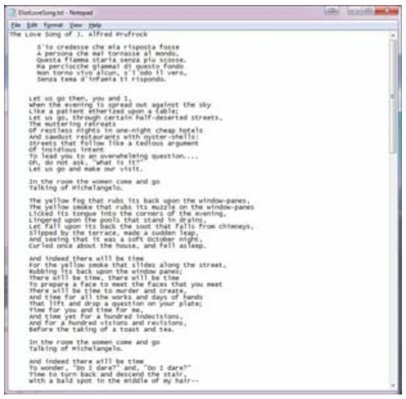
Review

- Data Visualization Process

Text Analysis/Text Mining

- Derive high-quality information on patterns and trends in the text via statistical pattern learning
  - Word frequency analysis
  - Sentiment analysis
  - Text categorization
  - Text clustering
- Related fields
  - Computational Linguistics
  - Natural Language Processing
  - Information Retrieval
  - Machine Learning
  - Artificial Intelligence

Acquire



EliotLoveSong.txt - Notepad

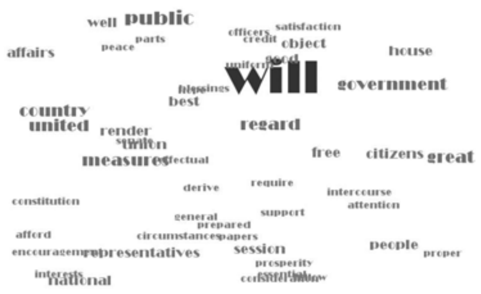
```
The Love Song of J. Alfred Prufrock

      S'io credesse che mia risposta fosse
      A persona che mai tornasse al mondo,
      Questa fiamma staria senza piu scosse.
      Ma perciocche giammai di questo fondo
      Non torno vivo alcun, s'i'odo il vero,
      Senza tema d'infamia ti rispondo.

    Let us go then, you and I,
    When the evening is spread out against the sky
    Like a patient etherized upon a table;
    Let us go, through certain half-deserted streets,
    The muttering retreats
    Of restless nights in one-night cheap hotels
    And sawdust restaurants with oyster-shells:
    Streets that follow like a tedious argument
    Of insidious intent
    To lead you to an overwhelming question....
    Oh, do not ask, "What is it?"
    Let us go and make our visit.

    In the room the women come and go
    Talking of Michelangelo.

    The yellow fog that rubs its back upon the window-panes,
    The yellow smoke that rubs its muzzle on the window-panes
    Licked its tongue into the corners of the evening,
    Lingered upon the pools that stand in drains,
    Let fall upon its back the soot that falls from chimneys,
    Slipped by the terrace, made a sudden leap,
    And seeing that it was a soft October night,
    Curled once about the house, and fell asleep.

    And indeed there will be time
    For the yellow smoke that slides along the street,
    Rubbing its back upon the window panes;
    There will be time, there will be time
    To prepare a face to meet the faces that you meet
    There will be time to murder and create,
    And time for all the works and days of hands
    That lift and drop a question on your plate;
    Time for you and time for me,
    And time yet for a hundred indecisions,
    And for a hundred visions and revisions,
    Before the taking of a toast and tea.

    In the room the women come and go
    Talking of Michelangelo.

    And indeed there will be time
    To wonder, "Do I dare?" and, "Do I dare?"
    Time to turn back and descend the stair,
    With a bald spot in the middle of my hair--
```

Parse and Filter

```
String raw;
String delimiters = " ,.?!;:-\'\"()*![]{}|\\~`@#$%^&";
String[] fileText, words;
int[] freqs;

void setup() {
  fileText = loadStrings("EliotLoveSong.txt");
  println("Read " + fileText.length + " lines.");

  raw = join(fileText, " ");
  raw = raw.toLowerCase();

  words = splitTokens(raw, delimiters);
  println("Found " + words.length + " words.");
}
```

Mine

```
String raw;
String delimiters = " ,.?!;:-\'\"()*![]{}|\\~`@#$%^&";
String[] fileText, words, uniqueWords;
int[] freqs;

void setup() {
 fileText = loadStrings("EliotLoveSong.txt");
  println("Read " + fileText.length + " lines.");

  raw = join(fileText, " ");
  raw = raw.toLowerCase();

  words = splitTokens(raw, delimiters);
  println("Found " + words.length + " words.");

  freqs = makeUnique(words);
  println("Found "+uniqueWords.length+" unique words.");
}
```



Data Structures

- Ways of storing and organizing data
- Arrays
  - Must know the size ahead of time
  - Can not grow and shrink at will

Built-in Collection Classes

- ArrayList
  - A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
  - Objects in an ArrayList are accessed by **index** [0..size-1]
- HashMap
  - A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
  - Objects in a HashMap are accessed by a **key**, which can be another Object, frequently a String.

ArrayList

- Constructors

```
ArrayList lst1 = new ArrayList();
ArrayList lst2 = new ArrayList(int initialSize);
```

- Fields
- Methods

```
size()                        // Returns the num of items held.
add(Object o)                 // Appends o to end.
add(int idx, Object o)        // Inserts o at pos idx.
remove(int idx)               // Removes item at pos idx.
get(int idx)                  // Gets items at idx. No removal.
set(int idx, Object o)        // Replaces item at idx with o.
clear()                       // Removes all items.
isEmpty()                     // true if empty.
toArray()                     // returns an array that contains
                              // the contents of the list
```

ArrayList Example – Box Dropper

```
// Box Dropper
ArrayList boxes = new ArrayList();

void setup() { size(500, 500); }

void draw() {
  background(0);

  for (int i = boxes.size()-1; i>=0; i--) {
    //boxes.get(i).draw();      // Fails. Why?
    Box b = (Box)boxes.get(i);  // Type cast Object->Box
    if(b.update()) {
      boxes.remove(i);
      println(boxes.size() + " boxes remaining");
    }
    else {
      b.draw();
    }
  }
}

void mousePressed() {
  Box b = new Box(mouseX, mouseY);
  boxes.add(b);
  println( boxes.size() + " boxes in ArrayList" );
}
```

```
// A simple Box class
class Box {
  float x, y, v;

  Box(float tx, float ty) {
    x = tx;  // x position
    y = ty;  // y position
    v = 0.0; // y velocity
  }

  void draw() {
    fill(200);
    rect(x, y, 20, 20);
  }

  boolean update(){
    y += v;
    v += 0.02;
    return (y>height);
  }
}
```

- Why can we not call draw directly on item in ArrayList?
- Why do we loop over ArrayList backwards?

ArrayList Example - Fireworks

HashMap

- Constructors

```
HashMap map1 = new HashMap();
HashMap map2 = new HashMap(int initialCapacity);
```

- Fields
- Methods

```
size()                        // Returns num of items held.
put(Object key, Object o)     // Puts o in map at key
remove(Object key)            // Remove Object at key
get(Object key)               // Get Object at key
containsKey(Object key)       // True if map contains key
containsValue(Object val)     // True if map contains val
clear()                       // Removes all items.
isEmpty()                     // true if empty.
```

HashMap Example – High Score

```
// HighScore
HashMap scores = new HashMap();

void setup() {
  size(500, 500);

  // Init HashMap
  scores.put("Fred", 2);
  scores.put("Wilma", 4);
  scores.put("Barney", 10);
  scores.put("Betty", 5);
  scores.put("BamBam", 6);
  scores.put("Pebbles", 5);

  // Draw once
  noLoop();
  drawMap(scores);
}

void draw() { }
```

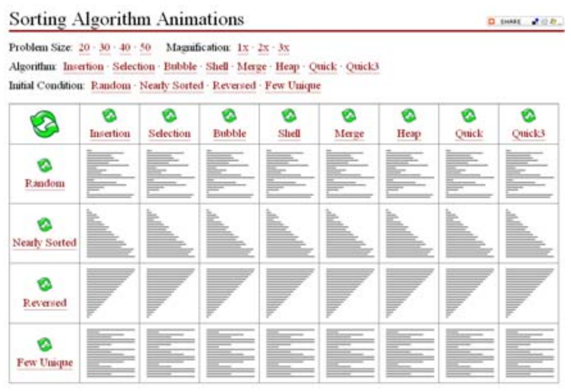
```
// Draw the HashMap to the sketch
void drawMap(HashMap hm) {
  background(0);
  fill(255);
  textSize(20);

  // Display all scores
  text( buildScore("Fred", scores), 100, 100);
  text( buildScore("Wilma", scores), 100, 150);
  text( buildScore("Barney", scores), 100, 200);
  text( buildScore("Betty", scores), 100, 250);
  text( buildScore("BamBam", scores), 100, 300);
  text( buildScore("Pebbles", scores), 100, 350);

  redraw();
}

// Build a return a String for displaying a Score
String buildScore(String name, HashMap hm) {
  String msg = name + ":" + hm.get(name).toString();
  return msg;
}
```

Sorting

- Any process of arranging items in sequence
- Build-in **`sort()`**
  - Works on arrays of simple types, i.e. `int`, `float` and `String`
  - `float[] a = { 3.4, 3.6, 2, 0, 7.1 };`
  - `a = sort(a);`
  - `String[] s = { "deer", "elephant", "bear", "aardvark", "cat" };`
  - `s = sort(s, 3);`
- Convenient, but not very flexible

Implement your own sort

- Many sorting algorithms
- Bubble Sort
  - Looks at items in successive pairs
  - Swap if in the wrong order
- Selection Sort
  - Scan a list top to bottom and find the value that should come first
  - Swap that item with the top position
  - Repeat scan starting at next lowest item in the list
  - Works best when swapping is expensive



Sorting Algorithm Animations

Problem Size: 20 · 30 · 40 · 50   Magnification: 1x · 2x · 3x

Algorithm: Insertion · Selection · Bubble · Shell · Merge · Heap · Quick · Quick3

Initial Condition: Random · Nearly Sorted · Reversed · Few Unique

http://www.sorting-algorithms.com/