UDACITY

Return to Classroom

Investigate a Dataset

REVIEW

HISTORY

Requires Changes

5 specifications require changes

This is an excellent first submission for this project *(The bulk of the work has been completed, only minor issues remain to be resolved to meet all requirements.* ***Note: This review contains a lot of suggestions. Suggestions are not required changes. The required changes are listed clearly****)*.

- Extensive analysis of the factors affecting attendance has been performed throughout your report. Kudos!
- You take a complex dataset and reveal some interesting, and unusual, patterns *(for example: "As people get older, they tend to show up less often for their appointments. However, it's essential to note any outliers in the data that could skew the analysis")*.

Excellent work!

However, *reviewers are ruled by the* ***Rubric*** and there are still main requirements that have not been met. The main issues to be addressed relate to:

- Include both a notebook and a HTML file (both have to be completed, with results showing)
- Include an introduction, with clearly stated questions

- Include *at least* one function in your script, **to replace repetitive coding**.
- There are **two parts** to this project. ***Answering research questions*** and:
  - "Variables are investigated using **both** single-variable (1d) and multiple-variable (2d) explorations."
  - Ensure that **variables** are investigated using **both** single-variable (1d) **and** multiple-variable (2d) explorations.
- Document (i.e. explain) in your commentary the wrangling decisions that you made.

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

Required Changes are typical *(both here at **Udacity**, or in a work environment)*. They are not a reflection of the quality of your work, they merely signify that the submission is not *exactly* in sync with the requirements. *(They do not affect your progress, and they are not included in your records.)*

- 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. All other Notes and Tips are included as guides, to help make the required changes.***
- Sections marked `Notes` 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 solely provided as **more detailed guides or explanations**). If no change is required, these are **optional**.*

Code Functionality

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

This Section Requires Changes

CRITERIA: ALL CODE IS FUNCTIONAL AND PRODUCES NO ERRORS WHEN RUN.

CRITERIA: THE CODE GIVEN IS SUFFICIENT TO REPRODUCE THE RESULTS DESCRIBED.

There is no notebook included in your submission.

This file is required.

1. **There should be a completed *data exploration* notebook (or *python* scripts), that you used to create your report, included in your submission *(Not Included)*** , and
2. **There should also be a HTML/pdf file *(Included)***

*(Where **"completed"** means that all rubric items have been attempted).*

Both files have to be included in all submissions *(As each submission is reviewed independently by different reviewers).*

- That is, you create a zip file of the folder that contains **both your HTML/pdf file and notebook**. For macos, just right-click on the folder and select `Compress`, for Windows, follow these instructions. And then submit the .zip file

---

Here are the files in your submission:

Investigate_a_Dataset.html

Here are the submission requirements:

Submission

What to include in your submission

1. A PDF or HTML file containing your analysis. This file should include:
   - A note specifying which dataset you analyzed
   - A statement of the question(s) you posed
   - A description of what you did to investigate those questions
   - Documentation of any data wrangling you did
   - Summary statistics and plots communicating your final results
2. Code you used to perform your analysis. If you used a Jupyter notebook, you can submit your `.ipynb`. Otherwise, you should submit the code separately in `.py` file(s).
3. A list of Web sites, books, forums, blog posts, github repositories, etc. that you referred to or used in creating your submission (add N/A if you did not use any such resources).

ADDITIONAL NOTES

- To help develop your skills using python, I highly recommend working through the examples in this free online text
- For guides for most aspects of the *Python Programming Language* this site is a great resource
- Finally, a guide to developing good coding habits in python

IMPORTANT

All of the results have to be **visible** in the notebook and HTML files submitted:

> **Tip**: Welcome to the Investigate a Dataset project! You will find tips in quoted sections like this to help organize your approach to your investigation. Once you complete this project, remove these **Tip** sections from your report before submission. First things first, you might want to double-click this Markdown cell and change the title so that it reflects your dataset and investigation.

Project: Investigate a Dataset - [No-Show Appt 2016]

Table of Contents

- Introduction
- Data Wrangling
- Exploratory Data Analysis
- Conclusions

Introduction

Dataset Description

> **Tip**: In this section of the report, provide a brief introduction to the dataset you've selected/downloaded for analysis. Read through the description available on the `Project Details` page of `Investigate a Dataset` lesson for this course. List all column names in each table, and their significance. In case of multiple tables, describe the relationship between tables.

Question(s) for Analysis

> **Tip**: Clearly state one or more questions that you plan on exploring over the course of the report. You will address these questions in the **data analysis** and **conclusion** sections. Try to build your report around the analysis of at least one dependent variable and three independent variables. If you're not sure what questions to ask, then make sure you familiarize yourself with the dataset, its variables and the dataset context for ideas of what to explore.

> **Tip**: Once you start coding, use NumPy arrays, Pandas Series, and DataFrames where appropriate rather than Python lists and dictionaries. Also, **use good coding practices**, such as, define and use functions to avoid repetitive code. Use appropriate comments within the code cells, explanation in the mark-down cells, and meaningful variable names.

In [ ]:
```
# Use this cell to set up import statements for all of the packages that you
#   plan to use.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# While optional for current notebooks, if you are having trouble with visualizations,
#   remember to include a 'magic word' so that your visualizations are plotted
#   inline with the notebook. See this page for more:
#   http://ipython.readthedocs.io/en/stable/interactive/magics.html
%matplotlib inline
```

Data Wrangling

> **Tip**: In this section of the report, you will load in the data, check for cleanliness, and then trim and clean your dataset for analysis. Make sure that you **document your data cleaning steps in mark-down cells precisely and justify your cleaning decisions.**

General Properties

> **Tip**: You should *not* perform too many operations in each cell. Create cells freely to explore your data. One option that you can take with this project is to do a lot of explorations initially. This does not have to be organized, but make sure you use enough comments to understand the purpose of each code cell. Then, after you're done with your analysis, trim the excess and organize your steps so that you have a flowing, cohesive report.

In [ ]:
```
# Load your data and print out a few lines. What is the size of your dataframe?
#   Perform operations to inspect data types and look for instances of missing
#   or possibly errant data. There are at least 4 – 6 methods you can call on your
```

> **Important:** *After running all of the code in the notebook,* `Save and Checkpoint` your notebook - under `File` in the notebook menu (the saved notebook and HTML file is made from the last saved state of your notebook).

- **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.**

This Section Meets Specifications

✓ CRITERIA: THE PROJECT USES NUMPY ARRAYS AND PANDAS SERIES AND DATAFRAMES WHERE APPROPRIATE RATHER THAN PYTHON LISTS AND DICTIONARIES.

✓ CRITERIA: WHERE POSSIBLE, VECTORIZED OPERATIONS AND BUILT-IN FUNCTIONS ARE USED INSTEAD OF LOOPS.

Where appropriate, your project uses and Pandas Series and DataFrames, rather than Python lists and dictionaries. Great work!

Pandas' *built-in functions*, which use vectorized operations, instead of loops, are used where possible. Great job!

IMPORTANT *(THESE TIPS WILL HELP YOU TO ANSWER YOUR QUESTIONS WITH LESS, AND MORE EFFICIENT, CODE)*

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.

GROUPBY AND VALUE COUNTS

The most important vectorized method for *discrete variables* is *.groupby()*

- It groups the observations for *each* category in the variable that you are grouping by.
- You can then apply a function to those groups.

.value_counts() will count the number of observations in each category.

- To get the proportions, you can use the option *normalize=True*

57]:
```
# groupby gender
# count no-show
df.groupby(['Gender'])['No-show'].value_counts()
```

57]:
```
Gender  No-show
F       No         57245
        Yes        14594
M       No         30962
        Yes         7725
Name: No-show, dtype: int64
```

58]:
```
# groupby gender
# proportion no-show
df.groupby(['Gender'])['No-show'].value_counts(normalize=True)
```

58]:
```
Gender  No-show
F       No         0.796851
        Yes        0.203149
M       No         0.800321
        Yes        0.199679
Name: No-show, dtype: float64
```

Click On Images To Enlarge Them

PLOTTING GROUPBY STATEMENTS

Another 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:

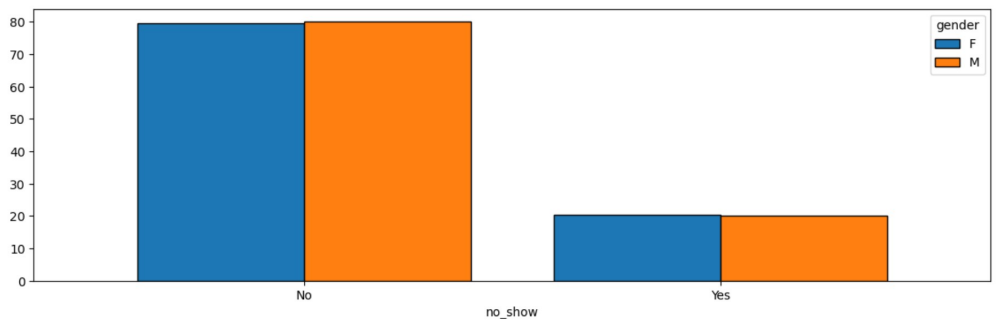
```
In [54]: # groupby gender, get count for show/no show
         df.groupby('gender')['no_show'].value_counts()

Out[54]: gender  no_show
         F       No         57245
                 Yes        14594
         M       No         30962
                 Yes         7725
         Name: no_show, dtype: int64
```

```
]: # groupby gender, get proportion show/no show, mulitply by 100 for percentages
   df.groupby('gender')['no_show'].value_counts(normalize=True).mul(100)

]: gender  no_show
   F       No         79.685129
           Yes        20.314871
   M       No         80.032052
           Yes        19.967948
   Name: no_show, dtype: float64

]: # Unstack
   df.groupby('gender')['no_show'].value_counts(normalize=True).mul(100).unstack('gender')
```

Not Required
(ONLY included to show
what the steps in the plot
code are doing)

gender	F	M
no_show		
No	79.685129	80.032052
Yes	20.314871	19.967948

```
]: # plot
   df.groupby('gender')['no_show'].value_counts(normalize=True).mul(100).unstack('gender').plot.bar(edgecolor='black',figsize=[14, 4], rot=0, width=0.8);
```



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 reduced 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"` *)*

RELEVANCE TO YOUR REPORT

counts obviously offer important information. However, they can be difficult/impossible to interpret quickly *when comparing categories* if there is a large difference in the numbers in each group.

proportions/percentages are easy to interpret quickly:

For example, for the SMS analysis:

1.4.1 Research Question 1 (Does sending SMS reduce or increase no-shows?)

[26]:
```
# Use this, and more code cells, to explore your data. Don't forget to add
#   Markdown cells to document your observations and findings.
df.groupby(["NoShow", "SMS_Received"]).size()
```

t[26]:
```
NoShow  SMS_Received
No      False          62509
        True           25698
Yes     False          12535
        True            9784
dtype: int64
```

[33]:
```
# Use this, and more code cells, to explore your data. Don't forget to add
#   Markdown cells to document your observations and findings.
df.groupby(["SMS_Received"])["NoShow"].value_counts(normalize=True).mul(100).unstack()
```

t[33]:

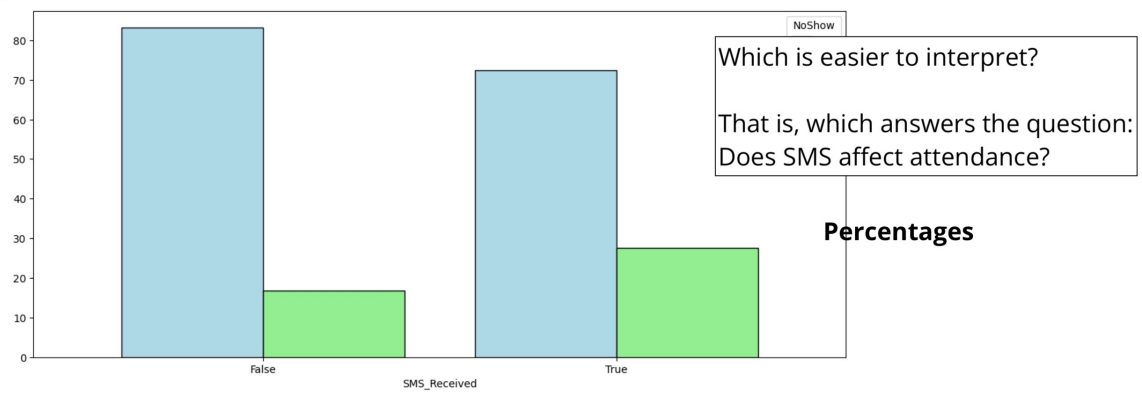
NoShow	No	Yes
SMS_Received		
False	83.296466	16.703534
True	72.425455	27.574545

1.4.2 Analysis: This shows that there are 35422 patients that received sms and only 9784 actu

Click On Images To Enlarge Them

and

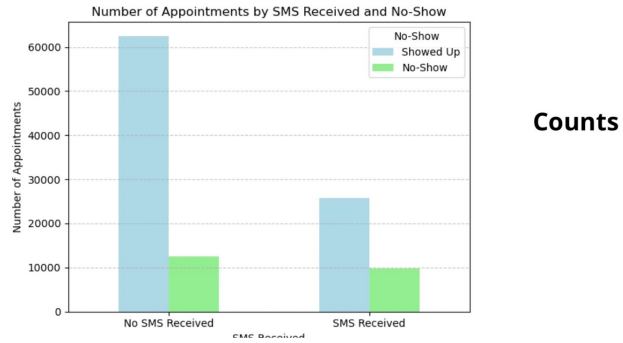
[35]:
```
# Use this, and more code cells, to explore your data. Don't forget to add
#   Markdown cells to document your observations and findings.
df.groupby(["SMS_Received"])["NoShow"].value_counts(normalize=True).mul(100).unstack().plot.bar(edgecolor='black',figsize=[14, 6],rot=0,width=0.8,color=['lightblue', 'lightgreen']);
```



1.4.2 Analysis: This shows that there are 35422 patients that received sms and only 9784 actually showed up for their appts while 25698 did not show up.

[27]:
```
# Group data by 'SMS_Received' and 'NoShow', and count the occurrences
sms_no_show_counts = df.groupby(['SMS_Received', 'NoShow']).size().unstack(fill_value=0)

# Plot the grouped bar plot
plt.figure(figsize=(8, 6))
sms_no_show_counts.plot(kind='bar', color=['lightblue', 'lightgreen'])
plt.title('Number of Appointments by SMS Received and No-Show')
plt.xlabel('SMS Received')
plt.ylabel('Number of Appointments')
plt.xticks([0, 1], ['No SMS Received', 'SMS Received'], rotation=0)
plt.legend(title='No-Show', labels=['Showed Up', 'No-Show'])
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



Counts

Click On Images To Enlarge Them

As you can see, *percentages are easy to interpret,* there is no quick way to interpret the counts.

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

ADDITIONAL NOTES

The popularity of *Pandas* in data analysis is a result of two features: It has a simple but flexible data representation; and It provides a toolbox of in-built methods that allows you to easily analyze your data (which are efficient and easy to use).

- Understanding Vectorization in NumPy and Pandas
- How to Speed Up Pandas Data Operations Using Vectorized Operations

- **The code makes use of at least 1 function to avoid repetitive code.**
- **The code contains good comments and meaningful variable names, making it easy to read.**

This Section Requires Changes

CRITERIA: THE PROJECT USES FUNCTIONS AND LOOPS WHERE POSSIBLE TO REDUCE REPETITIVE CODE. COMMENTS AND DOCSTRINGS ARE USED AS NEEDED TO DOCUMENT CODE FUNCTIONALITY.

- The comments in your code, and in your Markdown that relate to the code, makes it easy to follow your code. Very nice work!

One of the requirements for this specification is:

> *"The code makes use of **at least 1** function to **avoid repetitive code.**"*

You don't use a function in your code. There is significant repetitive coding that would benefit from using functions.

CHANGE(S) REQUIRED

- Include at least one function to replace repetitive code

TIPS

Overtime, you will build up a library of functions- eventually creating your own modules to load them from external scripts - that you will use across different work projects. Using them will free up time to focus on other details of your analysis.

- 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
- This guide gives an excellent overview of creating functions in python

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

```
[63]: def myValueCountsPlot(df,gVar,normalize=True,mul=100, xticks=['Without', 'With'],color=['lightblue', 'lightgreen'], index=None):
          '''
          docstring: IMPORTA[illegible]ction here (not completed here)
          inputs: ...
          output: ...
          '''
          # plot
          df.groupby(gVar)['NoShow'].value_counts(normalize=normalize).mul(mul).unstack().reindex(index=index).plot.bar(edgecolor='black',figsize=[14, 4],rot=0,width=0.8, color=color);
          # OPTIONAL: [illegible] plotting, clean up variable names
          gVar=gVar.replace("_"," ") # replace _ with a space
          # Add title and format it
          plt.title(f'Effect of {gVar} on Patient Attendance'.title(), fontsize = 14, weight = "bold")
          # Add x label and for[illegible]
          plt.xlabel(gVar.title(), fontsize = 10, weight = "bold")
          # Add y la[illegible] and format it
          plt.ylabel(f'{"Percentage" if normalize else "Frequency"}'.title(), fontsize = 10, weight = "bold")
          # xticks
          plt.xticks(plt.xticks()[0], [f'{tick} {gVar}' for tick in xticks])
```

placeholder

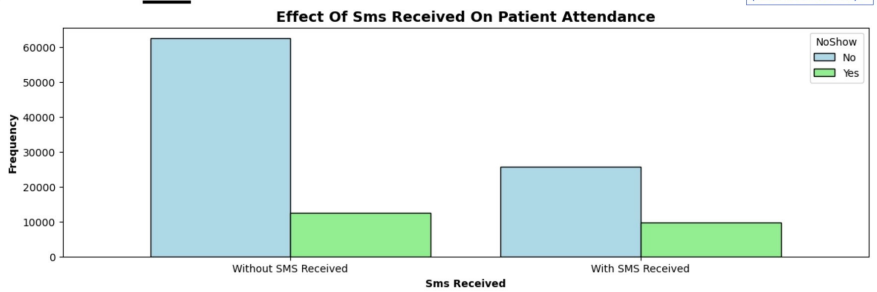
arguments with = are optional (their defaults are set)

plot

title/ labels

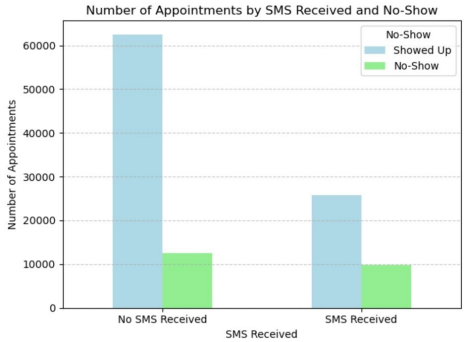
```
[64]: # call plotting function
      myValueCountsPlot(df,'SMS_Received',normalize=False,mul=1)
```

same plot, with placeholders (and title/labels)



1.4.2 Analysis: This shows that there are 35422 patients that received sms and only 9784 actually showed up for their appts while 25698 did not show up.

```
[45]: # Group data by 'SMS_Received' and 'NoShow', and count the occurrences
      sms_no_show_counts = df.groupby(['SMS_Received', 'NoShow']).size().unstack(fill_value=0)

      # Plot the grouped bar plot
      plt.figure(figsize=(8, 4))
      sms_no_show_counts.plot(kind='bar', color=['lightblue', 'lightgreen'])
      plt.title('Number of Appointments by SMS Received and No-Show')
      plt.xlabel('SMS Received')
      plt.ylabel('Number of Appointments')
      plt.xticks([0, 1], ['No SMS Received', 'SMS Received'], rotation=0)
      plt.legend(title='No-Show', labels=['Showed Up', 'No-Show'])
      plt.grid(axis='y', linestyle='--', alpha=0.7)
      plt.tight_layout()
      plt.show()

      <Figure size 800x400 with 0 Axes>
```

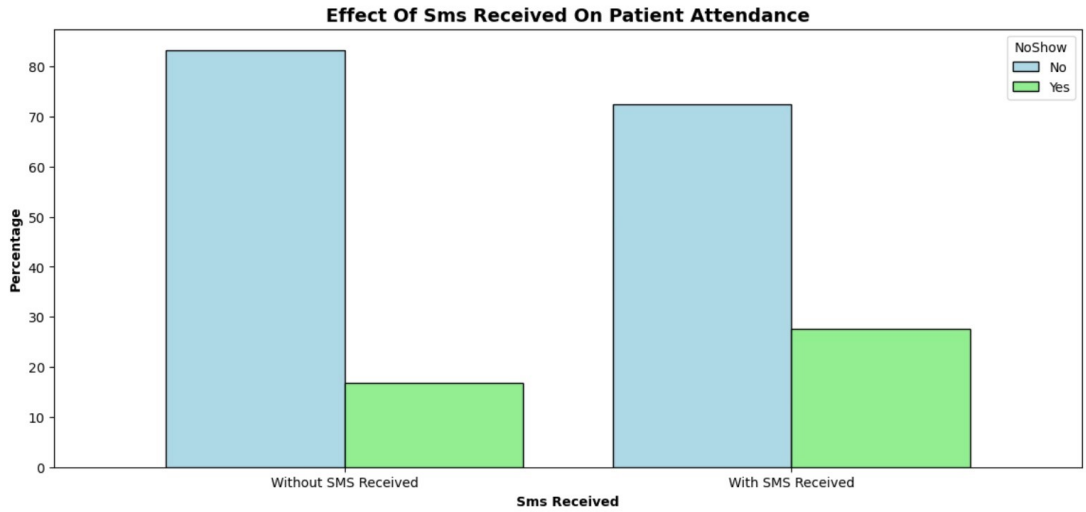


Click On Images To Enlarge Them

(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). ***Important*** *The outer quotes have to differ from quotes used inside the f-string (example: If you use a single quote to* ***create*** *the f-string, double quotes have to be used* ***inside*** *the f-string). Also, here is a useful cheatsheet for formating numbers using f-strings)*

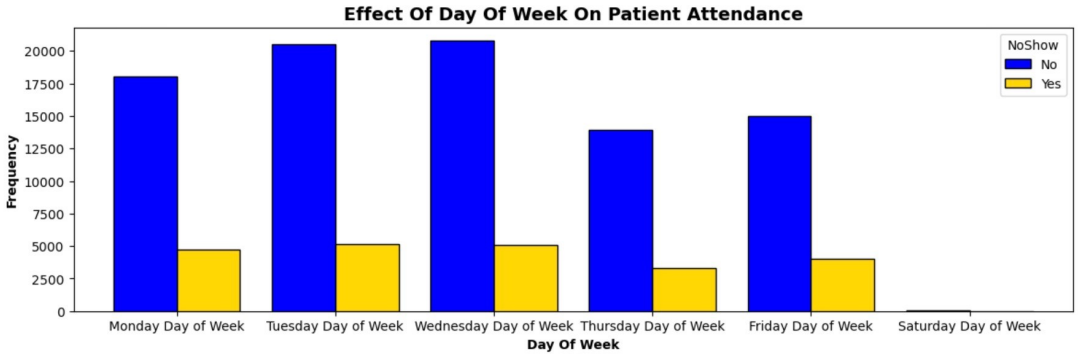
That function can then be used repeatedly, changing the *arguments when you call the function*, as noted earlier, you can create a **percentage** version of the plots above, using the option `normalize=True` (times 100 for percentages):

[2]:
```
# call plotting function
myValueCountsPlot(df,'SMS_Received')
```

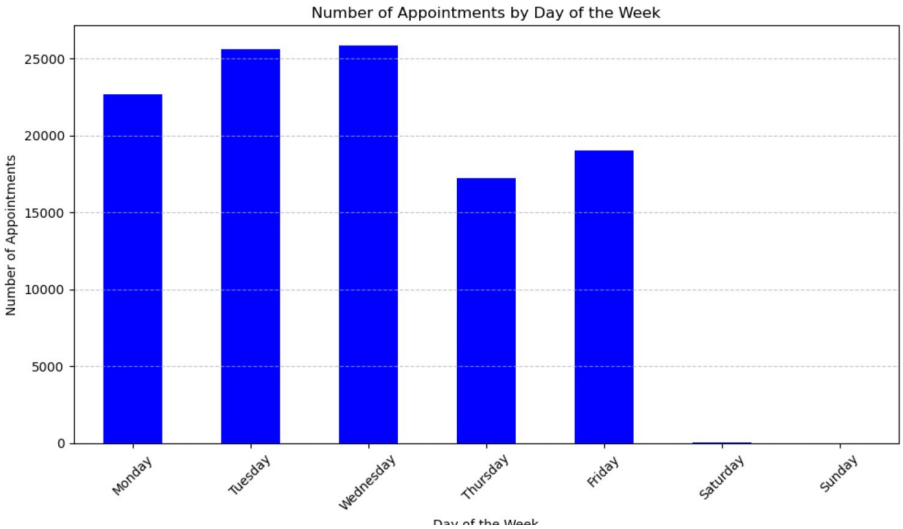


*and the same for **day of the week:***

[67]:
```
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

# call plotting function
myValueCountsPlot(df,'Day_of_Week',index=day_order, xticks=day_order, color=['b','gold'], normalize=False, mul=1)
```

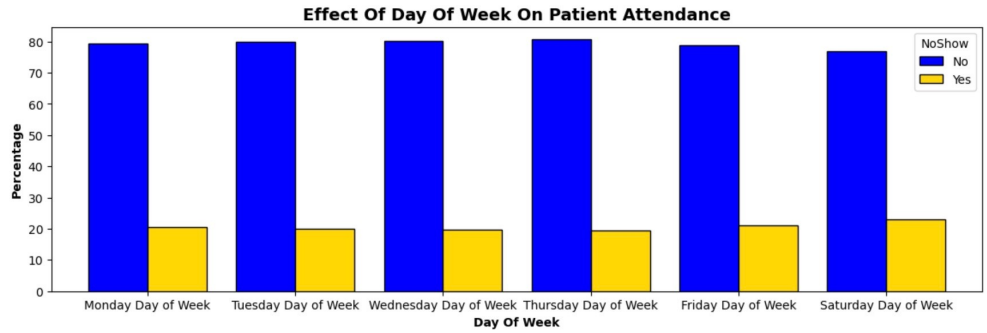


[29]:
```
# Plot the count of appointments for each day of the week
plt.figure(figsize=(10, 6))
appointment_counts.plot(kind='bar', color='blue')
plt.title('Number of Appointments by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

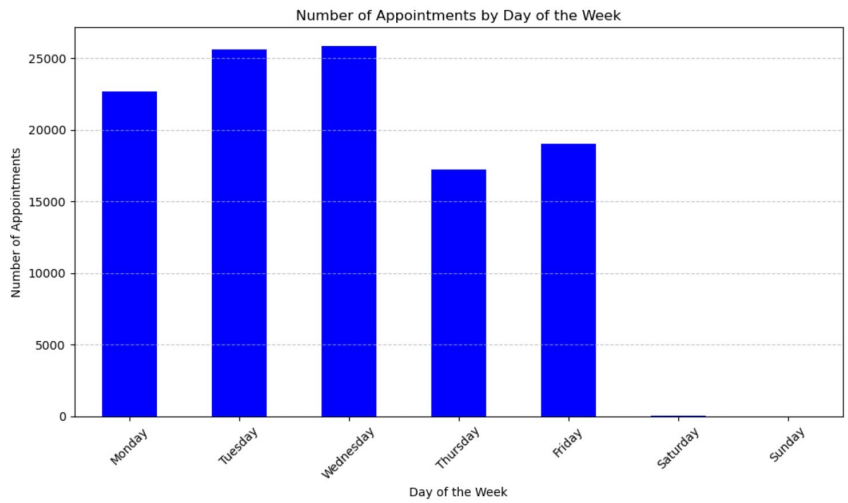


*and the **percentage** version:*

[66]:
```
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

# call plotting function
myValueCountsPlot(df,'Day_of_Week',index=day_order, xticks=day_order, color=['b','gold'])
```



[29]:
```
# Plot the count of appointments for each day of the week
plt.figure(figsize=(10, 6))
appointment_counts.plot(kind='bar', color='blue')
plt.title('Number of Appointments by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



*Obviously, this is **a tip/suggestion**, you can include any function that **replaces repetitive coding**.*

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 only takes a few minutes to clearly comment code like this but the benefits are substantial. Overtime you will built up a library of scripts/notebook that you will refer back to. With clearly commented code, you can scan those resources quickly.
    - Commenting code is 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.

    - 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

Again, nice work!!

- If you want to explore functions in python:
    - An *Udacity* guide to functions, with examples
    - Another introduction to functions, with examples
    - This free online text/tutorial provides a *fun* and ***practical*** approach to help you to develop this area of your coding skills
    - Here is a simple overview of functions

Quality of Analysis

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

This Section Requires Changes

CRITERIA: THE PROJECT CLEARLY STATES ONE OR MORE QUESTIONS, THEN ADDRESSES THOSE QUESTIONS IN THE REST OF THE ANALYSIS.

To meet this requirement, **clearly stated** research questions have to be included. *Currently, there are no research questions included in the introduction to your submission.*

- Also, in the report, there is a generic question that is too general to help communicate what direction your analysis will take.

At the start of your report, include clearly stated questions. You then develop your analysis around those questions.

CHANGE(S) REQUIRED

- Include an introduction
- Include **all questions** in your introduction.
- Include **one question** for each **different** phase of your analysis.

IMPORTANT

Remove all of the `Tip` sections - ***replacing them*** with your commentary *(all of those sections are **required**).* They are directed at you, the creator, not the reader of your report *(given that you are writing a report, to add to your portfolio of projects, that is evidence of your skills as a **Data Analyst**, the report should be replicate one typical of a work project **(without tips or suggestions)***

> **Tip**: Welcome to the Investigate a Dataset project! You will find tips in quoted sections like this to help organize your approach to your investigation. Once you complete this project, remove these **Tip** sections from your report before submission. First things first, you might want to double-click this Markdown cell and change the title so that it reflects your dataset and investigation.

1 Project: Investigate a Dataset - [No-Show Appt 2016]

1.1 Table of Contents

- Introduction
- Data Wrangling
- Exploratory Data Analysis
- Conclusions

1.2 Introduction

1.2.1 Dataset Description

> **Tip**: In this section of the report, provide a brief introduction to the dataset you've selected/downloaded for analysis. Read through the description available on the `Project Details` page of `Investigate a Dataset` lesson for this course. List all column names in each table, and their significance. In case of multiple tables, describe the relationship between tables.

1.2.2 Question(s) for Analysis

> **Tip**: Clearly state one or more questions that you plan on exploring over the course of the report. You will address these questions in the **data analysis** and **conclusion** sections. Try to build your report around the analysis of at least one dependent variable and three independent variables. If you're not sure what questions to ask, then make sure you familiarize yourself with the dataset, its variables and the dataset context for ideas of what to explore.

> **Tip**: Once you start coding, use NumPy arrays, Pandas Series, and DataFrames where appropriate rather than Python lists and dictionaries. Also, **use good coding practices**, such as, define and use functions to avoid repetitive code. Use appropriate comments within the code cells, explanation in the mark-down cells, and meaningful variable names.

```
[3]: # Use this cell to set up import statements for all of the packages that you
```

Click On Images To Enlarge Them

ADDITIONAL NOTES

Research questions are an important element of the research process.

By defining exactly what you, as a researcher, are interested in answering, these questions influence most of the rest of the steps that you take to conduct the research.

Importantly, they also inform the reader, giving them a *road map* for the direction that your report will take.

- A Simple Strategy For Asking Your Data The Right Questions

- Research question: the importance of your research question

Data Wrangling Phase

The project documents any changes that were made to clean the data, such as merging multiple files, handling missing values, etc.

This Section Requires Changes

CRITERIA: THE PROJECT DOCUMENTS ANY CHANGES THAT WERE MADE TO CLEAN THE DATA, SUCH AS MERGING MULTIPLE FILES, HANDLING MISSING VALUES, ETC.

You have wrangled your dataset. Your code

1. Identified missing/erroneous and duplicated data
2. Identified data that requires wrangling to be useful in your analysis and/or would bias your analysis

Then you resolved both issue to prepare your data for analysis. However, part of this requirement is to explain/document the steps that you took when cleaning your data.

CHANGE(S) REQUIRED

Add commentary, *in Markdown*, explaining the each phase of cleaning your dataset.

TIPS

Commentary has to be added to your report, explaining the steps that you took when wrangling the data (as **Markdown**):

`No Wrangling documentation`

```
Neighbourhood     0
Scholarship       0
Hipertension      0
Diabetes          0
Alcoholism        0
Handcap           0
SMS_received      0
No-show           0
dtype: int64
```

In [10]:
```
#Check for Duplicates
sum(df.duplicated())
```

Out[10]: 0

In [11]:
```
#Check to see range of ages
df.Age.max(), df.Age.min()
```

Out[11]: (115, -1)

1.3.2 Data Cleaning

> **Tip**: Make sure that you keep your reader informed on the steps that you are taking in your investigation. Follow every code cell, or every set of related code cells, with a markdown cell to describe to the reader what was t
> understand what they will be seeing in the following cell(s).

In [12]:
```
# After discussing the structure of the data and any problems that need to be
#   cleaned, perform those cleaning steps in the second part of this section.

#Query may not have been written properly or not registering
df.query('Age =="-1"')
```

Out[12]:

PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show

In [13]:
```
#Use python as my Query line is not working
filtered_df = df[df['Age'] == -1]

print(filtered_df)
```

```
          PatientId  AppointmentID Gender          ScheduledDay  \
99832  4.659432e+14        5775010      F  2016-06-06T08:58:13Z

             AppointmentDay  Age Neighbourhood  Scholarship  Hipertension  \
99832  2016-06-06T00:00:00Z   -1         ROMÃO            0             0

       Diabetes  Alcoholism  Handcap  SMS_received No-show
99832         0           0        0             0      No
```

In [14]:
```
#Drop -1 age record
df.drop(df.index[99832], inplace=True)
```

In [15]:
```
#Check to see if collumn dropped from 110527 to 110526
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 110526 entries, 0 to 110526
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   PatientId       110526 non-null  float64
 1   AppointmentID   110526 non-null  int64
 2   Gender          110526 non-null  object
 3   ScheduledDay    110526 non-null  object
 4   AppointmentDay  110526 non-null  object
 5   Age             110526 non-null  int64
 6   Neighbourhood   110526 non-null  object
 7   Scholarship     110526 non-null  int64
 8   Hipertension    110526 non-null  int64
 9   Diabetes        110526 non-null  int64
 10  Alcoholism      110526 non-null  int64
 11  Handcap         110526 non-null  int64
 12  SMS_received    110526 non-null  int64
 13  No-show         110526 non-null  object
dtypes: float64(1), int64(8), object(5)
memory usage: 12.6+ MB
```

In [16]:
```
#recheck the Age max and min
df.Age.max(), df.Age.min()
```

Out[16]: (115, 0)

In [17]:
```
#How many people in DS
df.PatientId.nunique()
```

Out[17]: 62298

In [18]:
```
#how many Appt ID are there? (is same as quantity of rows)
df.AppointmentID.nunique()
```

Out[18]: 110526

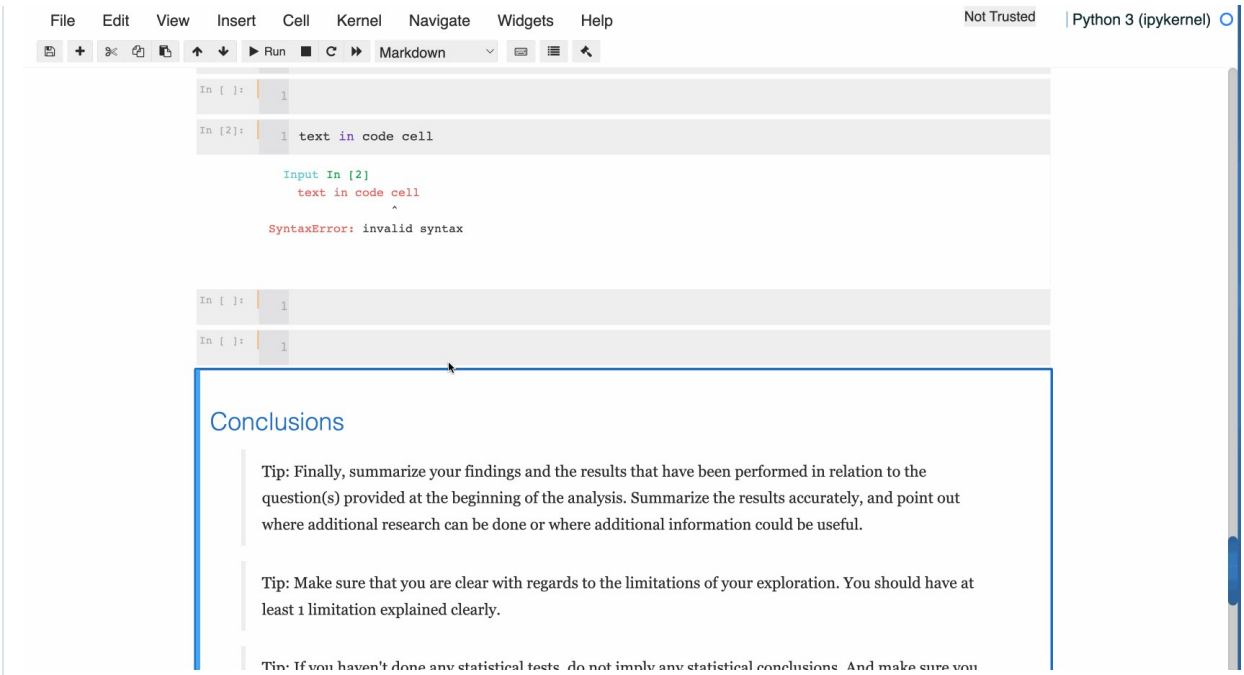
In [19]:
```
#Check Data first 5
df.head()
```

Out[19]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM DA PENHA	0	1	0	0	0	0	No
1	5.589978e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	0	0	0	0	0	No
2	4.262962e+12	5642549	F	2016-04-29T16:19:04Z	2016-04-29T00:00:00Z	62	MATA DA PRAIA	0	0	0	0	0	0	No
3	8.679512e+11	5642828	F	2016-04-29T17:29:31Z	2016-04-29T00:00:00Z	8	PONTAL DE CAMBURI	0	0	0	0	0	0	No
4	8.841186e+12	5642494	F	2016-04-29T16:07:23Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	1	1	0	0	0	No

- Markdown cells can be created (select **Cell** from the notebook menu, then select **Cell Type**)
  - Using *Markdown* syntax leads to report that is much easier to read. Here is a *""cheatsheet""* for basic markdown syntax (which shows the result of using that syntax)
  - Here is a guide to adding commentary to a *Jupyter Notebook*

*Here is an example of creating a **Markdown** cell in a notebook:*



All errors have to be resolved in order to pass this section.

ADDITIONAL NOTES

Any results that you generate from your analysis are only as valid as the data that you use is.

This highlights why data cleaning/wrangling is a fundamental feature of any data analysis.

Data wrangling seeks to ensure that your insights are valid by ensuring that data is reliable before you begin your analysis.

- Data Wrangling in Python
- Python - Data Wrangling

Exploration Phase

- The project investigates the stated question(s) from multiple angles.
- The project explores at least three variables in relation to the primary question. This can be an exploratory relationship between three variables of interest, or looking at how two independent variables relate to a single dependent variable of interest.
- The project performs both single-variable (1d) and multiple-variable (2d) explorations.

This Section Requires Changes

✓ CRITERIA: THE PROJECT INVESTIGATES THE STATED QUESTION(S) FROM MULTIPLE ANGLES.

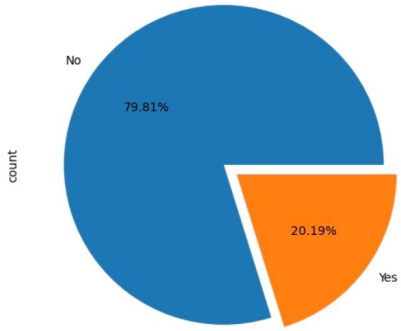
✓ CRITERIA: THE PROJECT EXPLORES AT LEAST THREE VARIABLES IN RELATION TO THE PRIMARY QUESTION. THIS CAN BE AN EXPLORATORY RELATIONSHIP BETWEEN THREE VARIABLES OF INTEREST, OR LOOKING AT HOW TWO INDEPENDENT VARIABLES RELATE TO A SINGLE DEPENDENT VARIABLE OF INTEREST.

☞ CRITERIA: THE PROJECT PERFORMS BOTH SINGLE-VARIABLE (1D) AND MULTIPLE-VARIABLE (2D) EXPLORATIONS.

You have explored your questions from multiple angles, using appropriate techniques to explore at least three variables that helped answer your research questions. Very nice work!

However, including **both** 1d and 2d explorations is required to meet this requirement.

[68]:
```
df['NoShow'].value_counts().plot(kind='pie',figsize=(14,6),explode=[0,0.1], autopct='%1.2f%%');
```

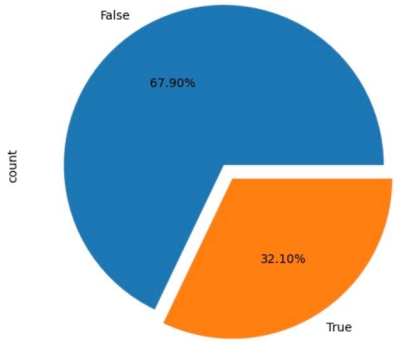


1d

Distribution of a single variable

To answer each of your questions, you will include multiple ways (graphically or statistically) to explore the data.
Beginning with a Univariate (1d) exploration, to explain the individual variables,
then moving on to your Multivariate (or 2d) explorations

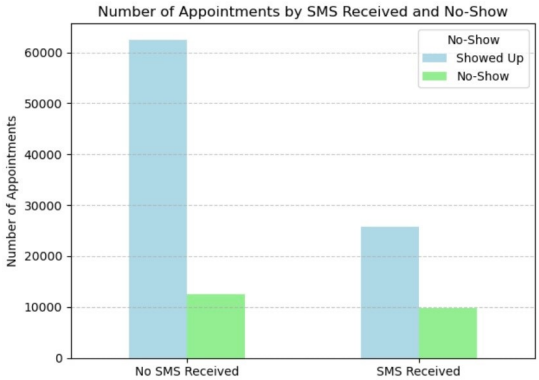
[69]:
```
df['SMS_Received'].value_counts().plot(kind='pie',figsize=(14,6),explode=[0,0.1], autopct='%1.2f%%');
```



1.4.2 Analysis: This shows that there are 35422 patients that received sms and only 9784 actually showed up for their appts while 25698 did not show up.

[45]:
```
# Group data by 'SMS_Received' and 'NoShow', and count the occurrences
sms_no_show_counts = df.groupby(['SMS_Received', 'NoShow']).size().unstack(fill_value=0)

# Plot the grouped bar plot
plt.figure(figsize=(8, 4))
sms_no_show_counts.plot(kind='bar', color=['lightblue', 'lightgreen'])
plt.title('Number of Appointments by SMS Received and No-Show')
plt.xlabel('SMS Received')
plt.ylabel('Number of Appointments')
plt.xticks([0, 1], ['No SMS Received', 'SMS Received'], rotation=0)
plt.legend(title='No-Show', labels=['Showed Up', 'No-Show'])
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
<Figure size 800x400 with 0 Axes>
```



2d

Relationship between 2 variables

CHANGE(S) REQUIRED

- Include `1d` analyses

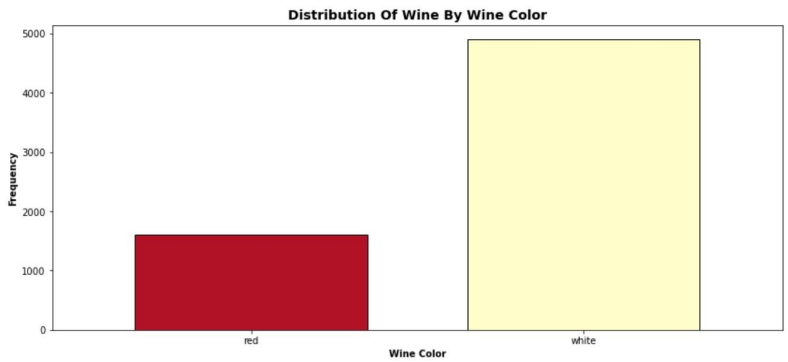
TIPS

Prior to exploring relationships between variables *(i.e. explorations involving more than one variable)*, you have to introduce your reader to the variables individually *(i.e. 1d or Univariate explorations)*.

This can be done statistically, or visually. For example, using a *wine dataset*, prior to examining the relationship between *quality* and *wine color*, you would show the reader the distribution of the wines of different colors:

```
(wine_df.groupby(['color'])
        .size()                # groupby color, get value counts of quality
        .plot.bar(color=['#b11226','#FFFFCC'],rot=0,width=0.7,edgecolor='black',figsize=[14, 6])); # plot
# Add title and format it
plt.title(f'Distribution of wine by wine color'.title(), fontsize = 14, weight = "bold")
# Add x label and format it
plt.xlabel('wine color'.title(), fontsize = 10, weight = "bold")
# Add y label and format it
plt.ylabel('Frequency'.title(), fontsize = 10, weight = "bold");
```

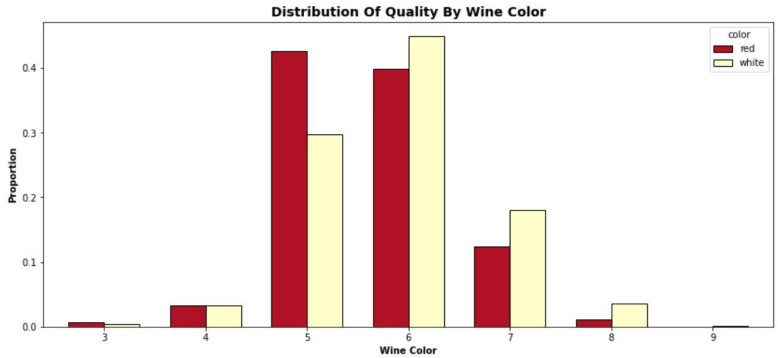
Prior to exploring relationships between variables (2d), introduce your reader to the properties of the indvidual variables (1d)



1 1d/Univariate Visualization

Color: What is the distribution of red and white wines in the dataset?

```
(wine_df.groupby(['color'])['quality']
        .value_counts(normalize=True)          # groupby color, get value counts of quality
        .unstack('color')                      # move color from the row index to column index
        .plot.bar(color=['#b11226','#FFFFCC'],rot=0,width=0.7,edgecolor='black',figsize=[14, 6])); # plot
# Add title and format it
plt.title(f'Distribution of quality by wine color'.title(), fontsize = 14, weight = "bold")
# Add x label and format it
plt.xlabel('wine color'.title(), fontsize = 10, weight = "bold")
# Add y label and format it
plt.ylabel('proportion'.title(), fontsize = 10, weight = "bold");
```



2 2d/Bivariate Visualization

Color & Quality: For red and white wines, what proportion are each quality level?

This approach ensures that your report offers your audience a comprehensive explanation of your data.

Both 1d **and** 2d explorations have to be included in order to fulfil this project's requirement.

ADDITIONAL NOTES

Investigating a dataset, i.e. performing an exploratory data analysis, is an approach to analysing data to summarize their main characteristics, using visual and statistical methods.

Data explorations are typically hierarchical, where the properties of the individual variables are examined first, followed by an analysis of the relationships between the variables.

Data explorations are less structured than model building. 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. This creates a broad picture of important trends and relationships.

- Data exploration summarized
- Data Exploration and Analysis Using Python

- **The project's visualizations are varied and show multiple comparisons and trends.**
- **At least two kinds of plots should be created as part of the explorations.**
- **Relevant statistics are computed throughout the analysis when an inference is made about the data.**

This Section Meets Specifications

✓ CRITERIA: THE PROJECT'S VISUALIZATIONS ARE VARIED AND SHOW MULTIPLE COMPARISONS AND TRENDS.

✓ CRITERIA: AT LEAST TWO KINDS OF PLOTS SHOULD BE CREATED AS PART OF THE EXPLORATIONS.

✓ CRITERIA: RELEVANT STATISTICS ARE COMPUTED THROUGHOUT THE ANALYSIS WHEN AN INFERENCE IS MADE ABOUT THE DATA.

Your selection of visualizations highlight interesting patterns, and you perform a range of statistical analyses. Both of which help to answer your research questions. Excellent work!

TIPS

PLOT DIMENSIONS

The plot dimensions change from plot to plot in your *data exploration*.

Using a fixed plot dimension, that is large enough to show all of the relevant information in the plot, for all of your visualizations helps (so that the reader doesn't have to re-focus on each visualization (and the increased width means that you don't have to angle some of your tick labels)).

TICK LABELS

Avoid angling tick labels. It makes visualizations difficult to interpret *(Imagine the viewer/reader angling their heads at the same time as trying to take in the rest of the information in the visualizations)*. There are two ways to resolve this:

1. Change the plot dimensions (in many cases, the tick labels in your visualizations did not need to be angled after an appropriate plot size was set),
2. Change the orientation of the visualization.

ADDITIONAL NOTES

Visual and statistical methods play two roles in data explorations:

1. They help uncover the properties of the data, and discover any relationships between variables.
2. They also are invaluable in *communicating* those result to colleagues and readers.

Python, and its modules, like *pandas* and *Seaborn*, are powerful tools that allow you to achieve these goals. *Obviously, the wide range of possible approaches are something that you will pick up over time - hopefully you found your introduction to these tools useful, interesting and productive.*

- The Python Graph Gallery
    - This site contains a gallery of most types of visualizations, *each example contains code that you can use as a template for your visualizations.*
- Plotting with categorical data

Conclusions Phase

- The Conclusions have reflected on the steps taken during the data exploration.
- The Conclusions have summarized the main findings in relation to the question(s) provided at the beginning of the analysis accurately.
- The project has pointed out where additional research can be done or where additional information could be useful.
- The conclusion should have at least 1 limitation explained clearly.
- The analysis does not state or imply that one change causes another based solely on a correlation.

This Section Meets Specifications

✓ CRITERIA: THE CONCLUSIONS HAVE REFLECTED ON THE STEPS TAKEN DURING THE DATA EXPLORATION.

✓ CRITERIA: THE CONCLUSIONS HAVE SUMMARIZED THE MAIN FINDINGS IN RELATION TO THE QUESTION(S) PROVIDED AT THE BEGINNING OF THE ANALYSIS ACCURATELY.

✓ CRITERIA: THE PROJECT HAS POINTED OUT WHERE ADDITIONAL RESEARCH CAN BE DONE OR WHERE ADDITIONAL INFORMATION COULD BE USEFUL.

✓ CRITERIA: THE CONCLUSION SHOULD HAVE AT LEAST 1 LIMITATION EXPLAINED CLEARLY.

✓ CRITERIA: THE ANALYSIS DOES NOT STATE OR IMPLY THAT ONE CHANGE CAUSES ANOTHER BASED SOLELY ON A CORRELATION.

Your report includes a conclusion section, summarizing the results of your analysis.

Throughout your analysis, you highlight the limitations of making direct causal statements (and the limitations due to the nature of features of the dataset). *You also include a discussion of limitations in your conclusion.* Excellent work!

ADDITIONAL NOTES

You will write your *Conclusion* (and *Introduction*) after you have completed your analysis.

The conclusion is typically the first part of a report that is read - even though it occupies a space at the end. Reading a conclusion first is the most efficient, and effective, way to read a report. It helps the reader organize their thoughts around a *'big picture'* perspective of the subject.

This is why it is important to make your *'last words count'* (because they will be the first words read).

- Conclusions
- Write Discussion and Conclusion

Communication

- **The code should have ideally the following sections: Introduction; Questions; Data Wrangling; Exploratory Data Analysis; Conclusions, Limitation.**
- **Reasoning is provided for each analysis decision, plot, and statistical summary.**
- **Interpretation of plots and application of statistical tests should be correct and without error.**
- **Comments are used within the code cells.**
- **Documented the flow of analysis in the mark-down cells.**

This Section Meets Specifications

✓ CRITERIA: THE CODE SHOULD HAVE IDEALLY THE FOLLOWING SECTIONS: INTRODUCTION; QUESTIONS; DATA WRANGLING; EXPLORATORY DATA ANALYSIS; CONCLUSIONS, LIMITATION.

✓ CRITERIA: REASONING IS PROVIDED FOR EACH ANALYSIS DECISION, PLOT, AND STATISTICAL SUMMARY.

✓ CRITERIA: INTERPRETATION OF PLOTS AND APPLICATION OF STATISTICAL TESTS SHOULD BE CORRECT AND WITHOUT ERROR.

✓ CRITERIA: COMMENTS ARE USED WITHIN THE CODE CELLS.

✓ CRITERIA: DOCUMENTED THE FLOW OF ANALYSIS IN THE MARK-DOWN CELLS.

Your commentary is clear and concise.

You have shown some very interesting results! Excellent job!

However, the discussion of those results is quite limited.

The commentary on your results is very important. It helps to clarify what you feel your visualizations and statistics are showing the reader. As this overview highlights

---

TIPS

- The commentary included should inform the reader of your thinking during your data exploration. A

key part of that process is the *Introduction*, i.e. what interests you about this dataset. Then, each stage of the *exploration* should be outlined. That is:

- Why (the visualization/statistic was created), and
- What (patterns in the data the visualization/statistic revealed)

  Currently, the commentary includes one or two facts about the visualization.

  The commentary doesn't have to be technical (or long) but it has to explain to the reader the role of the visualization in your exploration.

- **FYI**: Typically, when working on data exploration projects, in a work environment, you have more background knowledge of the data - which informs your commentary (i.e. allows you to tell a story about the role of the variables that you are analyzing.)

ADDITIONAL NOTES

The commentary on your results is very important. It helps to clarify what you feel your visualizations and statistics are showing the reader. As this overview highlights:

> 'However, tables and figures do not simply speak for themselves but you have to communicate to your readers what e.g. numbers and quantities mean; you also want your readers to accept your conclusions, which should follow logically from your results. If you do not comment on the results, your reader might interpret them differently, and your conclusions might appear rather strange or surprising.'

- How to comment on statistics
- Writing The Results
- Finally, here is an overview of interpreting visualizations

Visualizations made in the project depict the data in an appropriate manner (i.e., has appropriate labels, scale, legends, and plot type) that allows plots to be readily interpreted.

This Section Meets Specifications

✓ CRITERIA: VISUALIZATIONS MADE IN THE PROJECT DEPICT THE DATA IN AN APPROPRIATE MANNER (I.E., HAS APPROPRIATE LABELS, SCALE, LEGENDS, AND PLOT TYPE) THAT ALLOWS PLOTS TO BE READILY INTERPRETED.

All plots are clearly titled, and axes are labeled making your visualizations easy to interpret, excellent work!

ADDITIONAL NOTES

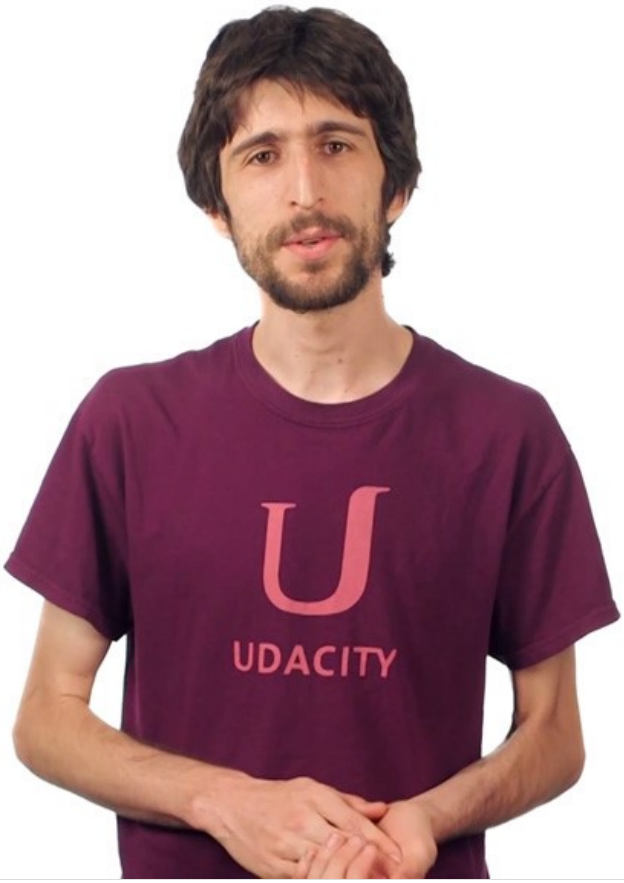
Meaningful titles and labels help your readers to interpret, and understand, the visualizations that you are using to communicate your results.

Titles and labels also help readers recall the message. The title is the most important, where specific, informative, titles reinforce the visualization's main message.

- Importance of Titles and Labels
- How to add labels and titles to a plot in matplotlib

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