COMP3217 Option A - Detection of Manipulated Pricing in Smart Energy CPS Scheduling Report

Student ID: 33376956

GITHUB:

Code and plots are uploaded in the GITHUB repository. https://github.com/mkumar-rk05/COMP3217.

Problem

The problem is based on a group of five people, each of whom owns 10 smart home appliances. Over the course of a 24-hour period, users' energy consumption will vary, and they will typically peak at the same time as other users. To reduce peak usage and balance the energy load, energy providers hike the cost of the energy used during peak hours. We need an intelligent scheduler is required to compute the scheduling solution which reduces the customer's consumption cost. Linear programming is used to determine what time appliances should be turned on and how much time they should be kept on running.

Some of the price curve rules for energy expenses over a 24-hour period are abnormal. Need to find this and based on that smart scheduler can schedule the appliances accordingly.

Predicting Model

We need to develop a model that is trained based on the given training pricing curves to classify the testing data into normal and abnormal. Various classification models are available that can be utilised for this purpose. Multiple models were tested in our case, and the one with the highest accuracy was chosen.

- Linear Discriminant Analysis (LDA),
- K-Nearest Neighbors (KNN),
- Support Vector Machines (SVM)

models were used. LDA produced the most accurate, steady, and consistent results of the models listed above.

Linear Discriminant Analysis (LDA):

Linear Discriminant Analysis, or LDA for short, is a machine learning technique for classification.

It operates by calculating summary statistics such as the mean and standard deviation for the input features by the class label. The model learned from the training data is represented by these statistics. In practice, matrix decomposition is utilised to efficiently calculate the relevant quantities using linear algebra procedures.

Based on the values of each input attribute, predictions are made by calculating the probability that a new example corresponds to each class label. The example is then assigned to the class with the highest probability. As a result, LDA can be thought of as a straightforward application of the Bayes Theorem to classification.

The input variables in LDA are assumed to be numeric, regularly distributed, and have the same variance (spread). If this is not the case, the data should be transformed into a Gaussian distribution and the data should be standardised or normalised before modelling.

The LDA model is multi-class by definition. This means it supports two-class classification problems and can be extended to more than two classes (multi-class classification) without needing to be modified or enhanced.

Like logistic regression, it's a linear classification algorithm. This means that lines or hyperplanes separate classes in the feature space. Extensions of the procedure, such as Quadratic Discriminant Analysis (QDA), which enables curved shapes in the decision boundary, can be employed.

In classification we consider K classes {1, 2, 3 , ..., K} and an input vector X. We classify X as belonging to the class i that maximizes P(Y=i | X).
LDA classification relies on Bayes theorem which states:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

The likelihood of X belonging to each class is related to the probability of input taking on the value of X in each class, according to this formula. We'll pretend we just have one input variable, in which case X is merely an integer.

P(X|Y) can be viewed as a probability density function like the one below, with K plots, one for each class, whose height indicates the probability of input taking on a value of X. There is a high possibility of getting a value X if the graph's height is large, and a low probability if the graph's height is small.

We get the normal distribution when we plug it into Bayes theorem.

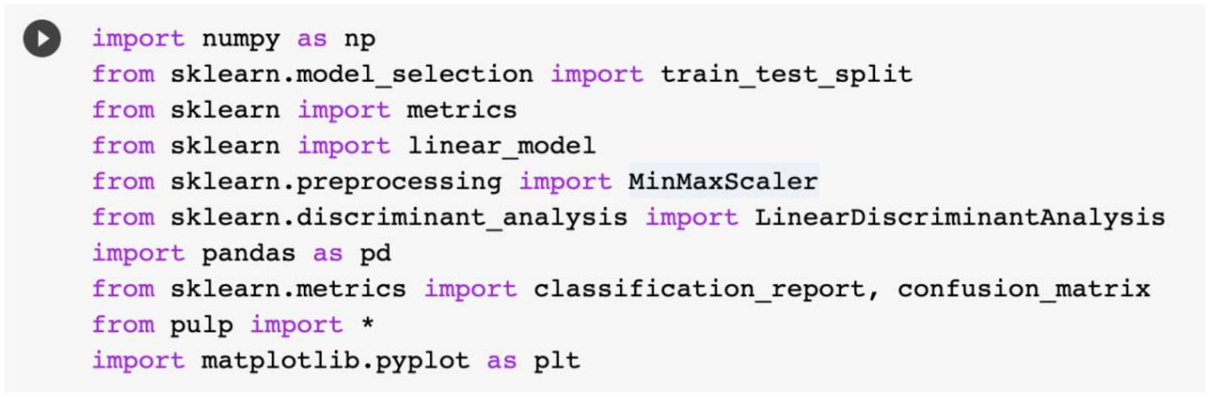
$$P(Y = i|X = a) = \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{X-\mu_i}{\sigma_i})^2} P(Y = i)}{P(X = a)}$$

$$\delta(X) = \frac{\mu_i X}{\sigma^2} - \frac{1}{2}\frac{{\mu_i}^2}{\sigma^2} + \log\big(P(Y = i)\big)$$

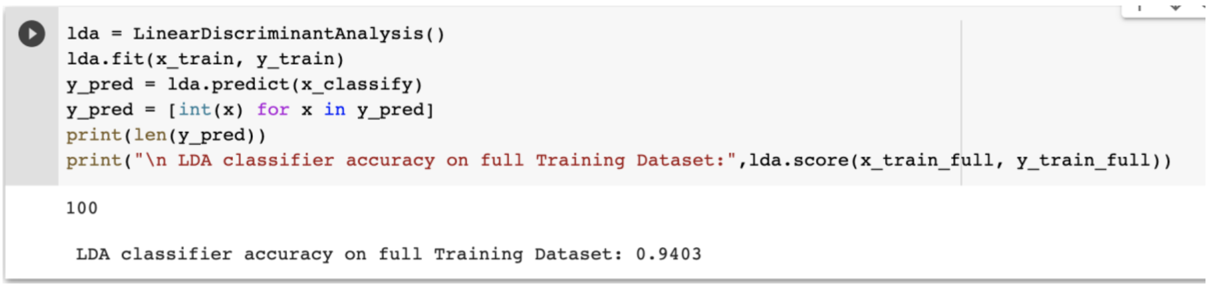
We can categorise X into the class with the highest discriminant function value. The name LDA comes from the fact that the formula is linear in X.

LDA Implementation

LDA algorithm is implemented using SKLEARN python library. Below is the source code for implementation of SKLEARN library. 10000 records are split into two like 80% is for training data and 20% is for testing data.



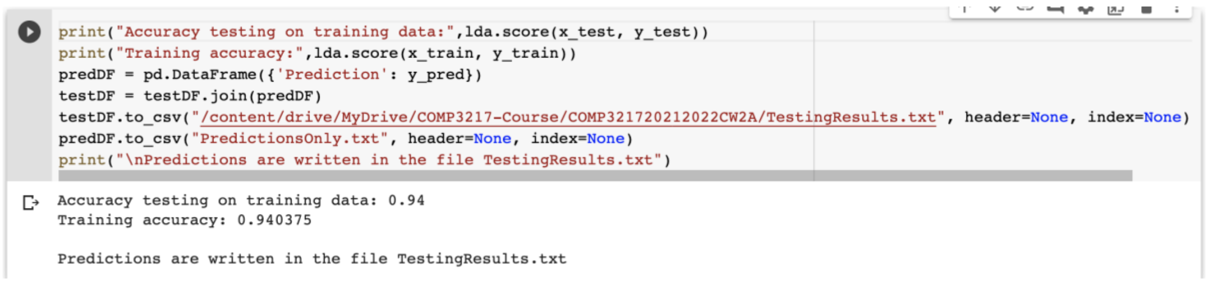
```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import linear_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix
from pulp import *
import matplotlib.pyplot as plt
```



```python
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
y_pred = lda.predict(x_classify)
y_pred = [int(x) for x in y_pred]
print(len(y_pred))
print("\n LDA classifier accuracy on full Training Dataset:",lda.score(x_train_full, y_train_full))
```

```
100

 LDA classifier accuracy on full Training Dataset: 0.9403
```



```python
print("Accuracy testing on training data:",lda.score(x_test, y_test))
print("Training accuracy:",lda.score(x_train, y_train))
predDF = pd.DataFrame({'Prediction': y_pred})
testDF = testDF.join(predDF)
testDF.to_csv("/content/drive/MyDrive/COMP3217-Course/COMP321720212022CW2A/TestingResults.txt", header=None, index=None)
predDF.to_csv("PredictionsOnly.txt", header=None, index=None)
print("\nPredictions are written in the file TestingResults.txt")
```

```
Accuracy testing on training data: 0.94
Training accuracy: 0.940375

Predictions are written in the file TestingResults.txt
```

The LDA model is created with the sklearn library's LinearDiscrimnantAnalysis() function. The model is trained with the labelled data provided by the fit function of the LDA object. 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 accomplish this. This is what the previously separated local testing results predicted. The score function can be used to determine the accuracy of these outcomes, and the associated training and testing accuracy is shown in the image above.

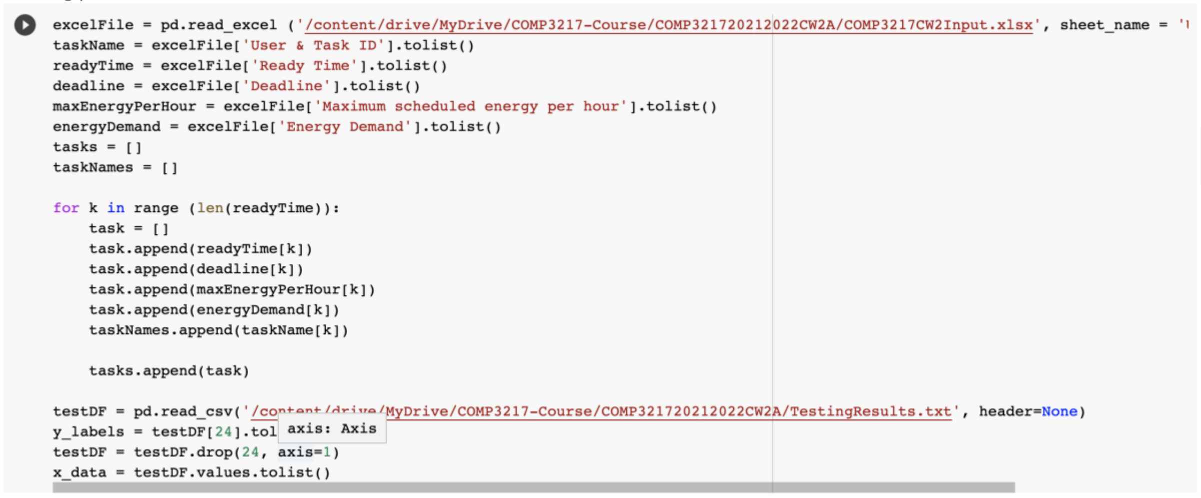
The provided testing data is now used to predict abnormal and normal cases. These projected outcomes are saved in TestingResults.txt, while the predictions are given below in a table.

Predictions for each day:

Day 1:	1	Day 26:	1	Day 51:	1	Day 76:	0
Day 2:	0	Day 27:	0	Day 52:	0	Day 77:	1
Day 3:	0	Day 28:	1	Day 53:	1	Day 78:	1
Day 4:	0	Day 29:	0	Day 54:	0	Day 79:	1

Day 5:	1	Day 30:	1	Day 55:	1	Day 80:	1
Day 6:	1	Day 31:	0	Day 56:	1	Day 81:	0
Day 7:	0	Day 32:	1	Day 57:	1	Day 82:	1
Day 8:	1	Day 33:	1	Day 58:	0	Day 83:	0
Day 9:	1	Day 34:	0	Day 59:	1	Day 84:	1
Day 10:	0	Day 35:	1	Day 60:	0	Day 85:	1
Day 11:	1	Day 36:	0	Day 61:	0	Day 86:	1
Day 12:	0	Day 37:	1	Day 62:	0	Day 87:	1
Day 13:	1	Day 38:	1	Day 63:	0	Day 88:	1
Day 14:	0	Day 39:	1	Day 64:	1	Day 89:	1
Day 15:	0	Day 40:	1	Day 65:	1	Day 90:	1
Day 16:	1	Day 41:	1	Day 66:	0	Day 91:	0
Day 17:	1	Day 42:	0	Day 67:	0	Day 92:	0
Day 18:	1	Day 43:	0	Day 68:	1	Day 93:	0
Day 19:	1	Day 44:	1	Day 69:	1	Day 94:	1
Day 20:	1	Day 45:	1	Day 70:	1	Day 95:	1
Day 21:	0	Day 46:	1	Day 71:	0	Day 96:	1
Day 22:	1	Day 47:	0	Day 72:	0	Day 97:	0
Day 23:	0	Day 48:	1	Day 73:	0	Day 98:	0
Day 24:	0	Day 49:	1	Day 74:	1	Day 99:	1
Day 25:	0	Day 50:	0	Day 75:	0	Day 100:	0

A linear programming method must be used to schedule all of the price curves that were projected to be anomalous. Each of the five users is assigned ten tasks that require them to operate the smart home appliances. The COMP3217CW2input.xlsx file contains this information. Data on User and Task ID, Start Time, Deadline, Maximum Energy Per Hour, and Energy Demand can all be derived from the "User & Task ID" sheet.



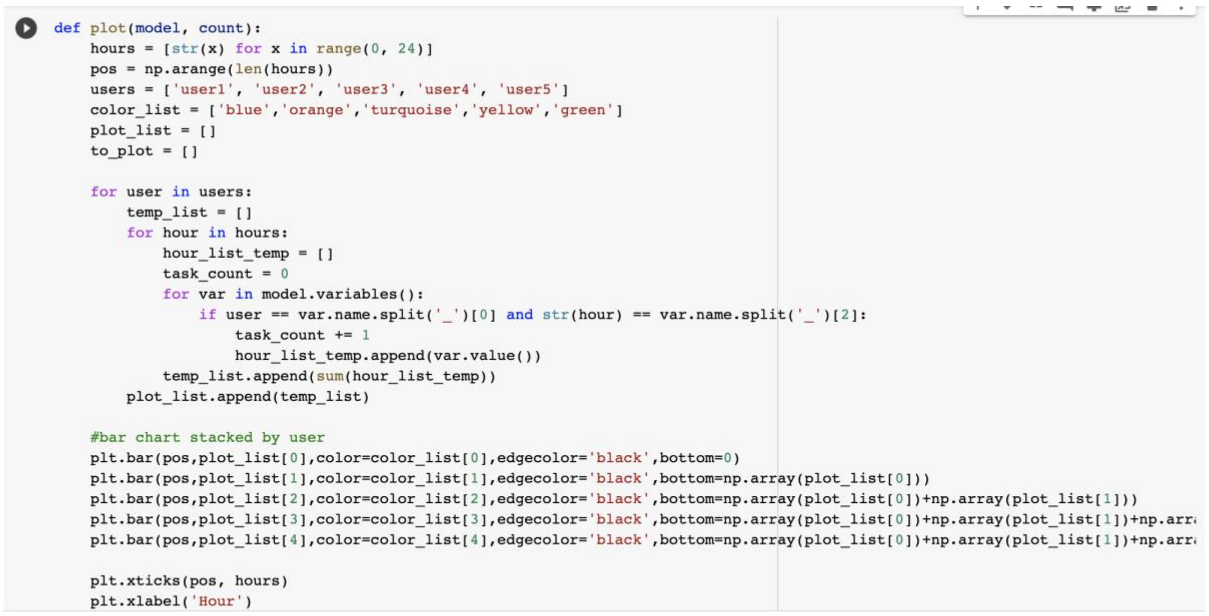
```
excelFile = pd.read_excel ('/content/drive/MyDrive/COMP3217-Course/COMP321720212022CW2A/COMP3217CW2Input.xlsx', sheet_name = '
taskName = excelFile['User & Task ID'].tolist()
readyTime = excelFile['Ready Time'].tolist()
deadline = excelFile['Deadline'].tolist()
maxEnergyPerHour = excelFile['Maximum scheduled energy per hour'].tolist()
energyDemand = excelFile['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)

testDF = pd.read_csv('/content/drive/MyDrive/COMP3217-Course/COMP321720212022CW2A/TestingResults.txt', header=None)
y_labels = testDF[24].tol axis: Axis
testDF = testDF.drop(24, axis=1)
x_data = testDF.values.tolist()
```

The relevant user duties and the abnormal guideline price for that hour should be utilised to generate a minimization linear programming objective function and the constraints on which the problem should work as follows for all abnormal curves.

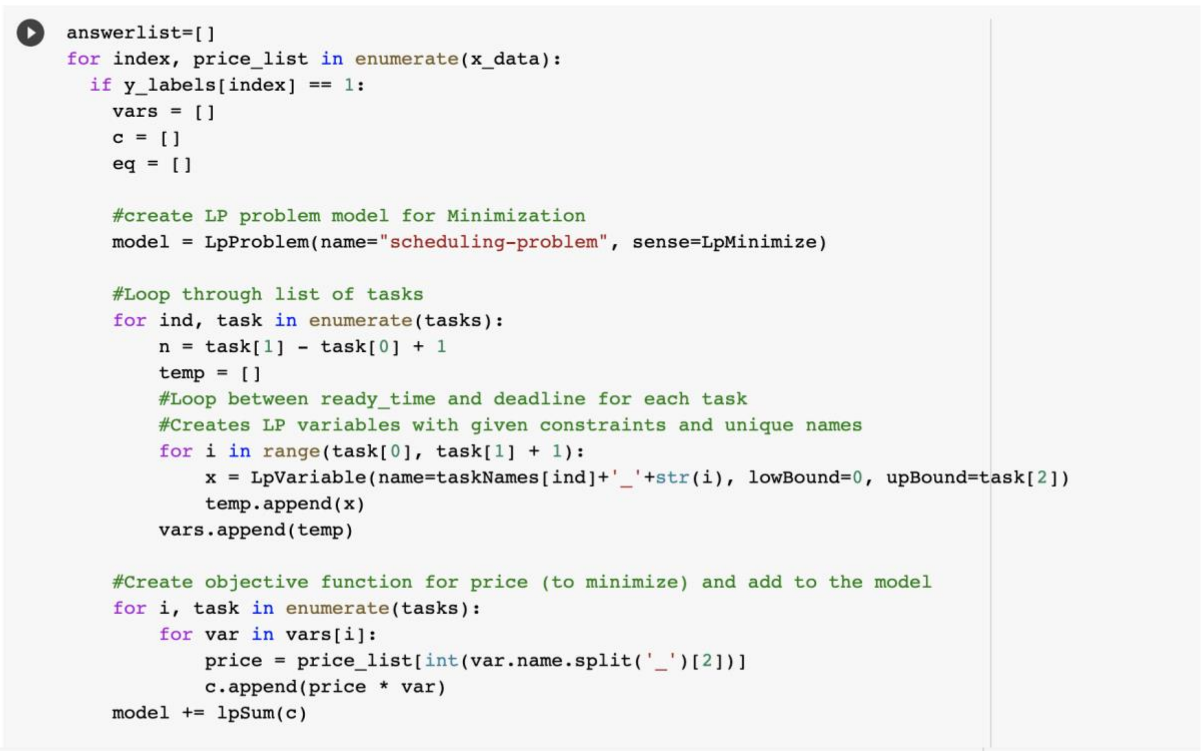


```python
def plot(model, count):
    hours = [str(x) for x in range(0, 24)]
    pos = np.arange(len(hours))
    users = ['user1', 'user2', 'user3', 'user4', 'user5']
    color_list = ['blue','orange','turquoise','yellow','green']
    plot_list = []
    to_plot = []

    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)

    #bar chart stacked by 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.arr
    plt.bar(pos,plot_list[4],color=color_list[4],edgecolor='black',bottom=np.array(plot_list[0])+np.array(plot_list[1])+np.arr

    plt.xticks(pos, hours)
    plt.xlabel('Hour')
```

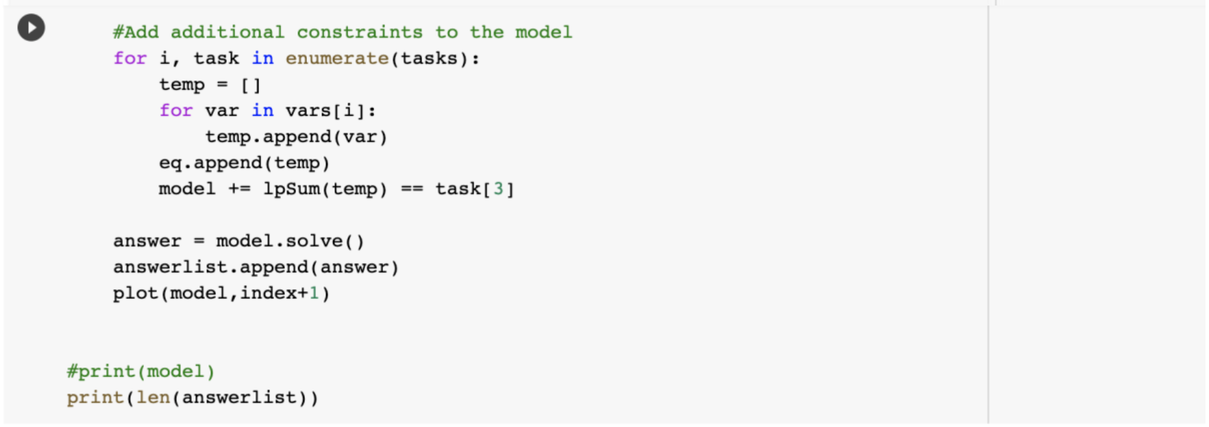


```python
answerlist=[]
for index, price_list in enumerate(x_data):
  if y_labels[index] == 1:
    vars = []
    c = []
    eq = []

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

    #Loop through list of tasks
    for ind, task in enumerate(tasks):
        n = task[1] - task[0] + 1
        temp = []
        #Loop between ready_time and deadline for each task
        #Creates 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)

    #Create objective function for price (to minimize) and add 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)
```

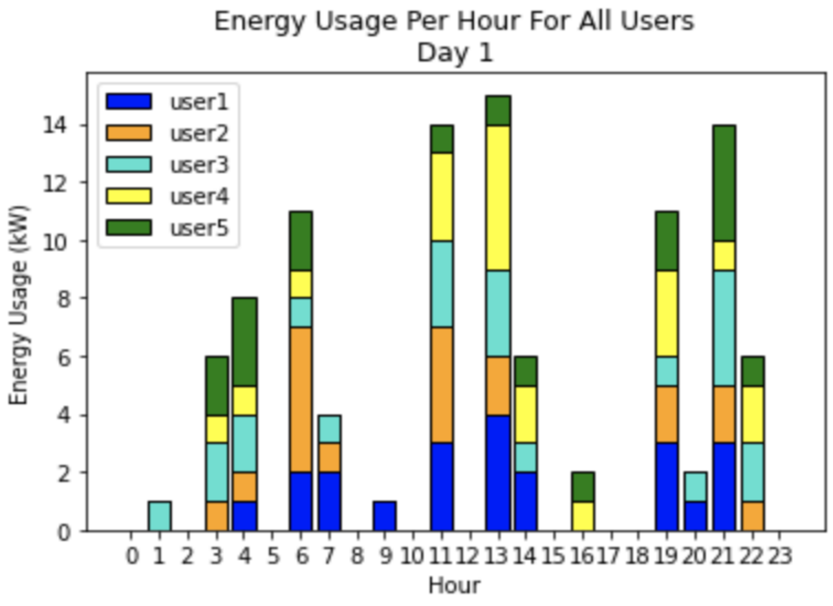


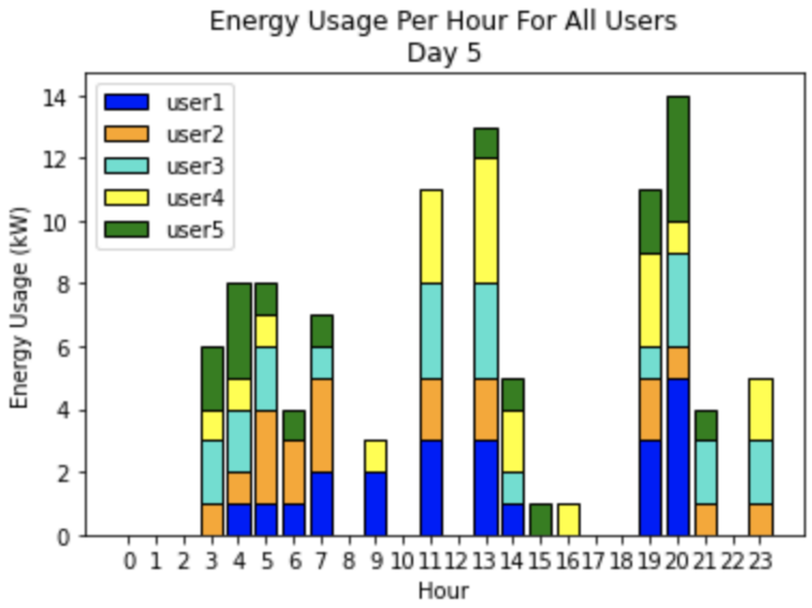
```python
    #Add additional constraints to the model
    for i, task in enumerate(tasks):
        temp = []
        for var in vars[i]:
            temp.append(var)
        eq.append(temp)
        model += lpSum(temp) == task[3]

    answer = model.solve()
    answerlist.append(answer)
    plot(model,index+1)


#print(model)
print(len(answerlist))
```

Bar charts can now be drawn using the solution to show the revised use schedule based on the anomalous pricing guideline. The x-axis of these bar charts is the hour, while the y-axis shows the amount of energy used. It displays the day on which the irregularity happened as well as the solution's lowest cost for that day. The following are some examples of charts.





The remaining plots are uploaded to the GITHUB repository inside the “plots” folder.

Reference:

[1] N. Chakrabarty, “Implementation of Gaussian Naive Bayes in Python from scratch,” Hackernoon, Jan. 23, 2019. https://hackernoon.com/implementation-of-gaussian-naive-bayes-in-python-from-scratch-c4ea64e3944d (accessed Apr. 28, 2022).

[2] C. Y. Huang, C. Y. Lai, and K. T. Cheng, “Fundamentals of algorithms,” Electronic Design Automation, pp. 173–234, Jan. 2009, doi: 10.1016/B978-0-12-374364-0.50011-4.

[3] https://machinelearningmastery.com/linear-discriminant-analysis-with-python/

[4] https://www.analyticsvidhya.com/blog/2021/08/a-brief-introduction-to-linear-discriminant-analysis/

[5] https://towardsdatascience.com/mathematical-insights-into-classification-using-linear-discriminant-analysis-9c822ad2fce2