Seaborn

Visualization with Seaborn: -

Intro to seaborn?

- **Seaborn** is another **python library for data visualization** that introduces new **chart types and layouts, and interacts well with matplotlib**.
- It is a very convenient library **designed to work with pandas dataframes**.
- **Seaborn** is a python library built for **easily visualizing pandas DataFrames**, taking away some of the 'drawing' required when using matplotlib.
- To use seaborn, we use it like **import seaborn as sns**. Every seaborn plotting function is the plot type followed by plot.
- The syntax of seaborn charts is pretty standardised like: -
    - **import seaborn as sns**
      sns.lineplot(
      x = "date",
      y = "LodgingRevenue"
      data=hotels,
      estimator=sum, → estimator is used to perform aggregations
      ci = None
)
- Seaborn will also **automatically aggregate results so it interacts well with pandas** so we don't need to do as much data prep upfront.

Basic formatting options: -

- We can use our **matplotlib formatting syntax** on seaborn charts.
- If we are working with a chart type **like a bar chart or line chart**, we can use the **matplotlib arguments as keyword arguments in seaborn plotting functions** to format our charts accordingly.

- So, these are **going to be passed to the matplotlib object** that seaborn creates internally like: -
    - **sns.lineplot(**
      x = "date",
      y = "LodgingRevenue"
      data=hotels,
      estimator=sum,
      ci = None,
      ls='--',
      color='green'
)
- Seaborn still has some useful chart formatting functions like **despine()** **which is used to remove our top and right borders** and we can use it like: -
    - **sns.lineplot(**
      x = "date",
      y = "LodgingRevenue"
      data=hotels,
      estimator=sum,
      ci = None → it specify the size of confidence levels
)
      sns.despine()
- **hue parameter** is used for grouping variable that will produce lines with different colors. Can be either categorical or numeric, although color mapping will behave differently in latter case. A legend will be added automatically after using this.
- **palette parameter** is used to change the default color map.

Bar charts and histograms: -

- **Bar charts** can be created with **Seaborn sns.barplot().**
    - Simply specify the **desired category labels** and series values as 'x' & 'y' arguments.
    - We can create the **barplot() in Seaborn as**: -
        - **sns.barplot(**
          x = 'cut',
          y = 'carat',
          data = diamonds
          );

- **Seaborn automatically aggregates the data for the plot**, using unique category values as the labels for the bars, the mean of each category for the bar length, and the column headers and the axis labels.
- **Seaborn behind the scene used pandas aggregate functions** such as **groupby() and agg()** to aggregate our dataframe before plotting them.
- To create a **horizontal bar chart** simply **swap the values of 'x' and 'y'** and we will get the horizontal bar chart.
    - We can create the horizontal bar chart as: -
        - **sns.barplot(**
          x = 'carat',
          y = 'cut',
          data = diamonds
          );
- **Grouped bar chart** can be created by specifying a **categorical column as "hue" argument**.
    - We can create grouped bar chart like: -
        - **sns.barplot(**
          x = "cut",
          y = "price",
          data = diamonds,
          hue="clarity",
          palette="husl"
          );
- **Histograms** can be created with **Seaborn sns.histplot() and a single 'X' argument**.
    - We can also specify the number of **"bins"** and add the **kernel density(kde=True)** it gives us a smooth form of a histogram**.**
    - We can create histograms like: -
        - **sns.histplot(**
          x = "price",
          data = diamonds.query("clarity in ['I1','IF']"),
          hue = "clarity",
          bins = 10,
          palette='husl',
          kde=True,
          stat = percent
          );

- The default style for Seaborn plots can be nicer than their matplotlib counterparts, and vice versa. So, choose the library that works best for each chart.
- Scatter plot can be created with Seaborn **sns.scatterplot().** And we can create scatter plot like: -
    - **sns.scatterplot(**
      x = "x",
      y = "price",
      data = diamonds,
      hue="carat",
      size = "carat",
      palette = 'husl'
      );

Box and Violin plots: -

- It is possible to create box plots using matplotlib, **but it is easy to create a Box plot using Seaborn**.
- Boxplots can be created using **Seaborn sns.boxplot()**
    - They help to visualize the **distribution of variable by plotting key statistics**.
    - In **boxplot tend to focus on what are called robust statistics** like median and percentile Vs mean and standard deviation.
- So, if we have a **distribution of data that has several outliers** and we want to **understand what key percentiles are**, box plots are a great way to go.
- Boxplot statistics: -
    - **Median** will be the centre line in the middle of our box.
    - **1st and 3rd quartiles** (25th and 75th percentiles) it forms the edges of the box. 50% of the data is inside 1st and 3rd quartile.
    - **The IQR (inter quartile range)** 1st quartile – 3rd quartile is used to determine the length of the whiskers.
    - The length of the whiskers is going to be either min or max values. or if the max or min are further away than 1.5 times the IQR, the min and max will be set at 1.5 times the IQR.
    - We still have the data points beyond the edge of the whisker and these are called as **Outliers**.

- This is how we create box plots: -
    - **sns.boxplot(**
      x = "price",
      color=”orange”,
      data = diamonds
)
    - **sns.boxplot(** → this will plot the boxplot along y axis
      y = "price",
      color=”orange”,
      data = diamonds
)
- We can also **specify the second variable in the boxplot** to create a separate boxplot by category like: -
    - **sns.boxplot(** → this will plot the boxplot along y axis
      y = "price",
      x = “cut”
      color=”orange”,
      data = diamonds
)
- **Violin plots are the variation of the boxplot**. They give us little more insights that traditional box plots.
- Violin plot can be created with **sns.violinplot()**.
    - They are boxplots with symmetrical kernel densities along their sides.
    - This is how we create **violinplots():** -
        - **sns.violinplot(**
          x = "cut",
          y = "price",
          data = diamonds
)

Linear Relationship charts: -

- If a **chart type exists in Matplotlib then it also exists in Seaborn**, but the reverse is not necessarily true.
- It is **possible to build every Seaborn chart from hand and Matplotlib**.
- Seaborn brings the new functions to the table that makes it quite easy.

- The **series of chart types that help us explore linear relationships** are: -
    - **sns.scatterplot(x, y, data)** → creates a scatterplot and **helps us understand the co-relation between the variables**.
    - **sns.regplot(x, y, data)** → creates a scatterplot with a **fitted regression line**. The regression line will be the relationship between x and y.
    - **sns.lmplot(x, y, hue, row, col, data)** → creates a scatterplot with a **fitted regression line, and can visualize multiple categories using color, or splitting into rows & columns**. Specify the **"hue" parameter to create a line for each category** in the specified column and set a different color for each category. **Specify the 'row' and 'column' to create regression** plots for each combination.
    - **sns.jointplot(x, y, kind, data)** → creates a scatterplot and **adds the distribution of each variable**. **kind='kde' parameter** will plot the **kernel densities** and **kind = 'reg'** plots the **regression line** give us the density of the data.
    - **sns.pairplot(cols, corner)** → creates a matrix of **scatterplots comparing multiple variables, and shows the distribution for each one along diagonal**. So, if we were just to pass in the entire dataframe and not select certain columns, we'll get the pairwise scatterplots for every single numeric data in your dataframe. **corner=True parameter** will help us the remove all the redundant charts.

HeatMaps: -

- To create Heatmap in Seaborn to visualize table of data we use **sns.heatmap()**.
- Pandas pivot_table method is a great way to setup the data needed for the heatmap.
- This is how we make the heatmap: -
    - **sns.heatmap(**
      diamonds_pivot,
      annot=True,
      fmt='g',
      cmap="RdYlGn");

- **sns.set(rc = {'figure.figsize': (15,8)})** → this is how we increase the size of the figure using Seaborn.
- **We can modify the rcParameters with the sns.set()**.
- We can create the correlation matrix using heatmap like: -
    - **sns.heatmap(**
      top5_countries_pivot.iloc[:,:10].corr(),
      annot=True,
      cmap="vlag"
)

FacetGrid: -

- Seaborn **FacetGrid is a convenient alternative to Matplotlib's subplot grids**. If we want to build a grid completely out of Matplotlib charts and **don't want to have to build subplots by looping through all of our columns** then Seaborn FacetGrid is a great way to do this.
- We build the **Seaborn FacetGrid** as **sns.FacetGrid(DataFrame, column, column wrap)**.
- We use the **map_dataframe(chart_type, x) method** to map the plot type to the FacetGrid.
- **Pandas cut() function** is used to **separate the array elements into different bins** . The cut function is **mainly used to perform statistical analysis** on scalar data.
- This is how we create the FacetGrid: -
    - **g = sns.FacetGrid(**
      ca_housing,
      col="region_name",
      col_wrap=2
)
      g.map_dataframe(sns.barplot, x = "price_bins", y="inventory")

Matplotlib integration: -

- We can **build Seaborn plots in Matplotlib objects**, which lets you customize and integrate Seaborn charts as if they were built using Matplotlib.
- Seaborn is built on top of Matplotlib so we can use all the knowledge we gain using Matplotlib.