DSCI 503 – Project 03 Instructions

Background

In this project, we will be working with the Diamonds dataset. This dataset contains information about several thousand diamonds sold in the United States. Each diamond in the dataset has 10 attributes recorded for it, but we will only be interested in the following 5 attributes:

- **`price`** - The sales price of the diamond, in US Dollars.
- **`carat`** - The weight of the diamond, measured in carats. One carat is 200 mg.
- **`cut`** - Quality of the cut of the diamond. The levels (from worst to best) are **Fair**, **Good**, **Very Good**, **Premium**, and **Ideal**.
- **`color`** - Level of the tint in the diamond. Colorless diamonds are generally preferred. The levels of this variable (from worst to best) are: **J**, **I**, **H**, **G**, **F**, **E**, and **D**.
- **`clarity`** - Indicates the level of internal defects in the diamond. The levels (from worst to best) are: **I1**, **SI2**, **SI1**, **VS2**, **VS1**, **VVS2**, **VVS1**, **IF**.

General Instructions

Create a new notebook named **`Project_03_YourLastName.ipynb`** and complete the instructions provided below.

Any set of instructions you see in this document with an orange bar to the left will indicate a place where you should create a markdown cell. If no instructions are provided regarding formatting, then the text should be unformatted.

Any set of instructions you see with a blue bar to the left will provide instructions for creating a single code cell.

Read the instructions carefully.

Any time that you are asked to display a DataFrame in this assignment, you should do so without using the **`print()`** function.

Assignment Header

Create a markdown cell with a level 1 header that reads: "DSCI 503 – Project 03". Add your name below that as a level 3 header.

Import the following packages using the standard aliases: **`numpy`**, **`pandas`**, and **`matplotlib.pyplot`**. No other packages should be used in this project.

Part 1: Loading the Dataset; Preliminary Analysis

In this section, we will load the data into a DataFrame, and will explore the structure of the data set.

Create a markdown cell that displays a level 2 header that reads: "**Part 1: Loading the Dataset; Preliminary Analysis**". Also add some text briefly describing the purpose of your code in this part.

The data is stored in the tab-delimited text file **`diamonds.txt`**. Download this file into the directory that contains your notebook, and then load the data into a DataFrame named **`diamonds`**. Use the **`head()`** method to display the first 10 rows of this DataFrame.

Add a markdown cell explaining that we will determine the size of the dataset.

Print the shape of the **`diamonds`** DataFrame.

We will now inspect the distribution of the columns in **`diamonds`**. Add a markdown cell to briefly explain this.

Call the DataFrame method **`describe()`** on **`diamonds`** to display a DataFrame containing descriptive statistics for each of the columns.

Part 2: Filtering and Sorting

In this part, you will be asked to use filtering and sorting techniques to display information for diamonds satisfying certain criteria.

Create a markdown cell that displays a level 2 header that reads: "**Part 2: Filtering and Sorting**". Also some text explaining that we will start by viewing information about the 5 most expensive diamonds in the dataset.

Complete the following steps by chaining DataFrame methods, and without creating any new DataFrame variables.

1. Select the columns **`price`**, **`carat`**, **`cut`**, **`color`**, and **`clarity`** from **`diamonds`**.
2. Sort the resulting DataFrame by **`price`**, in descending order.
3. Use **`head()`** to display the first five rows of the result.

Create a markdown cell explaining that we will now view information about the 5 least expensive diamonds in the dataset.

Complete the following steps by chaining DataFrame methods, and without creating any new DataFrame variables.

1. Select the columns **`price`**, **`carat`**, **`cut`**, **`color`**, and **`clarity`** from **`diamonds`**.
2. Sort the resulting DataFrame by **`price`**, in ascending order.
3. Use **`head()`** to display the first five rows of the result.

Create a markdown cell explaining that we will now view information about the 5 largest diamonds in the dataset with an ideal cut.

Complete the following steps by chaining DataFrame methods, and without creating any new DataFrame variables.

1. Select the columns **`price`**, **`carat`**, **`cut`**, **`color`**, and **`clarity`** from **`diamonds`**.
2. Use boolean masking to filter the DataFrame, keeping only records for diamonds with an ideal cut.
3. Sort the resulting DataFrame by **`carat`**, in descending order.
4. Use **`head()`** to display the first five rows of the result.

Create a markdown cell explaining that we will now view information about the 5 largest diamonds in the dataset with an fair cut.

Complete the following steps by chaining DataFrame methods, and without creating any new DataFrame variables.

1. Select the columns **`price`**, **`carat`**, **`cut`**, **`color`**, and **`clarity`** from **`diamonds`**.
2. Use boolean masking to filter the DataFrame, keeping only records for diamonds with an fair cut.
3. Sort the resulting DataFrame by **`carat`**, in descending order.
4. Use **`head()`** to display the first five rows of the result.

Part 3: Working with Categorical Variables

The columns **cut**, **color**, and **clarity** are categorical variables whose values represent discrete categories that the diamonds can be classified into. Any possible value that a categorical variable can take is referred to as a **level** of that variable.

As mentioned at the beginning of these instructions, the levels of each of the variables have a natural ordering, or ranking. However, Pandas will not understand the order that these levels should be in unless we specify the ordering ourselves.

Create a markdown cell that displays a level 2 header that reads: "**Part 3: Working with Categorical Variables**". Add some text explaining that we will be creating lists to specify the order for each of the three categorical variables.

Create three lists named **`clarity_levels`**, **`cut_levels`**, and **`color_levels`**. Each list should contain strings representing the levels of the associated categorical variable in order from worst to best.

We can specify the order for the levels of a categorical variable stored as a column in a DataFrame by using the **`pd.Categorical()`** function. To use this function, you will pass it two arguments: The first is the column whose levels you are setting, and the second is a list or array containing the levels in order. This function will return a new series object, which can be stored back in place of the original column. An example of this syntax is provided below:

```
df.some_column = pd.Categorical(df.some_column, levels_list)
```

Create a markdown cell explaining that we will now use these lists to communicate to Pandas the correct order for the levels of the three categorical variables.

Use **`pd.Categorical()`** to set the levels of the **`cut`**, **`color`**, and **`clarity`** columns. This will require three calls to **`pd.Categorical()`**.

Create a markdown cell explaining that we will now create lists of named colors to serve as palettes to be used for visualizations later in the notebook.

Create three lists named **`clarity_pal`**, **`color_pal`**, and **`cut_pal`**. Each list should contain a number of named colors equal to the number of levels found for the associated categorical variable. The colors within each list should be easy to distinguish from one-another.

Part 4: Displaying Counts for Categorical Variables

In this part, you will determine the number of diamonds with each level of each of the three categorical variables.

Every column (series) object in a Pandas DataFrame has a **`value_counts()`** method that can be used to return a series containing counts of the number of times each possible value occurred within this column. We can sort the contents in this series by the value names by calling the **`sort_index()`** method for the series. The syntax for this is as follows:

```
df.some_column.value_counts().sort_index()
```

Create a markdown cell that displays a level 2 header that reads: "**Part 4: Displaying Counts for Categorical Variables**". Add text explaining that you will start by counting the number of diamonds for each level of **`cut`**.

Without creating any new DataFrame variables, select the **`cut`** column from **`diamonds`**, and then call its **`value_counts()`** method, followed by the **`sort_index()`** method. Display the result.

Create a markdown cell explaining that you will now count the number of diamonds for each level of **`color`**.

Without creating any new DataFrame variables, select the **`color`** column from **`diamonds`**, and then call its **`value_counts()`** method, followed by the **`sort_index()`** method. Display the result.

Create a markdown cell explaining that you will now count the number of diamonds for each level of **`clarity`**.

Without creating any new DataFrame variables, select the **`clarity`** column from **`diamonds`**, and then call its **`value_counts()`** method, followed by the **`sort_index()`** method. Display the result.

Part 5: Scatterplots of Price Against Carat

In this part, you will explore the relationship between the **price** and **carat** attributes by generating scatter plots.

Create a markdown cell that displays a level 2 header that reads: "**Part 5: Scatterplots of Price Against Carat**". Add text explaining the purpose of this section.

Create a scatterplot of **`price`** against **`carat`** according to the following specifications:

- Set the figure size to `[8,6]`.
- Set a point size of 20 and an alpha level of 0.4.
- Select a named single color for the points.
- The x-axis should be labeled "**Carat Size**".
- The y-axis should be labeled "**Price (in $)**".
- The figure should be title "**Relationship between Diamond Price and Carat Size**".
- Do **NOT** set a color for the border of the points. With this many point, the borders end up cluttering the plot.

Display the figure using **`plt.show()`**.

Create a markdown cell explaining that we will reproduce the plot from above, but will now color the points according to their clarity.

Create a scatterplot of **price** against **`carat`** with the point color indicating the **`clarity`** level. You can do this by looping over the previously defined **`clarity_levels`** list and adding a new scatterplot to the figure for each item in **`clarity_levels`**. The colors of the points associated with each level of **`clarity`** should be set according to the **`clarity_pal`** list.

Add a legend to the figure. Aside from the legend and the point colors, this plot should otherwise be identical to your previous plot.

Create a markdown cell explaining that you will now separate the points for each of the eight clarity levels into its own subplot.

Create a figure containing a 2x4 grid of subplots. Each subplot should display a scatter plot for diamonds corresponding to exactly one of the levels of **`clarity`**. Create the figure according to the following specifications:

- Set the figure size to `[12,6]`.
- Set a point size of 20 and an alpha level of 0.4.
- The points in each subplot have a fill color determined by the appropriate element of **`clarity_pal`**.
- Within each subplot, the x-axis should be labeled "**Carat Size**" and the y-axis should be labeled "**Price (in $)**".
- The title of each subplot should be equal to the level of **`clarity`** being represented by that subplot.
- The x-limits of each subplot should be set to `[0,4]` and the y-limits should be set to `[0,20000]`.
- The subplots should not contain a legend.

Call **`plt.tight_layout()`** and then display the figure using **`plt.show()`**.

Part 6: Applying Logarithmic Transformations

In this part, you will apply logarithmic transformations to the **price** and **carat** columns, and will explore the distribution of these transformed variables.

Create a markdown cell that displays a level 2 header that reads: "**Part 6: Applying Logarithmic Transformations**". Add text explaining that you will start by adding two new columns to the **`diamonds`** in order to store the transformed variables.

Add two new columns named **ln_carat** and **`ln_price`** to **`diamonds`**. The values stored in these columns should be the natural logarithms of the **`carat`** and **`price`** columns.

Create a markdown cell explaining that you will now use histograms to explore the distribution of the diamond prices, and the log of the diamond prices.

Display two side-by-side histograms. The histogram on the left should be for `price` and the one on the right should be for `ln_price`. Create your figure according to the following specifications:

- Set the figure size to be `[12,4]`.
- Label the x-axes as "**Price**" and **"Natural Log of Price"**. Label the y-axes as "**Count**".
- The plots should be titled "**Histogram of Prices**" and **"Histogram of Log-Prices"**.
- In each plot, set the number of bins to 20.
- In each plot, set `edgecolor` to be black, and use a single named color for the bar colors.

Display the figure using `plt.show()`.

Create a markdown cell explaining that you will now use histograms to explore the distribution of the carat sizes, and the log of the carat sizes.

Display two side-by-side histograms. The histogram on the left should be for `carat` and the one on the right should be for `ln_carat`. Create your figure according to the following specifications:

- Set the figure size to be `[12,4]`.
- Label the x-axes a "**Carat Size**" and **"Natural Log of Carat Size"**. Label the y-axes as "**Count**".
- The plots should be titled "**Histogram of Carat Sizes**" and **"Histogram of Log-Carat Sizes"**.
- In each plot, set the number of bins to 20.
- In each plot, set `edgecolor` to be black, and use a single named color for the bar colors, different from the one used in the previous plot.

Display the figure using `plt.show()`.

Part 7: Scatterplot of Transformed Variables

In this part, you will recreate the second scatterplot from Part 5, but using the transformed variables instead of the original values.

Create a markdown cell that displays a level 2 header that reads: "**Part 7: Scatterplot of Transformed Variables**". Add text explaining that you will create a scatterplot of `ln_price` against `ln_carat`, with points colored according to their `clarity`.

Recreate the second figure from Part 5 (which contained a single scatter plot, with point colors determined by clarity), using `ln_price` and `ln_carat` instead of `price` and `carat`. The axes should be labeled "**Natural Log of Carat Size**" and "**Natural Log of Prioe**". The title of the plot should be set to "**Relationship between Log-Price and Log-Carat Size**".

You should notice that while the relationship between `price` and `carat` was very nonlinear, the relationship between the transformed variables is approximately linear.

Part 8: Grouping by Cut

In this part, you will calculate the mean price and carat size for each level of the variable `cut`.

Create a markdown cell that displays a level 2 header that reads: "**Part 8: Grouping by Cut**". Add text explaining that you will create a grouped DataFrame displaying the mean price and carat size for each cut level.

Create a new DataFrame named `gb_cut` as follows: Select the `cut`, `price`, and `carat` columns from `diamonds`, group the result by `cut`, and then calculate the grouped means. Display the resulting DataFrame.

Create a markdown cell explaining that you will now use bar charts to graphically display the information from the DataFrame above.

Create a figure containing two side-by-side bar charts. Each chart should have a bar for every level of **cut**. The left chart should display the mean price for each level, and the right chart should display the mean carat size for each level. Create the figure according to the following specifications:

- Set the figure size to `[12,4]`.
- Label the x-axes as "**Cut**" . Label the y-axes "**Mean Price**" and "**Mean Carat Size**".
- The title should be "**Mean Price by Cut**" and "**Mean Carat Size by Cu**t".
- The **`edgecolor`** in each plot should be set to black.
- The color of the bars in each plot should be set according to **`cut_pal`**.

Display the plot using **`plt.show()`**.

Part 9: Grouping by Color

In this part, you will calculate the mean price and carat size for each level of the variable **`color`**.

Create a markdown cell that displays a level 2 header that reads: "**Part 9: Grouping by Color**". Add text explaining that you will create a grouped DataFrame displaying the mean price and carat size for each color level.

Create a new DataFrame named **`gb_color`** as follows: Select the **`color`**, **`price`**, and **`carat`** columns from **`diamonds`**, group the result by **`color`**, and then calculate the grouped means. Display the resulting DataFrame.

Create a markdown cell explaining that you will now use bar charts to graphically display the information from the DataFrame above.

Create a figure containing two side-by-side bar charts. Each chart should have a bar for every level of **color**. The left chart should display the mean price for each level, and the right chart should display the mean carat size for each level. Create the figure according to the following specifications:

- Set the figure size to `[12,4]`.
- Label the x-axes as "**Color**" . Label the y-axes "**Mean Price**" and "**Mean Carat Size**".
- The title should be "**Mean Price by Color**" and "**Mean Carat Size by Color**".
- The **`edgecolor`** in each plot should be set to black.
- The color of the bars in each plot should be set according to **`color_pal`**.

Display the plot using **`plt.show()`**.

Part 10: Grouping by Clarity

In this part, you will calculate the mean price and carat size for each level of the variable **`clarity`**.

Create a markdown cell that displays a level 2 header that reads: "**Part 10: Grouping by Clarity**". Add text explaining that you will create a grouped DataFrame displaying the mean price and carat size for each clarity level.

Create a new DataFrame named **`gb_clarity`** as follows: Select the **`clarity`**, **`price`**, and **`carat`** columns from **`diamonds`**, group the result by **`clarity`**, and then calculate the grouped means. Display the resulting DataFrame.

Create a markdown cell explaining that you will now use bar charts to graphically display the information from the DataFrame above.

> Create a figure containing two side-by-side bar charts. Each chart should have a bar for every level of **clarity**. The left chart should display the mean price for each level, and the right chart should display the mean carat size for each level. Create the figure according to the following specifications:
>
> - Set the figure size to `[12,4]`.
> - Label the x-axes as "**Clarity**" . Label the y-axes "**Mean Price**" and "**Mean Carat Size**".
> - The title should be "**Mean Price by Clarity** " and "**Mean Carat Size by Clarity**".
> - The **`edgecolor`** in each plot should be set to black.
> - The color of the bars in each plot should be set according to **`clarity_pal`**.
>
> Display the plot using **`plt.show()`**.

In Parts 8 – 10, you should notice that for each of the three categorical variables, higher quality diamonds tend to have a lower average price. That seems counter-intuitive until you notice that higher quality diamonds also tend to be smaller. As it turns out, diamond size has much stronger effect of the price than cut, color, or clarity.

Submission Instructions

When you are done, click **Kernel > Restart and Run All**. If any cell produces an error, then manually run every cell after that one, in order. Save your notebook, and then export the notebook as an HTML file. Upload the HTML file to Canvas and upload the IPYNB file to CoCalc, placing it in the **Projects/Project 03** folder.