Exploratory Data Analysis

I opened the project in Kaggle via Google Colab. I noticed that running the first cell connected to the Kaggle API from the Colab platform retrieved the dataset to be used in the project. I found it weird to see this line of code '%matplotlib inline' as part of the imports but after making a research, I realized it is just a standard way of telling matplotlib to display its graphs directly in Jupyter notebooks and will not explicitly need plt.show() command. Unfortunately, the file could not be imported so I had to download the laptop_prices.csv file for exploration. I came across several files with the same name but after few observations, I discovered that some of them throws an error when the csv is read by pandas. This was encoding errors which was fixed by the encoding value set the the general 'latin-1'. However, most of the data types were objects, as opposed to their numerical features. After finding the best dataset, **pd.read_csv(filename)** allowed me to import the dataset directly into a pandas dataframe.

numeric = df.select_dtypes(include=[np.int64,np.float64])

categorical = df.select_dtypes(include=[np.object_])

The above lines of code allowed splitting the dataset into categorical and numeric columns to better understand how to operate upon each of them. Upon selecting groups based in datatypes, I noticed that numpy was used to call the value in the argument. This made me wonder how numpy and pandas are interlaced

Also, the pattern for creating visual plots is very simple. It uses .fig and pass keyword argument figsize. Then you use plt.option where option is the type of chart to be drawn. The data is passed to this method allowing the chart to be populated with specific data. Options like labels and legends can be set. Then we finally use .show to show the graph.

In the dataset, only two rows had Adroid as the value for OS, this is insignificant was was dropped by targeting those rows with [*categorical[categorical['OpSys'] == 'Android'].index] and categorical.drop(index, inplace=True)

There is also a connection between seaborn and pyplot in such a way that, pyplot can create the visuals and seaborn can feed it with the data. An example code is sns.countplot(x='SSD', data=numeric)

After the univariate analysis, where individual columns were targeted, the pd.concat statement allows combining multiple dataframes together.

One observation is how a for-loop was used to iterate over all the columns in the numeric datatypes, concurrently showing all graphs beautifully