PYTHON BASICS
Cheat Sheet

Data Structres

Numbers

```
# Integer (int)
>>> x = 2
# Float
>>> x = 2.3
# Complex
>>> x = 2j + 1
```

Operations

```
# Exponent
>>> 2**3
8

# Modulus / Remainder
>>> 22%8
6

# Integer Division
>>> 22//8
2

# Division
>>> 22/8
2.75

# Multiplication
>>> 3*3
9

# Subtraction
>>> 5-2
3

# Addition
>>> 2+2
4
```

Boolean

```
>>> a = 2
>>> b = 7

>>> a == b
False

>>> a != b
True

>>> a > b
False

>>> a >= b
False

>>> a < b
True

>>> a <= b
True

>>> (a > 1) & (b < 10)
True

>>> (a > 3) | (b > 5)
True

>>> a is b
False

>>> a is not b
True
```

Strings

```
>>> txt_1 = 'Hello World!'
'Hello World!'

>>> txt_2 = "HELLO WORLD!"
'HELLO WORLD!'

>>> long_txt = """
               Hello World,
               Welcome DSMLBC!
               """
'Hello World, \n Welcome DSMLBC!'
```

Indexing & Slicing

```
>>> txt_1[0]
'H'

>>> txt_1[-1]
'!'

>>> txt_1[1:4]
'ell'

>>> txt_1[:5]
'Hello'
```

String Methods

```
>>> len(txt_1)
12

>>> txt_1.upper()
'HELLO WORLD!'

>>> txt_2.lower()
'hello world!'

>>> txt_1.replace("World", "Era")
'Hello Era!'

>>> txt_1.split()
['Hello', 'World!']

>>> ' Hello World! '.strip()
'Hello World!'

>>> 'hello world!'.capitalize()
'Hello world!'
```

Dictionaries

```
>>> dict_1 = {"REG" : "Regression",
              "LOG" : "Logistic Regression",
              "CART": "Classification and Reg"}

>>> dict_2 = {"REG": ["RMSE", 10]
              "LOG": ["MSE", 20]
              "CART": ["SSE", 30]}
```

Key - Value Methods

```
>>> dict_1.keys()
dict_keys(['REG', 'LOG', 'CART'])

>>> dict_1.values()
dict_values(['Regression', 'Logistic Regression',
             'Classification and Reg'])

>>> dict_1.items()
dict_items([('REG', 'Regression'),
            ('LOG', 'Logistic Regression'),
            ('CART', 'Classification and Reg')])

>>> dict_1["REG"]
'Regression'

>>> dict_2["CART"][1]
30
```

Dictionary Methods

```
>>> dict_1 = {"REG": "Regression",
              "LOG": "Logistic Regression",
              "CART": "Classification and Reg"}

>>> dict_2 = {"REG": ["RMSE", 10]
              "LOG": ["MSE", 20]
              "CART":["SSE", 30]}
```

Lists

```
>>> list_1 = [1, 2, 3, "a", "b"]

>>> list_2 = [True, [1, 2, 3]]

>>> numbers = [4, 2, 1, 3]
```

Indexing & Slicing

```
>>> list_1[3]
'a'

>>>list_2[-1]
[1, 2, 3]

>>>list_1[0:4]
[2, 3, 'a']
```

List Methods

```
>>> list_1 + list_2
[1, 2, 3, 'a', 'b', True, [1, 2, 3]]

>>> list_1.append('c')
[1, 2, 3, 'a', 'b', 'c']

>>> list_2.remove('True')
[[1, 2, 3]]

>>> len(list_1)
5

>>> numbers.sort()
[1, 2, 3, 4]

>>> list_2.insert(1, False)
[True, False, [1, 2, 3]]

>>> numbers.pop()
[4, 2, 1]
```

Tuples

```
>>> tuple = ("john", "mark", 1, 2)

>>> tuple[0]
'john'

>>> tuple[1:3]
       ('mark', 1)

>> len(tuple)
4
```

Set

```
>>> set_1 = {1, 2, 2, 3, 3, 3}
{1, 2, 3}

>>> set_2 = set([3, 4, 4, 5, 5, 5, 6])
{3, 4, 5, 6}
```

Set Methods

```
>>> set_1.difference(set_2)
{1, 2}

>>> set_2.difference(set_1)
{4, 5, 6}

>>> set_1.symmetric_difference(set_2)
{1, 2, 4, 5, 6}

>>> set_1.intersection(set_2)
{3}

>>> set_1.union(set_2)
{1, 2, 3, 4, 5, 6}

>>> set_1.isdisjoint(set_2)
False

>>> set_1.issubset(set_2)
False

>>> set_1.issuperset(set_2)
False
```

Comprehensions

List Comprehensions

```
# Syntax
list = [expression for item in iterable if condition]
>>> squares = [x**2 for x in range(1, 11)]
>>> print(squares)
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

>>> evens = [f"{x}: even" for x in range(1,10) if x % 2 == 0]
>>> print(evens)
['2: even', '4: even', '6: even', '8: even']

>>> even_odd = [f"{x}: even" if x%2 == 0 else f"{x}: odd"
                for x in range(1,5)]

>>> print(even_odd)
['1: odd', '2: even', '3: odd', '4: even']
```

Dictionary Comprehensions

```
# Syntax
dict = {key_exp: value_exp for item in iterable if condition}

>>> dictionary = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

>>> {k: v ** 2 for (k, v) in dictionary.items()}
{'a': 1, 'b': 4, 'c': 9, 'd': 16}

>>> {k.upper(): v for (k, v) in dictionary.items()}
{'A': 1, 'B': 2, 'C': 3, 'D': 4}

>>> {k.upper(): v*2 for (k, v) in dictionary.items()}
{'A': 2, 'B': 4, 'C': 6, 'D': 8}
```

Loops

For Loop

```
# Syntax
for <variable> in <list>:
  <code block>

>>> students = ["John", "Mark", "Venessa"]

>>> for student in students:
...     print(student)
John
Mark
Venessa

>>> for index, student in enumerate(students):
...     print(index, student)

0 John
1 Mark
2 Venessa
```

While Loop

```
# Syntax
while <condition>:
  <code to execute while the condition is true>

>>> i = 1
>>> while i < 5:
...     print(i)
...     if i==3:
...         break
...     i+=1
1
2
3

>>> while i < 5:
...     i+=1
...     if i==3:
...         continue
...     print(i)
2
4
5
```

Conditions

```
>>> number = 3.14
>>> if number > 0:
...     print(f"{number} is positive.")
... elif number < 0:
...     print(f"{number} is negative.")
... else:
...     print(f"{number} is zero!")
3.14 is positive.
```

Functions

```
# Function_1
>>> def say_hi(name):
...     print(f'Merhaba {name}')

# Calling function
>>> say_hi('Miuul')
Merhaba Miuul

# Function_2
>>> def summer(num_1, num_2):
...     """
...     Sum of two numbers
...     Args:
...         num_1: int, float
...         num_22: int, float
...     Returns:
...         int, float
...     """
...     Return num_1 + num_2

# Calling function
>>> summer(3, 4)
7

# Function_3
>>> def find_volume(length=1, width=1, depth=1):
...     print(f'Length = {length}')
...     print(f'Width = {width}')
...     print(f'Depth = {depth}')
...     volume = length * width * depth

# Calling function
>>> find_volume(1, 2, 3)
Length = 1
Width = 2
Depth = 3
6

>>> find_volume(2, depth=3, width=4)
Length = 2
Width = 4
Depth = 3
24
...     return volume
```

Local & Global Variables

```
>>> list_store = [1, 2] # Global variable

>>> def add_element(a, b):
... c=a*b
... list_store.append(c)
... print(list_store)

>>> add_element(1, 9)
[1, 2, 9]
```

Built-in Functions

```
# lambda
>>> summer = lambda a, b: a + b
>>> summer(3, 5)
8

# map
>>> salaries = [1000, 2000, 3000, 4000, 5000]
>>> list(map(lambda x: x * 20 / 100 + x, salaries))
[1200.0, 2400.0, 3600.0, 4800.0, 6000.0]

# filter
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(filter(lambda x: x % 2 == 0, numbers))
[2, 4, 6, 8, 10]

# reduce
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> reduce(lambda a, b: a + b, numbers)
55

# zip
>>> students = ["John", "Mark", "Venessa"]
>>> departments = ["mathematics", "statistics", "physics"]
>>> list(zip(students, departments))
[('John', 'mathematics'), ('Mark', 'statistics'), ('Venessa', 'physics')]
```

PANDAS Cheat Sheet

```
import pandas as pd
```

Data Structures

Series

```
# Create a series
>>> s = pd.Series([1, 2, 3], index=['A' ,'B' ,'C'])
```

Index	Data
A	1
B	2
C	3

```
>>> s.index
Index(['A', 'B', 'C'], dtype='object')

>>> s.dtype
dtype('int64')

>>> s.size
3

>>> s.ndim
1

>>> s.values
array([1, 2, 3], dtype=int64)
```

DataFrame

```
# Create a DataFrame
>>> df = pd.DataFrame({'ID': [1, 2, 3],
                       'Name': ['Alex', 'Brian', 'David'],
                       'Profession': ['DA', 'DE', 'DS']},
                        index = ['1 (One)', '2 (Two)', '3 (Three)'])
```

	ID	Name	Profession
1 (One)	1	Alex	DA
2 (Two)	2	Brian	DE
3 (Three)	3	David	DS

```
# Get the shape of the DataFrame
>>> df.shape
(3, 3)

# Get the column names of the DataFrame
>>> df.columns
Index(['ID', 'Name', 'Profession'], dtype='object')

# Get the index information of the DataFrame
>>> df.index
Index(['1 (One)', '2 (Two)', '3 (Three)'], dtype='object')

# Get the data types of the DataFrame columns
>>> df.dtypes
ID            int64
Name         object
Profession   object
dtype: object

# Check if there are any missing values in the DataFrame
>>> df.isnull().values.any()
False

# Get the count of missing values in each column
>>> df.isnull().sum()
ID           0
Name         0
Profession   0
dtype: int64

# Get the count of non-null values in each column
>>> df.count()
ID           3
Name         3
Profession   3
dtype: int64

# Generate descriptive statistics of the DataFrame (transposed)
>>> df.describe().T
```

Importing & Exporting

```
# Read a CSV file and create a DataFrame
>>> pd.read_csv('filename.csv')

# Save the DataFrame to a CSV file
df.to_csv(filename.csv)

# Read an Excel file and create a DataFrame
>>> pd.read_excel('filename.xlsx')

# Save the DataFrame to an Excel file
>>> df.to_excel('filename.xlsx')
```

Data Manipulation

Selection

Selecting Rows

```
# Select a specific row by its label
>>> df.loc['1 (One)']:

# Select a specific row by its index
>>> df.loc[1]
```

Selecting Columns

```
# Select a single column by its name
>>> df['ID']

# Select multiple columns by their name
>>> df[['ID', 'Profession']]

# Select a single column using label-based indexing
>>> df.loc[:, 'Name']

# Select a single column using integer-based indexing
>>> df.iloc[:, 1]
```

Selecting Rows and Columns

```
# Select a specific cell by its label-based row and column indices
>>> df.loc['2 (Two)', 'Name']

# Select a specific cell by its integer-based row and column indices
>>> df.iloc[1, 1]

# Select specific rows and columns using label-based indexing
>>> df.loc['1 (One)', ['ID', 'Profession']]

# Select specific rows and columns using integer-based indexing
>>> df.iloc[1, ['ID', 'Profession']]
```

Conditional Selection

```
# Select rows based on a condition
>>> df[df['ID'] > 1]

# Select rows based on multiple conditions using logical operators
>>> df[(df['ID'] > 2) & (df['Profession'] == 'DS')]

# Select rows where a column value is present in a given list
>>> df.loc[df['Name'].isin(['Alex', 'David'])]
```

Deleting & Adding

```
# Drop rows with any missing values
>>> df.dropna()

# Drop columns with any missing values
>>> df.dropna(axis=1)

# Drop columns with fewer than n non-null values
>>> df.dropna(axis=1, thresh=n)

# Fill missing values with a specified value
df.fillna(value)
```

Filtering & Sorting

```
# Filter columns using regular expression matching
>>> df.filter(regex=regex)


# Sort DataFrame by values in a specific column in descending order
>>> df.sort_values('ID', ascending=False)
```

Editting

```
# Perform quantile-based discretization of values in 'col' into 3
bins with custom labels
>>> pd.qcut(df['col'], 3, labels=qcut_labels)

# Perform value-based discretization of values in 'col' into custom
bins with labels
>>> pd.cut(df['col'], bins=cut_bins, labels=cut_labels)

# Create a pivot table using 'col1' as index, 'col2' as columns, and
'col3' as values
>>> df.pivot(index='col1', columns='col2', values='col3')

# Reset the index of the DataFrame
>>> df.reset_index()

# Rename specific columns of the DataFrame
>>> df.rename(columns = rename_list)
```

Grouping & Aggregation

```
# Perform grouping operation on 'col' and obtain a GroupBy object
>>> df.groupby('col')


# Perform grouping operation on multiple columns ('col1' and 'col2')
and calculate the mean of 'col3'
>>> df.groupby(['col1', 'col2']).agg({"col3": "mean"})

# Perform grouping operation on multiple columns ('col1' and 'col2')
and calculate the mean of 'col3' and count of 'col4'
>>> df.groupby(['col1', 'col2']).agg({'col3': 'mean', 'col4':
'count'})


# Grouping the DataFrame by the first level of the index
>>> df.groupby(level=0)


# An example
>>> df = pd.DataFrame({
    'Name': ['John', 'Alice', 'John', 'Alice', 'Bob'],
    'City': ['New York', 'Paris', 'London', 'Paris', 'London'],
    'Age': [30, 25, 35, 28, 40],
    'Salary': [50000, 60000, 55000, 45000, 70000]})


>>> df.groupby('City').agg({'Salary': 'mean', 'Age': 'max'})
```

	Name	City	Age	Salary
0	John	New York	30	50000
1	Alice	Paris	25	60000
2	Jhon	London	35	55000
3	Alice	Paris	28	45000
4	Bob	London	40	70000

→

City	Salary	Age
London	62500.0	40
New York	50000.0	30
Paris	52500.0	28

Joining

```
# Merge operation
>>> df1 = pd.DataFrame({
        'key': ['A', 'B', 'C', 'D'],
        'value1': [1, 2, 3, 4]})

>>>> df2 = pd.DataFrame({
        'key': ['B', 'D', 'E', 'F'],
        'value2': [5, 6, 7, 8]})

>>> pd.merge(df1, df2, on='key', how='inner')
```

	Key	Value1
0	A	1
1	B	2
2	C	3
3	D	4

⊕

	Key	Value2
0	B	5
1	D	6
2	E	7
3	F	8

=

	Key	Value1	Value2
0	B	2	5
1	D	4	6

```
>>> df4 = pd.DataFrame({
        'key': ['B', 'D', 'E', 'F'],
        'value4': ['orange', 'grape', 'kiwi', 'lemon']})

>>> df3.set_index('key').join(df4.set_index('key'), how='inner')
```

	Key	Value3
0	A	apple
1	B	bannana
2	C	cherry
3	D	date

⊕

	Key	Value2
0	B	orange
1	D	grape
2	E	kiwi
3	F	lemon

=

Key	Value1	Value2
B	banana	orange
D	date	grape

```
# Example for concatenate operation
>>> df5 = pd.DataFrame({
        'A': ['A0', 'A1', 'A2'],
        'B': ['B0', 'B1', 'B2']})

>>> df6 = pd.DataFrame({
       'A': ['A3', 'A4', 'A5'],
       'B': ['B3', 'B4', 'B5']})

>>> pd.concat([df5, df6], axis=0)
```

	A	B
0	A0	B0
1	A1	B1
2	A2	B2

⊕

	A	B
0	A3	B3
1	A4	B4
2	A5	B5

=

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
0	A3	B3
1	A4	B4
2	A5	B5

Statistics

```
# Generate descriptive statistics of the DataFrame
>>> df.describe()

# Calculate the mean of each column in the DataFrame
>>> df.mean()

# Calculate the correlation matrix of the DataFrame
>>> df.corr()

# Count the number of non-null values in each column of the DataFrame
>>> df.count()

# Find the maximum value in each column of the DataFrame
>>> df.max()

# Find the minimum value in each column of the DataFrame
>>> df.min()

# Calculate the median of each column in the DataFrame
>>> df.median()

# Calculate the standard deviation of each column in the DataFrame
>>> df.std()
```

Data Visualization

Cheat Sheet

Importing Libraries

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Basic Line Plot

```
plt.plot(x, y, linestyle = '-', color = 'b')
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

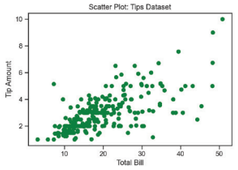
```
# Sample dataset
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May']
revenue = [10000, 15000, 12000, 18000, 20000]

# Create the line plot
plt.plot(months, revenue, marker='o', linestyle='-', color='b')
plt.xlabel('Months')
plt.ylabel('Revenue ($)')
plt.title('Monthly Revenue')
plt.grid(True)
plt.show()
```

Scatter Plot

```
plt.scatter(x, y, c='color')
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the tips dataset
df = sns.load_dataset("tips")

# Extract the total bill and tip amounts
total_bill = df["total_bill"].values
tip_amount = df["tip"].values

# Create a scatter plot
plt.scatter(total_bill, tip_amount, c='green')
plt.xlabel('Total Bill')
plt.ylabel('Tip Amount')
plt.title('Scatter Plot: Tips Dataset')
plt.show()
```

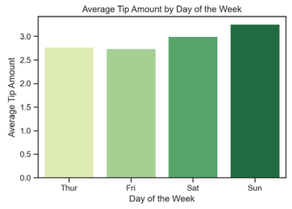


Bar Plot

```
plt.bar(x, y, color = 'b', width = 0.5, height = 0.3)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()

# Horizontal bar
plt.barh(x, y)
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Calculate the average tip amount for each day of the week
avg_tip_by_day = tips.groupby('day')['tip'].mean()

# Create the bar plot
sns.barplot(x=avg_tip_by_day.index, y=avg_tip_by_day.values)
plt.xlabel('Day of the Week')
plt.ylabel('Average Tip Amount')
plt.title('Average Tip Amount by Day of the Week')
plt.show()
```

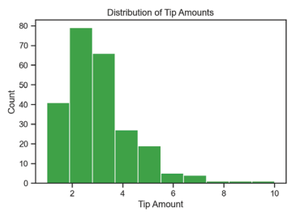


Histogram

```
plt.hist(data, bins=10)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Create the histogram
sns.histplot(data=tips, x='tip', bins=10, color='green')
plt.xlabel('Tip Amount')
plt.ylabel('Count')
plt.title('Distribution of Tip Amounts')
plt.show()
```

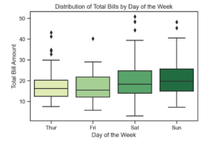


Box Plot

```
sns.boxplot(data, x, y, color)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Create the box plot with a different color
sns.boxplot(data=tips, x='day', y='total_bill',
color='green')
plt.xlabel('Day of the Week')
plt.ylabel('Total Bill Amount')
plt.title('Distribution of Total Bills by Day of the Week')
plt.show()
```

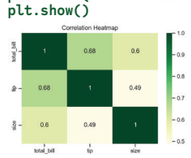


Heatmap

```
sns.heatmap(data, annot=True)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Calculate the correlation matrix
correlation_matrix = tips.corr()

# Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='YlGn')
plt.title('Correlation Heatmap')
plt.show()
```

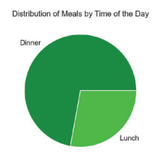


Pie Chart

```
plt.pie(y, labels=labels, colors=[colors])
plt.title('Title')
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Calculate the count of meals for each time of the day
meal_counts = tips['time'].value_counts()

# Specify custom labels and colors
labels = meal_counts.index
colors = ['darkgreen', 'aquamarine']

# Create the pie chart
plt.pie(meal_counts, labels=labels, colors=colors)
plt.title('Distribution of Meals by Time of the Day')
plt.show()
```

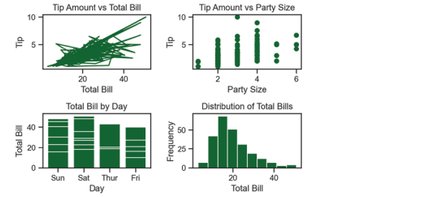


Multiple Subplots

```
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x1, y1)
axs[0, 1].scatter(x2, y2)
axs[1, 0].bar(x3, height3)
axs[1, 1].hist(data4, bins=10)
plt.show()
```

Example

```
# Load the 'tips' dataset from seaborn
tips = sns.load_dataset('tips')

# Prepare the data for plotting
x1 = tips['total_bill']
y1 = tips['tip']
x2 = tips['size']
y2 = tips['tip']
x3 = tips['day']
height3 = tips['total_bill']
data4 = tips['total_bill']

# Create the subplots
fig, axs = plt.subplots(2, 2)

# Plot 1: Line plot
axs[0, 0].plot(x1, y1)
axs[0, 0].set_xlabel('Total Bill')
axs[0, 0].set_ylabel('Tip')
axs[0, 0].set_title('Tip Amount vs Total Bill')

# Plot 2: Scatter plot
axs[0, 1].scatter(x2, y2)
axs[0, 1].set_xlabel('Party Size')
axs[0, 1].set_ylabel('Tip')
axs[0, 1].set_title('Tip Amount vs Party Size')

# Plot 3: Bar plot
axs[1, 0].bar(x3, height3)
axs[1, 0].set_xlabel('Day')
axs[1, 0].set_ylabel('Total Bill')
axs[1, 0].set_title('Total Bill by Day')

# Plot 4: Histogram
axs[1, 1].hist(data4, bins=10)
axs[1, 1].set_xlabel('Total Bill')
axs[1, 1].set_ylabel('Frequency')
axs[1, 1].set_title('Distribution of Total Bills')

# Adjust the layout and spacing
fig.tight_layout()

# Display the plot
plt.show()
```



Customizing Plots

```
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='red', linestyle='--', linewidth=2,
marker='o', markersize=6)
plt.xlabel('X-axis label', fontsize=12)
plt.ylabel('Y-axis label', fontsize=12)
plt.title('Title', fontsize=14)
plt.legend(['Legend'])
plt.grid(True)
plt.show()
```

Example

```
# Load the 'tip' dataset from seaborn
tips = sns.load_dataset('tips')

# Prepare the data for plotting
x = tips['total_bill']
y = tips['tip']

# Create the plot
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='red', linestyle='--', linewidth=2,
marker='o', markersize=6)
plt.xlabel('Total Bill', fontsize=12)
plt.ylabel('Tip', fontsize=12)
plt.title('Tip Amount vs Total Bill', fontsize=14)
plt.legend(['Tips'])
plt.grid(True)
plt.show()
```