**Introduction to Machine Learning**

My name: Pham Van Phuc

Student’s ID: 522H0068

**Homework: KNN Aglorithms**

1) Learn and present the theory

- KNN is a **supervised learning algorithm** used for both **classification** and **regression**. It is based on the principle that **similar data points tend to be close to each other**. When predicting a new data point, KNN finds the **K nearest points** in the training dataset and makes a decision based on them.

- Suppervised learning : is a machine learning method in which a model is trained on a dataset with labeled outputs. That is, each input data sample has a desired output, and the model learns to predict this output based on the input data.

- Steps to Find K Nearest Neighbors

1. Compute Distances: Calculate the distance between the query point and all points in the training dataset. Common distance metrics include Euclidean distance and Mahalanobis distance.
2. Sort by Distance: Order the training points based on their distance from the query point, starting with the nearest.
3. Select k Neighbors: Choose the k nearest points from the sorted list. These are the k nearest neighbors.
4. Determine Labels: For classification, assign the class label based on the majority vote among the k nearest neighbors. For regression, the output is the average of the values of the k nearest neighbors.

- KNN is a Method in Instance-Based Learning: Instance-based learning is often termed lazy learning, as there is typically no "transformation" of training instances into more general "statements”

- Distance Metrics:

The distance between two points  and  is often calculated using **Euclidean distance**:



2) Learn the model parameters in the libraries

Model Parameters in scikit-learn

* KNeighborsClassifier (For Classification)
* This is used for **classification tasks**, where the predicted label is determined by the majority class among the nearest neighbors.

class sklearn.neighbors.KNeighborsClassifier(n\_neighbors=5, \*, weights='uniform', algorithm='auto', leaf\_size=30, p=2, metric='minkowski', metric\_params=None, n\_jobs=None)

Parameters:

* n\_neighborsint: int, default=5

Number of neighbors to use by default for kneighbors queries.

* weights{‘uniform’, ‘distance’}, callable or None, default=’uniform’

Weight function used in prediction. Possible values:

* + ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
  + ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  + [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
* algorithm{‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}, default=’auto’

Algorithm used to compute the nearest neighbors:

* + ‘ball\_tree’ will use BallTree
  + ‘kd\_tree’ will use KDTree
  + ‘brute’ will use a brute-force search.
  + ‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method.
* leaf\_sizeint, default=30

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

* pfloat, default=2

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan\_distance (l1), and euclidean\_distance (l2) for p = 2. For arbitrary p, minkowski\_distance (l\_p) is used. This parameter is expected to be positive.

* metricstr or callable, default=’minkowski’

Metric to use for distance computation

* + 'minkowski' (default) → Use p to define Euclidean or Manhattan.
  + o 'euclidean' → Only use Euclidean distance (equivalent to minkowski with p=2).
  + o 'manhattan' → Only use Manhattan distance (equivalent to minkowski with p=1).
  + o 'chebyshev' → Chebyshev distance
  + o 'hamming' → Used for categorical data, measuring the difference between two vectors.
  + o 'precomputed' → Assume the input data is a distance matrix.
* metric\_paramsdict, default=None

Additional keyword arguments for the metric function.

* n\_jobsint, default=None

he number of parallel jobs to run for neighbors search. None means 1 unless in a joblib.parallel\_backend context. -1 means using all processors

* KNeighborsRegressor (For Regression):

- This is used for **regression tasks**, where the predicted value is computed as the average (or weighted average) of the nearest neighbors' values.

class sklearn.neighbors.KNeighborsRegressor(n\_neighbors=5, \*, weights='uniform', algorithm='auto', leaf\_size=30, p=2, metric='minkowski', metric\_params=None, n\_jobs=None)

Parameters:

* n\_neighborsint: int, default=5

Number of neighbors to use by default for kneighbors queries.

* weights{‘uniform’, ‘distance’}, callable or None, default=’uniform’

Weight function used in prediction. Possible values:

* + ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
  + ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  + [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
* algorithm{‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}, default=’auto’

Algorithm used to compute the nearest neighbors:

* + ‘ball\_tree’ will use BallTree
  + ‘kd\_tree’ will use KDTree
  + ‘brute’ will use a brute-force search.
  + ‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method.
* leaf\_sizeint, default=30

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

* pfloat, default=2

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan\_distance (l1), and euclidean\_distance (l2) for p = 2. For arbitrary p, minkowski\_distance (l\_p) is used. This parameter is expected to be positive.

* metricstr or callable, default=’minkowski’

Metric to use for distance computation

Metric to use for distance computation

* + 'minkowski' (default) → Use p to define Euclidean or Manhattan.
  + 'euclidean' → Only use Euclidean distance (equivalent to minkowski with p=2).
  + 'manhattan' → Only use Manhattan distance (equivalent to minkowski with p=1).
  + 'chebyshev' → Chebyshev distance
  + 'hamming' → Used for categorical data, measuring the difference between two vectors.
  + 'precomputed' → Assume the input data is a distance matrix.
* metric\_paramsdict, default=None

Additional keyword arguments for the metric function.

* n\_jobsint, default=None

he number of parallel jobs to run for neighbors search. None means 1 unless in a joblib.parallel\_backend context. -1 means using all processors

3) Apply to datasets in UCI

1. Iris
   1. Description

The Iris dataset is a classification dataset containing 150 samples of 3 species of Iris flowers:

* + Setosa
  + Versicolor
  + Virginica

Each sample has 4 features:

* + Sepal Length
  + Sepal Width
  + Petal Length
  + Petal Width
  1. Input

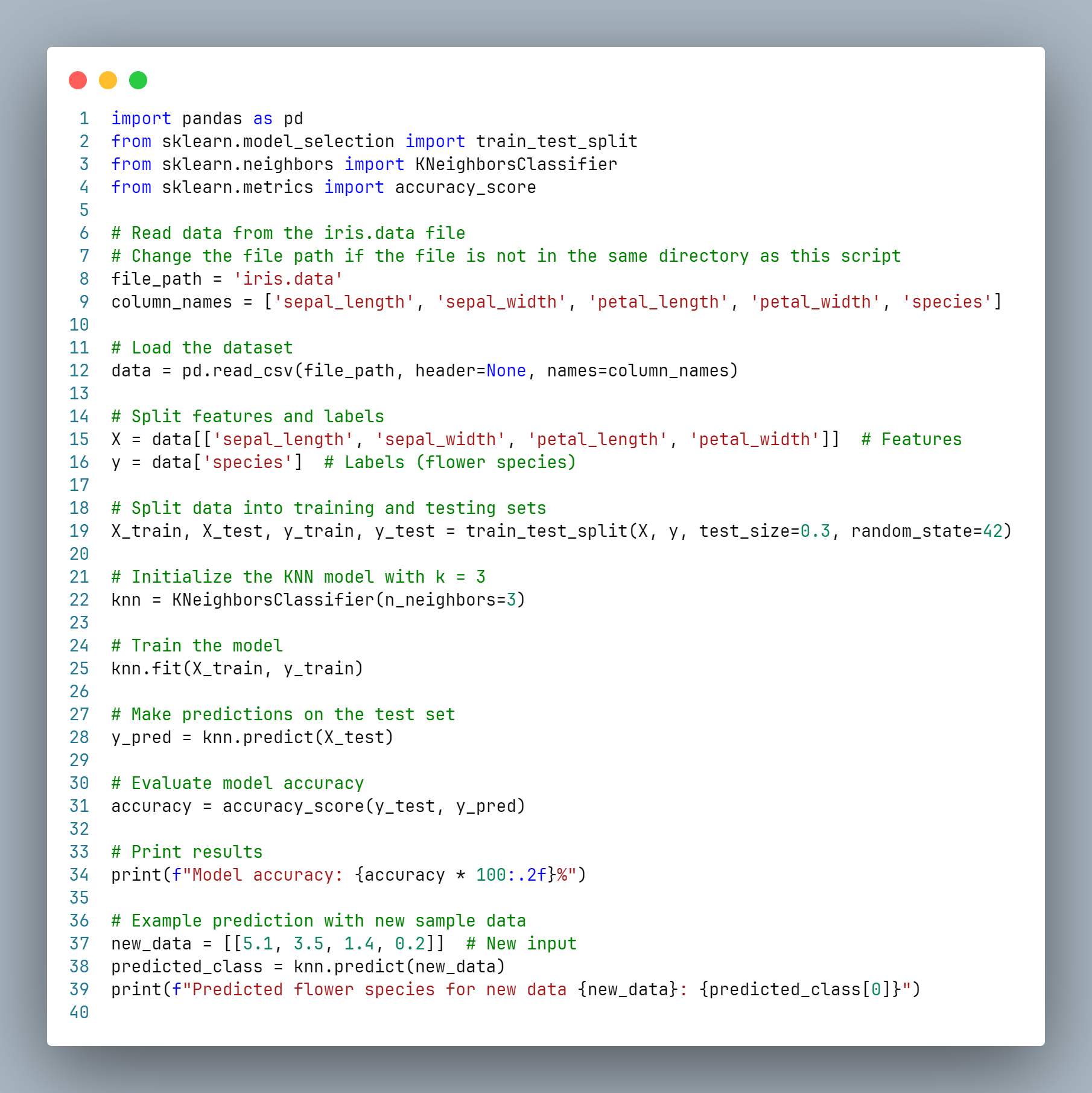
Data: 150 rows, 4 feature columns

Output label: 3 species of flowers (0: Setosa, 1: Versicolor, 2: Virginica)

Example of a row of data: 5.8,2.7,5.1,1.9,Iris-virginica

New sample is created to predict: 5.1, 3.5, 1.4, 0.2

* 1. Implement



* 1. Output

Predicted flower species for new data [[5.1, 3.5, 1.4, 0.2]]: Iris-setosa



1. Abalone
   1. Description

The Abalone dataset is used to predict the age of abalone based on its size and weight.

The dataset contains 4,177 samples with 8 features:

* + Sex (Male, Female, Infant)
  + Length
  + Diameter
  + Height
  + Whole weight
  + Shucked weight
  + Viscera weight
  + Shell weight

Target: Predict Rings (the number of rings, which determines the abalone's age). This is a regression problem

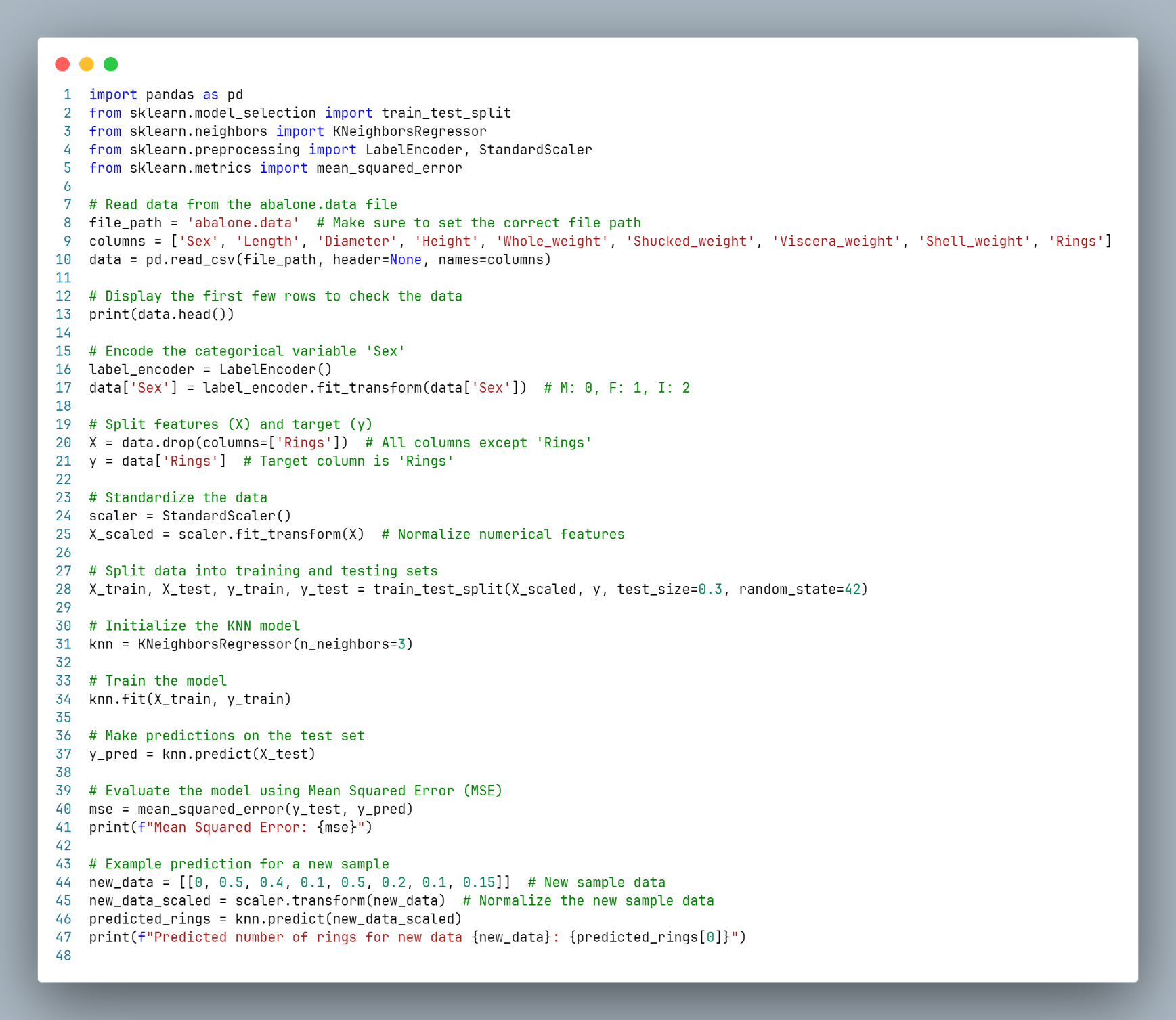
* 1. Input

Data: 4,177 rows, 8 feature columns

Target label: Rings (integer value representing age)

New sample is created to predict: 0, 0.5, 0.4, 0.1, 0.5, 0.2, 0.1, 0.15

* 1. Implement



* 1. Output

Predicted number of rings for new data [[0, 0.5, 0.4, 0.1, 0.5, 0.2, 0.1, 0.15]]: 11.0

