Basic Plotly Charts

Estimated time needed: 45 minutes

Objectives

After completing this lab, you will be able to:

- Use plotly graph objects and plotly express libraries to plot different types of charts
- Create interesting visualizations on Airline Reporting Carrier On-Time Performance Dataset

Plotly graph objects and Plotly express libraries to plot different types of charts

Plotly Libraries

plotly.graph_objects: This is a low level interface to figures, traces and layout. The Plotly graph objects module provides an automatically generated hierarchy of classes (figures, traces, and layout) called graph objects. These graph objects represent figures with a top-level class plotly.graph_objects.Figure.

plotly.express: Plotly express is a high-level wrapper for Plotly. It is a recommended starting point for creating the most common figures provided by Plotly using a simpler syntax. It uses graph objects internally. Now let us use these libraries to plot some charts We will start with plotly_graph_objects to plot line and scatter plots

> Note: You can hover the mouse over the charts whenever you want to view any statistics in the visualization charts

Exercise I: Get Started with Different Chart types in Plotly

In [1]:
```
# import piplite

# await piplite.install(['nbformat', 'plotly'])
```

In [2]:
```
# Import required libraries
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
```

1. Scatter Plot:

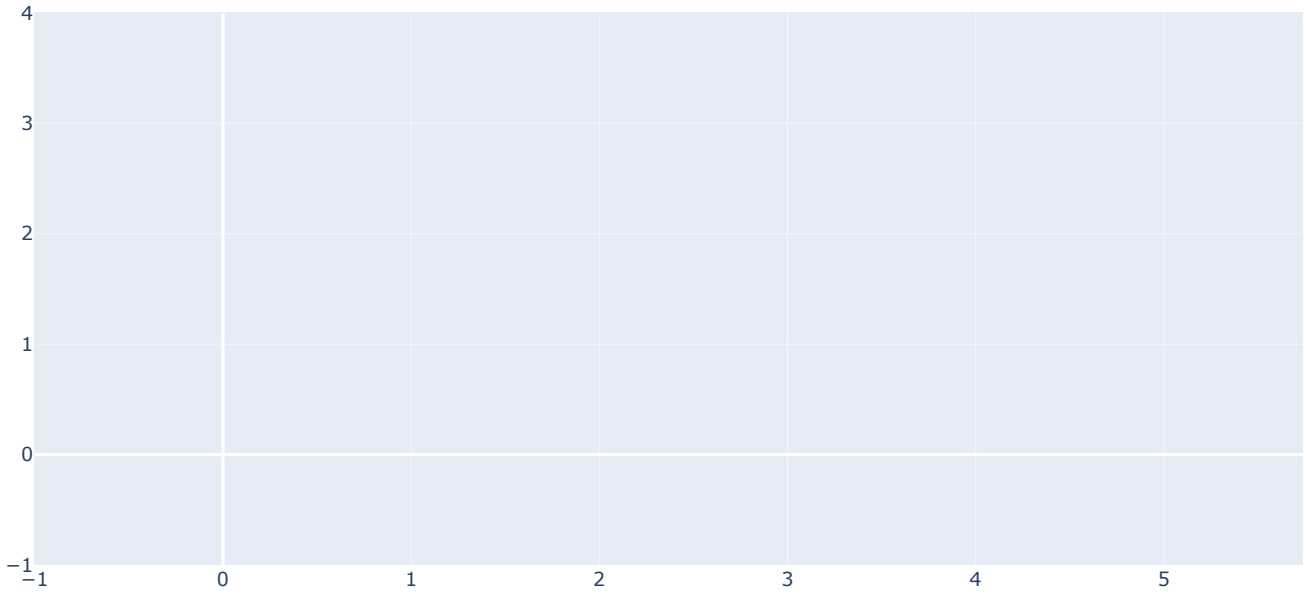
A scatter plot shows the relationship between 2 variables on the x and y-axis. The data points here appear scattered when plotted on a two-dimensional plane. Using scatter plots, we can create exciting visualizations to express various relationships, such as:

- Height vs weight of persons
- Engine size vs automobile price
- Exercise time vs Body Fat

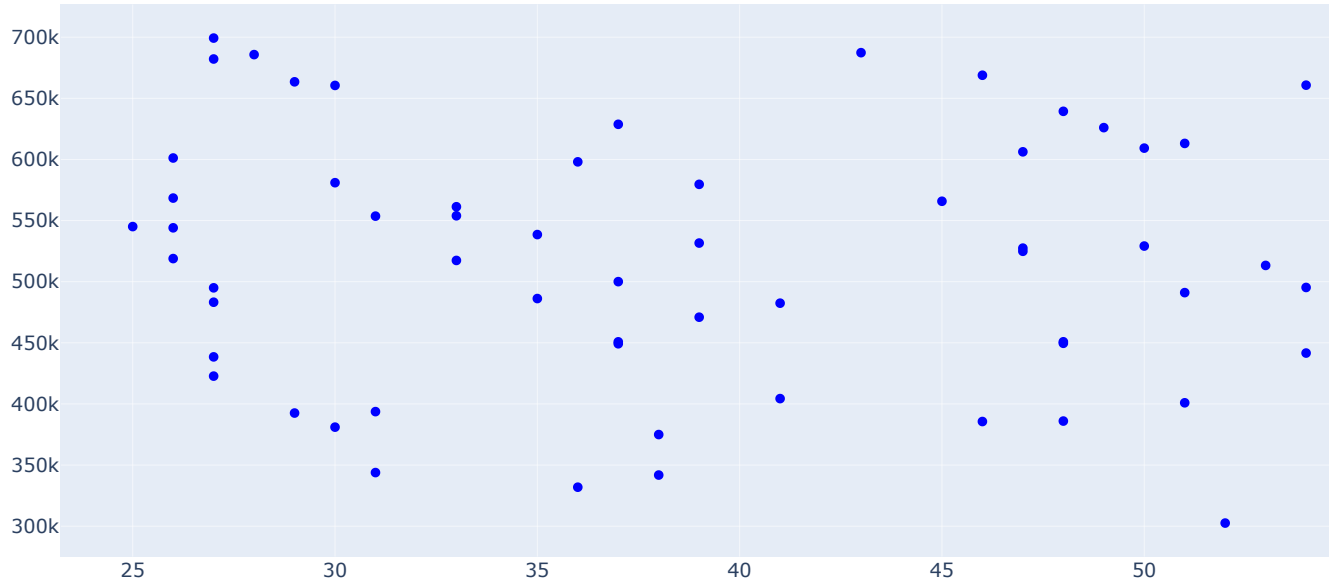
In [3]:
```
##Example 1: Let us illustrate the income vs age of people in a scatter plot

age_array=np.random.randint(25,55,60)
# Define an array containing salesamount values
income_array=np.random.randint(300000,700000,3000000)
```

In [4]:
```
##First we will create an empty figure using go.Figure()
fig=go.Figure()
fig
```

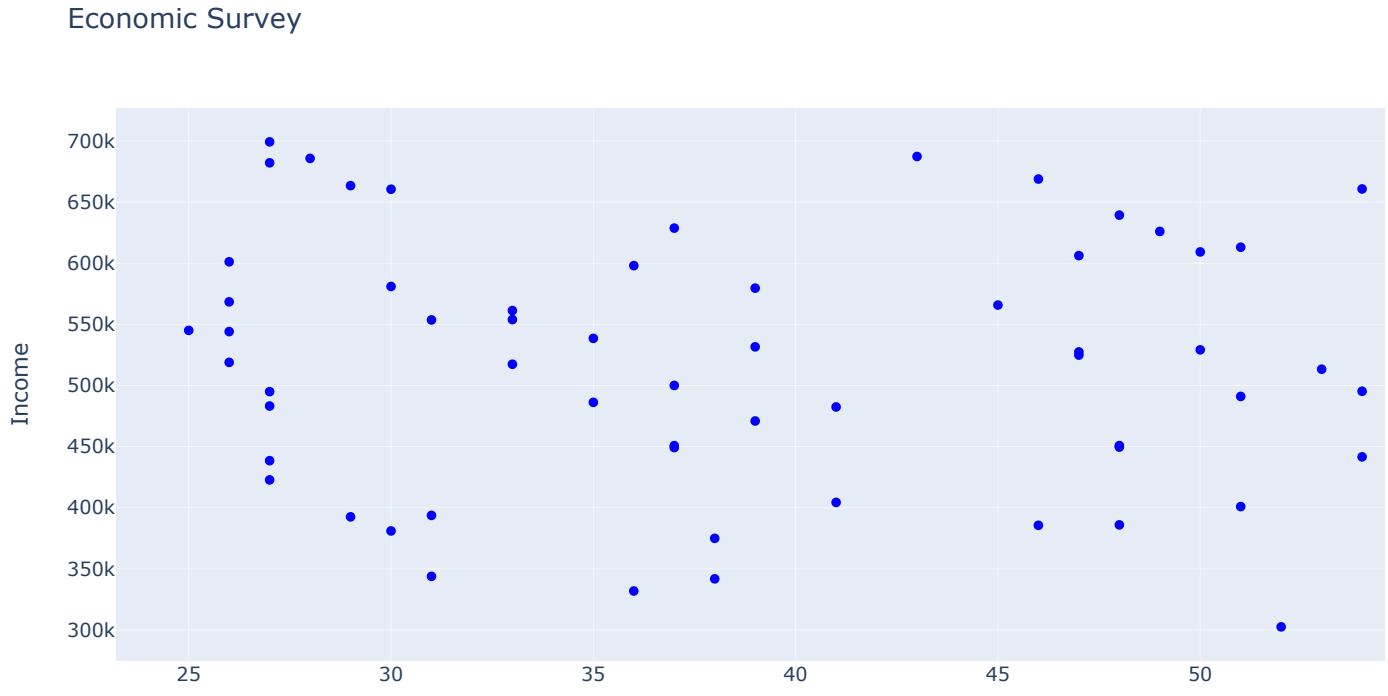


In [5]:
```
#Next we will create a scatter plot by using the add_trace function and use the go.scatter() function within it
# In go.Scatter we define the x-axis data,y-axis data and define the mode as markers with color of the marker as b
fig.add_trace(go.Scatter(x=age_array, y=income_array, mode='markers', marker=dict(color='blue')))
```



However in the previous output title, x-axis and y-axis labels are missing. Let us use the update_layout function to update the title and labels.

In [6]:
```
## Here we update these values under function attributes such as title,xaxis_title and yaxis_title
fig.update_layout(title='Economic Survey', xaxis_title='Age', yaxis_title='Income')
# Display the figure
fig.show()
```



Inferences:

From the above plot we find that the Income of a person is not correlated with age. We find that as the age increases the income may or not decrease.

2. Line Plot:

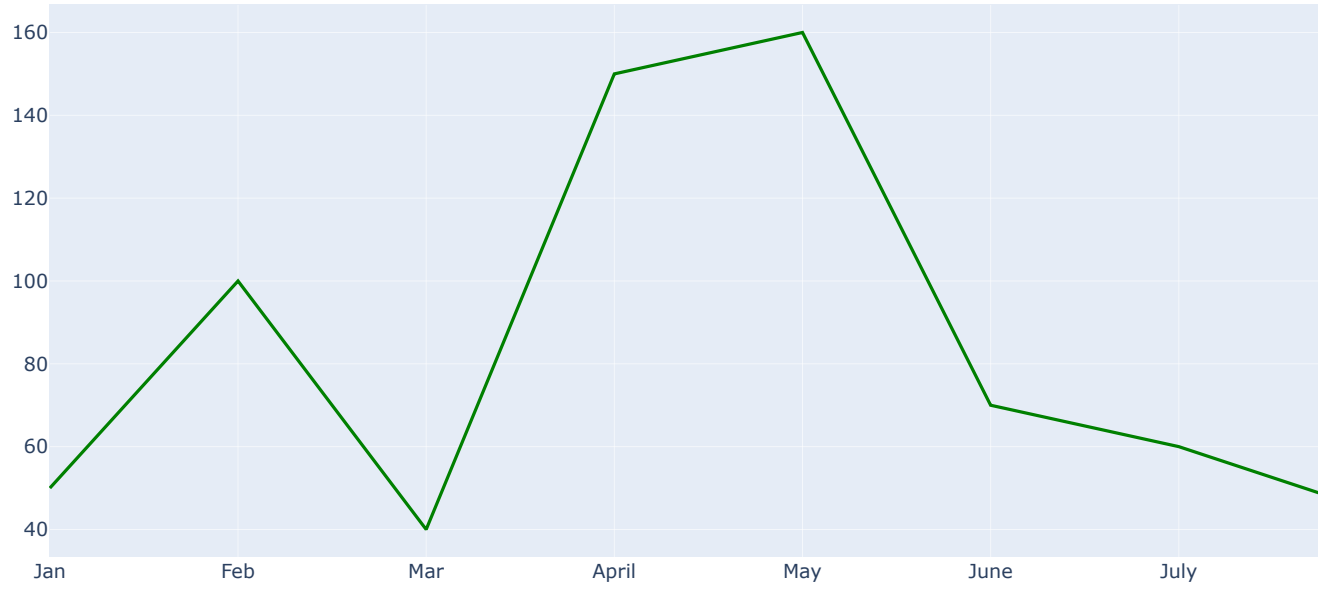
A line plot shows information that changes continuously with time. Here the data points are connected by straight lines. Line plots are also plotted on a two dimensional plane like scatter plots. Using line plots, we can create exciting visualizations to illustrate:

- Annual revenue growth
- Stock Market analysis over time
- Product Sales over time

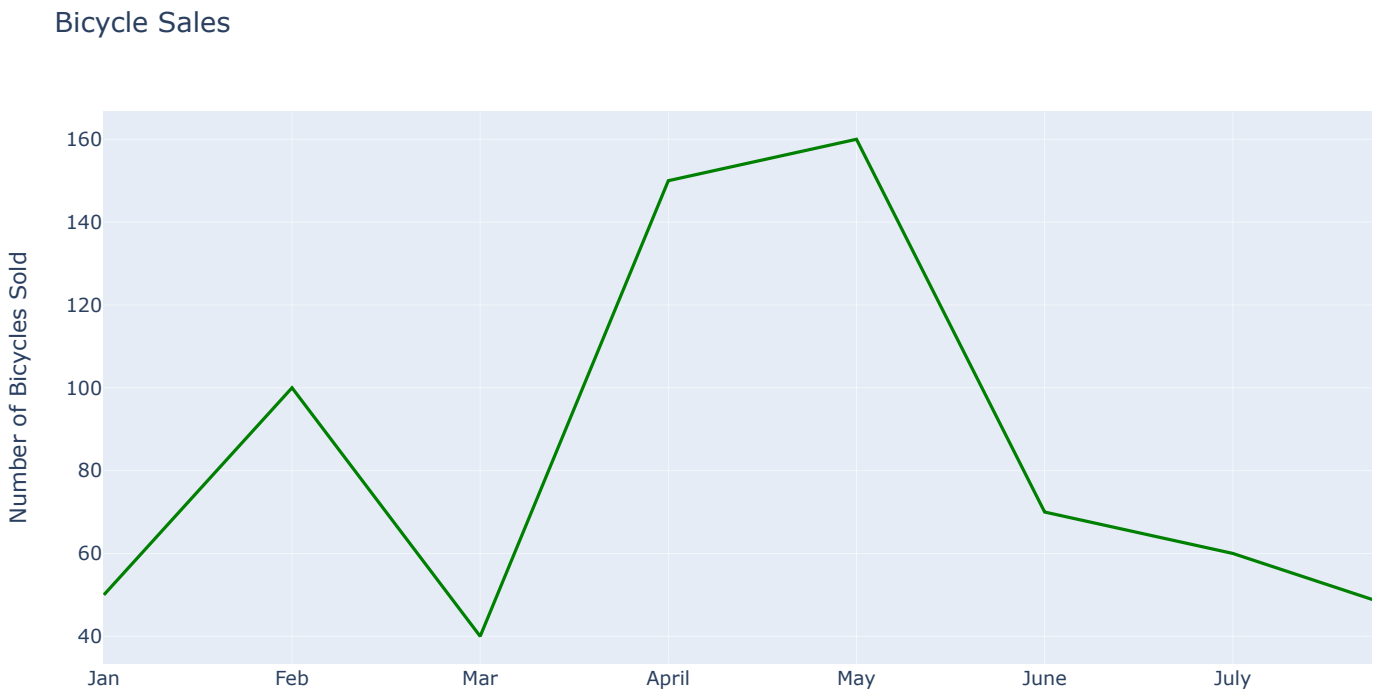
In [7]:
```
##Example 2: Let us illustrate the sales of bicycles from Jan to August last year using a line chart
# Define an array containing numberofbicyclessold
numberofbicyclessold_array=[50,100,40,150,160,70,60,45]
# Define an array containing months
months_array=["Jan","Feb","Mar","April","May","June","July","August"]
```

In [8]:
```
##First we will create an empty figure using go.Figure()
fig=go.Figure()
```

In [9]:
```
#Next we will create a line plot by using the add_trace function and use the go.scatter() function within it
# In go.Scatter we define the x-axis data,y-axis data and define the mode as lines with color of the marker as gre
fig.add_trace(go.Scatter(x=months_array, y=numberofbicyclessold_array, mode='lines', marker=dict(color='green')))
```



In [10]:
```
## Here we update these values under function attributes such as title,xaxis_title and yaxis_title
fig.update_layout(title='Bicycle Sales', xaxis_title='Months', yaxis_title='Number of Bicycles Sold')
# Display the figure
fig.show()
```



Inferences:

From the above plot we find that the sales is the highest in the month of May and then there is a decline in sales.

We will now use plotly express library to plot the other graphs

3.Bar Plot:

A bar plot represents categorical data in rectangular bars. Each category is defined on one axis, and the value counts for this category are represented on another axis. Bar charts are generally used to compare values.We can use bar plots in visualizing:

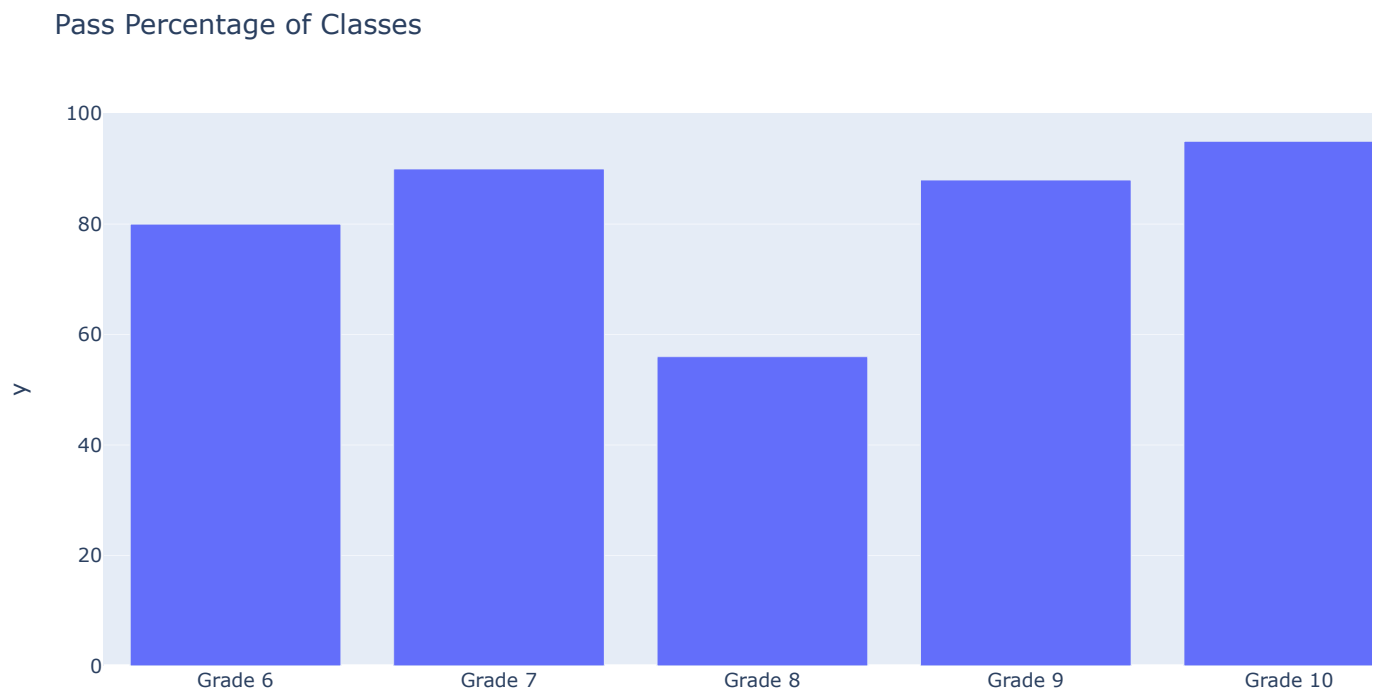
- Pizza delivery time in peak and non peak hours
- Population comparison by gender
- Number of views by movie name

In [11]:
```
##Example 3: Let us illustrate the average pass percentage of classes from grade 6 to grade 10

# Define an array containing scores of students
score_array=[80,90,56,88,95]
# Define an array containing Grade names
grade_array=['Grade 6','Grade 7','Grade 8','Grade 9','Grade 10']
```

In plotly express we set the axis values and the title within the same function call `px.<graphtype>(x=<xaxis value source>,y=<y-axis value source>,title=<appropriate title as a string>)` .In the below code we use `px.bar( x=grade_array, y=score_array, title='Pass Percentage of Classes')` .

In [12]:
```
# Use plotly express bar chart function px.bar.Provide input data, x and y axis variable, and title of the chart.
# This will give average pass percentage per class
fig = px.bar( x=grade_array, y=score_array, title='Pass Percentage of Classes')
fig.show()
```



Inferences:

From the above plot we find that Grade 8 has the lowest pass percentage and Grade 10 has the highest pass percentage

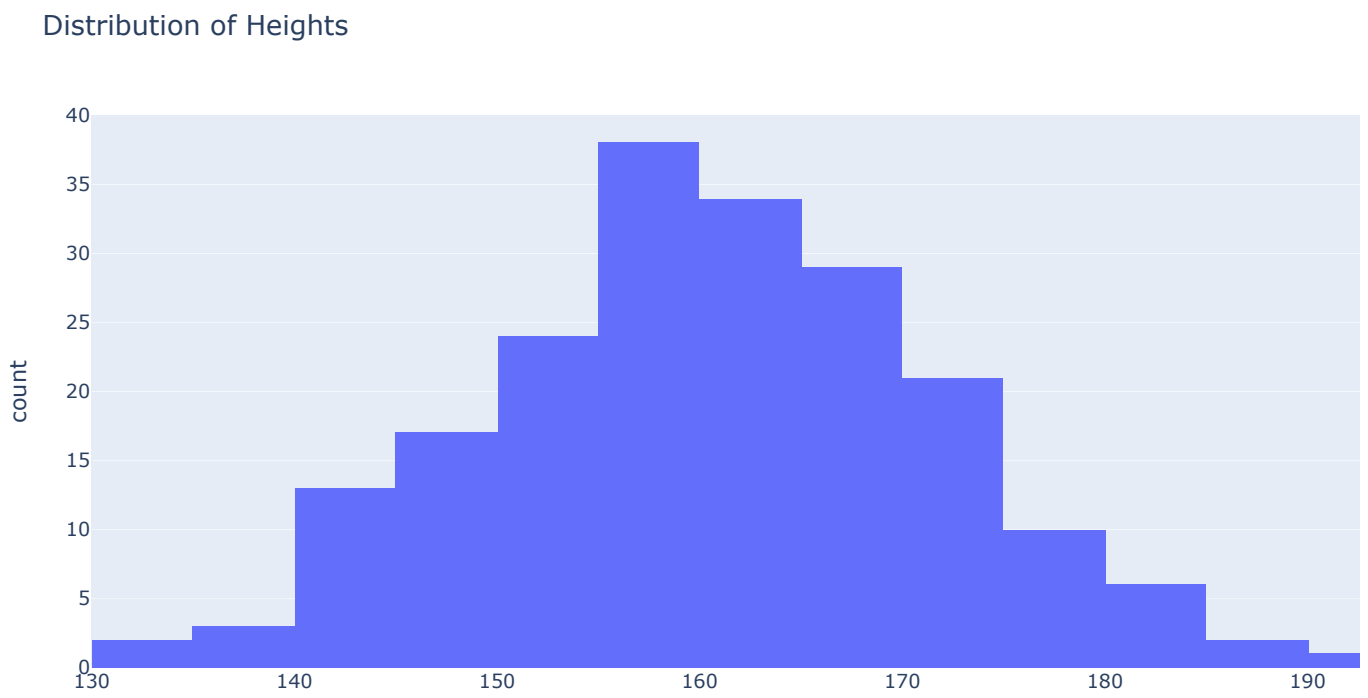
4.Histogram:

A histogram is used to represent continuous data in the form of bar. Each bar has discrete values in bar graphs, whereas in histograms, we have bars representing a range of values. Histograms show frequency distributions. We can use histograms to visualize:

- Students marks distribution
- Frequency of waiting time of customers in a Bank

In [13]:
```
##Example 4: Let us illustrate the distribution of heights of 200 people using a histogram

import numpy as np
#Here we will concentrate on heights which are 160 and the standard deviation is 11
heights_array = np.random.normal(160, 11, 200)
## Use plotly express histogram chart function px.histogram.Provide input data x to the histogram
fig = px.histogram(x=heights_array,title="Distribution of Heights")
fig.show()
```



Inferences

From this we can analyze that there are around 2 people who are at the height of 130cm and 45 people at the height of 160 cm

5. Bubble Plot:

A bubble plot is used to show the relationship between 3 or more variables. It is an extension of a scatter plot. Bubble plots are ideal for visualizing:

- Global Economic position of Industries
- Impact of viruses on Diseases

In [14]:
```
##Example 4: Let us illustrate crime statistics of US cities with a bubble chart

#Create a dictionary having city,numberofcrimes and year as 3 keys
crime_details = {
    'City' : ['Chicago', 'Chicago', 'Austin', 'Austin','Seattle','Seattle'],
    'Numberofcrimes' : [1000, 1200, 400, 700,350,1500],
    'Year' : ['2007', '2008', '2007', '2008','2007','2008'],
}

# create a Dataframe object with the dictionary
df = pd.DataFrame(crime_details)

df
```

Out[14]:

	City	Numberofcrimes	Year
0	Chicago	1000	2007
1	Chicago	1200	2008
2	Austin	400	2007
3	Austin	700	2008
4	Seattle	350	2007
5	Seattle	1500	2008

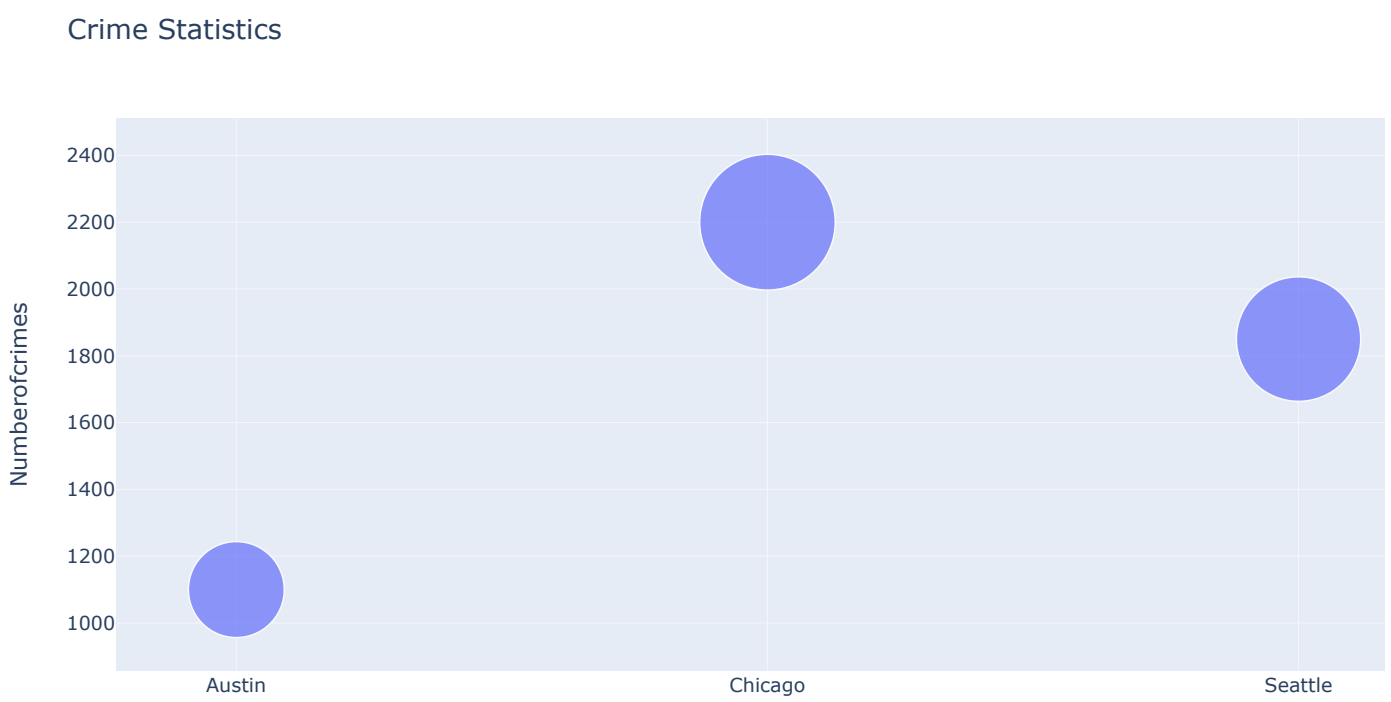
In [15]:
```
## Group the number of crimes by city and find the total number of crimes per city
bub_data = df.groupby('City')['Numberofcrimes'].sum().reset_index()
```

In [16]:
```
##Display the grouped dataframe
bub_data
```

Out[16]:

	City	Numberofcrimes
0	Austin	1100
1	Chicago	2200
2	Seattle	1850

In [17]:
```
## Bubble chart using px.scatter function with x ,y and size varibles defined.Title defined as Crime Statistics
fig = px.scatter(bub_data, x="City", y="Numberofcrimes", size="Numberofcrimes",
                 hover_name="City", title='Crime Statistics', size_max=60)
fig.show()
```



Inferences

The size of the bubble in the bubble chart indicates that Chicago has the highest crime rate when compared with the other 2 cities.

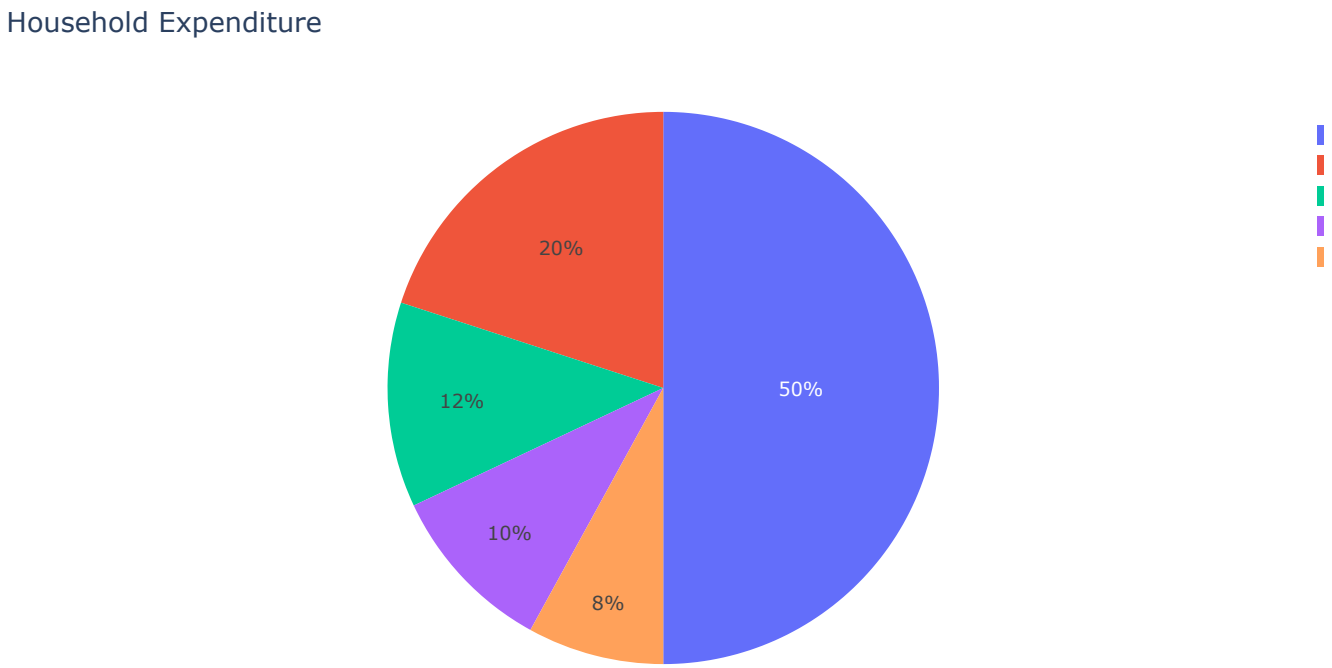
6.Pie Plot:

A pie plot is a circle chart mainly used to represent proportion of part of given data with respect to the whole data. Each slice represents a proportion and on total of the proportion becomes a whole. We can use bar plots in visualizing:

- Sales turnover percentatge with respect to different products
- Monthly expenditure of a Family

In [18]:
```
## Monthly expenditure of a family
# Random Data
exp_percent= [20, 50, 10,8,12]
house_holdcategories = ['Grocery', 'Rent', 'School Fees','Transport','Savings']
```

In [19]:
```
# Use px.pie function to create the chart. Input dataset.
# Values parameter will set values associated to the sector. 'exp_percent' feature is passed to it.
# labels for the sector are passed to the `house hold categoris` parameter.
fig = px.pie(values=exp_percent, names=house_holdcategories, title='Household Expenditure')
fig.show()
```



Inferences

From this pie chart we can find that the family expenditure is maximum for rent.

7.Sunburst Charts:

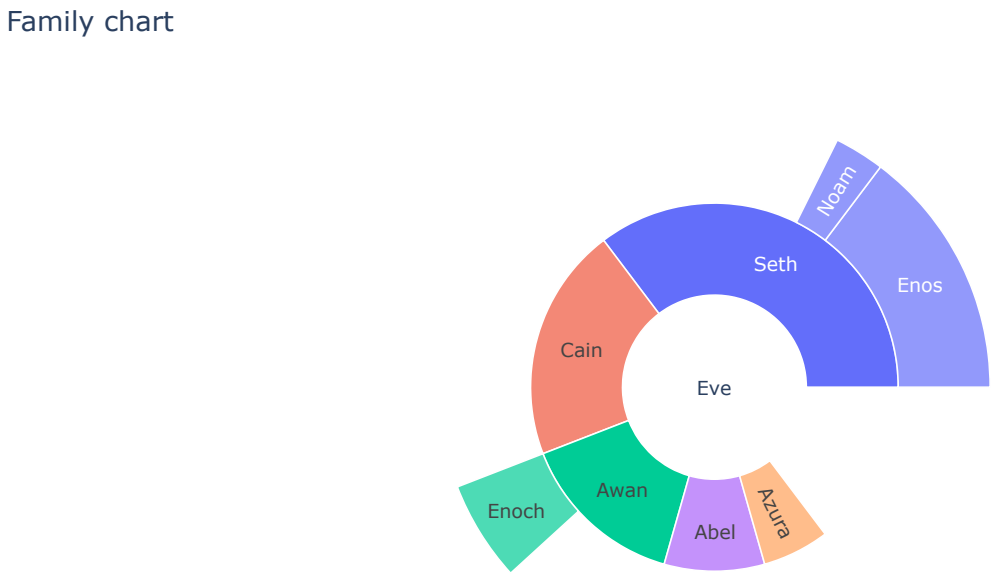
Sunburst charts represent hierarchial data in the form of concentric circles. Here the innermost circle is the root node which defines the parent, and then the outer rings move down the hierarchy from the centre. They are also called radial charts.We can use them to plot

- Worldwide mobile Sales where we can drill down as follows:
  - innermost circle represents total sales
  - first outer circle represents continentwise sales
  - second outer circle represents countrywise sales within each continent
- Disease outbreak hierarchy
- Real Estate Industrial chain

In [20]:
```
##Example 4: Let us illustrate plot the

#Create a dictionary having a set of people represented by a character array and the parents of these characters r
## array and the values are the values associated to the vectors.
data = dict(
    character=["Eve", "Cain", "Seth", "Enos", "Noam", "Abel", "Awan", "Enoch", "Azura"],
    parent=["", "Eve", "Eve", "Seth", "Seth", "Eve", "Eve", "Awan", "Eve" ],
    value=[10, 14, 12, 10, 2, 6, 6, 4, 4])

fig = px.sunburst(
    data,
    names='character',
    parents='parent',
    values='value',
    title="Family chart"
)
fig.show()
```



Inferences

It is found that here the innermost circle **Eve** represents the parent and the second outer circle represents his childrent **Cain**,**Seth** and so on.Further the outermost circle represents his grandchildren **Enoch** and **Enos**

II- Practice Exercises: Apply your Plotly Skills to an Airline Dataset

The Reporting Carrier On-Time Performance Dataset contains information on approximately 200 million domestic US flights reported to the United States Bureau of Transportation Statistics. The dataset contains basic information about each flight (such as date, time, departure airport, arrival airport) and, if applicable, the amount of time the flight was delayed and information about the reason for the delay. This dataset can be used to predict the likelihood of a flight arriving on time.

Preview data, dataset metadata, and data glossary here. (https://dax-cdn.cdn.appdomain.cloud/dax-airline/1.0.1/data-preview/index.html)

Read Data

In [21]:
```
# Read the airline data into pandas dataframe
# from js import fetch
# import io

URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-Skill
# resp = await fetch(URL)
# text = io.BytesIO((await resp.arrayBuffer()).to_py())

airline_data =  pd.read_csv(URL,
                            encoding = "ISO-8859-1",
                            dtype={'Div1Airport': str, 'Div1TailNum': str,
                                   'Div2Airport': str, 'Div2TailNum': str})

print('Data downloaded and read into a dataframe!')
```

```
Data downloaded and read into a dataframe!
```

In [22]:
```
# Preview the first 5 lines of the loaded data
airline_data.head()
```

Out[22]:

	Unnamed: 0	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	DOT_ID_Reporting_Airline	IATA_CODE_Reporting_
0	1295781	1998	2	4	2	4	1998-04-02	AS	19930	
1	1125375	2013	2	5	13	1	2013-05-13	EV	20366	
2	118824	1993	3	9	25	6	1993-09-25	UA	19977	
3	634825	1994	4	11	12	6	1994-11-12	HP	19991	
4	1888125	2017	3	8	17	4	2017-08-17	UA	19977	

5 rows × 110 columns

In [23]:
```
# Shape of the data
airline_data.shape
```

Out[23]:
```
(27000, 110)
```

In [24]:
```
# Randomly sample 500 data points. Setting the random state to be 42 so that we get same result.
data = airline_data.sample(n=500, random_state=42)
```

In [25]:
```
# Get the shape of the trimmed data
data.shape
```

Out[25]:
```
(500, 110)
```

It would be interesting if we visually capture details such as

- Departure time changes with respect to airport distance.
- Average Flight Delay time over the months
- Comparing number of flights in each destination state
- Number of flights per reporting airline
- Distrubution of arrival delay
- Proportion of distance group by month (month indicated by numbers)
- Hierarchical view in othe order of month and destination state holding value of number of flights

plotly.graph_objects¶

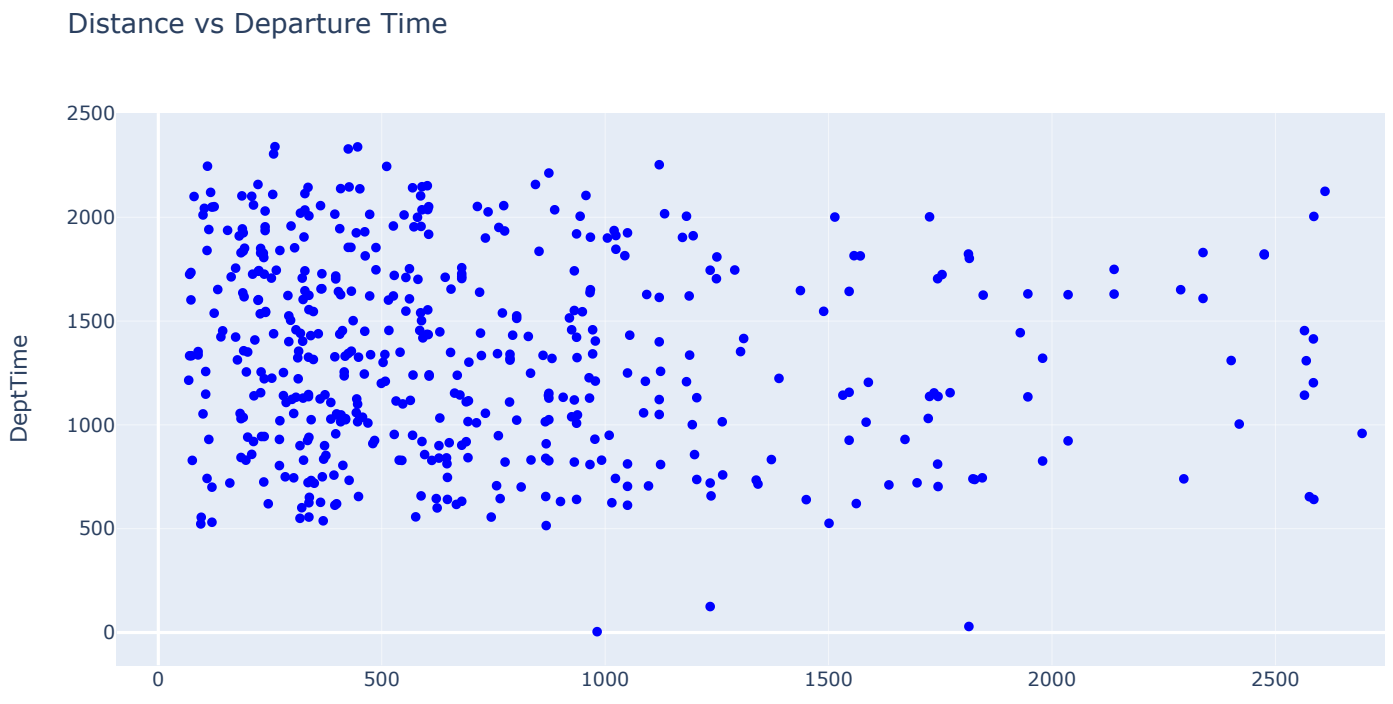
1. Scatter Plot

Let us use a scatter plot to represent departure time changes with respect to airport distance

This plot should contain the following

- Title as **Distance vs Departure Time**.
- x-axis label should be **Distance**
- y-axis label should be **DeptTime**
- **Distance** column data from the flight delay dataset should be considered in x-axis
- **DepTime** column data from the flight delay dataset should be considered in y-axis
- Scatter plot markers should be of red color

In [26]:
```
## Write your code here
fig1=go.Figure()
fig1.add_trace(go.Scatter(x=data['Distance'],y=data['DepTime'],mode='markers',marker=dict(color='blue')))
fig1.update_layout(title="Distance vs Departure Time",xaxis_title='Distance',yaxis_title='DeptTime')
fig1.show()
```



Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

It can be inferred that there are more flights round the clock for shorter distances. However, for longer distance there are limited flights through the day.

2. Line Plot

Let us now use a line plot to extract average monthly arrival delay time and see how it changes over the year.

This plot should contain the following

- Title as **Month vs Average Flight Delay Time**.
- x-axis label should be **Month**
- y-axis label should be **ArrDelay**
- A new dataframe **line_data** should be created which consists of 2 columns average **arrival delay time per month** and **month** from the dataset
- **Month** column data from the line_data dataframe should be considered in x-axis
- **ArrDelay** column data from the ine_data dataframeshould be considered in y-axis

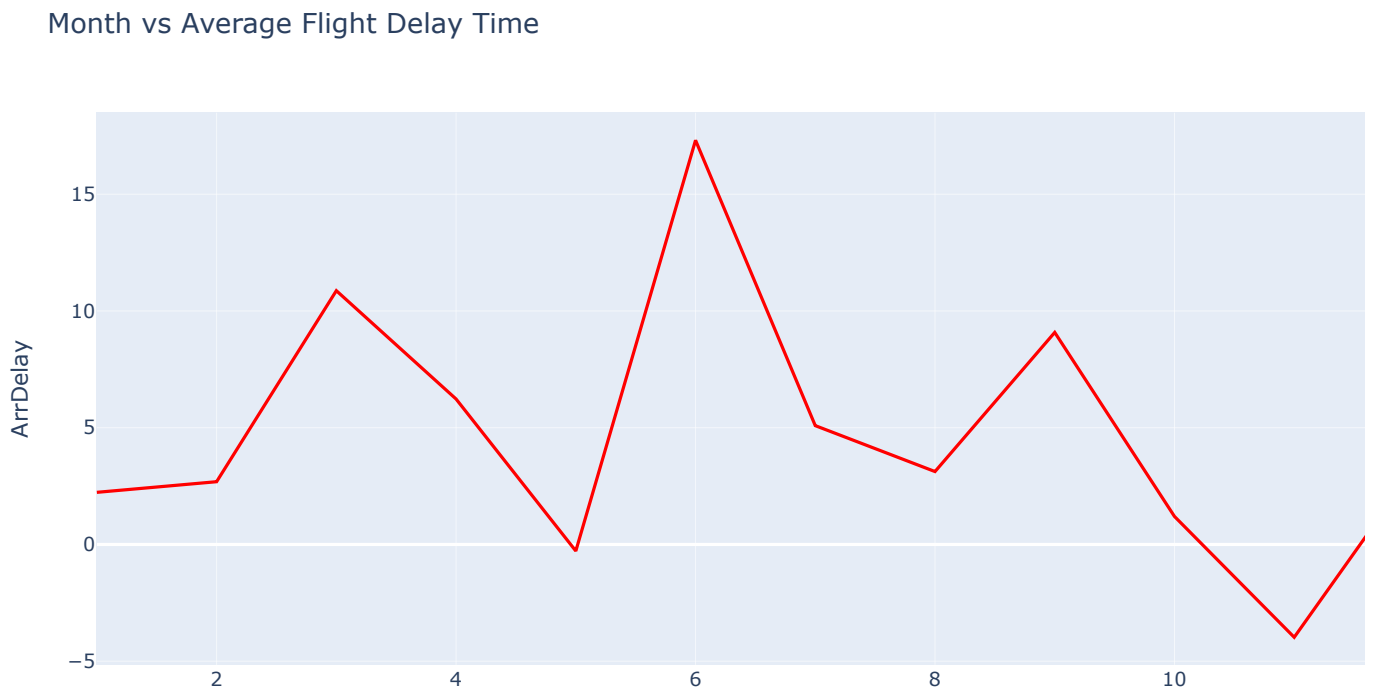
In [27]:
```
# Group the data by Month and compute average over arrival delay time.
line_data = data.groupby('Month')['ArrDelay'].mean().reset_index()
```

In [28]:
```
# Display the data
line_data
```

Out[28]:

	Month	ArrDelay
0	1	2.232558
1	2	2.687500
2	3	10.868421
3	4	6.229167
4	5	-0.279070
5	6	17.310345
6	7	5.088889
7	8	3.121951
8	9	9.081081
9	10	1.200000
10	11	-3.975000
11	12	3.240741

In [29]:
```
## Write your code here
fig2=go.Figure()
fig2.add_trace(go.Scatter(x=line_data['Month'],y=line_data['ArrDelay'],mode='lines',marker=dict(color="red")))
fig2.update_layout(title="Month vs Average Flight Delay Time",xaxis_title='Month',yaxis_title='ArrDelay')
fig2.show()
```



Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

It is found that in the month of June the average monthly delay time is the maximum

plotly.express¶

3. Bar Chart

Let us use a bar chart to extract number of flights from a specific airline that goes to a destination

This plot should contain the following

- Title as **Total number of flights to the destination state split by reporting air**.
- x-axis label should be **DestState**
- y-axis label should be **Flights**
- Create a new dataframe called **bar_data** which contains 2 columns **DestState** and **Flights**.Here **flights** indicate total number of flights in each combination.

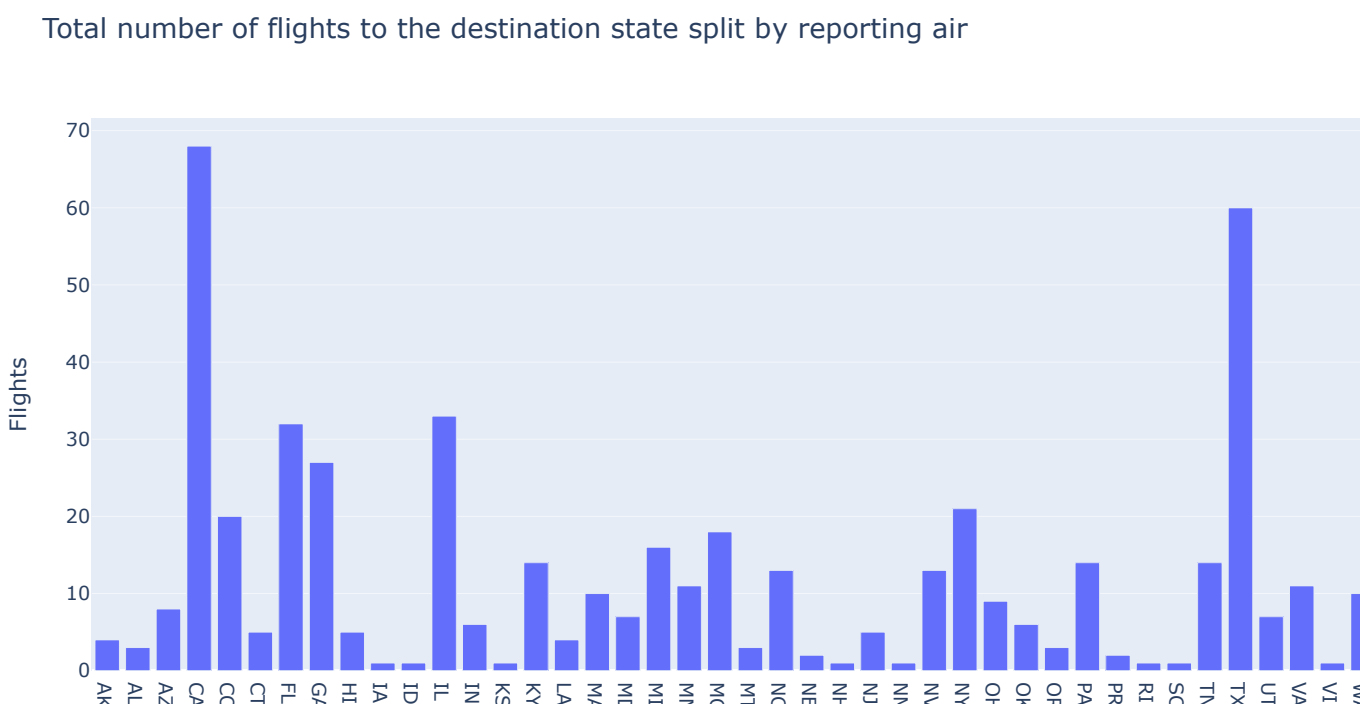
In [30]:
```
# Group the data by destination state and reporting airline. Compute total number of flights in each combination
bar_data = data.groupby('DestState')['Flights'].sum().reset_index()
```

In [31]:
```
# Display the data
bar_data
```

Out[31]:

	DestState	Flights
0	AK	4.0
1	AL	3.0
2	AZ	8.0
3	CA	68.0
4	CO	20.0
5	CT	5.0
6	FL	32.0
7	GA	27.0
8	HI	5.0
9	IA	1.0
10	ID	1.0
11	IL	33.0
12	IN	6.0
13	KS	1.0
14	KY	14.0
15	LA	4.0
16	MA	10.0
17	MD	7.0
18	MI	16.0
19	MN	11.0
20	MO	18.0
21	MT	3.0
22	NC	13.0
23	NE	2.0
24	NH	1.0
25	NJ	5.0
26	NM	1.0
27	NV	13.0
28	NY	21.0
29	OH	9.0
30	OK	6.0
31	OR	3.0
32	PA	14.0
33	PR	2.0
34	RI	1.0
35	SC	1.0
36	TN	14.0
37	TX	60.0
38	UT	7.0
39	VA	11.0
40	VI	1.0
41	WA	10.0
42	WI	8.0

In [32]:
```
## Write your code here
fig3=px.bar(bar_data,x='DestState',y='Flights',title="Total number of flights to the destination state split by re
fig3.show()
```



Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

It is found that maximum flights are to destination state **CA** which is around 68 and there is only 1 flight to destination state **VT**

4. Histogram

Let us represent the distribution of arrival delay using a histogram

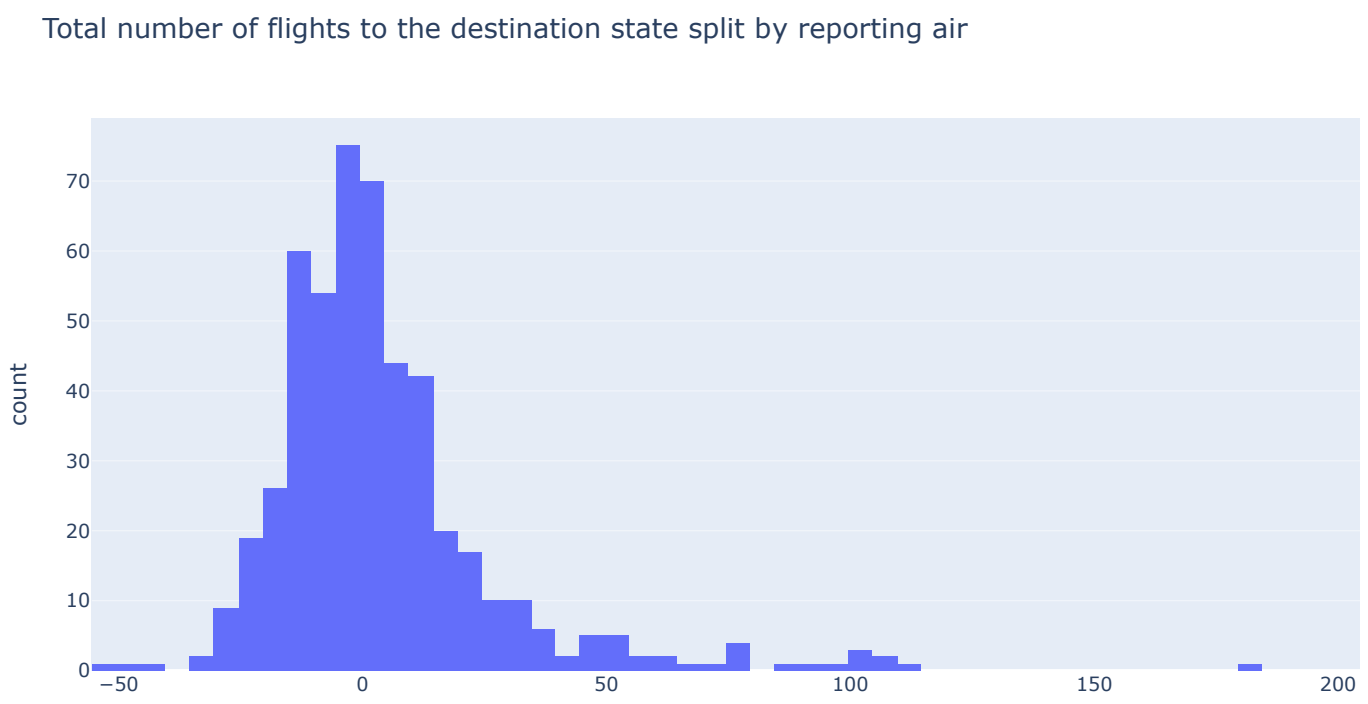
This plot should contain the following

- Title as **Total number of flights to the destination state split by reporting air**.
- x-axis label should be **ArrayDelay**
- y-axis will show the count of arrival delay

In [33]:
```
# Set missing values to 0
data['ArrDelay'] = data['ArrDelay'].fillna(0)
data['ArrDelay'].head(5)
```

Out[33]:
```
5312     32.0
18357    -1.0
6428     -5.0
15414    -2.0
10610   -11.0
Name: ArrDelay, dtype: float64
```

In [34]:
```
## Write your code here
fig4=px.histogram(data,x='ArrDelay',title="Total number of flights to the destination state split by reporting air
fig4.show()
```



Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

It is found that there is only max of 5 flights with an arrival delay of 50-54 minutes and around 17 flights with an arrival delay of 20-25 minutes

5. Bubble Chart

Let use a bubble plot to represent number of flights as per reporting airline

This plot should contain the following

- Title as **Reporting Airline vs Number of Flights**.
- x-axis label should be **Reporting_Airline**
- y-axis label should be **Flights**
- size of the bubble should be **Flights** indicating number of flights
- Name of the hover tooltip to `reporting_airline` using `hover_name` parameter.

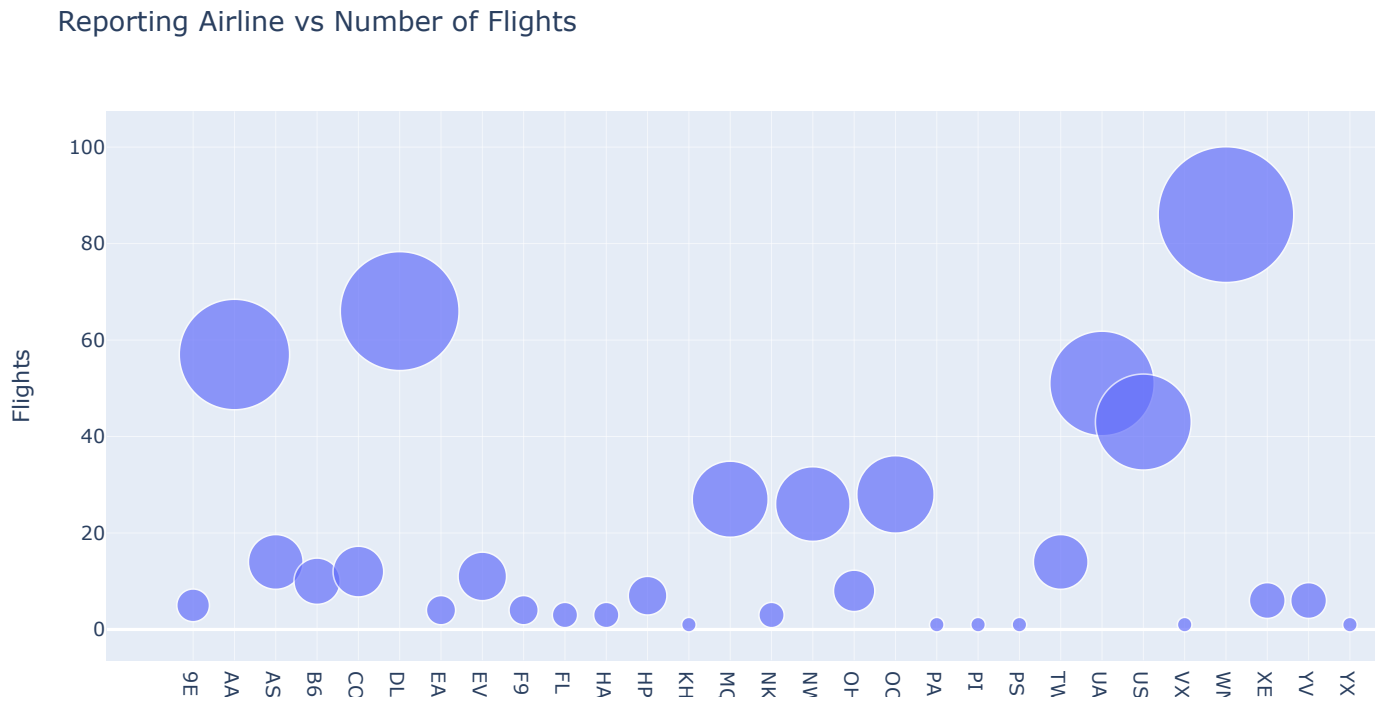
In [35]:
```
# Group the data by reporting airline and get number of flights
bub_data = data.groupby('Reporting_Airline')['Flights'].sum().reset_index()
```

In [36]:
```
bub_data
```

Out[36]:

	Reporting_Airline	Flights
0	9E	5.0
1	AA	57.0
2	AS	14.0
3	B6	10.0
4	CO	12.0
5	DL	66.0
6	EA	4.0
7	EV	11.0
8	F9	4.0
9	FL	3.0
10	HA	3.0
11	HP	7.0
12	KH	1.0
13	MQ	27.0
14	NK	3.0
15	NW	26.0
16	OH	8.0
17	OO	28.0
18	PA (1)	1.0
19	PI	1.0
20	PS	1.0
21	TW	14.0
22	UA	51.0
23	US	43.0
24	VX	1.0
25	WN	86.0
26	XE	6.0
27	YV	6.0
28	YX	1.0

In [37]:
```
## Write your code here
fig5=px.scatter(bub_data,x="Reporting_Airline",y="Flights",size="Flights",hover_name="Reporting_Airline",title="Re
fig5.show()
```



Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

It is found that the reporting airline **WN** has the highest number of flights which is around 86

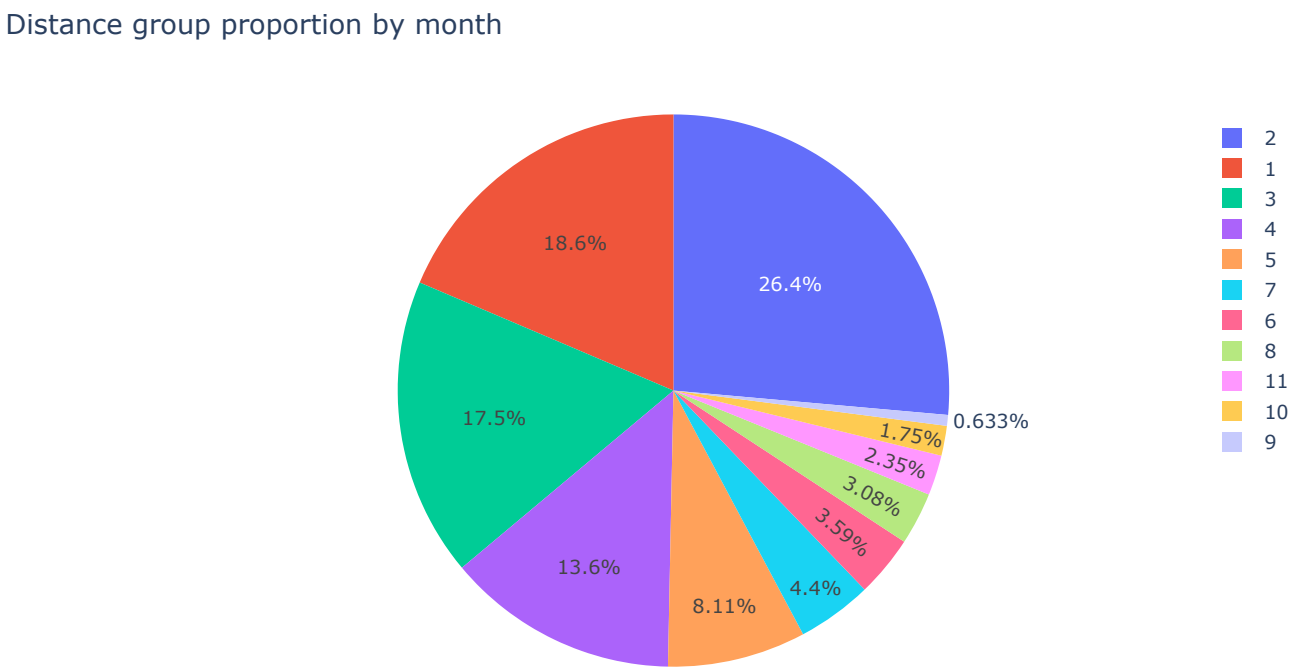
6. Pie Chart

Let us represent the proportion of distance group by month (month indicated by numbers)

This plot should contain the following

- Title as **Distance group proportion by month**.
- values should be **Month**
- names should be **DistanceGroup**

In [38]:
```
## Write your code here
fig6=px.pie(data,values="Month",names="DistanceGroup",title="Distance group proportion by month")
fig6.show()
```



Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

It is found that February month has the highest distance group proportion

7. SunBurst Charts

Let us represent the hierarchical view in othe order of month and destination state holding value of number of flights

This plot should contain the following

- Define hierarchy of sectors from root to leaves in `path` parameter. Here, we go from `Month` to `DestStateName` feature.
- Set sector values in `values` parameter. Here, we can pass in `Flights` feature.
- Show the figure.
- Title as **Flight Distribution Hierarchy**

In [39]:
```
## Write your code here
fig7=px.sunburst(data,path=["Month","DestStateName"],values="Flight",title="Flight Distribution Hierarchy")
fig7.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [39], line 2
      1 ## Write your code here
----> 2 fig7=px.sunburst(data,path=["Month","DestStateName"],values="Flight",title="Flight Distribution Hierarch
y")
      3 fig7.show()

File ~\anaconda3\lib\site-packages\plotly\express\_chart_types.py:1517, in sunburst(data_frame, names, values, pa
rents, path, ids, color, color_continuous_scale, range_color, color_continuous_midpoint, color_discrete_sequence,
color_discrete_map, hover_name, hover_data, custom_data, labels, title, template, width, height, branchvalues, ma
xdepth)
   1515 if path is not None and branchvalues is None:
   1516     branchvalues = "total"
-> 1517 return make_figure(
   1518     args=locals(),
   1519     constructor=go.Sunburst,
   1520     trace_patch=dict(branchvalues=branchvalues, maxdepth=maxdepth),
   1521     layout_patch=layout_patch,
   1522 )

File ~\anaconda3\lib\site-packages\plotly\express\_core.py:1933, in make_figure(args, constructor, trace_patch, l
ayout_patch)
   1930 layout_patch = layout_patch or {}
   1931 apply_default_cascade(args)
-> 1933 args = build_dataframe(args, constructor)
   1934 if constructor in [go.Treemap, go.Sunburst, go.Icicle] and args["path"] is not None:
   1935     args = process_dataframe_hierarchy(args)

File ~\anaconda3\lib\site-packages\plotly\express\_core.py:1405, in build_dataframe(args, constructor)
   1402     args["color"] = None
   1403 # now that things have been prepped, we do the systematic rewriting of `args`
-> 1405 df_output, wide_id_vars = process_args_into_dataframe(
   1406     args, wide_mode, var_name, value_name
   1407 )
   1409 # now that `df_output` exists and `args` contains only references, we complete
   1410 # the special-case and wide-mode handling by further rewriting args and/or mutating
   1411 # df_output
   1413 count_name = _escape_col_name(df_output, "count", [var_name, value_name])

File ~\anaconda3\lib\site-packages\plotly\express\_core.py:1207, in process_args_into_dataframe(args, wide_mode,
var_name, value_name)
   1205         if argument == "index":
   1206             err_msg += "\n To use the index, pass it in directly as `df.index`."
-> 1207         raise ValueError(err_msg)
   1208 elif length and len(df_input[argument]) != length:
   1209     raise ValueError(
   1210         "All arguments should have the same length. "
   1211         "The length of column argument `df[%s]` is %d, whereas the "
   (...)
   1218         )
   1219     )

ValueError: Value of 'values' is not the name of a column in 'data_frame'. Expected one of ['Unnamed: 0', 'Year',
'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'FlightDate', 'Reporting_Airline', 'DOT_ID_Reporting_Airline', 'IA
TA_CODE_Reporting_Airline', 'Tail_Number', 'Flight_Number_Reporting_Airline', 'OriginAirportID', 'OriginAirportSe
qID', 'OriginCityMarketID', 'Origin', 'OriginCityName', 'OriginState', 'OriginStateFips', 'OriginStateName', 'Ori
ginWac', 'DestAirportID', 'DestAirportSeqID', 'DestCityMarketID', 'Dest', 'DestCityName', 'DestState', 'DestState
Fips', 'DestStateName', 'DestWac', 'CRSDepTime', 'DepTime', 'DepDelay', 'DepDelayMinutes', 'DepDel15', 'Departure
DelayGroups', 'DepTimeBlk', 'TaxiOut', 'WheelsOff', 'WheelsOn', 'TaxiIn', 'CRSArrTime', 'ArrTime', 'ArrDelay', 'A
rrDelayMinutes', 'ArrDel15', 'ArrivalDelayGroups', 'ArrTimeBlk', 'Cancelled', 'CancellationCode', 'Diverted', 'CR
SElapsedTime', 'ActualElapsedTime', 'AirTime', 'Flights', 'Distance', 'DistanceGroup', 'CarrierDelay', 'WeatherDe
lay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay', 'FirstDepTime', 'TotalAddGTime', 'LongestAddGTime', 'DivA
irportLandings', 'DivReachedDest', 'DivActualElapsedTime', 'DivArrDelay', 'DivDistance', 'Div1Airport', 'Div1Airp
ortID', 'Div1AirportSeqID', 'Div1WheelsOn', 'Div1TotalGTime', 'Div1LongestGTime', 'Div1WheelsOff', 'Div1TailNum',
'Div2Airport', 'Div2AirportID', 'Div2AirportSeqID', 'Div2WheelsOn', 'Div2TotalGTime', 'Div2LongestGTime', 'Div2Wh
eelsOff', 'Div2TailNum', 'Div3Airport', 'Div3AirportID', 'Div3AirportSeqID', 'Div3WheelsOn', 'Div3TotalGTime', 'D
iv3LongestGTime', 'Div3WheelsOff', 'Div3TailNum', 'Div4Airport', 'Div4AirportID', 'Div4AirportSeqID', 'Div4Wheels
On', 'Div4TotalGTime', 'Div4LongestGTime', 'Div4WheelsOff', 'Div4TailNum', 'Div5Airport', 'Div5AirportID', 'Div5A
irportSeqID', 'Div5WheelsOn', 'Div5TotalGTime', 'Div5LongestGTime', 'Div5WheelsOff', 'Div5TailNum'] but received:
Flight
```

Double-click **here** for hint.

Double-click **here** for the solution.

Inferences

Here the **Month** numbers present in the innermost concentric circle is the root and for each month we will check the **number of flights** for the different **destination states** under it.

Summary

Congratulations for completing your lab.

In this lab, you have learnt how to use `plotly.graph_objects` and `plotly.express` for creating plots and charts.

Author(s)

[Saishruthi Swaminathan (https://www.linkedin.com/in/saishruthi-swaminathan/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork970-2022-01-01)

Lakshmi Holla

Other Contributor(s)

Lavanya T S

Changelog

Date	Version	Changed by	Change Description
12-18-2020	1.0	Nayef	Added dataset link and upload to Git
07-02-2023	1.1	Lakshmi Holla	Updated lab

In [ ]: