L2

January 8, 2018

1 Exploratory Data Analysis

CPSC 340: Machine Learning and Data Mining
The University of British Columbia
2017 Winter Term 2
Notebook by Mike Gelbart, based on slides by Mark Schmidt.

In [12]:
```
# lecture imports / dependencies
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import seaborn as sns
sns.set(style="ticks")
from sklearn.feature_extraction.text import CountVectorizer
from skimage.io import imread, imshow
```

1.1 Admin

- Get a CS ugrad account: https://www.cs.ubc.ca/getacct/
- Course website: https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/home
- Course Piazza sign-up: https://piazza.com/class/j9uk5ecmb7e4ks
- Tutorials start next week
- The lectures will be a mix of PowerPoint and jupyter notebook (this)
- both will be available online
- you can view the "static" notebook directly on GitHub
- you can run the notebook locally and play around with it

1.2 Typical steps of ML

1. Identify question / task
2. Collect data
3. Clean and preprocess data
4. Exploratory data anlysis (EDA)
5. Feature and model selection
6. Train model
7. Evaluate and communicate results
8. Deploy working system

(but not necessarily in this order...)
Today we'll discuss steps (3) and (4)

1.3 What does data look like?

Often, it is tabular (but certainly not always!).

```
In [13]: titanic = sns.load_dataset("titanic")
         titanic.head()
```

```
Out[13]:    survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
         0         0       3    male  22.0      1      0   7.2500        S  Third
         1         1       1  female  38.0      1      0  71.2833        C  First
         2         1       3  female  26.0      0      0   7.9250        S  Third
         3         1       1  female  35.0      1      0  53.1000        S  First
         4         0       3    male  35.0      0      0   8.0500        S  Third

              who  adult_male deck  embark_town alive  alone
         0    man        True  NaN  Southampton    no  False
         1  woman       False    C    Cherbourg   yes  False
         2  woman       False  NaN  Southampton   yes   True
         3  woman       False    C  Southampton   yes  False
         4    man        True  NaN  Southampton    no   True
```

- Each row is an **object** (or training example, or sample)
- Each column is a **feature** (or variable, covariate).

1.4 Types of features

- Categorical (e.g. `survived`, `embark_town`)
- Numerical (e.g. age, fare)
- Some are more ambiguous, like `pclass`: is this categorical or numerical?

Converting types:

- Many of our methods are meant to work with numerical features.
- We can convert categorical to numerical.

```
In [14]: pd.get_dummies(titanic, columns=["embarked"]).head()
```

```
Out[14]:    survived  pclass     sex   age  sibsp  parch     fare  class    who  \
         0         0       3    male  22.0      1      0   7.2500  Third    man
         1         1       1  female  38.0      1      0  71.2833  First  woman
         2         1       3  female  26.0      0      0   7.9250  Third  woman
         3         1       1  female  35.0      1      0  53.1000  First  woman
         4         0       3    male  35.0      0      0   8.0500  Third    man

            adult_male deck  embark_town alive  alone  embarked_C  embarked_Q  \
         0        True  NaN  Southampton    no  False           0           0
```

```
1       False    C    Cherbourg  yes  False              1            0
2       False  NaN  Southampton  yes   True              0            0
3       False    C  Southampton  yes  False              0            0
4        True  NaN  Southampton   no   True              0            0

   embarked_S
0           1
1           0
2           1
3           1
4           1
```

If we do this for all our features, we can now interpret objects as points in space.

```
In [15]: titanic_num = pd.get_dummies(titanic, columns=["sex","embarked","fare","class","who","a
         titanic_num.shape
```

```
Out[15]: (891, 280)
```

- So we now have 891 objects and 280 features.
- In other words, each object is a point in 280-dimensional space.
- This is why multivariable calculus is a prerequisite.

1.4.1 Other feature types: text data

```
In [16]: text = "The University of British Columbia (UBC) is a public research university with c
```

One approach: **bag of words** features.

```
In [17]: cv = CountVectorizer()
         feat = cv.fit_transform([text])
```

```
In [18]: for word, idx in cv.vocabulary_.items():
             print("%-14s%d" % (word, feat[0,idx]))
```

```
the           1
university    2
of            1
british       2
columbia      2
ubc           1
is            1
public        1
research      1
with          1
campuses      1
and           1
facilities    1
in            1
canada        1
```

- Bag of words ignores the order of words but still can work well.
- You can interpret each document as a point in space, compute distances.

1.4.2 Other feature types: images

```
In [19]: img = imread("https://upload.wikimedia.org/wikipedia/commons/8/86/Irving_K._Barber_Libr
         plt.xticks([])
         plt.yticks([])
         imshow(img);
```

Photo credit: Wikipedia: UBC by CjayD, CC BY 2.0.

```
In [20]: img.shape

Out[20]: (1344, 2048, 3)

In [21]: img[0:2,0:2,:]

Out[21]: array([[[45, 58, 90],
                 [45, 59, 88]],

                [[45, 59, 88],
                 [45, 59, 88]]], dtype=uint8)

In [22]: img.flatten().shape
```

```
Out[22]: (8257536,)
```

- Now, again, the image is a point in space.
- But now the space is 8,257,536-dimensional!
- We'll talk about this towards the end of the course.

1.5 Data Cleaning

- ML+DM typically assume "clean" data.
- Ways that data might not be "clean":
- noise (e.g., distortion on phone).
- outliers (e.g., data entry or instrument error).
- missing values (no value available or not applicable)
- duplicated data (repetitions, or different storage formats).
- Any of these can lead to problems in analyses.
- want to fix these issues, if possible.
- some ML methods are robust to these.
- often, ML is the best way to detect/fix these.

1.6 How much data do we need?

- A difficult if not impossible question to answer.
- Usual answer: "more is better".
- With the warning: "as long as the quality doesn't suffer".
- Another popular answer: "ten times the number of features".
- I don't like this view. Features are not the enemy!

1.7 Feature aggregation

- Combine features to form new ones
- Useful if there are few examples of a particular case

```
In [23]: titanic['deck'].value_counts()
```

```
Out[23]: C    59
         B    47
         D    33
         E    32
         A    15
         F    13
         G     4
         Name: deck, dtype: int64
```

```
In [24]: titanic_agg = titanic.copy()

         # aggregate decks A and B into the "upper" deck category
         titanic_agg["upper"] = titanic_agg['deck'].isin(("A","B"))
         titanic_agg.tail()
```

```
Out[24]:      survived  pclass     sex   age  sibsp  parch   fare embarked   class  \
         886         0       2    male  27.0      0      0  13.00        S  Second
         887         1       1  female  19.0      0      0  30.00        S   First
         888         0       3  female   NaN      1      2  23.45        S   Third
         889         1       1    male  26.0      0      0  30.00        C   First
         890         0       3    male  32.0      0      0   7.75        Q   Third

                who  adult_male deck  embark_town alive  alone  upper
         886    man        True  NaN  Southampton    no   True  False
         887  woman       False    B  Southampton   yes   True   True
         888  woman       False  NaN  Southampton    no  False  False
         889    man        True    C    Cherbourg   yes   True  False
         890    man        True  NaN   Queenstown    no   True  False
```

(Not shown: we should still fix up the NaNs here!)

1.8 Feature selection

```
In [25]: titanic_id = titanic.copy()

         # Adding an irrelevant feature
         titanic_id['id'] = titanic_id.index
         titanic_id.head()
```

```
Out[25]:    survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
         0         0       3    male  22.0      1      0   7.2500        S  Third
         1         1       1  female  38.0      1      0  71.2833        C  First
         2         1       3  female  26.0      0      0   7.9250        S  Third
         3         1       1  female  35.0      1      0  53.1000        S  First
         4         0       3    male  35.0      0      0   8.0500        S  Third

              who  adult_male deck  embark_town alive  alone  id
         0    man        True  NaN  Southampton    no  False   0
         1  woman       False    C    Cherbourg   yes  False   1
         2  woman       False  NaN  Southampton   yes   True   2
         3  woman       False    C  Southampton   yes  False   3
         4    man        True  NaN  Southampton    no   True   4
```

- Remove features that are not relevant to the task.
- `id` probably not relevant for prediction.

1.9 Feature transformation

Discretization (binning): turn numerical data into categorical

```
In [26]: titanic['age'].head()
```

```
Out[26]: 0    22.0
         1    38.0
```

```
2    26.0
3    35.0
4    35.0
Name: age, dtype: float64
```

In [27]:
```
ages = pd.cut(titanic['age'], bins=(0,20,30,100))
ages_cat = pd.get_dummies(ages)
pd.concat([titanic['age'], ages_cat],axis=1).head()
```

Out[27]:

	age	(0, 20]	(20, 30]	(30, 100]
0	22.0	0	1	0
1	38.0	0	0	1
2	26.0	0	1	0
3	35.0	0	0	1
4	35.0	0	0	1

Mathematical transformsations

- e.g. log, exp, square, sqrt, etc.
- also, scaling/normalization

In [28]:
```
titanic_mod = titanic.copy()

# fare --> sqrt(fare)
titanic_mod['fare'] = np.sqrt(titanic_mod['fare'])
titanic_mod.head()
```

Out[28]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	\
0	0	3	male	22.0	1	0	2.692582	S	Third	
1	1	1	female	38.0	1	0	8.442944	C	First	
2	1	3	female	26.0	0	0	2.815138	S	Third	
3	1	1	female	35.0	1	0	7.286975	S	First	
4	0	3	male	35.0	0	0	2.837252	S	Third	

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

Example use case: something needs to be non-negative (exp) or shouldn't be non-negative (log).

1.10 Exploratory data analysis (EDA)

- You should always "look" at the data first.
- But how do you "look" at features and high-dimensional objects?
- Summary statistics
- Visualization
- ML + DM (later in course)

1.11 Categorical summary statistics

- Some summary statistics for a categorical variable:
- **Frequencies** of different classes.
- **Mode**: category that occurs most often.

```
In [29]: titanic['deck'].value_counts(normalize=True) # frequencies
```

```
Out[29]: C    0.290640
         B    0.231527
         D    0.162562
         E    0.157635
         A    0.073892
         F    0.064039
         G    0.019704
         Name: deck, dtype: float64
```

```
In [30]: titanic['deck'].mode()[0]
```

```
Out[30]: 'C'
```

1.12 Continuous summary statistics

- Measures of location:
- **Mean**: average value.
- **Median**: value such that half points are larger/smaller.
- **Quantiles**: value such that t fraction of points are smaller.
- Measures of spread:
- **Range**: minimum and maximum values.
- **Variance**: measures how far values are from mean.
  - Square root of variance is **standard deviation**.
- **Intequantile ranges**: difference between quantiles

```
In [31]: titanic['fare'].mean()
```

```
Out[31]: 32.2042079685746
```

```
In [32]: titanic['fare'].median()
```

```
Out[32]: 14.4542
```

```
In [33]: titanic['fare'].quantile((0.25,0.5,0.75))
```

```
Out[33]: 0.25     7.9104
         0.50    14.4542
         0.75    31.0000
         Name: fare, dtype: float64
```

```
In [34]: titanic['fare'].min()
```

```
Out[34]: 0.0

In [35]: titanic['fare'].max()

Out[35]: 512.32920000000001

In [36]: titanic['fare'].var()

Out[36]: 2469.436845743117

In [37]: titanic['fare'].std()

Out[37]: 49.693428597180905
```

Notice that the mean and std are sensitive to extreme values:

```
In [38]: data = [0,1,2,3,3,5,7,8,9,10,14,15,17,200] # the "200" is an outlier
         print("Mean with outlier   :", np.mean(data))
         print("Mean without outlier:", np.mean(data[:-1]))

Mean with outlier   : 21.0
Mean without outlier: 7.23076923077


In [39]: print("Std with outlier   :", np.std(data))
         print("Std without outlier:", np.std(data[:-1]))

Std with outlier   : 49.9127810714
Std without outlier: 5.35154680952
```

Whereas the median is not:

```
In [40]: print("Median with outlier   :", np.median(data))
         print("Median without outlier:", np.median(data[:-1]))

Median with outlier   : 7.5
Median without outlier: 7.0
```

1.13 Distances and similarities

- There are also summary statistics between features.
- Hamming distance:
  - Number of elements in the vectors that aren't equal.
- Euclidean distance:
  - How far apart are the vectors?
- Correlation:
  - Does one increase/decrease linearly as the other increases?
  - Between -1 and 1.

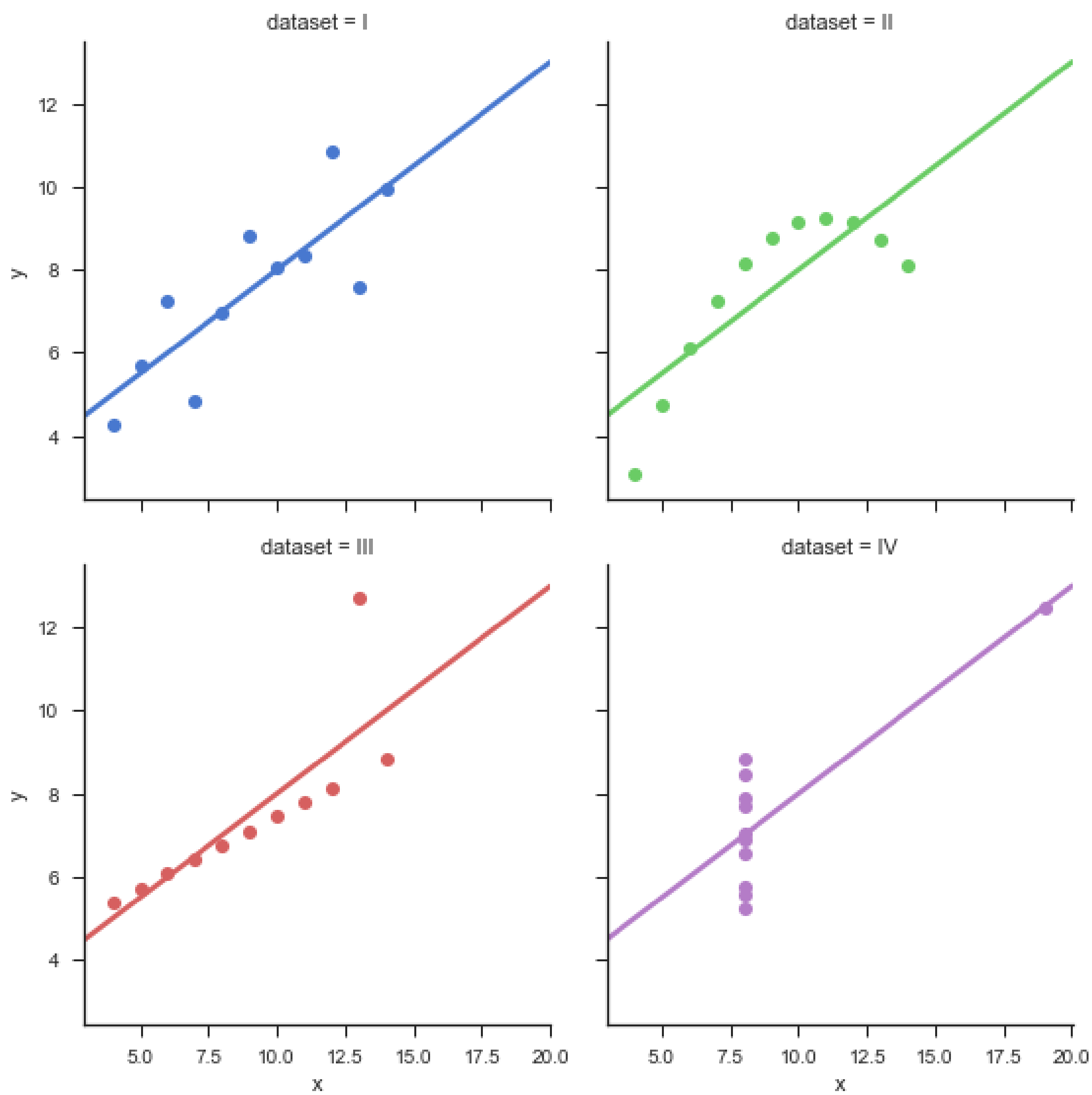
1.14 Limitations of summary statistics

- Summary statistics can be misleading
- A famous example is Anscombe's quartet, four datasets with:
- Almost same means.
- Almost same variances.
- Almost same correlations.
- Almost same linear fits.
- Look completely different.

In [41]:
```
# Code below from seaborn documentation: https://seaborn.pydata.org/examples/anscombes_

# Load the example dataset for Anscombe's quartet
anscombe = sns.load_dataset("anscombe")

# Show the results of a linear regression within each dataset
sns.lmplot(x="x", y="y", col="dataset", hue="dataset", data=anscombe,
           col_wrap=2, ci=None, palette="muted", size=4,
           scatter_kws={"s": 50, "alpha": 1});
```

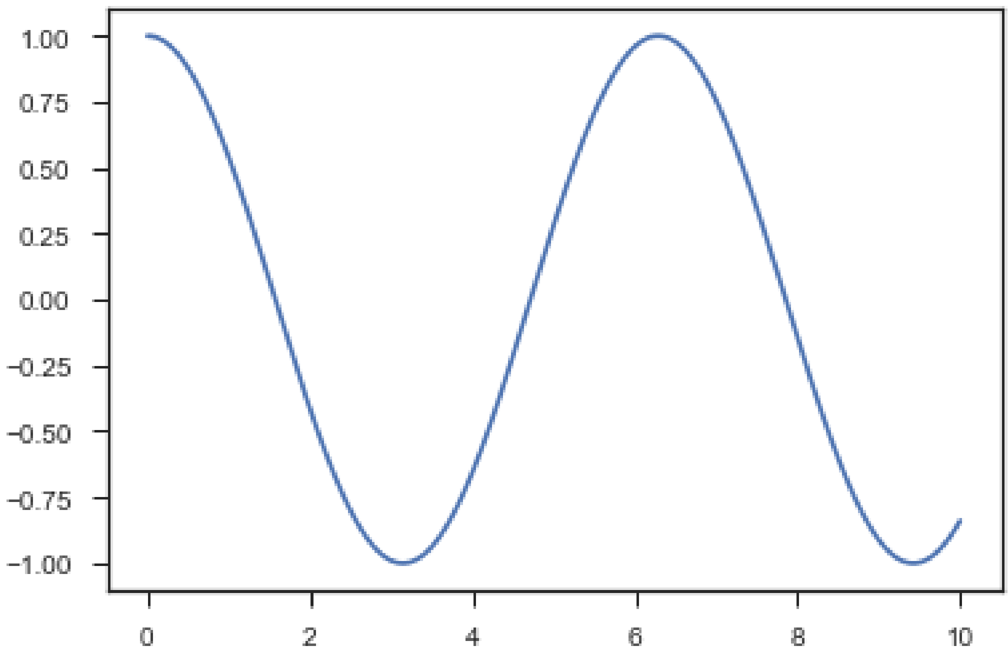


1.15 Visualization

- You can learn a lot from 2D plots of the data:
- Patterns, trends, outliers, unusual patterns.
- We'll use the `matplotlib` library to do most of our basic plotting.
- For fancier plots, you can try `seaborn`.

1.16 Basic plot

```
In [42]: x = np.linspace(0,10,1000)
         plt.plot(x, np.cos(x));
```

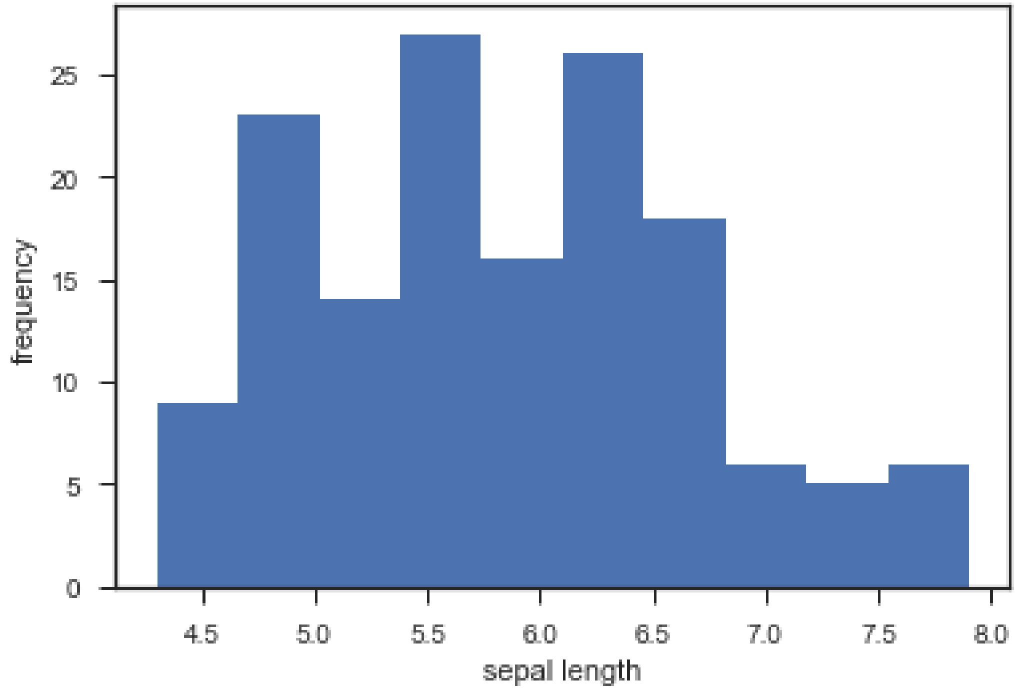



```
In [43]: iris = sns.load_dataset("iris") # iris flowers, a classic dataset
         iris.head()
```

```
Out[43]:    sepal_length  sepal_width  petal_length  petal_width 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
```

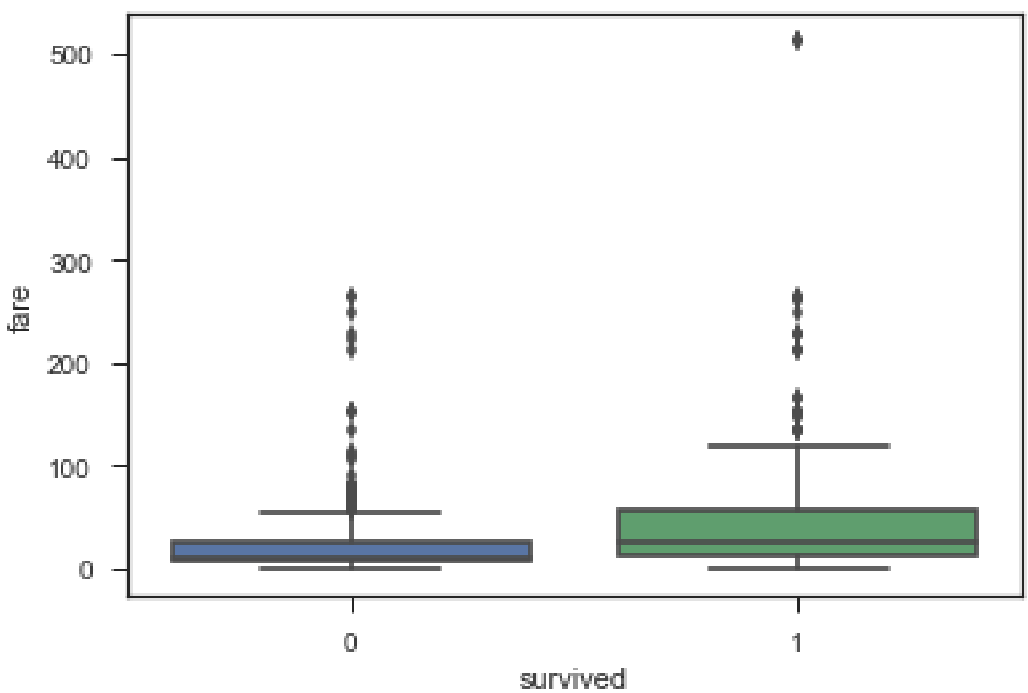
1.17 Histogram

```
In [44]: plt.hist(iris['sepal_length'])
         plt.xlabel('sepal length')
         plt.ylabel('frequency');
         # sns.distplot(iris["sepal_length"]);
```



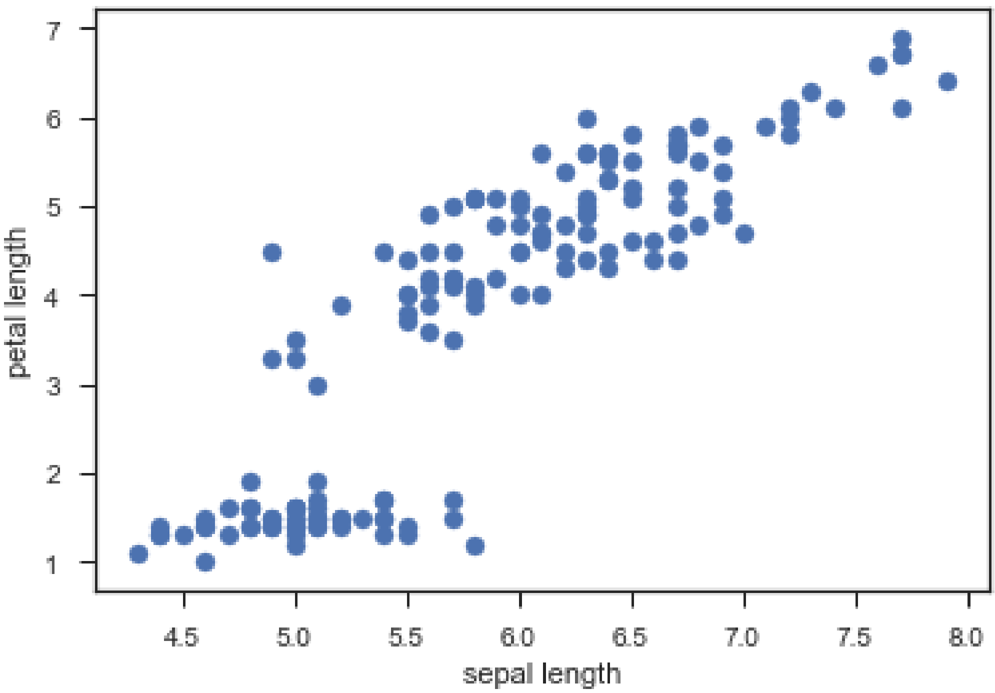
1.18 Box plot

```
In [45]: sns.boxplot(x="survived", y="fare", data=titanic);
```



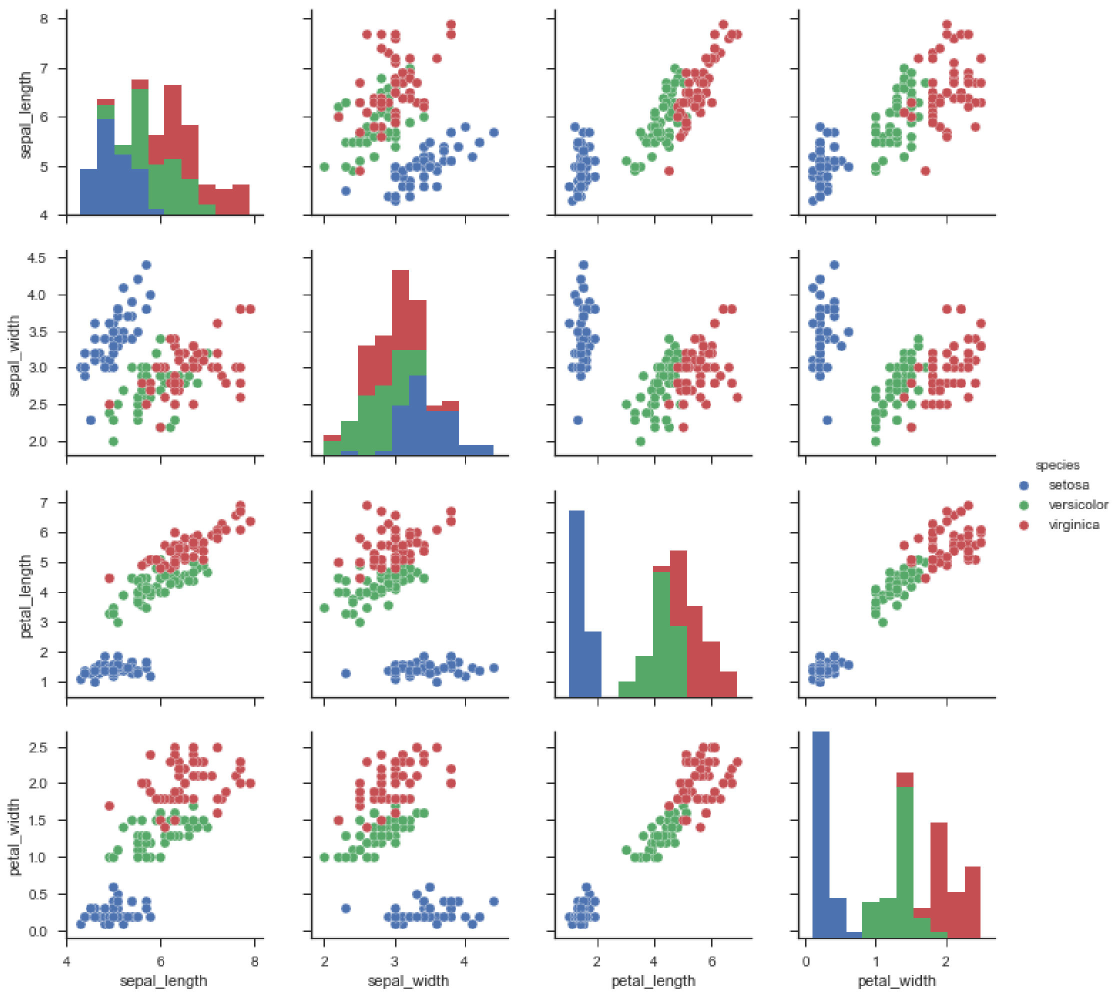
1.19 Scatterplot

```
In [46]: plt.scatter(iris['sepal_length'], iris['petal_length'])
         plt.xlabel('sepal length')
         plt.ylabel('petal length');
```



1.20 Scatterplot array

```
In [48]: sns.pairplot(iris, hue="species");
```



1.21 CPSC 340 meta-discussion

- This is the only CPSC 340 lecture on data cleaning and EDA.
- That is not representative of the time typically devoted to these tasks.
- In fact, data cleaning is often the most time intensive step.
- This is a weakness of the course.
- But not as bad if you're aware of it.

1.22 Summary

- Typical data mining steps:
- Involves data collection, preprocessing, analysis, and evaluation.
- Object-feature representation and categorical/numerical features.
- Transforming non-vector objects to vector representations.
- Feature transformations:
- To address coupon collecting or simplify relationships between variables.

- Exploring data:
- Summary statistics and data visualization.
- Post-lecture bonus slides: other visualization methods.