DATA VISUALIZATION

by Dr Juan H Klopper

- Research Fellow
- School for Data Science and Computational Thinking
- Stellenbosch University

INTRODUCTION

Visualizing data is not only a pleasing activity, but it can provide an even richer understanding of the data than summary statistics. Data visualisation is also part and parcel of data communication, a key activity in Data Science.

PACKAGES USED IN THIS NOTEBOOK

In this notebook, we take a look at one of the myriad plotting packages in Python called plotly. We choose it for this course due to its ability to produce modern, multi-use plots, ready for online reports and print documents such as reseach papers. Matplotlib, seaborn, and altair are some of the other established plotting packages.

```
1 import pandas as pd
```

Two of the main plotting modules in plotly are the `graph_objects` and the `express` modules. We import these with the commonly used namespace abbreviations `go` and `px`. We also import the `io` module to set a theme for our plots. We choose `plotly_white` since this notebook uses a white background. You can type `pio.templates` which will return a list of all the themes.

```
1 import plotly.graph_objects as go
```

```
import plotly.express as px
import plotly.io as pio
pio.templates.default = 'plotly_white'
```

```
pio.templates # A list of the available plot themes
```

```
Templates configuration
-----------------------
    Default template: 'plotly_white'
    Available templates:
        ['ggplot2', 'seaborn', 'simple_white', 'plotly',
         'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
         'ygridoff', 'gridon', 'none']
```

This notebook was created on a computer with a retina display. In order to plot high resolution plots, we use the %config magic command below.

```
%config InlineBackend.figure_format = "retina" # For Retina type displays
```

Finally, the `%load_ext` magic command is used to set the way tables are printed to the screen in Google Colab.

```
%load_ext google.colab.data_table
```

The `drive` function allows us to connect to the files in our Google Drive.

```
from google.colab import drive  # Connect to Google Drive
```

DATA IMPORT

Since our data is in the `data` folder in Google Drive, we need to mount the drive.

```
# Log on and list files in the DATA directory of our Google Drive
drive.mount('/gdrive')
%cd '/gdrive/My Drive/Stellenbosch University/School for Data Science and Comput
%ls
```

```
Mounted at /gdrive
/gdrive/My Drive/Stellenbosch University/School for Data Science and Computat
australia_rain.csv    crops.csv       DefaultMissingData.csv
bitcoin_ethereum.csv  customers.csv   kaggle_survey_2020_responses.csv
breast_cancer.csv     data.csv        MissingData.csv
client_data.csv       DatesTimes.csv  montague_gardens_construction.csv
```

The data set contains data on banking customers and is used in machine learning to predict

```
df = pd.read_csv('customers.csv')
```

Later, we will also use data about rainfall in Australia.

```
rain = pd.read_csv('australia_rain.csv')
```

```
df # Viewing a table of the customer data
```

1 to 25 of 10127 entries Filter

index	Attrition_Flag	Customer_Age	Dependent_count	Education_Level	Marital_Status	Income_
0	Existing Customer	49	4	Uneducated	Single	R80K - R
1	Attrited Customer	48	2	Graduate	Married	R60K - R
2	Existing Customer	44	3	Graduate	Unknown	R80K - R
3	Existing Customer	48	3	Graduate	Single	R80K - R
4	Existing Customer	49	1	Graduate	Married	R120K +

We consider some meta data about the dataframe object.

```
1 df.shape # Number of observations (rows) and variables (columns)
```

```
(10127, 14)
```

```
1 df.columns # Statistical variables (column names)
```

```
Index(['Attrition_Flag', 'Customer_Age', 'Dependent_count', 'Education_Level',
       'Marital_Status', 'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy'],
      dtype='object')
```

```
1 # Data types of each column
2 df.dtypes
```

```
Attrition_Flag             object
Customer_Age                int64
Dependent_count             int64
Education_Level            object
Marital_Status             object
Income_Category            object
Card_Category              object
Months_on_book              int64
Total_Relationship_Count    int64
Months_Inactive_12_mon      int64
Contacts_Count_12_mon       int64
Credit_Limit               object
Total_Revolving_Bal         int64
Avg_Open_To_Buy            object
dtype: object
```

A method that we have not used before is the `info` method. It returns a pandas dataframe object with a column of all the variables, a column of the number of non missing data, and a column of the data types.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Attrition_Flag            10127 non-null  object
 1   Customer_Age              10127 non-null  int64 
 2   Dependent_count           10127 non-null  int64 
 3   Education_Level           10127 non-null  object
 4   Marital_Status            10127 non-null  object
 5   Income_Category           10127 non-null  object
 6   Card_Category             10127 non-null  object
 7   Months_on_book            10127 non-null  int64 
 8   Total_Relationship_Count  10127 non-null  int64 
 9   Months_Inactive_12_mon    10127 non-null  int64 
 10  Contacts_Count_12_mon     10127 non-null  int64 
 11  Credit_Limit              10127 non-null  object
 12  Total_Revolving_Bal       10127 non-null  int64 
 13  Avg_Open_To_Buy           10127 non-null  object
dtypes: int64(7), object(7)
memory usage: 1.1+ MB
```

We start our journey introdcuing only the most commonly used plots. In fact, the plotly package is enormous and we cannot possibly look at all the plots that it can create. There is a lot more information at the plotly website.

BAR PLOT

Bar plots are great for indicating frequency and relative frequency. In other words, counting the sample space elements of categorical or discrete data.

One axis, usually the horizontal axis is reserved to indicate the sample space elements of the catgeorical variable. The other axis is used to show the frequency or relative frequency, i.e. the height of a bar. There are spaces in between the bars to indicate that the sample space elements are indeed not a continuity.

Below, we use the `value_counts` method on the series object of the `Attrition_Flag` variable.

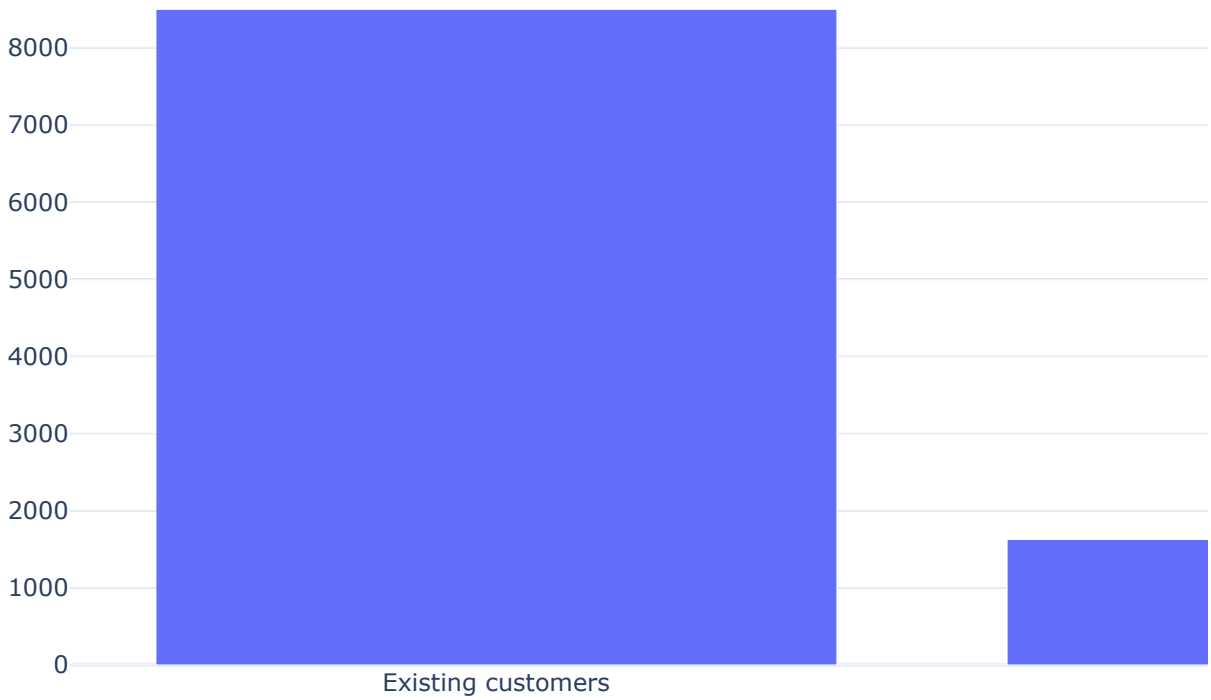
```
1 # Calculate the frequency count
2 df.Attrition_Flag.value_counts()
```

```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

A total of 8500 customers are still with the bank (`Existing Customer`) and 1627 have left the bank (`Attrited Customer`). We visualize this data as a bar blot. The `Figure` function in the

graph_objects module is assigned to a computer variable. We then use the `add_trace` method on this figure to add a bar chart using the `Bar` function. We set the `x` axis (sample space elements of the categorical variable as Python a list object) and the `y` axis values (frequencies also as a Python list object).

```
churn_fig = go.Figure() # Simple bar chart

churn_fig.add_trace(
    go.Bar(
        x=['Existing customers', 'Lost customers'],
        y=[8500, 1627]
    )
)
```



We entered the `x` and `y` values by hand. While this is easy enough for smaller sample element numbers, it is not always the case. We can get the sample space elements using the `unique()` method. It will be returned in the order in which the method discovers them in the relevant pandas series object.

```
df.Education_Level.unique()
```

```
array(['Uneducated', 'Graduate', 'College', 'Unknown', 'High School',
```

```
       'Post-Graduate', 'Doctorate'], dtype=object)
```

This might not be the order in which we want the bars to appear in.

With the default argument values of the `.value_counts()` method, we get the frequencies in descending order.

```
df.Education_Level.value_counts()
```

```
Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
```

This might also not be useful as in the case of a statistical variable such as *Month*. The bottom line is that we have to be careful with our code when designing plots.

The `Bar` function requires the frequency variable to be a list of values. We extract that below using the `value` property and the `tolist` method.

```
totals = df.Education_Level.value_counts().values.tolist()
totals
```

```
[3128, 2013, 1519, 1487, 1013, 516, 451]
```

Since we are using a series object, with the sample space elements as the index, we can use the `index` attribute. We can also convert it to a list using the `tolist` method.

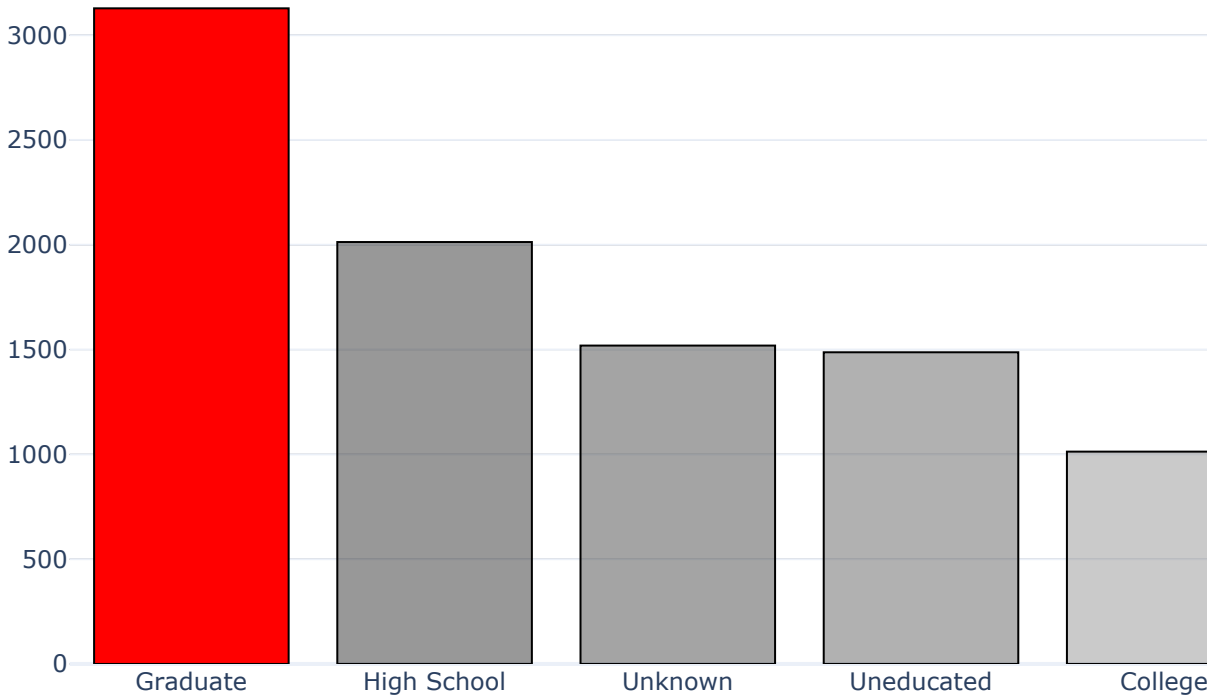
```
levels = df.Education_Level.value_counts().index.tolist()
levels
```

```
['Graduate',
 'High School',
 'Unknown',
 'Uneducated',
 'College',
 'Post-Graduate',
 'Doctorate']
```

With these lists saved, we can use them for our `x` and `y` arguments in the `Bar` function. We also use the `marker` argument and set its value using a dictionary. We specify seven colors and also a bar outline color and line thickness. Colors can be specified by their name such as `red` or

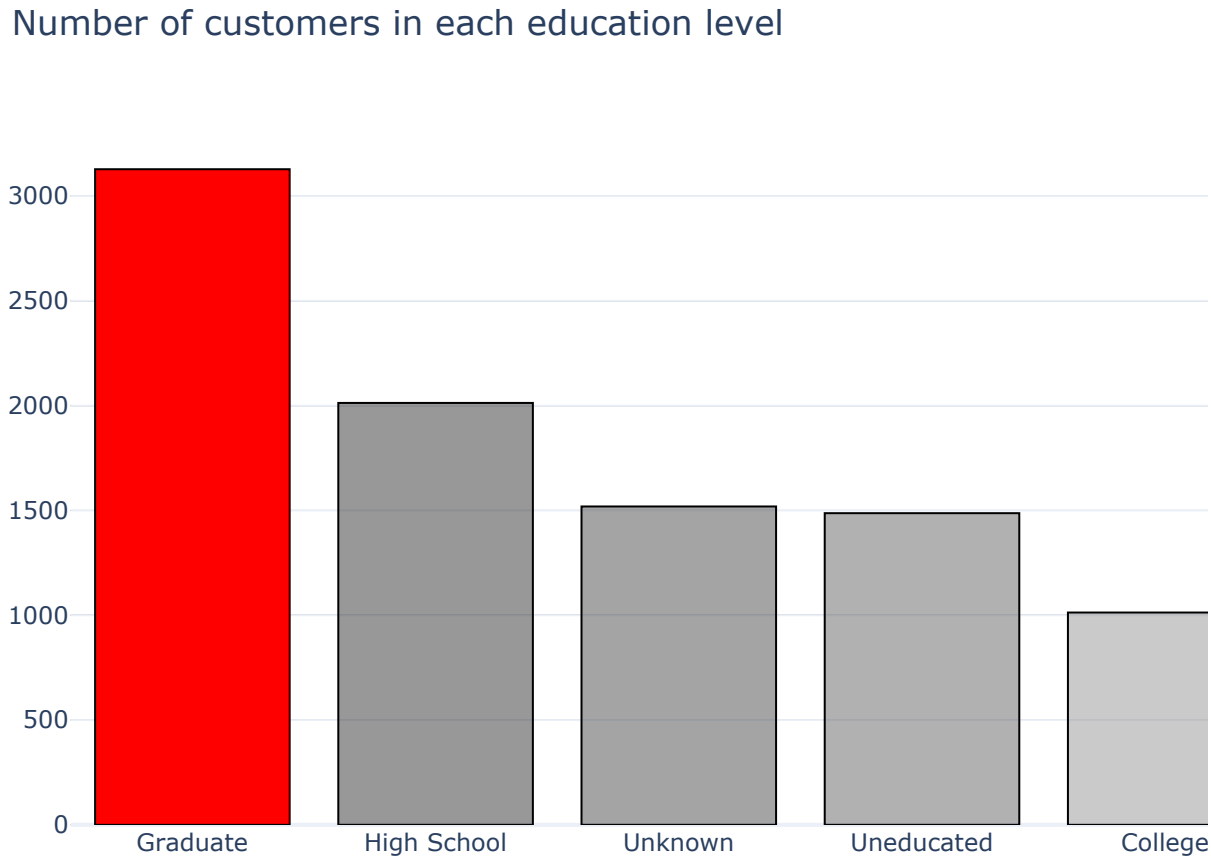
`blue` or by RGBA values. The latter uses four values. The first three are for red, green, and blue

```
# Simple bar plot
churn_fig = go.Figure()

churn_fig.add_trace(go.Bar(
    x=levels,
    y=totals,
    marker={'color':['red', 'rgba(50, 50, 50, 0.5)', 'rgba(75, 75, 75, 0.5)', 'r
            'line':{'color':'black', 'width':1}}
))

churn_fig.show()
```



The figure is assigned to a computer variable. We can use the `update_layout` method to add a title and axes labels.

```
# Add a title
churn_fig.update_layout(title='Number of customers in each education level')
churn_fig.show()
```




```
# Add axes labels
churn_fig.update_layout(xaxis=dict(title='Education level'),
                        yaxis=dict(title='Counts'))
churn_fig.show()
```

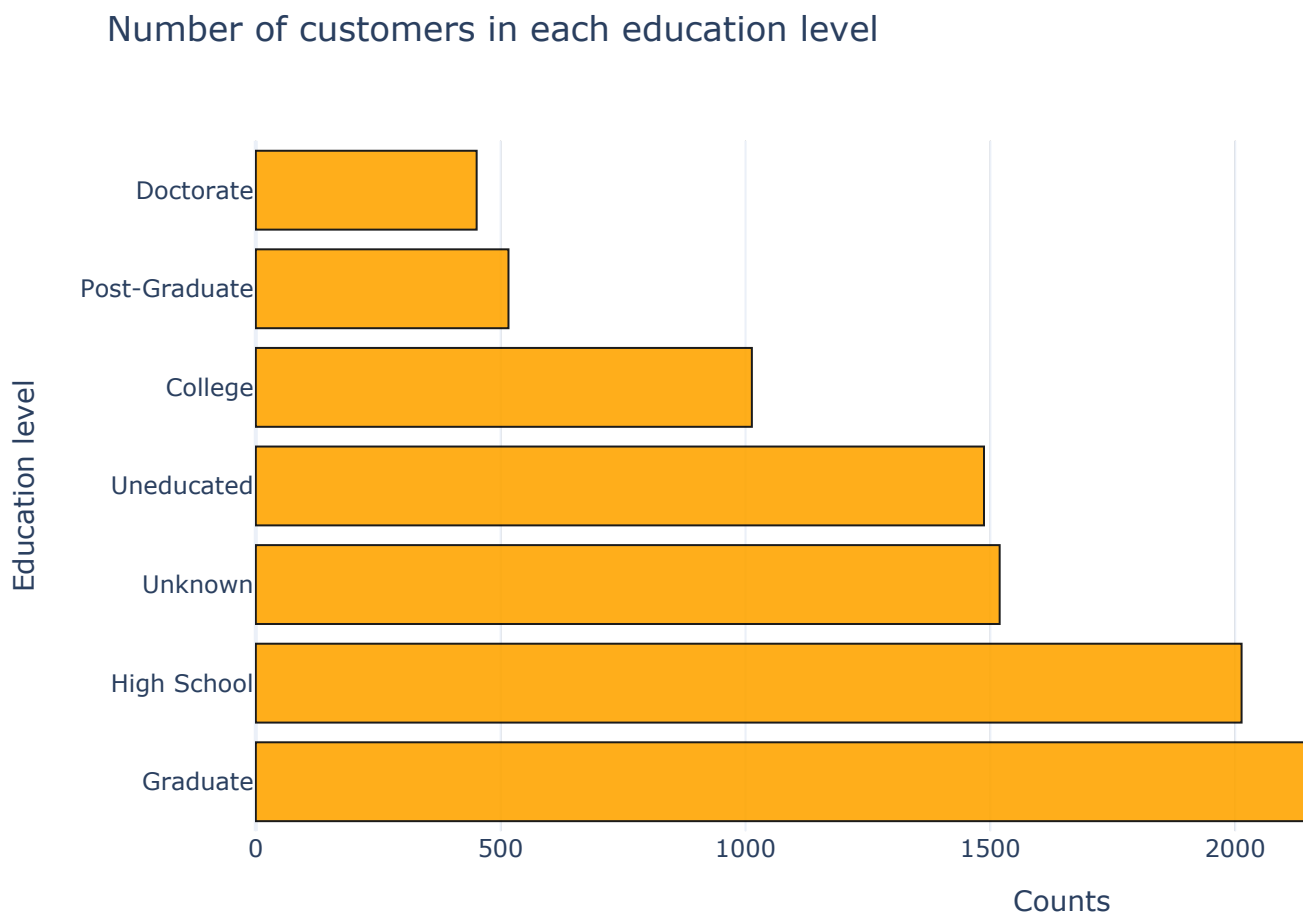
Number of customers in each education level

In the bar plot below, we add many more properties as illustration of what is possible.

```
churn_fig = go.Figure()

churn_fig.add_trace(go.Bar(
    x=levels,
    y=totals,
    text=levels,
    textposition='outside',
    marker={'color':'deepskyblue',
            'line':{'color':'black', 'width':1},
            'opacity':0.9}
))

churn_fig.update_layout(title='Number of customers in each education level')

churn_fig.update_layout(xaxis = dict(title='Education level'),
                        xaxis_tickangle=-25,
                        yaxis=dict(title='Counts'))

churn_fig.show()
```

We can also create horizontal bar plots using the `orientation` argument set to `h`, remembering to swicth the axes information.

```
churn_fig = go.Figure()

churn_fig.add_trace(go.Bar(
    y=levels,
    x=totals,
    marker={'color':'orange',
            'line':{'color':'black', 'width':1},
            'opacity':0.9},
    orientation = 'h'
))

churn_fig.update_layout(title='Number of customers in each education level')

churn_fig.update_layout(yaxis = dict(title='Education level'),
                        xaxis=dict(title='Counts'))

churn_fig.show()
```



We can group data by another categorical variable. Below, we plot the frequencies of the different educational levels but divided for each of the customers groups (lost versus existing). We start with the `crosstab` function in pandas. It returns a dataframe object. We pass two categorical variables. The variable listed first results in the rows and the sample space elements of the second variable becomes the column names.

```
1 attr_edu = pd.crosstab(df.Attrition_Flag, df.Education_Level)
2 attr_edu
```

1 to 2 of 2 entries Filter

Attrition_Flag	College	Doctorate	Graduate	High School	Post-Graduate	Uneducated	Unkn
Attrited Customer	154	95	487	306	92	237	
Existing Customer	859	356	2641	1707	424	1250	

Show 25 per page

Note that the order of the educational levels is now different from when we used the `value_counts` method on a series object. We save the educational level values as a list.

```
1 levels = attr_edu.columns.tolist()
2 levels
```

```
['College',
 'Doctorate',
 'Graduate',
 'High School',
 'Post-Graduate',
 'Uneducated',
 'Unknown']
```

Since our object is a dataframe, we can used indexing. Below, we do this by using `iloc` indexing. With it we specify the row and the column values we require. Remember that the colon symbol is short-hand for including all the values (columns in this case).

```
1 attr_values = attr_edu.iloc[0, :].tolist()
2 attr_values
```

```
[154, 95, 487, 306, 92, 237, 256]
```

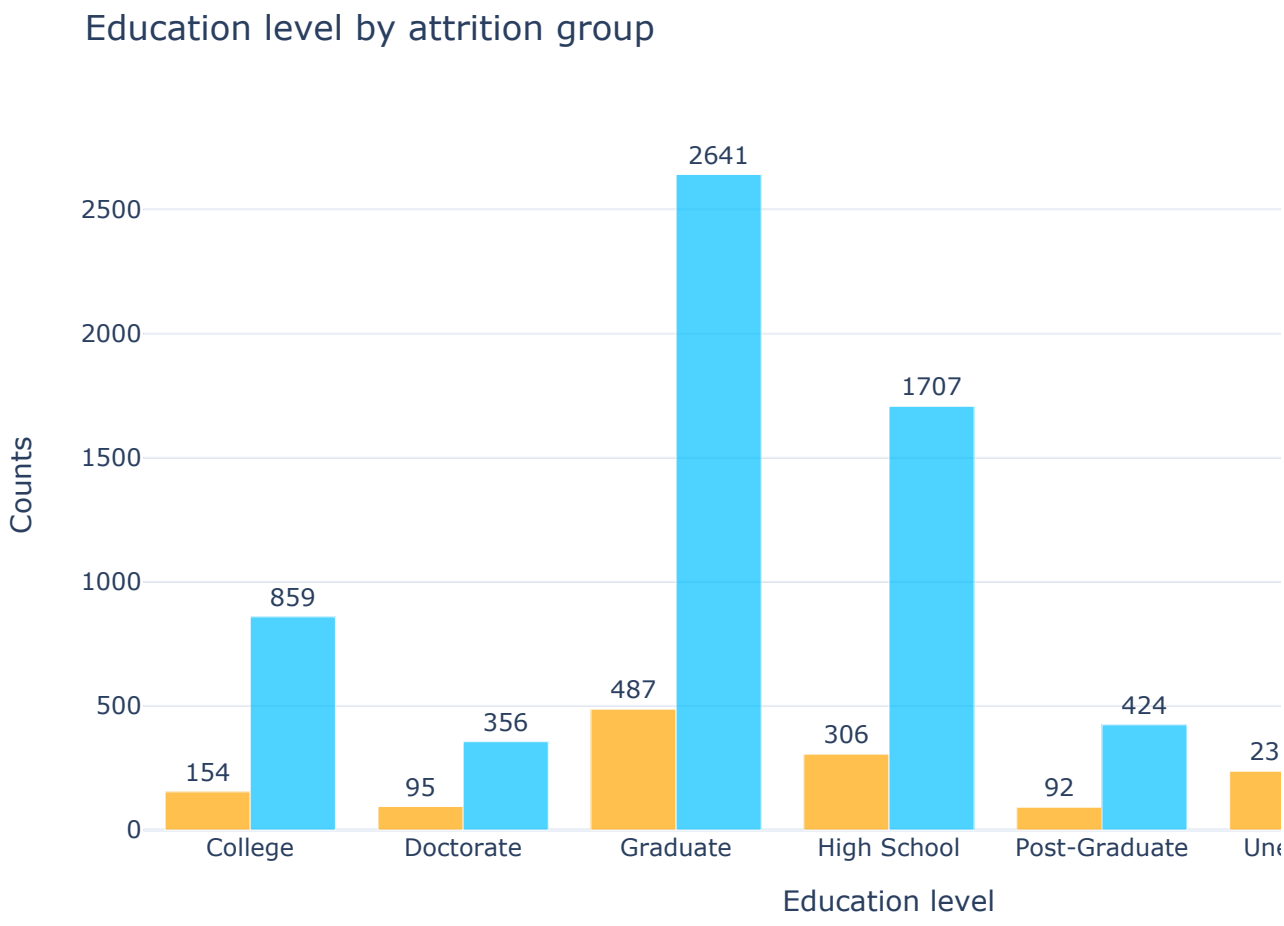
```
1 exis_values = attr_edu.iloc[1, :].tolist()
2 exis_values
```

```
[859, 356, 2641, 1707, 424, 1250, 1263]
```

No we can generate two traces using the `add_trace` function twice. We also set `barmode` argument to `group` in the `update_layout` method to create a grouped bar chart.

```
1 churn_edu_fig = go.Figure()
2
```

```
churn_edu_fig.add_trace(go.Bar(
    x=levels,
    y=attr_values,
    text=attr_values,
    textposition='outside',
    name='Lost customers',
    marker={'color':'orange', 'opacity':0.7}
))

churn_edu_fig.add_trace(go.Bar(
    x=levels,
    y=exis_values,
    text=exis_values,
    textposition='outside',
    name='Existing customers',
    marker={'color':'deepskyblue', 'opacity':0.7}
))

churn_edu_fig.update_layout(title='Education level by attrition group')

churn_edu_fig.update_layout(xaxis = dict(title='Education level'),
                            yaxis=dict(title='Counts'),
                            barmode='group')

churn_edu_fig.show()
```



Exercise (advanced)

Create the bar plot above, but group the horizontal axis by the customer groups.

Solution

One way to generate a plot for this exercise is to use the idea of creating arrays and lists so that we can loop over their elements.

We start by generating a dataframe object, using the reverse order of the categorical variables in the `crosstab` function.

```
attr_edu = pd.crosstab(df.Education_Level, df.Attrition_Flag)
attr_edu
```

1 to 7 of 7 entries Filter

Education_Level	Attrited Customer	Existing Customer
College	154	859
Doctorate	95	356
Graduate	487	2641
High School	306	1707
Post-Graduate	92	424
Uneducated	237	1250
Unknown	256	1263

Show 25 per page

We save the column values again.

```
groups = attr_edu.columns.tolist()
groups
```

```
['Attrited Customer', 'Existing Customer']
```

We can extract all the dataframe values as an array.

```
values = attr_edu.values
values
```

```
array([[ 154,  859],
       [  95,  356],
       [ 487, 2641],
       [ 306, 1707],
       [  92,  424],
```

```
            [ 237, 1250],
            [ 256, 1263]])
```

We also still need a list of all the education levels. This is stored as the index of the dataframe obejct.

```
levels = attr_edu.index.tolist()
levels
```

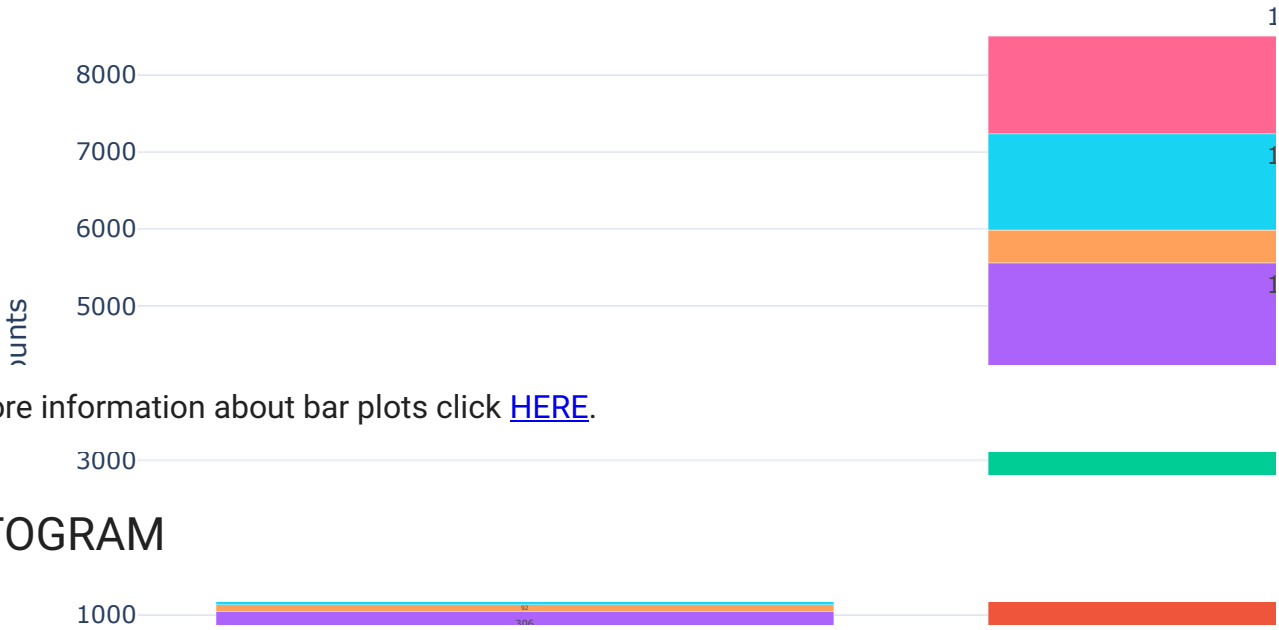
```
['College',
 'Doctorate',
 'Graduate',
 'High School',
 'Post-Graduate',
 'Uneducated',
 'Unknown']
```

Now we generate seven traces by looping over the seven elements in the (values) array and the (education level) list above. We could also have added each trace manualy. The idea is to use the power of the Python language to do all the hard work.

```
churn_edu_fig = go.Figure()

# Loop over elements in grps array object and nms list obejct
for i in range(len(values)):
  churn_edu_fig.add_trace(go.Bar(
      x=groups,
      y=values[i],
      text=values[i],
      textposition='outside',
      name=levels[i],
))

churn_edu_fig.update_layout(title='Educational level frequencies by customer gro

# Using alternate dictionary syntax
churn_edu_fig.update_layout({'xaxis':{'title':'Customer group'},
                             'yaxis':{'title':'Counts'},
                             'barmode':'group'})

churn_edu_fig.show()
```

Below, we turn this into a stacked bar chart.

```
churn_edu_fig = go.Figure()

# Loop over elements in grps array object and nms list obejct
for i in range(len(values)):
  churn_edu_fig.add_trace(go.Bar(
      x=groups,
      y=values[i],
      text=values[i],
      textposition='outside',
      name=levels[i],
))

churn_edu_fig.update_layout(title='Educational level frequencies by customer gro

# Using alternate dictionary syntax
churn_edu_fig.update_layout({'xaxis':{'title':'Customer group'},
                             'yaxis':{'title':'Counts'},
                             'barmode':'stack'})

churn_edu_fig.show()
```



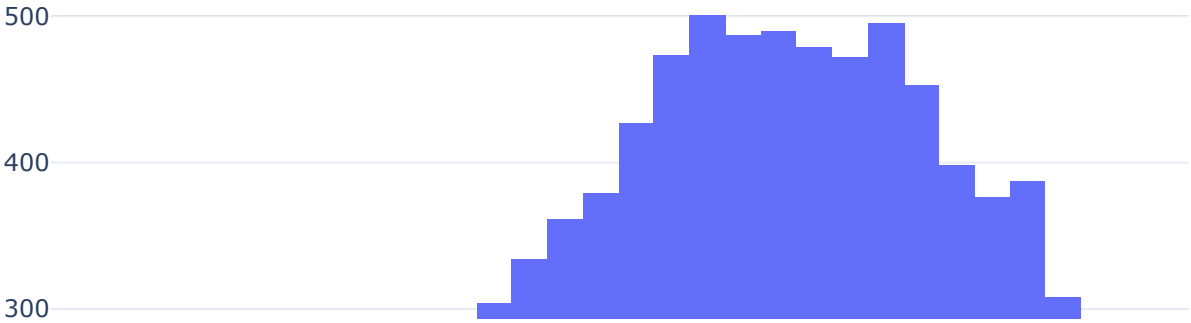
For more information about bar plots click HERE.

HISTOGRAM

A histogram is used for continuous numerical variables. By creating bins, we can count how many times a value in that interval appears. Binning is a method of dividing the range of values for the variable up into intervals and coonting how many times values *fall within* each interval.

Histograms are great at showing us the distribution of the data. Below, we plot a histogram of the age of our customers. This time, we make use of the plotly express library. The first argument is the dataframe object and the `x` argument is assigned to the `Customer_Age` column in the dataframe.

```
age_hist = px.histogram(df,
                        x='Customer_Age')
age_hist.show()
```



The bin size was set by default at one year increments. Note that there are no gaps between the bars as with a bar chart. This indicates the fact that the variable is continuous numerical.

We can create a **stacked** histogram by using the `color` argument to point to another categorical variable, `Attrition_Flag` in this case. The `labels` argument is set to a dictionary. It can be used to change text aspects of the axes labels of the plot. First, we see a plot without this argument and then with it to see the differences.

```
age_hist = px.histogram(
    df,
    x='Customer_Age',
    color='Attrition_Flag',
    title='Histogram of customer ages',
    opacity=0.7,
    marginal='rug'
)

age_hist.show()
```

Histogram of customer ages

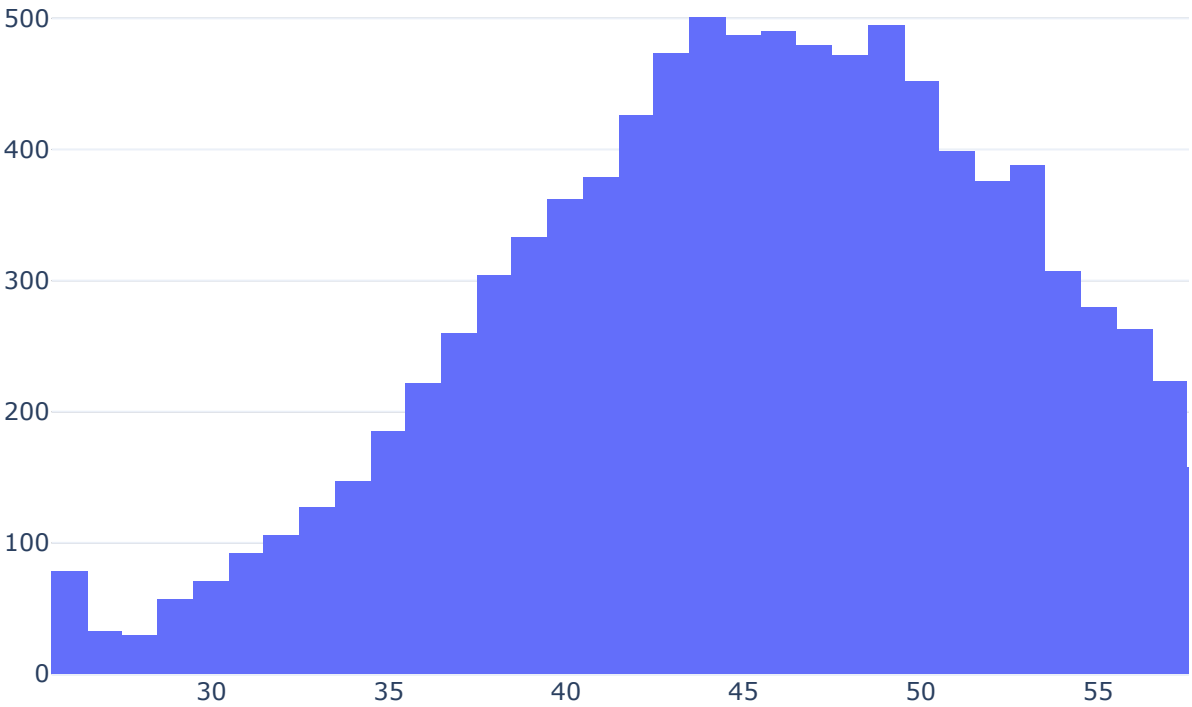
Note the `Customer_Age` title of the `x` axis, taken from the columns name. The dictionary is used below to replace this value. This is only availabe in the express module.

```
age_hist = px.histogram(
    df,
    x='Customer_Age',
    color='Attrition_Flag',
    title='Histogram of customer ages',
    opacity=0.7,
    marginal='rug',
    labels={
        'Attrition_Flag':'Customer group',
        'Customer_Age':'Customer age'
    }
)

age_hist.show()
```

The rug plot shows the actual values as small marks at the top of the plot. Since the values overlap (many customers with the same age), we see that the rug plot does not give us a good idea of the distribution of the data.

The graph objects module of plotly provides more felxibility.

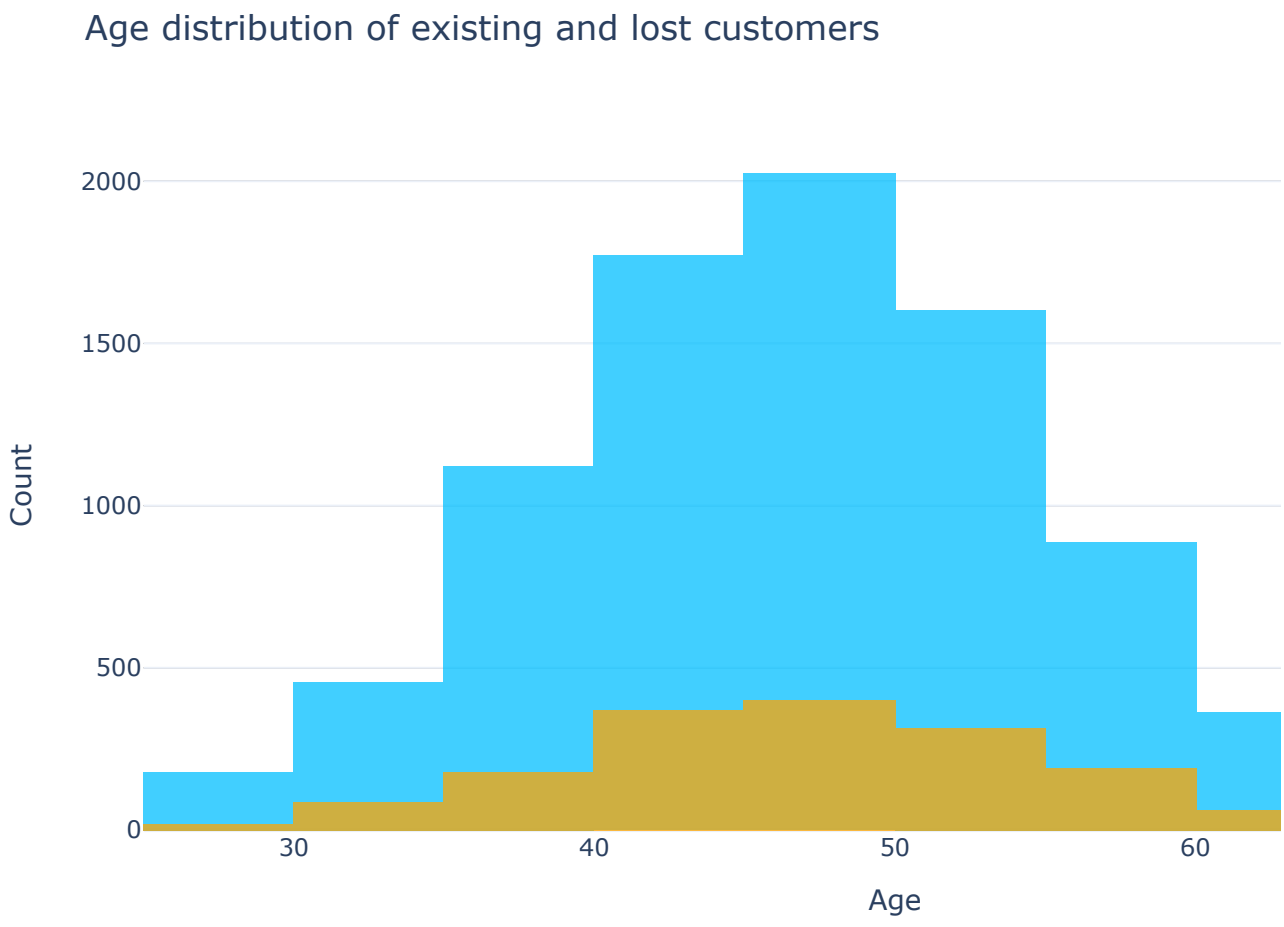
```
age_hist = go.Figure()

age_hist.add_trace(go.Histogram(
    x=df.Customer_Age
))

age_hist.show()
```



Let's look at the age distribution for each of the customer groups. Using the `barmode='overlay'` option, we create a **non-stacked** histogram. Below we set a start and end interval for the *x* axis and also a bins size of five year increments.

```
ages_cust = go.Figure()
```

```
ages_cust.add_trace(go.Histogram(
    x=df[df.Attrition_Flag == 'Existing Customer']['Customer_Age'],
    name='Existing customer',
    marker_color='deepskyblue',
    xbins=dict(start=25,
               end=80,
               size=5)
))
ages_cust.add_trace(go.Histogram(
    x=df[df.Attrition_Flag == 'Attrited Customer']['Customer_Age'],
    name='Lost customer',
    marker_color='orange',
    xbins=dict(start=25,
               end=80,
               size=5)
))

ages_cust.update_layout(barmode='overlay',
                        title='Age distribution of existing and lost customers
                        xaxis=dict(title='Age'),
                        yaxis=dict(title='Count'))

ages_cust.update_traces(opacity=0.75)

ages_cust.show()
```



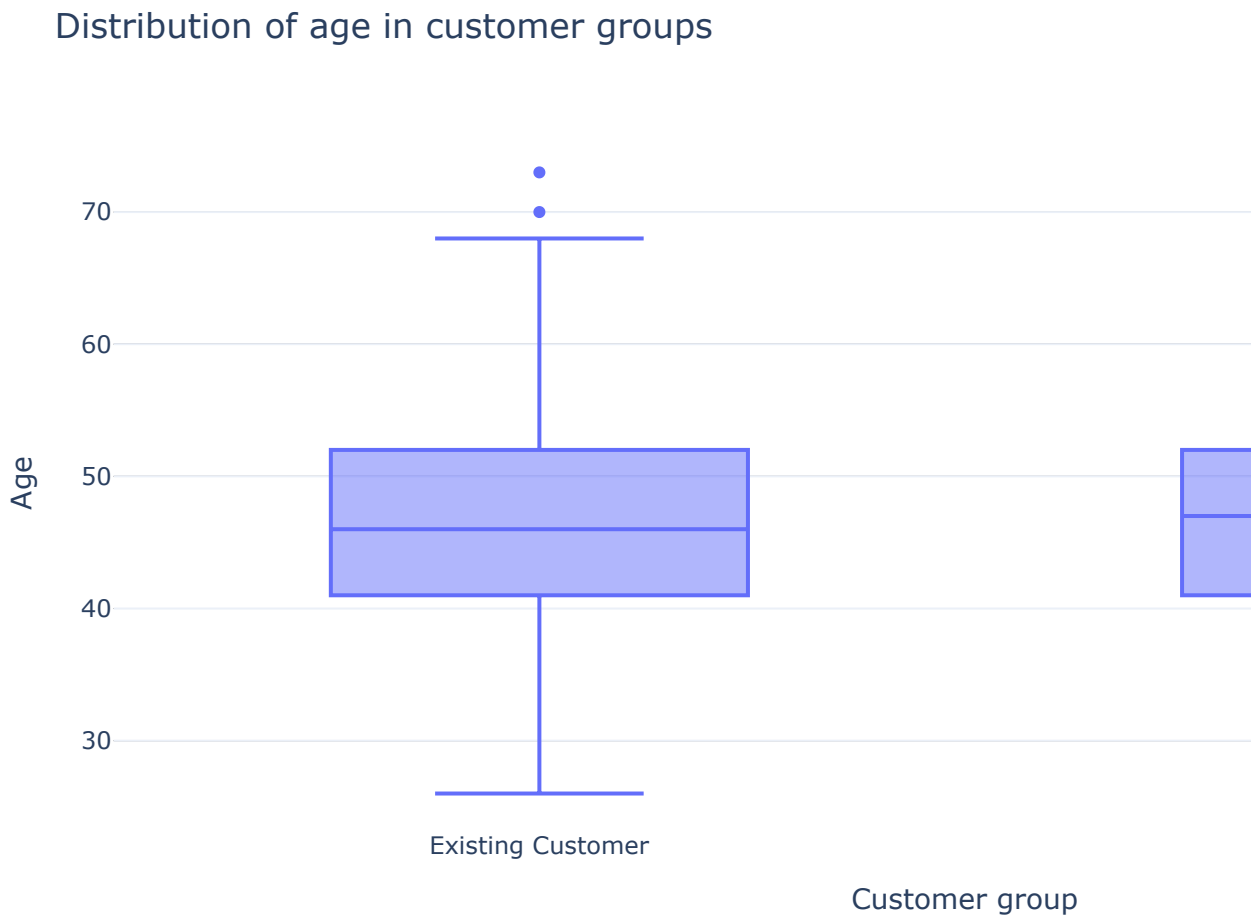
For more information on histograms from plotly click HERE.

BOX PLOT

Box-and-whisker plots are another visual representation of the distribution of a continuous numerical variable.

Below, we create a box plot of the ages of each customer group.

```
# Simple box plots using express
ages_churn_box_px = px.box(
    df,
    x='Attrition_Flag',
    y='Customer_Age',
    title='Distribution of age in customer groups',
    labels={'Customer_Age':'Age', 'Attrition_Flag':'Customer group'})
ages_churn_box_px.show()
```



With the `Box` function in the graph_objects module, we can take more control over each trace

```
# Extracting list objects
exis_age = df[df.Attrition_Flag == 'Existing Customer']['Customer_Age'].to_list(
churn_age = df[df.Attrition_Flag == 'Attrited Customer']['Customer_Age'].to_list
```

Now we create separate traces for each group. Code comments highlight the arguments.

```
# Adding separate traces and configuration
ages_churn_box = go.Figure()

ages_churn_box.add_trace(go.Box(
    y=exis_age,
    name='Existing customers',
    marker_color='green',
    boxmean=True, # Add a mean as a dotted line
    boxpoints='all' # Add all the age values as dots next to the box
))

ages_churn_box.add_trace(go.Box(
    y=churn_age,
    name='Lost customers',
    marker_color='red',
    boxmean='sd', # Add a mean and standard deviation as dotted lines
    boxpoints='all'
))

ages_churn_box.update_layout(title='Distribution of ages',
                             xaxis={'title':'Group'},
                             yaxis={'title':'Count'})

ages_churn_box.show()
```

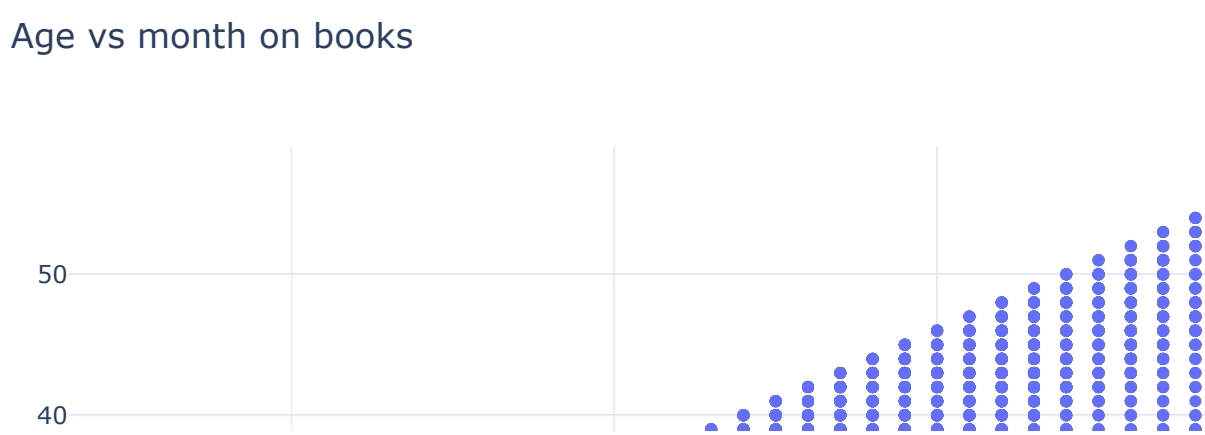
You can learn more about bo-and-whisker plots HERE.

SCATTER PLOT

Scatter plots allow us to view the difference between observations with respect to continuous numerical variables. If we restrict ourselves to two variables, each dot on a plane with and *x* and a *y* axis can represent the values for a pair of continuous numerical variables for each observation.

- Create a scatter plot of age (x axis) vs systolic blood pressure (y axis).

```
age_mob = go.Figure()

age_mob.add_trace(go.Scatter(
    x=df.Customer_Age,
    y=df.Months_on_book,
    mode='markers'
))

age_mob.update_layout(title="Age vs month on books",
                      xaxis=dict(title="Age"),
                      yaxis=dict(title="Months"))

age_mob.show()
```

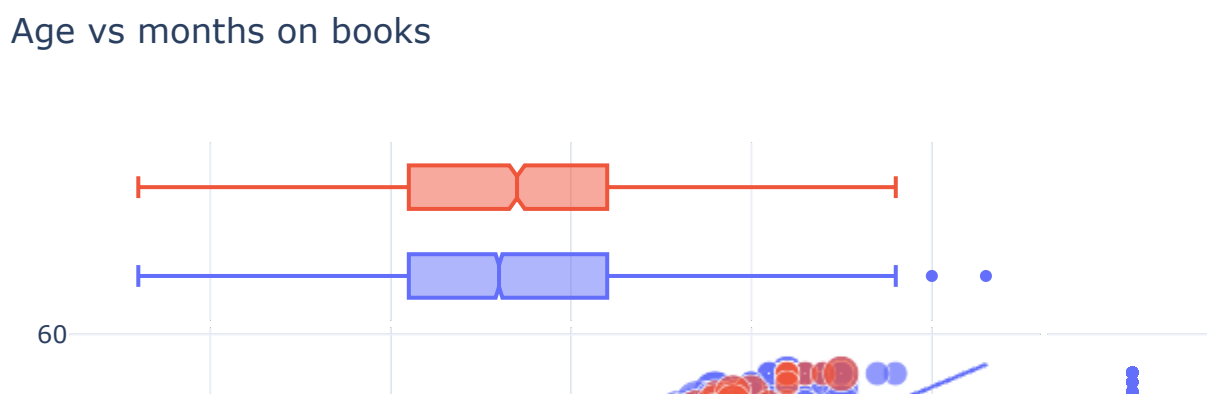


We can add a third variable in the form of the size of the marker. Below, we do just that and compare the age and months on books of the customers. To this, we add the numer of dependents as the size of the marker. We also split the customers by the sample space elements of the `Attrition_Flag` variable.

We can even further and add box-and-whisker plots, together with a linear model (using ordinary-least-squares).


```
# Box and whisker plots of the variables
age_mob_group_px = px.scatter(
    df,
    x='Customer_Age',
    y='Months_on_book',
    size='Dependent_count',  # Determines size of markers
    color='Attrition_Flag',  # Group by this variable
    marginal_y='box',
    marginal_x='box',
    trendline='ols',
    title='Age vs months on books',
    labels={'Customer_Age':'Age', 'Months_on_book':'Months'})  # Over-write colu
age_mob_group_px.show()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: Futu

pandas.util.testing is deprecated. Use the functions in the public API at pan
```



Instead of box-and-whisker plots, we can also add rug plots and histograms.

```
# Rug plot and histogram as marginal plots
px.scatter(
    df,
    x='Customer_Age',
    y='Months_on_book',
    size='Dependent_count',  # Determines size of markers
    color='Attrition_Flag',
    marginal_y='histogram',
    marginal_x='rug',
    trendline='ols',
    title='Comparing age vs months on books for each of the two groups',
    labels={'Customer_Age':'Age', 'Months_on_book':'Months'}).show()
```

Comparing age vs months on books for each of the two groups

Instead of the size of the marker as visual indicator of the third variable, we can also use color. Below, we also add the `facet_col` argument. This creates individual plots based on the sample space elements of a categorical variable and positions them as columns.

- Separate scatter plots
- *Third dimension* by color scale

```
px.scatter(
    df,
    x='Customer_Age',
    y='Months_on_book',
    color='Dependent_count',
    facet_col='Attrition_Flag',
    trendline='ols',
    title='Sperate scatter plots per group',
    labels={'Attrition_Flag':'Group', 'Months_on_book':'Months', 'Customer_Age':
    color_continuous_scale=px.colors.sequential.Viridis).show();
```

(Hover on the trendline to see the regression model and the coefficient of determination, R^2. We will learn more about this when we look at linear regression.)

You can learn more about scatter plots HERE.

TIME SERIES DATA

Time series contain dates or sequential data of some form. Here, we use the Australian rainfall data set. We start by investigating the difference in maximum daily temperatures between Darwin and Hobart.

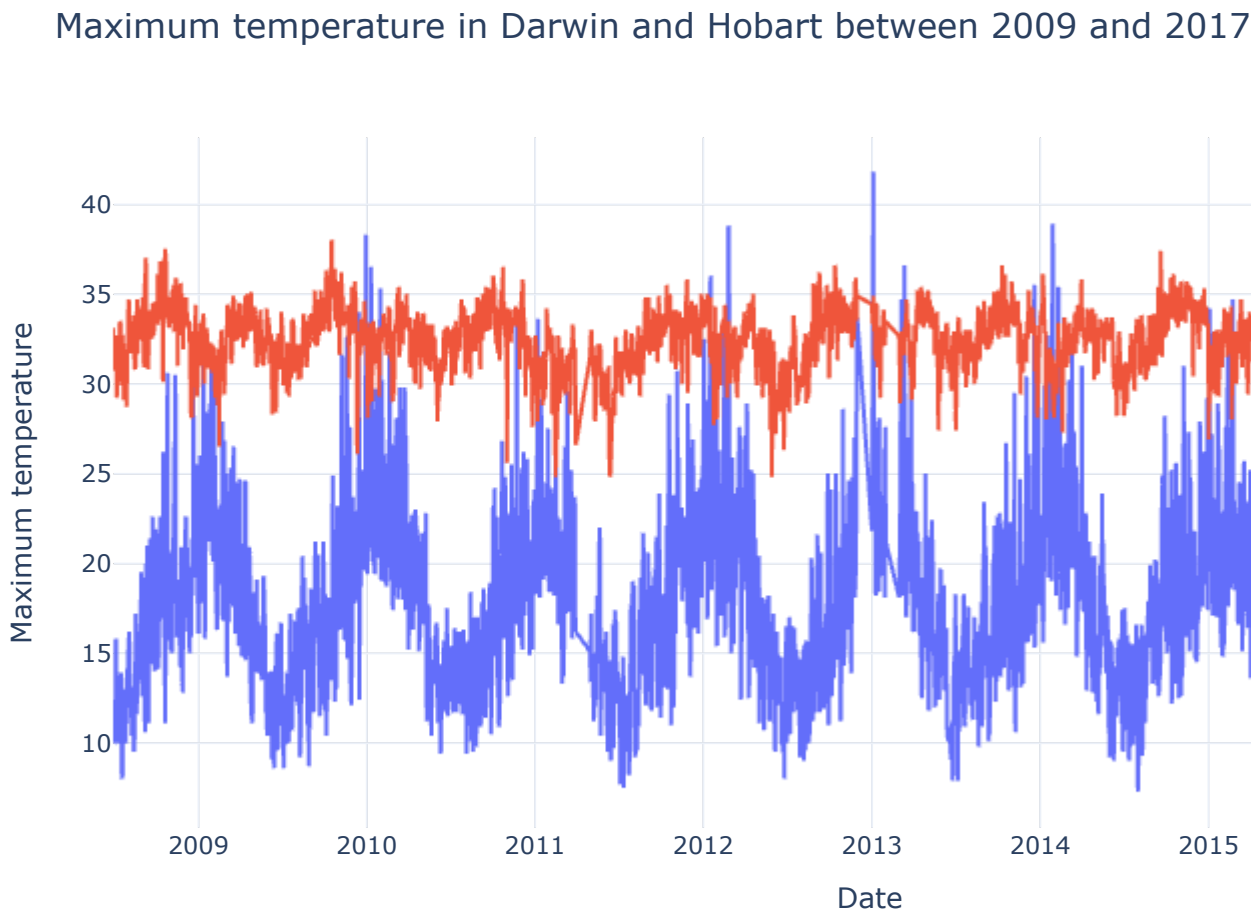
We start by using the `info` method to learn more about the dataframe object.

```
1 rain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object 
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object 
 10  WindDir3pm     141232 non-null  object 
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object 
 22  RainTomorrow   142193 non-null  object 
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

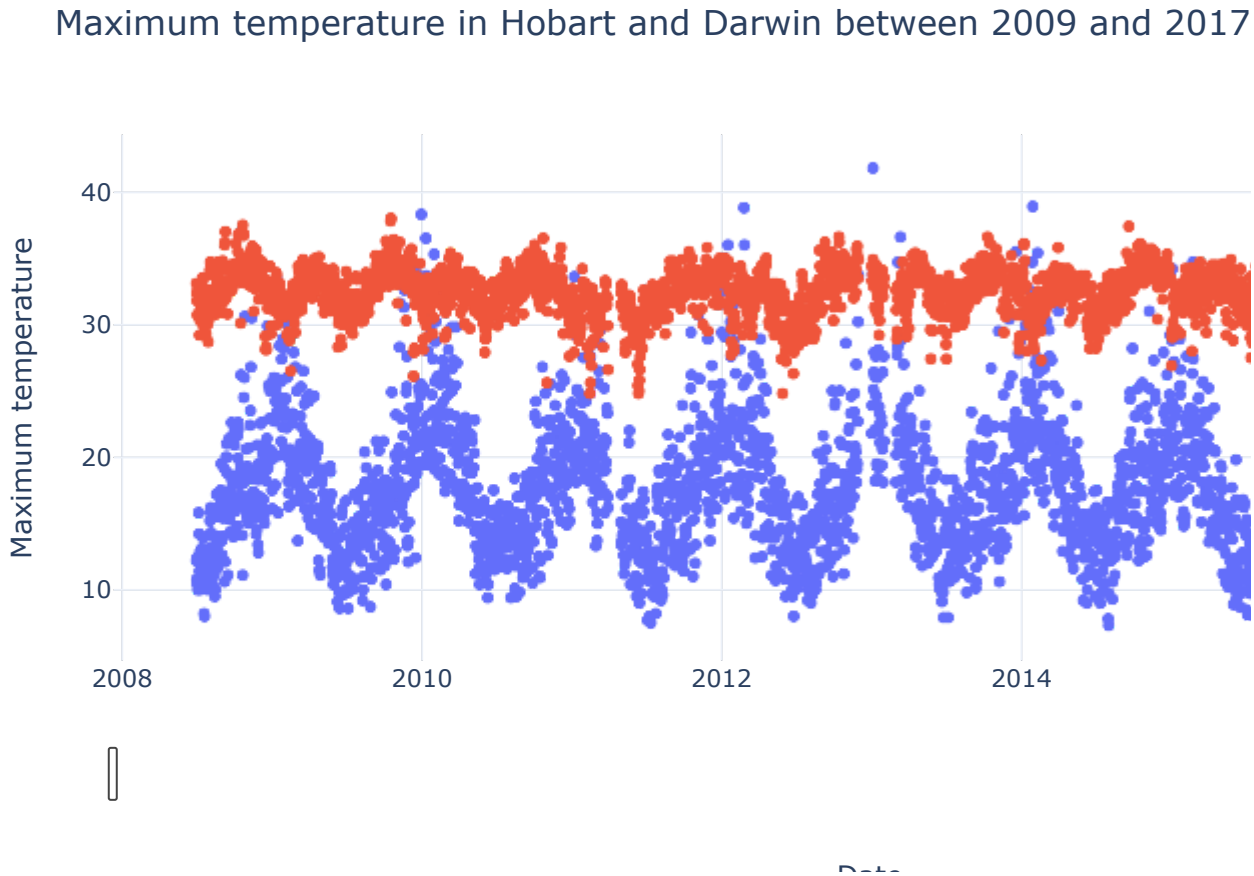
The data points for `MaxTemp`, the maximum temperature, are connect by line segments using the `line` function in the express module.

```
px.line(
    rain[(rain.Location == 'Darwin') | (rain.Location == 'Hobart')],
    x='Date',
    y='MaxTemp',
    color='Location',
    title='Maximum temperature in Darwin and Hobart between 2009 and 2017',
    labels={'MaxTemp':'Maximum temperature'}
)
```



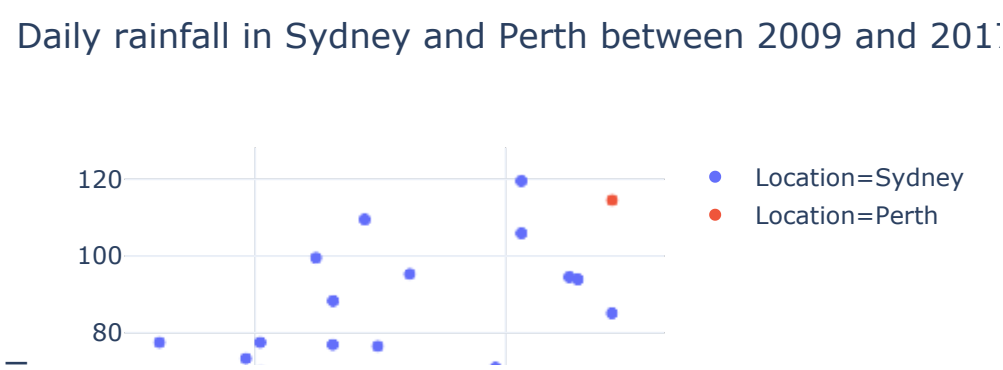
We can also view the individual data points as a scatter plot. We also add a range slider at the bottom of the plot.

```
px.scatter(
    rain[(rain.Location == 'Darwin') | (rain.Location == 'Hobart')],
    x='Date',
    y='MaxTemp',
    color='Location',
    title='Maximum temperature in Hobart and Darwin between 2009 and 2017',
    labels={'MaxTemp':'Maximum temperature'}
).update_xaxes(rangeslider_visible=True)
```



Lastly, let's look at the difference in daily rainfall between Perth in the West and Sydney in the East.

```
px.scatter(
    rain[(rain.Location == 'Perth') | (rain.Location == 'Sydney')],
    x='Date',
    y='Rainfall',
    color='Location',
    title='Daily rainfall in Sydney and Perth between 2009 and 2017'
).update_xaxes(rangeslider_visible=True)
```



You can learn more about plot for dates and time HERE.

CONCLUSION

There is so much more to learn about plotly and we will see new types of plots in later notebooks. Visit the homepage HERE. For references on all the arguments click HERE.

|| ||

1

Date

✓ 0s completed at 13:56