# University of Salford, MSc Data Science

**Module:** Big Data Tools and Techniques

**Date:** Trimester 2, 2024-2025

**Session:** Workshop Week 10

**Topic:** Using python libraries for data visualisation in Databricks

**Tools**: Databricks Community Edition

**Instructors:** Dr Kaveh Kiani, Dr Taha Mansouri, and Nathan Topping.

## Objectives:

After completing this workshop, you will be able to:

- Use matplotlib library in Databricks to visualise your big data
- Use seaborn library in Databricks to visualise your big data

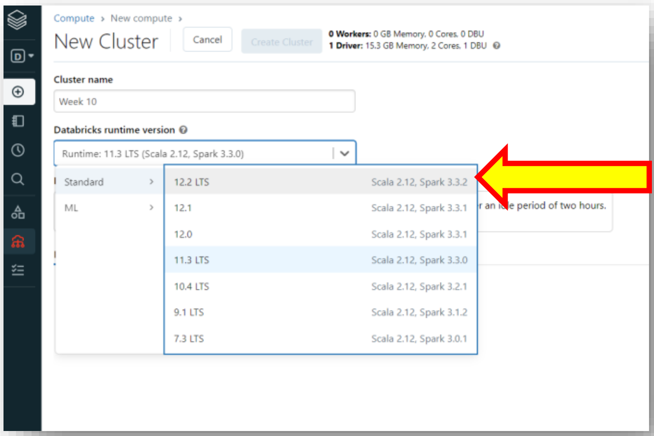
# Table of Contents

## Part 1: Fire up the Databricks workspace

1. Log in to your Databricks Community Edition account. Here is the link:

   [https://community.cloud.databricks.com/login.html?nextUrl=%2F](https://community.cloud.databricks.com/login.html?nextUrl=%2F)

2. Click on "Create Compute" and type in a new name for the cluster, any name that you like.
3. From the dropdown, select latest **standard** runtime. Note that it will take a few minutes to create the cluster. After some time, the green circle next to the cluster name will gain a green tick meaning the cluster has successfully started up.



New Cluster

**0 Workers:** 0 GB Memory, 0 Cores, 0 DBU
**1 Driver:** 15.3 GB Memory, 2 Cores, 1 DBU

Cluster name: Week 10

Databricks runtime version: Runtime: 11.3 LTS (Scala 2.12, Spark 3.3.0)

| | |
|---|---|
| 12.2 LTS | Scala 2.12, Spark 3.3.2 |
| 12.1 | Scala 2.12, Spark 3.3.1 |
| 12.0 | Scala 2.12, Spark 3.3.1 |
| 11.3 LTS | Scala 2.12, Spark 3.3.0 |
| 10.4 LTS | Scala 2.12, Spark 3.2.1 |
| 9.1 LTS | Scala 2.12, Spark 3.1.2 |
| 7.3 LTS | Scala 2.12, Spark 3.0.1 |

r an idle period of two hours.

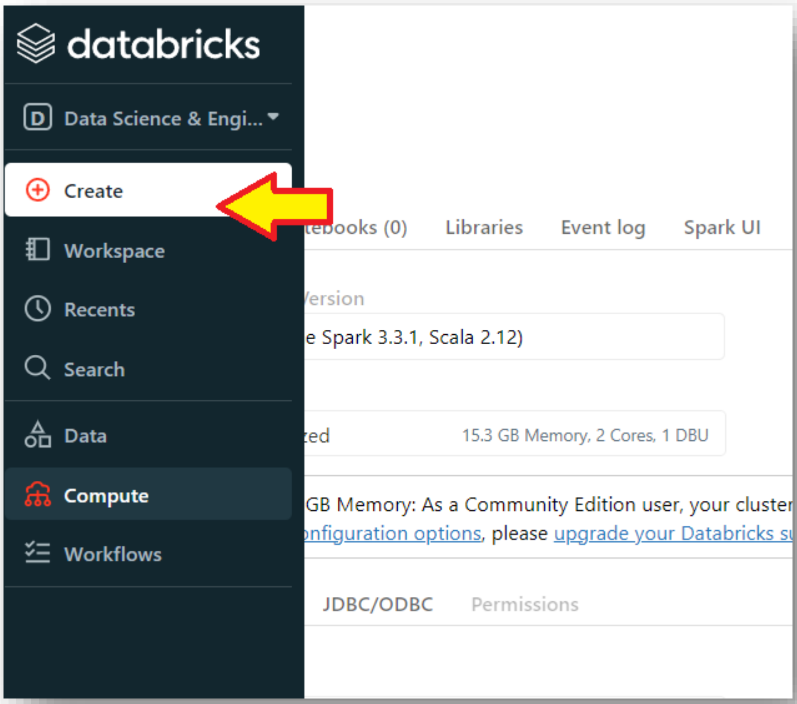
## Part 2: Creating a new notebook

Databricks programs are written in Notebooks. A notebook is a collection of runnable cells which contain your commands. Look at:

[https://docs.databricks.com/notebooks/notebooks-use.html](https://docs.databricks.com/notebooks/notebooks-use.html)

for an overview of how to use a Databricks notebook.

1. Click on the button "Create" and select "Notebook".



/ersion

e Spark 3.3.1, Scala 2.12)

ted 15.3 GB Memory, 2 Cores, 1 DBU

GB Memory: As a Community Edition user, your cluster
onfiguration options, please upgrade your Databricks su

2. Name your notebook, e.g. “Week 10 Data visualisation”, leave the language as the default Python and the cluster should be the cluster you have just created.

## Part 3: Matplotlib Library

Matplotlib is a comprehensive library for creating static, animated, and interactive visualisations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in Jupiterian.
- Use a rich array of third-party packages built on Matplotlib.

Matplotlib is one of the best python data visualisation libraries for generating powerful yet simple visualisation. It is a 2-D plotting library that can be used in various ways, including Spark, Python, iPython sheets, and Jupyter notebooks.

Key Features

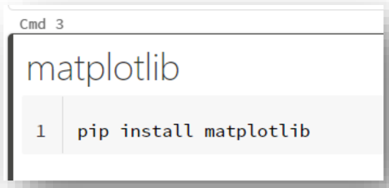
It supports various types of graphical representation, including line graphs, bar graphs, and histograms.

- It can work with the NumPy arrays and border SciPy stack.
- It has a huge number of plots for understanding trends and making correlations.

Pros and Cons

- Interactive platform
- Versatile library
- Not ideal for time series data

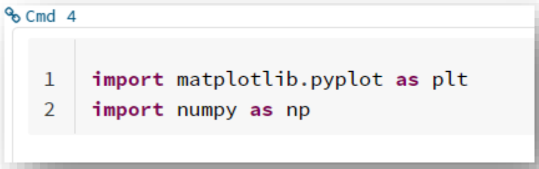
1. Install ***mtplotlib*** library.



matplotlib

```
pip install matplotlib
```

2. Import required libraries.

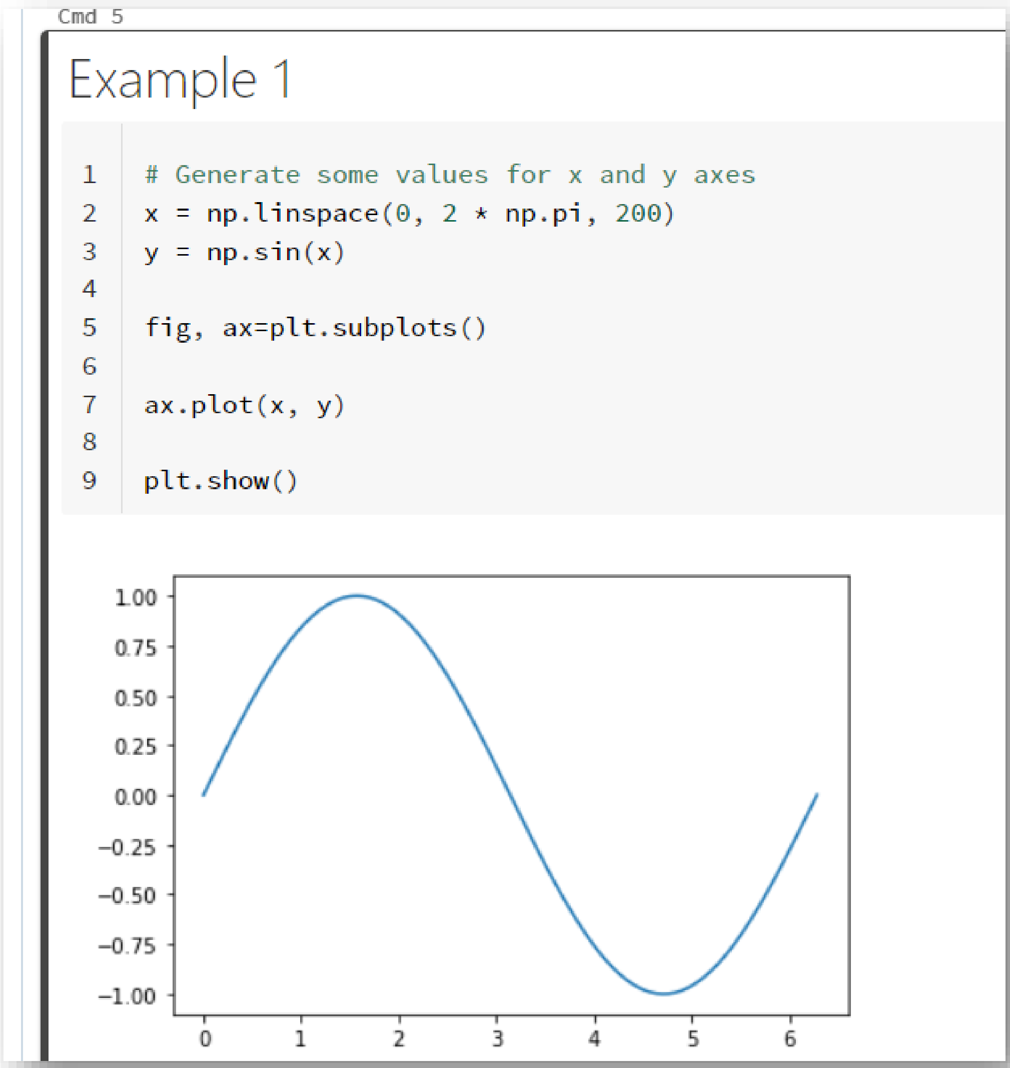


Cmd 4

```
import matplotlib.pyplot as plt
import numpy as np
```

Try to run the four examples and then change their parameters to understand the usage of each of them.

**Example 1:** Drawing a simple sin(x) plot



Cmd 5

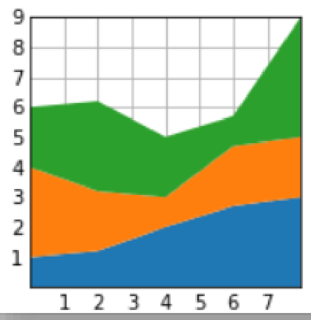
Example 1

```
# Generate some values for x and y axes
x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax=plt.subplots()

ax.plot(x, y)

plt.show()
```

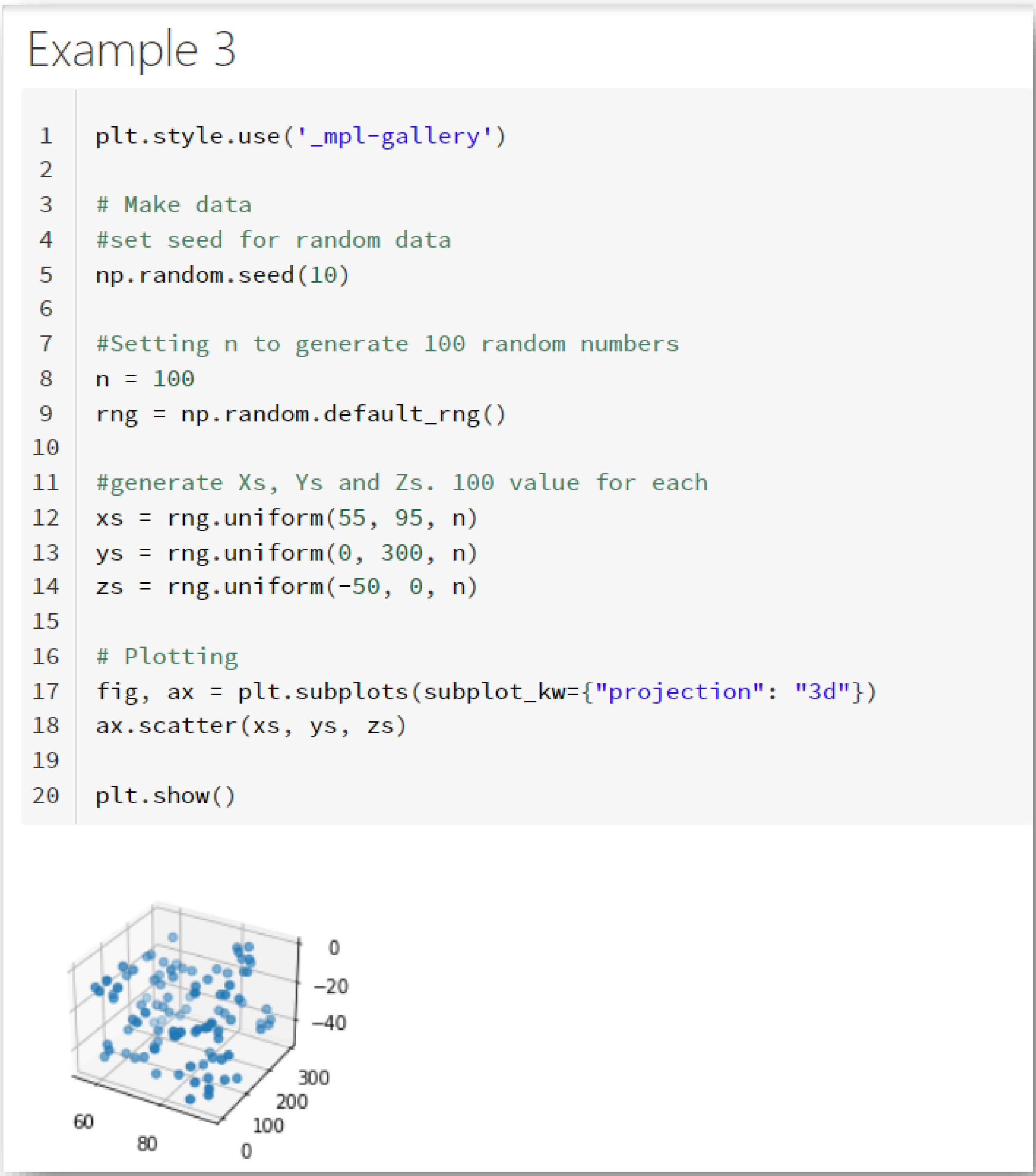
**Example 2:** Generating a stack bar chart.

## Example 2

```
plt.style.use('_mpl-gallery')
# make data
# X from 0 to 10 with step 2 (5 numbers or 0,2,4,6,8)
x = np.arange(0, 10, 2)

# 3 sets of Ys, following lists
ay = [1, 1.2, 2, 2.7, 3]
by = [3, 2, 1, 2, 2]
cy = [2, 3, 2, 1, 4]

# stacking Ys, first stack is (1,3,2). look at the graph at x=0
y = np.vstack([ay, by, cy])

# plotting
fig, ax = plt.subplots()

ax.stackplot(x, y)

# plot settings
ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 10))
plt.show()
```

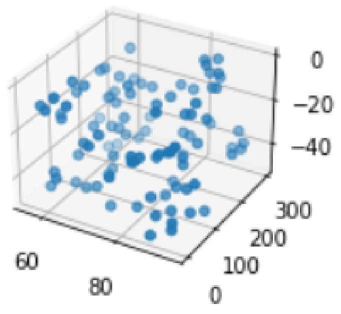


**Example 3:** Generating a 3D scatter plot.

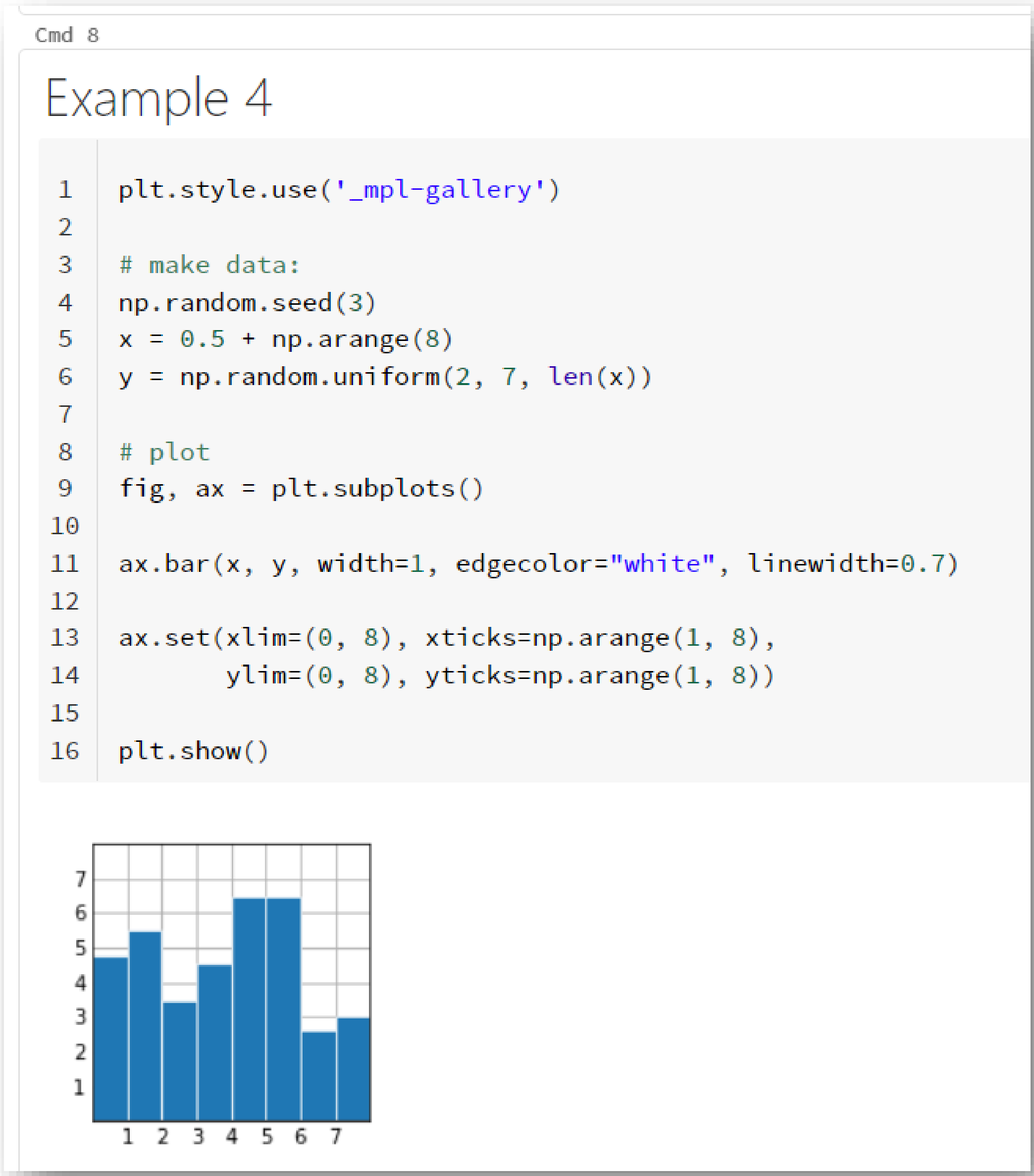


Example 3

```
plt.style.use('_mpl-gallery')

# Make data
#set seed for random data
np.random.seed(10)

#Setting n to generate 100 random numbers
n = 100
rng = np.random.default_rng()

#generate Xs, Ys and Zs. 100 value for each
xs = rng.uniform(55, 95, n)
ys = rng.uniform(0, 300, n)
zs = rng.uniform(-50, 0, n)

# Plotting
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.scatter(xs, ys, zs)

plt.show()
```



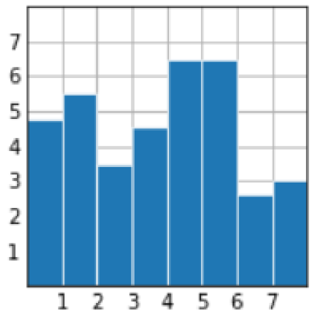
**Example 4:** Generating a bar chart.



Cmd 8

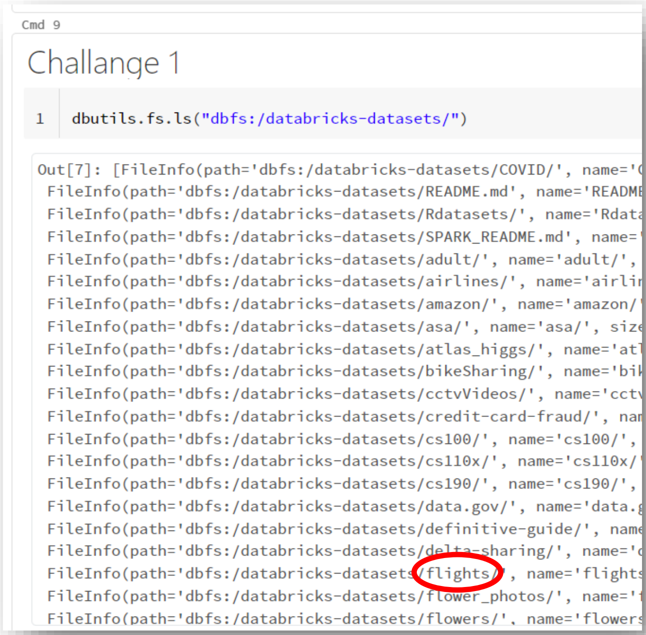
Example 4

```
plt.style.use('_mpl-gallery')

# make data:
np.random.seed(3)
x = 0.5 + np.arange(8)
y = np.random.uniform(2, 7, len(x))

# plot
fig, ax = plt.subplots()

ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



**Challenge 1**

Use one of the Databricks sample dataset, **flights,** to create a bar chart that shows first five most destination of the airline.

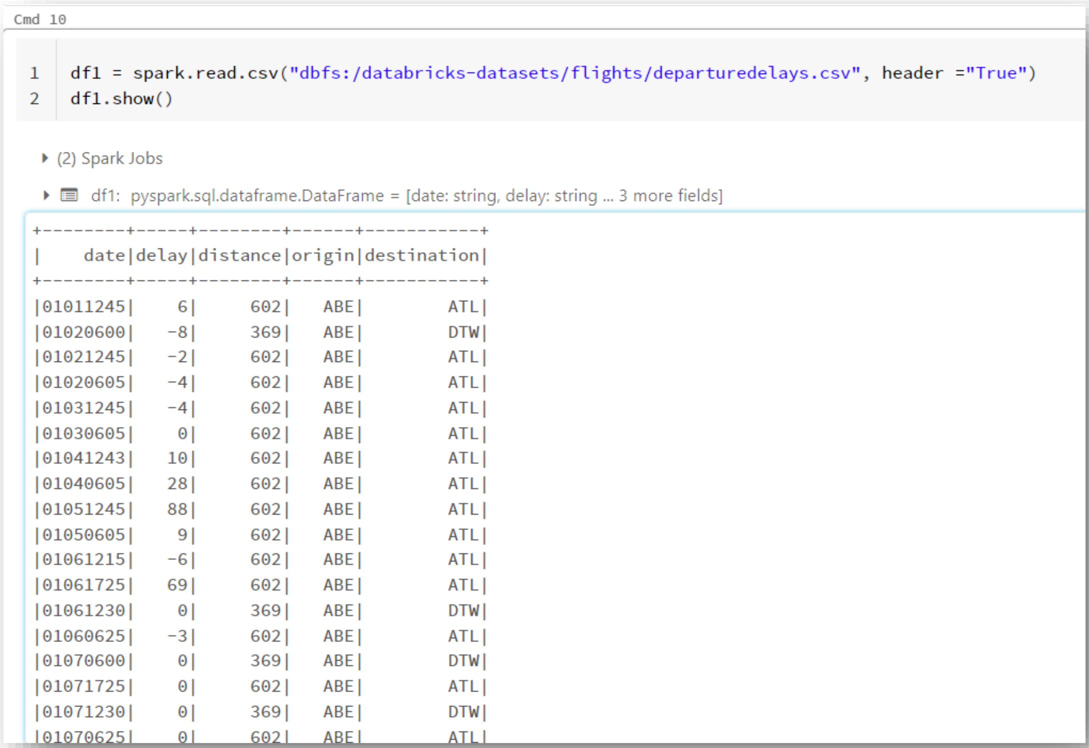


Cmd 9

Challange 1

```
dbutils.fs.ls("dbfs:/databricks-datasets/")
```

```
Out[7]: [FileInfo(path='dbfs:/databricks-datasets/COVID/', name='C
 FileInfo(path='dbfs:/databricks-datasets/README.md', name='README
 FileInfo(path='dbfs:/databricks-datasets/Rdatasets/', name='Rdata
 FileInfo(path='dbfs:/databricks-datasets/SPARK_README.md', name='
 FileInfo(path='dbfs:/databricks-datasets/adult/', name='adult/',
 FileInfo(path='dbfs:/databricks-datasets/airlines/', name='airlin
 FileInfo(path='dbfs:/databricks-datasets/amazon/', name='amazon/'
 FileInfo(path='dbfs:/databricks-datasets/asa/', name='asa/', size
 FileInfo(path='dbfs:/databricks-datasets/atlas_higgs/', name='atl
 FileInfo(path='dbfs:/databricks-datasets/bikeSharing/', name='bik
 FileInfo(path='dbfs:/databricks-datasets/cctvVideos/', name='cctv
 FileInfo(path='dbfs:/databricks-datasets/credit-card-fraud/', nam
 FileInfo(path='dbfs:/databricks-datasets/cs100/', name='cs100/',
 FileInfo(path='dbfs:/databricks-datasets/cs110x/', name='cs110x/'
 FileInfo(path='dbfs:/databricks-datasets/cs190/', name='cs190/',
 FileInfo(path='dbfs:/databricks-datasets/data.gov/', name='data.g
 FileInfo(path='dbfs:/databricks-datasets/definitive-guide/', name
 FileInfo(path='dbfs:/databricks-datasets/delta-sharing/', name='d
 FileInfo(path='dbfs:/databricks-datasets/flights/', name='flights
 FileInfo(path='dbfs:/databricks-datasets/flower_photos/', name='f
 FileInfo(path='dbfs:/databricks-datasets/flowers/', name='flowers
```



Cmd 10

```
df1 = spark.read.csv("dbfs:/databricks-datasets/flights/departuredelays.csv", header ="True")
df1.show()
```

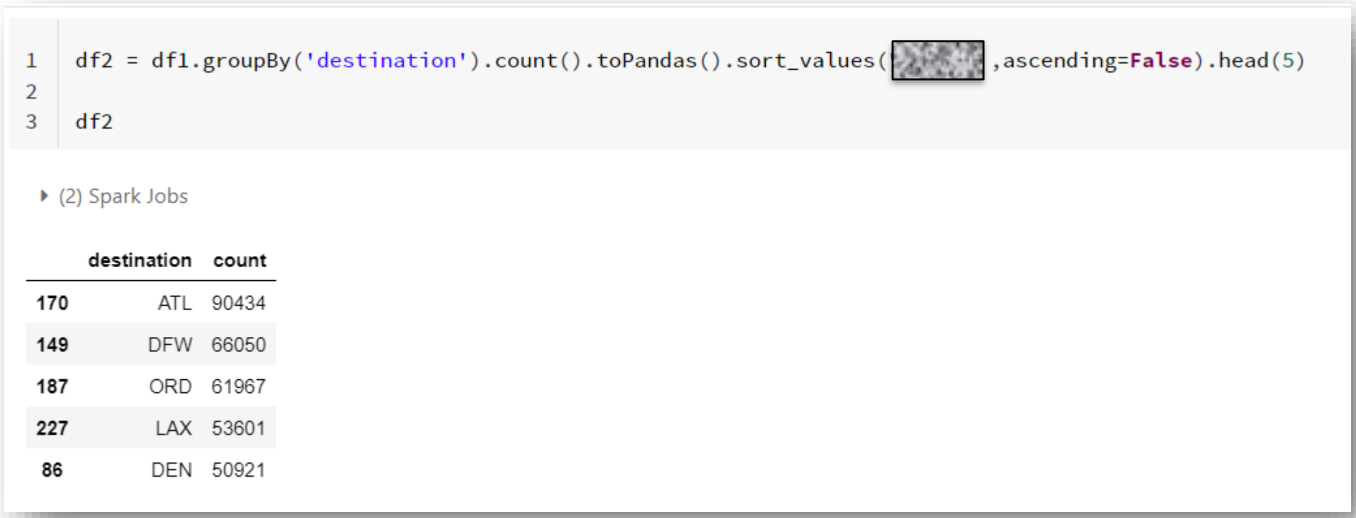
▸ (2) Spark Jobs

▸ df1: pyspark.sql.dataframe.DataFrame = [date: string, delay: string ... 3 more fields]

```
+--------+-----+--------+------+-----------+
|    date|delay|distance|origin|destination|
+--------+-----+--------+------+-----------+
|01011245|    6|     602|   ABE|        ATL|
|01020600|   -8|     369|   ABE|        DTW|
|01021245|   -2|     602|   ABE|        ATL|
|01020605|   -4|     602|   ABE|        ATL|
|01031245|   -4|     602|   ABE|        ATL|
|01030605|    0|     602|   ABE|        ATL|
|01041243|   10|     602|   ABE|        ATL|
|01040605|   28|     602|   ABE|        ATL|
|01051245|   88|     602|   ABE|        ATL|
|01050605|    9|     602|   ABE|        ATL|
|01061215|   -6|     602|   ABE|        ATL|
|01061725|   69|     602|   ABE|        ATL|
|01061230|    0|     369|   ABE|        DTW|
|01060625|   -3|     602|   ABE|        ATL|
|01070600|    0|     369|   ABE|        DTW|
|01071725|    0|     602|   ABE|        ATL|
|01071230|    0|     369|   ABE|        DTW|
|01070625|    0|     602|   ABE|        ATL|
```

Create a new DataFrame with only 2 columns, destination, and count. Convert Pyspark DataFrame to pandas DataFrame. Sort destination column based on the "**count**" column and select five top values then draw the bar chart.

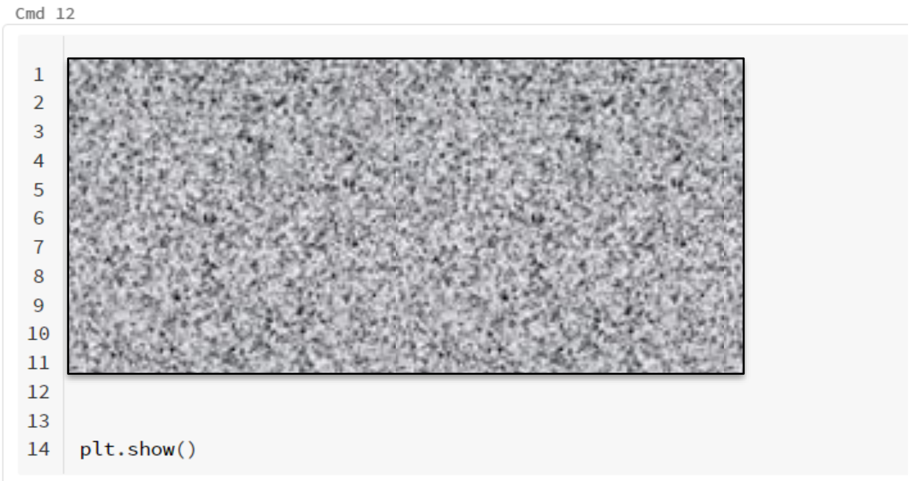
**Hint:** use example 4 and replace x and Y with proper values.



```
df2 = df1.groupBy('destination').count().toPandas().sort_values([redacted],ascending=False).head(5)

df2
```

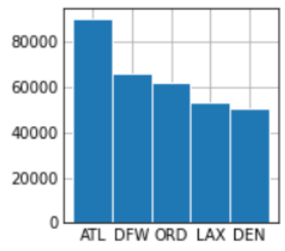
▸ (2) Spark Jobs

|     | destination | count |
|-----|-------------|-------|
| 170 | ATL | 90434 |
| 149 | DFW | 66050 |
| 187 | ORD | 61967 |
| 227 | LAX | 53601 |
| 86 | DEN | 50921 |



Cmd 12

```
[redacted]



plt.show()
```



## Part 4: Seaborn Library

Seaborn is a powerful Python library for creating beautiful and informative data visualisations. It was created by Michael Waskom in 2012 and has since become a popular choice for data visualisation in the Python community. Seaborn is built on top of the matplotlib library, which provides a low-level interface for creating plots. Seaborn provides a high-level interface that simplifies the process of creating complex visualisations and provides a wide range of customization options.

One of the most powerful features of Seaborn is its ability to create visualisations that reveal patterns in data. Seaborn provides a variety of statistical plots that allow you to explore relationships between variables in your data. For example, the scatter plot function in Seaborn allows you to visualise the relationship between two variables, with the option to add a linear regression line and a confidence interval. This can be useful for exploring correlations between variables and identifying outliers or other anomalies in the data.

Seaborn also provides functions for creating distributions, such as histograms and kernel density plots, which can be used to explore the distribution of a single variable. These plots can be customized with a variety of options, such as changing the bin size or bandwidth, to reveal detail in the distribution.

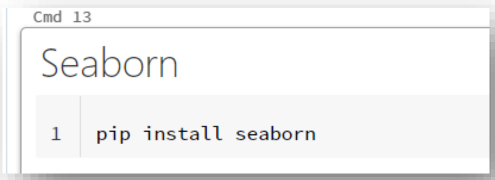
Another useful feature of Seaborn is its ability to create multi-panel plots, such as grids of scatter plots or histograms. These plots can be used to explore relationships between multiple variables in your data. Seaborn also provides functions for creating heatmaps and cluster maps, which can be used to visualise patterns in large datasets.

Seaborn provides a wide range of customization options for its plots. For example, you can change the colour palette, add annotations, or adjust the size and aspect ratio of the plot. Seaborn also provides functions for styling the plot axes, including changing the tick labels and adding grid lines.

One of the advantages of Seaborn over other data visualisation libraries is its integration with pandas, which is a popular library for data manipulation and analysis in Python. Seaborn can directly plot data from a pandas DataFrame, and provides functions for grouping and aggregating data before plotting.

In addition to its powerful visualisation capabilities, Seaborn is also known for its beautiful default styles. The default colour palettes and plot styles are designed to be aesthetically pleasing and easy to read, which can be especially important for creating visualisations that will be used in presentations or reports. Whether you are a beginner or an experienced data scientist, Seaborn is an essential tool in your data visualisation toolkit.

1. Instal seaborn library

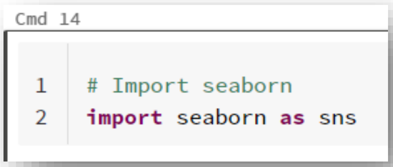


Cmd 13

Seaborn

```
pip install seaborn
```

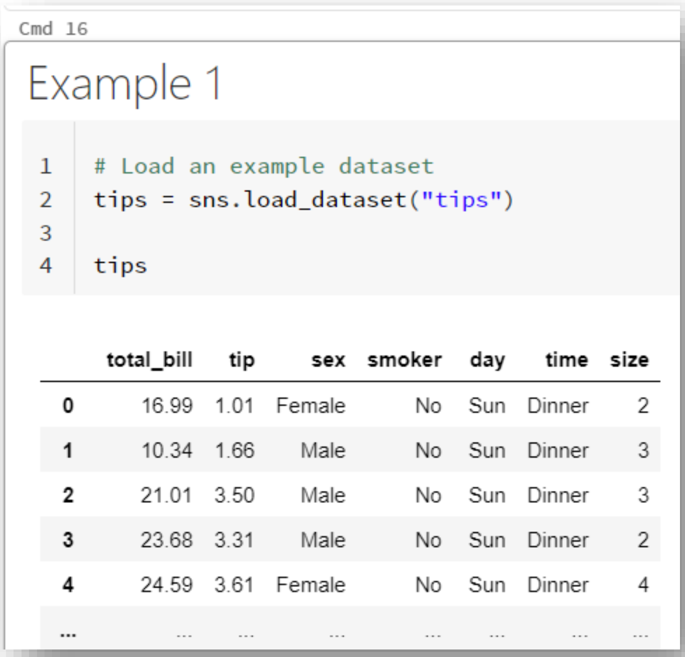
2. Import the library



Cmd 14

```
# Import seaborn
import seaborn as sns
```

**Example 5:** Load and review one of the seaborn's data sets, tips, for the first visualisation.



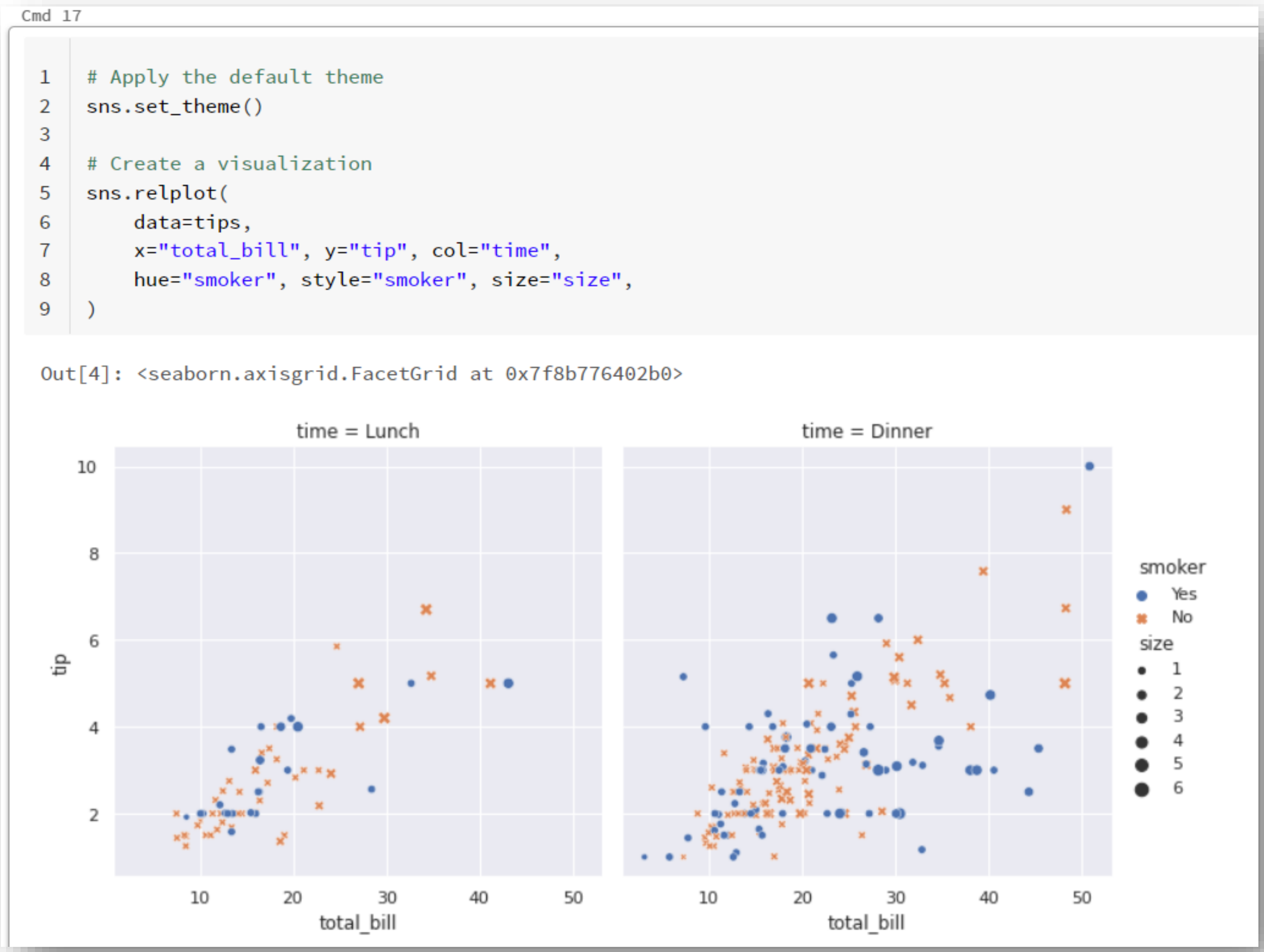
Cmd 16

Example 1

```
# Load an example dataset
tips = sns.load_dataset("tips")

tips
```

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |
| **...** | ... | ... | ... | ... | ... | ... | ... |

**Tips** dataset has 7 columns. We want to compare amount of **total_bill** and **tip** on lunch and dinner (**time** column) also try to highlight data points by **size** and **smoker** columns. So, we should use 5 columns in our visualisation.



Cmd 17

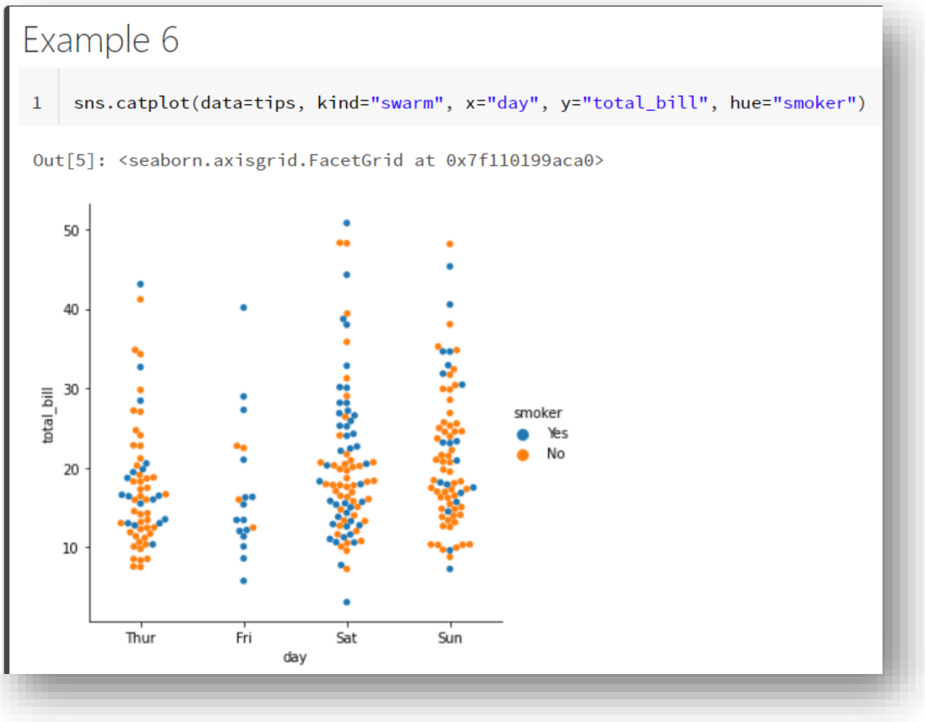
```
# Apply the default theme
sns.set_theme()

# Create a visualization
sns.relplot(
    data=tips,
    x="total_bill", y="tip", col="time",
    hue="smoker", style="smoker", size="size",
)
```

```
Out[4]: <seaborn.axisgrid.FacetGrid at 0x7f8b776402b0>
```

**col = “time”**: shows two graphs side by side in two columns (time = Lunch / time = Dinner)
**hue = “smoker”**: smoker(yes/no) has been separated by colour
**style = “smoker”**: other than smoker colour its shape also is different for yes/no
**size = “size”**: size variable has been separated by size of the dots (1 to 6)

You try to visualise amount of **total_bill** and **tip** on different **days** and highlight data points by **time** and **sex** columns. How many scatter plots do you have? 1, 2, 3, or 4?

**Example 6:** Let’s use ‘tips’ data again. Several specialized plot types in seaborn are oriented towards visualising categorical data. They can be accessed through catplot(). These plots offer different levels of granularity. At the finest level, you may wish to see every observation by drawing a “swarm” plot: a scatter plot that adjusts the positions of the points along the categorical axis so that they don’t overlap:

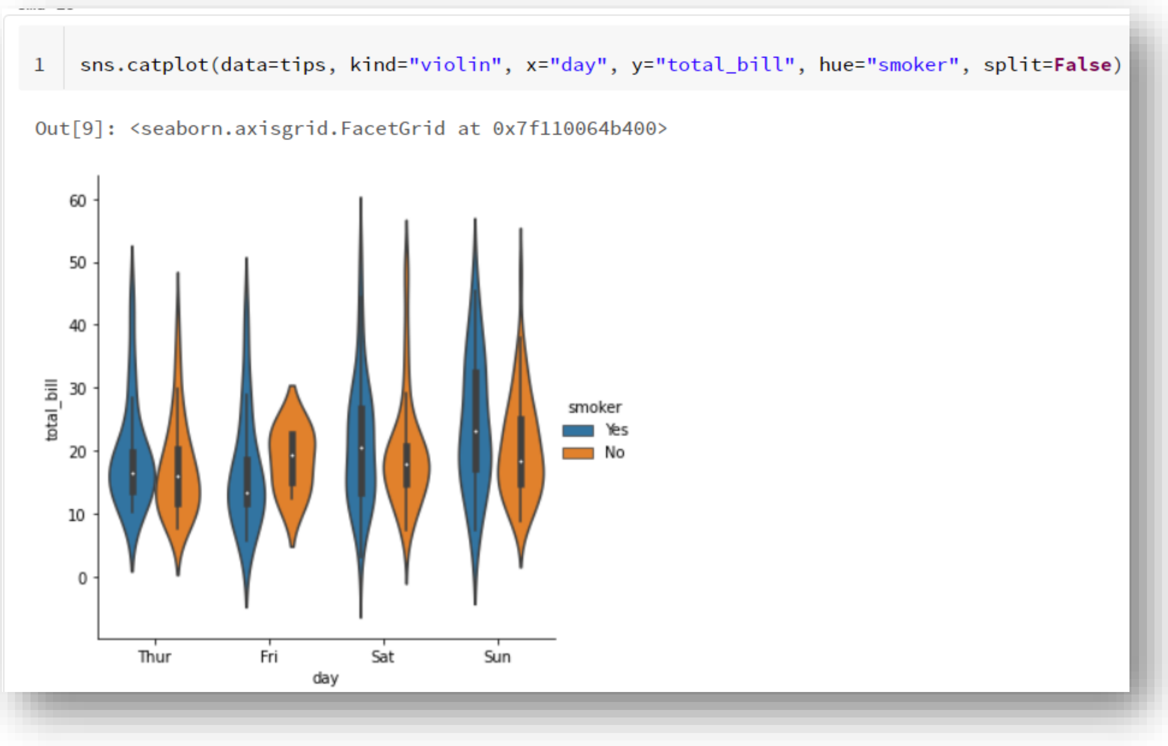


Example 6

```
sns.catplot(data=tips, kind="swarm", x="day", y="total_bill", hue="smoker")
```

```
Out[5]: <seaborn.axisgrid.FacetGrid at 0x7f110199aca0>
```

It seems smokers have bigger total_bill on average but possibly other graph types can present more clear view. Alternately, you could use kernel density estimation to represent the underlying distribution that the points are sampled from:



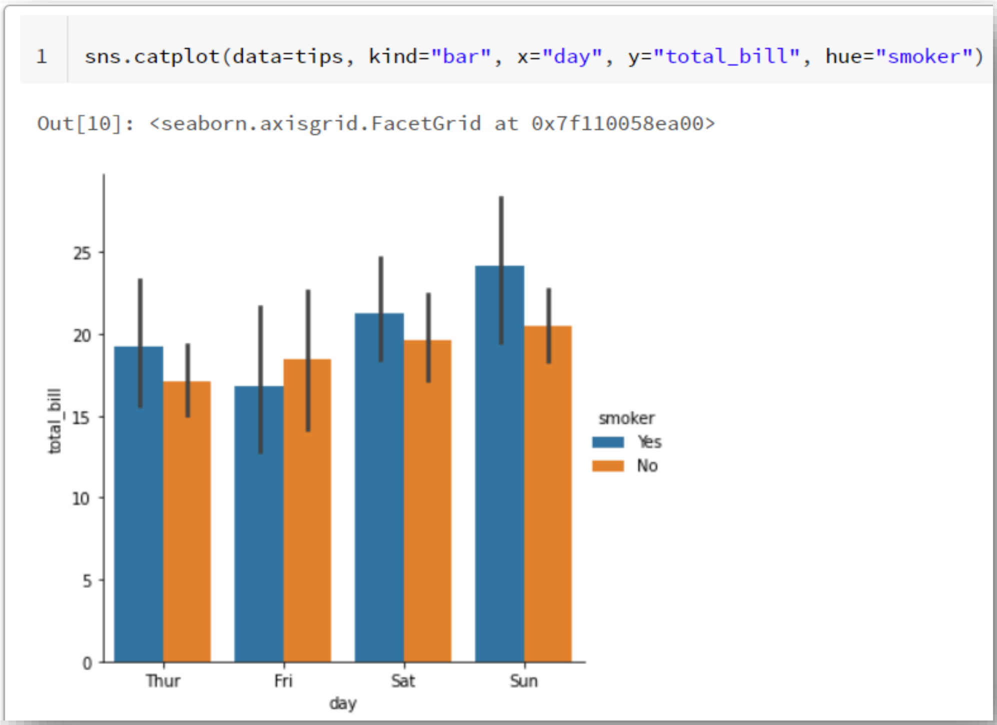
```
sns.catplot(data=tips, kind="violin", x="day", y="total_bill", hue="smoker", split=False)
```

```
Out[9]: <seaborn.axisgrid.FacetGrid at 0x7f110064b400>
```

What pattern you can see in this violin plot?

Let's have a final look and compare categories in a bar chart.

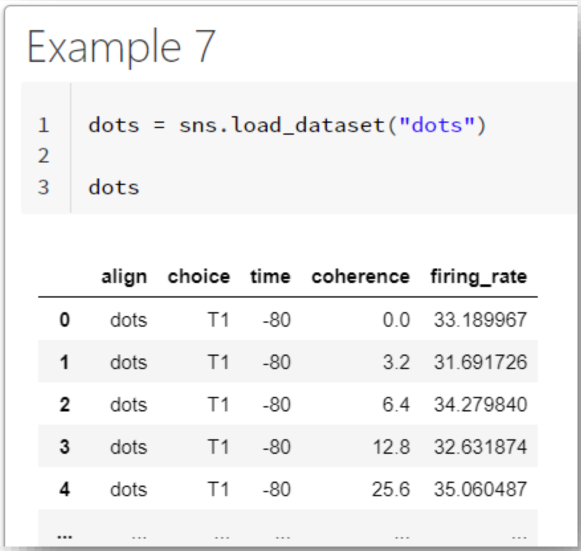
On average smokers have a higher total_bill except on Fridays. Any other conclusion?



```
sns.catplot(data=tips, kind="bar", x="day", y="total_bill", hue="smoker")
```

```
Out[10]: <seaborn.axisgrid.FacetGrid at 0x7f110058ea00>
```

**Example 7:** Load and review another seaborn's data sets, **dots**, for the second visualisation. The "dots" dataset comes from a trial study for a study about the response of monkeys' neurons to some visual discrimination reaction time tasks. The dataset has 5 columns about different variables and observations for each monkey's reaction and performance during the trial.



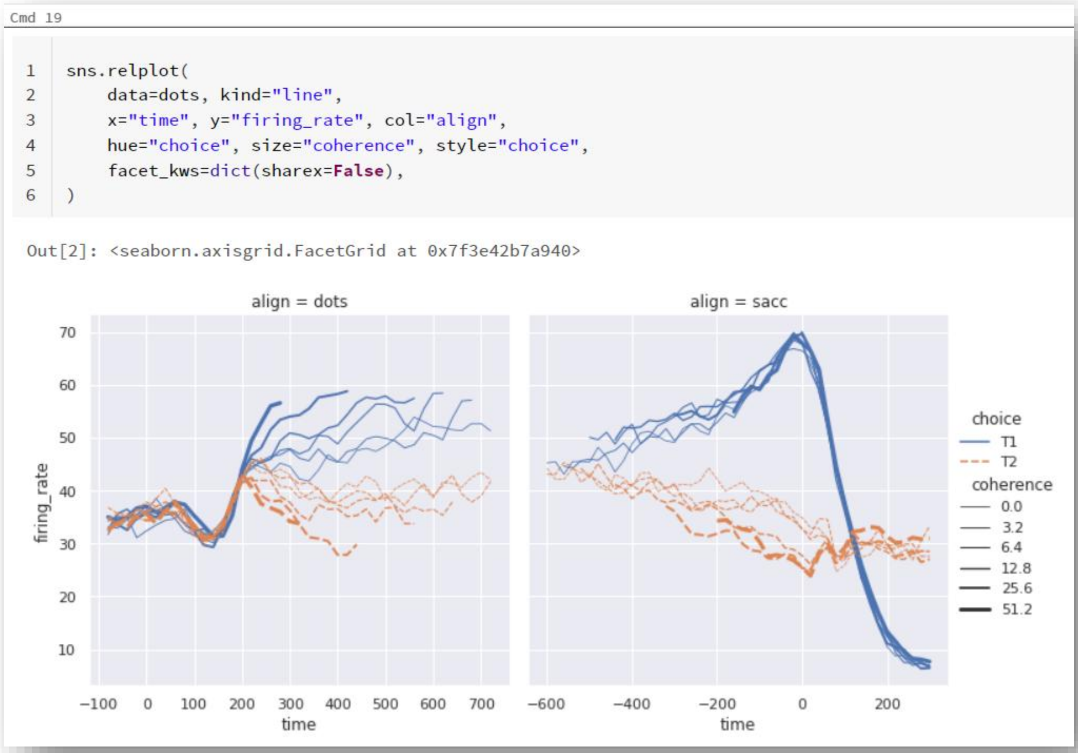
## Example 7

```
dots = sns.load_dataset("dots")

dots
```

| | align | choice | time | coherence | firing_rate |
|---|---|---|---|---|---|
| **0** | dots | T1 | -80 | 0.0 | 33.189967 |
| **1** | dots | T1 | -80 | 3.2 | 31.691726 |
| **2** | dots | T1 | -80 | 6.4 | 34.279840 |
| **3** | dots | T1 | -80 | 12.8 | 32.631874 |
| **4** | dots | T1 | -80 | 25.6 | 35.060487 |
| **...** | ... | ... | ... | ... | ... |

We want to show the relationship between reaction **time** and **firaring_rate** to different **alignments** and try to separate the visualised objects by **choice** and **coherence**. So, we are going to use all 5 columns. Let's choose **relplot** function that shows relations between variables in a line format and possibly the final plot will look like a bunch of neurons.

The function relplot() is named that way because it is designed to visualise many different statistical relationships. While scatter plots are often effective, relationships where one variable represents a measure of time are better represented by a line. The relplot() function has a convenient kind parameter that lets you easily switch to this alternate representation:

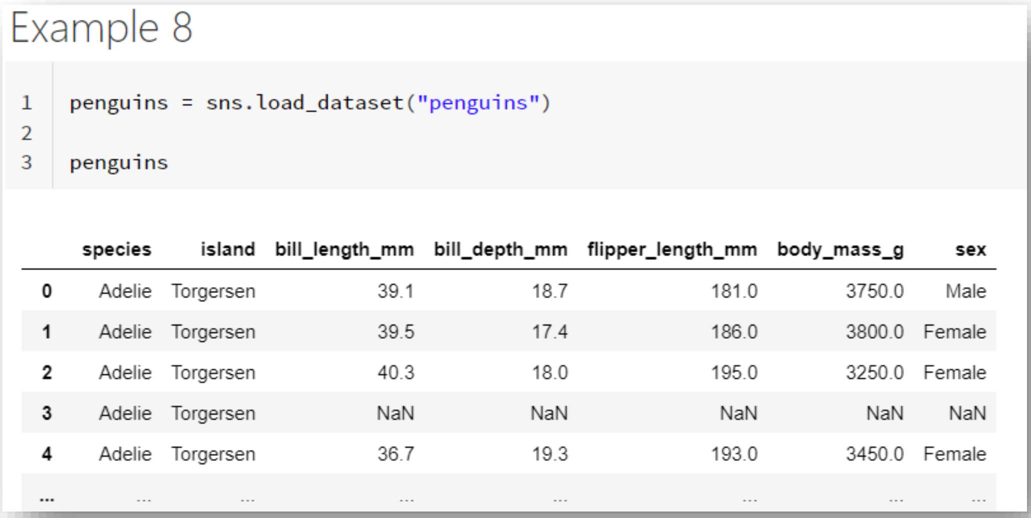

Cmd 19

```
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate", col="align",
    hue="choice", size="coherence", style="choice",
    facet_kws=dict(sharex=False),
)
```

```
Out[2]: <seaborn.axisgrid.FacetGrid at 0x7f3e42b7a940>
```

For both alignments seems monkeys have a higher firaing_rate for the T1 (blue line).

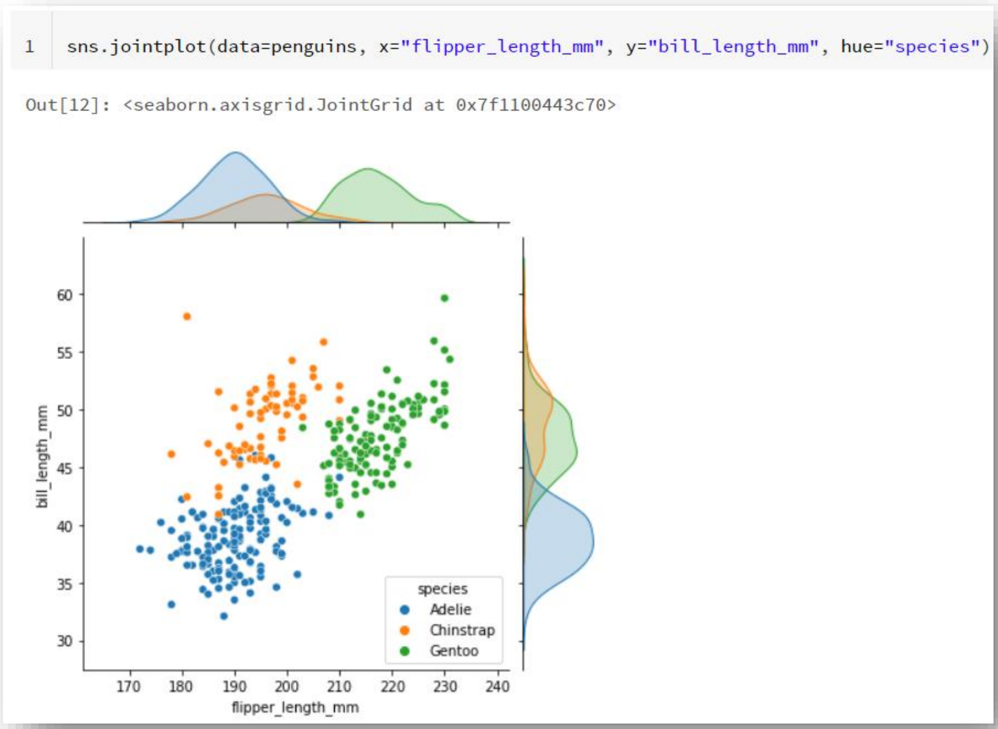
**Example 8:** Some seaborn functions combine multiple kinds of plots to quickly give informative summaries of a dataset. One, jointplot(), focuses on a single relationship. It plots the joint distribution between two variables along with each variable's marginal distribution:
Load and review another seaborn's data sets, **penguins.** The dataset has 7 columns.


Example 8

```
penguins = sns.load_dataset("penguins")

penguins
```

| | species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|---|---|---|---|---|---|---|---|
| **0** | Adelie | Torgersen | 39.1 | 18.7 | 181.0 | 3750.0 | Male |
| **1** | Adelie | Torgersen | 39.5 | 17.4 | 186.0 | 3800.0 | Female |
| **2** | Adelie | Torgersen | 40.3 | 18.0 | 195.0 | 3250.0 | Female |
| **3** | Adelie | Torgersen | NaN | NaN | NaN | NaN | NaN |
| **4** | Adelie | Torgersen | 36.7 | 19.3 | 193.0 | 3450.0 | Female |
| **...** | ... | ... | ... | ... | ... | ... | ... |

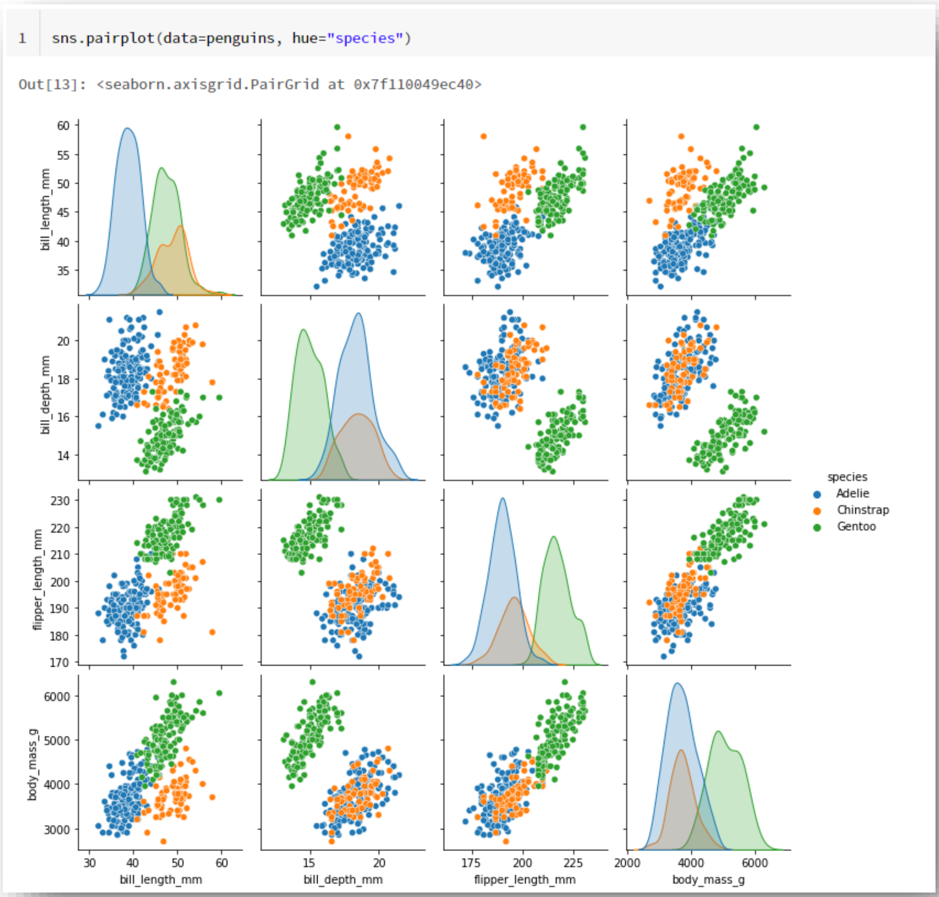
We want to study the body sizes of the different penguin species. It is evident that the 3 species have different body sizes. Try to interpret the graph briefly.



```
sns.jointplot(data=penguins, x="flipper_length_mm", y="bill_length_mm", hue="species")
```

```
Out[12]: <seaborn.axisgrid.JointGrid at 0x7f1100443c70>
```

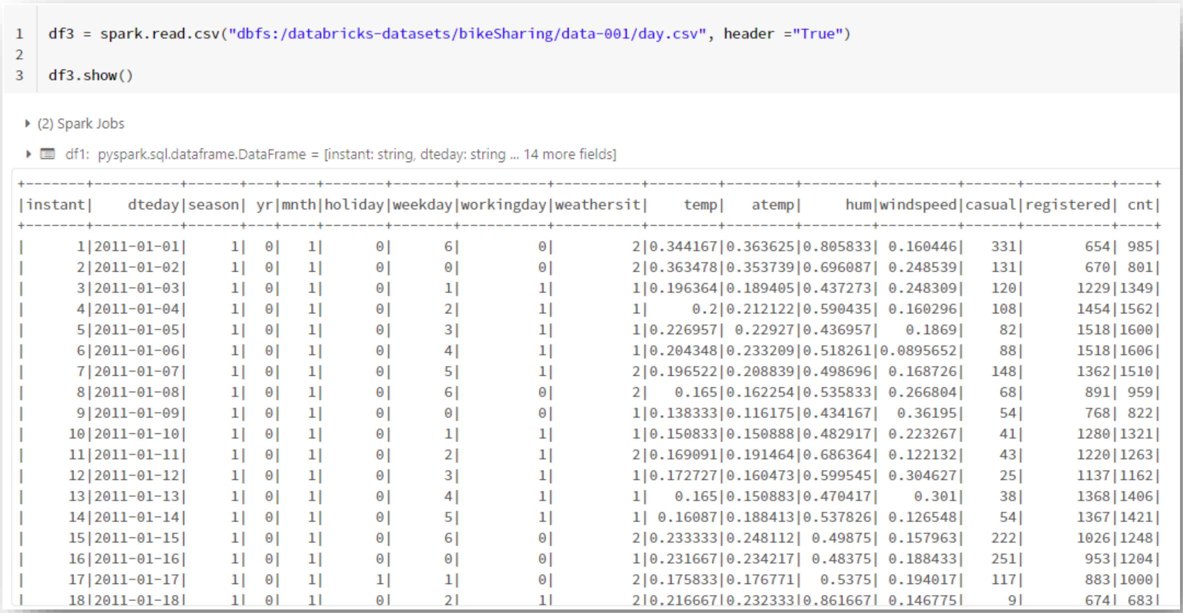
The other, **pairplot()**, takes a broader view: it shows joint and marginal distributions for all pairwise relationships and for each variable, respectively:



```
sns.pairplot(data=penguins, hue="species")
```

```
Out[13]: <seaborn.axisgrid.PairGrid at 0x7f110049ec40>
```

**Challenge 2:** Use one of the Databricks sample dataset, **bikeSharing.**

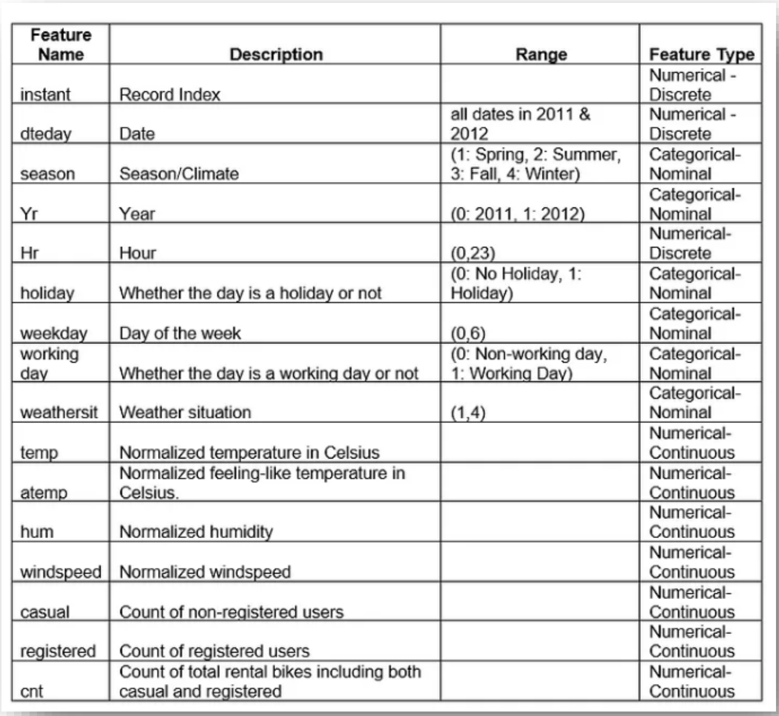


```
df3 = spark.read.csv("dbfs:/databricks-datasets/bikeSharing/data-001/day.csv", header ="True")

df3.show()
```

(2) Spark Jobs

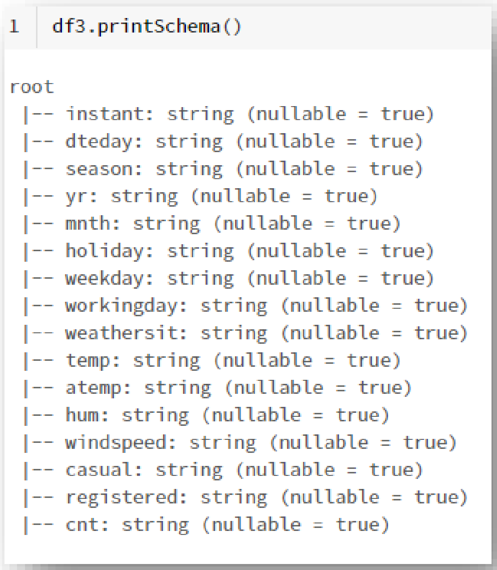
df1: pyspark.sql.dataframe.DataFrame = [instant: string, dteday: string ... 14 more fields]

```
+-------+----------+------+---+----+-------+-------+----------+----------+--------+--------+--------+---------+------+----------+----+
|instant|    dteday|season| yr|mnth|holiday|weekday|workingday|weathersit|    temp|   atemp|     hum|windspeed|casual|registered| cnt|
+-------+----------+------+---+----+-------+-------+----------+----------+--------+--------+--------+---------+------+----------+----+
|      1|2011-01-01|     1|  0|   1|      0|      6|         0|         2|0.344167|0.363625|0.805833| 0.160446|   331|       654| 985|
|      2|2011-01-02|     1|  0|   1|      0|      0|         0|         2|0.363478|0.353739|0.696087| 0.248539|   131|       670| 801|
|      3|2011-01-03|     1|  0|   1|      0|      1|         1|         1|0.196364|0.189405|0.437273| 0.248309|   120|      1229|1349|
|      4|2011-01-04|     1|  0|   1|      0|      2|         1|         1|     0.2|0.212122|0.590435| 0.160296|   108|      1454|1562|
|      5|2011-01-05|     1|  0|   1|      0|      3|         1|         1|0.226957| 0.22927|0.436957|   0.1869|    82|      1518|1600|
|      6|2011-01-06|     1|  0|   1|      0|      4|         1|         1|0.204348|0.233209|0.518261|0.0895652|    88|      1518|1606|
|      7|2011-01-07|     1|  0|   1|      0|      5|         1|         2|0.196522|0.208839|0.498696| 0.168726|   148|      1362|1510|
|      8|2011-01-08|     1|  0|   1|      0|      6|         0|         2|   0.165|0.162254|0.535833| 0.266804|    68|       891| 959|
|      9|2011-01-09|     1|  0|   1|      0|      0|         0|         1|0.138333|0.116175|0.434167|  0.36195|    54|       768| 822|
|     10|2011-01-10|     1|  0|   1|      0|      1|         1|         1|0.150833|0.150888|0.482917| 0.223267|    41|      1280|1321|
|     11|2011-01-11|     1|  0|   1|      0|      2|         1|         2|0.169091|0.191464|0.686364| 0.122132|    43|      1220|1263|
|     12|2011-01-12|     1|  0|   1|      0|      3|         1|         1|0.172727|0.160473|0.599545| 0.304627|    25|      1137|1162|
|     13|2011-01-13|     1|  0|   1|      0|      4|         1|         1|   0.165|0.150883|0.470417|    0.301|    38|      1368|1406|
|     14|2011-01-14|     1|  0|   1|      0|      5|         1|         1| 0.16087|0.188413|0.537826| 0.126548|    54|      1367|1421|
|     15|2011-01-15|     1|  0|   1|      0|      6|         0|         2|0.233333|0.248112| 0.49875| 0.157963|   222|      1026|1248|
|     16|2011-01-16|     1|  0|   1|      0|      0|         0|         1|0.231667|0.234217| 0.48375| 0.188433|   251|       953|1204|
|     17|2011-01-17|     1|  0|   1|      1|      1|         0|         2|0.175833|0.176771|  0.5375| 0.194017|   117|       883|1000|
|     18|2011-01-18|     1|  0|   1|      0|      2|         1|         2|0.216667|0.232333|0.861667| 0.146775|     9|       674| 683|
```



| Feature Name | Description | Range | Feature Type |
|---|---|---|---|
| instant | Record Index | | Numerical - Discrete |
| dteday | Date | all dates in 2011 & 2012 | Numerical - Discrete |
| season | Season/Climate | (1: Spring, 2: Summer, 3: Fall, 4: Winter) | Categorical-Nominal |
| Yr | Year | (0: 2011, 1: 2012) | Categorical-Nominal |
| Hr | Hour | (0,23) | Numerical-Discrete |
| holiday | Whether the day is a holiday or not | (0: No Holiday, 1: Holiday) | Categorical-Nominal |
| weekday | Day of the week | (0,6) | Categorical-Nominal |
| working day | Whether the day is a working day or not | (0: Non-working day, 1: Working Day) | Categorical-Nominal |
| weathersit | Weather situation | (1,4) | Categorical-Nominal |
| temp | Normalized temperature in Celsius | | Numerical-Continuous |
| atemp | Normalized feeling-like temperature in Celsius. | | Numerical-Continuous |
| hum | Normalized humidity | | Numerical-Continuous |
| windspeed | Normalized windspeed | | Numerical-Continuous |
| casual | Count of non-registered users | | Numerical-Continuous |
| registered | Count of registered users | | Numerical-Continuous |
| cnt | Count of total rental bikes including both casual and registered | | Numerical-Continuous |

1) check the data type of the columns:



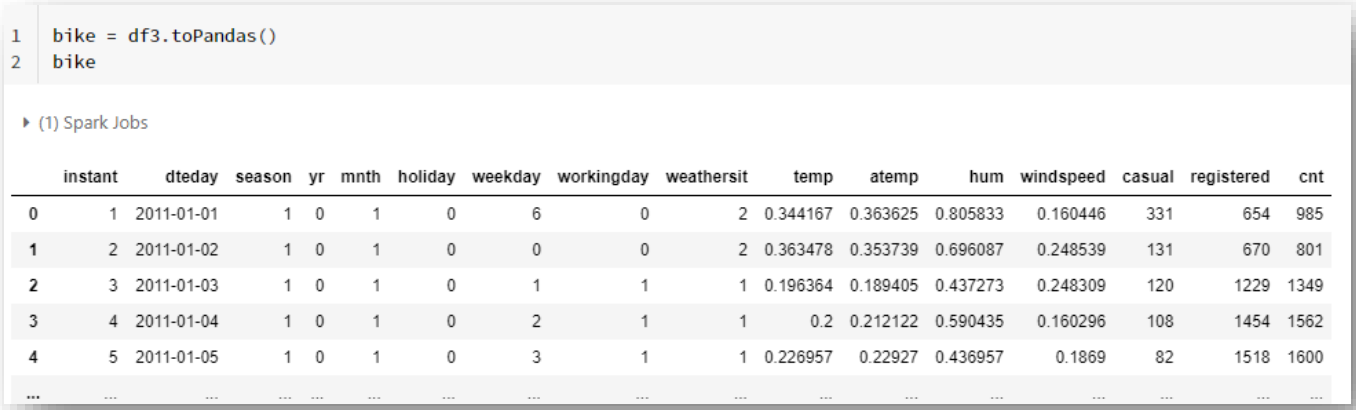
```
df3.printSchema()
```

```
root
 |-- instant: string (nullable = true)
 |-- dteday: string (nullable = true)
 |-- season: string (nullable = true)
 |-- yr: string (nullable = true)
 |-- mnth: string (nullable = true)
 |-- holiday: string (nullable = true)
 |-- weekday: string (nullable = true)
 |-- workingday: string (nullable = true)
 |-- weathersit: string (nullable = true)
 |-- temp: string (nullable = true)
 |-- atemp: string (nullable = true)
 |-- hum: string (nullable = true)
 |-- windspeed: string (nullable = true)
 |-- casual: string (nullable = true)
 |-- registered: string (nullable = true)
 |-- cnt: string (nullable = true)
```

All columns are string, and you cannot plot string variables. Change the data types of some of them.

```
df3 = df3.withColumn(("weekday"), df3.weekday.cast("int"))
df3 = df3.withColumn(("cnt"), df3.cnt.cast("int"))
df3 = df3.withColumn(("workingday"), df3.workingday.cast("int"))
df3 = df3.withColumn(("season"), df3.season.cast("int"))
df3 = df3.withColumn(("weathersit"), df3.weathersit.cast("int"))
```

2) Convert Pyspark DataFrame to Pandas DataFrame.



```
bike = df3.toPandas()
bike
```

▸ (1) Spark Jobs

| | instant | dteday | season | yr | mnth | holiday | weekday | workingday | weathersit | temp | atemp | hum | windspeed | casual | registered | cnt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 2011-01-01 | 1 | 0 | 1 | 0 | 6 | 0 | 2 | 0.344167 | 0.363625 | 0.805833 | 0.160446 | 331 | 654 | 985 |
| **1** | 2 | 2011-01-02 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0.363478 | 0.353739 | 0.696087 | 0.248539 | 131 | 670 | 801 |
| **2** | 3 | 2011-01-03 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0.196364 | 0.189405 | 0.437273 | 0.248309 | 120 | 1229 | 1349 |
| **3** | 4 | 2011-01-04 | 1 | 0 | 1 | 0 | 2 | 1 | 1 | 0.2 | 0.212122 | 0.590435 | 0.160296 | 108 | 1454 | 1562 |
| **4** | 5 | 2011-01-05 | 1 | 0 | 1 | 0 | 3 | 1 | 1 | 0.226957 | 0.22927 | 0.436957 | 0.1869 | 82 | 1518 | 1600 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

3) visualise relation between **weekday** and **cnt** to check if there is any relation between weekdays and number of bike renting.

4) Add **weathersit** as the 3rd variable to explore effect of the weather situation on the bike renting in different days.

Compare 3 kinds of the graph (swarm, violin, and bar), from which graph you can get a clearer inference?

## References

https://matplotlib.org/stable/index.html

https://seaborn.pydata.org/