Team 12 - Assignment 01: Data Preprocessing Project

Using a Python version that is 3.10+ globally or in a virtual environment

1. Update pip: `python.exe -m pip install --upgrade pip`
2. run the following command in your terminal: `pip install -r requirements.txt`

This will install the required dependencies to run our Python code

- ipykernal
- scikit-learn
- matplotlib
- pandas
- numpy
- notebook

Team 12 Member assignment and topics

Alicia Luna

- Missing values
- Outliers
- Duplicate data

assignment01_alicia_luna_jupData.ipynb

Derek Dilger

- Aggregation
- Discretization
- Principle Components Analysis
- Sampling

Ian Dilger

- Testing/trining split

Jesus Cervantes

- Saving a dataframe
- Dropping fields

- Means and Standard Deviation

Matthew Mendoza & Gary Young

- Missing values
- Concatenating rows and columns

assignment01_matthew_mendoza.ipynb

Thomas Jaramillo-Ocha

- Calculated fields
- Feature normalization

assignment01_thomas_jaramilloochoa.ipynb

Yunjeong Lee (Luna)

- Shuffling dataframes
- Sorting dataframes

assignment01_yunjeong_lee.ipynb

Alicia Luna

- Missing values
- Outliers
- Duplicate data

assignment01_alicia_luna_jupData.ipynb

In [ ]:
```
import numpy as nm
import matplotlib.pyplot as plt
import pandas as pd

# reads and copies the data from the csv file
data = pd.read_csv(r'.\data\uci_heartDisease\uci_heartDisease_changed.csv')
# prints out original data
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d\n' % (data.shape[1]))
print(data)
# ------------------------------------------------------------------------------
# detects duplicated data and deletes it in another copy of data
dups = data.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
print('Now discarding duplicate rows')
data2 = data.drop_duplicates()
print('Number of instances = %d\n' % (data2.shape[0]))
print(data2)
```

```
# ------------------------------------------------------------------------------
# Changes the missing values
data3 = data2.replace('?', nm.NaN)
print(data3)

# Counts number of missing values
print('Number of missing values:')
for col in data3.columns:
    print('\t%s: %d' % (col, data3[col].isna().sum()))

# Fills in missing values with the median of the columns
print('\nLet us fill in the missing values')
for col in data3.columns:
    column = '%s' % col
    datacol = data3[column]
    data3[column] = datacol.fillna(datacol.median())

# Recounts missing values and prints new dataset
print('\nNumber of missing values:')
for col in data3.columns:
    print('\t%s: %d' % (col, data3[col].isna().sum()))
print(data3)
# ------------------------------------------------------------------------------
# Finding outliers
for col in data3.columns:
    column = '%s' % col
    datacol = data3[column]
    data3[column] = pd.to_numeric(data3[column])
data3.boxplot(figsize=(20, 3))
plt.show()

# calculate z scores to delete outliers
Z = (data3-data3.mean())/data3.std()
print('Number of rows before discarding outliers = %d' % (Z.shape[0]))
Z2 = Z.loc[((Z > -3).sum(axis=1) == 14) & ((Z <= 3).sum(axis=1) == 14), :]
print('Number of rows after discarding outlier values = %d' % (Z2.shape[0]))
print(Z2)
```

```
Number of instances = 303
Number of attributes = 14

     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   1       145   233    1        2      150      0      2.3
1     67    1   4       160   286    0        2      108      1      1.5
2     67    1   4       120   229    0        2      129      1      2.6
3     37    1   3       130   250    0        0      187      0      3.5
4     41    0   2       130   204    0        2      172      0      1.4
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   45    1   1       110   264    0        0      132      0      1.2
299   68    1   4       144   193    1        0      141      0      3.4
300   57    1   4       130   131    0        0      115      1      1.2
301   57    0   2       130   236    0        2      174      0      0.0
302   38    1   3       138   175    0        0      173      0      0.0

     slope   ca  thal  num
0        3  0.0   6.0    0
1        2  3.0   3.0    2
2        2  2.0   7.0    1
3        3  0.0   3.0    0
4        1  0.0   3.0    0
..     ...  ...   ...  ...
298      2  0.0   7.0    1
299      2  2.0   7.0    2
300      2  1.0   7.0    3
301      2  1.0   3.0    1
302      1  NaN   3.0    0

[303 rows x 14 columns]
Number of duplicate rows = 0
Now discarding duplicate rows
Number of instances = 303

     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   1       145   233    1        2      150      0      2.3
1     67    1   4       160   286    0        2      108      1      1.5
2     67    1   4       120   229    0        2      129      1      2.6
3     37    1   3       130   250    0        0      187      0      3.5
4     41    0   2       130   204    0        2      172      0      1.4
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   45    1   1       110   264    0        0      132      0      1.2
299   68    1   4       144   193    1        0      141      0      3.4
300   57    1   4       130   131    0        0      115      1      1.2
301   57    0   2       130   236    0        2      174      0      0.0
302   38    1   3       138   175    0        0      173      0      0.0

     slope   ca  thal  num
0        3  0.0   6.0    0
1        2  3.0   3.0    2
2        2  2.0   7.0    1
3        3  0.0   3.0    0
4        1  0.0   3.0    0
..     ...  ...   ...  ...
298      2  0.0   7.0    1
299      2  2.0   7.0    2
```

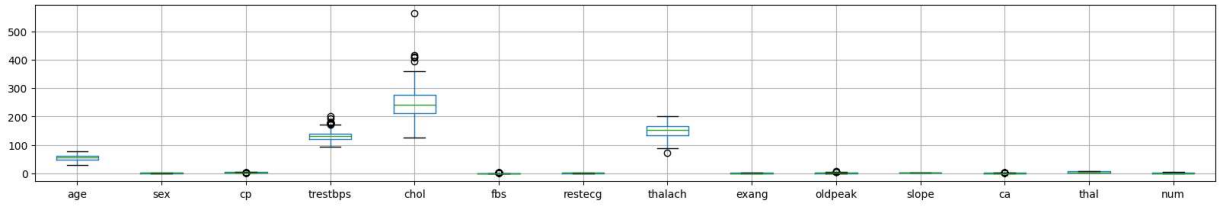
```
300      2  1.0   7.0    3
301      2  1.0   3.0    1
302      1  NaN   3.0    0

[303 rows x 14 columns]
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   1       145   233    1        2      150      0      2.3
1     67    1   4       160   286    0        2      108      1      1.5
2     67    1   4       120   229    0        2      129      1      2.6
3     37    1   3       130   250    0        0      187      0      3.5
4     41    0   2       130   204    0        2      172      0      1.4
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   45    1   1       110   264    0        0      132      0      1.2
299   68    1   4       144   193    1        0      141      0      3.4
300   57    1   4       130   131    0        0      115      1      1.2
301   57    0   2       130   236    0        2      174      0      0.0
302   38    1   3       138   175    0        0      173      0      0.0

     slope   ca  thal  num
0        3  0.0   6.0    0
1        2  3.0   3.0    2
2        2  2.0   7.0    1
3        3  0.0   3.0    0
4        1  0.0   3.0    0
..     ...  ...   ...  ...
298      2  0.0   7.0    1
299      2  2.0   7.0    2
300      2  1.0   7.0    3
301      2  1.0   3.0    1
302      1  NaN   3.0    0

[303 rows x 14 columns]
Number of missing values:
        age: 0
        sex: 0
        cp: 0
        trestbps: 0
        chol: 0
        fbs: 0
        restecg: 0
        thalach: 0
        exang: 0
        oldpeak: 0
        slope: 0
        ca: 4
        thal: 2
        num: 0

Let us fill in the missing values

Number of missing values:
        age: 0
        sex: 0
        cp: 0
        trestbps: 0
        chol: 0
```

```
fbs: 0
restecg: 0
thalach: 0
exang: 0
oldpeak: 0
slope: 0
ca: 0
thal: 0
num: 0
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
0	63	1	1	145	233	1	2	150	0	2.3	
1	67	1	4	160	286	0	2	108	1	1.5	
2	67	1	4	120	229	0	2	129	1	2.6	
3	37	1	3	130	250	0	0	187	0	3.5	
4	41	0	2	130	204	0	2	172	0	1.4	
..	...	...	..	...	...	...	...	...	...	...	
298	45	1	1	110	264	0	0	132	0	1.2	
299	68	1	4	144	193	1	0	141	0	3.4	
300	57	1	4	130	131	0	0	115	1	1.2	
301	57	0	2	130	236	0	2	174	0	0.0	
302	38	1	3	138	175	0	0	173	0	0.0	

	slope	ca	thal	num
0	3	0.0	6.0	0
1	2	3.0	3.0	2
2	2	2.0	7.0	1
3	3	0.0	3.0	0
4	1	0.0	3.0	0
..	...	...	...	...
298	2	0.0	7.0	1
299	2	2.0	7.0	2
300	2	1.0	7.0	3
301	2	1.0	3.0	1
302	1	0.0	3.0	0

```
[303 rows x 14 columns]
```



```
Number of rows before discarding outliers = 303
Number of rows after discarding outlier values = 294
          age       sex        cp  trestbps      chol       fbs   restecg  \
0    0.947160  0.685069 -2.248056  0.756274 -0.264463  2.390484  1.015005
1    1.389703  0.685069  0.876535  1.608559  0.759159 -0.416945  1.015005
2    1.389703  0.685069  0.876535 -0.664201 -0.341717 -0.416945  1.015005
3   -1.929372  0.685069 -0.164995 -0.096011  0.063869 -0.416945 -0.995103
4   -1.486829 -1.454889 -1.206525 -0.096011 -0.824558 -0.416945  1.015005
..        ...       ...       ...       ...       ...       ...       ...
298 -1.044285  0.685069 -2.248056 -1.232391  0.334260 -0.416945 -0.995103
299  1.500339  0.685069  0.876535  0.699455 -1.037008  2.390484 -0.995103
300  0.283345  0.685069  0.876535 -0.096011 -2.234453 -0.416945 -0.995103
301  0.283345 -1.454889 -1.206525 -0.096011 -0.206522 -0.416945  1.015005
302 -1.818736  0.685069 -0.164995  0.358541 -1.384653 -0.416945 -0.995103

      thalach    exang   oldpeak     slope        ca      thal       num
0    0.017169 -0.69548  1.085542  2.270822 -0.709957  0.658914 -0.762936
1   -1.818896  1.43311  0.396526  0.648041  2.500744 -0.888768  0.865019
2   -0.900864  1.43311  1.343924  0.648041  1.430510  1.174808  0.051041
3    1.634655 -0.69548  2.119067  2.270822 -0.709957 -0.888768 -0.762936
4    0.978917 -0.69548  0.310399 -0.974740 -0.709957 -0.888768 -0.762936
..        ...      ...       ...       ...       ...       ...       ...
298 -0.769716 -0.69548  0.138144  0.648041 -0.709957  1.174808  0.051041
299 -0.376274 -0.69548  2.032940  0.648041  1.430510  1.174808  0.865019
300 -1.512885  1.43311  0.138144  0.648041  0.360277  1.174808  1.678996
301  1.066349 -0.69548 -0.895381  0.648041  0.360277 -0.888768  0.051041
302  1.022633 -0.69548 -0.895381 -0.974740 -0.709957 -0.888768 -0.762936

[294 rows x 14 columns]
```

Derek Dilger

- Aggregation
- Discretization
- Principle Components Analysis
- Sampling

Aggregation

Our implementation of Aggregation

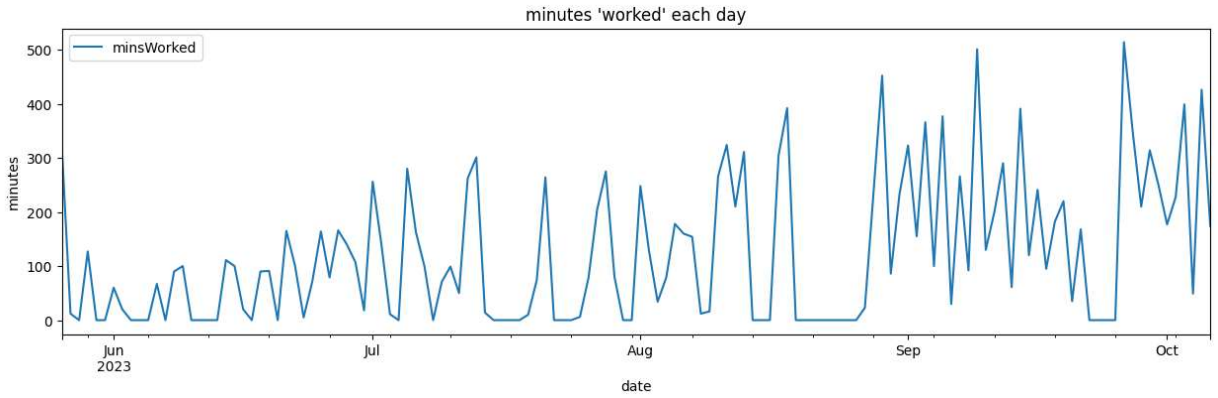
In [ ]:
```python
import pandas as pd
import numpy as np

TIME = pd.read_csv(r"data\derekswork\TIME.csv")  # obtaining our dataset
print(TIME)
```

```
          date  minsWorked
0    5/26/2023         319
1    5/27/2023          12
2    5/28/2023           0
3    5/29/2023         127
4    5/30/2023           0
..         ...         ...
129  10/2/2023         228
130  10/3/2023         399
131  10/4/2023          49
132  10/5/2023         426
133  10/6/2023         174

[134 rows x 2 columns]
```

In [ ]:
```
TIME.index = pd.to_datetime(TIME['date'])
TIME = TIME.drop(['date'], axis=1)
print(TIME)
```

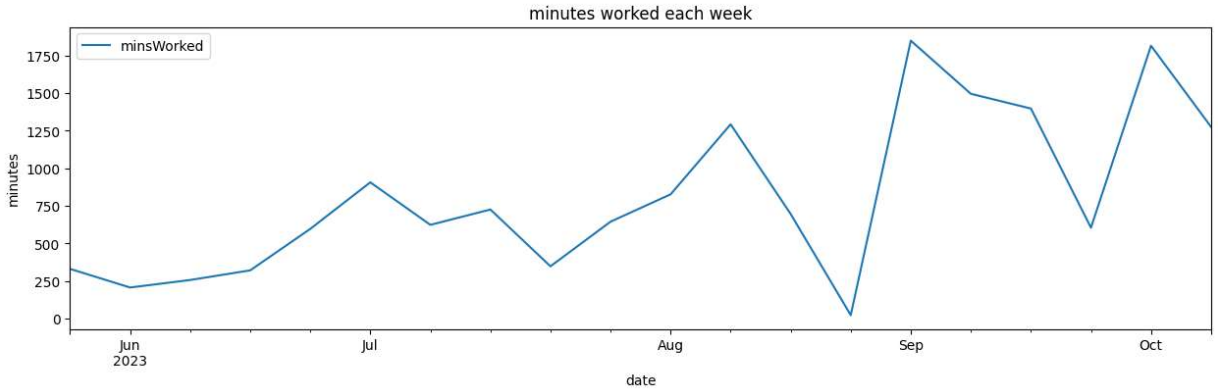
```
            minsWorked
date
2023-05-26         319
2023-05-27          12
2023-05-28           0
2023-05-29         127
2023-05-30           0
...                ...
2023-10-02         228
2023-10-03         399
2023-10-04          49
2023-10-05         426
2023-10-06         174

[134 rows x 1 columns]
```

In [ ]:
```
ax = TIME.plot(kind='line', figsize=(15, 4))
ax.set_title("minutes \'worked\' each day")
ax.set_ylabel("minutes")
ax.set_xlabel("date")
plt.show()
```



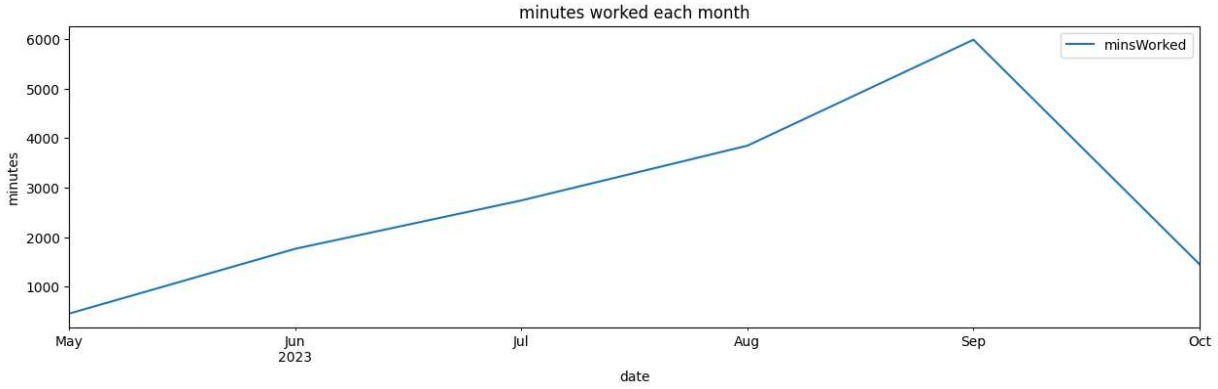
The above graph is not easy on the eyes

In [ ]:
```
weekly = TIME.groupby(pd.Grouper(freq='W')).sum()
ax = weekly.plot(kind='line', figsize=(15, 4))
ax.set_title('minutes worked each week')
ax.set_ylabel('minutes')
plt.show()
```



The above graph is easier on the eyes than 'daily'. Given that there are only 134 days in the dataset, weekly is my personal favorite. Notice the week right before the semester starts? The dip in late September is due to a family reunion.

In [ ]:
```
monthly = TIME.groupby(pd.Grouper(freq='M')).sum()
ax = monthly.plot(kind='line', figsize=(15, 4))
ax.set_title('minutes worked each month')
ax.set_ylabel('minutes')
plt.show()
```



October isn't over!

Discretization

In this example of discretization, we check winning lottery numbers from 2010 and later. Choose 5 numbers between 1 and 69, and one number between 1 and 26. This last number is the "powerball". In order to win the jackpot, you must match every number.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
powerball = pd.read_csv(
```

```
    r"data\derekswork\Lottery_Powerball_Winning_Numbers__Beginning_2010.csv")
powerball.head()
```

Out[ ]:

	Draw Date	Winning Numbers	Multiplier
0	09/26/2020	11 21 27 36 62 24	3.0
1	09/30/2020	14 18 36 49 67 18	2.0
2	10/03/2020	18 31 36 43 47 20	2.0
3	10/07/2020	06 24 30 53 56 19	2.0
4	10/10/2020	05 18 23 40 50 18	3.0

In [ ]:
```
columnOfWinNum = powerball['Winning Numbers']
print(type(columnOfWinNum))
columnOfWinNum.head()
```

```
<class 'pandas.core.series.Series'>
```

Out[ ]:
```
0    11 21 27 36 62 24
1    14 18 36 49 67 18
2    18 31 36 43 47 20
3    06 24 30 53 56 19
4    05 18 23 40 50 18
Name: Winning Numbers, dtype: object
```

In [ ]:
```
print(f"columnOfWinNum[0]: {columnOfWinNum[0]}")
print(f"Length of column win number: {len(columnOfWinNum)}")
print(
    f"demonstrating a method to isolate the \"powerball\" number as a string: {colu
```

```
columnOfWinNum[0]: 11 21 27 36 62 24
Length of column win number: 1533
demonstrating a method to isolate the "powerball" number as a string: 18
```

In [ ]:
```
listOfWinningPowerballNumbers = []  # putting all those powerballs into a list

for i in range(len(columnOfWinNum)):
    str = columnOfWinNum[i][-2:]
    listOfWinningPowerballNumbers.append(int(str))

# notice how some powerballs are above 26, even though it is impossible to win the
print(listOfWinningPowerballNumbers)
```

[24, 18, 20, 19, 18, 5, 23, 26, 6, 13, 24, 14, 18, 13, 20, 5, 11, 26, 18, 4, 10, 4,
20, 7, 13, 6, 18, 14, 5, 3, 13, 14, 2, 22, 14, 9, 4, 5, 8, 7, 20, 1, 4, 21, 18, 16,
18, 18, 2, 19, 4, 18, 6, 24, 5, 16, 3, 14, 25, 6, 22, 1, 14, 1, 4, 21, 7, 18, 10, 2,
14, 23, 3, 21, 24, 24, 24, 26, 18, 14, 9, 4, 4, 5, 6, 18, 8, 23, 5, 10, 5, 18, 18, 1
7, 14, 21, 16, 9, 11, 20, 19, 20, 24, 20, 18, 24, 19, 2, 17, 5, 4, 16, 11, 16, 13,
8, 8, 2, 23, 14, 13, 24, 11, 19, 3, 25, 6, 2, 13, 2, 18, 8, 18, 19, 3, 22, 7, 26, 1
8, 20, 14, 9, 19, 10, 25, 14, 18, 21, 23, 12, 26, 11, 25, 17, 22, 21, 4, 23, 12, 3,
23, 2, 4, 7, 4, 25, 22, 15, 11, 3, 6, 2, 18, 14, 1, 21, 20, 10, 3, 3, 13, 6, 6, 13,
19, 10, 21, 3, 1, 25, 8, 24, 21, 1, 14, 13, 18, 8, 8, 7, 2, 3, 26, 4, 26, 3, 2, 17,
6, 4, 1, 15, 10, 6, 8, 18, 4, 12, 5, 21, 11, 14, 14, 23, 21, 23, 13, 2, 1, 26, 25, 2
1, 13, 6, 10, 24, 20, 1, 24, 22, 19, 12, 25, 24, 26, 16, 17, 26, 10, 11, 26, 21, 18,
18, 5, 23, 24, 25, 11, 20, 4, 22, 23, 15, 15, 7, 15, 3, 22, 8, 18, 11, 3, 24, 21, 1
3, 9, 12, 8, 13, 16, 16, 24, 17, 2, 26, 13, 26, 1, 12, 11, 7, 22, 10, 24, 15, 25, 1
3, 9, 1, 14, 14, 7, 20, 21, 7, 25, 19, 14, 19, 17, 6, 8, 12, 19, 9, 15, 18, 20, 24,
21, 21, 24, 25, 7, 9, 22, 18, 5, 16, 6, 8, 2, 26, 20, 10, 6, 22, 7, 21, 3, 21, 24,
4, 26, 12, 24, 3, 25, 20, 11, 7, 6, 9, 6, 16, 4, 13, 26, 25, 18, 22, 13, 1, 10, 8,
4, 8, 11, 23, 7, 24, 15, 22, 19, 26, 24, 19, 22, 17, 26, 7, 15, 4, 13, 4, 19, 9, 11,
18, 12, 12, 9, 4, 17, 9, 26, 21, 16, 8, 10, 13, 21, 24, 13, 25, 6, 9, 15, 3, 9, 19,
8, 21, 9, 5, 26, 6, 15, 13, 24, 15, 13, 1, 11, 16, 9, 5, 16, 11, 4, 19, 17, 19, 2, 2
0, 19, 2, 5, 25, 10, 17, 22, 2, 12, 13, 24, 10, 4, 9, 4, 21, 3, 10, 15, 15, 20, 22,
7, 16, 3, 21, 7, 24, 2, 8, 5, 23, 24, 22, 23, 20, 16, 9, 15, 1, 1, 22, 17, 14, 14,
8, 21, 25, 12, 20, 16, 24, 8, 6, 12, 12, 2, 5, 24, 11, 5, 26, 10, 12, 18, 4, 6, 25,
17, 11, 10, 7, 20, 17, 12, 12, 15, 26, 11, 9, 23, 9, 3, 14, 17, 13, 8, 3, 22, 9, 25,
3, 19, 6, 25, 3, 21, 23, 3, 19, 6, 18, 5, 25, 25, 20, 5, 8, 23, 18, 21, 19, 24, 6, 1
0, 13, 17, 10, 22, 20, 25, 10, 6, 22, 1, 13, 9, 2, 21, 21, 6, 5, 18, 15, 17, 7, 10,
2, 11, 19, 1, 1, 9, 4, 17, 5, 5, 24, 7, 27, 8, 19, 16, 16, 12, 7, 7, 24, 28, 15, 19,
15, 35, 2, 11, 33, 11, 3, 18, 21, 25, 27, 7, 15, 7, 29, 25, 15, 11, 22, 23, 16, 25,
32, 7, 17, 30, 33, 6, 12, 18, 17, 30, 12, 29, 28, 1, 17, 14, 7, 8, 18, 27, 33, 32, 1
2, 8, 26, 22, 28, 19, 33, 22, 34, 1, 10, 23, 9, 22, 19, 10, 25, 35, 15, 22, 5, 15, 3
3, 10, 14, 26, 27, 35, 35, 17, 28, 11, 15, 34, 17, 16, 12, 16, 19, 20, 28, 18, 31,
7, 13, 26, 16, 23, 27, 13, 18, 35, 27, 19, 31, 3, 9, 24, 24, 24, 29, 29, 16, 3, 9, 3
5, 26, 14, 11, 35, 35, 26, 7, 30, 23, 15, 24, 1, 25, 18, 31, 22, 33, 1, 34, 26, 32,
20, 12, 21, 8, 20, 29, 29, 24, 17, 21, 15, 14, 9, 10, 30, 19, 13, 2, 4, 34, 3, 1, 2
9, 24, 7, 20, 2, 29, 25, 25, 34, 22, 14, 23, 18, 32, 8, 1, 34, 27, 17, 25, 29, 17, 1
0, 7, 10, 9, 32, 22, 17, 10, 29, 13, 5, 28, 4, 18, 5, 17, 5, 23, 13, 19, 20, 33, 24,
11, 20, 13, 19, 23, 23, 20, 13, 32, 15, 5, 6, 7, 6, 17, 6, 27, 11, 4, 16, 33, 28, 1
7, 1, 32, 19, 32, 14, 2, 25, 26, 11, 32, 16, 27, 17, 18, 5, 2, 20, 28, 33, 20, 24,
3, 27, 16, 31, 15, 16, 29, 12, 1, 16, 21, 29, 17, 29, 29, 3, 12, 34, 16, 18, 9, 35,
35, 7, 26, 25, 20, 14, 4, 11, 3, 14, 28, 21, 12, 3, 6, 34, 26, 20, 13, 32, 20, 25,
5, 33, 4, 10, 35, 18, 28, 35, 10, 8, 10, 33, 23, 1, 6, 29, 29, 22, 28, 23, 19, 1, 2
1, 1, 12, 18, 4, 13, 16, 20, 3, 13, 23, 29, 33, 22, 6, 30, 14, 26, 18, 27, 27, 33, 1
4, 14, 32, 30, 2, 22, 1, 23, 29, 33, 29, 16, 29, 2, 2, 18, 6, 27, 33, 8, 6, 30, 24,
5, 12, 35, 14, 7, 32, 5, 11, 11, 33, 7, 35, 13, 5, 7, 28, 1, 25, 24, 2, 34, 14, 19,
18, 33, 19, 9, 37, 8, 22, 28, 25, 11, 28, 8, 12, 26, 36, 38, 6, 13, 2, 1, 18, 31, 1
7, 31, 8, 26, 13, 11, 5, 24, 13, 2, 29, 14, 39, 36, 37, 4, 38, 6, 11, 19, 33, 30, 1
7, 25, 8, 6, 8, 32, 26, 12, 12, 18, 29, 8, 29, 21, 31, 11, 23, 36, 16, 16, 31, 15, 2
4, 3, 5, 19, 27, 7, 39, 11, 15, 23, 37, 27, 9, 8, 9, 3, 8, 21, 6, 10, 29, 36, 24, 1
9, 30, 5, 36, 2, 6, 23, 32, 31, 27, 24, 7, 36, 30, 24, 12, 14, 18, 23, 16, 22, 2, 3
9, 4, 1, 5, 29, 10, 33, 33, 39, 15, 34, 17, 11, 2, 35, 24, 25, 30, 30, 15, 13, 34, 3
3, 20, 38, 19, 13, 23, 26, 39, 31, 29, 30, 36, 15, 2, 27, 25, 3, 6, 4, 4, 3, 27, 25,
10, 39, 6, 30, 5, 25, 38, 10, 32, 29, 27, 20, 38, 31, 11, 20, 24, 5, 20, 15, 20, 34,
33, 4, 12, 10, 7, 9, 8, 8, 1, 15, 23, 30, 12, 32, 15, 1, 34, 4, 24, 3, 1, 8, 17, 24,
4, 8, 16, 20, 19, 26, 10, 25, 20, 1, 17, 14, 24, 8, 3, 10, 3, 11, 10, 5, 4, 14, 23,
16, 18, 26, 4, 15, 24, 26, 8, 17, 1, 24, 21, 21, 21, 25, 11, 19, 5, 19, 12, 11, 1, 1
5, 1, 5, 24, 15, 3, 12, 5, 20, 24, 17, 19, 8, 26, 9, 8, 14, 9, 23, 1, 19, 22, 17, 1
0, 17, 24, 2, 6, 24, 10, 4, 15, 1, 2, 15, 2, 19, 11, 7, 22, 17, 13, 9, 24, 13, 22, 1

```
0, 17, 16, 16, 11, 14, 15, 24, 10, 26, 7, 17, 16, 11, 3, 18, 19, 19, 12, 13, 2, 10,
6, 17, 6, 16, 24, 6, 3, 11, 18, 6, 18, 4, 26, 26, 18, 14, 8, 10, 4, 1, 8, 23, 10, 5,
18, 9, 13, 17, 11, 26, 17, 15, 22, 2, 16, 4, 7, 5, 25, 7, 25, 13, 12, 4, 16, 14, 26,
4, 13, 24, 9, 2, 21, 5, 22, 12, 5, 26, 12, 16, 11, 3, 12, 19, 2, 5, 24, 7, 9, 22, 1
4, 18, 1, 9, 25, 20, 4, 25, 25, 1, 7, 7, 14, 20, 11, 26, 22, 6, 5, 11, 6, 15, 25, 1
0, 18, 16, 24, 23, 13, 23, 20, 7, 6, 21, 11, 10, 2, 25, 3, 20, 10, 7, 14, 21, 19, 9,
25, 12, 12, 14, 8, 17, 1, 11, 22, 26, 14, 14, 5, 23, 3, 26, 22, 23, 18, 15, 5, 9, 1
0, 7, 9, 11, 2, 12, 23, 15, 7, 26, 23, 4, 16, 4, 4, 24, 4, 8, 11, 14, 5, 19, 14, 7,
17, 1, 11, 18, 14, 4, 18, 25, 11, 25, 3, 6, 8, 1, 23, 6, 12, 3, 11, 25, 14, 23, 25,
26, 4, 24, 14, 18, 20, 2, 4, 7, 4, 26, 1, 14, 19, 12, 4, 8, 3, 23, 18, 13, 20, 18, 2
1, 24, 7, 25, 14, 9, 2, 13, 5, 23, 22, 3, 26, 3, 1, 13, 16, 20, 5, 26, 4, 19, 13, 1,
25, 5, 18, 5, 9, 23, 21, 4, 7, 22, 5, 1, 19]
```

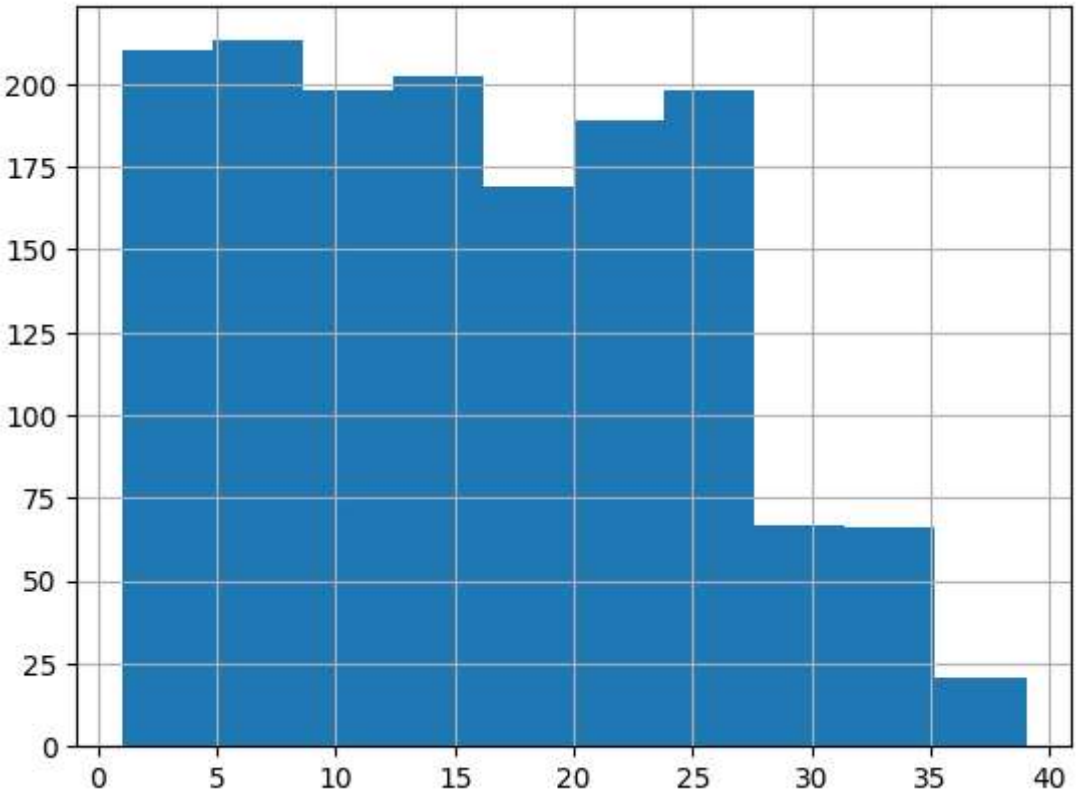
In [ ]:
```
dfOfWinningPowerballNumbers = pd.DataFrame(
    listOfWinningPowerballNumbers, columns=['num'])
dfOfWinningPowerballNumbers
```

Out[ ]:

	num
0	24
1	18
2	20
3	19
4	18
...	...
1528	7
1529	22
1530	5
1531	1
1532	19

1533 rows × 1 columns

In [ ]:
```
dfOfWinningPowerballNumbers['num'].hist(bins=10)
plt.show()
```



Principle Components Analysis

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

numImages = 8

fig = plt.figure(figsize=(7, 7))
imgData = np.zeros(shape=(numImages, 270000))

# walk through the images and add them to the plot in path data\dereks work\animals
for i, file in enumerate(os.listdir(r"data\dereks work\animals")):
    # if less than the number of images in folder
    if i < numImages:
        img = mpimg.imread(r"data\dereks work\animals\\" + file)
        imgData[i, :] = img.flatten()
        ax = fig.add_subplot(2, 4, i+1)
        ax.imshow(img)
        ax.set_title(file)
        ax.axis('off')
        imgplot = plt.imshow(img)
    else:
        break

plt.show()
```

Picture1.jpg

Picture2.jpg

Picture3.jpg

Picture4.jpg

Picture5.jpg

Picture6.jpg

Picture7.jpg

Picture8.jpg

In [ ]:

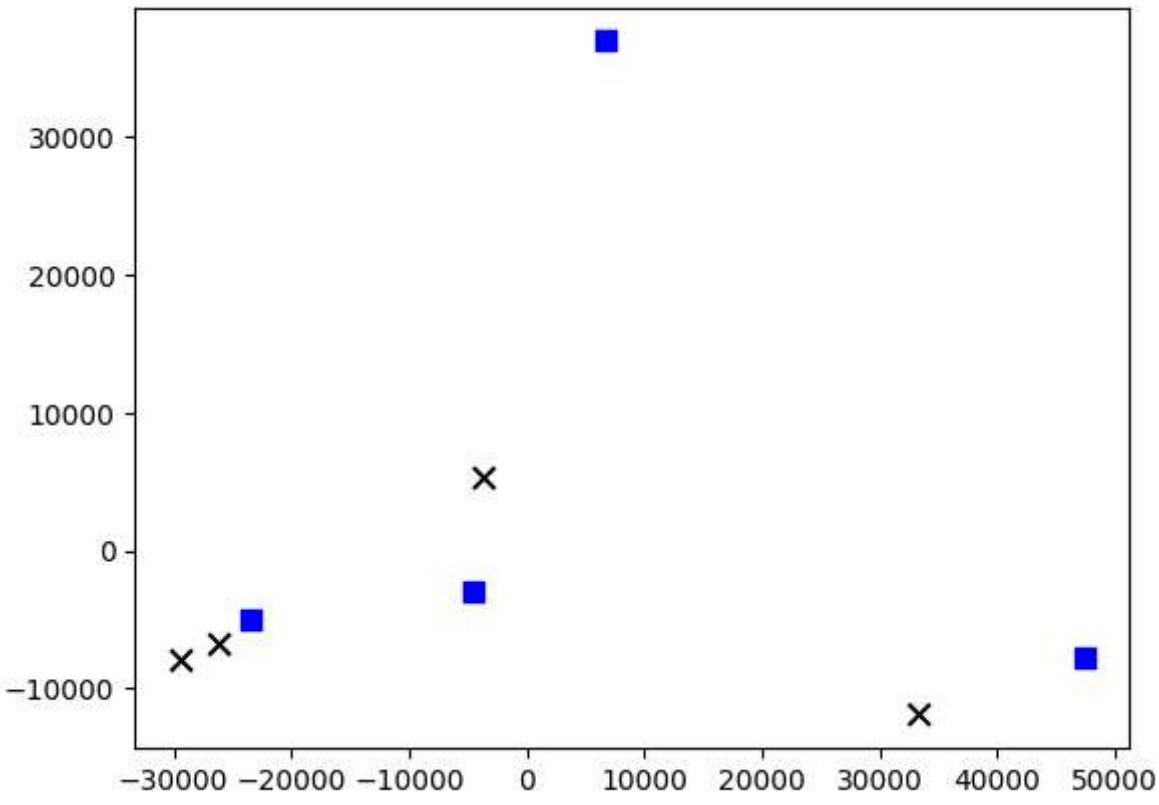
```
from sklearn.decomposition import PCA

numComponents = 2
pca = PCA(n_components=numComponents)
pca.fit(imgData)

projected = pca.transform(imgData)
projected = pd.DataFrame(
    projected, columns=['pc1', 'pc2'], index=range(1, numImages+1))
projected['animal'] = ['dog', 'dog', 'dog', 'dog', 'cat', 'cat', 'cat', 'cat']
projected
```

Out[ ]:

	pc1	pc2	animal
1	47418.908902	-7804.560588	dog
2	-4478.638904	-2924.532187	dog
3	-23518.205512	-4948.807255	dog
4	6648.144911	36919.058376	dog
5	-29466.102761	-7961.790671	cat
6	-26254.173143	-6703.680342	cat
7	-3660.149579	5312.618613	cat
8	33310.216085	-11888.305946	cat

In [ ]:

```
import matplotlib.pyplot as plt

colors = {'cat': 'k', 'dog': 'b'}
markerTypes = {'dog': 's', 'cat': 'x'}

for animalType in markerTypes:
    d = projected[projected['animal'] == animalType]
    plt.scatter(d['pc1'], d['pc2'], c=colors[animalType],
                s=60, marker=markerTypes[animalType])

plt.show()
```



Despite the above graph not providing much insight, this is a good example of when that would be the case, and a demonstration of the limits of PCA, and when it would be useful.

Some things to notice

1. The only input to the 'PCA function' (broadly speaking) was an 'array' of pixel colors. In the original example of PCA, the images had consistent color similarities between food types. The images in the example PCA seem chosen with colors in mind. However, in my implementation, the four images of cats and dogs were more randomly chosen: simply the first 4 images on an image search.
2. Do the chosen cat images and dog images contain significantly different color schemes? No. (Hence the PCA being unable to make groups of similar images effectively.)
3. One dog image and one cat image had a completely white background. Because the analysis depends on color, that would effect the location on the graph. Do you see a 'square' and an 'x' somewhat close together, but separate from the other images? I do.

Although I don't understand the math behind PCA, I feel like I learned something from this and hopefully you did too.

Sampling

Our implementation of Sampling

In [ ]:
```
import pandas as pd
import numpy as np

rain = pd.read_csv('./data/derekswork/DTW_prec.csv')

# That's a lot of data! If it was gigabytes, we could use just a sample for testing
rain
```

Out[ ]:

	DATE	PRCP
0	1/1/2001	0.00
1	1/2/2001	0.00
2	1/3/2001	0.00
3	1/4/2001	0.04
4	1/5/2001	0.14
...	...	...
6186	12/27/2017	0.00
6187	12/28/2017	0.00
6188	12/29/2017	0.00
6189	12/30/2017	0.00
6190	12/31/2017	0.00

6191 rows × 2 columns

Ian Dilger

- Testing/trining split

Training and validation

This is where we select a small segment of the data to train the remaining data on, and later validate it with

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

path = r".\data\uci_heartDisease"

filename = os.path.join(path, "uci_heartDisease_changed.csv")
df = pd.read_csv(filename, na_values=['NA', '?'])

# Showing only the first 10 rows of the table
df[0:10]
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	r
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	
5	56	1	2	120	236	0	0	178	0	0.8	1	0.0	3.0	
6	62	0	4	140	268	0	2	160	0	3.6	3	2.0	3.0	
7	57	0	4	120	354	0	0	163	1	0.6	1	0.0	3.0	
8	63	1	4	130	254	0	2	147	0	1.4	2	1.0	7.0	
9	53	1	4	140	203	1	2	155	1	3.1	3	0.0	7.0	

In [ ]:

```
# Splitting the data into the test and train sections
# The data is being trained based on the sex of the individual
x_train, x_test, y_train, y_test = train_test_split(
    df[
        ['age',
         'cp',
         'trestbps',
         'chol',
         'fbs',
         'restecg',
         'thalach',
         'exang',
         'oldpeak',
         'slope',
         'ca',
         'thal',
         'num'
        ]
    ],
    df['sex'],
    test_size=0.20,
    random_state=41
)

print(f"x_train.shape: {x_train.shape}")
print(f"y_train.shape: {y_train.shape}")
print(f"x_test.shape:  {x_test.shape}")
print(f"y_test.shape:  {y_test.shape}")
```

```
x_train.shape: (242, 13)
y_train.shape: (242,)
x_test.shape:  (61, 13)
y_test.shape:  (61,)
```

Jesus Cervantes

- Saving a dataframe
- Dropping fields
- Means and Standard Deviation

Dropping Fields

In [ ]:
```
import os
import pandas as pd
import numpy as np

path = r".\data\uci_heartDisease"

filename_read = os.path.join(path, "uci_heartDisease_changed_dropping.csv")
df = pd.read_csv(filename_read, na_values=['NA', '?'])

# Print data before dropping any columns
print("Data before dropping any columns:")
print(df.head())

# Remove the 'Unnamed: 15' column if it exists
if 'Unnamed: 15' in df.columns:
    df.drop('Unnamed: 15', axis=1, inplace=True)

# Check and drop the 'name' column if it exists
if 'name' in df.columns:
    df.drop('name', axis=1, inplace=True)
else:
    print("'name' column not found in the DataFrame")

# Print data after dropping the columns
print("\nData after dropping the unwanted columns:")
print(df.head())
```

```
Data before dropping any columns:
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0  1.0  1.0     145.0  233.0  1.0      2.0    150.0    0.0      2.3
1  67.0  1.0  4.0     160.0  286.0  0.0      2.0    108.0    1.0      1.5
2  67.0  1.0  4.0     120.0  229.0  0.0      2.0    129.0    1.0      2.6
3  37.0  1.0  3.0     130.0  250.0  0.0      0.0    187.0    0.0      3.5
4  41.0  0.0  2.0     130.0  204.0  0.0      2.0    172.0    0.0      1.4

   slope   ca  thal  num           name  Unnamed: 15
0    3.0  0.0   6.0  0.0    James Smith          NaN
1    2.0  3.0   3.0  2.0   Robert Jones          NaN
2    2.0  2.0   7.0  1.0    John Miller          NaN
3    3.0  0.0   3.0  0.0     Mary Davis          NaN
4    1.0  0.0   3.0  0.0  Emily Johnson          NaN

Data after dropping the unwanted columns:
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0  1.0  1.0     145.0  233.0  1.0      2.0    150.0    0.0      2.3
1  67.0  1.0  4.0     160.0  286.0  0.0      2.0    108.0    1.0      1.5
2  67.0  1.0  4.0     120.0  229.0  0.0      2.0    129.0    1.0      2.6
3  37.0  1.0  3.0     130.0  250.0  0.0      0.0    187.0    0.0      3.5
4  41.0  0.0  2.0     130.0  204.0  0.0      2.0    172.0    0.0      1.4

   slope   ca  thal  num
0    3.0  0.0   6.0  0.0
1    2.0  3.0   3.0  2.0
2    2.0  2.0   7.0  1.0
3    3.0  0.0   3.0  0.0
4    1.0  0.0   3.0  0.0
```

Mean and Standard Deviation

In [ ]:

```python
import os
import pandas as pd
from sklearn.model_selection import train_test_split

# Set up the file path
path = r".\data\uci_heartDisease"
filename_read = os.path.join(path, "uci_heartDisease_Spliting.csv")

# Read the CSV file into a DataFrame, handling missing values
df = pd.read_csv(filename_read, na_values=['NA', '?'])

# Split the DataFrame into two random subsets
# 50% split; you can adjust test_size as needed
df1, df2 = train_test_split(df, test_size=0.5, random_state=42)

# Calculate and print the mean and standard deviation of each subset
mean_df1 = df1.mean()
std_dev_df1 = df1.std()
mean_df2 = df2.mean()
std_dev_df2 = df2.std()

print(f"Mean of Subset 1:\n{mean_df1}")
print(f"Standard Deviation of Subset 1:\n{std_dev_df1}")
```

```
print(f"Mean of Subset 2:\n{mean_df2}")
print(f"Standard Deviation of Subset 2:\n{std_dev_df2}")
```

```
Mean of Subset 1:
age          54.960265
sex           0.635762
cp            3.158940
trestbps    132.569536
chol        244.311258
fbs           0.139073
restecg       1.013245
thalach     149.788079
exang         0.331126
oldpeak       0.956291
slope         1.562914
ca            0.651007
thal          4.604027
num           0.927152
dtype: float64
Standard Deviation of Subset 1:
age          9.383589
sex          0.482817
cp           0.931613
trestbps    18.629192
chol        45.195455
fbs          0.347174
restecg      0.999912
thalach     21.720224
exang        0.472184
oldpeak      1.190942
slope        0.606368
ca           0.936686
thal         1.920089
num          1.254854
dtype: float64
Mean of Subset 2:
age          53.921053
sex           0.723684
cp            3.157895
trestbps    130.815789
chol        249.059211
fbs           0.157895
restecg       0.967105
thalach     149.427632
exang         0.322368
oldpeak       1.122368
slope         1.638158
ca            0.693333
thal          4.861842
num           0.947368
dtype: float64
Standard Deviation of Subset 2:
age          8.682422
sex          0.448653
cp           0.990720
trestbps    16.529092
chol        57.630616
fbs          0.365848
restecg      0.992807
```

```
thalach     24.037940
exang        0.468928
oldpeak      1.128452
slope        0.625605
ca           0.940845
thal         1.956647
num          1.205888
dtype: float64
```

Matthew Mendoza & Gary Young

- Missing values
- Concatenating rows and columns

assignment01_matthew_mendoza.ipynb

I am to account for missing values and to concatenate rows and columns.

I will be using a dataset donated to UC Irvine Machine Learning Repository: Diabetes 130-US hospitals for years 1999-2008

The dataset represents a ten-year (1999-2008) clinical record of diabetes patients in 130 US hospitals, with the key goal of predicting early readmission within 30 days of discharge to address inconsistent diabetes management, which impacts both hospital costs and patient health.

Precautions, cavitates, and concerns

Some "massaging" of the data is required...

`age` values assigned are not natural numbers, but are given a range with an inclusive lower bound and an exclusive upper bound number in 10 year increments; for example, `[0-10)`, `[10-20)`, `[20-30)`, ect.

`weight`, like `age`, is also a range of numbers with an inclusive lower bound number and an exclusive upper bound number, but in 25 pounds increments (e.g `[0-25`, `[50-75)`, `[100-125)`, ...).

It should be noted that `weight` in the dataset is not prioritized; in addition, when there are records for `weight` only `Caucasian`s are those who were accounted for (excluding `Asian`, `African American`, `Hispanic`, and `other`)

For this reason I am forced to list the median weights of each category, regardless if they were male or female, of only `Caucasian` in the dataset.

No "real" analysis or extrapolation can be derived form the data produced. This is purely an exercise and demonstration of how to:

- Drop any rows with any NA values
- Replace missing values with the median value for that column
- Rows and columns can be concatenated together to form new data frames

Dataset citation

Clore,John, Cios,Krzysztof, DeShazo,Jon, and Strack,Beata. (2014). Diabetes 130-US hospitals for years 1999-2008. UCI Machine Learning Repository. https://doi.org/10.24432/C5230J.

Python dependencies

For analysis and processing of data I will be using the pandas package

Missing values

In this dataset there are missing values indicated by a question mark (`?`). The missing values that are missing are categorical in type and include the following:

- `race` : Caucasian, Asian, African American, Hispanic, and other
- `weight` : Weight in pounds
- `payer_code` : Integer identifier corresponding to 23 distinct values, for example, Blue Cross/Blue Shield, Medicare, and self-pay
- `medical_specialty` : Integer identifier of a specialty of the admitting physician, corresponding to 84 distinct values, for example, cardiology, internal medicine, family/general practice, and surgeon
- `diag_1` : The primary diagnosis (coded as first three digits of ICD9); 848 distinct values
- `diag_2` : Secondary diagnosis (coded as first three digits of ICD9); 923 distinct values
- `diag_3` : Additional secondary diagnosis (coded as first three digits of ICD9); 954 distinct values

In [ ]:
```
import pandas as pd
import os
import numpy as np

# Set the path to the data
DATA_PATH: str = os.getcwd() + \
    os.path.normpath(
        path="/data/diabetes+130-us+hospitals+for+years+1999-2008")

# Construct file paths
input_file = os.path.join(DATA_PATH, "diabetic_data.csv")
output_file = os.path.join(
    DATA_PATH, "diabetic_data_caucasian_median_weight.csv")

# Load the original CSV data
data = pd.read_csv(input_file, na_values='?')

# Filter the data to include only Caucasian patients
```

```
caucasian_data = data[data['race'] == 'Caucasian']


def calculate_median_weight(weight_range: str) -> float:
    """Calculate the median weight from a weight range or single value.

    Args:
        weight_range (str): A string containing a weight range or single value.

    Returns:
        float: The median weight calculated from the given weight range or value.
    """
    if isinstance(weight_range, str):
        bounds: list = [int(val) for val in weight_range.strip(
            '[]()').split('-') if val.isdigit()]
        if len(bounds) == 2:
            # If the value is a range, return the median
            return np.median(bounds)
    return np.nan  # Handle non-numeric values and floats as NaN


# Calculate the median for the 'weight' column
median = caucasian_data['weight'].apply(calculate_median_weight)

# Create a new DataFrame with the desired columns
output_data = caucasian_data[['encounter_id',
                              'patient_nbr', 'race', 'gender', 'age', 'weight']]

# Replace "NaN" values in the 'weight' column with the calculated median
output_data.loc[:, 'weight'] = output_data['weight'].apply(
    calculate_median_weight).fillna(median)

# Drop rows with missing values (NaN) in any column and update the DataFrame in-pla
output_data.dropna(inplace=True)

# Save the filtered data to a new CSV file
output_data.to_csv(output_file, index=False)
```

```
C:\Users\matrn\AppData\Local\Temp\ipykernel_48180\2407324495.py:16: DtypeWarning: Co
lumns (10) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv(input_file, na_values='?')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
d:\repo\csc177-group-project\assignment01_data_preprocessing_project\team12_assignment01_data_preprocessing.ipynb Cell 39 line 4
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%3D?line=36'>37</a>     return np.nan  # Handle non-numeric values and floats as NaN
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%3D?line=39'>40</a> # Calculate the median for the 'weight' column
---> <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%3D?line=40'>41</a> median = caucasian_data['weight'].apply(calculate_median_weight)
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%3D?line=42'>43</a> # Create a new DataFrame with the desired columns
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%3D?line=43'>44</a> output_data = caucasian_data[['encounter_id',
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%3D?line=44'>45</a>                                 'patient_nbr', 'race', 'gender', 'age', 'weight']]

File d:\repo\csc177-group-project\assignment01_data_preprocessing_project\.venv\Lib\site-packages\pandas\core\series.py:4760, in Series.apply(self, func, convert_dtype, args, by_row, **kwargs)
   4625 def apply(
   4626     self,
   4627     func: AggFuncType,
   (...)
   4632     **kwargs,
   4633 ) -> DataFrame | Series:
   4634     """
   4635     Invoke function on values of Series.
   4636
   (...)
   4751     dtype: float64
   4752     """
   4753     return SeriesApply(
   4754         self,
   4755         func,
   4756         convert_dtype=convert_dtype,
   4757         by_row=by_row,
   4758         args=args,
   4759         kwargs=kwargs,
-> 4760     ).apply()

File d:\repo\csc177-group-project\assignment01_data_preprocessing_project\.venv\Lib\site-packages\pandas\core\apply.py:1207, in SeriesApply.apply(self)
   1204     return self.apply_compat()
   1206 # self.func is Callable
-> 1207 return self.apply_standard()

File d:\repo\csc177-group-project\assignment01_data_preprocessing_project\.venv\Lib\site-packages\pandas\core\apply.py:1287, in SeriesApply.apply_standard(self)
```

```
   1281 # row-wise access
   1282 # apply doesn't have a `na_action` keyword and for backward compat reasons
   1283 # we need to give `na_action="ignore"` for categorical data.
   1284 # TODO: remove the `na_action="ignore"` when that default has been changed i
n
   1285 #  Categorical (GH51645).
   1286 action = "ignore" if isinstance(obj.dtype, CategoricalDtype) else None
-> 1287 mapped = obj._map_values(
   1288     mapper=curried, na_action=action, convert=self.convert_dtype
   1289 )
   1291 if len(mapped) and isinstance(mapped[0], ABCSeries):
   1292     # GH#43986 Need to do list(mapped) in order to get treated as nested
   1293     #  See also GH#25959 regarding EA support
   1294     return obj._constructor_expanddim(list(mapped), index=obj.index)

File d:\repo\csc177-group-project\assignment01_data_preprocessing_project\.venv\Lib
\site-packages\pandas\core\base.py:921, in IndexOpsMixin._map_values(self, mapper, n
a_action, convert)
    918 if isinstance(arr, ExtensionArray):
    919     return arr.map(mapper, na_action=na_action)
--> 921 return algorithms.map_array(arr, mapper, na_action=na_action, convert=conver
t)

File d:\repo\csc177-group-project\assignment01_data_preprocessing_project\.venv\Lib
\site-packages\pandas\core\algorithms.py:1814, in map_array(arr, mapper, na_action,
convert)
   1812 values = arr.astype(object, copy=False)
   1813 if na_action is None:
-> 1814     return lib.map_infer(values, mapper, convert=convert)
   1815 else:
   1816     return lib.map_infer_mask(
   1817         values, mapper, mask=isna(values).view(np.uint8), convert=convert
   1818     )

File lib.pyx:2917, in pandas._libs.lib.map_infer()

d:\repo\csc177-group-project\assignment01_data_preprocessing_project\team12_assignme
nt01_data_preprocessing.ipynb Cell 39 line 3
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=21'>22</a> def calculate_median_weight(weight_range: str) -> float:
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=22'>23</a>     """Calculate the median weight from a weight range or single
value.
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=23'>24</a>
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=24'>25</a>     Args:
   (...)
     <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=28'>29</a>         float: The median weight calculated from the given weight
range or value.
```

```
    <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=29'>30</a>     """
---> <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=30'>31</a>     if isinstance(weight_range, str):
    <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=31'>32</a>         bounds: list = [int(val) for val in weight_range.strip(
    <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=32'>33</a>             '[]()').split('-') if val.isdigit()]
    <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=33'>34</a>         if len(bounds) == 2:
    <a href='vscode-notebook-cell:/d%3A/repo/csc177-group-project/assignment01_data
_preprocessing_project/team12_assignment01_data_preprocessing.ipynb#Z2423sZmlsZQ%3D%
3D?line=34'>35</a>             # If the value is a range, return the median

TypeError: isinstance() arg 2 must be a type, a tuple of types, or a union
```

In [ ]:
```
pd.read_csv(input_file)  # Original data
```

Out[ ]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_i
0	2278392	8222157	Caucasian	Female	[0-10)	?	
1	149190	55629189	Caucasian	Female	[10-20)	?	
2	64410	86047875	AfricanAmerican	Female	[20-30)	?	
3	500364	82442376	Caucasian	Male	[30-40)	?	
4	16680	42519267	Caucasian	Male	[40-50)	?	
...	...	...	...	...	...	...	
101761	443847548	100162476	AfricanAmerican	Male	[70-80)	?	
101762	443847782	74694222	AfricanAmerican	Female	[80-90)	?	
101763	443854148	41088789	Caucasian	Male	[70-80)	?	
101764	443857166	31693671	Caucasian	Female	[80-90)	?	
101765	443867222	175429310	Caucasian	Male	[70-80)	?	

101766 rows × 50 columns

In [ ]:
```
data  # Original data (but with NaN values) - for comparison
```

Out[ ]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_i
0	2278392	8222157	Caucasian	Female	[0-10)	NaN	
1	149190	55629189	Caucasian	Female	[10-20)	NaN	
2	64410	86047875	AfricanAmerican	Female	[20-30)	NaN	
3	500364	82442376	Caucasian	Male	[30-40)	NaN	
4	16680	42519267	Caucasian	Male	[40-50)	NaN	
...	...	...	...	...	...	...	
101761	443847548	100162476	AfricanAmerican	Male	[70-80)	NaN	
101762	443847782	74694222	AfricanAmerican	Female	[80-90)	NaN	
101763	443854148	41088789	Caucasian	Male	[70-80)	NaN	
101764	443857166	31693671	Caucasian	Female	[80-90)	NaN	
101765	443867222	175429310	Caucasian	Male	[70-80)	NaN	

101766 rows × 50 columns

In [ ]:
```
# Filtered data (showing only Caucasian patients with median weight)
output_data
```

Out[ ]:

	encounter_id	patient_nbr	race	gender	age	weight

Concatenating rows and columns

Rows and columns can be concatenated together to form new data frames.

In [ ]:
```
# Create a new dataframe from patient number and weight

import os
import pandas as pd

# Set the path to the data
DATA_PATH: str = r"./data/diabetes+130-us+hospitals+for+years+1999-2008"
```

```
# Construct file paths
input_file = os.path.join(
    DATA_PATH, "diabetic_data_caucasian_median_weight.csv")
df = pd.read_csv(input_file, na_values=['NA', '?'])

column_patient_nbr = df['patient_nbr']
column_weight = df['weight']

# axis=1 for columns and axis=0 for rows
df_new = pd.concat([column_patient_nbr, column_weight], axis=1)

df_new  # New dataframe with patient number and weight only
```

Out[ ]:

	patient_nbr	weight
0	94466574	87.5
1	79874631	12.5
2	90421380	87.5
3	55628172	87.5
4	80041266	62.5
...	...	...
2900	97708986	62.5
2901	97508430	112.5
2902	62420742	62.5
2903	78208677	87.5
2904	98476776	162.5

2905 rows × 2 columns

Thomas Jaramillo-Ocha

- Calculated fields
- Feature normalization

assignment01_thomas_jaramilloochoa.ipynb

Calculated Fields

using lungCapacity.csv from https://www.kaggle.com/datasets/sulaimanahmed/lung-capacity-data/

converting weight (kg) to weight (lbs) and adding a new column

In [ ]:
```
import os
import pandas as pd
import numpy as np

path = os.getcwd() + os.path.normpath(path="/data/kaggle-lung-capacity-data")

filename = os.path.join(path, "lungcapacity.csv")

df = pd.read_csv(filename, na_values=['NaN', '?'])

df.insert(8, 'weight_lbs', (df['Weight (kg)']*2.20462).astype(float))
print(df)
```

```
     LungCap(cc)  Age( years)  Height(inches) Smoke  Gender Caesarean  \
0          6.475          6.0            62.1   NaN    male        no
1         10.125         18.0            74.7   yes  female        no
2          9.550         16.0            69.7    no  female       yes
3         11.125         14.0            71.0    no    male        no
4          4.800          5.0            56.9    no    male        no
..           ...          ...             ...   ...     ...       ...
741        9.925         16.0            68.3    no  female        no
742        8.725         19.0            68.4    no  female        no
743        7.075         11.0            66.7    no    male       yes
744        8.825         16.0            71.3   yes  female        no
745          NaN         17.0            68.8    no    male       yes

     No of children  Weight (kg)  weight_lbs
0                 3        85.70  188.935934
1                 0        98.75  217.706225
2                 0        11.01   24.272866
3                 1        29.78   65.653584
4                 4        72.84  160.584521
..              ...          ...         ...
741               3        68.30  150.575546
742               3        68.40  150.796008
743               4        66.70  147.048154
744               0        71.30  157.189406
745               2        68.80  151.677856

[746 rows x 9 columns]
```

Here we are normalizing the values so that they can be easily compared. Each Z score has been calculated for Weight (kg). This will help us determine which values are above or below average.

```
txt
LungCap(cc) Age( years) Height(inches) Smoke Gender Caesarean No of
children Weight (kg)
0 6.475 6.0 62.1 NaN male no 3 1.669226
1 10.125 18.0 74.7 yes female no 0 2.595413
2 9.550 16.0 69.7 no female yes 0 -3.631685
3 11.125 14.0 71.0 no male no 1 -2.299538
4 4.800 5.0 56.9 no male no 4 0.756524
.. ... ... ... ... ... ... ... ... ...
```

```
741 9.925 16.0 68.3 no female no 3 0.434310
742 8.725 19.0 68.4 no female no 3 0.441408
743 7.075 11.0 66.7 no male yes 4 0.320755
744 8.825 16.0 71.3 yes female no 0 0.647227
745 NaN 17.0 68.8 no male yes 2 0.469796
```

Yunjeong Lee (Luna)

- Shuffling dataframes
- Sorting dataframes

assignment01_yunjeong_lee.ipynb

Create a dataframe by reading a CSV file

In [ ]:
```
import os
import numpy as np
import pandas as pd

path = r".\data\uci_heartDisease"
filename_read = os.path.join(path, "uci_heartDisease_changed.csv")
df = pd.read_csv(filename_read, na_values=['NaN', '?'])
df
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.
...	...	...	...	...	...	...	...	...	...	...	...	...	
298	45	1	1	110	264	0	0	132	0	1.2	2	0.0	7.
299	68	1	4	144	193	1	0	141	0	3.4	2	2.0	7.
300	57	1	4	130	131	0	0	115	1	1.2	2	1.0	7.
301	57	0	2	130	236	0	2	174	0	0.0	2	1.0	3.
302	38	1	3	138	175	0	0	173	0	0.0	1	NaN	3.

303 rows × 14 columns

Shuffle the dataframe and save it

In [ ]:
```
df = df.reindex(np.random.permutation(df.index))
df
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
261	58	0	2	136	319	1	2	152	0	0.0	1	2.0	3.0
115	41	1	2	135	203	0	0	132	0	0.0	2	0.0	6.0
252	64	1	4	128	263	0	0	105	1	0.2	2	1.0	7.0
81	53	0	4	130	264	0	2	143	0	0.4	2	0.0	3.0
38	55	1	4	132	353	0	0	132	1	1.2	2	1.0	7.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
214	52	1	4	112	230	0	0	160	0	0.0	1	1.0	3.0
128	44	1	2	120	220	0	0	170	0	0.0	1	0.0	3.0
136	70	1	4	145	174	0	0	125	1	2.6	3	0.0	7.0
117	35	0	4	138	183	0	0	182	0	1.4	1	0.0	3.0
146	57	1	4	165	289	1	2	124	0	1.0	2	3.0	7.0

303 rows × 14 columns

Can use `reset_index` to reset index from 0 maintaining shuffled order

In [ ]:
```
df.reset_index(inplace=True, drop=True)
df
```

Out[ ]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	58	0	2	136	319	1	2	152	0	0.0	1	2.0	3.0
1	41	1	2	135	203	0	0	132	0	0.0	2	0.0	6.0
2	64	1	4	128	263	0	0	105	1	0.2	2	1.0	7.0
3	53	0	4	130	264	0	2	143	0	0.4	2	0.0	3.0
4	55	1	4	132	353	0	0	132	1	1.2	2	1.0	7.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	52	1	4	112	230	0	0	160	0	0.0	1	1.0	3.0
299	44	1	2	120	220	0	0	170	0	0.0	1	0.0	3.0
300	70	1	4	145	174	0	0	125	1	2.6	3	0.0	7.0
301	35	0	4	138	183	0	0	182	0	1.4	1	0.0	3.0
302	57	1	4	165	289	1	2	124	0	1.0	2	3.0	7.0

303 rows × 14 columns

In [ ]:
```
filename_write = os.path.join(path, "uci_heartDisease_changed-shuffled.csv")
df.to_csv(filename_write, index=False)
print("Saved file: {}".format(filename_write))
```

```
Saved file: .\data\uci_heartDisease\uci_heartDisease_changed-shuffled.csv
```

Sort the dataframe and save it

In [ ]:
```
print("Before sorting")
print(df['age'])  # print the age column before sorting it
df = df.sort_values(by='age', ascending=True)

print("\nAfter sorting")
print(df['age'])  # print the age column after sorting it
print("df['age'].iloc[0] is: {}".format(df['age'].iloc[0]))
print("df['age'].loc[0] is: {}".format(df['age'].loc[0]))
```

```
Before sorting
0      58
1      41
2      64
3      53
4      55
       ..
298    52
299    44
300    70
301    35
302    57
Name: age, Length: 303, dtype: int64

After sorting
246    29
252    34
68     34
160    35
301    35
       ..
109    71
129    71
25     74
126    76
255    77
Name: age, Length: 303, dtype: int64
df['age'].iloc[0] is: 29
df['age'].loc[0] is: 58
```

`iloc` gets rows (or columns) at particular positions in the index (so it only takes integers).
`loc` gets rows (or columns) with particular labels from the index.
Therefore, if you want to get a value of youngest age, you should use `.iloc[0]` after sort.

In [ ]:
```
filename_write = os.path.join(path, "uci_heartDisease_changed-sorted.csv")
# Specify index = false to not write row numbers
df.to_csv(filename_write, index=False)
print("Saved file: {}".format(filename_write))
```

```
Saved file: .\data\uci_heartDisease\uci_heartDisease_changed-sorted.csv
```