Introduction to Machine Learning

Homework 1
Due: Tuesday, 9/12, 5pm

Instructor:
Dr. Tom Arodz

HW1 - due 9/12, 5pm

- It involves a simple dataset with 2 classes (car vs. SUV) and 2 features (time 0-60mph, horsepower), 10 samples in total

	A	B	C	D	E
1	SampleName	ZeroToSixty	PowerHP	IsCar	IsSUV
2	c1	8.2	173	1	0
3	c2	8.4	176	1	0
4	c3	8.4	158	1	0
5	c4	9	142	1	0
6	c5	8.8	152	1	0
7	s1	8.7	189	0	1
8	s2	9.1	173	0	1
9	s3	8.6	194	0	1
10	s4	8.7	201	0	1
11	s5	8.7	198	0	1

normalized

	A	B	C	D	E
1	SampleNa	ZeroToSix	PowerHP	IsCar	IsSUV
2	c1	-1.6686	-0.12868	1	0
3	c2	-0.94312	0.019796	1	0
4	c3	-0.94312	-0.87105	1	0
5	c4	1.23331	-1.6629	1	0
6	c5	0.507833	-1.16799	1	0
7	s1	0.145095	0.663182	0	1
8	s2	1.596048	-0.12868	0	1
9	s3	-0.21764	0.910638	0	1
10	s4	0.145095	1.257077	0	1
11	s5	0.145095	1.108603	0	1

- Your task is to analyze the process of training a perceptron (the vector version, next slide)
  - $\mathbf{f(x_i)=sign(w_1 x_i^1 + w_2 x_i^2)}$
  - $w_{new}=w_{old} + c[y_i - f(x_i)]x_i$
- Python libraries to be used:
  - Pandas (reading in a csv file)
  - Numpy (storing vectors, doing the math with them)
  - Matplotlib (plotting diagrams of training progress)
  - ML libraries (e.g. sklearn, pytorch, tensforflow, others) not allowed

HW1 - due 9/12, 5pm

- **Data:** Two classes (y_i is -1 or +1),
  two features, i.e., 2D vector: x_i = (x_i^1 and x_i^2)
  Predictive model: $f(x_i) = \text{sign}(w^T x)$
  with two trainable parameters w = (w_1 , w_2)

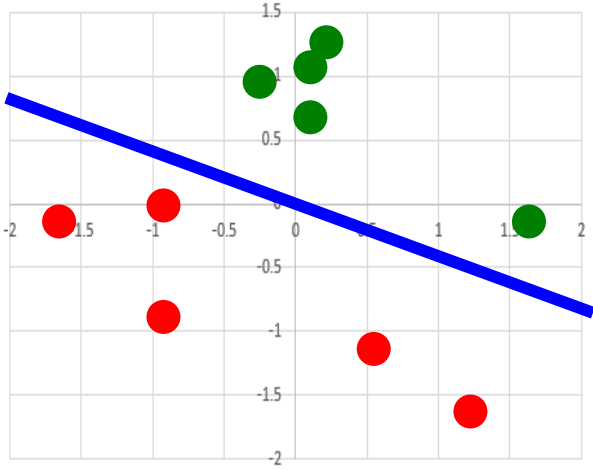
  - Set initial value of w_1 and w_2 (random, or a guess)
  - Repeat training epochs:
    - Loop over all 10 training samples once (one epoch):
      - Present a sample x_i and predict $f(x_i) = \text{sign}(w^T x)$
      - Compare true class y_i with predicted class $f(x_i)$
      - If prediction is right, go to next sample (i=i+1)
      - If prediction is wrong, update w
        - $w_{new} = w_{old} + c[y_i - f(x_i)]x_i$
    - Calculate current error rate:
      loop over all 10 training samples:
      - Present a sample x_i and predict $f(x_i) = \text{sign}(w^T x)$
      - Compare true class y_i with predicted class $f(x_i)$
      - Update the error rate if incorrect prediction

HW1 - due 9/12, 5pm

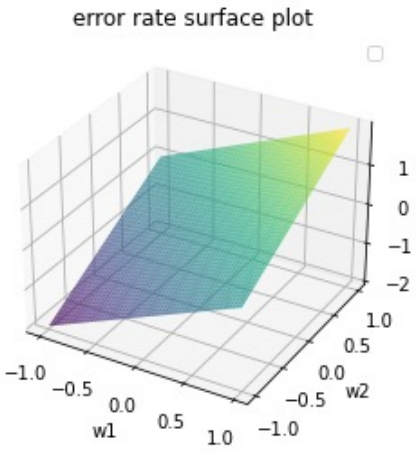
- It involves a simple dataset with 2 classes (car vs. SUV) and 2 features (time 0-60mph, horsepower), 10 samples in total
- Your task is to analyze the process of training a perceptron (the vector version)
  - $f(x_i)=sign(w_1 x_i^1 + w_2 x_i^2)$
  - $w_{new}=w_{old} + c[y_i - f(x_i)]x_i$
- Specifically:
  - Use pandas to import **carSUV_normalized.csv** file (file is in Canvas)
  - Extract numpy 2D (10 x 2) array X of features from pandas
  - Extract classes (10D vector Y) from pandas dataframe, convert them from 0/1 to -1/1
  - Pick a learning rate "c", if it's not working, try smaller (or larger), e.g. orders of 10 (…, 100, 10, 1, .1, .01, .001, …)
  - Run the perceptron algorithm as described above,
    - One epoch: presenting each of the 10 samples and updating the weights if needed after each sample
    - Perform a large number of epochs, until the model no longer improves
    - Use "error rate" as the measure of model quality:
      - At the end of each epoch, do prediction with current weights for each of the 10 samples, count how many are wrong, divide by total number of training samples (i.e., 10)

HW1 - due 9/12, 5pm

- Upload your submission through Canvas
  Provide:
  - Code (as jupyter-notebook file)
    for reading data and training the model
  - Final "error rate" and the "w_1 & w_2"
    resulting from training
  - Two plots:
    - 2D scatter plot with graphical depiction
      of the final classification model
      - with the two features as axes
      - with samples as points (use red for cars, green for SUV)
      - with decision line (based on your w_1,w_2) in blue

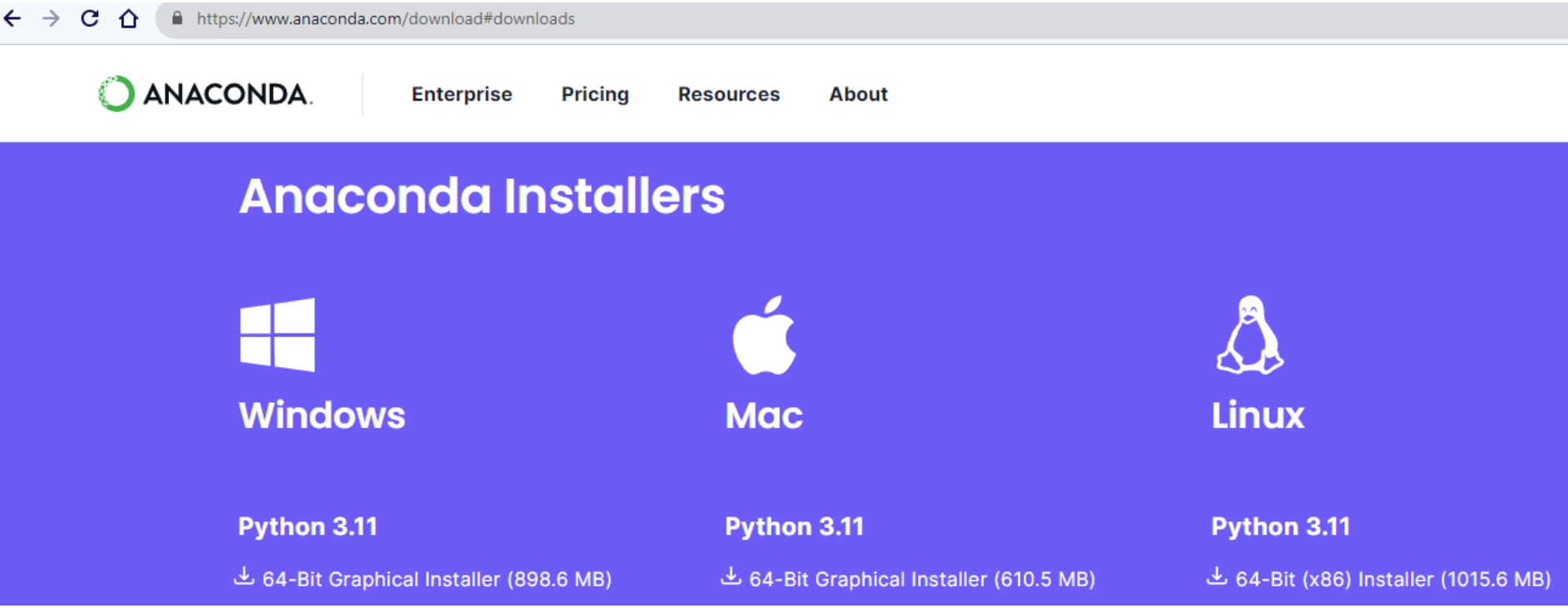


    - 2D surface plot of the "error rate" for all possible values
      of w1, w2 (on a grid, with very small grid step size)
      - "error rate" (values in [0,1]):
        number of incorrectly predicted samples
        divided by total number of training samples

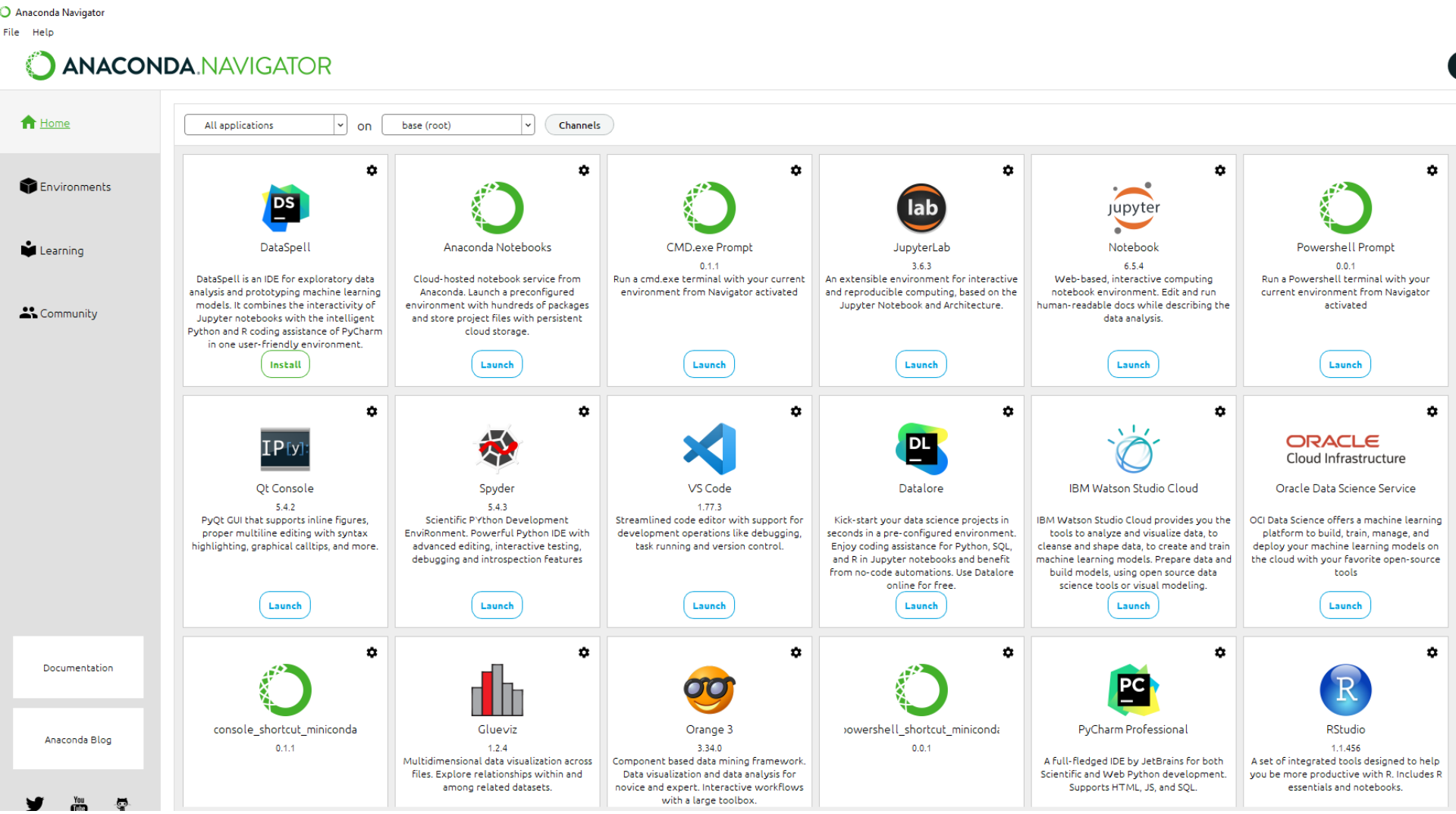


Python – brief into

Anaconda: Python 3 distribution tailored for machine learning and data science programming

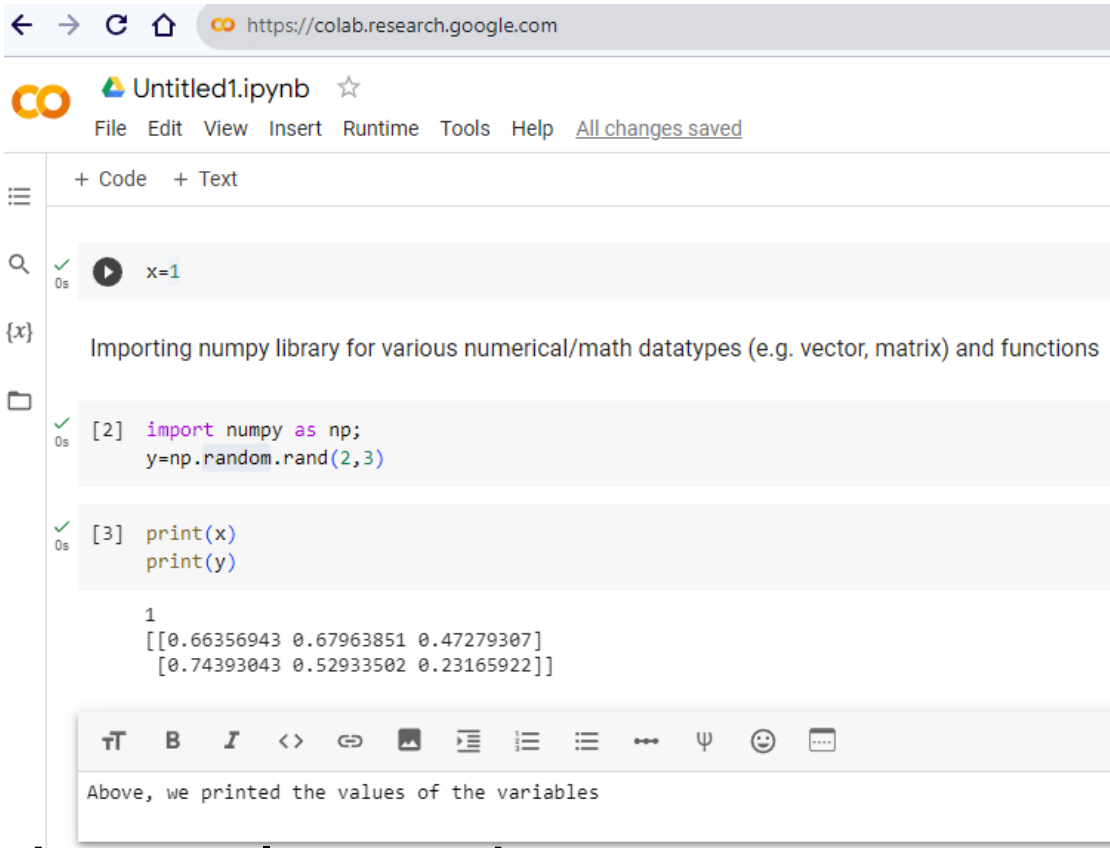


Anaconda Navigator (Windows)



Coding in python

Jupyter-notebook: combines text and code, runs in the browser

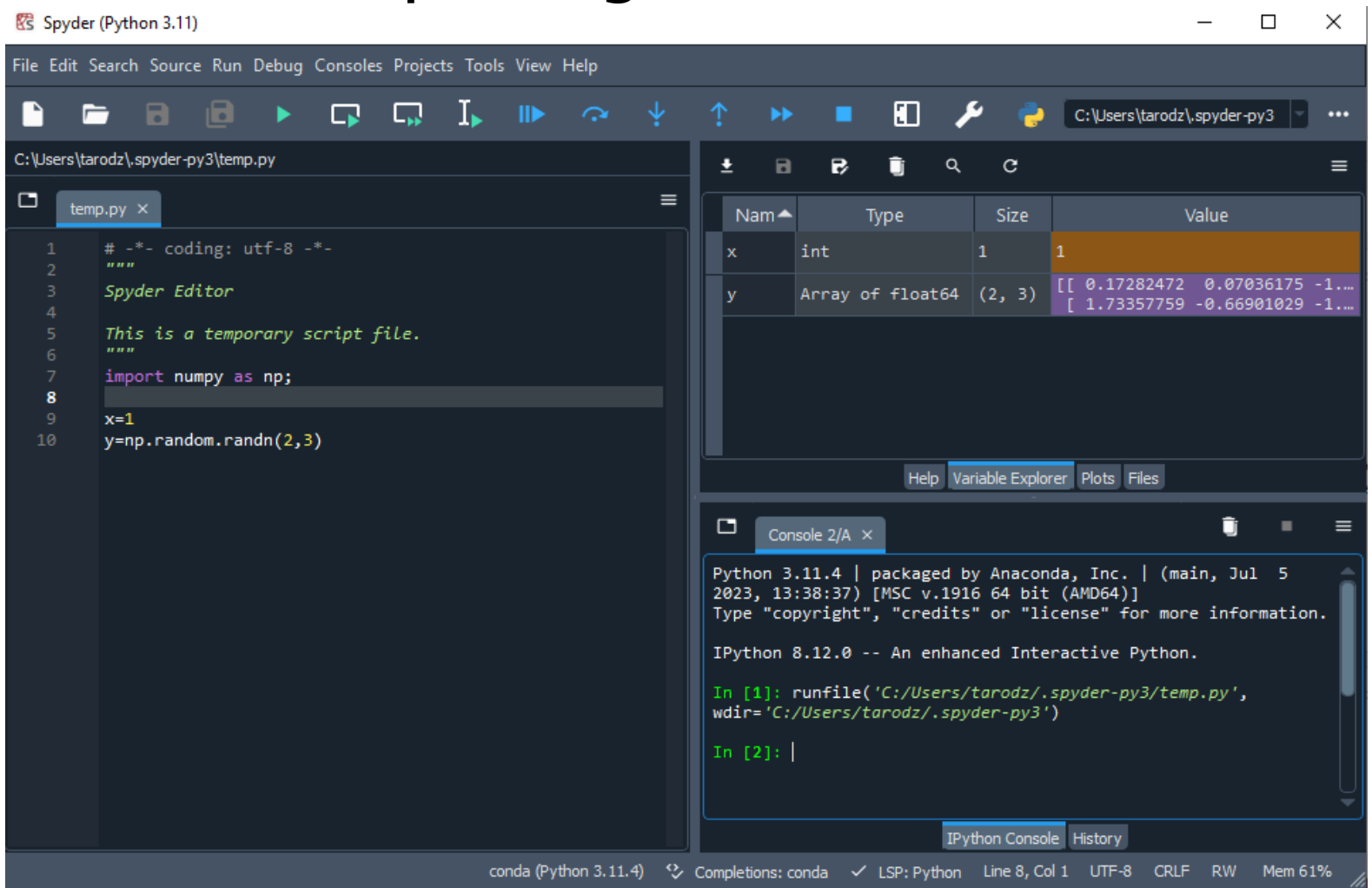


Can run locally
on your machine
(e.g. on Windows:
from Navigator)

Can run online
e.g. google colab:
https://colab.research.google.com/

Coding in python

Spyder: a Matlab-like layout, with panels for plots and inspecting data



Python intro

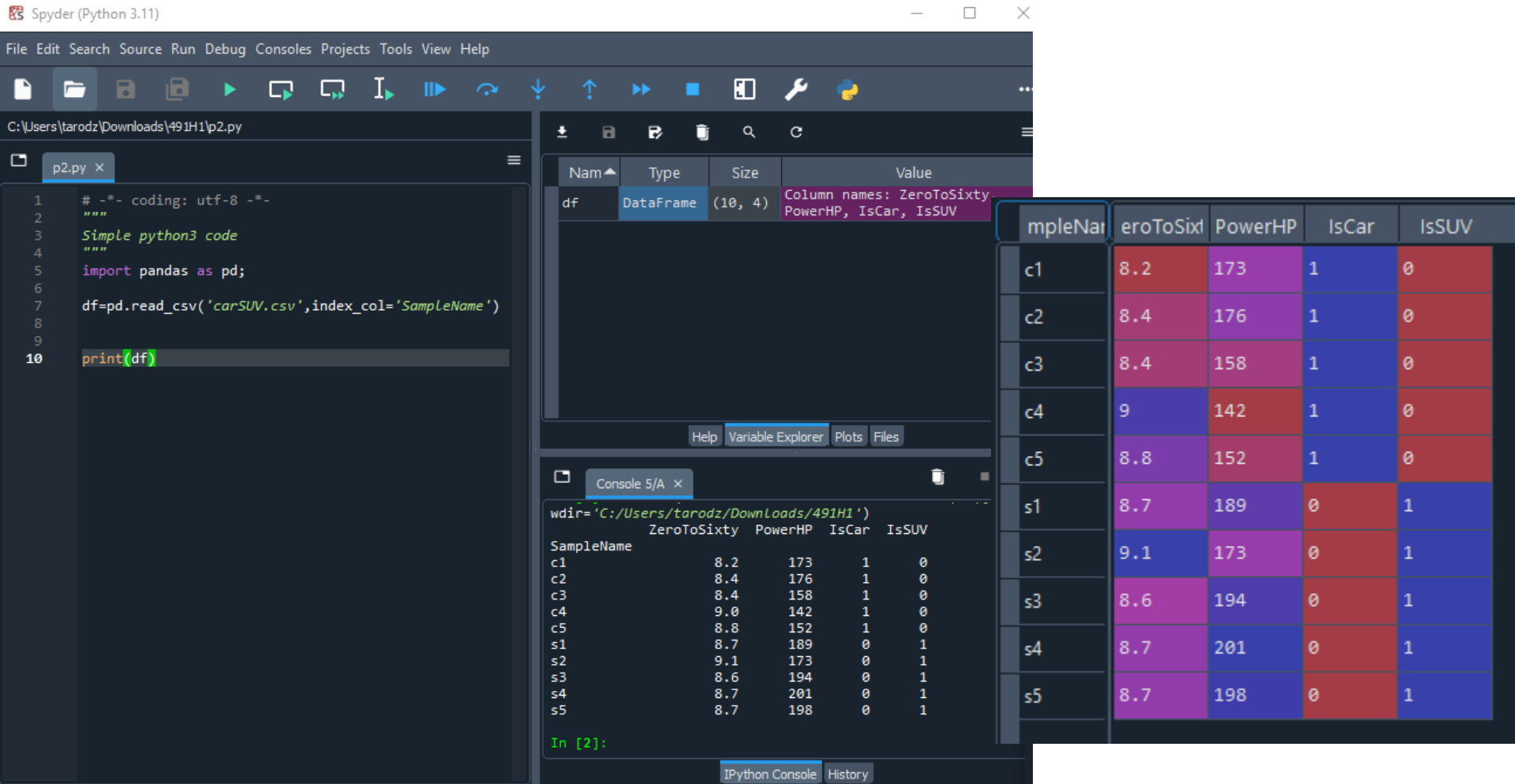
- Interpreted, no sperate compilation step
- Execution starts from line one (like a script)
- Variables do not need declaration and don't need type specification
- [a,b,c] is a list, (a,b,c) is a tuple
- There is automatic garbage collection
- Has standard concepts like functions (can return multiple values), loops, etc.
- Most of useful functionality is provided in libraries (e.g., numpy)
  - Libraries need to be imported, and can be given a shortcut name

```
import numpy as np;

x=1.1

y=0
for index in range(1,5):
    y=y+index

z='one'


def make2Dvector(w1,w2):
    w=np.zeros((2,1));
    w[0]=w1
    w[1]=w2
    return w

w=make2Dvector(0.1,-0.2)

print(x,type(x))
print(y,type(y))
print(z,type(z))
print(w,type(w))
```

```
1.1 <class 'float'>
10 <class 'int'>
one <class 'str'>
[[ 0.1]
 [-0.2]] <class 'numpy.ndarray'>
```

Pandas

- Provides data import and manipulation functionality (a bit like R)
- Main data type is “DataFrame”
  - Basically, a 2D array of data, but with column names, row names, various data types in the same array



Pandas

- Provides data import and manipulation functionality (a bit like R)
- Main data type is “DataFrame”
  - Basically, a 2D array of data, but with column names, row names, various data types in the same array

```
import pandas as pd;

df=pd.read_csv('carSUV.csv',index_col='SampleName’)

#can do all kinds of data analytics
print(df.mean())
```

printed:

```
ZeroToSixty      8.66
PowerHP        175.60
IsCar            0.50
IsSUV            0.50
```

```
# extract raw data as numpy matrix
myDataAsNumpyMatrix = df.to_numpy();
```

Numpy

- Vectors and matrix variables, reshaping (e.g., transposing) them
- Arithmetic and functions on vectors
- Much faster than doing math in raw python

```
import numpy as np;

# a two-by-one matrix (a vector)
w = np.ones((2,1))
print("shape of w",w.shape)

x = np.array( [[0.1,],[0.2,]])

print("x before",x)
x = 2*x-0.1
print("x after",x)

wT = np.transpose(w)

wTx = np.matmul(wT,x)
# compare with
wElementwiseX = np.multiply(w,x)

#squeeze() removes "useless" dimensions
# 1-by-1 matrix becomes just one number
wTxAsNumber = np.squeeze(wTx)
```

```
x before [[0.1]
 [0.2]]
x after [[0.1]
 [0.3]]
```

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$ ← 2D Vector (2x1)

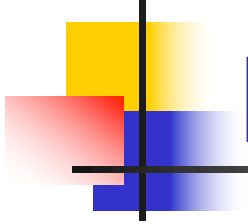
$$\mathbf{x} = \begin{bmatrix} x^1 \\ x^2 \end{bmatrix}$$ ← 2D Vector (2x1)

$$\mathbf{w}^T = [w_1, w_2]$$ ← Transposed vector (1x2)

$$\mathbf{w}^T\mathbf{x} = [w_1, w_2]\begin{bmatrix} x^1 \\ x^2 \end{bmatrix} = w_1x^1 + w_2x^2$$

Matrix multiplication

Name	Type	Size	Value
w	Array of float64	(2, 1)	[[1.] [1.]]
wElementwiseX	Array of float64	(2, 1)	[[0.1] [0.3]]
wT	Array of float64	(1, 2)	[[1. 1.]]
wTx	Array of float64	(1, 1)	[[0.4]]
wTxAsNumber	Array of float64	1	0.4
x	Array of float64	(2, 1)	[[0.1] [0.3]]



Matplotlib

Creates plots from data

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import Normalize

def plot_function_on_grid(function_to_plot, datasetX, datasetY):
    # Create a meshgrid
    w1 = np.arange(-1, 1, 0.01)
    w2 = np.arange(-1, 1, 0.01)
    W1, W2 = np.meshgrid(w1, w2)
    error_rates_values_for_W1W2 = function_to_plot(W1, W2, datasetX, datasetY)

    # Create a figure and a 3D axis
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Create the surface plot
    surface = ax.plot_surface(W1, W2, error_rates_values_for_W1W2, cmap='viridis', al

    # Add labels and title
    ax.set_xlabel('w1')
    ax.set_ylabel('w2')
    ax.set_zlabel('error rate')
    ax.set_title('error rate surface plot')

    return ax;
```

Matplotlib

Creates plots from data

```
def function_error_rate(w1, w2, datasetX, datasetY):
    # here add calculation of "error rate" for specific w1,w2 for the whole dataset
    return w1+w2
```

```
# plot function surface
ax = plot_function_on_grid(function_error_rate, None, None);
plt.legend()
plt.show()
```