```
#imports
import pandas as pd
import numpy as np
import math as mat
from google.colab import drive
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from matplotlib.colors import ListedColormap
from datetime import datetime


```

```
data1 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Assig 4 Logistics Reg wit
#data2 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Assig 4 Logistics Reg wi
```

```
data1.head(10)
```

	trip_id	duration	start_time	end_time	start_lat	start_lon	end_lat	e
0	101750280	35	2018-08-07 11:20:00	2018-08-07 11:55:00	33.748920	-118.275192	33.748920	-118.2
1	46560345	32	9/17/2017 17:51	9/17/2017 18:23	34.035679	-118.270813	34.047749	-118.2
2	120016336	6	2019-04-22 09:22:00	2019-04-22 09:28:00	34.046070	-118.233093	34.047749	-118.2
3	129547190	138	9/22/2019 11:27	9/22/2019 13:45	34.062580	-118.290092	34.059689	-118.2
4			2020 17:11	1/31/2020 17:25	34.026291	-118.277687	34.021660	-118.2
5	63406498	30	2017-12-16 15:18:00	2017-12-16 15:48:00	34.135250	-118.132370	34.135250	-118.1
6	25033469	11	2017-04-15 22:02:00	2017-04-15 22:13:00	34.045181	-118.250237	34.053570	-118.2
7	107479459	15	2018-10-16 17:27:00	2018-10-16 17:42:00	34.041130	-118.267982	34.045422	-118.2
8	132750788	19	2019-11-16 11:24:27	2019-11-16	34.046822	-118.248352	34.046822	-118.2

Se guardó correctamente

```
data1['passholder_type'].nunique
```

```
<bound method IndexOpsMixin.nunique of 0              Walk-up
```

```
1             Walk-up
2         Monthly Pass
3         One Day Pass
4         Monthly Pass
              ...
699995    Monthly Pass
699996    Monthly Pass
699997       Flex Pass
699998         Walk-up
699999         Walk-up
Name: passholder_type, Length: 693618, dtype: object>
```

```
1 data1.shape
```

```
(700000, 14)
```

```
1 data1.isna().sum()
```

```
trip_id                   0
duration                  0
start_time                0
end_time                  0
start_lat              5563
start_lon              5563
end_lat               18574
end_lon               18574
bike_id                   0
plan_duration           208
trip_route_category       0
passholder_type        2576
start_station             0
end_station               0
dtype: int64
```

```
1 data1['end_time'].max()
```

Se guardó correctamente

```
1 data1['end_time'].min()
```

```
'1/1/2017 0:23'
```

```
1
```

```
1 data1.dtypes
```

```
trip_id                 int64
duration                int64
start_time             object
end_time               object
start_lat             float64
start_lon             float64
end_lat               float64
end_lon               float64
```

```
bike_id                object
plan_duration         float64
trip_route_category    object
passholder_type        object
start_station           int64
end_station             int64
dtype: object
```

Feature preparation

```
1 data1['end_time'] = pd.to_datetime(data1['end_time'],yearfirst=True)
2
```

```
1 data1['start_time'] = pd.to_datetime(data1['start_time'],yearfirst=True)
```

```
1 def timemin(s1,s2):
2
3   return pd.Timedelta(s2 - s1).seconds/60
```

```
1 data1['time'] = data1.apply(lambda x: timemin(s2=x['end_time'], s1 = x['start_time']),
```

```
1 data1['time']
```

```
0          35.000000
1          32.000000
2           6.000000
3         138.000000
4          14.000000
             ...
699995     17.000000
699996      9.683333
699999     39.000000
Name: time, Length: 700000, dtype: float64
```



```
1 data1['time'].max()
```

```
1439.0
```

```
1 count = data1[data1['time'] >=500]
2 count.shape
```

```
(6395, 15)
```

```
1 data1 = data1[data1['time'] <=500]
```

```
1
```

```
1 import geopy.distance
```

```
1 df = data1.copy()
```

```
1 df = df.dropna()
```

```
1 def hess(a,al,b,bl):
2   coords_1 = (a, al)
3   coords_2 = (b, bl)
4
5   return geopy.distance.geodesic(coords_1, coords_2).km
```

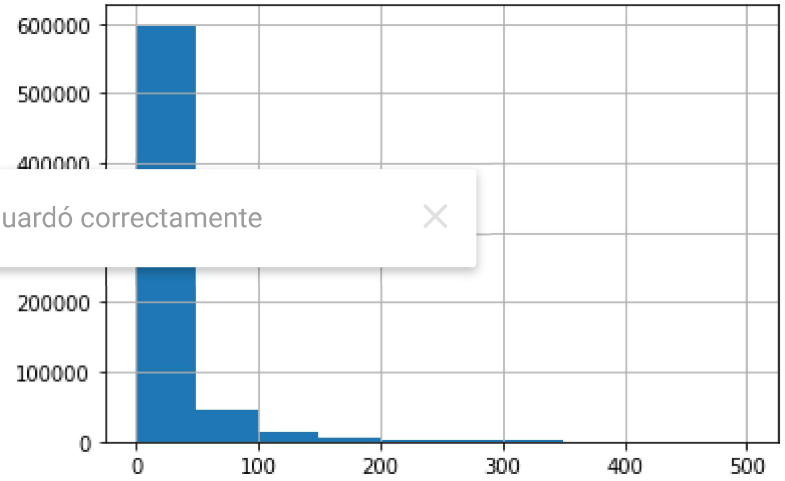
```
1 df['distance_km'] = df.apply(lambda x: hess(a=x['start_lat'], al = x['start_lon'], b=
```

```
1 df['distance_km'].head()
```

```
0    0.000000
1    2.882193
2    0.949087
3    0.529935
4    0.521920
Name: distance_km, dtype: float64
```

```
1 df['time'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f0cee2310>
```



Logistic Regression

```
1 df1 = df
```

```
1 df.head()
```

	trip_id	duration	start_time	end_time	start_lat	start_lon	end_lat	er
0	101750280	35	2018-08-07 11:20:00	2018-08-07 11:55:00	33.748920	-118.275192	33.748920	-118.2
1	46560345	32	2017-09-17 17:51:00	2017-09-17 18:23:00	34.035679	-118.270813	34.047749	-118.2
2	120016336	6	2019-04-22 09:22:00	2019-04-22 09:28:00	34.046070	-118.233093	34.047749	-118.2
3	129547190	138	2019-09-22 11:27:00	2019-09-22 13:45:00	34.062580	-118.290092	34.059689	-118.2
4	136619463	14	2020-01-31 17:11:00	2020-01-31 17:25:00	34.026291	-118.277687	34.021660	-118.2

```
1
```

```
1 df1[df1['distance_km'] >=100].shape
```

```
(98, 16)
```

```
1 df1 =df1[df1['distance_km'] <=100]
```

```
1 df1[df1['distance_km'] >=100].shape
```

```
(0, 16)
```

Se guardó correctamente

```
2     if x == "Monthly Pass":
3       return 1
4     else:
5       return 0
6
```

```
1 df1["passholder_type"] = df1['passholder_type'].map(transform)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  """Entry point for launching an IPython kernel.
```

```
df1['passholder_type'].nunique()
```

```
2
```

```
# Shuffle your dataset
shuffle_df = df1.sample(frac=1)

# Define a size for your train set
train_size = int(0.9 * len(df1))

# Split your dataset
train_df = shuffle_df[:train_size]
test_df = shuffle_df[train_size:]
```

```
def norma(data):
  return (data-data.min())/(data.max()-data.min())
```

```
X = train_df.loc[:,['time','distance_km']]
y = train_df.loc[:,'passholder_type']
```

```
X.shape
```

```
(604384, 2)
```

```
Xnorm = norma(X)
```

```
X_test = test_df.loc[:,['time','distance_km']]
y_test = test_df.loc[:,'passholder_type']
```

```
Xtestnorm = norma(X_test)
```

Se guardó correctamente

```
(67154, 2)
```

```
X = X.to_numpy()
y = y.to_numpy()
```

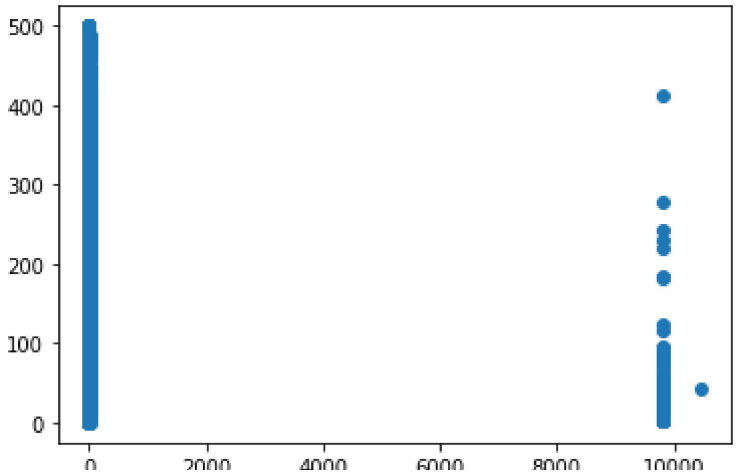
```
X_test = X_test.to_numpy()
y_test = y_test.to_numpy()
```

```
y_test
```

```
array([0, 1, 1, ..., 1, 0, 1])
```

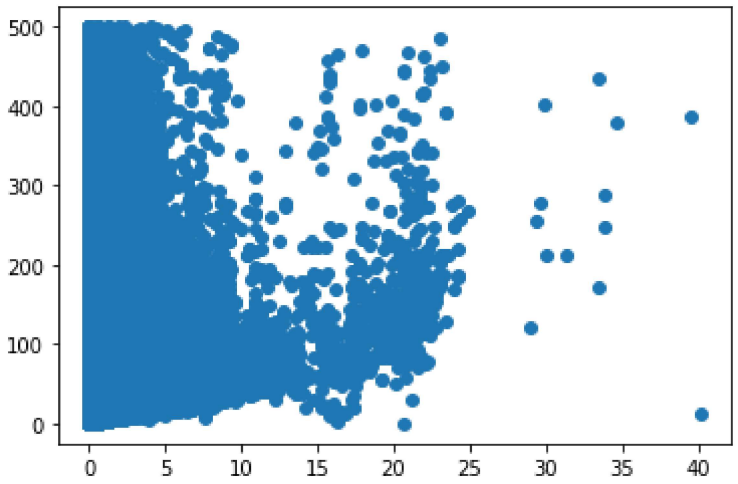
```
plt.scatter(df['distance_km'],df['time'])
```

```
<matplotlib.collections.PathCollection at 0x7f8f269cc310>
```




```
1 plt.scatter(df1['distance_km'],df1['time'])
```

```
<matplotlib.collections.PathCollection at 0x7f8ef74c2c90>
```




```
1 df1.max()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Droppi
                                                an IPython kernel.
                                    179457605
duration                                 1440
start_time                2021-12-31 22:57:00
end_time                  2022-01-01 21:35:00
start_lat                           55.705528
start_lon                           37.606541
end_lat                             55.705528
end_lon                             37.606541
plan_duration                           999.0
trip_route_category                Round Trip
passholder_type                             1
start_station                            4594
end_station                              4594
time                                    500.0
distance_km                         40.131137
dtype: object
```

Se guardó correctamente

Neuronal network

```
def linear(H):
  return H

def ReLU(H):
  return H*(H>0)

def sigmoid(H):
  return 1/(1+np.exp(-H))

def softmax(H):
  eH=np.exp(H)
  return eH/eH.sum(axis=1,keepdims=True)

def cross_entropy(Y, P_hat):
  return -(1/len(Y))*np.sum(Y*np.log(P_hat))

def OLS(Y, Y_hat):
  return (1/(2*len(Y)))*np.sum((Y-Y_hat)**2)

def one_hot_encode(y):
  N=len(y)
  K=len(set(y))
  Y=np.zeros((N,K))

  for i in range(N):
    Y[i, y[i]]=1

  return Y

def accuracy(y, y_hat):
  return np.mean(y==y_hat)

def R2(y, y_hat):
                                    p.sum((y -y.mean())**2)

# Derivatives of Activation Functions

def derivative(Z,a):
  if a==linear:
    return 1
  elif a==sigmoid:
    return Z*(1-Z)
  elif a==np.tanh:
    return 1-Z*Z
  elif a==ReLU:
    return (Z>0).astype(int)
  else:
    ValueError("Unknown Activation Function")
```

Se guardó correctamente

```
cmap_bold = ListedColormap(["#FF0000","#00FF00","#0000FF"])
cmap_light = ListedColormap(["#FFBBBB","#BBFFBB","#BBBBFF"])
```

```
class ANN():

  def __init__(self,architecture, activations=None, mode=0):
    self.mode=mode
    self.architecture=architecture
    self.activations=activations
    self.L = len(architecture)+1

  def fit(self, X, y, eta=1e-3, epochs=1e3, show_curve=False):
    epochs=int(epochs)
    if self.mode:
      Y=y
    else:
      Y=one_hot_encode(y)

    N,D =X.shape
    K=Y.shape[1]

    #Weights Init
    self.W={l: np.random.randn(M[0],M[1]) for l, M in enumerate
            (zip(([D]+self.architecture),(self.architecture+[K])),1)}
    self.b={l:np.random.randn(M) for l, M in enumerate(self.architecture +[K],1)}
    #Activations Loading
    if self.activations is None:
      self.a={l: ReLU for l in range(1,self.L)}
    else:
      self.a={l: act for l, act in enumerate(self.activations,1)}

    #Outputs
    if self.mode:
      self.a[self.L]=linear
    else:
      self.a[self.L]=softmax
```

Se guardó correctamente

```
    #GradDesc and Back Propagation
    for epoch in range(epochs):
      self.forward(X)
      if self.mode:
        J[epoch]= OLS(Y, self.Z[self.L])
      else:
        J[epoch]=cross_entropy(Y, self.Z[self.L])

      dH = (1/N)*(self.Z[self.L]-Y)

      for l in sorted(self.W.keys(), reverse=True):
        dW = self.Z[l-1].T@dH
        db = dH.sum(axis=0)
        #update rules
        self.W[l] -= eta*dW
        self.b[l] -= eta*db

        if l>1:
```

```
            dZ = dH@self.W[l].T
            dH = dZ*derivative(self.Z[l-1], self.a[l-1])

    if show_curve:
        plt.figure()
        plt.plot(J)
        plt.xlabel("epochs")
        plt.ylabel("$\mathcal{J}$")
        plt.show()

  def forward(self, X):
    self.Z={0:X}
    for l in sorted(self.W.keys()):
      self.Z[l]=self.a[l](self.Z[l-1]@self.W[l]+self.b[l])

  def predict(self, X):
    self.forward(X)
    if self.mode:
      return self.Z[self.L]
    else:
      return self.Z[self.L].argmax(axis=1)
```

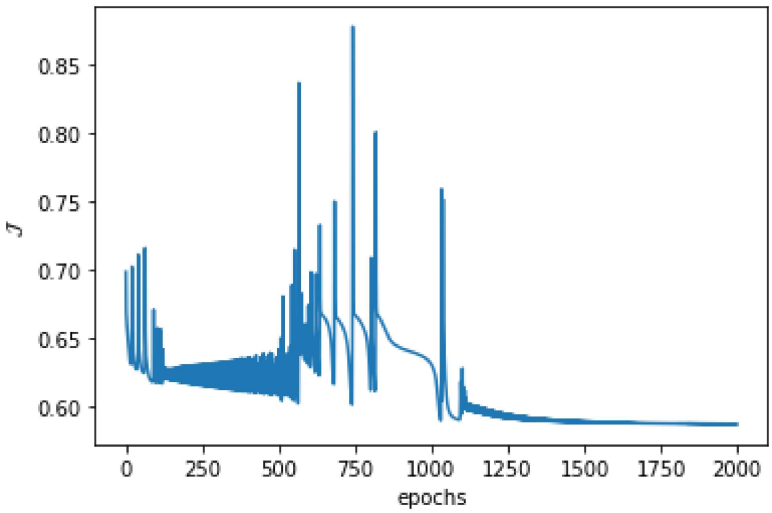
```
def main_class():
    D = 2
    K = 3
    N = int(K*1e3)


    ann=ANN([8, 8, 8],[ReLU, np.tanh, np.tanh])
    ann.fit(X,y, eta =3e-2, epochs=2e3, show_curve=True)
    y_hat = ann.predict(X)
    y_hatt = ann.predict(X_test)

    print(f"Training Accuracy: {accuracy(y, y_hat):0.4f}")
                              acy(y_test, y_hatt):0.4f}")

    x1 = np.linspace(X[:,0].min() - 1, X[:,0].max() + 1, 1000)
    x2 = np.linspace(X[:,1].min() - 1, X[:,1].max() + 1, 1000)

    xx1, xx2 = np.meshgrid(x1, x2)
    Z = ann.predict(np.c_[xx1.ravel(),xx2.ravel()]).reshape(*xx1.shape)

    plt.figure()
    plt.pcolormesh(xx1, xx2, Z, cmap = cmap_light)
    plt.scatter(X[:,0], X[:,1], c = y, cmap = cmap_bold,alpha=0.2)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    plt.show()
```

Se guardó correctamente

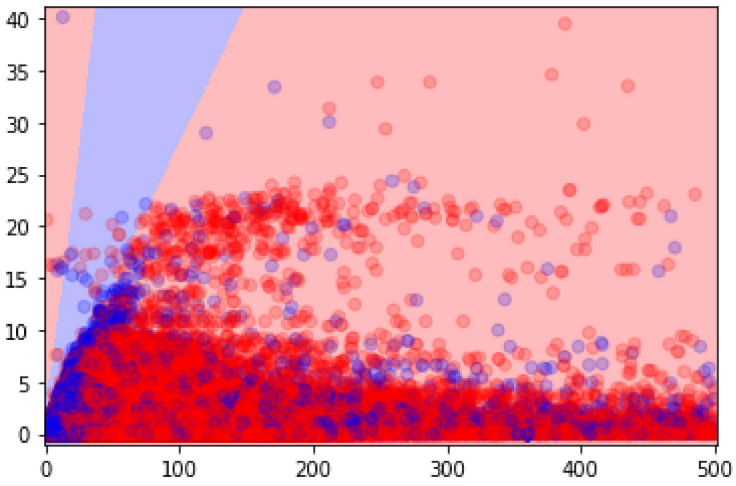
```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
```

```
5 else:
6   print(gpu_info)
```

```
1 if __name__ =="__main__":
2   main_class()
```




```
Training Accuracy: 0.6966
Test Accuracy: 0.6994
```



Se guardó correctamente

Data norm

```
1 def main_class2():
2     D = 2
3     K = 3
4     N = int(K*1e3)
5
6
7     ann=ANN([8, 8, 8],[ReLU, np.tanh, np.tanh])
8     ann.fit(Xnorm,y, eta =1e-2, epochs=2e3, show_curve=True)
9     y_hat = ann.predict(Xnorm)
10    y_hatt = ann.predict(Xtestnorm)
11
12    print(f"Training Accuracy: {accuracy(y, y_hat):0.4f}")
13    print(f"Test Accuracy: {accuracy(y_test, y_hatt):0.4f}")
14
```

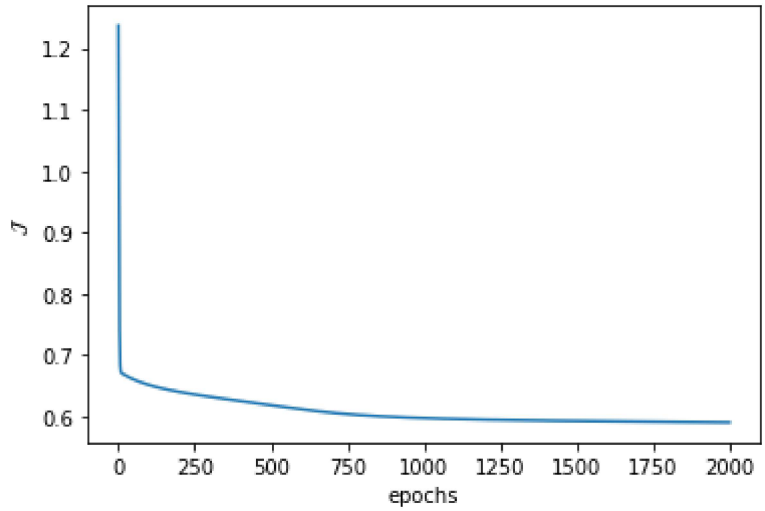
```
    x1 = np.linspace(X[:,0].min() - 1, X[:,0].max() + 1, 1000)
    x2 = np.linspace(X[:,1].min() - 1, X[:,1].max() + 1, 1000)

    xx1, xx2 = np.meshgrid(x1, x2)
    Z = ann.predict(np.c_[xx1.ravel(),xx2.ravel()]).reshape(*xx1.shape)

    plt.figure()
    plt.pcolormesh(xx1, xx2, Z, cmap = cmap_light)
    plt.scatter(X[:,0], X[:,1], c = y, cmap = cmap_bold,alpha=0.2)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    plt.show()
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

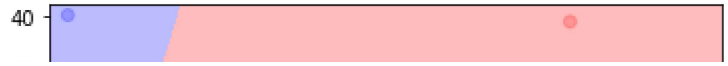
```
Wed Sep  7 01:36:00 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   36C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Se guardó correctamente

```
if __name__ =="__main__":
  main_class2()
```




```
Training Accuracy: 0.6994
Test Accuracy: 0.7011
```



80-20

[ ] ↳ *8 celdas ocultas*

Balance the 0 and 1

[ ] ↳ *13 celdas ocultas*

Other model

1

Se guardó correctamente

1

Productos pagados de Colab - Cancela los contratos aquí

✓ 0 s se ejecutó 22:59

Se guardó correctamente