Lecture 9:
• Quiz 4
• Intro to data visualization

Announcements:
• HW-3 out, due next Thurs
HW-3 pre-processing and exploration support functions

1. def distinct_values(table, column)
   • returns set of unique values in the given column of the table
   • can use Python sets to deal with duplicates
     
     ```python
     s = set()                      # create an empty set
     s.add(v)                      # add value v to the set (no change if v ∈ s)
     v in s                        # to check if value v is in the set
     ```

2. def remove_missing(table, columns)
   • removes all rows in the table with a missing value in one of the columns
   • where a missing value is represented as an empty string ("")

3. def duplicate_instances(table)
   • returns a data table with the duplicate rows in the given table
   • each duplicated row occurs only once in the result

4. def remove_duplicates(table)
   • removes the duplicate instances from the given table

5. def partition(table, columns)
   • returns a list of data tables whose rows form a partition of the original table
   • each partition (data table) has the same values for the given columns
   • this is similar to doing a “group by” – i.e., group rows by given columns
6. def summary_stat(table, column, function)
   • applies the function to the values in the given column (ignores missing values)
   • examples:
     summary_stat(t1, 'msrp', min)  ... find min 'msrp' value
     summary_stat(t1, 'msrp', sum)  ... return sum of all values
   • here function is "higher-order" ... function passed in as argument
   • simple example:
     def add_one(x):
       return x + 1
     def apply_fun(value, function):
       return function(value)  # note regular call
     y = apply_fun(3, add_one)  # y is now 4!

   • can also define "lambda" functions:
     add_one = lambda x: return x + 1
     sub_one = lambda x: return x - 1
     y = apply_fun(3, add_one)  # y is now 4!
     y = apply_fun(3, sub_one)  # y is now 3!
     y = apply_fun(3, lambda x: x*2)  # y is now 6!

7. def replace_missing(table, column, partition_columns, function)
   • replaces missing values in the column of the table
   • looks at “similar” rows based on partition columns
   • value used is result of applying function to “similar” rows
8. **def summary_stat_by_column(table, partition_col, stat_col, fun)**
   - returns two lists
   - one for each group value (given by partition column)
   - one for result of function applied to corresponding column row values

9. **def frequencies(table, partition_column)**
   - returns two lists
   - one for each group value (given by partition column)
   - the other that counts number of instances with that group value
Basic Data Visualization

We are going to use the matplotlib library ...

- to install: conda install -y matplotlib

We'll specifically use pyplot

```python
import matplotlib.pyplot as plt

xs = [1, 2, 3, 4, 5, 6, 7]
ys = [1, 1, 2, 3, 5, 8, 13]
plt.plot(xs, ys)
plt.show()
```

Can also save the “plot” to a file:

```python
import matplotlib.pyplot as plt

xs = [1, 2, 3, 4, 5, 6, 7]
ys = [1, 1, 2, 3, 5, 8, 13]
plt.plot(xs, ys)
plt.savefig('simple-plot.svg', format='svg')
```
Some notes on visualizations / graphing

1. Matplotlib is a large and comprehensive library …
   - we’ll only use a very small fraction of features

2. We will primarily be generating graphs, charts, plots
   - graph for (math) functions, plot for “points”, chart more general
   - e.g., line graph, scatter plot, pie chart, etc.

3. Be generous with labels!
   - Always label the plot/chart (title)
   - Always label all of the axes ... e.g., x and y axis
   - Always provide units (if applicable) ... unless obvious (like mpg)
   - If visualizing multiple things, good to provide a legend

4. Goal is to be informative and accurate
   - easy to “lie” with charts
   - classic example: change “scale” by cropping the $y$ and/or $x$ axes
Some basic pyplot functions:

- `plt.figure()` ... "resets" the current figure
- `plt.xlabel(text)` ... add x-axis label to figure
- `plt.ylabel(text)` ... add y-axis label to figure
- `plt.title(text)` ... add a title to figure
- `plt.grid(True)` ... add a background grid to chart
- `plt.grid(axis='y', color='0.85', zorder=0)` ... more options
- `plt.xticks(vals, labels)` ... set x-axis tick range/values
- `plt.yticks(vals, labels)` ... set y-axis tick range/values
- `plt.legend()` ... display/add a legend
1. Simple bar charts

The main function:

```python
plt.bar(xs, heights, width, align, ...)
```

Bar charts useful for comparing a category or value-range to a statistic

- each bar represents the statistic applied to instances in the category
- e.g.: Average MPG for each model year
- when the statistic is a count, often called a “histogram” (more later)

For more information (formatting options, etc):

```
matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html
```

includes additional examples (from bottom of page)
The full example:

```python
# reset figure
plt.figure()

# define x and y values
xs = [1, 2, 3, 4]
y = [100, 200, 400, 300]

# create a y-axis grid
plt.grid(axis='y', color='0.85', zorder=0)

# create the bar chart (.45 relative bar width)
plt.bar(xs, ys, width=0.45, align='center', zorder=3)

# define labels for x values
plt.xticks(xs, ['foo', 'bar', 'baz', 'qux'])

# x and y axis labels
plt.xlabel('Variables')
plt.ylabel('Values')

# title
plt.title('Example bar chart')

# display the plot
plt.show()
```

Bar charts are fairly generic

- we will see other types as well (histograms, box and whisker plots)

One way to title bar charts: "Y-values versus X-values"

- e.g.: Average Miles per Gallon (MPG) versus Number of Cylinders
2. Simple dot charts

The main function:

```python
plt.plot(xs, ys, color and shape, ...)
```

Useful for getting a quick sense of how data values are distributed

- useful for continuous data
- can also compare distributions by adding additional distributions
- but need to have a similar domain of values
The full example:

```python
# reset figure
plt.figure()

# define some x values
xs = [randint(0, round(i/2) for i in range(1,100)]

# create "dummy" y values
ys = [1] * len(xs)

# create the dot chart (alpha is transparency)
plt.plot(xs, ys, color='b', marker='.', alpha=0.2, markersize=16, linestyle='')

# get rid of the y axis
plt.gca().get_yaxis().set_visible(False)

# x label
plt.xlabel('Some values')

# title
plt.title('Example dot chart')

# display the plot
plt.show()
```

Title is often just what the values are

- e.g.: Miles per Gallon (MPG) Values
- or: Distribution of Miles per Gallon (MPG) Values
3. **Scatter plots**

One way to look at relationships between features

- primarily for comparing two continuous data attributes

![Example scatter plot](image)

The main function:

```python
plt.plot(xs, ys, color and shape, ...)
```

For more information:

The full example:

```python
# reset figure
plt.figure()

# define some x and y values
xs = [randint(0, round(i/2)) for i in range(1,100)]
ys = [randint(0, round(i/2)) for i in range(1,100)]

# add a grid
plt.grid(color='0.85', zorder=0)

# create the scatter plot
plt.plot(xs, ys, color='b', marker='.', alpha=0.2,
         markersize=16, linestyle='', zorder=3)

# x label
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# title
plt.title('Example scatter plot')

# display the plot
plt.show()
```

Similar to bar charts, can title as “Y-values vs X-values”
- e.g.: Miles per Gallon (MPG) versus Retail Price (MSRP)

Can change various style parameters
- change color, change marker, change transparency, marker size, add lines
- can also have multiple distributions, each with different styles, etc.
4. **Pie charts**

One way to display relative frequencies (as percentages)

- specifically the *percentage* of things by different partitions
- *note*: can use bar charts and “histograms” for frequencies as well

The main function:

```python
plt.pie(counts, labels=values, ...)
```

For more info:

- Variants: matplotlib.org/stable/gallery/pie_and_polar_charts/
The full example:

```python
# reset figure
plt.figure()

# define data
categories = ['a', 'b', 'c', 'd']
counts = [100, 200, 400, 300]

# create the bar chart (with pcts)
plt.pie(counts, labels=category, autopct='%1.1f%%')

# title
plt.title('Example pie chart')
```

Title should include how data is decomposed

- e.g.: Percentage of Cars by Model Year
5. **Box and whisker plots**

Boxplots display a summary of the dispersion (spread) of a distribution:

- a box representing interquartile range (2nd and 3rd quartiles)
- median marked by a line in the box (note median is 2nd quartile)
- two lines (called whiskers) outside box extend to smallest and largest values
- however, smallest and largest are those within $1.5 \times$ interquartile range
- remaining values displayed (marked) as “outliers”

*Typically one “box” per partition of a feature*

- e.g., MPG values of cars by number of cylinders

The main function:

```python
pyplot.boxplot(list_of_series)
```

For more information:

The full example:

```python
# reset figure
plt.figure()

# define data distributions
xs1 = [randint(0, round(i/2)) for i in range(1, 100)]
x2 = [randint(0, round(i/2)) for i in range(1, 100)]

# set a y-axis only grid
plt.grid(axis='y', color='0.85', zorder=0)

# create the box plot
plt.boxplot([xs1, xs2], zorder=3)

# set the x-axis distribution names
plt.xticks([1, 2], ['Partition 1', 'Partition 2'])

# set the x and y axis labels
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# title
plt.title('Example box plot')

# display plot
plt.show()
```

Plot title can use "Y-axis by X-axis"

- e.g.: Miles per Gallon by Number of Cylinders