Data Science

Numpy

Np.array is a central data structure of the NumPy library.

np.array([**1, 2, 3, 4, 5, 6**])

Np.arange is range function of numpy

x = np.arange(**1,10**)

[1 2 3 4 5 6 7 8 9]

Np.zeros creates arrays contains only zeros / also numpy has np.ones function too

np.zeros(**5**)

[0. 0. 0. 0. 0.]

x = np.zeros((**5,5**))

[[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]]

Np.linspace

np.linspace(**0,5,10**)

[0. 0.55555556 1.11111111 1.66666667 2.22222222 2.77777778

3.33333333 3.88888889 4.44444444 5. ]

Np.eye creates identity matrix

np.eye(**4**)

[[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. 1. 0.]

[0. 0. 0. 1.]]

Np.random.rand creates an array of the given shape you pass in and it’s going to populate it with random samples from uniform distribution over 0 to 1.

One dimension

np.random.rand(**5**)

[0.31119053 0.91550138 0.46115721 0.24515765 0.05212258]

Two dimensions

np.random.rand(**3,3**)

[[0.70115019 0.27511006 0.83921891]

[0.86969254 0.96426044 0.22163088]

[0.63076489 0.22895867 0.08046782]]

Np.random.randn

np.random.randn(**3,3**)

[[-0.5315388 0.32636329 0.18243067]

[-0.6846177 -0.57441501 0.01322321]

[ 0.13731295 -2.57329601 0.98334447]]

Np.random.randint(low,high,size,dtype)

np.random.randint(**0,100,10**)

[47 2 96 0 31 90 17 90 39 37]

Np.reshape

arr = np.arange(**1, 26**)

[ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]

x = arr.reshape(**5, 5**)

[[ 1 2 3 4 5]

[ 6 7 8 9 10]

[11 12 13 14 15]

[16 17 18 19 20]

[21 22 23 24 25]]

max() / min() function

ran\_arr = np.random.randint(**0, 50, 10**)

x = max(ran\_arr)

y = min(ran\_arr)

[32 23 40 4 25 15 17 13 34 47]

47-4

argmax() / argmin() function: it gives the index number of max and min values

ran\_arr = np.random.randint(**0, 50, 10**)

x = ran\_arr.argmax()

y = ran\_arr.argmin()

[26 4 17 13 48 49 11 44 4 6]

5-1

Shape function

arr = np.arange(**0, 25**)

newarr = arr.reshape(**5,5**)

ran\_arr = np.random.randint(**0, 50, 10**)

x = ran\_arr.shape

y = newarr.shape

ran\_arr == [26 24 9 35 20 43 48 0 22 44]

arr == [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]

x,y == (10,) -- (5, 5)

Slice function is the same as normal but it also changes the arr not only the slice.



But if we use copy() the arr won’t be affected.



--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

arr = np.array([[**5, 10, 15**]**,** [**20, 25, 30**]**,** [**35, 40, 45**]])

print(arr)

print(arr[**2,1**])

print(arr[:**2,1**:])

[[ 5 10 15]

[20 25 30]

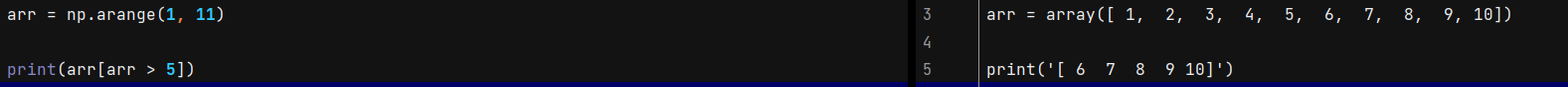
[35 40 45]]

40

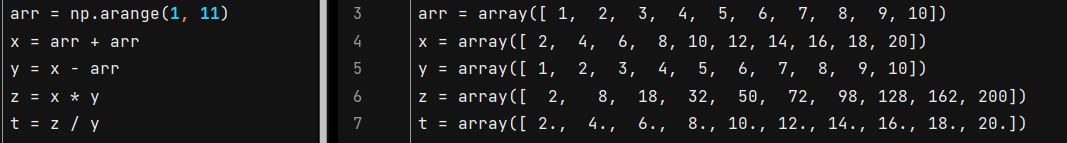
[[10 15]

[25 30]]

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



Operations in Numpy



Np.sqrt gives a square root of the values inside of an array



Np.exp gives a exponential values of array

arr = np.arange(**1, 10**)

exp = np.exp(arr)

[2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01

1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03

8.10308393e+03]

np.sin(), np.log() ,np.cos(), np.sum(), np.std(standard deviation) ext.

**Pandas**

pd.Series() creates series

labels = ["a"**,** "b"**,** "c"]

my\_data = [**10, 20, 30**]

x = pd.Series(data=my\_data**,** index=labels) # Also (my\_data,labels) gives the same output

a 10

b 20

c 30

dtype: int64

Also we can write builtin functions(sum,print,len,exc) inside of a function.

labels = ["a"**,** "b"**,** "c"]

my\_data = [**10, 20, 30**]

x = pd.Series(sum(my\_data)**,** labels)

y = pd.Series(len(my\_data)**,** labels)

a 60

b 60

c 60

dtype: int64

a 3

b 3

c 3

dtype: int64

Taking information from the series.

ser1 = pd.Series([**1, 2, 3, 4**]**,** ["USA"**,** "GERMANY"**,** "POLAND"**,** "JAPAN"])

print(ser1["POLAND"])

3

Sum

ser1 = pd.Series([**1, 2, 3, 4**]**,** ["USA"**,** "GERMANY"**,** "JAPAN"**,** "POLAND"])

ser2 = pd.Series([**1, 2, 5, 4**]**,** ["USA"**,** "GERMANY"**,** "DENMARK"**,** "POLAND"])

print(ser1 + ser2)

DENMARK NaN

GERMANY 4.0

JAPAN NaN

POLAND 8.0

USA 2.0

dtype: float64

pd.DataFrame is Two-dimensional, size-mutable, potentially heterogeneous tabular data. Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Input

np.random.seed(**101**)

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

Output

W X Y Z

A 2.706850 0.628133 0.907969 0.503826

B 0.651118 -0.319318 -0.848077 0.605965

C -2.018168 0.740122 0.528813 -0.589001

D 0.188695 -0.758872 -0.933237 0.955057

E 0.190794 1.978757 2.605967 0.683509

If we try to print W column

print(x["W"])

Or

print(x.W)

Output

W

A -0.901863

B -1.095917

C 0.401786

D -0.392336

E -0.021963

Name: W, dtype: float64

If we want to print many columns

print(x[["W"**,**"X"]])

W X

A -1.074898 -1.355301

B -0.700357 0.684699

C -0.269553 0.806788

D -2.147362 -0.244840

E 0.717607 1.292005

Adding new column

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

x["New"] = x["W"] + x["Z"]

W X Y Z New

A 2.706850 0.628133 0.907969 0.503826 3.210676

B 0.651118 -0.319318 -0.848077 0.605965 1.257083

C -2.018168 0.740122 0.528813 -0.589001 -2.607169

D 0.188695 -0.758872 -0.933237 0.955057 1.143752

E 0.190794 1.978757 2.605967 0.683509 0.874303

Removing column df.drop()

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

x["New"] = x["W"] + x["Z"]

print(x.drop("New"**,** axis=**1,**inplace=True))# If we don’t write inplace=True it won’t change the table eventually

W X Y Z

A 2.706850 0.628133 0.907969 0.503826

B 0.651118 -0.319318 -0.848077 0.605965

C -2.018168 0.740122 0.528813 -0.589001

D 0.188695 -0.758872 -0.933237 0.955057

E 0.190794 1.978757 2.605967 0.683509

For removing row

x.drop("E"**,** axis=**0,** inplace=True)

W X Y Z

A 2.706850 0.628133 0.907969 0.503826

B 0.651118 -0.319318 -0.848077 0.605965

C -2.018168 0.740122 0.528813 -0.589001

D 0.188695 -0.758872 -0.933237 0.955057

x.loc[] / x.iloc[] for selecting rows

print(x.loc["B"])

W 0.651118

X -0.319318

Y -0.848077

Z 0.605965

Name: B, dtype: float64

print(x.iloc[**1**])

W 0.651118

X -0.319318

Y -0.848077

Z 0.605965

Name: B, dtype: float64

Multiple

print(x.loc[["B"**,**"C"]])

W X Y Z

B 0.651118 -0.319318 -0.848077 0.605965

C -2.018168 0.740122 0.528813 -0.589001

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

print(x.loc[["B"**,**"C"]**,**["Y"**,**"Z"]])

Y Z

B -0.848077 0.605965

C 0.528813 -0.589001

Conditional selection in pandas

np.random.seed(**101**)

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

boolx = x > **0**

**print(x)**

W X Y Z

A 2.706850 0.628133 0.907969 0.503826

B 0.651118 -0.319318 -0.848077 0.605965

C -2.018168 0.740122 0.528813 -0.589001

D 0.188695 -0.758872 -0.933237 0.955057

E 0.190794 1.978757 2.605967 0.683509

print(boolx)

W X Y Z

A True True True True

B True False False True

C False True True False

D True False False True

E True True True True

print(x[boolx])

W X Y Z

A 2.706850 0.628133 0.907969 0.503826

B 0.651118 NaN NaN 0.605965

C NaN 0.740122 0.528813 NaN

D 0.188695 NaN NaN 0.955057

E 0.190794 1.978757 2.605967 0.683509

Multiple Conditions

np.random.seed(**101**)

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

df = x[(x["W"] > **0**) & (x["Y"] > **1**)]

print(df)

W X Y Z

E 0.190794 1.978757 2.605967 0.683509

x.reset\_index()

np.random.seed(**101**)

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

print(x.reset\_index())

index W X Y Z

0 A 2.706850 0.628133 0.907969 0.503826

1 B 0.651118 -0.319318 -0.848077 0.605965

2 C -2.018168 0.740122 0.528813 -0.589001

3 D 0.188695 -0.758872 -0.933237 0.955057

4 E 0.190794 1.978757 2.605967 0.683509

Adding new column

np.random.seed(**101**)

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

newind = "CA NY WY OR CO".split()

x["States"] = newind

print(x)

W X Y Z States

A 2.706850 0.628133 0.907969 0.503826 CA

B 0.651118 -0.319318 -0.848077 0.605965 NY

C -2.018168 0.740122 0.528813 -0.589001 WY

D 0.188695 -0.758872 -0.933237 0.955057 OR

E 0.190794 1.978757 2.605967 0.683509 CO

x.set\_index()

np.random.seed(**101**)

x = pd.DataFrame(np.random.randn(**5, 4**)**,** ["A"**,** "B"**,** "C"**,** "D"**,** "E"]**,** ["W"**,** "X"**,** "Y"**,** "Z"])

newind = "CA NY WY OR CO".split()

x["States"] = newind

print(x.set\_index("States"**,** inplace=True))

W X Y Z

States

CA 2.706850 0.628133 0.907969 0.503826

NY 0.651118 -0.319318 -0.848077 0.605965

WY -2.018168 0.740122 0.528813 -0.589001

OR 0.188695 -0.758872 -0.933237 0.955057

CO 0.190794 1.978757 2.605967 0.683509

Index Hierarchy

It goes outside to inside

outside = ["G1"**,** "G1"**,** "G1"**,** "G2"**,** "G2"**,** "G2"]

inside = [**1, 2, 3, 1, 2, 3**]

hier\_index = list(zip(outside**,**inside))

hier\_index = pd.MultiIndex.from\_tuples(hier\_index)

df = pd.DataFrame(np.random.randn(**6, 2**)**,**hier\_index**,**["A"**,**"B"])

print(df)

A B

G1 1 2.706850 0.628133

2 0.907969 0.503826

3 0.651118 -0.319318

G2 1 -0.848077 0.605965

2 -2.018168 0.740122

3 0.528813 -0.589001

For taking G1

print(df.loc["G1"])

A B

1 2.706850 0.628133

2 0.907969 0.503826

3 0.651118 -0.319318

Furthermore

print(df.loc["G1"].loc[**1**])

A 2.706850

B 0.628133

Name: 1, dtype: float64

If we try to see index names

print(df.index.names)

[None, None]

Than we give indexes a name

df.index.names = ["Groups"**,**"Num"]

print(df)

A B

Groups Num

G1 1 2.706850 0.628133

2 0.907969 0.503826

3 0.651118 -0.319318

G2 1 -0.848077 0.605965

2 -2.018168 0.740122

3 0.528813 -0.589001

For taking only one value(Example)

print(df.loc["G2"].loc[**2**]["B"])

0.7401220570561068

Cross section df.xs() we are gonna using this when multiindex

print(df.xs("G1"))

A B

Num

1 2.706850 0.628133

2 0.907969 0.503826

3 0.651118 -0.319318

We can also use xs like this to take 2 values from different outside indexes

print(df.xs(**1,**level="Num"))

A B

Groups

G1 2.706850 0.628133

G2 -0.848077 0.605965

Missing Data

d = {"A": [**1, 2,** np.nan]**,** "B": [**5,** np.nan**,** np.nan]**,** "C": [**1, 2, 3**]}

df = pd.DataFrame(d)

print(df)

A B C

0 1.0 5.0 1

1 2.0 NaN 2

2 NaN NaN 3

df.dropna() drops any columns with nan in it

df.dropna(inplace=True)

print(df)

A B C

0 1.0 5.0 1

Axis=1

df.dropna(axis=**1,**inplace=True)

print(df)

C

0 1

1 2

2 3

Thresh=(num) it keeps at least row num in it

df.dropna(thresh=**2,**inplace=True)

print(df)

A B C

0 1.0 5.0 1

1 2.0 NaN 2

df.fillna()

df.fillna(value="Fill"**,**inplace=True)

print(df)

A B C

0 1.0 5.0 1

1 2.0 Fill 2

2 Fill Fill 3

Filling just “A”

df["A"].fillna(value=df["A"].mean()**,**inplace=True)

print(df)

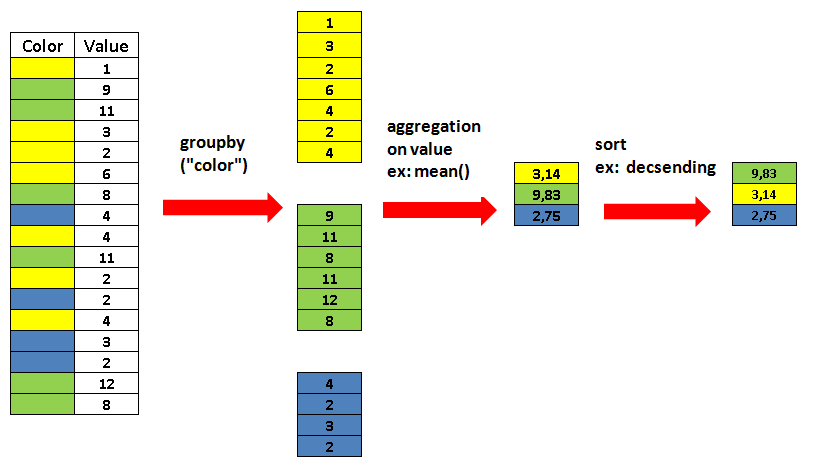
A B C

0 1.0 5.0 1

1 2.0 NaN 2

2 1.5 NaN 3

Groupby



data = {'Company': ['GOOG'**,** 'GOOG'**,** 'MSFT'**,** 'MSFT'**,** 'FB'**,** 'FB']**,**

'Person': ['Sam'**,** 'Charlie'**,** 'Amy'**,** 'Vanessa'**,** 'Carl'**,** 'Sarah']**,**

'Sales': [**200, 120, 340, 124, 243, 350**]}

df = pd.DataFrame(data)

print(df)

Company Person Sales

0 GOOG Sam 200

1 GOOG Charlie 120

2 MSFT Amy 340

3 MSFT Vanessa 124

4 FB Carl 243

5 FB Sarah 350

Sum of company

byComp = df.groupby("Company")

print(byComp.sum())

Sales

Company

FB 593

GOOG 320

MSFT 464

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

print(byComp.sum().loc["FB"])

Sales 593

Name: FB, dtype: int64

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

byComp = df.groupby("Company").sum().loc["FB"]

print(byComp)

Sales 593

Name: FB, dtype: int64

Gives bunch of explanations

byComp = df.groupby("Company").describe()

print(byComp)

count mean std min 25% 50% 75% max

Company

FB 2.0 296.5 75.660426 243.0 269.75 296.5 323.25 350.0

GOOG 2.0 160.0 56.568542 120.0 140.00 160.0 180.00 200.0

MSFT 2.0 232.0 152.735065 124.0 178.00 232.0 286.00 340.0

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

byComp = df.groupby("Company").describe().transpose()

print(byComp)

Company FB GOOG MSFT

Sales count 2.000000 2.000000 2.000000

mean 296.500000 160.000000 232.000000

std 75.660426 56.568542 152.735065

min 243.000000 120.000000 124.000000

25% 269.750000 140.000000 178.000000

50% 296.500000 160.000000 232.000000

75% 323.250000 180.000000 286.000000

max 350.000000 200.000000 340.000000

Process finished with exit code 0

Concatenation

Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on. You can use \*\*pd.concat\*\* and pass in a list of DataFrames to concatenate together:

import pandas as pd

df1 = pd.DataFrame({'A': ['A0'**,** 'A1'**,** 'A2'**,** 'A3']**,**

'B': ['B0'**,** 'B1'**,** 'B2'**,** 'B3']**,**

'C': ['C0'**,** 'C1'**,** 'C2'**,** 'C3']**,**

'D': ['D0'**,** 'D1'**,** 'D2'**,** 'D3']}**,**

index=[**0, 1, 2, 3**])

df2 = pd.DataFrame({'A': ['A4'**,** 'A5'**,** 'A6'**,** 'A7']**,**

'B': ['B4'**,** 'B5'**,** 'B6'**,** 'B7']**,**

'C': ['C4'**,** 'C5'**,** 'C6'**,** 'C7']**,**

'D': ['D4'**,** 'D5'**,** 'D6'**,** 'D7']}**,**

index=[**4, 5, 6, 7**])

df3 = pd.DataFrame({'A': ['A8'**,** 'A9'**,** 'A10'**,** 'A11']**,**

'B': ['B8'**,** 'B9'**,** 'B10'**,** 'B11']**,**

'C': ['C8'**,** 'C9'**,** 'C10'**,** 'C11']**,**

'D': ['D8'**,** 'D9'**,** 'D10'**,** 'D11']}**,**

index=[**8, 9, 10, 11**])

conc = pd.concat([df1**,**df2**,**df3])

print(conc)

A B C D

0 A0 B0 C0 D0

1 A1 B1 C1 D1

2 A2 B2 C2 D2

3 A3 B3 C3 D3

4 A4 B4 C4 D4

5 A5 B5 C5 D5

6 A6 B6 C6 D6

7 A7 B7 C7 D7

8 A8 B8 C8 D8

9 A9 B9 C9 D9

10 A10 B10 C10 D10

11 A11 B11 C11 D11

Axis = 1

conc = pd.concat([df1**,**df2**,**df3]**,**axis=**1**)

A B C D A B C D A B C D

0 A0 B0 C0 D0 NaN NaN NaN NaN NaN NaN NaN NaN

1 A1 B1 C1 D1 NaN NaN NaN NaN NaN NaN NaN NaN

2 A2 B2 C2 D2 NaN NaN NaN NaN NaN NaN NaN NaN

3 A3 B3 C3 D3 NaN NaN NaN NaN NaN NaN NaN NaN

4 NaN NaN NaN NaN A4 B4 C4 D4 NaN NaN NaN NaN

5 NaN NaN NaN NaN A5 B5 C5 D5 NaN NaN NaN NaN

6 NaN NaN NaN NaN A6 B6 C6 D6 NaN NaN NaN NaN

7 NaN NaN NaN NaN A7 B7 C7 D7 NaN NaN NaN NaN

8 NaN NaN NaN NaN NaN NaN NaN NaN A8 B8 C8 D8

9 NaN NaN NaN NaN NaN NaN NaN NaN A9 B9 C9 D9

10 NaN NaN NaN NaN NaN NaN NaN NaN A10 B10 C10 D10

11 NaN NaN NaN NaN NaN NaN NaN NaN A11 B11 C11 D11

Merging

The \*\*merge\*\* function allows you to merge DataFrames together using a similar logic as merging SQL Tables together. For example:

left = pd.DataFrame({'key': ['K0'**,** 'K1'**,** 'K2'**,** 'K3']**,**

'A': ['A0'**,** 'A1'**,** 'A2'**,** 'A3']**,**

'B': ['B0'**,** 'B1'**,** 'B2'**,** 'B3']})

right = pd.DataFrame({'key': ['K0'**,** 'K1'**,** 'K2'**,** 'K3']**,**

'C': ['C0'**,** 'C1'**,** 'C2'**,** 'C3']**,**

'D': ['D0'**,** 'D1'**,** 'D2'**,** 'D3']})

merg = pd.merge(left**,**right**,**how="inner"**,**on="key")

print(merg)

key A B C D

0 K0 A0 B0 C0 D0

1 K1 A1 B1 C1 D1

2 K2 A2 B2 C2 D2

3 K3 A3 B3 C3 D3

Double keys

left = pd.DataFrame({'key1': ['K0'**,** 'K0'**,** 'K1'**,** 'K2']**,**

'key2': ['K0'**,** 'K1'**,** 'K0'**,** 'K1']**,**

'A': ['A0'**,** 'A1'**,** 'A2'**,** 'A3']**,**

'B': ['B0'**,** 'B1'**,** 'B2'**,** 'B3']})

right = pd.DataFrame({'key1': ['K0'**,** 'K1'**,** 'K1'**,** 'K2']**,**

'key2': ['K0'**,** 'K0'**,** 'K0'**,** 'K0']**,**

'C': ['C0'**,** 'C1'**,** 'C2'**,** 'C3']**,**

'D': ['D0'**,** 'D1'**,** 'D2'**,** 'D3']})

merg = pd.merge(left**,** right**,** on=['key1'**,** 'key2'])

print(merg)

key1 key2 A B C D

0 K0 K0 A0 B0 C0 D0

1 K1 K0 A2 B2 C1 D1

2 K1 K0 A2 B2 C2 D2

Joining

Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

left = pd.DataFrame({'A': ['A0'**,** 'A1'**,** 'A2']**,**

'B': ['B0'**,** 'B1'**,** 'B2']}**,**

index=['K0'**,** 'K1'**,** 'K2'])

right = pd.DataFrame({'C': ['C0'**,** 'C2'**,** 'C3']**,**

'D': ['D0'**,** 'D2'**,** 'D3']}**,**

index=['K0'**,** 'K2'**,** 'K3'])

joinn = left.join(right)

print(joinn)

A B C D

K0 A0 B0 C0 D0

K1 A1 B1 NaN NaN

K2 A2 B2 C2 D2

Outer

joinn = left.join(right**,** how='outer')

print(joinn)

A B C D

K0 A0 B0 C0 D0

K1 A1 B1 NaN NaN

K2 A2 B2 C2 D2

K3 NaN NaN C3 D3

Operations

There are lots of operations with pandas that will be really useful to you, but don't fall into any distinct category. Let's show them here in this lecture:

df = pd.DataFrame({'col1':[**1,2,3,4**]**,**'col2':[**444,555,666,444**]**,**'col3':['abc'**,**'def'**,**'ghi'**,**'xyz']})

df.head()

print(df)

col1 col2 col3

0 1 444 abcfif

1 2 555 def

2 3 666 ghi

3 4 444 xyz

Unique values **\*\*unique()\*\***

uniq = df['col2'].unique()

print(uniq)

[444 555 666]

Number of uniques **\*\*nunique()\*\***

uniq = df['col2'].nunique()

print(uniq)

3

Counting values inside of column or row **\*\*value\_counts()\*\***

uniq = df['col2'].value\_counts()

print(uniq)

444 2

555 1

666 1

Name: col2, dtype: int64

Selecting Data

df = pd.DataFrame({'col1':[**1,2,3,4**]**,**'col2':[**444,555,666,444**]**,**'col3':['abc'**,**'def'**,**'ghi'**,**'xyz']})

#Select from DataFrame using criteria from multiple columns

newdf = df[(df['col1']>**2**) & (df['col2']==**444**)]

print(newdf)

col1 col2 col3

3 4 444 xyz

Applying Functions

df = pd.DataFrame({'col1': [**1, 2, 3, 4**]**,** 'col2': [**444, 555, 666, 444**]**,** 'col3': ['abc'**,** 'def'**,** 'ghi'**,** 'xyz']})

print(df)

col1 col2 col3

0 1 444 abc

1 2 555 def

2 3 666 ghi

3 4 444 xyz

Imagine we have a random function such as:

def times2(x):

return x \* **2**

We can apply this function on our data frame

newdf = df['col2'].apply(times2)

print(newdf)

0 888

1 1110

2 1332

3 888

Name: col2, dtype: int64

Or

newdf = df.apply(times2)

print(newdf)

col1 col2 col3

0 2 888 abcabc

1 4 1110 defdef

2 6 1332 ghighi

3 8 888 xyzxyz

Example len function

newdf = df["col3"].apply(len)

print(newdf)

0 3

1 3

2 3

3 3

Name: col3, dtype: int64

Sum function

newdf = df["col2"].sum()

print(newdf)

2109

Lambda function is very important

newdf = df["col2"].apply(lambda x: x \* **2**)

print(newdf)

0 888

1 1110

2 1332

3 888

Name: col2, dtype: int64

Sorting

new\_df = df.sort\_values("col2")

print(new\_df)

col1 col2 col3

0 1 444 abc

3 4 444 xyz

1 2 555 def

2 3 666 ghi

Data Input and Output

CSV / Excel / HTML / SQL

CSV

We have csv file inside of an our project’s folder

df = pd.read\_csv("example")

print(df)

a b c d

0 0 1 2 3

1 4 5 6 7

2 8 9 10 11

3 12 13 14 15

Excel

df = pd.read\_excel("Excel\_Sample.xlsx"**,**sheet\_name="Sheet1")

print(df)

Unnamed: 0 a b c d

0 0 0 1 2 3

1 1 4 5 6 7

2 2 8 9 10 11

3 3 12 13 14 15

HTML

df = pd.read\_html("https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list/")

print(df)

[ Bank NameBank ... FundFund

0 Almena State Bank ... 10538

1 First City Bank of Florida ... 10537

2 The First State Bank ... 10536

3 Ericson State Bank ... 10535

4 City National Bank of New Jersey ... 10534

.. ... ... ...

558 Superior Bank, FSB ... 6004

559 Malta National Bank ... 4648

560 First Alliance Bank & Trust Co. ... 4647

561 National State Bank of Metropolis ... 4646

562 Bank of Honolulu ... 4645

[563 rows x 7 columns]]

SQL

import pandas as pd

import sqlalchemy as sa

df = pd.read\_csv("example")

engine = sa.create\_engine("sqlite:///:memory:")

df.to\_sql("my\_table"**,** engine)

sqldf = pd.read\_sql("my\_table"**,** con=engine)

print(sqldf)

index a b c d

0 0 0 1 2 3

1 1 4 5 6 7

2 2 8 9 10 11

3 3 12 13 14 15

Examples

\*\* What is the name of the highest paid person (including benefits)?\*\*

import pandas as pd

df = pd.read\_csv("Salaries.csv")

az = df[df["TotalPayBenefits"] == df["TotalPayBenefits"].max()]["EmployeeName"] # Variation 1

ava = df.loc[df["TotalPayBenefits"].idxmax()]["EmployeeName"] # Variation 2

print(ava)

Id 1

EmployeeName NATHANIEL FORD

JobTitle GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY

BasePay 167411.18

OvertimePay 0.0

OtherPay 400184.25

Benefits NaN

TotalPay 567595.43

TotalPayBenefits 567595.43

Year 2011

Notes NaN

Agency San Francisco

Status NaN

Name: 0, dtype: object

\*\* What is the name of the lowest paid person?\*\*

df = pd.read\_csv("Salaries.csv")

ava = df.loc[df["TotalPayBenefits"].idxmin()]

print(ava)

Id 148654

EmployeeName Joe Lopez

JobTitle Counselor, Log Cabin Ranch

BasePay 0.0

OvertimePay 0.0

OtherPay -618.13

Benefits 0.0

TotalPay -618.13

TotalPayBenefits -618.13

Year 2014

Notes NaN

Agency San Francisco

Status NaN

Name: 148653, dtype: object

\*\* What was the average (mean) BasePay of all employees per year? (2011-2014) ? \*\*

df = pd.read\_csv("Salaries.csv")

ava = df.groupby("Year").mean()["BasePay"]

print(ava)

Year

2011 63595.956517

2012 65436.406857

2013 69630.030216

2014 66564.421924

Name: BasePay, dtype: float64

\*\* How many unique job titles are there? \*\*

import pandas as pd

df = pd.read\_csv("Salaries.csv")

zort = df["JobTitle"].nunique()

print(zort)

2159

\*\* What are the top 5 most common jobs? \*\*

zort = df["JobTitle"].value\_counts().head(**5**)

print(zort)

Transit Operator 7036

Special Nurse 4389

Registered Nurse 3736

Public Svc Aide-Public Works 2518

Police Officer 3 2421

Name: JobTitle, dtype: int64

\*\* How many Job Titles were represented by only one person in 2013? (e.g. Job Titles with only one occurence in 2013?) \*\*

zort = sum(df[df["Year"] == **2013**]["JobTitle"].value\_counts() == **1**)

print(zort)

202

\*\* How many people have the word Chief in their job title? (This is pretty tricky) \*\*

def chief\_str(title):

if "chief" in title.lower().split():

return True

else:

return False

zort = sum(df["JobTitle"].apply(lambda x: chief\_str(x)))

print(zort)

477

\*\* Hard: How many people have a credit card that expires in 2025? \*\*

import pandas as pd

df = pd.read\_csv("Ecommerce Purchases")

def expdate(year):

x = year.split("/")

intnum = int("".join(x))

if (intnum - **25**) % **10** == **0**:

return True

else:

return False

zort = sum(df["CC Exp Date"].apply(lambda x: expdate(x)))

print(zort)

1033

And the short version of it

import pandas as pd

df = pd.read\_csv("Ecommerce Purchases")

zort = sum(df["CC Exp Date"].apply(lambda x: x[**3**:] == "25"))

print(zort)

1033

\*\* Hard: What are the top 5 most popular email providers/hosts (e.g. gmail.com, yahoo.com, etc...) \*\*

import pandas as pd

df = pd.read\_csv("Ecommerce Purchases")

def mail\_provider(prov):

x = prov.split("@")

return x[**1**]

zort = df["Email"].apply(lambda x: mail\_provider(x)).value\_counts().head(**5**)

print(zort)

hotmail.com 1638

yahoo.com 1616

gmail.com 1605

smith.com 42

williams.com 37

Name: Email, dtype: int64

And the short version of it

import pandas as pd

df = pd.read\_csv("Ecommerce Purchases")

zort = df["Email"].apply(lambda x: x.split("@")[**1**]).value\_counts().head(**5**)

print(zort)

hotmail.com 1638

yahoo.com 1616

gmail.com 1605

smith.com 42

williams.com 37

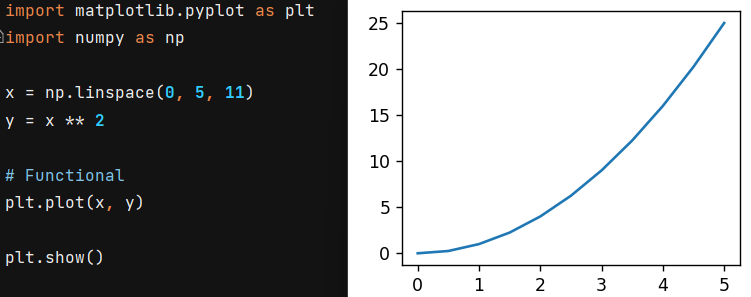
Name: Email, dtype: int64

**Matplotlib**

For importing matplotlib we use

import matplotlib.pyplot as plt

Than **plt.show()**



We can define color as “r-” , ”b-” etc.

plt.plot(x**,** y**,**"g-")

Furthermore

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

# Object Oriented

fig = plt.figure()

axes = fig.add\_axes([**0.1, 0.1, 0.8, 0.8**])

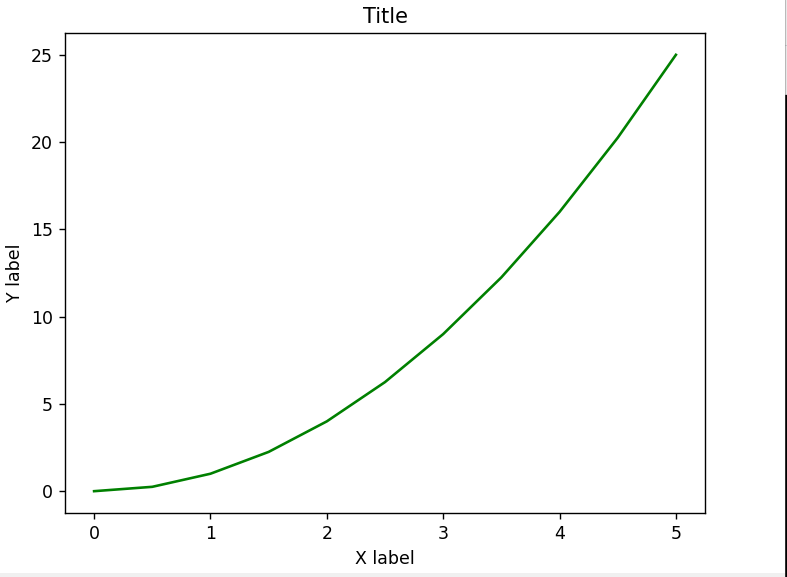
axes.plot(x**,** y**,** "g")

axes.set\_xlabel("X label")

axes.set\_ylabel("Y label")

axes.set\_title("Title")

plt.show()



import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

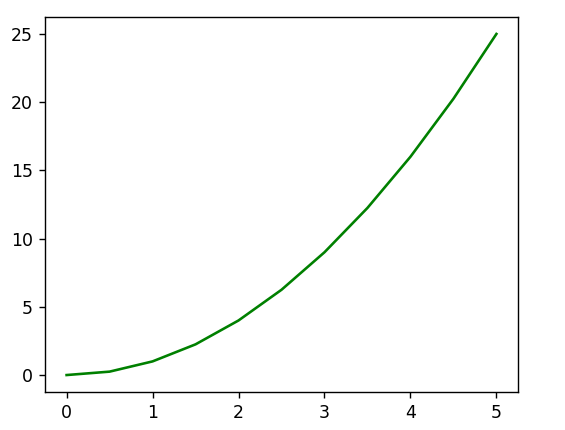
# Object Oriented

fig = plt.figure()

axes = fig.add\_axes([**0.1,0.1,0.8,0.8**])

axes.plot(x**,**y**,**"g")

plt.show()



**fig.add\_axes()**

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

# Object Oriented

fig = plt.figure()

axes1 = fig.add\_axes([**0.1, 0.1, 0.8, 0.8**])

axes2 = fig.add\_axes([**0.2, 0.5, 0.4, 0.3**])

axes1.plot(x**,** y**,** "g")

axes1.set\_xlabel("X label")

axes1.set\_ylabel("Y label")

axes1.set\_title("Bigger")

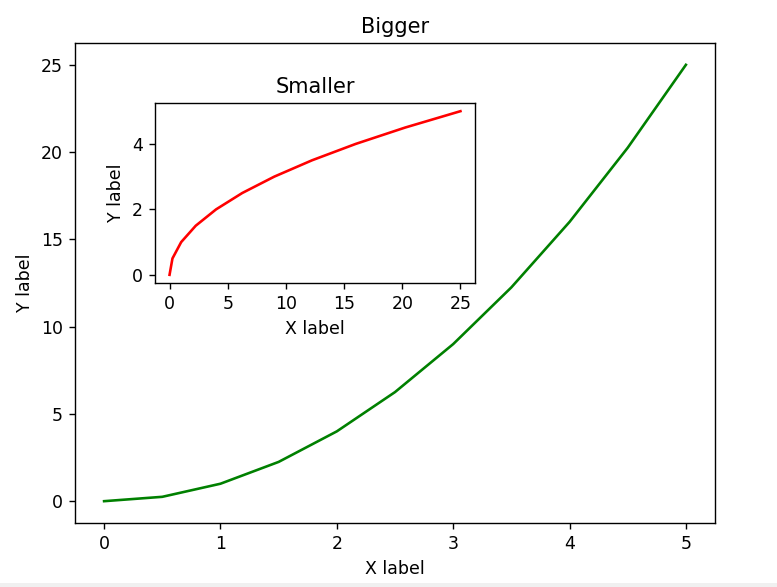
axes2.plot(y**,** x**,** "r")

axes2.set\_xlabel("X label")

axes2.set\_ylabel("Y label")

axes2.set\_title("Smaller")

plt.show()



Subplot

**plt.subplot(nrows, ncols, plot\_number)**

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

# Functional

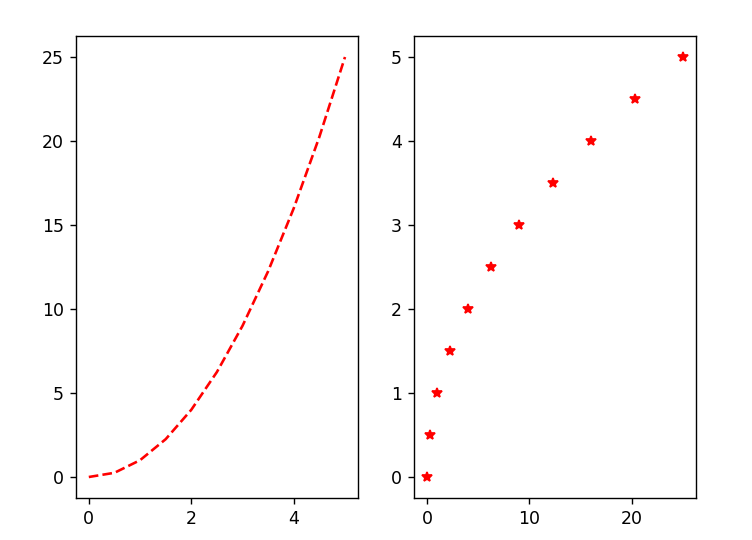
plt.subplot(**1, 2, 1**)

plt.plot(x**,** y**,** "r--")

plt.subplot(**1, 2, 2**)

plt.plot(y**,** x**,** "r\*")

plt.show()



import matplotlib.pyplot as plt

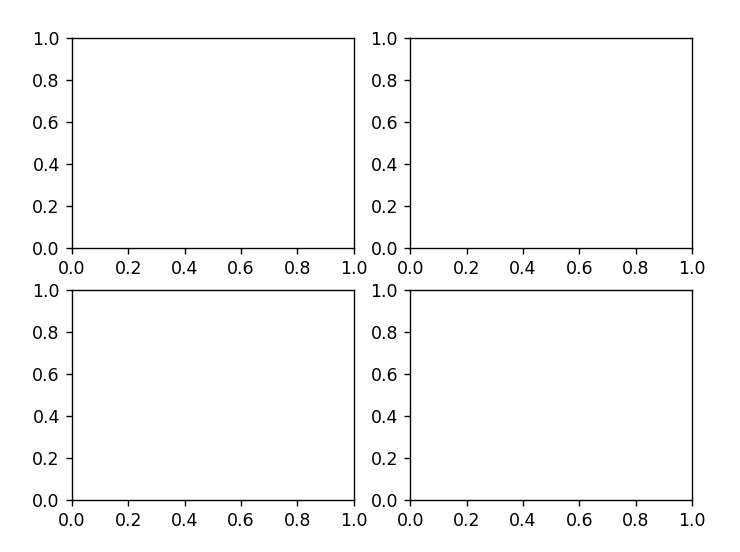
import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

fig**,** axes = plt.subplots(nrows=**2,** ncols=**2**)

plt.show()



**plt.tight\_layout() is fixing layouts**

Axes are iterable in this form and we can give index to reach them

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

fig**,** axes = plt.subplots(nrows=**1,** ncols=**2**)

axes[**0**].plot(x**,**y)

plt.show()

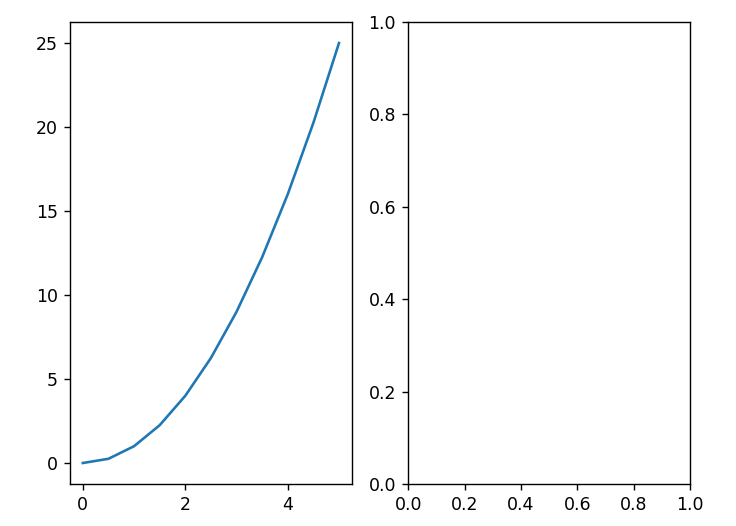


Figure Size and DPI

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

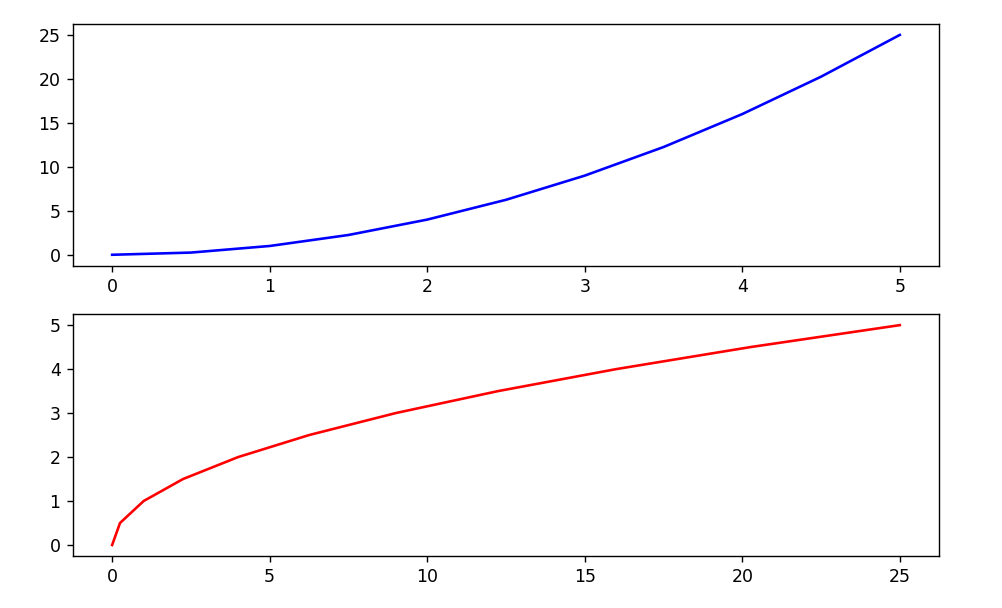
y = x \*\* **2**

fig**,** axes = plt.subplots(nrows=**2,** ncols=**1,** figsize=(**8, 2**))

axes[**0**].plot(x**,** y**,** "b")

axes[**1**].plot(y**,** x**,** "r")

plt.show()



To save the file

fig.savefig("my\_picture.png")

**plt.legend()**

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(**0, 5, 11**)

y = x \*\* **2**

fig = plt.figure()

ax = fig.add\_axes([**0.1, 0.1, 0.8, 0.8**])

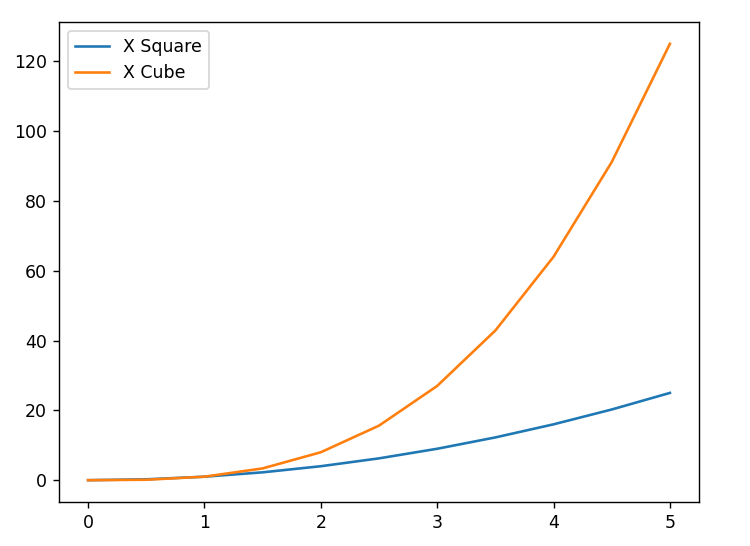
ax.plot(x**,** x \*\* **2,** label="X Square")

ax.plot(x**,** x \*\* **3,** label="X Cube")

plt.legend()

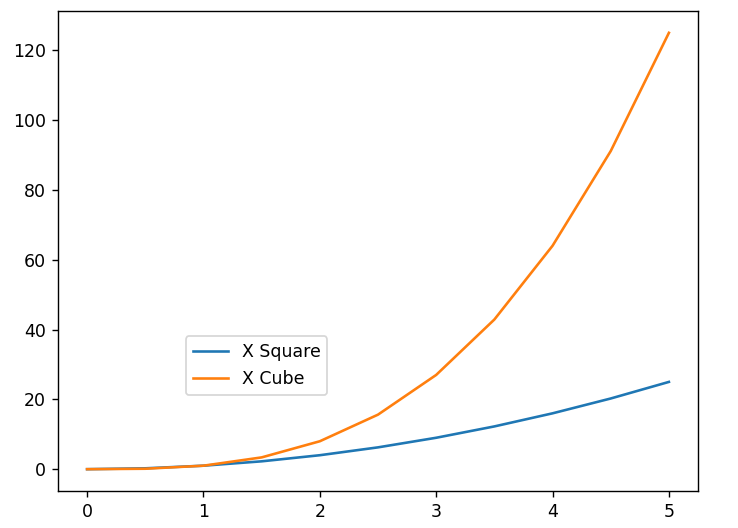
plt.tight\_layout()

plt.show()



We can give coordinates for legend like this

plt.legend(loc=(**0.2,0.2**))



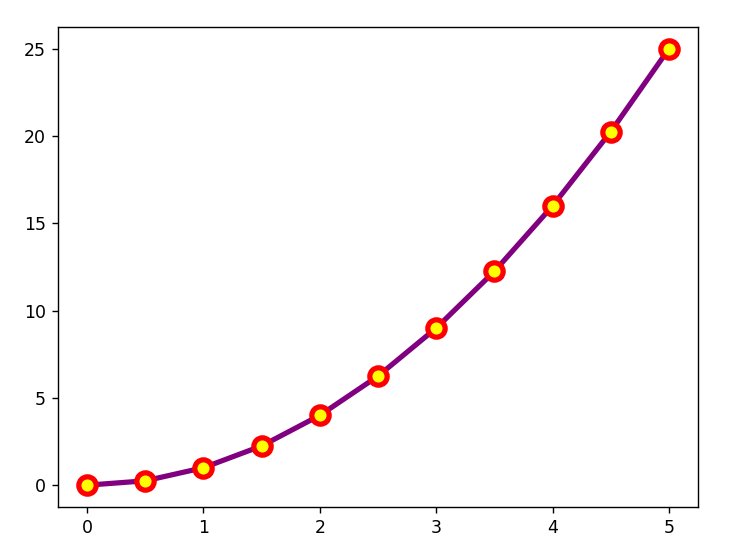
Changing the types

ax.plot(x**,** y**,** color="purple"**,** lw=**3,** alpha=**0.5,** linestyle="--")



With markers and its customization

ax.plot(x**,** y**,** color="purple"**,** lw=**3,** alpha=**1,** linestyle="-"**,** marker="o"**,** markersize=**10,** markerfacecolor="yellow"**,** markeredgewidth=**3,** markeredgecolor="red")



Setting Limits **set\_xlim()**

import matplotlib.pyplot as plt

import numpy as np

x = np.arange(**0, 100**)

y = x \* **2**

z = x \*\* **2**

fig = plt.figure()

ax1 = fig.add\_axes([**0, 0, 1, 1**])

ax1.plot(z**,** "r")

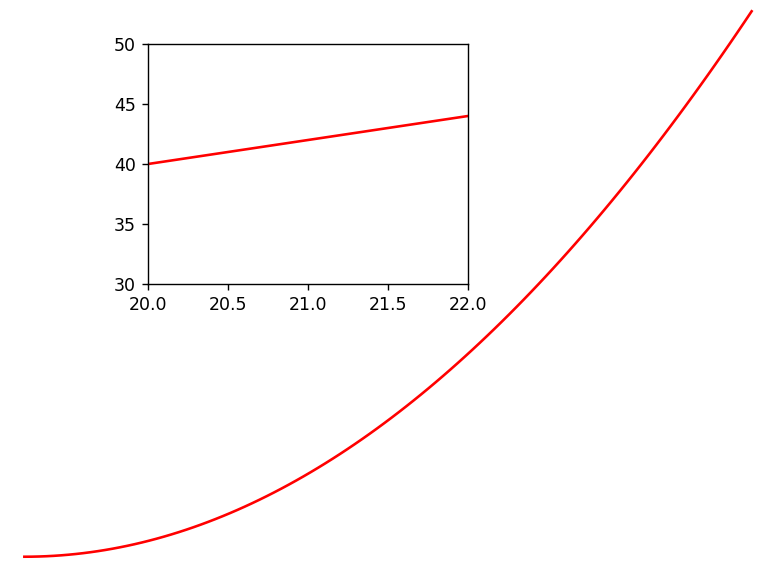
ax2 = fig.add\_axes([**0.2, 0.5, 0.4, 0.4**])

ax2.plot(x**,** y**,** "r")

ax2.set\_xlim([**20, 22**])

ax2.set\_ylim([**30, 50**])

plt.show()



**Seaborn**

Distribution Plot

import seaborn as sns

import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

tips.head()

print(tips.head())

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

The distplot shows the distribution of a univariate set of observations.

import seaborn as sns

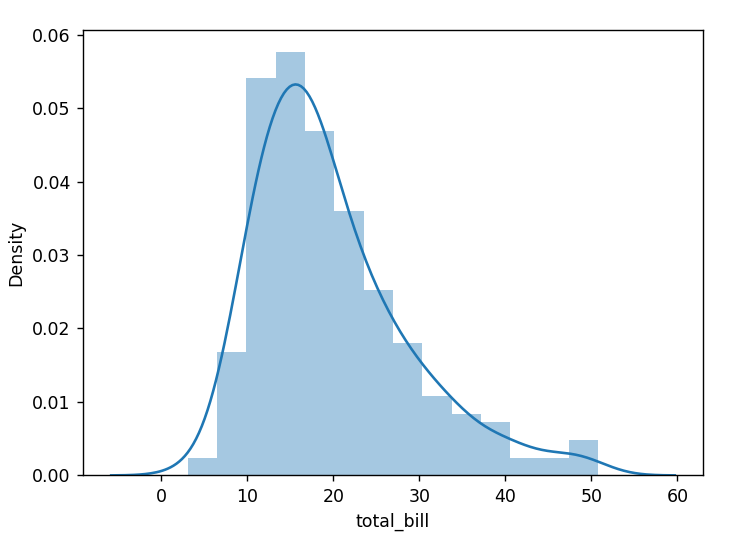
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

zort = sns.distplot(tips['total\_bill'])

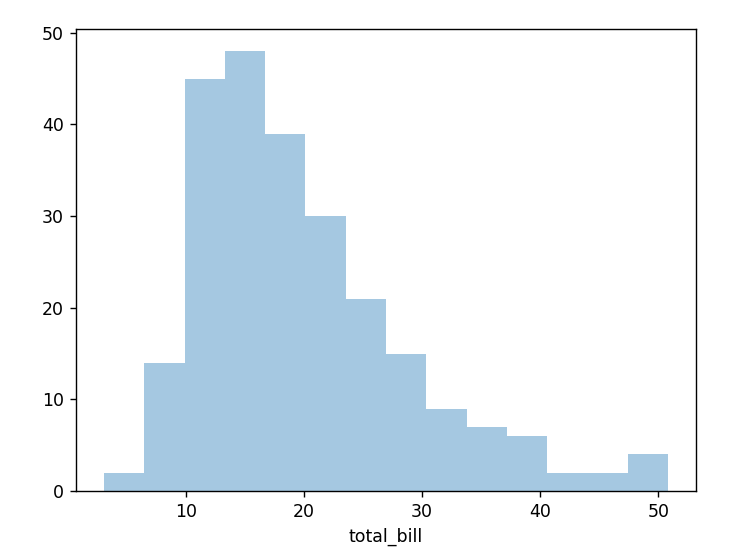
plt.show()

print(zort)



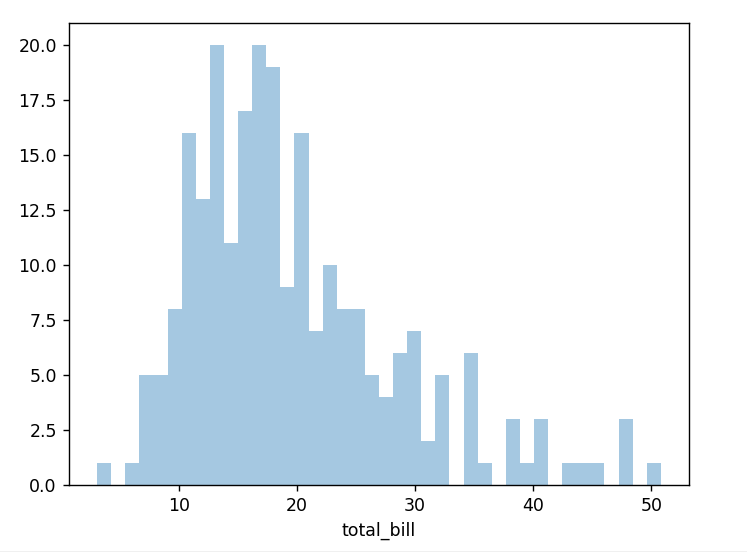
kde=False

zort = sns.distplot(tips['total\_bill']**,**kde=False)



bins=(int)

zort = sns.distplot(tips['total\_bill']**,**kde=False**,**bins=**40**)



Jointplot

Jointplot() allows you to basically match up two dist plots for bivariate data. With your choice of what \*\*kind\*\* parameter to compare with:

import seaborn as sns

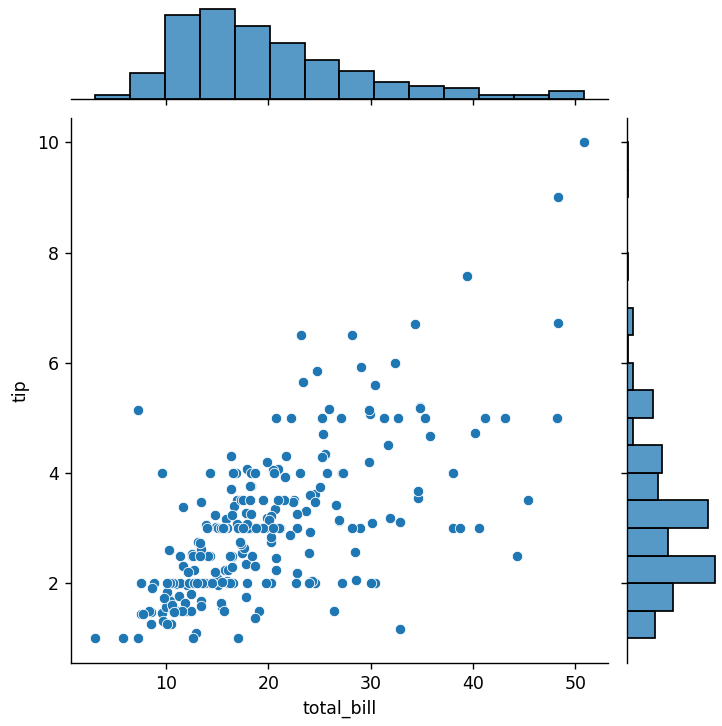
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

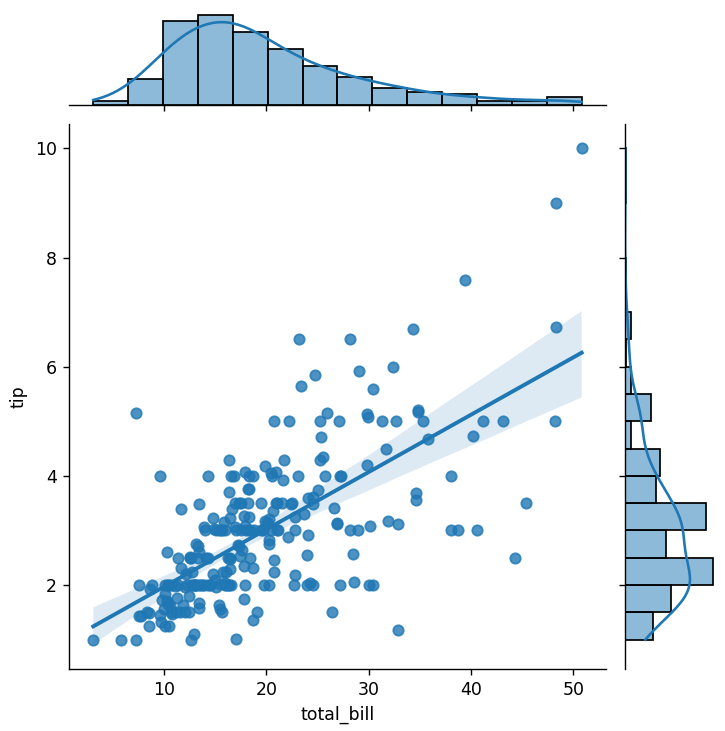
zort = sns.jointplot(x='total\_bill'**,** y='tip'**,** data=tips**,** kind='scatter')

plt.show()

print(zort)



zort = sns.jointplot(x='total\_bill'**,** y='tip'**,** data=tips**,** kind='reg')



Pairplot

Pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns) and supports a color hue argument (for categorical columns).

import seaborn as sns

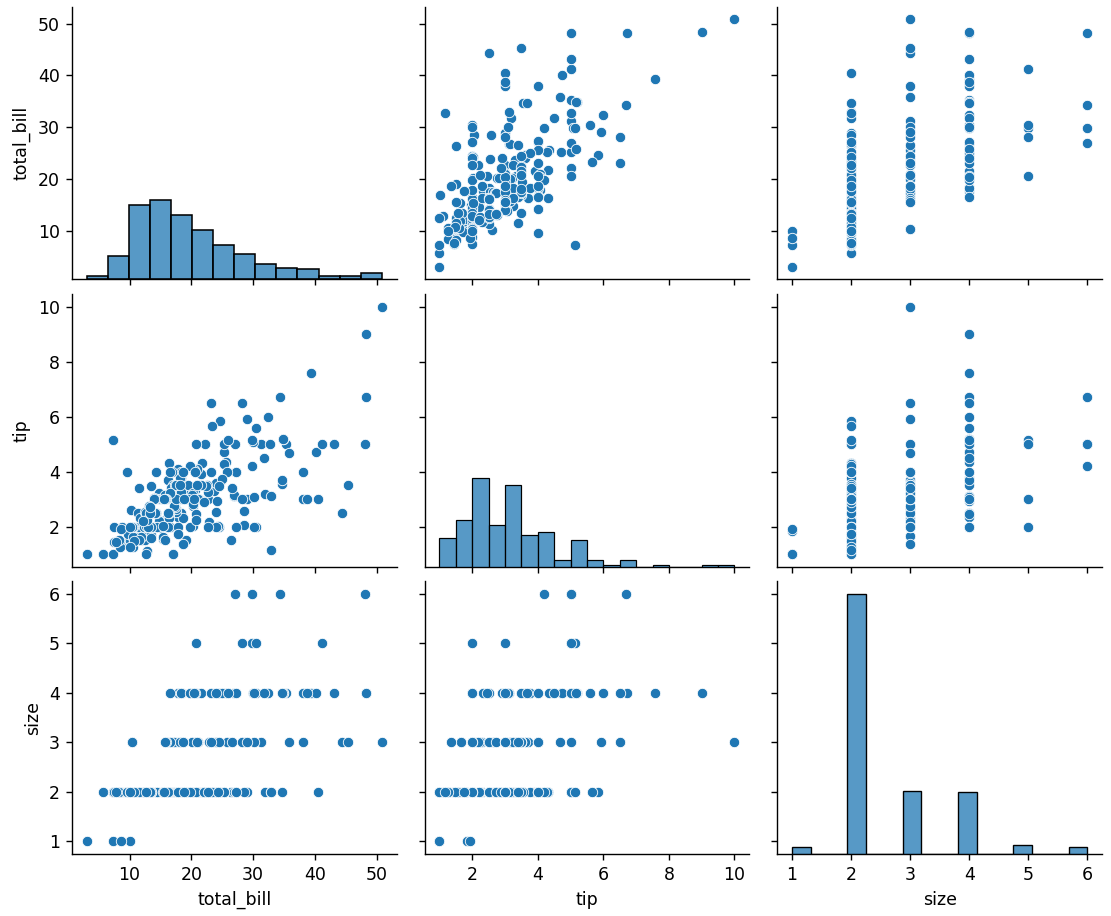
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

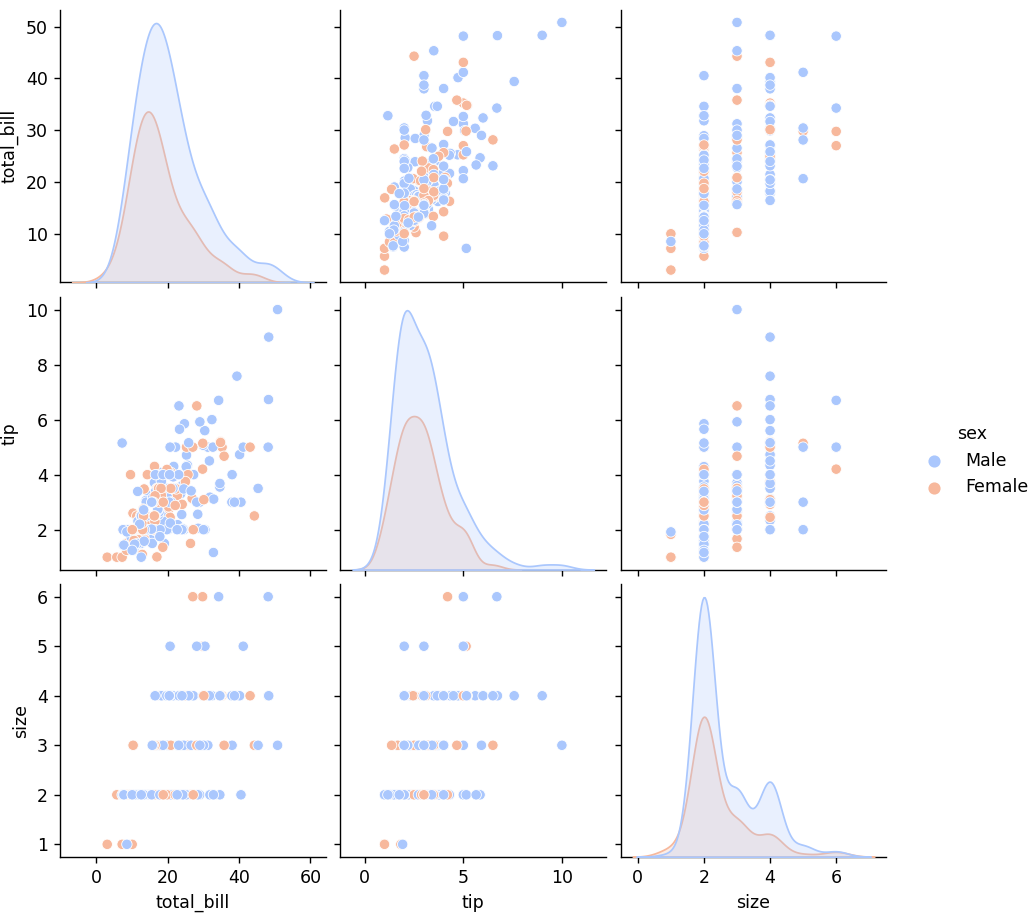
zort = sns.pairplot(tips)

plt.show()

print(zort)



zort = sns.pairplot(tips**,**hue='sex'**,**palette='coolwarm')



Rugplots

Rug plots are actually a very simple concept, they just draw a dash mark for every point on a univariate distribution. They are the building block of a KDE plot:

sns.rugplot

**Categorical Data Plots**

Barplot

These very similar plots allow you to get aggregate data off a categorical feature in your data. \*\*barplot\*\* is a general plot that allows you to aggregate the categorical data based off some function, by default the mean: We can change the default to (estimater=)

import seaborn as sns

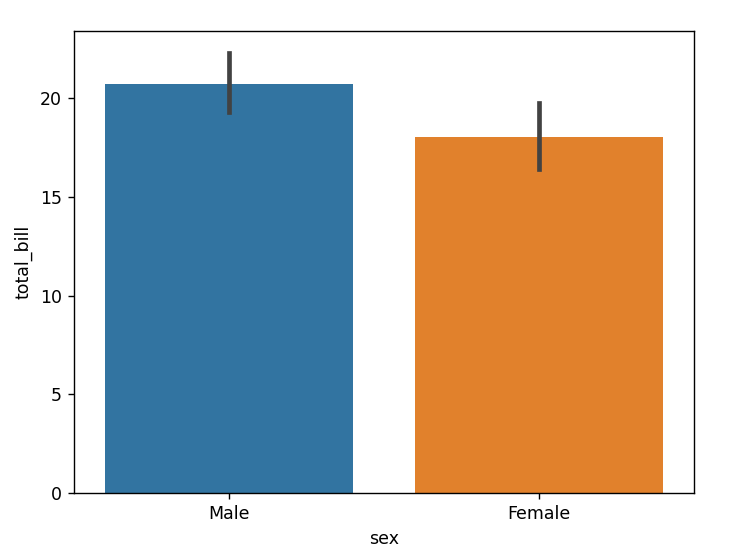
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

zort = sns.barplot(x='sex'**,**y='total\_bill'**,**data=tips)

plt.show()

print(zort)



Countplot

This is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why we only pass the x value:

import seaborn as sns

import matplotlib.pyplot as plt

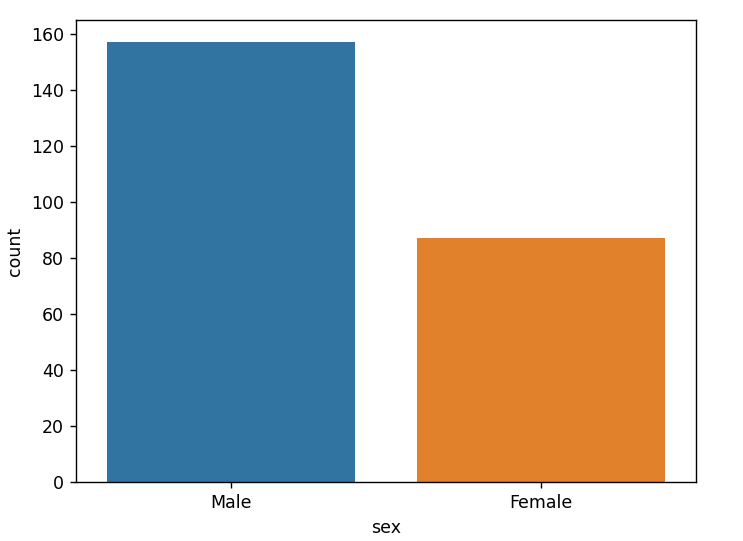
import numpy as np

tips = sns.load\_dataset('tips')

zort = sns.countplot(x='sex'**,**data=tips)

plt.show()

print(zort)



Boxplot and Violinplot

Boxplots and violinplots are used to show the distribution of categorical data. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers'' using a method that is a function of the inter-quartile range.

import seaborn as sns

import matplotlib.pyplot as plt

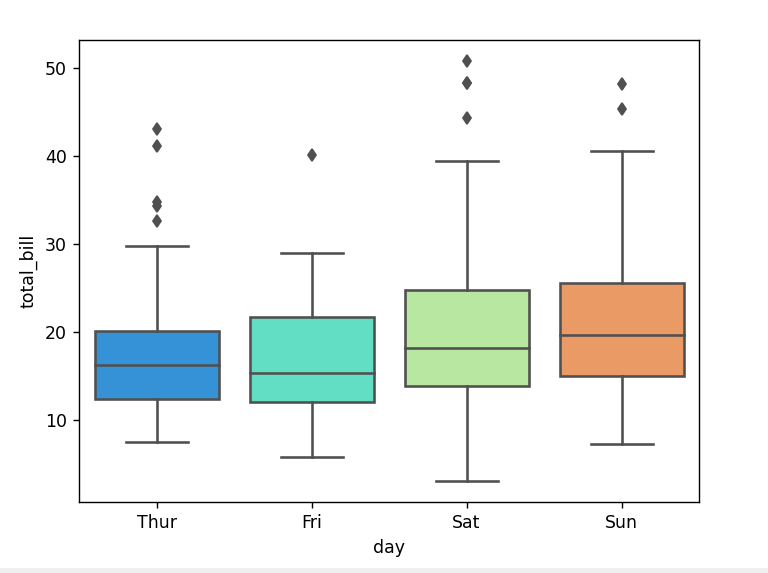
import numpy as np

tips = sns.load\_dataset('tips')

zort = sns.boxplot(x="day"**,** y="total\_bill"**,** data=tips**,**palette='rainbow')

plt.show()

print(zort)

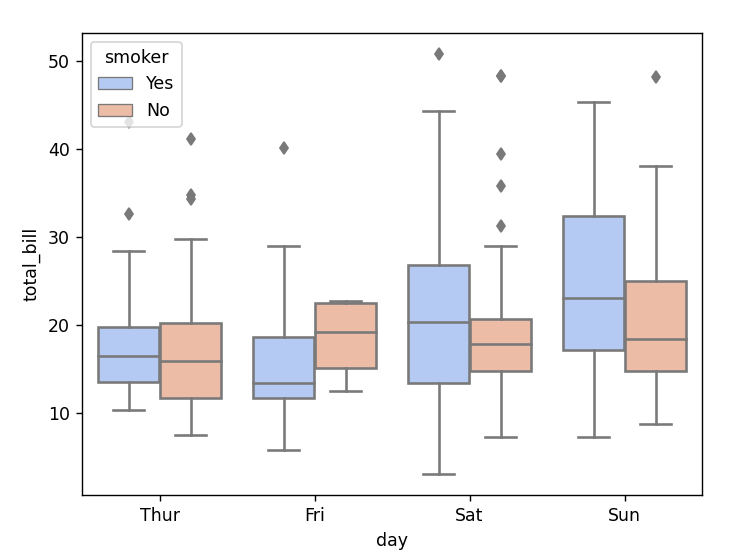


# Can do entire dataframe with orient='h'

zort = sns.boxplot(data=tips**,** palette='rainbow'**,** orient='h')



zort = sns.boxplot(x="day"**,** y="total\_bill"**,** hue="smoker"**,**data=tips**,** palette="coolwarm")



Violinplot

A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all of the plot components correspond to actual data points, the violin plot features a kernel density estimation of the underlying distribution.

import seaborn as sns

import matplotlib.pyplot as plt

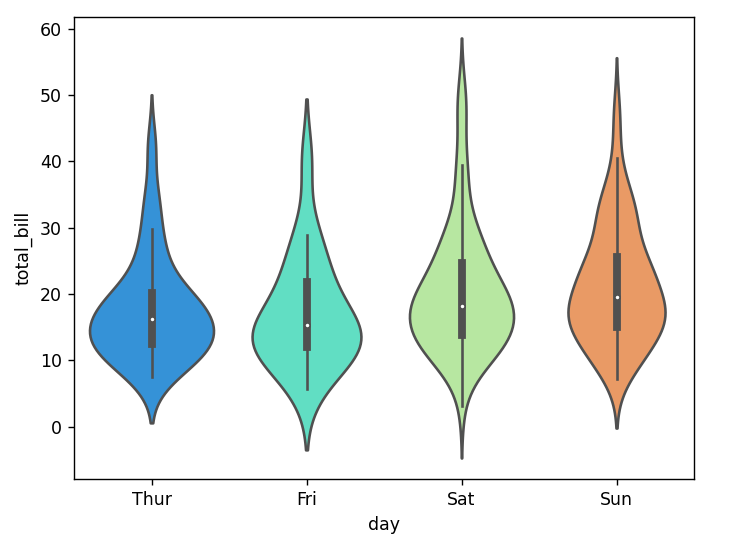
import numpy as np

tips = sns.load\_dataset('tips')

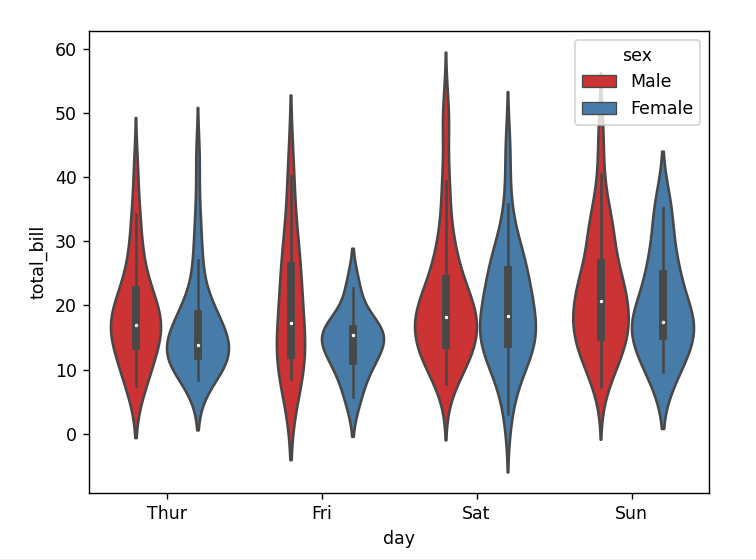
zort = sns.violinplot(x="day"**,** y="total\_bill"**,** data=tips**,**palette='rainbow')

plt.show()

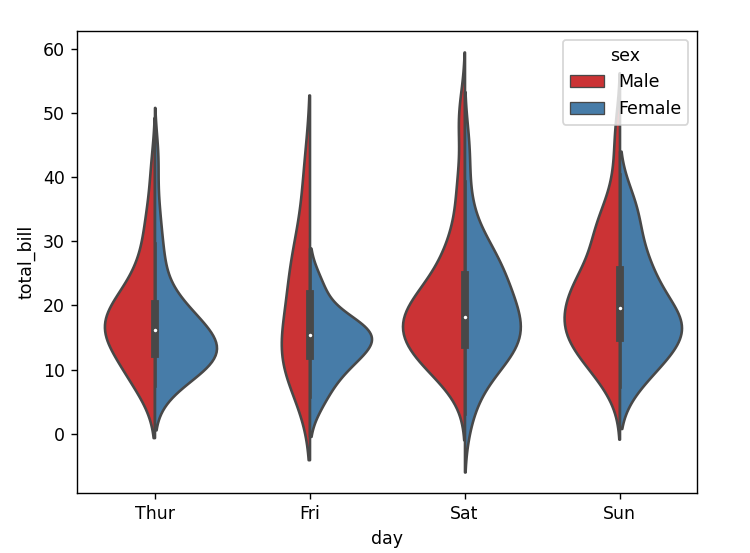
print(zort)



zort = sns.violinplot(x="day"**,** y="total\_bill"**,** data=tips**,**hue='sex'**,**palette='Set1')



zort = sns.violinplot(x="day"**,** y="total\_bill"**,** data=tips**,**hue='sex'**,**split=True**,**palette='Set1')



Stripplot and Swarmplot

The stripplot will draw a scatter plot where one variable is categorical. A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution. The swarmplot is similar to stripplot(), but the points are adjusted (only along the categorical axis) so that they don’t overlap. This gives a better representation of the distribution of values, although it does not scale as well to large numbers of observations (both in terms of the ability to show all the points and in terms of the computation needed to arrange them).

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

tips = sns.load\_dataset('tips')

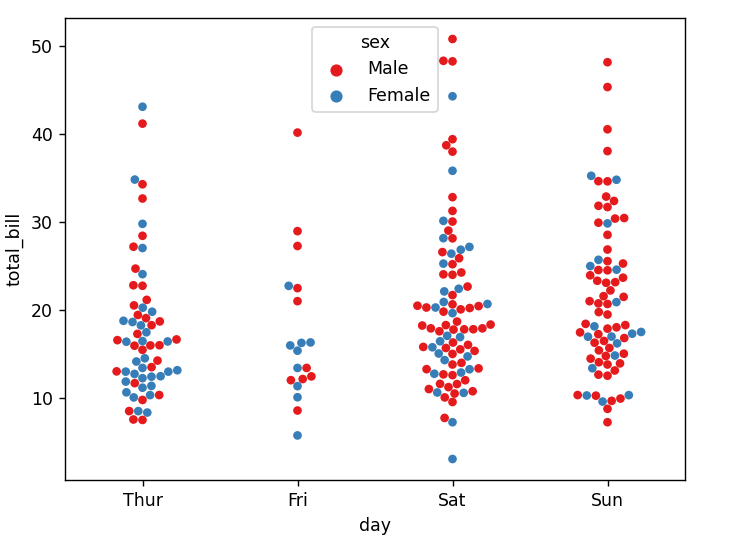
zort = sns.stripplot(x="day"**,** y="total\_bill"**,** data=tips**,**jitter=True**,**hue='sex'**,**palette='Set1')

plt.show()

print(zort)



zort = sns.swarmplot(x="day"**,** y="total\_bill"**,**hue='sex'**,**data=tips**,** palette="Set1")



Combining Categorical Plots

import seaborn as sns

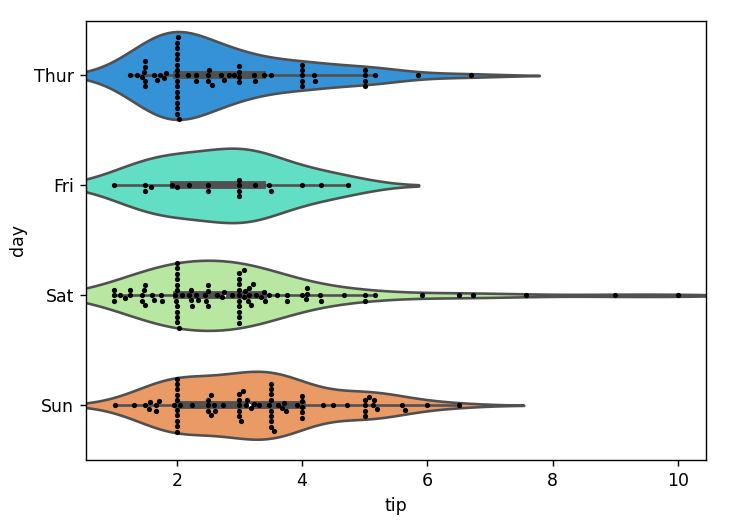
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

sns.violinplot(x="tip"**,** y="day"**,** data=tips**,**palette='rainbow')

sns.swarmplot(x="tip"**,** y="day"**,** data=tips**,**color='black'**,**size=**3**)

plt.show()



Factorplot

Factorplot is the most general form of a categorical plot. It can take in a \*\*kind\*\* parameter to adjust the plot type:

Matrix Plots

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

Heatmap

In order for a heatmap to work properly, your data should already be in a matrix form, the sns.heatmap function basically just colors it in for you. For example:

import seaborn as sns

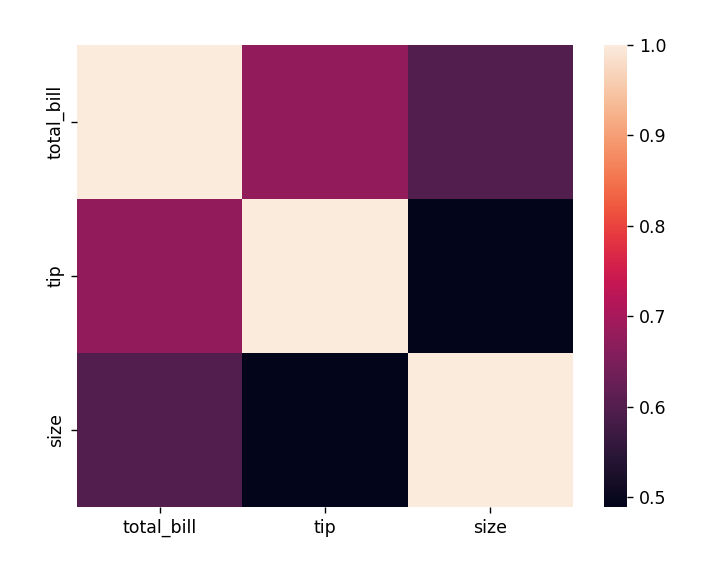
import matplotlib.pyplot as plt

flights = sns.load\_dataset('flights')

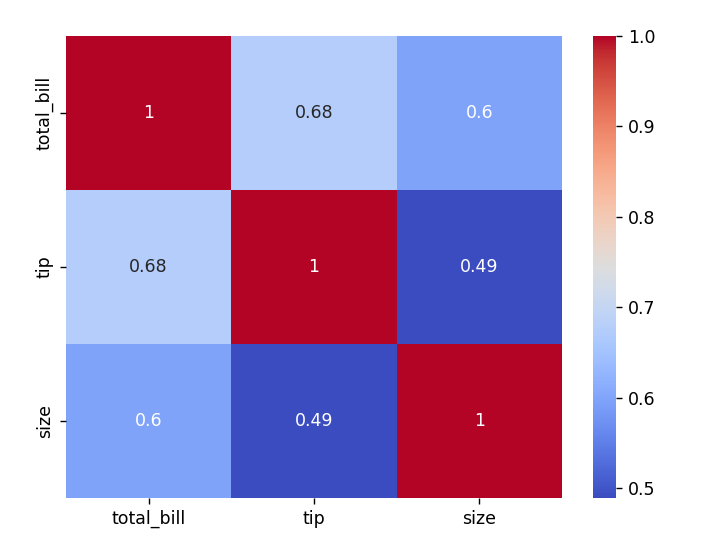
tips = sns.load\_dataset('tips')

sns.heatmap(tips.corr())

plt.show()



sns.heatmap(tips.corr()**,**cmap='coolwarm'**,**annot=True)



Or for the flights data:

import seaborn as sns

import matplotlib.pyplot as plt

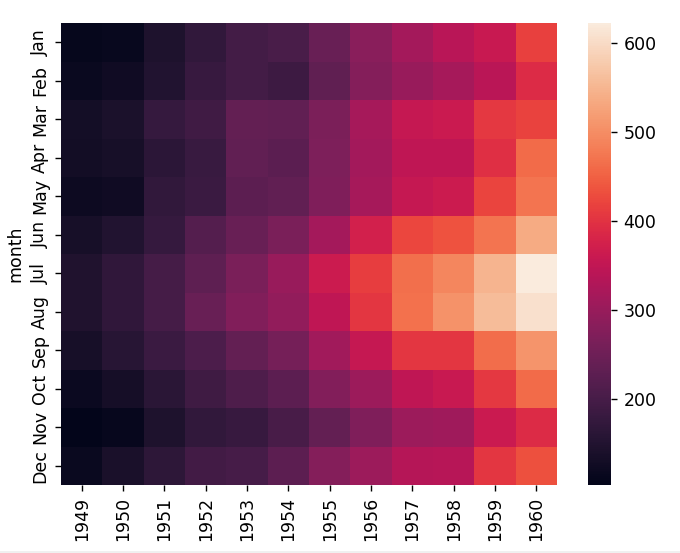
flights = sns.load\_dataset('flights')

tips = sns.load\_dataset('tips')

pvflights = flights.pivot\_table(values='passengers'**,**index='month'**,**columns='year')

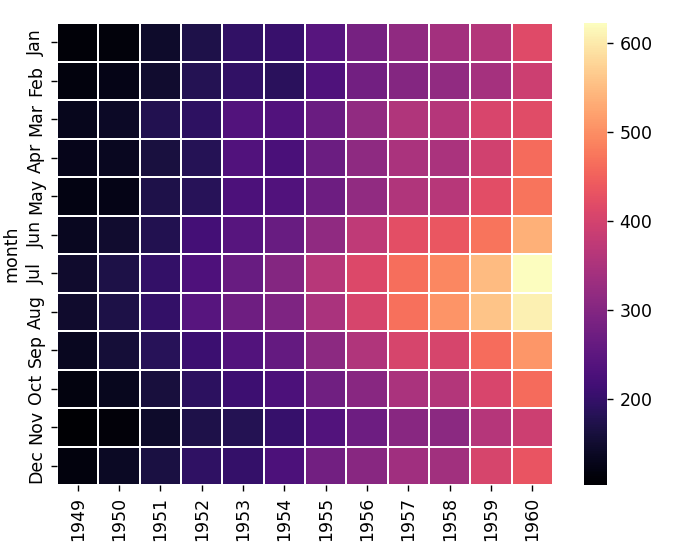
sns.heatmap(pvflights)

plt.show()



pvflights = flights.pivot\_table(values='passengers'**,**index='month'**,**columns='year')

sns.heatmap(pvflights**,**cmap='magma'**,**linecolor='white'**,**linewidths=**1**)



Clustermap

The clustermap uses hierarchical clustering to produce a clustered version of the heatmap. For example:

*import* seaborn *as* sns

*import* matplotlib.pyplot *as* plt

*import* scipy

flights = sns.load\_dataset('flights')

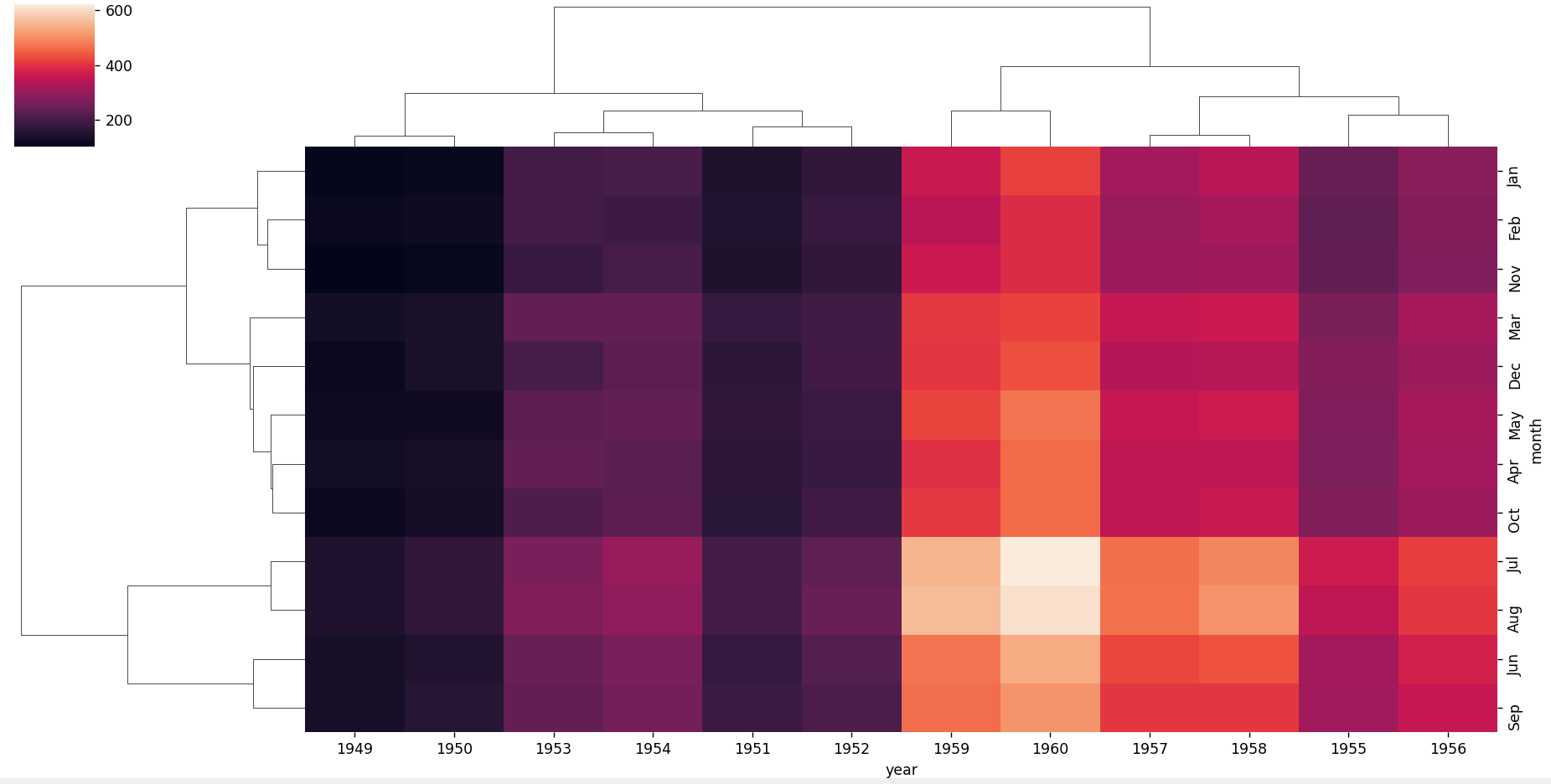
tips = sns.load\_dataset('tips')

pvflights = flights.pivot\_table(values='passengers',index='month',columns='year')

sns.heatmap(pvflights)

sns.clustermap(pvflights)

plt.show()



Grids

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

PairGrid

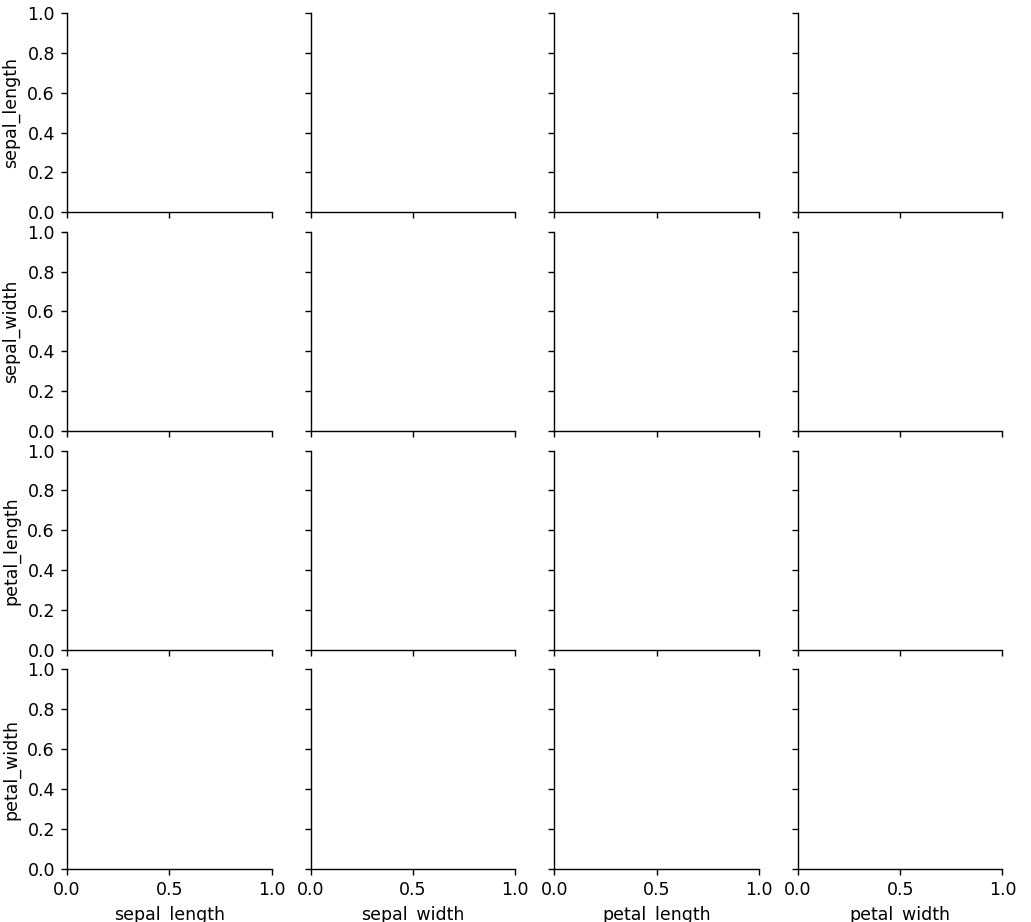
import seaborn as sns

import matplotlib.pyplot as plt

iris = sns.load\_dataset('iris')

sns.PairGrid(iris)

plt.show()



import seaborn as sns

import matplotlib.pyplot as plt

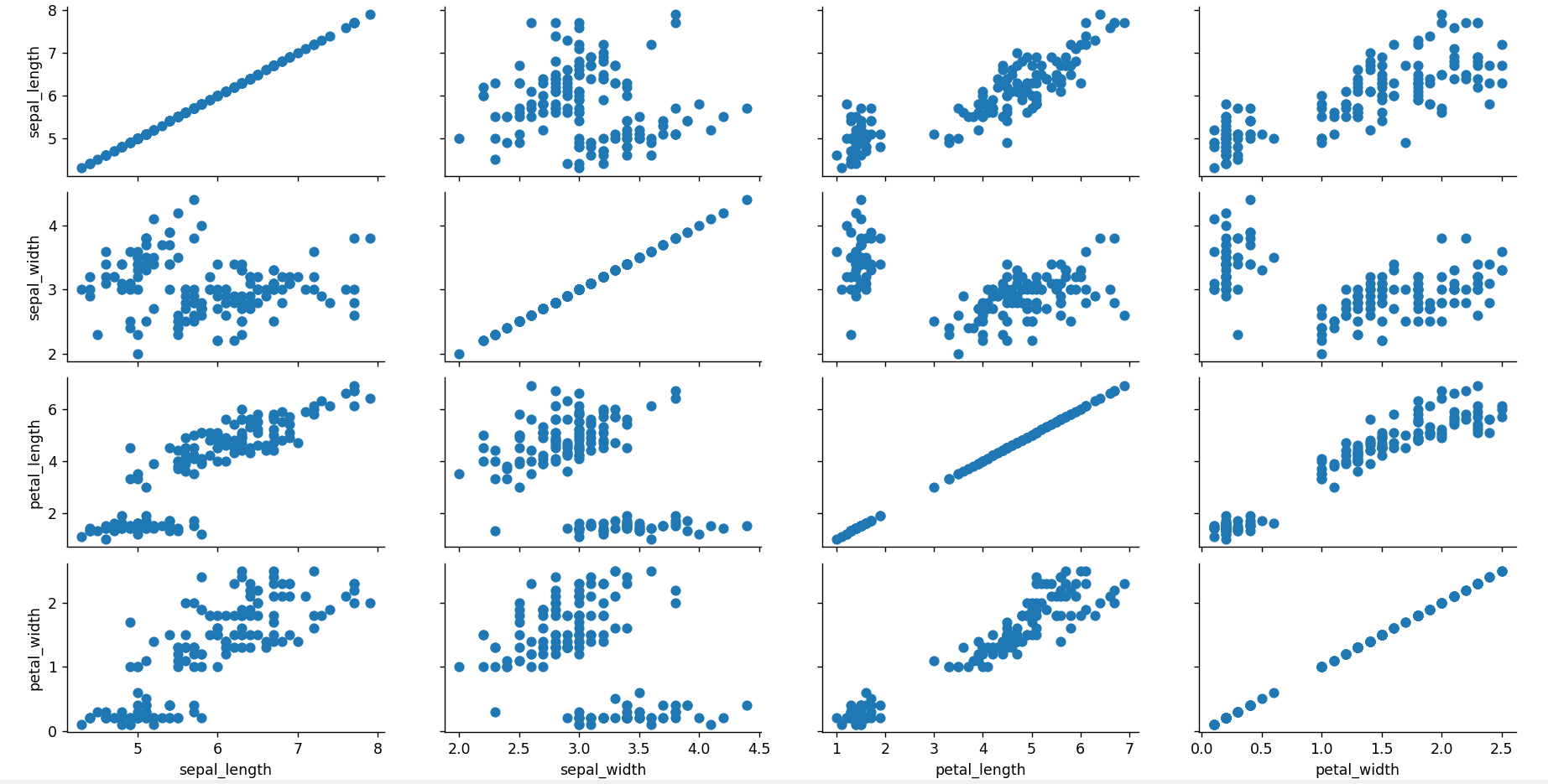
iris = sns.load\_dataset('iris')

# Then you map to the grid

g = sns.PairGrid(iris)

g.map(plt.scatter)

plt.show()



import seaborn as sns

import matplotlib.pyplot as plt

iris = sns.load\_dataset('iris')

# Map to upper,lower, and diagonal

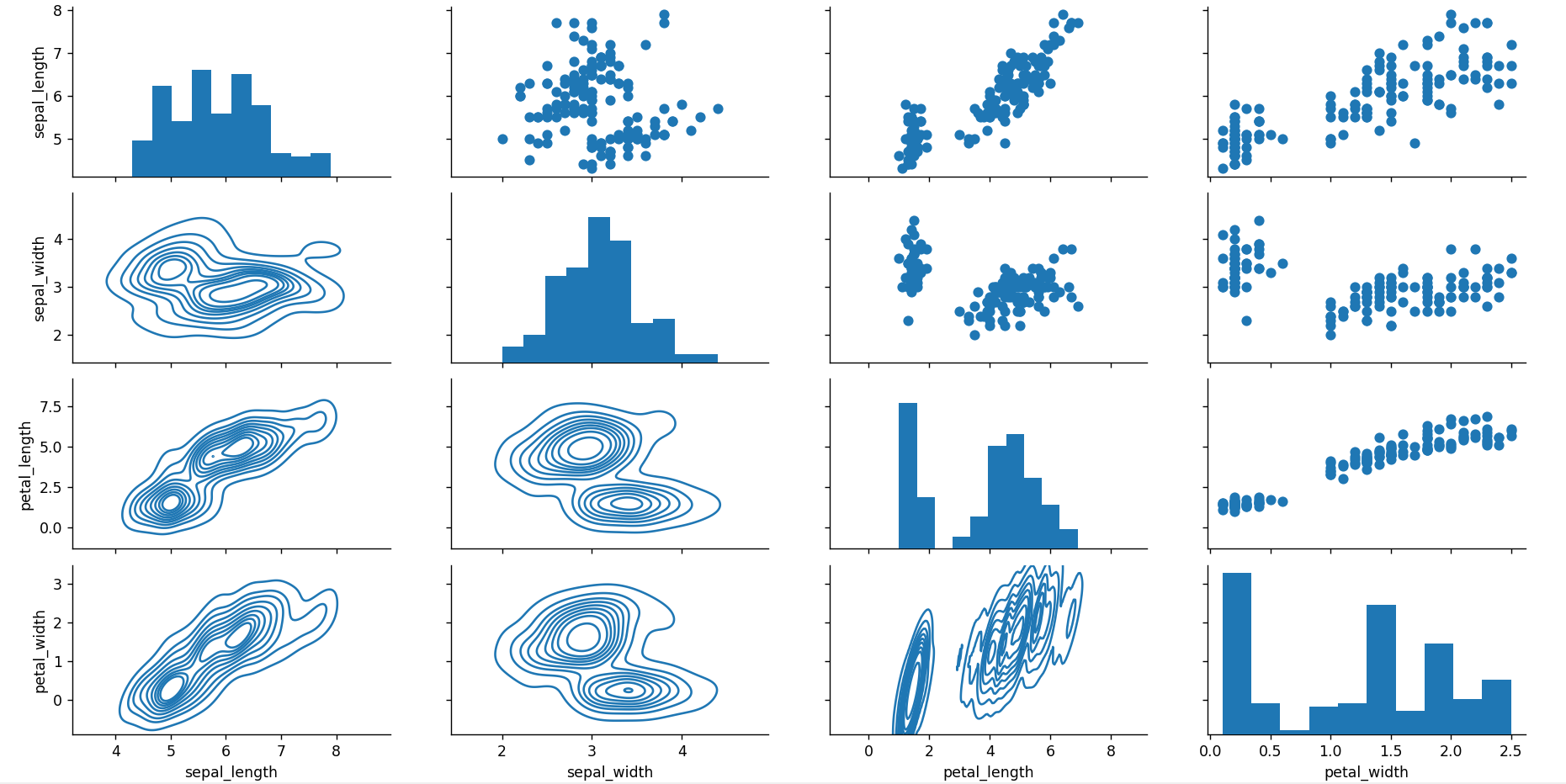
g = sns.PairGrid(iris)

g.map\_diag(plt.hist)

g.map\_upper(plt.scatter)

g.map\_lower(sns.kdeplot)

plt.show()



Pairplot

Pairplot is a simpler version of PairGrid (you'll use quite often)

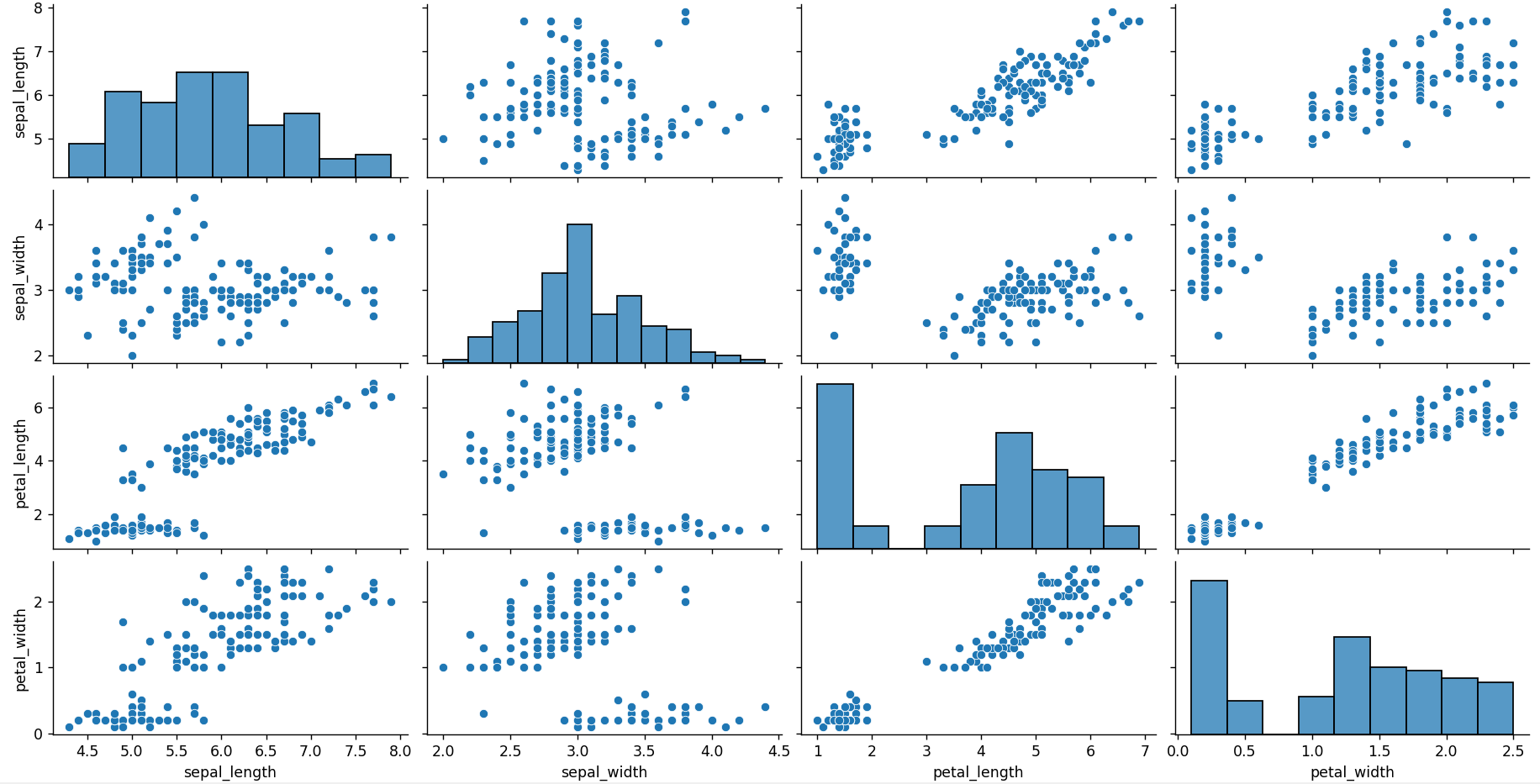
import seaborn as sns

import matplotlib.pyplot as plt

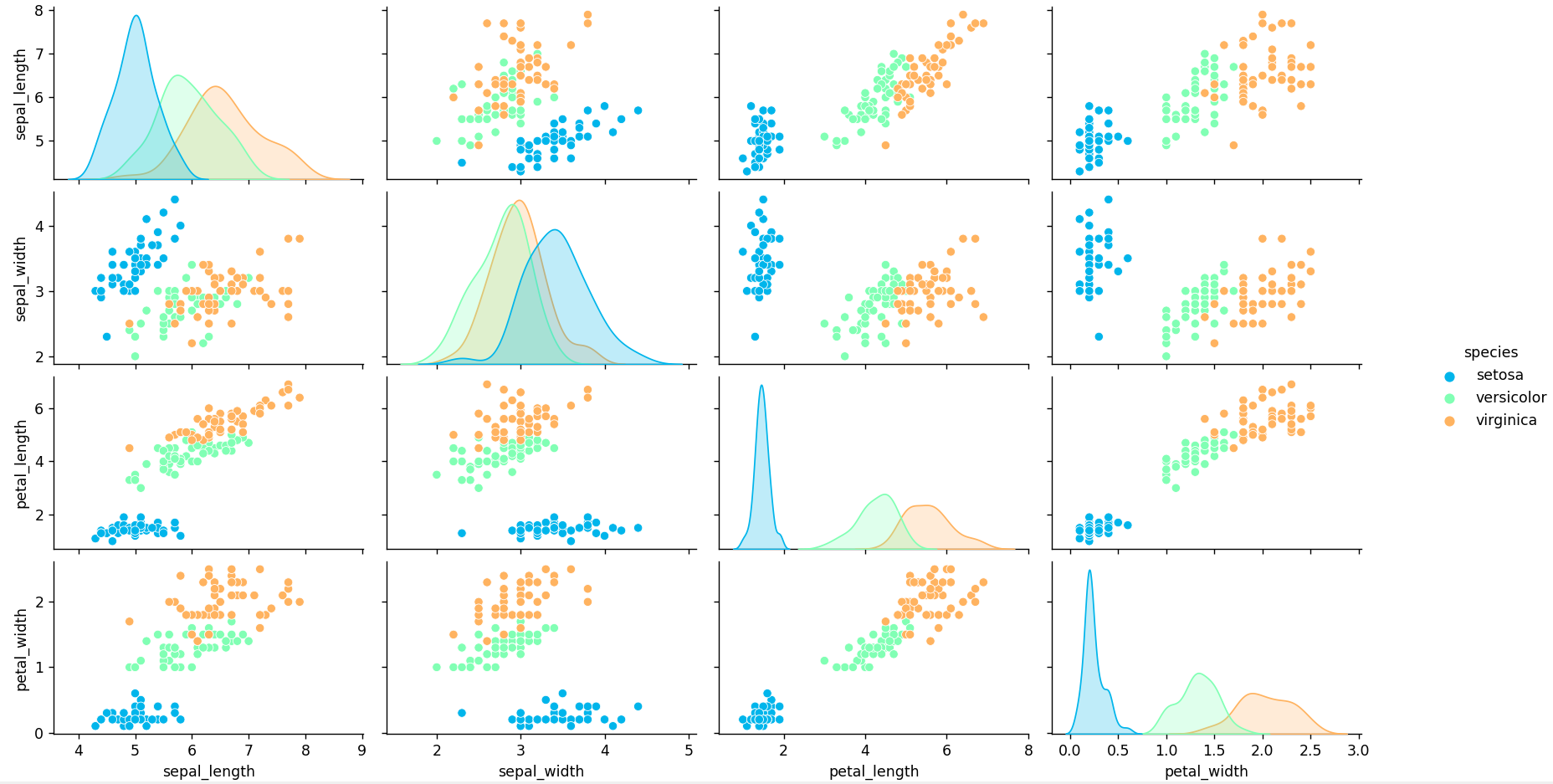
iris = sns.load\_dataset('iris')

sns.pairplot(iris)

plt.show()



sns.pairplot(iris**,**hue='species'**,**palette='rainbow')



Facet Grid

FacetGrid is the general way to create grids of plots based off of a feature:

import seaborn as sns

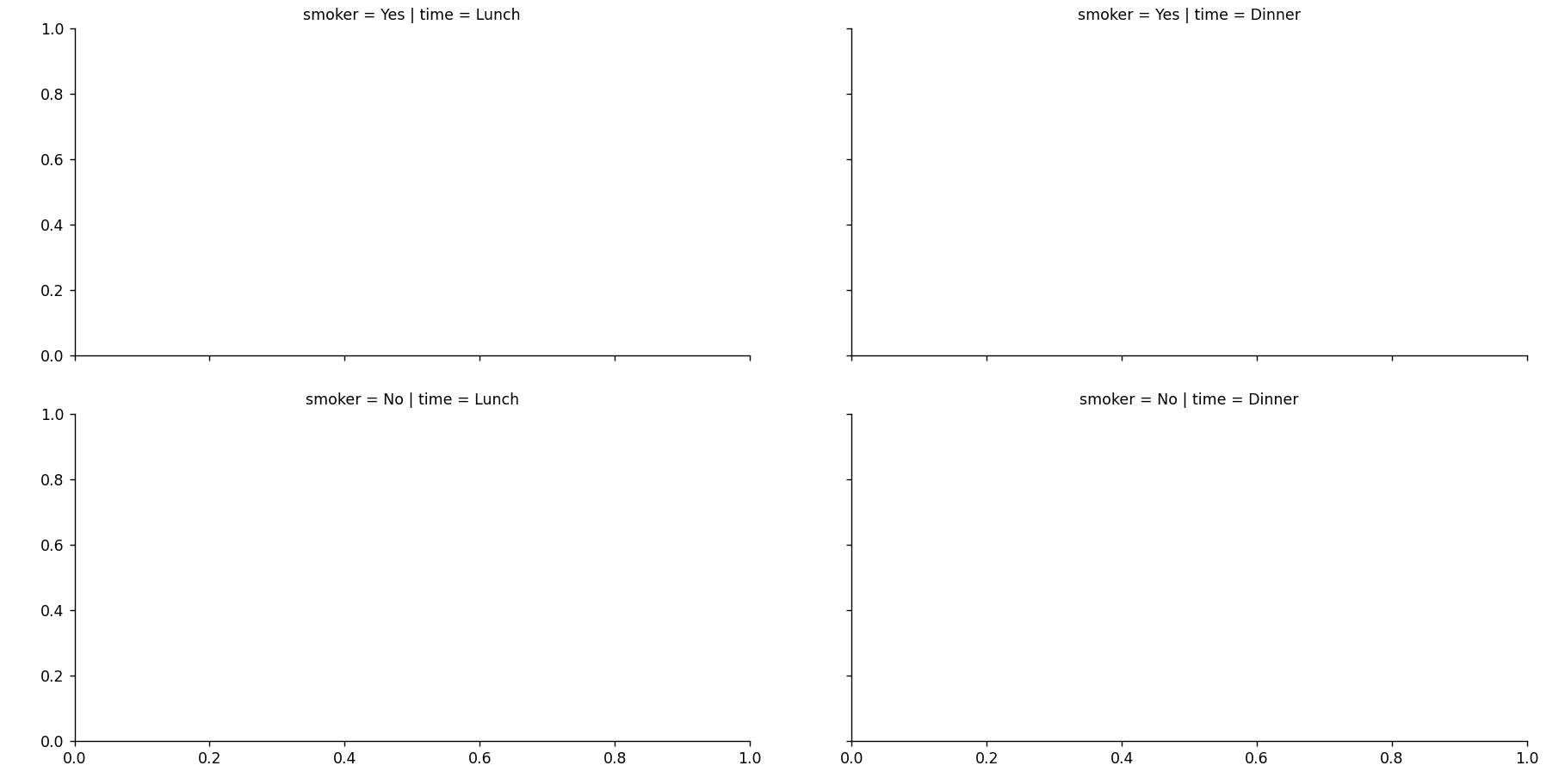
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

# Just the Grid

g = sns.FacetGrid(tips**,** col="time"**,** row="smoker")

plt.show()



import seaborn as sns

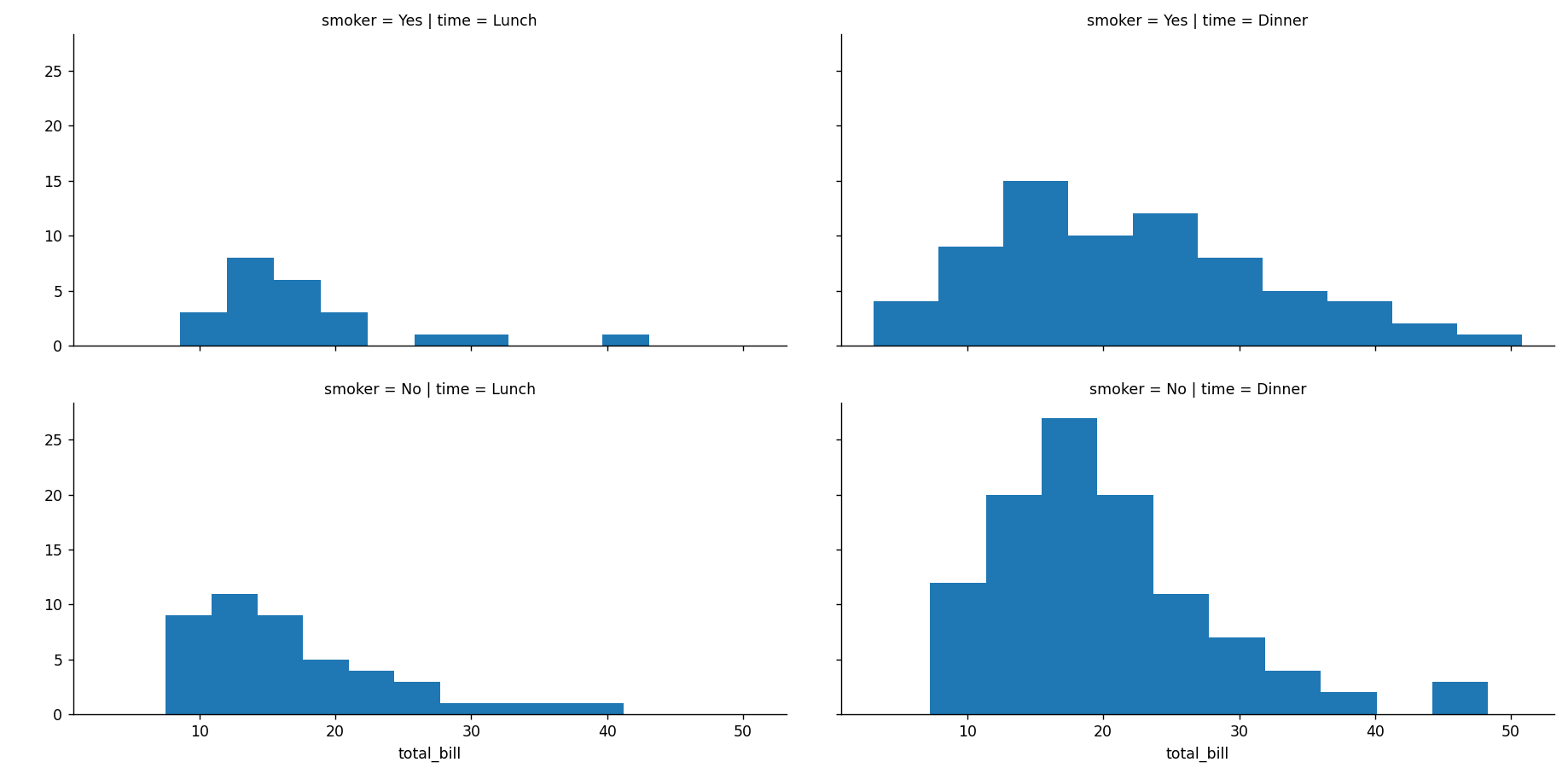
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

g = sns.FacetGrid(tips**,** col="sex")

g = g.map(plt.hist**,** "total\_bill")

plt.show()



import seaborn as sns

import matplotlib.pyplot as plt

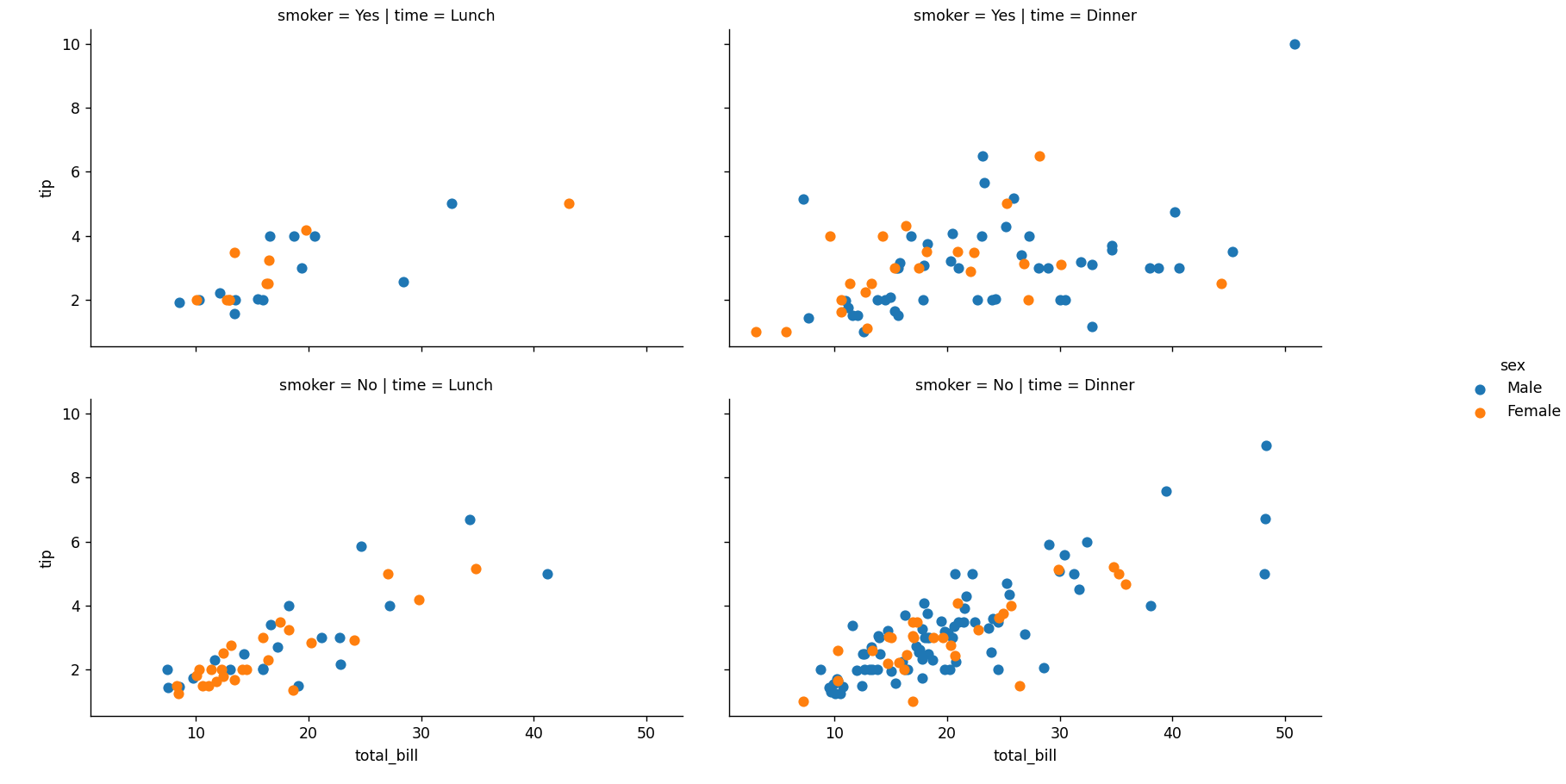
tips = sns.load\_dataset('tips')

g = sns.FacetGrid(tips**,** col="time"**,** row="smoker"**,**hue='sex')

# Notice how the arguments come after plt.scatter call

g = g.map(plt.scatter**,** "total\_bill"**,** "tip").add\_legend()

plt.show()



JointGrid

JointGrid is the general version for jointplot() type grids, for a quick example:

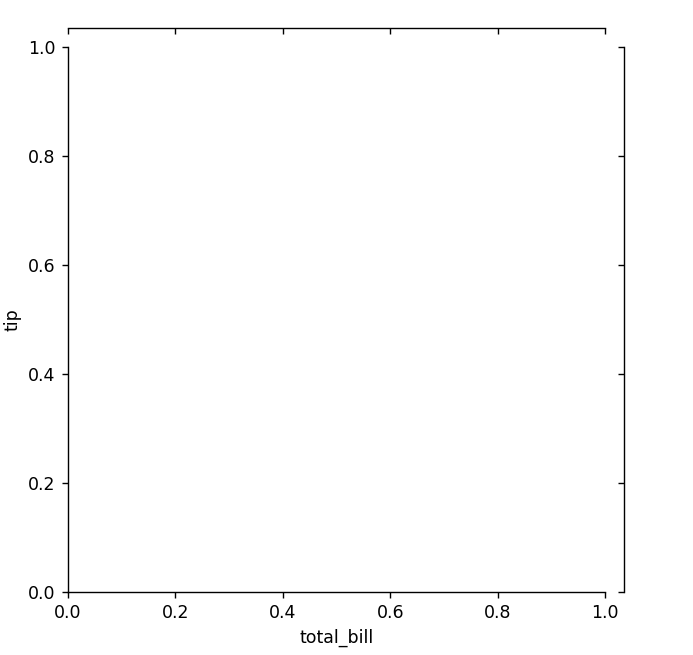
import seaborn as sns

import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

g = sns.JointGrid(x="total\_bill"**,** y="tip"**,** data=tips)

plt.show()



import seaborn as sns

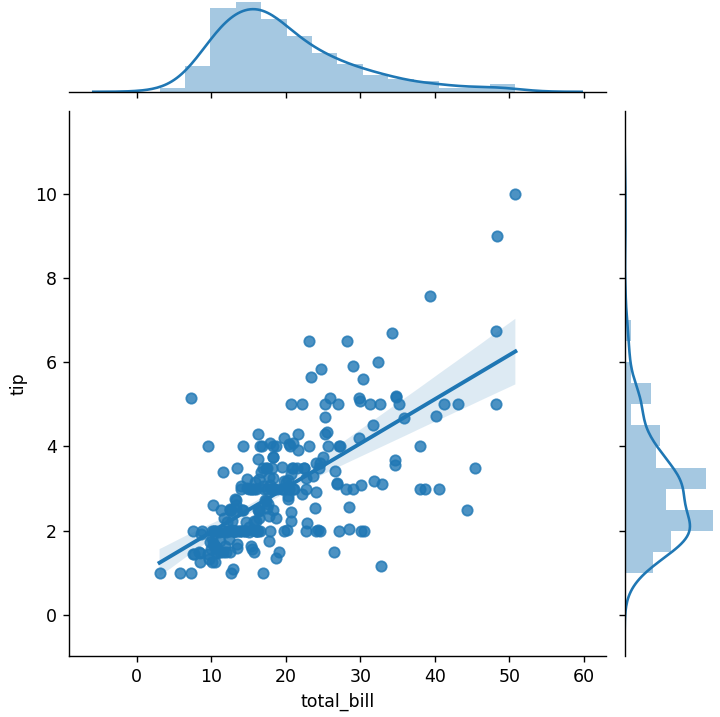
import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

g = sns.JointGrid(x="total\_bill"**,** y="tip"**,** data=tips)

g = g.plot(sns.regplot**,** sns.distplot)

plt.show()



Regression Plots

Seaborn has many built-in capabilities for regression plots, however we won't really discuss regression until the machine learning section of the course, so we will only cover the \*\*lmplot()\*\* function for now. \*\*lmplot\*\* allows you to display linear models, but it also conveniently allows you to split up those plots based off of features, as well as coloring the hue based off of features.

lmplot()

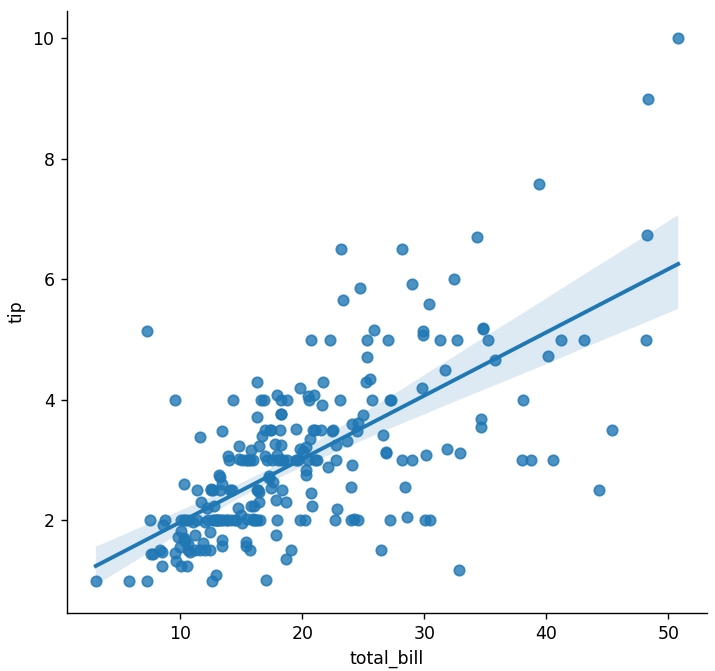
import seaborn as sns

import matplotlib.pyplot as plt

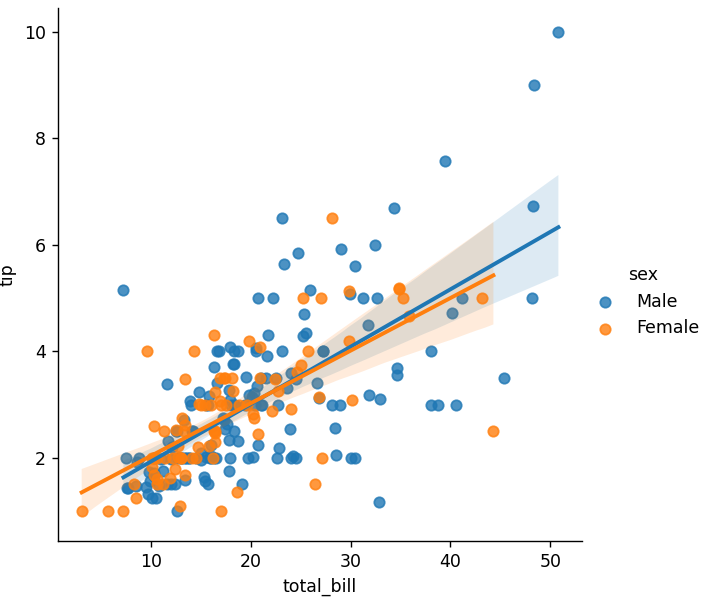
tips = sns.load\_dataset('tips')

sns.lmplot(x='total\_bill'**,**y='tip'**,**data=tips)

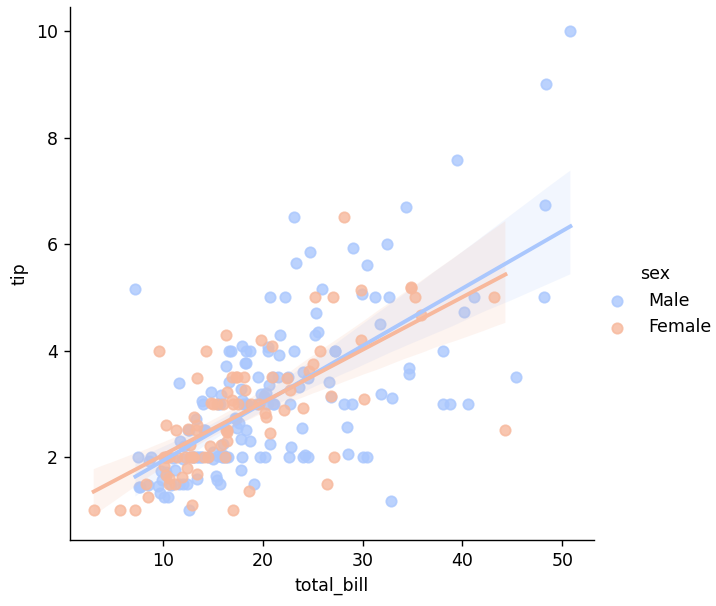
plt.show()



sns.lmplot(x='total\_bill'**,**y='tip'**,**data=tips**,**hue='sex')



sns.lmplot(x='total\_bill'**,**y='tip'**,**data=tips**,**hue='sex'**,**palette='coolwarm')



Working with Markers

lmplot kwargs get passed through to \*\*regplot\*\* which is a more general form of lmplot(). regplot has a scatter\_kws parameter that gets passed to plt.scatter. So you want to set the s parameter in that dictionary, which corresponds (a bit confusingly) to the squared markersize. In other words you end up passing a dictionary with the base matplotlib arguments, in this case, for the size of a scatter plot. In general, you probably won't remember this off the top of your head, but instead reference the documentation.

import seaborn as sns

import matplotlib.pyplot as plt

tips = sns.load\_dataset('tips')

sns.lmplot(x='total\_bill'**,**y='tip'**,**data=tips**,**hue='sex'**,**palette='coolwarm'**,** markers=['o'**,**'v']**,**scatter\_kws={'s':**100**})

plt.show()



Using a Grid

We can add more variable separation through columns and rows with the use of a grid. Just indicate this with the col or row arguments:

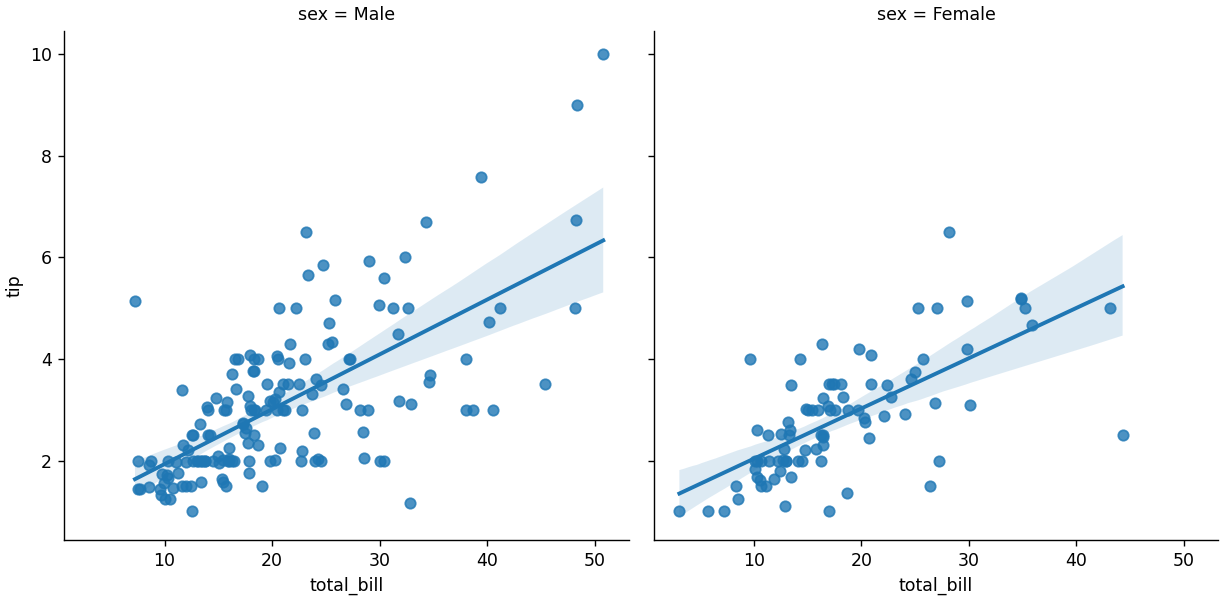
import seaborn as sns

import matplotlib.pyplot as plt

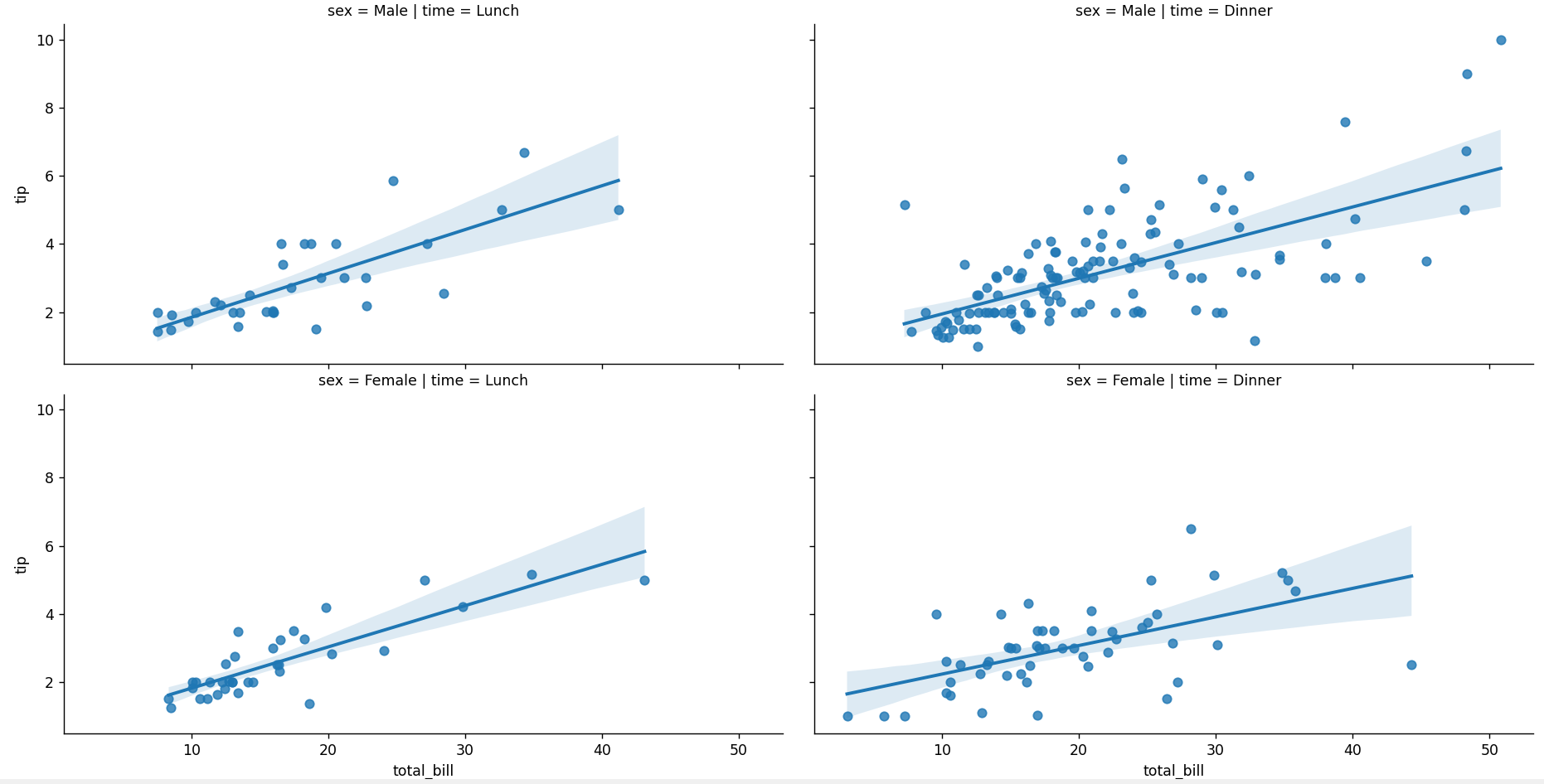
tips = sns.load\_dataset('tips')

sns.lmplot(x='total\_bill'**,**y='tip'**,**data=tips**,**col='sex')

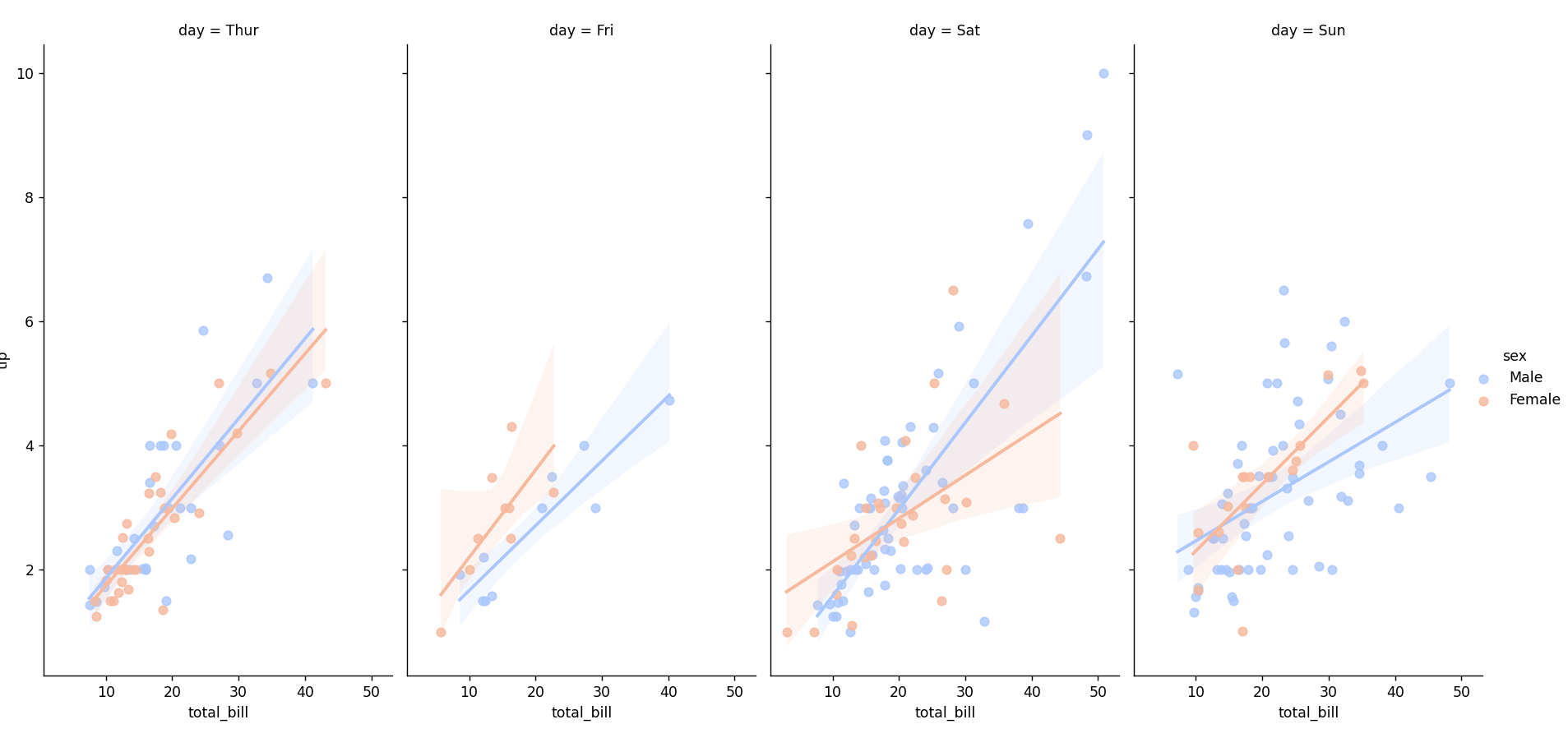
plt.show()



sns.lmplot(x="total\_bill"**,** y="tip"**,** row="sex"**,** col="time"**,**data=tips)



sns.lmplot(x='total\_bill'**,**y='tip'**,**data=tips**,**col='day'**,**hue='sex'**,**palette='coolwarm')



Aspect and Size

Seaborn figures can have their size and aspect ratio adjusted with the \*\*size\*\* and \*\*aspect\*\* parameters:

.

.

.

Style

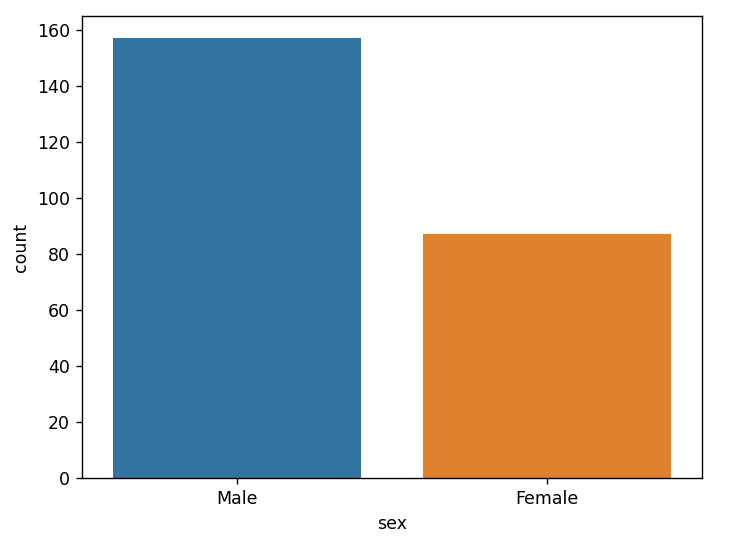
import seaborn as sns

import matplotlib.pyplot as plt

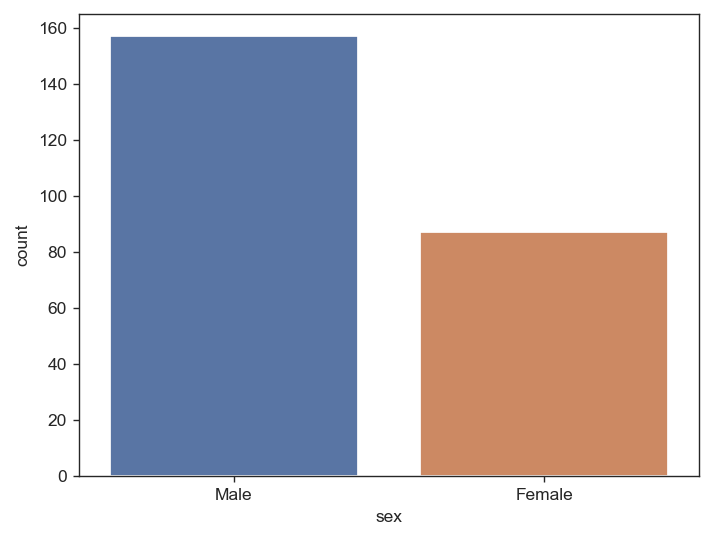
tips = sns.load\_dataset('tips')

sns.countplot(x='sex'**,**data=tips)

plt.show()



sns.countplot(x='sex'**,**data=tips**,**palette='deep')



Spine Removal

sns.despine(left=True , right=True)

.

.

.

Pandas Built-in Data Visualization

import matplotlib.pyplot as plt

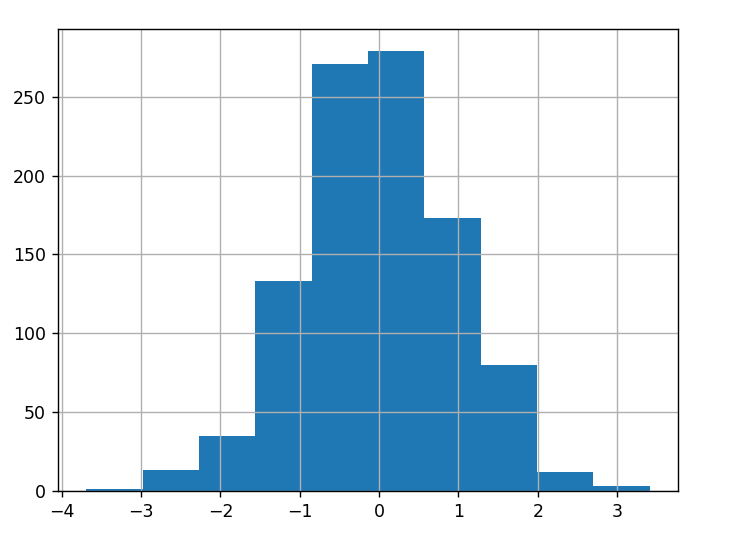
import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

df2 = pd.read\_csv('df2')

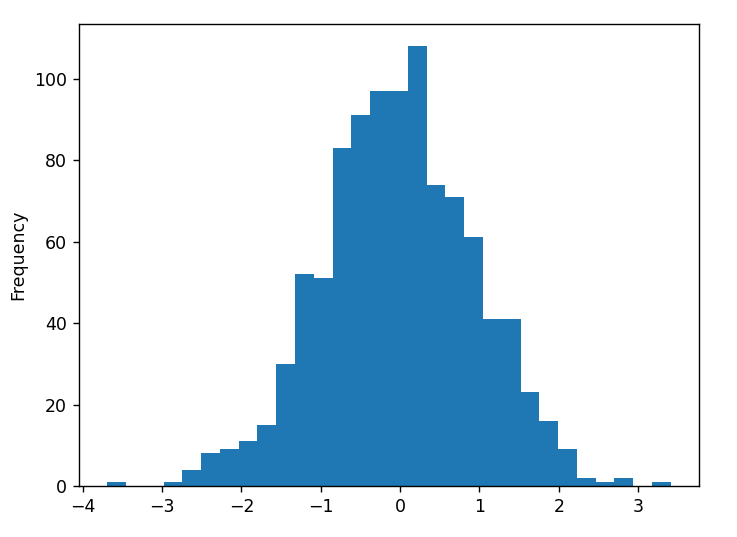
df1["A"].hist()

plt.show()



df1["A"].plot(kind="hist"**,**bins=**30**)

plt.show()



df.plot.area()

import matplotlib.pyplot as plt

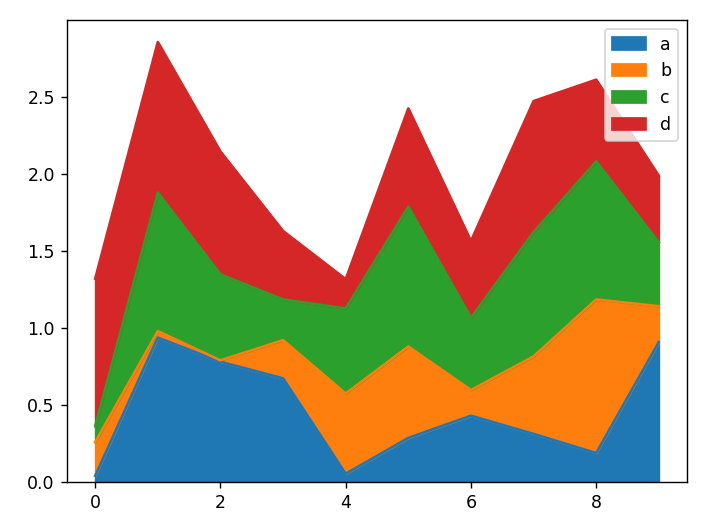
import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

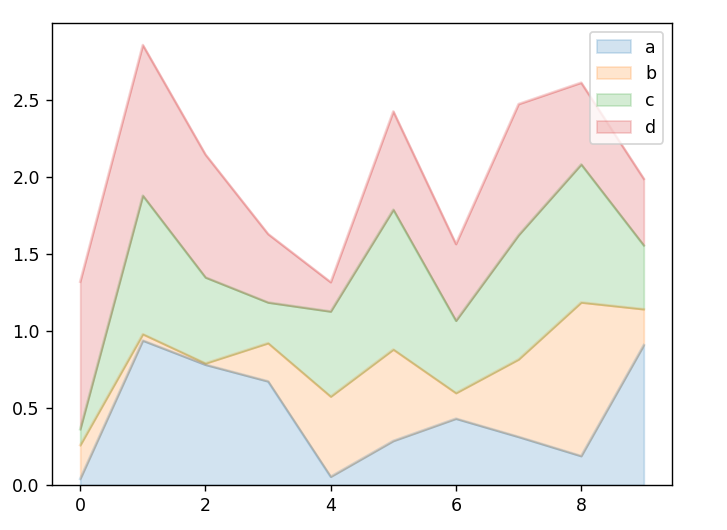
df2 = pd.read\_csv('df2')

df2.plot.area()

plt.show()



df2.plot.area(alpha=**0.2**)



df.plot.bar()

import matplotlib.pyplot as plt

import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

df2 = pd.read\_csv('df2')

df2.plot.bar()

plt.show()



df.plot.line(x= , y= )

import matplotlib.pyplot as plt

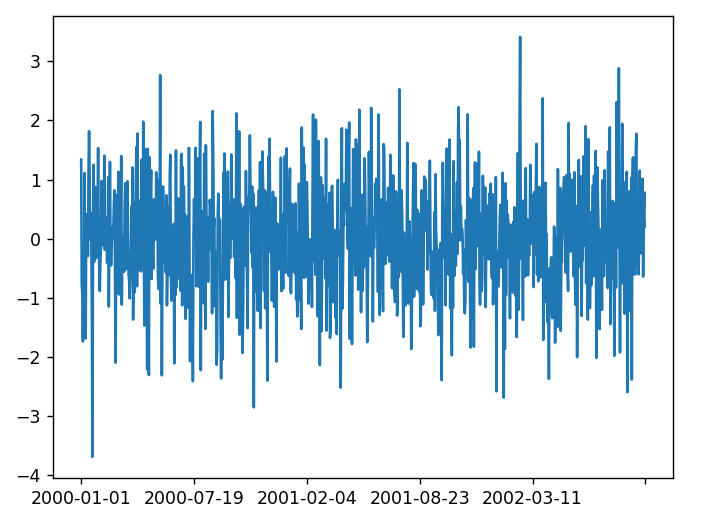
import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

df2 = pd.read\_csv('df2')

df1["A"].plot.line(x=df1.index**,** y="B")

plt.show()



df.plot.scatter()

import matplotlib.pyplot as plt

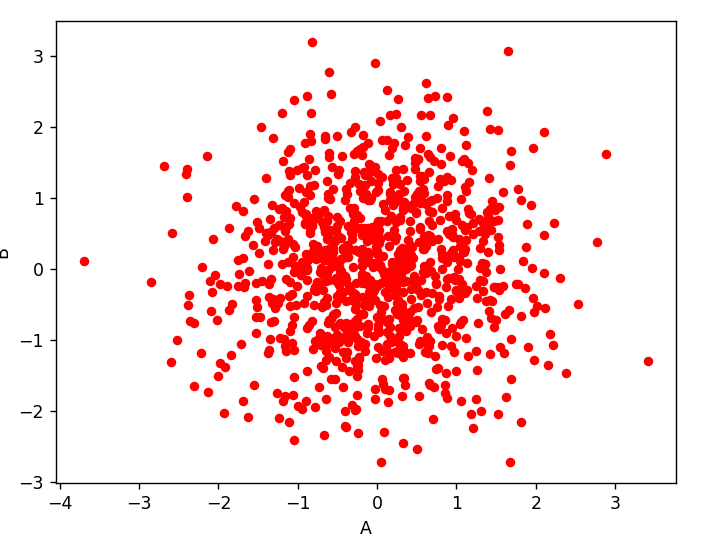
import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

df2 = pd.read\_csv('df2')

df1.plot.scatter(x="A"**,** y="B"**,**color="red")

plt.show()



df.plot.box()

import matplotlib.pyplot as plt

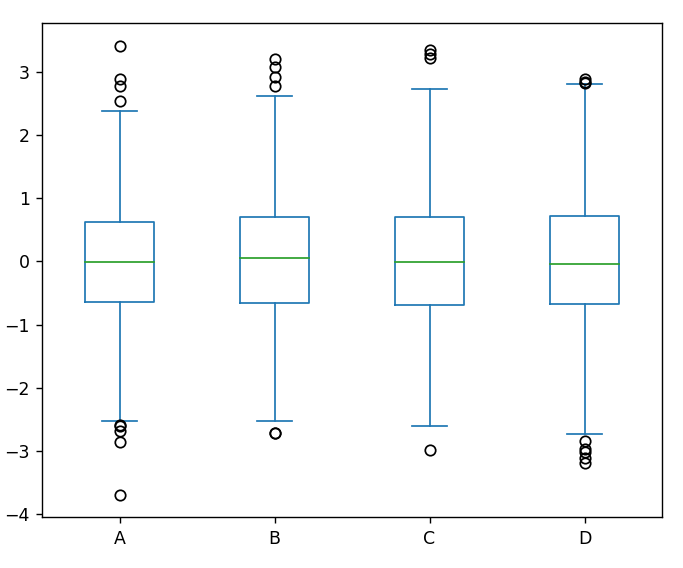
import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

df2 = pd.read\_csv('df2')

df1.plot.box()

plt.show()



import matplotlib.pyplot as plt

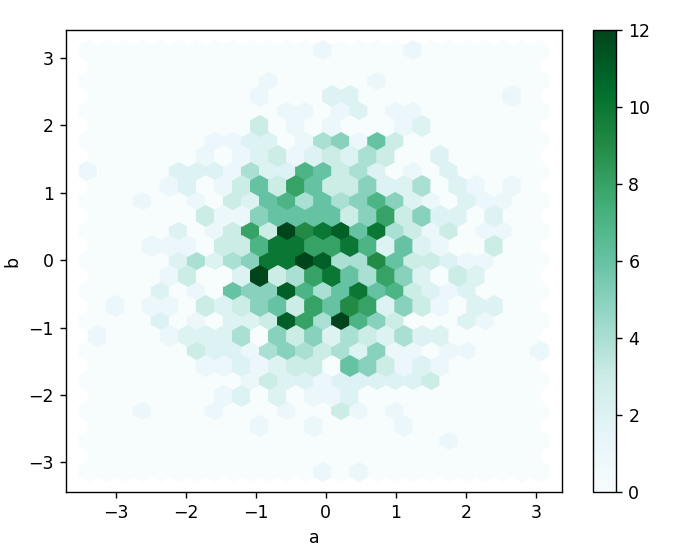
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(**1000, 2**)**,** columns=["a"**,** "b"])

df.plot.hexbin(x="a"**,** y="b"**,** gridsize=**25**)

plt.show()



density()

import matplotlib.pyplot as plt

import pandas as pd

df1 = pd.read\_csv('df1'**,** index\_col=**0**)

df2 = pd.read\_csv('df2')

df2.plot.density()

plt.show()

Plotly and Cufflinks

Plotly is a library that allows you to create interactive plots that you can use in dashboards or websites (you can save them as html files or static images).

Installation

import pandas as pd

import numpy as np

import cufflinks as cf

import plotly as py

df = pd.DataFrame(np.random.randn(**100, 4**)**,** columns="A B C D".split())

df2 = pd.DataFrame({"Category": ["A"**,** "B"**,** "C"]**,** "Values": [**32, 43, 50**]})

fig = df.iplot(kind='bar'**,** barmode='stack'**,** asFigure=True)

py.offline.plot(fig)

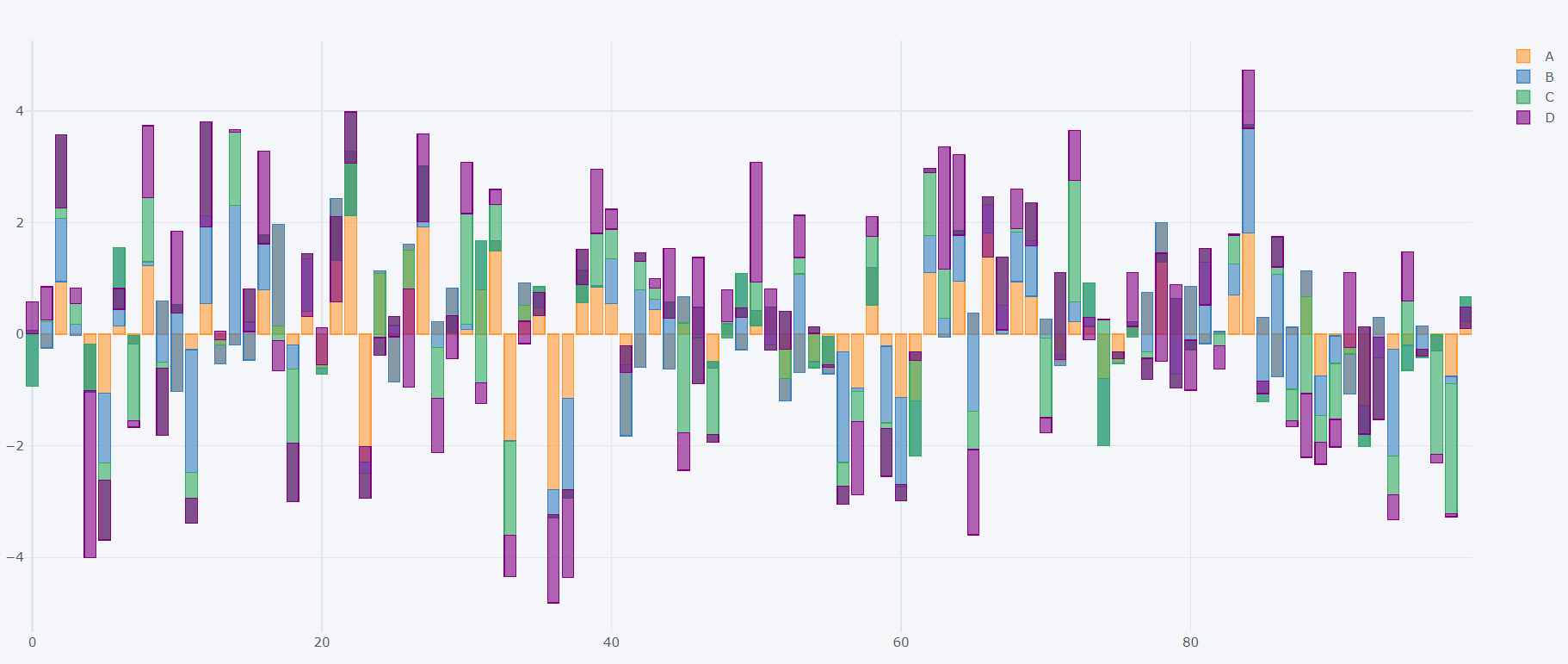


fig = df.iplot(kind='scatter'**,** x="A"**,** y="B"**,** mode="markers"**,** asFigure=True)

py.offline.plot(fig)

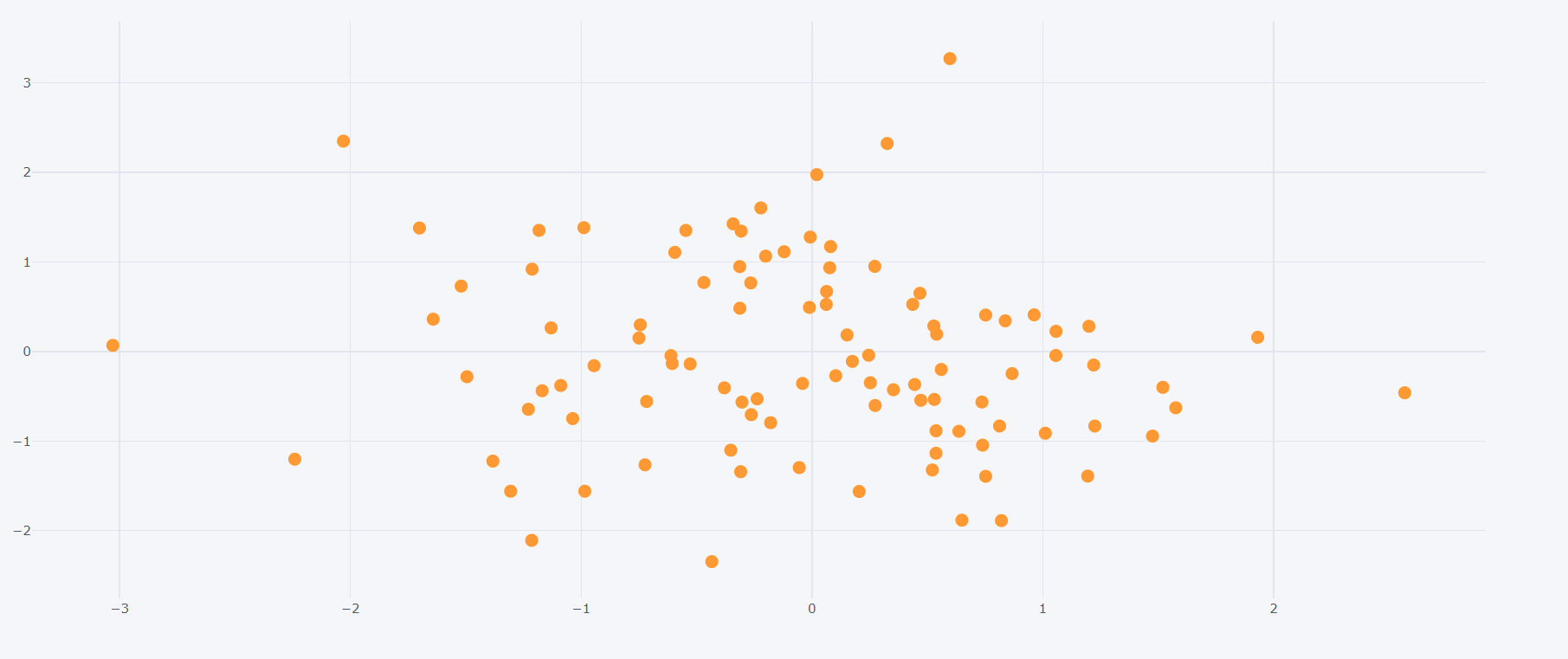
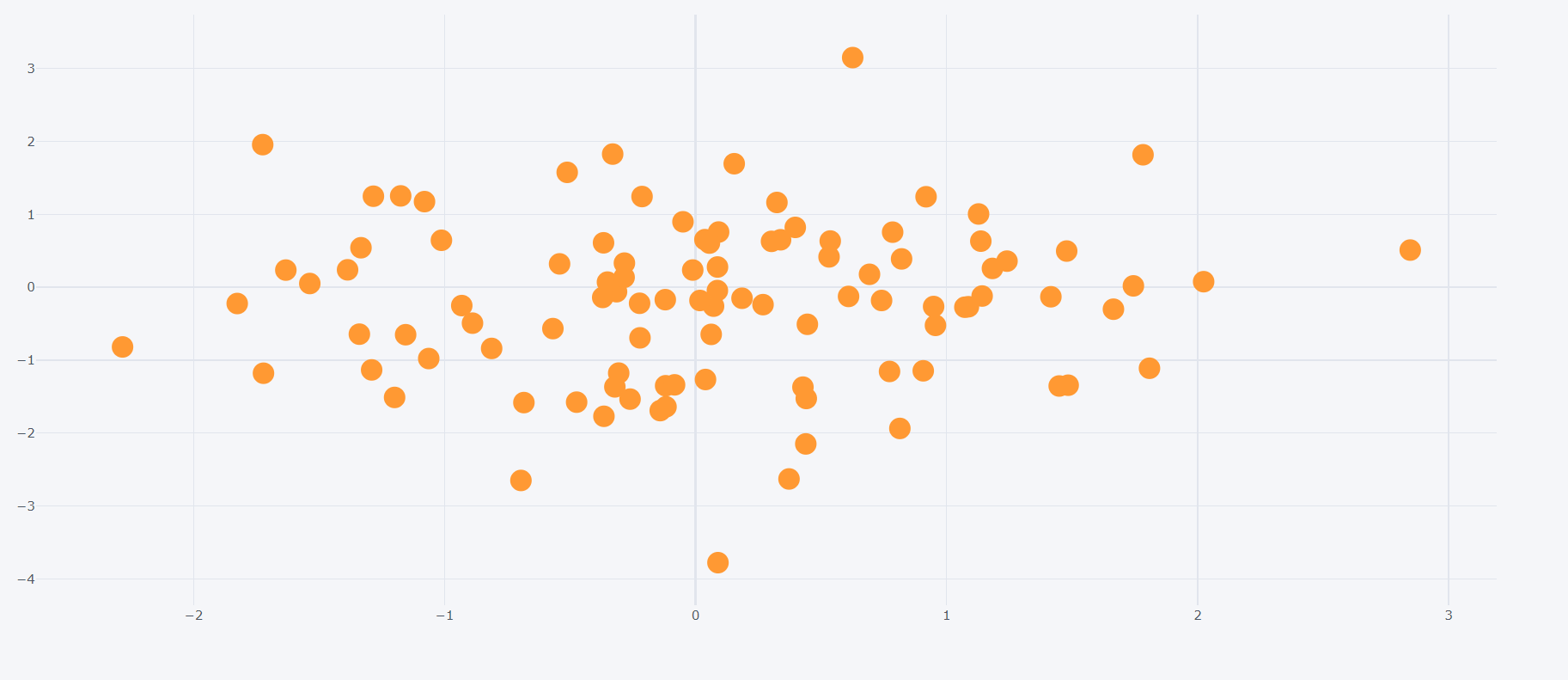


fig = df.iplot(kind='scatter'**,** x="A"**,** y="B"**,** mode="markers"**,** size=**20,** asFigure=True)

py.offline.plot(fig)



import pandas as pd

import numpy as np

import cufflinks as cf

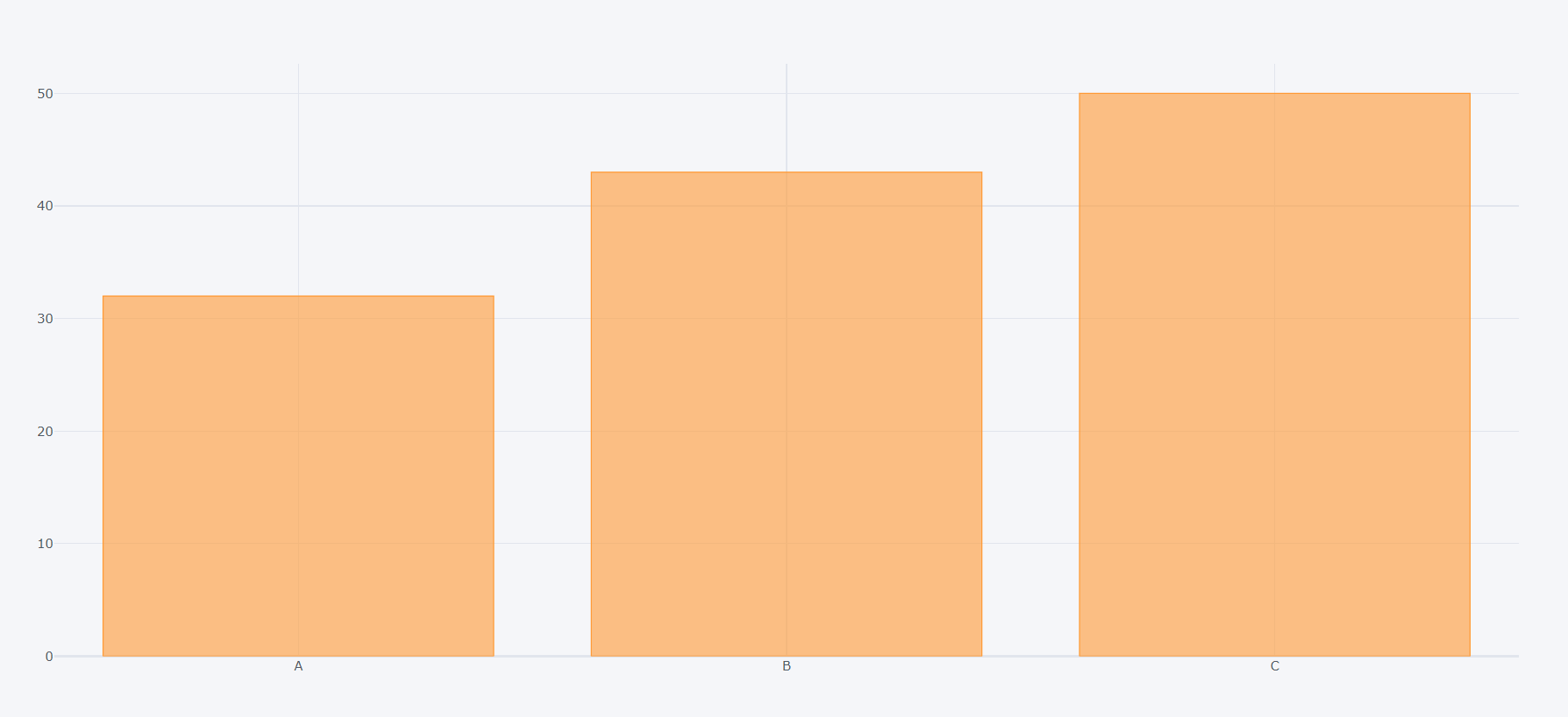
import plotly as py

df = pd.DataFrame(np.random.randn(**100, 4**)**,** columns="A B C D".split())

df2 = pd.DataFrame({"Category": ["A"**,** "B"**,** "C"]**,** "Values": [**32, 43, 50**]})

fig = df2.iplot(kind='bar'**,** x="Category"**,** y="Values"**,** asFigure=True)

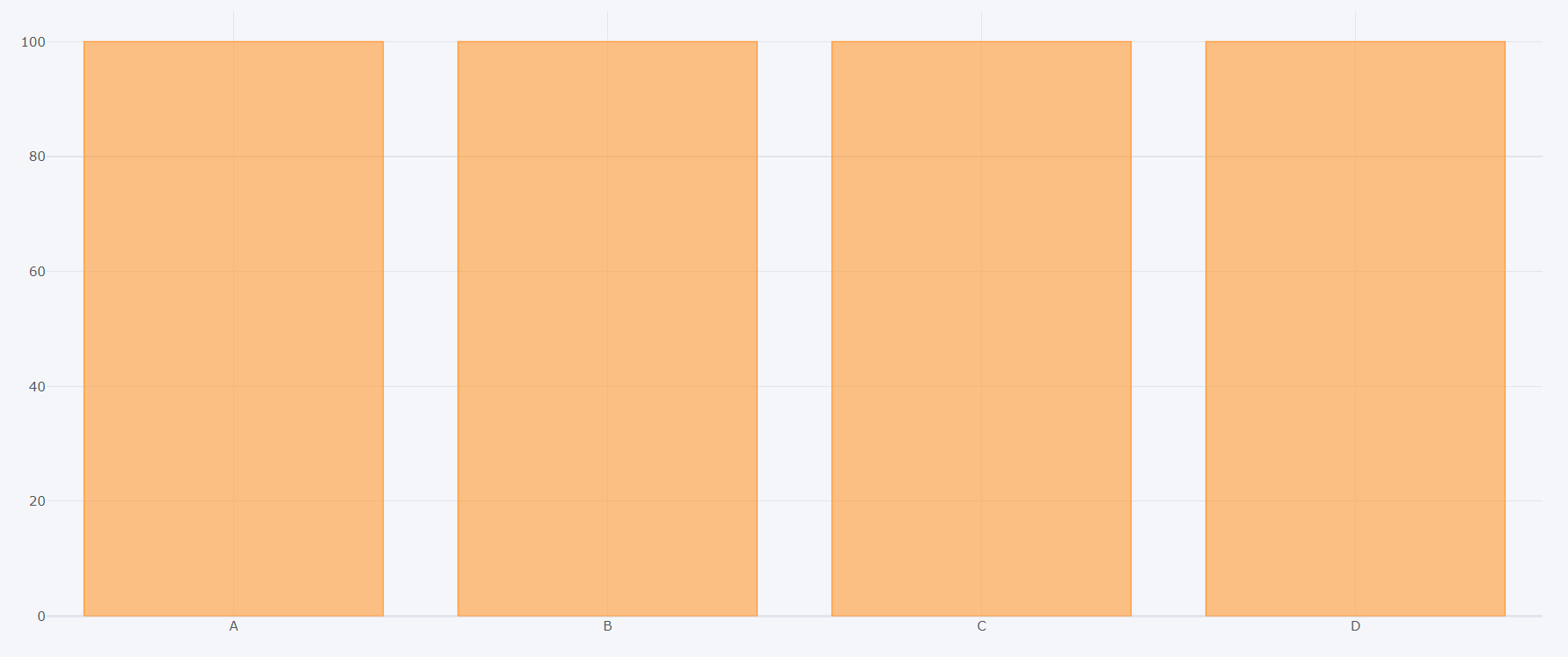
py.offline.plot(fig)



Count

fig = df.count().iplot(kind='bar'**,** asFigure=True)

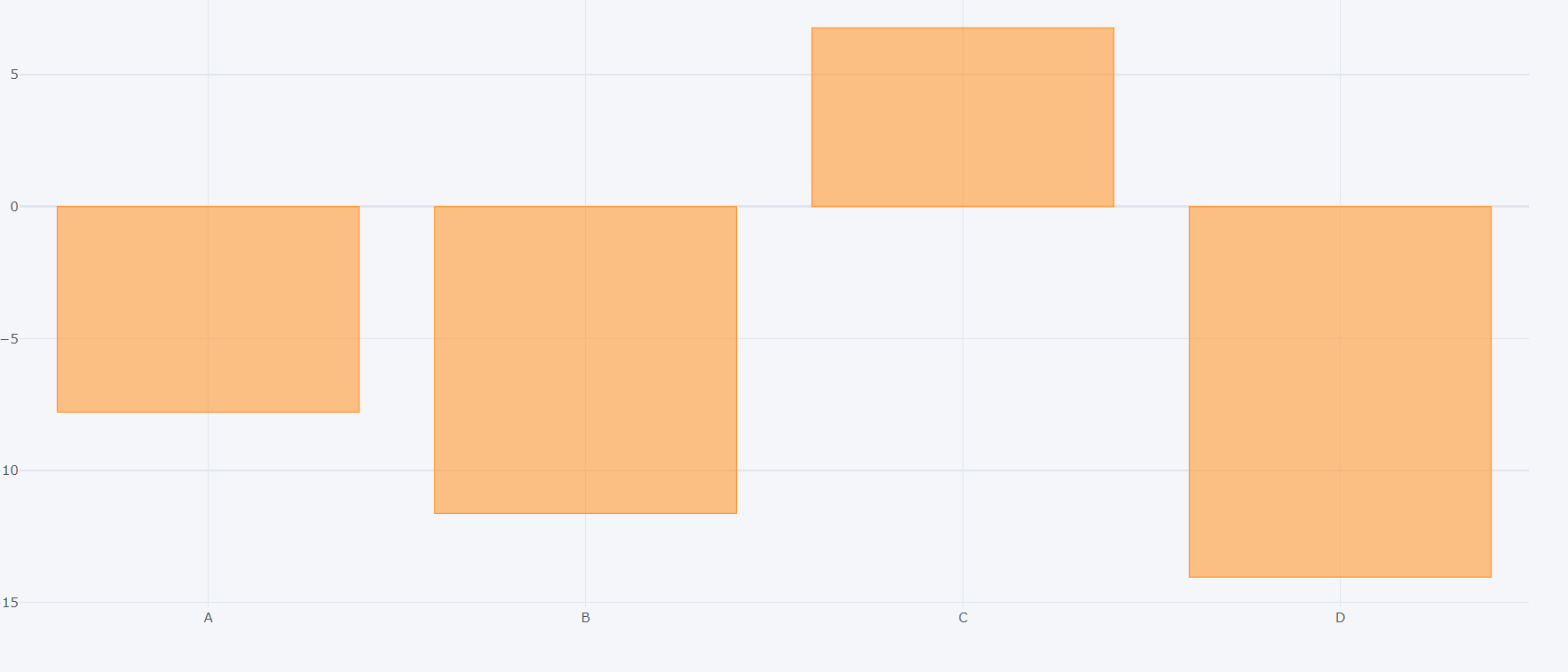
py.offline.plot(fig)



Sum

fig = df.sum().iplot(kind='bar'**,** asFigure=True)

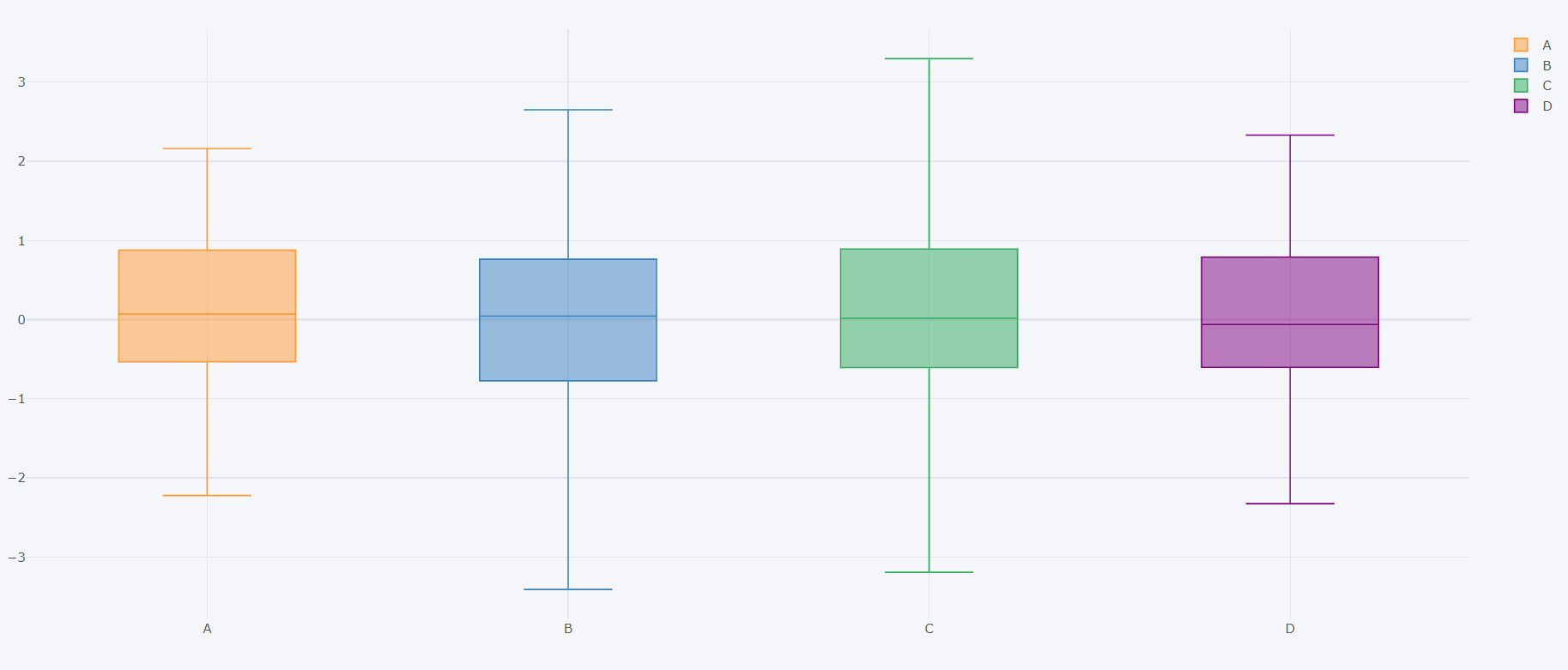
py.offline.plot(fig)



Box

fig = df.iplot(kind='box'**,** asFigure=True)

py.offline.plot(fig)



3 Dimension

import pandas as pd

import numpy as np

import cufflinks as cf

import plotly as py

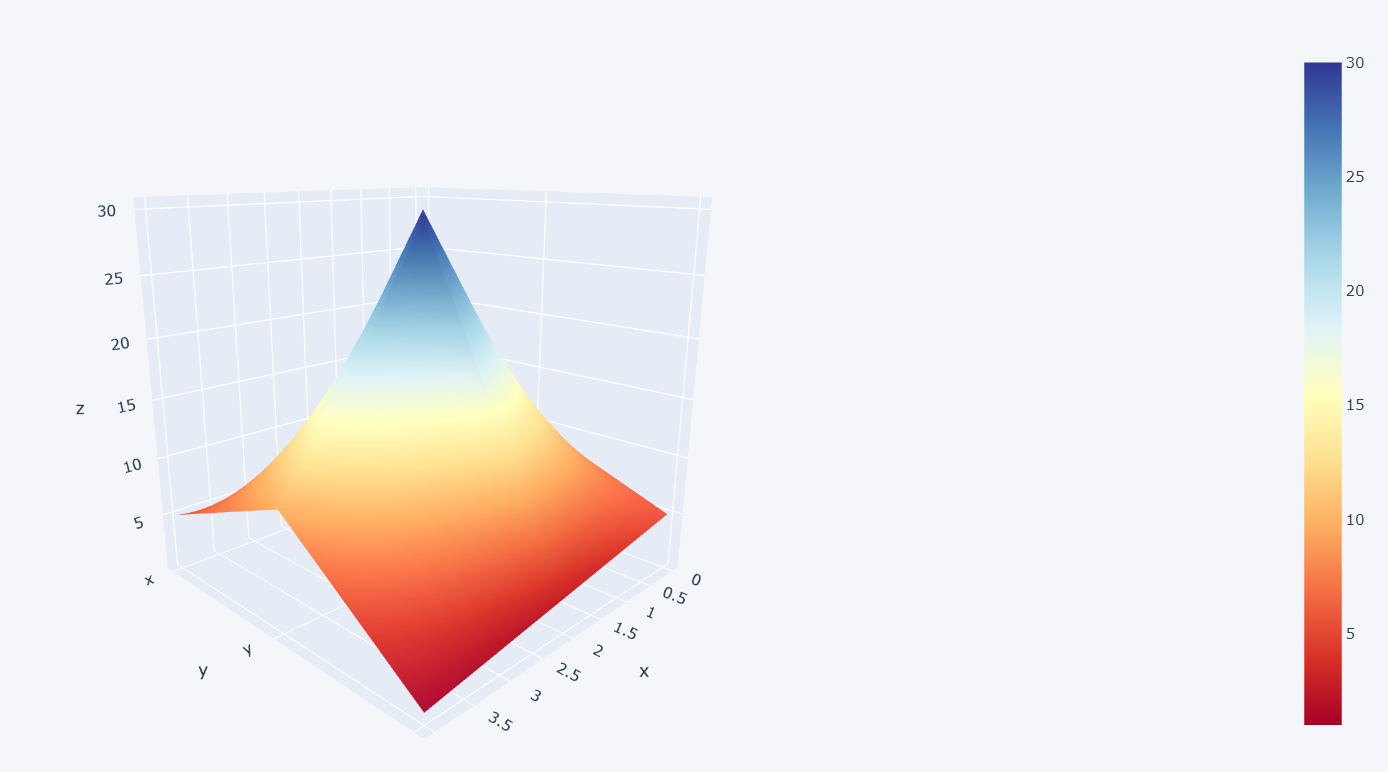
df = pd.DataFrame(np.random.randn(**100, 4**)**,** columns="A B C D".split())

df2 = pd.DataFrame({"Category": ["A"**,** "B"**,** "C"]**,** "Values": [**32, 43, 50**]})

df3 = pd.DataFrame({'x': [**1, 2, 3, 4, 5**]**,** 'y': [**10, 20, 30, 20, 10**]**,** 'z': [**5, 4, 3, 2, 1**]})

fig = df3.iplot(kind='surface'**,**colorscale="rdylbu" **,** asFigure=True)

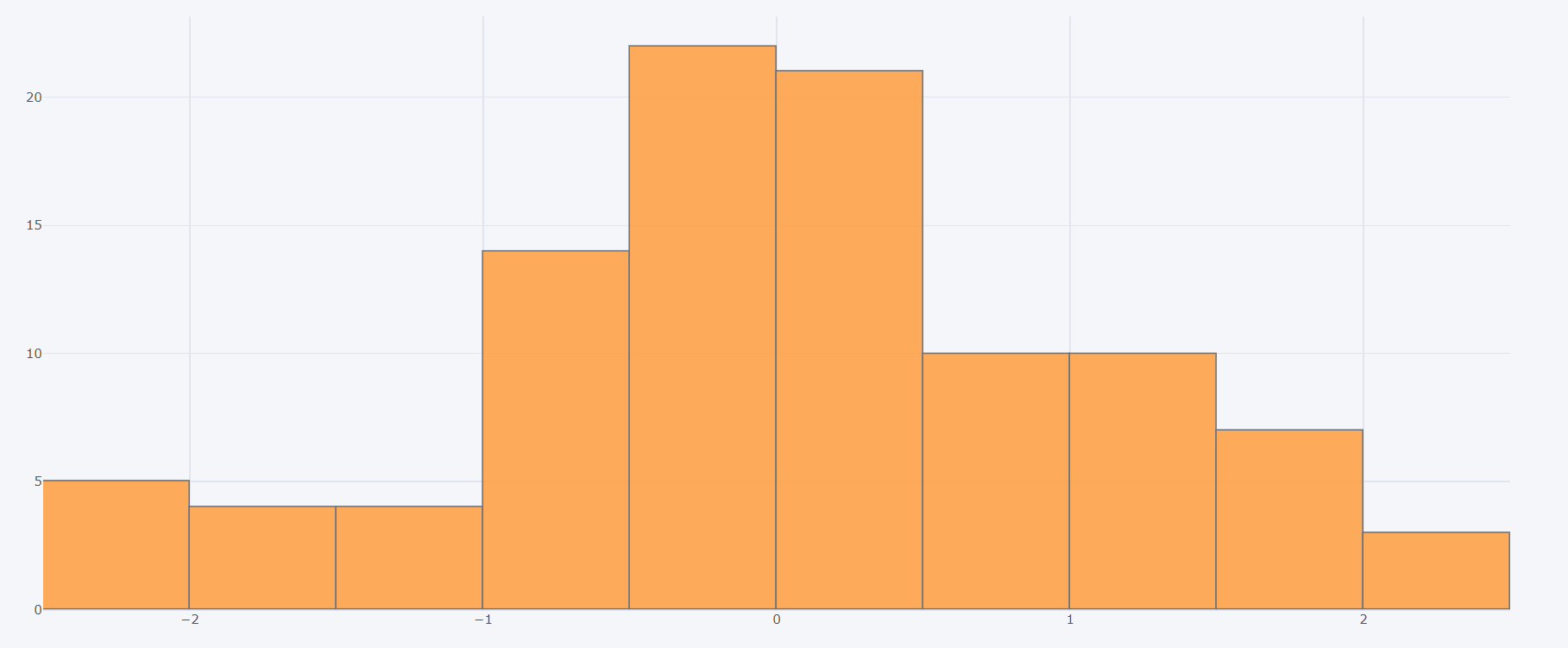
py.offline.plot(fig)



Histogram

fig = df["A"].iplot(kind='hist'**,**bins=**20,** asFigure=True)

py.offline.plot(fig)



You can close just by clicking on them

fig = df.iplot(kind='hist'**,**bins=**50,** asFigure=True)

py.offline.plot(fig)



fig = df[["A"**,** "B"]].iplot(kind='spread'**,** asFigure=True)

py.offline.plot(fig)

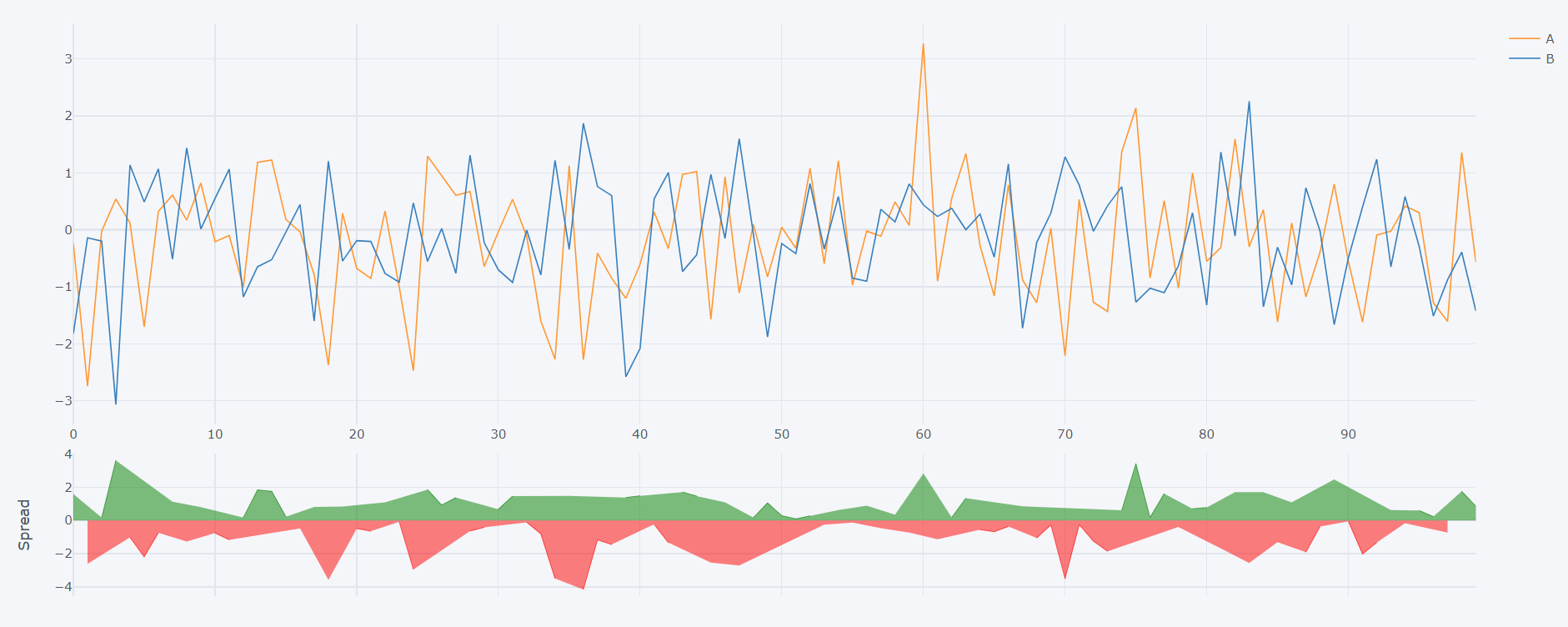
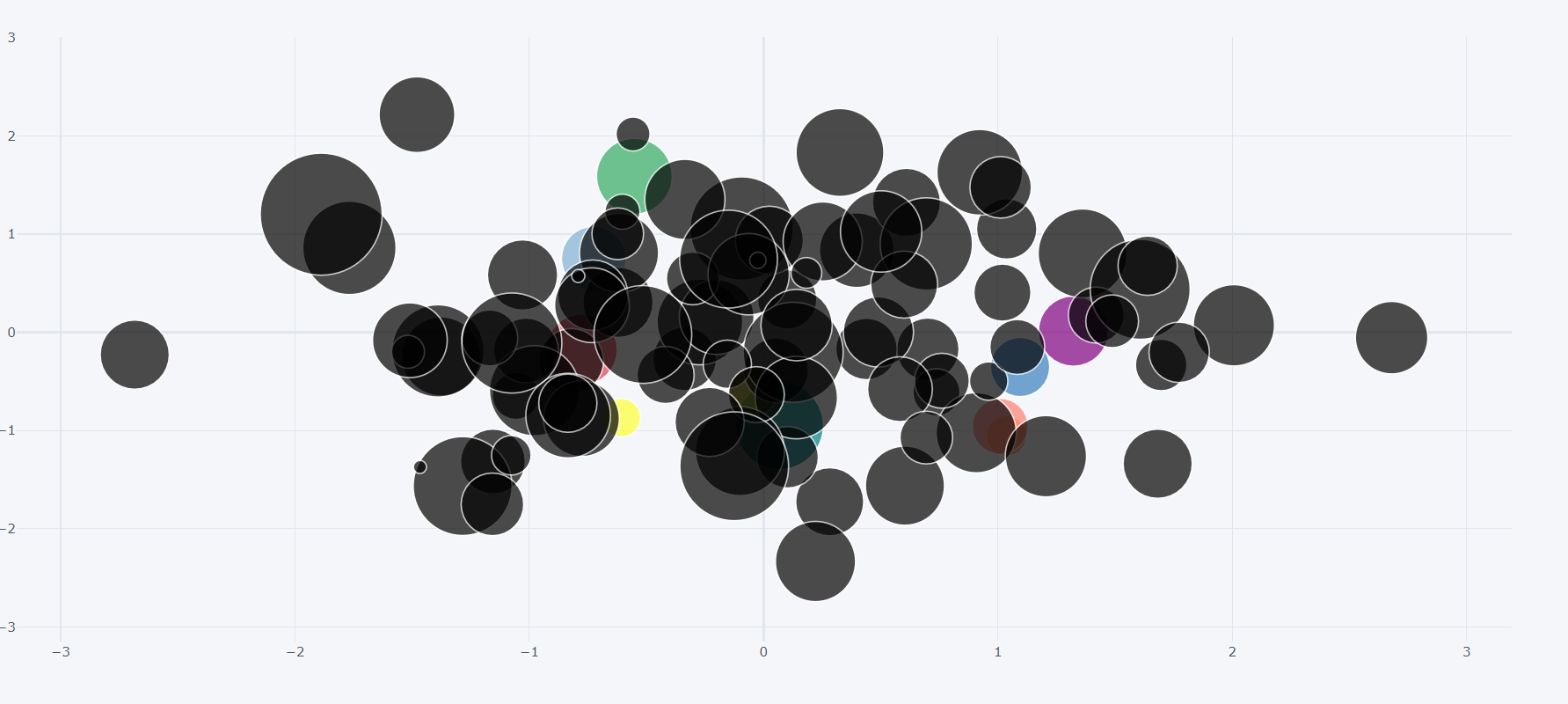


fig = df.iplot(kind='bubble'**,** x="A"**,** y="B"**,** size="C"**,** asFigure=True)

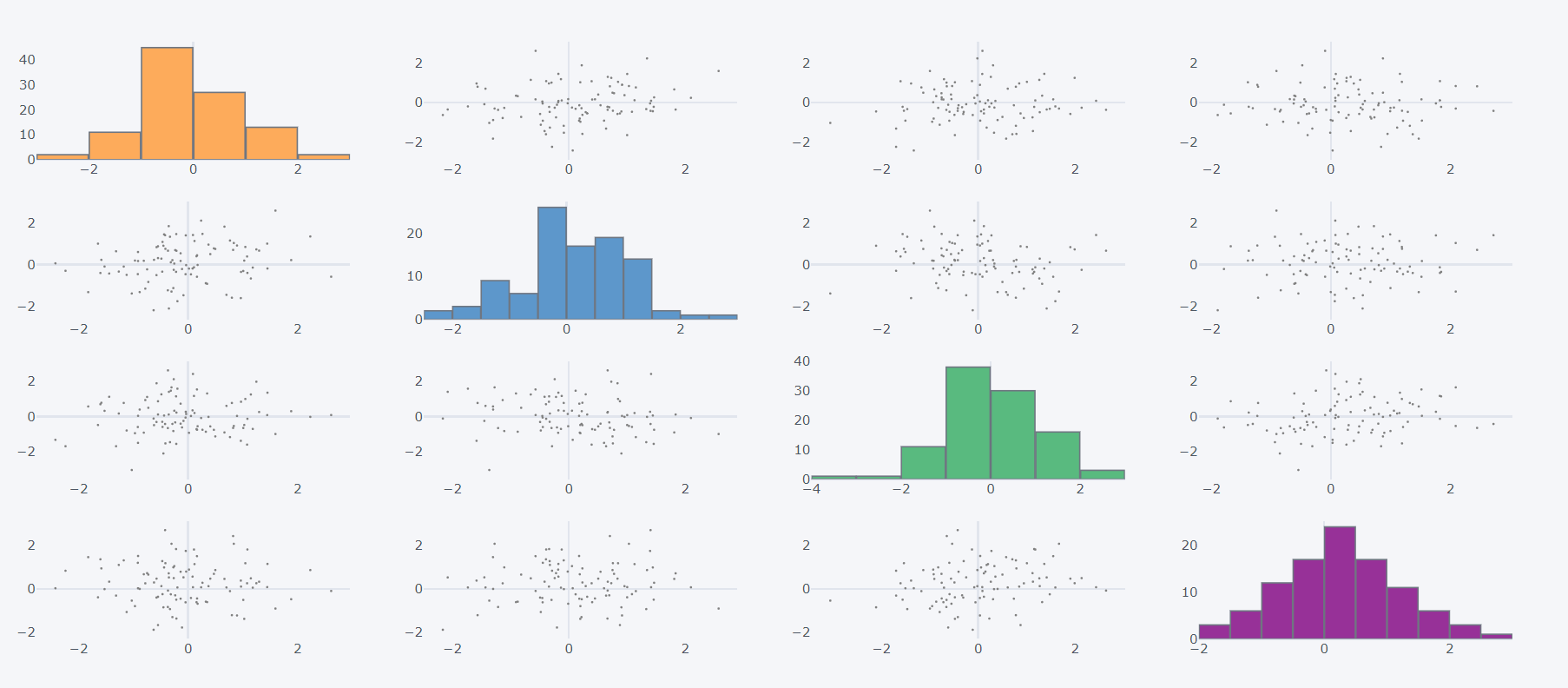
py.offline.plot(fig)



Scatter Matrix

fig = df.scatter\_matrix(asFigure=True)

py.offline.plot(fig)



Choropleth Maps

import plotly as py

data = dict(type="choropleth"**,**

locations=["AZ"**,** "CA"**,** "NY"]**,**

locationmode="USA-states"**,**

colorscale="Portland"**,**

text=['text1'**,** 'text2'**,** 'text3']**,**

z=[**1.0, 2.0, 3.0**]**,**

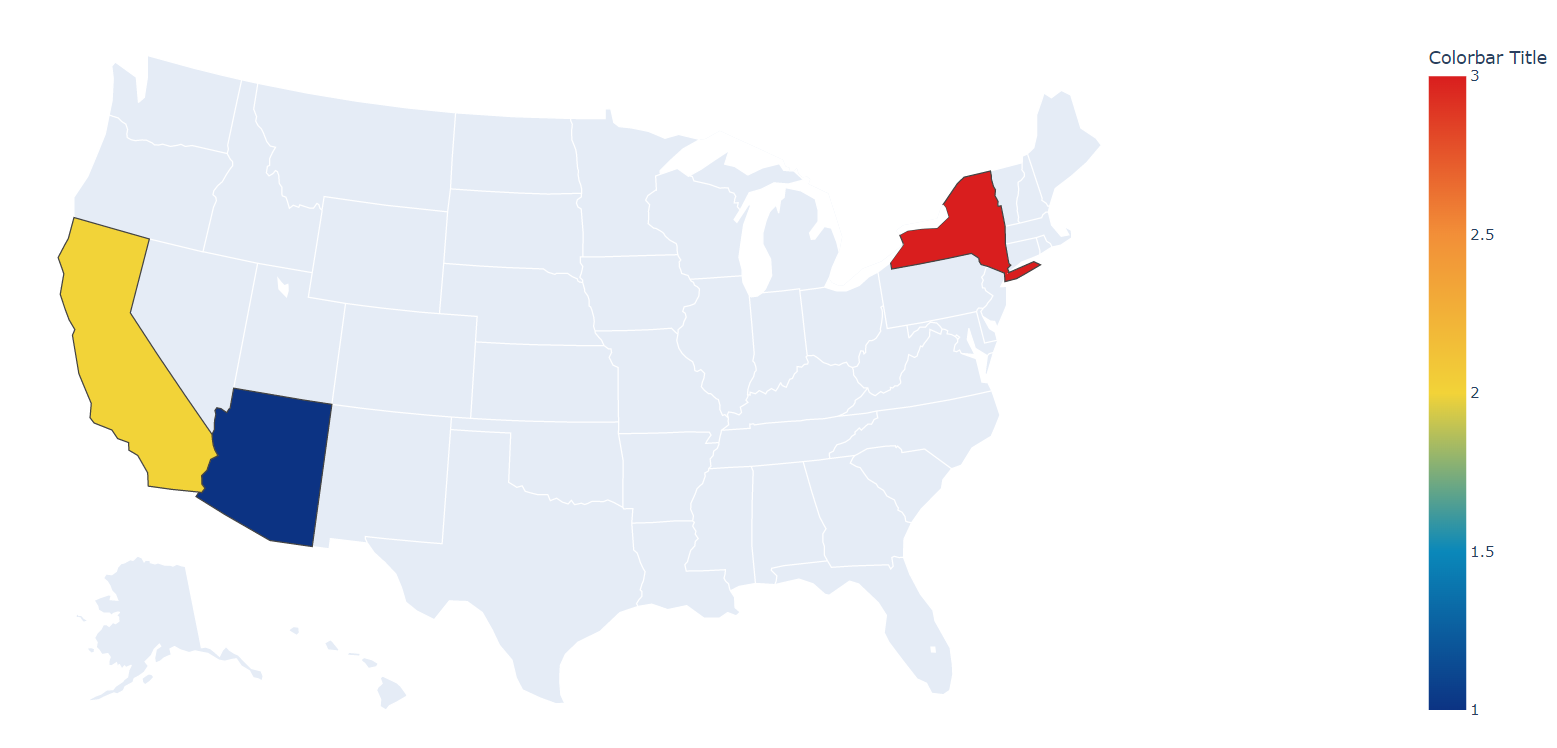
colorbar={'title': 'Colorbar Title'}

)

layout = dict(geo={'scope': 'usa'})

fig = py.graph\_objs.Figure(data=[data]**,** layout=layout)

py.offline.plot(fig)



import plotly as py

import pandas as pd

df = pd.read\_csv("2011\_US\_AGRI\_Exports")

data = dict(type='choropleth'**,**

colorscale='balance'**,**

locations=df['code']**,**

z=df['total exports']**,**

locationmode='USA-states'**,**

text=df['text']**,**

marker=dict(line=dict(color='rgb(255,255,255)'**,** width=**2**))**,**

colorbar={'title': "Millions USD"}

)

layout = dict(title='2011 US Agriculture Exports by State'**,**

geo=dict(scope='usa'**,**

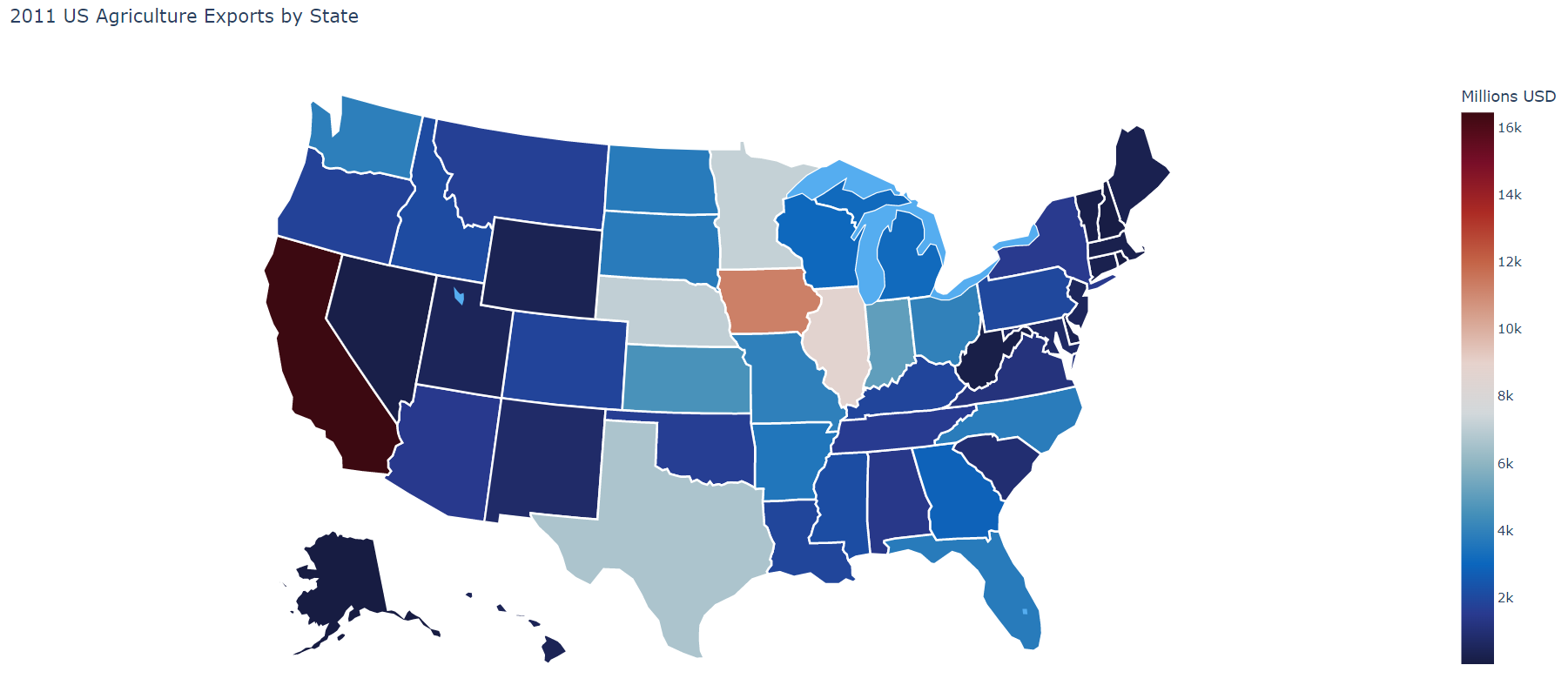
showlakes=True**,**

lakecolor='rgb(85,173,240)')

)

fig = py.graph\_objs.Figure(data=[data]**,** layout=layout)

py.offline.iplot(fig)



import plotly as py

import pandas as pd

df = pd.read\_csv("2014\_World\_GDP")

data = dict(type='choropleth'**,**

colorscale='balance'**,**

locations=df['CODE']**,**

z=df['GDP (BILLIONS)']**,**

text=df['COUNTRY']**,**

marker=dict(line=dict(color='rgb(255,255,255)'**,** width=**2**))**,**

colorbar={'title': "GDP Billions US"}

)

layout = dict(

title='2014 Global GDP'**,**

geo=dict(

showframe=False**,**

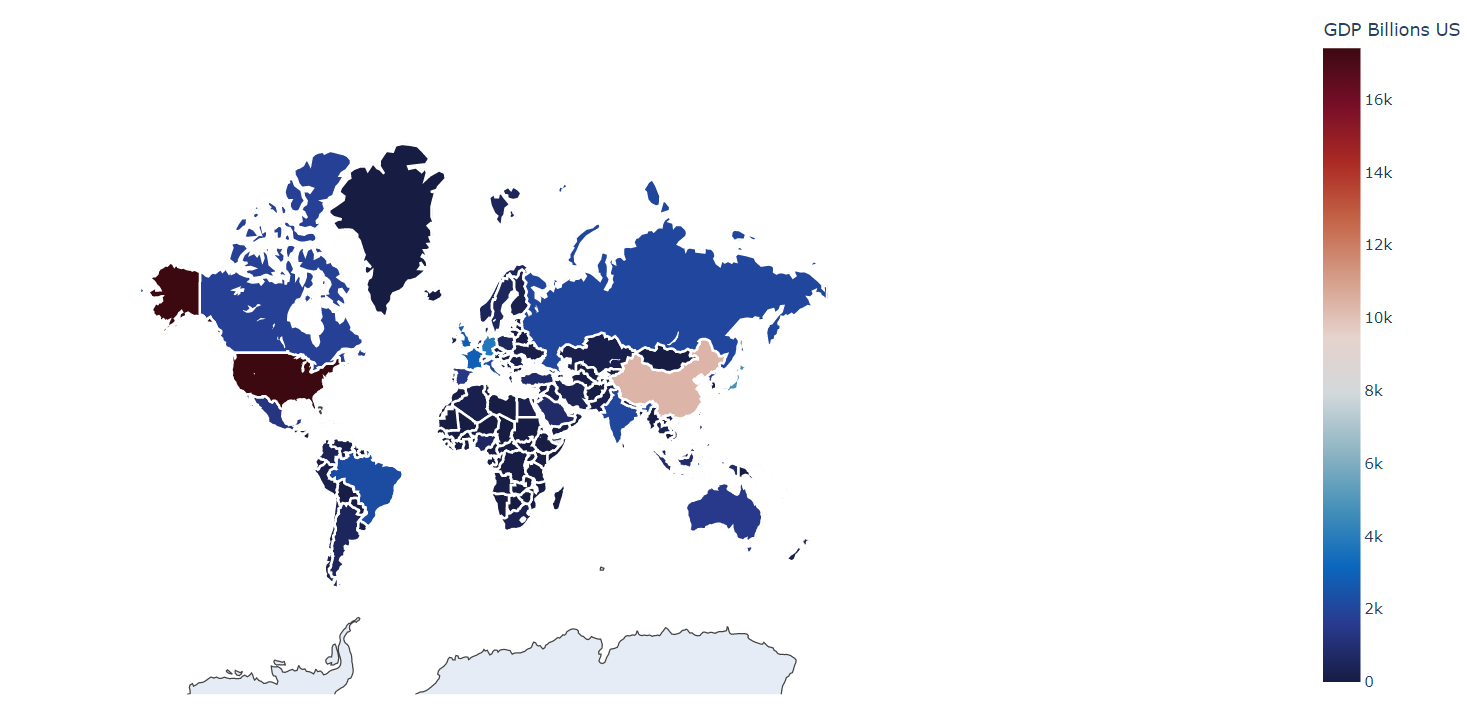
projection={'type': 'mercator'}

)

)

fig = py.graph\_objs.Figure(data=[data]**,** layout=layout)

py.offline.iplot(fig)



import plotly as py

import pandas as pd

df = pd.read\_csv("2014\_World\_GDP")

data = dict(type='choropleth'**,**

colorscale='inferno'**,**

locations=df['CODE']**,**

z=df['GDP (BILLIONS)']**,**

text=df['COUNTRY']**,**

marker=dict(line=dict(color='rgb(255,255,255)'**,** width=**2**))**,**

colorbar={'title': "GDP Billions US"}

)

layout = dict(

title='2014 Global GDP'**,**

geo=dict(

showframe=False**,**

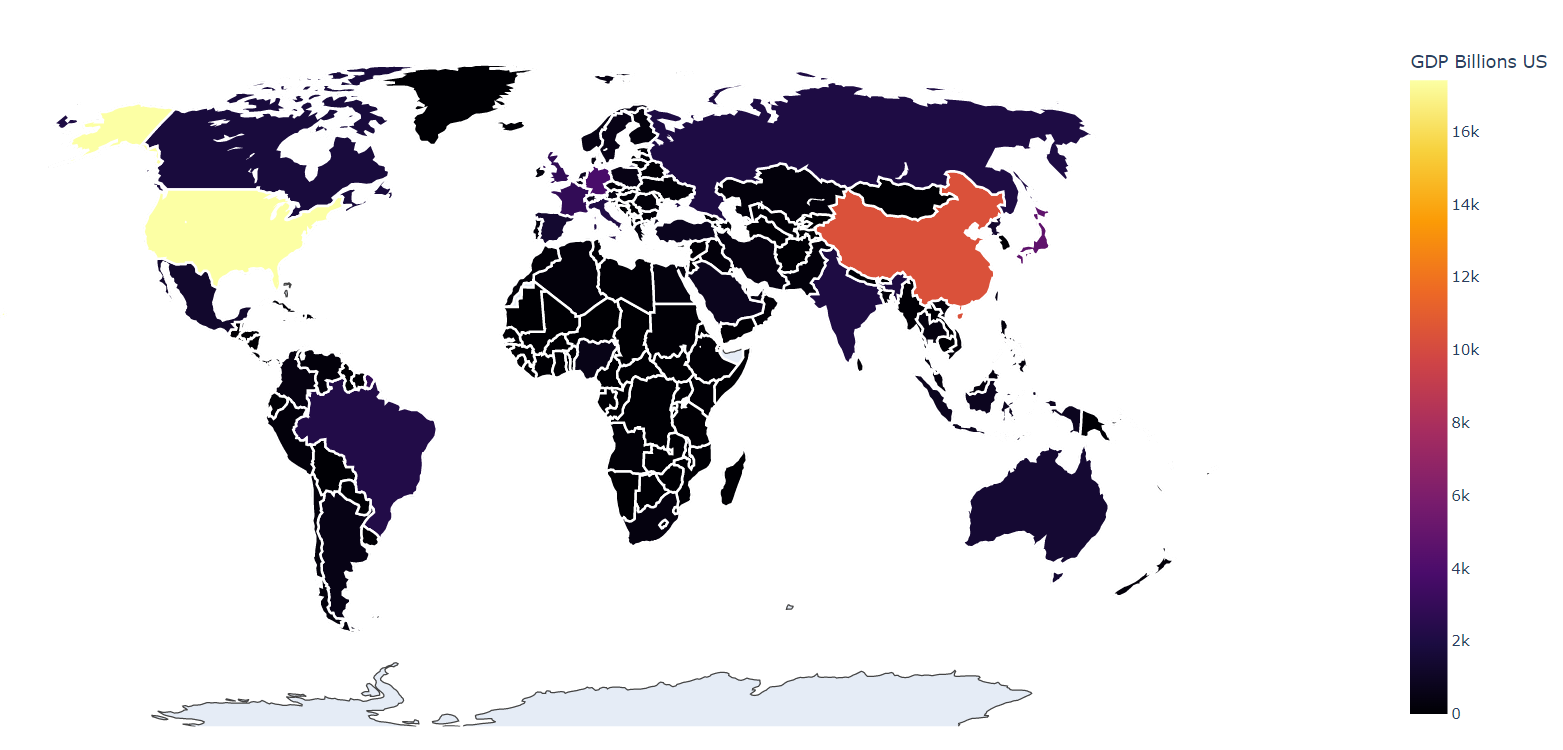
projection={'type': 'natural earth'}

)

)

fig = py.graph\_objs.Figure(data=[data]**,** layout=layout)

py.offline.iplot(fig)



https://youtube.com/shorts/-\_8DvtUbgDY?feature=shares

**Machine Learning**

**Linear Regression with Python**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

df = pd.read\_csv("USA\_Housing.csv")

X = df[['Avg. Area Income'**,** 'Avg. Area House Age'**,** 'Avg. Area Number of Rooms'**,** 'Avg. Area Number of Bedrooms'**,** 'Area Population']]

y = df["Price"]

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.4,** random\_state=**101**)

lm = LinearRegression()

lm.fit(X\_train**,** y\_train)

print(lm.intercept\_)

-2640159.7968526776

print(lm.coef\_)

[2.15282755e+01 1.64883282e+05 1.22368678e+05 2.23380186e+03

1.51504200e+01]

**Coefficient for each feature**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

df = pd.read\_csv("USA\_Housing.csv")

X = df[['Avg. Area Income'**,** 'Avg. Area House Age'**,** 'Avg. Area Number of Rooms'**,** 'Avg. Area Number of Bedrooms'**,** 'Area Population']]

y = df["Price"]

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.4,** random\_state=**101**)

lm = LinearRegression()

lm.fit(X\_train**,** y\_train)

cdf = pd.DataFrame(lm.coef\_**,** X.columns**,** columns=["Coef"])

print(cdf)

Coef

Avg. Area Income 21.528276

Avg. Area House Age 164883.282027

Avg. Area Number of Rooms 122368.678027

Avg. Area Number of Bedrooms 2233.801864

Area Population 15.150420

Predictions

Predictions of house prices

predictions = lm.predict(X\_test)

print(predictions)

[1260960.70567627 827588.75560329 1742421.24254344 ... 372191.40626917

1365217.15140899 1914519.5417888 ]

y\_test has the correct information about house prices

print(y\_test)1718 1.251689e+06

2511 8.730483e+05

345 1.696978e+06

2521 1.063964e+06

54 9.487883e+05

...

1776 1.489520e+06

4269 7.777336e+05

1661 1.515271e+05

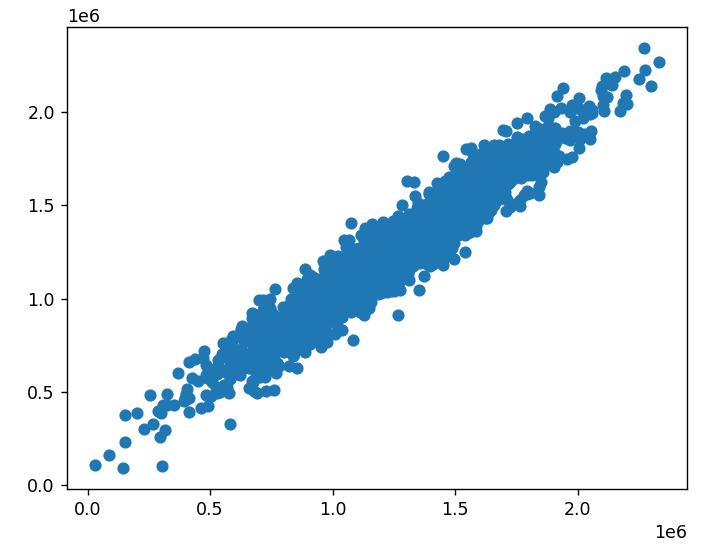
2410 1.343824e+06

2302 1.906025e+06

To compare predictions to true values we use scatter map here

plt.scatter(y\_test**,**predictions)

plt.show()

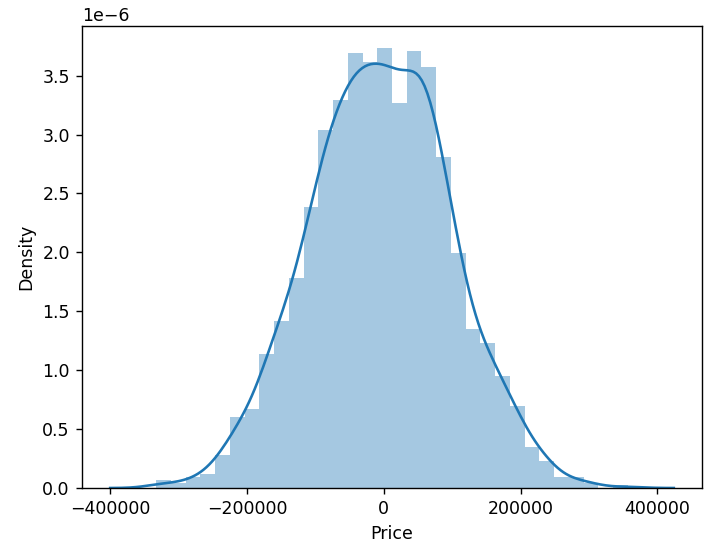


If the line up like this it’s means we have a good predictions

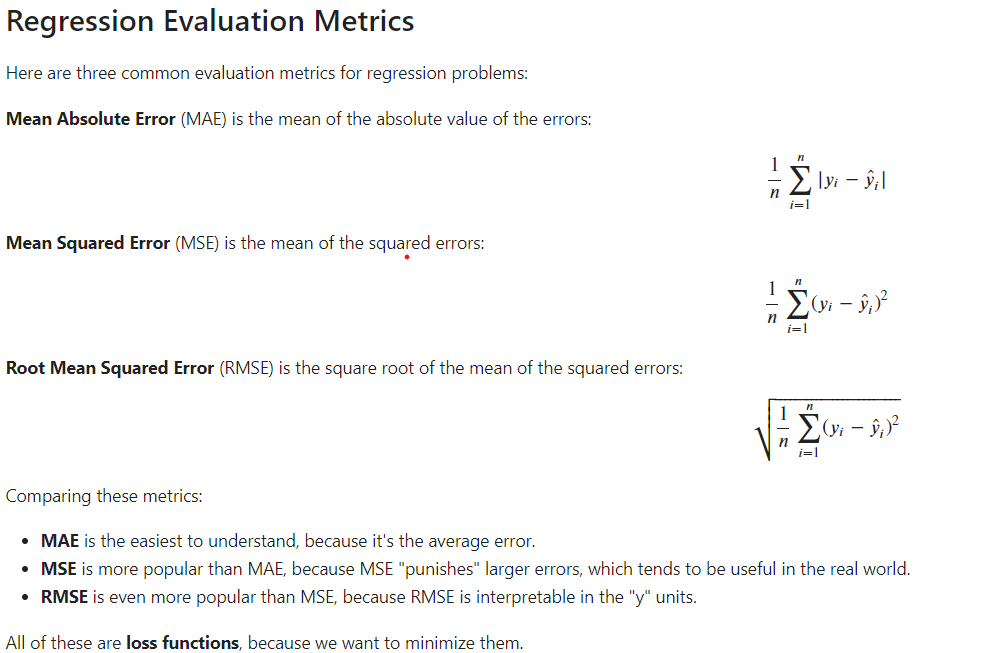
# Residuals = test-prediction

sns.distplot((y\_test-predicitons))

plt.show()



If residuals are distributed normally that's a good sign.



For this ve import from sklearn import metrics

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

# from sklearn.datasets import load\_breast\_cancer

df = pd.read\_csv("USA\_Housing.csv")

X = df[['Avg. Area Income'**,** 'Avg. Area House Age'**,** 'Avg. Area Number of Rooms'**,** 'Avg. Area Number of Bedrooms'**,** 'Area Population']]

y = df["Price"]

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.4,** random\_state=**101**)

lm = LinearRegression()

lm.fit(X\_train**,** y\_train)

cdf = pd.DataFrame(lm.coef\_**,** X.columns**,** columns=["Coef"])

predictions = lm.predict(X\_test)

print('MAE:'**,** metrics.mean\_absolute\_error(y\_test**,** predictions))

print('MSE:'**,** metrics.mean\_squared\_error(y\_test**,** predictions))

print('RMSE:'**,** np.sqrt(metrics.mean\_squared\_error(y\_test**,** predictions)))

MAE: 82288.22251914951

MSE: 10460958907.208992

RMSE: 102278.82922290904

**Logistic Regression**

Data Cleaning

First we’re looking empty data inside of Titanic dataset

import pandas as pd

import seaborn as sns

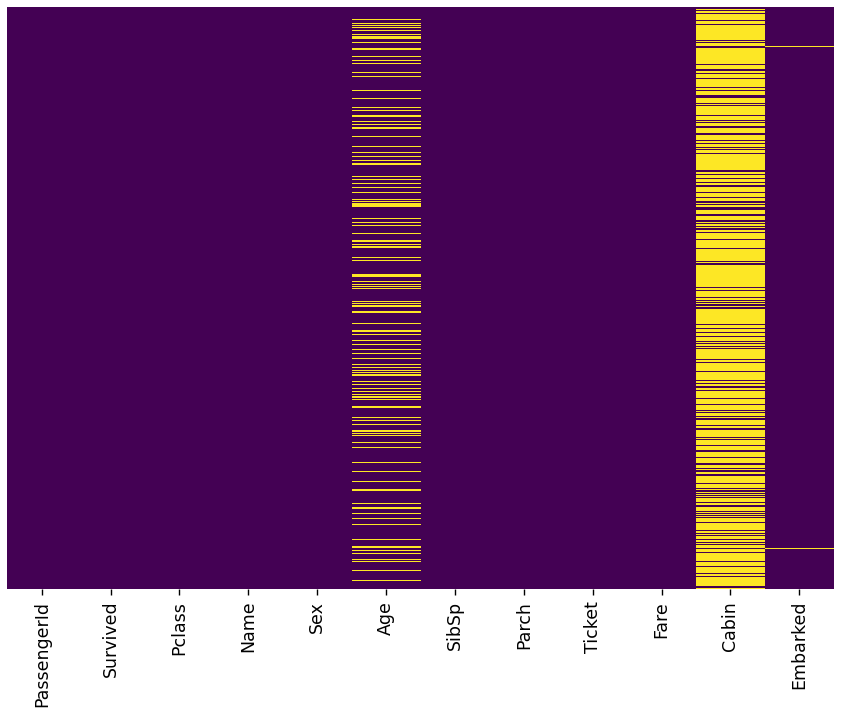
import matplotlib.pyplot as plt

df = pd.read\_csv("titanic\_train.csv")

sns.heatmap(df.isnull()**,** cmap="viridis"**,** cbar=False**,** yticklabels=False)

plt.tight\_layout()

plt.show()

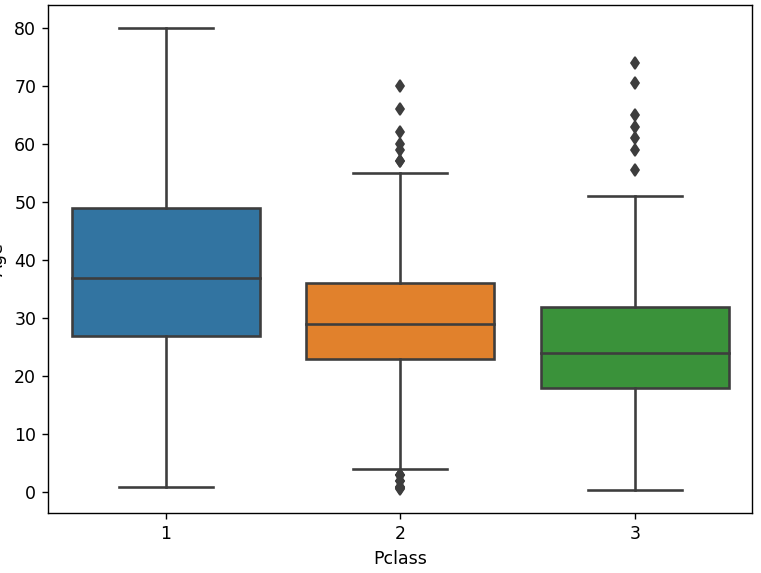


Now we will look at wealth and age correlation

sns.boxplot(x="Pclass"**,** y="Age"**,** data=df)

plt.tight\_layout()

plt.show()



As we can see when Class goes 3 to 1 age is also increased.

Now we will use the age-pclass correlation’s mean value to cover up null values

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv("titanic\_train.csv")

def ridder(cols):

Age = cols[**0**]

Pclass = cols[**1**]

if pd.isnull(Age):

if Pclass == **1**:

return **37**

if Pclass == **1**:

return **37**

elif Pclass == **2**:

return **29**

else:

return **24**

else:

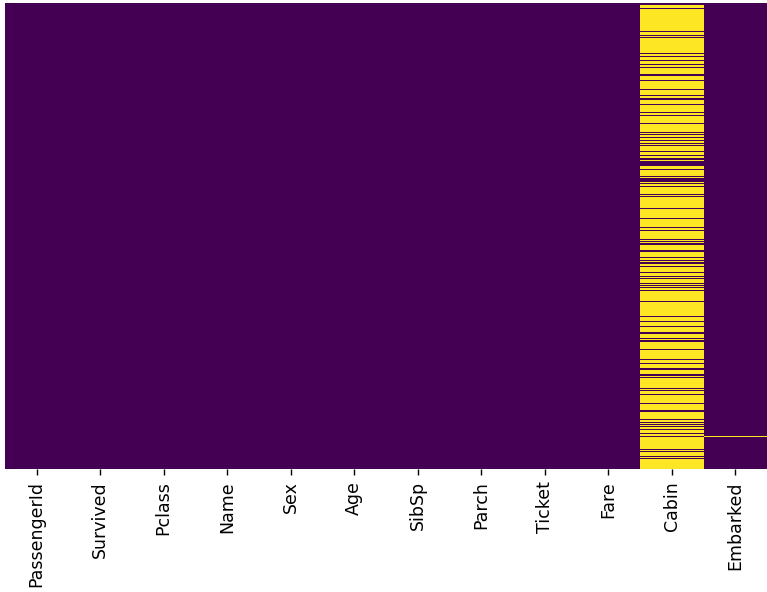
return Age

df["Age"] = df[["Age"**,** "Pclass"]].apply(ridder**,** axis=**1**)

sns.heatmap(df.isnull()**,** cmap="viridis"**,** yticklabels=False**,** cbar=False)

plt.tight\_layout()

plt.show()



As you can see we successfully removed null values inside of an age column

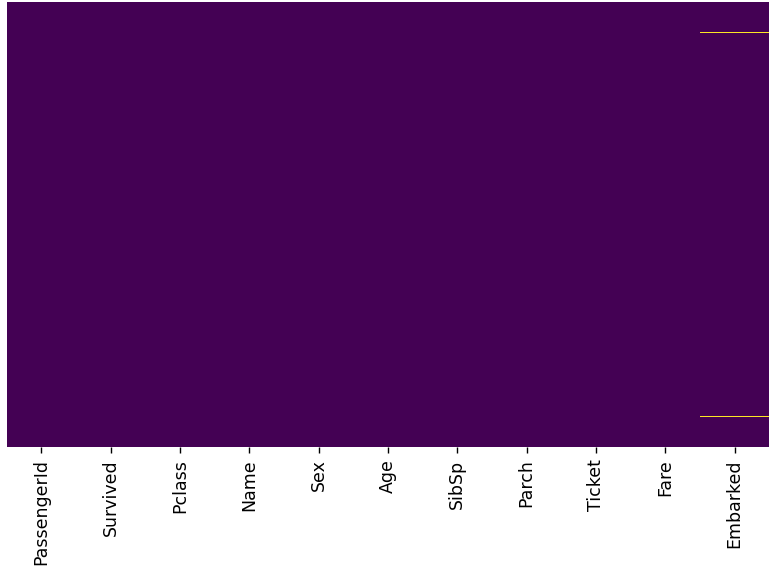
Now we will remove the all cabin column because the column nearly all full of null values

df.drop("Cabin"**,**axis=**1,** inplace=True)

sns.heatmap(df.isnull()**,** cmap="viridis"**,** yticklabels=False**,** cbar=False)

plt.tight\_layout()

plt.show()



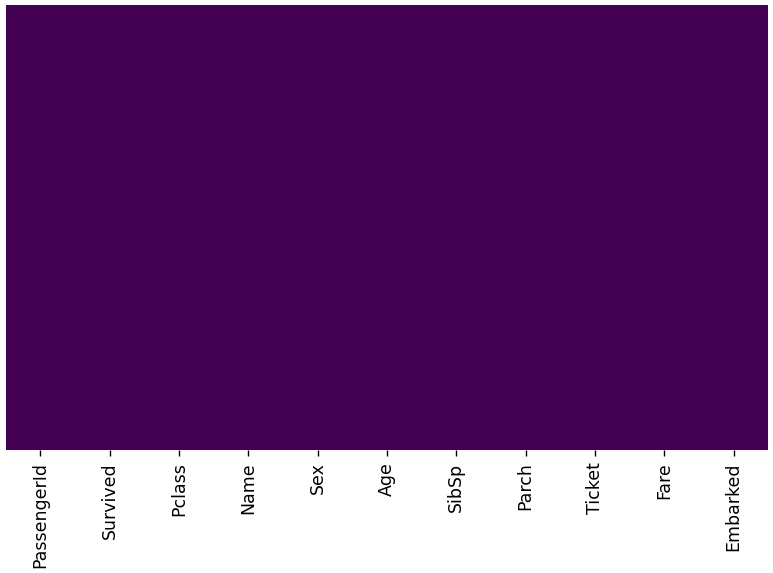
And in the final column as we see there are not many null values in it. So we just drop the null values and go on.

df.dropna(inplace=True)

sns.heatmap(df.isnull()**,** cmap="viridis"**,** yticklabels=False**,** cbar=False)

plt.tight\_layout()

plt.show()



Now we are good to go. There is no null value left in our dataset

Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

Normal

print(df["Sex"].head())

0 male

1 female

2 female

3 female

4 male

Dummies

sex\_dummies = pd.get\_dummies(df["Sex"])

print(sex\_dummies.head())

female male

0 0 1

1 1 0

2 1 0

3 1 0

4 0 1

We shaped the output to become understandable for machine learning operations

sex\_dummies = pd.get\_dummies(df["Sex"]**,**drop\_first=True)

male

0 1

1 0

2 0

3 0

4 1

Also for embark column

embarked\_dummies = pd.get\_dummies(df["Embarked"]**,**drop\_first=True)

print(embarked\_dummies.head())

Q S

0 0 1

1 0 0

2 0 1

3 0 1

4 0 1

And now we’re gonna add those columns to the dataframe and remove the columns we won’t gonna use

df.drop(["Sex"**,** "Embarked"**,** "Name"**,** "Ticket"]**,** axis=**1,**inplace=True)

print(df.head())

PassengerId Survived Pclass Age SibSp Parch Fare male Q S

0 1 0 3 22.0 1 0 7.2500 1 0 1

1 2 1 1 38.0 1 0 71.2833 0 0 0

2 3 1 3 26.0 0 0 7.9250 0 0 1

3 4 1 1 35.0 1 0 53.1000 0 0 1

4 5 0 3 35.0 0 0 8.0500 1 0 1

Now we have only numerical values in our dataframe and it’s ready to use for machine learning

y is the columns that we want to predict the actual label

And we have nice classification tool inside of sklearn

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

df = pd.read\_csv("titanic\_train.csv")

def ridder(cols):

Age = cols[**0**]

Pclass = cols[**1**]

if pd.isnull(Age):

if Pclass == **1**:

return **37**

if Pclass == **1**:

return **37**

elif Pclass == **2**:

return **29**

else:

return **24**

else:

return Age

df["Age"] = df[["Age"**,** "Pclass"]].apply(ridder**,** axis=**1**)

df.drop("Cabin"**,** axis=**1,** inplace=True)

df.dropna(inplace=True)

sex\_dummies = pd.get\_dummies(df["Sex"]**,** drop\_first=True)

embarked\_dummies = pd.get\_dummies(df["Embarked"]**,** drop\_first=True)

df = pd.concat([df**,** sex\_dummies**,** embarked\_dummies]**,** axis=**1**)

df.drop(["Sex"**,** "Embarked"**,** "Name"**,** "Ticket"**,** "PassengerId"]**,** axis=**1,** inplace=True)

X = df.drop("Survived"**,** axis=**1**)

y = df["Survived"]

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.3,** random\_state=**101**)

log = LogisticRegression()

log.fit(X\_train**,** y\_train)

predictions = log.predict(X\_test)

print(metrics.classification\_report(y\_test**,** predictions))

precision recall f1-score support

0 0.83 0.90 0.86 163

1 0.82 0.71 0.76 104

accuracy 0.83 267

macro avg 0.83 0.81 0.81 267

weighted avg 0.83 0.83 0.83 267

For confusion matrix

print(metrics.confusion\_matrix(y\_test**,** predictions))

[[147 16]

[ 30 74]]

KNN(K Nearest Neighbors with Python)

It’s a classification algorithm that operates on a very simple principle.

The training algorithm is very simple. You simply store all the data and the prediction algorithm for new test points. Works like this. You calculate the distance from X to all the points in your data X indicating that particular new data point. Then you sort the points near data by increasing distance from X. Then you predict the majority label of the K K being a number. Closest points choosing a K will affect what class a new point is assigned to

Worse for large datasets

Not good with high dimensional data

Categorical features don't work well

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

df = pd.read\_csv("Classified Data"**,**index\_col=**0**)

print(df.head())

WTT PTI EQW ... HQE NXJ TARGET CLASS

0 0.913917 1.162073 0.567946 ... 0.879422 1.231409 1

1 0.635632 1.003722 0.535342 ... 0.621552 1.492702 0

2 0.721360 1.201493 0.921990 ... 0.957877 1.285597 0

3 1.234204 1.386726 0.653046 ... 1.522692 1.153093 1

4 1.279491 0.949750 0.627280 ... 1.463812 1.419167 1

df = pd.read\_csv("Classified Data"**,**index\_col=**0**)

print(df.info())

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 WTT 1000 non-null float64

1 PTI 1000 non-null float64

2 EQW 1000 non-null float64

3 SBI 1000 non-null float64

4 LQE 1000 non-null float64

5 QWG 1000 non-null float64

6 FDJ 1000 non-null float64

7 PJF 1000 non-null float64

8 HQE 1000 non-null float64

9 NXJ 1000 non-null float64

10 TARGET CLASS 1000 non-null int64

Standard Version

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.preprocessing import StandardScaler

df = pd.read\_csv("Classified Data"**,** index\_col=**0**)

scaler = StandardScaler()

scaler.fit(df.drop("TARGET CLASS"**,** axis=**1**))

scaled\_features = scaler.transform(df.drop('TARGET CLASS'**,** axis=**1**))

df\_feat = pd.DataFrame(scaled\_features**,** columns=df.columns[:-**1**])

print(df\_feat.head())

WTT PTI EQW ... PJF HQE NXJ

0 -0.123542 0.185907 -0.913431 ... -1.482368 -0.949719 -0.643314

1 -1.084836 -0.430348 -1.025313 ... -0.202240 -1.828051 0.636759

2 -0.788702 0.339318 0.301511 ... 0.285707 -0.682494 -0.377850

3 0.982841 1.060193 -0.621399 ... 1.066491 1.241325 -1.026987

4 1.139275 -0.640392 -0.709819 ... -1.472352 1.040772 0.276510

For k=1

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report**,** confusion\_matrix

df = pd.read\_csv("Classified Data"**,** index\_col=**0**)

scaler = StandardScaler()

scaler.fit(df.drop("TARGET CLASS"**,** axis=**1**))

scaled\_features = scaler.transform(df.drop('TARGET CLASS'**,** axis=**1**))

df\_feat = pd.DataFrame(scaled\_features**,** columns=df.columns[:-**1**])

X = df\_feat

y = df["TARGET CLASS"]

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.3,** random\_state=**101**)

knn = KNeighborsClassifier(n\_neighbors=**1**)

knn.fit(X\_train**,** y\_train)

predictions = knn.predict(X\_test)

print(confusion\_matrix(y\_test**,** predictions))

print(classification\_report(y\_test**,** predictions))

[[151 8]

[ 15 126]]

precision recall f1-score support

0 0.91 0.95 0.93 159

1 0.94 0.89 0.92 141

accuracy 0.92 300

macro avg 0.92 0.92 0.92 300

weighted avg 0.92 0.92 0.92 300

Error Rate and Elbow Method

error\_rate = []

for i in range(**1, 40**):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(X\_train**,** y\_train)

pred\_i = knn.predict(X\_test)

error\_rate.append(np.mean(pred\_i != y\_test))

plt.figure(figsize=(**10, 6**))

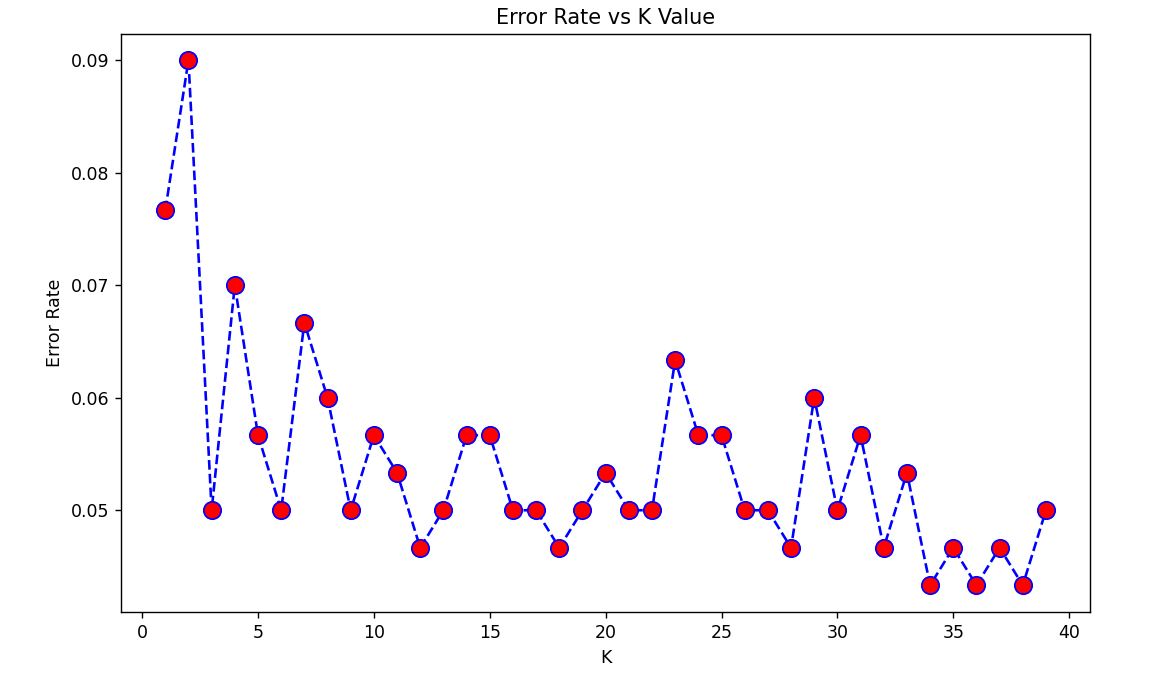
plt.plot(range(**1, 40**)**,** error\_rate**,** color="blue"**,** linestyle="dashed"**,** marker="o"**,** markerfacecolor="red"**,** markersize=**10**)

plt.title("Error Rate vs K Value")

plt.xlabel("K")

plt.ylabel("Error Rate")

plt.show()



As we can see error rates are bouncing but if we take one of the lowest k value and error rate node and than make an example of that output is gonna be like this:

knn = KNeighborsClassifier(n\_neighbors=**17**)

knn.fit(X\_train**,**y\_train)

predictions = knn.predict(X\_test)

print(confusion\_matrix(y\_test**,** predictions))

print()

print(classification\_report(y\_test**,** predictions))

[[153 6]

[ 9 132]]

precision recall f1-score support

0 0.94 0.96 0.95 159

1 0.96 0.94 0.95 141

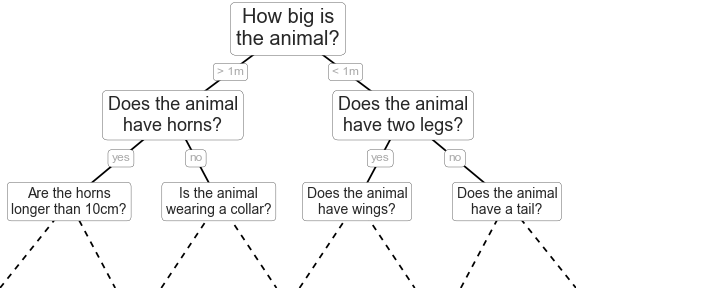
accuracy 0.95 300

macro avg 0.95 0.95 0.95 300

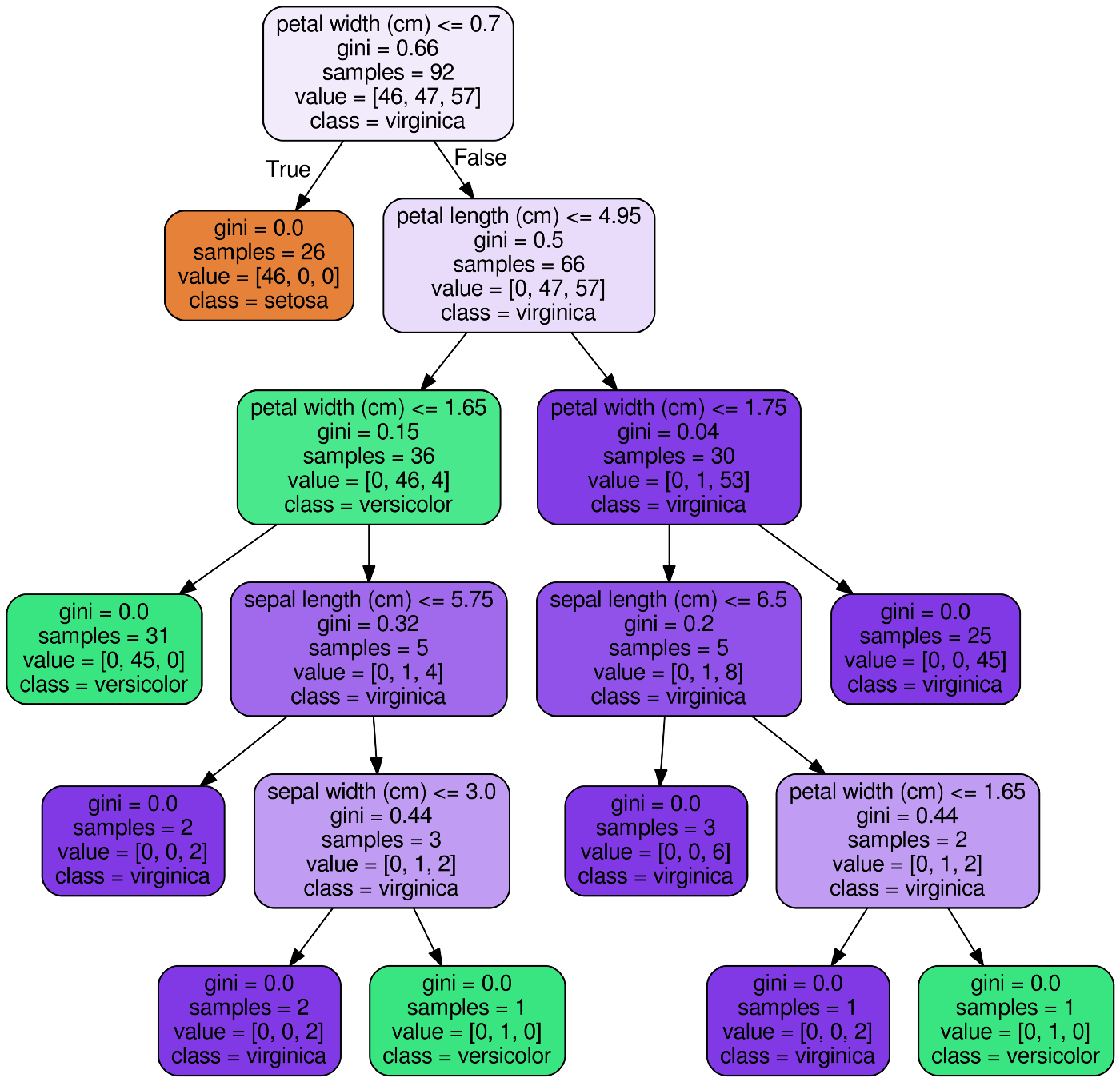
weighted avg 0.95 0.95 0.95 300

We went 92% accuracy to 95% accuracy

Decision Trees and Random Forests in Python



Or



The primary weakness of decision trees is that they don't tend to have the best predictive accuracy

Decision Tree

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report**,**confusion\_matrix

df = pd.read\_csv('kyphosis.csv')

X = df.drop('Kyphosis'**,** axis=**1**)

y = df['Kyphosis']

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.30**)

dtree = DecisionTreeClassifier()

dtree.fit(X\_train**,** y\_train)

predict = dtree.predict(X\_test)

print(classification\_report(y\_test**,**predict))

print()

print(confusion\_matrix(y\_test**,**predict))

precision recall f1-score support

absent 0.82 0.95 0.88 19

present 0.67 0.33 0.44 6

accuracy 0.80 25

macro avg 0.74 0.64 0.66 25

weighted avg 0.78 0.80 0.77 25

[[18 1]

[ 4 2]]

Random Forest

It’s better for larger datasets

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report**,** confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

df = pd.read\_csv('kyphosis.csv')

X = df.drop('Kyphosis'**,** axis=**1**)

y = df['Kyphosis']

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.30**)

rfc = RandomForestClassifier(n\_estimators=**200**)

rfc.fit(X\_train**,** y\_train)

predict = rfc.predict(X\_test)

print(classification\_report(y\_test**,** predict))

print()

print(confusion\_matrix(y\_test**,** predict))

precision recall f1-score support

absent 0.89 0.80 0.84 20

present 0.43 0.60 0.50 5

accuracy 0.76 25

macro avg 0.66 0.70 0.67 25

weighted avg 0.80 0.76 0.77 25

[[16 4]

[ 2 3]]

Function fun

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report**,** confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

# Decision Tree Classifier

def dtc(X**,** y):

dtree = DecisionTreeClassifier()

dtree.fit(X\_train**,** y\_train)

predict = dtree.predict(X\_test)

return predict

# Random Forest Classifier

def rfc(X**,** y):

rfc = RandomForestClassifier(n\_estimators=**200**)

rfc.fit(X\_train**,** y\_train)

predict = rfc.predict(X\_test)

return predict

df = pd.read\_csv("loan\_data.csv")

cat\_feats = ["purpose"]

final\_data = pd.get\_dummies(df**,** columns=cat\_feats**,** drop\_first=True)

X = final\_data.drop('not.fully.paid'**,** axis=**1**)

y = df['not.fully.paid']

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.30,** random\_state=**101**)

print(classification\_report(y\_test**,** rfc(X**,** y)))

precision recall f1-score support

0 0.85 1.00 0.92 2431

1 0.60 0.03 0.05 443

accuracy 0.85 2874

macro avg 0.72 0.51 0.48 2874

weighted avg 0.81 0.85 0.78 2874

Support Vector Machines

SVM’s are also known or supervised learning models of associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_breast\_cancer

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix**,** classification\_report

cancer = load\_breast\_cancer()

df\_feat = pd.DataFrame(cancer['data']**,** columns=cancer['feature\_names'])

X = df\_feat

y = cancer['target']

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.30,** random\_state=**101**)

svm = SVC()

svm.fit(X\_train**,** y\_train)

predict = svm.predict(X\_test)

precision recall f1-score support

0 0.95 0.85 0.90 66

1 0.91 0.97 0.94 105

accuracy 0.92 171

macro avg 0.93 0.91 0.92 171

weighted avg 0.93 0.92 0.92 171

Grid

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_breast\_cancer

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix**,** classification\_report

from sklearn.model\_selection import GridSearchCV

cancer = load\_breast\_cancer()

df\_feat = pd.DataFrame(cancer['data']**,** columns=cancer['feature\_names'])

X = df\_feat

y = cancer['target']

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.30,** random\_state=**101**)

svm = SVC()

svm.fit(X\_train**,** y\_train)

predict = svm.predict(X\_test)

print(classification\_report(y\_test**,** predict))

param\_grid = {"C": [**0.1, 1, 10, 100, 1000**]**,** "gamma": [**1, 0.1, 0.01, 0.001, 0.0001**]}

grid = GridSearchCV(SVC()**,** param\_grid**,** verbose=**3**)

grid.fit(X\_train**,** y\_train)

grid\_pre = grid.predict(X\_test)

print(classification\_report(y\_test**,** grid\_pre))

precision recall f1-score support

0 0.95 0.85 0.90 66

1 0.91 0.97 0.94 105

accuracy 0.92 171

macro avg 0.93 0.91 0.92 171

weighted avg 0.93 0.92 0.92 171

—----------------------------------------------------------------------------------------------------

precision recall f1-score support

0 0.94 0.89 0.91 66

1 0.94 0.96 0.95 105

f

accuracy 0.94 171

macro avg 0.94 0.93 0.93 171

weighted avg 0.94 0.94 0.94 171

K Mean Clustering

K means clustering is an unsupervised learning algorithm meaning it takes an unlabeled data it's going to attempt to group similar clusters together in your data.

import seaborn as sns

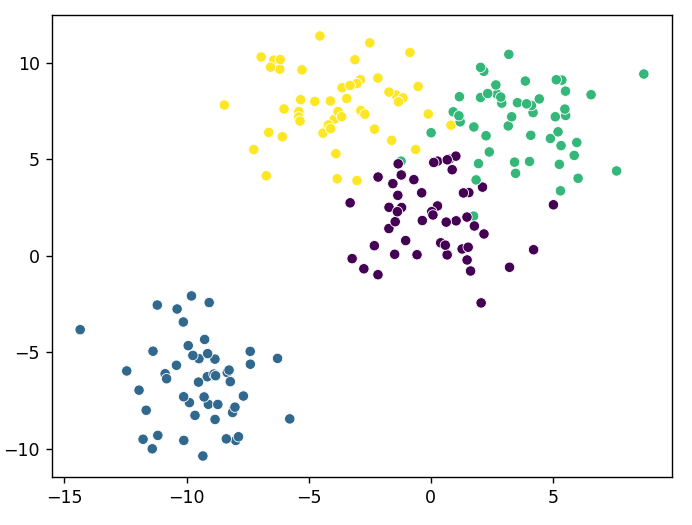
import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

data = make\_blobs(n\_samples=**200,** n\_features=**2,** centers=**4,** cluster\_std=**1.8,** random\_state=**101**)

sns.scatterplot(x=data[**0**][:**, 0**]**,** y=data[**0**][:**, 1**]**,** c=data[**1**]**,** palette="rainbow")

plt.show()



from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

data = make\_blobs(n\_samples=**200,** n\_features=**2,** centers=**4,** cluster\_std=**1.8,** random\_state=**101**)

kmeans = KMeans(n\_clusters=**4,** n\_init=**10**)

kmeans.fit(data[**0**])

print(kmeans.cluster\_centers\_)

print(kmeans.labels\_)

[[ 3.71749226 7.01388735]

[-9.46941837 -6.56081545]

[-0.0123077 2.13407664]

[-4.13591321 7.95389851]]

[3 0 2 0 0 1 0 2 0 2 3 2 0 0 3 2 0 2 1 3 1 2 2 1 3 1 1 2 0 0 3 1 0 2 2 3 1

1 1 2 1 3 3 3 2 0 3 2 1 2 2 3 0 2 1 3 2 2 3 0 1 0 1 3 0 2 1 0 0 1 0 2 1 2

1 0 0 2 3 2 2 1 0 1 2 2 2 3 2 1 1 1 1 2 2 1 0 3 1 0 2 1 2 2 0 2 1 0 1 1 0

3 3 0 1 0 3 3 0 3 2 3 2 3 2 0 3 2 1 3 3 3 2 1 1 3 0 3 0 2 1 0 1 3 3 0 2 1

3 3 3 3 2 0 2 3 0 0 0 2 0 2 2 3 1 3 2 0 3 2 0 2 3 0 2 3 0 0 1 0 3 1 1 3 1

1 1 1 1 2 1 0 0 3 1 2 0 0 1 2]

print(data[**1**])

[3 2 0 2 2 1 2 0 2 0 3 0 2 2 3 0 2 0 1 3 1 0 0 1 3 1 1 0 2 2 3 1 2 0 0 3 1

1 1 2 1 3 3 3 0 3 3 0 1 2 0 3 2 0 1 3 0 0 3 2 1 2 1 3 2 0 1 2 2 1 2 0 1 3

1 2 2 0 3 0 0 1 2 1 0 0 0 3 2 1 1 1 1 3 0 1 2 3 1 2 0 1 0 0 2 0 1 2 1 1 0

3 3 2 1 2 3 3 2 3 0 3 0 3 0 2 3 0 1 3 3 3 0 1 1 3 2 3 2 0 1 2 1 3 3 2 0 1

3 3 3 3 0 2 0 3 2 2 2 0 2 0 0 3 1 3 0 2 3 0 2 0 3 3 0 3 2 2 1 2 3 1 1 3 1

1 1 1 1 0 1 2 2 3 1 0 2 2 1 0]

Compare

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

data = make\_blobs(n\_samples=**200,** n\_features=**2,** centers=**4,** cluster\_std=**1.8,** random\_state=**101**)

kmeans = KMeans(n\_clusters=**4,** n\_init=**10**)

kmeans.fit(data[**0**])

fig**,** (ax1**,** ax2) = plt.subplots(**1, 2,** sharey="col"**,** figsize=(**10, 6**))

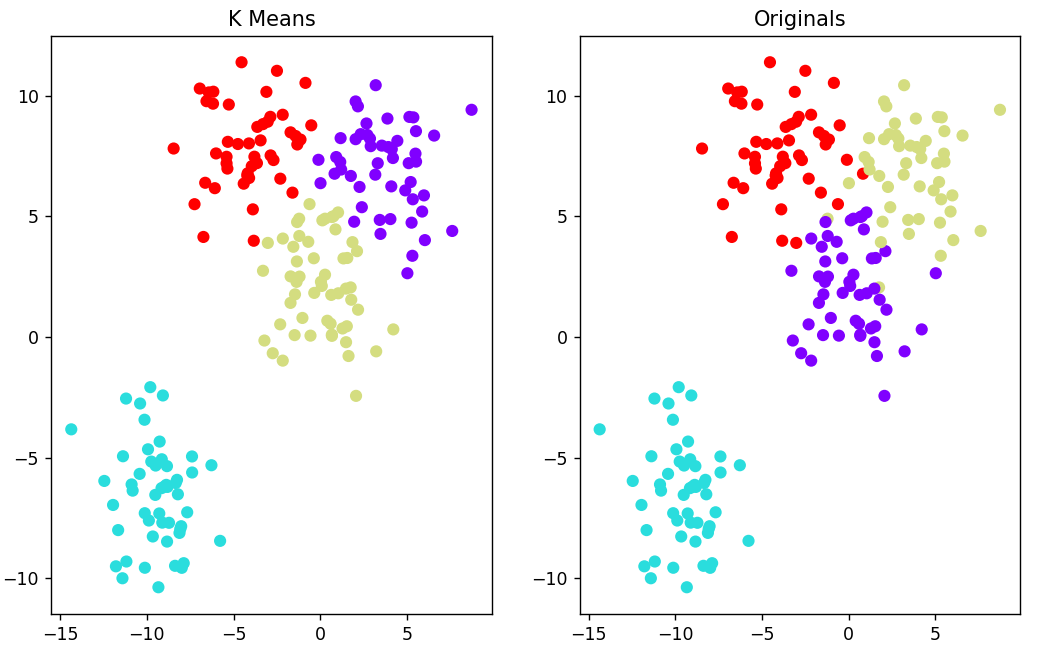
ax1.set\_title("K Means")

ax1.scatter(data[**0**][:**, 0**]**,** data[**0**][:**, 1**]**,** c=kmeans.labels\_**,** cmap="rainbow")

ax2.set\_title("Originals")

ax2.scatter(data[**0**][:**, 0**]**,** data[**0**][:**, 1**]**,** c=data[**1**]**,** cmap="rainbow")

plt.show()



import seaborn as sns

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cufflinks as cf

import plotly as py

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report

from sklearn.cluster import KMeans

df = pd.read\_csv("College\_Data")

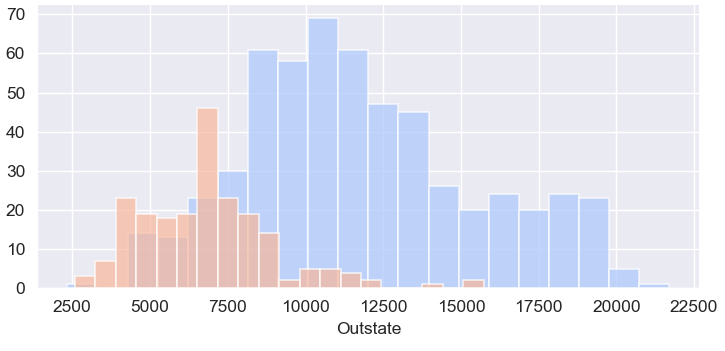
df['Grad.Rate']['Cazenovia College'] = **100**

sns.set\_style('darkgrid')

g = sns.FacetGrid(df**,**hue="Private"**,**palette='coolwarm'**,**aspect=**2**)

g = g.map(plt.hist**,**'Outstate'**,**bins=**20,**alpha=**0.7**)

plt.show()



Natural Language Processing

Imagine you work for Google News and you want to group news articles by topic or maybe you work for a legal firm and you need to sift through thousands of pages of legal documents to find the relevant ones. This is where natural language processing can help.

import nltk

import pandas as pd

# nltk.download\_shell()

messages = [line.strip() for line in open("SMSSpamCollection")]

for message\_no**,** message in enumerate(messages[:**5**]):

print(message\_no**,**message)

print()

0 ham Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...

1 ham Ok lar... Joking wif u oni...

2 spam Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's

3 ham U dun say so early hor... U c already then say...

4 ham Nah I don't think he goes to usf, he lives around here though

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import nltk

import pandas as pd

# nltk.download\_shell()

messages = pd.read\_csv("SMSSpamCollection"**,** sep="\t"**,** names=["label"**,**"message"])

print(messages.head())

label message

0 ham Go until jurong point, crazy.. Available only ...

1 ham Ok lar... Joking wif u oni...

2 spam Free entry in 2 a wkly comp to win FA Cup fina...

3 ham U dun say so early hor... U c already then say...

4 ham Nah I don't think he goes to usf, he lives aro...

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

print(messages.describe())

label message

count 5572 5572

unique 2 5169

top ham Sorry, I'll call later

freq 4825 30

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

print(messages.groupby("label").describe())

message

count unique top freq

label

ham 4825 4516 Sorry, I'll call later 30

spam 747 653 Please call our customer service representativ... 4

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

messages["length"] = messages["message"].apply(len)

print(messages.head())

label message length

0 ham Go until jurong point, crazy.. Available only ... 111

1 ham Ok lar... Joking wif u oni... 29

2 spam Free entry in 2 a wkly comp to win FA Cup fina... 155

3 ham U dun say so early hor... U c already then say... 49

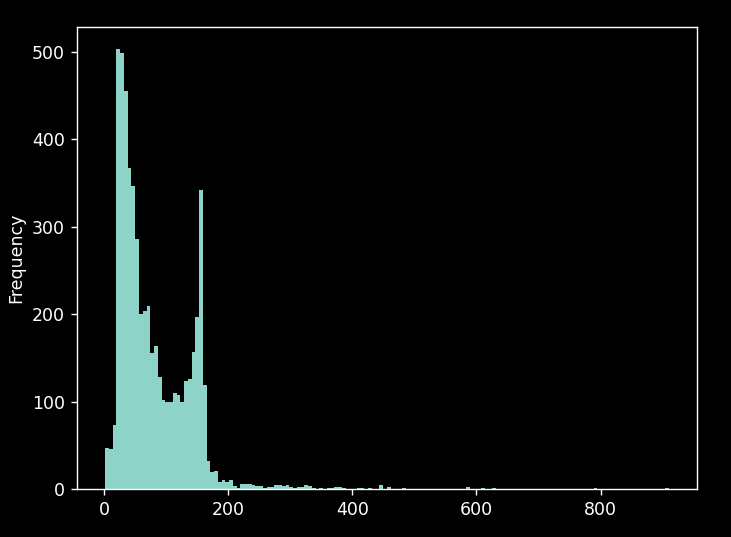
4 ham Nah I don't think he goes to usf, he lives aro... 61

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

plt.style.use("dark\_background")

messages["length"].plot.hist(bins=**150**)

plt.show()



-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

print(messages["length"].describe())

count 5572.000000

mean 80.489950

std 59.942907

min 2.000000

25% 36.000000

50% 62.000000

75% 122.000000

max 910.000000

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

zort = messages[messages["length"] == **910**]["message"].iloc[**0**]

print(zort)

For me the love should start with attraction.i should feel that I need her every time around me.she should be the first thing which comes in my thoughts.I would start the day and end it with her.she should be there every time I dream.love will be then when my every breath has her name.my life should happen around her.my life will be named to her.I would cry for her.will give all my happiness and take all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing the craziest things for her.love will be when I don't have to prove anyone that my girl is the most beautiful lady on the whole planet.I will always be singing praises for her.love will be when I start up making chicken curry and end up making sambar.life will be the most beautiful then.will get every morning and thank god for the day because she is with me.I would like to say a lot..will tell later..

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import nltk

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import style

# nltk.download\_shell()

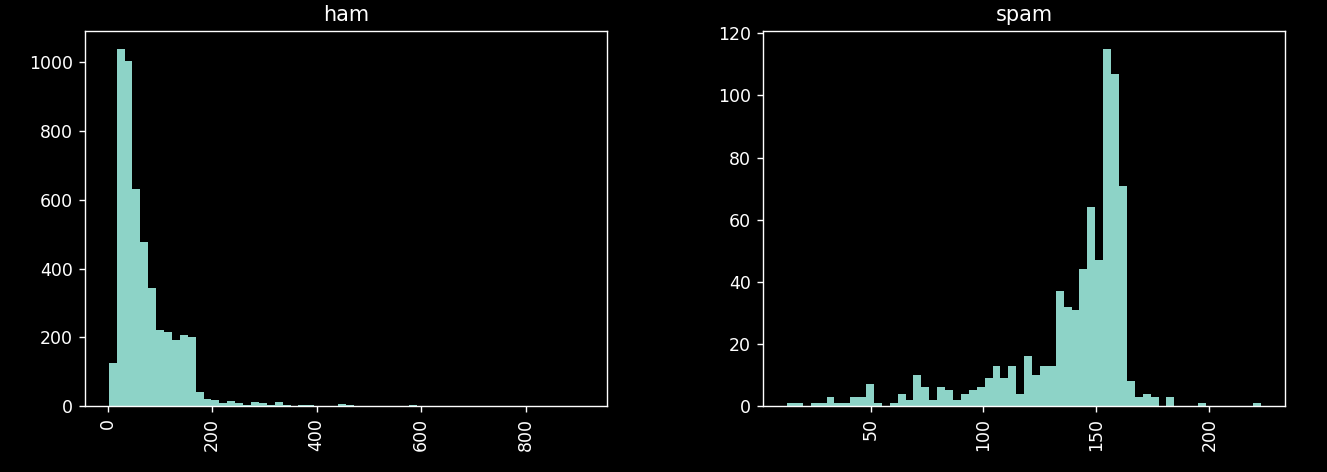
messages = pd.read\_csv("SMSSpamCollection"**,** sep="\t"**,** names=["label"**,** "message"])

messages["length"] = messages["message"].apply(len)

plt.style.use("dark\_background")

messages.hist(column="length"**,** by="label"**,** bins=**60,** figsize=(**12, 4**))

plt.show()



-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import nltk

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import style

import string

from nltk.corpus import stopwords

# nltk.download\_shell()

messages = pd.read\_csv("SMSSpamCollection"**,** sep="\t"**,** names=["label"**,** "message"])

messages["length"] = messages["message"].apply(len)

mess = "Sample message! Notice: it has punctuation."

def punc\_remover(sentence):

nopunc = []

for words in sentence:

if words in string.punctuation:

continue

else:

nopunc.append(words)

return "".join(nopunc)

print(punc\_remover(mess))

Sample message Notice it has punctuation

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

mess = "Sample message! Notice: it has punctuation."

def punc\_remover(sentence):

nopunc = []

for words in sentence:

if words in string.punctuation:

continue

else:

nopunc.append(words)

clean\_no\_punc = "".join(nopunc)

no\_stopwords = []

for words in clean\_no\_punc.split():

if words.lower() not in stopwords.words("english"):

no\_stopwords.append(words)

else:

continue

return " ".join(no\_stopwords)

print(punc\_remover(mess))

Sample message Notice punctuation

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

zort = messages["message"].apply(spammer)

print(zort.head())

0 Go jurong point crazy Available bugis n great ...

1 Ok lar Joking wif u oni

2 Free entry 2 wkly comp win FA Cup final tkts 2...

3 U dun say early hor U c already say

4 Nah dont think goes usf lives around though

Vectorization

import nltk

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import style

import string

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import CountVectorizer

# nltk.download\_shell()

messages = pd.read\_csv("SMSSpamCollection"**,** sep="\t"**,** names=["label"**,** "message"])

messages["length"] = messages["message"].apply(len)

def spammer(sentence):

nopunc = []

for words in sentence:

if words in string.punctuation:

continue

else:

nopunc.append(words)

clean\_no\_punc = "".join(nopunc)

no\_stopwords = []

for words in clean\_no\_punc.split():

if words.lower() not in stopwords.words("english"):

no\_stopwords.append(words)

else:

continue

return no\_stopwords

bow\_transformer = CountVectorizer(analyzer=spammer).fit(messages["message"])

mess4 = messages["message"][**3**]

print(mess4**,**"\n")

bow4 = bow\_transformer.transform([mess4])

print(bow4**,**"\n")

print(bow\_transformer.get\_feature\_names\_out()[**4068**])

U dun say so early hor... U c already then say...

(0, 4068) 2

(0, 4629) 1

(0, 5261) 1

(0, 6204) 1

(0, 6222) 1

(0, 7186) 1

(0, 9554) 2

U

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

bow\_transformer = CountVectorizer(analyzer=spammer).fit(messages["message"])

mess4 = messages["message"][**3**]

bow4 = bow\_transformer.transform([mess4])

messages\_bow = bow\_transformer.transform(messages)

tfidf4\_transformer = TfidfTransformer().fit(messages\_bow)

tfidf4 = tfidf4\_transformer.transform(bow4)

print(tfidf4)

(0, 9554) 0.554700196225229

(0, 7186) 0.2773500981126145

(0, 6222) 0.2773500981126145

(0, 6204) 0.2773500981126145

(0, 5261) 0.2773500981126145

(0, 4629) 0.2773500981126145

(0, 4068) 0.554700196225229

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

print(tfidf4\_transformer.idf\_[bow\_transformer.vocabulary\_["university"]])

2.386294361119891

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import nltk

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import style

import string

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

# nltk.download\_shell()

messages = pd.read\_csv("SMSSpamCollection"**,** sep="\t"**,** names=["label"**,** "message"])

messages["length"] = messages["message"].apply(len)

def spammer(sentence):

nopunc = []

for words in sentence:

if words in string.punctuation:

continue

else:

nopunc.append(words)

clean\_no\_punc = "".join(nopunc)

no\_stopwords = []

for words in clean\_no\_punc.split():

if words.lower() not in stopwords.words("english"):

no\_stopwords.append(words)

else:

continue

return no\_stopwords

bow\_transformer = CountVectorizer(analyzer=spammer).fit(messages["message"])

mess4 = messages["message"][**3**]

bow4 = bow\_transformer.transform([mess4])

messages\_bow = bow\_transformer.transform(messages["message"])

tfidf\_transformer = TfidfTransformer().fit(messages\_bow)

tfidf4 = tfidf\_transformer.transform(bow4)

messages\_tfidf = tfidf\_transformer.transform(messages\_bow)

spam\_detect\_model = MultinomialNB().fit(messages\_tfidf**,** messages["label"])

print('predicted:'**,** spam\_detect\_model.predict(tfidf4)[**0**])

print('expected:'**,** messages.label[**3**])

predicted: ham

expected: ham

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

all\_pred = spam\_detect\_model.predict(messages\_tfidf)

print(all\_pred)

['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']

Train

import nltk

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib import style

import string

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification\_report

# nltk.download\_shell()

messages = pd.read\_csv("SMSSpamCollection"**,** sep="\t"**,** names=["label"**,** "message"])

messages["length"] = messages["message"].apply(len)

def spammer(sentence):

nopunc = []

for words in sentence:

if words in string.punctuation:

continue

else:

nopunc.append(words)

clean\_no\_punc = "".join(nopunc)

no\_stopwords = []

for words in clean\_no\_punc.split():

if words.lower() not in stopwords.words("english"):

no\_stopwords.append(words)

else:

continue

return no\_stopwords

bow\_transformer = CountVectorizer(analyzer=spammer).fit(messages["message"])

mess4 = messages["message"][**3**]

bow4 = bow\_transformer.transform([mess4])

messages\_bow = bow\_transformer.transform(messages["message"])

tfidf\_transformer = TfidfTransformer().fit(messages\_bow)

tfidf4 = tfidf\_transformer.transform(bow4)

messages\_tfidf = tfidf\_transformer.transform(messages\_bow)

spam\_detect\_model = MultinomialNB().fit(messages\_tfidf**,** messages["label"])

all\_pred = spam\_detect\_model.predict(messages\_tfidf)

message\_train**,** message\_test**,** label\_train**,** label\_test = train\_test\_split(messages["message"]**,** messages["label"]**,** test\_size=**0.3**)

pipeline = Pipeline([("bow"**,** CountVectorizer(analyzer=spammer))**,** ("tfidf"**,** TfidfTransformer())**,** ("classifier"**,** MultinomialNB())])

pipeline.fit(message\_train**,** label\_train)

pip\_pre = pipeline.predict(message\_test)

print(classification\_report(label\_test**,**pip\_pre))

precision recall f1-score support

ham 0.96 1.00 0.98 1454

spam 1.00 0.73 0.84 218

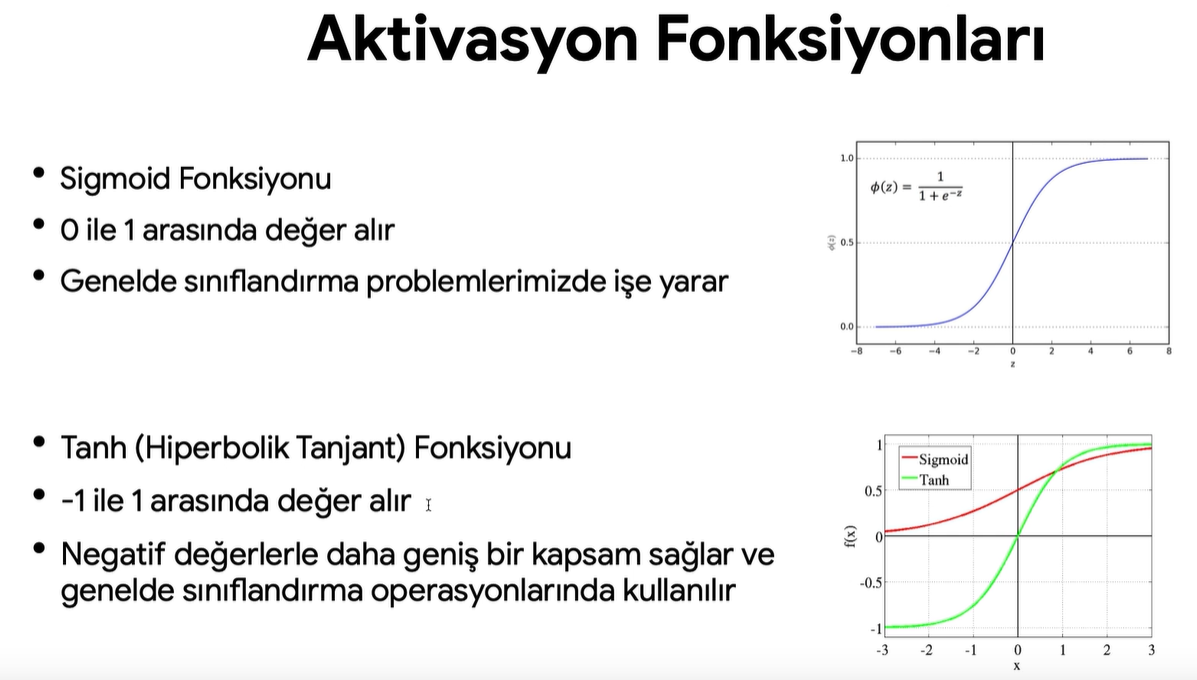
accuracy 0.96 1672

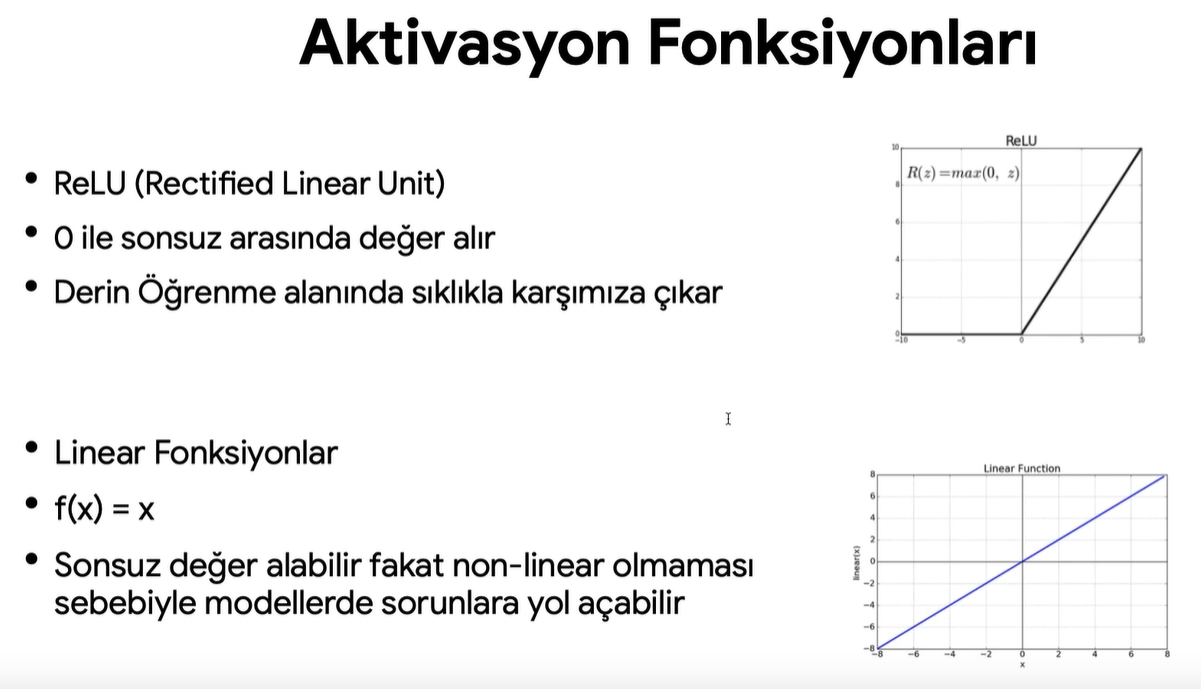
macro avg 0.98 0.86 0.91 1672

weighted avg 0.97 0.96 0.96 1672

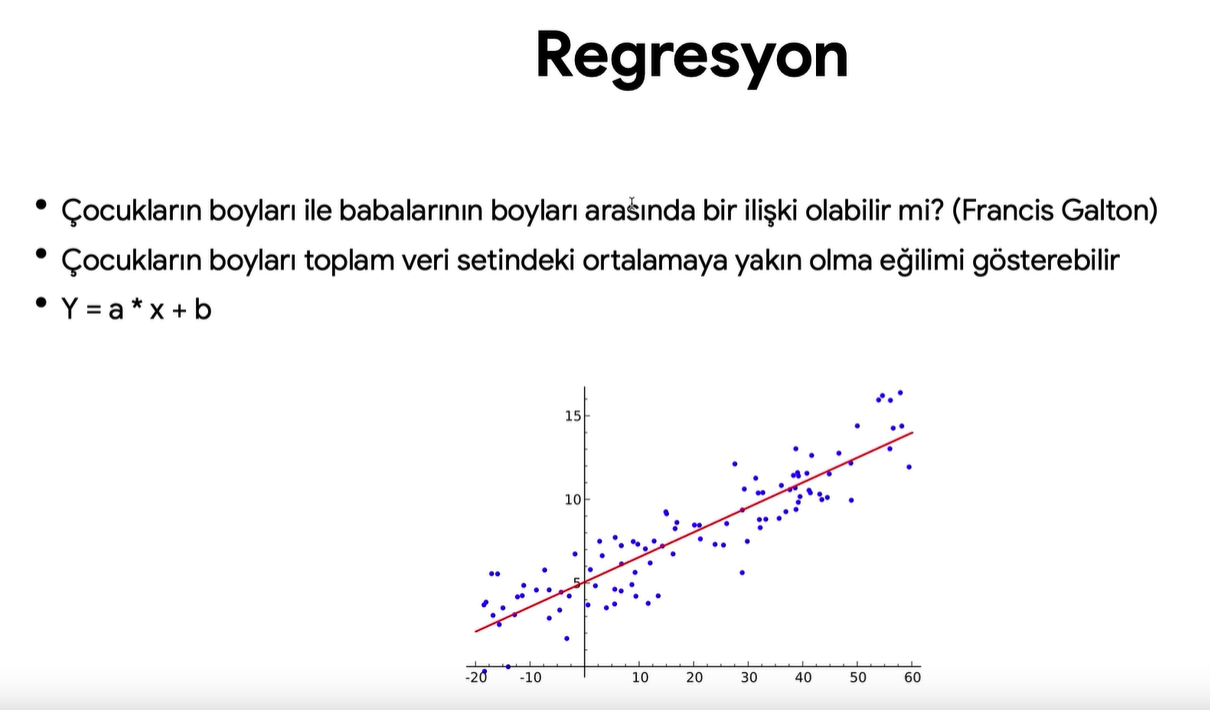
TensorFlow

Activation Functions

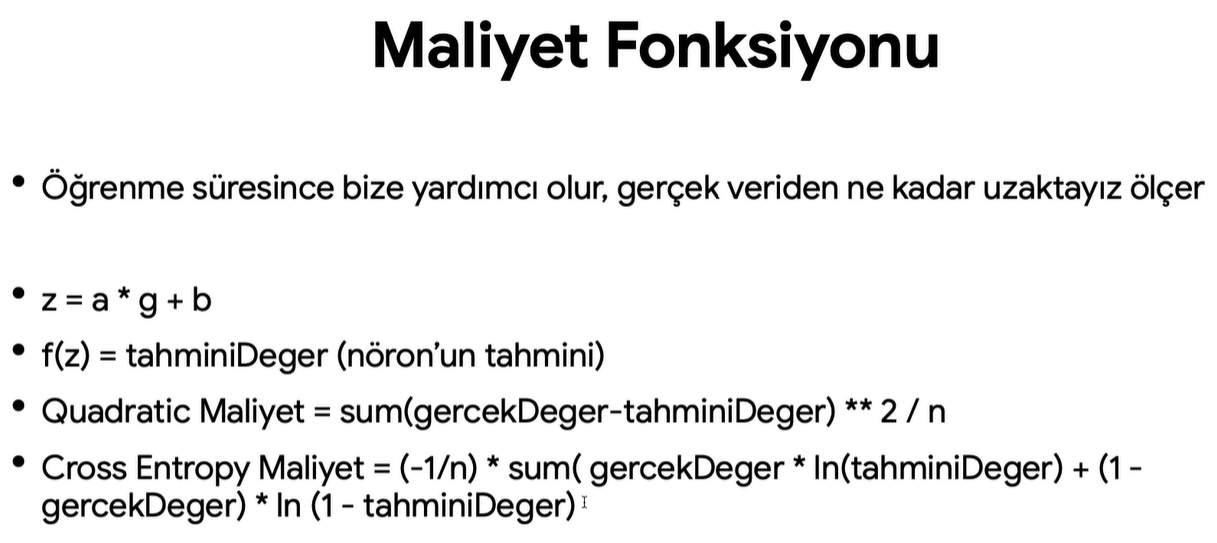




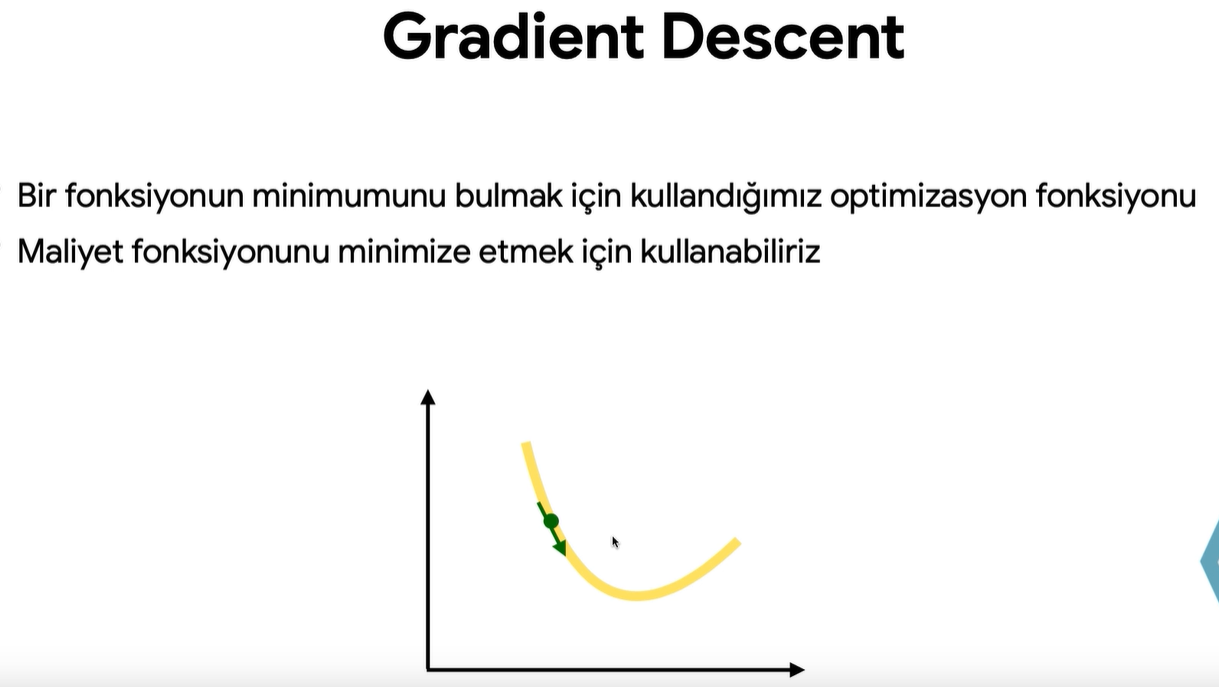
Regression

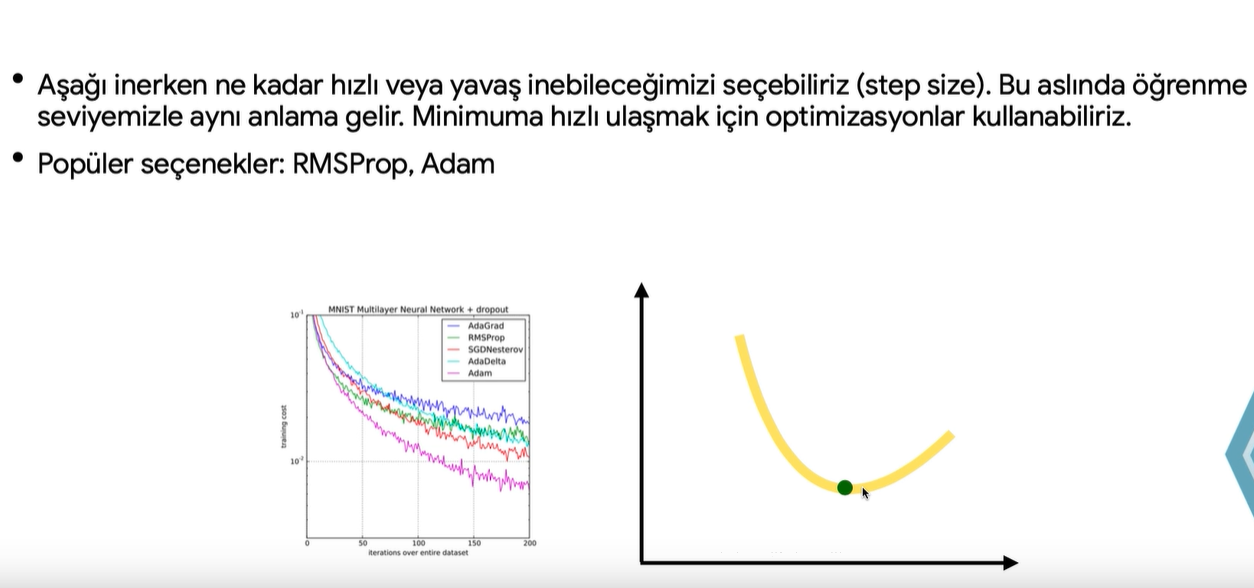


Cost Function



Gradient Descent





--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf

from tensorflow import keras

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import load\_model

# Load data

df = pd.read\_excel("bisiklet\_fiyatlari.xlsx")

# Split the data into X and y

X = df.drop("Fiyat", axis=1).values

y = df["Fiyat"].values

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=15)

# Scale the data

scaler = MinMaxScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

# Build the model

model = keras.Sequential()

model.add(keras.layers.Dense(4, activation="relu"))

model.add(keras.layers.Dense(4, activation="relu"))

model.add(keras.layers.Dense(4, activation="relu"))

model.add(keras.layers.Dense(1))

model.compile(optimizer="rmsprop", loss="mse")

# Train the model

model.fit(X\_train, y\_train, epochs=250)

# Evaluate the model on the training and test sets

loss = model.history.history["loss"]

print(df.describe(), "\n")

sns.lineplot(x=range(len(loss)), y=loss)

plt.show()

train\_loss = model.evaluate(x=X\_train, y=y\_train, verbose=0)

test\_loss = model.evaluate(x=X\_test, y=y\_test, verbose=0)

print(train\_loss)

print(test\_loss)

# Make predictions

test\_pred = model.predict(X\_test)

predicted\_Df = pd.DataFrame(y\_test, columns=["Real Y"])

predicted\_Df["Predicted Y"] = pd.Series(test\_pred.reshape(330,))

print(predicted\_Df)

sns.scatterplot(x="Real Y", y="Predicted Y", data=predicted\_Df)

plt.show()

# Print mean absolute error (MAE) and mean squared error (MSE)

print(mean\_absolute\_error(predicted\_Df["Real Y"], predicted\_Df["Predicted Y"]))

print(mean\_squared\_error(predicted\_Df["Real Y"], predicted\_Df["Predicted Y"]))

# Predict using new bicycle features

new\_biycle\_features = [[1753, 1751]]

new\_biycle\_features = scaler.transform(new\_biycle\_features)

print(model.predict(new\_biycle\_features))

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

This code starts with importing the required Python packages like numpy, pandas, matplotlib, seaborn, sklearn, and keras. The merc.xlsx file is read using pandas and stored in a dataframe named df. The correlation between the features and the price is obtained and stored in a descending order.

df is sorted in descending order based on the price feature. Rows from the top 131 are dropped since there are apparently outliers in the initial rows. The average price based on the year feature is obtained and stored in new\_df. The row where year=1970 is dropped since it seems to be an outlier. The transmission feature is removed from the dataframe using the drop() method.

Train and test data is split using the train\_test\_split() method from sklearn. The MinMaxScaler() method is used to scale the training and testing data. The multilayer perceptron (Dense Neural Network) model is defined using sequential models from Keras, there are 9 hidden dense layers in the model with the input layer. Each hidden dense layer uses the ReLU activation function. The loss is measured using the mean squared error and the optimization uses the Adam optimizer.

The model is fitted using the train data and validated using test data using mini-batch sizes of 300 and epoch number 200. The training and validation loss are plotted using the plot() function from Matplotlib.

The mean absolute error is calculated between the predicted and actual test sets. A scatter plot is made between actual and predicted prices using scatter() from matplotlib and a green line is also added using plot() to show where the predicted price ideally should be. To predict a new car price using the trained model, a row of data from the test set is taken as an example and the predict() method is used.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf

from tensorflow import keras

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import load\_model, Sequential

from keras.layers import Dense

# load data into pandas dataframe

df = pd.read\_excel("merc.xlsx")

# dropping the transmission column before calculating correlations with price

df.drop("transmission", axis=1).corr()["price"].sort\_values()

# Sorting values by price and then removing the top 131 entries. With the remaining data, dropping transmission column and grouping by year

new\_df = df.sort\_values("price", ascending=False).iloc[131:]

new\_df.drop("transmission", axis=1).groupby("year").mean()["price"]

# removing rows with year==1970

new\_df = new\_df[new\_df.year != 1970]

new\_df.drop("transmission", axis=1, inplace=True)

new\_df.groupby("year").mean()["price"]

# using train\_test\_split to split data into training and testing data

X = new\_df.drop("price", axis=1).values

y = new\_df["price"].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=10)

# scaling the X training and testing data using MinMaxScaler

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# building a neural network model with 9 hidden layers to predict car prices

model = Sequential()

model.add(Dense(50, activation="relu"))

model.add(Dense(50, activation="relu"))

model.add(Dense(50, activation="relu"))

model.add(Dense(50, activation="relu"))

model.add(Dense(100, activation="relu"))

model.add(Dense(100, activation="relu"))

model.add(Dense(100, activation="relu"))

model.add(Dense(75, activation="relu"))

model.add(Dense(75, activation="relu"))

model.add(Dense(1))

model.compile(optimizer="adam", loss="mse")

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), batch\_size=300, epochs=200)

# determining the errors in the model's predictions

loss\_data = pd.DataFrame(model.history.history)

loss\_data.plot()

plt.show()

predictions = model.predict(X\_test)

mean\_absolute\_error(y\_test, predictions)

mean\_squared\_error(y\_test, predictions)

# visualizing the model's predictions using a scatter plot

plt.scatter(y\_test,predictions)

plt.plot(y\_test,y\_test,"g")

plt.show()

new\_df.iloc[2]

# predicting the price of a new car stored as a pandas series

new\_car\_series = new\_df.drop("price",axis=1).iloc[2]

new\_car\_series = scaler.transform(new\_car\_series.values.reshape(-1,5))

model.predict(new\_car\_series)

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Classification

# Bu kod, bir veri setini yükleyip işleyerek bir sinir ağı modelini eğitir ve ardından sınıflandırma performansını değerlendirir.

# Bu kod bloğunda, kullanılacak olan Python kütüphaneleri içe aktarılır. Bu kütüphaneler, veri işleme, görselleştirme, makine öğrenimi ve derin öğrenme işlemleri için kullanılacak olan fonksiyonları sağlar.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf

from tensorflow import keras

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from keras.models import load\_model, Sequential

from keras.layers import Dense, Activation, Dropout

from keras.callbacks import EarlyStopping

from sklearn.metrics import classification\_report, confusion\_matrix

# Bu kod, "maliciousornot.xlsx" adlı Excel dosyasından veri setini yükler ve bir DataFrame nesnesine atar.

df = pd.read\_excel("maliciousornot.xlsx")

df.corr()["Type"].sort\_values()

sns.countplot(data=df, x="Type")

plt.show()

# Bu kod bloğu, "Type" özelliği ile diğer özellikler arasındaki korelasyonu hesaplar ve sıralar. Ardından, veri setindeki "Type" özelliğinin sınıf dağılımını çubuk grafiğiyle görselleştirir.

X = df.drop("Type", axis=1).values

y = df["Type"].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=15)

scaler = MinMaxScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

# Bu kod bloğunda, veri seti özellikleri (X) ve hedef değişken (y) ayrıştırılır. Ardından, veri seti, eğitim ve test kümelerine bölünür. Min-Max ölçeklendirme yöntemi kullanılarak özellik değerleri 0 ile 1 arasında ölçeklenir.

X\_train.shape # (383, 30)

model = Sequential()

model.add(Dense(units=30, activation="relu"))

model.add(Dense(units=15, activation="relu"))

model.add(Dense(units=15, activation="relu"))

model.add(Dense(units=1, activation="sigmoid"))

model.compile(loss="binary\_crossentropy", optimizer="adam")

model.fit(x=X\_train, y=y\_train, epochs=700,

validation\_data=(X\_test, y\_test), verbose=1)

# Bu kod bloğunda, keras kütüphanesini kullanarak bir sinir ağı modeli oluşturulur. Modelin katmanları ve aktivasyon fonksiyonları tanımlanır. Ardından, model derlenir ve eğitim verileri kullanılarak belirli bir sayıda epoch üzerinde eğitilir. Eğitim sırasında doğrulama verileri de kullanılır.

model\_loss = pd.DataFrame(model.history.history)

model\_loss.plot()

plt.show()

# Bu kod bloğu, eğitim sürecinde elde edilen kayıp (loss) değerlerini içeren bir DataFrame oluşturur ve bunu çizgi grafiği olarak görselleştirir.

model = Sequential()

model.add(Dense(units=30, activation="relu"))

model.add(Dense(units=15, activation="relu"))

model.add(Dense(units=15, activation="relu"))

model.add(Dense(units=1, activation="sigmoid"))

model.compile(loss="binary\_crossentropy", optimizer="adam")

early\_stop = EarlyStopping(

monitor="val\_loss", patience=25, mode="min", verbose=1)

model.fit(X\_train, y\_train, epochs=700, validation\_data=(

X\_test, y\_test), verbose=1, callbacks=[early\_stop])

# Bu kod bloğunda, aynı sinir ağı modeli bir kez daha oluşturulur ve eğitilir. Ayrıca, EarlyStopping geri çağrısı kullanılarak eğitim süreci erken durdurulabilir.

model\_loss2 = pd.DataFrame(model.history.history)

model\_loss2.plot()

plt.show()

# Bu kod bloğu, ikinci eğitim sürecinde elde edilen kayıp (loss) değerlerini içeren bir DataFrame oluşturur ve bunu çizgi grafiği olarak görselleştirir.

model = Sequential()

model.add(Dense(units=30, activation="relu"))

model.add(Dropout(rate=0.5))

model.add(Dense(units=15, activation="relu"))

model.add(Dropout(rate=0.5))

model.add(Dense(units=15, activation="relu"))

model.add(Dropout(rate=0.5))

model.add(Dense(units=1, activation="sigmoid"))

model.compile(loss="binary\_crossentropy", optimizer="adam")

model.fit(X\_train, y\_train, epochs=700, validation\_data=(

X\_test, y\_test), verbose=1, callbacks=[early\_stop])

# Bu kod bloğunda, dropout katmanları eklenerek regularizasyon uygulanan bir sinir ağı modeli oluşturulur ve eğitilir.

model\_loss3 = pd.DataFrame(model.history.history)

model\_loss3.plot()

plt.show()

# Bu kod bloğu, üçüncü eğitim sürecinde elde edilen kayıp (loss) değerlerini içeren bir DataFrame oluşturur ve bunu çizgi grafiği olarak görselleştirir.

predictions = model.predict(X\_test)

classes = np.argmax(predictions, axis=1)

print(classification\_report(y\_test, classes))

confusion\_matrix(y\_test, classes)

# Bu kod bloğunda, eğitilmiş model kullanılarak test verileri üzerinde tahminler yapılır. Ardından, sınıflandırma performansını değerlendirmek için sınıflandırma raporu (classification\_report) ve karışıklık matrisi (confusion\_matrix) hesaplanır ve ekrana yazdırılır.

# Bu kod, bir veri setinin yüklenmesi, ön işleme adımlarının uygulanması ve ardından sinir ağı modellerinin oluşturulması ve eğitimi ile performans değerlendirmesini içeren bir veri bilimi iş akışını temsil etmektedir. Her adımda yapılan işlemler açıklanmış ve sonuçlar görselleştirilmiştir.

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer

from sklearn import preprocessing

df = pd.read\_csv("veriler.csv")

df

# Here, an instance of the SimpleImputer class from scikit-learn is created. It is used to handle missing values (NaN values) in the data. It replaces missing values with the mean value of the respective column.

imputer = SimpleImputer(missing\_values=np.nan, strategy="mean")

# This line selects the values from columns 1 to 3 (index 1 to 3) in the DataFrame df and assigns them to the variable yas. The values attribute converts the selected data to a NumPy array. q

yas = df.iloc[:, 1:4].values

# The missing values in the yas array are replaced with the mean values using the imputer.transform method. The transformed array is then assigned back to yas.

imputer = imputer.fit(yas[:, 1:4])

yas[:, 1:4] = imputer.transform(yas[:, 1:4])

yas

# The missing values in the yas array are replaced with the mean values using the imputer.transform method. The transformed array is then assigned back to yas.

# This code selects the values from the first column (index 0) of the DataFrame df and assigns them to the variable ulke. The selected column represents the "country" data.

ulke = df.iloc[:, 0:1].values

print(ulke)

# A LabelEncoder instance named le is created. It is used to encode categorical variables into numerical values. The fit\_transform method of LabelEncoder is applied to the first column of df (representing the country data) to transform the country names into numerical labels. The transformed values are assigned back to ulke.

le = preprocessing.LabelEncoder()

ulke[:, 0] = le.fit\_transform(df.iloc[:, 0])

print(ulke)

# A OneHotEncoder instance named ohe is created. It is used to perform one-hot encoding on categorical variables. The fit\_transform method of OneHotEncoder is applied to ulke to transform the numerical labels into one-hot encoded representations. The resulting array is assigned back to ulke.

ohe = preprocessing.OneHotEncoder()

ulke = ohe.fit\_transform(ulke).toarray()

print(ulke)

# This line creates a pandas DataFrame called sonuc using the ulke array. The sonuc DataFrame has 22 rows and three columns named "fr", "tr", and "us". It contains the one-hot encoded country data.

sonuc = pd.DataFrame(data=ulke, index=range(22), columns=["fr", "tr", "us"])

print(sonuc)

# This line creates a pandas DataFrame called sonuc2 using the yas array. The sonuc2 DataFrame has 22 rows and three columns named "boy", "kilo", and "yas". It contains the imputed age data with missing values filled.

sonuc2 = pd.DataFrame(data=yas, index=range(

22), columns=["boy", "kilo", "yas"])

print(sonuc2)

# These lines assign the last column (gender data) of the df DataFrame to the variable cinsiyet. Then, it creates a pandas DataFrame called sonuc3 with 22 rows and one column named "cinsiyet" using the cinsiyet data. It contains the gender data.

cinsiyet = df.iloc[:, -1].values

sonuc3 = pd.DataFrame(data=cinsiyet, index=range(22), columns=["cinsiyet"])

print(sonuc3)

# This line creates a DataFrame called s by horizontally concatenating the sonuc and sonuc2 DataFrames. The concatenation is done along the columns (axis=1). The s DataFrame contains the one-hot encoded country data and the imputed age data.

s = pd.concat([sonuc, sonuc2], axis=1)

print(s)

# This line creates a DataFrame called final by horizontally concatenating the s and sonuc3 DataFrames. The concatenation is done along the columns (axis=1). The final DataFrame contains the one-hot encoded country data, imputed age data, and gender data.

final = pd.concat([s, sonuc3], axis=1)

print(final)

# These lines separate the independent variables (X) and the dependent variable (y) from the final DataFrame. The drop function is used to remove the "cinsiyet" column from the X DataFrame, creating X DataFrame with only the independent variables. The y DataFrame is defined with the dependent variable.

# Next, the data set is split into training and testing subsets using the train\_test\_split function. The test\_size=0.33 parameter splits the data into a test subset of 33% and uses the remaining data as the training subset. The random\_state=42 parameter is used to specify a seed value to obtain the same split when randomly dividing the data.

# Finally, the data is scaled using the StandardScaler. The fit\_transform method performs the scaling operation on the training data. The same scaling parameters are used to scale the test data.

# This code prepares the training and test data sets and performs data scaling after completing the data preprocessing steps.

X = final.drop("cinsiyet", axis=1)

y = final["cinsiyet"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.33, random\_state=42)

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.fit\_transform(X\_test)

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------