, and energy demand which are extracted to be used for the scheduling algorithm. The code associated with this is the Readingdata() function as shown below.



```
def Readingdata():

    #Reading the user task information from given excel file

    excelInput = pd.read_excel ('COMP3217CW2Input.xlsx', sheet_name = 'User & Task ID')
    taskName = excelInput['User & Task ID'].tolist()
    readyTime = excelInput['Ready Time'].tolist()
    deadline = excelInput['Deadline'].tolist()
    maxEnergyPerHour = excelInput['Maximum scheduled energy per hour'].tolist()
    energyDemand = excelInput['Energy Demand'].tolist()
    tasks = []
    taskNames = []

    for k in range (len(readyTime)):
        task = []
        task.append(readyTime[k])
        task.append(deadline[k])
        task.append(maxEnergyPerHour[k])
        task.append(energyDemand[k])
        taskNames.append(taskName[k])

        tasks.append(task)

    #Reading the output of testing data

    testingDF = pd.read_csv('TestingResults.txt', header=None)
    y_labels = testingDF[24].tolist()
    testingDF = testingDF.drop(24, axis=1)
    x_data = testingDF.values.tolist()

    return tasks, taskNames, x_data, y_labels
```

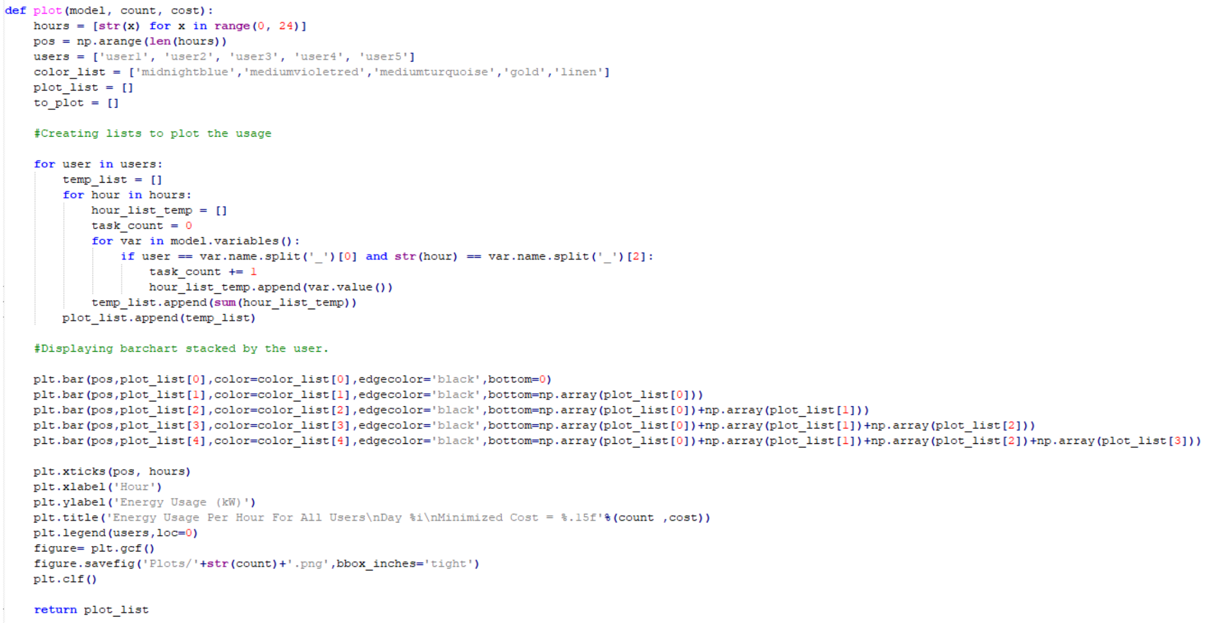
The relevant user tasks and the abnormal guideline price of that hour should be utilised to develop a minimization linear programming objective function for all abnormal curves. The conditions under which the function should operate must be specified, and the corresponding code using the creatingLPModel() function is displayed below.



```
#function for computing the linear problem and adding constraints
def creatingLPModel(tasks, taskNames):
    vars = []
    c = []
    eq = []

    #creating the LP problem model for Minimization
    model = LpProblem(name="scheduling-problem", sense=LpMinimize)

    #Looping through list of tasks
    for ind, task in enumerate(tasks):
        n = task[1] - task[0] + 1
        temp = []
        #Looping between ready_time and deadline for each task
        #Creation of LP variables with given constraints and unique names
        for i in range(task[0], task[1] + 1):
            x = LpVariable(name=taskNames[ind]+'_'+str(i), lowBound=0, upBound=task[2])
            temp.append(x)
        vars.append(temp)

    #Creating the objective function to minimize the price and adding to the model
    for i, task in enumerate(tasks):
        for var in vars[i]:
            price = price_list[int(var.name.split('_')[2])]
            c.append(price * var)
    model += lpSum(c)

    #Introducing additional constraints to the model
    for i, task in enumerate(tasks):
        temp = []
        for var in vars[i]:
            temp.append(var)
        eq.append(temp)
        model += lpSum(temp) == task[3]

    return model

tasks, task_names, x_data, y_labels = Readingdata()
answerlist=[]

for index, price_list in enumerate(x_data):
  #Scheduling and plotting the abnormal guideline pricing curves
  if y_labels[index] == 1:
  #Solving the returned LP model from the scheduling solution
        model = creatingLPModel(tasks, task_names)
        answer = model.solve()
        answerlist.append(answer)
        #Print LP model stats
        #Ploting the hourly usage for the scheduling solution
        plot(model,index+1,value(model.objective))
        print(LpStatus[answer], value(model.objective))

print("Total number of abnormal charts =", len(answerlist))
print("The plotted bar charts for the scheduling solution are available in the folder '/Plots'")
```

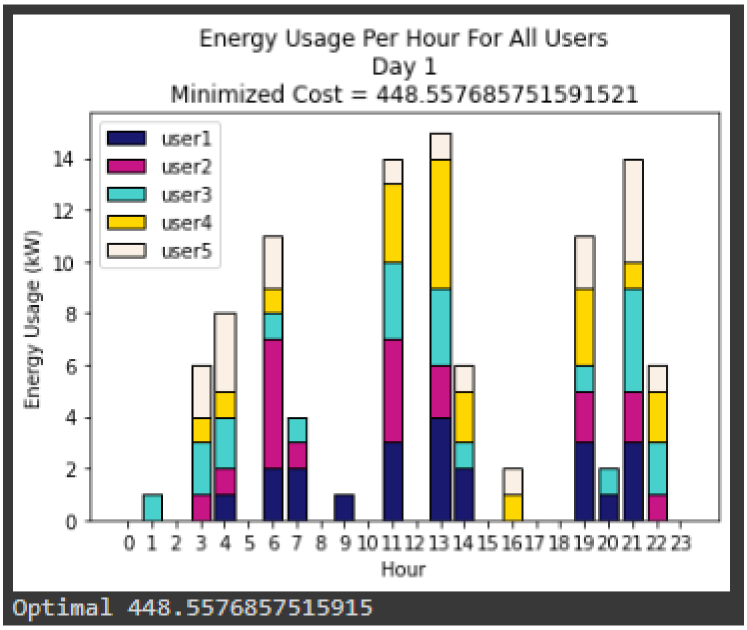
PLOTTING BAR CHARTS

The bar charts displaying the hourly energy usage of the 5 users need to be plotted based on the obtained linear programming solution. This will enable us to understand the corrected schedule of usage based on the abnormal price curve. The bar charts have hours and energy usage in the x and y axes respectively. The charts depict the day in which abnormality was observed and shows the minimized cost of that day according to the linear programming solution that was obtained. The bar charts are plotted using the plot function in the code as shown below

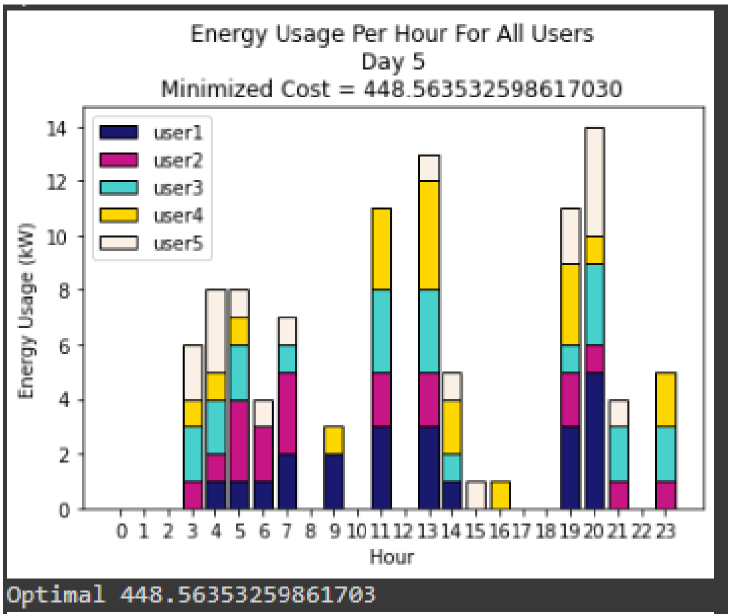


```
def plot(model, count, cost):
    hours = [str(x) for x in range(0, 24)]
    pos = np.arange(len(hours))
    users = ['user1', 'user2', 'user3', 'user4', 'user5']
    color_list = ['midnightblue','mediumvioletred','mediumturquoise','gold','linen']
    plot_list = []
    to_plot = []

    #Creating lists to plot the usage

    for user in users:
        temp_list = []
        for hour in hours:
            hour_list_temp = []
            task_count = 0
            for var in model.variables():
                if user == var.name.split('_')[0] and str(hour) == var.name.split('_')[2]:
                    task_count += 1
                    hour_list_temp.append(var.value())
            temp_list.append(sum(hour_list_temp))
        plot_list.append(temp_list)

    #Displaying barchart stacked by the user.

    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\nMinimized Cost = %.15f'%(count ,cost))
    plt.legend(users,loc=0)
    figure= plt.gcf()
    figure.savefig('Plots/'+str(count)+'.png',bbox_inches='tight')
    plt.clf()

    return plot_list
```

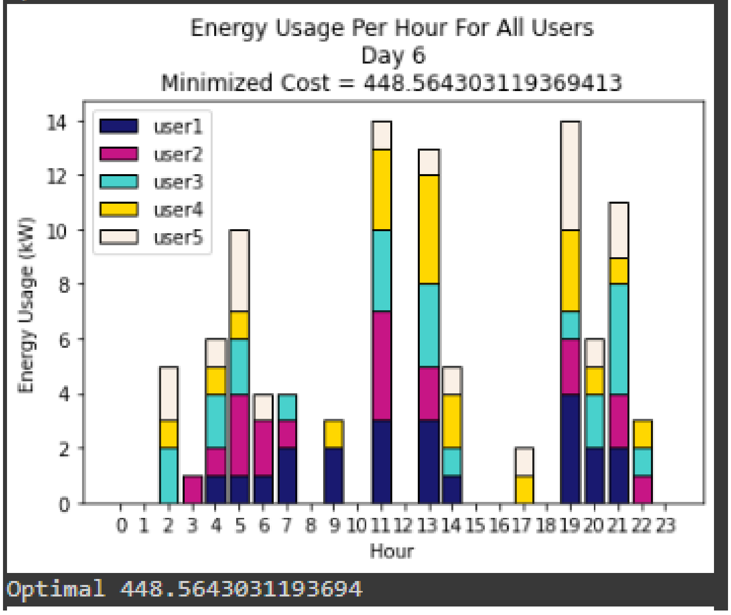
The following are some of the bar charts that were plotted, and the complete set of bar charts is saved in the /Plots folder.



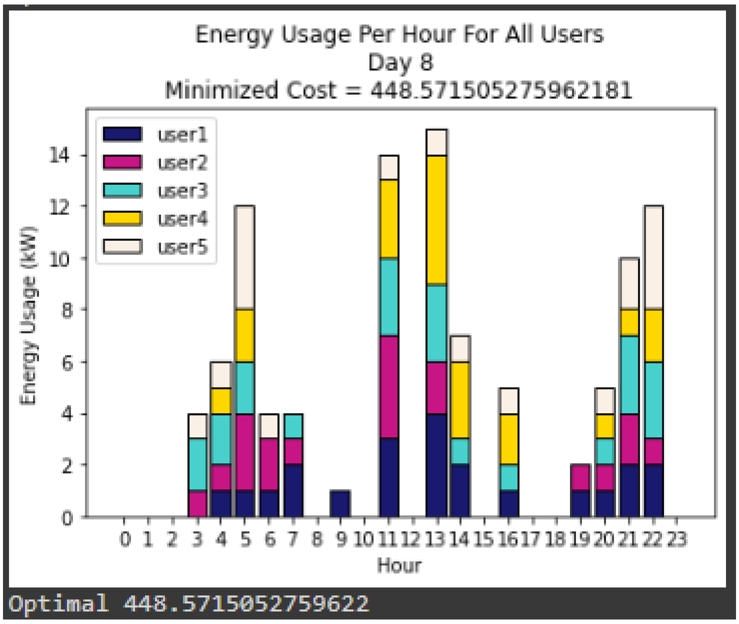
```
Optimal 448.5576857515915
```



```
Optimal 448.56353259861703
```

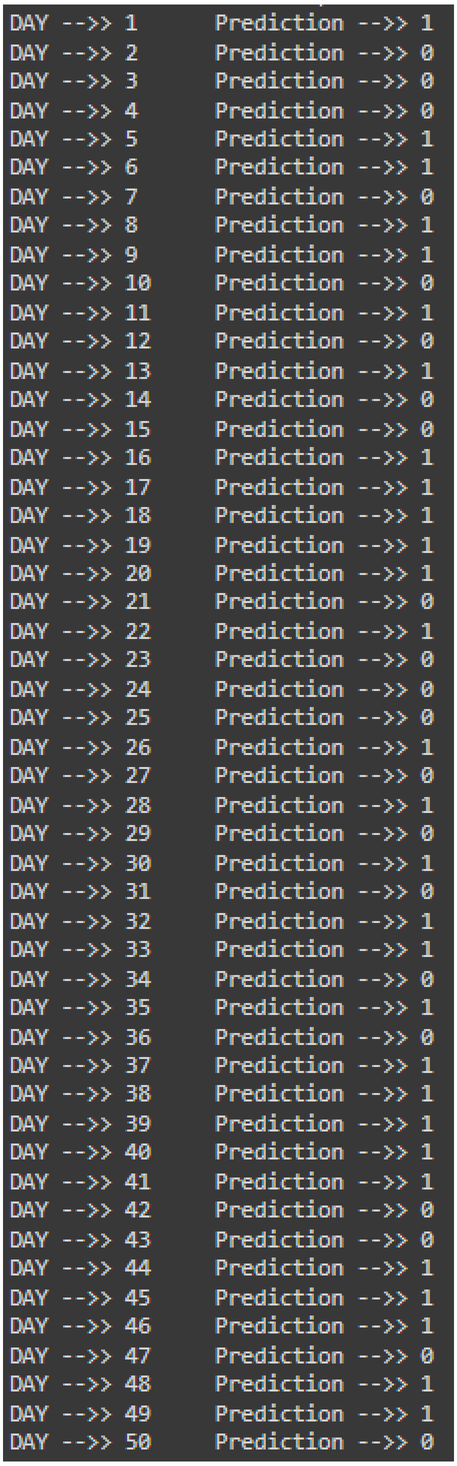


```
Optimal 448.5643031193694
```

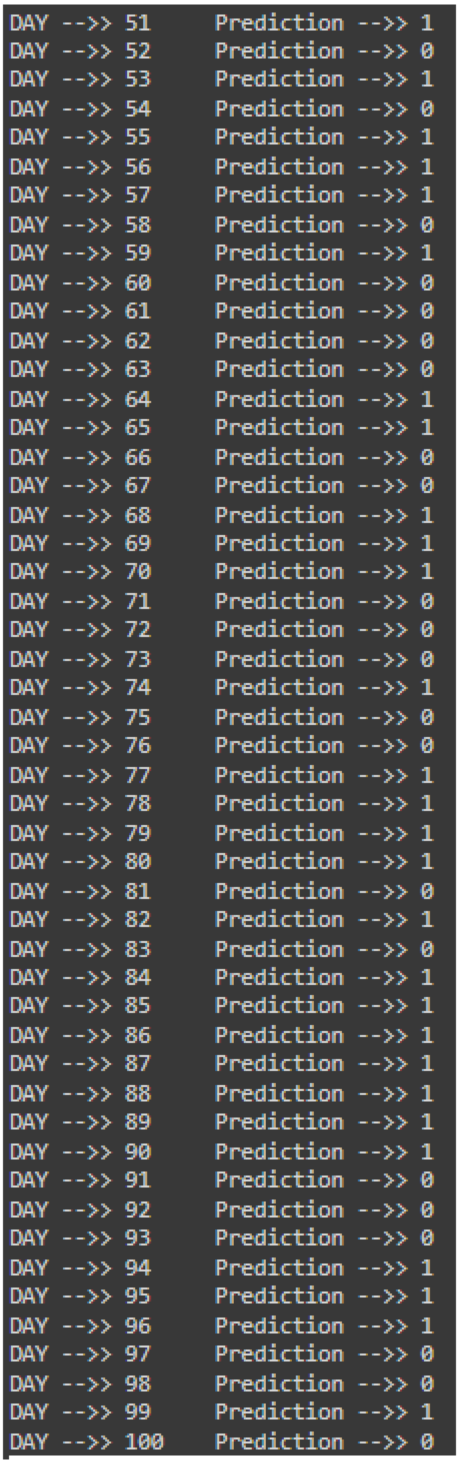


```
Optimal 448.5715052759622
```

COMPUTED LABEL PREDICTION



```
DAY -->> 1      Prediction -->> 1
DAY -->> 2      Prediction -->> 0
DAY -->> 3      Prediction -->> 0
DAY -->> 4      Prediction -->> 0
DAY -->> 5      Prediction -->> 1
DAY -->> 6      Prediction -->> 1
DAY -->> 7      Prediction -->> 0
DAY -->> 8      Prediction -->> 1
DAY -->> 9      Prediction -->> 1
DAY -->> 10     Prediction -->> 0
DAY -->> 11     Prediction -->> 1
DAY -->> 12     Prediction -->> 0
DAY -->> 13     Prediction -->> 1
DAY -->> 14     Prediction -->> 0
DAY -->> 15     Prediction -->> 0
DAY -->> 16     Prediction -->> 1
DAY -->> 17     Prediction -->> 1
DAY -->> 18     Prediction -->> 1
DAY -->> 19     Prediction -->> 1
DAY -->> 20     Prediction -->> 1
DAY -->> 21     Prediction -->> 0
DAY -->> 22     Prediction -->> 1
DAY -->> 23     Prediction -->> 0
DAY -->> 24     Prediction -->> 0
DAY -->> 25     Prediction -->> 0
DAY -->> 26     Prediction -->> 1
DAY -->> 27     Prediction -->> 0
DAY -->> 28     Prediction -->> 1
DAY -->> 29     Prediction -->> 0
DAY -->> 30     Prediction -->> 1
DAY -->> 31     Prediction -->> 0
DAY -->> 32     Prediction -->> 1
DAY -->> 33     Prediction -->> 1
DAY -->> 34     Prediction -->> 0
DAY -->> 35     Prediction -->> 1
DAY -->> 36     Prediction -->> 0
DAY -->> 37     Prediction -->> 1
DAY -->> 38     Prediction -->> 1
DAY -->> 39     Prediction -->> 1
DAY -->> 40     Prediction -->> 1
DAY -->> 41     Prediction -->> 1
DAY -->> 42     Prediction -->> 0
DAY -->> 43     Prediction -->> 0
DAY -->> 44     Prediction -->> 1
DAY -->> 45     Prediction -->> 1
DAY -->> 46     Prediction -->> 1
DAY -->> 47     Prediction -->> 0
DAY -->> 48     Prediction -->> 1
DAY -->> 49     Prediction -->> 1
DAY -->> 50     Prediction -->> 0
```



```
DAY -->> 51     Prediction -->> 1
DAY -->> 52     Prediction -->> 0
DAY -->> 53     Prediction -->> 1
DAY -->> 54     Prediction -->> 0
DAY -->> 55     Prediction -->> 1
DAY -->> 56     Prediction -->> 1
DAY -->> 57     Prediction -->> 1
DAY -->> 58     Prediction -->> 0
DAY -->> 59     Prediction -->> 1
DAY -->> 60     Prediction -->> 0
DAY -->> 61     Prediction -->> 0
DAY -->> 62     Prediction -->> 0
DAY -->> 63     Prediction -->> 0
DAY -->> 64     Prediction -->> 1
DAY -->> 65     Prediction -->> 1
DAY -->> 66     Prediction -->> 0
DAY -->> 67     Prediction -->> 0
DAY -->> 68     Prediction -->> 1
DAY -->> 69     Prediction -->> 1
DAY -->> 70     Prediction -->> 1
DAY -->> 71     Prediction -->> 0
DAY -->> 72     Prediction -->> 0
DAY -->> 73     Prediction -->> 0
DAY -->> 74     Prediction -->> 1
DAY -->> 75     Prediction -->> 0
DAY -->> 76     Prediction -->> 0
DAY -->> 77     Prediction -->> 1
DAY -->> 78     Prediction -->> 1
DAY -->> 79     Prediction -->> 1
DAY -->> 80     Prediction -->> 1
DAY -->> 81     Prediction -->> 0
DAY -->> 82     Prediction -->> 1
DAY -->> 83     Prediction -->> 0
DAY -->> 84     Prediction -->> 1
DAY -->> 85     Prediction -->> 1
DAY -->> 86     Prediction -->> 1
DAY -->> 87     Prediction -->> 1
DAY -->> 88     Prediction -->> 1
DAY -->> 89     Prediction -->> 1
DAY -->> 90     Prediction -->> 1
DAY -->> 91     Prediction -->> 0
DAY -->> 92     Prediction -->> 0
DAY -->> 93     Prediction -->> 0
DAY -->> 94     Prediction -->> 1
DAY -->> 95     Prediction -->> 1
DAY -->> 96     Prediction -->> 1
DAY -->> 97     Prediction -->> 0
DAY -->> 98     Prediction -->> 0
DAY -->> 99     Prediction -->> 1
DAY -->> 100    Prediction -->> 0
```

REFERENCES

[1]J. Brownlee, "Linear Discriminant Analysis for Machine Learning", *Machine Learning Mastery*, 2022. [Online]. Available: https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/.