week-1-summary

January 13, 2024

1 Week 1 Summary

NAME: <Insert Your Name Here>

PID: <Insert Your ID Here>

I certify that the following write-up is my own work, and have abided by the UCSD Academic Integrity Guidelines.

☒ Yes
☐ No

1.1 Key Takeaways from Week 1

Monday: Basics of Python

- Introduction to Python programming
- Key Data Structures: Lists, Tuples, Dictionaries, Sets
- Basic Python syntax and operations

Wednesday NumPy and Linear Algebra

- Introduction to NumPy for numerical computing
- Vector and matrix operations with NumPy
- Basics of linear algebra in Python

Friday: Pandas and Plotting

- Introduction to Pandas for data manipulation
- Creating and manipulating DataFrames
- Basic data visualization techniques

1.2 Monday, Jan 8th

On Monday, we covered the basics of Python programming. We started with a brief introduction to Python and Jupyter notebooks, and then moved on to the basics of Python syntax. We covered variables, loops, conditionals, and functions. We also covered the basics of the four main data structures in Python: lists, tuples, dictionaries, and sets.

Key concepts covered:

- **Data Structures:** Understanding of Lists, Tuples, Dictionaries, and Sets.
- **Basic Syntax and Operations:** Familiarity with Python syntax, including variables, loops, conditionals, and functions.

[1]:
```python
# List example
my_list = [1, 2, 3]
my_list.append(10) # Appending elements
my_list + [4, 5, 6] # Concatenating lists
```

[1]:
```
[1, 2, 3, 10, 4, 5, 6]
```

[2]:
```python
# Tuple example
my_tuple = (1, 2, 3)
# my_tuple.append(10) # Error! Tuples are immutable
my_tuple + (4, 5, 6) # Concatenating tuples
```

[2]:
```
(1, 2, 3, 4, 5, 6)
```

[3]:
```python
# Dictionary example
my_dict = {'a': 1, 'b': 2}
print(f'Keys: {my_dict.keys()}') # Returns a list of keys
print(f'Values: {my_dict.values()}') # Returns a list of values
print(f'Items: {my_dict.items()}') # Returns a list of tuples
```

```
Keys: dict_keys(['a', 'b'])
Values: dict_values([1, 2])
Items: dict_items([('a', 1), ('b', 2)])
```

[4]:
```python
# Set example
my_set_1 = {1, 2, 3}
my_set_2 = {2, 5, 3}
print(f'Union: {my_set_1 | my_set_2}') # Union
print(f'Intersection: {my_set_1 & my_set_2}') # Intersection
print(f'Difference: {my_set_1 - my_set_2}') # Difference
print(f'Symmetric difference: {my_set_1 ^ my_set_2}') # Symmetric difference
```

```
Union: {1, 2, 3, 5}
Intersection: {2, 3}
Difference: {1}
Symmetric difference: {1, 5}
```

Functions and Lambda Functions in Python - **Functions:** - Defined using the `def` keyword. - Can take arguments and return values.

- **Lambda Functions:** A concise way to write functions in a single line.
  - Useful for simple functions that are used once or a few times.
  - Often used with functions like `map()`, `filter()`, and in Pandas operations.

[5]:
```
# Defining a simple function
def sum_of_squares(a, b):
    return a**2 + b**2
sum_of_squares(3, 4)
```

[5]: 25

[6]:
```
# Lambda function equivalent
sum_of_squares_lambda = lambda a, b: a**2 + b**2

sum_of_squares_lambda(3, 4)
```

[6]: 25

1.3 Wed, Jan 10th

On Wednesday we did a refresher on linear algebra, and how to perform linear algebraic operations using the `NumPy` package.

This included:

- creating vectors and matrices,
- performing matrix multiplication,
- computing norms and dot products of vectors,
- computing determinants, trace and inverse of a matrix,
- computing eigenvalues and eigenvectors, and
- solving a system of linear equations.

Creating vectors, matrices, and some basic operations:

[7]:
```python
import numpy as np

# Creating vectors and matrices
vector = np.array([1, 2, 3])
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(f'Vector:\n {vector}\n')
print(f'Matrix:\n {matrix}\n')


product = np.dot(matrix, vector)
print(f'Product:\n {product}\n')
```

```
Vector:
 [1 2 3]

Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Product:
 [14 32 50]
```

Basic linear algebra operations:

[8]:
```python
print(f'Adding vectors: {vector + vector}\n')
print(f'dot product: {np.dot(vector, vector)}\n')
print(f'norm: {np.linalg.norm(vector)}\n')
print(f'Matrix transpose:\n {matrix.transpose()}\n')
print(f'Matrix inverse:\n {np.linalg.inv(matrix)}\n')
```

```
Adding vectors: [2 4 6]
```

```
dot product: 14

norm: 3.7416573867739413

Matrix transpose:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]

Matrix inverse:
 [[ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]
 [-6.30503948e+15  1.26100790e+16 -6.30503948e+15]
 [ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]]
```

Slightly more advanced linear algebra operations:

[9]:
```python
# Solving a system of linear equations
A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])
solution = np.linalg.solve(A, b)
print(f'Solution:\n {solution}\n')

# Matrix eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)
print(f'Eigenvalues:\n {eigenvalues}\n')
print(f'Eigenvectors:\n {eigenvectors}\n')
```

```
Solution:
 [2. 3.]

Eigenvalues:
 [3.61803399 1.38196601]

Eigenvectors:
 [[ 0.85065081 -0.52573111]
 [ 0.52573111  0.85065081]]
```

1.4 Fri, Jan 12th

On Friday, we focused on using the `Pandas` library for data manipulation, along with plotting using `Matplotlib`, `Seaborn` and `gr`.

This included:

- Creating and manipulating DataFrames with Pandas.
- Basic plotting.
- Advanced visualization with Seaborn.

Importing a `.csv` file as a Pandas DataFrame:

[10]:
```python
# iris dataset
import pandas as pd
url = 'https://gist.github.com/curran/a08a1080b88344b0c8a7/raw/
↪0e7a9b0a5d22642a06d3d5b9bcbad9890c8ee534/iris.csv'
df = pd.read_csv(url)

df.head(5) # First 5 rows of the iris dataset
```

[10]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[11]:
```python
# Summary statitsics
df.describe()
```

[11]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

[12]:
```python
# Subsetting example
df[(df['species'] == 'versicolor') & (df['sepal_length'] > 4)].head(5)
```

[12]:

	sepal_length	sepal_width	petal_length	petal_width	species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor

[13]:
```
# creating variables
df['species_color_code'] = df['species']\
    .map( lambda species: 0 if species == 'setosa' else 1 if species ==␣
 ↪'versicolor' else 2 )
```

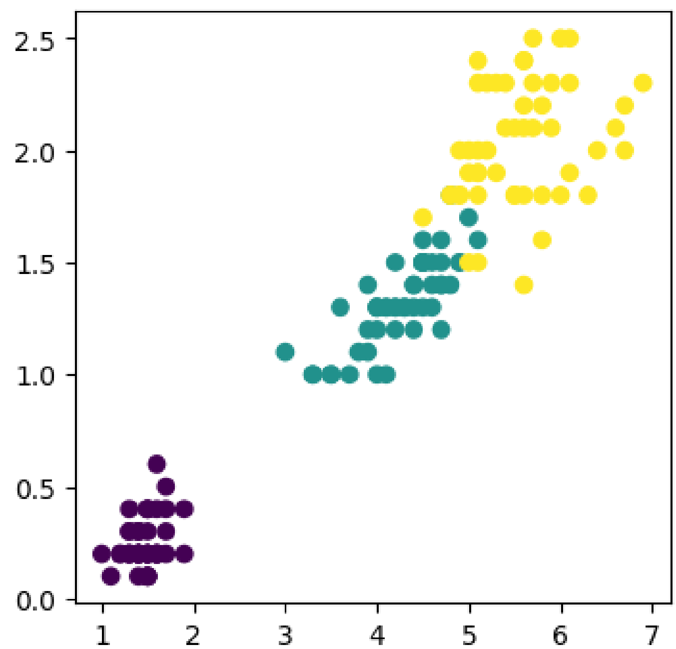
Scatterplot of petal length vs. petal width colored by species using `matplotlib`

[14]:
```
%matplotlib inline

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize=(4,4))
plt.scatter(df['petal_length'], df['petal_width'], c=df['species_color_code'])
```

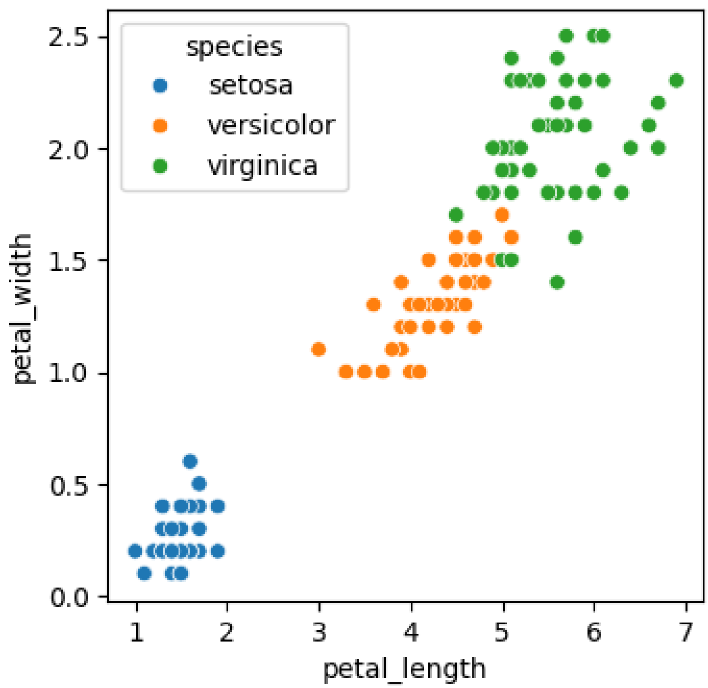
[14]:
```
<matplotlib.collections.PathCollection at 0x1196afcd0>
```



Scatterplot of petal length vs. petal width colored by species using `seaborn`

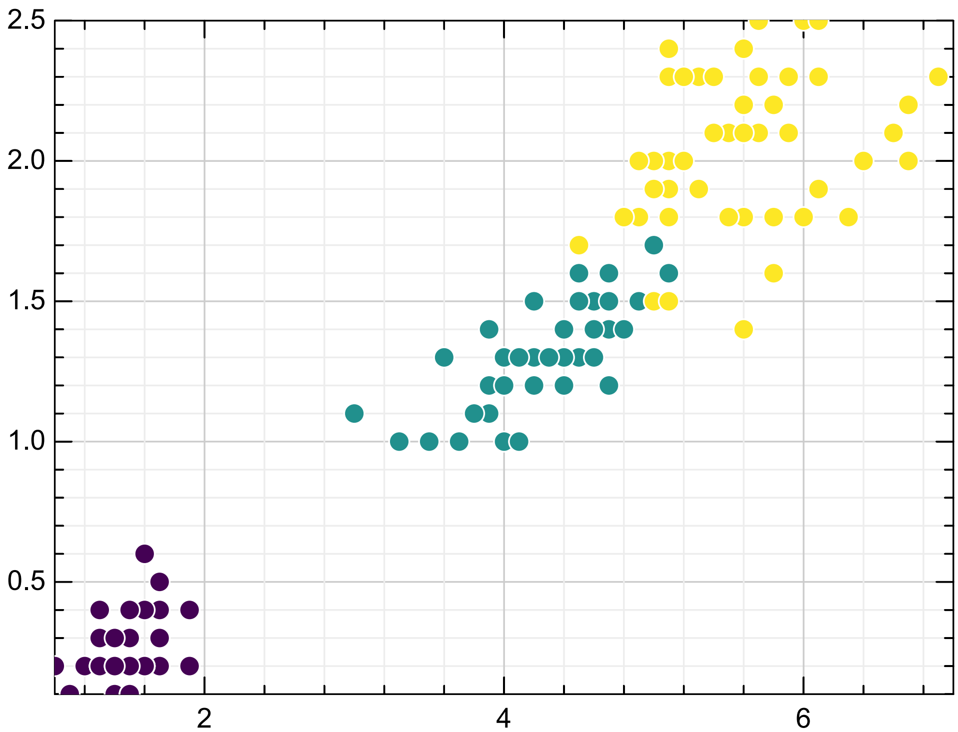
[15]:
```
# Scatterplot of sepal length vs. sepal width using seaborn colored by species
import seaborn as sns
fig, ax = plt.subplots(1,1, figsize=(4,4))
sns.scatterplot(x='petal_length', y='petal_width', hue='species', data=df,␣
 ↪ax=ax)
```

[15]:
```
<Axes: xlabel='petal_length', ylabel='petal_width'>
```



Scatterplot of petal length vs. petal width colored by species using gr

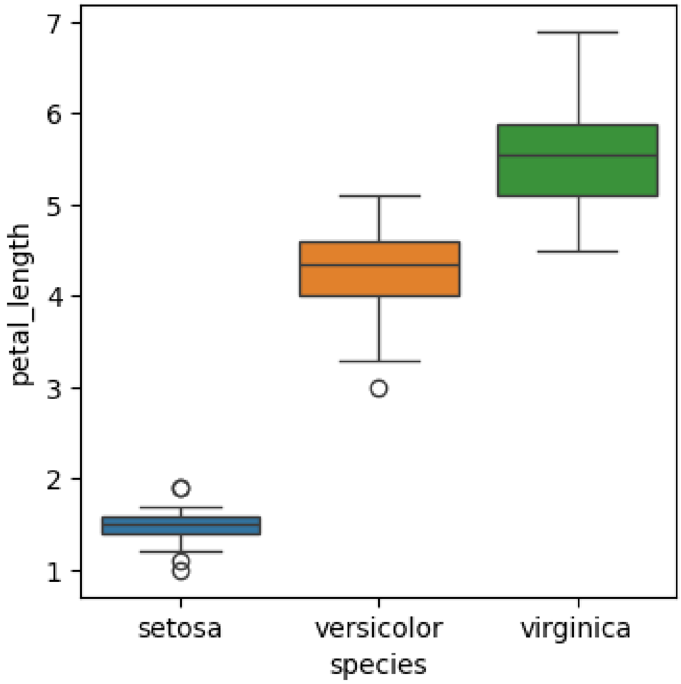
[16]:
```
# Scatterplot of sepal length vs. sepal width using gr colored by species
import gr
import gr.pygr as pygr
pygr.scatter(
    df['petal_length'],                    # X values
    df['petal_width'],                     # Y values
    200 * np.ones_like(df['species']),     # Size of the points in the scatterplot
    df['species_color_code']               # Color of the points in the scatterplot
)
```



Boxplot of petal length by species using seaborn.

[17]:
```
fig, ax = plt.subplots(1, 1, figsize=(4, 4))
sns.boxplot(x='species', y='petal_length', data=df, hue='species', ax=ax)
```

[17]: <Axes: xlabel='species', ylabel='petal_length'>



Histogram of petal length by species using seaborn.

[18]:
```
fig, axes = plt.subplots(1, 3, figsize=(9, 3))
for i, species in enumerate(df['species'].unique()):
    sns.histplot(df[df['species'] == species]['petal_length'], ax=axes[i],␣
 ↪kde=False)
    axes[i].set_title(species)
```