



[Return to Classroom](#)

# Investigate a Dataset

REVIEW

HISTORY

## Meets Specifications

### Dear Student

I would like to congratulate you on your awesome work. You are clearly a good data analyst and a Python programmer, your code quality is impeccable 🏆

I truly enjoyed reading your report. Well done 🍌

You have great potential, and I tried to help you as best as I can. Please refer to the project review below. You'll find some useful resources and suggestions to further enhance your coding skills.

This is the start of a wonderful journey. I wish you the best of luck 🍀

*A gentle reminder to rate this review, as I greatly value your feedback.*

### Code Functionality

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

Your code runs without error and produces the same results described in your analysis. Good work 🏆

## Suggestions

- When working with data in pandas, it's important to avoid hard-coding the full path to a data file in your scripts or notebooks.

*Hard-coding paths can lead to problems for several reasons:*

- 1. Portability:** If you share your script or notebook with someone else, or move it to a different location on your computer, it will likely fail because the file path on their computer or the new location will be different.
- 2. Reproducibility:** If someone else tries to run your code on their machine, they will encounter errors if the file doesn't exist at the hard-coded path. This makes it difficult for others to reproduce your results.
- 3. Security:** Hard-coding paths can expose sensitive information about your system's file structure, which could be a security risk.

Instead of hard-coding, use relative paths. If the .csv file is in the same folder as your notebook file, you can load your data as follows:

```
df = pd.read_csv("./data_file.csv")
```

Or better yet, make the path a variable that can be easily changed. Then use the relative path as a value for this variable:

```
file_path = "./data_file.csv"  
df = pd.read_csv(file_path)
```

- To convert your notebook to an HTML file programmatically, you can call `nbconvert` directly inside the final code cell. A pseudo-code would be:  

```
!jupyter nbconvert <Filename>.ipynb --to html
```

. You can also export the notebook as an HTML file from the `File > Download as... > HTML or PDF` menu.
- Always remember to avoid installing or updating packages inside your notebook.

*Updating packages like pandas within a Jupyter notebook is not a good practice. Here's why:*

- 1. Reproducibility:** Updating packages in the notebook makes it hard to reproduce the environment. It's better to state the required versions of all packages at the beginning of your notebook.
- 2. Clarity:** Notebooks should tell a clear story. Including code to update packages can distract from the main narrative.
- 3. Potential Errors:** Updating a package in the notebook could break the code in earlier cells. It's safer to update packages outside of the notebook.

*For best practices:*

- Always manage package versions outside the notebook, in a separate environment setup process. This could be done using virtual environments, docker containers, or conda environments.
- At the beginning of the notebook, state the versions of all major packages used. This can be done in a markdown cell for human readers, and optionally also in a code cell that prints out the actual

versions in use when the notebook is run. This can be achieved using the `watermark` extension for example, as shown here:

```
%load_ext watermark
%load watermark
watermark -p pandas,numpy,matplotlib
```

## Useful Links

- [Jupyter Notebook Keyboard Shortcuts](#).
- [Six easy ways to run your Jupyter Notebook in the cloud](#).
- `watermark` [extension documentation](#)

- 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 requirement was passed in a previous review 

## Suggestions



Some Pandas' built-in methods are very useful for exploring data. These include:

- [Boolean-Indexing](#)
- [Group-by](#)
- [Value-Counts](#)
- [Series.map](#)
- [pandas.cut](#)

## Useful Links

- [Getting started with Python for science](#)
- [Python Cheatsheet](#)
- [Pandas Cheat Sheet for Data Science in Python](#)
- [The Ultimate NumPy Tutorial for Data Science Beginners](#)

- 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.

- Great. You created a function to handle multiple wrangling tasks. 
- You added comments to describe the complicated bits of the code 

## Suggestions

In data analysis, we use functions to replace repetitive code statements for tasks such as:

- Getting unique values from a certain DataFrame column.
- Convert timestamps to datetime dtype.
- Extracting time features from datetime columns.
- Repetitive visuals for similar data types.

Here are some examples to use as references:

```
import pandas as pd

def unique_values(df, column_name):
    """
    Get unique values from a column in a pandas DataFrame.

    Parameters:
    df (pd.DataFrame): The pandas DataFrame to extract unique values from.
    column_name (str): The name of the column to extract unique values from.

    Returns:
    pd.Series: A pandas Series containing the unique values.
    """
    return df[column_name].unique()
```

```
from datetime import datetime as dt

def parse_date(date_string):
    """
    Parse a date string into a datetime object.

    Parameters:
    date_string (str): The date string to parse.

    Returns:
    datetime.datetime: A datetime object representing the parsed date, or None if
    the input is an empty string.
    """
    if not date_string:
        return None
```

```
else:
    return dt.strptime(date_string, "%Y-%m-%dT%H:%M:%S%z")
```

```
import matplotlib.pyplot as plt

def plot_histogram(df, column, title, xlabel, ylabel, bins=10):
    """
    Plot a histogram from a DataFrame column, with a title and axis labels.

    Parameters:
    df (pd.DataFrame): The pandas DataFrame containing the data.
    column (str): The column to create the histogram from.
    title (str): The title of the histogram.
    xlabel (str): The label for the x-axis.
    ylabel (str): The label for the y-axis.
    bins (int): The number of bins to use in the histogram. Default is 10.

    Returns:
    None
    """
    plt.hist(df[column], bins=bins)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show()
```

## Useful Links

- [Python Functions – How to Define and Call a Function](#)
- [Python for Data - Functions](#)
- [10 Tips for Writing Cleaner & Better Code](#)
- [PEP 257 – Docstring Conventions](#)

## Quality of Analysis

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

The project provided guiding questions at the outset of the analysis, and then addressed these questions in the remainder of the analysis. Good work 🍌

## Suggestions

Always remember to state your questions clearly at the beginning of your notebook.

*Ensuring that a research question is clearly stated at the beginning of any data analysis is crucial for several reasons:*

- **Guidance:** The research question guides the entire data analysis process. It determines what data is needed, what kind of analyses to run, and how to interpret the results.
- **Clarity:** Clearly stating the research question at the beginning helps anyone reading the notebook understand the purpose of the analysis. It sets the context and allows the reader to follow the logic of the study.
- **Focus:** The research question keeps the analysis focused. It's easy to get lost in the data and go off on tangents. A well-defined research question prevents this by reminding you of the main goal of your analysis.

## Useful Links

- [From Research Question to Exploratory Analysis](#)
- [The Importance of Structure, Coding Style, and Refactoring in Notebooks](#)
- [Seven Tricks for Better Data Storytelling: Part I](#)
- [Seven Tricks for Better Data Storytelling: Part II](#)

## Data Wrangling Phase

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

You performed the basic wrangling required for this project. 

## Suggestions

- I suggest that you *spend enough time wrangling your data in your projects*. [The State of Data Science 2020 report by Kaggle](#) found that data scientists spend an average of 45% of their time on data wrangling.

## Useful Links

- [What is Data Wrangling?: 6 Important Steps](#)
- [Data Wrangling with pandas, Cheat Sheet](#)
- [Generating Documentation for Data Wrangling Code](#)
- [12 Python Built-In Functions for Data Science and Analytics](#)

## 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.

- The project investigates the stated question(s) from multiple angles. ✓
- The project explores at least three variables in relation to the research questions. ✓
- The project performs both single-variable (1d) and multiple-variable (2d) explorations. ✓

You studied multiple aspects of the dataset with interesting insights. Good work 🙌

### Suggestions

*Here is a reminder of some single-variable (1d) and multiple-variable (2d) explorations techniques.*

#### *EXAMPLES OF SINGLE-VARIABLE EXPLORATION TECHNIQUES INCLUDE:*

1. **Descriptive Statistics:** This method involves calculating measures of central tendency (mean, median, mode) and measures of dispersion (range, variance, standard deviation) to summarize the main characteristics of one variable.
2. **Histograms:** This is a graphical representation of the frequency distribution of a single variable.
3. **Box Plots:** A box plot provides a graphical image of the concentration of the data. It displays the five-number summary of a dataset (minimum, first quartile, median, third quartile, and maximum), which can help identify outliers
4. **Frequency Distribution:** This is a tabular format that displays the number of observations within a specific range.
5. **Pie Charts:** Pie charts can be used to display the relative frequencies of different categories in a categorical variable.

#### *EXAMPLES OF MULTIPLE-VARIABLE (2D) EXPLORATIONS TECHNIQUES INCLUDE:*

1. **Scatter Plots:** This is a two-dimensional data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis. It's particularly useful for showing the relationship between two variables.
2. **Pair Plots:** Pair plots are an extension of scatter plots that allow you to visualize the relationship between multiple pairs of variables simultaneously. This can be useful if you have many variables and want to get a quick overview of the relationships between them. For example, you could use a pair plot to visualize the relationship between all pairs of variables in a customer dataset.
3. **Heatmap:** A heatmap is a two-dimensional graphical representation of data, where the individual values that are contained in a matrix are represented as colors. It's useful for visualizing large amounts of data and how variables interact with each other.
4. **Correlation Analysis:** This is a statistical technique that measures the strength and direction of the linear relationship between two variables. It can be efficiently visualized using heatmaps.

## Useful Links

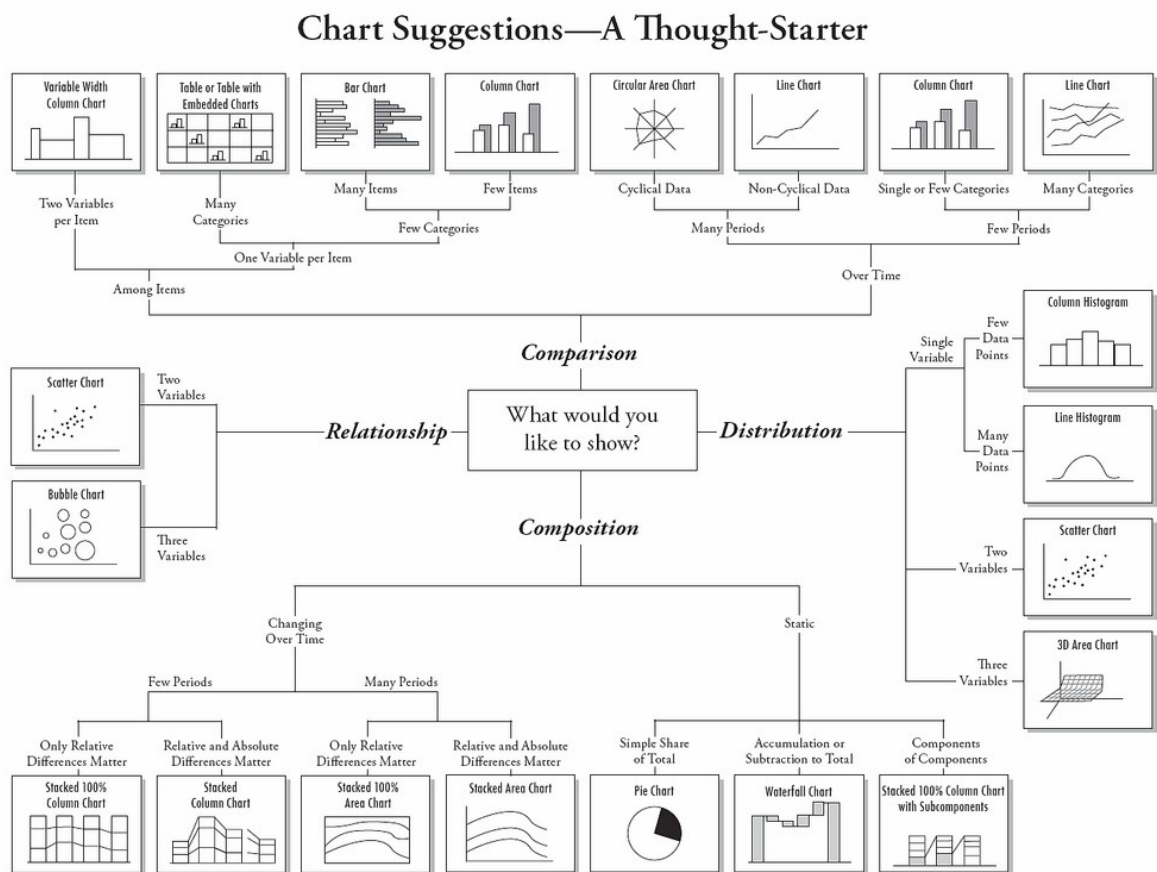
- [What is Univariate, Bivariate & Multivariate Analysis in Data Visualisation?](#)
- [EDA - Exploratory Data Analysis: Using Python Functions](#)
- [12 Python Built-In Functions for Data Science and Analytics](#)
- [Ten Simple Rules for Reproducible Research in Jupyter Notebooks](#)

- 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 requirement was passed in a previous review 

## Suggestions

Use this chart to guide you in choosing the correct visuals for your data.



## Useful Links



- [Matplotlib Cheat Sheet](#)
- [The Python Graph Gallery](#)
- [10 Do's and Don'ts of Infographic & Chart Design](#)
- [Do This, Not That: Data Visualization Before and After Examples](#)
- [How to pick more beautiful colors for your data visualizations](#)
- [Descriptive Statistics in Python](#)

## 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 requirement was passed in a previous review 

### Useful Links

- [How to Write Conclusions](#)
- [How to Present the Limitations of the Study](#)

## 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 requirement was passed in a previous review 

### Suggestions

- Get into the habit of utilizing the notebook capabilities in your future projects. ***Please use markdown cells to represent your wrangling and EDA decisions instead of using comments in the code cells.***
- Please refer to this [Project](#) to see how to properly document your analysis.

## Useful Links

- [Document Analysis Guide: Definition and How To Perform It](#)
- [Principles of Documenting Data](#)
- [15 Data Science Documentation Best Practices](#)

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.

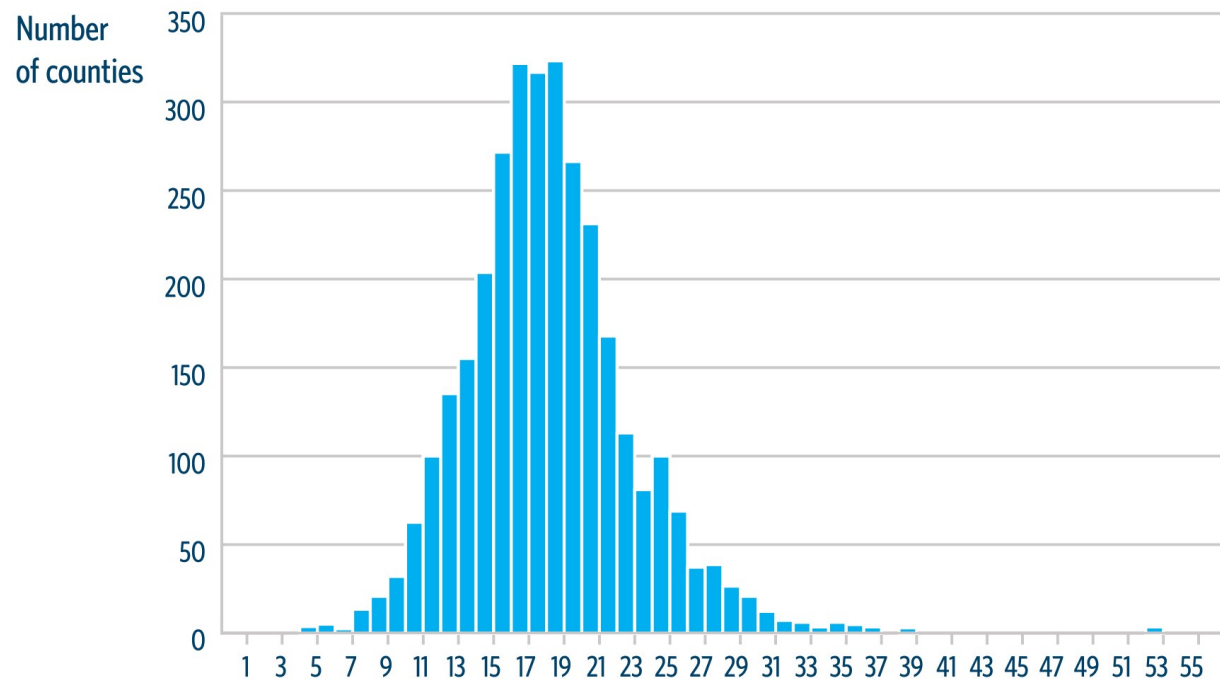
This requirement was passed in a previous review 

## Suggestions

- Always remember to add an informative title and clear axis labels to all your visuals.
- Always remember to add a legend when appropriate.

*Here are some examples to use as references:*

### Item 1: Percentage of Population Aged 65 and Older

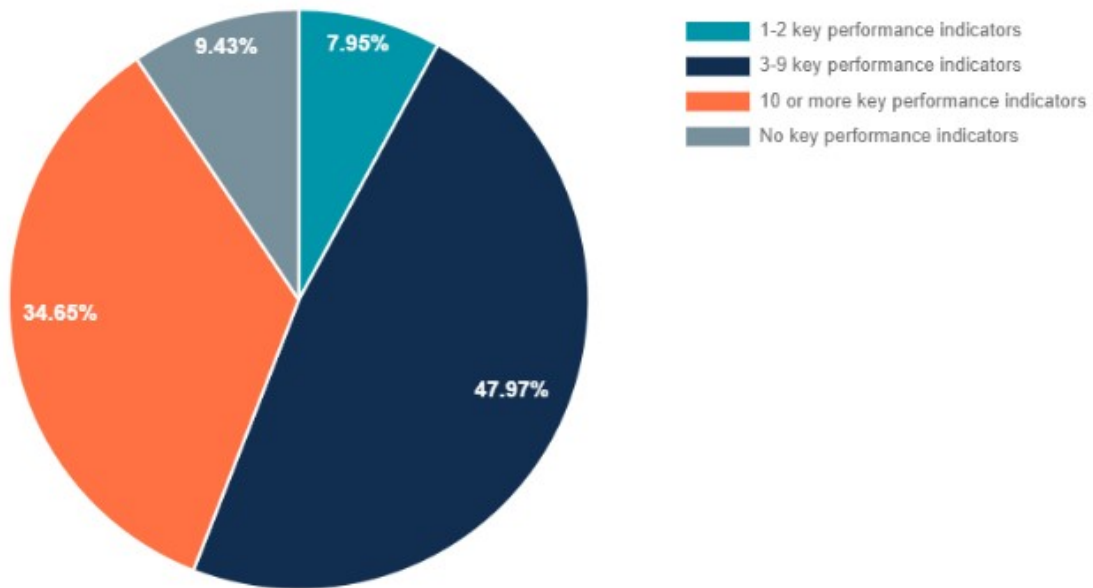


Percentage of seniors (aged 65 years and over) in each county

[factfinder.census.gov/bkrmk/table/1.0/en/PEP/2014/PEPAGESEX/0100000US.05000.003?slice=GEO~0500000US01005](https://factfinder.census.gov/bkrmk/table/1.0/en/PEP/2014/PEPAGESEX/0100000US.05000.003?slice=GEO~0500000US01005)

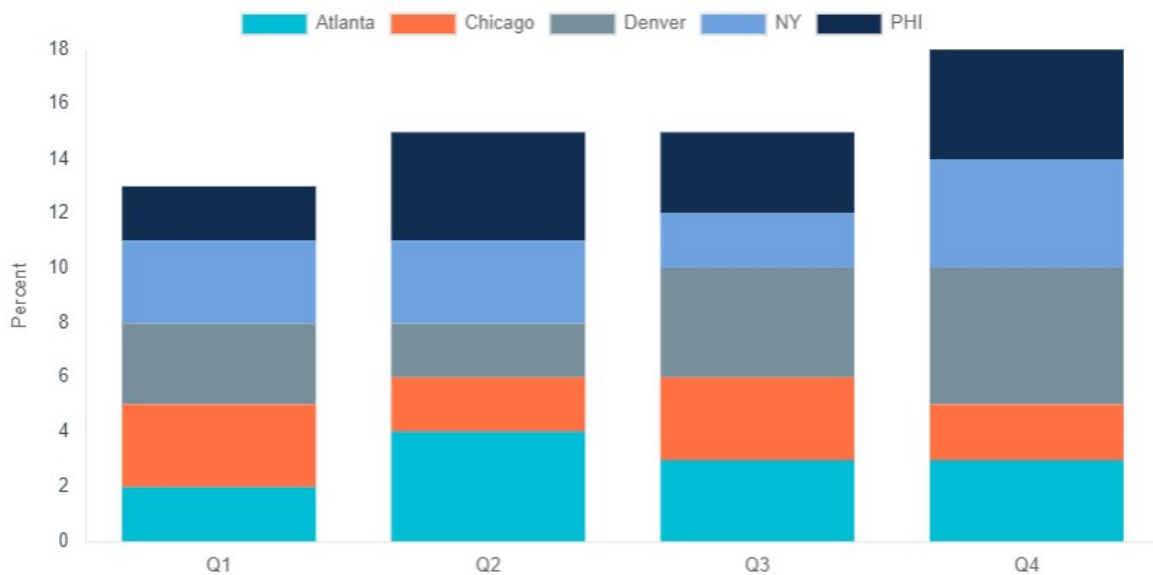
## Management in U.S. Manufacturing

How many key performance indicators were monitored at this establishment?



Source: U.S. Census Bureau, Massachusetts Institute of Technology, National Bureau of Economic Research, and Stanford University; 2015 Management and Organizational Practices Survey (MOPS). Accessed in March 2019.

## Population by Region



Source: None (fictional data)

## Useful Links

- [Data Visualization Standards - Titles](#)
- [Data Visualization Standards - Labels](#)
- [Data Visualization Standards - Legends](#)

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)

Rate this review

[START](#)