Introduction to Data Science

Assoc. Prof. Dr. Bora Canbula

bora.canbula@cbu.edu.tr

github.com/canbula/

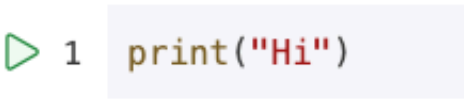
Two main types of files for Python programming

Script Files (.py)

```
print("Hi")
```

- Plain text files that contain Python code
- Designed to run from start to finish as a script

Notebook Files (.ipynb)



- Interactive files used primarily for Data Science
- Allows you to write and execute Python code in cells while combining them with markdown for notes and visualizations

Variables are symbols for memory addresses

Data Types

- Integer (3, 7, -5)
- Float (3.14, -0.01)
- Complex (3 + 5j)
- Boolean (True / False)
- Strings (“Data Science”)

Sequences

- List: [1, 2, 3]
- Tuple: (10.0, 20.0)
- Set: {1, 2, 3}
- Dict: {‘name’: Bora, age: 39}

Rules for Identifier Names

- Names are case sensitive
- Can be a combination of letters, digits, and underscore
- Cannot start with a number, must start with a letter or an underscore
- Keywords can not be used as a name

peps.python.org

Variable Names

```
an_integer = 3
a_float = 3.14
a_complex = 3 + 7j
a_bool = False
a_string = "Data Science"
a_list = [1, 2, 3]
a_tuple = (1, 2, 3)
a_set = {1, 2, 3}
a_dict = {"a": 1, "b": 2}
```

Slicing Notation

```
name = "Python Basics"
#####   0123456789012
print(name[0])  # P
print(name[7])  # B
print(len(name))  # 13
print(name[len(name)-1])  # s
print(name[-1])  # s
print(name[-3])  # i
print(name[-len(name)])  # P
print(name[0:6])  # Python
print(name[:6])  # Python
print(name[7:])  # Basics
print(name[:])  # Python Basics (copy)
print(name[0:6:2])  # Pto
print(name[8:1:-1])  # aB noht
print(name[::-1])  # scisaB nohtyP
```

Loops in Python

for Loop

Used to iterate over a sequence

for

```
sequence = [1, 2, 3, 4, 5]
for item in sequence:
    print(item)
```

range

```
for i in range(2, 10, 2):
    # Start with 2
    # Stop before 10
    # Step by 2
    print(i)
```

while Loop

Repeats as long as a condition True

while

```
count = 0
while count < 5:
    print(count)
    count += 1
```

break / continue

```
i = -1
while True:
    i += 1
    if i < 2:
        continue
    if i >= 10:
        break
    if i % 2 == 0:
        print(i)
```

list comprehension

```
squares = [x**2 for x in range(10) if x % 2 == 0]
squares_ = [
    x**2
    for x in range(10)
    if x % 2 == 0
]
```

Introductory Python : Data Structures in Python

Assoc. Prof. Dr. Bora Canbula
Manisa Celal Bayar University

Lists in Python:

Ordered and mutable sequence of values indexed by integers

Initializing
```
a_list = [] ## empty
a_list = list() ## empty
a_list = [3, 4, 5, 6, 7] ## filled
```
Finding the index of an item
```
a_list.index(5) ## 2 (the first occurence)
```
Accessing the items
```
a_list[0] ## 3
a_list[1] ## 4
a_list[-1] ## 7
a_list[-2] ## 6
a_list[2:] ## [5, 6, 7]
a_list[:2] ## [3, 4]
a_list[1:4] ## [4, 5, 6]
a_list[0:4:2] ## [3, 5]
a_list[4:1:-1] ## [7, 6, 5]
```
Adding a new item
```
a_list.append(9) ## [3, 4, 5, 6, 7, 9]
a_list.insert(2, 8) ## [3, 4, 8, 5, 6, 7, 9]
```
Update an item
```
a_list[2] = 1 ## [3, 4, 1, 5, 6, 7, 9]
```
Remove the list or just an item
```
a_list.pop() ## last item
a_list.pop(2) ## with index
del a_list[2] ## with index
a_list.remove(5) ## first occurence of 5
a_list.clear() ## returns an empty list
del a_list ## removes the list completely
```
Extend a list with another list
```
list_1 = [4, 2]
list_2 = [1, 3]
list_1.extend(list_2) ## [4, 2, 1, 3]
```
Reversing and sorting
```
list_1.reverse() ## [3, 1, 2, 4]
list_1.sort() ## [1, 2, 3, 4]
```
Counting the items
```
list_1.count(4) ## 1
list_1.count(5) ## 0
```
Copying a list
```
list_1 = [3, 4, 5, 6, 7]
list_2 = list_1
list_3 = list_1.copy()
list_1.append(1)
list_2 ## [3, 4, 5, 6, 7, 1]
list_3 ## [3, 4, 5, 6, 7]
```

Sets in Python:

Unordered and mutable collection of values with no duplicate elements. They support mathematical operations like union, intersection, difference and symmetric difference

Initializing
```
a_set = set() ## empty
a_set = {3, 4, 5, 6, 7} ## filled
```
No duplicate values
```
a_set = {3, 3, 3, 4, 4} ## {3, 4}
```
Adding and updating the items
```
a_set.add(5) ## {3, 4, 5}
set_1 = {1, 3, 5}
set_2 = {5, 7, 9}
set_1.update(set_2) ## {1, 3, 5, 7, 9}
```
Removing the items
```
a_set.pop() ## removes an item and returns it
a_set.remove(3) ## removes the item
a_set.discard(3) ## removes the item
```
If item does not exist in set,
remove() raises an error, discard() does not
```
a_set.clear() ## returns an empty set
del a_set ## removes the set completely
```
Mathematical operations
```
set_1 = {1, 2, 3, 5}
set_2 = {1, 2, 4, 6}
```
Union of two sets
```
set_1.union(set_2) ## {1, 2, 3, 4, 5, 6}
set_1 | set_2 ## {1, 2, 3, 4, 5, 6}
```
Intersection of two sets
```
set_1.intersection(set_2) ## {1, 2}
set_1 & set_2 ## {1, 2}
```
Difference between two sets
```
set_1.difference(set_2) ## {3, 5}
set_2.difference(set_1) ## {4, 6}
set_1 - set_2 ## {3, 5}
set_2 - set_1 ## {4, 6}
```
Symmetric difference between two sets
```
set_1.symmetric_difference(set_2) ## {3,4,5,6}
set_1 ^ set_2 ## {3, 4, 5, 6}
```
Update sets with mathematical operations
```
set_1.intersection_update(set_2) ## {1, 2}
set_1.difference_update(set_2) ## {3, 5}
set_1.symmetric_difference_update(set_2)
## {3, 4, 5, 6}
```
Copying a set

Same as lists

Dictionaries in Python:

Unordered and mutable set of key-value pairs

Initializing
```
a_dict = {} ## empty
a_dict = dict() ## empty
a_dict = {"name":"Bora"} ## filled
```
Accessing the items
```
a_dict["name"] ## "Bora"
a_dict.get("name") ## "Bora"
```
If the key does not exist in dictionary,
index notation raises an error, get() method does not

Accessing the items with views
```
other_dict = {"a": 3, "b": 5, "c": 7}
other_dict.keys() ## ['a', 'b', 'c']
other_dict.values() ## [3, 5, 7]
other_dict.items()
## [('a', 3), ('b', 5), ('c', 7)]
```
Adding a new item
```
a_dict["city"] = "Manisa"
a_dict["age"] = 37
## {"name":"Bora", "city":"Manisa", "age":37}
```
Update an item
```
a_dict["age"] = 38
## {"name":"Bora", "city":"Manisa", "age":38}
other_dict = {"age":39}
a_dict.update(other_dict)
## {"name":"Bora", "city":"Manisa", "age":39}
```
Removing the items
```
a_dict.popitem() ## last inserted item
a_dict.pop("city") ## with a key
a_dict.clear() ## returns an empty dictionary
del a_dict ## removes the dict completely
```
Initialize a dictionary with fromkeys
```
a_list = ['a', 'b', 'c']
a_dict = dict.fromkeys(a_list)
## {'a': None, 'b': None, 'c': None}
a_dict = dict.fromkeys(a_list, 0)
## {'a': 0, 'b': 0, 'c': 0}
a_tuple = (3, 'name', 7)
a_dict = dict.fromkeys(a_tuple, True)
## {3: True, 'name': True, 7: True}
a_set = {0, 1, 2}
a_dict = dict.fromkeys(a_set, False)
## {0: False, 1: False, 2: False}
```

Tuples in Python:

Ordered and immutable sequence of values indexed by integers

Initializing
```
a_tuple = () ## empty
a_tuple = tuple() ## empty
a_tuple = (3, 4, 5, 6, 7) ## filled
```
Finding the index of an item
```
a_tuple.index(5) ## 2 (the first occurence)
```
Accessing the items

Same index and slicing notation as lists

Adding, updating, and removing the items

Not allowed because tuples are immutable

Sorting

Tuples have no sort() method since they are immutable
```
sorted(a_tuple) ## returns a sorted list
```
Counting the items
```
a_tuple.count(7) ## 1
a_tuple.count(9) ## 0
```

Some iteration examples:

```
a_list = [3, 5, 7]
a_tuple = (4, 6, 8)
a_set = {1, 4, 7}
a_dict = {"a":1, "b":2, "c":3}
```
For ordered sequences
```
for i in range(len(a_list)):
    print(a_list[i])
for i, x in enumerate(a_tuple):
    print(i, x)
```
For ordered or unordered sequences
```
for a in a_set:
    print(a)
```
Only for dictionaries
```
for k in a_dict.keys():
    print(k)
for v in a_dict.values():
    print(v)
for k,v in zip(a_dict.keys(),a_dict.values()):
    print(k, v)
for k, v in a_dict.items():
    print(k, v)
```

Formatted Output

f-strings

```
name = "Bora"
age = 39
print(f"My name is {name} and I am {age:2d} years old.")
print(f"|{'Left':<10}|{'Right':>10}|{'Center':^10}|")
pi = 3.141592653589793
print(f"Pi is approximately {pi:.2f}.")
print(f"Pi is approximately {pi:.3f}.")
print(f"Pi is approximately {pi:.6e}.")
```

Functions

Basic Definition

```
def fn():
    print("Hi")

def squares(input_):
    return [x**2 for x in input_]
```

Default Values

```
def say_hello(from_=None, to="World"):
    if from_ is None:
        print(f"Hello, {to}!")
    else:
        print(f"Hello, {to}! This is {from_}.")
```

Type Annotations

```
def divide(x: int, y: int) -> float:
    """Divide x by y."""
    return x / y
```

Anonymous Functions

```
roots = lambda x: x**0.5
```

Classes

Defining

```
class FacultyMember:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Inheriting

```
class Professor(FacultyMember):
    def __init__(self, name, age, title):
        super().__init__(name, age)
        self.title = title if title else "Dr."

    def say_hello(self, to="World"):
        print(f"Hello, {to}! This is {self.title} {self.name}.")
```

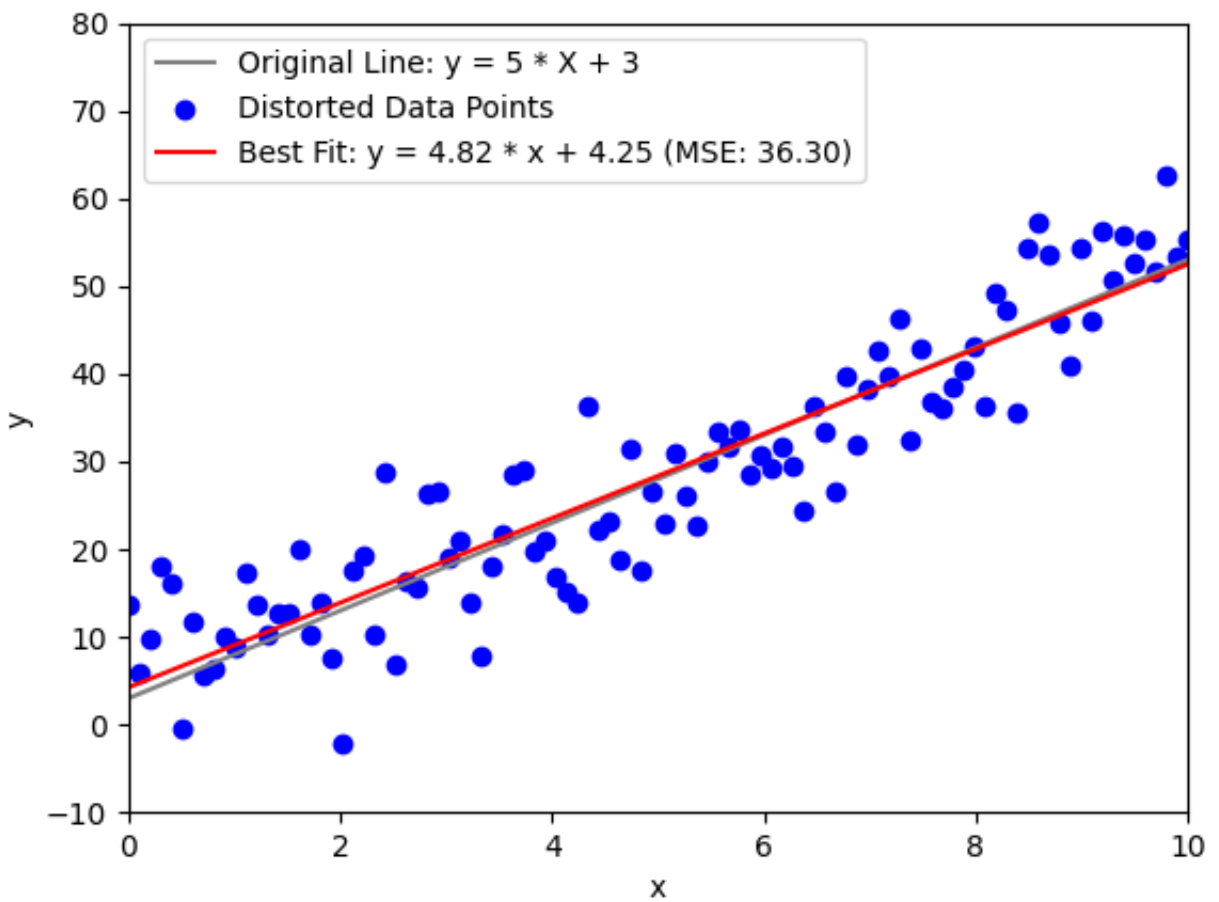
NumPy - Numerical Python

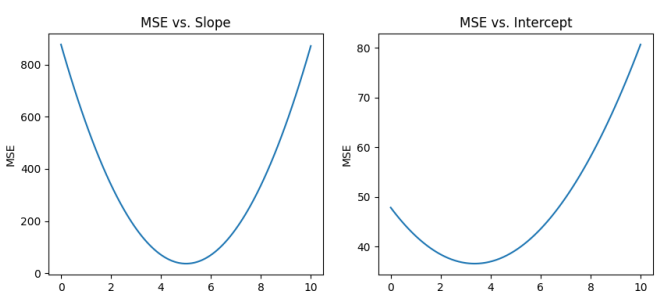
n-dim Arrays

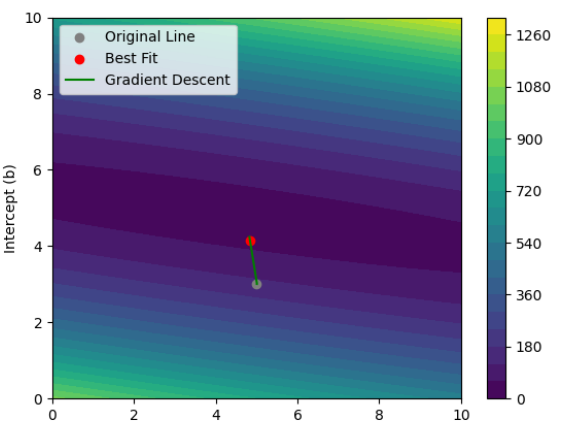
```python
arr = np.array([1, 2, 3, 4, 5])
print(arr)
# [1 2 3 4 5]
#
print(arr.dtype)  # int64
print(arr.shape)  # (5,)
print(arr.ndim)  # 1
print(arr.size)  # 5
```

```python
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
# [[1 2 3]
#  [4 5 6]]
print(arr.dtype)  # int64
print(arr.shape)  # (2, 3)
print(arr.ndim)  # 2
print(arr.size)  # 6
```

Example: Linear Regression with NumPy







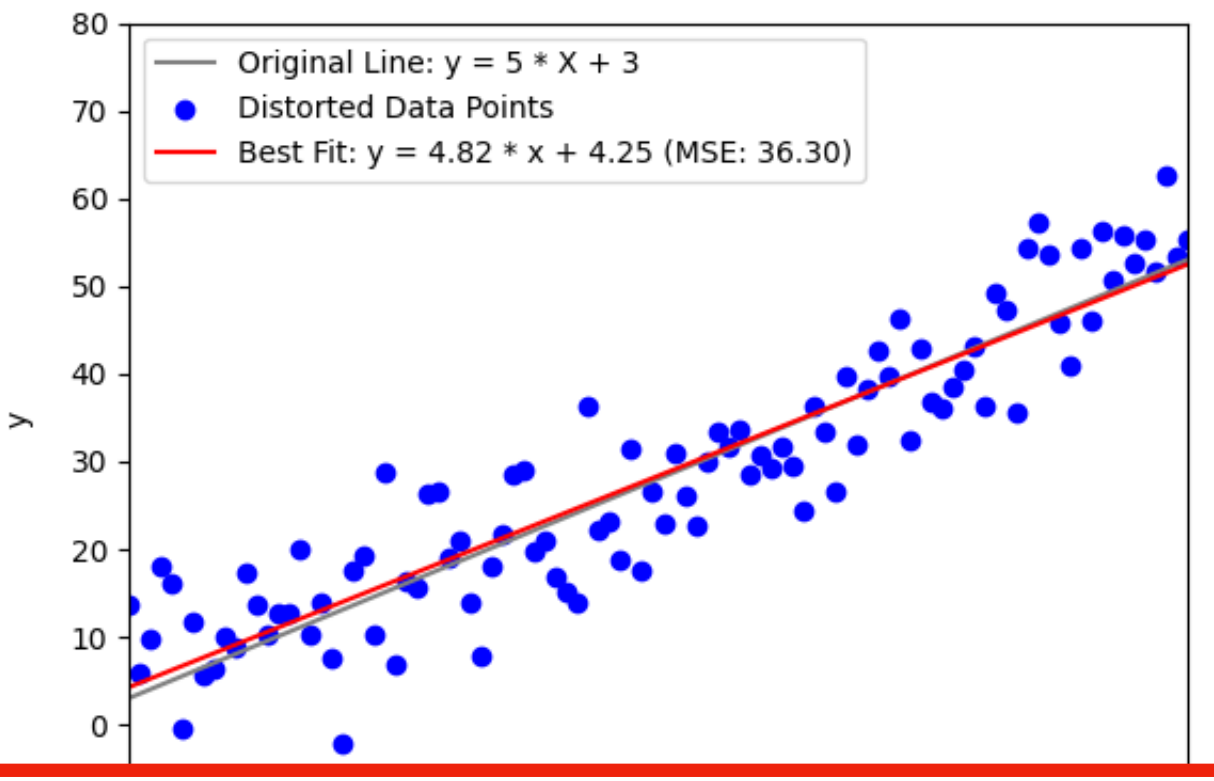
NumPy - Numerical Python

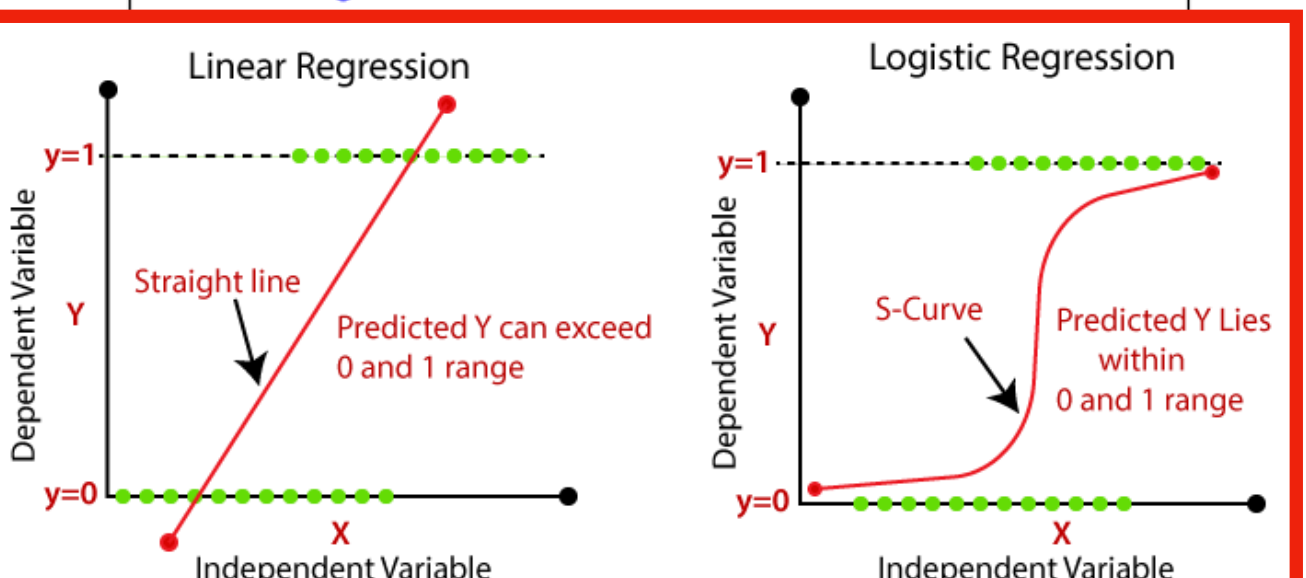
n-dim Arrays

```python
arr = np.array([1, 2, 3, 4, 5])
print(arr)
# [1 2 3 4 5]
#
print(arr.dtype)  # int64
print(arr.shape)  # (5,)
print(arr.ndim)  # 1
print(arr.size)  # 5
```

```python
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
# [[1 2 3]
#  [4 5 6]]
print(arr.dtype)  # int64
print(arr.shape)  # (2, 3)
print(arr.ndim)  # 2
print(arr.size)  # 6
```

Example: Linear Regression with NumPy





Pandas - Data Manipulation and Analysis

Series

```python
a_list = [1, 3, 5, np.nan, 6, 8]
a_pd_series = pd.Series(a_list)
# 0    1.0
# 1    3.0
# 2    5.0
# 3    NaN
# 4    6.0
# 5    8.0
print(a_pd_series[0])  # 1.0
print(a_pd_series.mean())  # 4.6
print(a_pd_series.median())  # 5.0
print(a_pd_series.std())  # 2.70
```

DataFrame

```python
dates = pd.date_range("20250101", periods=4)
a_dict = {
    "Code": ["06", "34", "35", "45"],
    "City": ["Ankara", "Istanbul", "Izmir", "Manisa"],
    "Population": [5000000, 15000000, 4000000, 1000000],
}
df = pd.DataFrame(a_dict, index=dates)
#             Code      City  Population
# 2025-01-01   06    Ankara     5000000
# 2025-01-02   34  Istanbul    15000000
# 2025-01-03   35     Izmir     4000000
# 2025-01-04   45    Manisa     1000000
```

Accessing Columns and Rows

```python
# Single column
print(df["City"])
# 2025-01-01       Ankara
# 2025-01-02     Istanbul
# 2025-01-03        Izmir
# 2025-01-04       Manisa
```

```python
# Multiple columns
print(df[["City", "Population"]])
#                 City  Population
# 2025-01-01    Ankara     5000000
# 2025-01-02  Istanbul    15000000
# 2025-01-03     Izmir     4000000
# 2025-01-04    Manisa     1000000
```

```python
# Slicing rows
print(df[0:2])
#             Code      City  Population
# 2025-01-01   06    Ankara     5000000
# 2025-01-02   34  Istanbul    15000000
```

```python
# Slicing rows and columns
print(df.iloc[0:2, 1:3])
#                 City  Population
# 2025-01-01    Ankara     5000000
# 2025-01-02  Istanbul    15000000
```

Add / Remove / Update a Column

```python
df["Area"] = [2000, 5000, 3000, 1000]
#             Code      City  Population  Area
# 2025-01-01   06    Ankara     5000000  2000
# 2025-01-02   34  Istanbul    15000000  5000
# 2025-01-03   35     Izmir     4000000  3000
# 2025-01-04   45    Manisa     1000000  1000
```

```python
df = df.drop(columns=["Area"])
#             Code      City  Population
# 2025-01-01   06    Ankara     5000000
# 2025-01-02   34  Istanbul    15000000
# 2025-01-03   35     Izmir     4000000
# 2025-01-04   45    Manisa     1000000
```

```python
df["Population"] = df["Population"] / 1000000
df["City"] = df["City"].str.upper()
#             Code      City  Population
# 2025-01-01   06    ANKARA         5.0
# 2025-01-02   34  ISTANBUL        15.0
# 2025-01-03   35     IZMIR         4.0
# 2025-01-04   45    MANISA         1.0
```

Filtering and Sorting

```python
df = df[df["Population"] > 5000000]
#             Code      City  Population
# 2025-01-02   34  Istanbul    15000000
```

```python
df.sort_values(
    by="Population",
    ascending=False,
    inplace=True
)
#             Code      City  Population
# 2025-01-02   34  Istanbul    15000000
# 2025-01-01   06    Ankara     5000000
# 2025-01-03   35     Izmir     4000000
# 2025-01-04   45    Manisa     1000000
```

Loading / Saving Data

```python
df = pd.read_csv("data.csv")
df.to_csv("data.csv", index=False)
```

Project: Airline Passenger Satisfaction

Data

test.csv
train.csv

kaggle

Gender: Gender of the passengers (Female, Male)

Customer Type: The customer type (Loyal customer, disloyal customer)

Age: The actual age of the passengers

Type of Travel: Purpose of the flight of the passengers (Personal Travel, Business Travel)

Class: Travel class in the plane of the passengers (Business, Eco, Eco Plus)

Flight distance: The flight distance of this journey

Inflight wifi service: Satisfaction level of the inflight wifi service (0:Not Applicable;1-5)

Departure/Arrival time convenient: Satisfaction level of Departure/Arrival time convenient

Ease of Online booking: Satisfaction level of online booking

Gate location: Satisfaction level of Gate location

Food and drink: Satisfaction level of Food and drink

Online boarding: Satisfaction level of online boarding

Seat comfort: Satisfaction level of Seat comfort

Inflight entertainment: Satisfaction level of inflight entertainment

On-board service: Satisfaction level of On-board service

Leg room service: Satisfaction level of Leg room service

Baggage handling: Satisfaction level of baggage handling

Check-in service: Satisfaction level of Check-in service

Inflight service: Satisfaction level of inflight service

Cleanliness: Satisfaction level of Cleanliness

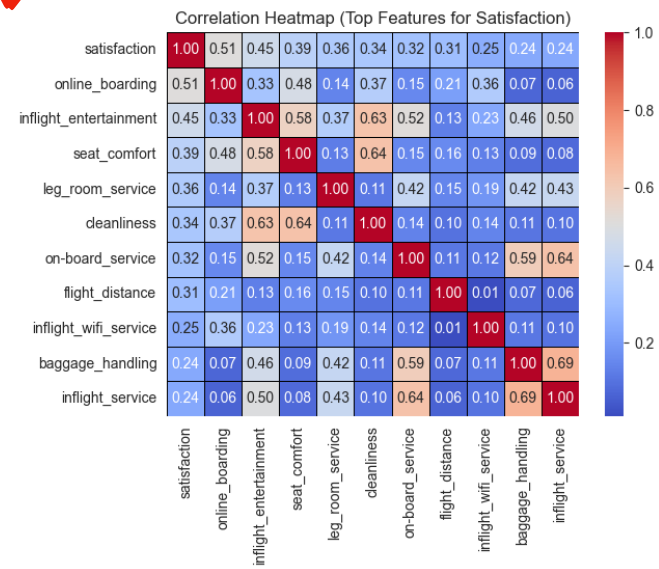
Departure Delay in Minutes: Minutes delayed when departure

Arrival Delay in Minutes: Minutes delayed when Arrival

Satisfaction: Airline satisfaction level(Satisfaction, neutral or dissatisfaction)

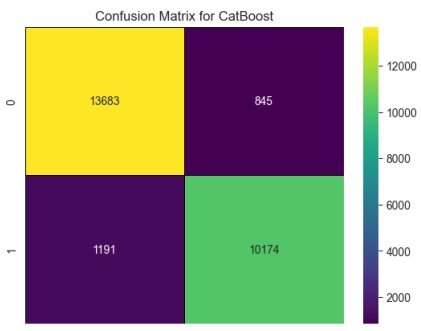
Data Science Workflow

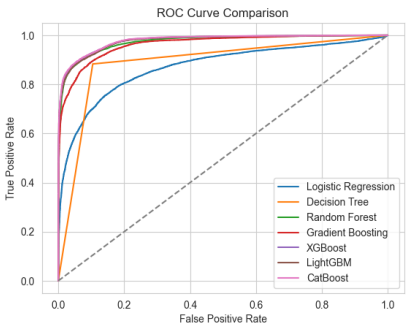
- ✓ **Data Loading and Overview**
- ✓ **Exploratory Data Analysis (EDA)**
- ✓ **Data Preprocessing**
- ✓ **Model Training**
- ✓ **Evaluation and Prediction**

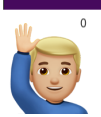


	precision	recall	f1-score	support
0	0.92	0.94	0.93	14528
1	0.92	0.90	0.91	11365
accuracy			0.92	25893
macro avg	0.92	0.92	0.92	25893
weighted avg	0.92	0.92	0.92	25893

```
Accuracy for Logistic Regression: 0.81
Accuracy for Decision Tree: 0.89
Accuracy for Random Forest: 0.92
Accuracy for Gradient Boosting: 0.90
Accuracy for XGBoost: 0.92
Accuracy for LightGBM: 0.92
Accuracy for CatBoost: 0.92
```







Good luck, see you again

Precision

The proportion of true positive predictions out of all positive predictions made by the model.

Recall

The proportion of true positive predictions out of all actual positive cases.

F1-Score

The harmonic mean of precision and recall, balancing the two metrics.

Accuracy

The proportion of all correctly classified instances out of the total instances.