Mid Journey V4 prompt: desktop, 32-inch monitor, workbench, programming environment, incredible detail, lots of detail --ar 3:2 --q 2 --upbeta --niji

Lecture 1-2: Programming Environment

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

What Is a Programming Environment?

I. Why Python?

II. Jupyter Notebook

III. Git and GitHub

IV. What is Unix shell?

V. Integrated Development Environment (IDE)

I. Why Python?

1. Interpreted vs. Compiled language

Interpreted language:

- **Examples: Python, Perl, R, Matlab**
- The interpreter executes the codes **line by line.**
- Faster **development process** as you can validate the results without compiling the entire program.
- Suitable for exploring new ideas or developing lightweight applications

I. Why Python?

1. Interpreted vs. Compiled language

Interpreted language:

- **Examples: Python, Perl, R, Matlab**
- The interpreter executes the codes **line by line.**
- Faster **development process** as you can validate the results without compiling the entire program.
- Suitable for exploring new ideas or developing lightweight applications

Compiled language:

- **Examples: C/C++, Java, Fortran, Visual Basic**
- The **entire program** will be compiled and built before the execution.
- Faster **execution time**.
- Suitable for developing large and complicated applications

I. Why Python?

2. Dynamic vs. Static typing

Python is a **dynamically-typed language**: A Python variable DOES NOT have a static type. A data type refers to an integer, a string (text), or a floating number.

I. Why Python?

2. Dynamic vs. Static typing

Python is a **dynamically-typed language**: A Python variable DOES NOT have a static type. A data type refers to an integer, a string (text), or a floating number.

Pros:

Higher flexibility for development

Cons:

Slower and less efficient (memory-wise) as the interpreter has to determine the type of a variable at runtime

I. Why Python?

2. Dynamic vs. Static typing

Python is a **dynamically-typed language**: A Python variable DOES NOT have a static type. A data type refers to an integer, a string (text), or a floating number.

Pros:

Higher flexibility for development

Cons:

Slower and less efficient (memory-wise) as the interpreter has to determine the type of a variable at runtime

```
x = 3
print(x) # 3
x = "hello"
print(x) # hello
```

I. Why Python?

2. Dynamic vs. Static typing

Python is a dynamically-typed language

C is a statically-typed language

```
1  def main():
2      x = 5
3      print("x is %d" % x)
4      return 0
```

I. Why Python?

2. Dynamic vs. Static typing

Python is a dynamically-typed language

C is a statically-typed language

```python
def main():
    x = 5
    print("x is %d" % x)
    return 0
```

```python
def main() -> int:
    x: int = 5
    print("x is %d" % x)
    return 0
```

I. Why Python?

2. Dynamic vs. Static typing

Python is a dynamically-typed language

C is a statically-typed language

```python
def main():
    x = 5
    print("x is %d" % x)
    return 0
```

The two statements are equivalent. The typing information is only an annotation to humans

```python
def main() -> int:
    x: int = 5
    print("x is %d" % x)
    return 0
```

Indicate that the function will return an integer

I. Why Python?

2. Dynamic vs. Static typing

Python is a dynamically-typed language

C is a statically-typed language

```python
def main():
    x = 5
    print("x is %d" % x)
    return 0
```

The two statements are equivalent. The typing information is only an annotation to humans

```python
def main() -> int:
    x: int = 5
    print("x is %d" % x)
    return 0
```

Indicate that the function will return an integer

```c
#include <stdio.h>

int main() {
    int x = 5;
    printf("x = %d", x);
    return 0;
}
```

I. Why Python?

2. Dynamic vs. Static typing

Python is a dynamically-typed language

C is a statically-typed language

```python
def main():
    x = 5
    print("x is %d" % x)
    return 0
```

The two statements are equivalent. The typing information is only an annotation to humans

```python
def main() -> int:
    x: int = 5
    print("x is %d" % x)
    return 0
```

Indicate that the function will return an integer

```c
#include <stdio.h>

int main() {
    int x = 5;
    printf("x = %d", x);
    return 0;
}
```

the main() function MUST return an integer value

int: specify the variable type as an integer

I. Why Python?

3. Large and active community

Rich resources are available to help you with your problems

I. Why Python?

3. Large and active community

Rich resources are available to help you with your problems

↓

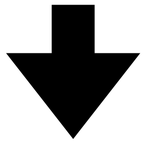
Wide range of libraries and frameworks

So that you don’t need to code everything yourself

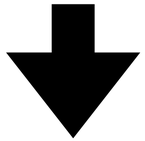
I. Why Python?

3. Large and active community

Rich resources are available to help you with your problems



Wide range of libraries and frameworks



So that you don’t need to code everything yourself

Scientific computation

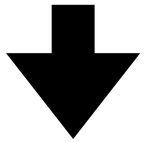
Visualization

Machine learning

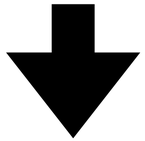
I. Why Python?

3. Large and active community

Rich resources are available to help you with your problems



Wide range of libraries and frameworks

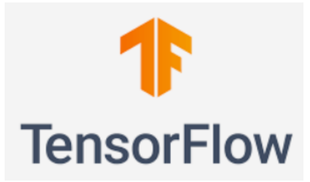


So that you don’t need to code everything yourself

Scientific computation

Visualization

Machine learning



I. Why Python?

3. Large and active community

https://numpy.org/doc/stable/index.html

https://pandas.pydata.org/docs/index.html

https://docs.scipy.org/doc/scipy/index.html

https://scikit-learn.org/stable/index.html

https://bokeh.org/

https://plotly.com/

https://opencv.org/

https://openai.com/

https://www.tensorflow.org/

https://pytorch.org/

I. Why Python?

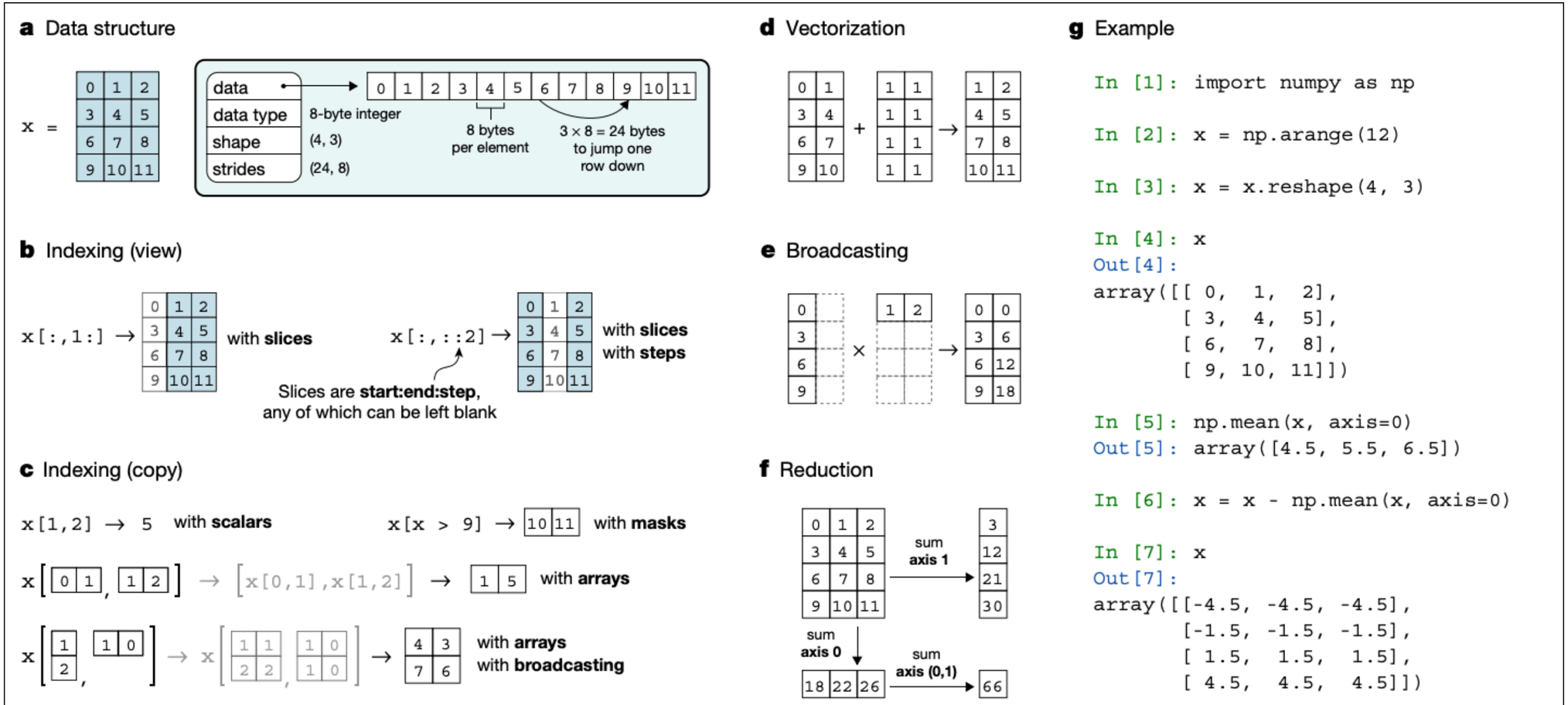
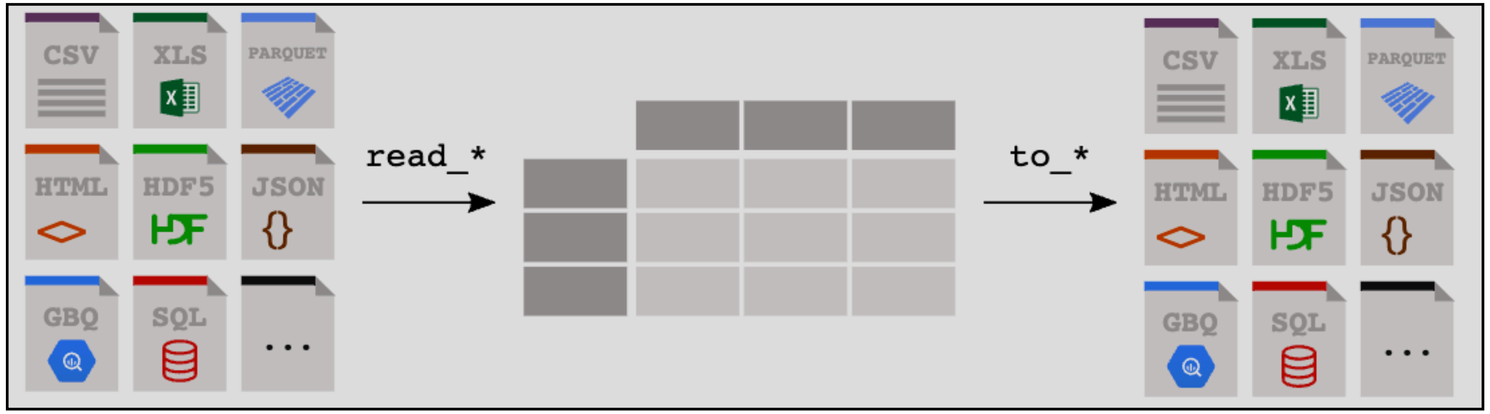


Fig. 1 | The NumPy array incorporates several fundamental array concepts. **a**, The NumPy array data structure and its associated metadata fields. **b**, Indexing an array with slices and steps. These operations return a 'view' of the original data. **c**, Indexing an array with masks, scalar coordinates or other arrays, so that it returns a 'copy' of the original data. In the bottom example, an array is indexed with other arrays; this broadcasts the indexing arguments before performing the lookup. **d**, Vectorization efficiently applies operations to groups of elements. **e**, Broadcasting in the multiplication of two-dimensional arrays. **f**, Reduction operations act along one or more axes. In this example, an array is summed along select axes to produce a vector, or along two axes consecutively to produce a scalar. **g**, Example NumPy code, illustrating some of these concepts.

https://www.nature.com/articles/s41586-020-2649-2

I. Why Python?

pandas




```
In [3]: titanic
Out[3]:
     PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0              1         0       3  ...   7.2500   NaN         S
1              2         1       1  ...  71.2833   C85         C
2              3         1       3  ...   7.9250   NaN         S
3              4         1       1  ...  53.1000  C123         S
4              5         0       3  ...   8.0500   NaN         S
..           ...       ...     ...  ...      ...   ...       ...
886          887         0       2  ...  13.0000   NaN         S
887          888         1       1  ...  30.0000   B42         S
888          889         0       3  ...  23.4500   NaN         S
889          890         1       1  ...  30.0000  C148         C
890          891         0       3  ...   7.7500   NaN         Q

[891 rows x 12 columns]
```

```
In [9]: titanic.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

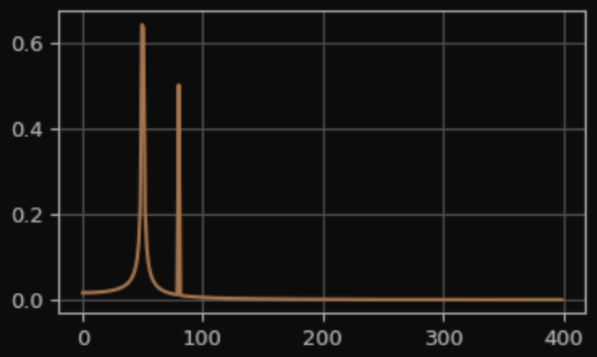
https://pandas.pydata.org/docs/getting_started/intro_tutorials/02_read_write.html

I. Why Python?

SciPy

1-D discrete Fourier transforms

```
>>> from scipy.fft import fft, fftfreq
>>> import numpy as np
>>> # Number of sample points
>>> N = 600
>>> # sample spacing
>>> T = 1.0 / 800.0
>>> x = np.linspace(0.0, N*T, N, endpoint=False)
>>> y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
>>> yf = fft(y)
>>> xf = fftfreq(N, T)[:N//2]
>>> import matplotlib.pyplot as plt
>>> plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
>>> plt.grid()
>>> plt.show()
```



Finding the inverse

The inverse of a matrix $\mathbf{A}$ is the matrix $\mathbf{B}$, such that $\mathbf{AB} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix consisting of ones down the main diagonal. Usually, $\mathbf{B}$ is denoted $\mathbf{B} = \mathbf{A}^{-1}$. In SciPy, the matrix inverse of the NumPy array, A, is obtained using `linalg.inv (A)`, or using `A.I` if `A` is a Matrix. For example, let

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix},$$

then

$$\mathbf{A}^{-1} = \frac{1}{25} \begin{bmatrix} -37 & 9 & 22 \\ 14 & 2 & -9 \\ 4 & -3 & 1 \end{bmatrix} = \begin{bmatrix} -1.48 & 0.36 & 0.88 \\ 0.56 & 0.08 & -0.36 \\ 0.16 & -0.12 & 0.04 \end{bmatrix}.$$

The following example demonstrates this computation in SciPy

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1,3,5],[2,5,1],[2,3,8]])
>>> A
array([[1, 3, 5],
       [2, 5, 1],
       [2, 3, 8]])
>>> linalg.inv(A)
array([[-1.48,  0.36,  0.88],
       [ 0.56,  0.08, -0.36],
       [ 0.16, -0.12,  0.04]])
>>> A.dot(linalg.inv(A)) #double check
array([[  1.00000000e+00,  -1.11022302e-16,  -5.55111512e-17],
       [  3.05311332e-16,   1.00000000e+00,   1.87350135e-16],
       [  2.22044605e-16,  -1.11022302e-16,   1.00000000e+00]])
```

https://docs.scipy.org/doc/scipy/tutorial/fft.html

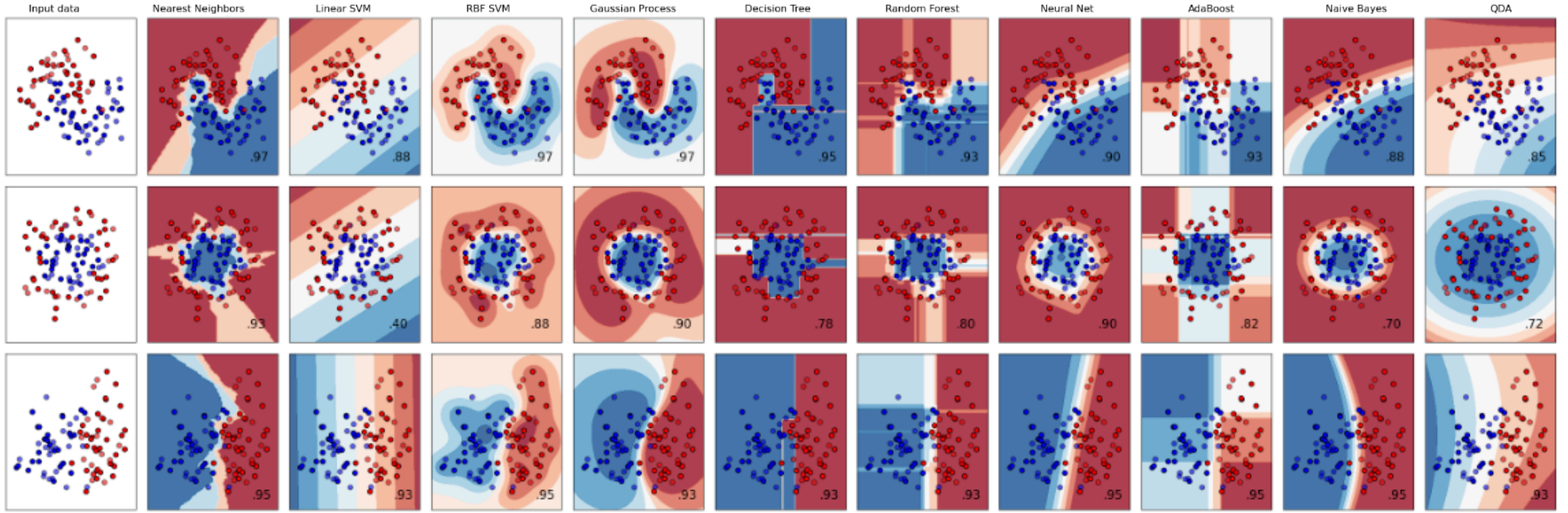
I. Why Python?

Classifier comparison

A comparison of a several classifiers in scikit-learn on synthetic datasets. The point of this example is to illustrate the nature of decision boundaries of different classifiers. This should be taken with a grain of salt, as the intuition conveyed by these examples does not necessarily carry over to real datasets.

Particularly in high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as naive Bayes and linear SVMs might lead to better generalization than is achieved by other classifiers.

The plots show training points in solid colors and testing points semi-transparent. The lower right shows the classification accuracy on the test set.

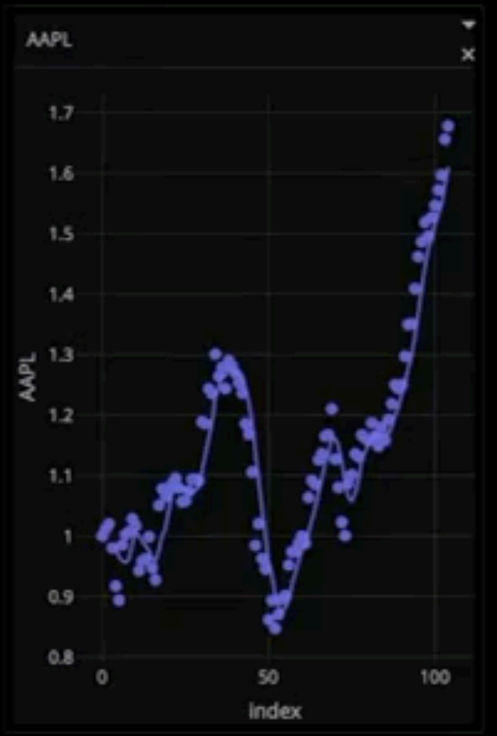


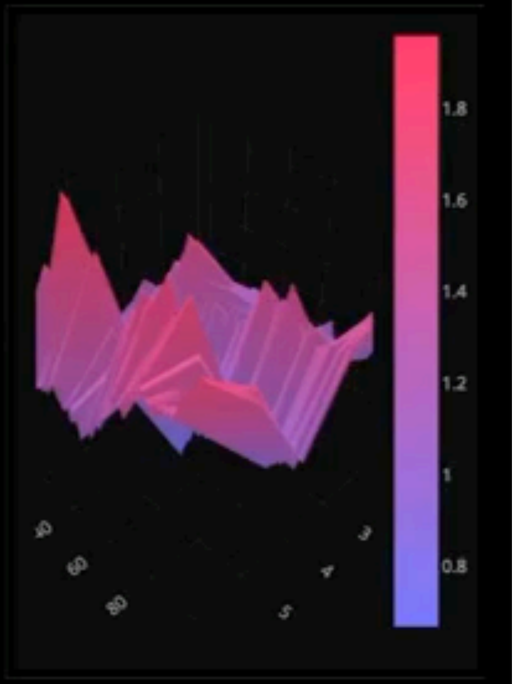
https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

I. Why Python?

plotly

```python
from dash import Dash, dcc, callback, Input, Output
import plotly.express as px
import dash_design_kit as ddk  # Dash Enterprise-only!
import plotly.graph_objs as go

df = px.data.stocks()   # Fetch data from anywhere

app = Dash(__name__)

# Full control over the look and feel of your apps
app.layout = ddk.App([
    ddk.Header(''),

    ddk.Card([
        dcc.Dropdown(df.columns, value='AAPL', id='dropdown'),
        ddk.Graph(id='graph-1')
    ], width=50),

    ddk.Card(
        ddk.Graph(figure=go.Figure(data=go.Surface(z=df.values))),
        width=50),

    ddk.Card(
        ddk.DataTable(data=df.round(2).to_dict('r')),
        width=100),

])

# Dash callback functions respond to user input,
# like selecting dropdowns or entering text
@callback(
    Output('graph-1', 'figure'),
    Input('dropdown', 'value'))
def update(stock_ticker):
    return px.scatter(
        df, y=stock_ticker,
        trendline='rolling',
        trendline_options=dict(window=5),
    )

app.run(debug=True)
```



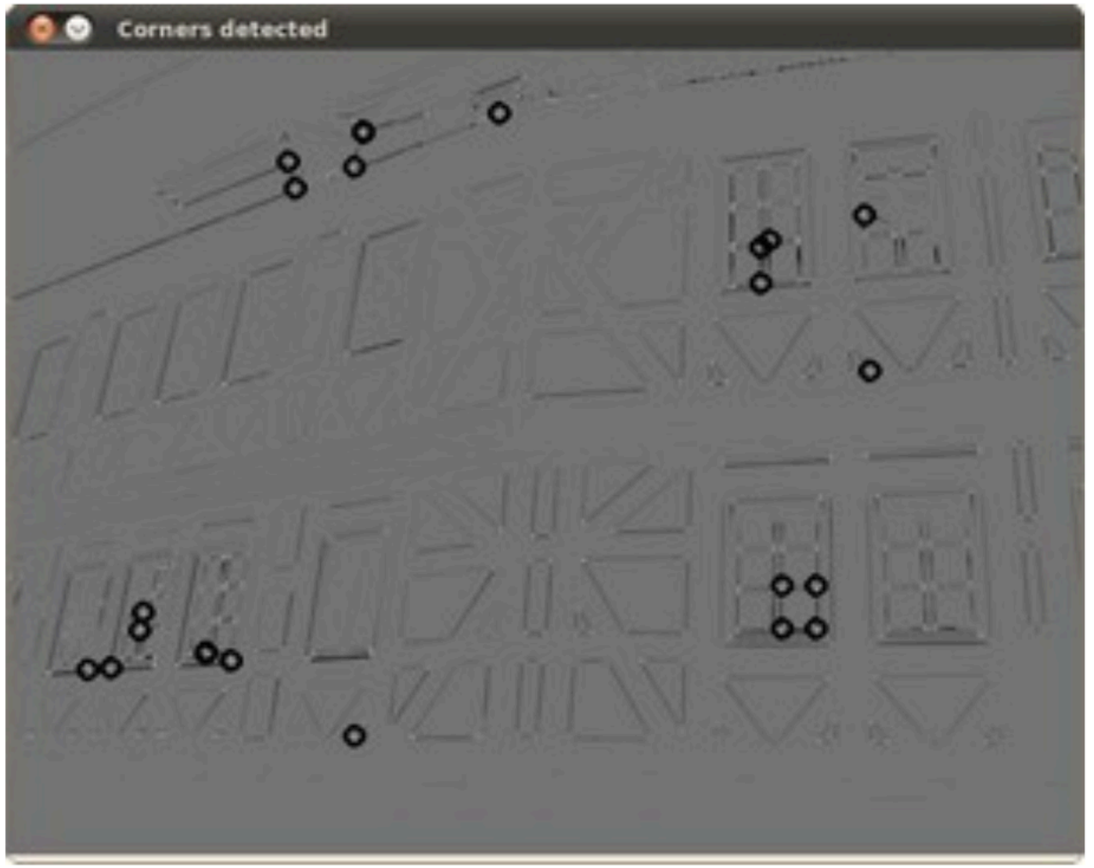


AAPL	AMZN	FB	GOOG	MSFT	NFLX	date
1	1	1	1	1	1	2018-01-01
1.01	1.06	0.96	1.02	1.02	1.05	2018-01-08
1.02	1.05	0.97	1.03	1.02	1.05	2018-01-15
0.98	1.14	1.02	1.07	1.07	1.31	2018-01-22
0.92	1.16	1.02	1.01	1.04	1.27	29
0.89	1.09	0.94	0.94	1	1.19	2-05
0.99	1.18	0.95	0.99	1.04	1.33	2018-02-12

I. Why Python?

OpenCV

Harris corner detector



https://docs.opencv.org/4.x/d4/d7d/tutorial_harris_detector.html

I. Why Python?

OpenAI

OpenAI

ChatGPT: Optimizing Language Models for Dialogue

We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests. ChatGPT is a sibling model to InstructGPT, which is trained to follow an instruction in a prompt and provide a detailed response.

TRY CHATGPT ↗

Generations

The image generations endpoint allows you to create an original image given a text prompt. Generated images can have a size of 256x256, 512x512, or 1024x1024 pixels. Smaller sizes are faster to generate. You can request 1-10 images at a time using the n parameter.

Generate an image — python — Copy

```
response = openai.Image.create(
  prompt="a white siamese cat",
  n=1,
  size="1024x1024"
)
image_url = response['data'][0]['url']
```

The more detailed the description, the more likely you are to get the result that you or your end user want. You can explore the examples in the DALL·E preview app for more prompting inspiration. Here's a quick example:

PROMPT	GENERATION
a white siamese cat	

https://beta.openai.com/docs/guides/images/usage

I. Why Python?

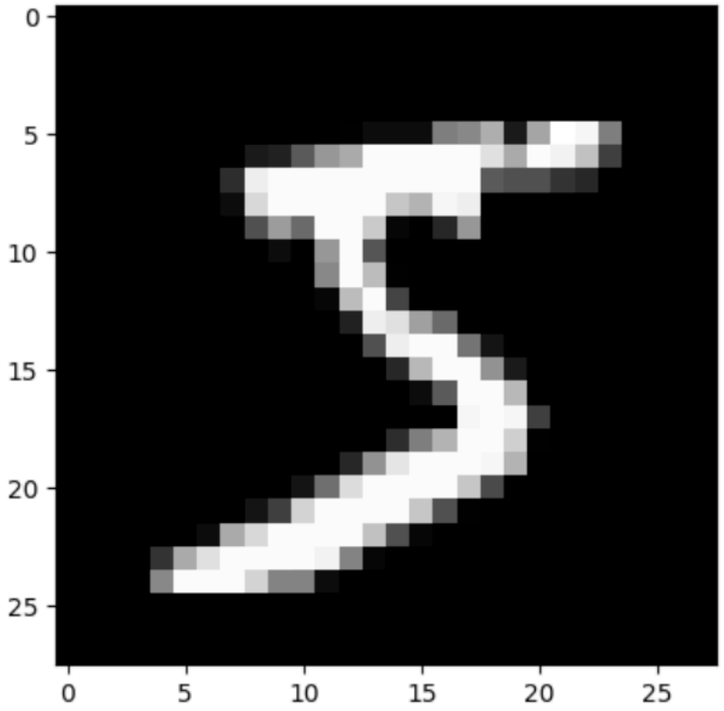
Google AI

Meta AI

TensorFlow

PyTorch

```
class Mnist_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(16, 16, kernel_size=3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(16, 10, kernel_size=3, stride=2, padding=1)

    def forward(self, xb):
        xb = xb.view(-1, 1, 28, 28)
        xb = F.relu(self.conv1(xb))
        xb = F.relu(self.conv2(xb))
        xb = F.relu(self.conv3(xb))
        xb = F.avg_pool2d(xb, 4)
        return xb.view(-1, xb.size(1))

lr = 0.1
```



II. Jupyter Notebook

It is a web-based interactive computing platform

- Good for prototyping and testing codes
- Easy to document and share your work

You can to share the notebook through **a PDF file** or **an interactive webpage** with others

Data

Multivariate normal distribution

First, we can simulate a structured data that is sampled from two different multivariate normal distributions.

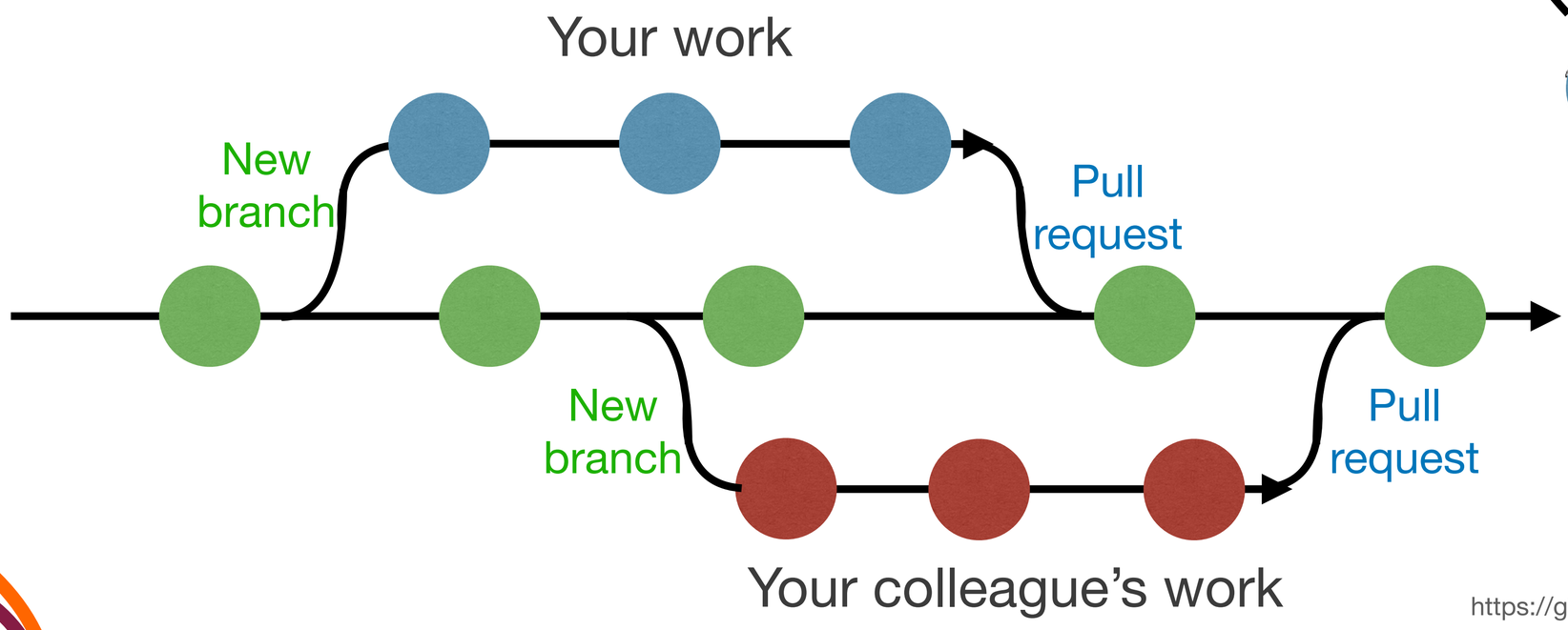
In [194]:
```
# number of clusters
k = 2
# number of data points in each cluster
n = 50
# total number of data points
N = n * k
# cluster proerties: means and standard deviation
means = [[40, 40], [60, 70]]
stds = [[[80, 0],
         [0, 70]],
        [[60, 10],
         [10, 90]]]

# use np.random.multivariate_normal() to sample the datasets
data = []
for mean, std in zip(means, stds):
    data += [np.random.multivariate_normal(mean=mean, cov=std, size=n)]
data = np.concatenate(data)

# validattion
print("Shape: ", data.shape)
print(data[:20])
```

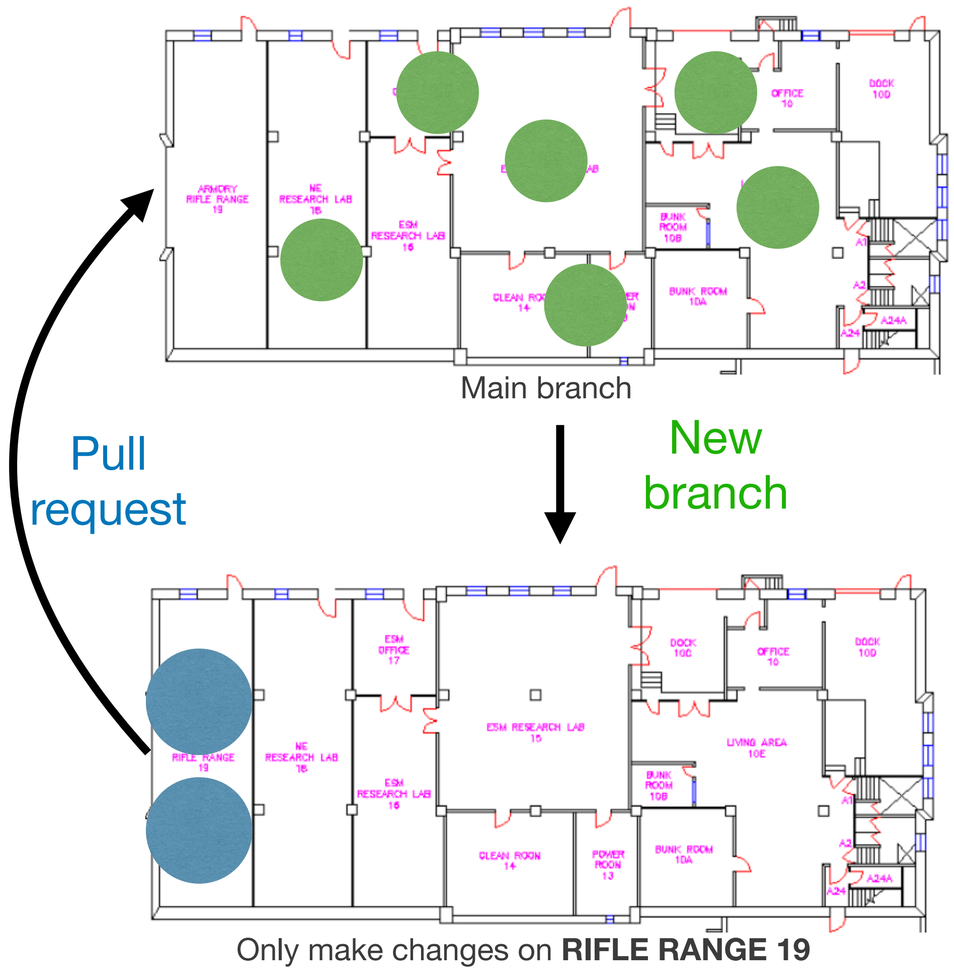
```
Shape:  (100, 2)
[[51.66251802 47.84412914]
 [39.71618565 50.12966424]
 [35.30204586 30.11518838]
 [41.21290983 37.45870684]
 [33.50222942 27.44831926]
 [45.59088463 45.32844912]
 [42.53108047 46.10790442]
 [28.29073007 52.30456678]
 [38.31237176 34.84785694]
 [29.25183562 43.67950456]
 [40.58293457 25.99292839]
 [45.3154831  45.79598333]
 [22.58292304 39.54979532]
 [33.92459233 43.18324273]
 [38.06019806 40.15521412]
 [38.81693873 47.34977002]
 [43.93189465 42.68070032]
 [46.3391774  51.42474463]
 [42.13973106 38.61929858]
 [45.20335028 38.9488359 ]]
```

III. Git and GitHub

A **time machine** for your coding project!

- **Git** is a system that allows you to track and manage changes to your code over time
- It also allows you to merge changes from multiple sources (collaborators) into a single codebase.





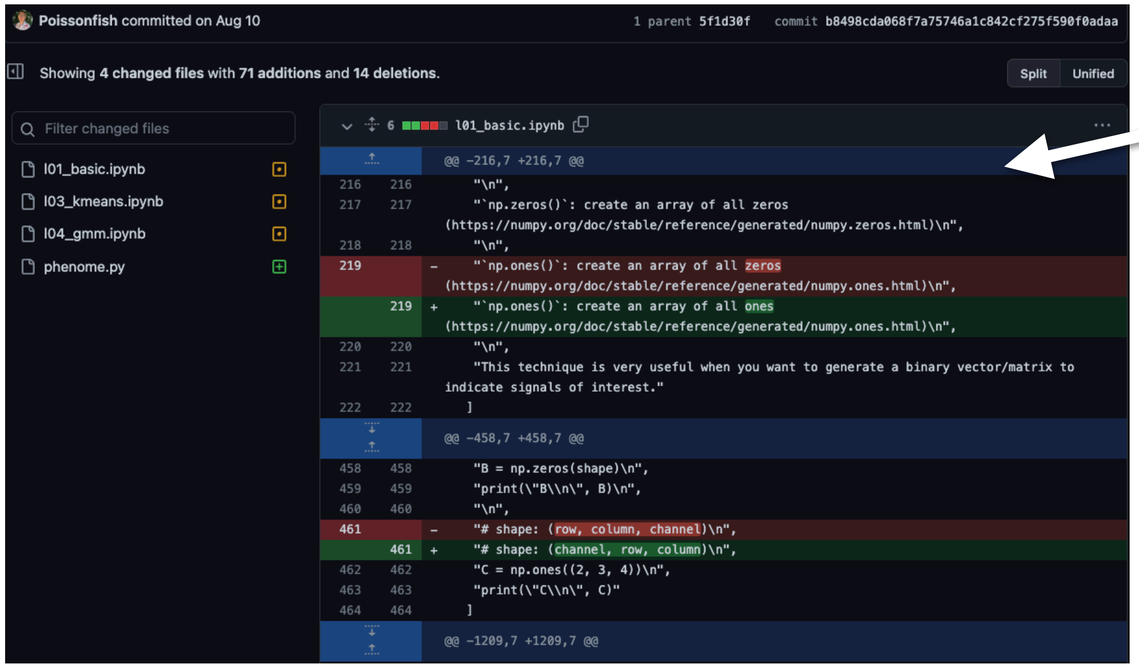
https://github.com/marcuswanner/vtfloorplans/blob/master/floorplans/0203sa00.gif

III. Git and GitHub

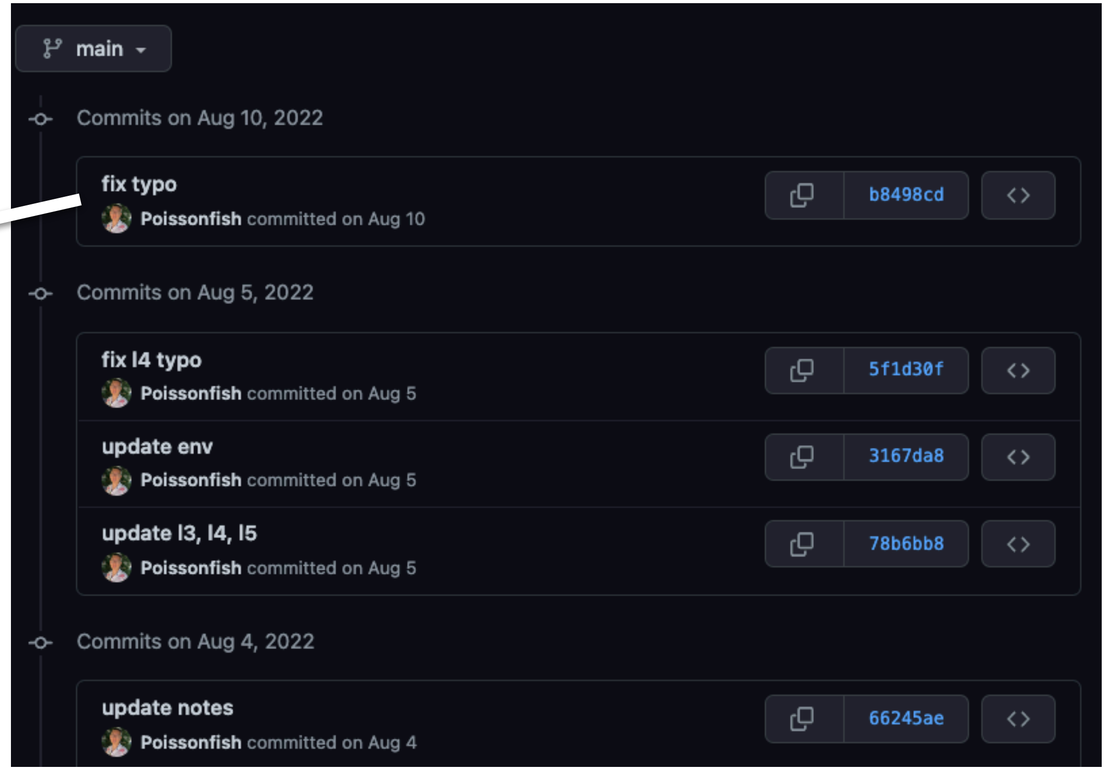
A **time machine** for your coding project!

- **Git** is a system that allows you to track and manage changes to your code over time
- It also allows you to merge changes from multiple sources (collaborators) into a single codebase.

GitHub is a web-based, public platform that implements Git system.



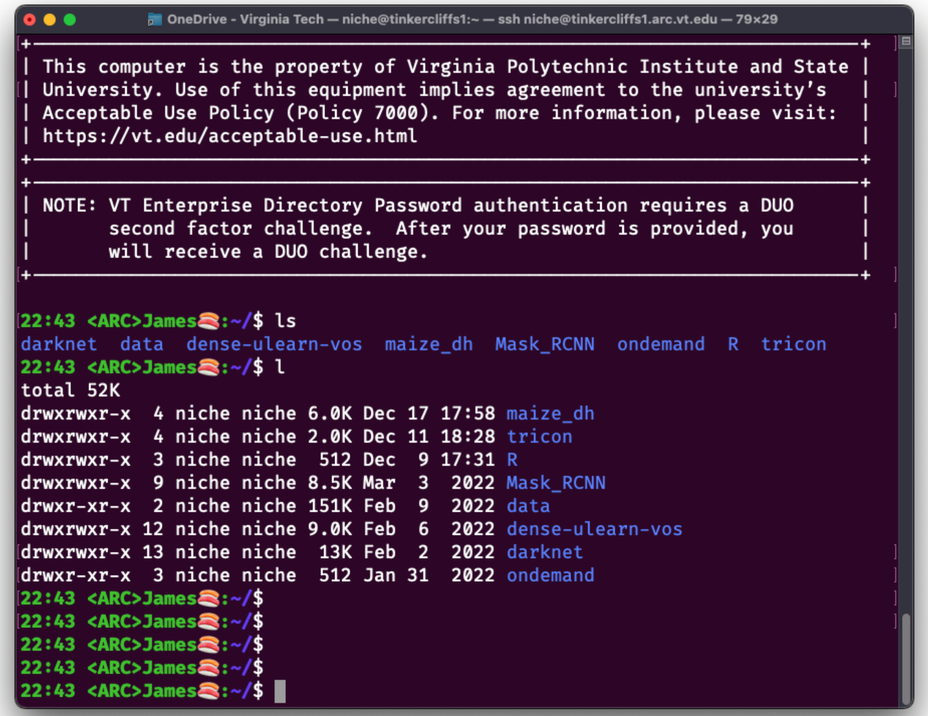
Track every change you made



Historical records of your repository

IV. Unix Shell

- **Unix shell** is a command-line interface that allows you to interact with the operating system.
- The major way to interact with a remote high-performance computer (e.g., VT ARC)
- You can use shell commands to:



A screenshot of the VT ARC remote server

Execute scripts

Copy files

Move / rename files

Delete files

Create folders

List files

Write texts to files

Display file contents

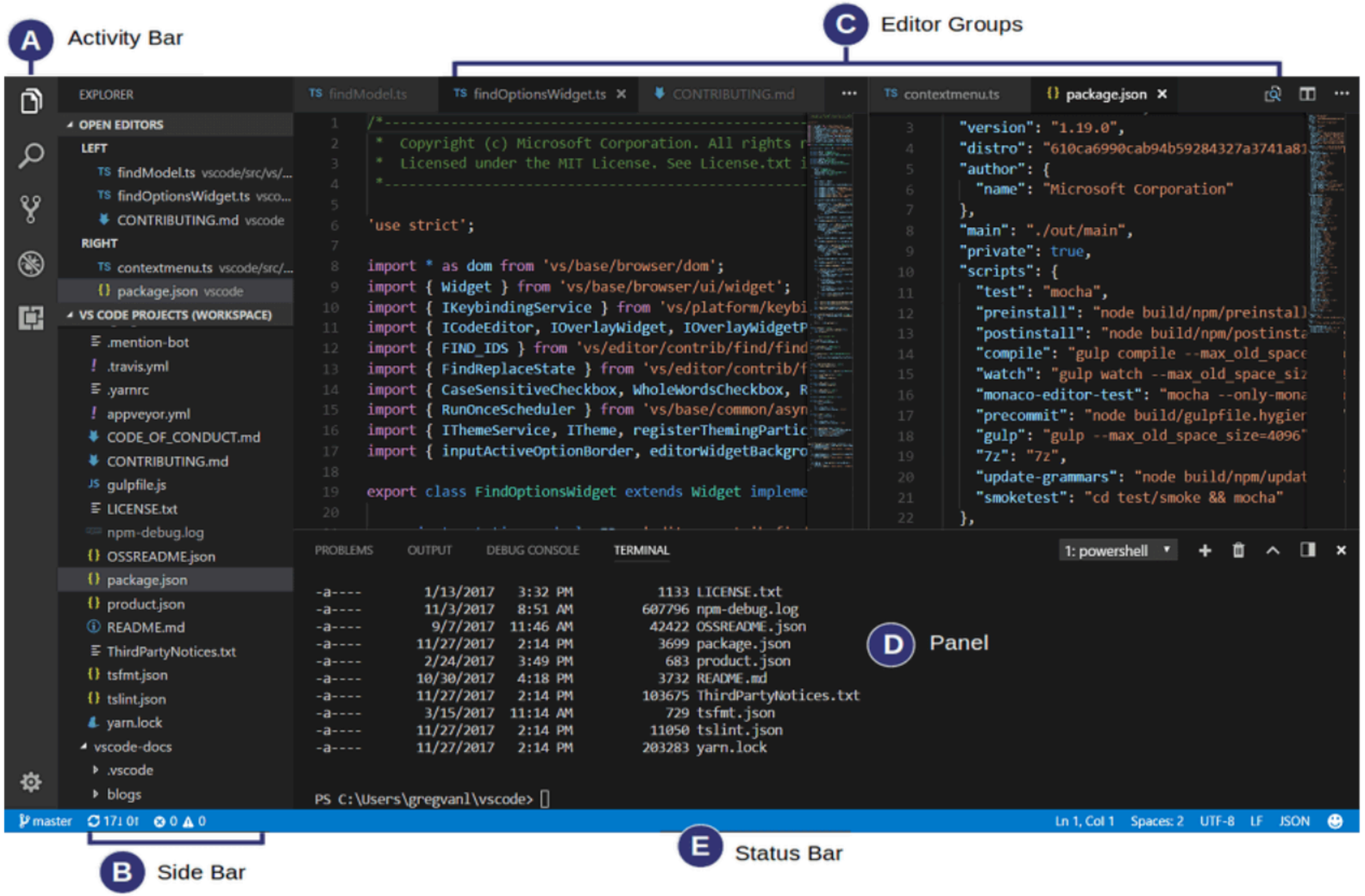
Change the working directory

Show the current working directory

V. Integrated Development Environment (IDE)

Integrated development environment (**IDE**) is a software application that provides a comprehensive set of tools for software development

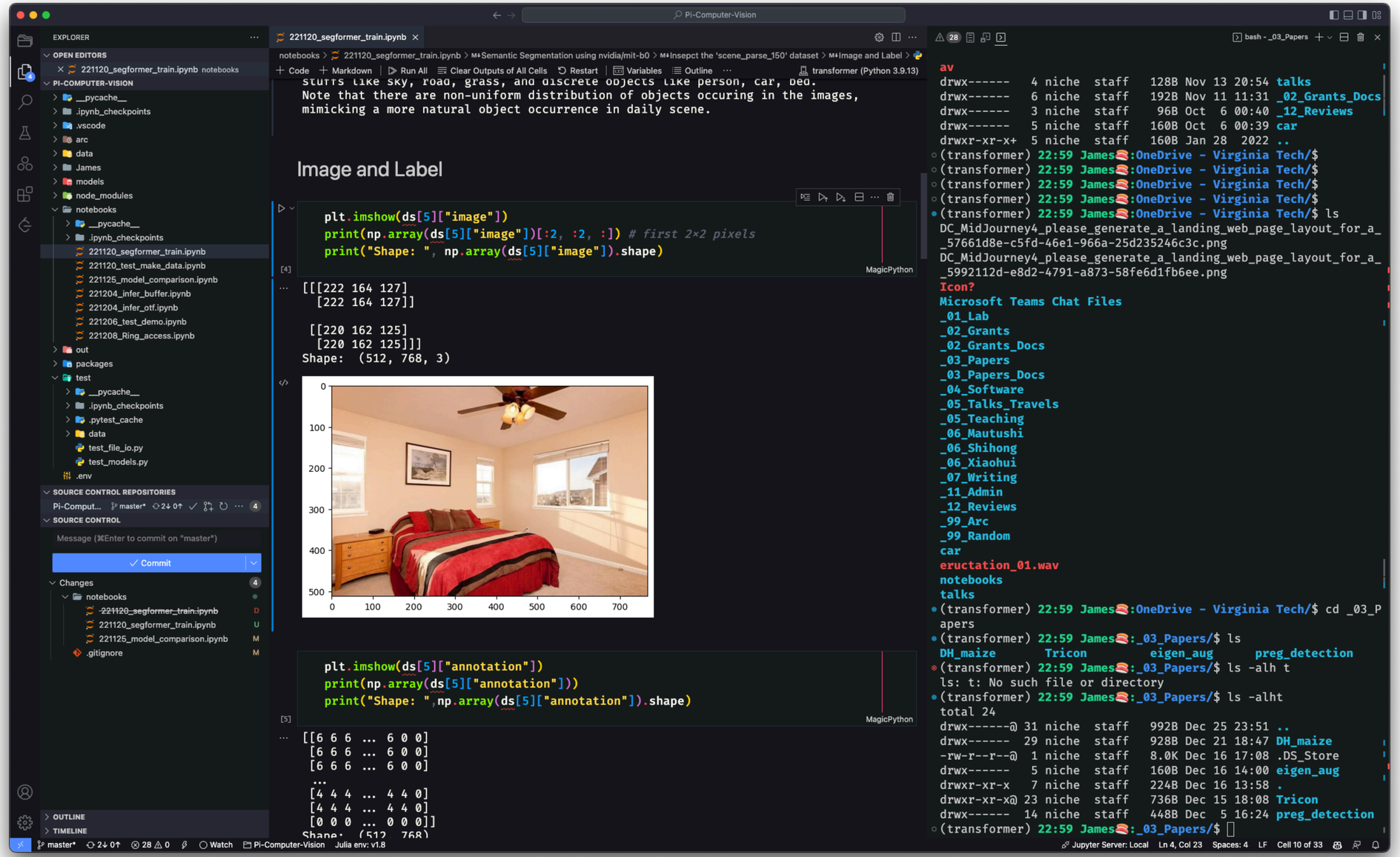
- Code editor
- Automation tools
- Warnings and suggestions
- Code builder (e.g., LaTeX)



https://code.visualstudio.com/docs/getstarted/userinterface

V. Integrated Development Environment (IDE)

IDE put everything together!



V. Integrated Development Environment (IDE)

IDE put everything together!