MET CS 688 Statistics & Probability II

Leila Ghaedi – Fall 2024

Creator: cemagraphics | Credit: Getty Images/iStockphoto

ANOVA
Analysis Of Variance

Suppose that, instead of an A/B test, we had a comparison of multiple groups, say A/B/C/D, each with numeric data. The statistical procedure that tests for a statistically significant difference among the groups is called analysis of variance, or ANOVA.

ANOVA

The H0 in ANOVA assumes that all groups' mean are equal.

H1 assumes at least two of group means are different.

$$H0: \mu 1 = \mu 2 = \mu 3$$

H1: Means are not all equal.

F-statistic: A standardized statistic that measures the extent to which differences among group means exceed what might be expected in a chance model.

F-statistic

$$F = MS_B / MS_W$$

Where:

MS_B = Sum of squares between samples (SS_B) / (k-1)

MS_W= Sum of squares within samples (SS_w) / (n-k)

k is the number of groups
n is the total number of observations

"Sum of squares," referring to deviations from some average value.

ANOVA

One-Way ANOVA: is a hypothesis test, which tests the equality of three of more population means simultaneously using variance.

- Number of observations could be different in each group.
- Number of independent variables is one.

Two-Way ANOVA: is a statistical technique which studies the interaction between factors, influencing variable.

- Number of observations need to be equal in each group.
- Number of independent variables is two.

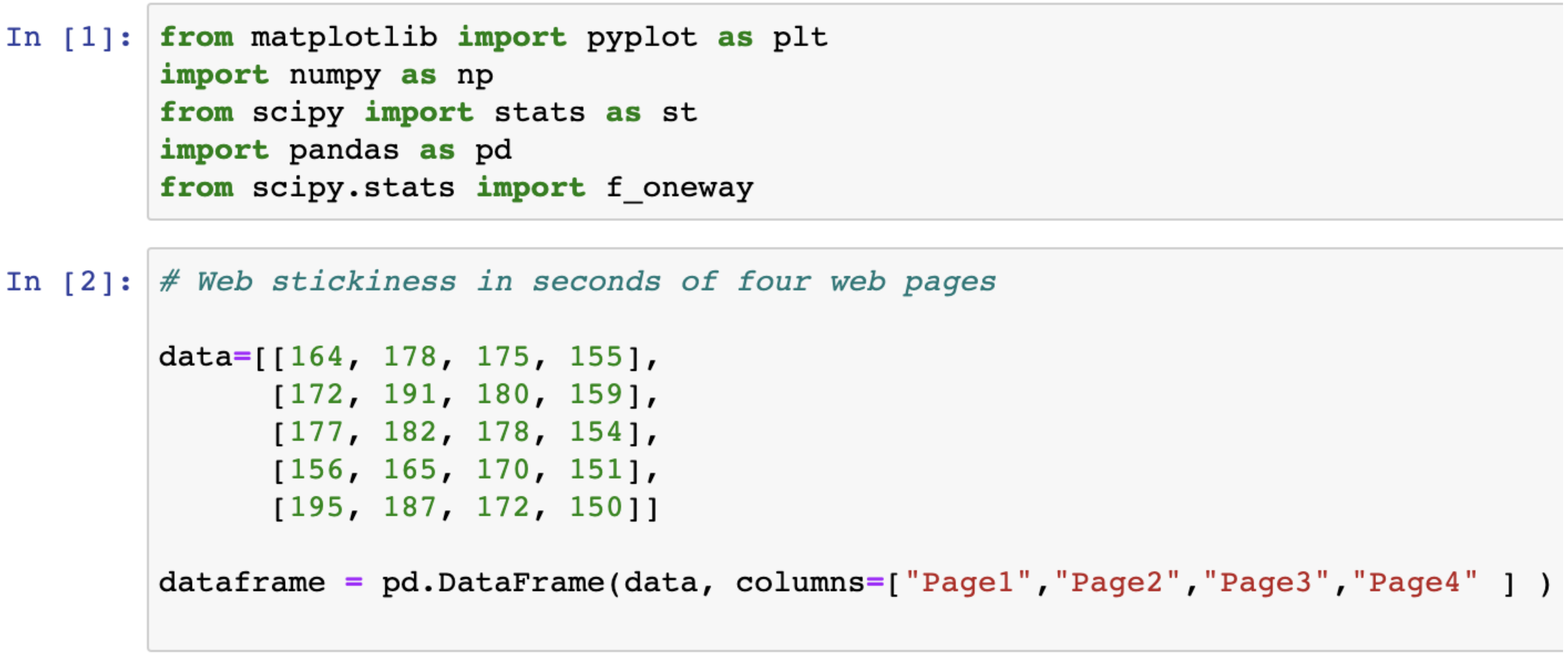
One Way ANOVA Assumptions

- The responses for each factor level have a normal population distribution.
- These distributions have the same variance.
- The data-points are independent.

ANOVA Example
Web Stickiness Data for 4 web pages (in seconds)

Page 1	Page 2	Page 3	Page 4
164	178	175	155
172	191	180	159
177	182	178	154
156	165	170	151
195	187	172	150

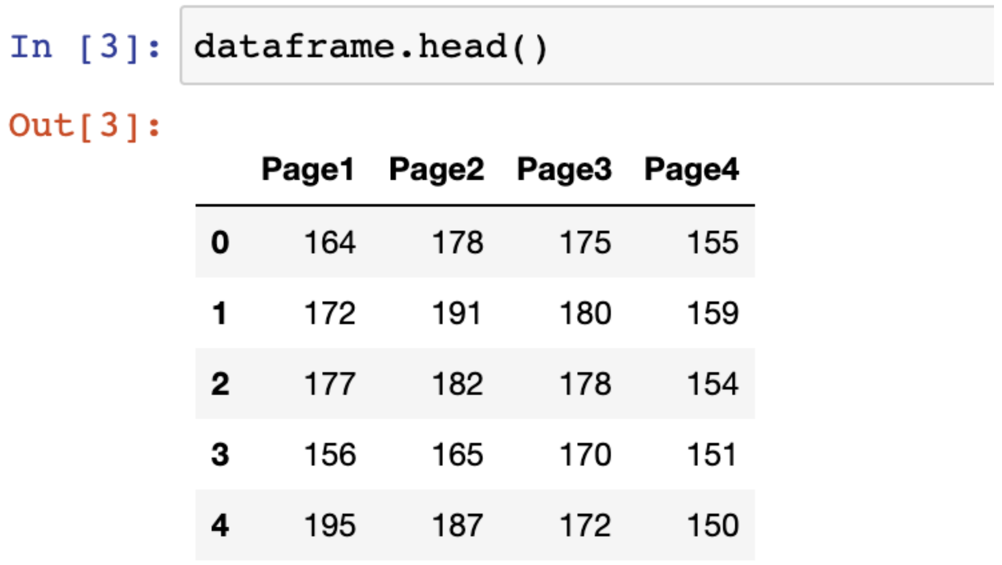
ANOVA Example
Web Stickiness Data for 4 web pages (in seconds)



```
In [1]: from matplotlib import pyplot as plt
        import numpy as np
        from scipy import stats as st
        import pandas as pd
        from scipy.stats import f_oneway
```

```
In [2]: # Web stickiness in seconds of four web pages

        data=[[164, 178, 175, 155],
              [172, 191, 180, 159],
              [177, 182, 178, 154],
              [156, 165, 170, 151],
              [195, 187, 172, 150]]

        dataframe = pd.DataFrame(data, columns=["Page1","Page2","Page3","Page4" ] )
```

ANOVA Example
Web Stickiness Data for 4 web pages (in seconds)

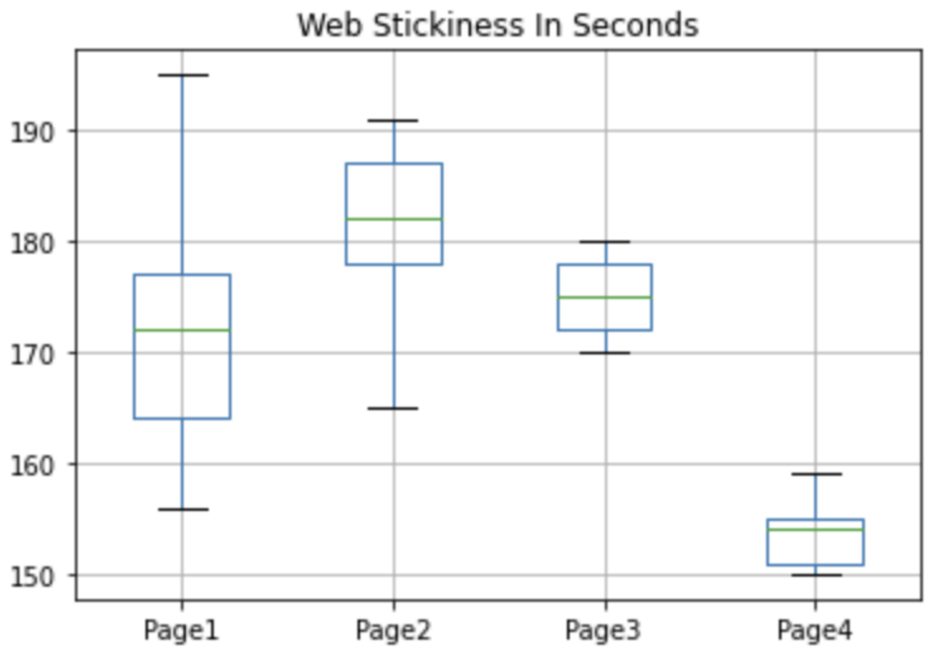


```
In [3]: dataframe.head()
Out[3]:
```

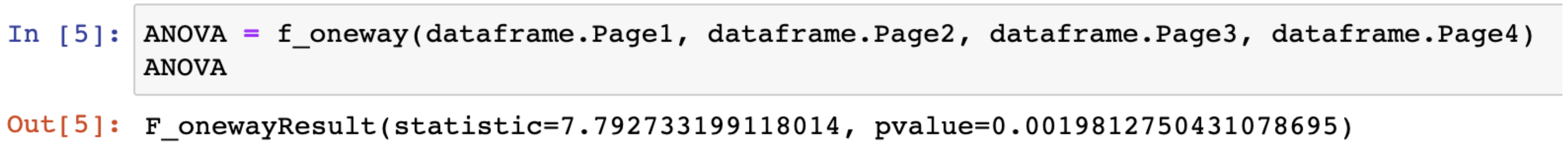
	Page1	Page2	Page3	Page4
0	164	178	175	155
1	172	191	180	159
2	177	182	178	154
3	156	165	170	151
4	195	187	172	150

ANOVA Example
Web Stickiness Data for 4 web pages (in seconds)

```
In [4]: dataframe.boxplot()
        plt.title('Web Stickiness In Seconds')

Out[4]: Text(0.5, 1.0, 'Web Stickiness In Seconds')
```



ANOVA Example
Web Stickiness Data for 4 web pages (in seconds)



```
In [5]: ANOVA = f_oneway(dataframe.Page1, dataframe.Page2, dataframe.Page3, dataframe.Page4)
        ANOVA

Out[5]: F_onewayResult(statistic=7.792733199118014, pvalue=0.0019812750431078695)
```

Post Hoc Test

Post Hoc means after this.

Post hoc tests are used after a statistically significant result has been found. They are used to determine where the differences came from.

- Bonferroni Procedure (Bonferroni Correction)
- Dunn's Multiple Comparison Test

χ^2 Test

1. Test of Independence between two categorical variable.

   H_0: The two categorical variables have **no relationship**

   H_1: There is **a relationship** between two categorical variables

2. Test for the Goodness-of-fit

This test operates based on a contingency table. **Contingency**, **Crosstab** or **RxC table** (read as R by C table, R stayed for row and C for column) is a table that presents frequency of different variables in a dataset and their relations together.
It calculates the *p*-value based on the differences between observed and expected values. Here is the calculation of expected value:

$$E = \frac{total\ row \times total\ column}{sample\ size}$$

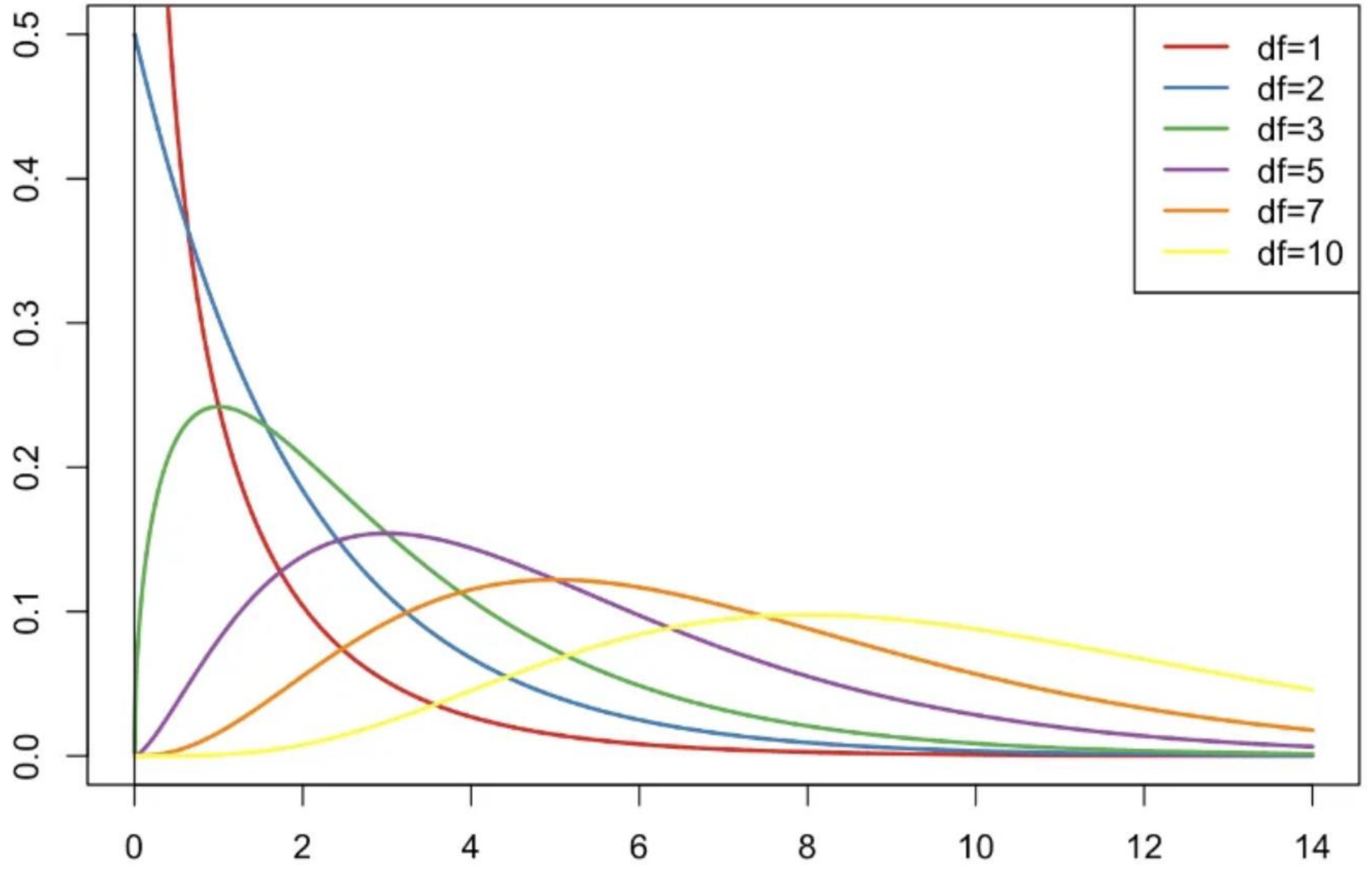
Chi-Square:Test of independence

After we have calculated the expected values, we can use the following formula to calculate the chi-square score:

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Chi-Square distribution with different degree of freedom

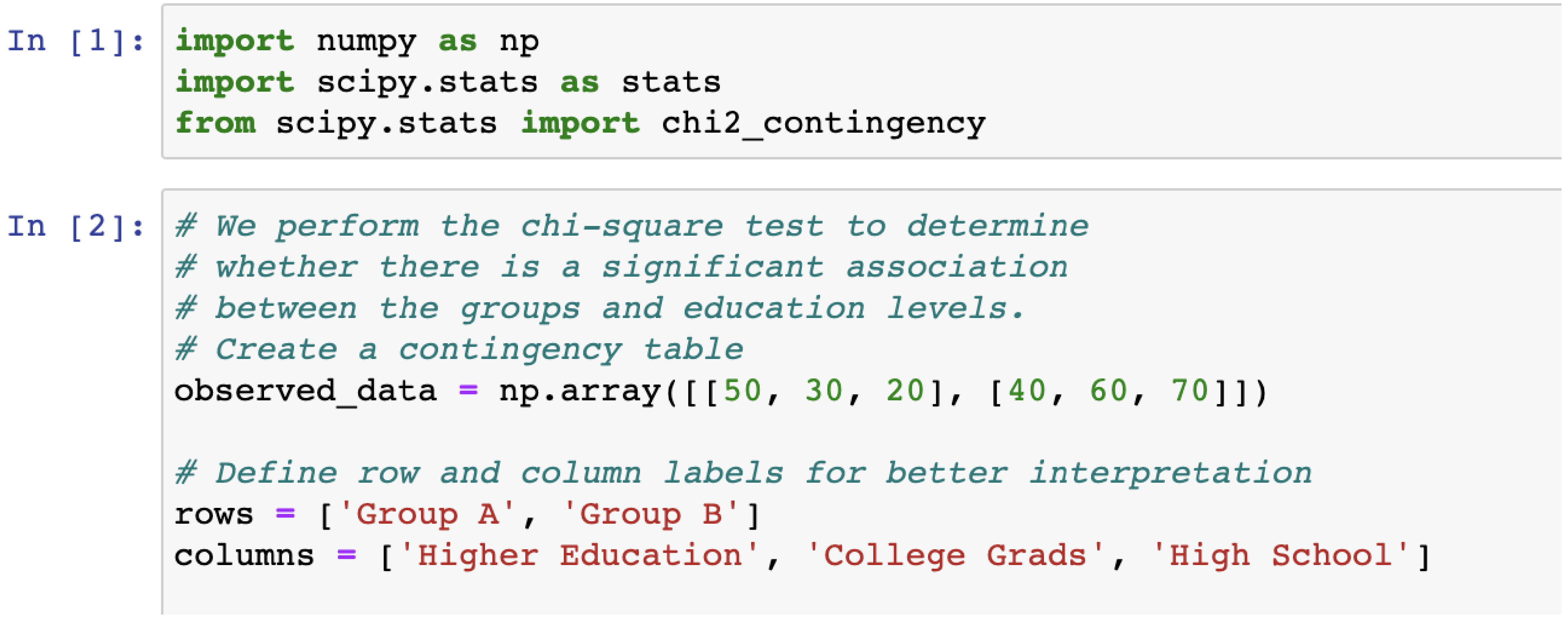
The χ2 distribution displays a right-skew, and as the degrees of freedom increase, the χ2 curve progressively resembles the normal distribution more closely.]
df = (r-1)(c-1) where r is the number of rows and c is the number of columns



Chi-Square Example

- We have two groups of people (A and B)
- We have the frequency of education levels for them ('Higher Education', 'College Grads', 'High School')
- H_0: There is no difference between A and B in terms of education.
- We are going to use Chi Square test to answer this question.

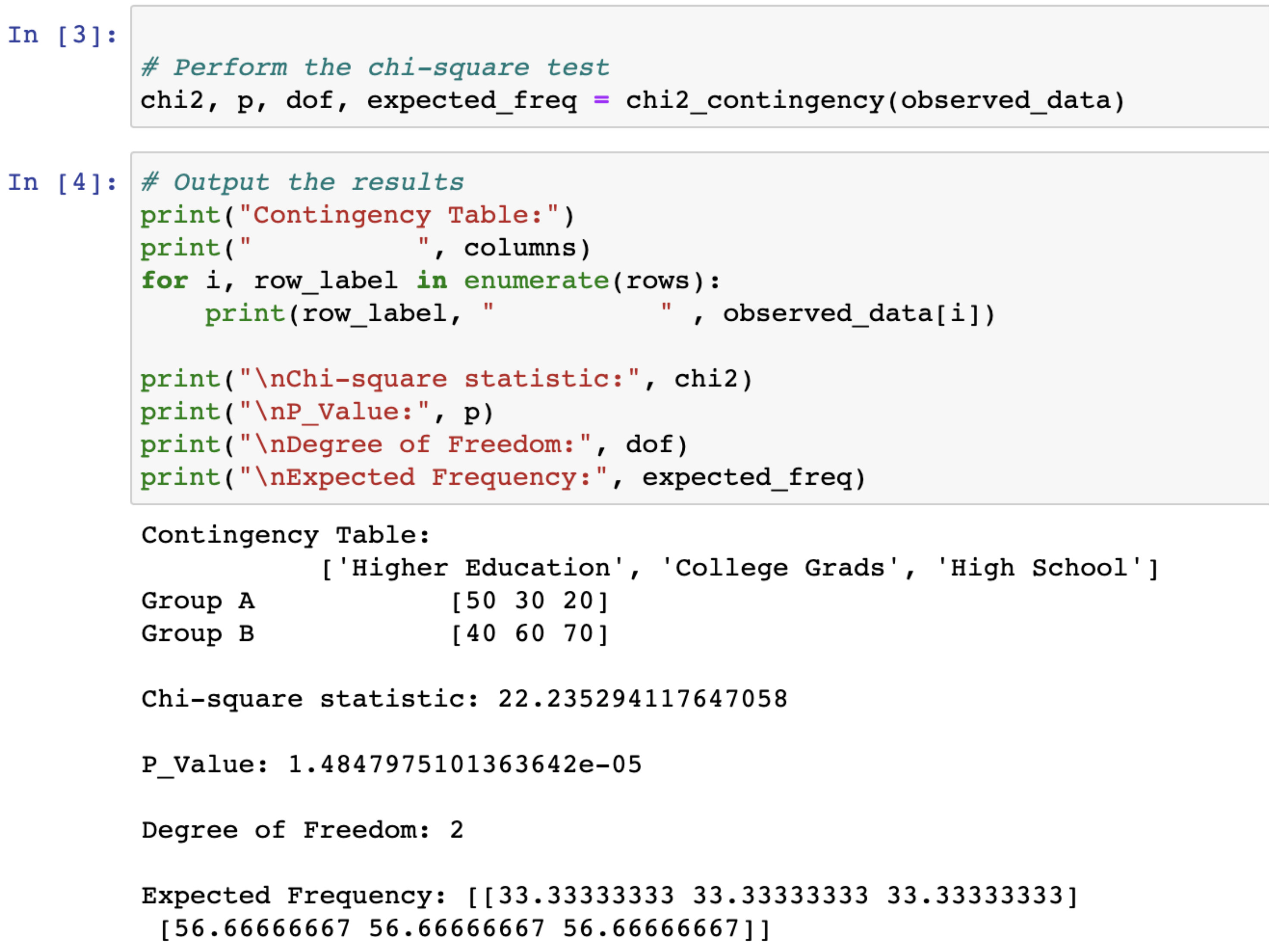
Chi-Square Example



```
In [1]: import numpy as np
        import scipy.stats as stats
        from scipy.stats import chi2_contingency
```

```
In [2]: # We perform the chi-square test to determine
        # whether there is a significant association
        # between the groups and education levels.
        # Create a contingency table
        observed_data = np.array([[50, 30, 20], [40, 60, 70]])

        # Define row and column labels for better interpretation
        rows = ['Group A', 'Group B']
        columns = ['Higher Education', 'College Grads', 'High School']
```

Chi-Square Example



```
In [3]:
# Perform the chi-square test
chi2, p, dof, expected_freq = chi2_contingency(observed_data)
```

```
In [4]:
# Output the results
print("Contingency Table:")
print("          ", columns)
for i, row_label in enumerate(rows):
    print(row_label, "          " , observed_data[i])

print("\nChi-square statistic:", chi2)
print("\nP_Value:", p)
print("\nDegree of Freedom:", dof)
print("\nExpected Frequency:", expected_freq)
```

```
Contingency Table:
           ['Higher Education', 'College Grads', 'High School']
Group A            [50 30 20]
Group B            [40 60 70]

Chi-square statistic: 22.235294117647058

P_Value: 1.4847975101363642e-05

Degree of Freedom: 2

Expected Frequency: [[33.33333333 33.33333333 33.33333333]
 [56.66666667 56.66666667 56.66666667]]
```

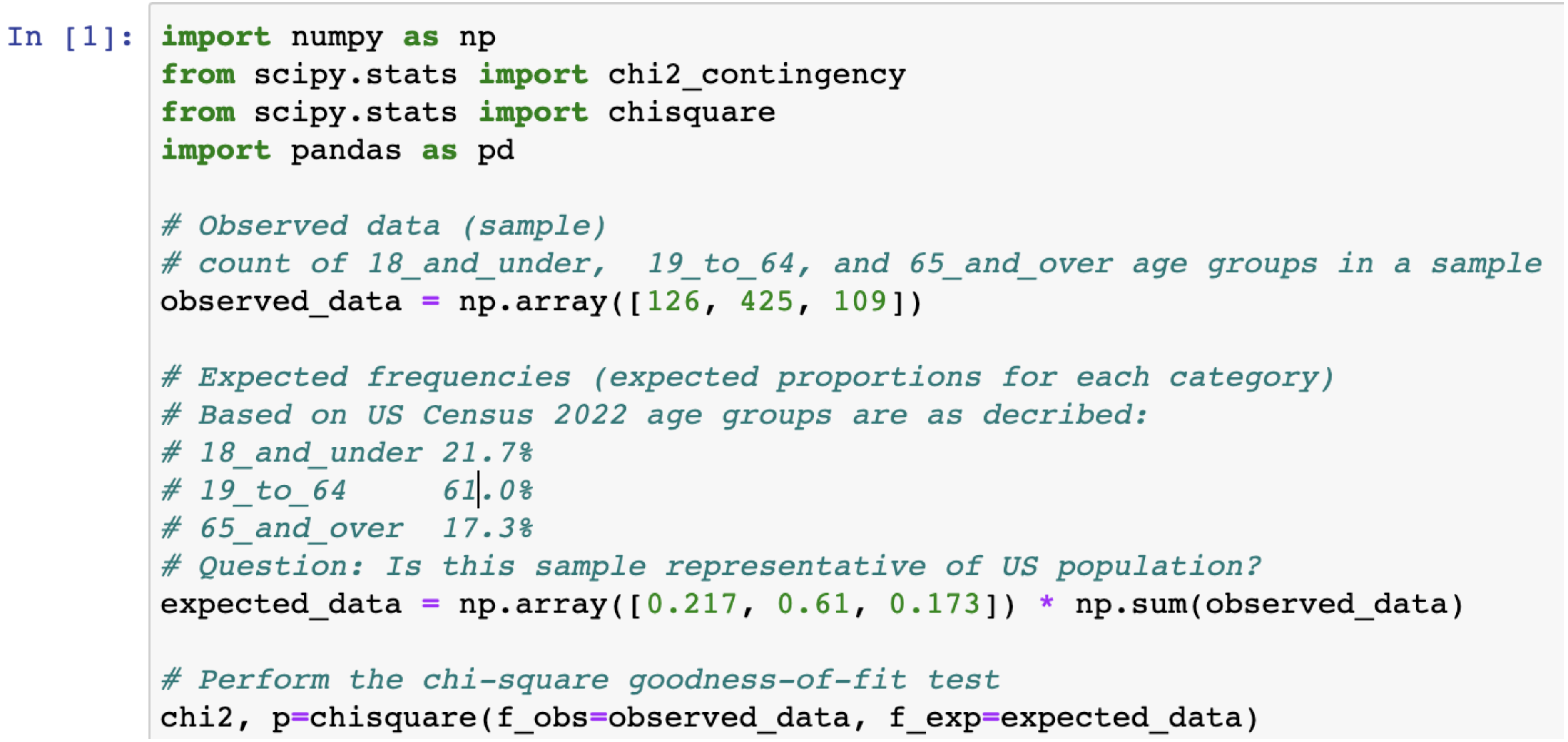
Chi-Square Example
Examine Goodness of Fit

- We know the breakout of age group for US population based on Census 2022.

Age Group	Percentage
18 and under	21.7%
19 to 64	61%
65 and over	17.3%

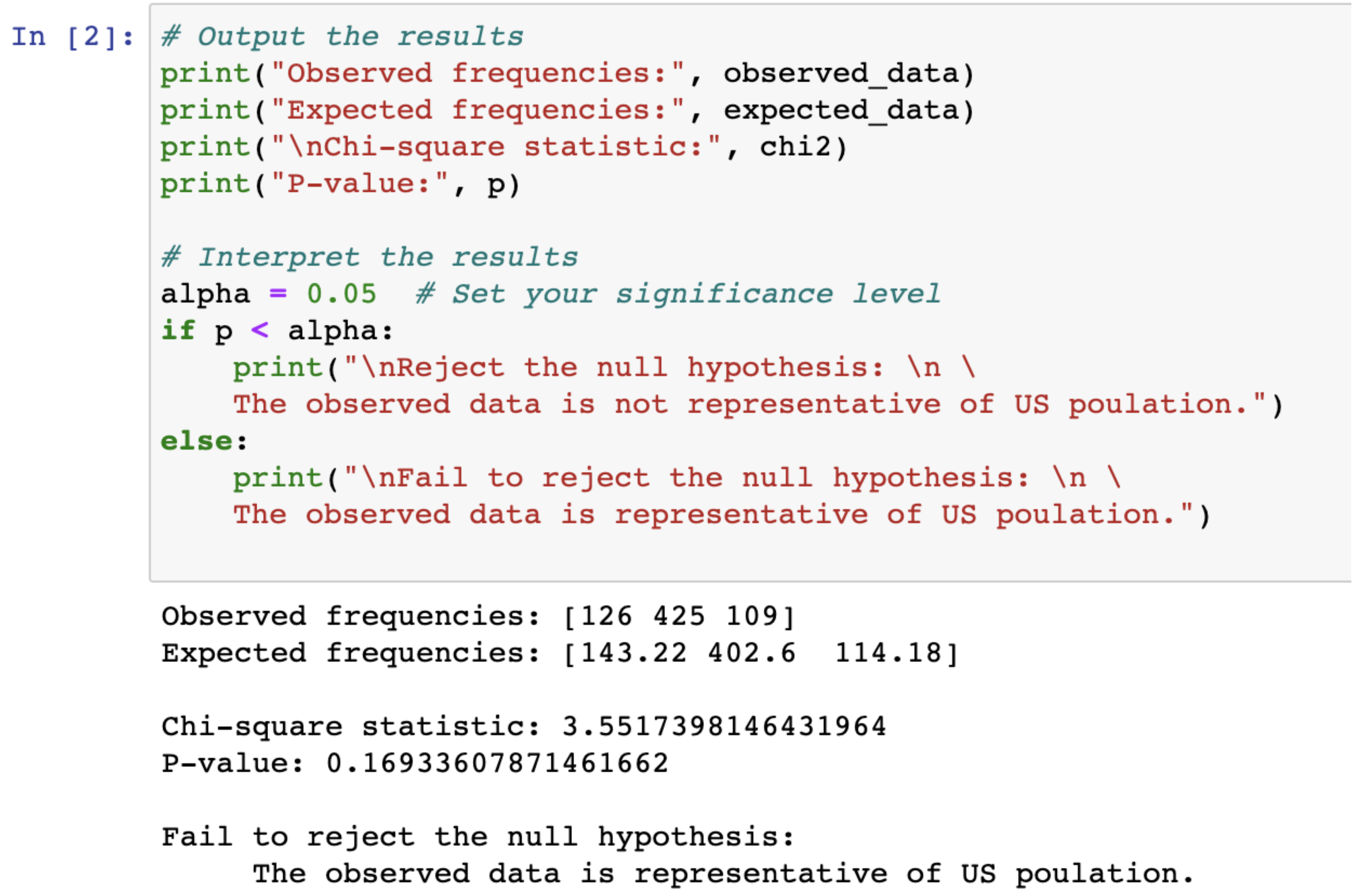
- We have a sample of people in the above age group [126, 425, 109].
- Is this sample representative of US population?

Chi-Square Example
Examine Goodness of Fit



```
In [1]: import numpy as np
        from scipy.stats import chi2_contingency
        from scipy.stats import chisquare
        import pandas as pd

        # Observed data (sample)
        # count of 18_and_under,  19_to_64, and 65_and_over age groups in a sample
        observed_data = np.array([126, 425, 109])

        # Expected frequencies (expected proportions for each category)
        # Based on US Census 2022 age groups are as decribed:
        # 18_and_under 21.7%
        # 19_to_64     61.0%
        # 65_and_over  17.3%
        # Question: Is this sample representative of US population?
        expected_data = np.array([0.217, 0.61, 0.173]) * np.sum(observed_data)

        # Perform the chi-square goodness-of-fit test
        chi2, p=chisquare(f_obs=observed_data, f_exp=expected_data)
```

Chi-Square Example
Examine Goodness of Fit



In [2]:

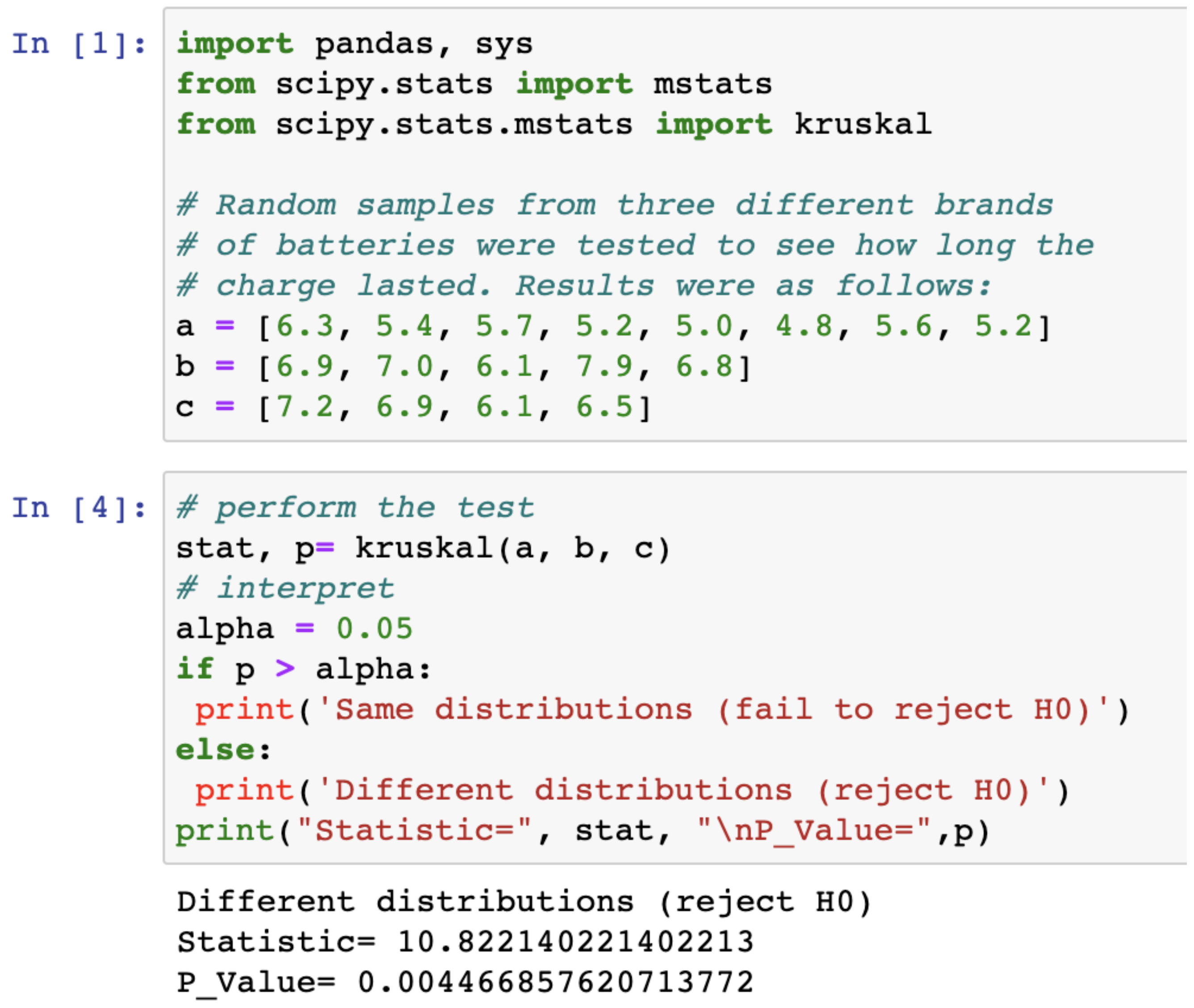
```python
# Output the results
print("Observed frequencies:", observed_data)
print("Expected frequencies:", expected_data)
print("\nChi-square statistic:", chi2)
print("P-value:", p)

# Interpret the results
alpha = 0.05  # Set your significance level
if p < alpha:
    print("\nReject the null hypothesis: \n \
    The observed data is not representative of US poulation.")
else:
    print("\nFail to reject the null hypothesis: \n \
    The observed data is representative of US poulation.")
```

```
Observed frequencies: [126 425 109]
Expected frequencies: [143.22 402.6  114.18]

Chi-square statistic: 3.5517398146431964
P-value: 0.16933607871461662

Fail to reject the null hypothesis: 
     The observed data is representative of US poulation.
```

Kruskal-Wallis Test (KW-Test)

- The Kruskal-Wallis Test is a non-parametric statistical method, like the one-way ANOVA. KW test assess and compare multiple samples.
  1. This test combines data from all samples.
  2. Ranks all samples in ascending order.
  3. Calculates the sum of ranks for each group.
  4. If two samples have equal mix of rank, they are assumed to be similar. Otherwise, if two samples are not having similar rank, they assumed to accept the null hypothesis.
- Test Statistic (H statistic): The Kruskal-Wallis test calculates an H statistic, which is a measure of the degree of separation or difference between the group distributions. A larger H value indicates a larger difference.

KW Test Example



```
In [1]: import pandas, sys
        from scipy.stats import mstats
        from scipy.stats.mstats import kruskal

        # Random samples from three different brands
        # of batteries were tested to see how long the
        # charge lasted. Results were as follows:
        a = [6.3, 5.4, 5.7, 5.2, 5.0, 4.8, 5.6, 5.2]
        b = [6.9, 7.0, 6.1, 7.9, 6.8]
        c = [7.2, 6.9, 6.1, 6.5]
```

```
In [4]: # perform the test
        stat, p= kruskal(a, b, c)
        # interpret
        alpha = 0.05
        if p > alpha:
         print('Same distributions (fail to reject H0)')
        else:
         print('Different distributions (reject H0)')
        print("Statistic=", stat, "\nP_Value=",p)
```

```
Different distributions (reject H0)
Statistic= 10.822140221402213
P_Value= 0.004466857620713772
```

Mann-Whitney-U Test

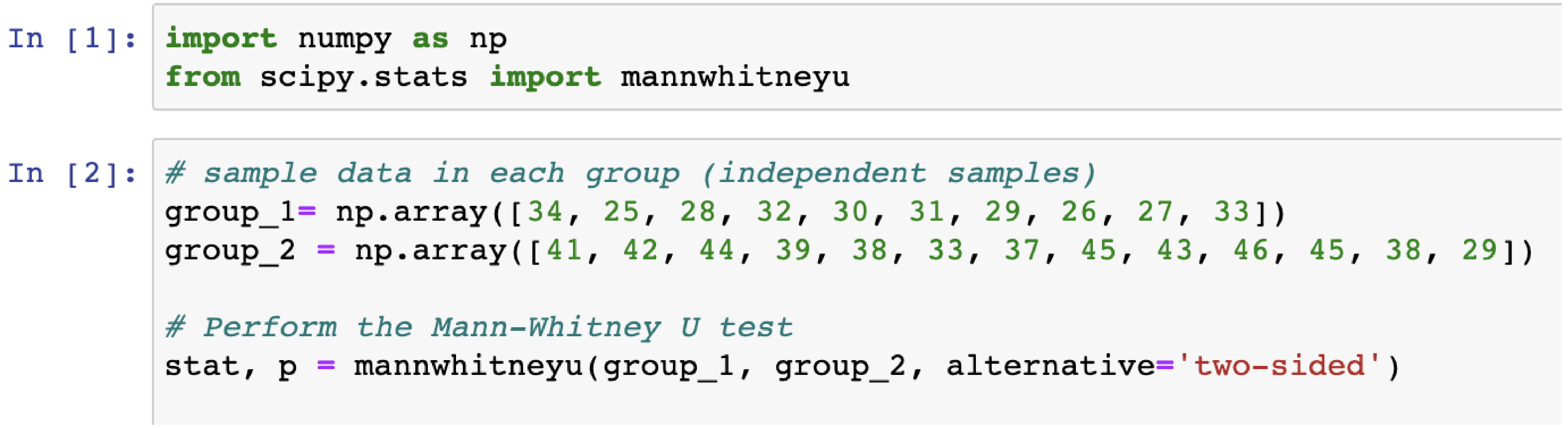
- Mann-Whitney (Mann-Whitney-Wilcoxon, Wilcoxon rank-sum test) is another non-parametric used for hypothesis test and identify exactly which sample is different from other samples. It is used to test two related samples, matches samples or samples from repeated measurement. **(Two-Sample T-Test)**

- KW-Test only identifies if they are similar or different, but it can not identify, exactly which sample dataset is different than others. Therefore, after the KW-Test rejects the H_0, we can use **Mann-Whitney-U Test (U-Test).** This test is also known as **Mann-Whitney- Wilcoxon** (**MWM-Test**) to find out which sample dataset is different than others. This method relies on ranks and their scores.

- To do this comparison we should run a test on each pair of the samples until we find which one is different than the other ones. This process is called **pairwise comparison** or **multiple comparison**. In particular, for U-Test we have the following assumptions.

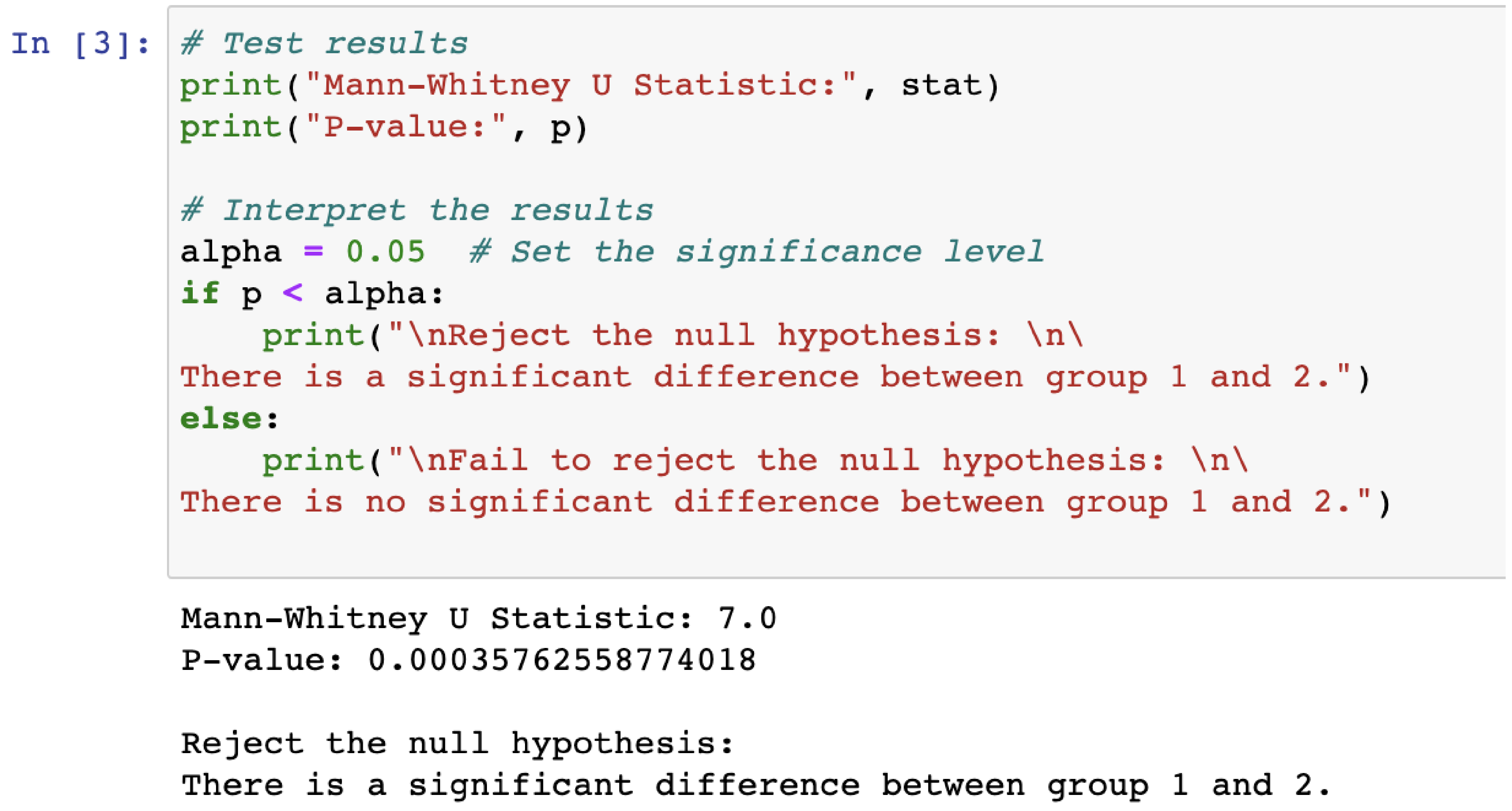
H_0: The distributions of both samples are equal.
H_1: The distributions of both samples are not equal.

Mann-Whitney-U Test Example



```
In [1]: import numpy as np
        from scipy.stats import mannwhitneyu

In [2]: # sample data in each group (independent samples)
        group_1= np.array([34, 25, 28, 32, 30, 31, 29, 26, 27, 33])
        group_2 = np.array([41, 42, 44, 39, 38, 33, 37, 45, 43, 46, 45, 38, 29])

        # Perform the Mann-Whitney U test
        stat, p = mannwhitneyu(group_1, group_2, alternative='two-sided')
```

Mann-Whitney-U Test Example



```
In [3]: # Test results
        print("Mann-Whitney U Statistic:", stat)
        print("P-value:", p)

        # Interpret the results
        alpha = 0.05  # Set the significance level
        if p < alpha:
            print("\nReject the null hypothesis: \n\
        There is a significant difference between group 1 and 2.")
        else:
            print("\nFail to reject the null hypothesis: \n\
        There is no significant difference between group 1 and 2.")
```

```
Mann-Whitney U Statistic: 7.0
P-value: 0.00035762558774018

Reject the null hypothesis:
There is a significant difference between group 1 and 2.
```

Odds Ratio

Outcome	Smoker	Non_Smoker
Lung Cancer	a	b
No Lung Cancer	c	d

Odds Ratio (OR) = (ad) / (bc)

OR = 1: The odds of the outcome are the same in both groups, indicating no association.

OR > 1: The odds of the outcome are higher in the exposed group compared to the unexposed group, suggesting a positive association.

OR < 1: The odds of the outcome are lower in the exposed group compared to the unexposed group, suggesting a negative association.

MET CS 688 Feature Engineering I

Leila Ghaedi – Fall 2024

Creator: cemagraphics | Credit: Getty Images/iStockphoto

Feature Engineering

- The question of how to represent your data best for a particular application is known as feature engineering, and it is one of the main tasks of data scientists and machine learning practitioners trying to solve real-world problems.
- No machine learning algorithm will be able to make a prediction on data for which it has no information. For example, if the only feature that you have for a patient is their last name, no algorithm will be able to predict their gender. This information is simply not contained in your data. (Garbage in, garbage out!)

Feature Engineering

- Transformations
- Normalization
- Feature Scaling
- Binning (convert continuous data into categorical)
- Extracting meaningful information (convert a raw pixel image data into information such as object recognition)
- Convolution (a kernel slides over an image)
- Bag of Words (BoW), word2vec (Used for Natural Language Processing)

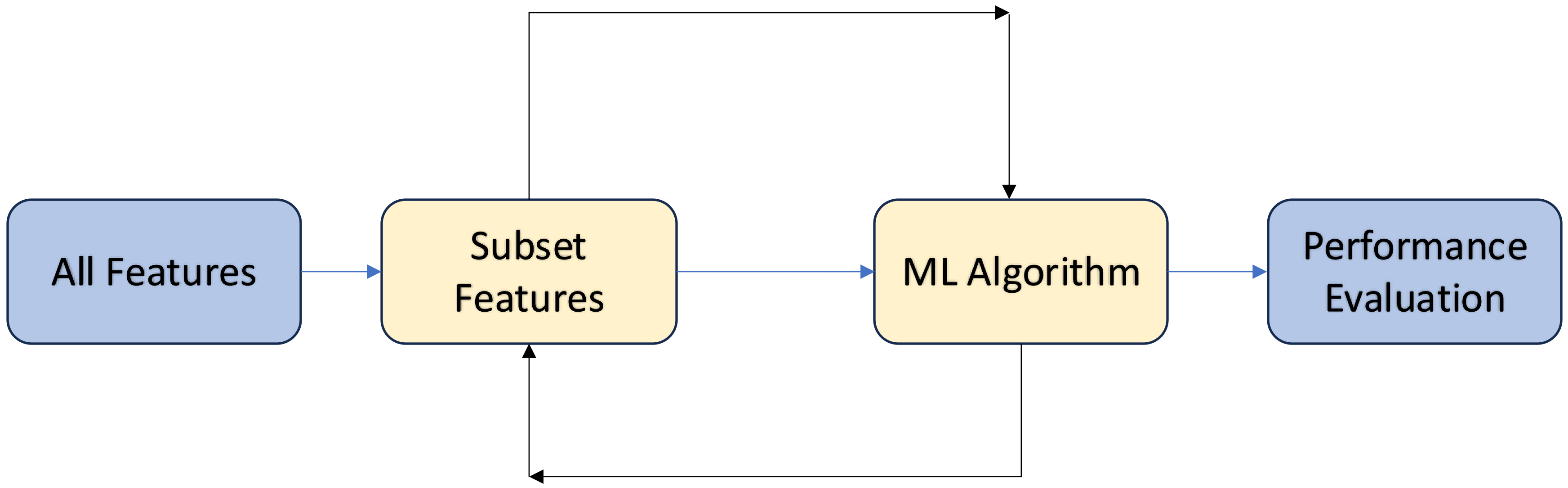
Filtering Methods

- There are three known feature selection methods:
  1. Filtering
  2. Wrapper Methods
  3. Embedded Methods
- Filtering method process features and removes the ones which are not relevant to the machine learning algorithm goal (such as classification, prediction)
- While we are dealing with numerical data, correlation analysis (Pearson, Kendall and Spearman) is one of the most convenient filtering methods to remove the redundant feature.
- Another filtering method is to use covariance. (positive covariance -> two features tend to change together)

Wrapper Method

- Wrapper methods are a machine learning algorithm which identify feature subsets from high-dimensional datasets.
- The wrapper method uses a search algorithm to locate possible subsets of features and measure the accuracy of each subset selection against a specific machine learning algorithm.
- The wrapper method considers the selection of feature sets as a search problem, where different combinations are prepared, evaluated and compared to other combinations.

Wrapper Method for Feature Selection



Wrapper Method

- There are two categories of wrapper methods:
    - Sequential search methods
    - Heuristic search methods
- A good example of heuristic methods are deep learning optimization algorithms that use gradient descent.

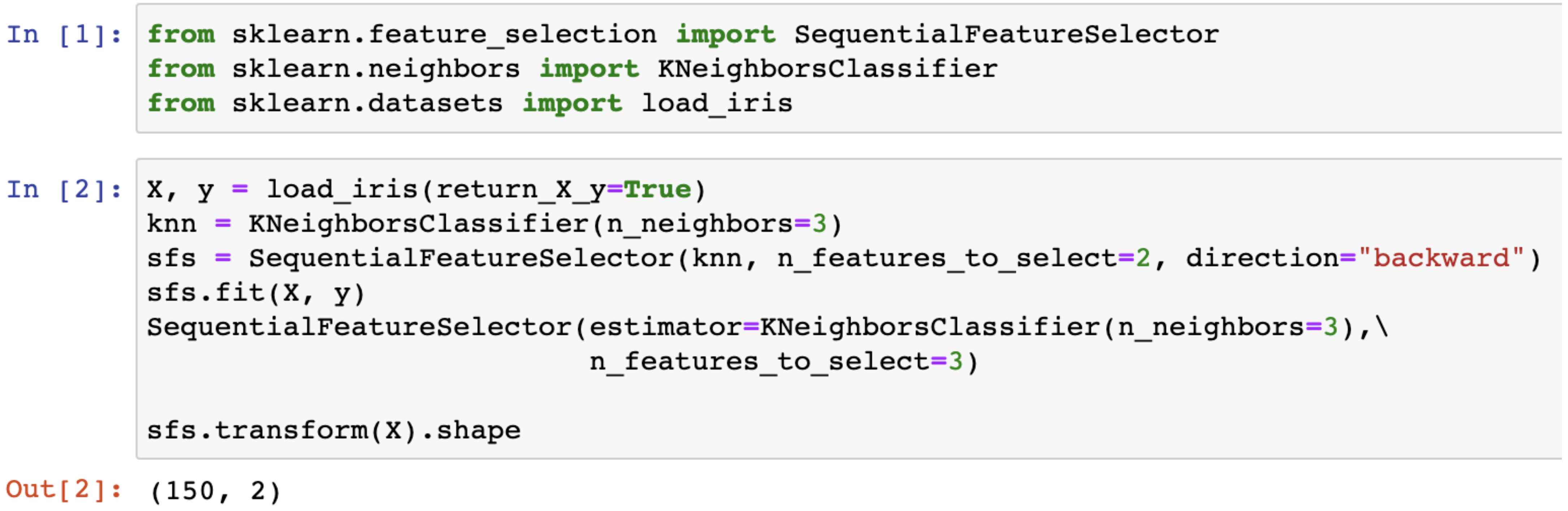
Sequential Feature Selection (SFS)

- Sequential feature selection:
    1. Sequential Forward Selection
    2. Sequential Backward Selection
- Sequential feature selection is a supervised approach to feature selection. It can be used to eliminate uninformative features from a large dataset.
- SFS is a greedy search algorithms.
- In forward selection, the process begins with an empty set of features, and after each iteration, it adds a feature and evaluates its performance to check whether it is improving the performance.

Sequential Backward Selection (SBS)

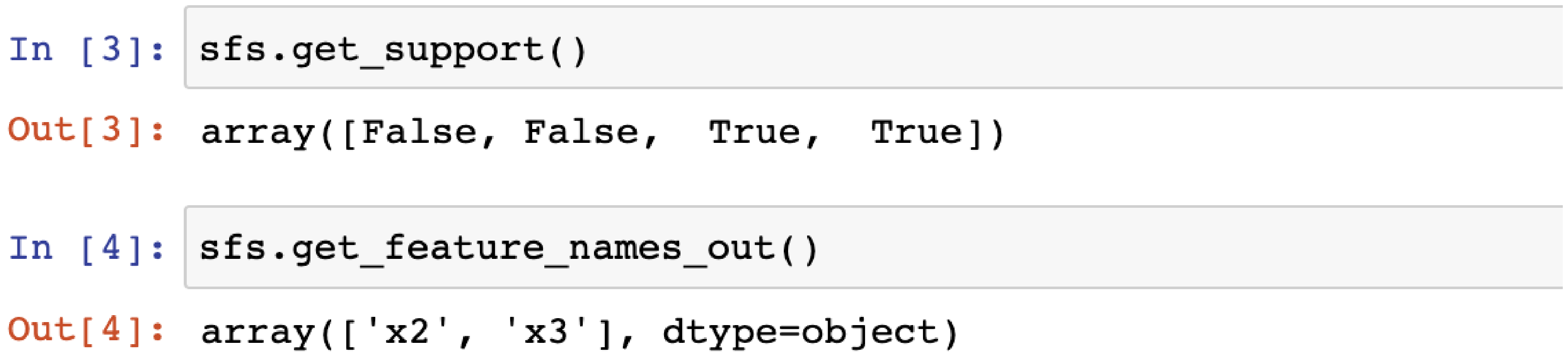
- It starts with the largest set of features, then removes features one by one and check the accuracy (or other objective function), until a desired accuracy has been achieved.
- It operates in contrast to SFS, because first it feeds all features into the algorithm, then it reduces them one by one to study changes in the objective function.

Sequential Feature Selection Example



```
In [1]: from sklearn.feature_selection import SequentialFeatureSelector
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.datasets import load_iris

In [2]: X, y = load_iris(return_X_y=True)
        knn = KNeighborsClassifier(n_neighbors=3)
        sfs = SequentialFeatureSelector(knn, n_features_to_select=2, direction="backward")
        sfs.fit(X, y)
        SequentialFeatureSelector(estimator=KNeighborsClassifier(n_neighbors=3),\
                                  n_features_to_select=3)

        sfs.transform(X).shape

Out[2]: (150, 2)
```

Sequential Feature Selection Example



```
In [3]: sfs.get_support()
Out[3]: array([False, False,  True,  True])

In [4]: sfs.get_feature_names_out()
Out[4]: array(['x2', 'x3'], dtype=object)
```

Embedded Methods for Feature Selection

- Embedded methods are like wrappers, but they use the knowledge of the machine learning algorithm to make the search more efficient.
- In embedded methods, feature selection is built into the classifier algorithm.
- Examples of Embedded Methods
    1. LASSO (Least Absolute Shrinkage and Selection Operator): L1 penalty
    2. RIDGE: L2 penalty
    3. Genetic Algorithm

LASSO Embedded Feature Selection

- L1 Penalty: sum of absolute values of regression coefficients.
- The penalty term is added to objective function of a machine learning algorithm. The penalty term in lasso regression forces some coefficient estimates to zero, causing variable selection.

Ridge Embedded Feature Selection

- L2 Penalty: sum of square values of regression coefficients.
- The penalty term is added to objective function of a machine learning algorithm. The penalty term in lasso regression forces some coefficient estimates to zero, causing variable selection.

Wrapper Methods
Heuristic Methods (Genetic Algorithms)

- Heuristics are techniques which try to solve a problem by finding an approximation instead of a best answer.
- Genetic Algorithms are well known methods to solve a problem in heuristic fashion.
- They are based on the Darwin‘s theory of Survival of the Fittest (Natural Selection). Genetic algorithms are used to find ’optimal‘ or ’near optimal‘ solution for a problem which is hard to solve (these problems are called NP-Hard problems).
- Genetic algorithms try to search for a solution by somehow a random search, but they perform better than purely random search or brute force search.

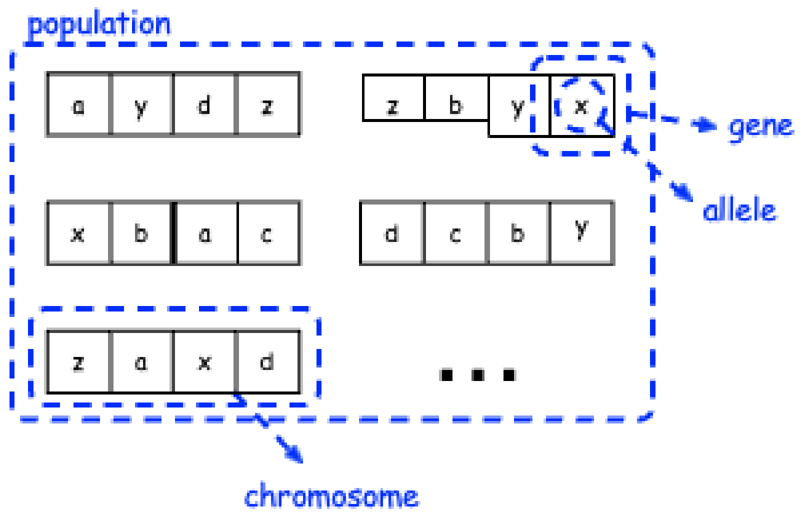
Genetic Algorithm

- We have a pool or population of solutions (or answers). Each solution is called chromosome, and each chromosome has a set of variables or parameters which is called gene. The value of each gene is called allele.
- Think of a solution as a set of parameters that their combination yield a result and this result will be represented as a fitness score.

gene = variable

allele = value

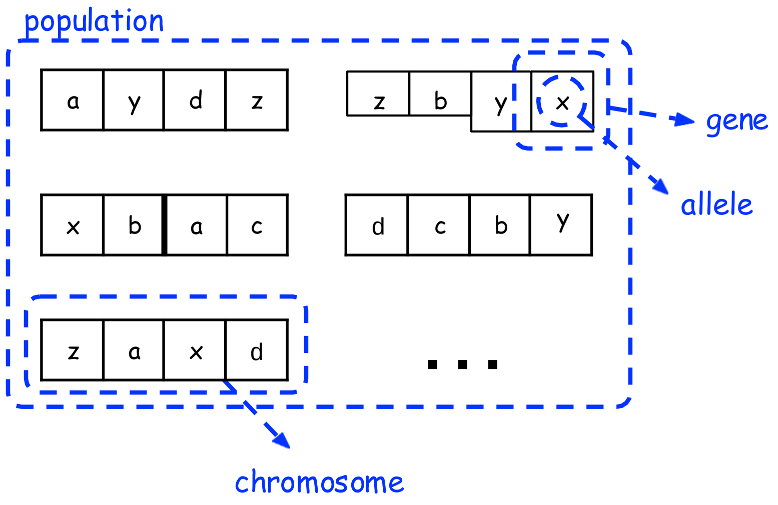
chromosome = set of variables



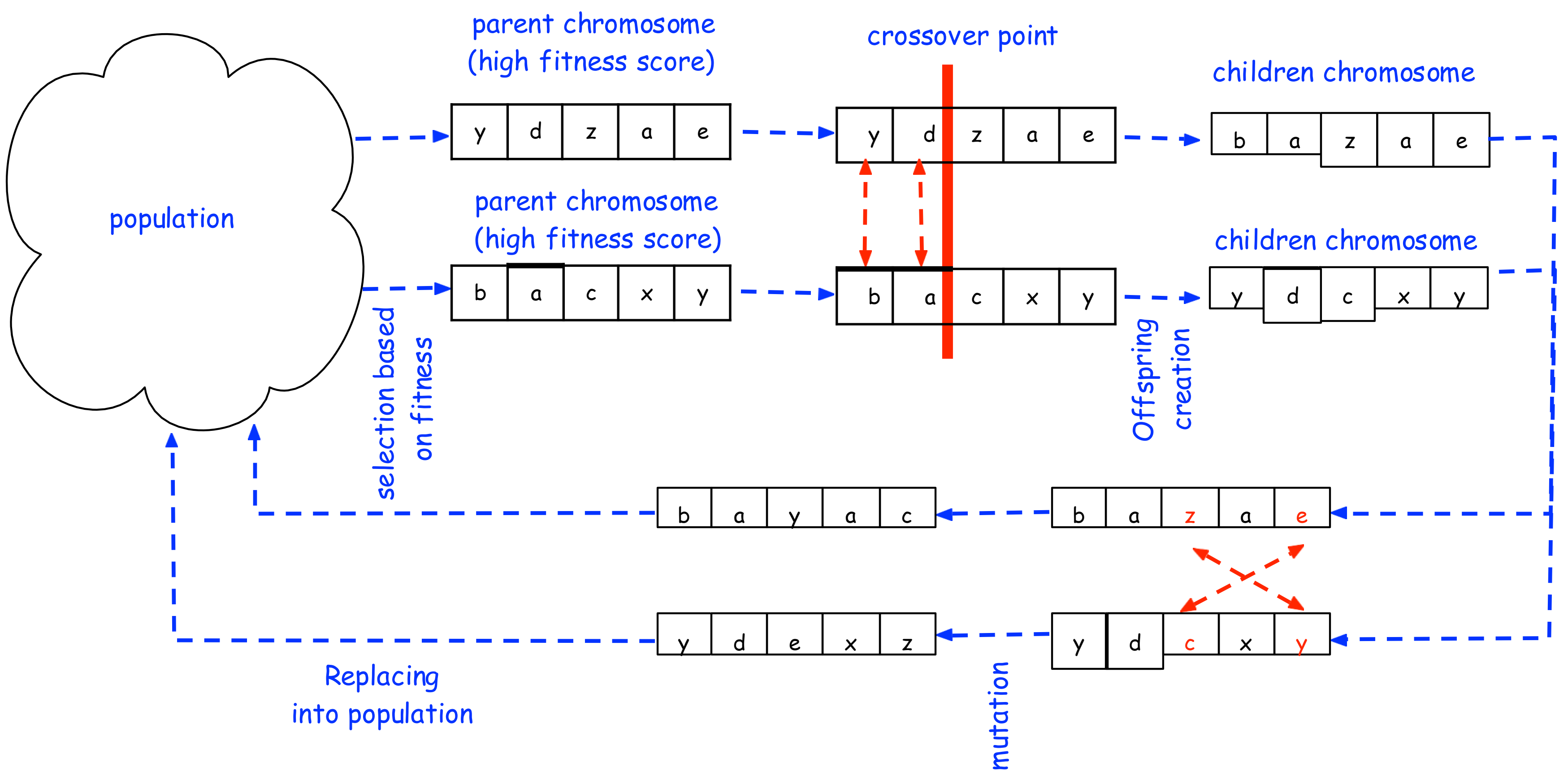
Entities which compose the population in genetic algorithms

Genetic Algorithm

Population: All possible solutions.
Chromosome: A single solution.
Gene: Variables or parameters, which characterize the chromosome.
Allele: The value inside a gene. In other word, the value of the variable (gene).
Fitness score & Fitness function: A cost of a solution (chromosome), is acquired by a function called fitness function and this cost is presented in a value that is called fitness score.
Selection: Is a process in which it tries to find chromosomes with a high fitness scores for mating. The mating process here means they pass their genes together and make new children.
Crossover: Is a process in which two chromosomes (parents) combine together and pass over their genes to create new children (offspring).

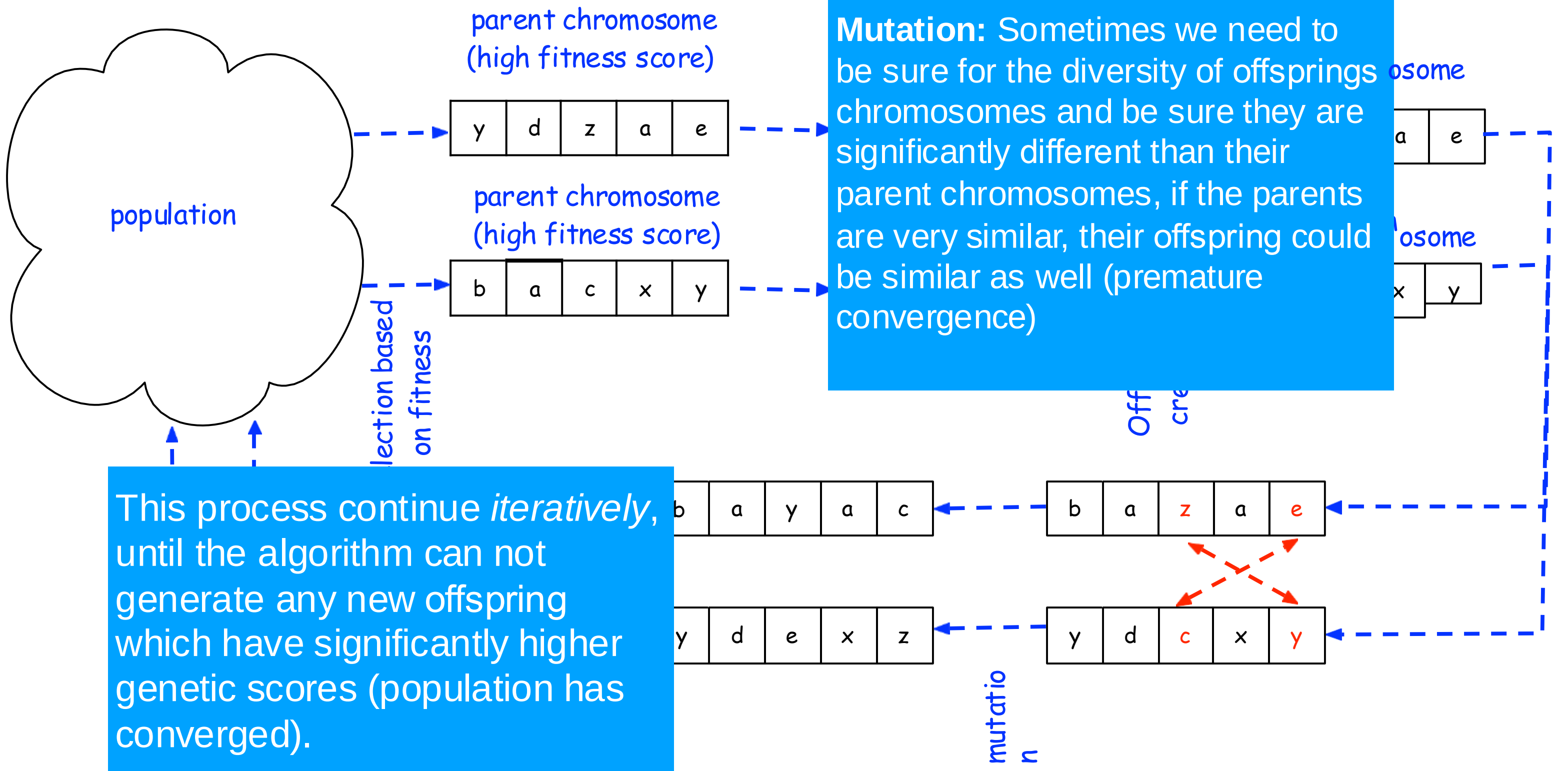


Genetic Algorithm



Two chromosome with high fitness score will be selected and based on the given crossover point two children chromosome will be created. Then **mutation will be done to increase their diversity**, and newly generated chromosomes will be replaced low fitness score chromosomes in the population.

Genetic Algorithm



Mutation: Sometimes we need to be sure for the diversity of offsprings chromosomes and be sure they are significantly different than their parent chromosomes, if the parents are very similar, their offspring could be similar as well (premature convergence)

This process continue *iteratively*, until the algorithm can not generate any new offspring which have significantly higher genetic scores (population has converged).

Two chromosome with high fitness score will be selected and based on the given crossover point two children chromosome will be created. Then mutation will be done to increase their diversity, and newly generated chromosomes will be replaced low fitness score chromosomes in the population.

Feature Generation

- Feature generation is the process of combining, mixing, merging, … two or more raw data objects and create a new data object from them.

- Why do we do that? Because of reducing the computational cost, while maintaining or even improving the accuracy of the data.

- Example: Fitness tracker and three axis data, X, Y, Z …

- A very simple feature generation method multiply all features together, **interaction features** or **polynomial feature creation**. For instance, if we have following features: x_1, x_2, x_3, x_4

- Their interaction feature could be as follows:

$$x_1^2, x_1 . x_2, x_1 . x_3, x_1 . x_4, x_2^2, x_2 . x_3, x_2 . x_4, x_3^2, x_3 . x_4, x_4^2 .$$

Feature Engineering for Numerical Data

- Sanity check checks whether the data is in proper form. For example, it checks whether there is a negative value, missing values, etc.

- Normalizing is the process of converting data objects from different format into a comparable and unit-independent format. The normalization could also include identifying the dataset‘s maximum and minimum boundaries, binning or quantizing the data as well.

- Quantizing the data or binning the data means substituting the original values of the data with a discrete and more generalized value. For example, we can quantize the ”time of the day“ into four bins as follows:

  * original data: {00:00, 01:00 , 02:00, … 23:59 }

  * quantized data: {mid night, morning, afternoon, evening, night}

- Discretizing the data is converting continuous data into a discrete unit of data.

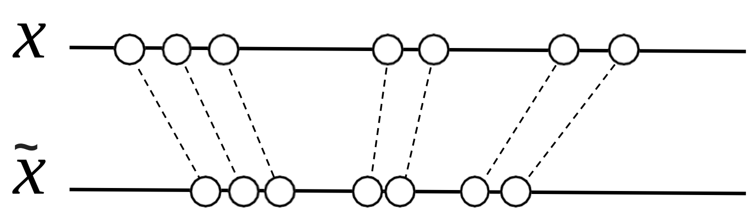
  For example, 11.22 (float) —> 11 (integer), 13.3462 (float)—> 13 (int)

Scaling Numerical Data

- While working with ML algorithm we always have the challenge to deal with computer resources, and to mitigate this challenge one solution is to scale data, e.g. getting rid of large floating numbers.

- Min_Max Scaling: is used to compress (squeeze) or stretch feature values, Assume. We have ***x***, which is vector of variable its Min-Max scaling factorial be shown as $\tilde{x}$.

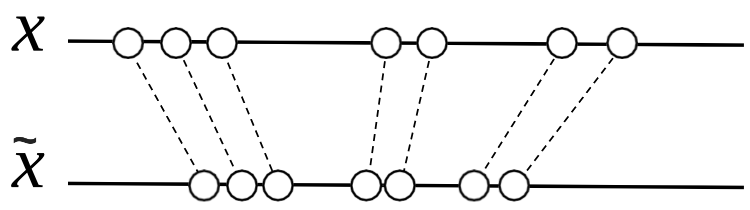
$$\tilde{x} = \frac{x - min(x)}{max(x) - min(x)}$$



Scaling Numerical Data

(1) **Min-Max scaling** is used to compress (squeeze) or stretch feature values. Assume we have X, which is vector of variable its Min-Max scaling factorial be shown as $\tilde{X}$.

$$\tilde{x} = \frac{x - min(x)}{max(x) - min(x)}$$



(2) **Standardization** or **variance scaling** or **z-score**

$$\tilde{x} = \frac{x - \bar{x}_{\text{mean}}}{\sigma_{\text{sd}}}$$

(3) **L2-norm** or **Euclidean norm**

$$||x||_2 = \sqrt{x_1^2 + x_2^2 + \ldots x_n^2}$$

$$\tilde{x} = \frac{x}{||x||_2}$$

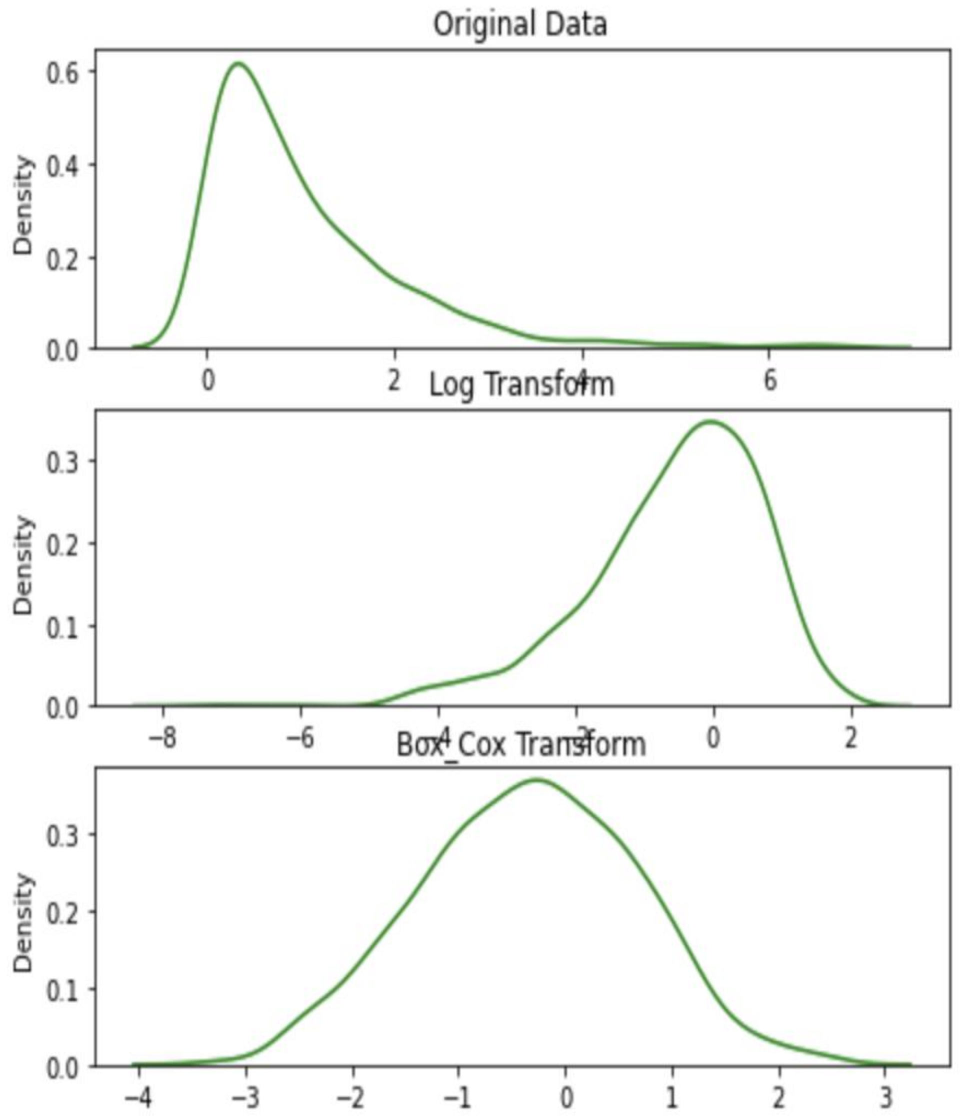
Transformation

(1) Logarithm Transformation It is used to reduce the emphasis on high density data and increase the emphasize on low density data.

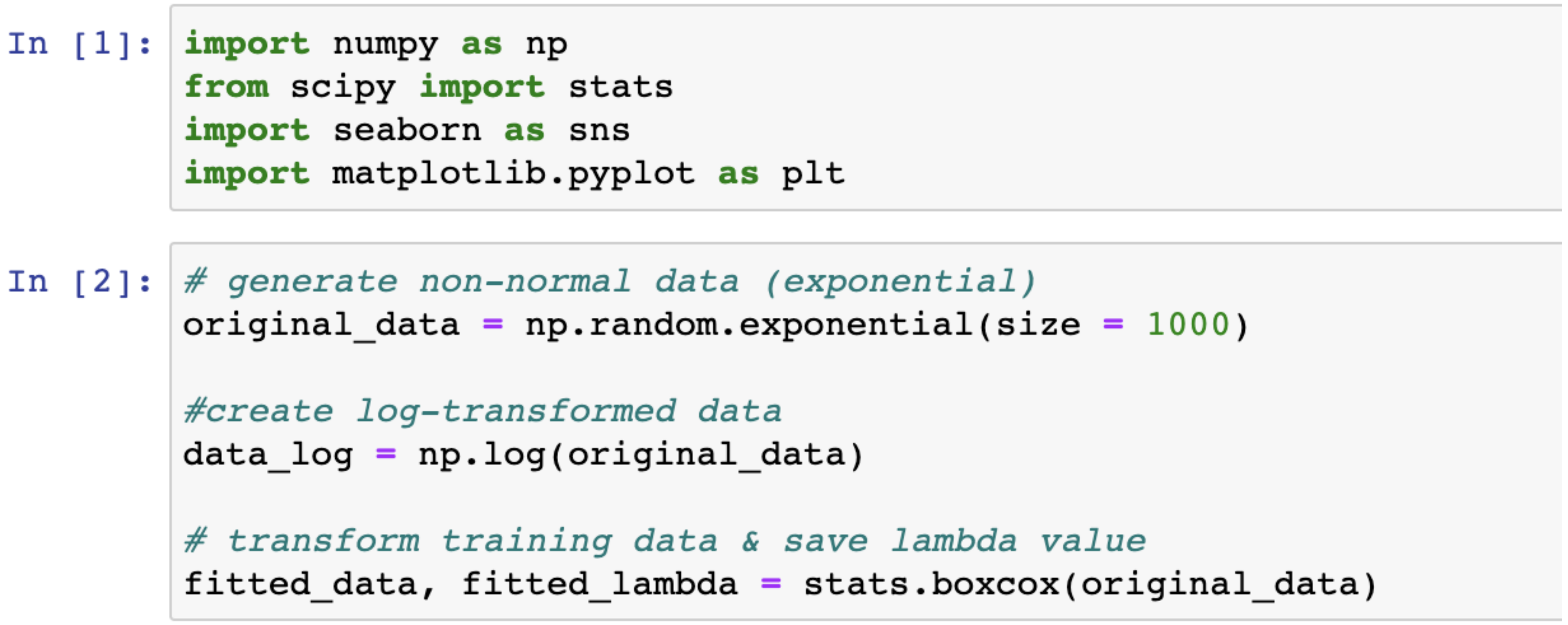
(2) Box-Cox Transformation It trying to convert the shape of non-normal distribution into a normal distribution.

$$y(\lambda) = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda} & if \quad \lambda \neq 0 \\ log(y) & if \quad \lambda = 0 \end{cases}$$

The value of λ varies between -5 and +5.



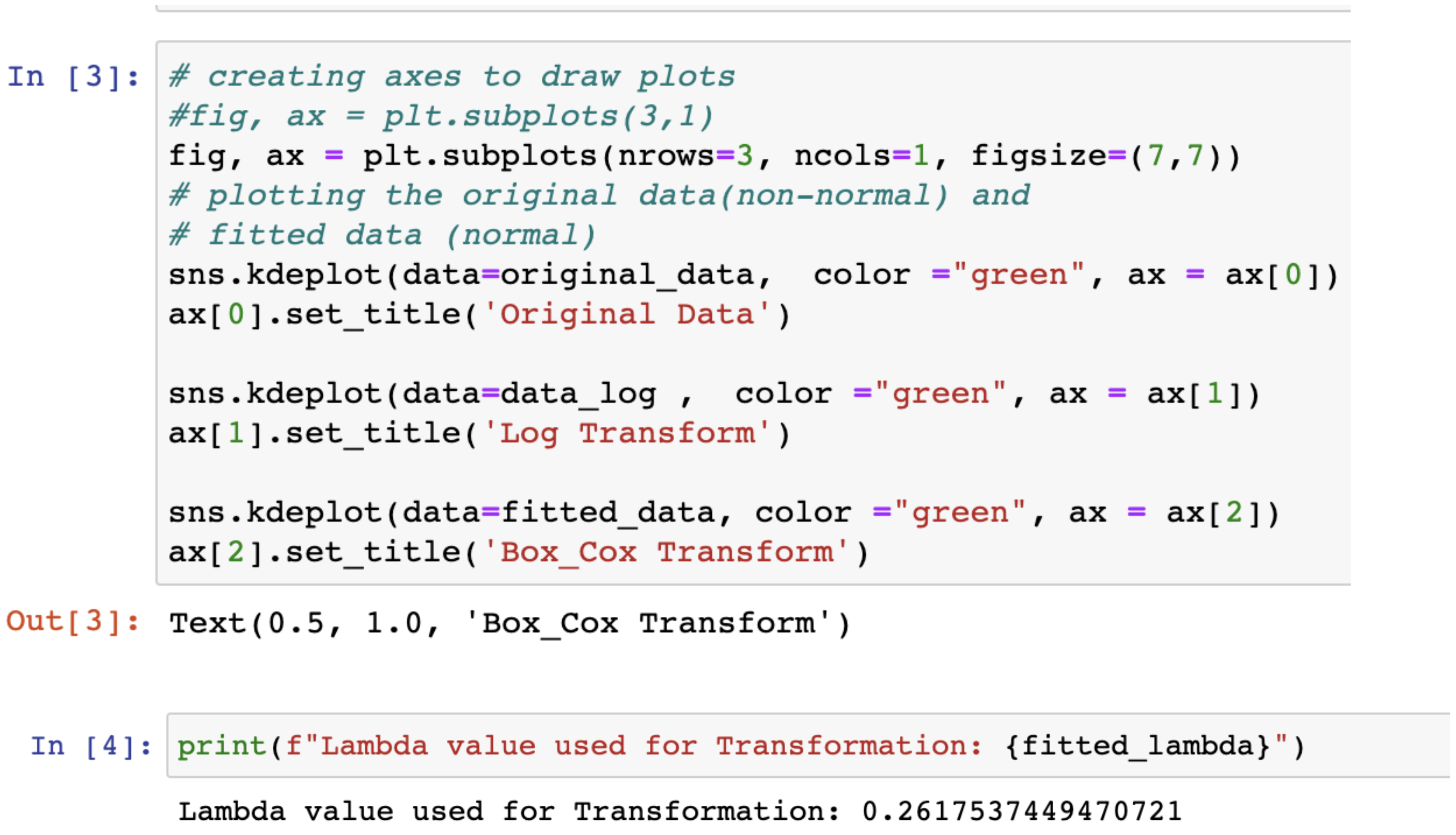
Transformation



```
In [1]: import numpy as np
        from scipy import stats
        import seaborn as sns
        import matplotlib.pyplot as plt
```

```
In [2]: # generate non-normal data (exponential)
        original_data = np.random.exponential(size = 1000)

        #create log-transformed data
        data_log = np.log(original_data)

        # transform training data & save lambda value
        fitted_data, fitted_lambda = stats.boxcox(original_data)
```

Transformation



```
In [3]: # creating axes to draw plots
        #fig, ax = plt.subplots(3,1)
        fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(7,7))
        # plotting the original data(non-normal) and
        # fitted data (normal)
        sns.kdeplot(data=original_data,  color ="green", ax = ax[0])
        ax[0].set_title('Original Data')

        sns.kdeplot(data=data_log ,  color ="green", ax = ax[1])
        ax[1].set_title('Log Transform')

        sns.kdeplot(data=fitted_data, color ="green", ax = ax[2])
        ax[2].set_title('Box_Cox Transform')

Out[3]: Text(0.5, 1.0, 'Box_Cox Transform')
```

```
In [4]: print(f"Lambda value used for Transformation: {fitted_lambda}")

        Lambda value used for Transformation: 0.2617537449470721
```

Some Notes About Transformation

- Both Box-cox and log transform operates when the data is positive. If you are dealing with a dataset that includes negative values as well, you can add a fixed number (constant) to all number (the number could be equal to the positive value of the lowest number) and to all data points and make all your data objects positive.
- The logarithmic transform and box-cox transform are both members of a group called power-transform. These transformations are operating based on ”stabilizing the variance“ of the dataset. In other words, power transformation changes the distribution in a way that variance is no longer dependent on the mean.

Some Notes About Transformation

- Feature selection can increase the bias as well. Usually, in deep learning algorithms, there will be no feature selection and we give all features to the model. Then the algorithm sorts out the useful features itself. However, we do not advocate generalizing a method that works for deep learning algorithms to all other algorithms. Especially, for battery powered devices that have energy limitations having to check too many features is a major challenge.
- Some literature suggests using transformation to change the shape of data into the normal distribution and thus add more flexibility to statistical analysis.

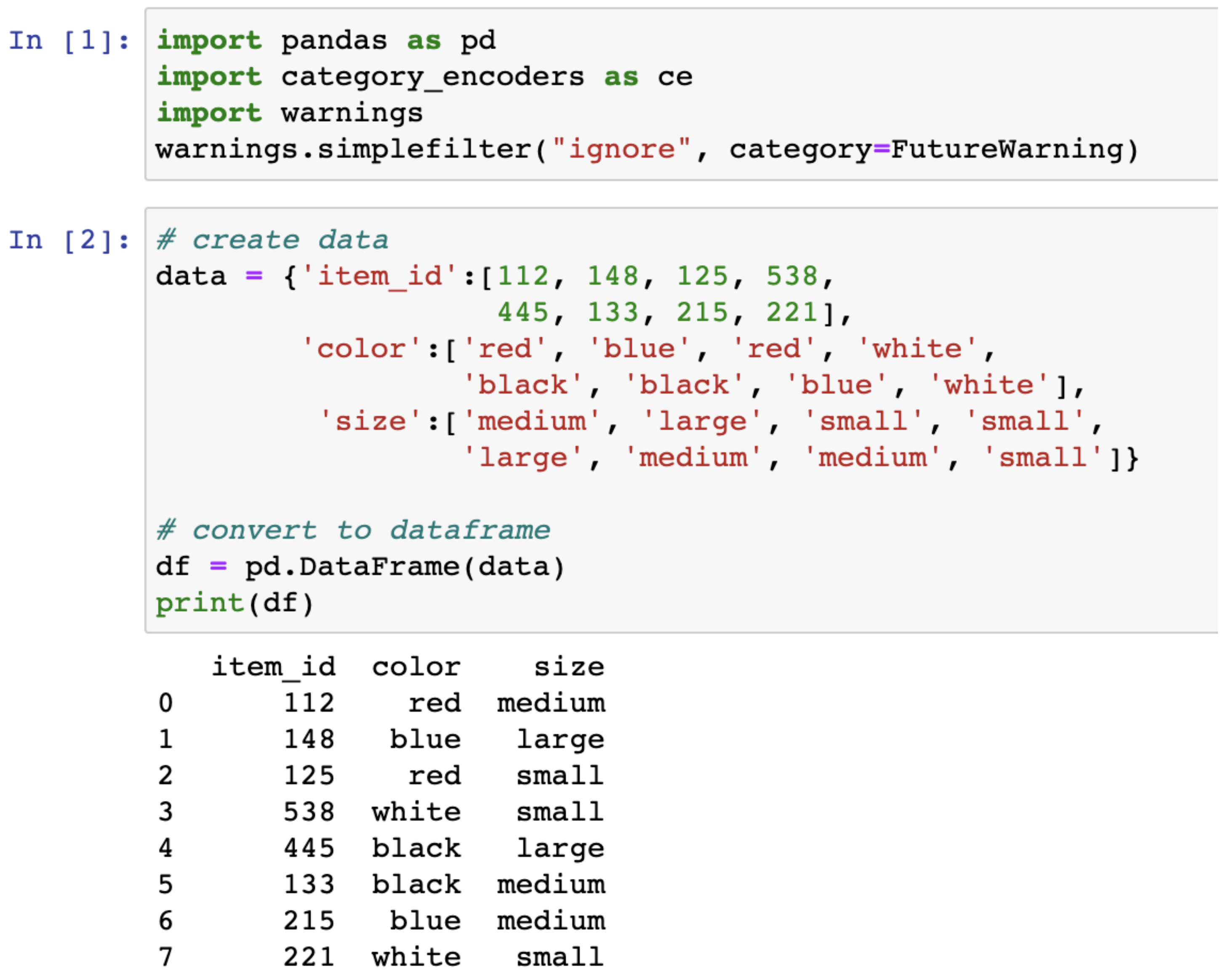
Feature Engineering for Categorical Data

- Categorical data is non-numerical data that represents qualitative information, such as gender, color, or product categories. Before using categorical data in computational models, it must be preprocessed into numerical values. This process is called encoding.
- One_Hot Encoding: Converts any possible values of the data set into a bit(0,1).

Feature Engineering for Categorical Data

- One_Hot Encoding:
  - Converts any possible values of the data set into a bit(0,1).
  - Represents a categorical value with k levels in k bits
- Dummy Encoding:
  - Is like one hot encoding only represents a categorical value with k levels in k-1 bits.
- Effect Encoding:
  - Is like dummy coding, but the reference category (all 0 bits) are presented with -1. It makes interpretation of results for linear regression easier.

One Hot Encoding

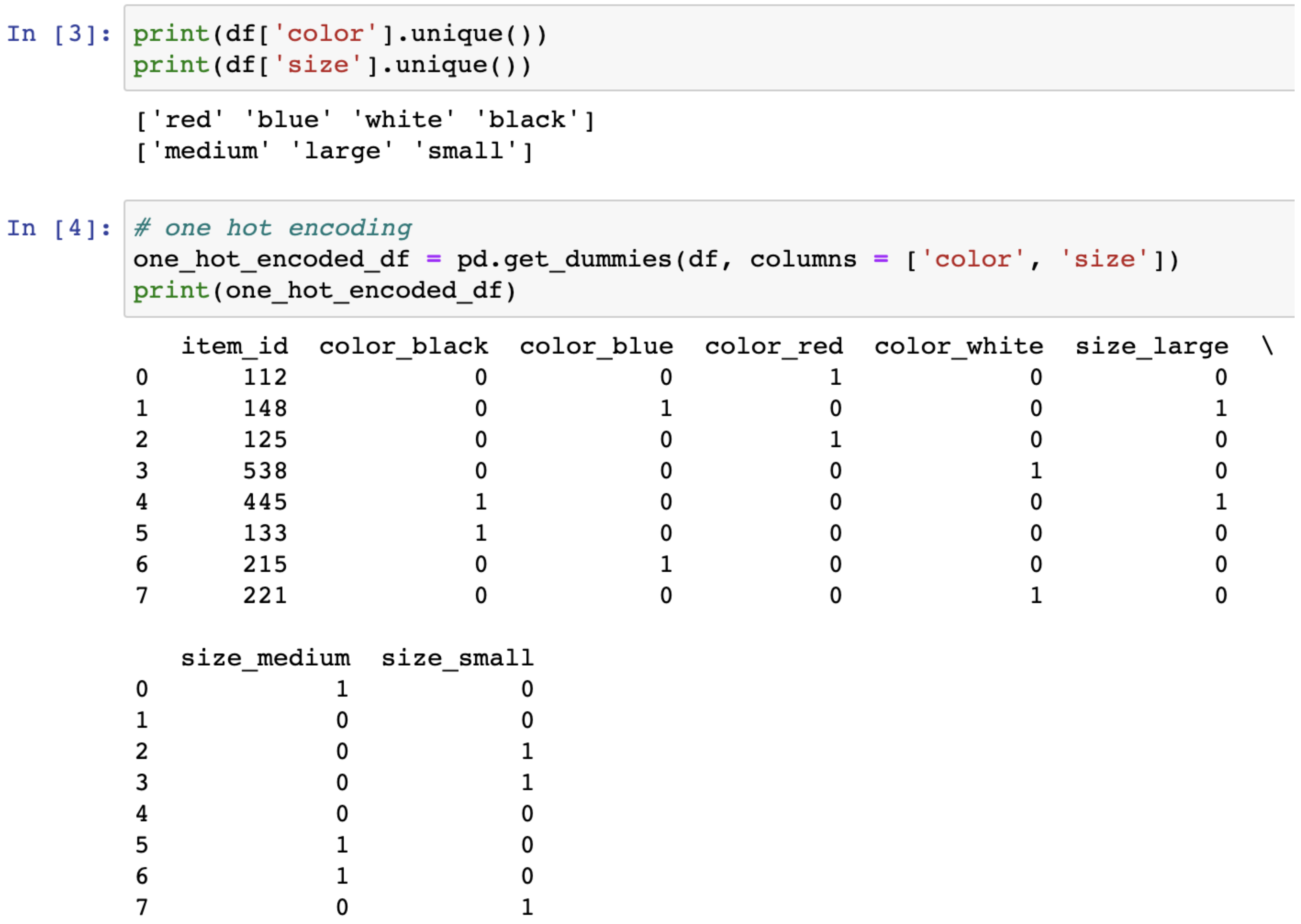


```
In [1]: import pandas as pd
        import category_encoders as ce
        import warnings
        warnings.simplefilter("ignore", category=FutureWarning)
```

```
In [2]: # create data
        data = {'item_id':[112, 148, 125, 538,
                           445, 133, 215, 221],
                'color':['red', 'blue', 'red', 'white',
                         'black', 'black', 'blue', 'white'],
                 'size':['medium', 'large', 'small', 'small',
                         'large', 'medium', 'medium', 'small']}

        # convert to dataframe
        df = pd.DataFrame(data)
        print(df)
```

```
   item_id  color    size
0      112    red  medium
1      148   blue   large
2      125    red   small
3      538  white   small
4      445  black   large
5      133  black  medium
6      215   blue  medium
7      221  white   small
```

One Hot Encoding



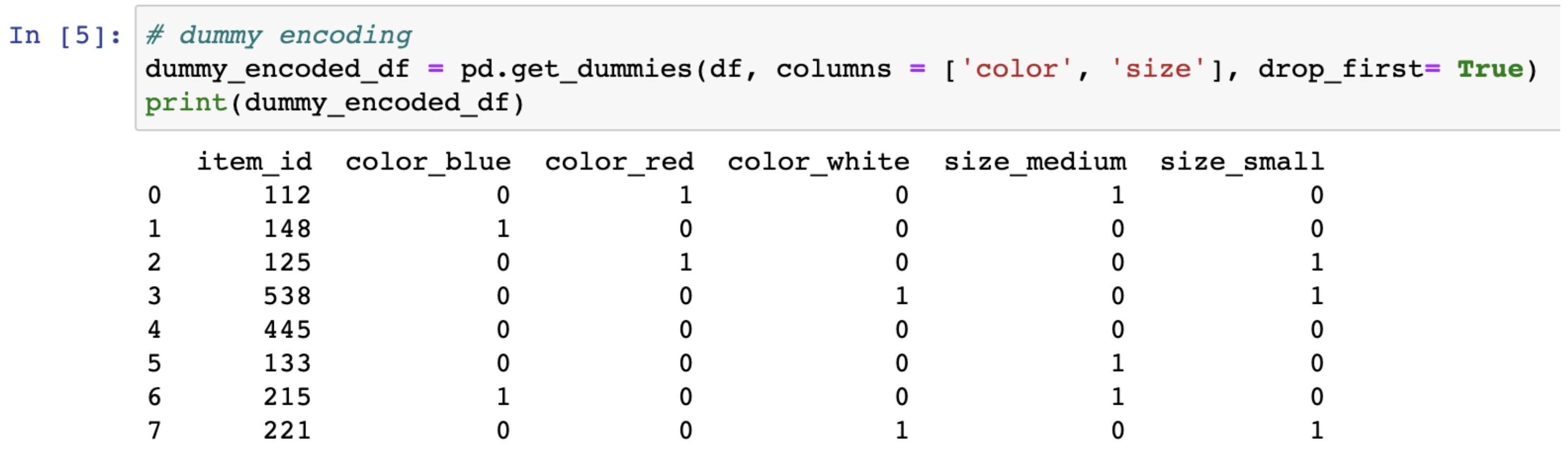
```
In [3]: print(df['color'].unique())
        print(df['size'].unique())
```

```
['red' 'blue' 'white' 'black']
['medium' 'large' 'small']
```

```
In [4]: # one hot encoding
        one_hot_encoded_df = pd.get_dummies(df, columns = ['color', 'size'])
        print(one_hot_encoded_df)
```

```
   item_id  color_black  color_blue  color_red  color_white  size_large  \
0      112            0           0          1            0           0
1      148            0           1          0            0           1
2      125            0           0          1            0           0
3      538            0           0          0            1           0
4      445            1           0          0            0           1
5      133            1           0          0            0           0
6      215            0           1          0            0           0
7      221            0           0          0            1           0

   size_medium  size_small
0            1           0
1            0           0
2            0           1
3            0           1
4            0           0
5            1           0
6            1           0
7            0           1
```

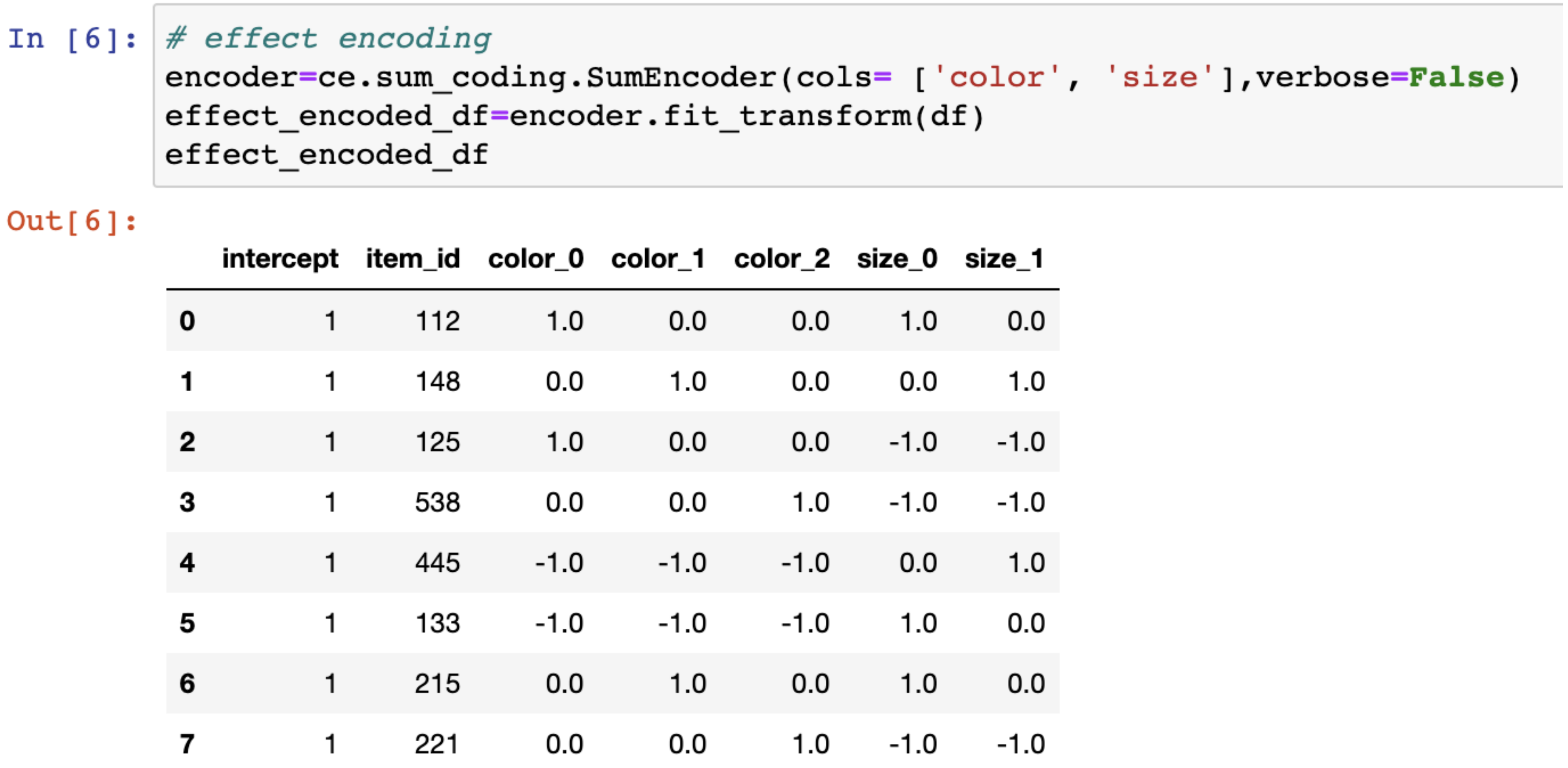
Dummy Encoding



```
In [5]: # dummy encoding
        dummy_encoded_df = pd.get_dummies(df, columns = ['color', 'size'], drop_first= True)
        print(dummy_encoded_df)
```

```
   item_id  color_blue  color_red  color_white  size_medium  size_small
0      112           0          1            0            1           0
1      148           1          0            0            0           0
2      125           0          1            0            0           1
3      538           0          0            1            0           1
4      445           0          0            0            0           0
5      133           0          0            0            1           0
6      215           1          0            0            1           0
7      221           0          0            1            0           1
```

Effect Encoding



```
In [6]: # effect encoding
        encoder=ce.sum_coding.SumEncoder(cols= ['color', 'size'],verbose=False)
        effect_encoded_df=encoder.fit_transform(df)
        effect_encoded_df
```

Out[6]:

	intercept	item_id	color_0	color_1	color_2	size_0	size_1
0	1	112	1.0	0.0	0.0	1.0	0.0
1	1	148	0.0	1.0	0.0	0.0	1.0
2	1	125	1.0	0.0	0.0	-1.0	-1.0
3	1	538	0.0	0.0	1.0	-1.0	-1.0
4	1	445	-1.0	-1.0	-1.0	0.0	1.0
5	1	133	-1.0	-1.0	-1.0	1.0	0.0
6	1	215	0.0	1.0	0.0	1.0	0.0
7	1	221	0.0	0.0	1.0	-1.0	-1.0

Extracting Features from Text Variables

Text (Unstructured Data)

- Text data is unstructured, which means it does not follow a pattern, like the tabular pattern.
- Text may also vary in length and content, and the writing style may be different. How can we extract information from text variables to inform our predictive models?
- The techniques we will cover here belong to the realm of Natural Language Processing (NLP). NLP is a subfield of linguistics and computer science, concerned with the interactions between computer and human language.
- NLP includes a multitude of techniques to understand the syntax, semantics, and discourse of text.

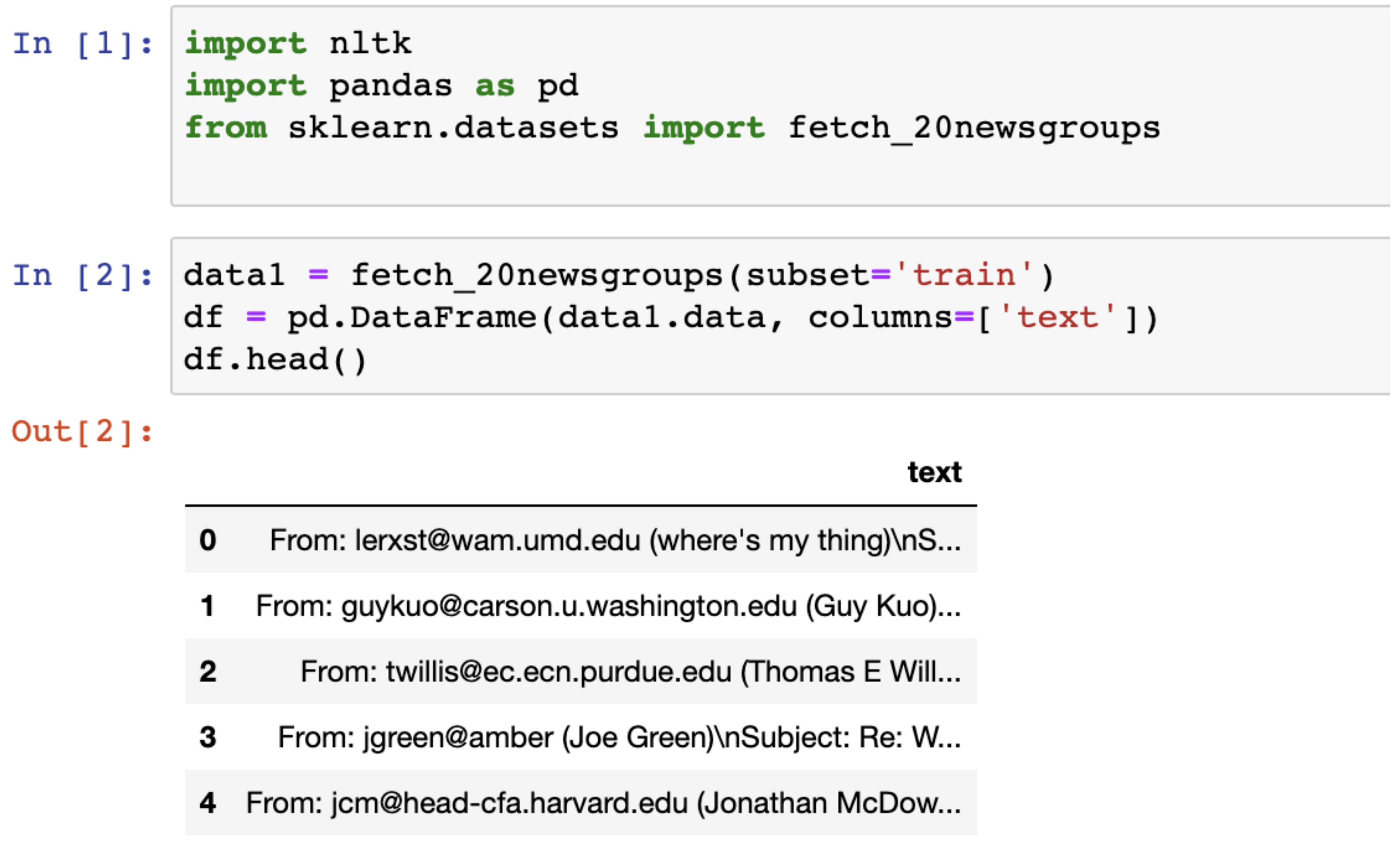
Text Feature Engineering

- Here we discuss techniques that allow us to quickly extract features from short pieces of text, in the context of providing input to predictive models.
- Specifically, we will discuss how to capture text complexity by looking at some statistical parameters of the text such as the word length and count, the number of words and unique words used, the number of sentences, and so on.
- Our tools:
  - pandas
  - scikit-learn libraries
  - Natural Language Toolkit (NLTK)

Text Feature Engineering

- We can capture text complexity by extracting the following information:
  - The total number of characters in the text
  - The total number of words
  - The total number of unique words
  - Lexical diversity = total number of words / number of unique words Word average length = number of characters / number of words

Text Feature Engineering Example



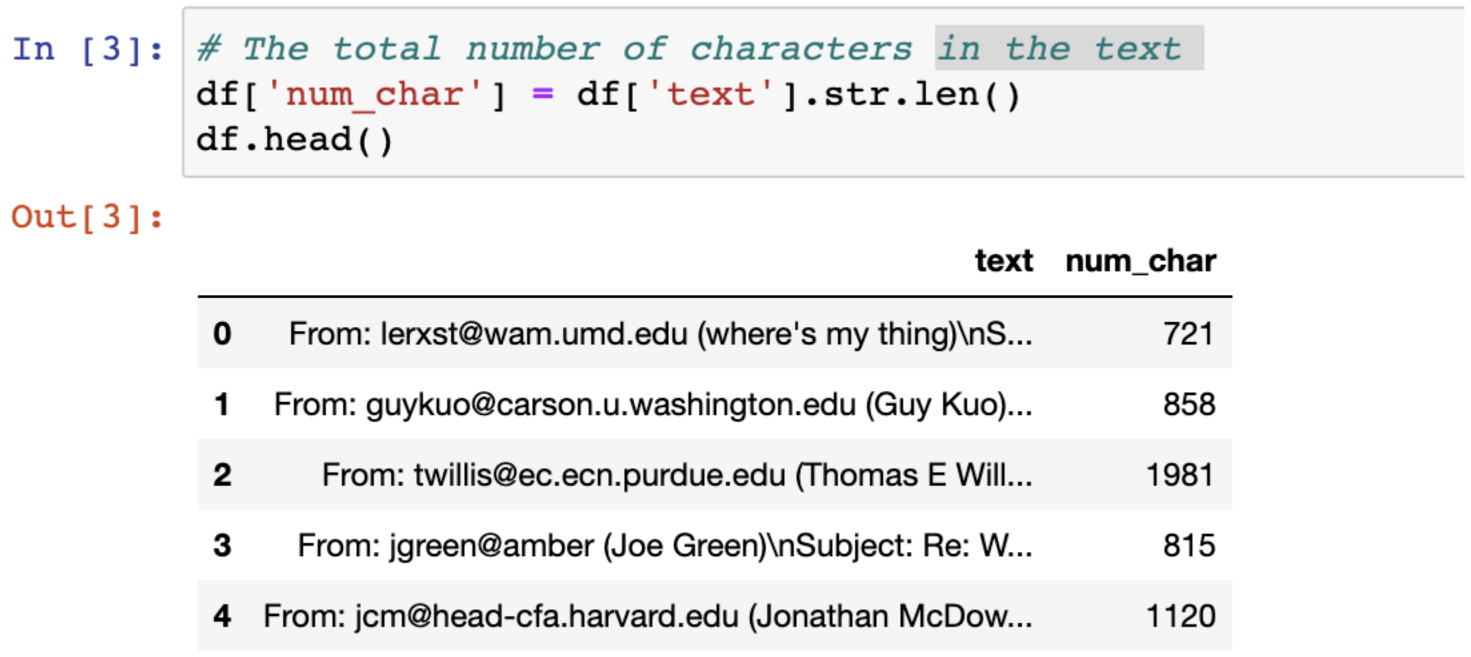
```
In [1]: import nltk
        import pandas as pd
        from sklearn.datasets import fetch_20newsgroups
```

```
In [2]: data1 = fetch_20newsgroups(subset='train')
        df = pd.DataFrame(data1.data, columns=['text'])
        df.head()
```

Out[2]:

	text
0	From: lerxst@wam.umd.edu (where's my thing)\nS...
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...

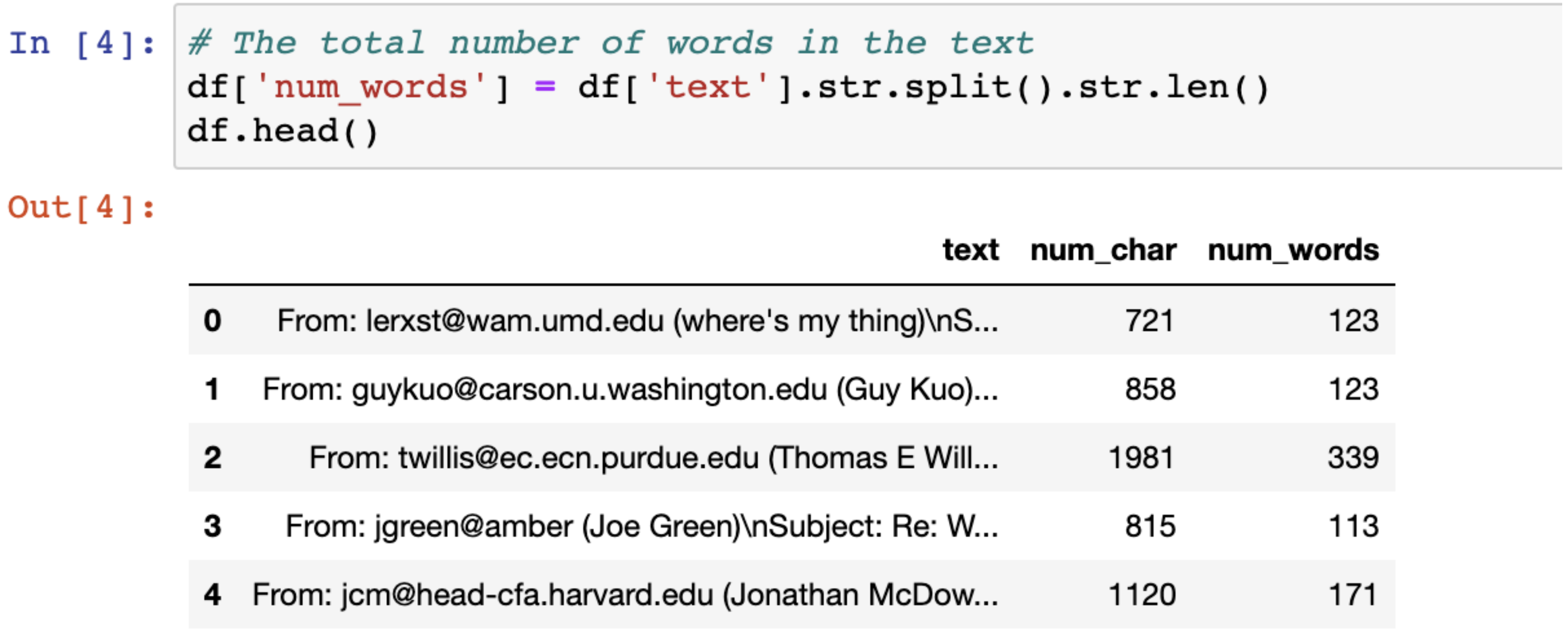
Text Feature Engineering Example



```
In [3]: # The total number of characters in the text
        df['num_char'] = df['text'].str.len()
        df.head()

Out[3]:
```

	text	num_char
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	721
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	858
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	1981
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	815
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	1120

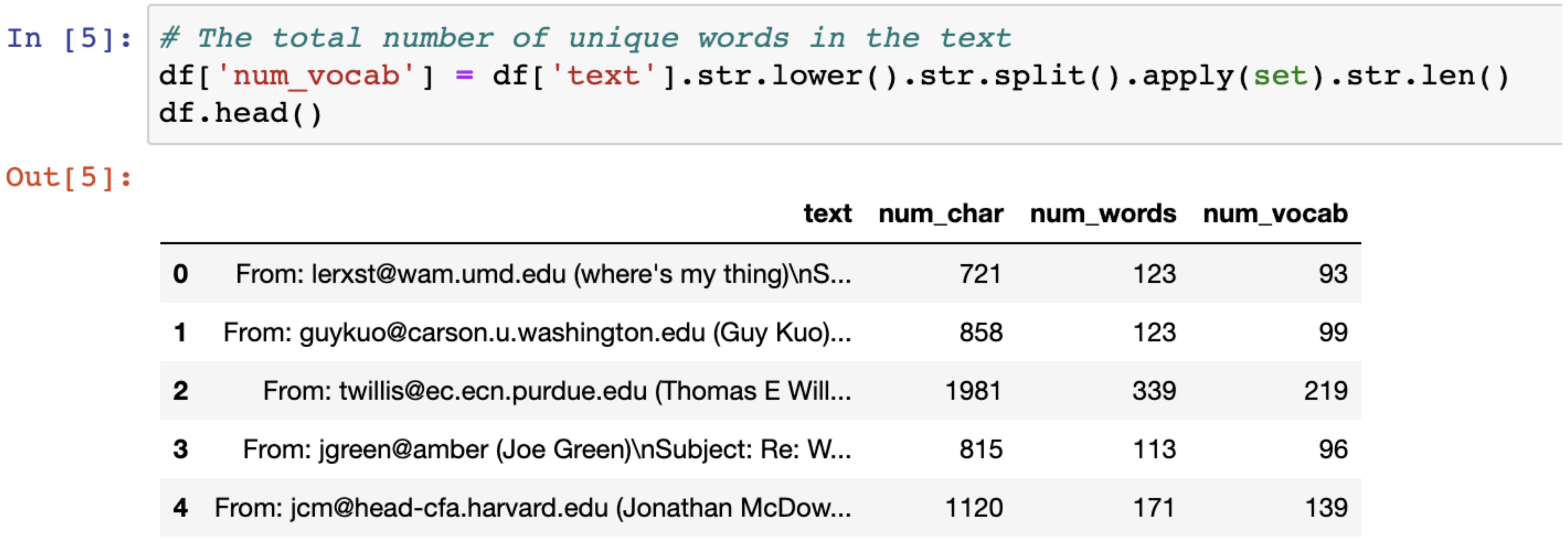
Text Feature Engineering Example



```
In [4]: # The total number of words in the text
        df['num_words'] = df['text'].str.split().str.len()
        df.head()
Out[4]:
```

	text	num_char	num_words
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	721	123
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	858	123
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	1981	339
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	815	113
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	1120	171

Text Feature Engineering Example

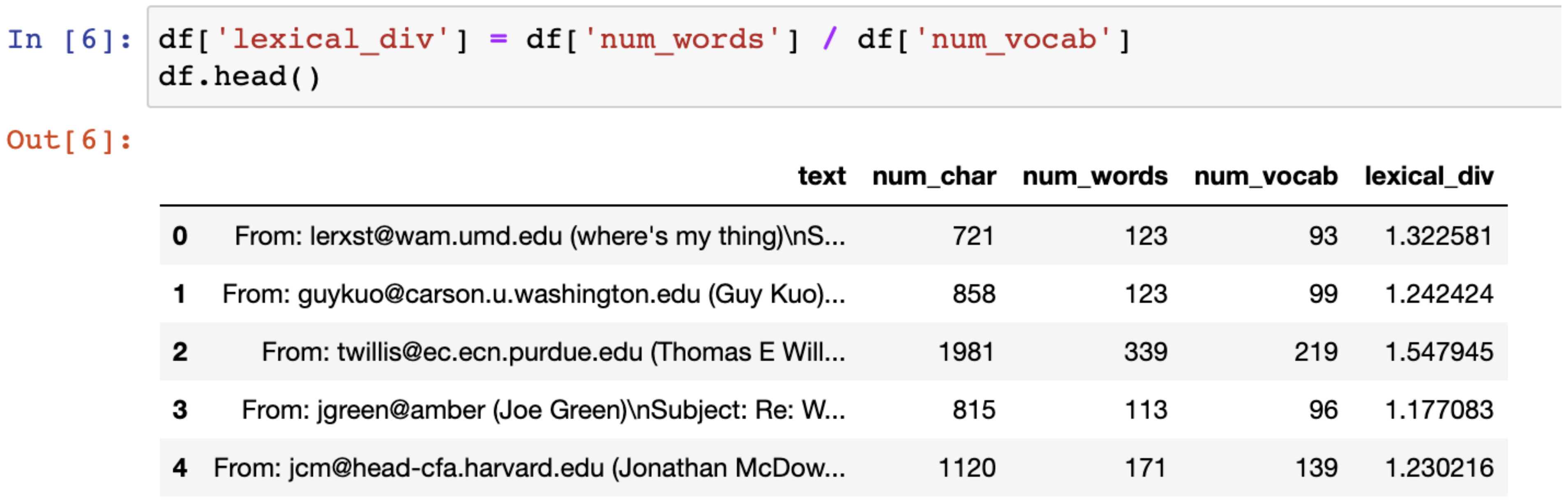


```
In [5]: # The total number of unique words in the text
        df['num_vocab'] = df['text'].str.lower().str.split().apply(set).str.len()
        df.head()
```

Out[5]:

	text	num_char	num_words	num_vocab
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	721	123	93
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	858	123	99
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	1981	339	219
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	815	113	96
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	1120	171	139

Text Feature Engineering Example



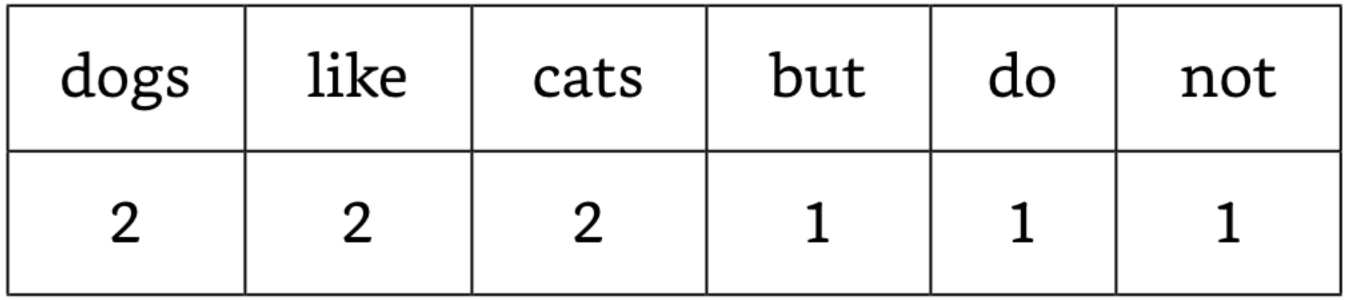
```
In [6]: df['lexical_div'] = df['num_words'] / df['num_vocab']
        df.head()
```

Out[6]:

	text	num_char	num_words	num_vocab	lexical_div
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	721	123	93	1.322581
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	858	123	99	1.242424
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	1981	339	219	1.547945
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	815	113	96	1.177083
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	1120	171	139	1.230216

Bag of Words (BoW)

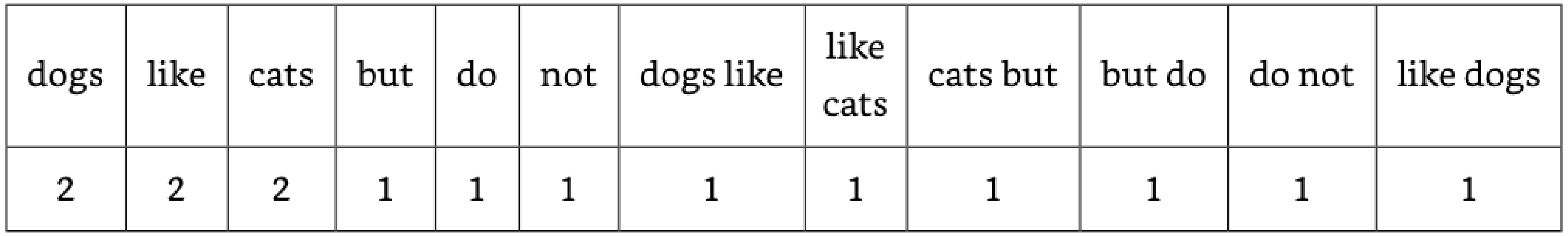
- A bag-of-words (BoW), is a simplified representation of a text that captures the words that are present in the text and the number of times each word appears in the text.
- Example: “Dogs like cats, but cats do not like dogs”



dogs	like	cats	but	do	not
2	2	2	1	1	1

n-grams

- To capture some syntax, BoW can be used together with n-grams. An n-gram is a contiguous sequence of n items in a given text. Continuing with the sentence “Dogs like cats, but cats do not like dogs”, the derived 2-grams are as follows:
  - Dogs like
  - like cats
  - cats but
  - but do
  - do not
  - like dogs

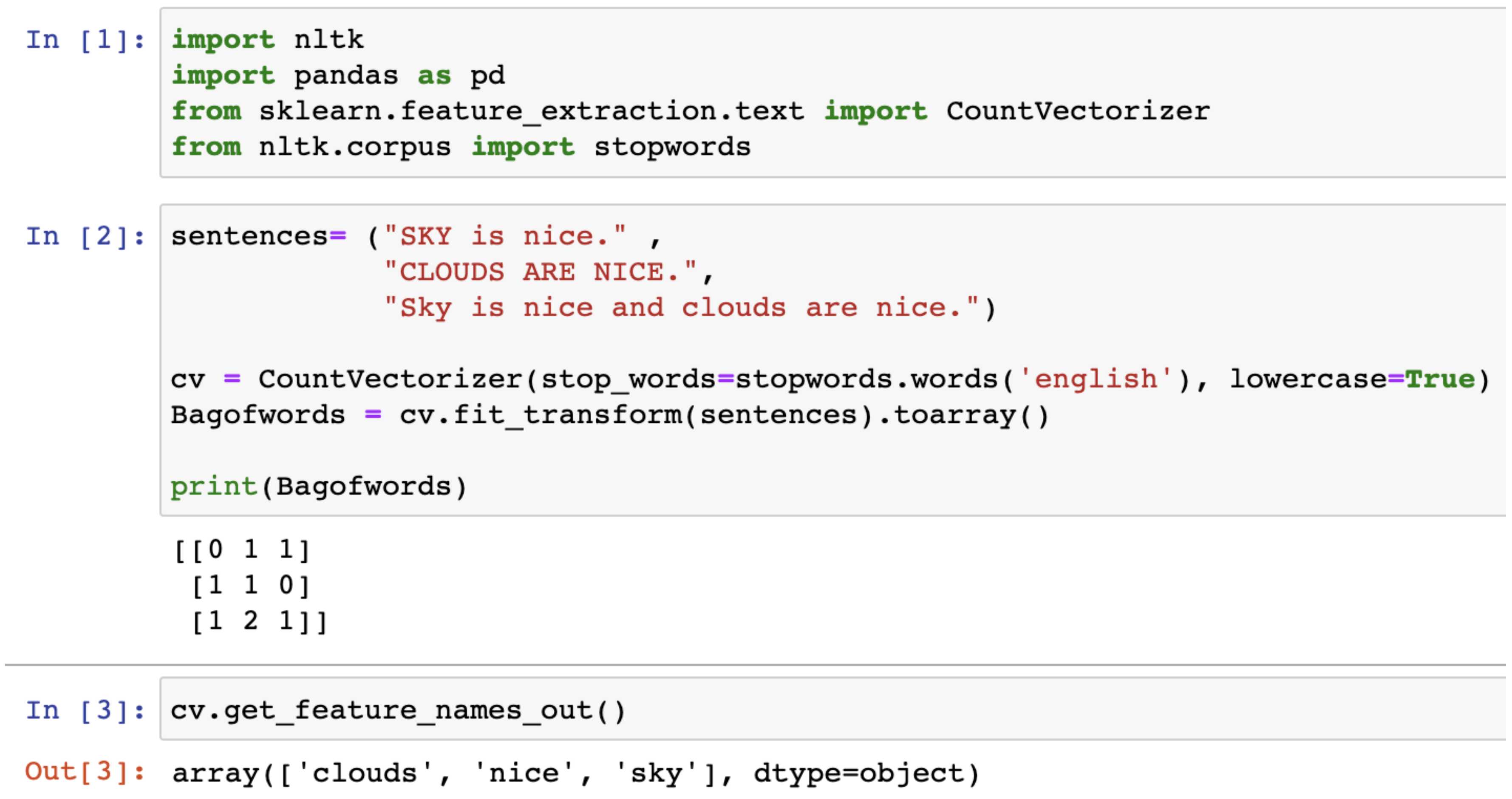


dogs	like	cats	but	do	not	dogs like	like cats	cats but	but do	do not	like dogs
2	2	2	1	1	1	1	1	1	1	1	1

Bag of Words (BoW), n_grams

- CountVectorizer() is a transformer from scikit-learn which sets words in lowercase, removes stop words, retains words with a minimum acceptable frequency, and capture n-grams all together.

Example BoW



In [1]:
```python
import nltk
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
```

In [2]:
```python
sentences= ("SKY is nice." ,
            "CLOUDS ARE NICE.",
            "Sky is nice and clouds are nice.")

cv = CountVectorizer(stop_words=stopwords.words('english'), lowercase=True)
Bagofwords = cv.fit_transform(sentences).toarray()

print(Bagofwords)
```

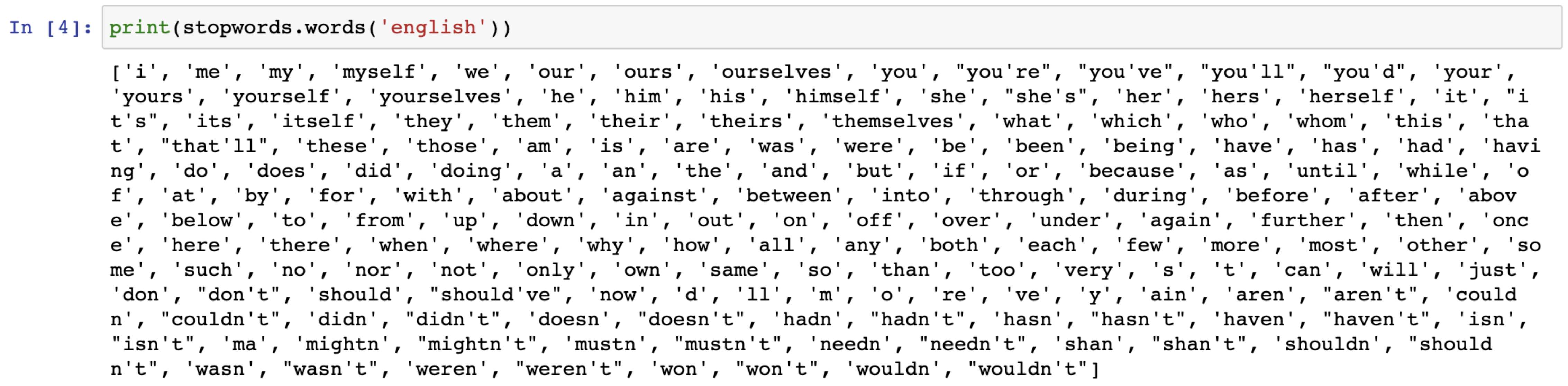
```
[[0 1 1]
 [1 1 0]
 [1 2 1]]
```

In [3]:
```python
cv.get_feature_names_out()
```

Out[3]:
```
array(['clouds', 'nice', 'sky'], dtype=object)
```

Stop Words

Stop words are very frequently used words to make sentences flow, but that, per se, do not carry any useful information.



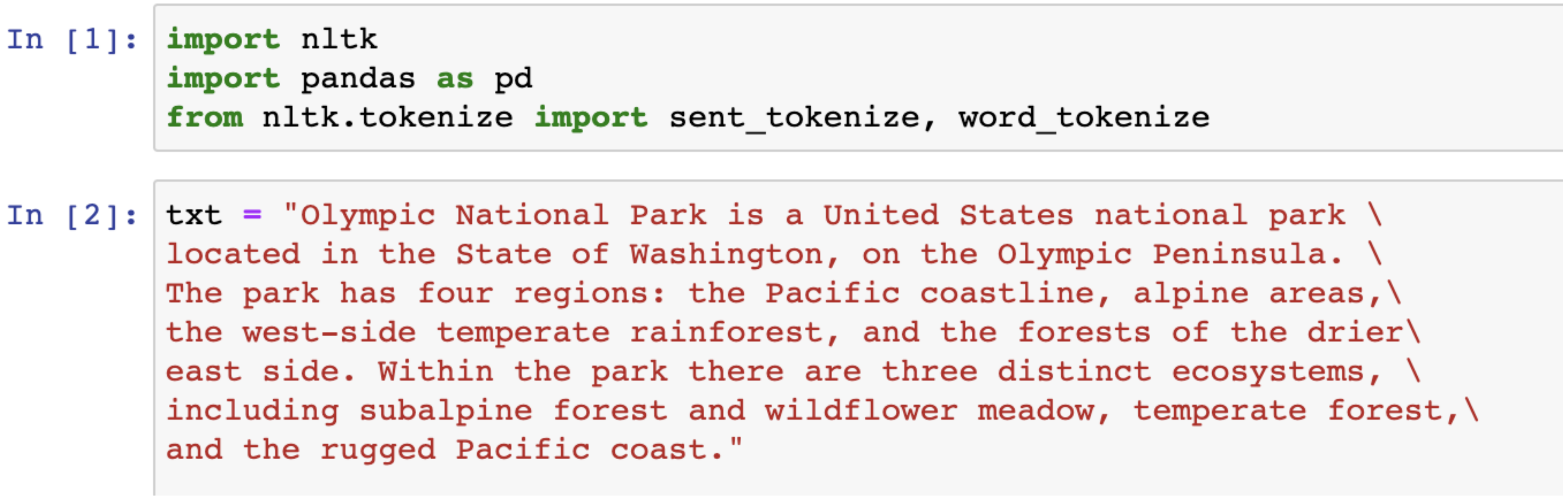
```
In [4]: print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "i
t's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havi
ng', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'o
f', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'abov
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'onc
e', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'so
me', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'could
n', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Tokenization

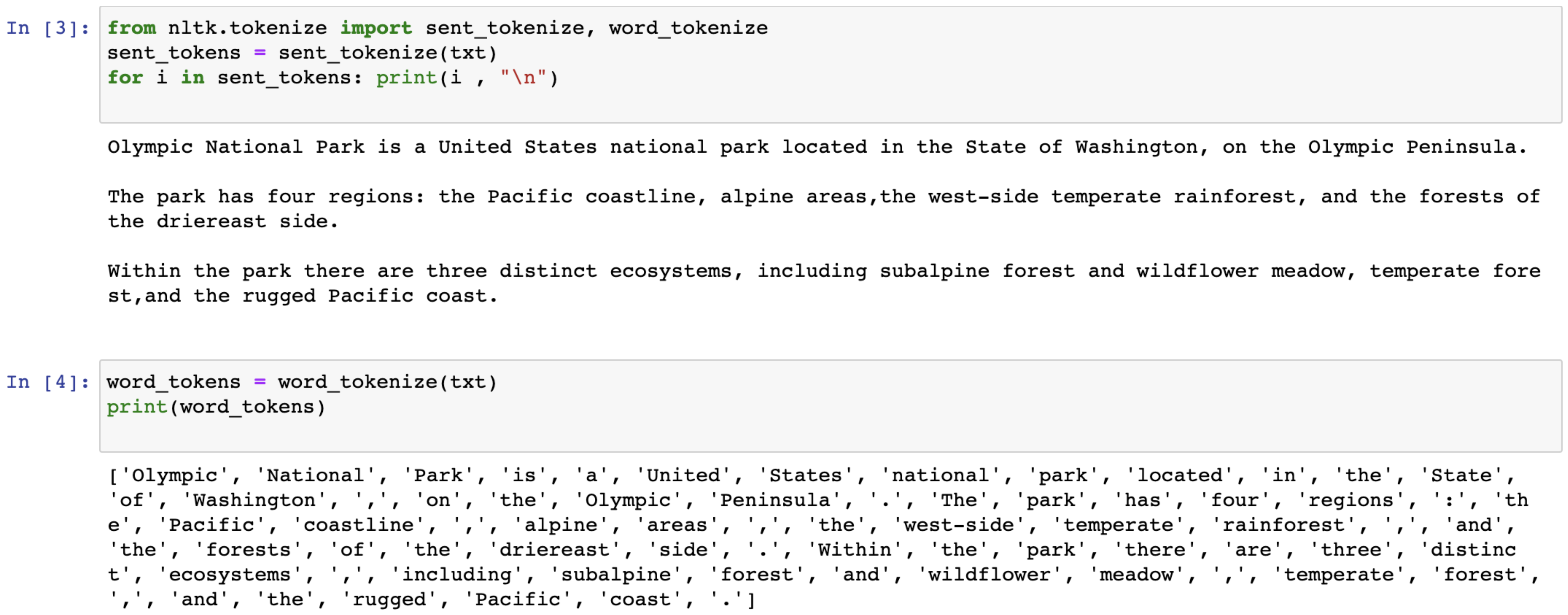
- Tokenization is used in NLP to split paragraphs and sentences into smaller units that can be more easily processed.
- The first step of the NLP process is gathering the data (text) and breaking it into understandable parts (sentences, words).

Tokenization Example



```
In [1]: import nltk
        import pandas as pd
        from nltk.tokenize import sent_tokenize, word_tokenize

In [2]: txt = "Olympic National Park is a United States national park \
        located in the State of Washington, on the Olympic Peninsula. \
        The park has four regions: the Pacific coastline, alpine areas,\
        the west-side temperate rainforest, and the forests of the drier\
        east side. Within the park there are three distinct ecosystems, \
        including subalpine forest and wildflower meadow, temperate forest,\
        and the rugged Pacific coast."
```

Tokenization Example



```
In [3]: from nltk.tokenize import sent_tokenize, word_tokenize
        sent_tokens = sent_tokenize(txt)
        for i in sent_tokens: print(i , "\n")
```

```
Olympic National Park is a United States national park located in the State of Washington, on the Olympic Peninsula.

The park has four regions: the Pacific coastline, alpine areas,the west-side temperate rainforest, and the forests of
the driereast side.

Within the park there are three distinct ecosystems, including subalpine forest and wildflower meadow, temperate fore
st,and the rugged Pacific coast.
```

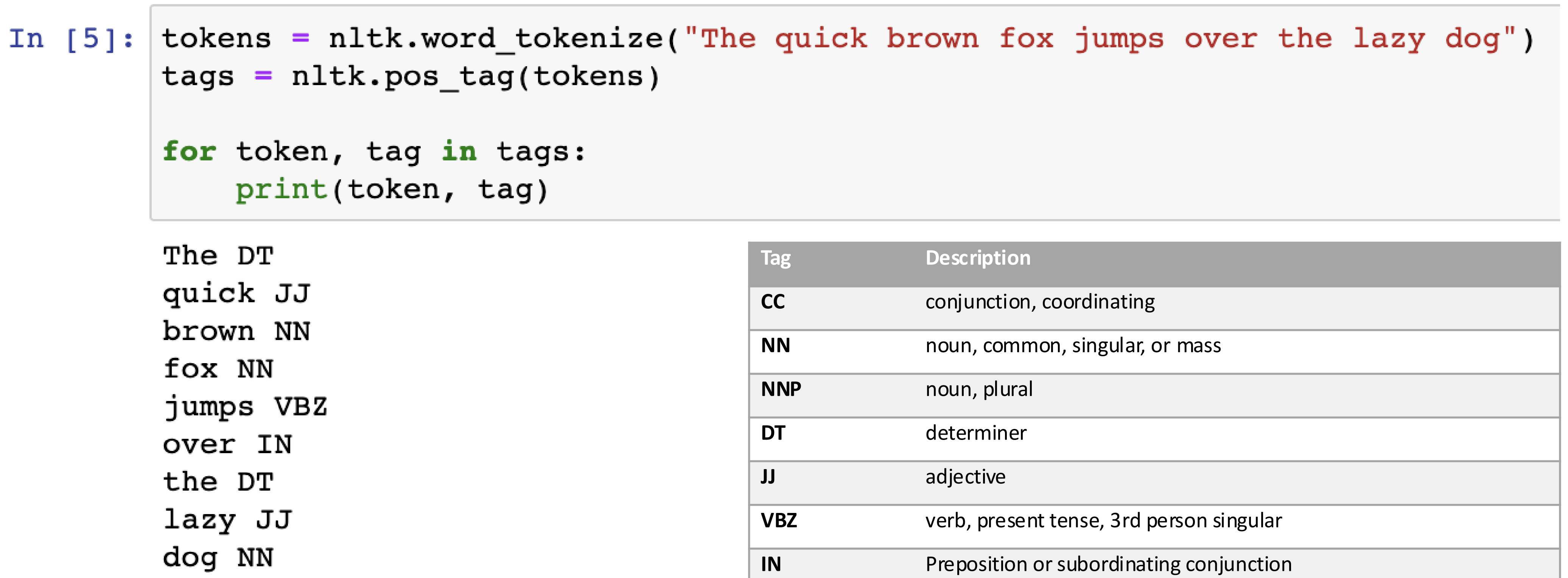
```
In [4]: word_tokens = word_tokenize(txt)
        print(word_tokens)
```

```
['Olympic', 'National', 'Park', 'is', 'a', 'United', 'States', 'national', 'park', 'located', 'in', 'the', 'State',
'of', 'Washington', ',', 'on', 'the', 'Olympic', 'Peninsula', '.', 'The', 'park', 'has', 'four', 'regions', ':', 'th
e', 'Pacific', 'coastline', ',', 'alpine', 'areas', ',', 'the', 'west-side', 'temperate', 'rainforest', ',', 'and',
'the', 'forests', 'of', 'the', 'driereast', 'side', '.', 'Within', 'the', 'park', 'there', 'are', 'three', 'distinc
t', 'ecosystems', ',', 'including', 'subalpine', 'forest', 'and', 'wildflower', 'meadow', ',', 'temperate', 'forest',
',', 'and', 'the', 'rugged', 'Pacific', 'coast', '.']
```

(Part of Speech) PoS Tagging

- Part of speech tagging is an NLP technique that assigns a part of speech tag to each word in a sentence. The part of speech tag indicates the grammatical category of the word, such as noun, verb, adjective, adverb, etc.

PoS Tagging Example



```
In [5]: tokens = nltk.word_tokenize("The quick brown fox jumps over the lazy dog")
        tags = nltk.pos_tag(tokens)

        for token, tag in tags:
            print(token, tag)
```

```
The DT
quick JJ
brown NN
fox NN
jumps VBZ
over IN
the DT
lazy JJ
dog NN
```

Tag	Description
CC	conjunction, coordinating
NN	noun, common, singular, or mass
NNP	noun, plural
DT	determiner
JJ	adjective
VBZ	verb, present tense, 3rd person singular
IN	Preposition or subordinating conjunction

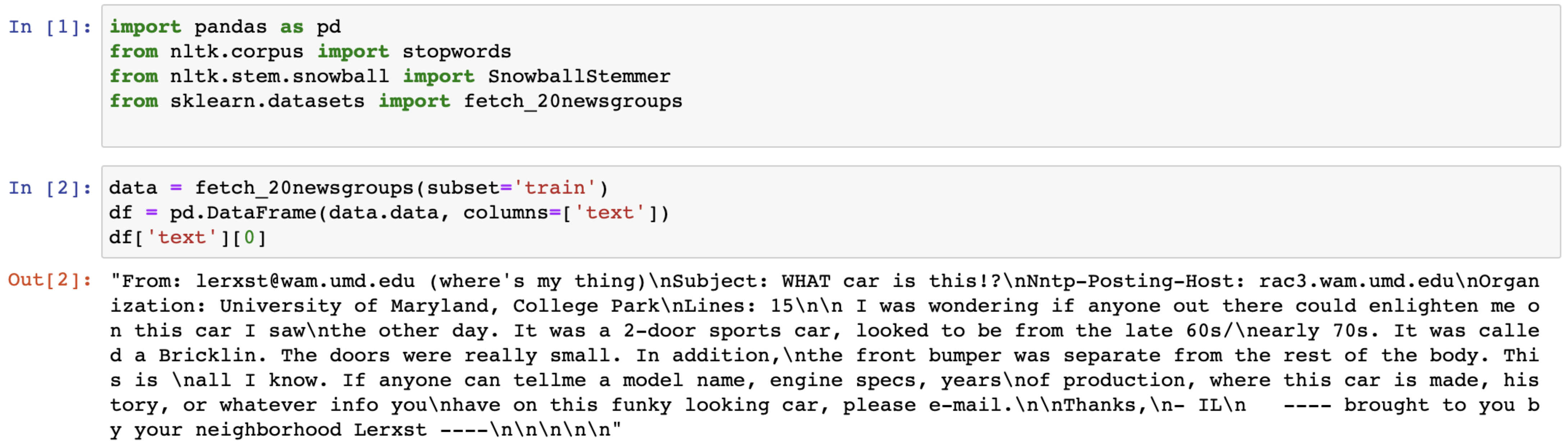
Text Preprocessing Steps

- Text cleaning or text preprocessing involves:
  - Punctuation removal
  - Elimination of stop words
  - Character case setting
  - Word stemming

Word Stemming

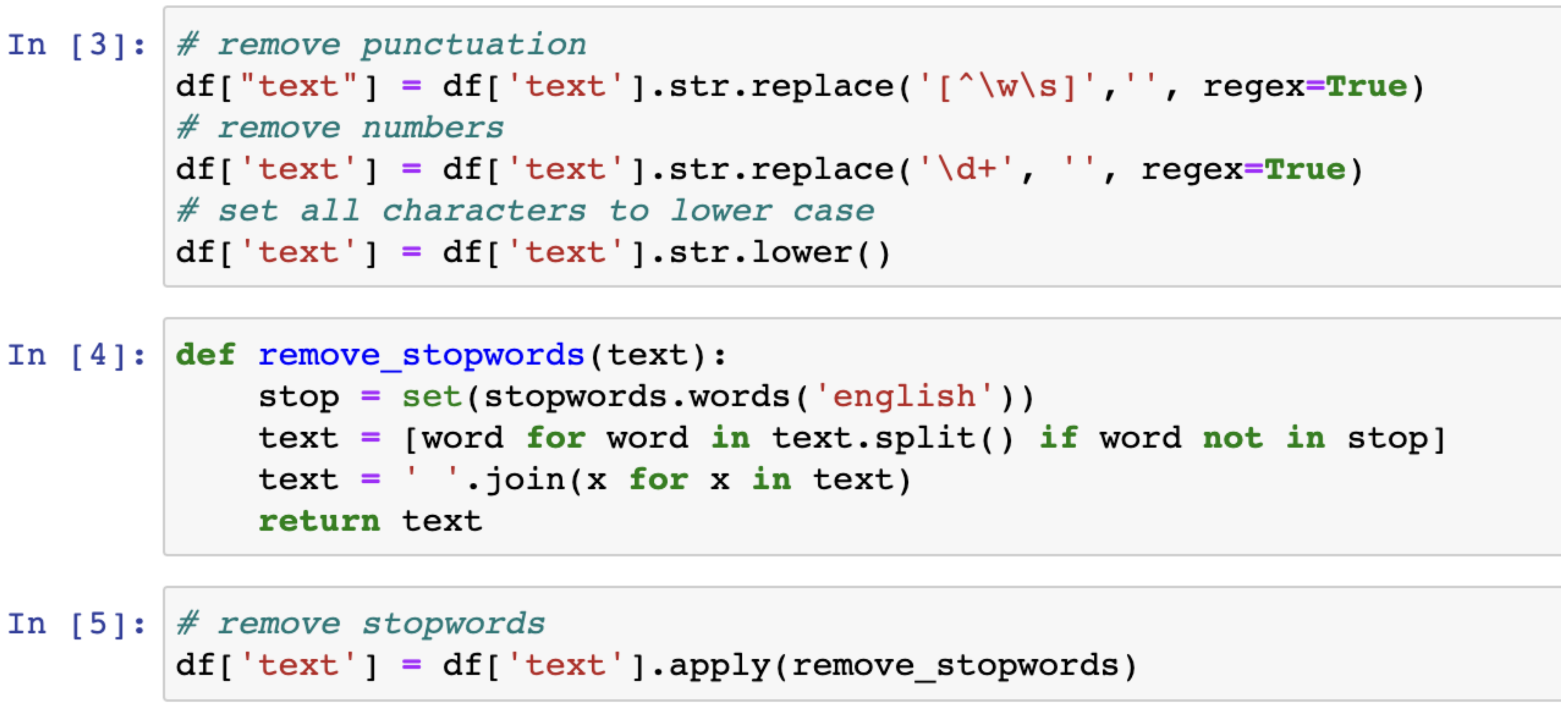
- Word stemming refers to reducing each word to its root or base so that the words playing, plays, and played become play, which, in essence, convey the same or very similar meaning.
    - had —> have
    - bananas —> banana
    - swimming —> swim

Text Preprocessing Example



```
In [1]: import pandas as pd
        from nltk.corpus import stopwords
        from nltk.stem.snowball import SnowballStemmer
        from sklearn.datasets import fetch_20newsgroups

In [2]: data = fetch_20newsgroups(subset='train')
        df = pd.DataFrame(data.data, columns=['text'])
        df['text'][0]

Out[2]: "From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car is this!?\nNntp-Posting-Host: rac3.wam.umd.edu\nOrgan
        ization: University of Maryland, College Park\nLines: 15\n\n I was wondering if anyone out there could enlighten me o
        n this car I saw\nthe other day. It was a 2-door sports car, looked to be from the late 60s/\nearly 70s. It was calle
        d a Bricklin. The doors were really small. In addition,\nthe front bumper was separate from the rest of the body. Thi
        s is \nall I know. If anyone can tellme a model name, engine specs, years\nof production, where this car is made, his
        tory, or whatever info you\nhave on this funky looking car, please e-mail.\n\nThanks,\n- IL\n   ---- brought to you b
        y your neighborhood Lerxst ----\n\n\n\n\n"
```

Text Preprocessing Example

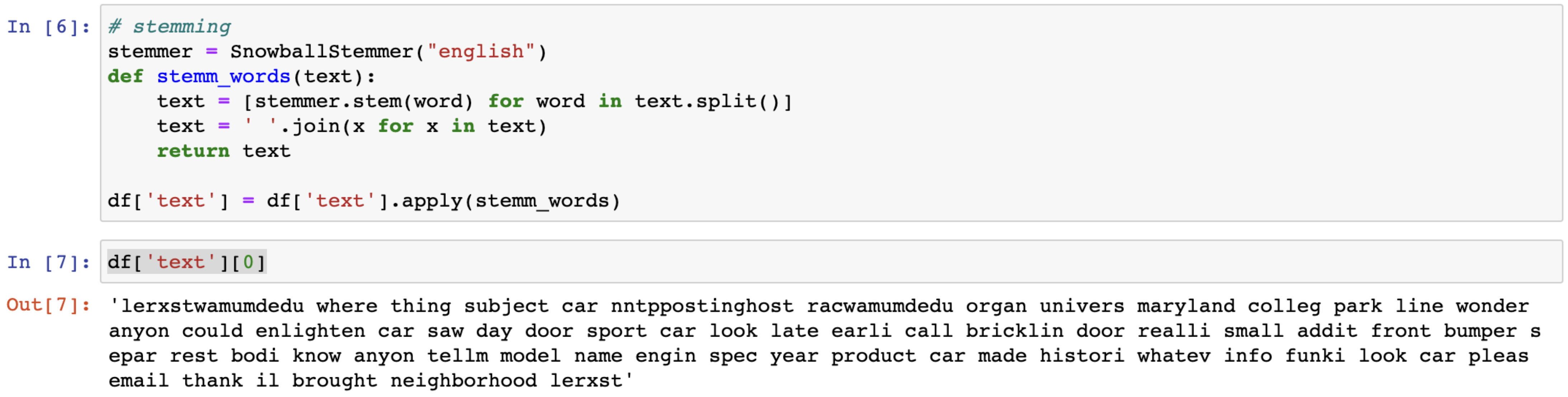


```
In [3]: # remove punctuation
        df["text"] = df['text'].str.replace('[^\w\s]','', regex=True)
        # remove numbers
        df['text'] = df['text'].str.replace('\d+', '', regex=True)
        # set all characters to lower case
        df['text'] = df['text'].str.lower()
```

```
In [4]: def remove_stopwords(text):
            stop = set(stopwords.words('english'))
            text = [word for word in text.split() if word not in stop]
            text = ' '.join(x for x in text)
            return text
```

```
In [5]: # remove stopwords
        df['text'] = df['text'].apply(remove_stopwords)
```

Text Preprocessing Example



```
In [6]: # stemming
        stemmer = SnowballStemmer("english")
        def stemm_words(text):
            text = [stemmer.stem(word) for word in text.split()]
            text = ' '.join(x for x in text)
            return text

        df['text'] = df['text'].apply(stemm_words)

In [7]: df['text'][0]

Out[7]: 'lerxstwamumdedu where thing subject car nntppostinghost racwamumdedu organ univers maryland colleg park line wonder
        anyon could enlighten car saw day door sport car look late earli call bricklin door realli small addit front bumper s
        epar rest bodi know anyon tellm model name engin spec year product car made histori whatev info funki look car pleas
        email thank il brought neighborhood lerxst'
```

Text Preprocessing

- The cleaning steps performed, resulted in strings containing the original text, without punctuation or numbers, in lowercase, without common words, and with the root of the word instead of its conjugated form.
- The data, as it is returned, can be used to derive features as described in the counting characters, words, and vocabulary or to create BoWs and TI-IDF matrices.

Term Frequency-Inverse Document Frequency (TF-IDF)

- TF-IDF is a numerical statistic that captures how relevant a word is in a document, with respect to the entire collection of documents.
- Some words will appear a lot within a text document as well as across documents, for example, the English words the, a, and is. These words generally convey little information about the actual content of the document and don't make it stand out of the crowd.
- TF-IDF provides a way to weigh the importance of a word, by examining how many times it appears in a document, with respect to how often it appears across documents.
- Hence, commonly occurring words such as the, a, and is will have a low weight, and words more specific to a topic, such as leopard, will have a higher weight.

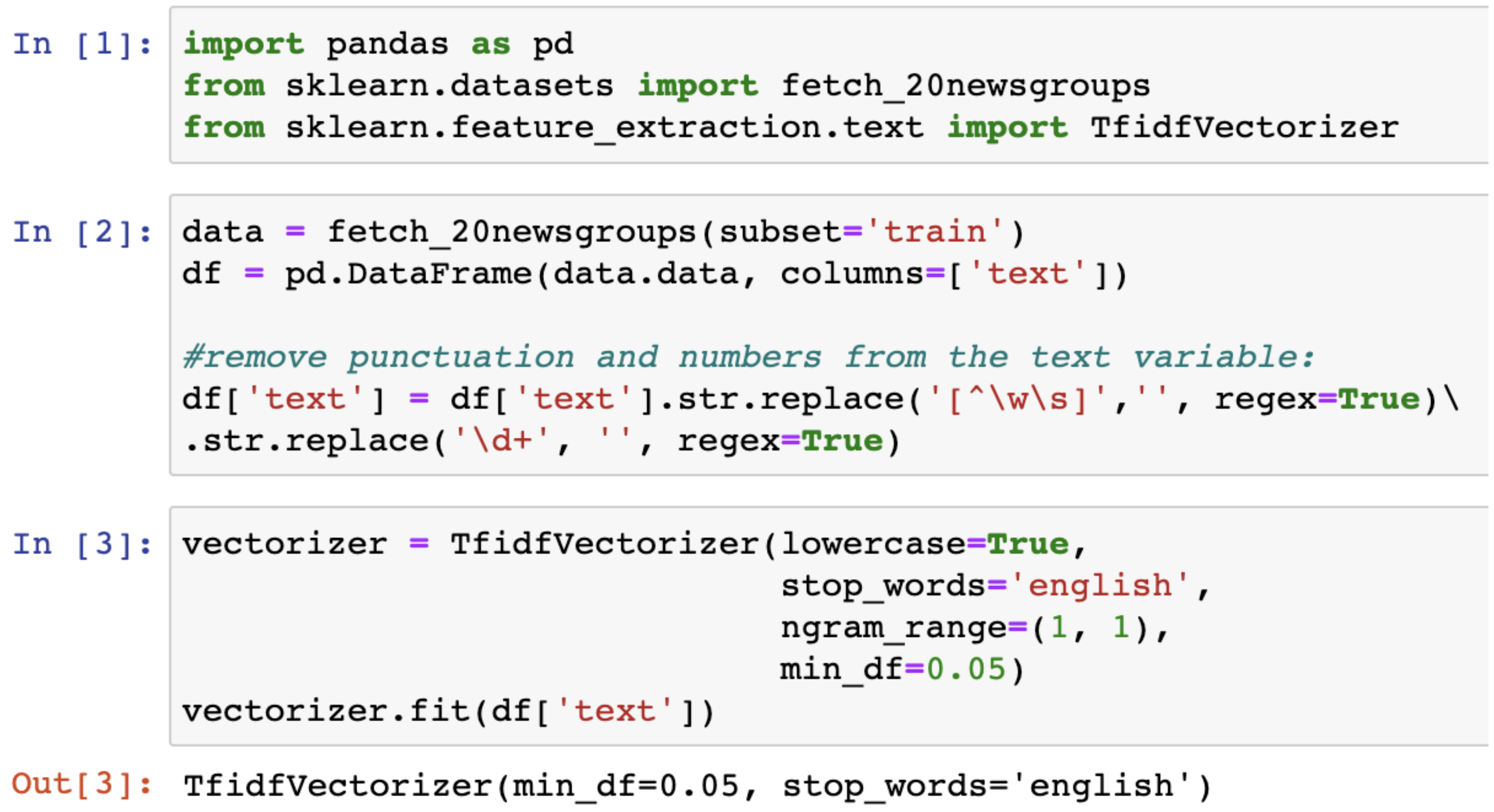
TF-IDF

- TF-IDF = term frequency * inverse document frequency
- Term frequency is, the count of the word in an individual text. So, for term t, the term frequency is calculated as tf(t) = count(t) and is determined text by text.
- The inverse document frequency is a measure of how common the word is across all documents and is usually calculated on a logarithmic scale.

$$idf(t) = log(\frac{n}{1 + df(t)})$$

- Here, n is the total number of documents and df(t) the number of documents in which the term t appears.

TF-IDF Example

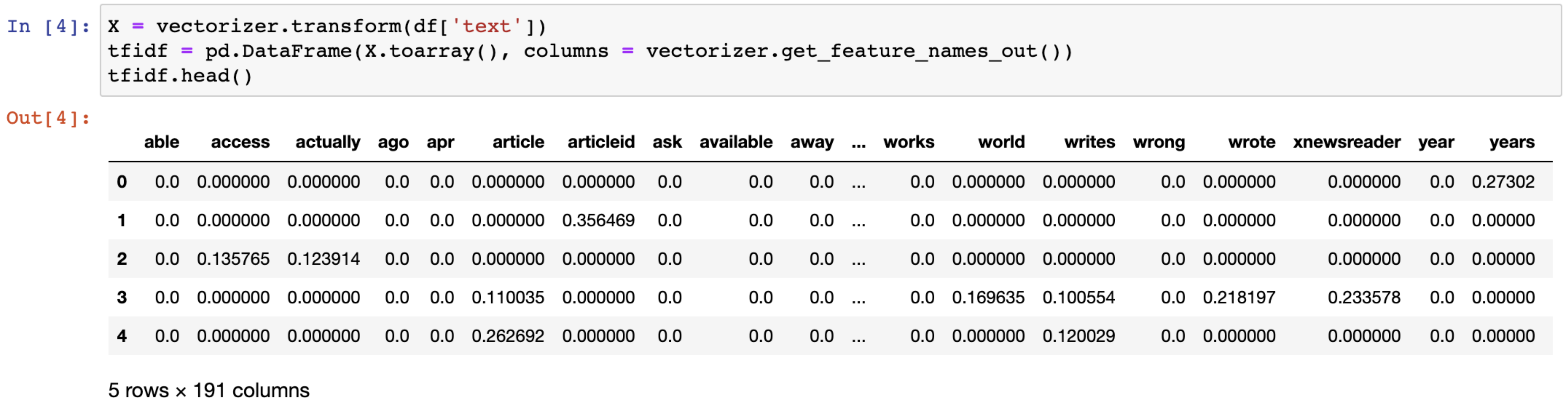


```
In [1]: import pandas as pd
        from sklearn.datasets import fetch_20newsgroups
        from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [2]: data = fetch_20newsgroups(subset='train')
        df = pd.DataFrame(data.data, columns=['text'])

        #remove punctuation and numbers from the text variable:
        df['text'] = df['text'].str.replace('[^\w\s]','', regex=True)\
        .str.replace('\d+', '', regex=True)
```

```
In [3]: vectorizer = TfidfVectorizer(lowercase=True,
                                     stop_words='english',
                                     ngram_range=(1, 1),
                                     min_df=0.05)
        vectorizer.fit(df['text'])

Out[3]: TfidfVectorizer(min_df=0.05, stop_words='english')
```

TF-IDF Example



In [4]:
```python
X = vectorizer.transform(df['text'])
tfidf = pd.DataFrame(X.toarray(), columns = vectorizer.get_feature_names_out())
tfidf.head()
```

Out[4]:

	able	access	actually	ago	apr	article	articleid	ask	available	away	...	works	world	writes	wrong	wrote	xnewsreader	year	years
0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.27302
1	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.356469	0.0	0.0	0.0	...	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.00000
2	0.0	0.135765	0.123914	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.00000
3	0.0	0.000000	0.000000	0.0	0.0	0.110035	0.000000	0.0	0.0	0.0	...	0.0	0.169635	0.100554	0.0	0.218197	0.233578	0.0	0.00000
4	0.0	0.000000	0.000000	0.0	0.0	0.262692	0.000000	0.0	0.0	0.0	...	0.0	0.000000	0.120029	0.0	0.000000	0.000000	0.0	0.00000

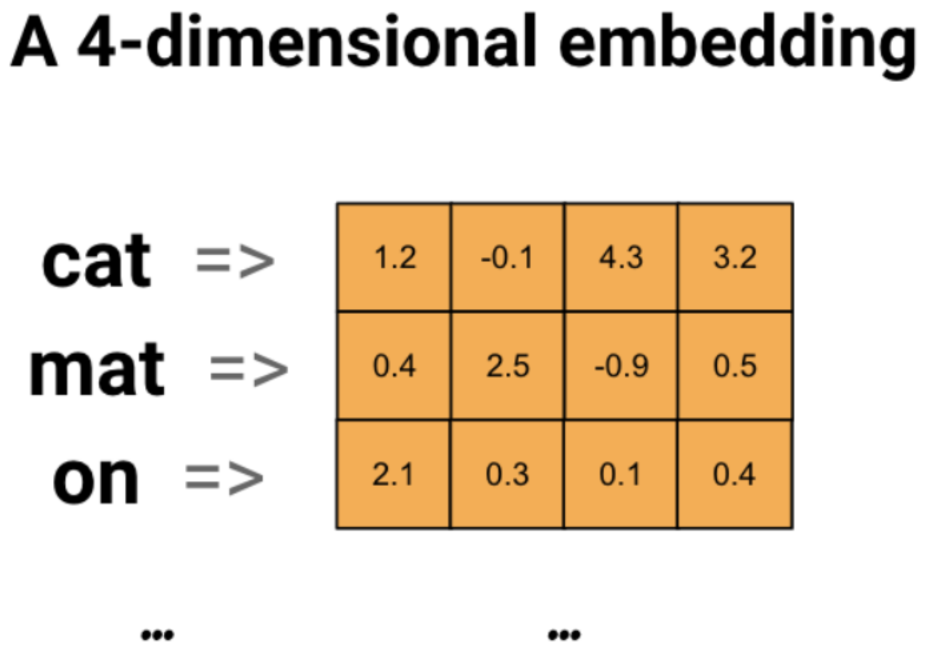
5 rows × 191 columns

Word Embedding

- Word embeddings give us a way to use an efficient, dense representation in which similar words have a similar encoding.
- Instead of specifying the values for the embedding manually, they are trainable parameters (weights learned by the model during training, in the same way a model learns weights for a dense layer).
- It is common to see word embeddings that are 8-dimensional (for small datasets), up to 1024-dimensions when working with large datasets.

Word Embedding

- Each word is represented as a 4-dimensional vector of floating point values.
- Another way to think of an embedding is as "lookup table".
- After these weights have been learned, you can encode each word by looking up the dense vector it corresponds to in the table.



A 4-dimensional embedding

cat	=>	1.2	-0.1	4.3	3.2
mat	=>	0.4	2.5	-0.9	0.5
on	=>	2.1	0.3	0.1	0.4
...		...			

References

Illowsky, Barbara; Dean, Susan. Introductory Statistics. XanEdu Publishing Inc.

Bruce, Peter; Bruce, Andrew; Gedeck, Peter. Practical Statistics for Data Scientists . O'Reilly Media.

Müller, Andreas C.; Guido, Sarah. Introduction to Machine Learning with Python. O'Reilly Media. Kindle Edition.