1A: Automatic Alarm system

```
from gpio import *
from time import *
def main():
    pinMode(0,IN)
    while True:
        if digitalRead(0) == HIGH:
        print("Alarm Activated")
        digitalWrite(1, HIGH)
    else:
        print("Alarm DeActivated")
        digitalWrite(1, LOW)
    sleep(1)
if __name__ == "__main__":
    main()
```

1B: Timer based buzzer

```
from gpio import *
from time import *
def main():
    while True:
        print("Program Started")
        digitalWrite(0, HIGH)
        sleep(1)
        digitalWrite(0, LOW)
        sleep(1)
if __name__=="__main__":
    main()
```

1C: Sensor based Counting device

```
from gpio import *
from time import *
def main():
    count = 0
    pinMode(0,IN)
    while True:
        if digitalRead(0) == HIGH:
            print(count)
            sleep(1)
            count += 1
if __name__ == "__main__":
    main()
```

3: To send ticket before entering the bus.

```
from gpio import *
from time import *
from ioeclient import IoEClient
def main():
    pinMode(0,OUT)
    pinMode(1,IN)
    while True:
            customWrite(0,"Waiting")
            rfid = analogRead(A1)
            if (rfid==0):
                    customWrite(0,"Sucessful")

            delay(3000)

if __name__ == "__main__":
    main()
```

6: Raise an alarm whenever with going to rain outside based on the weather prediction data

```
from gpio import *
from time import *

def main():
    while True:
        val = digitalRead(0)
        print (val)
        if val >= 600:
        digitalWrite(1, HIGH)
        print("status : Carry your Umbrella")
    else:
        digitalWrite(1, LOW)
        print("status : no need to carry Umbrella")
    delay(500)

if __name__ =="__main__":
    main()
```

8: Monitoring water levels in tanks

```
from options import Options
from time import *
import math
from physical import *
from gpio import *
from environment import Environment
from ioeclient import IoEClient
from pyjs import *

def setup ():
    pinMode(0, OUTPUT)
    pinMode(1, OUTPUT)
def loop ():
    waterLevel = math.floor(js_map(analogRead(A0), 0, 1023, 0, 20) + 0.5)
    if waterLevel >= 5:
        digitalWrite(0, HIGH)
        digitalWrite(1, LOW)
    else:
        digitalWrite(0, LOW)
        digitalWrite(1, HIGH)
    delay(1000)
if __name__ == "__main__":
    setup()
    while True:
        loop()
        idle()
```

pyjs.py

```
class JsObject(dict):
  def __init__(self, d):
    for k in d.keys():
      setattr(self, k, d[k])

def js_map(x, inMin, inMax, outMin, outMax):
  return (x - inMin) * (outMax - outMin) / (inMax - inMin) + outMin
```

10: Motion detection.

```
from gpio import *
from time import *
def main():
        while True:
                value = digitalRead(0)
                if value >=1:
                        customWrite(0, "2")
                        print("Fan ON")
                else:
                        customWrite(0, "0")
                        print("Fan OFF")
                delay(500)
if __name__ == "__main__":
        main()
```

MACHINE LEARNING

1: Linear Regression (Diabetes Dataset)

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
   % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
   % r2_score(diabetes_y_test, diabetes_y_pred))

#Scatter Plot
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
# plt.xticks(())
# plt.yticks(())
plt.title("Linear regression Diabeties Dataset")
plt.show()
```

4: Implement SVM classifier (Iris Dataset)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Define the col names
colnames=["sepal_length_in_cm", "sepal_width_in_cm","petal_length_in_cm","petal_width_in_cm", "class"]
#Read the dataset
dataset = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header = None, names= colnames )
#Data
dataset.head()

#Encoding the categorical column
dataset = dataset.replace({"class":  {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})
#Visualize the new dataset
dataset.head()

plt.figure(1)
sns.heatmap(dataset.corr())
plt.title('Correlation On iris Classes')

X = dataset.iloc[:,:-1]
y = dataset.iloc[:, -1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

#Create the SVM model
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
#Fit the model for the data
classifier.fit(X_train, y_train)
#Make the prediction
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

5: Train and fine-tune a Decision Tree for the Moons Dataset

```
import numpy as np
import matplotlib.pyplot as plt

def plot_dataset(X, y, axes):
   plt.figure(figsize=(10,6))
   plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs",alpha = 0.5)
   plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^",alpha = 0.2)
   plt.axis(axes)
```

```
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)

from sklearn.datasets import make_moons
X, y = make_moons(n_samples=10000, noise=0.4, random_state=21)
plot_dataset(X, y, [-3, 5, -3, 3])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)

from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier()

from sklearn.model_selection import GridSearchCV
parameter = {
                        'criterion' : ["gini", "entropy"],
                        'max_leaf_nodes': list(range(2, 50)),
                        'min_samples_split': [2, 3, 4]
                    }
clf = GridSearchCV(tree_clf, parameter, cv = 5,scoring = "accuracy",return_train_score=True,n_jobs=-1)
clf.fit(X_train, y_train)

clf.best_params_
{'criterion': 'gini', 'max_leaf_nodes': 37, 'min_samples_split': 2}

cvres = clf.cv_results_
for mean_score, params in zip(cvres["mean_train_score"], cvres["params"]):
            print (mean_score, params)

clf.score(X_train, y_train)

from sklearn.metrics import confusion_matrix
pred = clf.predict(X_train)
confusion_matrix(y_train,pred)

from sklearn.metrics import precision_score, recall_score
pre = precision_score(y_train, pred)
re  = recall_score(y_train, pred)
print(f"Precision: {pre}  Recall:{re}")

from sklearn.metrics import f1_score
f1_score(y_train, pred)
clf.score(X_test, y_test)
```

7: Implement Batch Gradient Descent with early stopping for Softmax Regression

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

iris=load_iris()
X=iris['data']
y=iris['target']

X_with_bias = np.c_[np.ones([len(X), 1]), X]
np.random.seed(1234)
test_ratio = 0.2
validation_ratio = 0.2
```

```
total_size = len(X_with_bias)
test_size = int(total_size * test_ratio)
validation_size = int(total_size * validation_ratio)
train_size = total_size - test_size - validation_size
rnd_indices = np.random.permutation(total_size)

X_train = X_with_bias[rnd_indices[:train_size]]
y_train = y[rnd_indices[:train_size]]
X_valid = X_with_bias[rnd_indices[train_size:-test_size]]
y_valid = y[rnd_indices[train_size:-test_size]]
X_test = X_with_bias[rnd_indices[-test_size:]]
y_test = y[rnd_indices[-test_size:]]

def one_hot(Y):
   nclasses=Y.max()+1
   m = len(Y)
   Y_one_hot=np.zeros((m,nclasses))
   Y_one_hot[np.arange(m),Y]=1
   return Y_one_hot

y_valid[:10]
one_hot(y_valid[:10])
y_train_prob = one_hot(y_train)
y_valid_prob = one_hot(y_valid)
y_test_prob = one_hot(y_test)

def softmax(sk_X):
   top = np.exp(sk_X)
   bottom = np.sum(top,axis=1,keepdim=True)
   return top/bottom
n_inputs = X_train.shape[1]
n_outputs = len(np.unique(y_train))
print (n_inputs, n_outputs)
```

9: Classification of images of clothing using Tensorflow (Fashion MNIST dataset)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)

fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

train_images.shape
len(train_labels)
train_labels
test_images.shape
len(test_labels)

plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

```
train_images = train_images / 255.0
test_images = test_images / 255.0

plt.figure(figsize=(10,10))
for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(train_images[i], cmap=plt.cm.binary)
   plt.xlabel(class_names[train_labels[i]])
plt.show()
```