MESSIAH UNIVERSITY | DEPARTMENT OF COMPUTING, MATHEMATICS AND PHYSICS

wisdom expertise

How to clean the data

Prasanna Ranjith Christodoss
Associate Professor of Computer Science
Messiah University, PA - USA

Objectives

Applied

1. Identify the data problems in a DataFrame by using the info(), unique(), nunique(), and value_counts() methods.
2. Remove duplicate rows from a DataFrame.
3. Drop unnecessary rows and columns in a DataFrame.
4. Rename columns so the names are easier to understand.
5. Find and fix the missing values in a DataFrame.
6. Find and fix datetime and numeric columns that are imported as object data types.
7. Replace the invalid values in a column and convert the column to the right data type.
8. Find and fix outliers.

Objectives (continued)

Knowledge

1. Describe the ways that you can simplify the data in a DataFrame.
2. Describe the ways that you can handle missing values.
3. Describe two common data type problems and how best to fix them.
4. Describe the ways to handle outliers.
5. Describe the category data type.

A general plan for cleaning a DataFrame

Identify the problems

1. Review whatever documentation is available to help you understand the data.
2. Examine the data after you import it into a DataFrame.
3. Plan the cleaning that needs to be done.

Simplify the data

4. Drop duplicate rows.
5. Drop rows that aren't needed for the analysis.
6. Drop columns that aren't needed for the analysis.
7. Rename columns so the names are easier to understand.

A general plan for cleaning a DataFrame

Fix data problems

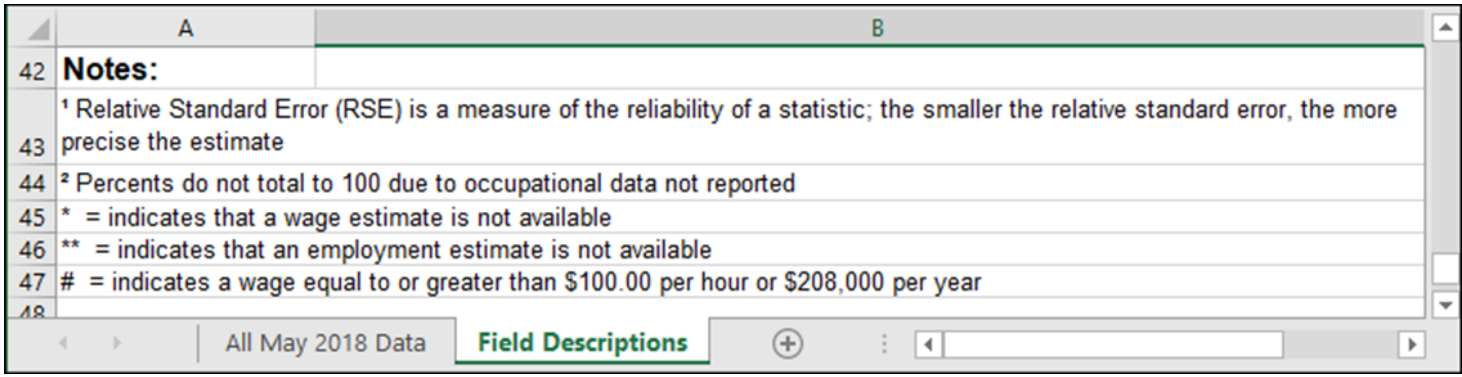
8. Find and fix missing values.
9. Fix data type problems like dates or numbers that are imported as strings.
10. Find and fix outliers.

But before you start cleaning...

- Be sure that you've set the goals for your analysis.

That will help you decide what cleaning needs to be done.

Documentation that identifies some data problems in the Jobs data



	A	B
42	**Notes:**	
43	¹ Relative Standard Error (RSE) is a measure of the reliability of a statistic; the smaller the relative standard error, the more precise the estimate	
44	² Percents do not total to 100 due to occupational data not reported	
45	* = indicates that a wage estimate is not available	
46	** = indicates that an employment estimate is not available	
47	# = indicates a wage equal to or greater than $100.00 per hour or $208,000 per year	
48		

All May 2018 Data | Field Descriptions

Three parameters of the info() method

Parameter	Description
`verbose=True`	Ensures that the method returns info for all of the columns.
`memory_usage='deep'`	Ensures that the memory usage information is accurate.
`show_counts=True`	Ensures that the non-null counts for each column will be displayed.

Partial results for the info() method when run on the Polls data

```
polls.info(verbose=True, memory_usage='deep', show_counts=True)
============================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12624 entries, 0 to 12623
Data columns (total 27 columns):
forecastdate        12624 non-null object
startdate           12624 non-null object
enddate             12624 non-null object
samplesize          12621 non-null float64
rawpoll_mcmullin    90 non-null float64
adjpoll_mcmullin    90 non-null float64
dtypes: float64(10), int64(3), object(14)
memory usage: 12.9 MB
```

Likely dates stored as strings

3 rows have null values: Why?

Mostly null values: Why?

Partial results for the info() method when run on the Jobs data

```
jobs.info(verbose=True, memory_usage='deep', show_counts=True)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 403895 entries, 0 to 403894
Data columns (total 30 columns):
area              403895 non-null int64
loc_quotient      227944 non-null object
pct_total         169080 non-null object
a_mean            403895 non-null object
a_median          403895 non-null object
annual            15682 non-null object
hourly            729 non-null object
dtypes: int64(3), object(27)
memory usage: 629.5 MB
```

Likely dates stored as strings

Mostly null values: Why?

What you should be looking for

- Object or string variables that look like they should be dates or numbers.
- Non-null counts that are lower than the non-null counts of other columns may indicate missing data.
- Small numbers of non-nulls or nulls may indicate problems with related columns or rows.

The nunique() and unique() methods

Method	Description
`nunique(dropna=False)`	Returns the number of unique values in each column.
`unique()`	Returns the unique values in each column.

Partial results for the nunique() method when run on the Polls data

```
polls.nunique()
==========================================================================
cycle                    1
branch                   1
type                     3
matchup                  1
forecastdate             1
state                   57
startdate              352
enddate                345
pollster               196
samplesize            1767
population               4
rawpoll_clinton       1312
rawpoll_trump         1385
...
multiversions            2
poll_id               4208
question_id           4208
```

One unique value: Delete?

Small number of uniques: Often groups or categories.

id columns: These have 1/3 as many uniques as the dataset has rows. Why?

Partial results for a unique() method that gets the unique values

```
polls.apply(pd.unique)
```

```
==========================================================================
cycle                                                             [2016]
branch                                                       [President]
type                                  [polls-plus, now-cast, polls-only]
matchup                                  [Clinton vs. Trump vs. Johnson]
forecastdate                                                 [11/8/2016]
state              [U.S., New Mexico, Virginia, Iowa, Wisconsin, ...
startdate          [11/3/2016, 11/1/2016, 11/2/2016, 11/4/2016, 1...
enddate            [11/6/2016, 11/7/2016, 11/5/2016, 11/4/2016, 1...
pollster           [ABC News/Washington Post, Google Consumer Sur...
grade                     [A+, B, A-, B-, A, nan, B+, C+, C-, C, D]
samplesize         [2220.0, 26574.0, 2195.0, 3677.0, 16639.0, 129...
population                                                 [lv, rv, a, v]
poll_wt            [8.720654, 7.628472, 6.424334, 6.087135, 5.316...
rawpoll_clinton    [47.0, 38.03, 42.0, 45.0, 48.0, 44.0, 46.0, 41...
rawpoll_trump      [43.0, 35.69, 39.0, 41.0, 44.0, 40.0, 42.7, 46...
```

The value_counts() method

Method	Description
`value_counts(params)`	Returns the count for each of the unique values in a series.

Parameters of the value_counts() method

Parameter	Description
`normalize`	If True, returns the counts as a percent of the total rather than the actual counts.
`dropna`	If True (the default), it ignores null values. Can only be used with Series objects.

How to use the value_counts() method

```
polls.state.value_counts().head(4)
==================================
U.S.                3318
Florida              444
Pennsylvania         375
North Carolina       375
Name: state, dtype: int64
```

How to use the normalize parameter

```
polls.state.value_counts(normalize=True).head(4)
=================================================
U.S.                0.262833
Florida             0.035171
Pennsylvania        0.029705
North Carolina      0.029705
Name: state, dtype: float64
```

How to use the dropna parameter

```
polls.multiversions.value_counts(dropna=False)
==============================================
NaN    12588
*         36
Name: multiversions, dtype: int64
```

How to use the value_counts() method with multiple columns

```
polls[['state','grade']].value_counts()
========================================
state          grade
U.S.           A-        1215
               C+         546
               B          162
               C-         126
               A+         114
                         ...
Nebraska CD-1  B            3
Indiana        A            3
Nebraska CD-2  B            3
Missouri       A            3
Mississippi    C            3
Name: count, Length: 321, dtype: int64
```

How to drop rows based on conditions

By directly accessing a column

```
polls = polls[polls.type == 'now-cast']
```

With the query() method

```
polls = polls.query('type == "now-cast"')
```

How to drop rows based on conditions

```
import pandas as pd
# Creating a simple bookstore dataset
data = {
    'Title': ['Python Basics', 'Data Science 101', 'Old Ledger',
                                'Advanced AI', 'Empty Journal'],
    'Price': [25.00, 40.00, 15.00, 60.00, 0.00],
    'In_Stock': [True, True, False, True, False],
    'Category': ['Tech', 'Tech', 'History', 'Tech', 'Misc']
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

How to drop rows based on conditions

Boolean indexing

```
# We keep only the rows where In_Stock is True
df_cleaned = df[df['In_Stock'] == True]

print("After dropping out-of-stock books:")
print(df_cleaned)
```

Query() method

```
# We query for books where price is greater than 0
df_cleaned = df.query('Price > 0')

print("After dropping free/zero-price books:")
print(df_cleaned)
```

Dropping Rows Using the .drop() Method

```
# Dropping by index label
df_dropped = df.drop(2)
print("After dropping index 2:")
print(df_dropped)
```

The duplicated() method

Method	Description
`duplicated(params)`	Identifies duplicated rows.

Parameters of the duplicated() method

Parameter	Description
`subset`	List of columns to be used for identifying duplicates; otherwise, all columns are used.
`keep`	If 'first' (the default), all duplicates are marked as True except for the first occurrence in each set. If 'last', all duplicates are marked as True except for the last occurrence. If False, all duplicates are marked as True.

The drop_duplicates() method

Method	Description
`drop_duplicates(params)`	Returns the DataFrame with duplicate rows removed.

Parameters of the drop_duplicates() method

Parameter	Description
`subset`	Same as for the duplicated() method.
`keep`	Same as for the duplicated() method except that the duplicate rows are dropped, not marked.

How to drop duplicate rows

How to find duplicate rows

```
fires[fires.duplicated(keep=False)]
```

How to drop duplicate rows

```
fires = fires.drop_duplicates(keep='first')
```

The drop() method

Method	Description
`drop(params)`	Drops the columns that are specified by the columns parameter.

Parameters of the drop() method

Parameter	Description
`columns`	List of columns to be dropped.
`errors`	What to do if a specified column doesn't exist: 'ignore' (the default) or 'raise'.

How the nunique() method helps you identify columns that you may want to drop

```
============================================================
cycle                    1
branch                   1
type                     3
matchup                  1
forecastdate             1
state                   57
startdate              352
enddate                345
pollster               196
grade                   10
samplesize            1766
population               4
...
multiversions            2
poll_id               4208
question_id           4208
```

Only one value in each column

One unique value in each row after the DataFrame has been cleaned down to 4208 rows

How to use the drop() method to drop columns

```
polls = polls.drop(columns=['cycle','forecastdate'])
```

The rename() method for columns

Method	Description
`rename(params)`	Renames the columns of a DataFrame.

Parameters of the rename() method

Parameter	Description
`columns`	A dictionary that contains the columns to be renamed and the new names for those columns.

How to rename columns with the rename()

How to rename columns with an embedded dictionary

```
polls = polls.rename(columns={
    'rawpoll_clinton':'clinton_pct',
    'rawpoll_trump':'trump_pct'})
```

How to rename columns with a separate dictionary

```
polls_names_dict = {'rawpoll_clinton':'clinton_pct',
                    'rawpoll_trump':'trump_pct'}
polls = polls.rename(columns = polls_names_dict)
```

How to rename columns with the Python replace() method

How to use a simple expression with the replace() method

```
polls.columns = polls.columns.str.replace('_pct',’’)
```

For example, a column named clinton_pct would become simply clinton

How to use regular expressions with the replace()

```
jobs.columns = jobs.columns \
    .str.replace('^a_','annual_', regex=True) \
    .str.replace('^h_','hourly_', regex=True) \
    .str.replace('_pct','_percent')
```

The isnull(), notnull(), and any() methods

Method	Description
`isnull()`	Returns a Boolean same-sized object indicating whether the values are NA.
`notnull()`	Returns a Boolean same-sized object indicating whether the values are not NA.
`any(axis=1)`	Returns True if any column in a row contains a missing value.

A ten-row DataFrame named mortality_data that has some missing values

	Year	AgeGroup	DeathRate
0	1900.0	01-04 Years	1983.8
1	NaN	01-04 Years	1695.0
2	1902.0	01-04 Years	NaN
3	1903.0	01-04 Years	1542.1
4	NaN	01-04 Years	NaN

	Year	AgeGroup	DeathRate
5	1905.0	01-04 Years	1498.9
6	NaN	01-04 Years	NaN
7	1907.0	01-04 Years	NaN
8	1908.0	01-04 Years	1396.8
9	1909.0	01-04 Years	1348.9

How to display the count of the missing values

```
missing_count = mortality_data.shape[0] - mortality_data.count()
print(missing_count)
```

```
==========================================================
Year          3
AgeGroup      0
DeathRate     4
```

How to display all rows that contain NA values

```
mortality_data[mortality_data.isnull().any(axis=1)]
```

	Year	AgeGroup	DeathRate
1	NaN	01-04 Years	1695.0
2	1902.0	01-04 Years	NaN
4	NaN	01-04 Years	NaN
6	NaN	01-04 Years	NaN
7	1907.0	01-04 Years	NaN

How to display all rows with NA values in the DeathRate column

```
mortality_data[mortality_data.DeathRate.isnull()]
```

How to display all rows that don't have NA values in the DeathRate column

```
mortality_data[mortality_data.DeathRate.notnull()]
```

The dropna() method

Method	Description
`dropna(params)`	Drops the rows that are specified by the parameters.

Parameters of the dropna() method

Parameter	Description
`subset`	A list of the columns that the drop should be based on.
`how`	If 'any' (the default), a row is dropped if any column contains an NA value. If 'all', a row is dropped only if all columns contain NA values.
`thresh`	An integer that specifies how many columns need to contain NA values before the row is dropped.

How to drop all rows that contain an NA value

```
mortality_data = mortality_data.dropna()
```

	Year	AgeGroup	DeathRate
0	1900.0	01-04 Years	1983.8
3	1903.0	01-04 Years	1542.1
5	1905.0	01-04 Years	1498.9
8	1908.0	01-04 Years	1396.8
9	1909.0	01-04 Years	1348.9

How to drop only the rows that have two or more NA values

```
mortality_data = mortality_data.dropna(thresh=2)
```

How to drop only the rows that have NAs in the DeathRate column

```
mortality_data = mortality_data.dropna(subset=['DeathRate'])
```

The ffill() method

Method	Description
`ffill(params)`	Forward fills the NA values in a DataFrame with the value in the row before the missing value.

Parameters of the ffill() method

Parameter	Description
`limit`	The maximum number of consecutive NAs in a column to fill.

How to forward fill NA values with a limit of 2 consecutive fills

```
mortality_data = mortality_data.ffill(limit=2)
mortality_data.head(8)
```

	Year	AgeGroup	DeathRate
0	1900.0	01-04 Years	1983.8
1	1900.0	01-04 Years	1695.0
2	1902.0	01-04 Years	1695.0
3	1903.0	01-04 Years	1542.1
4	1903.0	01-04 Years	1542.1
5	1905.0	01-04 Years	1498.9
6	1905.0	01-04 Years	1498.9
7	1907.0	01-04 Years	1498.9

The interpolate() method

Method	Description
`interpolate()`	Uses linear interpolation to fill the missing values in a DataFrame.

How to fill NA values with linear interpolation

```
mortality_data[['Year','DeathRate']] =
    mortality_data[['Year','DeathRate']].interpolate()
mortality_data.head(4)
```

	Year	AgeGroup	DeathRate
0	1900.0	01-04 Years	1983.80
1	1901.0	01-04 Years	1695.00
2	1902.0	01-04 Years	1618.55
3	1903.0	01-04 Years	1542.10

Example for ffill() and interpolate()

```
import pandas as pd
import numpy as np
data = {
    'Time': ['9:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00'],
    'Temp': [70, 72, np.nan, np.nan, np.nan, np.nan, 80]
}
df = pd.DataFrame(data)
```

Try these

```
df['Temp'].ffill()
df['limit_result'] = df['Temp'].ffill(limit=2)
df['interp_result'] = df['Temp'].interpolate()
```

The data types that Pandas applies to imported columns

Data type	When the column contains…
`int64`	Only integers or NAs
`float64`	Only floating-point numbers or NAs
`bool`	Only Boolean values (True or False)
`object`	Strings or data that doesn't import as another data type

The select_dtypes() method of a DataFrame

Method	Description
`select_dtypes(type)`	Selects all columns with the specified data type.

Polls data: Dates stored as objects

```
polls.select_dtypes('object').head(2)
```

	branch	type	matchup	state	startdate	enddate	pollster	grade	population	multiversions	
4208	President	now-cast	Clinton vs. Trump vs. Johnson	U.S.	11/3/2016	11/6/2016	ABC News/ Washington Post	A+	lv	NaN	https://www.washingt
4209	President	now-cast	Clinton vs. Trump vs. Johnson	U.S.	11/1/2016	11/7/2016	Google Consumer Surveys	B	lv	NaN	https://datastudio.googl

Jobs data: Numeric and Boolean values stored as objects

```
jobs.select_dtypes('object').head(2)
```

	area_title	naics	naics_title	i_group	occ_code	occ_title	o_group	tot_emp	emp_prse	jobs_1000	...	h_median	h_percent75	h_percent90	a_perc
0	U.S.	000000	Cross-industry	cross-industry	00-0000	All Occupations	total	144733270	0.1	NaN	...	18.58	30.06	47.31	
1	U.S.	000000	Cross-industry	cross-industry	11-0000	Management Occupations	major	7616650	0.2	NaN	...	50.11	72.93	#	

How Pandas applies data types to imported columns

- If a column consists entirely of numbers and NA values, Pandas imports the column with either the int64 or float64 data type.
- If a column consists entirely of True and False values, Pandas imports the column with the bool data type.
- Pandas imports all other columns with the object data type.

The Pandas to_datetime() method

Method	Description
`to_datetime()`	Converts columns that contain date strings to datetime objects.

Four of the date columns in the polls DataFrame

```
date_cols =
['startdate','enddate','createddate','timestamp']
```

	startdate	enddate	createddate	timestamp
4208	11/3/2016	11/6/2016	11/7/16	09:24:53 8 Nov 2016
4209	11/1/2016	11/7/2016	11/7/16	09:24:53 8 Nov 2016

How to convert dates to datetime objects with inferred formatting

```
polls[['startdate','enddate']] =
    polls[['startdate','enddate']].apply(pd.to_datetime)
polls[['timestamp']] =
polls[['timestamp']].apply(pd.to_datetime)
```

	startdate	enddate	createddate	timestamp
4208	11/3/2016	11/6/2016	11/7/16	09:24:53 8 Nov 2016
4209	11/1/2016	11/7/2016	11/7/16	09:24:53 8 Nov 2016

A few of the formatting codes for the Python strftime() method

Code	Description
`%m`	The month of the year as an integer
`%d`	The day of the month as an integer
`%y`	A 2-digit year
`%Y`	A 4-digit year

How to convert dates to datetime objects with specified formatting

```
polls[['createddate']] =
    polls[['createddate']].apply(pd.to_datetime,
format='%m/%d/%Y')
```

	startdate	enddate	createddate	timestamp
4208	2016-11-03	2016-11-06	2016-11-07	2016-11-08 09:24:53
4209	2016-11-01	2016-11-07	2016-11-07	2016-11-08 09:24:53

How to use the Python strftime() method to format dates as strings

```
polls['startdate'] = polls.startdate.dt.strftime("%m/%d/%Y")
polls['enddate'] = polls.enddate.dt.strftime("%m-%d-%y")
```

	startdate	enddate	createddate	timestamp
4208	11/03/2016	11-06-16	2016-11-07	2016-11-08 09:24:53
4209	11/01/2016	11-07-16	2016-11-07	2016-11-08 09:24:53

The to_numeric() method

Method	Description
`to_numeric(columns,errors)`	Tries to convert the data in a column to a numeric type.

Three options for the errors parameter

Option	Description
`raise`	Raises an error if there is any non-numeric data in a column except for NA values. This is the default.
`ignore`	Ignores the error and doesn't change the column at all.
`coerce`	Converts non-numeric values to NaN.

How to convert columns with valid numeric strings to numeric data types

```
pd.to_numeric(jobs.tot_emp)
```

How to coerce columns with invalid strings into numeric data types

The original data

```
jobs.tot_emp.tail(3)
==========================================================
403892    170
403893    130
403894     **
Name: tot_emp, dtype: object
```

Without an errors parameter, an error is raised

```
pd.to_numeric(jobs.tot_emp)
==========================================================
ValueError: Unable to parse string "**" at position 1906
```

With the errors parameter set to coerce

```
pd.to_numeric(jobs.tot_emp, errors='coerce').tail(3)
==========================================================
403892    170.0
403893    130.0
403894      NaN
Name: tot_emp, dtype: float64
```

The astype() method for converting data types

Method	Description
`astype(type,errors)`	Tries to convert the data to the specified data type. Errors are raised by default, but they can be ignored by setting the errors parameter to 'ignore'.

How to convert object columns to the category data type

```
fires.state = fires.state.astype('category')
```

Example – astype()

```
import pandas as pd
data = {
    'Product_ID': ['101', '102', '103', '104'],          # Currently Strings
    'Price': ['12.50', '15.00', '9.99', '20.00'],        # Currently Strings
    'In_Stock': [1, 0, 1, 1],                            # Currently Integers (1/0)
    'Department': ['Books', 'Tech', 'Books', 'Tech']  # Currently Strings
}

df = pd.DataFrame(data)

# Before conversion, we can't do math on 'Price'
# print(df['Price'].sum())
```

Example – astype()

```
import pandas as pd
data = {
   'Product_ID': ['101', '102', '103', '104'],          # Currently Strings
   'Price': ['12.50', '15.00', '9.99', '20.00'],        # Currently Strings
   'In_Stock': [1, 0, 1, 1],                            # Currently Integers (1/0)
   'Department': ['Books', 'Tech', 'Books', 'Tech']  # Currently Strings
}

df = pd.DataFrame(data)

# Before conversion, we can't do math on 'Price'
# print(df['Price'].sum())
```

```
# Convert Price to float (decimal) and ID to int
df['Price'] = df['Price'].astype(float)
df['Product_ID'] = df['Product_ID'].astype(int)

# Convert to Boolean
df['In_Stock'] = df['In_Stock'].astype(bool)

# Convert to Category
df['Department'] = df['Department'].astype('category')
```

The remove_unused_categories() method

Method	Description
`remove_unused_categories()`	Removes unused categories from a table.

How to drop rows for specific categories

Drop the rows for the categories that aren't in the query

```
filteredStatCounts = statCounts.query(
    'wrkstat in ["working fulltime","working
parttime","retired"]')
filteredStatCounts.wrkstat.value_counts()
=========================================================
=========
retired              31892
working parttime      9121
working fulltime      6719
other                    0
keeping house            0
school                   0
unempl, laid off         0
temp not working         0
Name: wrkstat, dtype: int64
```

How to drop rows for specific categories (cont.)

Drop the unused categories

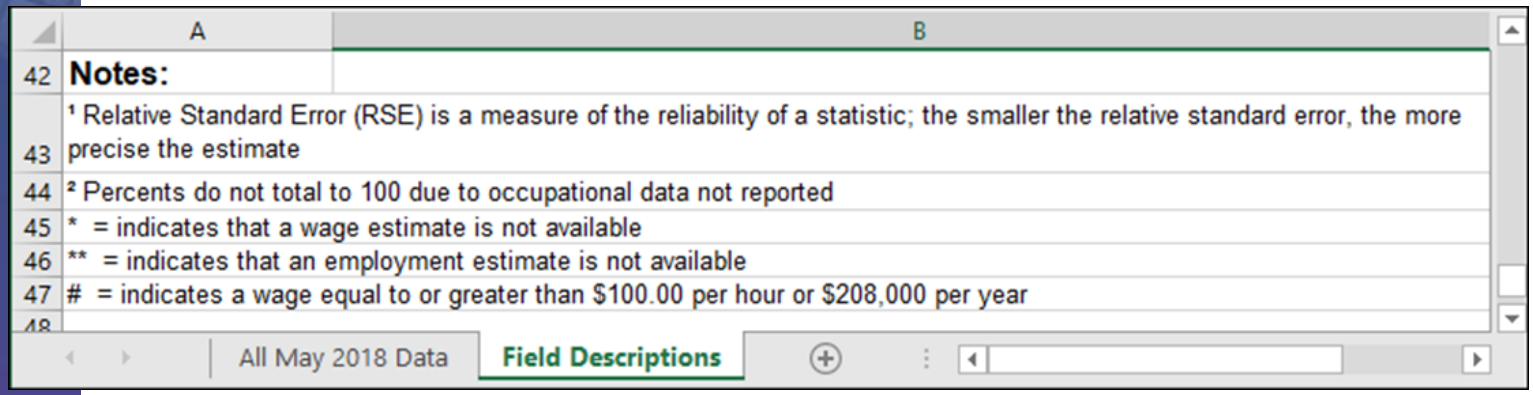
```
filteredStatCounts['wrkstat'] = \

filteredStatCounts.wrkstat.cat.remove_unused_categories()
filteredStatCounts.wrkstat.value_counts()
=========================================================
=
retired              31892
working parttime      9121
working fulltime      6719
Name: wrkstat, dtype: int64
```

The Pandas replace() method

Method	Description
`replace(to_replace,value)`	Replaces the strings in the to_replace parameter with the strings in the value parameter or replaces the old values in a dictionary with the new values.

Documentation that identifies some data problems in the Jobs data



	A	B
42	**Notes:**	
43	¹ Relative Standard Error (RSE) is a measure of the reliability of a statistic; the smaller the relative standard error, the more precise the estimate	
44	² Percents do not total to 100 due to occupational data not reported	
45	* = indicates that a wage estimate is not available	
46	** = indicates that an employment estimate is not available	
47	# = indicates a wage equal to or greater than $100.00 per hour or $208,000 per year	

All May 2018 Data | Field Descriptions

How to replace * or ** values in a single column with NaN

```
import numpy as np
jobs.tot_emp = jobs.tot_emp.replace(to_replace=['*','**'],
                    value=[np.nan,np.nan])
```

How to use a dictionary to replace * or ** values in a single column with NaN

```
jobs.tot_emp = jobs.tot_emp.replace({'*':np.nan,
   '**':np.nan})
```

How to replace # values with a numeric value

```
jobs.h_median = jobs.h.replace(
    to_replace='#', value=100)
```

How to replace NaN values with False in what should be a Boolean column

```
jobs.hourly = jobs.hourly.replace(to_replace=np.nan,
   value=False)
```

Some useful parameters of the read_csv()

Parameter	Description
`header`	The row and column names where the import should start.
`usecols`	The columns to be imported.
`nrows`	The number of rows to import.
`parse_dates`	The columns that the to_datetime() method should try to convert.
`na_values`	The strings that should be treated as NAs.
`true_values`	The values that the dataset uses for True.
`false_values`	The values that the dataset uses for False.

How to parse object columns that contain dates and times

The date columns to be parsed

```
date_cols = ['forecastdate','startdate','enddate','createddate','timestamp']
```

The url for the file

```
polls_url = '../../data/president_polls_2016.csv'
```

An import with no changes

```
polls = pd.read_csv(polls_url)
polls[date_cols].head(3)
```

	forecastdate	startdate	enddate	createddate	timestamp
0	11/8/16	11/3/2016	11/6/2016	11/7/16	09:35:33 8 Nov 2016
1	11/8/16	11/1/2016	11/7/2016	11/7/16	09:35:33 8 Nov 2016
2	11/8/16	11/2/2016	11/6/2016	11/8/16	09:35:33 8 Nov 2016

How to parse object columns that contain dates and times (continued)

An import with parsed dates

```
polls_new = pd.read_csv(polls_url, parse_dates=date_cols)
polls_new[date_cols].head(3)
```

	forecastdate	startdate	enddate	createddate	timestamp
0	2016-11-08	2016-11-03	2016-11-06	2016-11-07	2016-11-08 09:35:33
1	2016-11-08	2016-11-01	2016-11-07	2016-11-07	2016-11-08 09:35:33
2	2016-11-08	2016-11-02	2016-11-06	2016-11-08	2016-11-08 09:35:33

The mortality data for the 15-19 age group

```
mortality_group = \
    mortality_data.query('AgeGroup == "15-19 Years"')
mortality_group.head(3)
```

	Year	AgeGroup	DeathRate
357	1900	15-19 Years	484.8
358	1901	15-19 Years	454.4
359	1902	15-19 Years	421.5

How to use the describe() method to find outliers

```
mortality_group.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Year	119.0	1959.00000	34.496377	1900.0	1929.50	1959.0	1988.50	2018.0
DeathRate	119.0	177.37479	138.362290	44.8	85.45	106.9	283.65	777.4

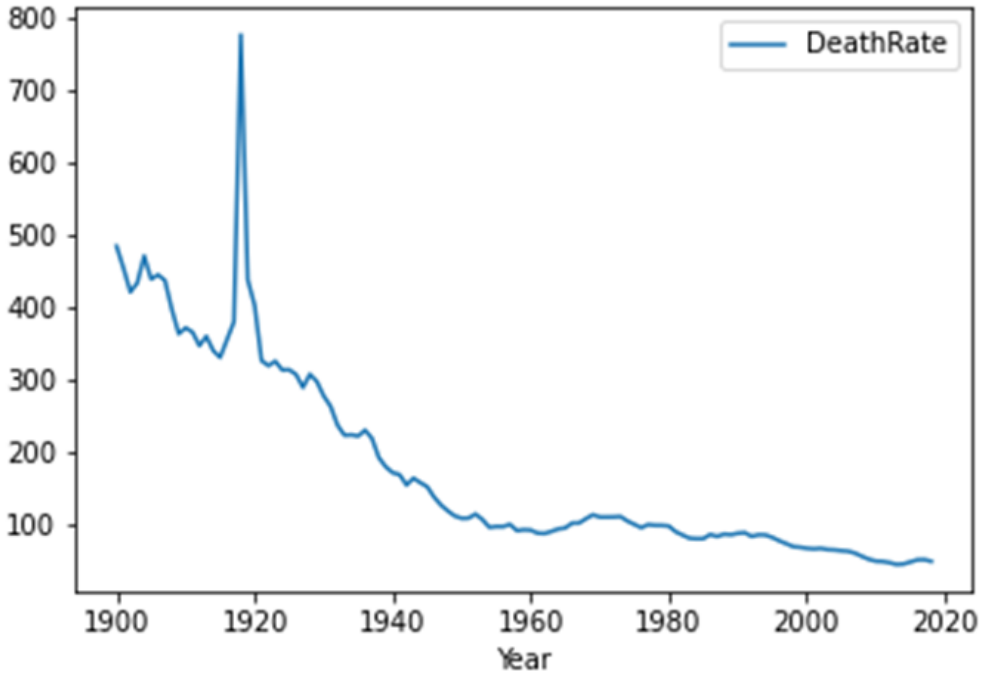
Find outliers in the below dataframe

```
import pandas as pd
data = {
    'Runner_ID': [1, 2, 3, 4, 5],
    'Age': [24, 29, 31, 22, 145],
    'Finish_Time_Min': [240, 255, 230, 210, 10]
}
df = pd.DataFrame(data)

# Transpose the summary
print(df.describe().T)
```

How to use the plot() method to find outliers

```
mortality_group.plot(x='Year', y='DeathRate')
```



How to review the outliers in the DeathRate column

```
mortality_group.query('DeathRate > 500')
```

	Year	AgeGroup	DeathRate
375	1918	15-19 Years	777.4

Three ways to handle outliers that are valid

- Drop the outliers.
- Cap the outliers at values that won't skew the results.
- Assign new values to the outliers like the mean of the column or a linear interpolation.

How to drop the rows for the outliers

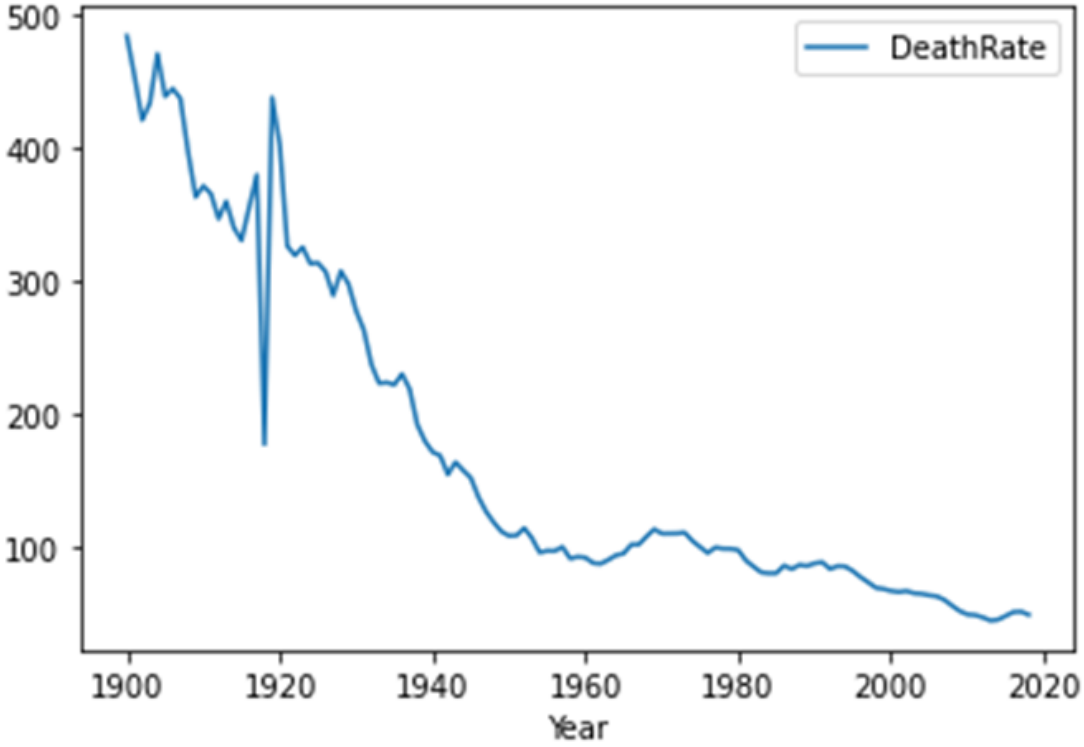
```
mortality_group = mortality_group.query('DeathRate <= 500')
```

How to cap the values of the outliers at 450

```
mortality_group.loc[mortality_group.DeathRate > 500,'DeathRate'] = 4
```

How to assign the mean of the column to the outliers

```
mortality_group.loc[mortality_group.DeathRate > 500,'DeathRate'] \
    = mortality_group.DeathRate.mean()
mortality_group.plot(x='Year', y='DeathRate')
```



How to use interpolation to smooth the outliers

```
mortality_group.loc[mortality_group.Year.isin([1917,1918,1919,1920]),
                    'DeathRate'] = None
mortality_group = mortality_group.interpolate()
mortality_group.plot(x='Year', y='DeathRate')
```