UDACITY

< Return to Classroom

DISCUSS ON STUDENT HUB >

Investigate a Dataset

REVIEW

HISTORY

Requires Changes

4 specifications require changes

This is an excellent first submission for this project.

- You take a complex dataset and reveal some interesting, and unusual, patterns *(for example: "comparing both genders who scheduled appointments, both genders have roughly the same show up rate")*.

Excellent work!

However, *reviewers are ruled by the* ***Rubric*** and there are still main requirements that have not been met. However, many of the changes are minor *(where* ***"very minor issues"*** *take just a few minutes to resolve)* and others overlap (that is, when you fix one then another requirement will be met). The main issues to be addressed relate to:

- Include *at least* one function in your script.
- In your conclusion, add a discussion of the limitations of drawing broader conclusions from the results of your analysis.
- All visualizations have to have titles and axes labels (with units of measurement were relevant).

Each of these are explained in the sections below.

Note: Given that the structure of your project is sound (there are no major changes required), it will not take long to make the changes required to meet all of the requirements.

Overall, very nice work!! Best wishes for your re-submission!

*p.s if you have any questions about any aspect of this review, you can ask mentors questions on **Knowledge***

Important

- Where changes are required, a simple list of required changes is provided.
    - ***If you feel comfortable making those changes, all other comments can be ignored. They are included as guides, to help make the required changes.***
- Sections marked `Note` are **"for your information"** *(and, if you wish, can be read later, after you have finished your project).*
- Sections marked `Suggestions/Tip` detail ways that you can improve your project *(which you can incorporate into your project but are **optional**).*

Code Functionality

All code is functional and produces no errors when run. The code given is sufficient to reproduce the results described.

The code in your notebook evaluates as expected (without errors). Nice work!

TIPS

DATA PATH

```
[0]: 1 df.to_csv('C:/Users/user/Desktop/1.coding_datascience/UDACITY NANO DEGREE data analyst/Project 2/df_cleaned.csv', index=False)
```

2 Exploratory Data Analysis

> with our cleaned data, we are ready to explore. we will create visualizations with the goal of addressing the research questions that we posed in the Introduction section.

ad the cleaned data and check first few rows and colums.

```
[1]: 1 df = pd.read_csv('C:/Users/user/Desktop/1.coding_datascience/UDACITY NANO DEGREE data analyst/Project 2/df_cleaned.csv')
     2 df.head()
```

Click On Images To Enlarge Them

- When sharing code, set a `path` variable at the top of your script. You can then use that every time that you want to load or save data.
- You can then use the `os` module's `os.path.join()` to combine the data path and the data file names (link)

Why is this useful?

When you share your work, your colleague will only have to change one variable (as long as you share the data with your code).

PATH VARIABLE

Import all necessary libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import re
import os

#To display entire dataset
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows',  None)

# path to data
path="C:/Users/user/Desktop/1.coding_datascience/UDACITY NANO DEGREE data analyst/Project 2/"
```

when sharing code, set a 'path' variable at the start of your script

that way, your colleague only has to change one line of code (use the module 'os' to access that path).

1.3 Data Wrangling

RE-LOADING SAVED DATA

There is no need to, and it can be harmful to, reload data that you just saved.

If you save data, that means that the data must *already be **in python's memory***, so there is no need to reload it.

Why might it be harmful? Saving data to `.csv` files does **not** save the **data types** with the data. As can be seen here:

2.1 Cleaned Data

great work saving the cleaned data!!

let us save the cleaned data below so that we can use it for the Exploratory Data Analysis

In [90]:
```
# save the data, to the path specified earlier
df.to_csv(os.path.join(path,'df_cleaned.csv'), index=False)
```

In [91]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110526 entries, 0 to 110526
Data columns (total 14 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   patient_id       110526 non-null  object
 1   appointment_id   110526 non-null  object
 2   gender           110526 non-null  object
 3   scheduled_day    110526 non-null  datetime64[ns]
 4   appointment_day  110526 non-null  datetime64[ns]
 5   age              110526 non-null  int64
 6   neighbourhood    110526 non-null  object
 7   scholarship      110526 non-null  bool
 8   hypertension     110526 non-null  bool
 9   diabetes         110526 non-null  bool
 10  alcoholism       110526 non-null  bool
 11  sms_received     110526 non-null  int64
 12  handicap         110526 non-null  bool
 13  showed_up        110526 non-null  bool
dtypes: bool(6), datetime64[ns](2), int64(2), object(4)
memory usage: 8.2+ MB
```

2.2 Exploratory Data Analysis

> with our cleaned data, we are ready to explore. we will create visualizations with the goal of addressing the research questions that we posed in the Introduction section.

Load the cleaned data and check first few rows and colums.

In [93]:
```
df = pd.read_csv(os.path.join(path,'df_cleaned.csv'))
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110526 entries, 0 to 110525
Data columns (total 14 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   patient_id       110526 non-null  int64
 1   appointment_id   110526 non-null  int64
 2   gender           110526 non-null  object
 3   scheduled_day    110526 non-null  object
 4   appointment_day  110526 non-null  object
 5   age              110526 non-null  int64
 6   neighbourhood    110526 non-null  object
 7   scholarship      110526 non-null  bool
 8   hypertension     110526 non-null  bool
 9   diabetes         110526 non-null  bool
 10  alcoholism       110526 non-null  bool
 11  sms_received     110526 non-null  int64
 12  handicap         110526 non-null  bool
 13  showed_up        110526 non-null  bool
dtypes: bool(6), int64(4), object(4)
memory usage: 7.4+ MB
```

there is no need to reload the cleaned data, it is already in memory (and the data types are not saved with the file - so you may lose information)

So you will actually **lose information** by reloading that data.

Instead, just continue using the version in python's memory:

2.1 Cleaned Data

let us save the cleaned data below so that we can use it for the Exploratory Data Analysis

In [122]:
```
# save the data, to the path specified earlier
df.to_csv(os.path.join(path,'df_cleaned.csv'), index=False)
```

In [123]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110526 entries, 0 to 110526
Data columns (total 14 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   patient_id       110526 non-null  object
 1   appointment_id   110526 non-null  object
 2   gender           110526 non-null  object
 3   scheduled_day    110526 non-null  datetime64[ns]
 4   appointment_day  110526 non-null  datetime64[ns]
 5   age              110526 non-null  int64
 6   neighbourhood    110526 non-null  object
 7   scholarship      110526 non-null  bool
 8   hypertension     110526 non-null  bool
 9   diabetes         110526 non-null  bool
 10  alcoholism       110526 non-null  bool
 11  sms_received     110526 non-null  int64
 12  handicap         110526 non-null  bool
 13  showed_up        110526 non-null  bool
dtypes: bool(6), datetime64[ns](2), int64(2), object(4)
memory usage: 8.2+ MB
```

2.2 Exploratory Data Analysis ¶

> with our cleaned data, we are ready to explore. we will create visualizations with the goal of addressing the research questions that

Load the cleaned data and check first few rows and colums.

In [124]:
```
# df = pd.read_csv(os.path.join(path,'df_cleaned.csv'))
df.info()
```

commented out, continue using the original cleaned data

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110526 entries, 0 to 110526
Data columns (total 14 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   patient_id       110526 non-null  object
 1   appointment_id   110526 non-null  object
 2   gender           110526 non-null  object
 3   scheduled_day    110526 non-null  datetime64[ns]
 4   appointment_day  110526 non-null  datetime64[ns]
 5   age              110526 non-null  int64
 6   neighbourhood    110526 non-null  object
 7   scholarship      110526 non-null  bool
 8   hypertension     110526 non-null  bool
 9   diabetes         110526 non-null  bool
 10  alcoholism       110526 non-null  bool
 11  sms_received     110526 non-null  int64
 12  handicap         110526 non-null  bool
 13  showed_up        110526 non-null  bool
dtypes: bool(6), datetime64[ns](2), int64(2), object(4)
memory usage: 8.2+ MB
```

As you can see, in each of the examples above, I use `os.path.join()` to join the `path` to the data (which is different for me) with the file name.

- To help develop your skills using python, I highly recommend working through the examples in this free online text

Your Questions

> "how to generate research questions"

The idea of a *data exploration* is that, before you begin, you note down some thoughts that you have about the data that you are going to explore - some immediate questions that pop into your mind. Then you use that as a base for creating visualizations - to answer those questions. Often, those visualizations will lead to other questions. So that the data exploration becomes like a *"stream of consciousness"* of questions and answers.

So, it is your curiosity that will determine the questions *(there is no one fixed set of questions for all data, it is the data that drives both your curiosity and, therefore, the questions)*

Obviously, in a work environment, there will be a reason that you are studying the data. That reason will typically motivate your research questions.

> "how to answer research questions using various visualizations"

As you will see later, to answer this question *is in fact a course on its own*.

When you get to the **Communicate Data Findings** project, you will be introduced to the different approaches to *communicating* data using visualizations.

As it is a complete course, there is no real way for me to answer that succinctly.

"during EDA, how many visualizations is appropriate to answer a research question?"

This depends on the number of variables *(of interest)* in the dataset.

Typically, you will have more *Univariate* visualizations (one for each of the variables that you use in the *Bivariate* and *Multivariate* visualizations). *Data explorations* are hierarchical in that way (variables used in later sections are explored also in earlier sections).

So, if you have, say, 4 multivariate visualizations (i.e. 4 interesting multivariate relationships), then you will have at least 4 bivariate and 8 univariate visualizations.

Again, this is data dependent (if there is only one multivariate relationship that you are really interested in then you will divide the numbers above by 4).

The project uses NumPy arrays and Pandas Series and DataFrames where appropriate rather than Python lists and dictionaries. Where possible, vectorized operations and built-in functions are used instead of loops.

pandas is used for most data wrangling tasks, so vectorized operations rather than loops are used. Nice work.

TIP *(NOT REQUIRED)*

(You can read this later, if you wish)

- Explore vectorized operations methods

As you get more experience using *pandas*, you will find that, whatever task you want to achieve, there will be a *vectorized operation* that will allow you to achieve it.

One handy operation on *pandas* series/dataframes is `.unstack()` *(link)*. It moves row indices to column indices, which doesn't sound too interesting except when you see how this makes plotting easier:

In [24]:
```
# Replace Gender categories
df.Gender.replace({"F":"Female","M":"Male"}, inplace = True)
# groupby by and count number in each category
df.groupby(['Gender','No_show']).size()
```

```
Gender  No_show
Female  No         57246
        Yes        14594
Male    No         30962
        Yes         7725
dtype: int64
```

Only included to show what the plot code does

In [25]:
```
# unstack - move row index to columns
df.groupby(['Gender','No_show']).size().unstack('Gender')
```

Gender	Female	Male
No_show		
No	57246	30962
Yes	14594	7725

In [26]:
```
# .apply() is "applied" down columns by default, use axis = 1 for across columns
df.groupby(['Gender','No_show']).size().unstack('Gender').apply(lambda x: x/x.sum())
```

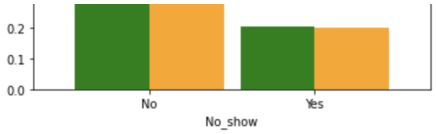
Gender	Female	Male
No_show		
No	0.796854	0.800321
Yes	0.203146	0.199679

annonymous function

x is the argument

return is x/sum of x's

In [27]:
```
# .apply() is "applied" down columns by default, use axis = 1 for across columns
(df.groupby(['Gender','No_show']).size()
                                 .unstack('Gender')
                                 .apply(lambda x: x/x.sum())
                                 .plot.bar(rot=0, width = 0.9, color = ['green','orange']));
```



Click On Images To Enlarge Them

The code is broken into steps, solely to highlight what each role each step is performing.

- That is, the code in the screenshot above could be reduce to 1 statement *(as you can see).*

This visualization shows you:

1. What percentage from **each category** are patients who **do show** *(No)*,
2. What percentage from **each category** are patients who **do not show** *(Yes)*,

As you can see, you could use the same code, changing only `Gender`, to analyze all of the other categorical variables. *(If you do this, it would be best to generate a function from that code, where the only argument is a placeholder for* `"Gender"` *)*

(Obviously, these approaches are something that you will pick up over time - hopefully you find this useful/interesting).

The code makes use of functions to avoid repetitive code. The code contains good comments and variable names, making it easy to read.

One of the requirement for this specification is:

> The code makes use of functions to avoid repetitive code.

Another of the requirements for this specification is:

> "The code contains good comments and variable names, making it easy to read."

There is very little commenting of code. It should sufficient for me to follow your code without reading much of it line-by-line *(especially complex code, or code that relates to decisions that you made during wrangling)*.

CHANGE REQUIRED (MINOR)

- Code comments should be added
- Include at least one function to replace repetitive code

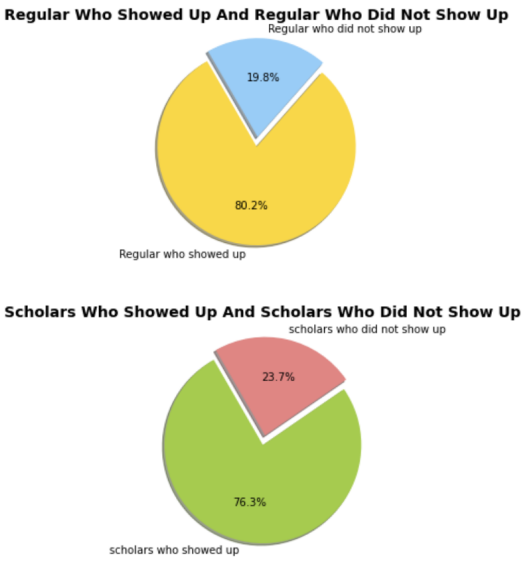
TIP

- The repetitive code forms the base of your function
- Compare the two cells were you repeat code
    - The only differences between the repeated code will become the arguments for your function
    - All of the rest of the code will remain the same
- Using **any** name (not already used) for your function that will help you remember what the function does
    - Do the same for the arguments

For example, you repeatedly use the same code for pie charts (where the only differences are the variables included):

```
[40]:
def myPieCharts(data,labels,colors):
    '''
    docstring: IMPORTANT ... explain function here
    '''
    # plot
    plt.pie(data, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=120)
    plt.title(f"{' and '.join(labels)}".title(),fontsize=14,weight='bold')
    plt.axis('equal')
```

```
[42]:
plt.figure(0)

# Data to plot
label1 = 'Regular who showed up', 'Regular who did not show up'
data1  = Bosnia.loc[False].values
color1 = ['gold','lightskyblue']
explode = (0.1, 0.0)  # explode 1st slice

myPieCharts(data1,label1,color1)

plt.figure(1)
# Data to plot
label2 = 'scholars who showed up', 'scholars who did not show up'
data2  = Bosnia.loc[True].values
color2 = ['yellowgreen','lightcoral']

myPieCharts(data2,label2,color2)
```



Click On Images To Enlarge Them

(where I use f-strings (link) which allows for string substitution (i.e. everything in `{}` *is a variable, but the string*

version will be included in the string, and `.join()` *to join the entries in a list as a string))*

As you can see above, that function can then be used repeatedly

---

ADDITIONAL NOTES

In data analysis, in a work environment, commenting code, so that your colleagues understand the intent of the code that you write, is a basic necessity.

- *(It is always more difficult to read other people's code).*
- Here is a summary of good commenting etiquette
- Here is a more detailed discussion, including a guide to commenting and a response to the idea that you do not need to comment code

Commenting code is a requirement for this project, and a good habit to develop, because it communicates to colleagues, or to yourself at some future time, the intent of the code that you have written (in a succinct way that avoids you having to examine the code line by line).

It is difficult to overstate the importance of clear code commenting, in a work environment, in data analysis.

To fix this, it is best to write brief comments for each chunk of code, and more detailed comments for complex code.

It takes about 5 minutes to comment a script like this, but it saves much more time for a colleague when they are reading your code.

Quality of Analysis

✓

The project clearly states one or more questions, then addresses those questions in the rest of the analysis.

You list four questions:

- Does Gender affect the rate of show up?
- Does the Bosnia Familia Scholarship affect the rate of show up?
- What is the effect of disability on show up?
- How was the bosnia scholarship distributed among patients?

at the start of your analysis and develop your analysis around those questions throughout your script. Very nice work!

Data Wrangling Phase

✓

The project documents any changes that were made to clean the data, such as merging multiple files,

handling missing values, etc.

Excellent work documenting the cleaning of your dataset. You have

1. Identified missing and duplicated data

2. Identified data that requires wrangling to be useful in your analysis (`genres` and `production_companies`) and/or would bias your analysis (revenue and budget values of 0)

Then you resolved both issue to prepare your data for analysis. Very nice work!

Important

Never hard-code values in a data analysis script.

Code that you write in data analyses will, typically, be applied to different versions of the same data. Any change to the data that your code is run on will make any hard-coded values invalid.

Instead, extract the values from the variables that calculated them:

In [130]:

```
plt.figure(0)

# Data to plot
label1 = 'Regular who showed up', 'Regular who did not show up'
data1  = [79924, 19741]
color1 = ['gold','lightskyblue']
explode = (0.0, 0.0)  # explode 1st slice

# Plot
plt.pie(data1, explode=explode, labels=label1, colors=color1, autopct='%1.1f%%', shadow=True, startangle=120)

plt.axis('equal')

plt.figure(1)

# Data to plot
label2 = 'scholars who showed up', 'scholars who did not show up'
data2  = [8283, 2578]
color2 = ['yellowgreen','lightcoral']
explode = (0.0, 0.0)  # explode 1st slice

# Plot
plt.pie(data2, explode=explode, labels=label2, colors=color2, autopct='%1.1f%%', shadow=True, startangle=120)

plt.axis('equal')

plt.show()
```

never hard code values
- any change to the original data (or to your cleaning process) will make those values invalid.

the Bosnia Familia Scholarship has an effect on rate of show up. Patients who do not have scholarship tend to show up more for their appointment.

UPDATED

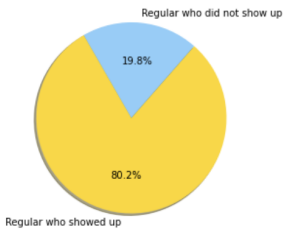
```
In [136]:
# use .loc[] to access each category
Bosnia.loc[False]

showed_up
True     79924
False    19741
Name: showed_up, dtype: int64
```

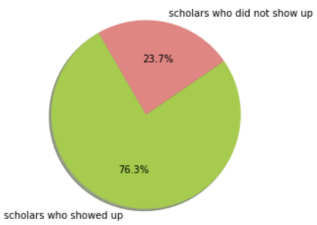
Instead, access the values from the variables that calculated them

```
In [137]:
# use .values to access the values
Bosnia.loc[False].values

array([79924, 19741])
```

```
In [138]:
Bosnia.loc[True].values

array([8283, 2578])
```

```
In [141]:
plt.figure(0)

# Data to plot
label1 = 'Regular who showed up', 'Regular who did not show up'
# data1  = [79924, 19741]
data1  = Bosnia.loc[False].values
color1 = ['gold','lightskyblue']
explode = (0.0, 0.0)  # explode 1st slice
# Plot
plt.pie(data1, explode=explode, labels=label1, colors=color1, autopct='%1.1f%%', shadow=True, startangle=120)
plt.axis('equal')

plt.figure(1)

# Data to plot
label2 = 'scholars who showed up', 'scholars who did not show up'
# data2  = [8283, 2578]
data2  = Bosnia.loc[True].values
color2 = ['yellowgreen','lightcoral']
explode = (0.0, 0.0)  # explode 1st slice

# Plot
plt.pie(data2, explode=explode, labels=label2, colors=color2, autopct='%1.1f%%', shadow=True, startangle=120)
plt.axis('equal')
plt.show()
```





Exploration Phase

The project investigates the stated question(s) from multiple angles. At least three variables are investigated using both single-variable (1d) and multiple-variable (2d) explorations.

You have explored your questions using 1d and 2d explorations using appropriate techniques that helped answer those questions *(showing the underlying distribution for each variable, prior to your 2d analyses)*. Very nice work!

The project's visualizations are varied and show multiple comparisons and trends. Relevant statistics are computed throughout the analysis when an inference is made about the data.

At least two kinds of plots should be created as part of the explorations.

You include a selection of visualizations that highlight interesting patterns, and perform a range of statistical analyses, that help to answer your research questions. Nice work!

ADDITIONAL NOTES

If you are interested in experimenting with different types of visualization, this site contains a gallery of most types of visualizations, *each example contains code that you can use as a template for your visualizations.*

One gallery that I suggest that you look at is for the Seaborn module (link). It is developed with the sole purpose for making aesthetically pleasing visualizations easily.

Also, here is an overview of a range of other modules for visualizations in *python*, which highlights the range of professional level graphical analysis you can apply using *python*.

TIP

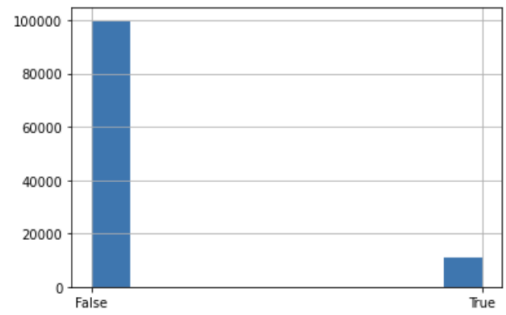
The `.hist()` function is not very good at displaying the distribution for **discrete** variables.

Instead, use panda's bar plotting function, along with the `.value_counts()` method

let's check how many patients have scholarship

In [128]:
```
1 print(df.scholarship.value_counts())
2 df.scholarship.astype(str).hist();
```

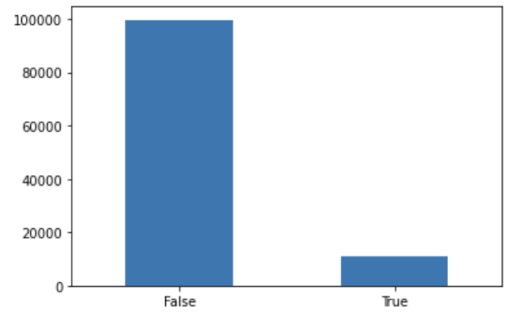
```
False    99665
True     10861
Name: scholarship, dtype: int64
```



In [144]:
```
1 print(df.scholarship.value_counts())
2 df.scholarship.value_counts().plot.bar(rot=0);
```

```
False    99665
True     10861
Name: scholarship, dtype: int64
```

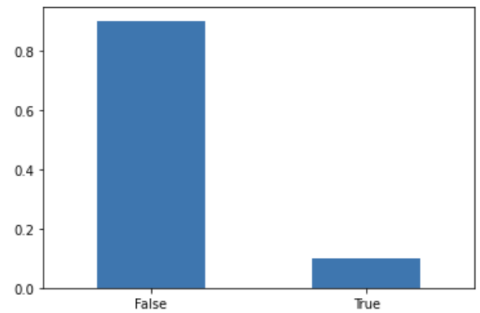
VS.



In [145]:
```
1 print(df.scholarship.value_counts(normalize=True))
2 df.scholarship.value_counts(normalize=True).plot.bar(rot=0);
```

relative frequency

```
False    0.901734
True     0.098266
Name: scholarship, dtype: float64
```



10861 patients have the scholarship.

let us now check for the percentage of patients with scholarship who showed up for their appointment

Also, as you can see, you can use the `normalize=True` option in `.value_counts()` to derive the relative frequency.

Conclusions Phase

The results of the analysis are presented such that any limitations are clear. The analysis does not state or imply that one change causes another based solely on a correlation.

Throughout your analysis, you present *(very interesting)* visualizations and statistics. However, there is little discussion of the limitations of your analysis.

CHANGE REQUIRED (VERY MINOR)

- In your conclusion, add a discussion of the limitations of drawing broader conclusions from the results of your analysis.

ADDITIONAL NOTES

One of the requirements for this specification is:

> "The results of the analysis are presented such that any limitations are clear. "

The limitations of any data analysis are:

- Issues with the dataset
  - we *(almost always)* are working with sample data, which implies uncertainty (a different sample may lead to different results)
  - we *(almost always)* find issues with the sample of data that we are working with (missing observations, data that appears to be inconsistent

Other limitations involve:

- Issues with the methods of analysis
  - Typically, a type of analysis that we would like to perform is not possible due to data limitations.

Added to these, there is no statistical inference performed in your analysis (to test the significance of the results that you found). *While this is not required*, it is still a limitation of your analysis.

Communication

Reasoning is provided for each analysis decision, plot, and statistical summary.

Your commentary makes it easy to follow your thought processes as you work through your data exploration. It is clear and concise. The questions that you answer throughout your report uncovers interesting patterns in the data. Excellent work!

Visualizations made in the project depict the data in an appropriate manner that allows plots to be readily interpreted.

The requirement for this specification is:

> "Visualizations made in the project depict the data in an appropriate manner that allows plots to be readily interpreted."

When communicating information using visualizations, the visualizations have to be *self-contained*. That is, when a reader looks at the visualization:

1. Do they have all of the information required to interpret the visualization?
2. Is the visualization the correct type (different plot types are appropriate for different data, allowing the patterns in the data to be easily interpreted)?

CHANGE REQUIRED (VERY MINOR)

- Add labels and titles to your plots, so that the reader can interpret the plot
- Include units of measurement in label axes, where relevant

ADDITIONAL NOTES

TITLES AND LABELS

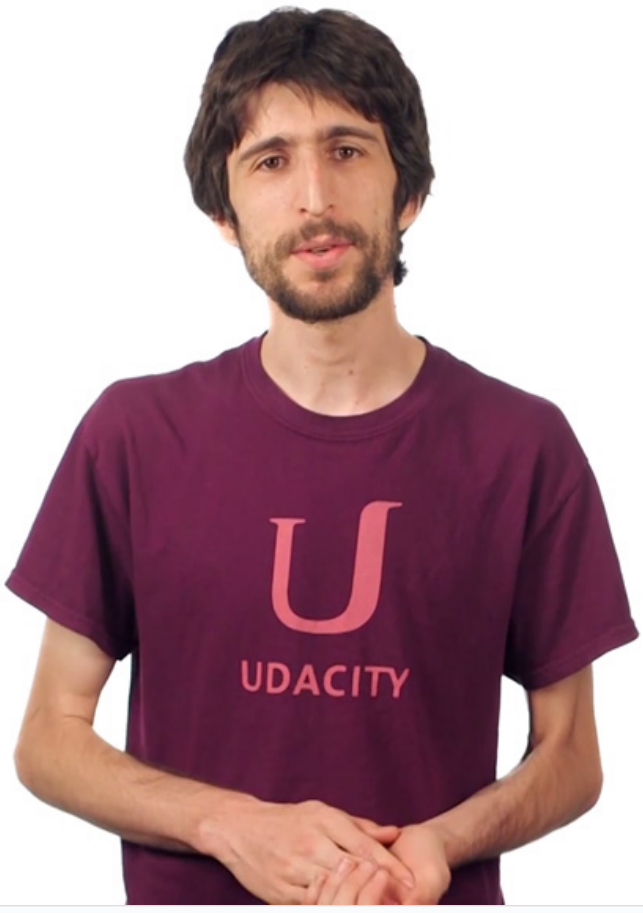
It is **much** easier to interpret visualizations if you include titles and labels:

```
#### rest of plot code
```

```
# Add title and format it
plt.title('Title'.title(),
                fontsize = 14, weight = "bold")
# Add x label and format it
plt.xlabel('X_label'.title(),
                fontsize = 10, weight = "bold")
# Add y label and format it
plt.ylabel('Y_Label'.title(),
                fontsize = 10, weight = "bold")
# (If there is a legend, you can use this) Change legend title and format it, move l
egend
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5),
           title="Legend_Title", title_fontsize = 12);
```

RESUBMIT

DOWNLOAD PROJECT



Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review

START