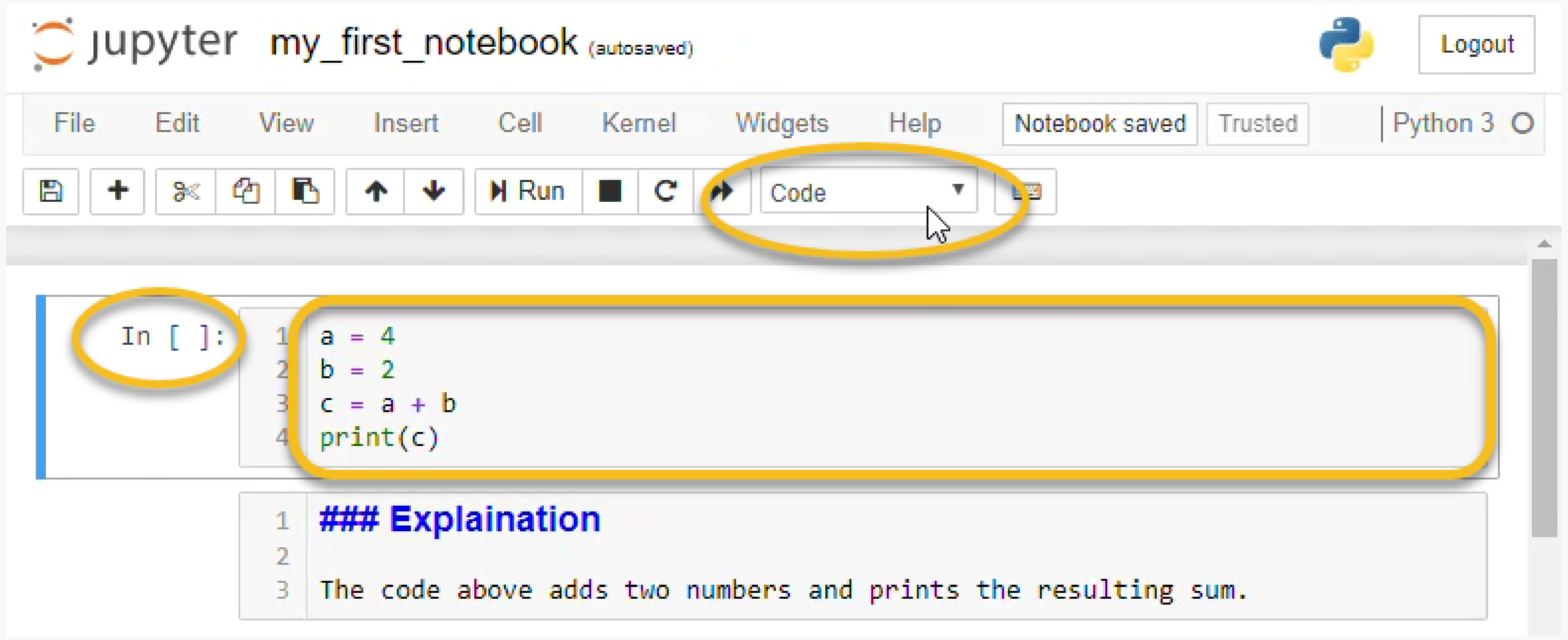
Machine learning algorithms: Practice 1.

Python libraries for ML
Classification algorithms

Alexandr Gavrilko
MLA Course for CS-2227

Developing environment: Interactive notebooks


jupyter my_first_notebook (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Notebook saved Trusted Python 3
Run Code
In [ ]:

```
a = 4
b = 2
c = a + b
print(c)
```

```
### Explaination

The code above adds two numbers and prints the resulting sum.
```

Jupyter Notebook / Jupyter Lab

Self-hosted / Cloud

```
> pip install jupyterlab
> jupyter lab

> pip install notebook
> jupyter notebook
```

Google Colab

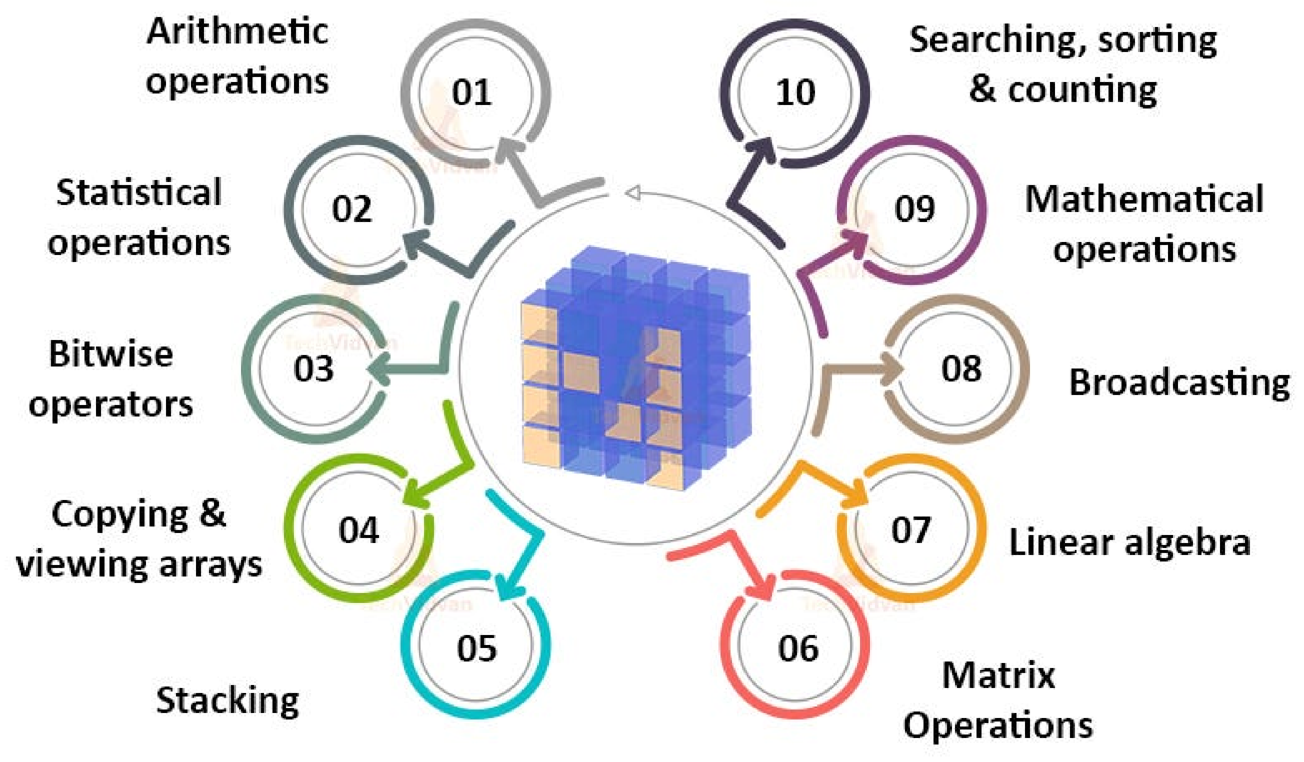
Cloud

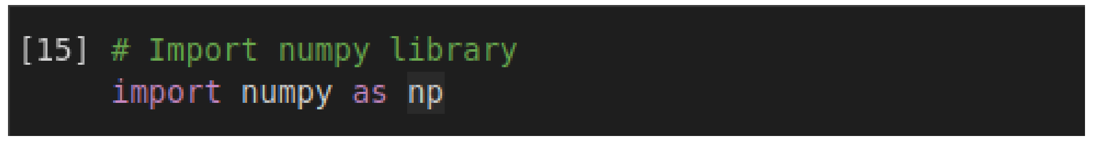
Python libraries for Machine Learning

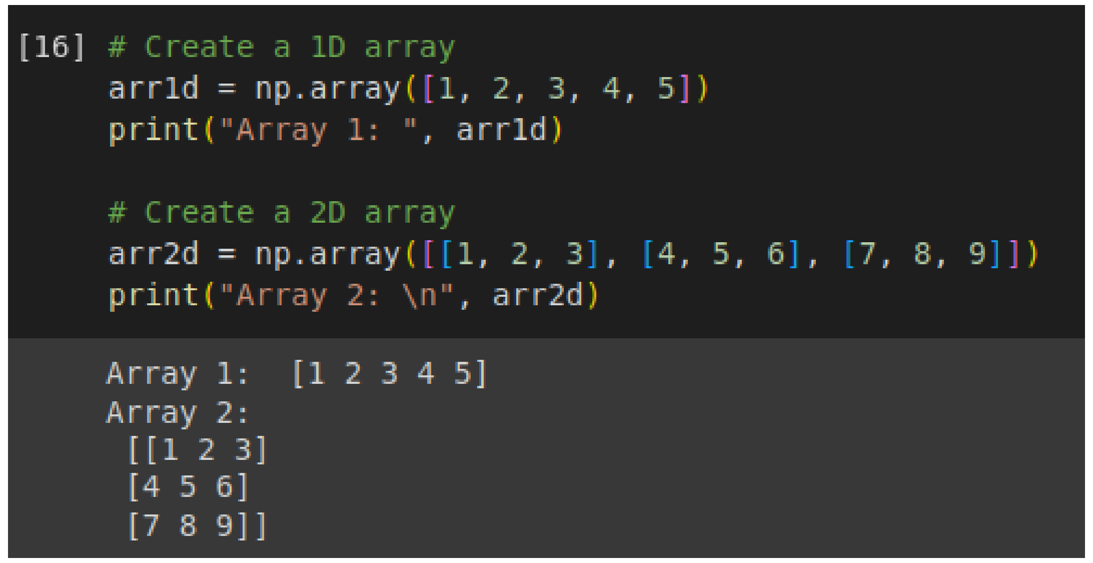
NumPy

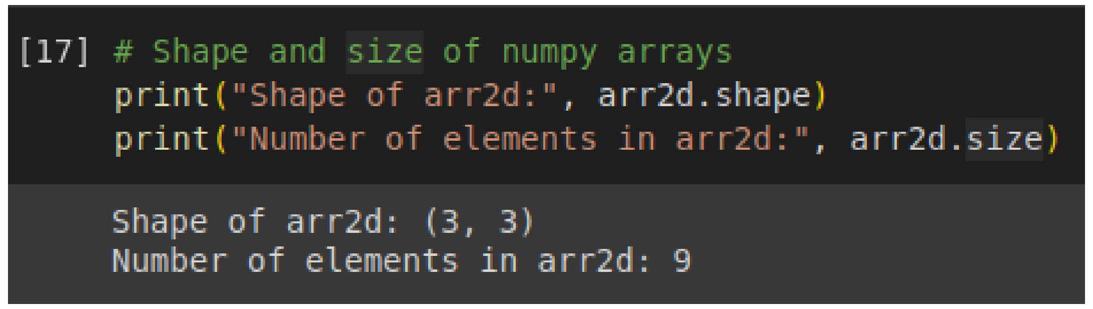
> pip install numpy

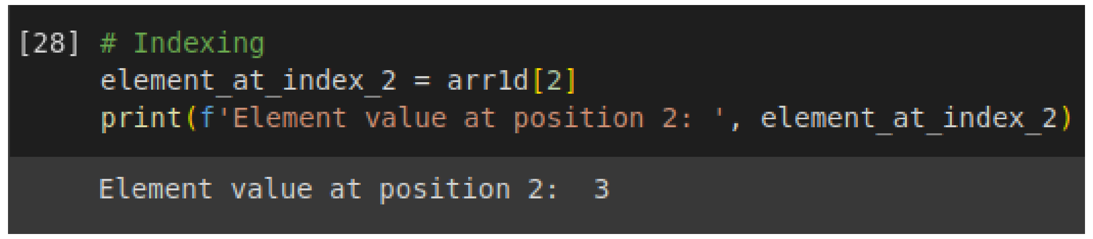
Uses of NumPy

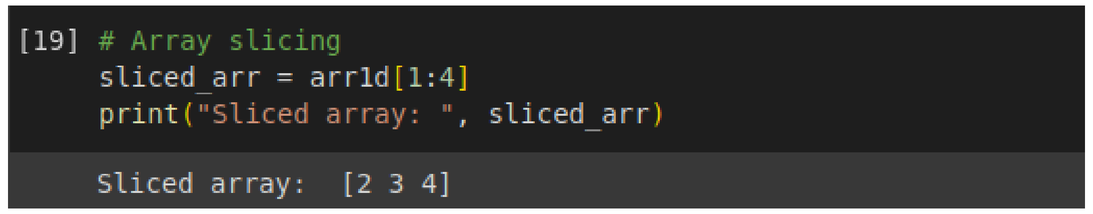


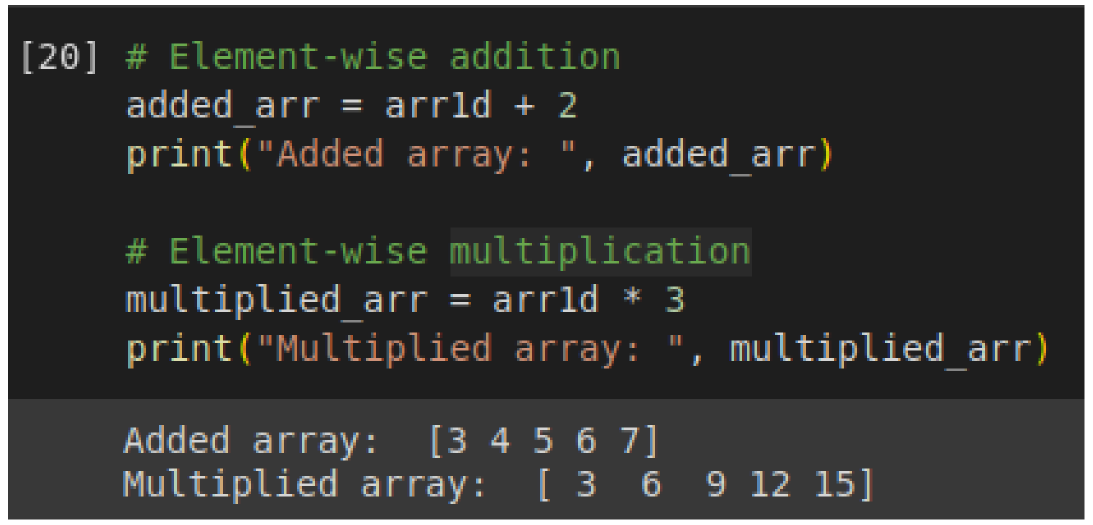

```
[15] # Import numpy library
     import numpy as np
```

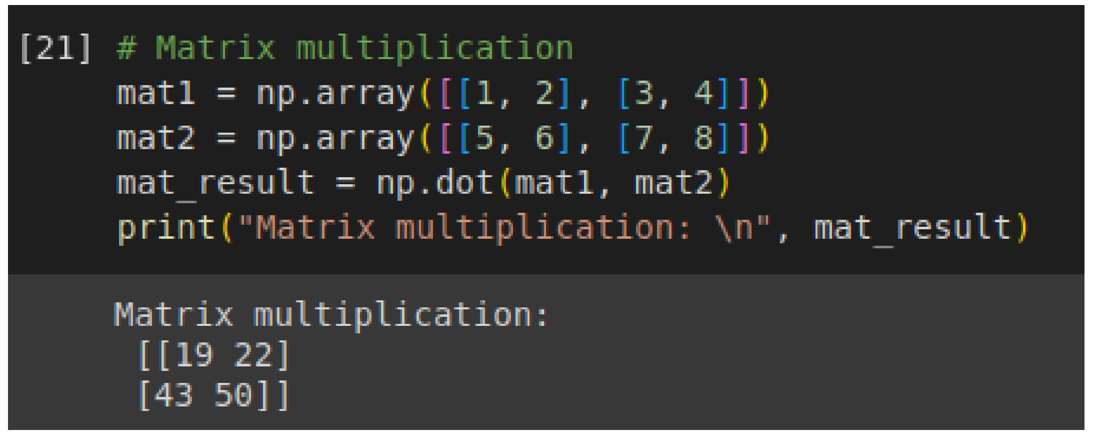

```
[16] # Create a 1D array
     arr1d = np.array([1, 2, 3, 4, 5])
     print("Array 1: ", arr1d)

     # Create a 2D array
     arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
     print("Array 2: \n", arr2d)

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

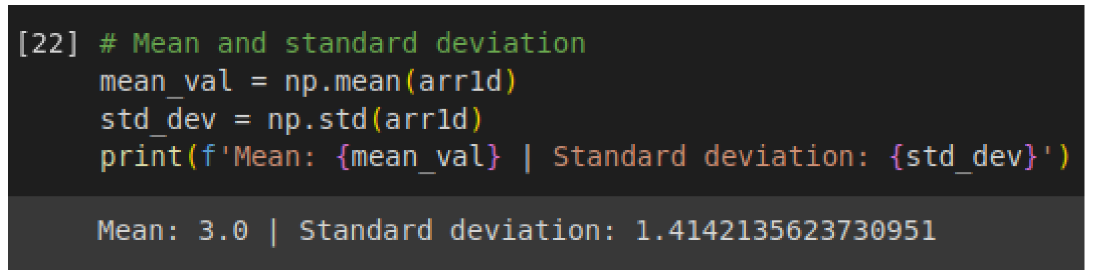

```
[17] # Shape and size of numpy arrays
     print("Shape of arr2d:", arr2d.shape)
     print("Number of elements in arr2d:", arr2d.size)

     Shape of arr2d: (3, 3)
     Number of elements in arr2d: 9
```

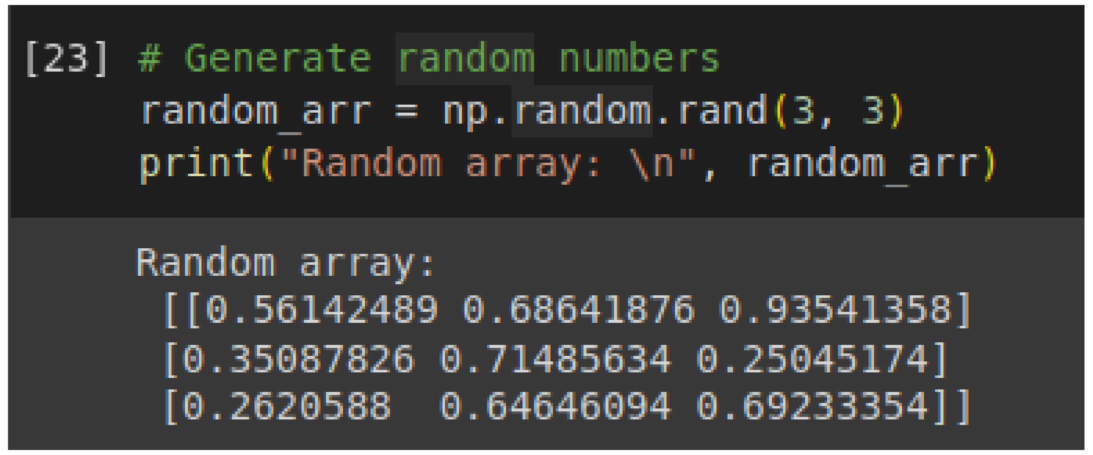

```
[28] # Indexing
     element_at_index_2 = arr1d[2]
     print(f'Element value at position 2: ', element_at_index_2)

     Element value at position 2:  3
```


```
[19] # Array slicing
     sliced_arr = arr1d[1:4]
     print("Sliced array: ", sliced_arr)

     Sliced array:  [2 3 4]
```


```
[20] # Element-wise addition
     added_arr = arr1d + 2
     print("Added array: ", added_arr)

     # Element-wise multiplication
     multiplied_arr = arr1d * 3
     print("Multiplied array: ", multiplied_arr)

     Added array:  [3 4 5 6 7]
     Multiplied array:  [ 3  6  9 12 15]
```


```
[21] # Matrix multiplication
     mat1 = np.array([[1, 2], [3, 4]])
     mat2 = np.array([[5, 6], [7, 8]])
     mat_result = np.dot(mat1, mat2)
     print("Matrix multiplication: \n", mat_result)

     Matrix multiplication:
      [[19 22]
      [43 50]]
```

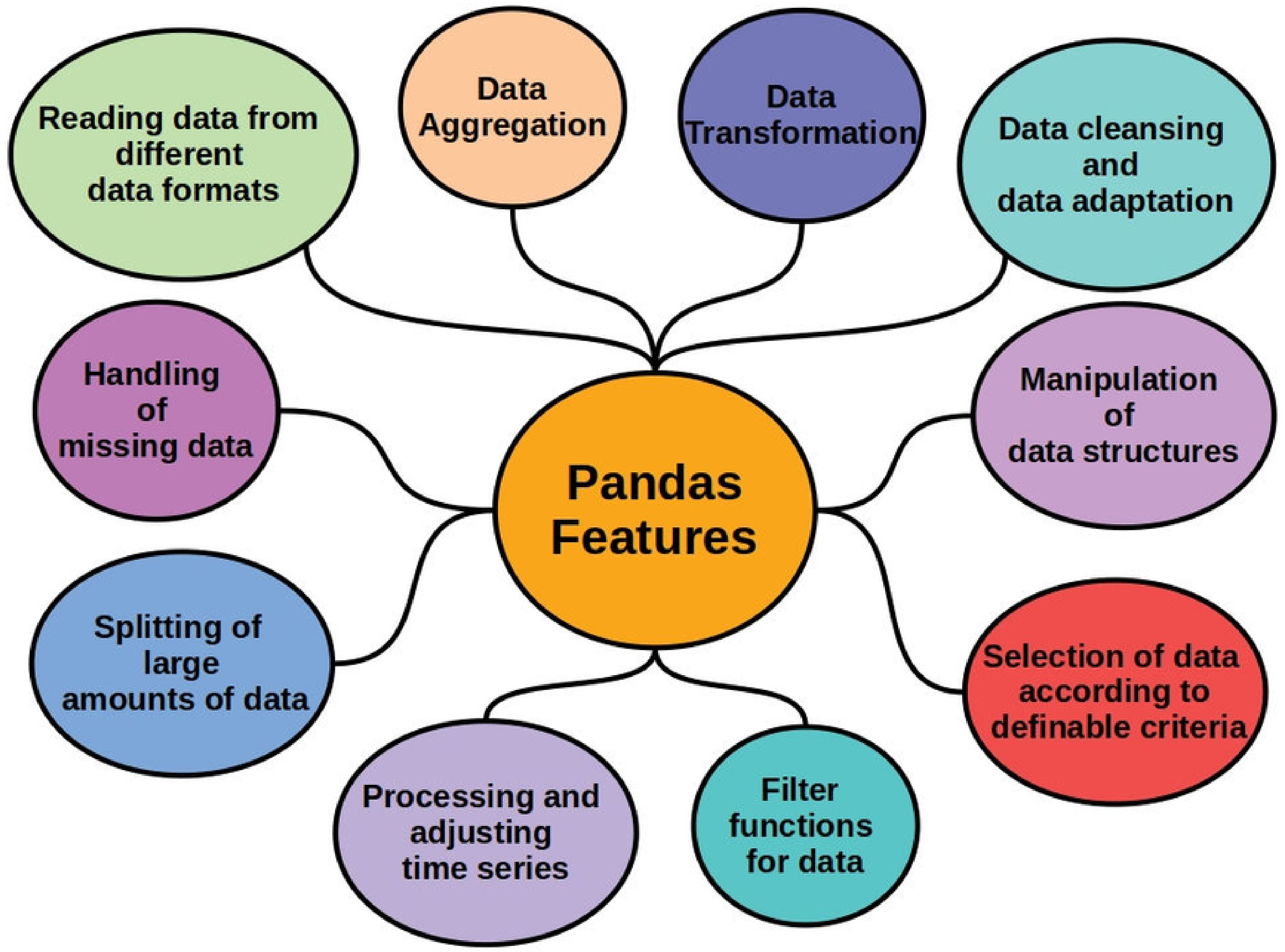

```
[22] # Mean and standard deviation
     mean_val = np.mean(arr1d)
     std_dev = np.std(arr1d)
     print(f'Mean: {mean_val} | Standard deviation: {std_dev}')

     Mean: 3.0 | Standard deviation: 1.4142135623730951
```

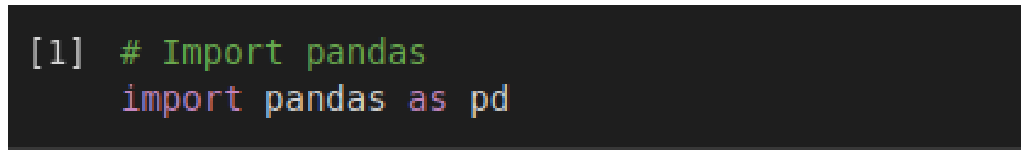

```
[23] # Generate random numbers
     random_arr = np.random.rand(3, 3)
     print("Random array: \n", random_arr)

     Random array:
      [[0.56142489 0.68641876 0.93541358]
      [0.35087826 0.71485634 0.25045174]
      [0.2620588  0.64646094 0.69233354]]
```

Python libraries for Machine Learning

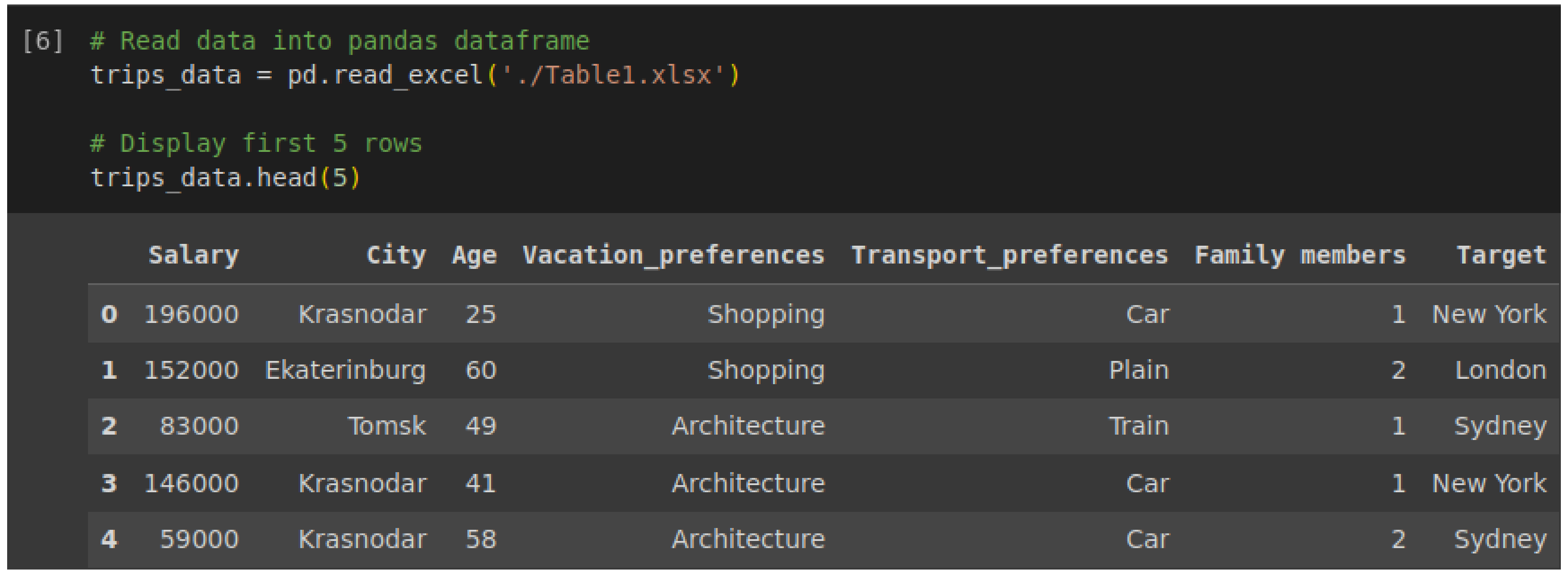
pandas

> pip install pandas


Pandas Features
Reading data from different data formats
Data Aggregation
Data Transformation
Data cleansing and data adaptation
Handling of missing data
Manipulation of data structures
Splitting of large amounts of data
Selection of data according to definable criteria
Processing and adjusting time series
Filter functions for data

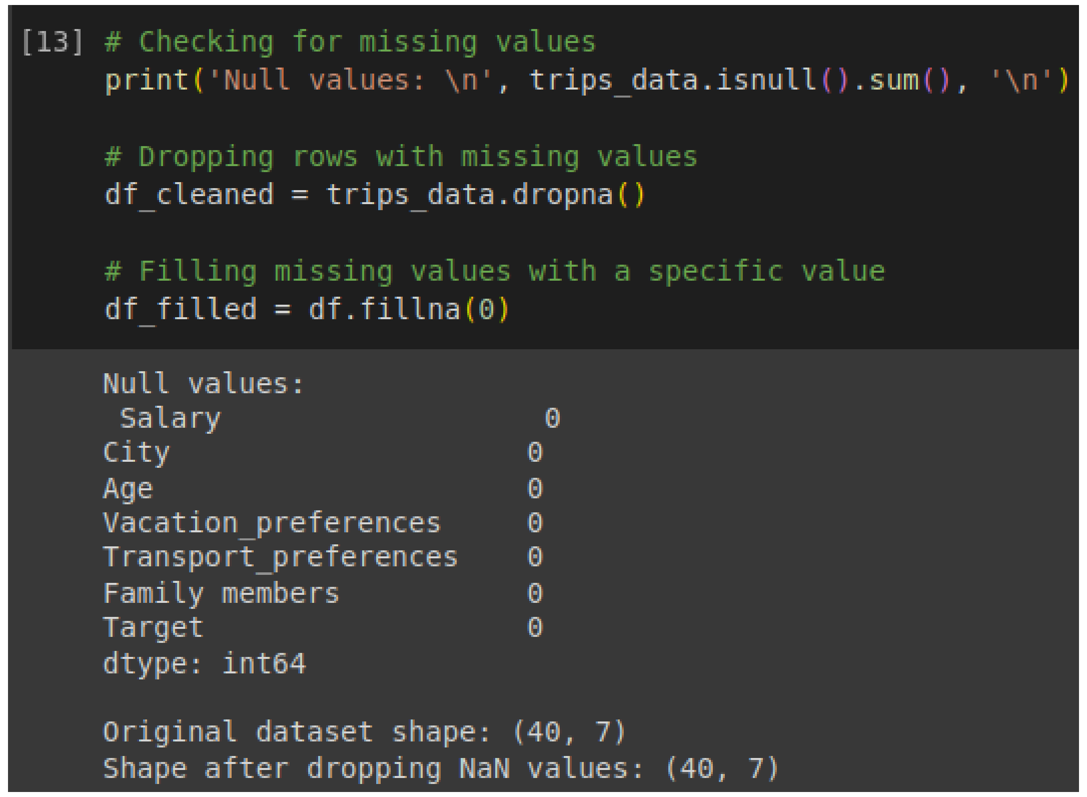


```
[1]  # Import pandas
     import pandas as pd
```



```
[6]  # Read data into pandas dataframe
     trips_data = pd.read_excel('./Table1.xlsx')

     # Display first 5 rows
     trips_data.head(5)
```

	Salary	City	Age	Vacation_preferences	Transport_preferences	Family members	Target
0	196000	Krasnodar	25	Shopping	Car	1	New York
1	152000	Ekaterinburg	60	Shopping	Plain	2	London
2	83000	Tomsk	49	Architecture	Train	1	Sydney
3	146000	Krasnodar	41	Architecture	Car	1	New York
4	59000	Krasnodar	58	Architecture	Car	2	Sydney



```
[13] # Checking for missing values
     print('Null values: \n', trips_data.isnull().sum(), '\n')

     # Dropping rows with missing values
     df_cleaned = trips_data.dropna()

     # Filling missing values with a specific value
     df_filled = df.fillna(0)
```

```
Null values:
 Salary                   0
City                     0
Age                      0
Vacation_preferences     0
Transport_preferences    0
Family members           0
Target                   0
dtype: int64

Original dataset shape: (40, 7)
Shape after dropping NaN values: (40, 7)
```

Introduction to k-Nearest Neighbors algorithm (kNN)

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. But mostly for classification.

Main features of kNN:

Lazy learning algorithm - no training phase, just use training data to predict classes

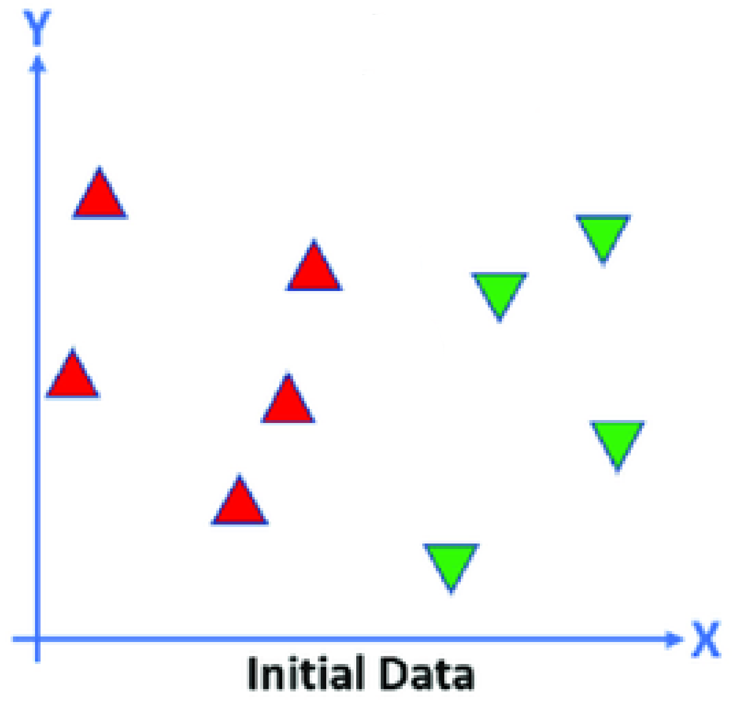
Non-parametric algorithm - algorithm doesn't assume anything about training data

Fit-predict pipeline of kNN:

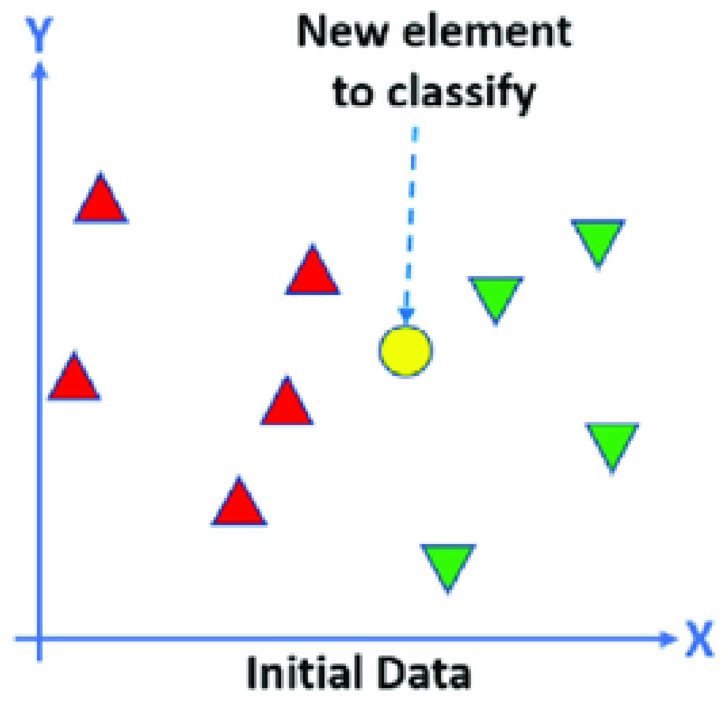
1.Set up hyper-parameters of kNN algorithm:

- k - number of neighbors
(from 1 to Num. Observations/2)
- Distance metric
(euclidean, cosine, Minkowsky, Manhattan, etc.)
- Distance weight (optional)
(equal, inverse, squared inverse)
- Standardize data (optional)
(Yes or No)

2.Fit the data into kNN algorithm:

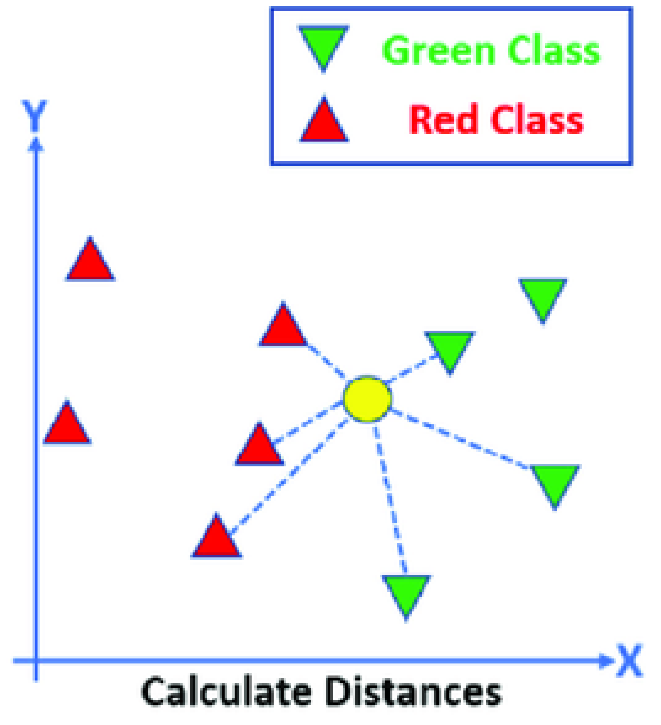


3.Put new data example with unknown class on the plot:

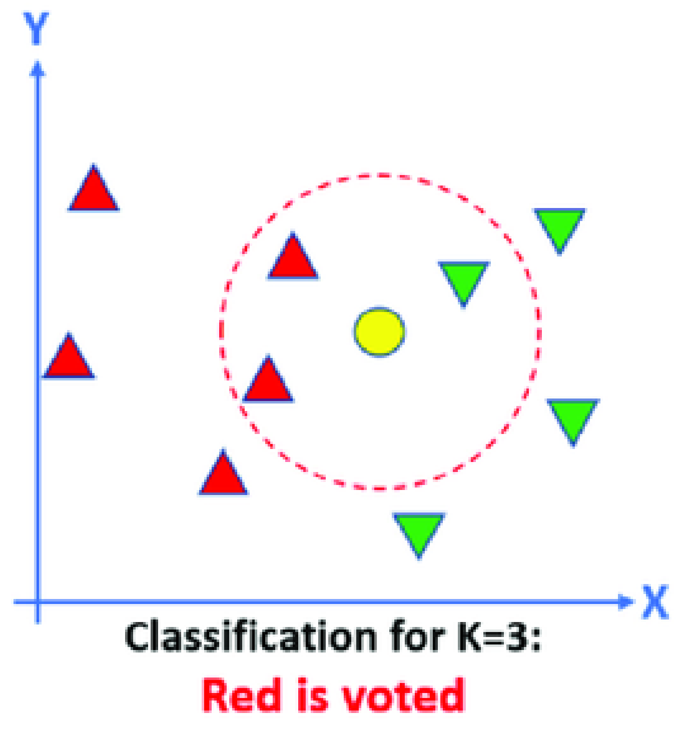


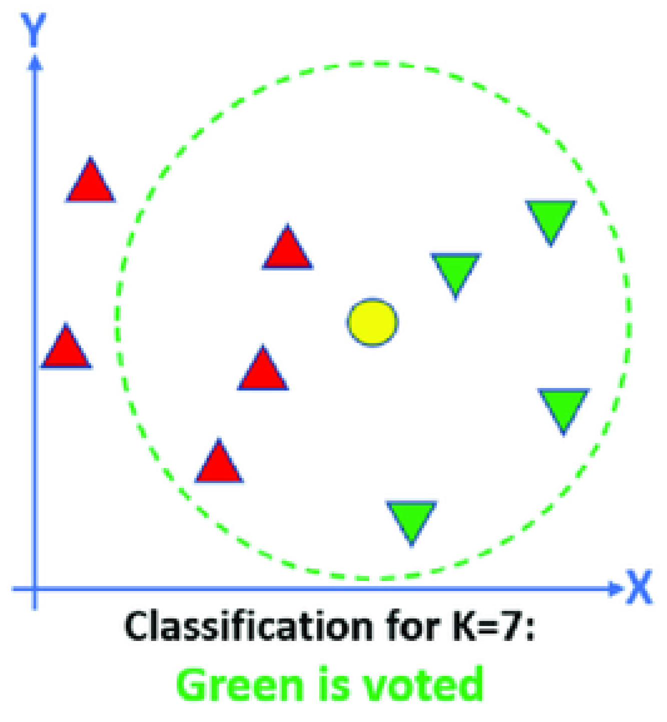
Fit-predict pipeline of kNN:

4.Calculate **distances** and find **k** nearest data points to new point:



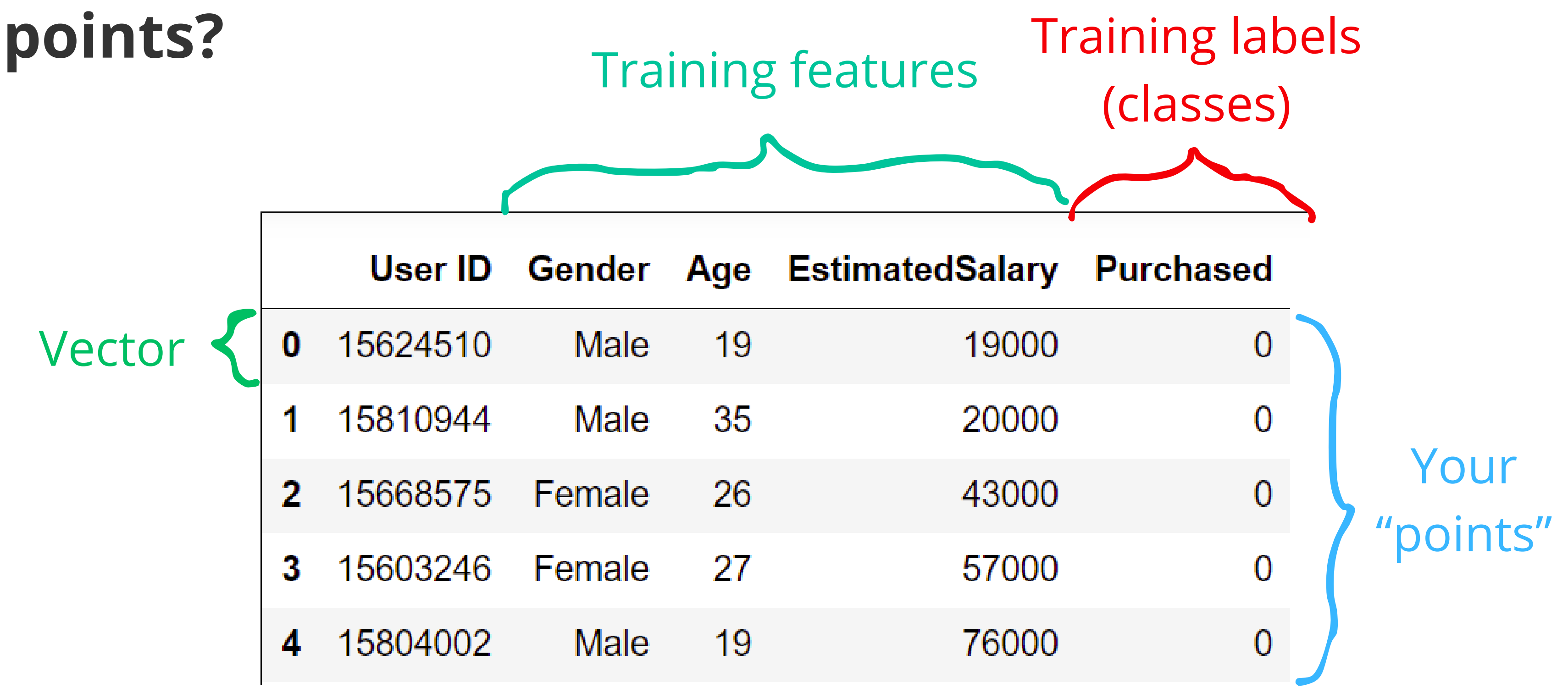
5.Define the dominant class of points which are the neighbors of your new point:





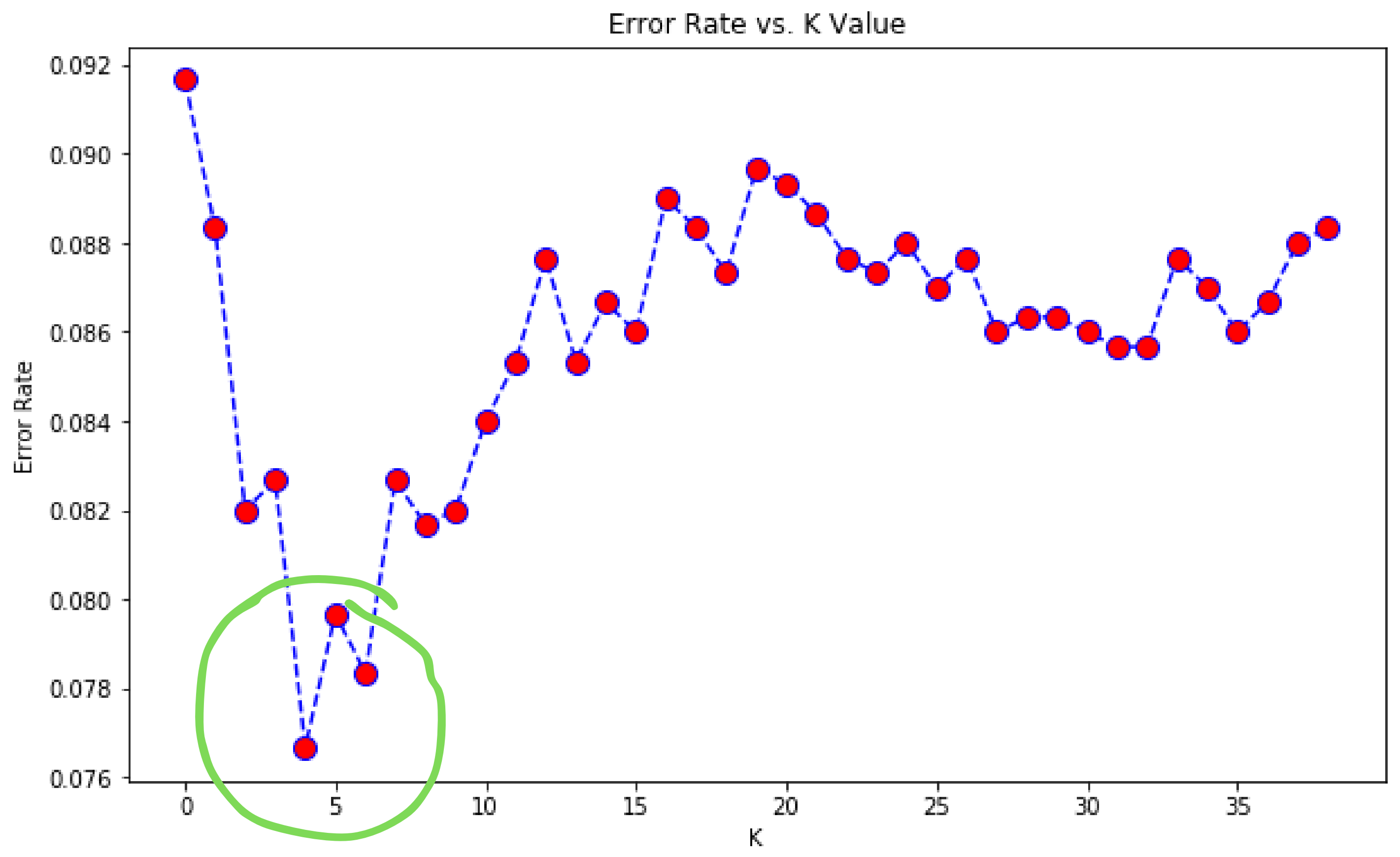
What are the data points?

- All the instances correspond to points in an n-dimensional feature space.
- Each instance is represented with a set of numerical attributes.
- Each of the training data consists of a set of vectors and a class label associated with each vector.


Training features
Training labels (classes)
Vector
Your "points"

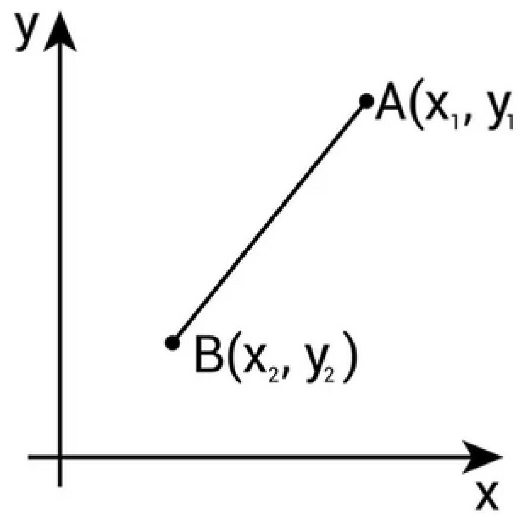
	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Find *k* hyper-parameter experimentally:
Measure accuracy on validation dataset and define the best value for *k*



Distance metrics

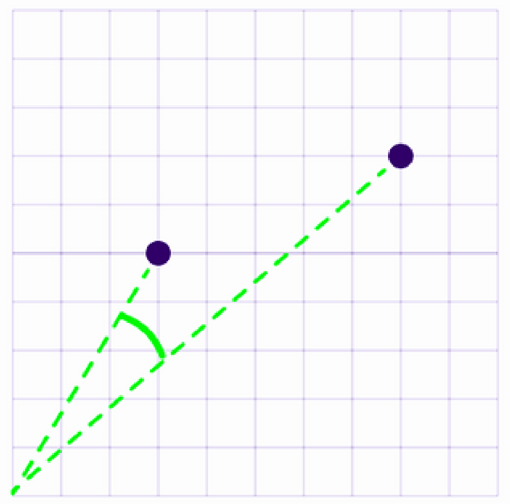
Euclidean distance:



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Works well when feature distribution is balanced
- Measures the physical distance between two points

Cosine Distance



$$1 - \frac{A \cdot B}{||A|| \quad ||B||}$$

- Works well with sparse vectors (a lot of zero values)
- Measures how closely the two vectors point in the same direction

Final notes about k-Nearest Neighbors algorithm:

- **Does not work well with large dataset:** In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm

- **Does not work well with high dimensions:** The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

- **Need feature scaling:** We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

- **Sensitive to noisy data, missing values and outliers:** KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

Introduction to k-Means algorithm (kMeans)

K-means clustering is a method for grouping n observations into K clusters. It uses vector quantization and aims to assign each observation to the cluster with the nearest mean or centroid.

Main features of kMeans:

Eager learner - there is training phase, during which algorithm group data by clusters

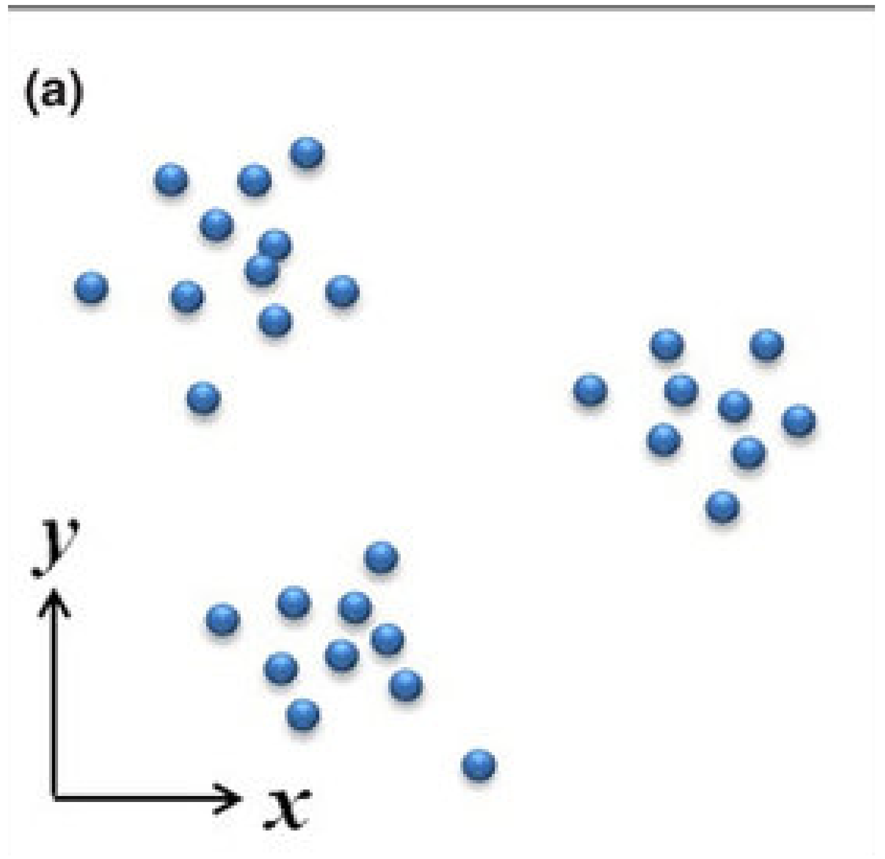
Non-parametric algorithm - algorithm doesn’t assume anything about training data

Fit-predict pipeline of kMeans:

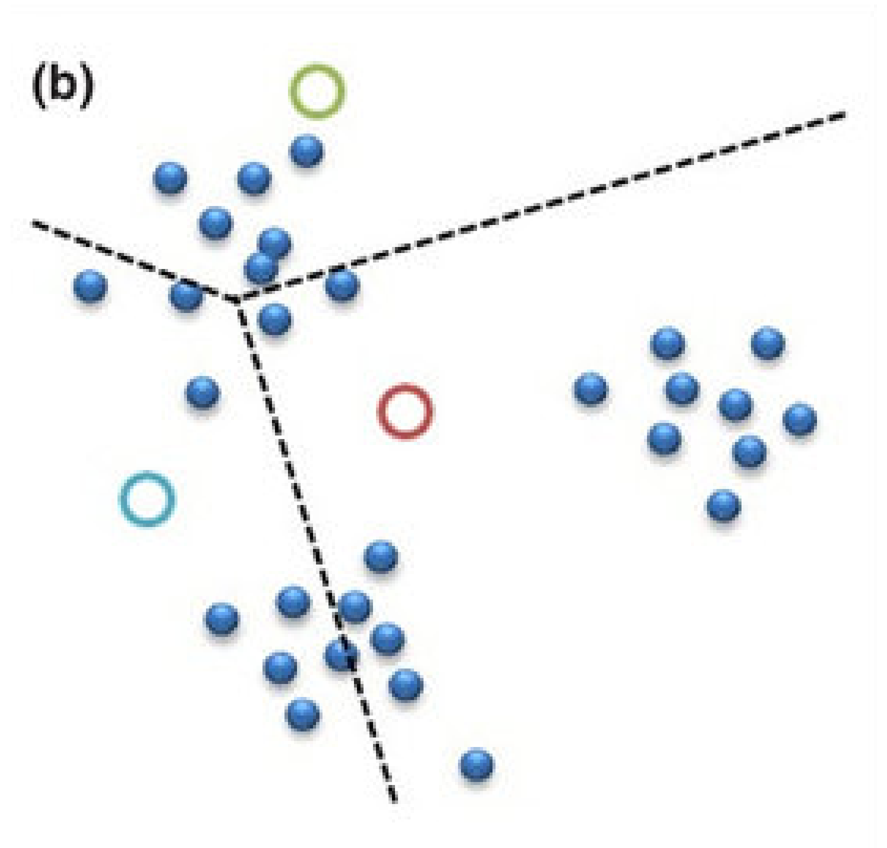
1.Set up hyper-parameters of kNN algorithm:

- k - number of clusters
(from 2 to Num. Observations)
- Distance metric - euclidean
- Max iterations
(from 1 to infinity)
- Initialization algorithm (optional)
(random or kmeans++)

2.Fit the data into kMeans algorithm:

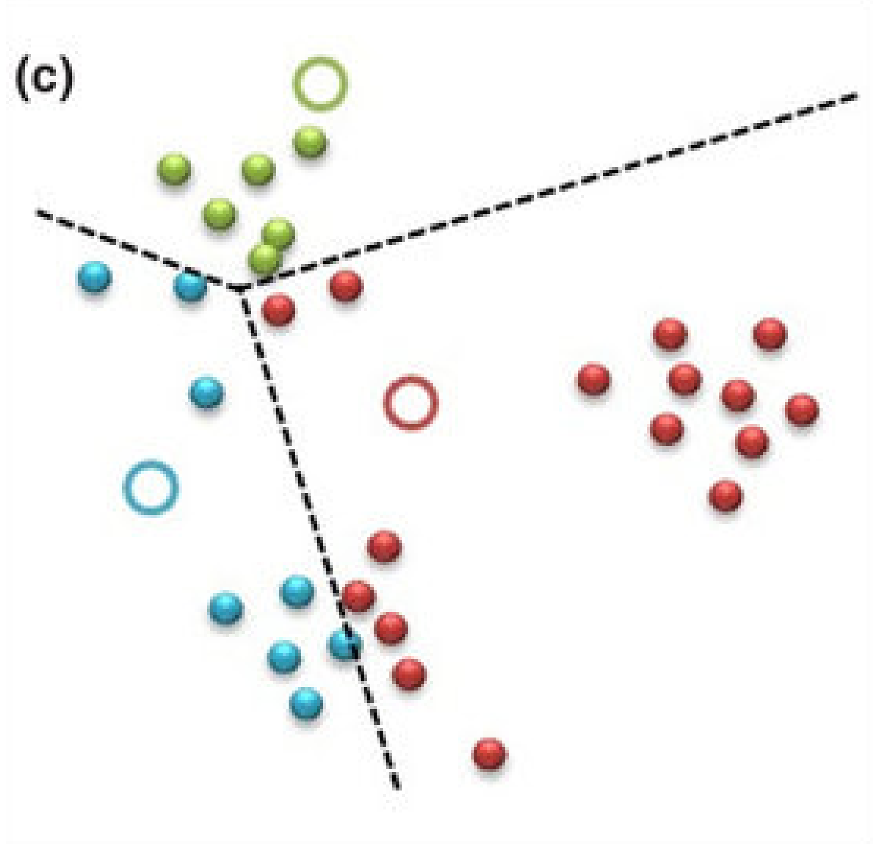


3.Initialize centroids (centers of clusters)

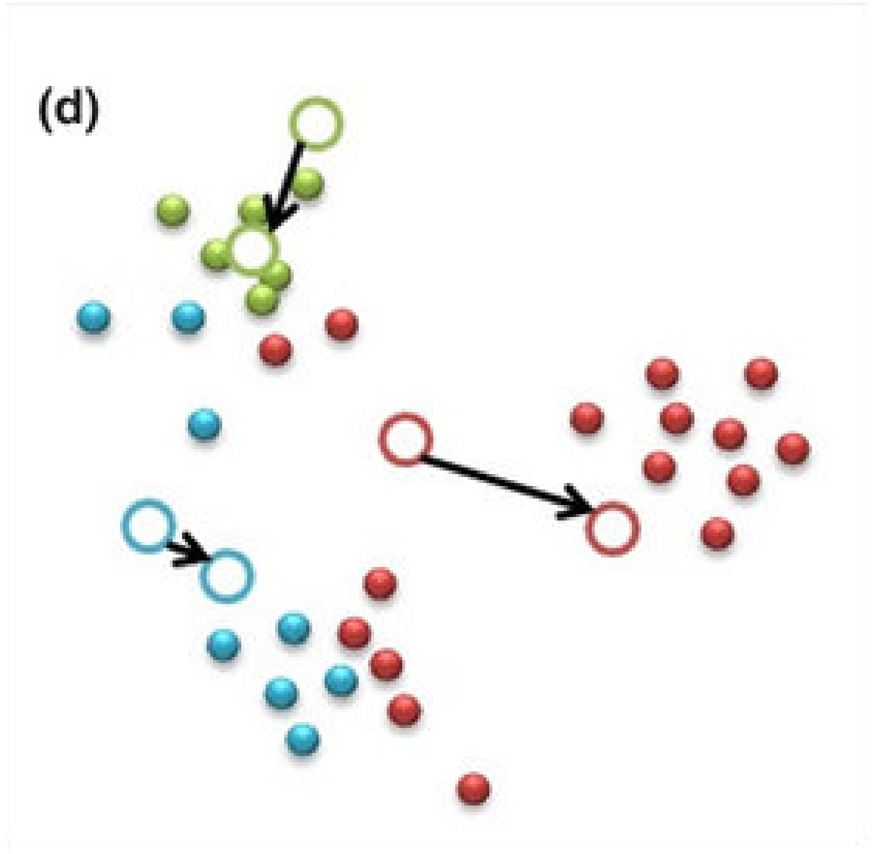


Fit-predict pipeline of kMeans:

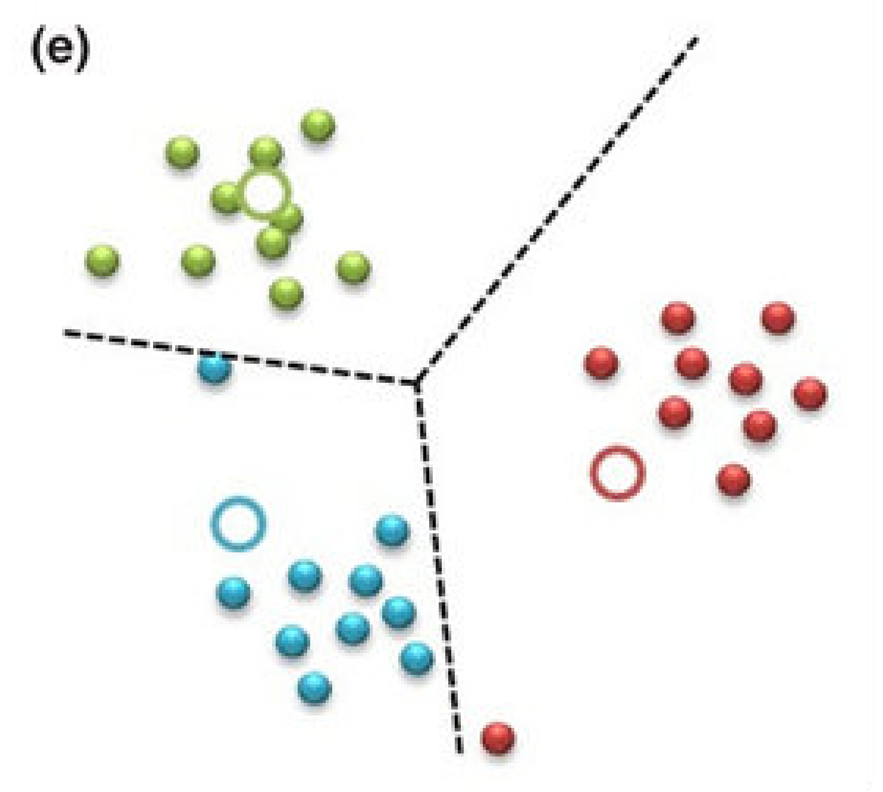
4.Assign each vector to nearest cluster



5.Replace centroids in the center of centroid's vectors



6.Repeat step 4 and step 5 ***i*** times to improve clustering results

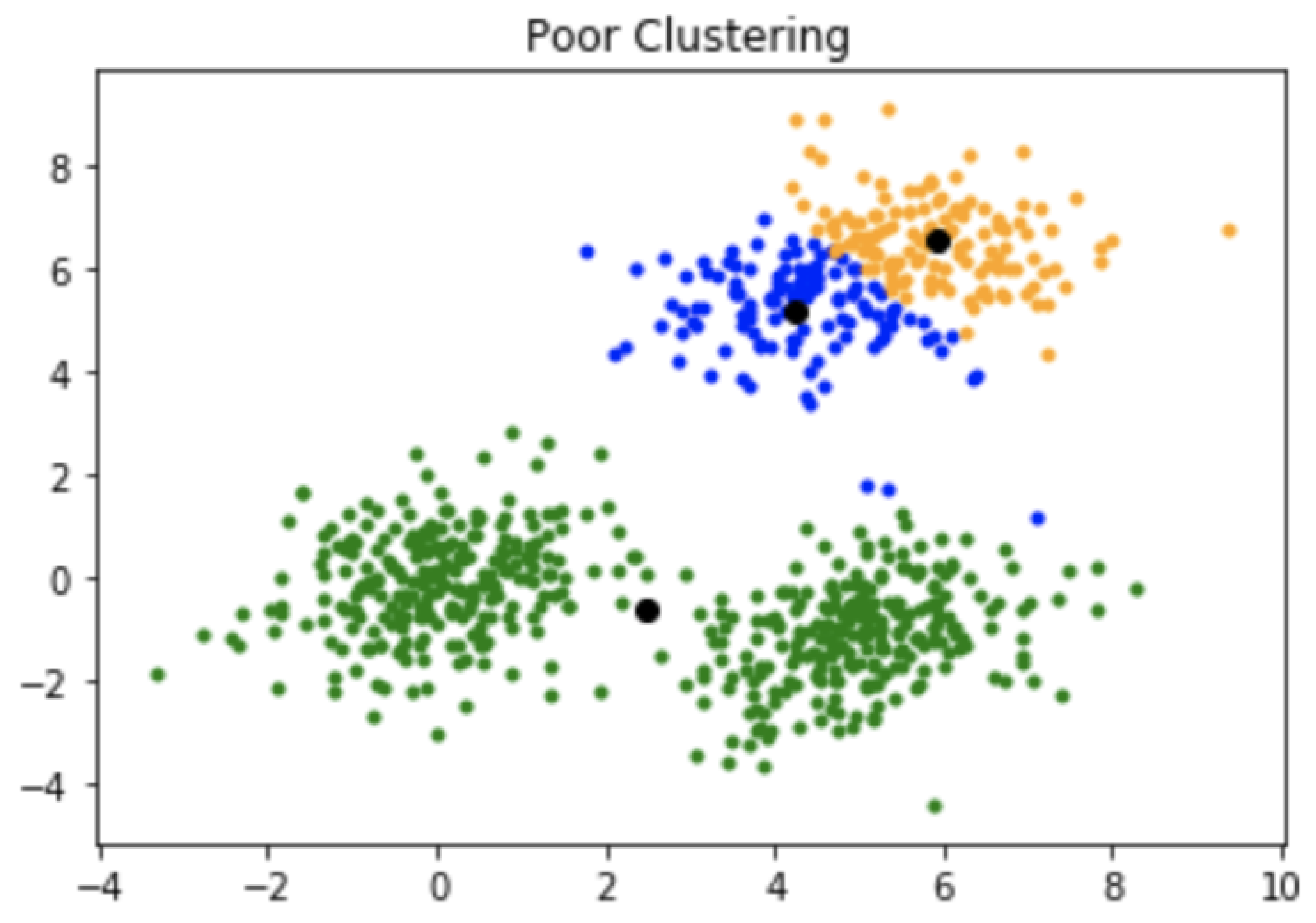


Stopping Criteria for K-Means Clustering:

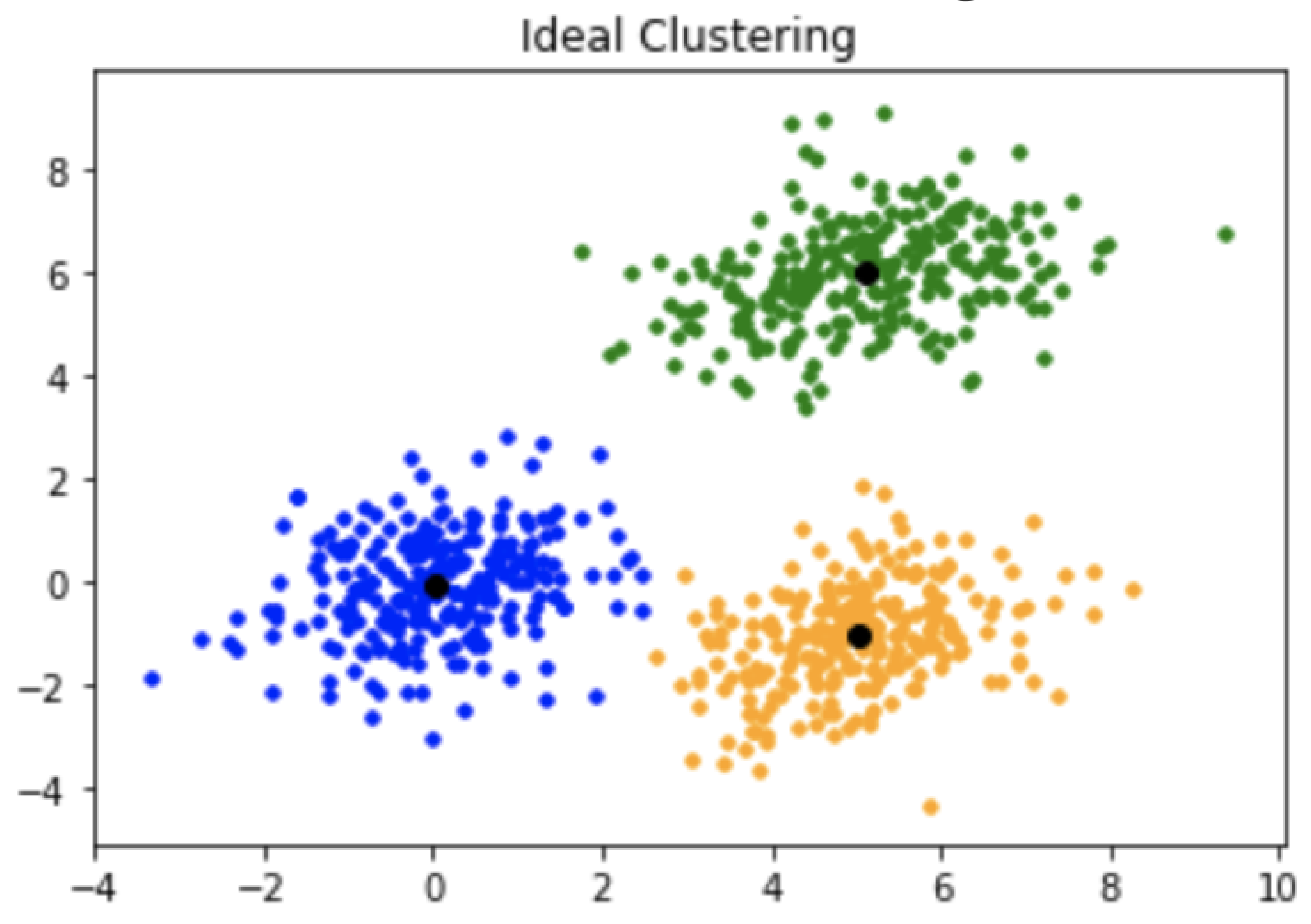
- Centroids of newly formed clusters do not change
- Points remain in the same cluster
- Maximum number of iterations is reached

kMeans++ algorithm of centroids initialization

Random initialization of centroids



Initialization of centroids using kmeans++



How kmeans++ works?

1. Randomly select the first centroid from the data points.
2. For each data point compute its distance from the nearest, previously chosen centroid.
3. Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid. (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)
4. Repeat steps 2 and 3 until k centroids have been sampled