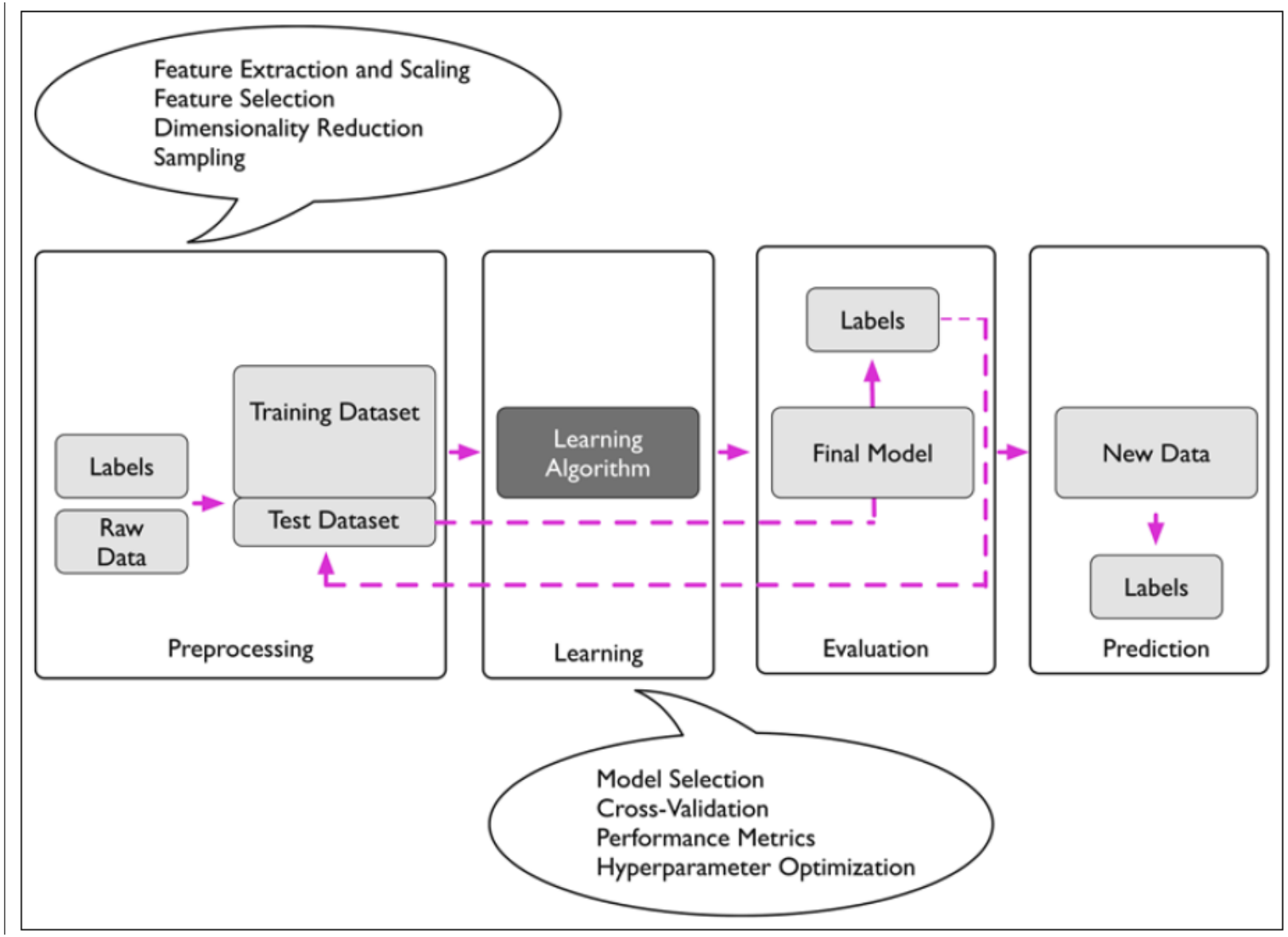
MODULE 3:

EXPLORATORY DATA ANALYSIS

1.4 Roadmap for building machine learning systems



1.4.1 Preprocessing – getting data into shape

- Raw data rarely comes in the form and shape that is necessary for the optimal performance of a learning algorithm
- Preprocessing of the data is one of the most crucial steps
- Sometimes called ‘data munging’ or ‘data wrangling’ or ‘cleaning’
- Normalization/Scaling - Many machine learning algorithms also require that the selected features are on the same scale for optimal performance, which is often achieved by transforming the features in the range [0, 1] or a standard normal distribution with zero mean and unit variance
- Dimensionality reduction to eliminate high correlation between features or reduce irrelevant features

Preprocessing aka Exploratory Data Analysis

- John W. Tukey – American statistician (1915-2000)
- Credited with inventing the Box Plot as well as many other statistical techniques
- Coined the term ‘bit’ for binary digit and first published use of the term “software” (1958).
- Published *Exploratory Data Analysis* (1977).
- Worked with John von Neumann and Claude Shannon
- Served at Princeton University and Bell Labs

- Photo By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=17099473

Understand the problem by understanding the data

- This module is based on Chapter 2 in the Bowles textbook

- Bowles, Michael. *Machine Learning in Python*. 2015. Wiley. Available free from Safari Books Online

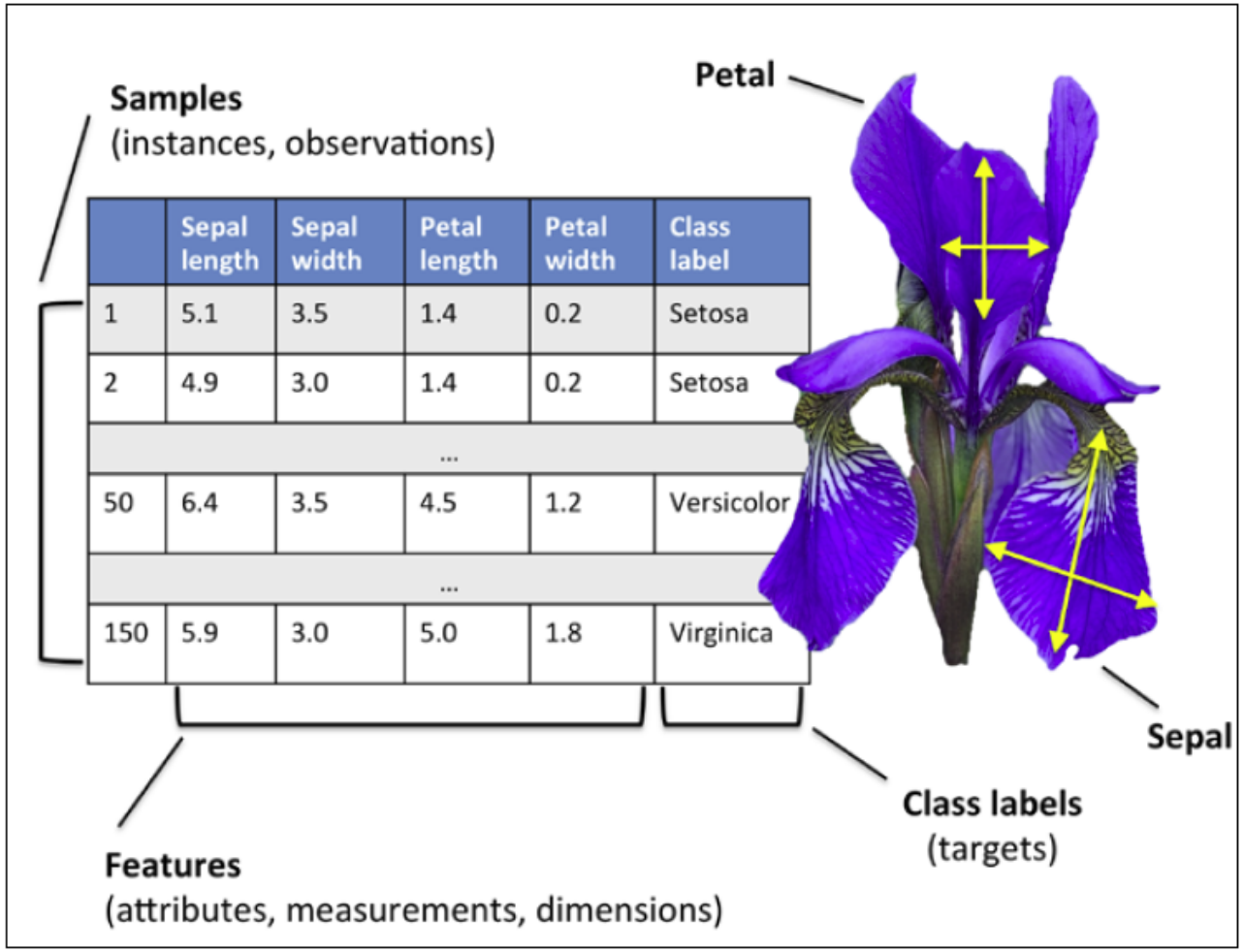
- If you are having difficulty accessing the textbook online, I will provide a pdf of this chapter.

A new data set is like a wrapped gift….

“It’s full of promise and anticipation at the miracles you can work once you’ve solved it. But it remains a mystery until you’ve opened it.

This chapter is about opening up your new data set so you can see what’s inside, get an appreciation for what you’ll be able to do with the data, and start thinking about how you’ll approach model building with it.”

1.3 Basic terminology and notations



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Rocks versus Mines dataset

- Gorman, R. P., and Sejnowski, T. J. (1988). UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar,+Mines+vs.+Rocks%29
- The file "sonar.mines" contains 111 patterns obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions. The file "sonar.rocks" contains 97 patterns obtained from rocks under similar conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. The data set contains signals obtained from a variety of different aspect angles, spanning 90 degrees for the cylinder and 180 degrees for the rock.

  Each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time. The integration aperture for higher frequencies occur later in time, since these frequencies are transmitted later during the chirp.

  The label associated with each record contains the letter "R" if the object is a rock and "M" if it is a mine (metal cylinder). The numbers in the labels are in increasing order of aspect angle, but they do not encode the angle directly.

Anatomy of a New Problem

Table 2-1: Data for a Machine Learning Problem

USERID	ATTRIBUTE 1	ATTRIBUTE 2	ATTRIBUTE 3	LABELS
001	6.5	Male	12	$120
004	4.2	Female	17	$270
007	5.7	Male	3	$75
008	5.8	Female	8	$600

What’s in a name

- The data are arranged into rows and columns. Each row represents an individual case (also called an *instance*, *example*, or *observation*).
- Attributes (the variables being used to make predictions) are also known as the following:
  - Predictors
  - Features
  - Independent variables
  - Inputs
- Labels are also known as the following:
  - Outcomes
  - Targets
  - Dependent variables
  - Response variables

Different Types of Attributes and Labels Drive Modeling Choices

Two different types of data labels:

- numeric variables
- categorical (or factor) variables

Table 2-2: Numeric Targets versus Categorical Targets

TABLE 1 LABELS	>$200 ?
$120	False
$270	True
$75	False
$600	True

Things to Notice about Your New Data Set

The following is a checklist and a sequence of things to learn about your data set to familiarize yourself with the data and to formulate the predictive model development steps that you want to follow.

Items to Check

- Number of rows and columns - dimensionality
- Number of categorical variables and number of unique values for each
- Missing values
- Summary statistics for attributes and labels

Listing 2-1 Sizing up a new dataset

```
#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

sys.stdout.write("Number of Rows of Data = " + str(len(xList)) + '\n')
sys.stdout.write("Number of Columns of Data = " + str(len(xList[1])))
```

```
Output:
Number of Rows of Data = 208
Number of Columns of Data = 61
```

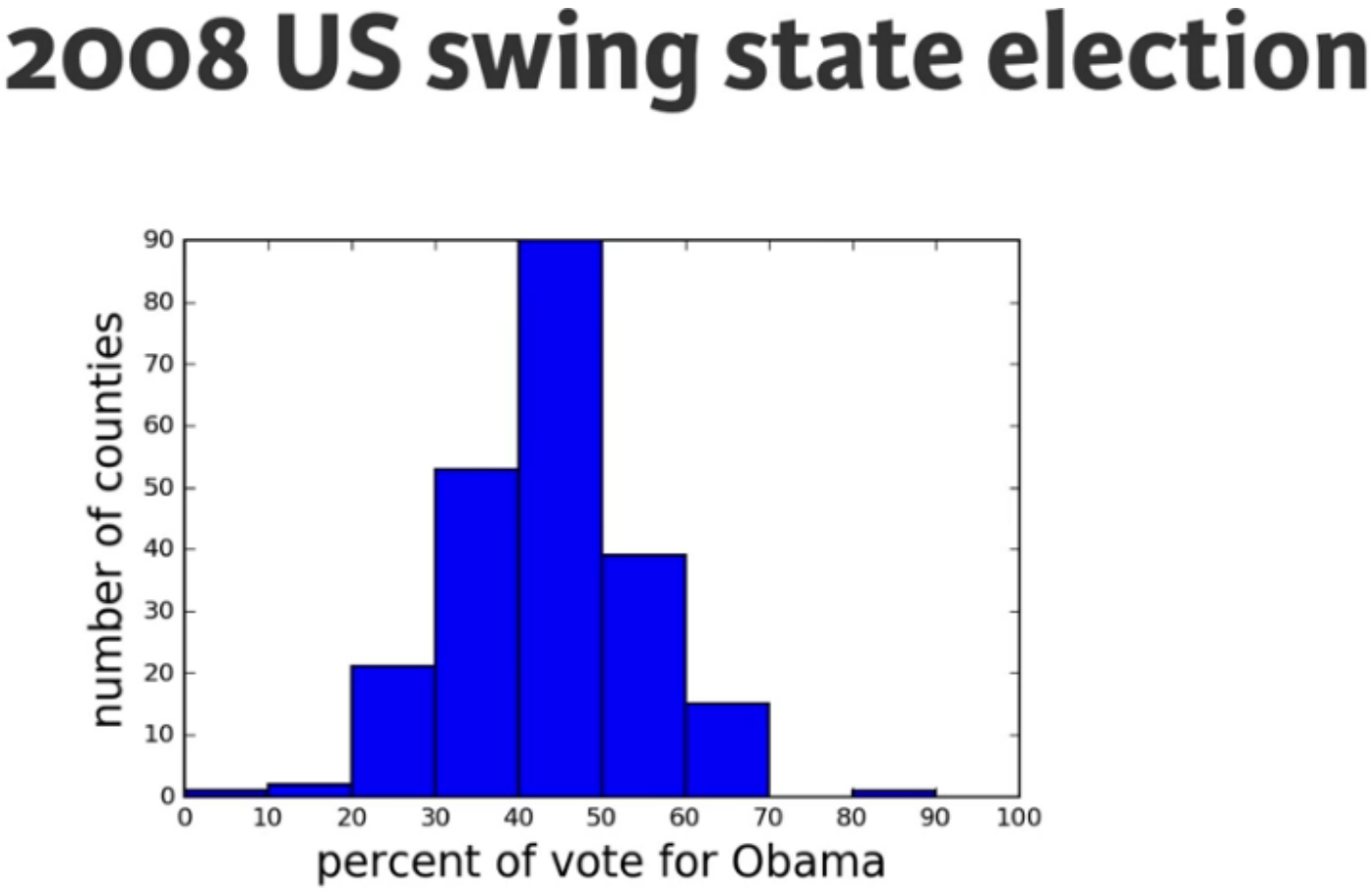
Determining the data types

- Int
- Float
- Strings
- Boolean
- Other {byte, tuple, list, set, dict}

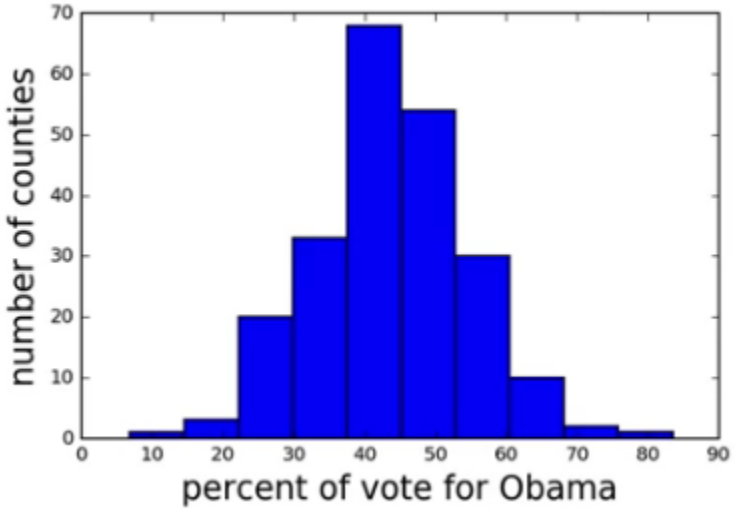
Listing 2-3 Summary statistics

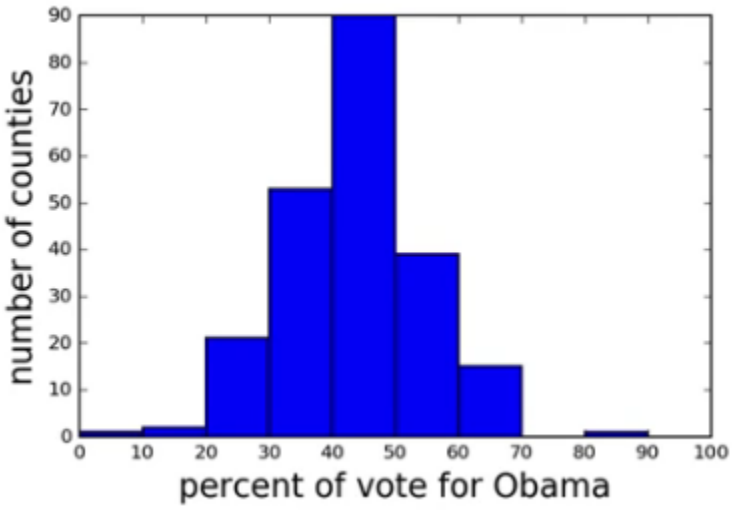
```
Output:
Mean =    0.053892307      Standard Deviation =     0.046415983

Boundaries for 4 Equal Percentiles
 [0.005799999999999996, 0.02437500000000001, 0.04404999999999999,
 0.06450000000000002, 0.4264]

Boundaries for 10 Equal Percentiles
[0.00579999999999, 0.0141, 0.022740000000, 0.027869999999,
0.036220000000, 0.044049999999, 0.050719999999, 0.059959999999,
0.077940000000, 0.10836, 0.4264]
Unique Label Values
set(['R', 'M'])

Counts for Each Value of Categorical Label
['R', 'M']
[97, 111]
```

Histograms

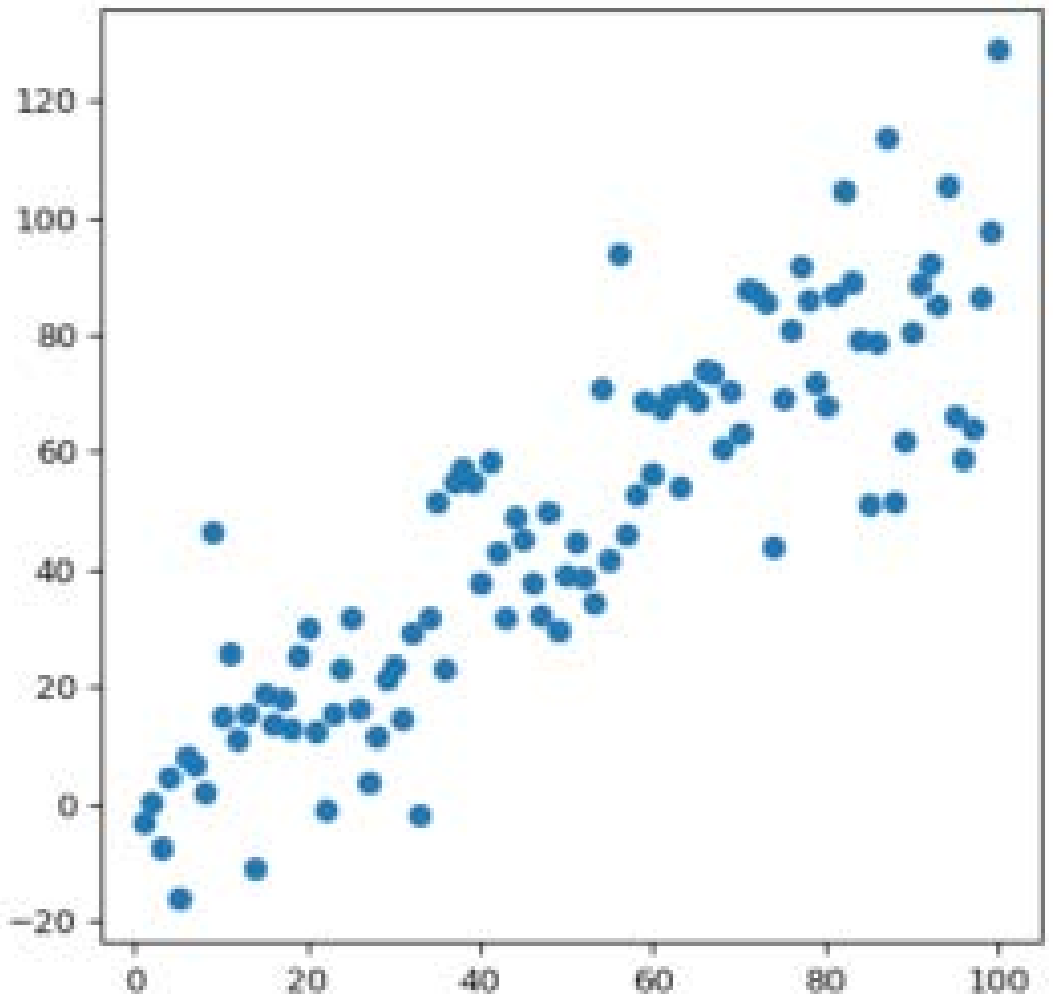

2008 US swing state election
number of counties
0
10
20
30
40
50
60
70
80
90
0
10
20
30
40
50
60
70
80
90
100
percent of vote for Obama

Binning bias – same data looks different with different number of bins

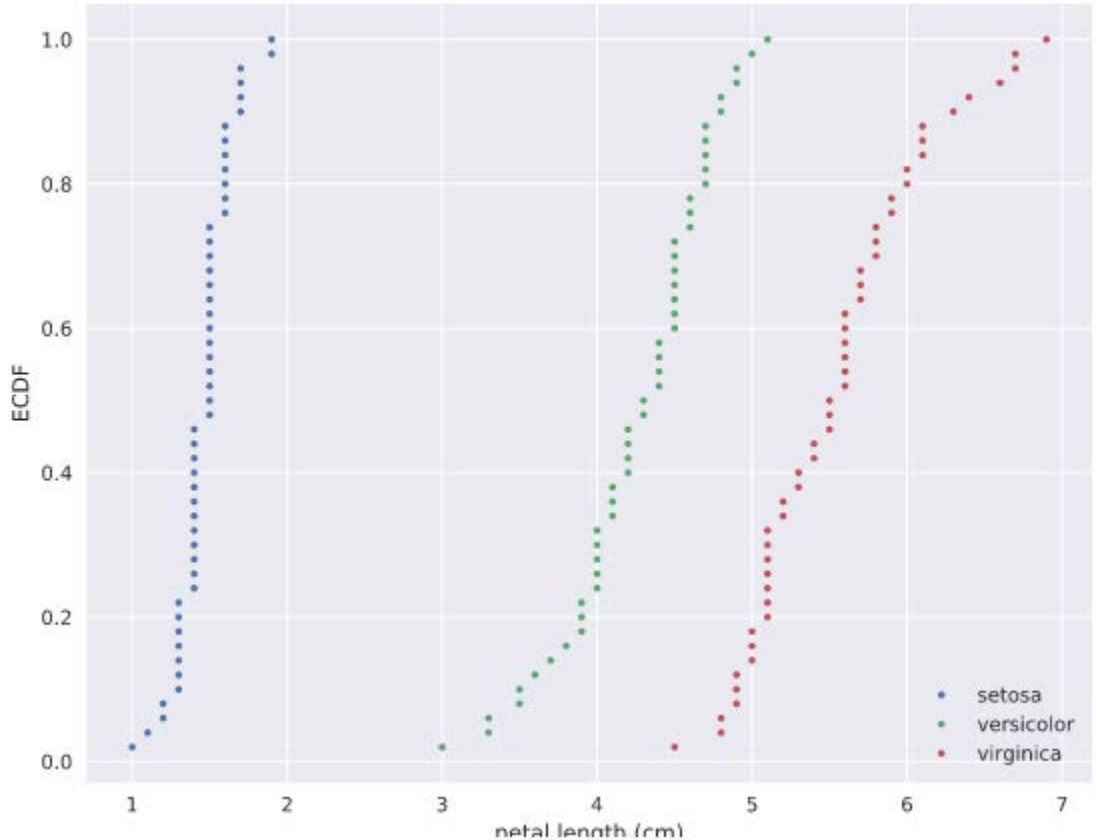




Scatter plot



ECDF – empirical cumulative distribution function



Quantile-Quantile Plot

- If the data being analyzed comes from a Gaussian distribution, the point being plotted will lie on a straight line.
- The tails of the rocks versus mines data contain more examples than the tails of a Gaussian density.

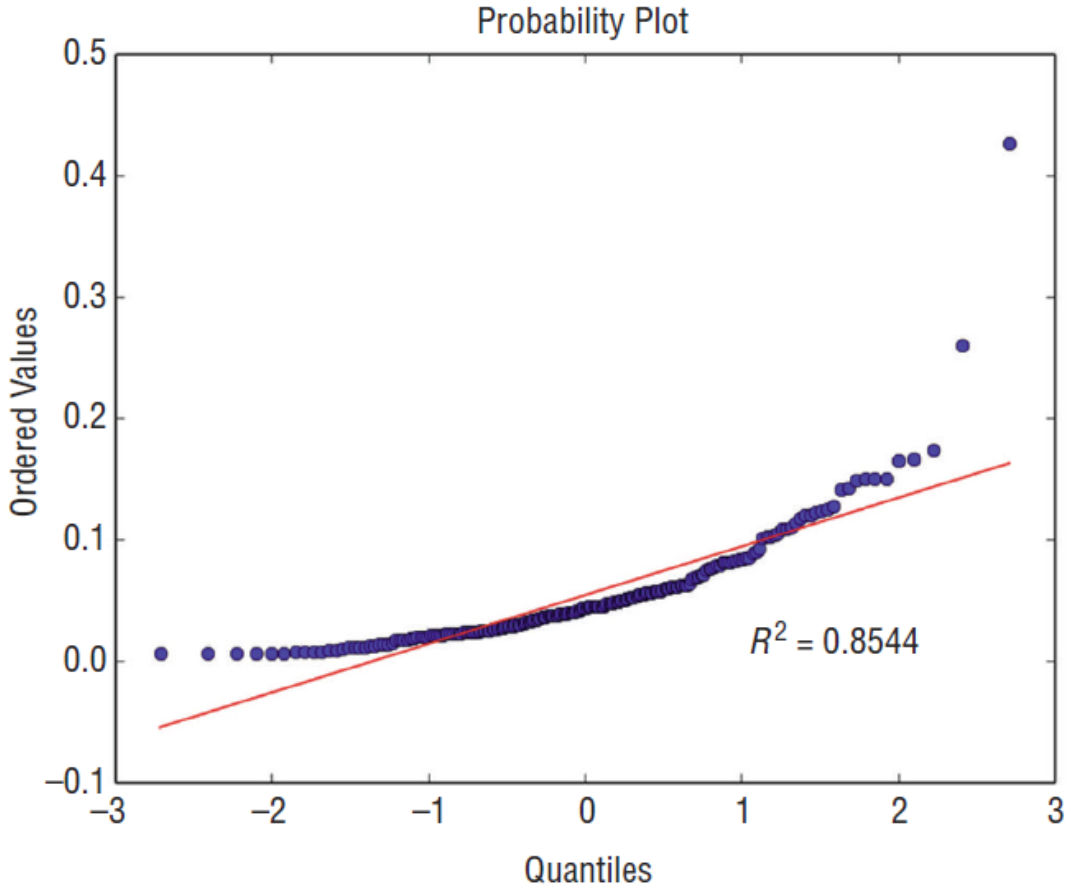
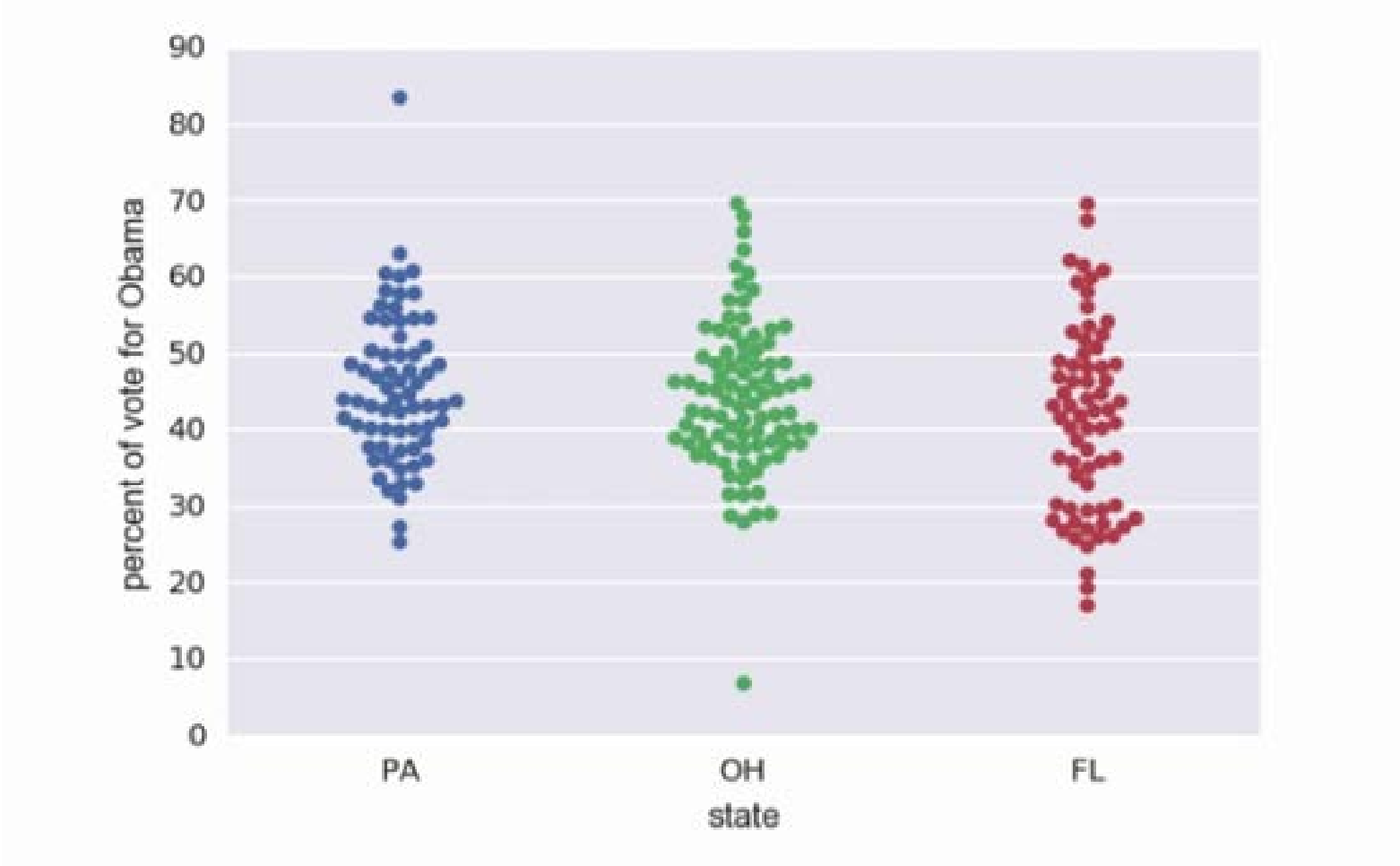
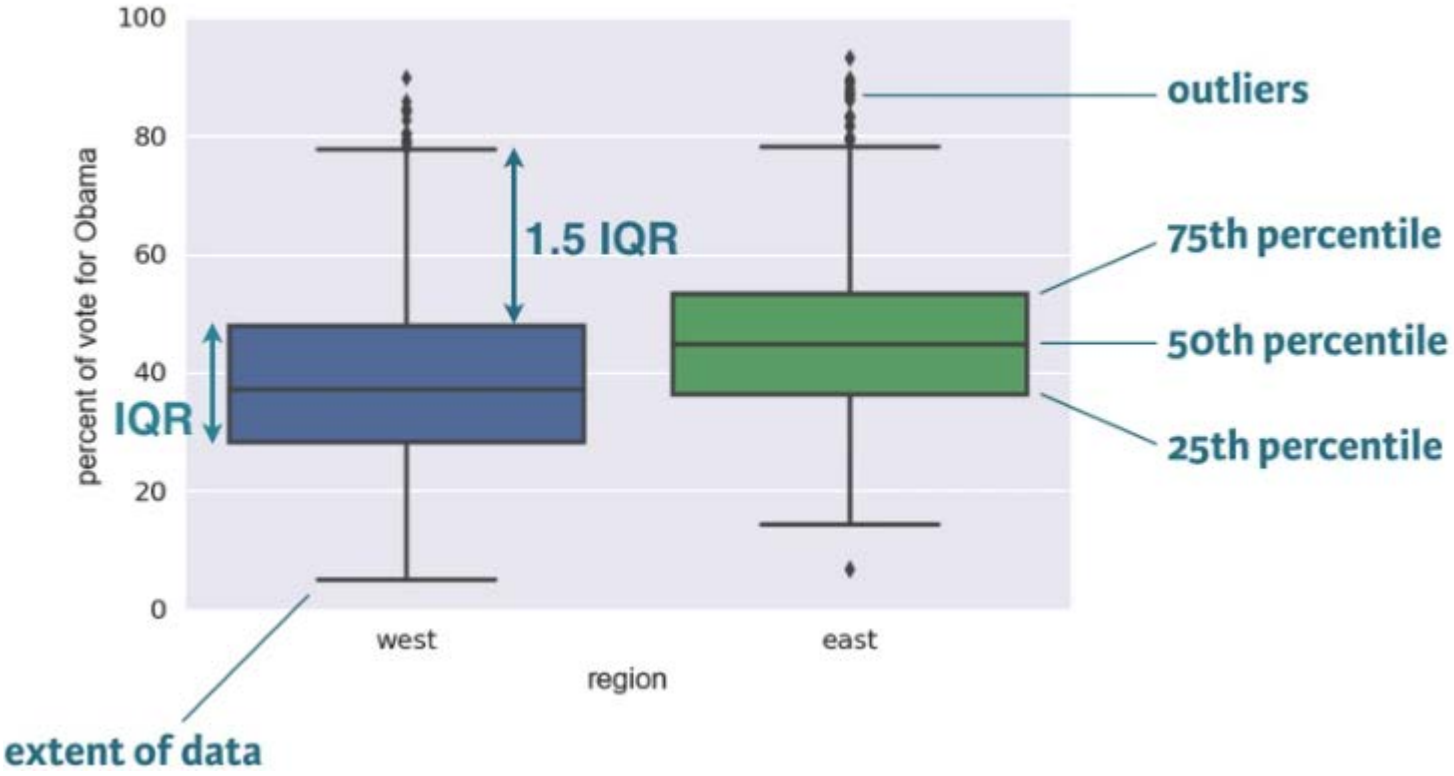


Figure 2-1: Quantile-quantile plot of attribute 4 from rocks versus mines data

Bee swarm plot



Box plot


100
80
60
40
20
0
percent of vote for Obama
1.5 IQR
IQR
west
east
region
outliers
75th percentile
50th percentile
25th percentile
extent of data

Dealing with missing values

- One of the easiest ways to deal with missing data is to simply remove the corresponding features (columns) or samples (rows) from the dataset entirely (**Dropna** method in Python)
- Replace by imputation – one of the most common interpolation techniques is **mean imputation**, where we simply replace the missing value with the mean value of the entire feature column.
- **Imputer** class from scikit-learn -- Other options for the strategy parameter are *median* or *most_frequent*. This is useful for imputing categorical feature values, for example, a feature column that stores an encoding of color names, such as red, green, and blue.

Listing 2-5 Using pandas to summarize data

```
#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#print head and tail of data frame
print(rocksVMines.head())
print(rocksVMines.tail())

#print summary of data frame
summary = rocksVMines.describe()
print(summary)
```

Parallel Coordinates plot

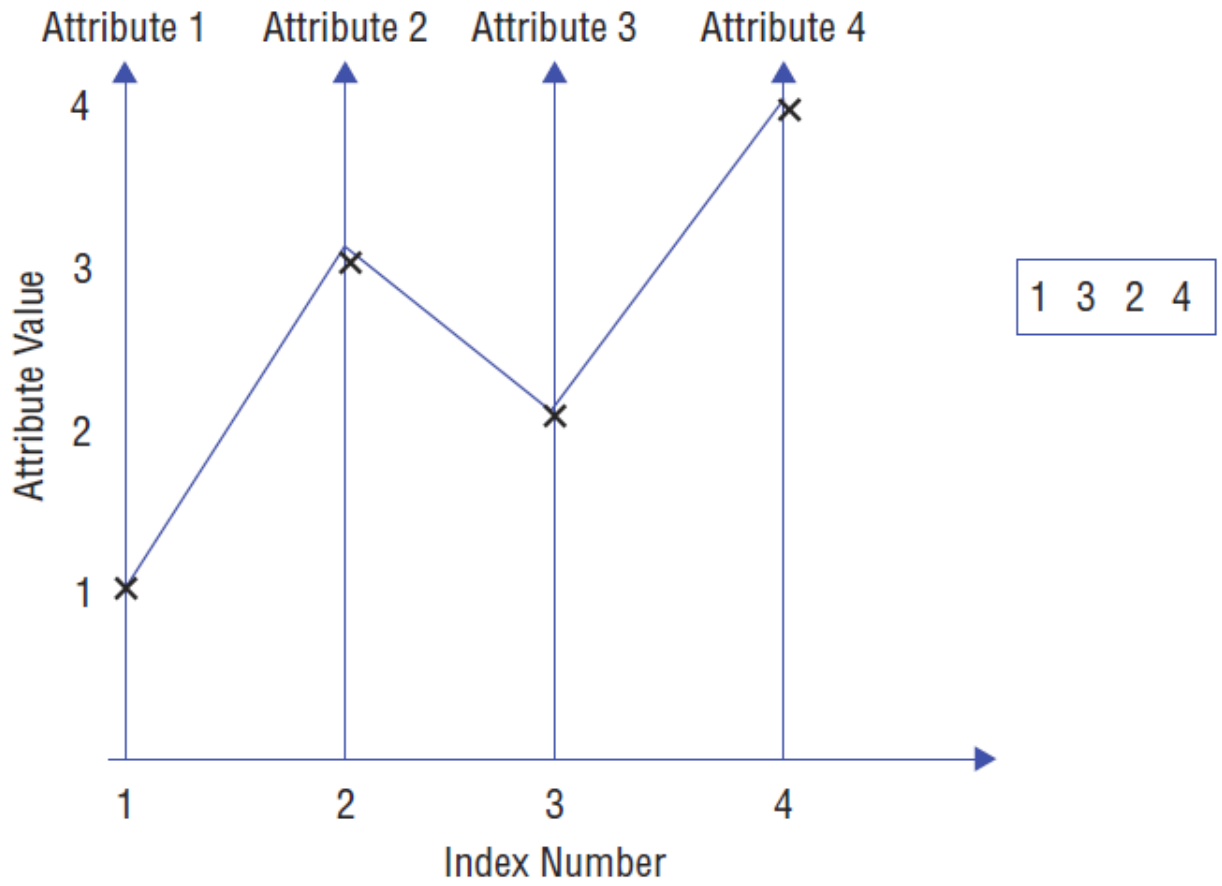


Figure 2-2: Constructing a parallel coordinates plot

Visualizing Properties of the Rocks versus Mines Data

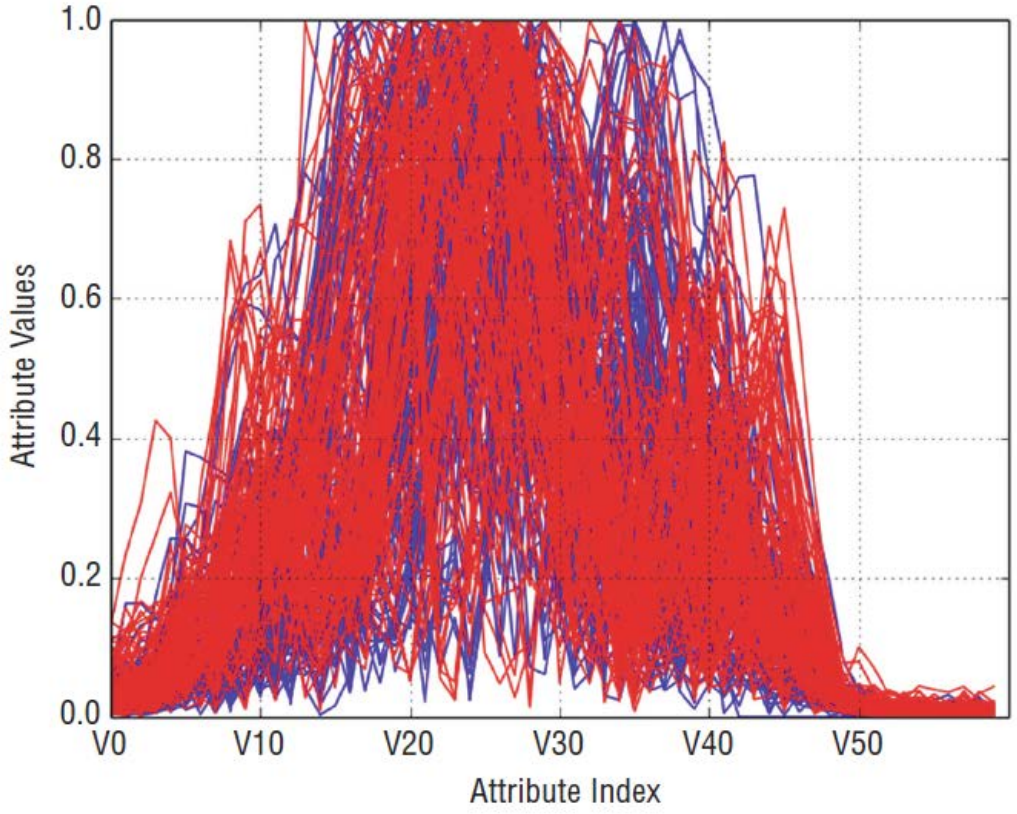


Figure 2-3: Parallel coordinates graph of rocks versus mines attributes

Visualizing Interrelationships between Attributes and Labels

Attributes 2 and 3

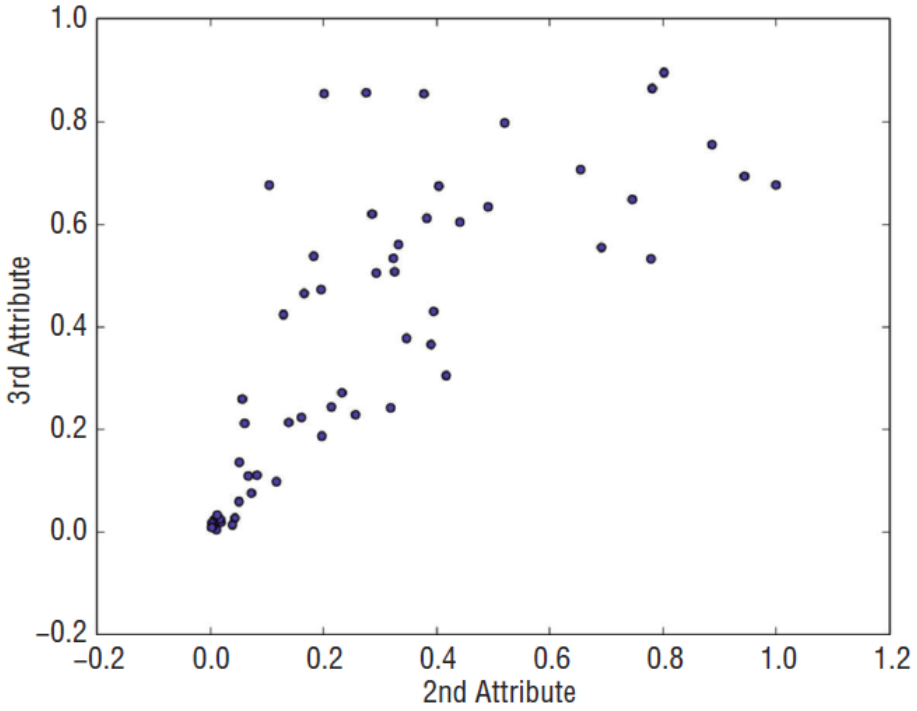


Figure 2-4: Cross-plot of rocks versus mines attributes 2 and 3

Attributes 2 and 21

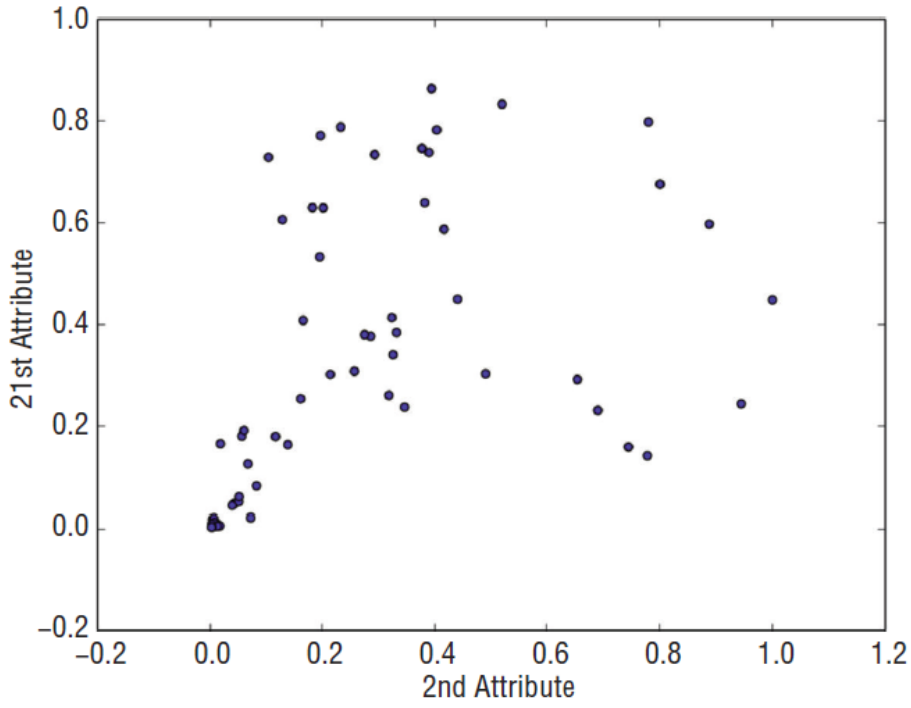
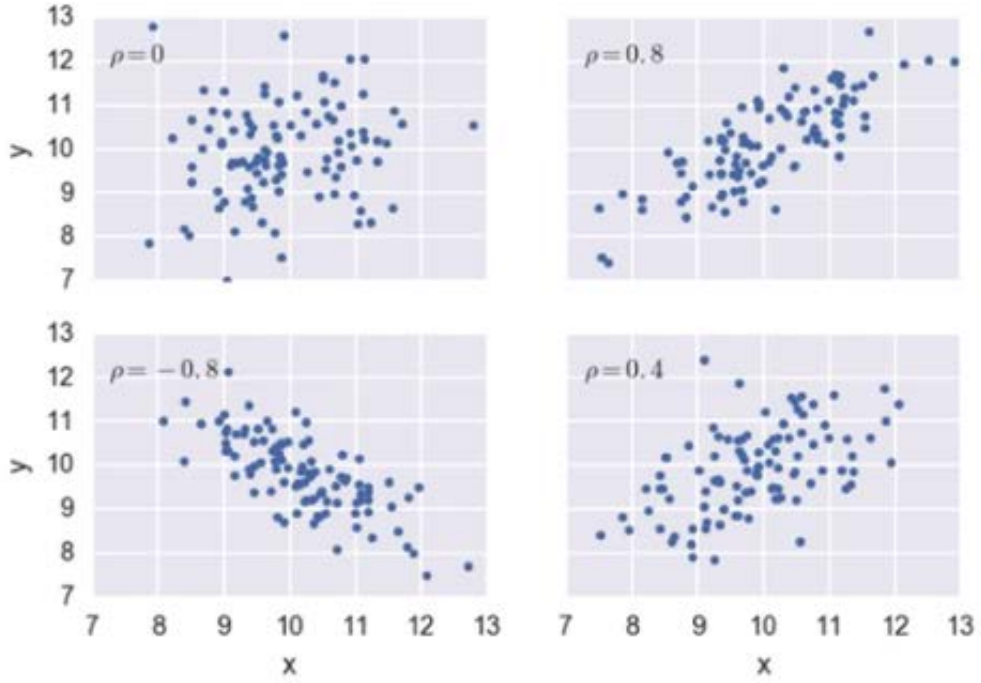


Figure 2-5: Cross- plot of rocks versus mines attributes 2 and 21

Pearson's correlation

$$\rho = \text{Pearson correlation} = \frac{\text{covariance}}{(\text{std of } x)\,(\text{std of } y)}$$



Visualizing Attribute and Label Correlations Using a Heat Map

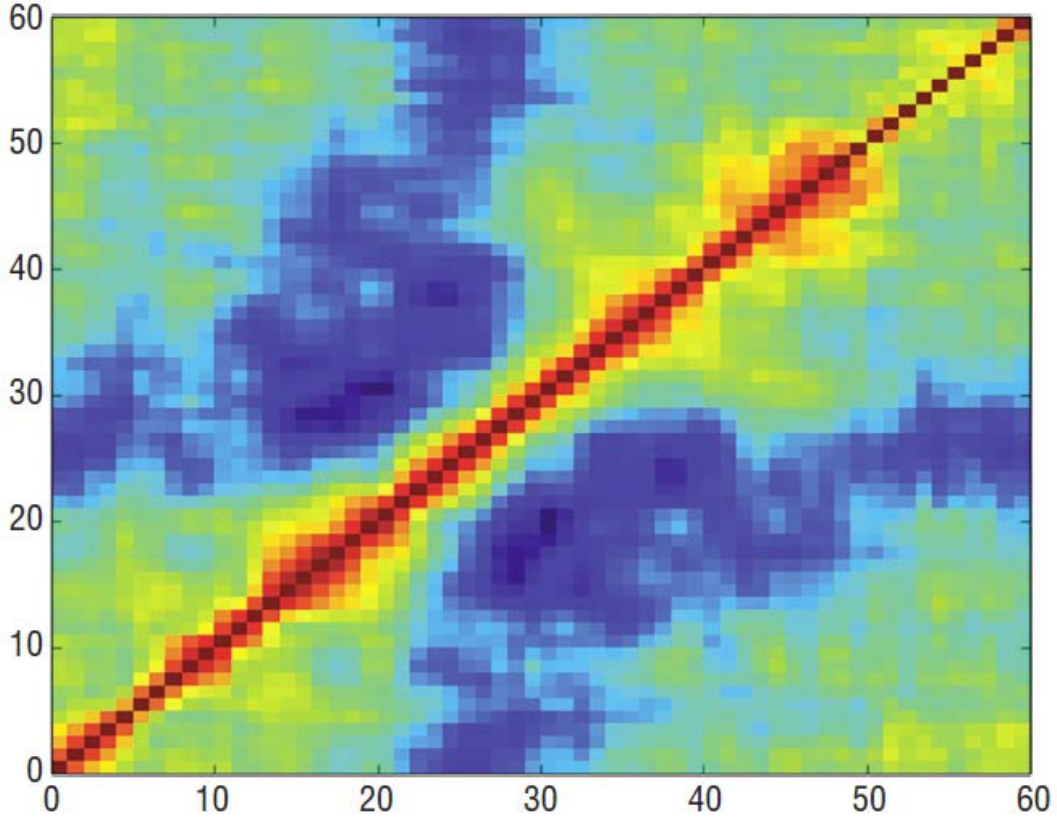


Figure 2-8: Heat map showing attribute cross-correlations

For next time

- Implement an EDA for the rocks v mines dataset
- Use the code presented in Listings 2-1 through 2-10
- Dataset is available from UCI or from Compass
- You may use pandas, numpy, scikit-learn, seaborn, matplotlib or any other packages you wish.
- We are not fitting any models this week, just producing EDAs

Bringing features onto the same scale

- Feature scaling is important for some models (KNN) and not others (decision trees)
- The majority of machine learning and optimization algorithms behave much better if features are on the same scale
- There are two common approaches to bring different features onto the same scale:

  normalization and **standardization**. Those terms are often used quite loosely in different fields, and the meaning has to be derived from the context.

Normalization

- Rescaling of the features to a range of [0, 1], which is a special case of **min-max scaling**

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{\min}}{x_{\max} - x_{\min}}$$

Here, $x^{(i)}$ is a particular sample, $x_{\min}$ is the smallest value in a feature column, and $x_{\max}$ the largest value.

The min-max scaling procedure is implemented in scikit-learn and can be used as follows:

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> mms = MinMaxScaler()
>>> X_train_norm = mms.fit_transform(X_train)
>>> X_test_norm = mms.transform(X_test)
```

Standardization

- can be more practical for many ML algorithms, especially for optimization algorithms such as gradient descent and logistic regression.
- centers the feature columns at mean 0 with standard deviation 1 so that the feature columns takes the form of a normal distribution.
- maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling, which scales the data to a limited range of values.
- standardization can be expressed by the following equation:

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

Feature scaling in scikit-learn

You can perform the standardization and normalization using the following code:

```
>>> ex = np.array([0, 1, 2, 3, 4, 5])
>>> print('standardized:', (ex - ex.mean()) / ex.std())
standardized: [-1.46385011 -0.87831007 -0.29277002  0.29277002
0.87831007  1.46385011]
>>> print('normalized:', (ex - ex.min()) / (ex.max() - ex.min()))
normalized: [ 0.   0.2  0.4  0.6  0.8  1. ]
```

Similar to the `MinMaxScaler` class, scikit-learn also implements a class for standardization:

```
>>> from sklearn.preprocessing import StandardScaler
>>> stdsc = StandardScaler()
>>> X_train_std = stdsc.fit_transform(X_train)
>>> X_test_std = stdsc.transform(X_test)
```

Last thoughts…

"Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone."

—John Tukey

Always label your axes! --- M.N. Lane

Deliverables due Sunday midnight

- Complete the online quiz
- Submit two posts to the discussion board – based on Tukey reading
- Homework has two parts this week:
  - Perform EDA on the rocks v mines dataset, upload code to Github, submit screenshots
  - Two chapters in Datacamp - Statistical Thinking in Python (Part 1)
  - Graphical EDA and Quantitative EDA.