**STATISTICAL FUNDAMENTALS**

**Data Wrangling and Storytelling: Exploratory Data Analysis**

Learn

Learn to Submit assignments via GitHub (save work to GitHub)

GitHub is a website where you can save code or other files either for personal use or for sharing with others. The website is used primarily for storing “open-source” project files so that users can work together on large code bases without overwriting each other’s work. You will be using GitHub to collaborate on large projects, both with other students and in your career.

Overview

In order to help you get familiar with this tool we have structured our assignment submission process around the typical GitHub workflow to try and mimic how this tool is used. The following process is the workflow that you will follow in order to submit your assignments so that the Team Leads can view your work and give you daily feedback.

Follow Along

1) Fork the Repository for that Sprint at the beginning of the Sprint

**NOTE: You will only do this step a single time at the beginning of each sprint.**

Go to [http://github.com/lambdaschool](https://github.com/lambdaschool)

All of our data science curriculum can be accessed through this page.

In the search bar start typing:

DS-Unit-1-Sprint-1-Data-Wrangling-and-Storytelling

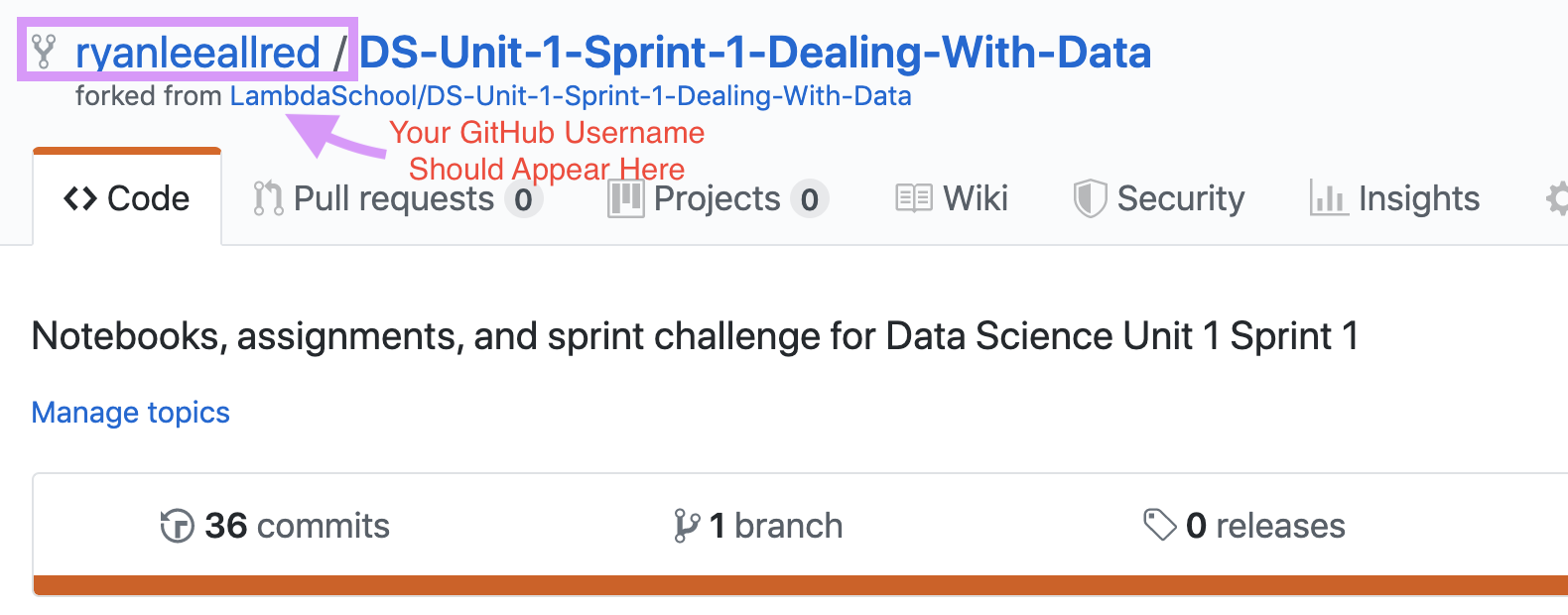
Repositories that don’t match what you are typing in the search bar will be filtered out, eventually leaving this sprint’s repository.

“Repository” is fancy work that just means: folder where we are going to store some files on GitHub. You’ll hear people say “repo” for short.

**At the beginning of each sprint you will need to find that sprint’s corresponding repository and “fork” it to your personal GitHub account.** “Forking” a repository is GitHub lingo for “Make a copy.” If you click the fork button on the top right corner of the webpage, GitHub will make a copy of the folder of files that we will be using for that sprint to your personal GitHub account. You will be doing your work and saving your changes to the copied version on your account.

You can tell when you have successfully forked a repository because you should briefly see an animation appear that looks like a book is being photocopied with a fork stuck in it and then you will be redirected to your copy of the repository.

You can always tell when you’re looking at the forked version on your personal github account by looking at the name of the repository and looking at the username that is just to the left of it in the filepath:



2) Open one of the files and make a change to it.

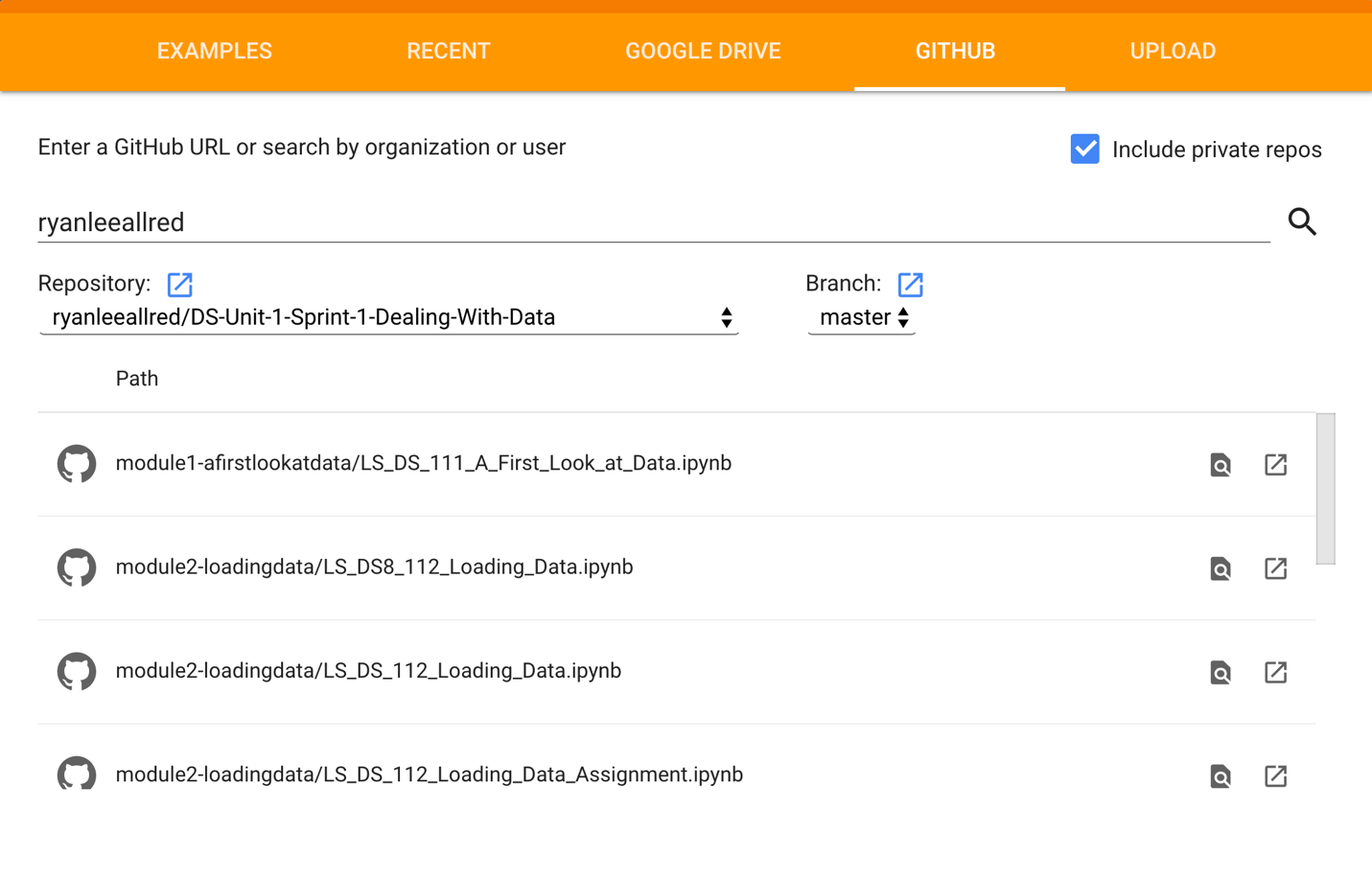
The files that we will be working with primarily during the course have the file extension: .ipynb for “IPython Notebook” any of these are notebooks that we can open in Google Colab.

To open one of these notebook files in Google Colab go to:

<https://colab.research.google.com/github/>

If you haven’t done so already, give Google permission to access your GitHub account from your Google Account.

Once you have all of the permissions sorted out, select the repository that you’re most interested in from the dropdown menu. Once you select a repository Google Colab will look through it to find all of the .ipynb files and will list them below:



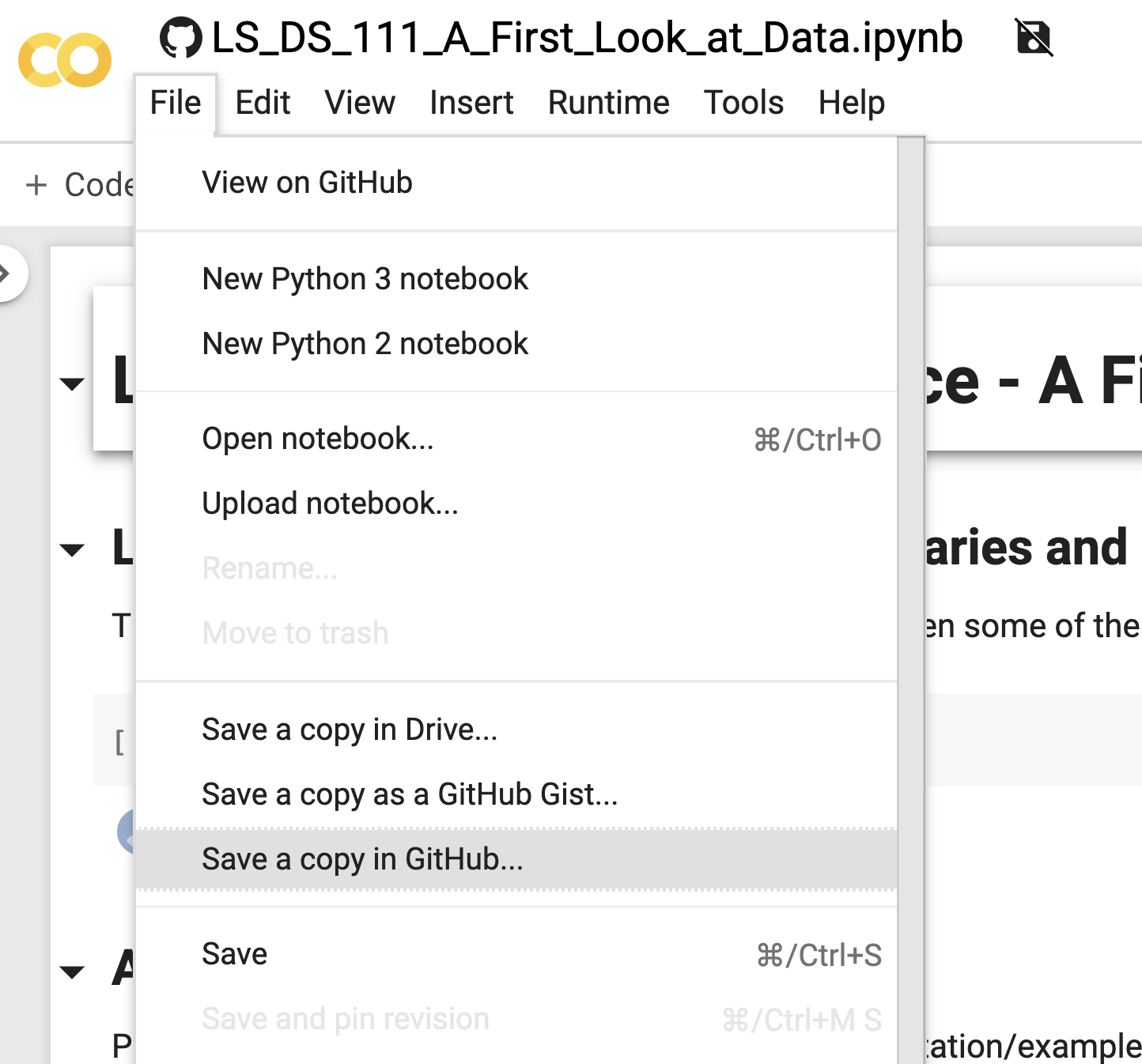
If you don’t like going to this link everyday to open your notebooks, there is also a Google Chrome extension that you can use to easily open any .ipynb file from GitHub directly in Google Colab:

[Google Chrome Extension to Open .ipynb files easily in Google Colab](https://chrome.google.com/webstore/detail/open-in-colab/iogfkhleblhcpcekbiedikdehleodpjo?hl=en)

3) Save your changes back to your forked repository on Github.

Once you have finished making all of the changes that you want to the notebook, you can save your work back to GitHub by selecting File » Save a copy in GitHub from the dropdown menu.

When you select this a new tab will open in your browser to show you the saved file on GitHub to let you know that the save has been completed successfully.



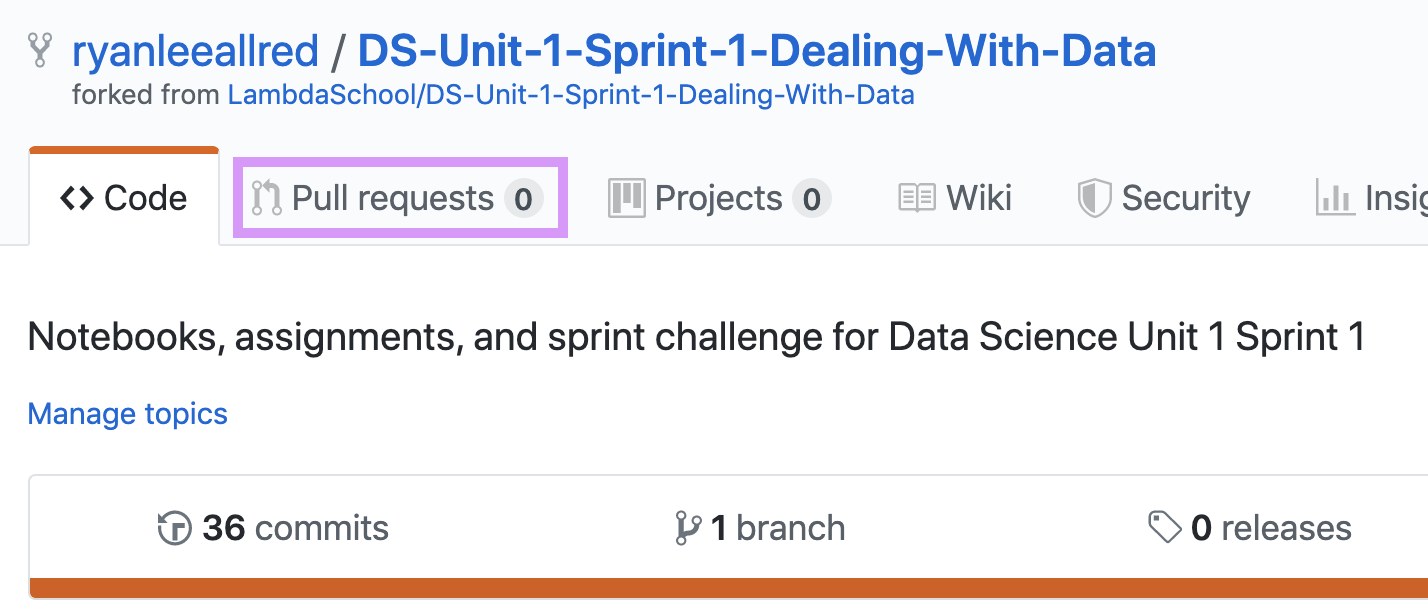
You will do steps 2 and 3 of this process every day as you work on your assignment work, however, you will only do steps 1 and 4 at the beginning of a sprint.

4) Submit a “Pull Request” of your work.

The final step in submitting your work is to open a “Pull Request” GitHub won’t allow you to complete this step until you have saved some changes to your version of the repository on GitHub.

Opening a Pull Request is something that only needs to be done once per week (typically at the beginning of the week). This pull request is what ties your work back to the original Lambda School repository and makes it easy for the Team Leads to find your work.

In order to open a pull request, navigate to your repository on GitHub and select the “Pull Requests” tab at the top of the page.



To open a new pull request you will need to click the green “New Pull Request” button and give your pull request a title. Please include your name and Cohort number i.e. DS8, DS9, or DS10, etc. at the beginning of the pull request title so that the Team Leads can easily identify your Pull Request. Once you have filled out the title, just click the remaining large green buttons until the pull request has been submitted.

In Summary

1) Fork the Repository (make a copy to your personal account)

2) Open the Repository in Google Colab and make changes to the files (work on your assignment).

3) Save the changes back to github using the dropdown menu.

4) Make sure that sometime before the end of the first day of the sprint that you have submitted a Pull Request so that the TLs can find your work.

Challenge

You’ll have to do follow this process or one very close to it every day/week for the next nine months. If this feels a little bit overwhelming at first, don’t worry about it! We will be doing this everyday and you have your Team Leads and classmates to lean on for help. You’ll be a pro at using GitHub in no time.

If you’re already familiar with GitHub and or Git via the command line, feel free to use the tools that you are most comfortable with, but you still need to save your work to GitHub every day.

Learn to Load a dataset (CSV) from a URL using pandas.read\_csv()

Many datasets are hosted online for easy consumption, as such it’s important that we know how to load a dataset via its url. This is perhaps the easiest way to load a dataset into a Pandas DataFrame via the pd.read\_csv() method.

Overview

In order to practice Loading Datasets into Google Colab, we’re going to use the Flags Dataset from UCI to show both loading the dataset via its URL and from a local file.

Follow Along

Steps for loading a dataset:

1) Learn as much as you can about the dataset:

* Number of rows
* Number of columns
* Column headers (Is there a “data dictionary”?)
* Is there missing data?
* **OPEN THE RAW FILE AND LOOK AT IT. IT MAY NOT BE FORMATTED IN THE WAY THAT YOU EXPECT.**

2) Try loading the dataset using pandas.read\_csv() and if things aren’t acting the way that you expect, investigate until you can get it loading correctly.

3) Keep in mind that functions like pandas.read\_csv() have a lot of optional parameters that might help us change the way that data is read in. If you get stuck, google, read the documentation, and try things out.

4) You might need to type out column headers by hand if they are not provided in a neat format in the original dataset. It can be a drag.

Challenge

You’ll get very good at reading documentation, Googling, asking for help, troubleshooting, debugging, etc. by the time you’re done here at Lambda School. Our goal is to turn you into a data scientist that can solve their own problems.

Learn to Load a dataset (CSV) from a local file using pandas.read\_csv()

We won’t always have CSVs hosted on the interwebs for us. We need to be able to upload files from our local machines as well. With Google Colab this is trickier than it is with other software (like Jupyter Notebooks for example. Because the main file system backing Google Colab is Google Drive, we can’t use a filepath to the file on our computers in order to access our data. We have to upload our files to Google Colab before we can start working with them.

Overview

There are two main methods of uploading local files to Google Colab. One is to use the code snippet

Copy

from google.colab import files

uploaded = files.upload()

And then use the prompt to select a file from your computer.

The second method involves using the Google Colab GUI interface found in the left-hand sidebar to upload and manage files.

Follow Along

Method 1: Use the GUI tool in the left-hand sidebar of Google Colab to upload a CSV file to the notebook’s memory and then simply pass in a string of the filename to pd.read\_csv()

Method 2:

Run the following code snippet in a code cell in order to be prompted with an upload form that you can use to browse your machine and upload the file to Google Colab.

Copy

from google.colab import files

uploaded = files.upload()

Challenge

On the assignment this afternoon you’ll get to choose a new dataset and try both of these methods, we will load hundreds of datasets into notebooks by the time the class is over, you’ll be pro at it in no time.

Learn to Use basic Pandas functions for Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypotheses and to check assumptions with the help of summary statistics and graphical representations.

Overview

Exploratory Data Analysis is often the first thing that we’ll do when starting out with a new dataset. How we treat our data, the models we choose, the approach we take to analyzing our data and in large part the entirety of our data science methodology and next steps are driven by the discoveries that we make during this stage of the process.

Follow Along

What can we discover about this dataset?

Below are some of the most popular Pandas EDA Functions. Lets try them on a dataset to see what we can learn.

* df.shape
* df.head()
* df.dtypes
* df.describe()
* Numeric
* Non-Numeric
* df[‘column’].value\_counts()
* df.isnull().sum()
* df.fillna()
* df.dropna()
* df.drop()
* pd.crosstab()

Challenge

Hopefully a lot of the above functions are review for you from the precourse material, but if not, again, don’t worry. We’ll be using these again on the assignment and most days of class -whenever we need to wrap our head around a new dataset.

Learn to Generate Basic Visualizations (graphs) with Pandas

One of the cornerstones of Exploratory Data Analysis (EDA) is visualizing our data in order to understand its distributions and how our variables are interrelated. Our brains are amazing pattern detection machines and sometimes the “eyeball test” is the most efficient one. In this section we’ll look at some of the most basic kinds of “exploratory visualizations” to help us better understand our data.

Overview

Lets demonstrate creating a:

* Line Plot
* Histogram
* Scatter Plot
* Density Plot
* Making plots of our crosstabs

Follow Along

How does each of these plots show us something different about the data?

Why might it be important for us to be able to visualize how our data is distributed?

Challenge

These are some of the most basic and important types of data visualizations. They’re so important that they’re built straight into Pandas and can be accessed with some very concise code. At the beginning our data exploration is about understanding the characteristics of our dataset, but over time it becomes about communicating insights in as effective and digestable a manner as possible, and that typically means using graphs in one way or another. See how intuitive of a graph you can make using a crosstab on this dataset.

Review

Class Recordings

You can use class recordings to help you master the material.

* [**Exploratory Data Analysis for DS15 w/ Ryan Allred**](https://youtu.be/MdZU5wacLvc)

04/06/2020

* [All previous recordings](https://learn.lambdaschool.com/archive/DS/module/rec81pljcvgboWB6U)

Demonstrate Mastery

To demonstrate mastery of this module, you need to complete and pass a code review on each of the following:

* Objective challenge:

You’ll have to do follow this process or one very close to it every day/week for the next nine months. If this feels a little bit overwhelming at first, don’t worry about it! We will be doing this everyday and you have your Team Leads and classmates to lean on for help. You’ll be a pro at using GitHub in no time.

If you’re already familiar with GitHub and or Git via the command line, feel free to use the tools that you are most comfortable with, but you still need to save your work to GitHub every day.

* Objective challenge:

You’ll get very good at reading documentation, Googling, asking for help, troubleshooting, debugging, etc. by the time you’re done here at Lambda School. Our goal is to turn you into a data scientist that can solve their own problems.

* Objective challenge:

On the assignment this afternoon you’ll get to choose a new dataset and try both of these methods, we will load hundreds of datasets into notebooks by the time the class is over, you’ll be pro at it in no time.

* Objective challenge:

Hopefully a lot of the above functions are review for you from the precourse material, but if not, again, don’t worry. We’ll be using these again on the assignment and most days of class -whenever we need to wrap our head around a new dataset.

* Objective challenge:

These are some of the most basic and important types of data visualizations. They’re so important that they’re built straight into Pandas and can be accessed with some very concise code. At the beginning our data exploration is about understanding the characteristics of our dataset, but over time it becomes about communicating insights in as effective and digestable a manner as possible, and that typically means using graphs in one way or another. See how intuitive of a graph you can make using a crosstab on this dataset.

**Data Wrangling and Storytelling: Make Features**

## Learn

#### Learn to understand the purpose of feature engineering

Feature Engineering is vital to improving the accuracy of machine learning models. It consists of finding creative ways to create new features (columns) of a dataset with the goal of exposing our model to patterns in the data that it hasn’t accounted for yet.

Feature Engineering is one of the areas where you can let your creativity, ingenuity, and domain knowledge shine as a data scientist. But before we can really dive in, we have to make sure that we’re comfortable creating new columns using all kinds of data.

##### Overview

Feature Engineering is the process of using a combination of domain knowledge, creativity and the pre-existing columns of a dataset to create completely new columns.

Machine Learning models try to detect patterns in the data and then associate those patterns with certain predictions. The hope is that by creating new columns on our dataset that we can expose our model to new patterns in the data so that it can make better and better predictions.

This is largely a matter of understanding how to work with individual columns of a dataframe with Pandas –which is what we’ll be practicing today!

##### Follow Along

Columns of a dataframe hold each hold a specific type of data. Lets inspect some of the common datatypes found in datasets and then we’ll make a new feature on a dataset using pre-existing columns.

Copy

import pandas as pd

Copy

### Lets take a look at the Ames Iowa Housing Dataset:

df = pd.read\_csv('https://raw.githubusercontent.com/ryanleeallred/datasets/master/Ames%20Housing%20Data/train.csv')

print(df.shape)

df.head()

Copy

(1460, 81)

|  | **Id** | **MSSubClass** | **MSZoning** | **LotFrontage** | **LotArea** | **Street** | **Alley** | **LotShape** | **LandContour** | **Utilities** | **LotConfig** | **LandSlope** | **Neighborhood** | **Condition1** | **Condition2** | **BldgType** | **HouseStyle** | **OverallQual** | **OverallCond** | **YearBuilt** | **YearRemodAdd** | **RoofStyle** | **RoofMatl** | **Exterior1st** | **Exterior2nd** | **MasVnrType** | **MasVnrArea** | **ExterQual** | **ExterCond** | **Foundation** | **BsmtQual** | **BsmtCond** | **BsmtExposure** | **BsmtFinType1** | **BsmtFinSF1** | **BsmtFinType2** | **BsmtFinSF2** | **BsmtUnfSF** | **TotalBsmtSF** | **Heating** | **...** | **CentralAir** | **Electrical** | **1stFlrSF** | **2ndFlrSF** | **LowQualFinSF** | **GrLivArea** | **BsmtFullBath** | **BsmtHalfBath** | **FullBath** | **HalfBath** | **BedroomAbvGr** | **KitchenAbvGr** | **KitchenQual** | **TotRmsAbvGrd** | **Functional** | **Fireplaces** | **FireplaceQu** | **GarageType** | **GarageYrBlt** | **GarageFinish** | **GarageCars**GarageAreaGarageQualGarageCondPavedDriveWoodDeckSFOpenPorchSFEnclosedPorch3SsnPorchScreenPorchPoolAreaPoolQCFenceMiscFeatureMiscValMoSoldYrSoldSaleTypeSaleConditionSalePrice |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 60 | RL | 65.0 | 8450 | Pave | NaN | Reg | Lvl | AllPub | Inside | Gtl | CollgCr | Norm | Norm | 1Fam | 2Story | 7 | 5 | 2003 | 2003 | Gable | CompShg | VinylSd | VinylSd | BrkFace | 196.0 | Gd | TA | PConc | Gd | TA | No | GLQ | 706 | Unf | 0 | 150 | 856 | GasA | ... | Y | SBrkr | 856 | 854 | 0 | 1710 | 1 | 0 | 2 | 1 | 3 | 1 | Gd | 8 | Typ | 0 | NaN | Attchd | 2003.0 | RFn | 2548TATAY0610000NaNNaNNaN022008WDNormal208500 |
| **1** | 2 | 20 | RL | 80.0 | 9600 | Pave | NaN | Reg | Lvl | AllPub | FR2 | Gtl | Veenker | Feedr | Norm | 1Fam | 1Story | 6 | 8 | 1976 | 1976 | Gable | CompShg | MetalSd | MetalSd | None | 0.0 | TA | TA | CBlock | Gd | TA | Gd | ALQ | 978 | Unf | 0 | 284 | 1262 | GasA | ... | Y | SBrkr | 1262 | 0 | 0 | 1262 | 0 | 1 | 2 | 0 | 3 | 1 | TA | 6 | Typ | 1 | TA | Attchd | 1976.0 | RFn | 2460TATAY29800000NaNNaNNaN052007WDNormal181500 |
| **2** | 3 | 60 | RL | 68.0 | 11250 | Pave | NaN | IR1 | Lvl | AllPub | Inside | Gtl | CollgCr | Norm | Norm | 1Fam | 2Story | 7 | 5 | 2001 | 2002 | Gable | CompShg | VinylSd | VinylSd | BrkFace | 162.0 | Gd | TA | PConc | Gd | TA | Mn | GLQ | 486 | Unf | 0 | 434 | 920 | GasA | ... | Y | SBrkr | 920 | 866 | 0 | 1786 | 1 | 0 | 2 | 1 | 3 | 1 | Gd | 6 | Typ | 1 | TA | Attchd | 2001.0 | RFn | 2608TATAY0420000NaNNaNNaN092008WDNormal223500 |
| **3** | 4 | 70 | RL | 60.0 | 9550 | Pave | NaN | IR1 | Lvl | AllPub | Corner | Gtl | Crawfor | Norm | Norm | 1Fam | 2Story | 7 | 5 | 1915 | 1970 | Gable | CompShg | Wd Sdng | Wd Shng | None | 0.0 | TA | TA | BrkTil | TA | Gd | No | ALQ | 216 | Unf | 0 | 540 | 756 | GasA | ... | Y | SBrkr | 961 | 756 | 0 | 1717 | 1 | 0 | 1 | 0 | 3 | 1 | Gd | 7 | Typ | 1 | Gd | Detchd | 1998.0 | Unf | 3642TATAY035272000NaNNaNNaN022006WDAbnorml140000 |
| **4** | 5 | 60 | RL | 84.0 | 14260 | Pave | NaN | IR1 | Lvl | AllPub | FR2 | Gtl | NoRidge | Norm | Norm | 1Fam | 2Story | 8 | 5 | 2000 | 2000 | Gable | CompShg | VinylSd | VinylSd | BrkFace | 350.0 | Gd | TA | PConc | Gd | TA | Av | GLQ | 655 | Unf | 0 | 490 | 1145 | GasA | ... | Y | SBrkr | 1145 | 1053 | 0 | 2198 | 1 | 0 | 2 | 1 | 4 | 1 | Gd | 9 | Typ | 1 | TA | Attchd | 2000.0 | RFn | 3836TATAY192840000NaNNaNNaN0122008WDNormal250000 |

5 rows × 81 columns

##### Specific Columns hold specific kinds of data

Copy

pd.set\_option('display.max\_rows', 400)

### We can look at column datatypes using `.dtypes()`

df.dtypes

Copy

Id int64

MSSubClass int64

MSZoning object

LotFrontage float64

LotArea int64

Street object

Alley object

LotShape object

LandContour object

Utilities object

LotConfig object

LandSlope object

Neighborhood object

Condition1 object

Condition2 object

BldgType object

HouseStyle object

OverallQual int64

OverallCond int64

YearBuilt int64

YearRemodAdd int64

RoofStyle object

RoofMatl object

Exterior1st object

Exterior2nd object

MasVnrType object

MasVnrArea float64

ExterQual object

ExterCond object

Foundation object

BsmtQual object

BsmtCond object

BsmtExposure object

BsmtFinType1 object

BsmtFinSF1 int64

BsmtFinType2 object

BsmtFinSF2 int64

BsmtUnfSF int64

TotalBsmtSF int64

Heating object

HeatingQC object

CentralAir object

Electrical object

1stFlrSF int64

2ndFlrSF int64

LowQualFinSF int64

GrLivArea int64

BsmtFullBath int64

BsmtHalfBath int64

FullBath int64

HalfBath int64

BedroomAbvGr int64

KitchenAbvGr int64

KitchenQual object

TotRmsAbvGrd int64

Functional object

Fireplaces int64

FireplaceQu object

GarageType object

GarageYrBlt float64

GarageFinish object

GarageCars int64

GarageArea int64

GarageQual object

GarageCond object

PavedDrive object

WoodDeckSF int64

OpenPorchSF int64

EnclosedPorch int64

3SsnPorch int64

ScreenPorch int64

PoolArea int64

PoolQC object

Fence object

MiscFeature object

MiscVal int64

MoSold int64

YrSold int64

SaleType object

SaleCondition object

SalePrice int64

dtype: object

Some columns hold integer values like the BedroomAbvGr which stands for “Bedrooms Above Grade.” This is the number of non-basement bedrooms in the home.

For more information on specific column meanings view the [data dictionary](https://github.com/ryanleeallred/datasets/blob/master/Ames%20Housing%20Data/data_description.txt).

Copy

### Look at the first ten rows of the `BedroomAbvGr` column.

### Looks like integers to me!

df['BedroomAbvGr'].head(10)

Copy

0 3

1 3

2 3

3 3

4 4

5 1

6 3

7 3

8 2

9 2

Name: BedroomAbvGr, dtype: int64

Some columns hold float values like the LotFrontage column.

Copy

### Look at the first ten rows of the `BedroomAbvGr` column.

df['LotFrontage'].head(10)

Copy

0 65.0

1 80.0

2 68.0

3 60.0

4 84.0

5 85.0

6 75.0

7 NaN

8 51.0

9 50.0

Name: LotFrontage, dtype: float64

Hmmm, do the values above look like floats to you?

They all have .0 on them so technically they’re being stored as floats, but should they be stored as floats?

Lets see what all of the possible values for this column are.

Copy

df['LotFrontage'].value\_counts(dropna=False)

Copy

NaN 259

60.0 143

70.0 70

80.0 69

50.0 57

75.0 53

65.0 44

85.0 40

78.0 25

21.0 23

90.0 23

68.0 19

24.0 19

64.0 19

73.0 18

72.0 17

79.0 17

63.0 17

55.0 17

100.0 16

51.0 15

66.0 15

74.0 15

52.0 14

59.0 13

71.0 12

67.0 12

57.0 12

82.0 12

43.0 12

40.0 12

76.0 11

69.0 11

53.0 10

92.0 10

88.0 10

34.0 10

86.0 10

77.0 9

35.0 9

44.0 9

84.0 9

62.0 9

93.0 8

61.0 8

98.0 8

96.0 8

107.0 7

95.0 7

120.0 7

58.0 7

41.0 6

105.0 6

54.0 6

94.0 6

48.0 6

110.0 6

30.0 6

89.0 6

81.0 6

91.0 6

36.0 6

47.0 5

83.0 5

37.0 5

87.0 5

32.0 5

56.0 5

102.0 4

42.0 4

49.0 4

99.0 3

45.0 3

104.0 3

130.0 3

103.0 3

108.0 3

124.0 2

174.0 2

97.0 2

134.0 2

129.0 2

313.0 2

118.0 2

122.0 2

121.0 2

101.0 2

109.0 2

115.0 2

116.0 2

114.0 2

153.0 1

149.0 1

150.0 1

111.0 1

182.0 1

46.0 1

112.0 1

141.0 1

33.0 1

152.0 1

160.0 1

168.0 1

128.0 1

144.0 1

39.0 1

106.0 1

38.0 1

138.0 1

140.0 1

137.0 1

Name: LotFrontage, dtype: int64

Looks to me like the LotFrontage column originally held integer values but was cast to a float meaning that each original integer values was converted to its corresponding float representation.

Any guesses as to why that would have happened?

HINT: What’s the most common LotFrontage value for this column?

Copy

### NaN is the most common value in this column. What is a NaN

df["LotFrontage"].value\_counts(dropna=False).head()

Copy

NaN 259

60.0 143

70.0 70

80.0 69

50.0 57

Name: LotFrontage, dtype: int64

NaN stands stands for “Not a Number” and is the default missing value indicator with Pandas. This means there were cells in this column that didn’t have a LotFrontage value recorded for those homes.

This is where domain knowledge starts to come in. Think about the context we’re working with here: houses. What might a null or blank cell representing “Linear feet of street connected to property” mean in the context of a housing dataset?

Ok, so maybe it makes seanse to have some NaNs in this column. What is the datatype of a NaN value?

Perhaps some of this data is truly missing or unrecorded data, but sometimes NaNs are more likely to indicate something that was “NA” or “Not Applicable” to a particular observation. There could be multiple reasons why there was no value recorded for a particular feature.

Remember, that Pandas tries to maintain a single datatype for all values in a column, and therefore…

Copy

import numpy as np

### What is the datatype of NaN?

type(np.NaN)

Copy

float

The datatype of a NaN is float! This means that if we have a column of integer values, but the column has even a single NaN that column will not be treated with the integer datatype but all of the integers will be converted to floats in order to try and preserve the same datatype throughout the entire column.

You can see already how understanding column datatypes is crucial to understanding how Pandas help us manage our data.

##### Making new Features

Lets slim down the dataset and consider just a few specific columns:

* TotalBsmtSF
* 1stFlrSF
* 2ndFlrSF
* SalePrice1

Copy

### I can make a smaller dataframe with a few specific column headers

### by passing a list of column headers inside of the square brackets

small\_df = df[['TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'SalePrice']].copy()

small\_df.head()

|  | **TotalBsmtSF** | **1stFlrSF** | **2ndFlrSF** | **SalePrice** |
| --- | --- | --- | --- | --- |
| **0** | 856 | 856 | 854 | 208500 |
| **1** | 1262 | 1262 | 0 | 181500 |
| **2** | 920 | 920 | 866 | 223500 |
| **3** | 756 | 961 | 756 | 140000 |
| **4** | 1145 | 1145 | 1053 | 250000 |

##### Syntax for creating new columns

When making a new column on a dataframe, we have to use the square bracket syntax of accessing a column. We can’t use “dot syntax” here.

Copy

### Lets add up all of the square footage to get a single square footage

### column for the entire dataset

### Using bracket syntax to make a new 'TotalSquareFootage' column

small\_df['TotalSquareFootage'] = small\_df['TotalBsmtSF'] + small\_df['1stFlrSF'] + small\_df['2ndFlrSF']

small\_df.head()

|  | **TotalBsmtSF** | **1stFlrSF** | **2ndFlrSF** | **SalePrice** | **TotalSquareFootage** |
| --- | --- | --- | --- | --- | --- |
| **0** | 856 | 856 | 854 | 208500 | 2566 |
| **1** | 1262 | 1262 | 0 | 181500 | 2524 |
| **2** | 920 | 920 | 866 | 223500 | 2706 |
| **3** | 756 | 961 | 756 | 140000 | 2473 |
| **4** | 1145 | 1145 | 1053 | 250000 | 3343 |

Copy

### Lets make a nother new column that is 'PricePerSqFt' by

### dividing the price by the square footage

small\_df['PricePerSqFt'] = (small\_df['SalePrice'] / small\_df['TotalSquareFootage'])

small\_df.head()

|  | **TotalBsmtSF** | **1stFlrSF** | **2ndFlrSF** | **SalePrice** | **TotalSquareFootage** | **PricePerSqFt** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 856 | 856 | 854 | 208500 | 2566 | 81.254871 |
| **1** | 1262 | 1262 | 0 | 181500 | 2524 | 71.909667 |
| **2** | 920 | 920 | 866 | 223500 | 2706 | 82.594235 |
| **3** | 756 | 961 | 756 | 140000 | 2473 | 56.611403 |
| **4** | 1145 | 1145 | 1053 | 250000 | 3343 | 74.783129 |

Ok, we have made two new columns on our small dataset.

* What does a **high** PricePerSqFt say about a home that the square footage and price alone don’t capture as directly?
* What does a **low** PricePerSqFt say about a home that the square footage and price alone don’t directly capture?

##### Challenge

I hope you can see how we have used existing columns to create a new column on a dataset that say something new about our unit of observation. This is what making new features (columns) on a dataset is all about and why it’s so essential to data science –particularly predictive modeling “Machine Learning.”

We’ll spend the rest of the lecture and assignment today trying to get as good as we can at manipulating (cleaning) and creating new columns on datasets.

#### Learn to work with strings in pandas

Strings are the second most common datatype that we will work with as data scientists and as such we need to be comfortable manipulating them in common ways. Working with strings will also provide us the opportunity to practice using .apply() functions, lambda functions, and list comprehensions.

##### Overview

So far we have worked with numeric datatypes (ints and floats) but we haven’t worked with any columns containing string values. We can’t simply use arithmetic to manipulate string values, so we’ll need to learn some more techniques in order to work with this datatype.

Read Python Data Science Handbook [Chapter 3.10](https://jakevdp.github.io/PythonDataScienceHandbook/03.10-working-with-strings.html), Vectorized String Operations

##### Follow Along

We’re going to import a new dataset here to work with. This dataset is from LendingClub and holds information about loans issued in Q4 of 2018. This dataset is a bit messy so it will give us plenty of opportunities to clean up existing columns as well as create new ones.

The !wget shell command being used here does exactly the same thing that your browser does when you type a URL in the address. It makes a request or “gets” the file at that address. However, in our case the file isn’t a webpage, it’s a compressed CSV file.

Try copying and pasting the URL from below into your browser, did it start an automatic download? Any URLs like this that start automatic downloads when navigated to can be used along with the !wget command to bring files directly into your notebook’s memory.

##### Load a new dataset

Copy

!wget https://resources.lendingclub.com/LoanStats\_2018Q4.csv.zip

Copy

--2019-11-20 07:09:52-- https://resources.lendingclub.com/LoanStats\_2018Q4.csv.zip

Resolving resources.lendingclub.com (resources.lendingclub.com)... 64.48.1.20

Connecting to resources.lendingclub.com (resources.lendingclub.com)|64.48.1.20|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: unspecified [application/zip]

Saving to: ‘LoanStats\_2018Q4.csv.zip.1’

LoanStats\_2018Q4.cs [ <=> ] 21.67M 1.83MB/s in 12s

2019-11-20 07:10:04 (1.81 MB/s) - ‘LoanStats\_2018Q4.csv.zip.1’ saved [22727580]

We need to use the !unzip command to extract the csv from the zipped folder.

Copy

!unzip LoanStats\_2018Q4.csv.zip

Copy

Archive: LoanStats\_2018Q4.csv.zip

replace LoanStats\_2018Q4.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y

inflating: LoanStats\_2018Q4.csv

We can also use bash/shell commands to look at the raw file using the !head and !tail commands

Copy

!head LoanStats\_2018Q4.csv

Copy

Notes offered by Prospectus (https://www.lendingclub.com/info/prospectus.action)

"id","member\_id","loan\_amnt","funded\_amnt","funded\_amnt\_inv","term","int\_rate","installment","grade","sub\_grade","emp\_title","emp\_length","home\_ownership","annual\_inc","verification\_status","issue\_d","loan\_status","pymnt\_plan","url","desc","purpose","title","zip\_code","addr\_state","dti","delinq\_2yrs","earliest\_cr\_line","inq\_last\_6mths","mths\_since\_last\_delinq","mths\_since\_last\_record","open\_acc","pub\_rec","revol\_bal","revol\_util","total\_acc","initial\_list\_status","out\_prncp","out\_prncp\_inv","total\_pymnt","total\_pymnt\_inv","total\_rec\_prncp","total\_rec\_int","total\_rec\_late\_fee","recoveries","collection\_recovery\_fee","last\_pymnt\_d","last\_pymnt\_amnt","next\_pymnt\_d","last\_credit\_pull\_d","collections\_12\_mths\_ex\_med","mths\_since\_last\_major\_derog","policy\_code","application\_type","annual\_inc\_joint","dti\_joint","verification\_status\_joint","acc\_now\_delinq","tot\_coll\_amt","tot\_cur\_bal","open\_acc\_6m","open\_act\_il","open\_il\_12m","open\_il\_24m","mths\_since\_rcnt\_il","total\_bal\_il","il\_util","open\_rv\_12m","open\_rv\_24m","max\_bal\_bc","all\_util","total\_rev\_hi\_lim","inq\_fi","total\_cu\_tl","inq\_last\_12m","acc\_open\_past\_24mths","avg\_cur\_bal","bc\_open\_to\_buy","bc\_util","chargeoff\_within\_12\_mths","delinq\_amnt","mo\_sin\_old\_il\_acct","mo\_sin\_old\_rev\_tl\_op","mo\_sin\_rcnt\_rev\_tl\_op","mo\_sin\_rcnt\_tl","mort\_acc","mths\_since\_recent\_bc","mths\_since\_recent\_bc\_dlq","mths\_since\_recent\_inq","mths\_since\_recent\_revol\_delinq","num\_accts\_ever\_120\_pd","num\_actv\_bc\_tl","num\_actv\_rev\_tl","num\_bc\_sats","num\_bc\_tl","num\_il\_tl","num\_op\_rev\_tl","num\_rev\_accts","num\_rev\_tl\_bal\_gt\_0","num\_sats","num\_tl\_120dpd\_2m","num\_tl\_30dpd","num\_tl\_90g\_dpd\_24m","num\_tl\_op\_past\_12m","pct\_tl\_nvr\_dlq","percent\_bc\_gt\_75","pub\_rec\_bankruptcies","tax\_liens","tot\_hi\_cred\_lim","total\_bal\_ex\_mort","total\_bc\_limit","total\_il\_high\_credit\_limit","revol\_bal\_joint","sec\_app\_earliest\_cr\_line","sec\_app\_inq\_last\_6mths","sec\_app\_mort\_acc","sec\_app\_open\_acc","sec\_app\_revol\_util","sec\_app\_open\_act\_il","sec\_app\_num\_rev\_accts","sec\_app\_chargeoff\_within\_12\_mths","sec\_app\_collections\_12\_mths\_ex\_med","sec\_app\_mths\_since\_last\_major\_derog","hardship\_flag","hardship\_type","hardship\_reason","hardship\_status","deferral\_term","hardship\_amount","hardship\_start\_date","hardship\_end\_date","payment\_plan\_start\_date","hardship\_length","hardship\_dpd","hardship\_loan\_status","orig\_projected\_additional\_accrued\_interest","hardship\_payoff\_balance\_amount","hardship\_last\_payment\_amount","debt\_settlement\_flag","debt\_settlement\_flag\_date","settlement\_status","settlement\_date","settlement\_amount","settlement\_percentage","settlement\_term"

"","","35000","35000","35000"," 36 months"," 14.47%","1204.23","C","C2","Staff Physician","8 years","MORTGAGE","360000","Verified","Dec-2018","Fully Paid","n","","","credit\_card","Credit card refinancing","336xx","FL","19.9","0","Apr-1995","1","","","24","0","57259","43.2%","51","w","0.00","0.00","38187.0468373662","38187.05","35000.00","3187.05","0.0","0.0","0.0","Aug-2019","29882.16","","Sep-2019","0","","1","Individual","","","","0","0","828060","0","6","1","2","9","112027","29","4","8","19118","36","132400","3","2","5","11","34503","67341","46","0","0","173","284","8","8","3","8","","0","","0","11","11","13","20","23","16","24","11","24","0","0","0","5","100","30.8","0","0","1222051","169286","124600","258401","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","5000","5000","5000"," 36 months"," 22.35%","191.86","D","D5","Director of Sales","10+ years","MORTGAGE","72000","Source Verified","Dec-2018","Fully Paid","n","","","debt\_consolidation","Debt consolidation","333xx","FL","20.12","0","Mar-2010","0","","","13","0","11720","47.1%","26","f","0.00","0.00","5615.9776735688","5615.98","5000.00","615.98","0.0","0.0","0.0","Jul-2019","4474.13","","Aug-2019","0","","1","Individual","","","","0","534","189279","0","1","0","1","18","22698","","0","0","4056","47","24900","1","0","1","2","14560","8163","55.1","0","0","105","90","29","8","5","40","","9","","0","6","11","6","8","4","11","17","11","13","0","0","0","1","100","50","0","0","218686","34418","18200","37786","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","10000","10000","10000"," 60 months"," 23.40%","284.21","E","E1","","< 1 year","RENT","55000","Source Verified","Dec-2018","Current","n","","","debt\_consolidation","Debt consolidation","902xx","CA","13.51","0","Apr-2007","0","44","88","9","1","11859","53.9%","11","w","9131.55","9131.55","2538.39","2538.39","868.45","1669.94","0.0","0.0","0.0","Sep-2019","284.21","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","21235","0","1","0","1","20","9376","76","0","3","3122","62","22000","1","0","0","4","2359","1119","89.3","0","0","140","140","13","13","0","13","","15","44","0","4","7","4","5","2","8","9","7","9","0","0","0","0","90.9","100","1","0","34386","21235","10500","12386","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","17100","17100","17100"," 36 months"," 18.94%","626.3","D","D2","Receptionist ","10+ years","RENT","38000","Verified","Dec-2018","Current","n","","","debt\_consolidation","Debt consolidation","150xx","PA","38.09","0","Mar-1998","1","47","","14","0","15323","53%","21","w","13682.21","13682.21","5609.71","5609.71","3417.79","2191.92","0.0","0.0","0.0","Sep-2019","626.3","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","43351","1","2","1","1","10","28028","67","1","5","7533","53","29170","0","0","2","6","3096","4150","77","0","0","125","230","5","5","2","5","","5","","0","4","12","5","5","5","12","14","9","14","","0","0","2","95","75","0","0","70954","43351","16600","41784","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","4000","4000","4000"," 36 months"," 10.72%","130.43","B","B2","Extrusion assistant ","10+ years","MORTGAGE","56000","Verified","Dec-2018","Current","n","","","credit\_card","Credit card refinancing","301xx","GA","31.03","0","Sep-2006","0","","","7","0","4518","28.6%","11","w","3116.62","3116.62","1160.78","1160.78","883.38","277.40","0.0","0.0","0.0","Sep-2019","130.43","Oct-2019","Oct-2019","1","","1","Individual","","","","0","136","192983","0","2","0","2","14","66857","","0","3","1608","29","15800","3","0","0","5","27569","7835","36.3","0","0","147","29","22","14","1","22","","17","","0","3","4","3","3","6","4","4","4","7","0","0","0","0","100","0","0","0","221310","71375","12300","77865","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","10475","10475","10475"," 36 months"," 11.31%","344.48","B","B3","Teacher","6 years","RENT","66150","Source Verified","Dec-2018","Current","n","","","debt\_consolidation","Debt consolidation","786xx","TX","7.4","0","Jun-1991","0","28","","10","0","10499","40.2%","17","w","8177.98","8177.98","3090.45","3090.45","2297.02","793.43","0.0","0.0","0.0","Sep-2019","344.48","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","10499","1","0","0","0","84","0","","3","5","3841","40","26100","2","1","5","5","1312","14847","36.8","0","0","88","330","3","3","2","7","","1","28","0","5","8","7","8","2","10","13","8","10","0","0","0","3","88.2","60","0","0","26100","10499","23500","0","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","7500","7500","7500"," 36 months"," 11.31%","246.65","B","B3","Supervisor","7 years","MORTGAGE","40500","Verified","Dec-2018","In Grace Period","n","","","debt\_consolidation","Debt consolidation","604xx","IL","31.61","0","Sep-1996","1","","","23","0","14218","16.3%","33","w","6032.58","6032.58","1983.44","1983.44","1467.42","516.02","0.0","0.0","0.0","Sep-2019","246.65","Nov-2019","Oct-2019","0","","1","Individual","","","","0","0","123779","3","2","1","1","12","8096","29","3","6","2972","19","87000","0","0","2","7","5626","21078","23.9","0","0","59","267","1","1","1","6","","6","","0","4","6","7","10","2","20","30","6","23","0","0","0","4","100","0","0","0","221569","22314","27700","28409","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","3600","3600","3600"," 36 months"," 11.80%","119.23","B","B4","Analyst","5 years","MORTGAGE","50000","Source Verified","Dec-2018","Current","n","","","other","Other","427xx","KY","16.11","0","Jan-2002","2","","","10","0","5865","17.4%","17","w","2908.20","2908.20","968.0","968.00","691.80","276.20","0.0","0.0","0.0","Sep-2019","119.23","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","243415","3","3","1","2","3","68184","95","1","3","0","53","33700","2","1","10","6","24342","17300","0","0","0","203","198","4","3","1","18","","3","","1","0","1","4","6","6","6","10","1","10","0","0","0","3","93.8","0","0","0","272629","74049","17300","68929","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

Copy

!head LoanStats\_2018Q4.csv

Copy

Notes offered by Prospectus (https://www.lendingclub.com/info/prospectus.action)

"id","member\_id","loan\_amnt","funded\_amnt","funded\_amnt\_inv","term","int\_rate","installment","grade","sub\_grade","emp\_title","emp\_length","home\_ownership","annual\_inc","verification\_status","issue\_d","loan\_status","pymnt\_plan","url","desc","purpose","title","zip\_code","addr\_state","dti","delinq\_2yrs","earliest\_cr\_line","inq\_last\_6mths","mths\_since\_last\_delinq","mths\_since\_last\_record","open\_acc","pub\_rec","revol\_bal","revol\_util","total\_acc","initial\_list\_status","out\_prncp","out\_prncp\_inv","total\_pymnt","total\_pymnt\_inv","total\_rec\_prncp","total\_rec\_int","total\_rec\_late\_fee","recoveries","collection\_recovery\_fee","last\_pymnt\_d","last\_pymnt\_amnt","next\_pymnt\_d","last\_credit\_pull\_d","collections\_12\_mths\_ex\_med","mths\_since\_last\_major\_derog","policy\_code","application\_type","annual\_inc\_joint","dti\_joint","verification\_status\_joint","acc\_now\_delinq","tot\_coll\_amt","tot\_cur\_bal","open\_acc\_6m","open\_act\_il","open\_il\_12m","open\_il\_24m","mths\_since\_rcnt\_il","total\_bal\_il","il\_util","open\_rv\_12m","open\_rv\_24m","max\_bal\_bc","all\_util","total\_rev\_hi\_lim","inq\_fi","total\_cu\_tl","inq\_last\_12m","acc\_open\_past\_24mths","avg\_cur\_bal","bc\_open\_to\_buy","bc\_util","chargeoff\_within\_12\_mths","delinq\_amnt","mo\_sin\_old\_il\_acct","mo\_sin\_old\_rev\_tl\_op","mo\_sin\_rcnt\_rev\_tl\_op","mo\_sin\_rcnt\_tl","mort\_acc","mths\_since\_recent\_bc","mths\_since\_recent\_bc\_dlq","mths\_since\_recent\_inq","mths\_since\_recent\_revol\_delinq","num\_accts\_ever\_120\_pd","num\_actv\_bc\_tl","num\_actv\_rev\_tl","num\_bc\_sats","num\_bc\_tl","num\_il\_tl","num\_op\_rev\_tl","num\_rev\_accts","num\_rev\_tl\_bal\_gt\_0","num\_sats","num\_tl\_120dpd\_2m","num\_tl\_30dpd","num\_tl\_90g\_dpd\_24m","num\_tl\_op\_past\_12m","pct\_tl\_nvr\_dlq","percent\_bc\_gt\_75","pub\_rec\_bankruptcies","tax\_liens","tot\_hi\_cred\_lim","total\_bal\_ex\_mort","total\_bc\_limit","total\_il\_high\_credit\_limit","revol\_bal\_joint","sec\_app\_earliest\_cr\_line","sec\_app\_inq\_last\_6mths","sec\_app\_mort\_acc","sec\_app\_open\_acc","sec\_app\_revol\_util","sec\_app\_open\_act\_il","sec\_app\_num\_rev\_accts","sec\_app\_chargeoff\_within\_12\_mths","sec\_app\_collections\_12\_mths\_ex\_med","sec\_app\_mths\_since\_last\_major\_derog","hardship\_flag","hardship\_type","hardship\_reason","hardship\_status","deferral\_term","hardship\_amount","hardship\_start\_date","hardship\_end\_date","payment\_plan\_start\_date","hardship\_length","hardship\_dpd","hardship\_loan\_status","orig\_projected\_additional\_accrued\_interest","hardship\_payoff\_balance\_amount","hardship\_last\_payment\_amount","debt\_settlement\_flag","debt\_settlement\_flag\_date","settlement\_status","settlement\_date","settlement\_amount","settlement\_percentage","settlement\_term"

"","","35000","35000","35000"," 36 months"," 14.47%","1204.23","C","C2","Staff Physician","8 years","MORTGAGE","360000","Verified","Dec-2018","Fully Paid","n","","","credit\_card","Credit card refinancing","336xx","FL","19.9","0","Apr-1995","1","","","24","0","57259","43.2%","51","w","0.00","0.00","38187.0468373662","38187.05","35000.00","3187.05","0.0","0.0","0.0","Aug-2019","29882.16","","Sep-2019","0","","1","Individual","","","","0","0","828060","0","6","1","2","9","112027","29","4","8","19118","36","132400","3","2","5","11","34503","67341","46","0","0","173","284","8","8","3","8","","0","","0","11","11","13","20","23","16","24","11","24","0","0","0","5","100","30.8","0","0","1222051","169286","124600","258401","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","5000","5000","5000"," 36 months"," 22.35%","191.86","D","D5","Director of Sales","10+ years","MORTGAGE","72000","Source Verified","Dec-2018","Fully Paid","n","","","debt\_consolidation","Debt consolidation","333xx","FL","20.12","0","Mar-2010","0","","","13","0","11720","47.1%","26","f","0.00","0.00","5615.9776735688","5615.98","5000.00","615.98","0.0","0.0","0.0","Jul-2019","4474.13","","Aug-2019","0","","1","Individual","","","","0","534","189279","0","1","0","1","18","22698","","0","0","4056","47","24900","1","0","1","2","14560","8163","55.1","0","0","105","90","29","8","5","40","","9","","0","6","11","6","8","4","11","17","11","13","0","0","0","1","100","50","0","0","218686","34418","18200","37786","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","10000","10000","10000"," 60 months"," 23.40%","284.21","E","E1","","< 1 year","RENT","55000","Source Verified","Dec-2018","Current","n","","","debt\_consolidation","Debt consolidation","902xx","CA","13.51","0","Apr-2007","0","44","88","9","1","11859","53.9%","11","w","9131.55","9131.55","2538.39","2538.39","868.45","1669.94","0.0","0.0","0.0","Sep-2019","284.21","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","21235","0","1","0","1","20","9376","76","0","3","3122","62","22000","1","0","0","4","2359","1119","89.3","0","0","140","140","13","13","0","13","","15","44","0","4","7","4","5","2","8","9","7","9","0","0","0","0","90.9","100","1","0","34386","21235","10500","12386","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","17100","17100","17100"," 36 months"," 18.94%","626.3","D","D2","Receptionist ","10+ years","RENT","38000","Verified","Dec-2018","Current","n","","","debt\_consolidation","Debt consolidation","150xx","PA","38.09","0","Mar-1998","1","47","","14","0","15323","53%","21","w","13682.21","13682.21","5609.71","5609.71","3417.79","2191.92","0.0","0.0","0.0","Sep-2019","626.3","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","43351","1","2","1","1","10","28028","67","1","5","7533","53","29170","0","0","2","6","3096","4150","77","0","0","125","230","5","5","2","5","","5","","0","4","12","5","5","5","12","14","9","14","","0","0","2","95","75","0","0","70954","43351","16600","41784","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","4000","4000","4000"," 36 months"," 10.72%","130.43","B","B2","Extrusion assistant ","10+ years","MORTGAGE","56000","Verified","Dec-2018","Current","n","","","credit\_card","Credit card refinancing","301xx","GA","31.03","0","Sep-2006","0","","","7","0","4518","28.6%","11","w","3116.62","3116.62","1160.78","1160.78","883.38","277.40","0.0","0.0","0.0","Sep-2019","130.43","Oct-2019","Oct-2019","1","","1","Individual","","","","0","136","192983","0","2","0","2","14","66857","","0","3","1608","29","15800","3","0","0","5","27569","7835","36.3","0","0","147","29","22","14","1","22","","17","","0","3","4","3","3","6","4","4","4","7","0","0","0","0","100","0","0","0","221310","71375","12300","77865","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","10475","10475","10475"," 36 months"," 11.31%","344.48","B","B3","Teacher","6 years","RENT","66150","Source Verified","Dec-2018","Current","n","","","debt\_consolidation","Debt consolidation","786xx","TX","7.4","0","Jun-1991","0","28","","10","0","10499","40.2%","17","w","8177.98","8177.98","3090.45","3090.45","2297.02","793.43","0.0","0.0","0.0","Sep-2019","344.48","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","10499","1","0","0","0","84","0","","3","5","3841","40","26100","2","1","5","5","1312","14847","36.8","0","0","88","330","3","3","2","7","","1","28","0","5","8","7","8","2","10","13","8","10","0","0","0","3","88.2","60","0","0","26100","10499","23500","0","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","7500","7500","7500"," 36 months"," 11.31%","246.65","B","B3","Supervisor","7 years","MORTGAGE","40500","Verified","Dec-2018","In Grace Period","n","","","debt\_consolidation","Debt consolidation","604xx","IL","31.61","0","Sep-1996","1","","","23","0","14218","16.3%","33","w","6032.58","6032.58","1983.44","1983.44","1467.42","516.02","0.0","0.0","0.0","Sep-2019","246.65","Nov-2019","Oct-2019","0","","1","Individual","","","","0","0","123779","3","2","1","1","12","8096","29","3","6","2972","19","87000","0","0","2","7","5626","21078","23.9","0","0","59","267","1","1","1","6","","6","","0","4","6","7","10","2","20","30","6","23","0","0","0","4","100","0","0","0","221569","22314","27700","28409","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

"","","3600","3600","3600"," 36 months"," 11.80%","119.23","B","B4","Analyst","5 years","MORTGAGE","50000","Source Verified","Dec-2018","Current","n","","","other","Other","427xx","KY","16.11","0","Jan-2002","2","","","10","0","5865","17.4%","17","w","2908.20","2908.20","968.0","968.00","691.80","276.20","0.0","0.0","0.0","Sep-2019","119.23","Oct-2019","Oct-2019","0","","1","Individual","","","","0","0","243415","3","3","1","2","3","68184","95","1","3","0","53","33700","2","1","10","6","24342","17300","0","0","0","203","198","4","3","1","18","","3","","1","0","1","4","6","6","6","10","1","10","0","0","0","3","93.8","0","0","0","272629","74049","17300","68929","","","","","","","","","","","","N","","","","","","","","","","","","","","","N","","","","","",""

As we look at the raw file itself, do you see anything that might cause us trouble as we read in the CSV file to a dataframe?

Copy

df = pd.read\_csv('LoanStats\_2018Q4.csv')

print(df.shape)

df.head()

Copy

/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (0,1,2,3,4,7,13,18,19,24,25,27,28,29,30,31,32,34,36,37,38,39,40,41,42,43,44,46,49,50,51,53,54,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,113,114,115,116,117,118,119,120,121,123,124,125,126,127,128,129,130,131,132,133,134,135,136,141,142,143) have mixed types. Specify dtype option on import or set low\_memory=False.

interactivity=interactivity, compiler=compiler, result=result)

(128415, 1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Notes offered by Prospectus (https://www.lendingclub.com/info/prospectus.action) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **id** | **member\_id** | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **url** | **desc** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **total\_rec\_prncp** | **total\_rec\_int** | **total\_rec\_late\_fee** | **recoveries** | **collection\_recovery\_fee** | **last\_pymnt\_d** | **last\_pymnt\_amnt** | **next\_pymnt\_d** | **last\_credit\_pull\_d** | **collections\_12\_mths\_ex\_med** | **mths\_since\_last\_major\_derog** | **policy\_code** | **application\_type** | **annual\_inc\_joint** | **dti\_joint** | **verification\_status\_joint** | **acc\_now\_delinq** | **tot\_coll\_amt** | **tot\_cur\_bal** | **open\_acc\_6m** | **open\_act\_il** | **open\_il\_12m** | **open\_il\_24m**mths\_since\_rcnt\_iltotal\_bal\_ilil\_utilopen\_rv\_12mopen\_rv\_24mmax\_bal\_bcall\_utiltotal\_rev\_hi\_liminq\_fitotal\_cu\_tlinq\_last\_12macc\_open\_past\_24mthsavg\_cur\_balbc\_open\_to\_buybc\_utilchargeoff\_within\_12\_mthsdelinq\_amntmo\_sin\_old\_il\_acctmo\_sin\_old\_rev\_tl\_opmo\_sin\_rcnt\_rev\_tl\_opmo\_sin\_rcnt\_tlmort\_accmths\_since\_recent\_bcmths\_since\_recent\_bc\_dlqmths\_since\_recent\_inqmths\_since\_recent\_revol\_delinqnum\_accts\_ever\_120\_pdnum\_actv\_bc\_tlnum\_actv\_rev\_tlnum\_bc\_satsnum\_bc\_tlnum\_il\_tlnum\_op\_rev\_tlnum\_rev\_acctsnum\_rev\_tl\_bal\_gt\_0num\_satsnum\_tl\_120dpd\_2mnum\_tl\_30dpdnum\_tl\_90g\_dpd\_24mnum\_tl\_op\_past\_12mpct\_tl\_nvr\_dlqpercent\_bc\_gt\_75pub\_rec\_bankruptciestax\_lienstot\_hi\_cred\_limtotal\_bal\_ex\_morttotal\_bc\_limittotal\_il\_high\_credit\_limitrevol\_bal\_jointsec\_app\_earliest\_cr\_linesec\_app\_inq\_last\_6mthssec\_app\_mort\_accsec\_app\_open\_accsec\_app\_revol\_utilsec\_app\_open\_act\_ilsec\_app\_num\_rev\_acctssec\_app\_chargeoff\_within\_12\_mthssec\_app\_collections\_12\_mths\_ex\_medsec\_app\_mths\_since\_last\_major\_deroghardship\_flaghardship\_typehardship\_reasonhardship\_statusdeferral\_termhardship\_amounthardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_term |
| **NaN** | **NaN** | **35000** | **35000** | **35000** | **36 months** | **14.47%** | **1204.23** | **C** | **C2** | **Staff Physician** | **8 years** | **MORTGAGE** | **360000** | **Verified** | **Dec-2018** | **Fully Paid** | **n** | **NaN** | **NaN** | **credit\_card** | **Credit card refinancing** | **336xx** | **FL** | **19.9** | **0** | **Apr-1995** | **1** | **NaN** | **NaN** | **24** | **0** | **57259** | **43.2%** | **51** | **w** | **0.00** | **0.00** | **38187.0468373662** | **38187.05** | **35000.00** | **3187.05** | **0.0** | **0.0** | **0.0** | **Aug-2019** | **29882.16** | **NaN** | **Sep-2019** | **0** | **NaN** | **1** | **Individual** | **NaN** | **NaN** | **NaN** | **0** | **0** | **828060** | **0** | **6** | **1** | **2**91120272948191183613240032511345036734146001732848838NaN0NaN0111113202316241124000510030.8001222051169286124600258401NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **5000** | **5000** | **5000** | **36 months** | **22.35%** | **191.86** | **D** | **D5** | **Director of Sales** | **10+ years** | **MORTGAGE** | **72000** | **Source Verified** | **Dec-2018** | **Fully Paid** | **n** | **NaN** | **NaN** | **debt\_consolidation** | **Debt consolidation** | **333xx** | **FL** | **20.12** | **0** | **Mar-2010** | **0** | **NaN** | **NaN** | **13** | **0** | **11720** | **47.1%** | **26** | **f** | **0.00** | **0.00** | **5615.9776735688** | **5615.98** | **5000.00** | **615.98** | **0.0** | **0.0** | **0.0** | **Jul-2019** | **4474.13** | **NaN** | **Aug-2019** | **0** | **NaN** | **1** | **Individual** | **NaN** | **NaN** | **NaN** | **0** | **534** | **189279** | **0** | **1** | **0** | **1**1822698NaN0040564724900101214560816355.10010590298540NaN9NaN06116841117111300011005000218686344181820037786NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **10000** | **10000** | **10000** | **60 months** | **23.40%** | **284.21** | **E** | **E1** | **NaN** | **< 1 year** | **RENT** | **55000** | **Source Verified** | **Dec-2018** | **Current** | **n** | **NaN** | **NaN** | **debt\_consolidation** | **Debt consolidation** | **902xx** | **CA** | **13.51** | **0** | **Apr-2007** | **0** | **44** | **88** | **9** | **1** | **11859** | **53.9%** | **11** | **w** | **9131.55** | **9131.55** | **2538.39** | **2538.39** | **868.45** | **1669.94** | **0.0** | **0.0** | **0.0** | **Sep-2019** | **284.21** | **Oct-2019** | **Oct-2019** | **0** | **NaN** | **1** | **Individual** | **NaN** | **NaN** | **NaN** | **0** | **0** | **21235** | **0** | **1** | **0** | **1**20937676033122622200010042359111989.3001401401313013NaN15440474528979000090.91001034386212351050012386NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **17100** | **17100** | **17100** | **36 months** | **18.94%** | **626.3** | **D** | **D2** | **Receptionist** | **10+ years** | **RENT** | **38000** | **Verified** | **Dec-2018** | **Current** | **n** | **NaN** | **NaN** | **debt\_consolidation** | **Debt consolidation** | **150xx** | **PA** | **38.09** | **0** | **Mar-1998** | **1** | **47** | **NaN** | **14** | **0** | **15323** | **53%** | **21** | **w** | **13682.21** | **13682.21** | **5609.71** | **5609.71** | **3417.79** | **2191.92** | **0.0** | **0.0** | **0.0** | **Sep-2019** | **626.3** | **Oct-2019** | **Oct-2019** | **0** | **NaN** | **1** | **Individual** | **NaN** | **NaN** | **NaN** | **0** | **0** | **43351** | **1** | **2** | **1** | **1**102802867157533532917000263096415077001252305525NaN5NaN04125551214914NaN00295750070954433511660041784NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |

The extra rows at the top and bottom of the file have done two things:

1) The top row has made it so that the entire dataset is being interpreted as column headers

2) The bottom two rows have been read into the ‘id’ column and are causing every column to have at least two NaN values in it.

Copy

### We can fix the header problem by using the 'skiprows' parameter

df = pd.read\_csv('LoanStats\_2018Q4.csv', skiprows=1)

print(df.shape)

df.head()

Copy

/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (0,123,124,125,128,129,130,133) have mixed types. Specify dtype option on import or set low\_memory=False.

interactivity=interactivity, compiler=compiler, result=result)

(128414, 144)

|  | **id** | **member\_id** | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **url** | **desc** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **...** | **percent\_bc\_gt\_75** | **pub\_rec\_bankruptcies** | **tax\_liens** | **tot\_hi\_cred\_lim** | **total\_bal\_ex\_mort** | **total\_bc\_limit** | **total\_il\_high\_credit\_limit** | **revol\_bal\_joint** | **sec\_app\_earliest\_cr\_line** | **sec\_app\_inq\_last\_6mths** | **sec\_app\_mort\_acc** | **sec\_app\_open\_acc** | **sec\_app\_revol\_util** | **sec\_app\_open\_act\_il** | **sec\_app\_num\_rev\_accts** | **sec\_app\_chargeoff\_within\_12\_mths** | **sec\_app\_collections\_12\_mths\_ex\_med** | **sec\_app\_mths\_since\_last\_major\_derog** | **hardship\_flag** | **hardship\_type** | **hardship\_reason**hardship\_statusdeferral\_termhardship\_amounthardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_term |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | NaN | NaN | 35000.0 | 35000.0 | 35000.0 | 36 months | 14.47% | 1204.23 | C | C2 | Staff Physician | 8 years | MORTGAGE | 360000.0 | Verified | Dec-2018 | Fully Paid | n | NaN | NaN | credit\_card | Credit card refinancing | 336xx | FL | 19.90 | 0.0 | Apr-1995 | 1.0 | NaN | NaN | 24.0 | 0.0 | 57259.0 | 43.2% | 51.0 | w | 0.00 | 0.00 | 38187.046837 | 38187.05 | ... | 30.8 | 0.0 | 0.0 | 1222051.0 | 169286.0 | 124600.0 | 258401.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **1** | NaN | NaN | 5000.0 | 5000.0 | 5000.0 | 36 months | 22.35% | 191.86 | D | D5 | Director of Sales | 10+ years | MORTGAGE | 72000.0 | Source Verified | Dec-2018 | Fully Paid | n | NaN | NaN | debt\_consolidation | Debt consolidation | 333xx | FL | 20.12 | 0.0 | Mar-2010 | 0.0 | NaN | NaN | 13.0 | 0.0 | 11720.0 | 47.1% | 26.0 | f | 0.00 | 0.00 | 5615.977674 | 5615.98 | ... | 50.0 | 0.0 | 0.0 | 218686.0 | 34418.0 | 18200.0 | 37786.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **2** | NaN | NaN | 10000.0 | 10000.0 | 10000.0 | 60 months | 23.40% | 284.21 | E | E1 | NaN | < 1 year | RENT | 55000.0 | Source Verified | Dec-2018 | Current | n | NaN | NaN | debt\_consolidation | Debt consolidation | 902xx | CA | 13.51 | 0.0 | Apr-2007 | 0.0 | 44.0 | 88.0 | 9.0 | 1.0 | 11859.0 | 53.9% | 11.0 | w | 9131.55 | 9131.55 | 2538.390000 | 2538.39 | ... | 100.0 | 1.0 | 0.0 | 34386.0 | 21235.0 | 10500.0 | 12386.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **3** | NaN | NaN | 17100.0 | 17100.0 | 17100.0 | 36 months | 18.94% | 626.30 | D | D2 | Receptionist | 10+ years | RENT | 38000.0 | Verified | Dec-2018 | Current | n | NaN | NaN | debt\_consolidation | Debt consolidation | 150xx | PA | 38.09 | 0.0 | Mar-1998 | 1.0 | 47.0 | NaN | 14.0 | 0.0 | 15323.0 | 53% | 21.0 | w | 13682.21 | 13682.21 | 5609.710000 | 5609.71 | ... | 75.0 | 0.0 | 0.0 | 70954.0 | 43351.0 | 16600.0 | 41784.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **4** | NaN | NaN | 4000.0 | 4000.0 | 4000.0 | 36 months | 10.72% | 130.43 | B | B2 | Extrusion assistant | 10+ years | MORTGAGE | 56000.0 | Verified | Dec-2018 | Current | n | NaN | NaN | credit\_card | Credit card refinancing | 301xx | GA | 31.03 | 0.0 | Sep-2006 | 0.0 | NaN | NaN | 7.0 | 0.0 | 4518.0 | 28.6% | 11.0 | w | 3116.62 | 3116.62 | 1160.780000 | 1160.78 | ... | 0.0 | 0.0 | 0.0 | 221310.0 | 71375.0 | 12300.0 | 77865.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |

5 rows × 144 columns

Lets look at the NaN values of each column so that you can see the problem that the extrows at the bottom of the file are creating for us

Copy

### Sum null values by column and sort from least to greatest

df.isnull().sum().sort\_values()

Copy

inq\_fi 2

delinq\_amnt 2

chargeoff\_within\_12\_mths 2

acc\_open\_past\_24mths 2

inq\_last\_12m 2

total\_cu\_tl 2

total\_rev\_hi\_lim 2

max\_bal\_bc 2

open\_rv\_24m 2

open\_rv\_12m 2

total\_bal\_il 2

open\_il\_24m 2

hardship\_flag 2

open\_act\_il 2

open\_acc\_6m 2

tot\_cur\_bal 2

tot\_coll\_amt 2

acc\_now\_delinq 2

application\_type 2

policy\_code 2

mo\_sin\_old\_rev\_tl\_op 2

mo\_sin\_rcnt\_rev\_tl\_op 2

mo\_sin\_rcnt\_tl 2

mort\_acc 2

total\_il\_high\_credit\_limit 2

total\_bc\_limit 2

total\_bal\_ex\_mort 2

tot\_hi\_cred\_lim 2

tax\_liens 2

pub\_rec\_bankruptcies 2

pct\_tl\_nvr\_dlq 2

num\_tl\_op\_past\_12m 2

num\_tl\_90g\_dpd\_24m 2

collections\_12\_mths\_ex\_med 2

num\_tl\_30dpd 2

num\_rev\_tl\_bal\_gt\_0 2

num\_rev\_accts 2

num\_op\_rev\_tl 2

num\_il\_tl 2

num\_bc\_tl 2

num\_bc\_sats 2

num\_actv\_rev\_tl 2

num\_actv\_bc\_tl 2

num\_accts\_ever\_120\_pd 2

num\_sats 2

last\_pymnt\_amnt 2

open\_il\_12m 2

debt\_settlement\_flag 2

pub\_rec 2

open\_acc 2

grade 2

sub\_grade 2

home\_ownership 2

inq\_last\_6mths 2

annual\_inc 2

verification\_status 2

earliest\_cr\_line 2

delinq\_2yrs 2

issue\_d 2

addr\_state 2

zip\_code 2

title 2

loan\_status 2

pymnt\_plan 2

installment 2

revol\_bal 2

term 2

collection\_recovery\_fee 2

loan\_amnt 2

recoveries 2

total\_rec\_late\_fee 2

funded\_amnt 2

funded\_amnt\_inv 2

total\_acc 2

total\_rec\_int 2

purpose 2

total\_rec\_prncp 2

total\_pymnt\_inv 2

total\_pymnt 2

out\_prncp\_inv 2

out\_prncp 2

initial\_list\_status 2

int\_rate 2

last\_credit\_pull\_d 3

avg\_cur\_bal 15

all\_util 39

revol\_util 158

last\_pymnt\_d 161

dti 239

mths\_since\_recent\_bc 1593

bc\_open\_to\_buy 1693

percent\_bc\_gt\_75 1694

bc\_util 1756

num\_tl\_120dpd\_2m 2861

mths\_since\_rcnt\_il 4480

mo\_sin\_old\_il\_acct 4480

emp\_length 11706

mths\_since\_recent\_inq 16049

next\_pymnt\_d 19465

il\_util 20276

emp\_title 20949

mths\_since\_last\_delinq 72198

mths\_since\_recent\_revol\_delinq 91632

mths\_since\_last\_major\_derog 99234

mths\_since\_recent\_bc\_dlq 103245

sec\_app\_open\_acc 111632

sec\_app\_mort\_acc 111632

sec\_app\_inq\_last\_6mths 111632

sec\_app\_earliest\_cr\_line 111632

revol\_bal\_joint 111632

annual\_inc\_joint 111632

dti\_joint 111632

sec\_app\_collections\_12\_mths\_ex\_med 111632

sec\_app\_chargeoff\_within\_12\_mths 111632

sec\_app\_num\_rev\_accts 111632

sec\_app\_open\_act\_il 111632

sec\_app\_revol\_util 111890

mths\_since\_last\_record 112964

verification\_status\_joint 113566

sec\_app\_mths\_since\_last\_major\_derog 123260

debt\_settlement\_flag\_date 128199

settlement\_date 128199

settlement\_amount 128199

settlement\_status 128199

settlement\_term 128199

settlement\_percentage 128199

hardship\_status 128318

deferral\_term 128318

hardship\_amount 128318

hardship\_start\_date 128318

hardship\_end\_date 128318

payment\_plan\_start\_date 128318

hardship\_length 128318

hardship\_dpd 128318

hardship\_loan\_status 128318

hardship\_reason 128318

hardship\_payoff\_balance\_amount 128318

hardship\_last\_payment\_amount 128318

hardship\_type 128318

orig\_projected\_additional\_accrued\_interest 128322

id 128412

url 128414

member\_id 128414

desc 128414

dtype: int64

Copy

### Address the extra NaNs in each column by skipping the footer as well.

df = pd.read\_csv('LoanStats\_2018Q4.csv',

skiprows=1,

skipfooter=2,

engine='python')

print(df.shape)

df.head()

Copy

(128412, 144)

|  | **id** | **member\_id** | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **url** | **desc** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **...** | **percent\_bc\_gt\_75** | **pub\_rec\_bankruptcies** | **tax\_liens** | **tot\_hi\_cred\_lim** | **total\_bal\_ex\_mort** | **total\_bc\_limit** | **total\_il\_high\_credit\_limit** | **revol\_bal\_joint** | **sec\_app\_earliest\_cr\_line** | **sec\_app\_inq\_last\_6mths** | **sec\_app\_mort\_acc** | **sec\_app\_open\_acc** | **sec\_app\_revol\_util** | **sec\_app\_open\_act\_il** | **sec\_app\_num\_rev\_accts** | **sec\_app\_chargeoff\_within\_12\_mths** | **sec\_app\_collections\_12\_mths\_ex\_med** | **sec\_app\_mths\_since\_last\_major\_derog** | **hardship\_flag** | **hardship\_type** | **hardship\_reason**hardship\_statusdeferral\_termhardship\_amounthardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_term |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | NaN | NaN | 35000 | 35000 | 35000.0 | 36 months | 14.47% | 1204.23 | C | C2 | Staff Physician | 8 years | MORTGAGE | 360000.0 | Verified | Dec-2018 | Fully Paid | n | NaN | NaN | credit\_card | Credit card refinancing | 336xx | FL | 19.90 | 0 | Apr-1995 | 1 | NaN | NaN | 24 | 0 | 57259 | 43.2% | 51 | w | 0.00 | 0.00 | 38187.046837 | 38187.05 | ... | 30.8 | 0 | 0 | 1222051 | 169286 | 124600 | 258401 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **1** | NaN | NaN | 5000 | 5000 | 5000.0 | 36 months | 22.35% | 191.86 | D | D5 | Director of Sales | 10+ years | MORTGAGE | 72000.0 | Source Verified | Dec-2018 | Fully Paid | n | NaN | NaN | debt\_consolidation | Debt consolidation | 333xx | FL | 20.12 | 0 | Mar-2010 | 0 | NaN | NaN | 13 | 0 | 11720 | 47.1% | 26 | f | 0.00 | 0.00 | 5615.977674 | 5615.98 | ... | 50.0 | 0 | 0 | 218686 | 34418 | 18200 | 37786 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **2** | NaN | NaN | 10000 | 10000 | 10000.0 | 60 months | 23.40% | 284.21 | E | E1 | NaN | < 1 year | RENT | 55000.0 | Source Verified | Dec-2018 | Current | n | NaN | NaN | debt\_consolidation | Debt consolidation | 902xx | CA | 13.51 | 0 | Apr-2007 | 0 | 44.0 | 88.0 | 9 | 1 | 11859 | 53.9% | 11 | w | 9131.55 | 9131.55 | 2538.390000 | 2538.39 | ... | 100.0 | 1 | 0 | 34386 | 21235 | 10500 | 12386 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **3** | NaN | NaN | 17100 | 17100 | 17100.0 | 36 months | 18.94% | 626.30 | D | D2 | Receptionist | 10+ years | RENT | 38000.0 | Verified | Dec-2018 | Current | n | NaN | NaN | debt\_consolidation | Debt consolidation | 150xx | PA | 38.09 | 0 | Mar-1998 | 1 | 47.0 | NaN | 14 | 0 | 15323 | 53% | 21 | w | 13682.21 | 13682.21 | 5609.710000 | 5609.71 | ... | 75.0 | 0 | 0 | 70954 | 43351 | 16600 | 41784 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **4** | NaN | NaN | 4000 | 4000 | 4000.0 | 36 months | 10.72% | 130.43 | B | B2 | Extrusion assistant | 10+ years | MORTGAGE | 56000.0 | Verified | Dec-2018 | Current | n | NaN | NaN | credit\_card | Credit card refinancing | 301xx | GA | 31.03 | 0 | Sep-2006 | 0 | NaN | NaN | 7 | 0 | 4518 | 28.6% | 11 | w | 3116.62 | 3116.62 | 1160.780000 | 1160.78 | ... | 0.0 | 0 | 0 | 221310 | 71375 | 12300 | 77865 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |

5 rows × 144 columns

Copy

df.isnull().sum().sort\_values()

Copy

inq\_fi 0

delinq\_amnt 0

chargeoff\_within\_12\_mths 0

acc\_open\_past\_24mths 0

inq\_last\_12m 0

total\_cu\_tl 0

total\_rev\_hi\_lim 0

max\_bal\_bc 0

open\_rv\_24m 0

open\_rv\_12m 0

total\_bal\_il 0

open\_il\_24m 0

hardship\_flag 0

open\_act\_il 0

open\_acc\_6m 0

tot\_cur\_bal 0

tot\_coll\_amt 0

acc\_now\_delinq 0

application\_type 0

policy\_code 0

mo\_sin\_old\_rev\_tl\_op 0

mo\_sin\_rcnt\_rev\_tl\_op 0

mo\_sin\_rcnt\_tl 0

mort\_acc 0

total\_il\_high\_credit\_limit 0

total\_bc\_limit 0

total\_bal\_ex\_mort 0

tot\_hi\_cred\_lim 0

tax\_liens 0

pub\_rec\_bankruptcies 0

pct\_tl\_nvr\_dlq 0

num\_tl\_op\_past\_12m 0

num\_tl\_90g\_dpd\_24m 0

collections\_12\_mths\_ex\_med 0

num\_tl\_30dpd 0

num\_rev\_tl\_bal\_gt\_0 0

num\_rev\_accts 0

num\_op\_rev\_tl 0

num\_il\_tl 0

num\_bc\_tl 0

num\_bc\_sats 0

num\_actv\_rev\_tl 0

num\_actv\_bc\_tl 0

num\_accts\_ever\_120\_pd 0

num\_sats 0

last\_pymnt\_amnt 0

open\_il\_12m 0

debt\_settlement\_flag 0

pub\_rec 0

open\_acc 0

grade 0

sub\_grade 0

home\_ownership 0

inq\_last\_6mths 0

annual\_inc 0

verification\_status 0

earliest\_cr\_line 0

delinq\_2yrs 0

issue\_d 0

addr\_state 0

zip\_code 0

title 0

loan\_status 0

pymnt\_plan 0

installment 0

revol\_bal 0

term 0

collection\_recovery\_fee 0

loan\_amnt 0

recoveries 0

total\_rec\_late\_fee 0

funded\_amnt 0

funded\_amnt\_inv 0

total\_acc 0

total\_rec\_int 0

purpose 0

total\_rec\_prncp 0

total\_pymnt\_inv 0

total\_pymnt 0

out\_prncp\_inv 0

out\_prncp 0

initial\_list\_status 0

int\_rate 0

last\_credit\_pull\_d 1

avg\_cur\_bal 13

all\_util 37

revol\_util 156

last\_pymnt\_d 159

dti 237

mths\_since\_recent\_bc 1591

bc\_open\_to\_buy 1691

percent\_bc\_gt\_75 1692

bc\_util 1754

num\_tl\_120dpd\_2m 2859

mths\_since\_rcnt\_il 4478

mo\_sin\_old\_il\_acct 4478

emp\_length 11704

mths\_since\_recent\_inq 16047

next\_pymnt\_d 19463

il\_util 20274

emp\_title 20947

mths\_since\_last\_delinq 72196

mths\_since\_recent\_revol\_delinq 91630

mths\_since\_last\_major\_derog 99232

mths\_since\_recent\_bc\_dlq 103243

sec\_app\_open\_acc 111630

sec\_app\_mort\_acc 111630

sec\_app\_inq\_last\_6mths 111630

sec\_app\_earliest\_cr\_line 111630

revol\_bal\_joint 111630

annual\_inc\_joint 111630

dti\_joint 111630

sec\_app\_collections\_12\_mths\_ex\_med 111630

sec\_app\_chargeoff\_within\_12\_mths 111630

sec\_app\_num\_rev\_accts 111630

sec\_app\_open\_act\_il 111630

sec\_app\_revol\_util 111888

mths\_since\_last\_record 112962

verification\_status\_joint 113564

sec\_app\_mths\_since\_last\_major\_derog 123258

debt\_settlement\_flag\_date 128197

settlement\_date 128197

settlement\_amount 128197

settlement\_status 128197

settlement\_term 128197

settlement\_percentage 128197

hardship\_status 128316

deferral\_term 128316

hardship\_amount 128316

hardship\_start\_date 128316

hardship\_end\_date 128316

payment\_plan\_start\_date 128316

hardship\_length 128316

hardship\_dpd 128316

hardship\_loan\_status 128316

hardship\_reason 128316

hardship\_payoff\_balance\_amount 128316

hardship\_last\_payment\_amount 128316

hardship\_type 128316

orig\_projected\_additional\_accrued\_interest 128320

url 128412

member\_id 128412

desc 128412

id 128412

dtype: int64

For good measure, we’ll also drop some columns that are made up completely of NaN values.

Why might LendingClub have included columns in their dataset that are 100% blank?

Copy

df = df.drop(['url', 'member\_id', 'desc', 'id'], axis=1)

print(df.shape)

df.head()

Copy

(128412, 140)

|  | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **total\_rec\_prncp** | **total\_rec\_int** | **total\_rec\_late\_fee** | **recoveries** | **...** | **percent\_bc\_gt\_75** | **pub\_rec\_bankruptcies** | **tax\_liens** | **tot\_hi\_cred\_lim** | **total\_bal\_ex\_mort** | **total\_bc\_limit** | **total\_il\_high\_credit\_limit** | **revol\_bal\_joint** | **sec\_app\_earliest\_cr\_line** | **sec\_app\_inq\_last\_6mths** | **sec\_app\_mort\_acc** | **sec\_app\_open\_acc** | **sec\_app\_revol\_util** | **sec\_app\_open\_act\_il** | **sec\_app\_num\_rev\_accts** | **sec\_app\_chargeoff\_within\_12\_mths** | **sec\_app\_collections\_12\_mths\_ex\_med** | **sec\_app\_mths\_since\_last\_major\_derog** | **hardship\_flag** | **hardship\_type** | **hardship\_reason**hardship\_statusdeferral\_termhardship\_amounthardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_term |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 35000 | 35000 | 35000.0 | 36 months | 14.47% | 1204.23 | C | C2 | Staff Physician | 8 years | MORTGAGE | 360000.0 | Verified | Dec-2018 | Fully Paid | n | credit\_card | Credit card refinancing | 336xx | FL | 19.90 | 0 | Apr-1995 | 1 | NaN | NaN | 24 | 0 | 57259 | 43.2% | 51 | w | 0.00 | 0.00 | 38187.046837 | 38187.05 | 35000.00 | 3187.05 | 0.0 | 0.0 | ... | 30.8 | 0 | 0 | 1222051 | 169286 | 124600 | 258401 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **1** | 5000 | 5000 | 5000.0 | 36 months | 22.35% | 191.86 | D | D5 | Director of Sales | 10+ years | MORTGAGE | 72000.0 | Source Verified | Dec-2018 | Fully Paid | n | debt\_consolidation | Debt consolidation | 333xx | FL | 20.12 | 0 | Mar-2010 | 0 | NaN | NaN | 13 | 0 | 11720 | 47.1% | 26 | f | 0.00 | 0.00 | 5615.977674 | 5615.98 | 5000.00 | 615.98 | 0.0 | 0.0 | ... | 50.0 | 0 | 0 | 218686 | 34418 | 18200 | 37786 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **2** | 10000 | 10000 | 10000.0 | 60 months | 23.40% | 284.21 | E | E1 | NaN | < 1 year | RENT | 55000.0 | Source Verified | Dec-2018 | Current | n | debt\_consolidation | Debt consolidation | 902xx | CA | 13.51 | 0 | Apr-2007 | 0 | 44.0 | 88.0 | 9 | 1 | 11859 | 53.9% | 11 | w | 9131.55 | 9131.55 | 2538.390000 | 2538.39 | 868.45 | 1669.94 | 0.0 | 0.0 | ... | 100.0 | 1 | 0 | 34386 | 21235 | 10500 | 12386 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **3** | 17100 | 17100 | 17100.0 | 36 months | 18.94% | 626.30 | D | D2 | Receptionist | 10+ years | RENT | 38000.0 | Verified | Dec-2018 | Current | n | debt\_consolidation | Debt consolidation | 150xx | PA | 38.09 | 0 | Mar-1998 | 1 | 47.0 | NaN | 14 | 0 | 15323 | 53% | 21 | w | 13682.21 | 13682.21 | 5609.710000 | 5609.71 | 3417.79 | 2191.92 | 0.0 | 0.0 | ... | 75.0 | 0 | 0 | 70954 | 43351 | 16600 | 41784 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |
| **4** | 4000 | 4000 | 4000.0 | 36 months | 10.72% | 130.43 | B | B2 | Extrusion assistant | 10+ years | MORTGAGE | 56000.0 | Verified | Dec-2018 | Current | n | credit\_card | Credit card refinancing | 301xx | GA | 31.03 | 0 | Sep-2006 | 0 | NaN | NaN | 7 | 0 | 4518 | 28.6% | 11 | w | 3116.62 | 3116.62 | 1160.780000 | 1160.78 | 883.38 | 277.40 | 0.0 | 0.0 | ... | 0.0 | 0 | 0 | 221310 | 71375 | 12300 | 77865 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN |

5 rows × 140 columns

##### Clean up the int\_rate column

When we’re preparing a dataset for a machine learning model we typically want to represent don’t want to leave any string values in our dataset –because it’s hard to do math on words.

Specifically, we have a column that is representing a numeric value, but currently doesn’t have a numeric datatype. Lets look at the first 10 values of the int\_rate column

Copy

### Look at the first 10 values of the int\_rate column

df['int\_rate'][:10]

Copy

0 14.47%

1 22.35%

2 23.40%

3 18.94%

4 10.72%

5 11.31%

6 11.31%

7 11.80%

8 11.80%

9 18.94%

Name: int\_rate, dtype: object

Copy

### Look at a specific value from the int\_rate column

df['int\_rate'][0]

Copy

' 14.47%'

Problems that we need to address with this column:

* String column that should be numeric
* Percent Sign % included with the number
* Leading space at the beginning of the string

However, we’re not going to try and write exactly the right code to fix this column in one go. We’re going to methodically build up to the code that will help us address these problems.

Copy

### Lets start with just fixing a single string.

### If we can fix one, we can usually fix all of them

int\_rate = ' 14.17%'

Copy

int\_rate.strip()

Copy

'14.17%'

Copy

int\_rate.strip('%')

Copy

' 14.17'

Copy

int\_rate.strip().strip('%')

Copy

'14.17'

Copy

type(int\_rate.strip().strip('%'))

Copy

str

Copy

### "Cast" the string value to a float

float(int\_rate.strip().strip('%'))

Copy

14.17

Copy

type(float(int\_rate.strip().strip('%')))

Copy

float

##### Write a function to make our solution reusable!

Copy

### Write a function that can do what we have written above to any

### string that is passsed to it.

def int\_rate\_to\_float(cell\_contents):

return float(cell\_contents.strip().strip('%'))

Copy

### Test out our function by calling it on our example

int\_rate\_to\_float(int\_rate)

Copy

14.17

Copy

### is the data type correct?

type(int\_rate\_to\_float(int\_rate))

Copy

float

##### Apply our solution to every cell in a column

Copy

df['int\_rate\_float'] = df['int\_rate'].apply(int\_rate\_to\_float)

df.head()

|  | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **total\_rec\_prncp** | **total\_rec\_int** | **total\_rec\_late\_fee** | **recoveries** | **...** | **pub\_rec\_bankruptcies** | **tax\_liens** | **tot\_hi\_cred\_lim** | **total\_bal\_ex\_mort** | **total\_bc\_limit** | **total\_il\_high\_credit\_limit** | **revol\_bal\_joint** | **sec\_app\_earliest\_cr\_line** | **sec\_app\_inq\_last\_6mths** | **sec\_app\_mort\_acc** | **sec\_app\_open\_acc** | **sec\_app\_revol\_util** | **sec\_app\_open\_act\_il** | **sec\_app\_num\_rev\_accts** | **sec\_app\_chargeoff\_within\_12\_mths** | **sec\_app\_collections\_12\_mths\_ex\_med** | **sec\_app\_mths\_since\_last\_major\_derog** | **hardship\_flag** | **hardship\_type** | **hardship\_reason** | **hardship\_status**deferral\_termhardship\_amounthardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_termint\_rate\_float |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 35000 | 35000 | 35000.0 | 36 months | 14.47% | 1204.23 | C | C2 | Staff Physician | 8 years | MORTGAGE | 360000.0 | Verified | Dec-2018 | Fully Paid | n | credit\_card | Credit card refinancing | 336xx | FL | 19.90 | 0 | Apr-1995 | 1 | NaN | NaN | 24 | 0 | 57259 | 43.2% | 51 | w | 0.00 | 0.00 | 38187.046837 | 38187.05 | 35000.00 | 3187.05 | 0.0 | 0.0 | ... | 0 | 0 | 1222051 | 169286 | 124600 | 258401 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN14.47 |

5 rows × 141 columns

Copy

### What type of data is held in our new column?

### Look at the datatypes of the last 5 columns

df.dtypes[-5:]

Copy

settlement\_date object

settlement\_amount float64

settlement\_percentage float64

settlement\_term float64

int\_rate\_float float64

dtype: object

##### Challenge

We can create a new column with our cleaned values or overwrite the original, whatever we think best suits our needs. On your assignment you will take the same approach in trying to methodically build up the complexity of your code until you have a few lines that will work for any cell in a column. At that point you’ll contain all of that functionality in a reusable function block and then use the .apply() function to… well… apply those changes to an entire column.

#### Learn to modify or create dataframe columns using the .apply() function

We’re already seen one example of using the .apply() function to clean up a column. This function lets us write a function that works on a single cell of a dataframe but then make that change uniformly to an entire column. It’s a very powerful method. Let’s see if we can do it again, but this time on a slightly more complicated use case.

##### Overview

Remember, the goal here is to write a function that will work correctly on any **individual** cell of a specific column. Then we can reuse that function on those individual cells of a dataframe column via the .apply() function.

Lets clean up the emp\_title “Employment Title” column

##### Follow Along

First we’ll try and diagnose how bad the problem is and what improvements we might be able to make.

Copy

### Look at the top 20 employment titles

df['emp\_title'].value\_counts(dropna=False, ascending=False)[:20]

Copy

NaN 20947

Teacher 2090

Manager 1773

Registered Nurse 952

Driver 924

RN 726

Supervisor 697

Sales 580

Project Manager 526

General Manager 523

Office Manager 521

Owner 420

Director 402

Truck Driver 387

Operations Manager 387

Nurse 326

Engineer 325

Sales Manager 304

manager 301

Supervisor 270

Name: emp\_title, dtype: int64

Copy

### How many different unique employment titles are there currently?

len(df['emp\_title'].value\_counts())

Copy

43892

Copy

### How often is the employment\_title null?

df['emp\_title'].isnull().sum()

Copy

20947

What are some possible reasons as to why a person’s employment title may have not been provided?

Copy

### Create some examples that represent the cases that we want to clean up

examples = ['owner', 'Supervisor', ' Project Manager', np.NaN]

Copy

### Write a function to clean up these use cases and increase uniformity.

def clean\_title(title):

if isinstance(title, str):

return title.strip().title()

else:

return "Unknown"

for example in examples:

print(clean\_title(example))

Copy

Owner

Supervisor

Project Manager

Unknown

Copy

### list comprehensions can combine function calls and for loops over lists

### into one succinct and fairly readable single line of code.

[clean\_title(example) for example in examples]

Copy

['Owner', 'Supervisor', 'Project Manager', 'Unknown']

Copy

### We have a function that works as expected. Lets apply it to our column.

### This time we'll overwrite the original column

df['emp\_title'] = df['emp\_title'].apply(clean\_title)

df.head()

|  | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **total\_rec\_prncp** | **total\_rec\_int** | **total\_rec\_late\_fee** | **recoveries** | **...** | **pub\_rec\_bankruptcies** | **tax\_liens** | **tot\_hi\_cred\_lim** | **total\_bal\_ex\_mort** | **total\_bc\_limit** | **total\_il\_high\_credit\_limit** | **revol\_bal\_joint** | **sec\_app\_earliest\_cr\_line** | **sec\_app\_inq\_last\_6mths** | **sec\_app\_mort\_acc** | **sec\_app\_open\_acc** | **sec\_app\_revol\_util** | **sec\_app\_open\_act\_il** | **sec\_app\_num\_rev\_accts** | **sec\_app\_chargeoff\_within\_12\_mths** | **sec\_app\_collections\_12\_mths\_ex\_med** | **sec\_app\_mths\_since\_last\_major\_derog** | **hardship\_flag** | **hardship\_type** | **hardship\_reason** | **hardship\_status**deferral\_termhardship\_amounthardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_termint\_rate\_float |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 35000 | 35000 | 35000.0 | 36 months | 14.47% | 1204.23 | C | C2 | Staff Physician | 8 years | MORTGAGE | 360000.0 | Verified | Dec-2018 | Fully Paid | n | credit\_card | Credit card refinancing | 336xx | FL | 19.90 | 0 | Apr-1995 | 1 | NaN | NaN | 24 | 0 | 57259 | 43.2% | 51 | w | 0.00 | 0.00 | 38187.046837 | 38187.05 | 35000.00 | 3187.05 | 0.0 | 0.0 | ... | 0 | 0 | 1222051 | 169286 | 124600 | 258401 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN14.47 |
| **1** | 5000 | 5000 | 5000.0 | 36 months | 22.35% | 191.86 | D | D5 | Director Of Sales | 10+ years | MORTGAGE | 72000.0 | Source Verified | Dec-2018 | Fully Paid | n | debt\_consolidation | Debt consolidation | 333xx | FL | 20.12 | 0 | Mar-2010 | 0 | NaN | NaN | 13 | 0 | 11720 | 47.1% | 26 | f | 0.00 | 0.00 | 5615.977674 | 5615.98 | 5000.00 | 615.98 | 0.0 | 0.0 | ... | 0 | 0 | 218686 | 34418 | 18200 | 37786 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN22.35 |
| **2** | 10000 | 10000 | 10000.0 | 60 months | 23.40% | 284.21 | E | E1 | Unknown | < 1 year | RENT | 55000.0 | Source Verified | Dec-2018 | Current | n | debt\_consolidation | Debt consolidation | 902xx | CA | 13.51 | 0 | Apr-2007 | 0 | 44.0 | 88.0 | 9 | 1 | 11859 | 53.9% | 11 | w | 9131.55 | 9131.55 | 2538.390000 | 2538.39 | 868.45 | 1669.94 | 0.0 | 0.0 | ... | 1 | 0 | 34386 | 21235 | 10500 | 12386 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN23.40 |
| **3** | 17100 | 17100 | 17100.0 | 36 months | 18.94% | 626.30 | D | D2 | Receptionist | 10+ years | RENT | 38000.0 | Verified | Dec-2018 | Current | n | debt\_consolidation | Debt consolidation | 150xx | PA | 38.09 | 0 | Mar-1998 | 1 | 47.0 | NaN | 14 | 0 | 15323 | 53% | 21 | w | 13682.21 | 13682.21 | 5609.710000 | 5609.71 | 3417.79 | 2191.92 | 0.0 | 0.0 | ... | 0 | 0 | 70954 | 43351 | 16600 | 41784 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN18.94 |
| **4** | 4000 | 4000 | 4000.0 | 36 months | 10.72% | 130.43 | B | B2 | Extrusion Assistant | 10+ years | MORTGAGE | 56000.0 | Verified | Dec-2018 | Current | n | credit\_card | Credit card refinancing | 301xx | GA | 31.03 | 0 | Sep-2006 | 0 | NaN | NaN | 7 | 0 | 4518 | 28.6% | 11 | w | 3116.62 | 3116.62 | 1160.780000 | 1160.78 | 883.38 | 277.40 | 0.0 | 0.0 | ... | 0 | 0 | 221310 | 71375 | 12300 | 77865 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN10.72 |

5 rows × 141 columns

We can use the same code as we did earlier to see how much progress was made.

Copy

### Look at the top 20 employment titles

df['emp\_title'].value\_counts(dropna=False, ascending=False)[:20]

Copy

Unknown 20947

Teacher 2557

Manager 2395

Registered Nurse 1418

Driver 1258

Supervisor 1160

Truck Driver 920

Rn 834

Office Manager 805

Sales 803

General Manager 791

Project Manager 720

Owner 625

Director 523

Operations Manager 518

Sales Manager 500

Police Officer 440

Nurse 425

Technician 420

Engineer 412

Name: emp\_title, dtype: int64

Copy

### How many different unique employment titles are there currently?

len(df['emp\_title'].value\_counts())

Copy

34902

Copy

### How often is the employment\_title null (NaN)?

df['emp\_title'].isnull().sum()

Copy

0

##### Challenge

Using the .apply() function isn’t always about creating new columns on a dataframe, we can use it to clean up or modify existing columns as well.

#### Learn to work with dates and times in pandas

Pandas has powerful functionality for working with dates and times -datetime objects. This is a unique strength of pandas is that it makes it extremely easy to parse dates and then create new columns on our dataframes that correspond to the year, month, day, second, quarter, day of the week, etc.

##### Overview

This section will demonstrate how to take a column of date strings, convert it to a datetime object and then use the datetime formatting .dt to access specific parts of the date (year, month, day) to generate useful columns on a dataframe.

Read Python Data Science Handbook [Chapter 3.11](https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html), Working with Time Series

##### Follow Along

### Work with Dates

pandas documentation

* [to\_datetime](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html)
* [Time/Date Components](https://pandas.pydata.org/pandas-docs/stable/timeseries.html#time-date-components) “You can access these properties via the .dt accessor”

Many of the most useful date columns in this dataset have the suffix \_d to indicate that they correspond to dates.

We’ll use a list comprehension to print them out

Copy

[col for col in df if col.endswith('\_d')]

Copy

['issue\_d', 'last\_pymnt\_d', 'next\_pymnt\_d', 'last\_credit\_pull\_d']

Lets look at the string format of the issue\_d column

Copy

df['issue\_d'][:10]

Copy

0 Dec-2018

1 Dec-2018

2 Dec-2018

3 Dec-2018

4 Dec-2018

5 Dec-2018

6 Dec-2018

7 Dec-2018

8 Dec-2018

9 Dec-2018

Name: issue\_d, dtype: object

Because this string format %m-%y is a common datetime format, we can just let Pandas detect this format and translate it to the appropriate datetime object.

Copy

df['issue\_d'] = pd.to\_datetime(df['issue\_d'], infer\_datetime\_format=True)

df.dtypes[:15]

Copy

loan\_amnt int64

funded\_amnt int64

funded\_amnt\_inv float64

term object

int\_rate object

installment float64

grade object

sub\_grade object

emp\_title object

emp\_length object

home\_ownership object

annual\_inc float64

verification\_status object

issue\_d datetime64[ns]

loan\_status object

dtype: object

Now we can see that the issue\_d column has been changed to hold datetime objects.

Lets look at one of the cells specifically to see what a datetime object looks like:

Copy

df['issue\_d'].iloc[0]

Copy

Timestamp('2018-12-01 00:00:00')

You can see how the month and year have been indicated by the strings that were contained in the column previously, and that the rest of the values have been inferred.

Copy

df['issue\_d'].head().values

Copy

array(['2018-12-01T00:00:00.000000000', '2018-12-01T00:00:00.000000000',

'2018-12-01T00:00:00.000000000', '2018-12-01T00:00:00.000000000',

'2018-12-01T00:00:00.000000000'], dtype='datetime64[ns]')

We can use the .dt accessor to now grab specific parts of the datetime object. Lets grab just the year from the all of the cells in the issue\_d column

Copy

df['issue\_d'].dt.year

Copy

0 2018

1 2018

2 2018

3 2018

4 2018

...

128407 2018

128408 2018

128409 2018

128410 2018

128411 2018

Name: issue\_d, Length: 128412, dtype: int64

Now the month.

Copy

df['issue\_d'].dt.month

Copy

0 12

1 12

2 12

3 12

4 12

..

128407 10

128408 10

128409 10

128410 10

128411 10

Name: issue\_d, Length: 128412, dtype: int64

It’s just that easy! Now, instead of printing them out, lets add these year and month values as new columns on our dataframe. Again, you’ll have to scroll all the way over to the right in the table to see the new columns.

Copy

df['issue\_year'] = df['issue\_d'].dt.year

df['issue\_month'] = df['issue\_d'].dt.month

df.head()

|  | **loan\_amnt** | **funded\_amnt** | **funded\_amnt\_inv** | **term** | **int\_rate** | **installment** | **grade** | **sub\_grade** | **emp\_title** | **emp\_length** | **home\_ownership** | **annual\_inc** | **verification\_status** | **issue\_d** | **loan\_status** | **pymnt\_plan** | **purpose** | **title** | **zip\_code** | **addr\_state** | **dti** | **delinq\_2yrs** | **earliest\_cr\_line** | **inq\_last\_6mths** | **mths\_since\_last\_delinq** | **mths\_since\_last\_record** | **open\_acc** | **pub\_rec** | **revol\_bal** | **revol\_util** | **total\_acc** | **initial\_list\_status** | **out\_prncp** | **out\_prncp\_inv** | **total\_pymnt** | **total\_pymnt\_inv** | **total\_rec\_prncp** | **total\_rec\_int** | **total\_rec\_late\_fee** | **recoveries** | **...** | **tot\_hi\_cred\_lim** | **total\_bal\_ex\_mort** | **total\_bc\_limit** | **total\_il\_high\_credit\_limit** | **revol\_bal\_joint** | **sec\_app\_earliest\_cr\_line** | **sec\_app\_inq\_last\_6mths** | **sec\_app\_mort\_acc** | **sec\_app\_open\_acc** | **sec\_app\_revol\_util** | **sec\_app\_open\_act\_il** | **sec\_app\_num\_rev\_accts** | **sec\_app\_chargeoff\_within\_12\_mths** | **sec\_app\_collections\_12\_mths\_ex\_med** | **sec\_app\_mths\_since\_last\_major\_derog** | **hardship\_flag** | **hardship\_type** | **hardship\_reason** | **hardship\_status** | **deferral\_term** | **hardship\_amount**hardship\_start\_datehardship\_end\_datepayment\_plan\_start\_datehardship\_lengthhardship\_dpdhardship\_loan\_statusorig\_projected\_additional\_accrued\_interesthardship\_payoff\_balance\_amounthardship\_last\_payment\_amountdebt\_settlement\_flagdebt\_settlement\_flag\_datesettlement\_statussettlement\_datesettlement\_amountsettlement\_percentagesettlement\_termint\_rate\_floatissue\_yearissue\_month |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 35000 | 35000 | 35000.0 | 36 months | 14.47% | 1204.23 | C | C2 | Staff Physician | 8 years | MORTGAGE | 360000.0 | Verified | 2018-12-01 | Fully Paid | n | credit\_card | Credit card refinancing | 336xx | FL | 19.90 | 0 | Apr-1995 | 1 | NaN | NaN | 24 | 0 | 57259 | 43.2% | 51 | w | 0.00 | 0.00 | 38187.046837 | 38187.05 | 35000.00 | 3187.05 | 0.0 | 0.0 | ... | 1222051 | 169286 | 124600 | 258401 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN14.47201812 |
| **1** | 5000 | 5000 | 5000.0 | 36 months | 22.35% | 191.86 | D | D5 | Director Of Sales | 10+ years | MORTGAGE | 72000.0 | Source Verified | 2018-12-01 | Fully Paid | n | debt\_consolidation | Debt consolidation | 333xx | FL | 20.12 | 0 | Mar-2010 | 0 | NaN | NaN | 13 | 0 | 11720 | 47.1% | 26 | f | 0.00 | 0.00 | 5615.977674 | 5615.98 | 5000.00 | 615.98 | 0.0 | 0.0 | ... | 218686 | 34418 | 18200 | 37786 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN22.35201812 |
| **2** | 10000 | 10000 | 10000.0 | 60 months | 23.40% | 284.21 | E | E1 | Unknown | < 1 year | RENT | 55000.0 | Source Verified | 2018-12-01 | Current | n | debt\_consolidation | Debt consolidation | 902xx | CA | 13.51 | 0 | Apr-2007 | 0 | 44.0 | 88.0 | 9 | 1 | 11859 | 53.9% | 11 | w | 9131.55 | 9131.55 | 2538.390000 | 2538.39 | 868.45 | 1669.94 | 0.0 | 0.0 | ... | 34386 | 21235 | 10500 | 12386 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN23.40201812 |
| **3** | 17100 | 17100 | 17100.0 | 36 months | 18.94% | 626.30 | D | D2 | Receptionist | 10+ years | RENT | 38000.0 | Verified | 2018-12-01 | Current | n | debt\_consolidation | Debt consolidation | 150xx | PA | 38.09 | 0 | Mar-1998 | 1 | 47.0 | NaN | 14 | 0 | 15323 | 53% | 21 | w | 13682.21 | 13682.21 | 5609.710000 | 5609.71 | 3417.79 | 2191.92 | 0.0 | 0.0 | ... | 70954 | 43351 | 16600 | 41784 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN18.94201812 |
| **4** | 4000 | 4000 | 4000.0 | 36 months | 10.72% | 130.43 | B | B2 | Extrusion Assistant | 10+ years | MORTGAGE | 56000.0 | Verified | 2018-12-01 | Current | n | credit\_card | Credit card refinancing | 301xx | GA | 31.03 | 0 | Sep-2006 | 0 | NaN | NaN | 7 | 0 | 4518 | 28.6% | 11 | w | 3116.62 | 3116.62 | 1160.780000 | 1160.78 | 883.38 | 277.40 | 0.0 | 0.0 | ... | 221310 | 71375 | 12300 | 77865 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | N | NaN | NaN | NaN | NaN | NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNNaNNaNNaNNaNNaNNaN10.72201812 |

5 rows × 143 columns

Because all of these dates come from Q4 of 2018, the issue\_d column isn’t all that interesting. Lets look at the earliest\_cr\_line column, which is also a string, but that could be converted to datetime format.

We’re going to create a new column called days\_from\_earliest\_credit\_to\_issue

It’s a long column header, but think about how valuable this piece of information could be. This number will essentially indicate the length of a person’s credit history and if that is correlated with repayment or other factors could be a valuable predictor!

Copy

df['earliest\_cr\_line'].head()

Copy

0 Apr-1995

1 Mar-2010

2 Apr-2007

3 Mar-1998

4 Sep-2006

Name: earliest\_cr\_line, dtype: object

Copy

df['earliest\_cr\_line'] = pd.to\_datetime(df['earliest\_cr\_line'],

infer\_datetime\_format=True)

What we’re about to do is so cool! Pandas’ datetime format is so smart that we can simply use the subtraction operator - in order to calculate the amount of time between two dates.

Think about everything that’s going on under the hood in order to give us such straightforward syntax! Handling months of different lengths, leap years, etc. Pandas datetime objects are seriously powerful!

Copy

df['days\_from\_earliest\_credit\_to\_issue'] = (df['issue\_d'] - df['earliest\_cr\_line']).dt.days

Copy

df['days\_from\_earliest\_credit\_to\_issue'].head()

Copy

0 8645

1 3197

2 4262

3 7580

4 4474

Name: days\_from\_earliest\_credit\_to\_issue, dtype: int64

What’s oldest credit history that was involved in Q4 2018?

Copy

df['days\_from\_earliest\_credit\_to\_issue'].describe()

Copy

count 128412.000000

mean 5859.891490

std 2886.535578

min 1126.000000

25% 4049.000000

50% 5266.000000

75% 7244.000000

max 25171.000000

Name: days\_from\_earliest\_credit\_to\_issue, dtype: float64

25,171 days is ~ 68.96 years of credit history!

##### Challenge

Pandas’ datetime format is so easy to work with that there’s really no excuse for not using dates to make features on a dataframe! Get ready to practice more of this on your assignment.

## Review

### Class Recordings

You can use class recordings to help you master the material.

* [**Make Features for DS14 w/ Ryan Allred**](https://youtu.be/sxW-yR14spA)

Unit 1 Sprint 1 Module 2

* [All previous recordings](https://learn.lambdaschool.com/archive/DS/module/recD0PnM64cmU06N1)

### Demonstrate Mastery

To demonstrate mastery of this module, you need to complete and pass a code review on each of the following:

* Objective challenge:

I hope you can see how we have used existing columns to create a new column on a dataset that say something new about our unit of observation. This is what making new features (columns) on a dataset is all about and why it’s so essential to data science –particularly predictive modeling “Machine Learning.”

We’ll spend the rest of the lecture and assignment today trying to get as good as we can at manipulating (cleaning) and creating new columns on datasets.

* Objective challenge:

We can create a new column with our cleaned values or overwrite the original, whatever we think best suits our needs. On your assignment you will take the same approach in trying to methodically build up the complexity of your code until you have a few lines that will work for any cell in a column. At that point you’ll contain all of that functionality in a reusable function block and then use the .apply() function to… well… apply those changes to an entire column.

* Objective challenge:

Using the .apply() function isn’t always about creating new columns on a dataframe, we can use it to clean up or modify existing columns as well.

* Objective challenge:

Pandas’ datetime format is so easy to work with that there’s really no excuse for not using dates to make features on a dataframe! Get ready to practice more of this on your assignment.

# **Join and Reshape Data**

Preparing data isn’t just about importing it and imputing missing values. Some of a Data Scientist’s most time consuming activities involved taking data from one format and transforming it into the format that is optimized for use by an algorithm or some other process. This is what’s known as “Data Wrangling” or “Data Munging” it’s all about “transforming and mapping data from one “raw” data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes. (Wikipedia - Data Wrangling)

**At the end of this module, you should be able to:**

* concatenate data with pandas
* merge data with pandas
* understand tidy data formatting
* melt and pivot data with pandas

#### Pro Tip

Before you put something on social media, ask yourself - “Would I be okay if this appeared on the front page of the New York Times?”

## Prepare

Review each preclass resource before class.

## Learn

#### Learn to concatenate data with pandas

Concatenating dataframes with Pandas is a vital data wrangling skill. This involves appending a dataframe to another either by row or by column. Often a significant amount of reshaping must occur before the two dataframes are ready to be concatenated. The need for concatenating dataframes often occurs when datasets are given as separate CSV files or when trying to re-combine a subset of a dataframe with the original.

##### Overview

Read Python Data Science Handbook [Chapter 3.6](https://jakevdp.github.io/PythonDataScienceHandbook/03.06-concat-and-append.html), Combining Datasets: Concat and Append

Review this Chis Albon documentation about [concatenating dataframes by row and by column](https://chrisalbon.com/python/data_wrangling/pandas_join_merge_dataframe/)

##### Follow Along

“Concatenate” is a fancy word for joining two things together. For example, we can concatenate two strings together using the + operator.

Copy

'We can join/concatenate two strings together ' + 'using the "+" operator.'

Copy

'We can join/concatenate two strings together using the "+" operator.'

When we “concatenate” two dataframes we will “stick them together” either by rows or columns. Lets look at some simple examples:

Copy

import pandas as pd

Copy

df1 = pd.DataFrame({'a': [1,2,3,4], 'b': [4,5,6,7], 'c': [7,8,9,10]})

df2 = pd.DataFrame({'a': [6,4,8,7], 'b': [9,4,3,2], 'c': [1,6,2,9]})

Copy

df1.head()

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| **0** | 1 | 4 | 7 |
| **1** | 2 | 5 | 8 |
| **2** | 3 | 6 | 9 |
| **3** | 4 | 7 | 10 |

Copy

df2.head()

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| **0** | 6 | 9 | 1 |
| **1** | 4 | 4 | 6 |
| **2** | 8 | 3 | 2 |
| **3** | 7 | 2 | 9 |

### Concatenate by Rows

concatenating by rows is the default behavior of pd.concat() This is often the most common form of concatenation.

Copy

concatenated\_by\_rows = pd.concat([df1,df2])

concatenated\_by\_rows.head(8)

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| **0** | 1 | 4 | 7 |
| **1** | 2 | 5 | 8 |
| **2** | 3 | 6 | 9 |
| **3** | 4 | 7 | 10 |
| **0** | 6 | 9 | 1 |
| **1** | 4 | 4 | 6 |
| **2** | 8 | 3 | 2 |
| **3** | 7 | 2 | 9 |

### Concatenate by Columns

Copy

concatenated\_by\_columns = pd.concat([df1,df2], axis=1)

concatenated\_by\_columns.head()

|  | **a** | **b** | **c** | **a** | **b** | **c** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 4 | 7 | 6 | 9 | 1 |
| **1** | 2 | 5 | 8 | 4 | 4 | 6 |
| **2** | 3 | 6 | 9 | 8 | 3 | 2 |
| **3** | 4 | 7 | 10 | 7 | 2 | 9 |

When concatenating dataframes, it is done using the column headers and row index values to match rows up. If these don’t match up, then NaN values will be added where matches can’t be found.

Copy

df3 = pd.DataFrame({'a': [4,3,2,1], 'b': [4,5,6,7], 'c': [7,8,9,10]})

df4 = pd.DataFrame({'a': [6,4,8,7,8], 'b': [9,4,3,2,1], 'd': [1,6,2,9,5]})

Copy

df3.head()

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| **0** | 4 | 4 | 7 |
| **1** | 3 | 5 | 8 |
| **2** | 2 | 6 | 9 |
| **3** | 1 | 7 | 10 |

Copy

df4.head()

|  | **a** | **b** | **d** |
| --- | --- | --- | --- |
| **0** | 6 | 9 | 1 |
| **1** | 4 | 4 | 6 |
| **2** | 8 | 3 | 2 |
| **3** | 7 | 2 | 9 |
| **4** | 8 | 1 | 5 |

### Concatenate by rows when not all column headers match

Copy

concatenated\_by\_rows = pd.concat([df3,df4], sort=True)

concatenated\_by\_rows.head(9)

|  | **a** | **b** | **c** | **d** |
| --- | --- | --- | --- | --- |
| **0** | 4 | 4 | 7.0 | NaN |
| **1** | 3 | 5 | 8.0 | NaN |
| **2** | 2 | 6 | 9.0 | NaN |
| **3** | 1 | 7 | 10.0 | NaN |
| **0** | 6 | 9 | NaN | 1.0 |
| **1** | 4 | 4 | NaN | 6.0 |
| **2** | 8 | 3 | NaN | 2.0 |
| **3** | 7 | 2 | NaN | 9.0 |
| **4** | 8 | 1 | NaN | 5.0 |

### Concatenate by columns when not all row indexes match

Copy

concatenated\_by\_columns = pd.concat([df3,df4], axis=1)

concatenated\_by\_columns.head()

|  | **a** | **b** | **c** | **a** | **b** | **d** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 4.0 | 4.0 | 7.0 | 6 | 9 | 1 |
| **1** | 3.0 | 5.0 | 8.0 | 4 | 4 | 6 |
| **2** | 2.0 | 6.0 | 9.0 | 8 | 3 | 2 |
| **3** | 1.0 | 7.0 | 10.0 | 7 | 2 | 9 |
| **4** | NaN | NaN | NaN | 8 | 1 | 5 |

Whenever we are combining dataframes, if appropriate values cannot be found based on the rules of the method we are using, then missing values will be filled with NaNs.

We’ll work with a dataset of [3 Million Instacart Orders, Open Sourced](https://tech.instacart.com/3-million-instacart-orders-open-sourced-d40d29ead6f2)!

The files that we will be working with are in a folder of CSVs, we need to load that folder of CSVs, explore the CSVs to make sure that we understand what we’re working with, and where the important data lies, and then work to combine the dataframes together as necessary.

Our goal is to reproduce this table which holds the first two orders for user id 1.

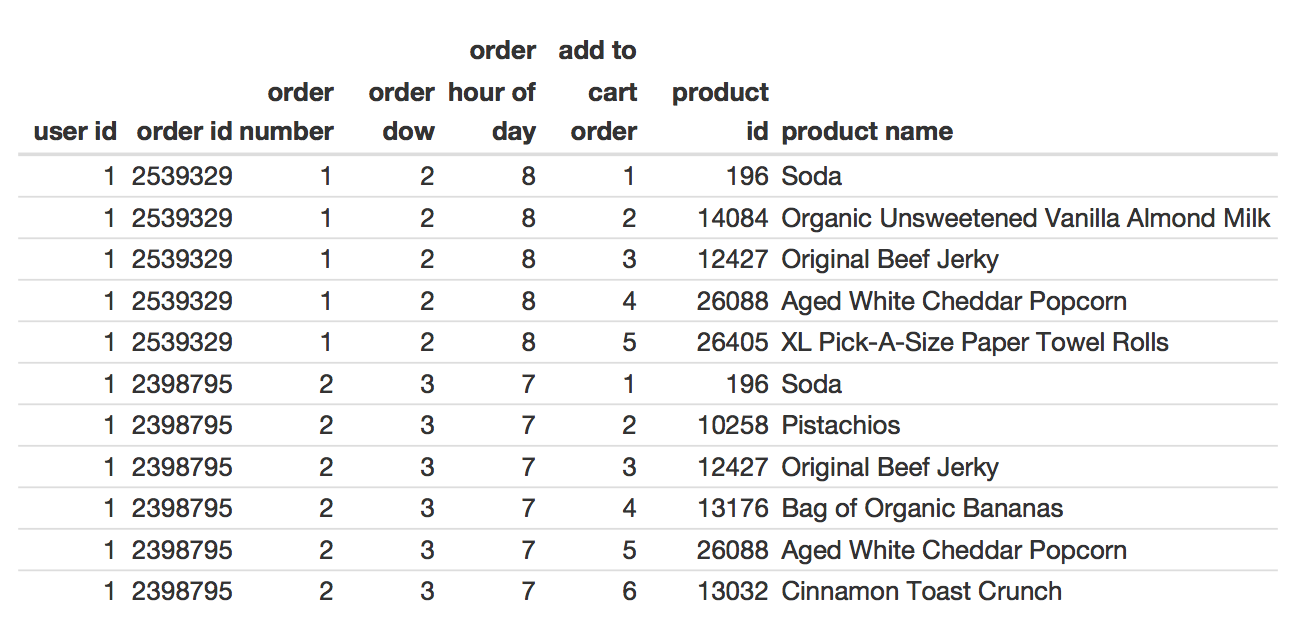
Copy

from IPython.display import display, Image

url = 'https://cdn-images-1.medium.com/max/1600/1\*vYGFQCafJtGBBX5mbl0xyw.png'

example = Image(url=url, width=600)

display(example)



Copy

!wget https://s3.amazonaws.com/instacart-datasets/instacart\_online\_grocery\_shopping\_2017\_05\_01.tar.gz

Copy

--2019-11-26 04:53:08-- https://s3.amazonaws.com/instacart-datasets/instacart\_online\_grocery\_shopping\_2017\_05\_01.tar.gz

Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.110.213

Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.110.213|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 205548478 (196M) [application/x-gzip]

Saving to: ‘instacart\_online\_grocery\_shopping\_2017\_05\_01.tar.gz.2’

instacart\_online\_gr 100%[===================>] 196.03M 16.3MB/s in 13s

2019-11-26 04:53:22 (14.8 MB/s) - ‘instacart\_online\_grocery\_shopping\_2017\_05\_01.tar.gz.2’ saved [205548478/205548478]

Copy

!tar --gunzip --extract --verbose --file=instacart\_online\_grocery\_shopping\_2017\_05\_01.tar.gz

Copy

instacart\_2017\_05\_01/

instacart\_2017\_05\_01/.\_aisles.csv

instacart\_2017\_05\_01/aisles.csv

instacart\_2017\_05\_01/.\_departments.csv

instacart\_2017\_05\_01/departments.csv

instacart\_2017\_05\_01/.\_order\_products\_\_prior.csv

instacart\_2017\_05\_01/order\_products\_\_prior.csv

instacart\_2017\_05\_01/.\_order\_products\_\_train.csv

instacart\_2017\_05\_01/order\_products\_\_train.csv

instacart\_2017\_05\_01/.\_orders.csv

instacart\_2017\_05\_01/orders.csv

instacart\_2017\_05\_01/.\_products.csv

instacart\_2017\_05\_01/products.csv

Copy

%cd instacart\_2017\_05\_01

Copy

/content/instacart\_2017\_05\_01

Copy

!ls -lh \*.csv

Copy

-rw-r--r-- 1 502 staff 2.6K May 2 2017 aisles.csv

-rw-r--r-- 1 502 staff 270 May 2 2017 departments.csv

-rw-r--r-- 1 502 staff 551M May 2 2017 order\_products\_\_prior.csv

-rw-r--r-- 1 502 staff 24M May 2 2017 order\_products\_\_train.csv

-rw-r--r-- 1 502 staff 104M May 2 2017 orders.csv

-rw-r--r-- 1 502 staff 2.1M May 2 2017 products.csv

### aisles

We don’t need anything from aisles.csv

Copy

aisles = pd.read\_csv('aisles.csv')

print(aisles.shape)

aisles.head()

Copy

(134, 2)

|  | **aisle\_id** | **aisle** |
| --- | --- | --- |
| **0** | 1 | prepared soups salads |
| **1** | 2 | specialty cheeses |
| **2** | 3 | energy granola bars |
| **3** | 4 | instant foods |
| **4** | 5 | marinades meat preparation |

### departments

We don’t need anything from departments.csv

Copy

departments = pd.read\_csv('departments.csv')

print(departments.shape)

departments.head()

Copy

(21, 2)

|  | **department\_id** | **department** |
| --- | --- | --- |
| **0** | 1 | frozen |
| **1** | 2 | other |
| **2** | 3 | bakery |
| **3** | 4 | produce |
| **4** | 5 | alcohol |

### order\_products\_\_prior

We need:

* order id
* proudct id
* add to cart order

Everything except for ‘reordered’

Copy

order\_products\_\_prior = pd.read\_csv('order\_products\_\_prior.csv')

print(order\_products\_\_prior.shape)

order\_products\_\_prior.head()

Copy

(32434489, 4)

|  | **order\_id** | **product\_id** | **add\_to\_cart\_order** | **reordered** |
| --- | --- | --- | --- | --- |
| **0** | 2 | 33120 | 1 | 1 |
| **1** | 2 | 28985 | 2 | 1 |
| **2** | 2 | 9327 | 3 | 0 |
| **3** | 2 | 45918 | 4 | 1 |
| **4** | 2 | 30035 | 5 | 0 |

### order\_products\_\_train

We need:

* order id
* proudct id
* add to cart order

Everything except for ‘reordered’

Do you see anything similar between order\_products\_\_train and order\_products\_\_prior?

Copy

order\_products\_\_train = pd.read\_csv('order\_products\_\_train.csv')

print(order\_products\_\_train.shape)

order\_products\_\_train.head()

Copy

(1384617, 4)

|  | **order\_id** | **product\_id** | **add\_to\_cart\_order** | **reordered** |
| --- | --- | --- | --- | --- |
| **0** | 1 | 49302 | 1 | 1 |
| **1** | 1 | 11109 | 2 | 1 |
| **2** | 1 | 10246 | 3 | 0 |
| **3** | 1 | 49683 | 4 | 0 |
| **4** | 1 | 43633 | 5 | 1 |

### orders

We need:

* order id
* user id
* order number
* order dow
* order hour of day

Copy

orders = pd.read\_csv('orders.csv')

print(orders.shape)

orders.head()

Copy

(3421083, 7)

|  | **order\_id** | **user\_id** | **eval\_set** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **days\_since\_prior\_order** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN |
| **1** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 |
| **2** | 473747 | 1 | prior | 3 | 3 | 12 | 21.0 |
| **3** | 2254736 | 1 | prior | 4 | 4 | 7 | 29.0 |
| **4** | 431534 | 1 | prior | 5 | 4 | 15 | 28.0 |

### products

We need:

* product id
* product name

Copy

products = pd.read\_csv('products.csv')

print(products.shape)

products.head()

Copy

(49688, 4)

|  | **product\_id** | **product\_name** | **aisle\_id** | **department\_id** |
| --- | --- | --- | --- | --- |
| **0** | 1 | Chocolate Sandwich Cookies | 61 | 19 |
| **1** | 2 | All-Seasons Salt | 104 | 13 |
| **2** | 3 | Robust Golden Unsweetened Oolong Tea | 94 | 7 |
| **3** | 4 | Smart Ones Classic Favorites Mini Rigatoni Wit... | 38 | 1 |
| **4** | 5 | Green Chile Anytime Sauce | 5 | 13 |

### Concatenate order\_products\_\_prior and order\_products\_\_train

Copy

order\_products\_\_prior.shape

Copy

(32434489, 4)

Copy

order\_products\_\_train.shape

Copy

(1384617, 4)

Copy

order\_products = pd.concat([order\_products\_\_prior, order\_products\_\_train])

print(order\_products.shape)

order\_products.head()

Copy

(33819106, 4)

|  | **order\_id** | **product\_id** | **add\_to\_cart\_order** | **reordered** |
| --- | --- | --- | --- | --- |
| **0** | 2 | 33120 | 1 | 1 |
| **1** | 2 | 28985 | 2 | 1 |
| **2** | 2 | 9327 | 3 | 0 |
| **3** | 2 | 45918 | 4 | 1 |
| **4** | 2 | 30035 | 5 | 0 |

##### Challenge

Concatenating dataframes means to stick two dataframes together either by rows or by columns. The default behavior of pd.concat() is to take the rows of one dataframe and add them to the rows of another dataframe. If we pass the argument axis=1 then we will be adding the columns of one dataframe to the columns of another dataframe.

Concatenating dataframes is most useful when the columns are the same between two dataframes or when we have matching row indices between two dataframes.

Be ready to use this method to combine dataframes together during your assignment.

#### Learn to merge data with pandas

Whenever two dataframes hold different kinds of information about the same item but share a common feature/column we can use the linking feature as an index and merge two dataframes together based on the common column. To accomplish this we can use either the .merge() or .join() operations.

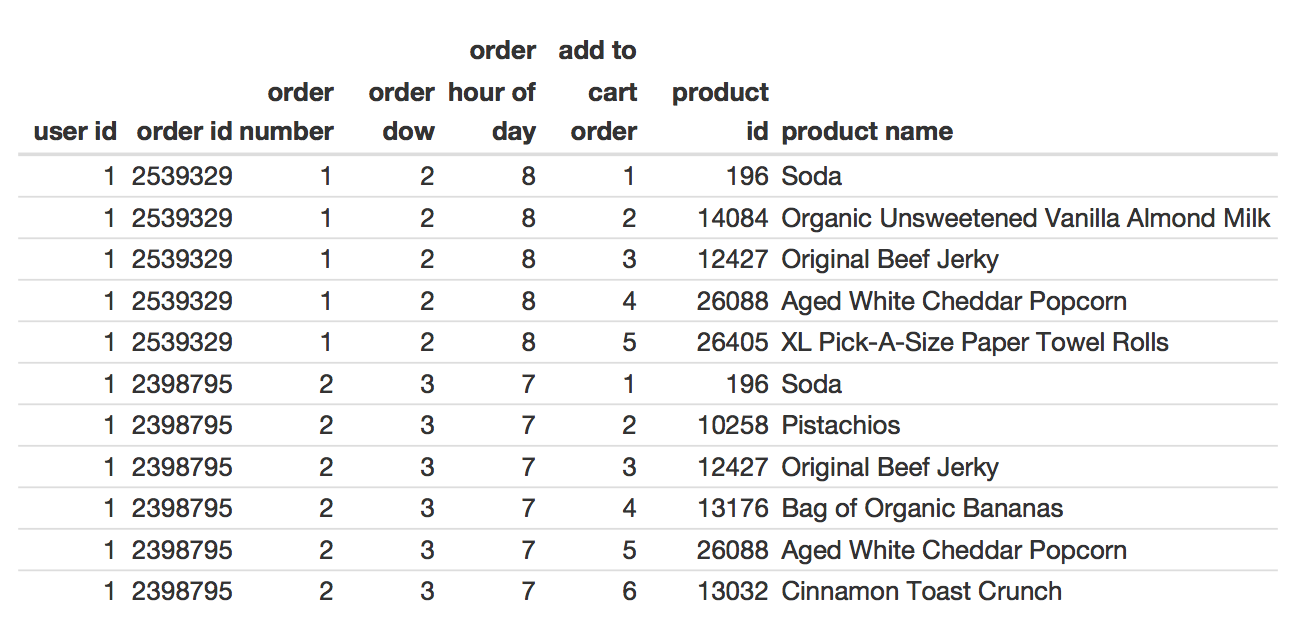
Review this section in the [Pandas Cheat Sheet](https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf): Combine Data Sets: Standard Joins

Read Python Data Science Handbook [Chapter 3.7](https://jakevdp.github.io/PythonDataScienceHandbook/03.07-merge-and-join.html), Combining Datasets: Merge and Join

##### Overview

Copy

display(example)



Before we can continue we need to understand where the data in the above table is coming from and what why specific pieces of data are held in the specific dataframes.

Each of these CSVs has a specific unit of observation (row). The columns that we see included in each CSV were selected purposefully. For example, everything each row of the orders dataframe is a specific and unique order -telling us who made the order, and when they made it. Every row in the products dataframe tells us about a specific and unique product that thestore offers. And everything in the order\_products dataframe tells us about how products are associated with specific orders -including when the product was added to the shopping cart.

### The Orders Dataframe

Holds information about specific orders, things like who placed the order, what

* user\_id
* order\_id
* order\_number
* order\_dow
* order\_hour\_of\_day

### The Products Dataframe

Holds information about individual products.

* product\_id
* product\_name

### The Order\_Products Dataframe

Tells us how products are associated with specific orders since an order is a group of products.

* order\_id
* product\_id
* add\_to\_cart\_order

As we look at the table that we’re trying to recreate, we notice that we’re not looking at specific orders or products, but at a specific **USER**. We’re looking at the first two orders for a specific user and the products associated with those orders, so we’ll need to combine dataframes to get all of this data together into a single table.

**The key to combining all of this information is that we need values that exist in both datasets that we can use to match up rows and combine dataframes.**

##### Follow Along

We have two dataframes, so we’re going to need to merge our data twice. As we approach merging datasets together we will take the following approach.

1) Identify which to dataframes we would like to combine.

2) Find columns that are common between both dataframes that we can use to match up information.

3) Slim down both of our dataframes so that they only relevant data before we merge.

4) Merge the dataframes.

##### First Merge

1) Combine orders and order\_products

2) We will use the order\_id column to match information between the two datasets

3) Lets slim down our dataframes to only the information that we need. We do this because the merge process is complex. Why would we merge millions of rows together if we know that we’re only going to need 11 rows when we’re done

What specific conditions could we use to slim down the orders dataframe?

user\_id == 1 and order\_id <=2

or

order\_id == 2539329 and order\_id == 2398795

Copy

### An example of dataframe filtering

### Create a condition

condition = (orders['order\_id'] <= 5)

### Pass that condition into the square brackets

### that we use to access portions of a dataframe

### only the rows where that condition evaluates to \*TRUE\*

### will be retained in the dataframe

orders\_subset = orders[condition]

### Look at the subsetted dataframe

orders\_subset.head()

|  | **order\_id** | **user\_id** | **eval\_set** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **days\_since\_prior\_order** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1868044** | 1 | 112108 | train | 4 | 4 | 10 | 9.0 |
| **2593373** | 5 | 156122 | prior | 42 | 6 | 16 | 9.0 |
| **2958007** | 4 | 178520 | prior | 36 | 1 | 9 | 7.0 |
| **3355525** | 2 | 202279 | prior | 3 | 5 | 9 | 8.0 |
| **3417191** | 3 | 205970 | prior | 16 | 5 | 17 | 12.0 |

Copy

condition = ((orders['user\_id'] == 1) & (orders['order\_number'] <= 2))

orders\_subset = orders[condition]

print(orders\_subset.shape)

orders\_subset.head()

Copy

(2, 7)

|  | **order\_id** | **user\_id** | **eval\_set** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **days\_since\_prior\_order** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN |
| **1** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 |

Copy

### We don't necessarily have to save our condition to the variable "condition"

### we can pass the condition into the square brackest directly

### I just wanted to be clear what was happening inside of the square brackets

orders\_subset = orders[((orders['user\_id'] == 1) & (orders['order\_number'] <= 2))]

print(orders\_subset.shape)

orders\_subset.head()

Copy

(2, 7)

|  | **order\_id** | **user\_id** | **eval\_set** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **days\_since\_prior\_order** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN |
| **1** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 |

Remember there are multiple ways that we could have filtered this dataframe. We also could have done it by specific order\_ids

Copy

orders\_subset = orders[((orders['order\_id'] == 2539329) | (orders['order\_id'] == 2398795))]

print(orders\_subset.shape)

orders\_subset.head(15)

Copy

(2, 7)

|  | **order\_id** | **user\_id** | **eval\_set** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **days\_since\_prior\_order** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN |
| **1** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 |

Now we’ll filter down the order\_products dataframe

What conditions could we use for subsetting that table?

We can use order\_id again.

Copy

order\_products\_subset = order\_products[((order\_products['order\_id'] == 2539329) | (order\_products['order\_id'] == 2398795))]

print(order\_products\_subset.shape)

order\_products\_subset.head(11)

Copy

(11, 4)

|  | **order\_id** | **product\_id** | **add\_to\_cart\_order** | **reordered** |
| --- | --- | --- | --- | --- |
| **22742744** | 2398795 | 196 | 1 | 1 |
| **22742745** | 2398795 | 10258 | 2 | 0 |
| **22742746** | 2398795 | 12427 | 3 | 1 |
| **22742747** | 2398795 | 13176 | 4 | 0 |
| **22742748** | 2398795 | 26088 | 5 | 1 |
| **22742749** | 2398795 | 13032 | 6 | 0 |
| **24076664** | 2539329 | 196 | 1 | 0 |
| **24076665** | 2539329 | 14084 | 2 | 0 |
| **24076666** | 2539329 | 12427 | 3 | 0 |
| **24076667** | 2539329 | 26088 | 4 | 0 |
| **24076668** | 2539329 | 26405 | 5 | 0 |

4) Now we’re ready to merge these two tables together.

Copy

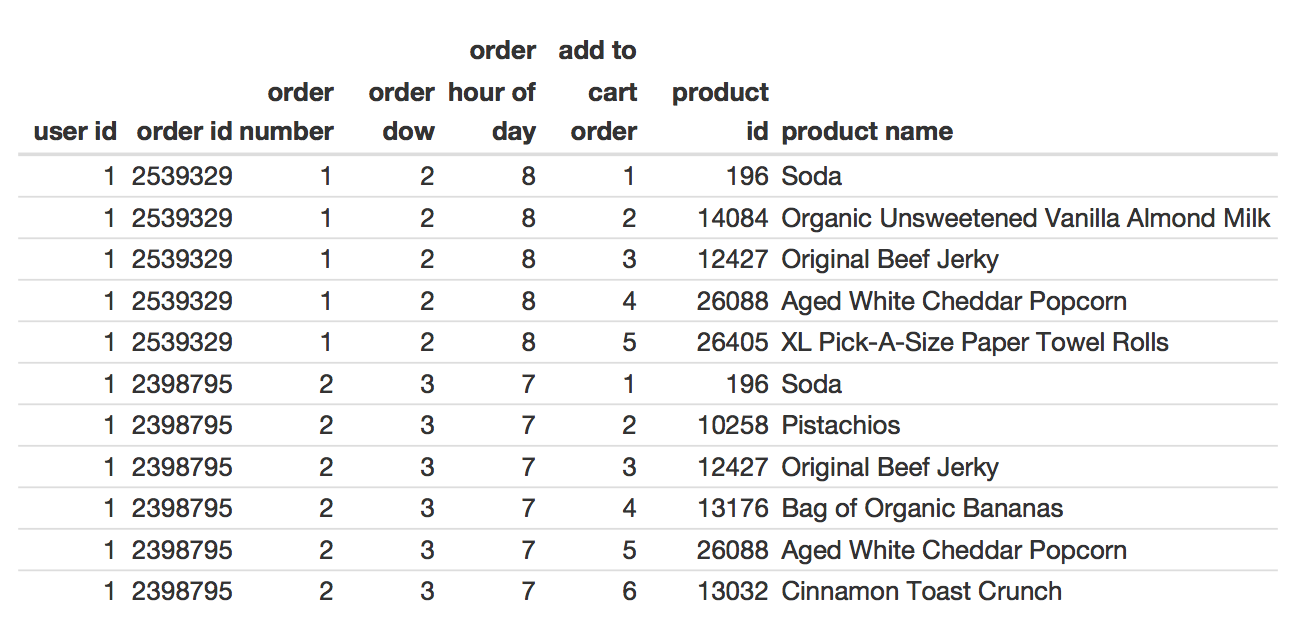
orders\_and\_products = pd.merge(orders\_subset, order\_products\_subset, on='order\_id', how='inner')

orders\_and\_products.head(11)

|  | **order\_id** | **user\_id** | **eval\_set** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **days\_since\_prior\_order** | **product\_id** | **add\_to\_cart\_order** | **reordered** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN | 196 | 1 | 0 |
| **1** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN | 14084 | 2 | 0 |
| **2** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN | 12427 | 3 | 0 |
| **3** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN | 26088 | 4 | 0 |
| **4** | 2539329 | 1 | prior | 1 | 2 | 8 | NaN | 26405 | 5 | 0 |
| **5** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 | 196 | 1 | 1 |
| **6** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 | 10258 | 2 | 0 |
| **7** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 | 12427 | 3 | 1 |
| **8** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 | 13176 | 4 | 0 |
| **9** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 | 26088 | 5 | 1 |
| **10** | 2398795 | 1 | prior | 2 | 3 | 7 | 15.0 | 13032 | 6 | 0 |

Copy

display(example)



Copy

### Remove columns that we don't need

orders\_and\_products = orders\_and\_products.drop(['eval\_set', 'reordered', 'days\_since\_prior\_order'], axis='columns')

orders\_and\_products.head(11)

|  | **order\_id** | **user\_id** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **product\_id** | **add\_to\_cart\_order** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | 1 | 2 | 8 | 196 | 1 |
| **1** | 2539329 | 1 | 1 | 2 | 8 | 14084 | 2 |
| **2** | 2539329 | 1 | 1 | 2 | 8 | 12427 | 3 |
| **3** | 2539329 | 1 | 1 | 2 | 8 | 26088 | 4 |
| **4** | 2539329 | 1 | 1 | 2 | 8 | 26405 | 5 |
| **5** | 2398795 | 1 | 2 | 3 | 7 | 196 | 1 |
| **6** | 2398795 | 1 | 2 | 3 | 7 | 10258 | 2 |
| **7** | 2398795 | 1 | 2 | 3 | 7 | 12427 | 3 |
| **8** | 2398795 | 1 | 2 | 3 | 7 | 13176 | 4 |
| **9** | 2398795 | 1 | 2 | 3 | 7 | 26088 | 5 |
| **10** | 2398795 | 1 | 2 | 3 | 7 | 13032 | 6 |

Okay, we’re looking pretty good, we’re missing one more column product\_name so we’re going to need to merge one more time

1) merge orders\_and\_products with products

2) Use product\_id as our identifier in both tables

3) We need to slim down the products dataframe

Copy

orders\_and\_products['product\_id']

Copy

0 196

1 14084

2 12427

3 26088

4 26405

5 196

6 10258

7 12427

8 13176

9 26088

10 13032

Name: product\_id, dtype: int64

Copy

orders\_and\_products['product\_id'].isin([196, 26088])

Copy

0 True

1 False

2 False

3 True

4 False

5 True

6 False

7 False

8 False

9 True

10 False

Name: product\_id, dtype: bool

Copy

condition = products['product\_id'].isin(orders\_and\_products['product\_id'])

products\_subset = products[condition]

products\_subset

|  | **product\_id** | **product\_name** | **aisle\_id** | **department\_id** |
| --- | --- | --- | --- | --- |
| **195** | 196 | Soda | 77 | 7 |
| **10257** | 10258 | Pistachios | 117 | 19 |
| **12426** | 12427 | Original Beef Jerky | 23 | 19 |
| **13031** | 13032 | Cinnamon Toast Crunch | 121 | 14 |
| **13175** | 13176 | Bag of Organic Bananas | 24 | 4 |
| **14083** | 14084 | Organic Unsweetened Vanilla Almond Milk | 91 | 16 |
| **26087** | 26088 | Aged White Cheddar Popcorn | 23 | 19 |
| **26404** | 26405 | XL Pick-A-Size Paper Towel Rolls | 54 | 17 |

Copy

final = pd.merge(orders\_and\_products, products\_subset, on='product\_id', how='inner')

final

|  | **order\_id** | **user\_id** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **product\_id** | **add\_to\_cart\_order** | **product\_name** | **aisle\_id** | **department\_id** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | 1 | 2 | 8 | 196 | 1 | Soda | 77 | 7 |
| **1** | 2398795 | 1 | 2 | 3 | 7 | 196 | 1 | Soda | 77 | 7 |
| **2** | 2539329 | 1 | 1 | 2 | 8 | 14084 | 2 | Organic Unsweetened Vanilla Almond Milk | 91 | 16 |
| **3** | 2539329 | 1 | 1 | 2 | 8 | 12427 | 3 | Original Beef Jerky | 23 | 19 |
| **4** | 2398795 | 1 | 2 | 3 | 7 | 12427 | 3 | Original Beef Jerky | 23 | 19 |
| **5** | 2539329 | 1 | 1 | 2 | 8 | 26088 | 4 | Aged White Cheddar Popcorn | 23 | 19 |
| **6** | 2398795 | 1 | 2 | 3 | 7 | 26088 | 5 | Aged White Cheddar Popcorn | 23 | 19 |
| **7** | 2539329 | 1 | 1 | 2 | 8 | 26405 | 5 | XL Pick-A-Size Paper Towel Rolls | 54 | 17 |
| **8** | 2398795 | 1 | 2 | 3 | 7 | 10258 | 2 | Pistachios | 117 | 19 |
| **9** | 2398795 | 1 | 2 | 3 | 7 | 13176 | 4 | Bag of Organic Bananas | 24 | 4 |
| **10** | 2398795 | 1 | 2 | 3 | 7 | 13032 | 6 | Cinnamon Toast Crunch | 121 | 14 |

Copy

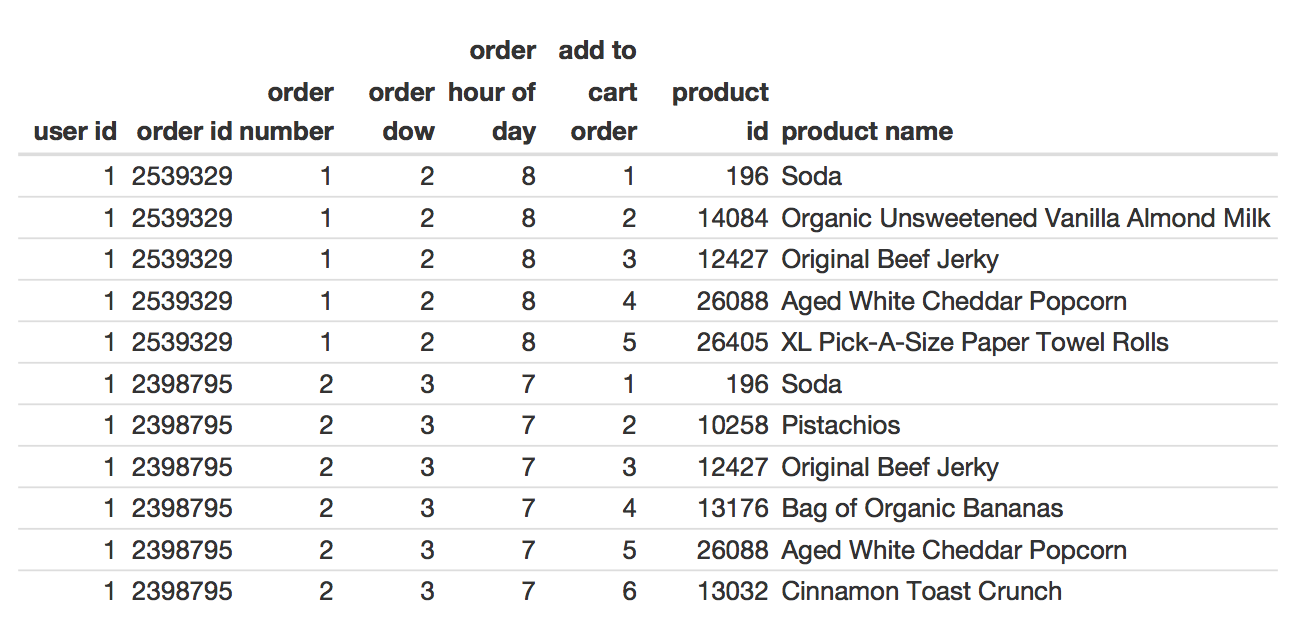
final = final.drop(['aisle\_id', 'department\_id'], axis=1)

final

|  | **order\_id** | **user\_id** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **product\_id** | **add\_to\_cart\_order** | **product\_name** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | 1 | 2 | 8 | 196 | 1 | Soda |
| **1** | 2398795 | 1 | 2 | 3 | 7 | 196 | 1 | Soda |
| **2** | 2539329 | 1 | 1 | 2 | 8 | 14084 | 2 | Organic Unsweetened Vanilla Almond Milk |
| **3** | 2539329 | 1 | 1 | 2 | 8 | 12427 | 3 | Original Beef Jerky |
| **4** | 2398795 | 1 | 2 | 3 | 7 | 12427 | 3 | Original Beef Jerky |
| **5** | 2539329 | 1 | 1 | 2 | 8 | 26088 | 4 | Aged White Cheddar Popcorn |
| **6** | 2398795 | 1 | 2 | 3 | 7 | 26088 | 5 | Aged White Cheddar Popcorn |
| **7** | 2539329 | 1 | 1 | 2 | 8 | 26405 | 5 | XL Pick-A-Size Paper Towel Rolls |
| **8** | 2398795 | 1 | 2 | 3 | 7 | 10258 | 2 | Pistachios |
| **9** | 2398795 | 1 | 2 | 3 | 7 | 13176 | 4 | Bag of Organic Bananas |
| **10** | 2398795 | 1 | 2 | 3 | 7 | 13032 | 6 | Cinnamon Toast Crunch |

Copy

display(example)



##### Some nitpicky cleanup:

Copy

### sort rows

final = final.sort\_values(by=['order\_number', 'add\_to\_cart\_order'])

final

|  | **order\_id** | **user\_id** | **order\_number** | **order\_dow** | **order\_hour\_of\_day** | **product\_id** | **add\_to\_cart\_order** | **product\_name** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2539329 | 1 | 1 | 2 | 8 | 196 | 1 | Soda |
| **2** | 2539329 | 1 | 1 | 2 | 8 | 14084 | 2 | Organic Unsweetened Vanilla Almond Milk |
| **3** | 2539329 | 1 | 1 | 2 | 8 | 12427 | 3 | Original Beef Jerky |
| **5** | 2539329 | 1 | 1 | 2 | 8 | 26088 | 4 | Aged White Cheddar Popcorn |
| **7** | 2539329 | 1 | 1 | 2 | 8 | 26405 | 5 | XL Pick-A-Size Paper Towel Rolls |
| **1** | 2398795 | 1 | 2 | 3 | 7 | 196 | 1 | Soda |
| **8** | 2398795 | 1 | 2 | 3 | 7 | 10258 | 2 | Pistachios |
| **4** | 2398795 | 1 | 2 | 3 | 7 | 12427 | 3 | Original Beef Jerky |
| **9** | 2398795 | 1 | 2 | 3 | 7 | 13176 | 4 | Bag of Organic Bananas |
| **6** | 2398795 | 1 | 2 | 3 | 7 | 26088 | 5 | Aged White Cheddar Popcorn |
| **10** | 2398795 | 1 | 2 | 3 | 7 | 13032 | 6 | Cinnamon Toast Crunch |

Copy

### reorder columns

final = final[['user\_id', 'order\_id', 'order\_number', 'order\_dow', 'order\_hour\_of\_day', 'add\_to\_cart\_order', 'product\_id', 'product\_name']]

Copy

### remove underscores from column headers

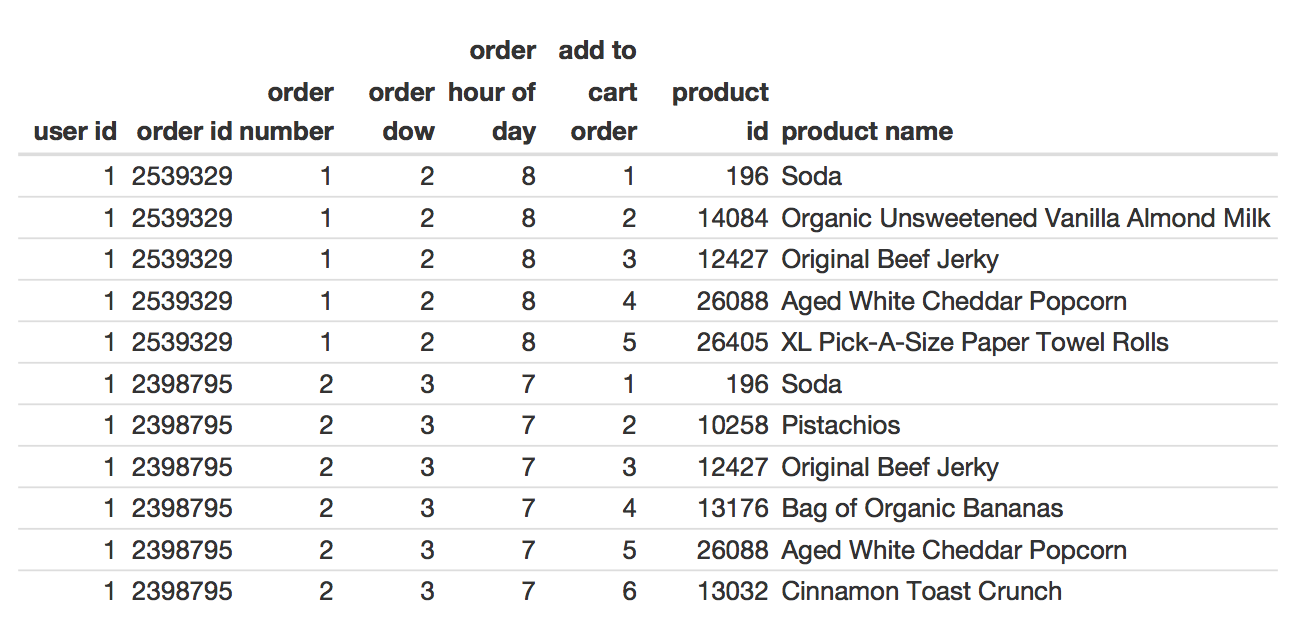
final.columns = [column.replace('\_', ' ') for column in final]

final

|  | **user id** | **order id** | **order number** | **order dow** | **order hour of day** | **add to cart order** | **product id** | **product name** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 2539329 | 1 | 2 | 8 | 1 | 196 | Soda |
| **2** | 1 | 2539329 | 1 | 2 | 8 | 2 | 14084 | Organic Unsweetened Vanilla Almond Milk |
| **3** | 1 | 2539329 | 1 | 2 | 8 | 3 | 12427 | Original Beef Jerky |
| **5** | 1 | 2539329 | 1 | 2 | 8 | 4 | 26088 | Aged White Cheddar Popcorn |
| **7** | 1 | 2539329 | 1 | 2 | 8 | 5 | 26405 | XL Pick-A-Size Paper Towel Rolls |
| **1** | 1 | 2398795 | 2 | 3 | 7 | 1 | 196 | Soda |
| **8** | 1 | 2398795 | 2 | 3 | 7 | 2 | 10258 | Pistachios |
| **4** | 1 | 2398795 | 2 | 3 | 7 | 3 | 12427 | Original Beef Jerky |
| **9** | 1 | 2398795 | 2 | 3 | 7 | 4 | 13176 | Bag of Organic Bananas |
| **6** | 1 | 2398795 | 2 | 3 | 7 | 5 | 26088 | Aged White Cheddar Popcorn |
| **10** | 1 | 2398795 | 2 | 3 | 7 | 6 | 13032 | Cinnamon Toast Crunch |

Copy

display(example)



##### Challenge

Review this Chis Albon documentation about [concatenating dataframes by row and by column](https://chrisalbon.com/python/data_wrangling/pandas_join_merge_dataframe/) and then be ready to master this function and practice using different how parameters on your assignment.

#### Learn to understand tidy data formatting

### Why reshape data?

Because some libraries prefer data in different formats.

For example, the Seaborn data visualization library prefers data in “Tidy” format often (but not always).

“[Seaborn will be most powerful when your datasets have a particular organization.](https://seaborn.pydata.org/introduction.html#organizing-datasets) This format ia alternately called “long-form” or “tidy” data and is described in detail by Hadley Wickham. The rules can be simply stated:

* Each variable is a column
* Each observation is a row

A helpful mindset for determining whether your data are tidy is to think backwards from the plot you want to draw. From this perspective, a “variable” is something that will be assigned a role in the plot.”

#### Data science is often about putting square pegs in round holes

Here’s an inspiring [video clip from Apollo 13](https://www.youtube.com/watch?v=ry55--J4_VQ): “Invent a way to put a square peg in a round hole.” It’s a good metaphor for data wrangling!

### Hadley Wickham’s Examples

From his paper, [Tidy Data](http://vita.had.co.nz/papers/tidy-data.html)

Copy

%matplotlib inline

import pandas as pd

import numpy as np

import seaborn as sns

table1 = pd.DataFrame(

[[np.nan, 2],

[16, 11],

[3, 1]],

index=['John Smith', 'Jane Doe', 'Mary Johnson'],

columns=['treatmenta', 'treatmentb'])

“Table 1 provides some data about an imaginary experiment in a format commonly seen in the wild.

The table has two columns and three rows, and both rows and columns are labelled.”

Copy

table1

|  | **treatmenta** | **treatmentb** |
| --- | --- | --- |
| **John Smith** | NaN | 2 |
| **Jane Doe** | 16.0 | 11 |
| **Mary Johnson** | 3.0 | 1 |

“There are many ways to structure the same underlying data.

Table 2 shows the same data as Table 1, but the rows and columns have been transposed. The data is the same, but the layout is different.”

Copy

table2 = table1.T

table2

|  | **John Smith** | **Jane Doe** | **Mary Johnson** |
| --- | --- | --- | --- |
| **treatmenta** | NaN | 16.0 | 3.0 |
| **treatmentb** | 2.0 | 11.0 | 1.0 |

“Table 3 reorganises Table 1 to make the values, variables and obserations more clear.

Table 3 is the tidy version of Table 1. Each row represents an observation, the result of one treatment on one person, and each column is a variable.”

| **name** | **trt** | **result** |
| --- | --- | --- |
| John Smith | a | - |
| Jane Doe | a | 16 |
| Mary Johnson | a | 3 |
| John Smith | b | 2 |
| Jane Doe | b | 11 |
| Mary Johnson | b | 1 |

Review this section in the [Pandas Cheat Sheet](https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf): Reshaping Data

Read these sections in the Python Data Science Handbook

* [Chapter 3.8](https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html), Aggregation and Grouping
* [Chapter 3.9](https://jakevdp.github.io/PythonDataScienceHandbook/03.09-pivot-tables.html), Pivot Tables

##### Overview

Review this section in the [Pandas Cheat Sheet](https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf): Tidy Data

Read about [Tidy Data](https://en.wikipedia.org/wiki/Tidy_data) on Wikipedia

##### Follow Along

##### Table 1 –> Tidy

We can use the pandas melt function to reshape Table 1 into Tidy format.

Copy

### Take the row index, and add it as a new column

table1 = table1.reset\_index()

table1

|  | **index** | **treatmenta** | **treatmentb** |
| --- | --- | --- | --- |
| **0** | John Smith | NaN | 2 |
| **1** | Jane Doe | 16.0 | 11 |
| **2** | Mary Johnson | 3.0 | 1 |

Copy

### What is the unique identifier for each row

### Where is the data at that I want to be in my single "tidy" column

tidy1 = table1.melt(id\_vars='index', value\_vars=['treatmenta', 'treatmentb'])

tidy1

|  | **index** | **variable** | **value** |
| --- | --- | --- | --- |
| **0** | John Smith | treatmenta | NaN |
| **1** | Jane Doe | treatmenta | 16.0 |
| **2** | Mary Johnson | treatmenta | 3.0 |
| **3** | John Smith | treatmentb | 2.0 |
| **4** | Jane Doe | treatmentb | 11.0 |
| **5** | Mary Johnson | treatmentb | 1.0 |

Copy

tidy1 = tidy1.rename(columns={

'index': 'name',

'variable': 'trt',

'value': 'result'

})

tidy1

|  | **name** | **trt** | **result** |
| --- | --- | --- | --- |
| **0** | John Smith | treatmenta | NaN |
| **1** | Jane Doe | treatmenta | 16.0 |
| **2** | Mary Johnson | treatmenta | 3.0 |
| **3** | John Smith | treatmentb | 2.0 |
| **4** | Jane Doe | treatmentb | 11.0 |
| **5** | Mary Johnson | treatmentb | 1.0 |

Copy

tidy1.trt = tidy1.trt.str.replace('treatment', '')

tidy1

|  | **name** | **trt** | **result** |
| --- | --- | --- | --- |
| **0** | John Smith | a | NaN |
| **1** | Jane Doe | a | 16.0 |
| **2** | Mary Johnson | a | 3.0 |
| **3** | John Smith | b | 2.0 |
| **4** | Jane Doe | b | 11.0 |
| **5** | Mary Johnson | b | 1.0 |

##### Tidy –> Table 1

The pivot\_table function is the inverse of melt.

Copy

### index: unique identifier

### columns: What do you want to differentiate the columns in wide format

### values: Where are the numbers at - go in the middle of the wide dataframe

wide = tidy1.pivot\_table(index='name', columns='trt', values='result')

wide

| **trt** | **a** | **b** |
| --- | --- | --- |
| **name** |  |  |
| **Jane Doe** | 16.0 | 11.0 |
| **John Smith** | NaN | 2.0 |
| **Mary Johnson** | 3.0 | 1.0 |

##### Challenge

On your assignment, be prepared to take table2 (the transpose of table1) and reshape it to be in tidy data format using .melt() and then put it back in “wide format” using .pivot\_table()

#### Learn to melt and pivot data with pandas

Tidy data format can be particularly useful with certain plotting libraries like Seaborn for example. Lets practice reshaping our data and show how this can be extremely useful in preparing our data for plotting.

Remember that tidy data format means:

* Each variable is a column
* Each observation is a row

A helpful mindset for determining whether your data are tidy is to think backwards from the plot you want to draw. From this perspective, a “variable” is something that will be assigned a role in the plot.” When plotting, this typically means that the values that we’re most interested in and that represent the same thing will all be in a single column. You’ll see that in the different examples that we show. The important data will be in a single column.

Copy

### Look at some of the awesome out-of-the-box seaborn functionality:

sns.catplot(x='trt', y='result', col='name',

kind='bar', data=tidy1, height=2);

##### Overview

Now with Instacart Data. We’re going to try and reproduce a small part of this visualization:

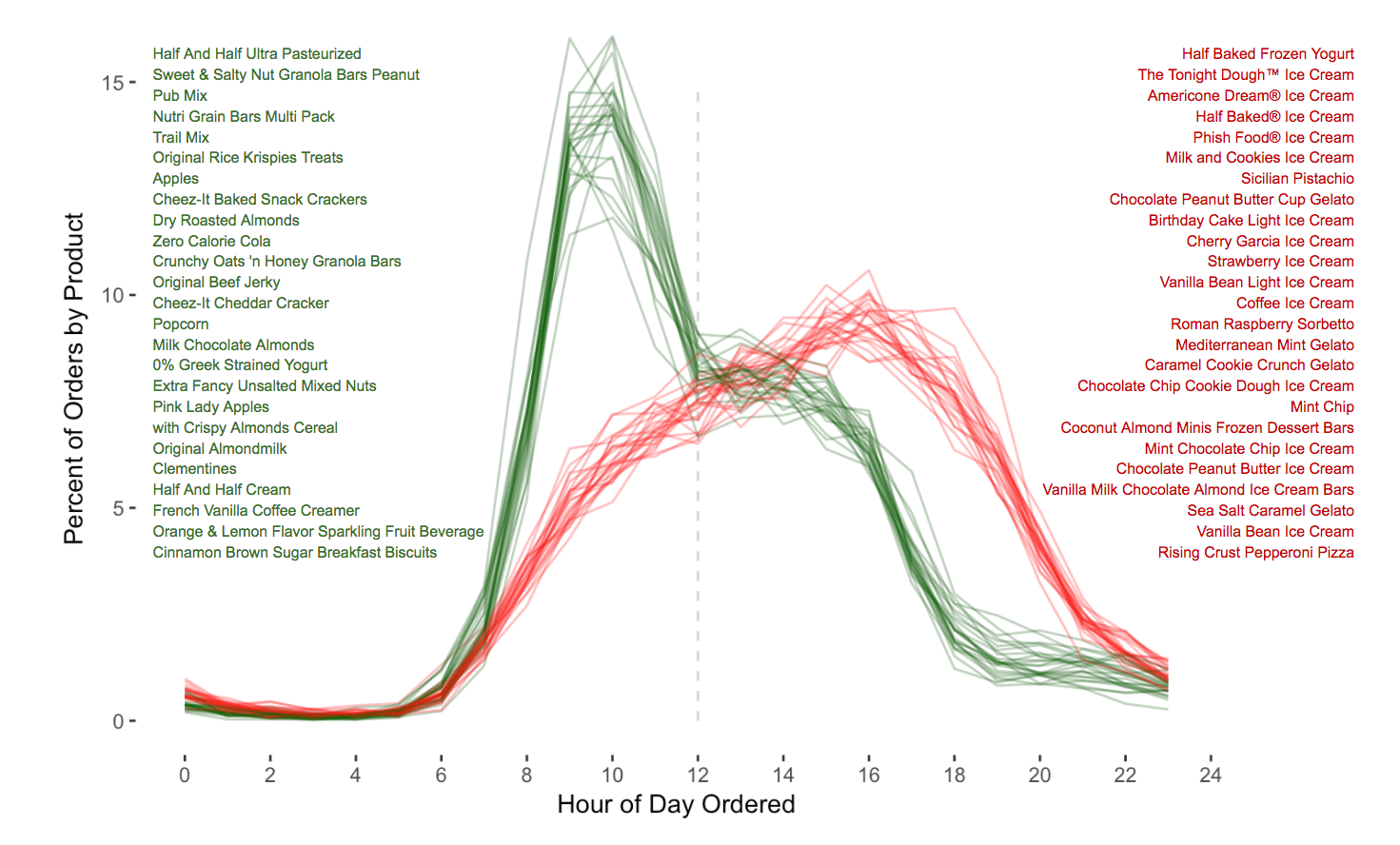
Copy

from IPython.display import display, Image

url = 'https://cdn-images-1.medium.com/max/1600/1\*wKfV6OV-\_1Ipwrl7AjjSuw.png'

example = Image(url=url, width=600)

display(example)



Instead of a plot with 50 products, we’ll just do two — the first products from each list

* Half And Half Ultra Pasteurized
* Half Baked Frozen Yogurt

So, given a product\_name we need to calculate its order\_hour\_of\_day pattern.

##### Follow Along

Copy

products = pd.read\_csv('products.csv')

order\_products = pd.concat([pd.read\_csv('order\_products\_\_prior.csv'),

pd.read\_csv('order\_products\_\_train.csv')])

orders = pd.read\_csv('orders.csv')

### Subset and Merge

One challenge of performing a merge on this data is that the products and orders datasets do not have any common columns that we can merge on. Due to this we will have to use the order\_products dataset to provide the columns that we will use to perform the merge.

Here’s the two products that we want to work with.

Copy

product\_names = ['Half Baked Frozen Yogurt', 'Half And Half Ultra Pasteurized']

Lets remind ourselves of what columns we have to work with:

Copy

products.columns.to\_list()

Copy

['product\_id', 'product\_name', 'aisle\_id', 'department\_id']

Copy

orders.columns.to\_list()

Copy

['order\_id',

'user\_id',

'eval\_set',

'order\_number',

'order\_dow',

'order\_hour\_of\_day',

'days\_since\_prior\_order']

Copy

order\_products.columns.to\_list()

Copy

['order\_id', 'product\_id', 'add\_to\_cart\_order', 'reordered']

This might blow your mind, but we’re going to subset the dataframes to select specific columns **and** merge them all in one go. Ready?

Copy

merged = (products[['product\_id', 'product\_name']]

.merge(order\_products[['order\_id', 'product\_id']])

.merge(orders[['order\_id', 'order\_hour\_of\_day']]))

merged.head()

|  | **product\_id** | **product\_name** | **order\_id** | **order\_hour\_of\_day** |
| --- | --- | --- | --- | --- |
| **0** | 1 | Chocolate Sandwich Cookies | 1107 | 11 |
| **1** | 769 | Sliced American Cheese | 1107 | 11 |
| **2** | 6184 | Clementines | 1107 | 11 |
| **3** | 8048 | Packaged Grape Tomatoes | 1107 | 11 |
| **4** | 9007 | Frosted Flakes | 1107 | 11 |

Ok, so we were a little bit lazy and probably should have subsetted our the rows of our dataframes before we merged them. We are going to filter after the fact. This is something that you can try out for practice. Can you figure out how to filter these dataframes **before** merging rather than after?

Copy

condition = ((merged['product\_name']=='Half Baked Frozen Yogurt') |

(merged['product\_name']=='Half And Half Ultra Pasteurized'))

merged = merged[condition]

print(merged.shape)

merged.head()

Copy

(5978, 4)

|  | **product\_id** | **product\_name** | **order\_id** | **order\_hour\_of\_day** |
| --- | --- | --- | --- | --- |
| **25086** | 30668 | Half Baked Frozen Yogurt | 595220 | 21 |
| **29409** | 30668 | Half Baked Frozen Yogurt | 3252348 | 16 |
| **33914** | 30668 | Half Baked Frozen Yogurt | 677455 | 17 |
| **34412** | 30668 | Half Baked Frozen Yogurt | 1821824 | 14 |
| **35652** | 30668 | Half Baked Frozen Yogurt | 1225489 | 17 |

Again, there are multiple effective ways to write conditions.

Copy

product\_names = ['Half Baked Frozen Yogurt', 'Half And Half Ultra Pasteurized']

condition = merged['product\_name'].isin(product\_names)

subset = merged[condition]

print(subset.shape)

subset.head()

Copy

(5978, 4)

|  | **product\_id** | **product\_name** | **order\_id** | **order\_hour\_of\_day** |
| --- | --- | --- | --- | --- |
| **25086** | 30668 | Half Baked Frozen Yogurt | 595220 | 21 |
| **29409** | 30668 | Half Baked Frozen Yogurt | 3252348 | 16 |
| **33914** | 30668 | Half Baked Frozen Yogurt | 677455 | 17 |
| **34412** | 30668 | Half Baked Frozen Yogurt | 1821824 | 14 |
| **35652** | 30668 | Half Baked Frozen Yogurt | 1225489 | 17 |

Copy

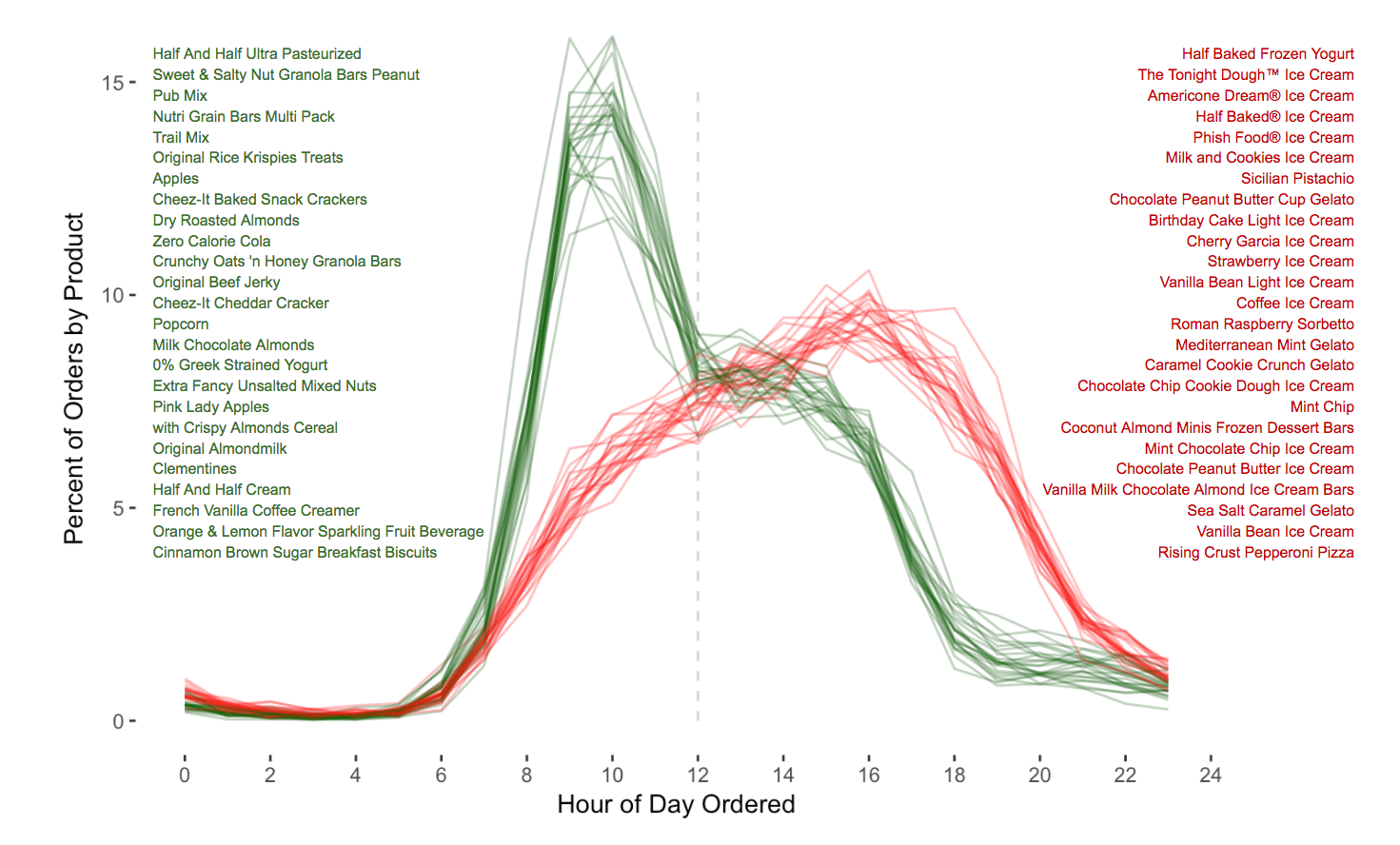
froyo = subset[subset['product\_name']=='Half Baked Frozen Yogurt']

cream = subset[subset['product\_name']=='Half And Half Ultra Pasteurized']

### 4 ways to reshape and plot

Copy

display(example)



1) The .value\_counts() approach.

Remember, that we’re trying to get the key variables (values) listed as a single column.

Copy

cream['order\_hour\_of\_day'].value\_counts(normalize=True).sort\_index()

Copy

0 0.002003

1 0.000334

2 0.000334

4 0.000334

5 0.001336

6 0.008347

7 0.031720

8 0.107846

9 0.160267

10 0.143239

11 0.097496

12 0.066778

13 0.071119

14 0.071786

15 0.074124

16 0.072788

17 0.037062

18 0.018698

19 0.009349

20 0.010684

21 0.007679

22 0.004007

23 0.002671

Name: order\_hour\_of\_day, dtype: float64

Copy

(cream['order\_hour\_of\_day']

.value\_counts(normalize=True)

.sort\_index()

.plot())

(froyo['order\_hour\_of\_day']

.value\_counts(normalize=True)

.sort\_index()

.plot());

2) Crosstab

Copy

pd.crosstab(subset['order\_hour\_of\_day'],

subset['product\_name'],

normalize='columns').plot();

3) Pivot Table

Copy

subset.pivot\_table(index='order\_hour\_of\_day',

columns='product\_name',

values='order\_id',

aggfunc=len).plot();

4) Melt

We’ve got to get it into wide format first. We’ll use a crosstab which is a specific type of pivot\_table.

Copy

table = pd.crosstab(subset['order\_hour\_of\_day'],

subset['product\_name'],

normalize=True)

table

| **product\_name** | **Half And Half Ultra Pasteurized** | **Half Baked Frozen Yogurt** |
| --- | --- | --- |
| **order\_hour\_of\_day** |  |  |
| **0** | 0.001004 | 0.002676 |
| **1** | 0.000167 | 0.001338 |
| **2** | 0.000167 | 0.001338 |
| **3** | 0.000000 | 0.000502 |
| **4** | 0.000167 | 0.000335 |
| **5** | 0.000669 | 0.001673 |
| **6** | 0.004182 | 0.003011 |
| **7** | 0.015892 | 0.009870 |
| **8** | 0.054031 | 0.014888 |
| **9** | 0.080294 | 0.021412 |
| **10** | 0.071763 | 0.029441 |
| **11** | 0.048846 | 0.033791 |
| **12** | 0.033456 | 0.032452 |
| **13** | 0.035631 | 0.039311 |
| **14** | 0.035965 | 0.042155 |
| **15** | 0.037136 | 0.047508 |
| **16** | 0.036467 | 0.046002 |
| **17** | 0.018568 | 0.042322 |
| **18** | 0.009368 | 0.042656 |
| **19** | 0.004684 | 0.033121 |
| **20** | 0.005353 | 0.024590 |
| **21** | 0.003847 | 0.014386 |
| **22** | 0.002007 | 0.008197 |
| **23** | 0.001338 | 0.006022 |

Copy

melted = table.reset\_index().melt(id\_vars='order\_hour\_of\_day').rename(columns={

'order\_hour\_of\_day': 'Hour of Day Ordered',

'product\_name': 'Product',

'value': 'Percent of Orders by Product'

})

melted

|  | **Hour of Day Ordered** | **Product** | **Percent of Orders by Product** |
| --- | --- | --- | --- |
| **0** | 0 | Half And Half Ultra Pasteurized | 0.001004 |
| **1** | 1 | Half And Half Ultra Pasteurized | 0.000167 |
| **2** | 2 | Half And Half Ultra Pasteurized | 0.000167 |
| **3** | 3 | Half And Half Ultra Pasteurized | 0.000000 |
| **4** | 4 | Half And Half Ultra Pasteurized | 0.000167 |
| **5** | 5 | Half And Half Ultra Pasteurized | 0.000669 |
| **6** | 6 | Half And Half Ultra Pasteurized | 0.004182 |
| **7** | 7 | Half And Half Ultra Pasteurized | 0.015892 |
| **8** | 8 | Half And Half Ultra Pasteurized | 0.054031 |
| **9** | 9 | Half And Half Ultra Pasteurized | 0.080294 |
| **10** | 10 | Half And Half Ultra Pasteurized | 0.071763 |
| **11** | 11 | Half And Half Ultra Pasteurized | 0.048846 |
| **12** | 12 | Half And Half Ultra Pasteurized | 0.033456 |
| **13** | 13 | Half And Half Ultra Pasteurized | 0.035631 |
| **14** | 14 | Half And Half Ultra Pasteurized | 0.035965 |
| **15** | 15 | Half And Half Ultra Pasteurized | 0.037136 |
| **16** | 16 | Half And Half Ultra Pasteurized | 0.036467 |
| **17** | 17 | Half And Half Ultra Pasteurized | 0.018568 |
| **18** | 18 | Half And Half Ultra Pasteurized | 0.009368 |
| **19** | 19 | Half And Half Ultra Pasteurized | 0.004684 |
| **20** | 20 | Half And Half Ultra Pasteurized | 0.005353 |
| **21** | 21 | Half And Half Ultra Pasteurized | 0.003847 |
| **22** | 22 | Half And Half Ultra Pasteurized | 0.002007 |
| **23** | 23 | Half And Half Ultra Pasteurized | 0.001338 |
| **24** | 0 | Half Baked Frozen Yogurt | 0.002676 |
| **25** | 1 | Half Baked Frozen Yogurt | 0.001338 |
| **26** | 2 | Half Baked Frozen Yogurt | 0.001338 |
| **27** | 3 | Half Baked Frozen Yogurt | 0.000502 |
| **28** | 4 | Half Baked Frozen Yogurt | 0.000335 |
| **29** | 5 | Half Baked Frozen Yogurt | 0.001673 |
| **30** | 6 | Half Baked Frozen Yogurt | 0.003011 |
| **31** | 7 | Half Baked Frozen Yogurt | 0.009870 |
| **32** | 8 | Half Baked Frozen Yogurt | 0.014888 |
| **33** | 9 | Half Baked Frozen Yogurt | 0.021412 |
| **34** | 10 | Half Baked Frozen Yogurt | 0.029441 |
| **35** | 11 | Half Baked Frozen Yogurt | 0.033791 |
| **36** | 12 | Half Baked Frozen Yogurt | 0.032452 |
| **37** | 13 | Half Baked Frozen Yogurt | 0.039311 |
| **38** | 14 | Half Baked Frozen Yogurt | 0.042155 |
| **39** | 15 | Half Baked Frozen Yogurt | 0.047508 |
| **40** | 16 | Half Baked Frozen Yogurt | 0.046002 |
| **41** | 17 | Half Baked Frozen Yogurt | 0.042322 |
| **42** | 18 | Half Baked Frozen Yogurt | 0.042656 |
| **43** | 19 | Half Baked Frozen Yogurt | 0.033121 |
| **44** | 20 | Half Baked Frozen Yogurt | 0.024590 |
| **45** | 21 | Half Baked Frozen Yogurt | 0.014386 |
| **46** | 22 | Half Baked Frozen Yogurt | 0.008197 |
| **47** | 23 | Half Baked Frozen Yogurt | 0.006022 |

Now, with Seaborn:

Copy

sns.relplot(x='Hour of Day Ordered',

y='Percent of Orders by Product',

hue='Product',

data=melted,

kind='line');

##### Challenge

As a stretch goal on your assignment, you can try and complete the rest of this graphic in its entirety.

## Review

### Class Recordings

You can use class recordings to help you master the material.

* [**Join and Reshape Data for DS13 w/Ryan Allred**](https://youtu.be/-TjuSfv4g20)

Join and Reshape Data for DS13 w/Ryan Allred

* [All previous recordings](https://learn.lambdaschool.com/archive/DS/module/recLMyOUPqsW4wm90)

### Demonstrate Mastery

To demonstrate mastery of this module, you need to complete and pass a code review on each of the following:

* Objective challenge:

Concatenating dataframes means to stick two dataframes together either by rows or by columns. The default behavior of pd.concat() is to take the rows of one dataframe and add them to the rows of another dataframe. If we pass the argument axis=1 then we will be adding the columns of one dataframe to the columns of another dataframe.

Concatenating dataframes is most useful when the columns are the same between two dataframes or when we have matching row indices between two dataframes.

Be ready to use this method to combine dataframes together during your assignment.

* Objective challenge:

Review this Chis Albon documentation about [concatenating dataframes by row and by column](https://chrisalbon.com/python/data_wrangling/pandas_join_merge_dataframe/) and then be ready to master this function and practice using different how parameters on your assignment.

* Objective challenge:

On your assignment, be prepared to take table2 (the transpose of table1) and reshape it to be in tidy data format using .melt() and then put it back in “wide format” using .pivot\_table()

* Objective challenge:

As a stretch goal on your assignment, you can try and complete t

# **Make Explanatory Visualizations**

**At the end of this module, you should be able to:**

* identify misleading visualizations and how to fix them.
* use appropriate terminology when referring to parts of a Matplotlib graph
* differentiate between Matplotlib syntaxes
* use Matplotlib to control basic visual aspects of their plot so as to mimic popular plotting styles (FiveThirtyEight) including: plot, plot stylesheet, title, subtitle, axis labels, axis tick marks, background colors, text annotations.

#### Pro Tip

The way to end up with a good plan is not to start with a good plan, it’s to start with some plan, and then slam that plan against reality until reality hands you a better plan.

## Prepare

Review each preclass resource before class.

## Learn

#### Learn to identify misleading visualizations and how to fix them.

We want to both recognize and avoid creating misleading visualizations. Lets make sure that we’re familiar with a few of the most common pitfalls so that we don’t make the same mistakes in our own work.

##### Overview

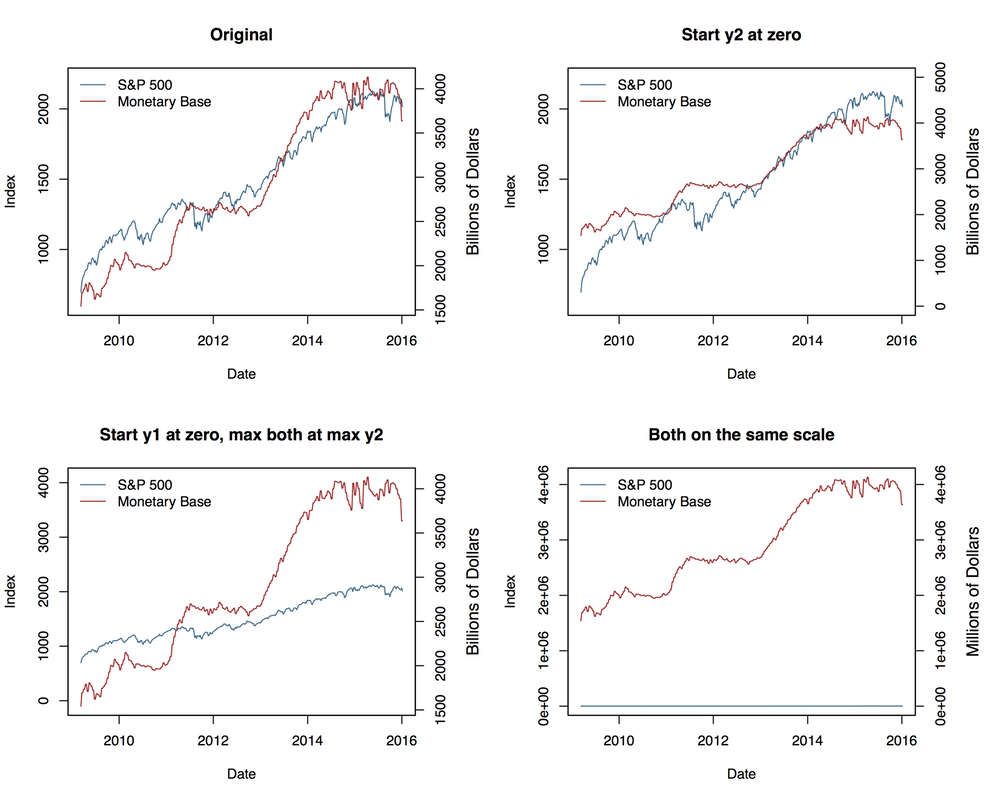
Some of the examples in this section have been taken from the following article:

[5 Ways Writers Use Misleading Graphs To Manipulate You](https://venngage.com/blog/misleading-graphs/)

But many others have come from our own internet surfing.

##### Follow Along

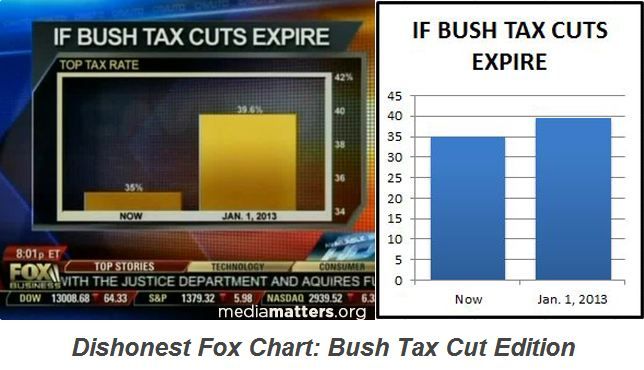
### Two y-axes



Other Examples:

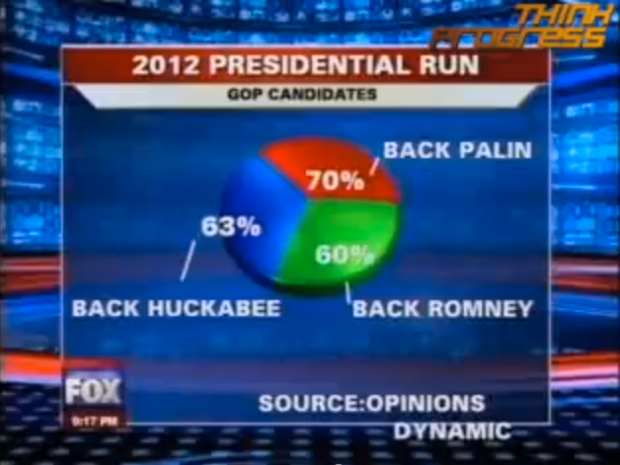
* [Spurious Correlations](https://tylervigen.com/spurious-correlations)
* <https://blog.datawrapper.de/dualaxis/>
* <https://kieranhealy.org/blog/archives/2016/01/16/two-y-axes/>
* <http://www.storytellingwithdata.com/blog/2016/2/1/be-gone-dual-y-axis>

### Y-axis doesn’t start at zero.



### Pie Charts are not very insightful

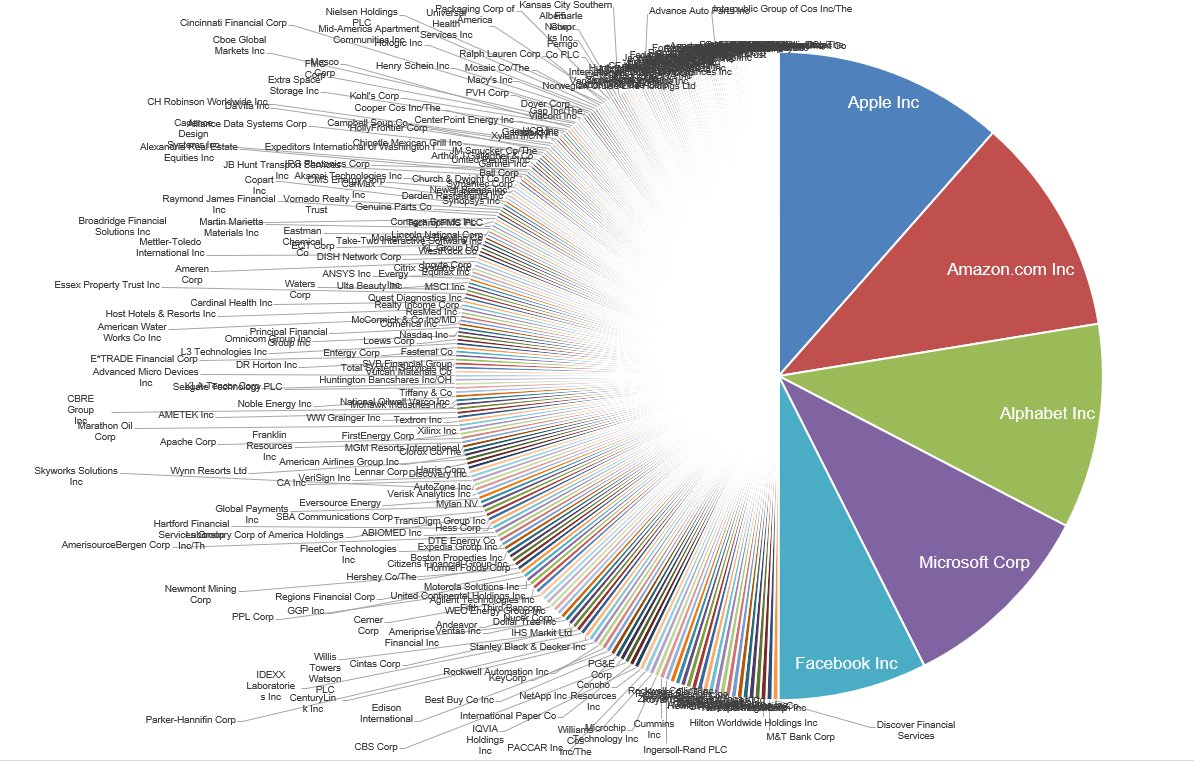
Pie charts are not a very insightful type of visualization, but they’re just plain bad if you use them incorrectly.



### Pie charts that omit data are extra bad

* A guy makes a misleading chart that goes viral

What does this chart imply at first glance? You don’t want your user to have to do a lot of work in order to be able to interpret you graph correctly. You want that first-glance conclusions to be the correct ones.



<https://twitter.com/michaelbatnick/status/1019680856837849090?lang=en>

* It gets picked up by overworked journalists (assuming incompetency before malice)

<https://www.marketwatch.com/story/this-1-chart-puts-mega-techs-trillions-of-market-value-into-eye-popping-perspective-2018-07-18>

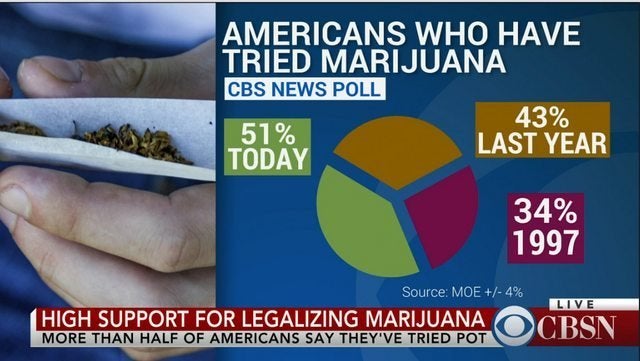
* Even after the chart’s implications have been refuted, it’s hard a bad (although compelling) visualization from being passed around.

<https://www.linkedin.com/pulse/good-bad-pie-charts-karthik-shashidhar/>

[**“yea I understand a pie chart was probably not the best choice to present this data.”**](https://twitter.com/michaelbatnick/status/1037036440494985216)

https://external-preview.redd.it/tAurUM7z0TjJnHnJiR4TUVaTqg0m3oKxgufIxdMA7j0.jpg?auto=webp&s=3b0322910a35ab9c9c46e07f4c0ea69f94c29411

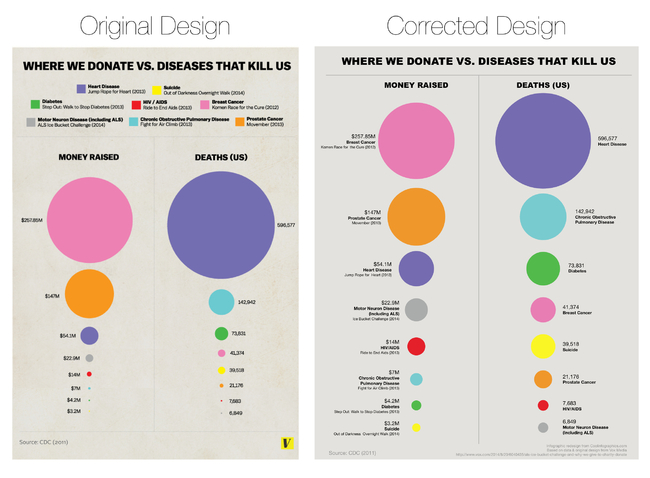
### Pie Charts that compare unrelated things are next-level extra bad



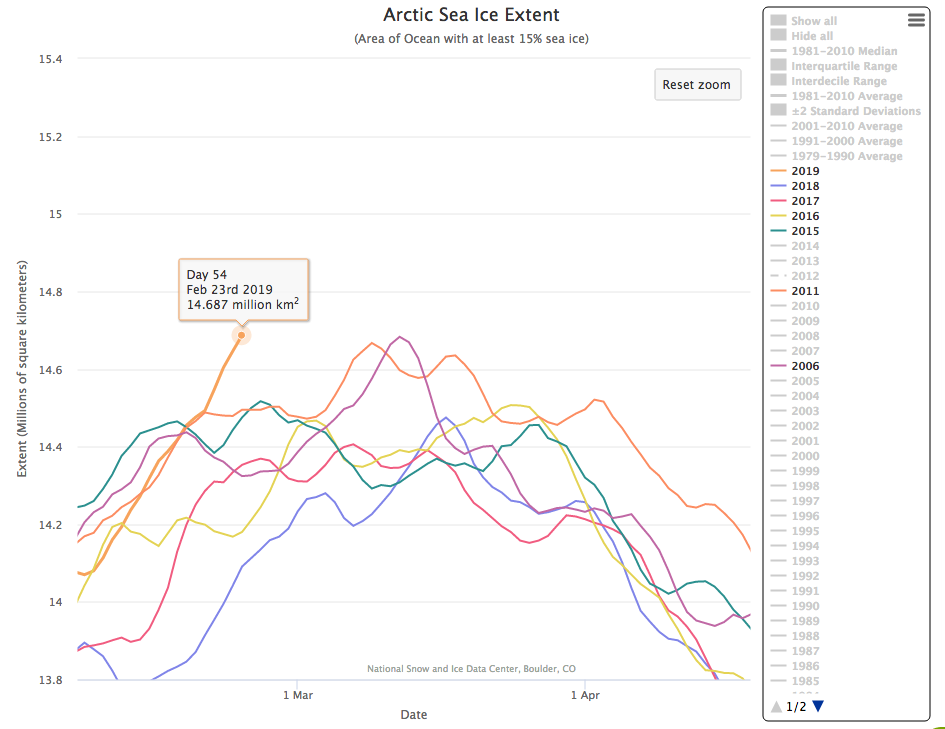
What kinds of charts would have been a better choice for displaying this data?

### Be careful about how you use volume to represent quantities:

radius vs diameter vs volume



### Don’t cherrypick timelines or specific subsets of your data:



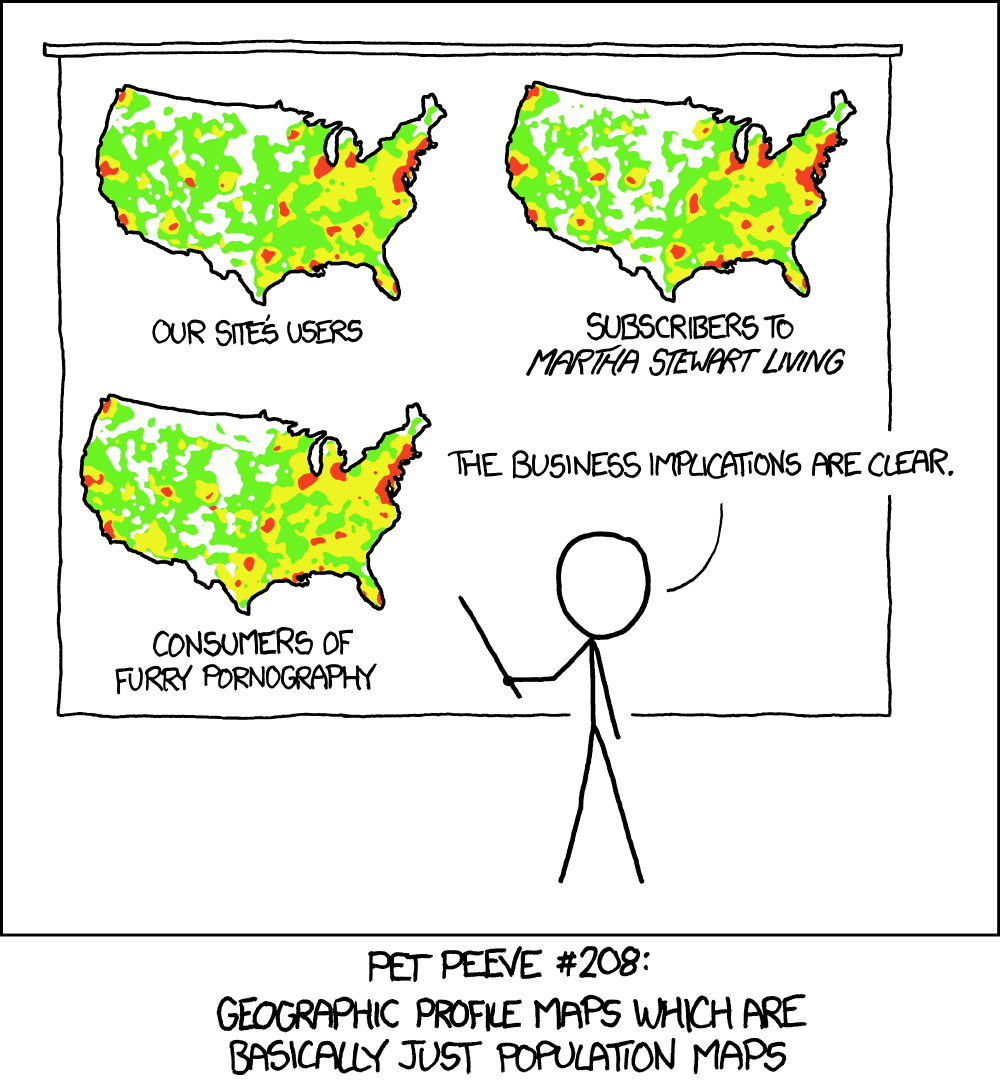
Look how specifically the writer has selected what years to show in the legend on the right side.

<https://wattsupwiththat.com/2019/02/24/strong-arctic-sea-ice-growth-this-year/>

Try the tool that was used to make the graphic for yourself

<http://nsidc.org/arcticseaicenews/charctic-interactive-sea-ice-graph/>

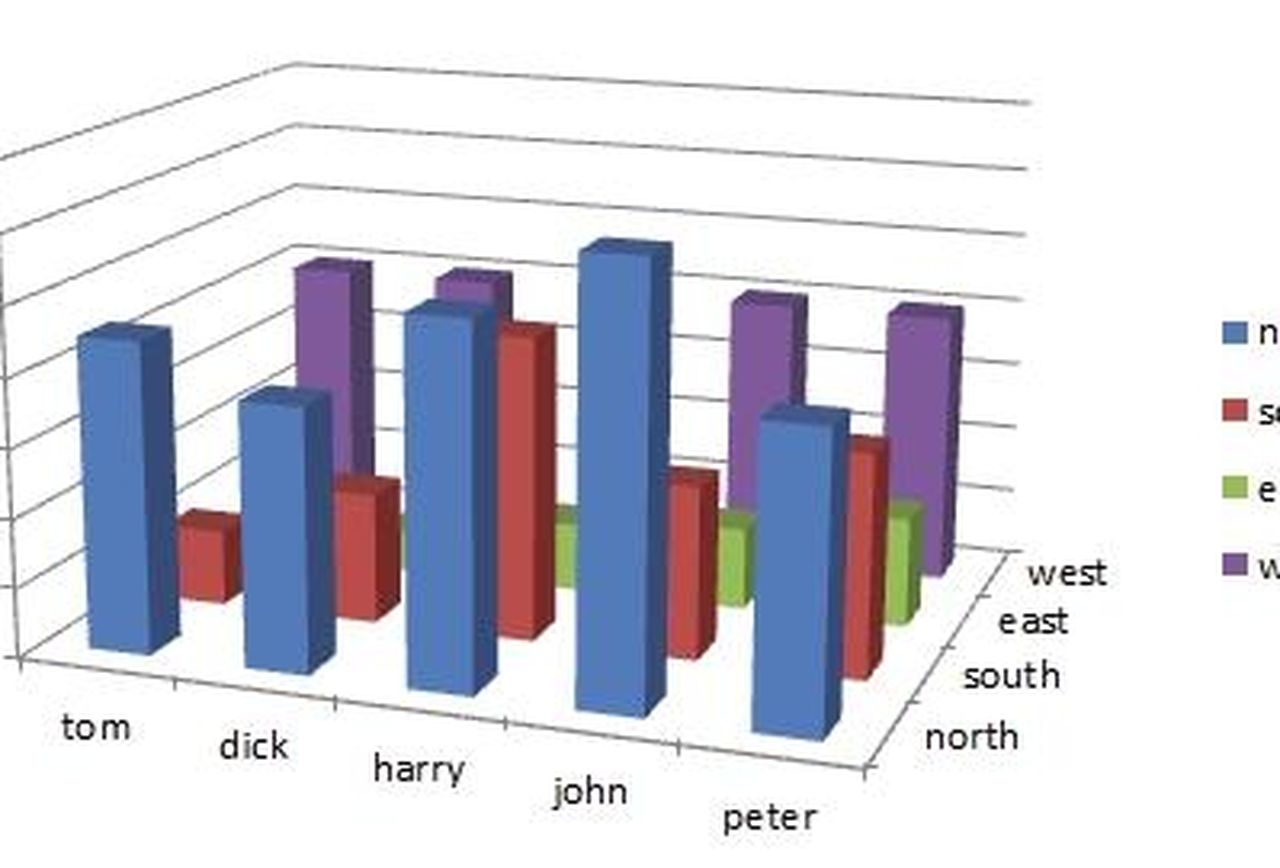
### Use Relative units rather than Absolute Units



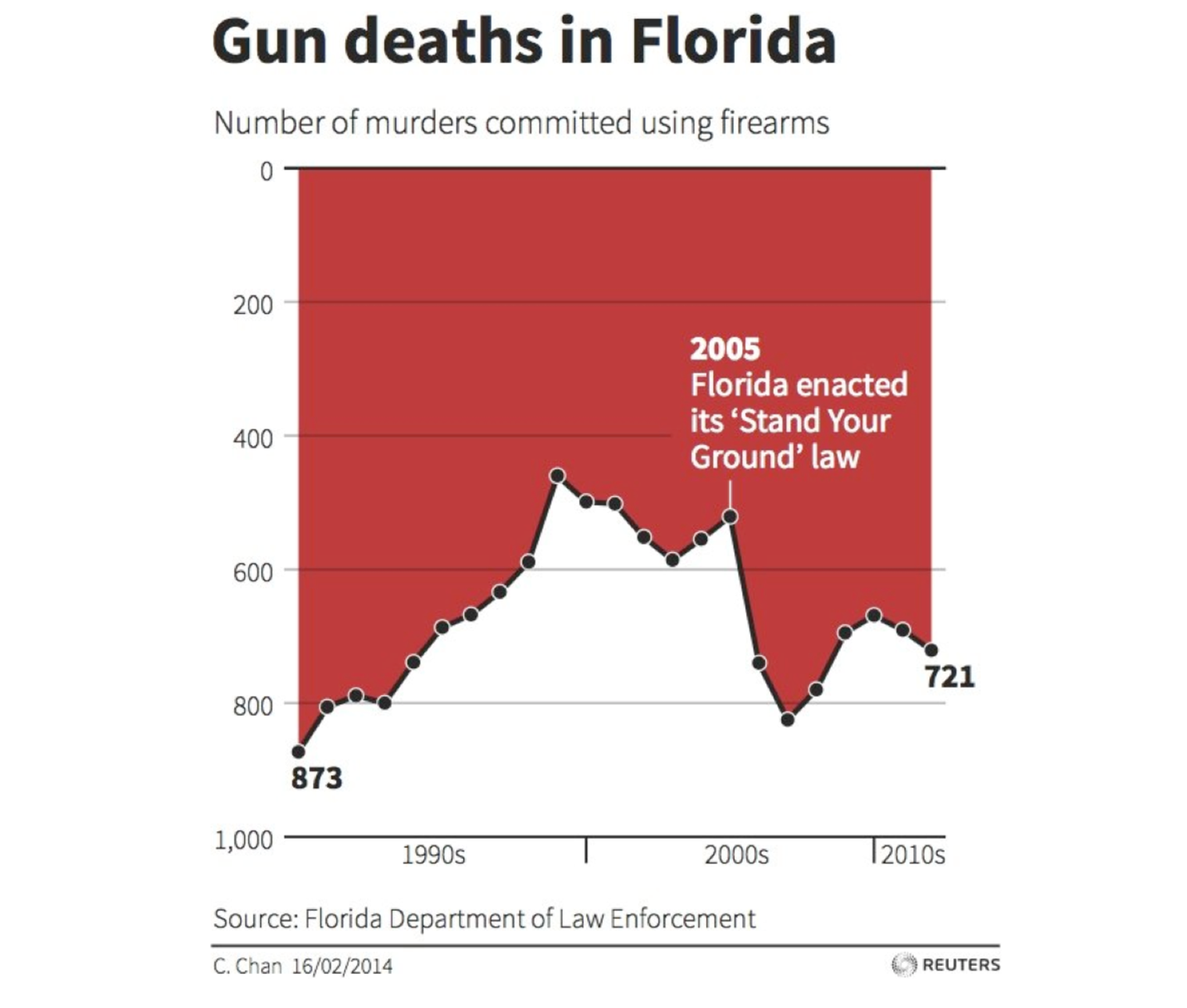
### Avoid 3D graphs unless having the extra dimension is effective

Usually you can Split 3D graphs into multiple 2D graphs

3D graphs that are interactive can be very cool. (See Plotly and Bokeh)



### Don’t go against typical conventions



### Use Appropriate “Visual Vocabulary”

[Visual Vocabulary - Vega Edition](https://ft.com/vocabulary)

### What are the properties of your data?

* Is your primary variable of interest continuous or discrete?
* Is in wide or long (tidy) format?
* Does your visualization involve multiple variables?
* How many dimensions do you need to include on your plot?

Can you express the main idea of your visualization in a single sentence?

How hard does your visualization make the user work in order to draw the intended conclusion?

##### Challenge

At the end of the unit you will be embarking on a week long “Data Storytelling Portfolio Project” and it would greatly please me to not see any of the common pitfalls that we have just covered be fallen into during that week.

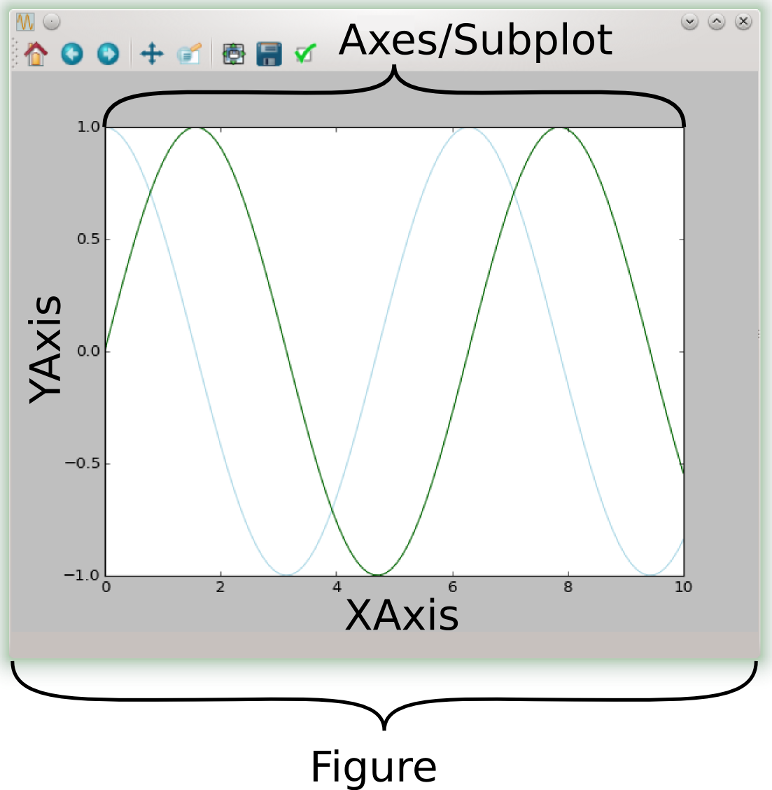
#### Learn to use appropriate terminology when referring to parts of a Matplotlib graph

The Matplotlib library has assigned specific names to different portions of the graphs that we will be creating. It is important to generally know what these parts of the graph are called because we will use that same terminology when we write our code, when we look up documentation, google questions, etc.

##### Overview

### The Anatomy of a Matplotlib Plot

Every plot has an inner section called the “Axes” this is where our data is actually displayed. This is sometimes referred to as the “subplot” if we’re displaying multiple axes within a single “figure.” The figure is outer background that wraps everything.



Here’s an example of a figure with subplots:

Each of the four subplots within this figure also has its on individual axes. The data within the axes - the green, blue, black and red lines is referred to as the “plot.”

The plot (data) lives inside the axes, which lives inside of the figure. These names are important because we understanding this organization will help us know what part of the graph to try and access in order to modify parts of the graph.

##### Follow Along

The following code is included mostly to create the plot that lives below it. I don’t expect you to understand everything that’s happening in this code cell, so don’t be intimidated by it. However, you can use the code below to learn more about matplotlib. Try commenting out specific lines and then watch how the graph below is altered to learn more about how Matplotlib plots are generated.

This learning strategy is sometimes called “learning by breaking things.”

Copy

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.ticker import AutoMinorLocator, MultipleLocator, FuncFormatter

np.random.seed(19680801)

X = np.linspace(0.5, 3.5, 100)

Y1 = 3+np.cos(X)

Y2 = 1+np.cos(1+X/0.75)/2

Y3 = np.random.uniform(Y1, Y2, len(X))

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(1, 1, 1, aspect=1)

def minor\_tick(x, pos):

if not x % 1.0:

return ""

return "%.2f" % x

ax.xaxis.set\_major\_locator(MultipleLocator(1.000))

ax.xaxis.set\_minor\_locator(AutoMinorLocator(4))

ax.yaxis.set\_major\_locator(MultipleLocator(1.000))

ax.yaxis.set\_minor\_locator(AutoMinorLocator(4))

ax.xaxis.set\_minor\_formatter(FuncFormatter(minor\_tick))

ax.set\_xlim(0, 4)

ax.set\_ylim(0, 4)

ax.tick\_params(which='major', width=1.0)

ax.tick\_params(which='major', length=10)

ax.tick\_params(which='minor', width=1.0, labelsize=10)

ax.tick\_params(which='minor', length=5, labelsize=10, labelcolor='0.25')

ax.grid(linestyle="--", linewidth=0.5, color='.25', zorder=-10)

ax.plot(X, Y1, c=(0.25, 0.25, 1.00), lw=2, label="Blue signal", zorder=10)

ax.plot(X, Y2, c=(1.00, 0.25, 0.25), lw=2, label="Red signal")

ax.plot(X, Y3, linewidth=0,

marker='o', markerfacecolor='w', markeredgecolor='k')

ax.set\_title("Anatomy of a figure", fontsize=20, verticalalignment='bottom')

ax.set\_xlabel("X axis label")

ax.set\_ylabel("Y axis label")

ax.legend()

def circle(x, y, radius=0.15):

from matplotlib.patches import Circle

from matplotlib.patheffects import withStroke

circle = Circle((x, y), radius, clip\_on=False, zorder=10, linewidth=1,

edgecolor='black', facecolor=(0, 0, 0, .0125),

path\_effects=[withStroke(linewidth=5, foreground='w')])

ax.add\_artist(circle)

def text(x, y, text):

ax.text(x, y, text, backgroundcolor="white",

ha='center', va='top', weight='bold', color='blue')

### Minor tick

circle(0.50, -0.10)

text(0.50, -0.32, "Minor tick label")

### Major tick

circle(-0.03, 4.00)

text(0.03, 3.80, "Major tick")

### Minor tick

circle(0.00, 3.50)

text(0.00, 3.30, "Minor tick")

### Major tick label

circle(-0.15, 3.00)

text(-0.15, 2.80, "Major tick label")

### X Label

circle(1.80, -0.27)

text(1.80, -0.45, "X axis label")

### Y Label

circle(-0.27, 1.80)

text(-0.27, 1.6, "Y axis label")

### Title

circle(1.60, 4.13)

text(1.60, 3.93, "Title")

### Blue plot

circle(1.75, 2.80)

text(1.75, 2.60, "Line\n(line plot)")

### Red plot

circle(1.20, 0.60)

text(1.20, 0.40, "Line\n(line plot)")

### Scatter plot

circle(3.20, 1.75)

text(3.20, 1.55, "Markers\n(scatter plot)")

### Grid

circle(3.00, 3.00)

text(3.00, 2.80, "Grid")

### Legend

circle(3.70, 3.80)

text(3.70, 3.60, "Legend")

### Axes

circle(0.5, 0.5)

text(0.5, 0.3, "Axes")

### Figure

circle(-0.3, 0.65)

text(-0.3, 0.45, "Figure")

color = 'blue'

ax.annotate('Spines', xy=(4.0, 0.35), xytext=(3.3, 0.5),

weight='bold', color=color,

arrowprops=dict(arrowstyle='->',

connectionstyle="arc3",

color=color))

ax.annotate('', xy=(3.15, 0.0), xytext=(3.45, 0.45),

weight='bold', color=color,

arrowprops=dict(arrowstyle='->',

connectionstyle="arc3",

color=color))

ax.text(4.0, -0.4, "Made with http://matplotlib.org",

fontsize=10, ha="right", color='.5')

plt.show()

Here we see some additional terminology like:

* title
* legend
* y axis label
* x axis label
* Major and Minor Ticks
* Major and Minor Tick Labels
* Spines

All of the blue text has been added at specific location as “text annotations” We’ll learn how to work with those today as well.

##### Challenge

Knowing this terminology will save you a lot of headache when it comes time to write Matplotlib code since this terminology is reflected in the code that you will need to write throughout the class attributes and methods of the library.

#### Learn to differentiate between Matplotlib syntaxes

There are multiple ways of working with the Matplotlib library -this is something that can be really confusing at the beginning. This goes against a strong Python convention of “having one clear way of doing things.” In spite of this challenge with this particular library we are going to take a “minimally sufficient” approach and teach you just one syntax that will provide you a great amount of control over the visual aspects of your graphs.

##### Overview

In order to try and avoid confusion, I am going to only teach you one of the ways of working with Matplotlib, but I’m going to show you how two of the most common syntax styles work so that if you’re out Googling things later today, you will know how to differentiate between the two when you’re googling things or reading blog posts and things later this afternoon:

##### Follow Along

##### Matplotlib Pyplot Syntax

Pyplot syntax is very similar to how we build visualizations with Pandas and is most useful for building quick, simple graphs. Today we’re going to try and learn how to heavily customize our graphs so I don’t want you to use this syntax during your work today.

It’s called Pyplot syntax because of the commonly used Matplotlib module pyplot.

You may have seen common import statements like the following:

import matplotlib.pyplot as plt

When we import matplotlib in this way, we’re typically creating a single graph at a time, and all of our graph manipulations happen on the plt object. Let me show you what I mean.

Copy

import matplotlib.pyplot as plt

sample\_data = [1,4,2,5,3,6]

plt.plot(sample\_data)

plt.title("The Title")

plt.xlabel('x label')

plt.ylabel('y label')

plt.show()

more\_data = [6,4,5,3,4,2]

plt.plot(more\_data)

plt.show()

##### Matplotlib Figure, Axes syntax

This syntax is a little bit easier to use when we’re really trying to customize and control everything about a graph. We will create separate Figure and Axes objects instead of one big plt object and then when we are changing attributes about our graph we’ll only either access the figure or the axes object.

I’m going to make a new plot with separate figure and axes objects and then color their backgrounds so that you can see the difference between the two very easily

“figure” and “axes” are commonly abbreviated as fig and ax when we’re referring to them with code. This isn’t a hard convention, but it’s a somewhat popular one. Remember that these are just variable names and you can technically name them whatever you want.

Copy

### generate separate figure and axes objects.

fig, ax = plt.subplots()

### Make the figure green

fig.patch.set(facecolor='green')

### Plot some fake data on the graph

ax.plot(sample\_data)

### make the figure yellow

ax.set(facecolor="yellow")

### Set the graph title, xlabel, and ylabel

### Notice that we set these on the AXES and not on the figure

### even though they stick out past the edges of the axes a little bit.

ax.set\_title("Graph Title")

ax.set\_ylabel('my y label')

ax.set\_xlabel('my x label');

##### Challenge

As you are googling things and looking through tutorials and documentations this afternoon, please ignore any resources that use the “pyplot” syntax and continue looking until you find resources that help you to use the “figure, axes” syntax. The pyplot syntax is still useful for making quick graphs but doesn’t allow the same level of customizability as the “figure, axes” syntax.

#### Learn to use Matplotlib to control basic visual aspects of their plot so as to mimic popular plotting styles (FiveThirtyEight) including: plot, plot stylesheet, title, subtitle, axis labels, axis tick marks, background colors, text annotations.

In order to practice controlling the visual aspects of our graphs, we are going to try and mimic the plotting style of some visualization experts. This learning method of emulating the styles of individuals that we look up to is sometimes called the “Benjamin Franklin Method.”

##### Overview

Today we will reproduce this [example by FiveThirtyEight:](https://fivethirtyeight.com/features/al-gores-new-movie-exposes-the-big-flaw-in-online-movie-ratings/)

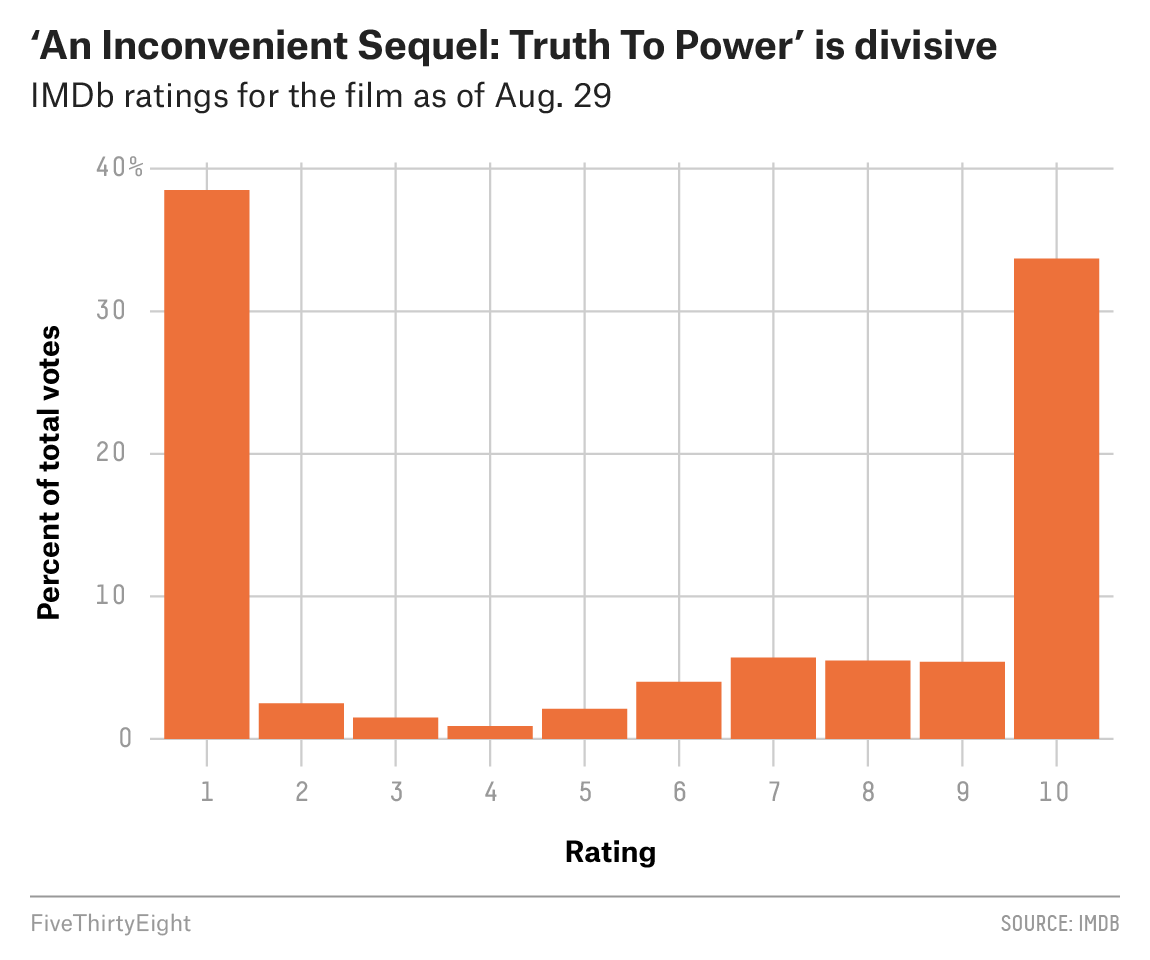
Copy

from IPython.display import display, Image

url = 'https://fivethirtyeight.com/wp-content/uploads/2017/09/mehtahickey-inconvenient-0830-1.png'

example = Image(url=url, width=400)

display(example)



Using this data: https://github.com/fivethirtyeight/data/tree/master/inconvenient-sequel

Links

* [Strong Titles Are The Biggest Bang for Your Buck](http://stephanieevergreen.com/strong-titles/)
* [Remove to improve (the data-ink ratio)](https://www.darkhorseanalytics.com/blog/data-looks-better-naked)
* [How to Generate FiveThirtyEight Graphs in Python](https://www.dataquest.io/blog/making-538-plots/)

##### Follow Along

One of the cool things about FiveThirtyEight is that they provide a lot of the raw datasets for the visualizations that they create. We’ll look at more of these during the assignment.

Instead of diving into data wrangling, I want to get straight to the plotting, so we’re going to make up our own data that gets us in the ballpark of what we need to start plotting these graphs, and then we’ll go back and explore the dataset afterwards as time permits.

I would call these first two graphics “Exploratory” visualizations because they’re done in the quick and dirty way. The more that we customize the look and feel of our plot and really improve the quality of it, then our visualizations become “Explanatory” visualziations.

Copy

%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

fake = pd.Series([38, 3, 2, 1, 2, 4, 6, 5, 5, 33],

index=range(1,11))

fake.plot.bar(color='C1', width=0.9);

Copy

fake2 = pd.Series(

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

2, 2, 2,

3, 3, 3,

4, 4,

5, 5, 5,

6, 6, 6, 6,

7, 7, 7, 7, 7,

8, 8, 8, 8,

9, 9, 9, 9,

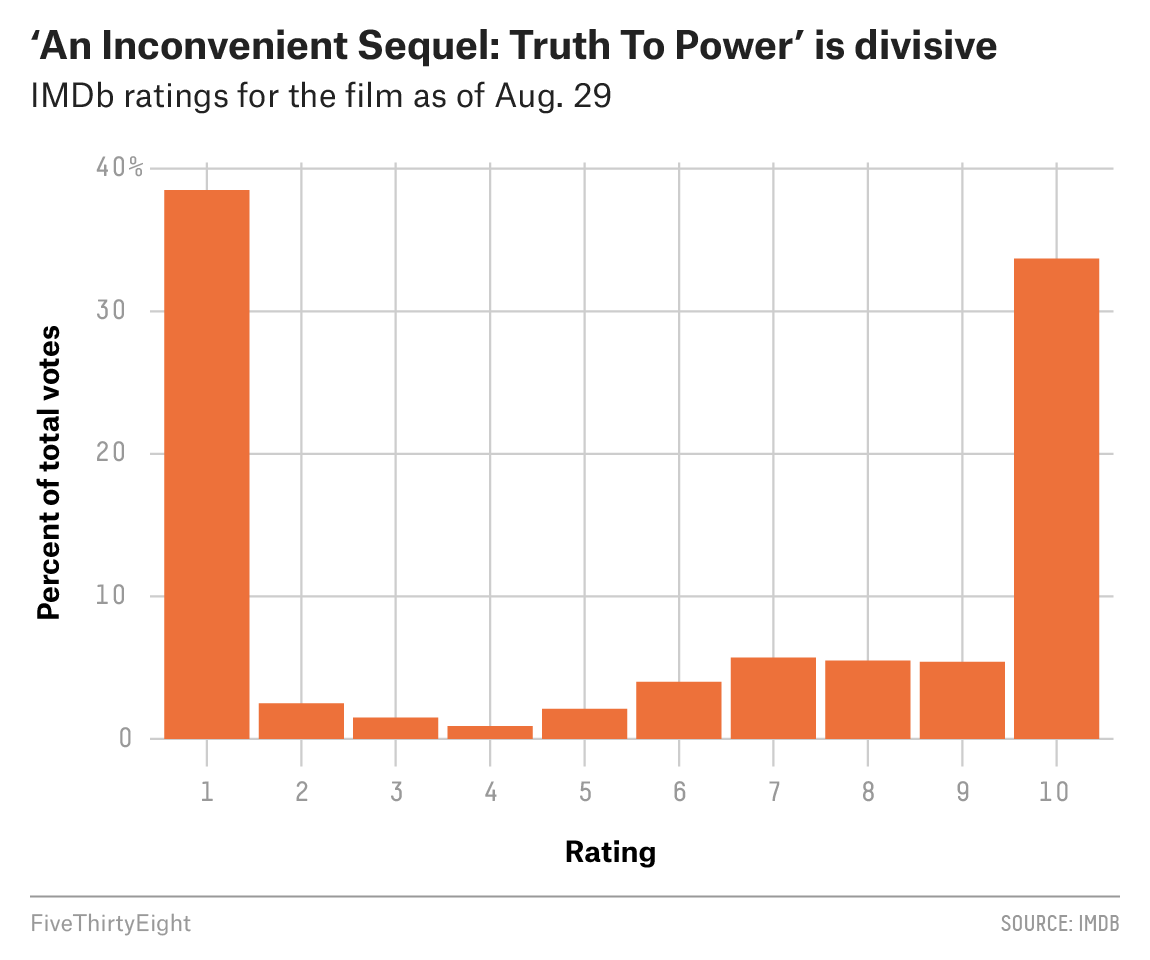
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10])

### fake2.value\_counts().sort\_index().plot.bar(color='C1', width=0.9);

fake2.value\_counts().sort\_index().plot.bar(color='C1', width=0.9);

Copy

display(example)



##### Mimic the style of FiveThirtyEight

What piece of graph anatomy do you want to start with? Maybe we can give ourselves a headstart if we pick a good starting [stylesheet](https://matplotlib.org/3.1.1/gallery/style_sheets/style_sheets_reference.htmlhttps:/)?

Copy

### Set stylesheet

plt.style.use('fivethirtyeight')

### figure axes syntax

fig, ax = plt.subplots()

### Make the figure background color white

fig.patch.set(facecolor='white')

### put bars on the axes

fake.plot.bar(width=0.9, color='#ED713A')

### Set axes background color

ax.set(facecolor='white')

### Set plot title

ax.set\_title("'An Inconvenient Sequel: truth To Power' is divisive", fontsize=12, fontweight='bold', x=.35, y=1.1)

### Set plot ylabel

ax.set\_ylabel('Percent of total votes', fontsize=9, fontweight='bold')

### Set plot xlabel

ax.set\_xlabel('Rating', fontsize=9, fontweight='bold')

### Set subtitle

ax.text(s='IMDb ratings for the film as of Aug. 29', y=41.5, x=-1.8, fontsize=12)

### set y axis labels, tick marks and grid lines

ax.set\_yticklabels(['0', '10', '20', '30', '40%'], fontsize=10)

ax.set\_yticks(range(0,50,10))

### Set x axis labels, tick marks and grid lines

ax.set\_xticklabels(range(1,11,1), rotation='horizontal', fontsize=10)

### Show the plot and squelch the funny output

fig.show()

##### Reproduce with “real” data (if time permitting)

Copy

df = pd.read\_csv('https://raw.githubusercontent.com/fivethirtyeight/data/master/inconvenient-sequel/ratings.csv')

Copy

df.head()

|  | **timestamp** | **respondents** | **category** | **link** | **average** | **mean** | **median** | **1\_votes** | **2\_votes** | **3\_votes** | **4\_votes** | **5\_votes** | **6\_votes** | **7\_votes** | **8\_votes** | **9\_votes** | **10\_votes** | **1\_pct** | **2\_pct** | **3\_pct** | **4\_pct** | **5\_pct** | **6\_pct** | **7\_pct** | **8\_pct** | **9\_pct** | **10\_pct** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2017-07-17 12:28:32.785639 | 402 | Males | http://www.imdb.com/title/tt6322922/ratings-male | 4.6 | 5.0 | 2 | 197 | 7 | 7 | 3 | 7 | 7 | 11 | 8 | 20 | 135 | 49.0 | 1.7 | 1.7 | 0.7 | 1.7 | 1.7 | 2.7 | 2.0 | 5.0 | 33.6 |
| **1** | 2017-07-17 12:28:33.025600 | 78 | Females | http://www.imdb.com/title/tt6322922/ratings-fe... | 6.9 | 7.7 | 10 | 16 | 1 | 0 | 1 | 1 | 0 | 3 | 4 | 3 | 49 | 20.5 | 1.3 | 0.0 | 1.3 | 1.3 | 0.0 | 3.8 | 5.1 | 3.8 | 62.8 |
| **2** | 2017-07-17 12:28:33.273919 | 4 | Aged under 18 | http://www.imdb.com/title/tt6322922/ratings-age\_1 | 4.2 | 4.2 | 3 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 50.0 | 0.0 | 0.0 | 0.0 | 25.0 | 0.0 | 0.0 | 0.0 | 0.0 | 25.0 |
| **3** | 2017-07-17 12:28:33.495325 | 4 | Males under 18 | http://www.imdb.com/title/tt6322922/ratings-ma... | 4.2 | 4.2 | 3 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 50.0 | 0.0 | 0.0 | 0.0 | 25.0 | 0.0 | 0.0 | 0.0 | 0.0 | 25.0 |
| **4** | 2017-07-17 12:28:33.722849 | 130 | Aged 18-29 | http://www.imdb.com/title/tt6322922/ratings-age\_2 | 6.3 | 6.5 | 9 | 41 | 0 | 3 | 1 | 2 | 3 | 6 | 4 | 6 | 64 | 31.5 | 0.0 | 2.3 | 0.8 | 1.5 | 2.3 | 4.6 | 3.1 | 4.6 | 49.2 |

Copy

df.tail()

|  | **timestamp** | **respondents** | **category** | **link** | **average** | **mean** | **median** | **1\_votes** | **2\_votes** | **3\_votes** | **4\_votes** | **5\_votes** | **6\_votes** | **7\_votes** | **8\_votes** | **9\_votes** | **10\_votes** | **1\_pct** | **2\_pct** | **3\_pct** | **4\_pct** | **5\_pct** | **6\_pct** | **7\_pct** | **8\_pct** | **9\_pct** | **10\_pct** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **80048** | 2017-08-29 23:10:05.369510 | 8 | IMDb staff | http://www.imdb.com/title/tt6322922/ratings-im... | 8.4 | 7.2 | 8 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2 | 1 | 12.5 | 0.0 | 0.0 | 0.0 | 0.0 | 12.5 | 12.5 | 25.0 | 25.0 | 12.5 |
| **80049** | 2017-08-29 23:10:05.598331 | 41 | Top 1000 voters | http://www.imdb.com/title/tt6322922/ratings-to... | 4.6 | 4.7 | 5 | 11 | 4 | 2 | 2 | 5 | 4 | 5 | 1 | 1 | 6 | 26.8 | 9.8 | 4.9 | 4.9 | 12.2 | 9.8 | 12.2 | 2.4 | 2.4 | 14.6 |
| **80050** | 2017-08-29 23:10:05.794141 | 925 | US users | http://www.imdb.com/title/tt6322922/ratings-usa | 4.8 | 5.0 | 4 | 405 | 34 | 20 | 8 | 9 | 30 | 54 | 46 | 60 | 259 | 43.8 | 3.7 | 2.2 | 0.9 | 1.0 | 3.2 | 5.8 | 5.0 | 6.5 | 28.0 |
| **80051** | 2017-08-29 23:10:06.022268 | 565 | Non-US users | http://www.imdb.com/title/tt6322922/ratings-in... | 6.4 | 6.5 | 7 | 129 | 11 | 10 | 8 | 32 | 47 | 52 | 37 | 35 | 204 | 22.8 | 1.9 | 1.8 | 1.4 | 5.7 | 8.3 | 9.2 | 6.5 | 6.2 | 36.1 |
| **80052** | 2017-08-29 23:10:06.218251 | 2662 | IMDb users | http://www.imdb.com/title/tt6322922/ratings | 5.4 | 5.6 | 7 | 1021 | 69 | 38 | 25 | 55 | 110 | 154 | 147 | 146 | 897 | 38.4 | 2.6 | 1.4 | 0.9 | 2.1 | 4.1 | 5.8 | 5.5 | 5.5 | 33.7 |

Copy

### Convert timestamps strings to actual datetime objects

df['timestamp'] = pd.to\_datetime(df['timestamp'])

Copy

### Use the timestamp as the unique index identifier

### so that we can select rows by timestamp

df.set\_index('timestamp', inplace=True)

Copy

### grab only the rows corresponding to the last day

lastday = df['2017-08-29']

Copy

### get the demographic breakdowns for all IMDb users on the last day

lastday\_filtered = lastday[lastday['category'] == 'IMDb users']

lastday\_filtered.tail()

|  | **respondents** | **category** | **link** | **average** | **mean** | **median** | **1\_votes** | **2\_votes** | **3\_votes** | **4\_votes** | **5\_votes** | **6\_votes** | **7\_votes** | **8\_votes** | **9\_votes** | **10\_votes** | **1\_pct** | **2\_pct** | **3\_pct** | **4\_pct** | **5\_pct** | **6\_pct** | **7\_pct** | **8\_pct** | **9\_pct** | **10\_pct** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **timestamp** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2017-08-29 22:30:06.423181** | 2662 | IMDb users | http://www.imdb.com/title/tt6322922/ratings | 5.4 | 5.6 | 7 | 1021 | 69 | 38 | 25 | 55 | 110 | 154 | 147 | 146 | 897 | 38.4 | 2.6 | 1.4 | 0.9 | 2.1 | 4.1 | 5.8 | 5.5 | 5.5 | 33.7 |
| **2017-08-29 22:40:06.233659** | 2662 | IMDb users | http://www.imdb.com/title/tt6322922/ratings | 5.4 | 5.6 | 7 | 1021 | 69 | 38 | 25 | 55 | 110 | 154 | 147 | 146 | 897 | 38.4 | 2.6 | 1.4 | 0.9 | 2.1 | 4.1 | 5.8 | 5.5 | 5.5 | 33.7 |
| **2017-08-29 22:50:06.592571** | 2662 | IMDb users | http://www.imdb.com/title/tt6322922/ratings | 5.4 | 5.6 | 7 | 1021 | 69 | 38 | 25 | 55 | 110 | 154 | 147 | 146 | 897 | 38.4 | 2.6 | 1.4 | 0.9 | 2.1 | 4.1 | 5.8 | 5.5 | 5.5 | 33.7 |
| **2017-08-29 23:00:05.829558** | 2662 | IMDb users | http://www.imdb.com/title/tt6322922/ratings | 5.4 | 5.6 | 7 | 1021 | 69 | 38 | 25 | 55 | 110 | 154 | 147 | 146 | 897 | 38.4 | 2.6 | 1.4 | 0.9 | 2.1 | 4.1 | 5.8 | 5.5 | 5.5 | 33.7 |
| **2017-08-29 23:10:06.218251** | 2662 | IMDb users | http://www.imdb.com/title/tt6322922/ratings | 5.4 | 5.6 | 7 | 1021 | 69 | 38 | 25 | 55 | 110 | 154 | 147 | 146 | 897 | 38.4 | 2.6 | 1.4 | 0.9 | 2.1 | 4.1 | 5.8 | 5.5 | 5.5 | 33.7 |

Copy

### just grab the very last line (latest timestamp) of IMDb user ratings

### this should be the most up to date data from the dataset

final = lastday\_filtered.tail(1)

final.T

| **timestamp** | **2017-08-29 23:10:06.218251** |
| --- | --- |
| **respondents** | 2662 |
| **category** | IMDb users |
| **link** | http://www.imdb.com/title/tt6322922/ratings |
| **average** | 5.4 |
| **mean** | 5.6 |
| **median** | 7 |
| **1\_votes** | 1021 |
| **2\_votes** | 69 |
| **3\_votes** | 38 |
| **4\_votes** | 25 |
| **5\_votes** | 55 |
| **6\_votes** | 110 |
| **7\_votes** | 154 |
| **8\_votes** | 147 |
| **9\_votes** | 146 |
| **10\_votes** | 897 |
| **1\_pct** | 38.4 |
| **2\_pct** | 2.6 |
| **3\_pct** | 1.4 |
| **4\_pct** | 0.9 |
| **5\_pct** | 2.1 |
| **6\_pct** | 4.1 |
| **7\_pct** | 5.8 |
| **8\_pct** | 5.5 |
| **9\_pct** | 5.5 |
| **10\_pct** | 33.7 |

Copy

### Grab only the percentage columns since we don't care about the raw

### counts in making our graph

pct\_columns = ['1\_pct', '2\_pct', '3\_pct', '4\_pct', '5\_pct',

'6\_pct', '7\_pct', '8\_pct', '9\_pct', '10\_pct']

final[pct\_columns].T

| **timestamp** | **2017-08-29 23:10:06.218251** |
| --- | --- |
| **1\_pct** | 38.4 |
| **2\_pct** | 2.6 |
| **3\_pct** | 1.4 |
| **4\_pct** | 0.9 |
| **5\_pct** | 2.1 |
| **6\_pct** | 4.1 |
| **7\_pct** | 5.8 |
| **8\_pct** | 5.5 |
| **9\_pct** | 5.5 |
| **10\_pct** | 33.7 |

Copy

### Reset the index so that it's numeric again

### and rename the percent column for easy access in our plotting

plot\_data = final[pct\_columns].T

plot\_data.index = range(1,11)

plot\_data.columns = ['percent']

plot\_data

|  | **percent** |
| --- | --- |
| **1** | 38.4 |
| **2** | 2.6 |
| **3** | 1.4 |
| **4** | 0.9 |
| **5** | 2.1 |
| **6** | 4.1 |
| **7** | 5.8 |
| **8** | 5.5 |
| **9** | 5.5 |
| **10** | 33.7 |

Copy

import matplotlib.pyplot as plt

### Set stylesheet

plt.style.use('fivethirtyeight')

### figure axes syntax

fig, ax = plt.subplots()

### Make the figure background color white

fig.patch.set(facecolor='white')

### put bars on the axes

ax.bar(x=range(1,11), height=plot\_data['percent'], width=0.9, color='#ED713A')

### Set axes background color

ax.set(facecolor='white')

### Set plot title

ax.set\_title("'An Inconvenient Sequel: truth To Power' is divisive", fontsize=12, fontweight='bold', x=.35, y=1.1)

### Set plot ylabel

ax.set\_ylabel('Percent of total votes', fontsize=9, fontweight='bold')

### Set plot xlabel

ax.set\_xlabel('Rating', fontsize=9, fontweight='bold')

### Set subtitle

ax.text(s='IMDb ratings for the film as of Aug. 29', y=42.5, x=-1, fontsize=12)

### set y axis labels, tick marks and grid lines

ax.set\_yticklabels(['0', '10', '20', '30', '40%'], fontsize=10)

ax.set\_yticks(range(0,50,10))

### Set x axis labels, tick marks and grid lines

ax.set\_xticklabels(range(1,11,1), rotation='horizontal', fontsize=10, fontweight=550)

ax.set\_xticks(range(1,11))

### Show the plot and squelch the funny output

fig.show()

##### Challenge

Take this further on your assignment by making the above example as pixel perfect as possible. Once you are satisfied with your creation you will pick one other graph from the website FiveThirtyEight’s shared data repository in order to practice at least one graphing example completely on your own.

## Review

### Class Recordings

You can use class recordings to help you master the material.

* [**Make Explanatory Visualizations for DS14 w/ Ryan Allred**](https://youtu.be/hneds69oFME)

Unit 1 Sprint 1 Module 4

* [All previous recordings](https://learn.lambdaschool.com/archive/DS/module/recK7pBbrxfDphtSd)

### Demonstrate Mastery

To demonstrate mastery of this module, you need to complete and pass a code review on each of the following:

* Objective challenge:

At the end of the unit you will be embarking on a week long “Data Storytelling Portfolio Project” and it would greatly please me to not see any of the common pitfalls that we have just covered be fallen into during that week.

* Objective challenge:

Knowing this terminology will save you a lot of headache when it comes time to write Matplotlib code since this terminology is reflected in the code that you will need to write throughout the class attributes and methods of the library.

* Objective challenge:

As you are googling things and looking through tutorials and documentations this afternoon, please ignore any resources that use the “pyplot” syntax and continue looking until you find resources that help you to use the “figure, axes” syntax. The pyplot syntax is still useful for making quick graphs but doesn’t allow the same level of customizability as the “figure, axes” syntax.

* Objective challenge:

Take this further on your assignment by making the above example as pixel perfect as possible. Once you are satisfied with your creation you will pick one other graph from the website FiveThirtyEight’s shared data repository in order to practice at least one graphing example completely on your own.