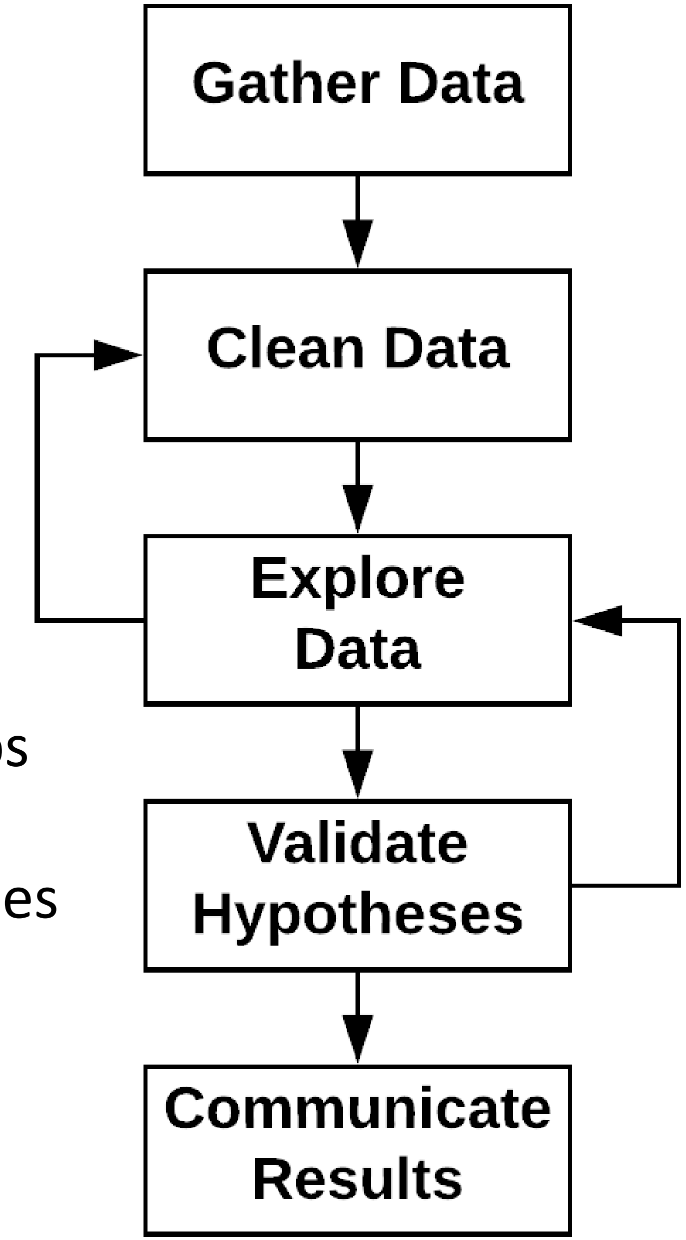
Data Cleaning

CSC2621 Introduction to Data Science

Data Science Process

- Application of the scientific method to data
- Data Science vs Machine Learning
  - Data Science: Understand the data and its Relationships better. Uses machine learning to explore data and validate those relationships
  - Machine Learning: Goal is to build a predictive model. Data Science is used to identify variables for the models and evaluate the models by creating an experimental framework



Data Cleaning Steps

1. Parse the file format to read in the data. At the end of this stage, for each record, you should be able to separate the values belonging to each field
2. Convert data to consistent representations (e.g., spelling of state names, format of dates)
3. Convert values to the right types (e.g., floats, datetimes, categorical)
4. Convert values to the right units (e.g., everything in inches)
5. Remove outlier records

Structuring Data

Properly Structured Data

- After we acquire the data, we need to clean it
- A clean data set is organized into a table such that:
  - Each sample is one row
  - Each variable is one column
  - All values are represented by their proper type (e.g., datetime, float, categorical)
  - All values in each column have the same types (e.g., int, float)
  - All values in each column have the same units (e.g., inches)

Representing Data as a Table

```
df.head(10)
```

	CHROM	POS	N_ALLELES	N_CHR	FREQ_1	FREQ_2
0	2R	808	2	2	1.0	0.0
1	2R	810	2	2	1.0	0.0
2	2R	821	2	2	1.0	0.0
3	2R	1297	2	4	0.5	0.5
4	2R	2124	2	2	0.5	0.5
5	2R	2400	2	2	1.0	0.0
6	2R	2740	2	4	0.0	1.0
7	2R	4485	2	2	0.0	1.0
8	2R	4800	2	2	0.0	1.0
9	2R	4803	2	2	0.0	1.0

Observations are in Rows

```
df.head(10)
```

	CHROM	POS	N_ALLELES	N_CHR	FREQ_1	FREQ_2
0	2R	808	2	2	1.0	0.0
1	2R	810	2	2	1.0	0.0
2	2R	821	2	2	1.0	0.0
3	2R	1297	2	4	0.5	0.5
4	2R	2124	2	2	0.5	0.5
5	2R	2400	2	2	1.0	0.0
6	2R	2740	2	4	0.0	1.0
7	2R	4485	2	2	0.0	1.0
8	2R	4800	2	2	0.0	1.0
9	2R	4803	2	2	0.0	1.0

Variables are in Columns

```
df.head(10)
```

	CHROM	POS	N_ALLELES	N_CHR	FREQ_1	FREQ_2
0	2R	808	2	2	1.0	0.0
1	2R	810	2	2	1.0	0.0
2	2R	821	2	2	1.0	0.0
3	2R	1297	2	4	0.5	0.5
4	2R	2124	2	2	0.5	0.5
5	2R	2400	2	2	1.0	0.0
6	2R	2740	2	4	0.0	1.0
7	2R	4485	2	2	0.0	1.0
8	2R	4800	2	2	0.0	1.0
9	2R	4803	2	2	0.0	1.0

Loading Data

Loading Data with Pandas

- Before we can access the data in a file, we need to be able to read it
- Generally, you can use:
  - df = pd.read_csv() – read a CSV or TSV file
  - df = pd.read_json() – read a JSON file
  - df = pd.read_excel() – read an Excel file
  - df = pd.read_sql() – read a result of a SQL query or table
- This only works if the file formats are properly formatted or the data is available in common file formats

Bad or Uncommon File Formats

- Files may have bad formats for multiple reasons
  - Excel spreadsheet doesn't strictly organize data into a single table of columns and rows
  - Someone wrote a custom JSON writer and didn't escape newlines and quote (") marks in Strings properly
  - Data is from multiple sources and written in a mix of encodings (e.g., UTF-8) in the same file
- Files may also be in uncommon file formats
  - e.g., domain-specific file formats like FASTA and VCF for genomic data

Loading Data (incorrectly)

```
df = pd.read_csv("/scratch1/rnowling/pedigree-linkage-mapping/linkage_mapping/data/mafs/Pfin_2.freq",
                 delim_whitespace=True)
```

Incorrectly Read Data

```
df.head(10)
```

	CHROM	POS	N_ALLELES	N_CHR	{FREQ}
2R	808	2	2	1.0	0.0
2R	810	2	2	1.0	0.0
2R	821	2	2	1.0	0.0
2R	1297	2	4	0.5	0.5
2R	2124	2	2	0.5	0.5
2R	2400	2	2	1.0	0.0
2R	2740	2	4	0.0	1.0
2R	4485	2	2	0.0	1.0
2R	4800	2	2	0.0	1.0
2R	4803	2	2	0.0	1.0

Raw Text File

```
rnowling@parker:/scratch1/rnowling/pedigree-linkage-mapping/linkage_mapping/data/mafs$ head 1997_BP02.freq
CHROM   POS     N_ALLELES       N_CHR   {FREQ}
2R      217     2       2       0.5     0.5
2R      356     2       2       1       0
2R      371     2       2       1       0
2R      1222    2       2       1       0
2R      1236    2       2       1       0
2R      1473    2       4       0       1
2R      3670    2       8       0       1
2R      3937    2       6       0.666667        0.333333
2R      4584    2       6       0       1
```

Loading Data (correctly)

```
df = pd.read_csv("/scratch1/rnowling/pedigree-linkage-mapping/linkage_mapping/data/mafs/Pfin_2.freq",
                  delim_whitespace=True,
                 header=None,
                  skiprows=1,
                 names=["CHROM", "POS", "N_ALLELES", "N_CHR", "FREQ_1", "FREQ_2"])
```

Correctly Read Data

```
df.head(10)
```

	CHROM	POS	N_ALLELES	N_CHR	FREQ_1	FREQ_2
0	2R	808	2	2	1.0	0.0
1	2R	810	2	2	1.0	0.0
2	2R	821	2	2	1.0	0.0
3	2R	1297	2	4	0.5	0.5
4	2R	2124	2	2	0.5	0.5
5	2R	2400	2	2	1.0	0.0
6	2R	2740	2	4	0.0	1.0
7	2R	4485	2	2	0.0	1.0
8	2R	4800	2	2	0.0	1.0
9	2R	4803	2	2	0.0	1.0

Custom Parsers

- In some cases, you will need to write custom code to read a file format and create the DataFrame yourself
- Store the data in lists (one per column)
- And create a DataFrame yourself:

```
df = pd.DataFrame({ "column1" : column1,
                    "column2" : column2 })
```

Custom Parsers

You can also create DataFrames from a list of tuples

```
rows = []
for ln in fl:
    ...
    rows.append((value1, value2, value3))
df = DataFrame.from_records(rows,
                            columns=["col_1", "col_2",
                                     "col_3"])
```

Basic Data Exploration

Look at Your Data!

- Look at the first few records (head() on the table)
- Use Pandas' info() method to get column info (e.g., name, type, nulls)
- Use Pandas' describe() method to get descriptive / summary statistics:
  - Average / mean
  - Standard deviation
  - Min, Max
  - Count, unique, mode for categorical types

Head

```
df.head(10)
```

	CHROM	POS	N_ALLELES	N_CHR	FREQ_1	FREQ_2
0	2R	808	2	2	1.0	0.0
1	2R	810	2	2	1.0	0.0
2	2R	821	2	2	1.0	0.0
3	2R	1297	2	4	0.5	0.5
4	2R	2124	2	2	0.5	0.5
5	2R	2400	2	2	1.0	0.0
6	2R	2740	2	4	0.0	1.0
7	2R	4485	2	2	0.0	1.0
8	2R	4800	2	2	0.0	1.0
9	2R	4803	2	2	0.0	1.0

Info

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3191955 entries, 0 to 3191954
Data columns (total 6 columns):
CHROM        object
POS          int64
N_ALLELES    int64
N_CHR        int64
FREQ_1       float64
FREQ_2       float64
dtypes: float64(2), int64(3), object(1)
memory usage: 146.1+ MB
```

Describe

```
df.describe()
```

	POS	N_ALLELES	N_CHR	FREQ_1	FREQ_2
count	3.191955e+06	3191955.0	3.191955e+06	3.191955e+06	3.191955e+06
mean	2.463528e+07	2.0	5.155352e+00	3.823677e-01	6.176323e-01
std	1.364909e+07	0.0	3.966836e+00	4.028822e-01	4.028822e-01
min	7.100000e+01	2.0	2.000000e+00	0.000000e+00	0.000000e+00
25%	1.365195e+07	2.0	2.000000e+00	0.000000e+00	2.500000e-01
50%	2.414605e+07	2.0	4.000000e+00	2.500000e-01	7.500000e-01
75%	3.490815e+07	2.0	6.000000e+00	7.500000e-01	1.000000e+00
max	6.154446e+07	2.0	4.000000e+01	1.000000e+00	1.000000e+00

Unification

Data Representation Unification

- Data of the same type might not be stored in a consistent format
- A common example are dates and times. Dates follow a number of patterns:
  - YYYY-MM-DD
  - MM-DD-YY
- We need to convert these to a single representation before we can parse them
- We call this unification

Data Representation Unification

- Another example are state names:
  - Mississippi
  - MS
  - Miss.
- People can easily misspell or use different abbreviations for state names
- We call this name unification

Handling Varying Data Representations

- Your goal is to convert every representation to a single, canonical representation for each piece of information (e.g., single state name)
- For example, with dates, you would need to write a custom function to guess the date format for each String and parse it to produce a DateTime object accordingly
- In many cases, the various representations do not belong to a standard so you need to use a guess-and-check approach
  - Look at the data by eye!

Example: Parsing Date and Times

```
def parse_date(date_string):
    """
    Parse datetimes like

    'Tue, 10 Apr 2007 19:25:38 +0000'
    'Fri, 18 May 2007 12:15:02 -0700 (PDT)'
    '14 May 2007 04:27:55 +0000'
    '2007-06-02 17:35:21'
    """

    if date_string is None:
        return None

    # remove trailing "(GMT)"
    if date_string.endswith(")"):
        idx = date_string.rfind(" ")
        date_string = date_string[:idx]

    # remove preceding day of the week
    if "," in date_string:
        idx = date_string.find(",") + 2
        date_string = date_string[idx:]

    # remove timezone offset
    idx = date_string.rfind(" ")
    date_string = date_string[:idx]

    try:
        fmt = "%d %b %Y %H:%M:%S"
        return dt.datetime.strptime(date_string, fmt)
    except:
        try:
            fmt = "%Y-%m-%d %H:%M:%S"
            return dt.datetime.strptime(date_string, fmt)
        except:
            return None
```

Choosing the Right Types

- Text (String)
- Numerical
  - Integer (int32, int64)
  - Float (float32, float64)
- Boolean (bool)
- Categorical
- Date / Times
  - timedelta64[ns]
  - datetime64[ns]
  - Timestamp

Choosing the Right Types

- Pandas tries to infer the data types when using load_csv() or similar
- Is it is likely that Pandas will guess incorrectly at times
- You can change types using astype():

```
df["at_bats"] = df["at_bats"].astype(np.float32)
df["state"] = df["state_names"].astype("category")
```

Should It Be Categorical?

- Some data sets encoded categorical data as integers
  - 1 – Red
  - 2 – Green
  - 3 – Blue
- This data is mis-encoded. Integers imply an ordering. Categories do not have orderings.
- Similarly, Strings are not usable for plots, statistical tests, or machine learning.
- If the values of a String field contains many duplicated values (e.g., state names), it should probably be categorical.

Text Data

- If String data is complex (e.g., street addresses, email bodies, or comments), it cannot be used directly
- Variables can be extracted from text
- For example, given addresses like "1234 Madison St.", we can extract:
  - Home numbers
  - Street names
  - Street types (street, court, way, etc.)

Unit Conversions

- All of the values in a single field should be in the same unit (e.g., inches, dollars)
- Date times and currencies are usually the most common problems:
  - Times may be in different time zones. Time zones are very complicated (based on geography, time of year)
  - International data will use different currencies. Currency exchange rates vary in time.

Bad or Missing Values

Errors vs Artifacts

- Fields can contain bad values:
  - Dates outside of expected ranges (e.g., in the future)
  - NaNs
  - Missing values
- We can classify causes into two categories:
  - Errors
  - Artifacts

Errors vs Artifacts

Errors

- Information fundamentally lost in acquisition
- E.g., Missing logs because a server crashed
- Information cannot be recovered

Artifacts

- Systematic problems arising from processing
- e.g., bad date formats
- Cannot be corrected so long as the original data is still available

Missing Values

- Fields may contain missing values
- Statistical and machine learning methods do not work with nulls
- We can check for missing values (represented as nulls):

```
In [6]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61097 entries, 0 to 61096
Data columns (total 7 columns):
bodies            61097 non-null object
spam              61097 non-null int64
to                60709 non-null object
from              61097 non-null object
message_number    61097 non-null int64
user_agent        61097 non-null object
date              61096 non-null object
dtypes: int64(2), object(5)
memory usage: 3.3+ MB
```

Missing Values: Meaningful

- Missing values have multiple causes
- Missing value has a structural meaning:
  - "Prior activity" field in glucose notebook. Missing value means no prior activity was performed.
  - Survey questions. A survey may have a question that should only be answered if a previous question was answered.
  - These missing values should be instead represented by a placeholder categorical value ("no activity", "not applicable")

Missing Values: Data Cleaning Errors

- Missing values may be present because of badly-formatted data
- Data cleaning processes are often "guess and check"
  - Fail on edge cases and needs to be updated
  - To keep the data cleaning process from crashing, bad values are skipped and replaced with nulls
  - Need to fix the data cleaning process and re-process the data

Missing Values: Just Missing

- In the third case, missing values are just missing
  - e.g., server crashed and logs are lost for a period of time
- Remove records with missing values across many fields
- Impute the values
  - Replace the missing values with an estimated value
  - Need to think carefully about which imputation strategies make sense

Imputation: Mean or Mode

- A simple imputation strategy
  - Calculate the mean, median, or mode of known values
  - Replace missing values with calculated value
- Tends to be safe in that imputed values will not bias ML models
- Most common practice
- May not be appropriate:
  - e.g., using an average birth year for for historical figures

Imputation: Nearest Neighbor

- More complex strategy
  - Find similar records using other fields with known values
  - Use known values from similar records to impute missing value
- Not many good (robust) implementations available
  - May fail when a large number of records have missing values
- Complex
  - Requires transforming data (feature engineering, scaling) to work well
  - Basically doing machine learning at that point

When to Impute

- When in the data science process should you impute?
- Many resources describe imputation during the exploratory data analysis process; I disagree
- The goal of many analyses is to use data we have to make predictions about data we haven't seen yet
  - Imputing missing values using all data will violate our experimental setup (train / test split) for machine learning
  - Better to impute using the training set
- EDA is generally concerned with one or two variables at a time – easier to just ignore missing values in those columns when making plots
- Most plotting libraries ignoring missing values by default