Visualization

31 January 2020 18:43

import matplotlib.pyplot as plt

%matplotlib inline
Or plt.show()

plt.style.available
plt.style.use("fivethirtyeight")

http://matplotlib.org/users/colormaps.html

plt.tight_layout()
plt.tight_layout()
plt.figure(figsize=(12,3))

plt.title('titanic.corr()')

Ax.Legend(loc=0)

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

sns.set_style()

Pandas - Visualisation

07 February 2020 13:45

import matplotlib.pyplot as plt

plt.style.available

plt.style.use("fivethirtyeight")

Pandas default plot method.. **.plot() on df and series.**

Df..plot(kind = "pie", legend = True)

df1['A'].hist()

- **df.plot.area**
- df.plot.barh
- df.plot.density
- df.plot.hist
- df.plot.line
- df.plot.scatter
- df.plot.bar
- df.plot.box
- df.plot.hexbin
- df.plot.kde
- df.plot.pie

You can also just call **df.plot(kind='hist')** or replace that kind argument with any of the key terms shown in the list above (e.g. 'box','barh', etc..)

Matplotlib

02 February 2020 16:26

Matplotlib: popular plotting library for python.

Functional method and object oriented method...

Special Plot Types:

```
plt.scatter(x,y)
plt.hist(data)
plt.boxplot(data,vert=True,patch_artist=True);
```

plt.style.available
plt.style.use("fivethirtyeight")

plt.tight_layout()
plt.figure(figsize=(12,3))

```
plt.title('titanic.corr()')

fig = plt.figure(figsize=(8,4), dpi=100)

axis.set(xlim=[0.5, 4.5], ylim=[-2, 8], title='An Example
Axes',
     ylabel='Y-Axis', xlabel='X-Axis')

ax.fill_between(x, y, color='lightblue')
```

create a figure canvas and add axes instances to the canvas::

plt.plot()

```
plt.plot(x, y, 'r') # 'r' is the color red
plt.xlabel('X Axis Title Here')
plt.ylabel('Y Axis Title Here')
plt.title('String Title Here')
```

Plt.subplot()

```
plt.subplot(2,2,1)
plt.plot(x, y, 'r--') # More on color options later
plt.subplot(2,2,2)
plt.plot(y, x, 'g*-')
```

fig = plt.figure() #**canvas plt.figure(figsize=(12,3))**
axes = fig.add_axes([.1, .1, .8, .8]) #**multiple axes**.
# Plot on that set of axes
axes.plot(x, y, 'r')
axes.set_xlabel('Set X Label') # Notice the use of set_ to begin methods
axes.set_ylabel('Set y Label')
axes.set_title('Set Title')

plt.tight_layout()

Plt.subplots() #fig, axes = plt.subplots(figsize=(12,3), dpi =100)

fig, axes = plt.subplots() #subplots.
axes is an array of axis to plot on.

 # Add_axes on figure as well.. Fig.add_axes([coordinates])
Saving figures::
Fig.savefig("filename.ext")

 # multiple plots on same axes...
ax.plot(x, x**2, label="x squared")
ax.plot(x, x**3, label="x cubed")
ax.legend()

Axes.legend()
ax.legend(loc=0) # let matplotlib decide the optimal location

label
Color, linewidth, alpha, linestyle

```
# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8,
markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
      markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green")
```

axes[1].plot(x, x**2, x, x**3)

axes[1].axis('tight')	**For fitting axis limits**

axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)

axes[2].set_ylim([0, 60])	**Fitting range of y axis**

axes[2].set_xlim([2, 5])	**Fitting range of x axis.**

```
axes[2].set_title("custom axes range")
```

Seaborn

08 January 2020 19:01

Seaborn: is a statistical plotting library.

Distribution Plots

Let's discuss some plots that allow us to visualize the distribution of a data set. These plots are:

Sns.distplot()	Single variable	**sns.distplot**(tips['total_bill'],kde=False,bins=30)
Sns.jointplot()	jointplot() allows you to basically match up two distplots for bivariate data. With your choice of what **kind** parameter to compare with: • "scatter" • "reg" • "resid" • "kde" • "hex"	**sns.jointplo**t(x='total_bill',y='tip',data=tips,kind='scatter') sns.jointplot(x='total_bill',y='tip',data=tips,kind='reg')
Sns.pairplot()	pairplot will plot pairwise relationships across an **entire dataframe (for the numerical columns) and supports a color hue argument (for categorical columns).**	**sns**.**pairplot**(tips) **sns.pairplot(tips,hue='sex',palette='coolwarm')** **sns.pairplot(iris,hue='species',palette='rainbow')**
rugplot		
kdeplot		

Categorical Data Plots¶

Sns.barplot()	These very similar plots allow you t**o get aggregate data off a categorical feature** in your data. **barplot** is a general plot that allows you to aggregate the categorical data based off some function, by default the **mean**:	sns.barplot(x='sex',y='total_bill',data=tips,**estimator**=np.std, hue='smoker')
Sns.countplot()	This is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why we only pass the x value:	
Sns.boxplot()	boxplots and violinplots are used to shown the distribution of categorical data. A box plot (or box-and-whisker plot) **shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.** The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the **inter-quartile range.**	sns.boxplot(x="day", y="total_bill", **hue**="smoker",data=tips, palette="coolwarm")
Sns.violinplot()	A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all of the plot components correspond to actual datapoints, the violin plot features a **kernel density estimation** of the underlying distribution.	sns.violinplot(x="day", y="total_bill", data=tips, hue='sex',palette='Set1') sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',**split**=True,palette='Set1')
sns.stripplot() Sns.swarmplot()	The stripplot will draw a **scatterplot** where one variable is categorical. A strip plot can be drawn on its own, but it is also a **good complement to a box or violin plot i**n cases where you want to show all observations along with some representation of the underlying distribution. The swarmplot is similar to stripplot(), but the points are adjusted (only along the categorical axis) so that they don't overlap. This gives a better representation of the distribution of values, although it does not scale as well to large numbers of observations (both in terms of the ability to show all the points and in terms of the computation needed to arrange them).	sns.stripplot(x="day", y="total_bill", data=tips,**jitter**=True,hue='sex',palette='Set1',**dodge**=True) # sns.violinplot(x="day", y="total_bill", data=tips, palette='rainbow') # sns.swarmplot(x="day", y="total_bill", data=tips, hue = 'sex', dodge=False, color = 'red') **Combination of 2::** sns.violinplot(x="tip", y="day", data=tips,**palette**='rainbow') sns.swarmplot(x="tip", y="day", data=tips,color='black',size=3) sns.boxplot(y="tip", x="day", data=tips,palette='rainbow', hue = 'sex') sns.stripplot(y="tip", x="day", data=tips, jitter = True , hue = 'sex', dodge = True)
Sns.factorplot()	factorplot is the most general form of a categorical plot. It can take in a ****kind**** parameter to adjust the plot type:	sns.factorplot(x='sex',y='total_bill',data=tips,**kind**='bar')

sns.stripplot(x="day", y="total_bill", data=tips,**jitter**=True,**hue**='sex',**palette**='Set1',**split**=True)

sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',split=True,palette='Set1')

sns.swarmplot(x="day", y="total_bill",**hue**='sex',data=tips, **palette**="Set1", **split**=True)

Matrix Plots

Matrix plots allow you to plot data as color-encoded matrices and can also be used to indicate clusters within the data (later in the machine learning section we will learn how to formally cluster data).

sns.heatmap()		tips.corr() sns.heatmap(tips.corr()) sns.heatmap(tips.corr(),cmap='coolwarm',**annot**=True) flights.pivot_table(index='month',columns='year', values='passengers') sns.heatmap(pvflights,cmap='magma',linecolor='white',linewidths=1)
sns.clustermap()		sns.clustermap(pvflights) sns.clustermap(pvflights,cmap='coolwarm',**standard_scale=1)**

Grids

Grids are general types of plots that allow you to **map plot types to rows and columns of a grid, this helps you create similar plots separated by features.**

sns.PairGrid()	Pairgrid is a subplot grid for plotting pairwise relationships in a dataset.
	
	More flexibility wrt to sns.**pairplot()**
		# Then you map to the grid
		g = sns.PairGrid(iris)
		g.map_diag(plt.hist)
		g.map_upper(plt.scatter)
		g.map_lower(sns.kdeplot)
		
		sns.pairplot(iris,hue='species',palette='rainbow')

sns.FacetGrid()	FacetGrid is the general way to create grids of plots based off of a feature:	g = sns.FacetGrid(tips, **col="time", row="smoker"**) g = g.map(plt.hist, "total_bill") g = sns.FacetGrid(tips, col="time", row="smoker",hue='sex') # Notice two arguments come after plt.scatter call g.map(**plt.scatter**, "total_bill", "tip")**.add_legend()** g = sns.FacetGrid(tips, col="time", row="smoker") g.map(sns.distplot,'total_bill')
Sns.JointGrid	JointGrid is the general version for jointplot() type grids, for a quick example:	g = sns.JointGrid(x="total_bill", y="tip", data=tips) g.plot(sns.regplot, sns.distplot)

Regression Plots

Sns.lmplot()		sns.lmplot(x='total_bill',y='tip',data=tips,**hue**='sex',palette='coolwarm')
	markers	sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex',palette='coolwarm', **markers**=['o','v'],scatter_kws={'s':100})
	splitting on row and column	sns.lmplot(x="total_bill", y="tip", row="sex",**col="time",data=tips)**
	Aspect and size	sns.lmplot(x='total_bill',y='tip',data=tips,col='day',hue='sex',palette='coolwarm', aspect=0.6,size=8)

Style and colour

sns.set_style()		sns.set_style('white') sns.countplot(x='sex',data=tips)
Remove spine		sns.countplot(x='sex',data=tips) **sns.despine()**

Size and Aspect	You can use matplotlib's **plt.figure(figsize=(width,height)* * to change the size of most seaborn plots.
	You can control the size and aspect ratio of most seaborn grid plots by passing in parameters: size, and aspect. For example:
		plt.figure(figsize=(12,3))
		sns.countplot(x='sex',data=tips)
		
		# Grid Type Plot
		sns.lmplot(x='total_bill',y='tip',height=2,aspect=4,data=tips)

Scale and Context	The **set_context()** allows you to override default parameters: Default is **notebook**	**sns.set_context('poster',font_scale=2)** sns.countplot(x='sex',data=tips,palette='coolwarm')

Plotly and Cufflinks

02 February 2020 17:51

Plotly is an interactive visualization library.

Cufflinks connects Plotly with Pandas.