# Assignment 1 BUDA 451

Collin Edwards

2025-02-09

## Data Structures

### Question 1a

#### Given list s = "Hamburg", how do you index/slice "mbur" from list s?

In order to slice the character is at mburg out of hamburg, you have to first note where the charcters are.[1] since `m` is at the 2nd position,`b` is in position 3,`u` is in position 4 and `r` is at position 5.

```
s = "hamburg"
substring = s[2:6]
print(substring)  # Output: "mbur"
```

```
mbur
```

### Question 1b

#### What the major differences between two Python data structures: list and set. Give a or two application examples where we might need to convert a list to a set.

**List** *Maintains the order* if elements and allows you to index via normal means. Can also contain duplicate elements.

**Set** *Unordered* collection of elements. Does *not* support indexing or slicing since the order is not guaranteed. Automatically removes the duplicate elements since each one is unique

[1]Since python like many programming languages start counting from 0 you have to account for that when slicing a matrix of anything.

**Application examples for converting a list to a set**

1. **Removing Duplicates** For instance if you have a bunch of email addresses in your list and you want to include only unique entries
2. **Membership Testing** If you want to frequently check whether an element exists within a collection, if you convert the list to a set this can reduce run time and improve performance

## Question 1c

**If you are working with a toy dataset as show in following**

| student_id | gpa |
|---|---|
| 001 | 4.6 |
| 002 | 4.2 |
| 003 | 4.0 |

**and you want to use a `dictionary` to store the data for further processing. Write Python code to demonstrate how to create such a `dictionary` in Python, and to iterate over the `key: value` pairs.**

```python
''' creating the dictionary. essentially storing the values in variables and
associating the variables with the other variable'''
gpa_dict = {
    "001": 4.6,
    "002": 4.2,
    "003": 4.0
}

# iterating over the key-value pairs with a for statement
for student_id, gpa in gpa_dict.items():
    print(f"Student ID: {student_id}, GPA: {gpa}")
    '''the .items() method used in the line above will return a view object and
    a list of the key-value tuple pairs. Tuples are also immutable compared to
    lists'''
```

```
Student ID: 001, GPA: 4.6
Student ID: 002, GPA: 4.2
Student ID: 003, GPA: 4.0
```

**This code will print each of the student ID's along with their GPA**

## Function and library

### Question 2a Write a Python function to calculate the Euclidean distance between two data points `x` and `y`.

The Euclidean distance between two data points = [ 1, 2, … , ] and = [ 1, 2, … , ] is defined as **Euclidean Distance Formula**

The Euclidean distance between two data points
$\mathbf{x} = [x_1, x_2, \dots, x_n]$
and
$\mathbf{y} = [y_1, y_2, \dots, y_n]$
is defined as:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

```
import math # importing for math operations

def euc_dist(x, y):
    # ensuring that the two lists have the same length
    if len(x) != len(y):
        raise ValueError("The data points must have the same dimension.")

    sum_squared = 0
    for xi, yi in zip(x, y):#using for to iterate over paired elements
        sum_squared += math.pow(xi - yi, 2)
    dist = math.sqrt(sum_squared)#calculates square root to get euclidean distance
    return dist

# given data points from the question
x = [1, 2, 4, -4]
y = [10.5, 12.1, 13.2, 105.8]

# Calculate the Euclidean distance
distance = euc_dist(x, y)
print("Euclidean distance:", distance)
```

```
Euclidean distance: 111.05377075993412
```

## Numpy and Pandas

**Question 3a create a 10x10 array with random numbers, and name it as `np_data`. Print out the array.**

```python
import numpy as np
import pandas as pd

'''creating a 10x10 array with random numbers and print it. Note: np.random.rand
generates numbers in [0, 1)'''

np_data = np.random.rand(10, 10)
print("np_data:")
print(np_data)
```

```
np_data:
[[0.18766857 0.6521975  0.21111353 0.45507086 0.92039183 0.00460018
  0.7610065  0.32497228 0.70508584 0.37629859]
 [0.29722373 0.42411183 0.8903304  0.40764577 0.22309156 0.90618175
  0.38760479 0.40789548 0.01603669 0.41371331]
 [0.55946161 0.516955   0.94127821 0.8918773  0.45478488 0.00149964
  0.995376   0.49382916 0.51612088 0.80041298]
 [0.17501483 0.58147718 0.9401548  0.83869378 0.41842164 0.64132846
  0.7301493  0.30991517 0.14349295 0.84468953]
 [0.53054481 0.4542115  0.52624097 0.9860977  0.18058438 0.04409628
  0.46212581 0.65172012 0.49099624 0.77524234]
 [0.51808458 0.2723701  0.83708736 0.95884106 0.64752155 0.12581512
  0.94489438 0.64264484 0.49406551 0.36833369]
 [0.61158502 0.79813456 0.43885388 0.87782716 0.51071474 0.93003944
  0.245111   0.88737396 0.93937602 0.49381396]
 [0.70222476 0.28028905 0.30292349 0.59180161 0.94187037 0.04587023
  0.76027305 0.52487624 0.92594873 0.4670616 ]
 [0.11666859 0.25177581 0.33430992 0.07206836 0.85858438 0.79984938
  0.39512678 0.82857294 0.51870026 0.57250749]
 [0.5840482  0.98936444 0.82738966 0.92422701 0.25771903 0.44055579
  0.66858243 0.97781167 0.92867175 0.42826707]]
```

**Question 3a Part II slice the sub-array of 6th-10th rows, 1st-5th columns of `np_data`, and name it as `subdata` (`subdata` should be a 5x5 array). Print out the sub-array.**

```
#    Since Python uses zero-based indexing as mentioned earlier:
#    - 6th to 10th rows correspond to indices 5 to 9.
#    - 1st to 5th columns correspond to indices 0 to 4.
subdata = np_data[5:10, 0:5]
print("\nSub-array (6th-10th rows, 1st-5th columns):")
print(subdata)
```

```
Sub-array (6th-10th rows, 1st-5th columns):
[[0.51808458 0.2723701  0.83708736 0.95884106 0.64752155]
 [0.61158502 0.79813456 0.43885388 0.87782716 0.51071474]
 [0.70222476 0.28028905 0.30292349 0.59180161 0.94187037]
 [0.11666859 0.25177581 0.33430992 0.07206836 0.85858438]
 [0.5840482  0.98936444 0.82738966 0.92422701 0.25771903]]
```

**Question 3b create a `Pandas DataFrame` from `np_data` above, and name it as `pd_data`. Print out `pd_data`. Then slice the sub-DataFrame of 1st-5th rows, 6th-10th columns of `pd_data`, and name it as `subdf` (`subdf` should be a 5x5 dataframe). Print out `subdf`.**

```
# Creating a df from np_data and print it.
pd_data = pd.DataFrame(np_data)
print("\npd_data DataFrame:")
print(pd_data)
pd_data = pd.DataFrame(np_data)
subdf = pd_data.iloc[0:5, 5:10]
print("\nSub-DataFrame (1st-5th rows, 6th-10th columns):")
print(subdf)
```

```
pd_data DataFrame:
          0         1         2         3         4         5         6  \
0  0.187669  0.652198  0.211114  0.455071  0.920392  0.004600  0.761006
1  0.297224  0.424112  0.890330  0.407646  0.223092  0.906182  0.387605
2  0.559462  0.516955  0.941278  0.891877  0.454785  0.001500  0.995376
```

```
3  0.175015  0.581477  0.940155  0.838694  0.418422  0.641328  0.730149
4  0.530545  0.454212  0.526241  0.986098  0.180584  0.044096  0.462126
5  0.518085  0.272370  0.837087  0.958841  0.647522  0.125815  0.944894
6  0.611585  0.798135  0.438854  0.877827  0.510715  0.930039  0.245111
7  0.702225  0.280289  0.302923  0.591802  0.941870  0.045870  0.760273
8  0.116669  0.251776  0.334310  0.072068  0.858584  0.799849  0.395127
9  0.584048  0.989364  0.827390  0.924227  0.257719  0.440556  0.668582

          7         8         9
0  0.324972  0.705086  0.376299
1  0.407895  0.016037  0.413713
2  0.493829  0.516121  0.800413
3  0.309915  0.143493  0.844690
4  0.651720  0.490996  0.775242
5  0.642645  0.494066  0.368334
6  0.887374  0.939376  0.493814
7  0.524876  0.925949  0.467062
8  0.828573  0.518700  0.572507
9  0.977812  0.928672  0.428267

Sub-DataFrame (1st-5th rows, 6th-10th columns):
          5         6         7         8         9
0  0.004600  0.761006  0.324972  0.705086  0.376299
1  0.906182  0.387605  0.407895  0.016037  0.413713
2  0.001500  0.995376  0.493829  0.516121  0.800413
3  0.641328  0.730149  0.309915  0.143493  0.844690
4  0.044096  0.462126  0.651720  0.490996  0.775242
```

I use *iloc* because I wanted to select rows and columns based on their order in the df rather than their labels. It's like slicing a list. e.g df.iloc[0:5] gives you the first 5 rows. df.iloc[:, 5:10] gives you the columns in positions 6 through 10.

## Tabular data

**Question 4 You are given the UCI Parkinsons Data Set:This dataset is composed of a range ofbiomedical voice measurements from 31 people, 23 with Parkinson's disease (PD). Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals ("name" column). The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD. The data is in ASCII CSV format. You may find more introduction about the data here: https://archive.ics.uci.edu/ml/datasets/Parkinsons**

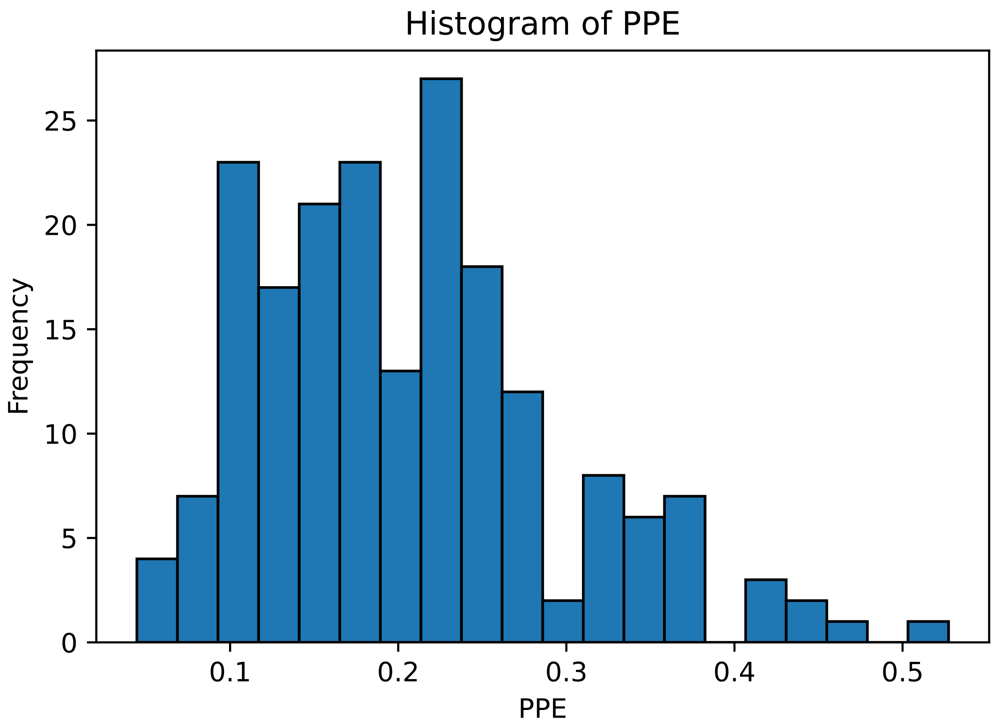
```
df = pd.read_csv("~/Downloads/parkinsons(2).data")
print(df.head)
# (b) Display summary statistics for all attributes.
print("Summary statistics for all attributes:")
print(df.describe())

# (c) Plot histograms for the attributes: PPE, DFA, MDVP:Fo(Hz)
import matplotlib.pyplot as plt

attributes = ['PPE', 'DFA', 'MDVP:Fo(Hz)']

for attr in attributes:
    plt.figure(figsize=(6, 4))
    plt.hist(df[attr].dropna(), bins=20, edgecolor='black')
    plt.title(f'Histogram of {attr}')
    plt.xlabel(attr)
    plt.ylabel('Frequency')
    plt.show()

# (d) Plot two histograms for each attribute (one for status=1 and one for status=0)
for attr in attributes:
    # Filter the data for each status
    df_status1 = df[df['status'] == 1]
    df_status0 = df[df['status'] == 0]

    plt.figure(figsize=(12, 4))

    # Histogram for instances with status = 1 (Parkinson's)
    plt.subplot(1, 2, 1)
    plt.hist(df_status1[attr].dropna(), bins=20, color='tomato', edgecolor='black')
```

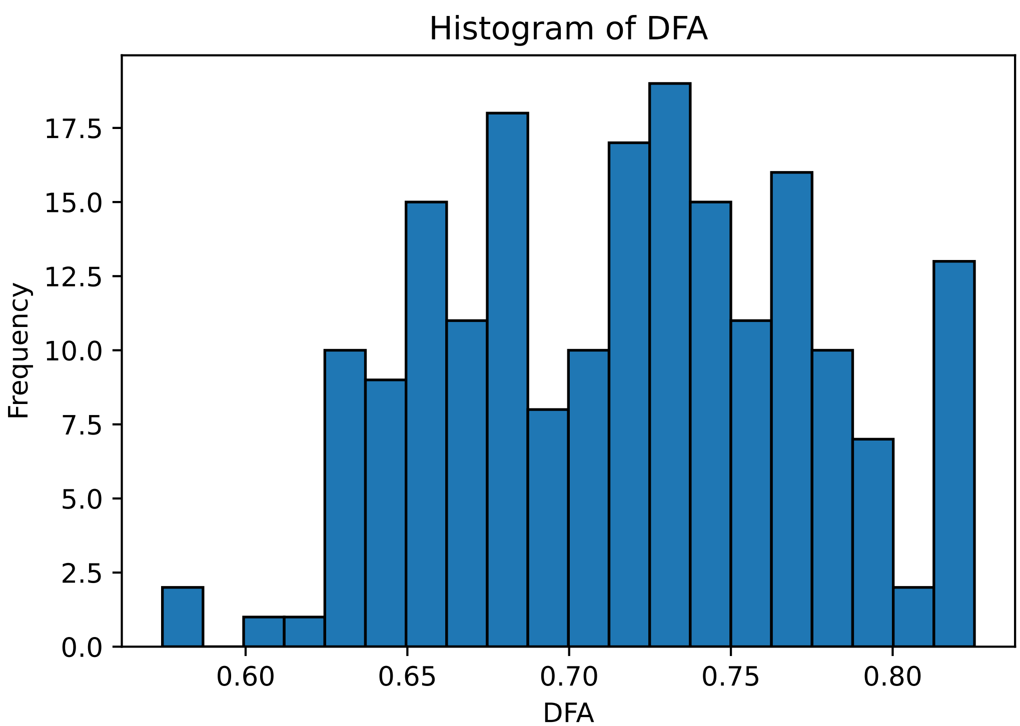
```
    plt.title(f'{attr} (Status = 1)')
    plt.xlabel(attr)
    plt.ylabel('Frequency')

    # Histogram for instances with status = 0 (Healthy)
    plt.subplot(1, 2, 2)
    plt.hist(df_status0[attr].dropna(), bins=20, color='skyblue', edgecolor='black')
    plt.title(f'{attr} (Status = 0)')
    plt.xlabel(attr)
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()
```

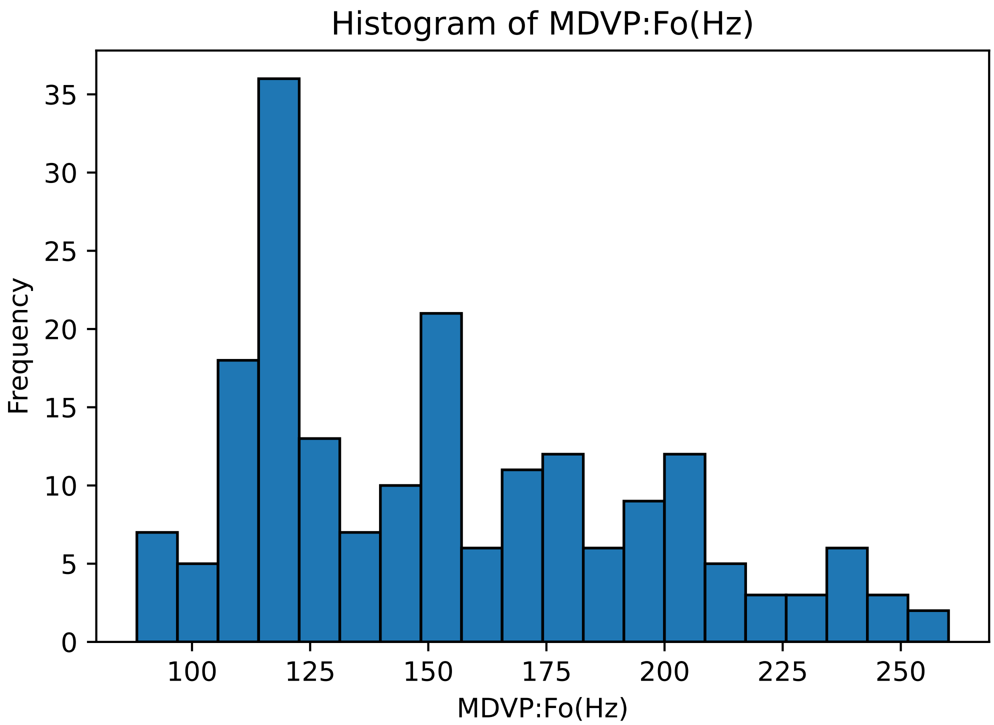
```
<bound method NDFrame.head of                 name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  I
0    phon_R01_S01_1      119.992       157.302        74.997        0.00784
1    phon_R01_S01_2      122.400       148.650       113.819        0.00968
2    phon_R01_S01_3      116.682       131.111       111.555        0.01050
3    phon_R01_S01_4      116.676       137.871       111.366        0.00997
4    phon_R01_S01_5      116.014       141.781       110.655        0.01284
..              ...          ...           ...           ...            ...
190  phon_R01_S50_2      174.188       230.978        94.261        0.00459
191  phon_R01_S50_3      209.516       253.017        89.488        0.00564
192  phon_R01_S50_4      174.688       240.005        74.287        0.01360
193  phon_R01_S50_5      198.764       396.961        74.904        0.00740
194  phon_R01_S50_6      214.289       260.277        77.973        0.00567

     MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  ...  \
0             0.00007   0.00370   0.00554     0.01109       0.04374  ...
1             0.00008   0.00465   0.00696     0.01394       0.06134  ...
2             0.00009   0.00544   0.00781     0.01633       0.05233  ...
3             0.00009   0.00502   0.00698     0.01505       0.05492  ...
4             0.00011   0.00655   0.00908     0.01966       0.06425  ...
..                ...       ...       ...         ...           ...  ...
190           0.00003   0.00263   0.00259     0.00790       0.04087  ...
191           0.00003   0.00331   0.00292     0.00994       0.02751  ...
192           0.00008   0.00624   0.00564     0.01873       0.02308  ...
193           0.00004   0.00370   0.00390     0.01109       0.02296  ...
194           0.00003   0.00295   0.00317     0.00885       0.01884  ...

     Shimmer:DDA      NHR     HNR  status      RPDE       DFA   spread1  \
0        0.06545  0.02211  21.033       1  0.414783  0.815285 -4.813031
```

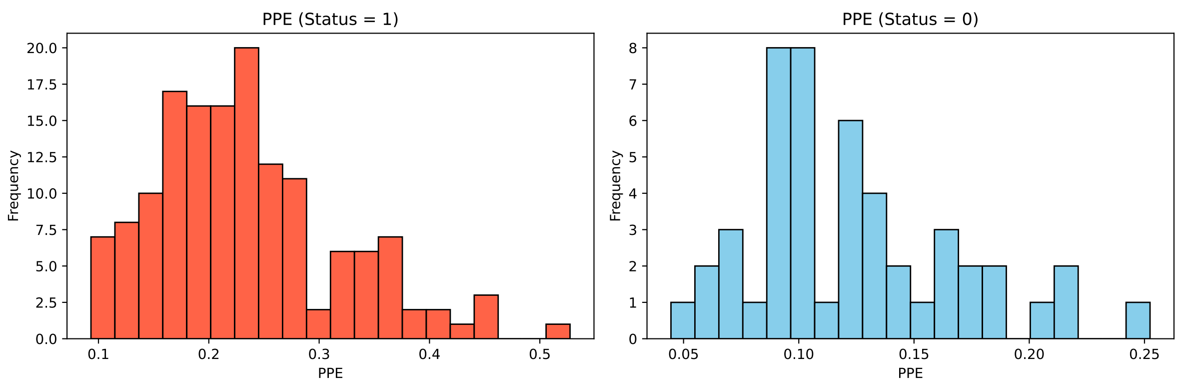
```
1        0.09403  0.01929  19.085       1  0.458359  0.819521 -4.075192
2        0.08270  0.01309  20.651       1  0.429895  0.825288 -4.443179
3        0.08771  0.01353  20.644       1  0.434969  0.819235 -4.117501
4        0.10470  0.01767  19.649       1  0.417356  0.823484 -3.747787
..           ...      ...     ...     ...       ...       ...       ...
190      0.07008  0.02764  19.517       0  0.448439  0.657899 -6.538586
191      0.04812  0.01810  19.147       0  0.431674  0.683244 -6.195325
192      0.03804  0.10715  17.883       0  0.407567  0.655683 -6.787197
193      0.03794  0.07223  19.020       0  0.451221  0.643956 -6.744577
194      0.03078  0.04398  21.209       0  0.462803  0.664357 -5.724056

      spread2        D2       PPE
0    0.266482  2.301442  0.284654
1    0.335590  2.486855  0.368674
2    0.311173  2.342259  0.332634
3    0.334147  2.405554  0.368975
4    0.234513  2.332180  0.410335
..        ...       ...       ...
190  0.121952  2.657476  0.133050
191  0.129303  2.784312  0.168895
192  0.158453  2.679772  0.131728
193  0.207454  2.138608  0.123306
194  0.190667  2.555477  0.148569

[195 rows x 24 columns]>
Summary statistics for all attributes:
       MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
count   195.000000    195.000000    195.000000      195.000000
mean    154.228641    197.104918    116.324631        0.006220
std      41.390065     91.491548     43.521413        0.004848
min      88.333000    102.145000     65.476000        0.001680
25%     117.572000    134.862500     84.291000        0.003460
50%     148.790000    175.829000    104.315000        0.004940
75%     182.769000    224.205500    140.018500        0.007365
max     260.105000    592.030000    239.170000        0.033160

       MDVP:Jitter(Abs)    MDVP:RAP    MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
count        195.000000  195.000000  195.000000  195.000000    195.000000
mean           0.000044    0.003306    0.003446    0.009920      0.029709
std            0.000035    0.002968    0.002759    0.008903      0.018857
min            0.000007    0.000680    0.000920    0.002040      0.009540
25%            0.000020    0.001660    0.001860    0.004985      0.016505
50%            0.000030    0.002500    0.002690    0.007490      0.022970
```

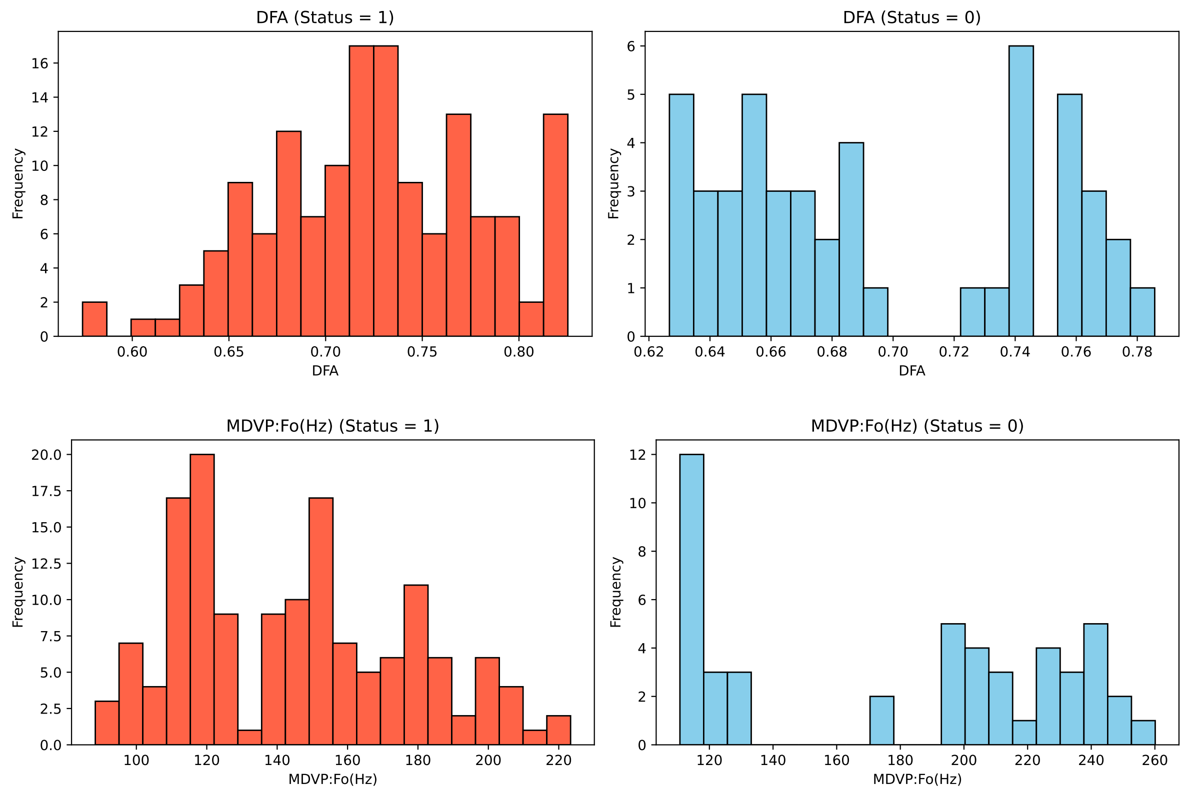
```
75%          0.000060    0.003835    0.003955    0.011505      0.037885
max          0.000260    0.021440    0.019580    0.064330      0.119080

       MDVP:Shimmer(dB)  ...  Shimmer:DDA         NHR         HNR      status  \
count        195.000000  ...   195.000000  195.000000  195.000000  195.000000
mean           0.282251  ...     0.046993    0.024847   21.885974    0.753846
std            0.194877  ...     0.030459    0.040418    4.425764    0.431878
min            0.085000  ...     0.013640    0.000650    8.441000    0.000000
25%            0.148500  ...     0.024735    0.005925   19.198000    1.000000
50%            0.221000  ...     0.038360    0.011660   22.085000    1.000000
75%            0.350000  ...     0.060795    0.025640   25.075500    1.000000
max            1.302000  ...     0.169420    0.314820   33.047000    1.000000

             RPDE         DFA     spread1     spread2          D2         PPE
count  195.000000  195.000000  195.000000  195.000000  195.000000  195.000000
mean     0.498536    0.718099   -5.684397    0.226510    2.381826    0.206552
std      0.103942    0.055336    1.090208    0.083406    0.382799    0.090119
min      0.256570    0.574282   -7.964984    0.006274    1.423287    0.044539
25%      0.421306    0.674758   -6.450096    0.174351    2.099125    0.137451
50%      0.495954    0.722254   -5.720868    0.218885    2.361532    0.194052
75%      0.587562    0.761881   -5.046192    0.279234    2.636456    0.252980
max      0.685151    0.825288   -2.434031    0.450493    3.671155    0.527367

[8 rows x 23 columns]
```


Histogram of PPE
Frequency
0
5
10
15
20
25
0.1
0.2
0.3
0.4
0.5
PPE


Histogram of DFA
Frequency
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.60
0.65
0.70
0.75
0.80
DFA


Histogram of MDVP:Fo(Hz)
Frequency
MDVP:Fo(Hz)
0
5
10
15
20
25
30
35
100
125
150
175
200
225
250


PPE (Status = 1)
Frequency
PPE
PPE (Status = 0)
Frequency
PPE




This code first prints out a summary of all the data so you can see basic stats like the average and spread for each column. Then, it creates simple bar graphs (histograms) for three specific columns (PPE, DFA, and MDVP:Fo(Hz)) to show how the data values are distributed. Finally, it splits the data into two groups based on whether the person has Parkinson's (status = 1) or is healthy (status = 0) and makes separate histograms for each group. This makes it easier to compare how the numbers differ between the two groups.