COMPSCI 389
Introduction to Machine Learning

Data Analysis

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Corresponding Jupyter Notebook

- This slide deck covers the contents of:

$$\texttt{4 Data Analysis.ipynb}$$

- Although these slides were used in lecture, for review I recommend going through the Jupyter Notebook directly.

Data Analysis

- **Data Analysis** is the process of inspecting, cleaning, transforming, and modeling data to extract useful insights, identify patterns, and support decision-making.

- Machine learning relies on data, and the first step is to perform basic data analysis in preparation for using the data with machine learning algorithms.

- Modern libraries like pandas make data analysis easy!

- This lecture will cover:
    - Using pandas to loading data sets and performing basic data analysis
    - Standard data sets that we will use for much of this course

What is pandas?

- Pandas stands for "**pan**el **da**ta", an econometric term for data sets.

- It provides two main objects, a **DataFrame** and a **Series**.
  - A DataFrame object sores a 2-dimensional table of data
  - A Series stores a 1-dimensional vector of data

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry	gpa
0	622.60	491.56	439.93	707.64	663.65	557.09	711.37	731.31	509.80	1.33333
1	538.00	490.58	406.59	529.05	532.28	447.23	527.58	379.14	488.64	2.98333
2	455.18	440.00	570.86	417.54	453.53	425.87	475.63	476.11	407.15	1.97333
3	756.91	679.62	531.28	583.63	534.42	521.40	592.41	783.76	588.26	2.53333
4	584.54	649.84	637.43	609.06	670.46	515.38	572.52	581.25	529.04	1.58667
...	...	...	...	...	...	...	...	...	...	...
43298	519.55	622.20	660.90	543.48	643.05	579.90	584.80	581.25	573.92	2.76333
43299	816.39	851.95	732.39	621.63	810.68	666.79	705.22	781.01	831.76	3.81667
43300	798.75	817.58	731.98	648.42	751.30	648.67	662.05	773.15	835.25	3.75000
43301	527.66	443.82	545.88	624.18	420.25	676.80	583.41	395.46	509.80	2.50000
43302	512.56	415.41	517.36	532.37	592.30	382.20	538.35	448.02	496.39	3.16667

43303 rows × 10 columns

```
0         1.33333
1         2.98333
2         1.97333
3         2.53333
4         1.58667
           ...
43298     2.76333
43299     3.81667
43300     3.75000
43301     2.50000
43302     3.16667
Name: gpa, Length: 43303, dtype: float64
```

What is pandas?

- Pandas provides useful functions for working with DataFrames and Series objects including functions for:
  - Loading data sets from files and storing them in DataFrame and/or Series objects.
  - Manipulating DataFrame and Series objects (e.g., adding or removing features).
  - Computing statistics of the data (e.g., the minimum and maximum values of features).
- Pandas has become so common that many other ML libraries in python are built to be compatible with it.

Installation

- The packages that we will use can be installed using pip.
- In general you can install the package called PackageName by running the following in a command prompt or terminal:

```
pip3 install PackageName
```

- For pandas the command is:

```
pip3 install pandas
```

- The Jupyter notebooks that we provide include installation instructions for the packages they use.
  - These will only be (slightly) more complicated later, when we use packages to train models on the GPU rather than CPU.

GPA Data

- Data about undergraduate students from the *Universidade Federal do Rio Grande do Sul* (UFRGS) in Brazil.

- **Input**: Scores on 9 entrance exams:
  - Physics, biology, history, English, geography, literature, Portuguese, math, chemistry.

- **Output**: GPA on a 4.0 scale during the first three semesters at university.
  - **Regression**: Predict the GPA
  - **(Binary) Classification**: Predict if the GPA is at least 3.0

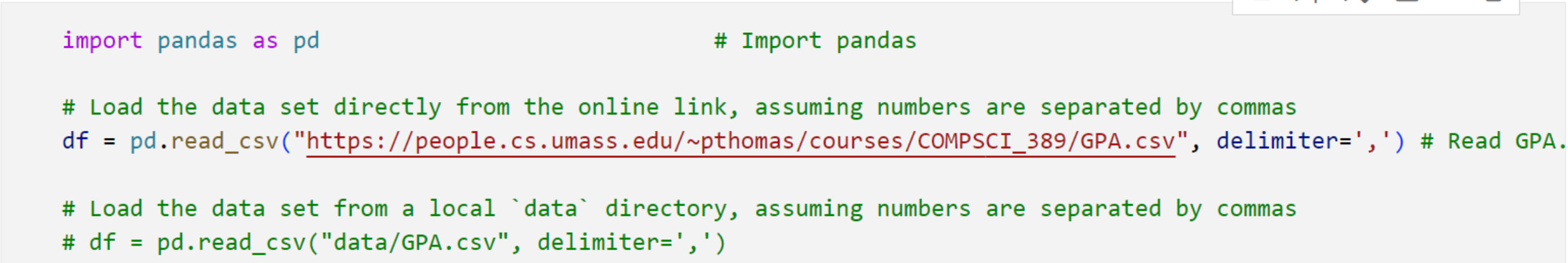- **Data Set Size**: 43,303 (rows/instances/points)

GPA Data

- You can download the GPA data set here:

  https://people.cs.umass.edu/~pthomas/courses/COMPSCI_389/GPA.csv
  - If you do, place it in a directory called `data` next to your .ipynb file.

- You can ask Pandas to fetch the CSV file from online for you!

```python
import pandas as pd                                            # Import pandas

# Load the data set directly from the online link, assuming numbers are separated by commas
df = pd.read_csv("https://people.cs.umass.edu/~pthomas/courses/COMPSCI_389/GPA.csv", delimiter=',') # Read GPA.

# Load the data set from a local `data` directory, assuming numbers are separated by commas
# df = pd.read_csv("data/GPA.csv", delimiter=',')
```

GPA Data: Printing DataFrames

- The Python `print` function works, printing the DataFrame to the console.

```
        physics   biology   history   English   geography   literature   Portuguese  \
0        622.60    491.56    439.93    707.64      663.65       557.09       711.37
1        538.00    490.58    406.59    529.05      532.28       447.23       527.58
2        455.18    440.00    570.86    417.54      453.53       425.87       475.63
3        756.91    679.62    531.28    583.63      534.42       521.40       592.41
4        584.54    649.84    637.43    609.06      670.46       515.38       572.52
...         ...       ...       ...       ...         ...          ...          ...
43298    519.55    622.20    660.90    543.48      643.05       579.90       584.80
43299    816.39    851.95    732.39    621.63      810.68       666.79       705.22
43300    798.75    817.58    731.98    648.42      751.30       648.67       662.05
43301    527.66    443.82    545.88    624.18      420.25       676.80       583.41
43302    512.56    415.41    517.36    532.37      592.30       382.20       538.35
```

`print(df)`

```
          math   chemistry       gpa
0       731.31      509.80   1.33333
1       379.14      488.64   2.98333
2       476.11      407.15   1.97333
3       783.76      588.26   2.53333
4       581.25      529.04   1.58667
...        ...         ...       ...
43298   581.25      573.92   2.76333
43299   781.01      831.76   3.81667
43300   773.15      835.25   3.75000
43301   395.46      509.80   2.50000
43302   448.02      496.39   3.16667

[43303 rows x 10 columns]
```

GPA Data: Displaying DataFrames

- Jupyter Notebooks have a function `display` that is like `print`.
  - Outputs HTML that can then be rendered nicely as a table.

`display(df)`

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry	gpa
0	622.60	491.56	439.93	707.64	663.65	557.09	711.37	731.31	509.80	1.33333
1	538.00	490.58	406.59	529.05	532.28	447.23	527.58	379.14	488.64	2.98333
2	455.18	440.00	570.86	417.54	453.53	425.87	475.63	476.11	407.15	1.97333
3	756.91	679.62	531.28	583.63	534.42	521.40	592.41	783.76	588.26	2.53333
4	584.54	649.84	637.43	609.06						
...	...	...	...	...						
43298	519.55	622.20	660.90	543.48						
43299	816.39	851.95	732.39	621.63						
43300	798.75	817.58	731.98	648.42						
43301	527.66	443.82	545.88	624.18	420.25	676.80	583.41	395.46	509.80	2.50000
43302	512.56	415.41	517.36	532.37	592.30	382.20	538.35	448.02	496.39	3.16667

Is each column numerical, categorical, text, or an image? Continuous, discrete, nominal, or ordinal?

If the GPAs were binned into letter grades, A, B, C, ..., F, would that change?

Notice that pandas views this as a **table** with **rows** and **columns**. Hence features *and* labels are viewed as "columns" when using pandas.

43303 rows × 10 columns

Manipulating DataFrames

- DataFrames have many built-in functions to facilitate manipulating and accessing data.
- **`iloc`**: Integer-location based indexing for selection by position
  - The first argument contains the rows to retrieve and the second contains the columns.
  - A = df.iloc[3,7]
    - Get the entry in the 4th row and 8th column.
    - This returns a numpy.float64 object.
  - B = df.iloc[0:3, :]
    - Get the first three rows (0, 1, and 2) and all the columns.
    - When returning a table of values (as in this example), `iloc` returns a new DataFrame object.
  - X = df.iloc[:, :-1]
    - Get all rows, and all but the last column.
  - y = df.iloc[:, -1]
    - Get all rows, but only the last column.
    - When returning a single list of values (as in this example), `iloc` returns a Series object.

Not rows 0,1,2, and 3!

We will use these two often, separating the features from the labels

```
display(X)
```

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry
0	622.60	491.56	439.93	707.64	663.65	557.09	711.37	731.31	509.80
1	538.00	490.58	406.59	529.05	532.28	447.23	527.58	379.14	488.64
2	455.18	440.00	570.86	417.54	453.53	425.87	475.63	476.11	407.15
3	756.91	679.62	531.28	583.63	534.42	521.40	592.41	783.76	588.26
4	584.54	649.84	637.43	609.06	670.46	515.38	572.52	581.25	529.04
...	...	...	...	...	...	...	...	...	...
43298	519.55	622.20	660.90	543.48	643.05	579.90	584.80	581.25	573.92
43299	816.39	851.95	732.39	621.63	810.68	666.79	705.22	781.01	831.76
43300	798.75	817.58	731.98	648.42	751.30	648.67	662.05	773.15	835.25
43301	527.66	443.82	545.88	624.18	420.25	676.80	583.41	395.46	509.80
43302	512.56	415.41	517.36	532.37	592.30	382.20	538.35	448.02	496.39

43303 rows × 9 columns

```
display(y)
```

```
0          1.33333
1          2.98333
2          1.97333
3          2.53333
4          1.58667
            ...
43298      2.76333
43299      3.81667
43300      3.75000
43301      2.50000
43302      3.16667
Name: gpa, Length: 43303, dtype: float64
```

Indexing by Column Names

```
display(df['physics'])
```

```
0          622.60
1          538.00
2          455.18
3          756.91
4          584.54
           ...
43298      519.55
43299      816.39
43300      798.75
43301      527.66
43302      512.56
Name: physics, Length: 43303, dtype: float64
```

Adding Columns

```python
df['STEM Total'] = df['physics'] + df['biology'] + df['math'] + df['chemistry'];
df['Non-STEM Total'] = df['history'] + df['English'] + df['geography'] + df['literature'] + df['Portuguese'];
display(df);
```

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry	gpa	STEM Total	Non-STEM Total
0	622.60	491.56	439.93	707.64	663.65	557.09	711.37	731.31	509.80	1.33333	2355.27	3079.68
1	538.00	490.58	406.59	529.05	532.28	447.23	527.58	379.14	488.64	2.98333	1896.36	2442.73
2	455.18	440.00	570.86	417.54	453.53	425.87	47				1778.44	2343.43
3	756.91	679.62	531.28	583.63	534.42	521.40	59				2808.55	2763.14
4	584.54	649.84	637.43	609.06	670.46	515.38	572.52	581.25	529.04	1.58667	2344.67	3004.85
...	...	...	...	...	...	...	...	...	...	...	...	...
43298	519.55	622.20	660.90	543.48	643.05	579.90	584.80	581.25	573.92	2.76333	2296.92	3012.13
43299	816.39	851.95	732.39	621.63	810.68	666.79	705.22	781.01	831.76	3.81667	3281.11	3536.71
43300	798.75	817.58	731.98	648.42	751.30	648.67	662.05	773.15	835.25	3.75000	3224.73	3442.42
43301	527.66	443.82	545.88	624.18	420.25	676.80	583.41	395.46	509.80	2.50000	1876.74	2850.52
43302	512.56	415.41	517.36	532.37	592.30	382.20	538.35	448.02	496.39	3.16667	1872.38	2562.58

Note that the label is now in the middle of the table!

43303 rows × 12 columns

Remove a column with `pop`

```
df['gpa'] = df.pop('gpa');
display(df)
```

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry	STEM Total	Non-STEM Total	gpa
0	622.60	491.56	439.93	707.64	663.65	557.09	711.37	731.31	509.80	2355.27	3079.68	1.33333
1	538.00	490.58	406.59	529.05	532.28	447.23	527.58	379.14	488.64	1896.36	2442.73	2.98333
2	455.18	440.00	570.86	417.54	453.53	425.87	475.63	476.11	407.15	1778.44	2343.43	1.97333
3	756.91	679.62	531.28	583.63	534.42	521.40	592.41	783.76	588.26	2808.55	2763.14	2.53333
4	584.54	649.84	637.43	609.06	670.46	515.38	572.52	581.25	529.04	2344.67	3004.85	1.58667
...	...	...	...	...	...	...	...	...	...	...	...	...
43298	519.55	622.20	660.90	543.48	643.05	579.90	584.80	581.25	573.92	2296.92	3012.13	2.76333
43299	816.39	851.95	732.39	621.63	810.68	666.79	705.22	781.01	831.76	3281.11	3536.71	3.81667
43300	798.75	817.58	731.98	648.42	751.30	648.67	662.05	773.15	835.25	3224.73	3442.42	3.75000
43301	527.66	443.82	545.88	624.18	420.25	676.80	583.41	395.46	509.80	1876.74	2850.52	2.50000
43302	512.56	415.41	517.36	532.37	592.30	382.20	538.35	448.02	496.39	1872.38	2562.58	3.16667

43303 rows × 12 columns

The `describe` function

```
display(df.describe())
```

- `describe`: Computes descriptive statistics of the DataFrame and stores them in a new DataFrame object.

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry
count	43303.000000	43303.000000	43303.000000	43303.000000	43303.000000	43303.000000	43303.000000	43303.000000	43303.000000
mean	576.122666	568.660142	580.826884	573.993046	574.494550	583.302079	551.041012	579.195005	571.711380
std	115.153301	101.443621	94.214466	86.872201	90.757127	92.895977	87.149764	114.682663	112.170338
min	299.340000	262.990000	265.020000	222.710000	224.870000	239.110000	151.590000	297.990000	300.470000
25%	482.790000	492.400000	516.100000	517.700000	510.230000	516.770000	491.880000	489.410000	484.540000
50%	565.610000	566.440000	578.940000	580.280000	575.470000	587.070000	553.570000	571.890000	565.510000
75%	662.800000	634.780000	650.190000	640.560000	637.270000	648.670000	613.060000	665.160000	655.420000
max	952.090000	966.570000	925.760000	858.440000	941.840000	904.770000	825.530000	1072.120000	1001.900000

Transposition of a Table or Matrix

```
display(df.describe().T);
```

- Transposition flips the rows and columns.
- The $\mathbb{T}$ function performs transposition on DataFrames.

	count	mean	std	min	25%	50%	75%	max
physics	43303.0	576.122666	115.153301	299.34	482.790	565.61	662.800	952.09
biology	43303.0	568.660142	101.443621	262.99	492.400	566.44	634.780	966.57
history	43303.0	580.826884	94.214466	265.02	516.100	578.94	650.190	925.76
English	43303.0	573.993046	86.872201	222.71	517.700	580.28	640.560	858.44
geography	43303.0	574.494550	90.757127	224.87	510.230	575.47	637.270	941.84
literature	43303.0	583.302079	92.895977	239.11	516.770	587.07	648.670	904.77
Portuguese	43303.0	551.041012	87.149764	151.59	491.880	553.57	613.060	825.53
math	43303.0	579.195005	114.682663	297.99	489.410	571.89	665.160	1072.12
chemistry	43303.0	571.711380	112.170338	300.47	484.540	565.51	655.420	1001.90
STEM Total	43303.0	2295.689193	382.953878	1439.80	1986.405	2263.72	2562.520	3700.08
Non-STEM Total	43303.0	2863.657572	328.000253	1840.05	2640.610	2865.60	3087.245	3891.23
gpa	43303.0	2.785727	0.820757	0.00	2.280	2.92	3.430	4.00

```
# Add additional statistics like the variance and skewness:
statistics = df.describe().T;
statistics['variance'] = df.var()                          # Get the variance of each column
statistics['skew'] = df.skew()                             # Get the skew of each column

# Notice that we
display(statistics
```

- DataFrames have *many* other built-in functions

- Here we show variance and skew

Negative skew

Positive skew
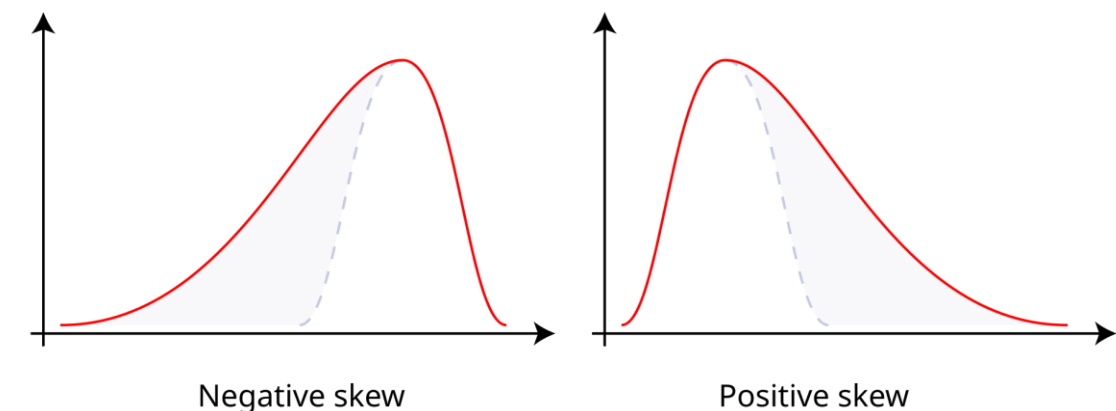
	mean	50%	variance	skew
physics	576.122666	565.61	13260.282681	0.385449
biology	568.660142	566.44	10290.808255	0.294475
history	580.826884	578.94	8876.365696	0.016084
English	573.993046	580.28	7546.779298	-0.258281
geography	574.494550	575.47	8236.856029	-0.017728
literature	583.302079	587.07	8629.662535	-0.111834
Portuguese	551.041012	553.57	7595.081410	-0.160546
math	579.195005	571.89	13152.113255	0.304221
chemistry	571.711380	565.51	12582.184723	0.358615
gpa	2.785727	2.92	0.673641	-0.822257

Matplotlib – Visualization with Python

- We will use matplotlib for most of our plotting.
- You can install matplotlib by running the following command in the command line or terminal:

```
pip install matplotlib
```

```python
import matplotlib.pyplot as plt

plt.hist(df['gpa'], bins=20, color='blue', edgecolor='black', alpha=0.7)
plt.title('Histogram of GPAs')
plt.xlabel('GPA')
plt.ylabel('Frequency')
plt.show()
```

- We often use **historgrams** to visualize the distributions of features, labels, or other statistics.

Histogram of GPAs
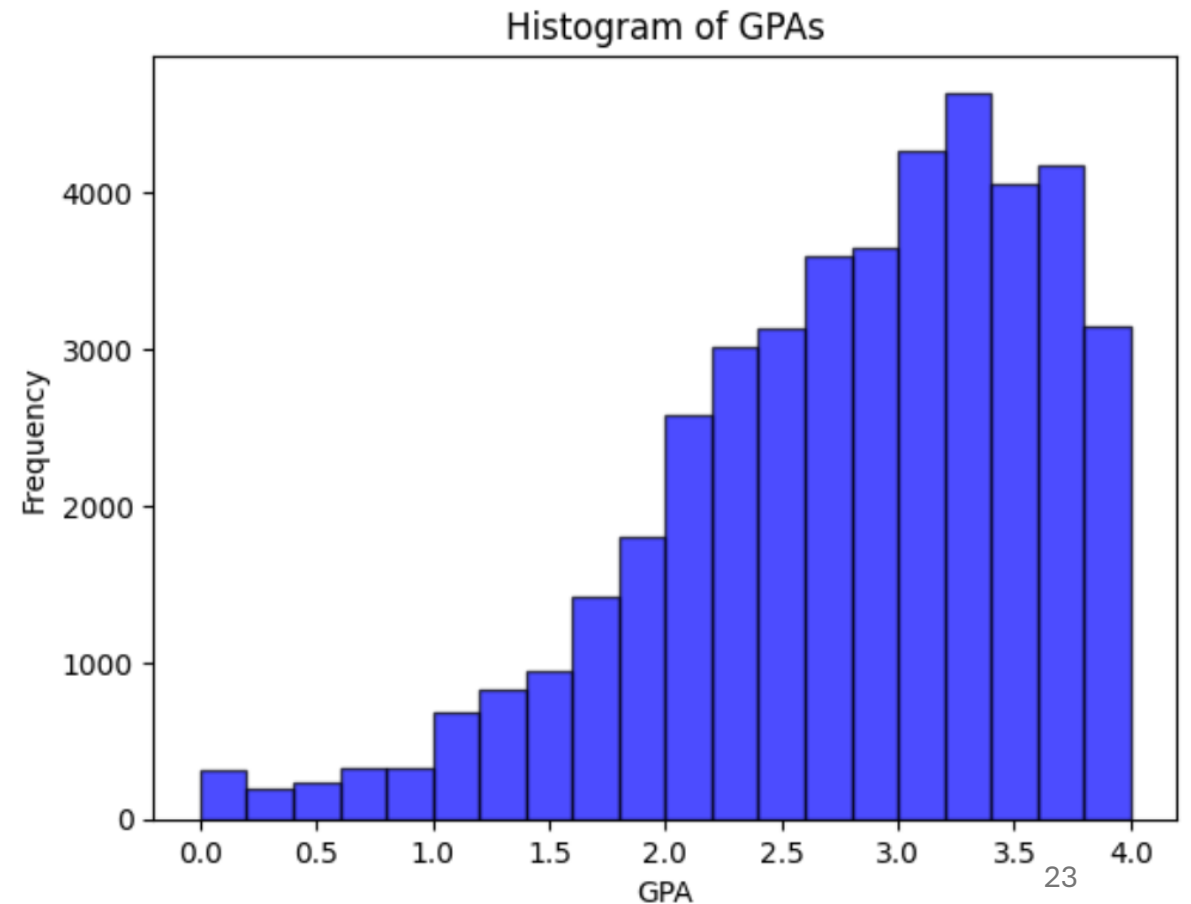
```python
df['gpa'].plot(kind='density')
plt.title('Distribution of GPAs')    # Add a title
plt.xlabel('GPA')                     # Add a horizontal label
plt.xlim(0, 4.0)                      # Set the x-axis limit
plt.ylabel('Density')                 # Add a veritcal label
plt.show()                            # Show the plot
```

- For continuous features we sometimes use **density** plots, which estimate the *probability density function* (PDF).

Distribution of GPAs
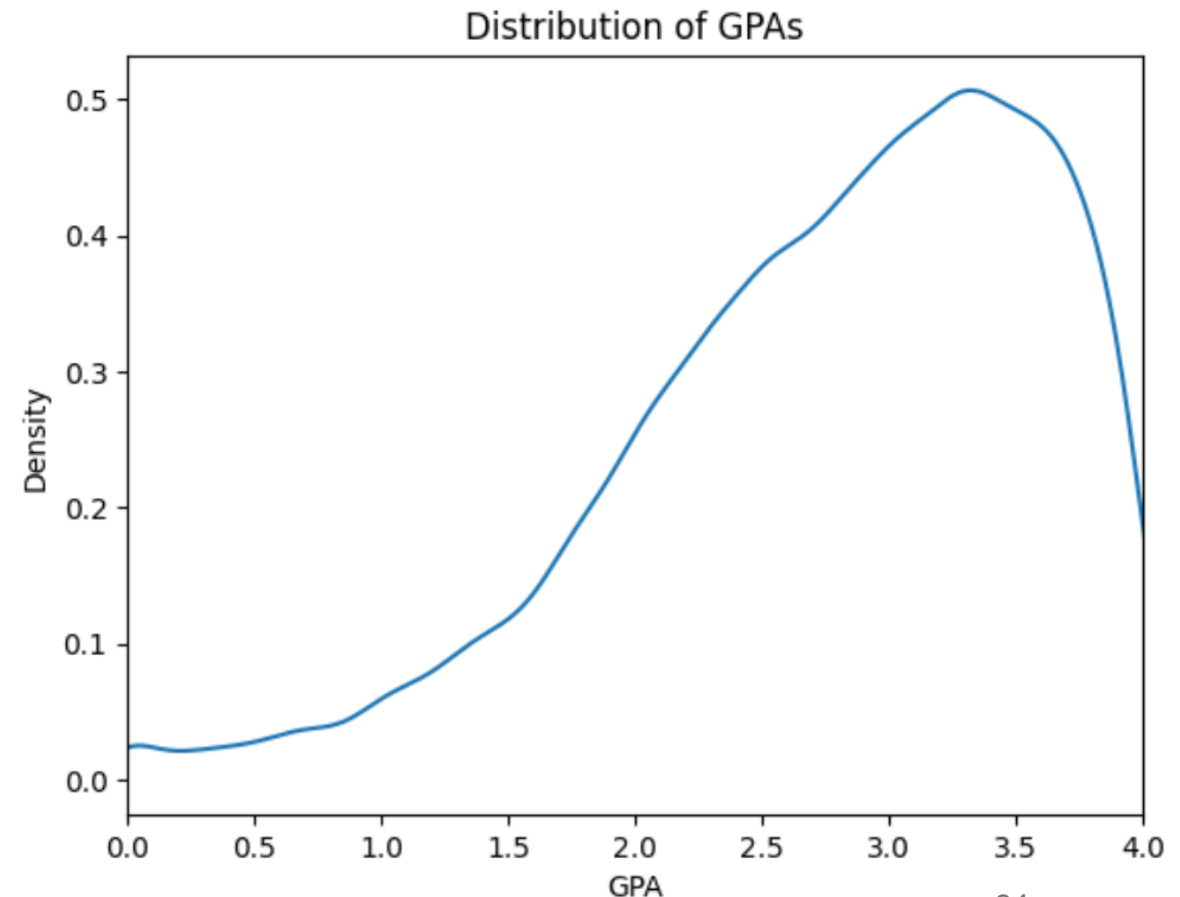
Visualizing pairs of features

- Histograms and density plots typically only show a single feature or column.

- You can visualize pairs of features with scatter plots.

- As an example, let's investigate how the scores on STEM exams and the scores on non-STEM exams relate to GPA.

- Recall:

```
df['STEM Total'] = df['physics'] + df['biology'] + df['math'] + df['chemistry'];
df['Non-STEM Total'] = df['history'] + df['English'] + df['geography'] + df['literature'] + df['Portuguese'];
```

	physics	biology	history	English	geography	literature	Portuguese	math	chemistry	STEM Total	Non-STEM Total	gpa
0	622.60	491.56	439.93	707.64	663.65	557.09	711.37	731.31	509.80	2355.27	3079.68	1.33333
1	538.00	490.58	406.59	529.05	532.28	447.23	527.58	379.14	488.64	1896.36	2442.73	2.98333

```
# Plot STEM Total versus GPA
plt.scatter(df['STEM Total'], df['gpa'], s=10, alpha=0.05)
plt.title('STEM Total vs GPA')
plt.xlabel('STEM Total')
plt.ylabel('GPA')
plt.show()
```
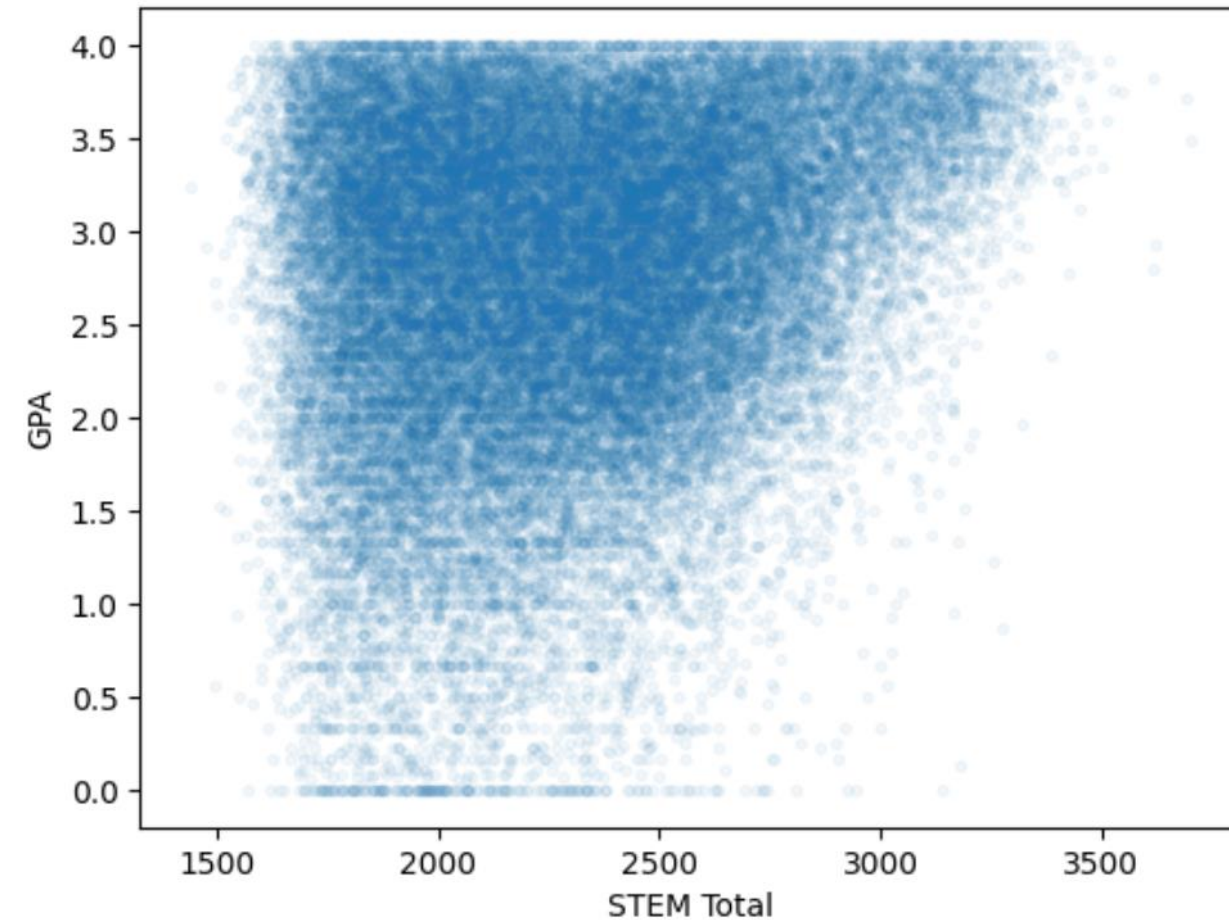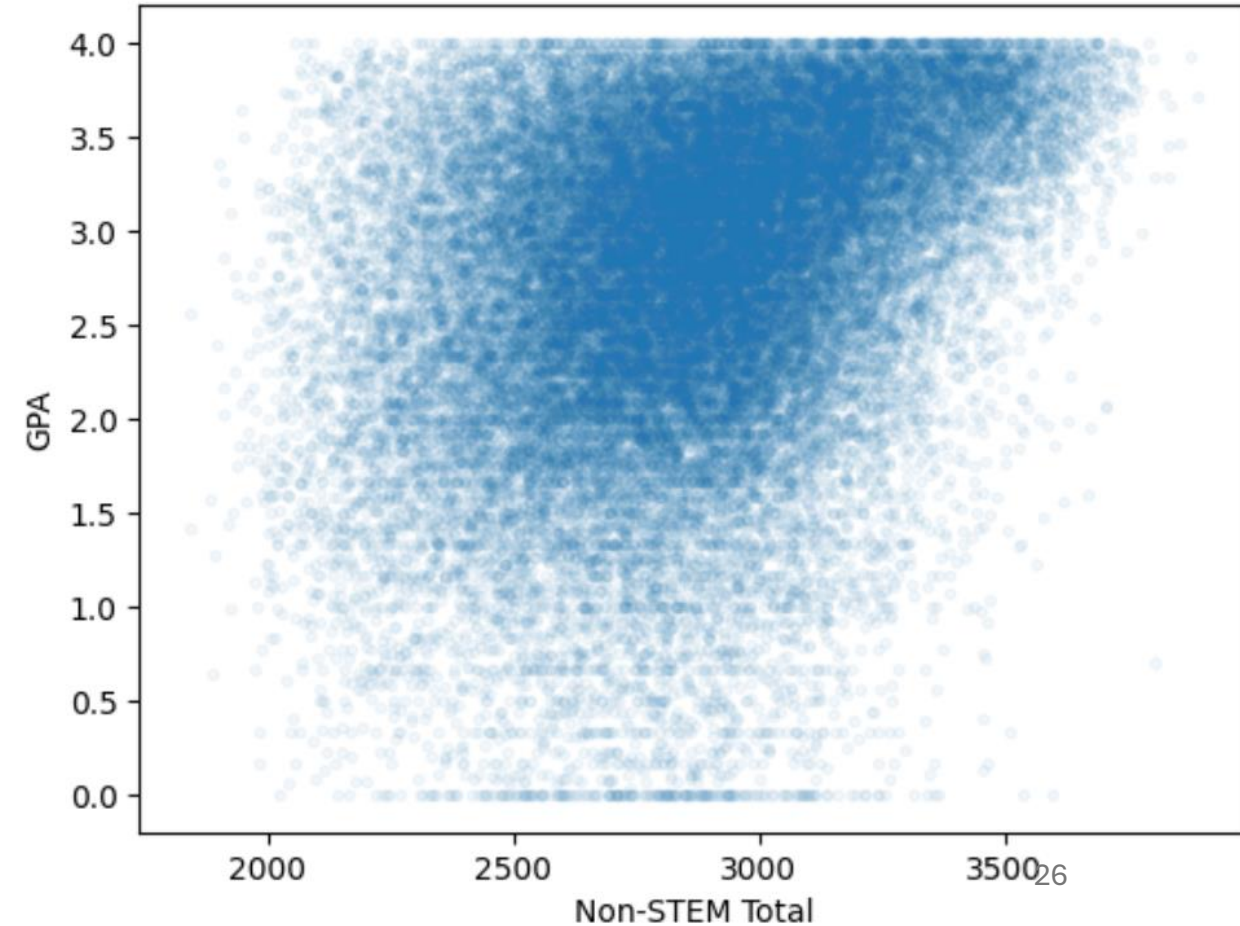
Do you see any patterns or trends here?

Note the different axis scales!
(4 STEM exams, 5 non-STEM)

STEM Total vs GPA

Non-STEM Total vs GPA

Pearson Correlation Coefficient

- Let's *quantify* the correlation between STEM and non-STEM exam scores and GPA.

- The **Pearson correlation coefficient** is a number that quantifies how correlated two quantities are.
  - $= 0$: Not correlated
  - > 0: Correlated. When one increases the other tends to increase
  - < 0: Correlated. When one increases the other tends to decrease
  - Always in $[-1,1]$.

Pearson Correlation Coefficient
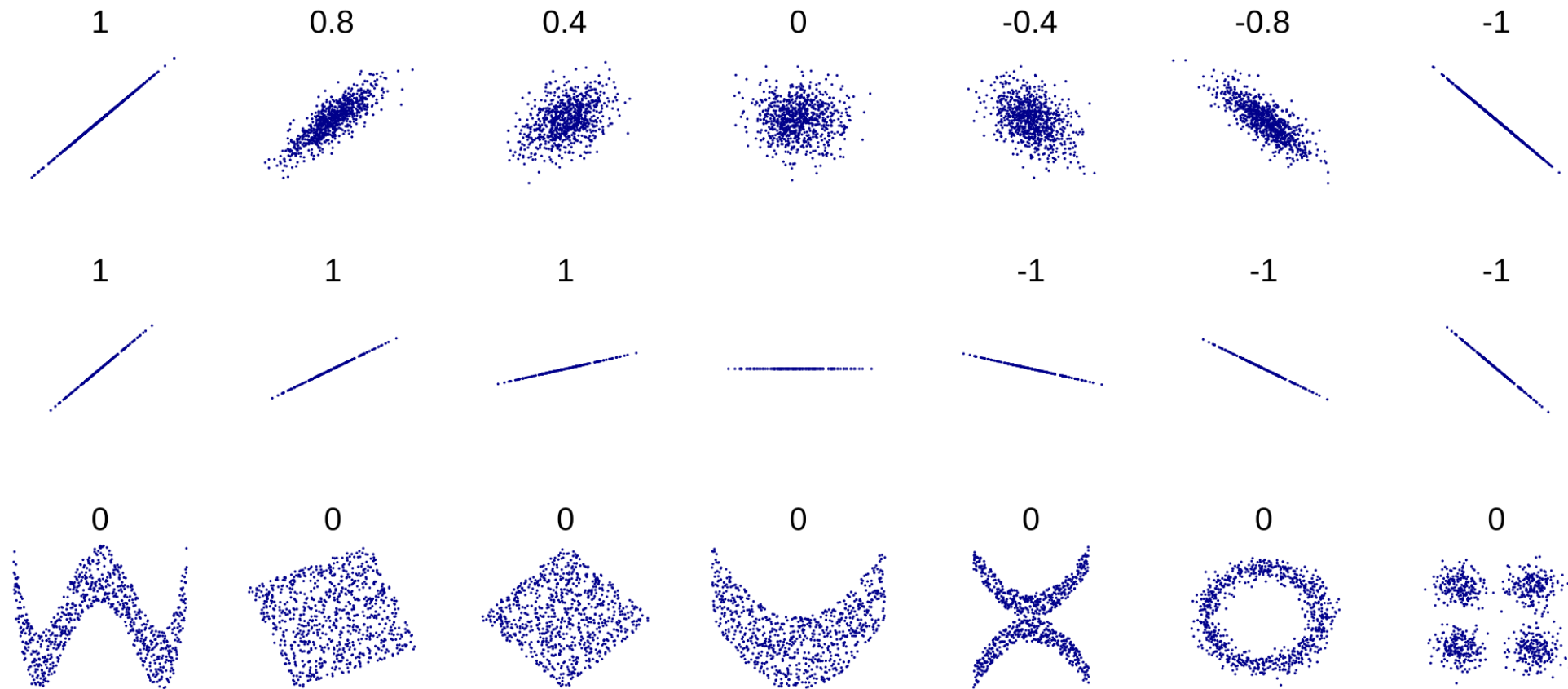
Pearson Correlation Coefficient

$$\rho_{X,Y} = \frac{\mathbf{E}\big[(X - \mu_x)(Y - \mu_y)\big]}{\sigma_X \sigma_Y}$$

- σ_X and σ_Y are the standard deviations of random variables X and Y.

- μ_X and μ_Y are the means of X and Y.

- Numerator:
    - Positive if one being larger than its mean implies the other tends to be as well.
    - Negative if one being larger than its mean implies the other tends to be smaller than its mean.

- Denominator:
    - Rescales to $[-1,1]$, accounting for how much each tends to vary from its mean

Pearson Correlation Coefficient

$$\rho_{X,Y} = \frac{\mathbf{E}\big[(X - \mu_x)(Y - \mu_y)\big]}{\sigma_X \sigma_Y}$$

- The means and standard deviations are not usually known, but can be estimates from n *independent and identically distributed* (i.i.d.) samples of (X, Y).
  - $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
- The sample Person correlation coefficient is then:

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Computing the Correlation Coefficient

- DataFrame objects have a `corr()` function that computes the correlation coefficient for you!

```python
stem_gpa_correlation = df['STEM Total'].corr(df['gpa'])
non_stem_gpa_correlation = df['Non-STEM Total'].corr(df['gpa'])

# Print the two correlations
print(f"Correlation between STEM Total and GPA: {stem_gpa_correlation}")
print(f"Correlation between Non-STEM Total and GPA: {non_stem_gpa_correlation}")
```

```
Correlation between STEM Total and GPA: 0.21260938426832135
Correlation between Non-STEM Total and GPA: 0.3312547367642783
```

Q: What can you conclude from these correlations?
A: The cumulative non-STEM exam score is a stronger predictor of GPA than the cumulative STEM exam score
Note: There are more non-STEM exams!

31

Idea: Compute the correlation between all pairs of columns!

- With 12 columns, there are 144 correlations to compute.
- We could loop over pairs of columns and print the correlations manually.
  - Would be hard to understand a list of 144 numbers!
- Solution:
  - Use pandas to compute **correlation matrix**, which contains the correlations between all pairs of columns.
  - Use the `seaborn` package to plot the matrix as a heatmap.
    - Install with: `pip install seaborn`

Calling `corr()` with no arguments defaults to computing Pearson's correlation coefficient (other options exist)

```python
import seaborn as sns

correlation_matrix = df.corr()

# Plot the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Columns')
plt.show()
```

Creates 12" by 8" figure

`annot=True` displays numbers in each cell

- Which columns has the highest correlation with GPA?
  - Is this surprising?
- Why is the correlation between physics and STEM Total so high?
- Why do you think the correlation between physics and STEM Total is higher than the correlation between math and STEM Total?
- Which columns have the lowest correlation? Is this surprising?
- Is it surprising that no columns have negative correlations?

Correlation Matrix of Columns

34
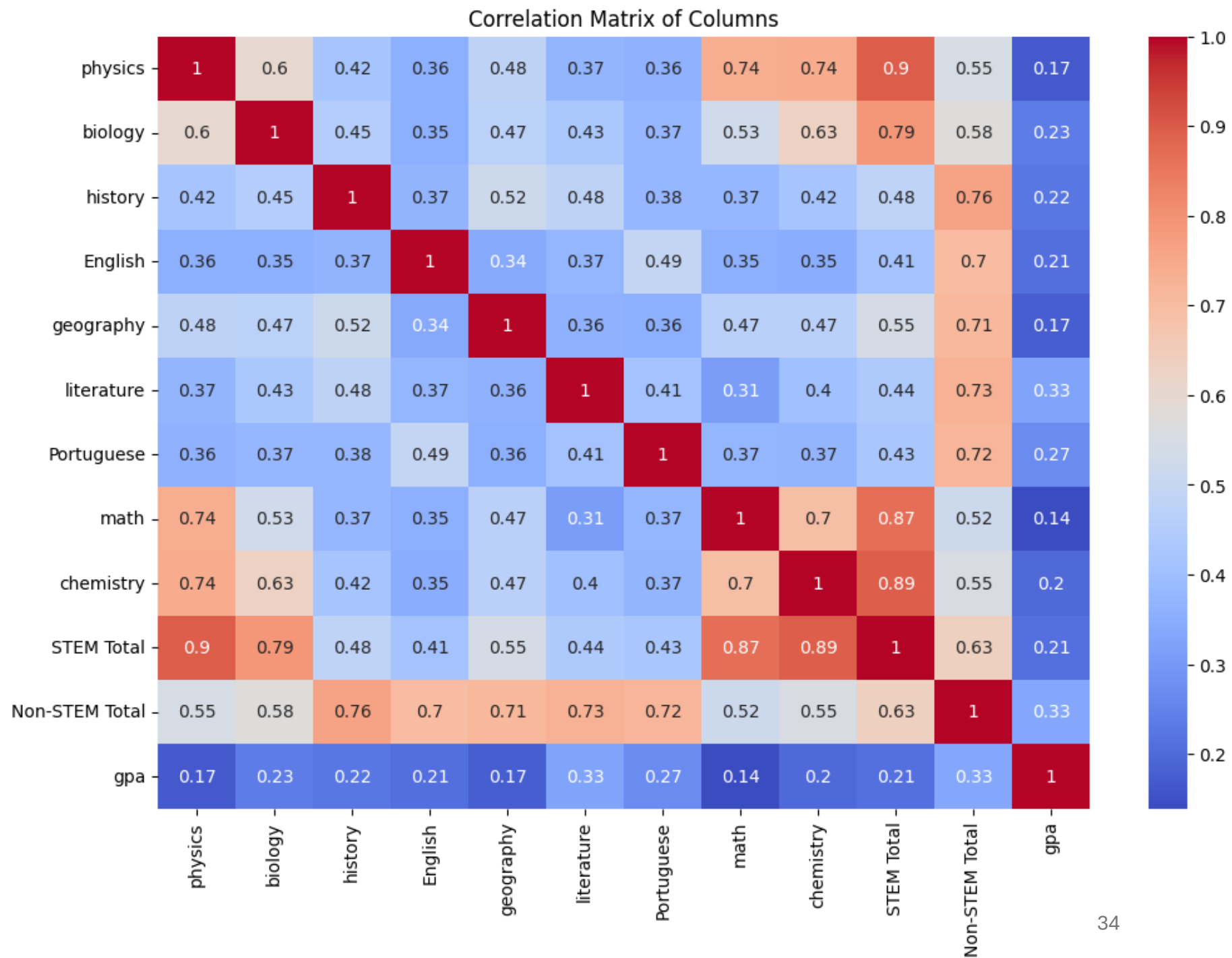
You can perform your own data analysis!

- These slides along with the corresponding Jupyter notebook are (or will soon be) on the course web page.

- You can perform your own additional data analysis to look for interesting trends.
  - E.g., you could further investigate STEM vs non-STEM exams and their relation to GPA:

```
Average GPA for high STEM scorers: 2.906325636524872
Average GPA for low STEM scorers: 2.6651347056253463
Average GPA for high Non-STEM scorers: 2.996511625250566
Average GPA for low Non-STEM scorers: 2.574952882149455
```

Histogram of GPAs (Split by STEM Scores) · Histogram of GPAs (Split by Non-STEM Scores)
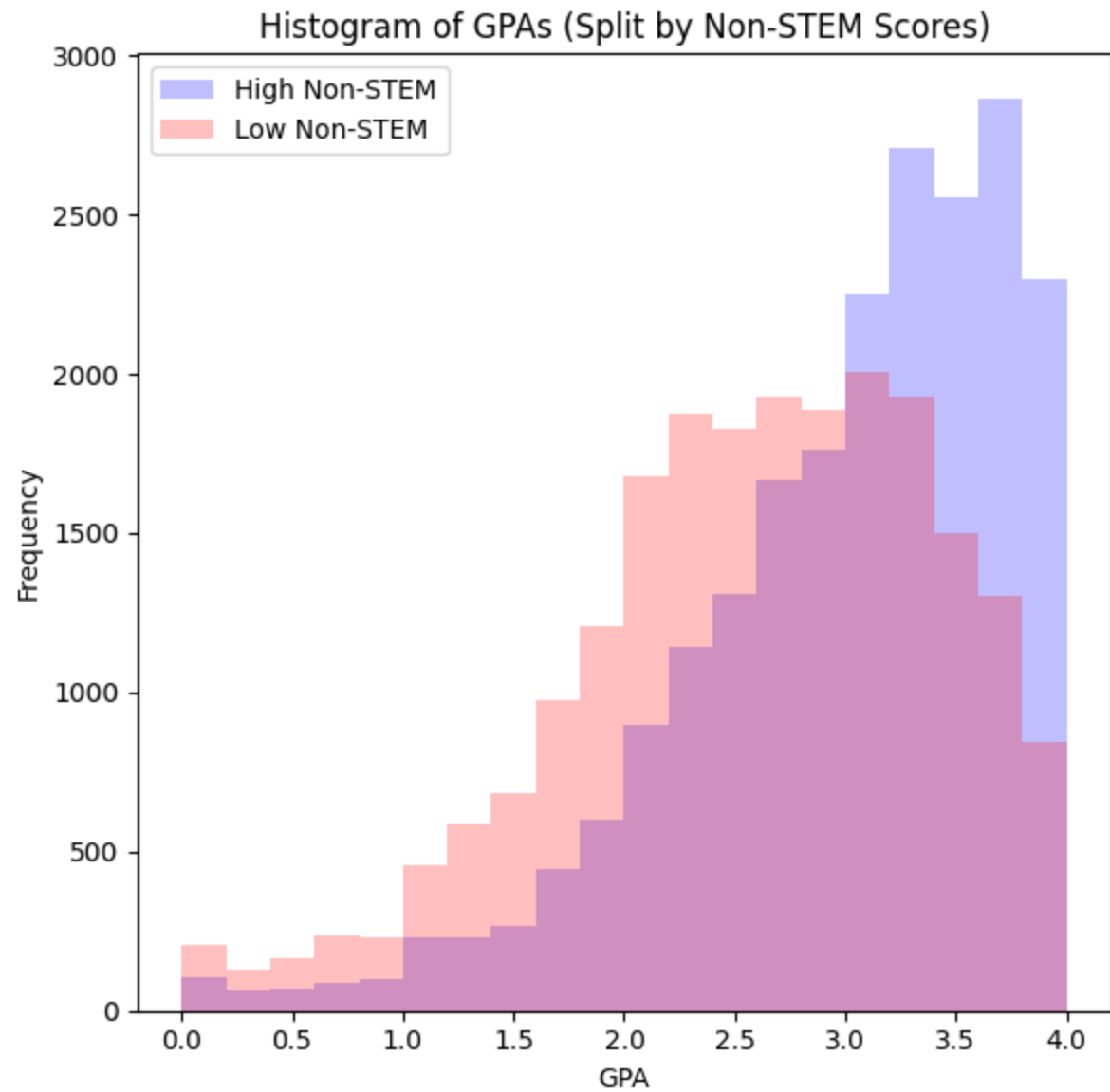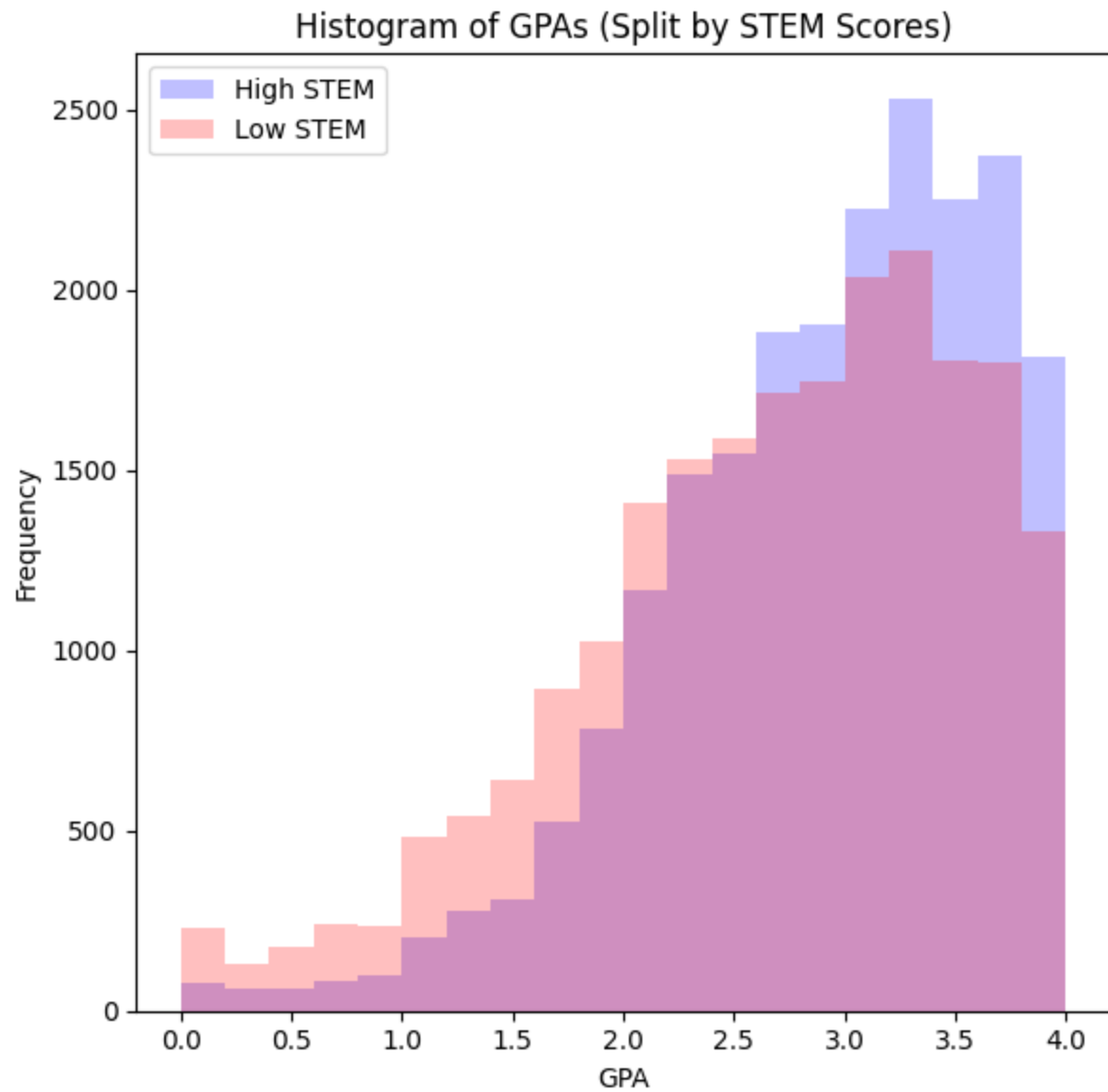
Fisher's Iris Data Set

- Often called just **Iris**.

- Introduced by British statistician Ronald Fisher in 1936.

- Often used for testing classification algorithms.
  - A standard in introductory ML education.

- Contains measurements of Iris flowers.

- The goal is to predict the species based on the measurements.

Fisher's Iris Data Set

- Columns:
  - **Sepal Length**: The length of the sepal in cm.
  - **Sepal Width**: The width of the sepal in cm.
  - **Petal Length**: The length of the petal in cm.
  - **Petal Width**: The width of the petal in cm.
  - **Species**: The species of the flower (Setosa, Versicolor, Virginica)
- Number of rows: 150

Mature flower
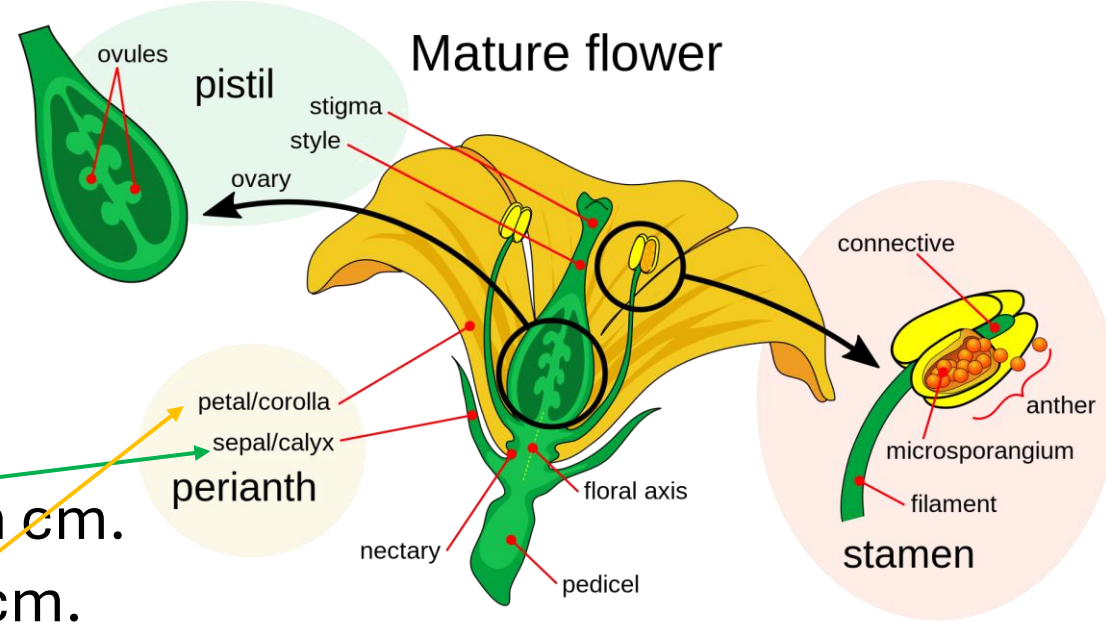
Iris setosa

Iris versicolor

Iris virginica

Scikit-learn: Machine Learning in Python

- Scikit-learn is a popular library for basic machine learning in Python.

- It includes some data sets like Iris.

- Installation:

Here `iris` is **not** a DataFrame.
It has member variables:
- `iris.feature_names`
  - Column names
- `iris.data`
  - Column data for features
- `iris.target`
  - Column data for label as integer
- `iris.target_names`
  - Column data for label as string

```
pip install scikit-learn
```

```python
from sklearn.datasets import load_iris


iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
display(df);
```

39

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

```
# We could have instead viewed the species by their names:
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
display(df);
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...				...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Question: Is each column numerical, categorical, text, or an image? Continuous, discrete, nominal, or ordinal?

150 rows × 5 columns

Adult Data Set

- Also known as the *Census Income* data set

- Often used in ML research (not just education)

- Part of the UCI machine learning repository ([link](#))
  - Repository contains $\geq$ 664 data sets you can browse!

- OpenML ([link](#)) is a platform that hosts many data sets, including many from the UCI ML repository.

- Scikit-learn provides functions for loading data sets from OpenML

  scikit-learn$\rightarrow$OpenML$\rightarrow$UCI ML repository

- We will use this approach, but you *could* download the data set from the UCI ML repository directly.

Adult Data Set

- Extracted from the 1994 Census database of the United States.
- Commonly used for predicting whether an individual's income exceeds $50,000 per year.
  - Typically framed as binary classification

Adult Data Set: 14 Features and Label

1. **Age**: The age of the individual.
2. **Workclass**: The type of employer the individual has (e.g., private, federal-gov).
3. **fnlwgt**: Final weight.
   - The number of people the census believes the entry represents.
   - E.g., a value of 50 means that the person's data represents 50 people in the real population with similar attributes.
   - Individual people can be represented by multiple rows, so `fnlwgt` does not sum to the US population.
   - This "binning" of people helps to preserve privacy.
4. **Education**: The highest level of education achieved by the individual.
5. **Education-Num**: The highest level of education in numerical form.
6. **Marital Status**: Marital status of the individual.

7. **Occupation**: The individual's occupation.
8. **Relationship**: The individual's role in the family.
9. **Race**: The race of the individual.
10. **Sex**: The gender of the individual.
11. **Capital Gain**: Income from investment sources, apart from wages/salary.
12. **Capital Loss**: Losses from investment sources, apart from wages/salary.
13. **Hours per Week**: Number of hours worked per week.
14. **Native Country**: Country of origin for the individual.

Label: Whether the individual earns more than $50,000 per year

Store data as a DataFrame

```
from sklearn.datasets import fetch_openml

adult = fetch_openml(name='adult', version=2, as_frame=True, parser='auto')
df = adult.data
df['income'] = adult.target
display(df)
```

Will try to parse data for pandas (the default anyway when as_frame=True), but has some fallbacks if there are errors. Only really needed to avoid a warning.

Note that both OpenML and skikit-learn return data sets with the features (data) and labels (target) stored separately. For now we are putting them into one table, but when we start using the data for ML we often will separate the features from the labels, and so this combination into one DataFrame could be skipped.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	NaN	103497	Some-college	10	Never-married	NaN	Own-child	White	Female	0	0	30	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
48840	22	Private	201490	HS-grad	9							0	20	United-States	<=50K
48841	52	Self-emp-inc	287927	HS-grad								0	40	United-States	>50K

Question: Is each column numerical, categorical, text, or an image? Continuous, discrete, nominal, or ordinal?

Column Types

- If you load a data set and do not know the type of a column, you can query DataFrame.dtype

```python
# We can get the types of columns using .dtype. This is a type object, not a string.
print('Second column .dtype is:');
print('\tType: ', type(df['workclass'].dtype))
print('\tValue: ', df['workclass'].dtype)

# We can get the string representation of the type with dtype.name.
# This is useful when we want to compare the type to a string.
print('Second column .dtype.name is:');
print('\tType: ', type(df['workclass'].dtype.name))
print('\tValue: ', df['workclass'].dtype.name)
```

```
Second column .dtype is:
        Type:   <class 'pandas.core.dtypes.dtypes.CategoricalDtype'>
        Value:  category
Second column .dtype.name is:
        Type:   <class 'str'>
        Value:  category
```

Common values of dtype:
- object (string)
- int64
- Float64
- bool
- datetime64
- category

46

Querying possible categories for categorical columns

```
df['workclass'].cat.categories
```

Functions/attributes for categorical features.

```
Index(['Federal-gov', 'Local-gov', 'Never-worked', 'Private', 'Self-emp-inc',
       'Self-emp-not-inc', 'State-gov', 'Without-pay'],
      dtype='object')
```

Index is an immutable (const) array in Pandas
dtype='object' tells us the type of the entries in the index. 'object' means Python string.

MNIST 784 Dataset

- Often called just MNIST

- Subset of a larger set available from NIST (National Institute of Standards and Technology)

- One of the most widely used data sets in ML for benchmarking classification algorithms

- Contains 70,000 images of handwritten digits (0 through 9)

- Each is a grayscale picture of a handwritten letter
  - 28x28 pixels flattened into a 1-dimensional, 784-element array
  - Each pixel is an integer from 0 to 255 (0 = white/background, 255 = black)

MNIST 784 Dataset

```python
mnist = fetch_openml('mnist_784', version=1, as_frame=True, parser="auto")
df = mnist.data
df['letter'] = mnist.target
display(df)
```

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel776	pixel777	pixel778	p
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0												0	0	
4	0												0	0	
...	...												...	...	
69995	0												0	0	
69996	0												0	0	
69997	0												0	0	
69998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
69999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

Label: 5   Label: 0   Label: 4   Label: 1   Label: 9

70000 rows × 785 columns

Intermission

- Class will resume in 5 minutes.

- Feel free to:
  - Stand up and stretch.
  - Leave the room.
  - Talk to those around you.
  - **Write a question on a notecard and add it to the stack at the front of the room.**