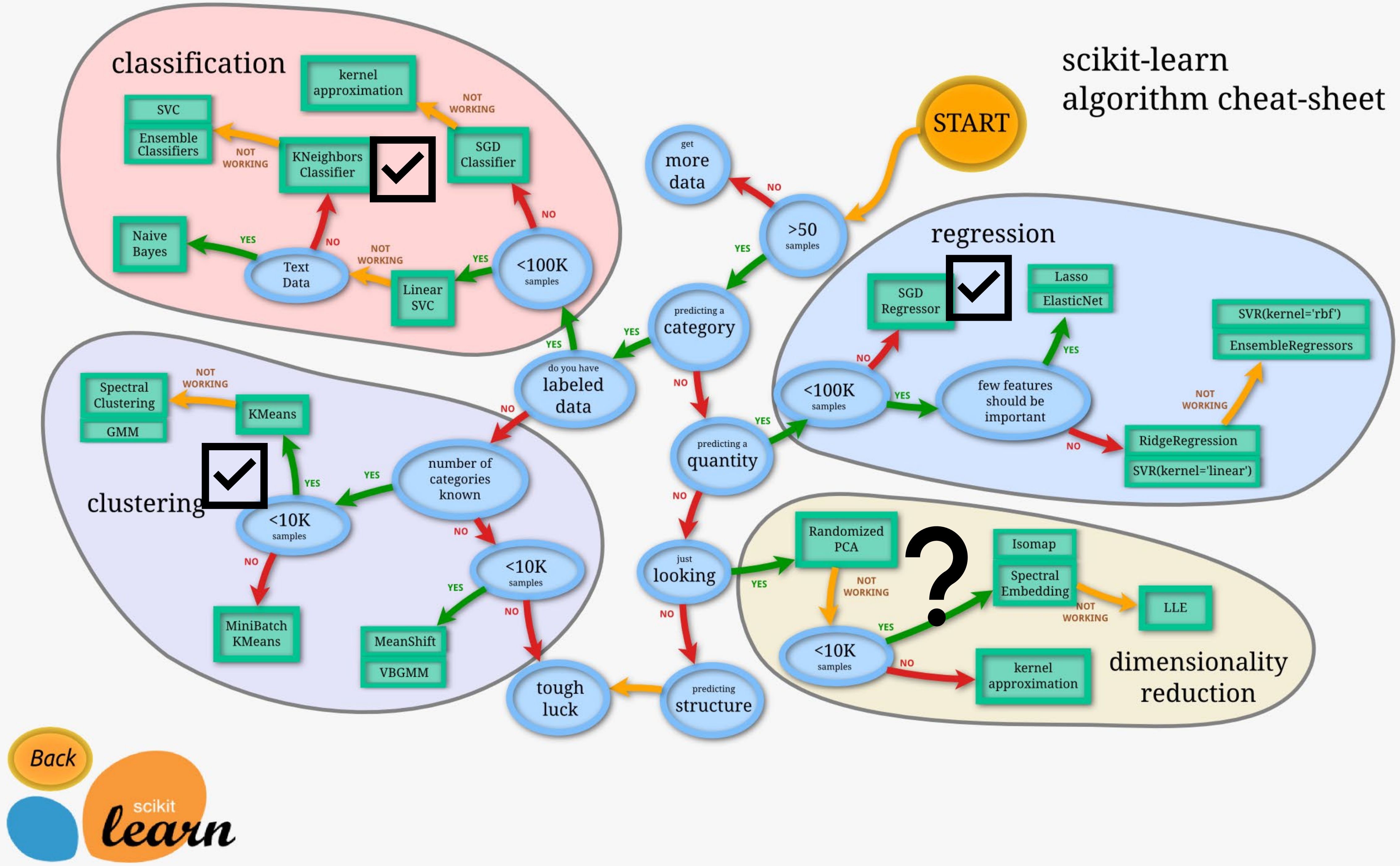
EM 538-001: PRACTICAL MACHINE LEARNING FOR ENGINEERING ANALYTICS

LECTURE 010

Fred Livingston, Ph.D.

scikit-learn algorithm cheat-sheet



DIMENSIONALITY REDUCTION

- Feature’s Variance
- Covariance Matrix
- Principal Component Analysis (PCA)
- PCA + Kmean Example
- Homework 2 (Assigned Next Class)

DIMENSIONALITY REDUCTION

Why do we care?

DIMENSIONALITY REDUCTION

Why do we care?

Curse of dimensionality

Computational efficiency

Easier data collection

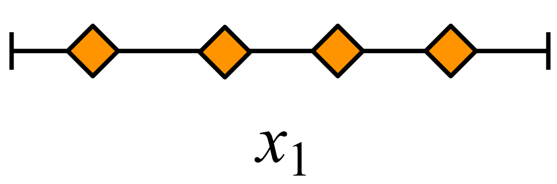
Storage space

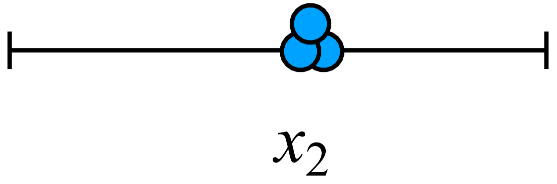
Interpretability

FEATURE VARIANCE

- Compute the variance of each feature

- Assume that features with a higher variance may contain more useful information

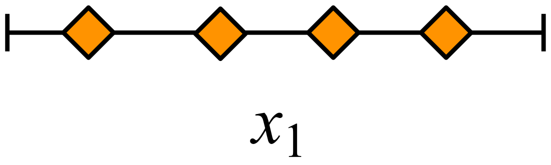


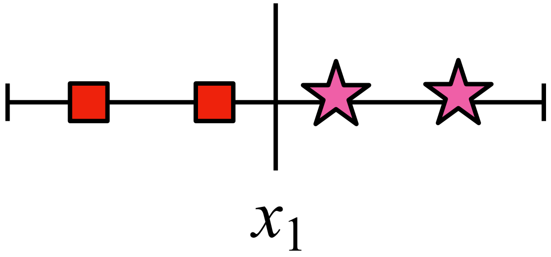


FEATURE VARIANCE

- Compute the variance of each feature

- Assume that features with a higher variance may contain more useful information





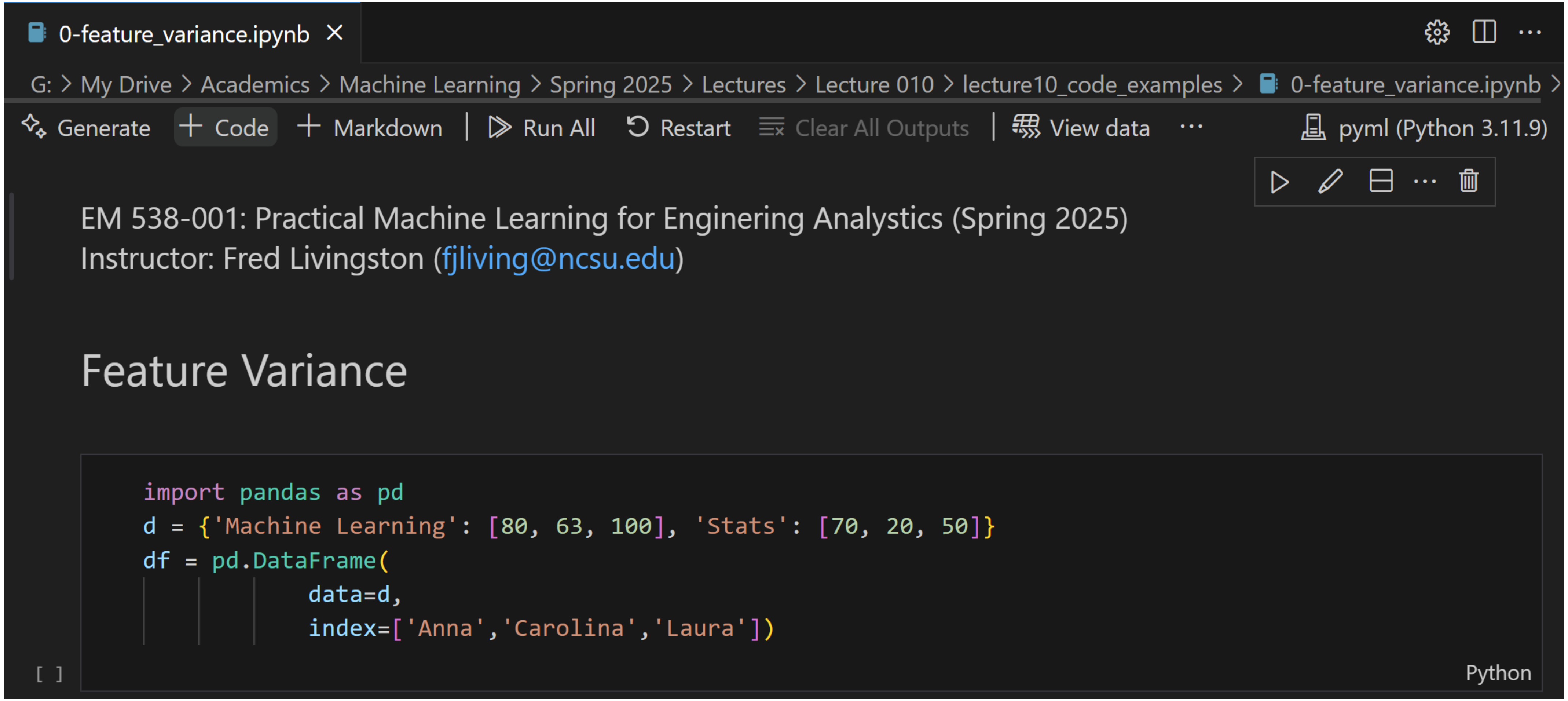
SAMPLE VARIANCE

E.g., dataset with n datapoints (for sample variance, *n-1*)

$$\boxed{\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Student	Machine Learning (X1)	Statics (X2)
Anna	80	70
Caroline	63	20
Laura	100	50

SAMPLE VARIANCE



0-feature_variance.ipynb
G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 010 > lecture10_code_examples > 0-feature_variance.ipynb >
Generate
+ Code
+ Markdown
Run All
Restart
Clear All Outputs
View data
pyml (Python 3.11.9)
[ ]
Python

EM 538-001: Practical Machine Learning for Enginering Analystics (Spring 2025)
Instructor: Fred Livingston (fjliving@ncsu.edu)

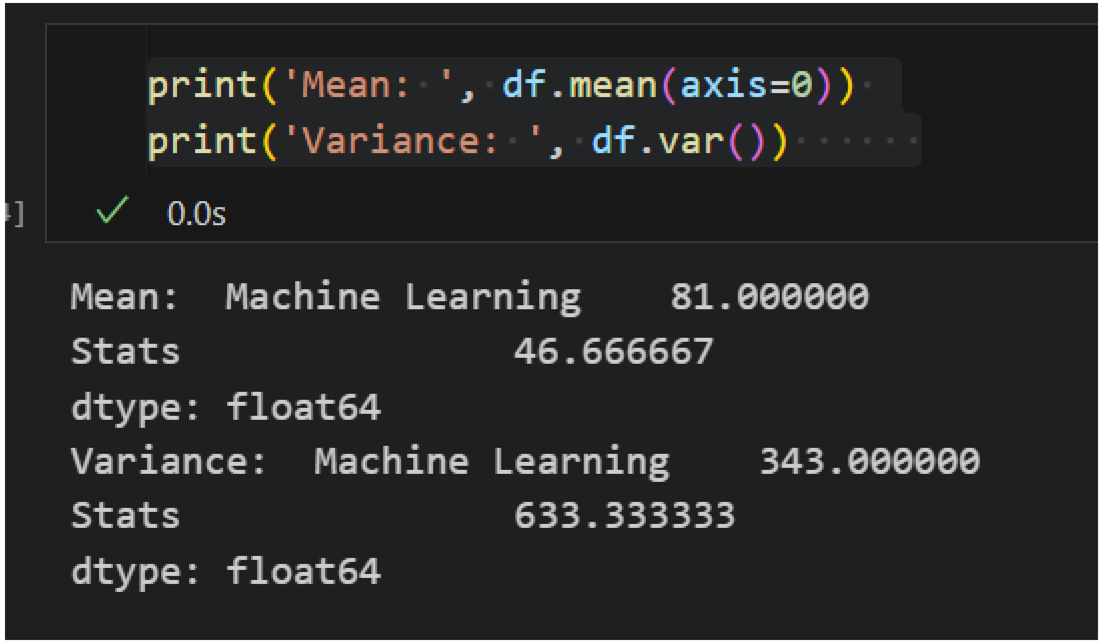
Feature Variance

```python
import pandas as pd
d = {'Machine Learning': [80, 63, 100], 'Stats': [70, 20, 50]}
df = pd.DataFrame(
            data=d,
            index=['Anna','Carolina','Laura'])
```

BE AWARE OF FEATURE SCALING!

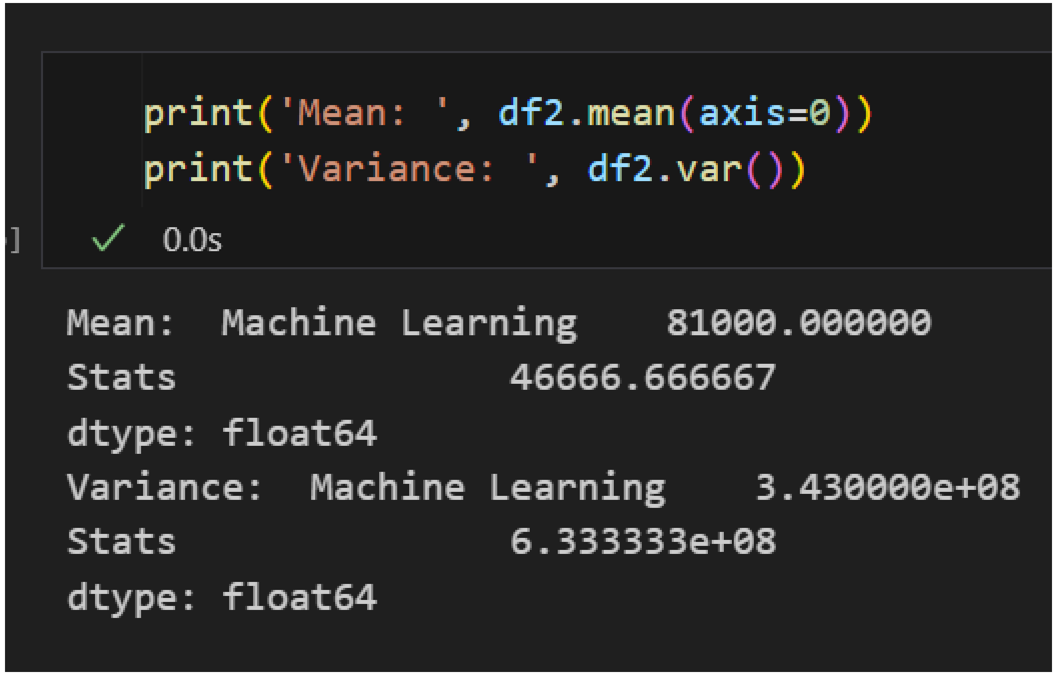
E.g., dataset with n datapoints (for sample variance, *n-1*)

$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2$$



```
print('Mean: ', df.mean(axis=0))
print('Variance: ', df.var())
```

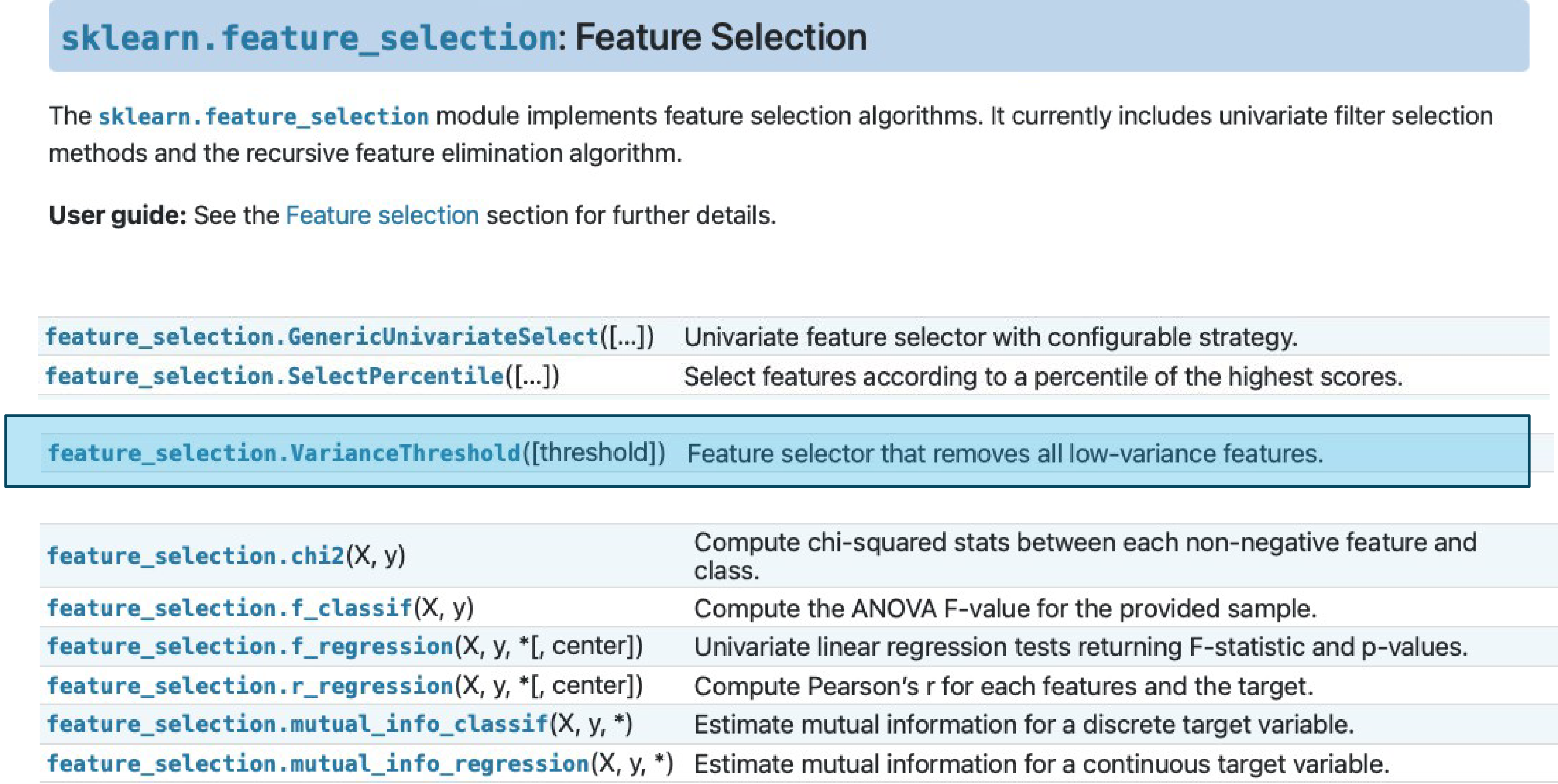
```
Mean:  Machine Learning    81.000000
Stats                      46.666667
dtype: float64
Variance:  Machine Learning    343.000000
Stats                          633.333333
dtype: float64
```



```
print('Mean: ', df2.mean(axis=0))
print('Variance: ', df2.var())
```

```
Mean:  Machine Learning    81000.000000
Stats                      46666.666667
dtype: float64
Variance:  Machine Learning    3.430000e+08
Stats                          6.333333e+08
dtype: float64
```

FEATURE SELCTION USING VARIANCE THRESOLD



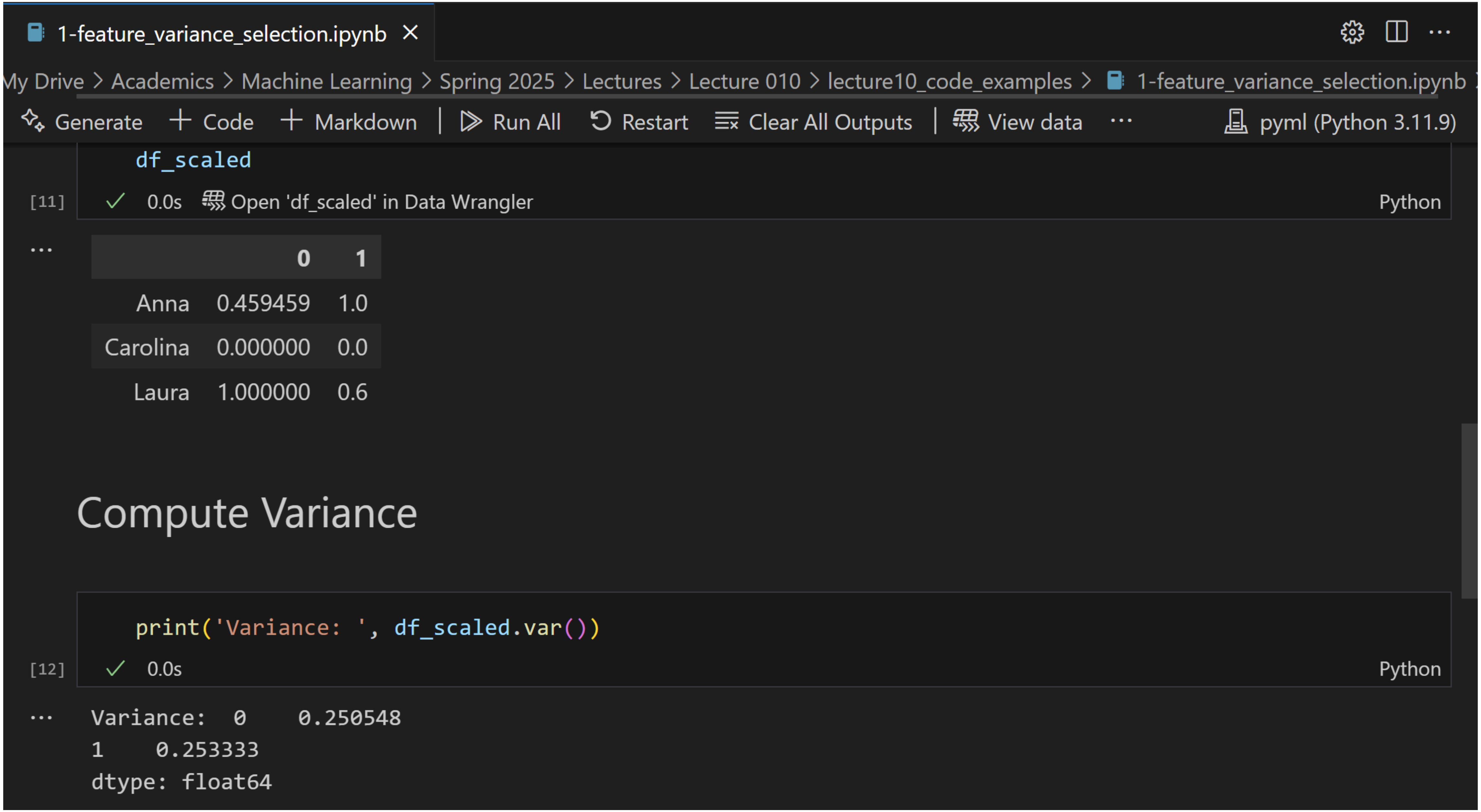
sklearn.feature_selection: Feature Selection

The sklearn.feature_selection module implements feature selection algorithms. It currently includes univariate filter selection methods and the recursive feature elimination algorithm.

User guide: See the Feature selection section for further details.

feature_selection.GenericUnivariateSelect([...])	Univariate feature selector with configurable strategy.
feature_selection.SelectPercentile([...])	Select features according to a percentile of the highest scores.
feature_selection.VarianceThreshold([threshold])	Feature selector that removes all low-variance features.
feature_selection.chi2(X, y)	Compute chi-squared stats between each non-negative feature and class.
feature_selection.f_classif(X, y)	Compute the ANOVA F-value for the provided sample.
feature_selection.f_regression(X, y, *[, center])	Univariate linear regression tests returning F-statistic and p-values.
feature_selection.r_regression(X, y, *[, center])	Compute Pearson's r for each features and the target.
feature_selection.mutual_info_classif(X, y, *)	Estimate mutual information for a discrete target variable.
feature_selection.mutual_info_regression(X, y, *)	Estimate mutual information for a continuous target variable.

https://scikit-learn.org/stable/modules/classes.html?highlight=feature%20selection#module-sklearn.feature_selection


1-feature_variance_selection.ipynb
My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 010 > lecture10_code_examples > 1-feature_variance_selection.ipynb
Generate
+ Code
+ Markdown
Run All
Restart
Clear All Outputs
View data
pyml (Python 3.11.9)
[11]
✓ 0.0s
Open 'df_scaled' in Data Wrangler
Python
[12]
✓ 0.0s
Python

```
df_scaled
```

	0	1
Anna	0.459459	1.0
Carolina	0.000000	0.0
Laura	1.000000	0.6

Compute Variance

```
print('Variance: ', df_scaled.var())
```

```
Variance:  0    0.250548
1    0.253333
dtype: float64
```

VARIANCE THRESHOLD

- Compute the variance of each feature

- Assume that features with a higher variance may contain more useful information

- Select the subset of features based on a user-specified threshold ("keep if greater or equal to *x*” or “keep the the top *k* features with largest variance”)

- **Good:** fast!

- **Bad:** does not take the relationship among features into account

COVARIANCE MATRIX

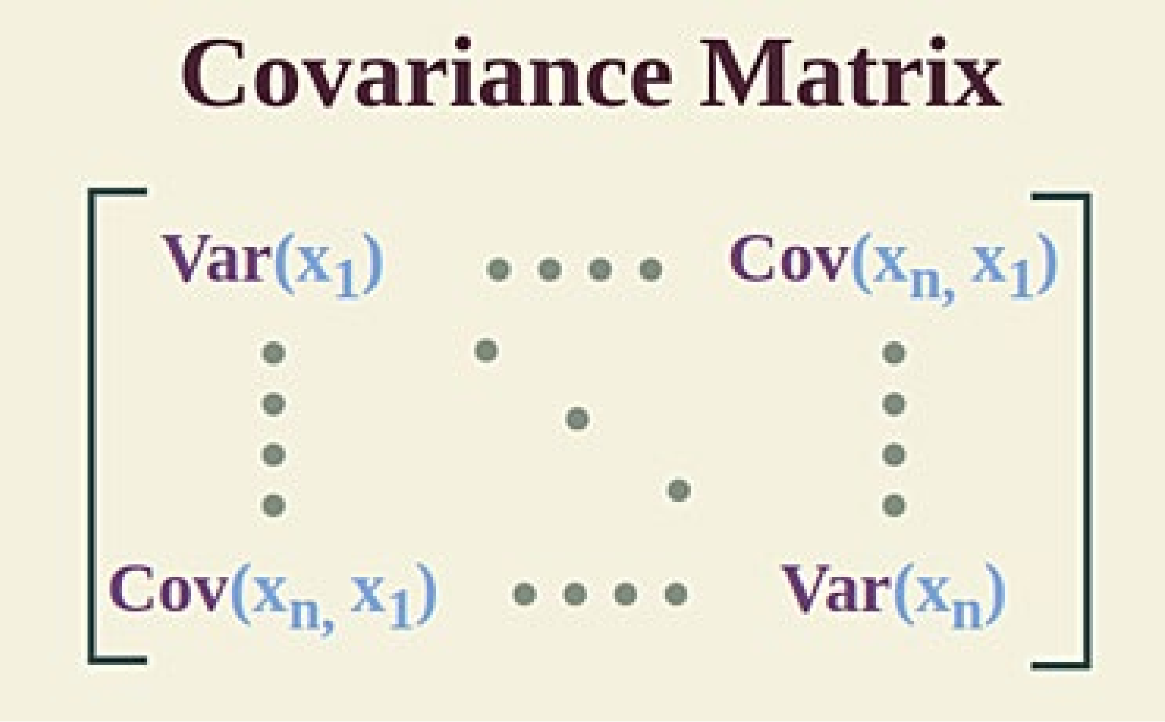
COVARIANCE MATRIX

Covariance Matrix is a type of matrix used to describe the covariance values between two items in a random vector. It is also known as the variance-covariance matrix because the variance of each element is represented along the matrix's major diagonal and the covariance is represented among the non-diagonal elements.

It's particularly important in fields like data science, machine learning, and finance, where understanding relationships between multiple variables is crucial and comes in handy when it comes to stochastic modeling and principal component analysis.

WHAT IS COVARIANCE MATRIX

The variance-covariance matrix is a square matrix with diagonal elements that represent the variance and the non-diagonal components that express covariance. The covariance of a variable can take any real value-positive, negative, or zero. A positive covariance suggests that the two variables have a positive relationship, whereas a negative covariance indicates that they do not. If two elements do not vary together, they have a zero covariance.



$$\begin{bmatrix} \mathrm{Var}(x_1) & \cdots\cdots & \mathrm{Cov}(x_{n,} x_1) \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}(x_{n,} x_1) & \cdots\cdots & \mathrm{Var}(x_n) \end{bmatrix}$$

HOW TO FIND COVARIANCE MATRIX

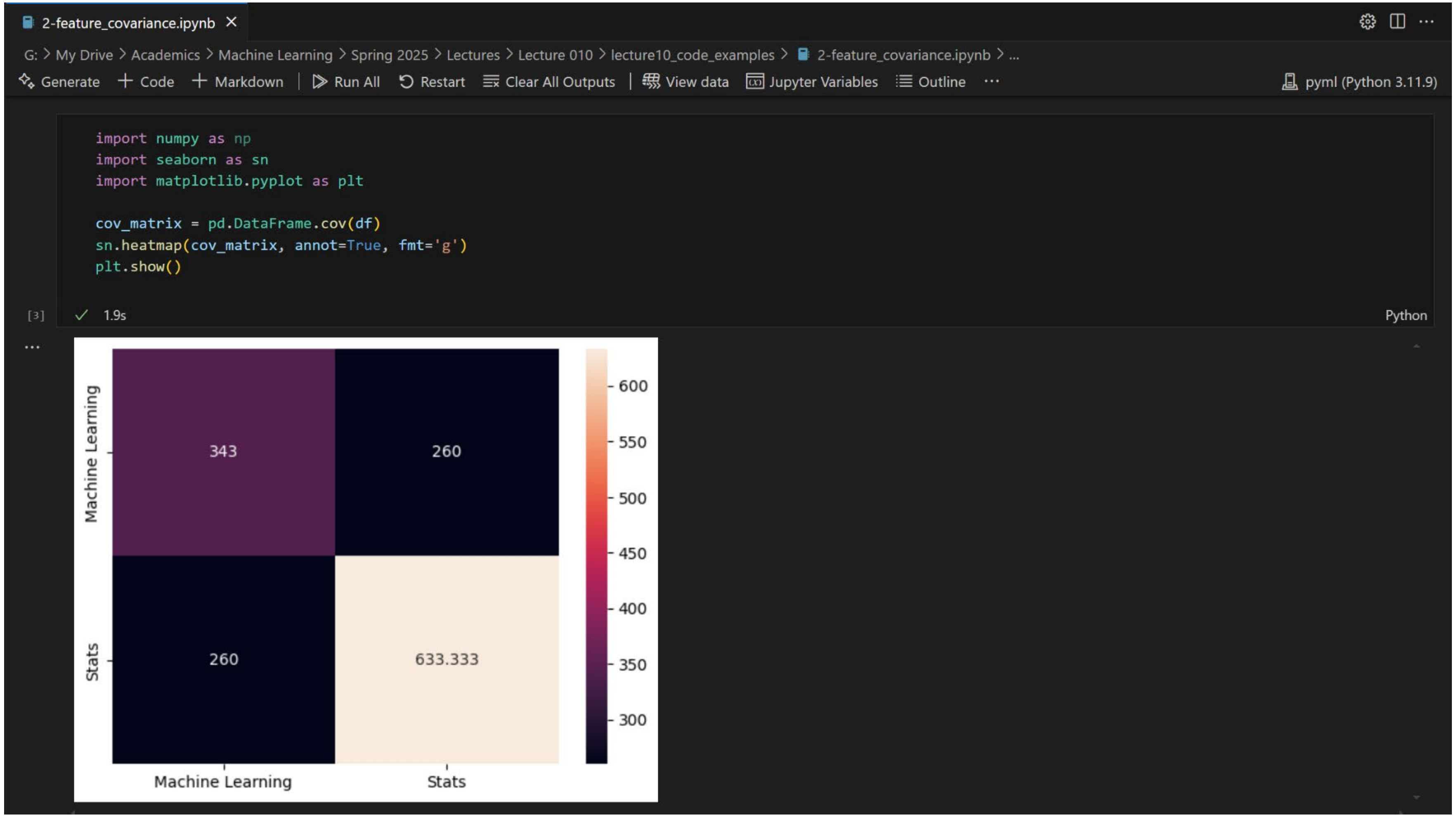
The dimensions of a covariance matrix are determined by the number of variables in a given data set. If there are only two variables in a set, then the covariance matrix would have two rows and two columns. Similarly, if a data set has three variables, then its covariance matrix would have three rows and three columns.

Student	Machine Learning (X1)	Statics (X2)
Anna	80	70
Caroline	63	20
Laura	100	50

COVARIANCE MATRIX WORKED EXAMPLE

Student	Machine Learning (X1)	Statics (X2)
Anna	80	70
Caroline	63	20
Laura	100	50

COVARIANCE MATRIX IN PYTHON



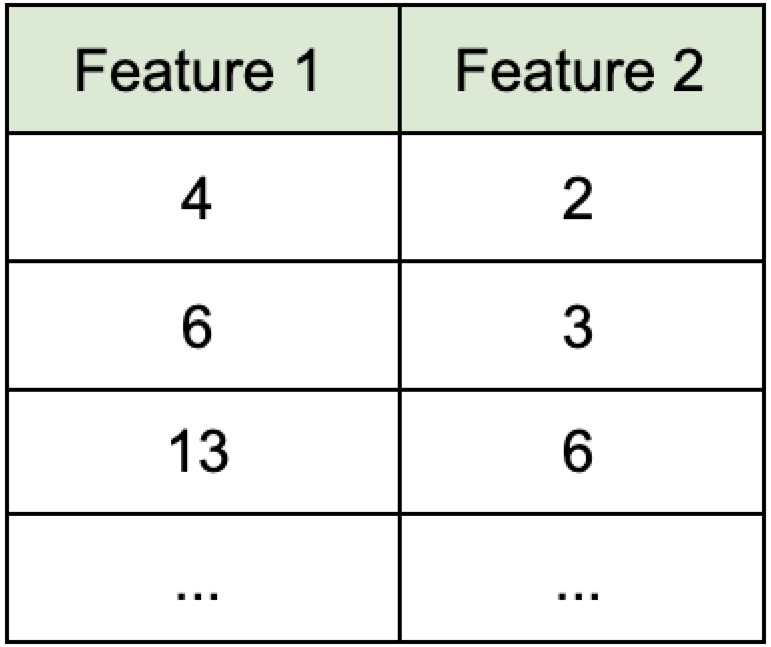
```python
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

cov_matrix = pd.DataFrame.cov(df)
sn.heatmap(cov_matrix, annot=True, fmt='g')
plt.show()
```

PRINCIPAL COMPONENT ANALYSIS (PCA)

PRINCIPAL COMPONENT ANALYSIS (PCA)

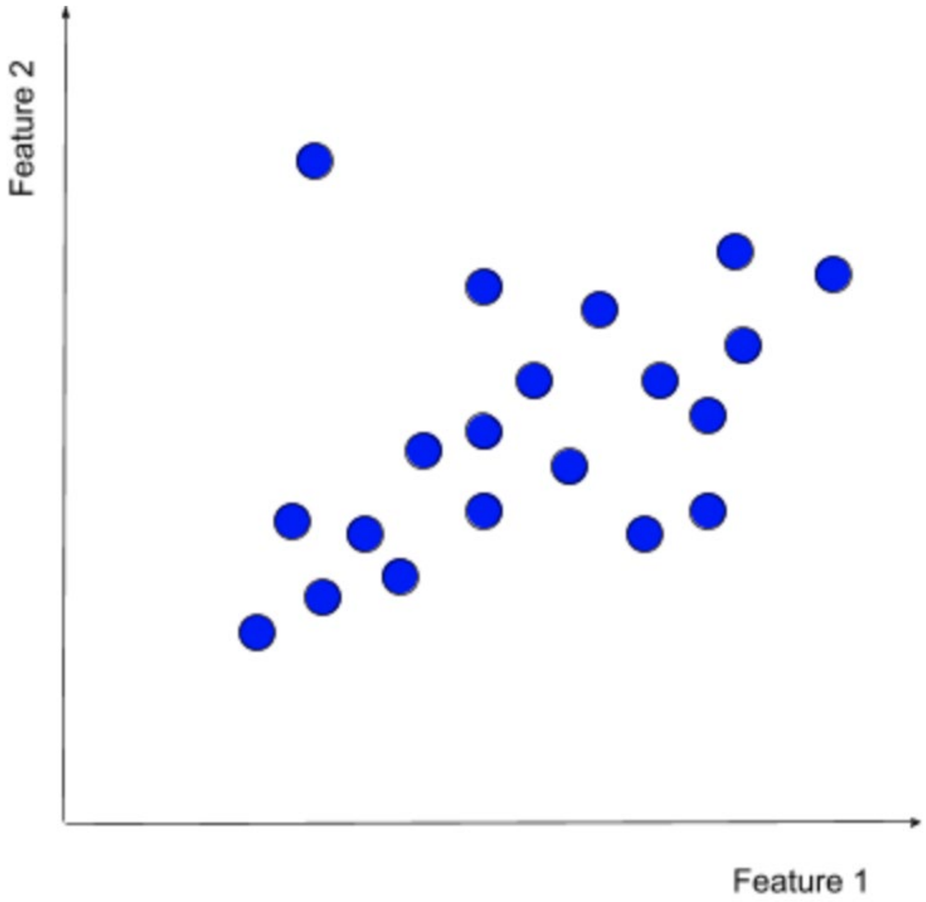
Principal Component Analysis (PCA) is one of the most commonly used unsupervised machine learning algorithms across a variety of applications: exploratory data analysis, dimensionality reduction, information compression, data de-noising, and plenty more.



Feature 1	Feature 2
4	2
6	3
13	6
...	...

PRINCIPAL COMPONENT ANALYSIS (PCA)

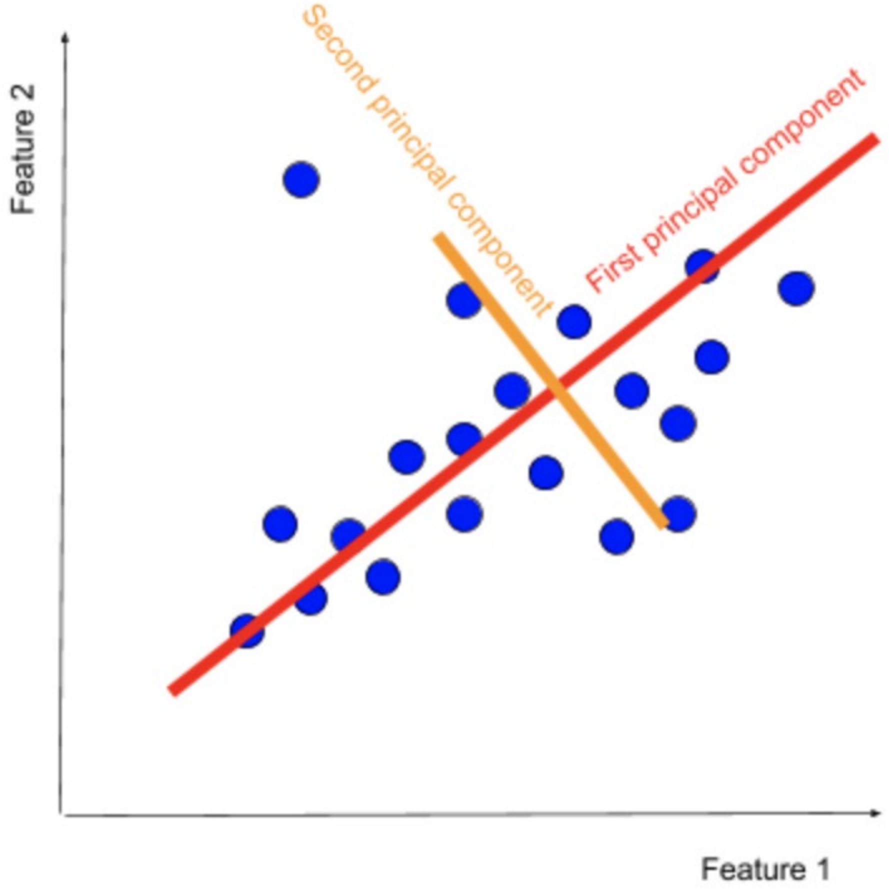
The main aim of PCA is to find such principal components, which can describe the data points with a set of principal components. The principal components are **vectors**, but they are not chosen at random. The **first principal component** is computed so that it explains the **greatest amount of variance** in the original features. The **second component** is **orthogonal** to the first, and it explains the greatest amount of variance left after the first principal component.



PRINCIPAL COMPONENT ANALYSIS (PCA)

The original data can be represented as feature vectors. PCA allows us to go a step further and represent the data as linear combinations of principal components. Getting principal components is equivalent to a linear transformation of data from the feature1 x feature2 axis to a PCA1 x PCA2 axis.

in very **large datasets** (where the number of dimensions can surpass 100 different variables), principal components remove noise by **reducing a large number of features** to just a couple of principal components. Principal components are orthogonal projections of data onto lower-dimensional space.



WHAT IS PCA USED FOR?

The algorithm can be used on its own, or it can serve as a data cleaning or data preprocessing technique used before another machine learning algorithm.

On its own, PCA is used across a variety of use cases:

1. **Visualize multidimensional data.** Data visualizations are a great tool for communicating multidimensional data as 2- or 3-dimensional plots.
2. **Compress information.** Principal Component Analysis is used to compress information to store and transmit data more efficiently. For example, it can be used to compress images without losing too much quality, or in signal processing. The technique has successfully been applied across a wide range of compression problems in pattern recognition (specifically face recognition), image recognition, and more.
3. **Simplify complex business decisions.** PCA has been employed to simplify traditionally complex business decisions. For example, traders use over 300 financial instruments to manage portfolios. The algorithm has proven successful in the risk management of interest rate derivative portfolios, lowering the number of financial instruments from more than 300 to just 3-4 principal components.
4. **Clarify convoluted scientific processes.** The algorithm has been applied extensively in the understanding of convoluted and multidirectional factors, which increase the probability of neural ensembles to trigger action potentials.

PCA FOR PREPROCESSING

When PCA is used as part of preprocessing, the algorithm is applied to:

1. Reduce the number of dimensions in the training dataset.
2. **De-noise the data.** Because PCA is computed by finding the components which explain the greatest amount of variance, it captures the signal in the data and omits the noise.

HOW IS PCA CALCULATED?

There are multiple ways to calculate PCA:

1. Eigendecomposition of the covariance matrix
2. Singular value decomposition of the data matrix
3. Eigenvalue approximation via power iterative computation
4. Non-linear iterative partial least squares (NIPALS) computation
5. … and more.

EIGENDECOMPOSITION OF COVARIANCE MATRIX

1. **Feature standardization.** We standardize each feature to have a mean of 0 and a variance of 1. As we explain later in assumptions and limitations, features with values that are on different orders of magnitude prevent PCA from computing the best principal components.

2. **Obtain the covariance matrix computation.** The covariance matrix is a square matrix, of d x d dimensions, where d stands for “dimension” (or feature or column, if our data is tabular). It shows the pairwise feature correlation between each feature.

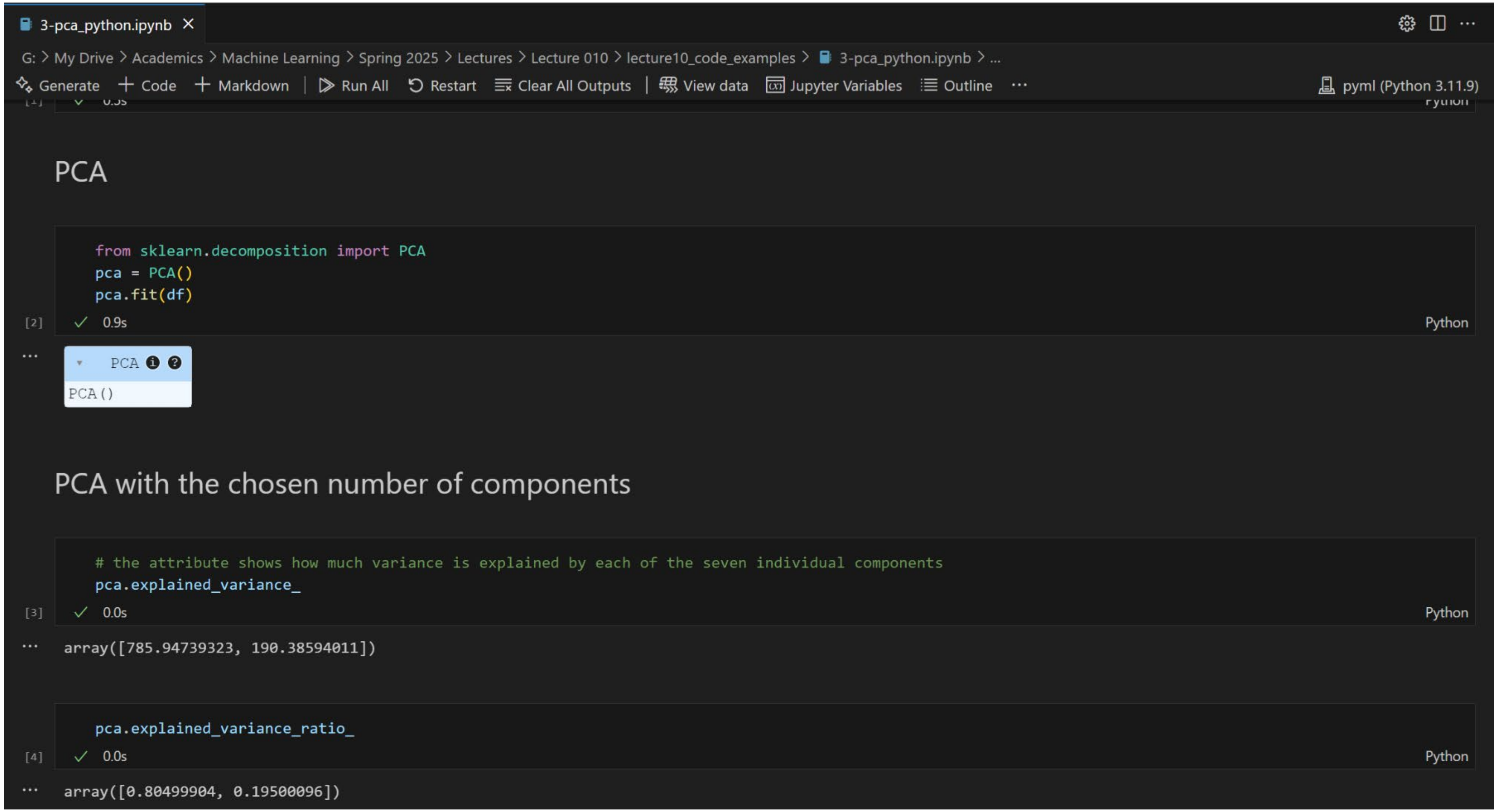
EIGENDECOMPOSITION OF COVARIANCE MATRIX

3. **Calculate the eigendecomposition of the covariance matrix.** We calculate the eigenvectors (unit vectors) and their associated eigenvalues (scalars by which we multiply the eigenvector) of the covariance matrix.
4. **Sort the eigenvectors from the highest eigenvalue to the lowest.** The eigenvector with the highest eigenvalue is the first principal component. Higher eigenvalues correspond to greater amounts of shared variance explained.

EIGENDECOMPOSITION OF COVARIANCE MATRIX

5. **Select the number of principal components.** Select the top N eigenvectors (based on their eigenvalues) to become the N principal components. The optimal number of principal components is both subjective and problem-dependent. Usually, we look at the cumulative amount of shared variance explained by the combination of principal components and pick that number of components, which still significantly explains the shared variance.

PCA WORKED EXAMPLE



PCA

```python
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(df)
```

```
PCA
PCA()
```

PCA with the chosen number of components

```python
# the attribute shows how much variance is explained by each of the seven individual components
pca.explained_variance_
```

```
array([785.94739323, 190.38594011])
```

```python
pca.explained_variance_ratio_
```

```
array([0.80499904, 0.19500096])
```

ADVANTAGES OF PCA

1. **Easy to compute.** PCA is based on linear algebra, which is computationally easy to solve by computers.
2. **Speeds up other machine learning algorithms.** Machine learning algorithms converge faster when trained on principal components instead of the original dataset.
3. **Counteracts the issues of high-dimensional data**. High-dimensional data causes regression-based algorithms to overfit easily. By using PCA beforehand to lower the dimensions of the training dataset, we prevent the predictive algorithms from overfitting.

DISADVANTAGES OF PCA

1. **Low interpretability of principal components.** Principal components are linear combinations of the features from the original data, but they are not as easy to interpret. For example, it is difficult to tell which are the most important features in the dataset after computing principal components.

2. **The trade-off between information loss and dimensionality reduction.** Although dimensionality reduction is useful, it comes at a cost. Information loss is a necessary part of PCA. Balancing the trade-off between dimensionality reduction and information loss is unfortunately a necessary compromise that we have to make when using PCA.

ASSUMPTIONS AND LIMITATIONS OF PCA

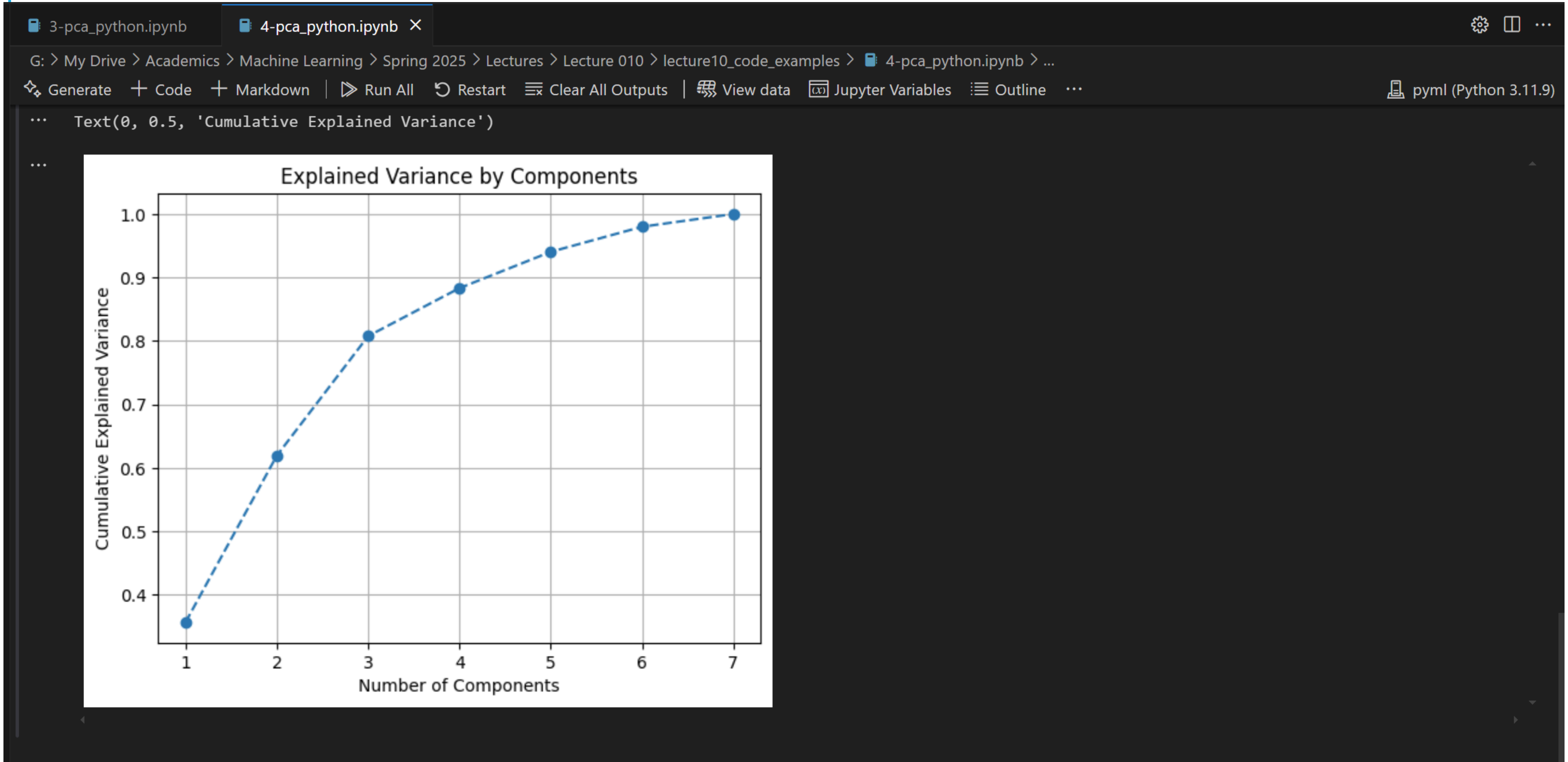
1. **PCA assumes a correlation between features.** If the features (or dimensions or columns, in tabular data) are not correlated, PCA will be unable to determine principal components.
2. **PCA is sensitive to the scale of the features.** Imagine we have two features - one takes values between 0 and 1000, while the other takes values between 0 and 1. PCA will be extremely biased towards the first feature being the first principle component, regardless of the actual maximum variance within the data. This is why it's so important to standardize the values first.
3. **PCA is not robust against outliers.** Similar to the point above, the algorithm will be biased in datasets with strong outliers. This is why it is recommended to remove outliers before performing PCA.

ASSUMPTIONS AND LIMITATIONS OF PCA

1. **PCA assumes a linear relationship between features.** The algorithm is not well suited to capturing non-linear relationships. That’s why it’s advised to turn non-linear features or relationships between features into linear, using the standard methods such as log transforms.

2. **Technical implementations often assume no missing values.** When computing PCA using statistical software tools, they often assume that the feature set has no missing values (no empty rows). Be sure to remove those rows and/or columns with missing values, or impute missing values with a close approximation (e.g. the mean of the column).

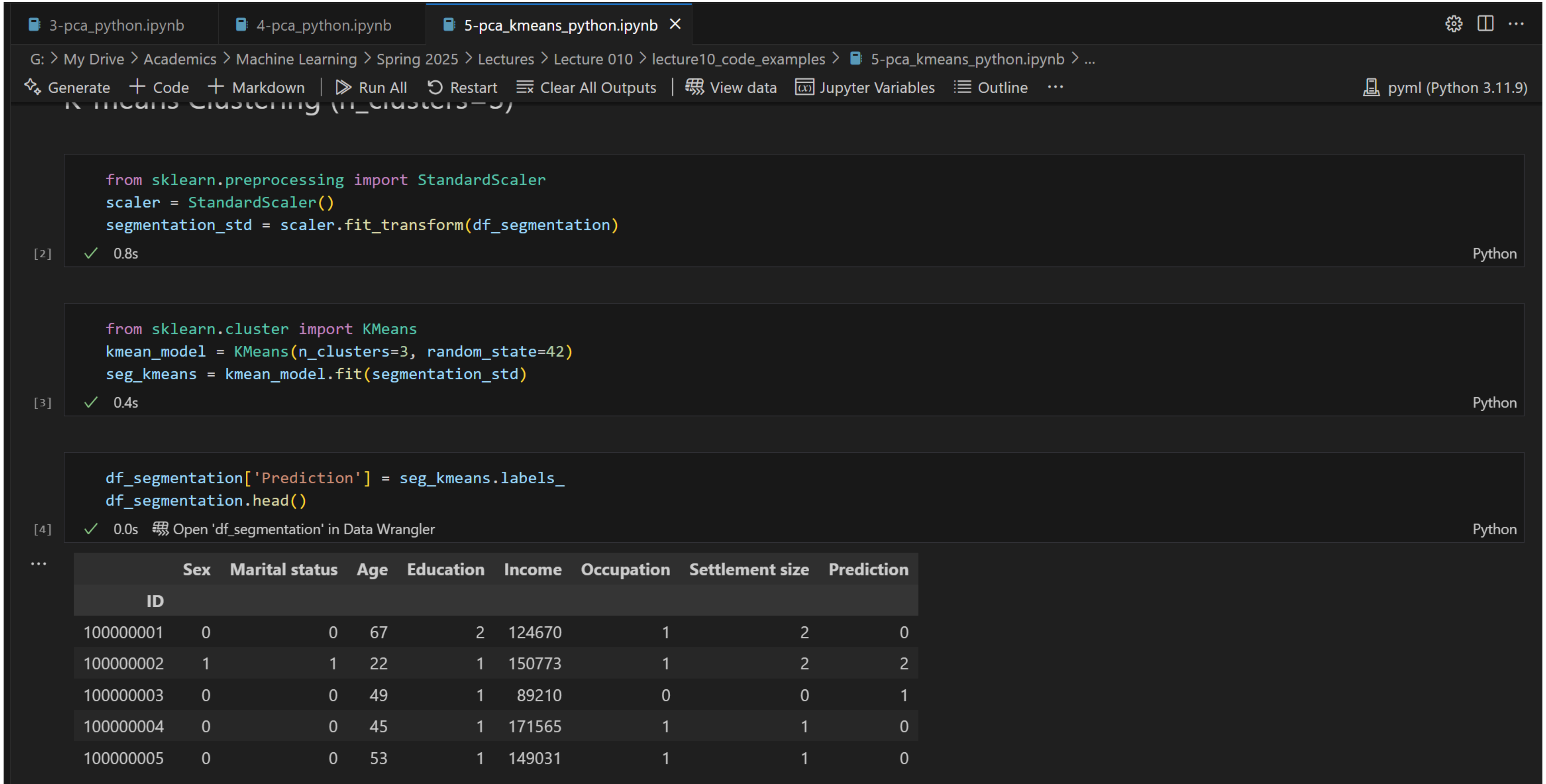
PCA IN PYTHON



```
Text(0, 0.5, 'Cumulative Explained Variance')
```

3-pca_python.ipynb
4-pca_python.ipynb
G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 010 > lecture10_code_examples > 4-pca_python.ipynb > ...
Generate
Code
Markdown
Run All
Restart
Clear All Outputs
View data
Jupyter Variables
Outline
pyml (Python 3.11.9)
Explained Variance by Components
Cumulative Explained Variance
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
Number of Components

PCA + CLUSTERING IN PYTHON



3-pca_python.ipynb
4-pca_python.ipynb
5-pca_kmeans_python.ipynb
G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 010 > lecture10_code_examples > 5-pca_kmeans_python.ipynb > ...
Generate
+ Code
+ Markdown
Run All
Restart
Clear All Outputs
View data
Jupyter Variables
Outline
pyml (Python 3.11.9)

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
segmentation_std = scaler.fit_transform(df_segmentation)
```

[2] ✓ 0.8s Python

```python
from sklearn.cluster import KMeans
kmean_model = KMeans(n_clusters=3, random_state=42)
seg_kmeans = kmean_model.fit(segmentation_std)
```

[3] ✓ 0.4s Python

```python
df_segmentation['Prediction'] = seg_kmeans.labels_
df_segmentation.head()
```

[4] ✓ 0.0s Open 'df_segmentation' in Data Wrangler Python

	Sex	Marital status	Age	Education	Income	Occupation	Settlement size	Prediction
ID								
100000001	0	0	67	2	124670	1	2	0
100000002	1	1	22	1	150773	1	2	2
100000003	0	0	49	1	89210	0	0	1
100000004	0	0	45	1	171565	1	1	0
100000005	0	0	53	1	149031	1	1	0

Next Class: Other Feature Selection Algorithms

Q/A?