

COIT20277 Introduction to Artificial Intelligence

Week 2

- Manage Anaconda Environments
- Scikit-Learn Installation and Basics
- Python for Supervised Learning



BE WHAT YOU WANT TO BE
cqu.edu.au

Acknowledgement of Country

I respectfully acknowledge the Traditional Custodians of the land on which we live, work and learn. I pay my respects to the First Nations people and their Elders, past, present and future

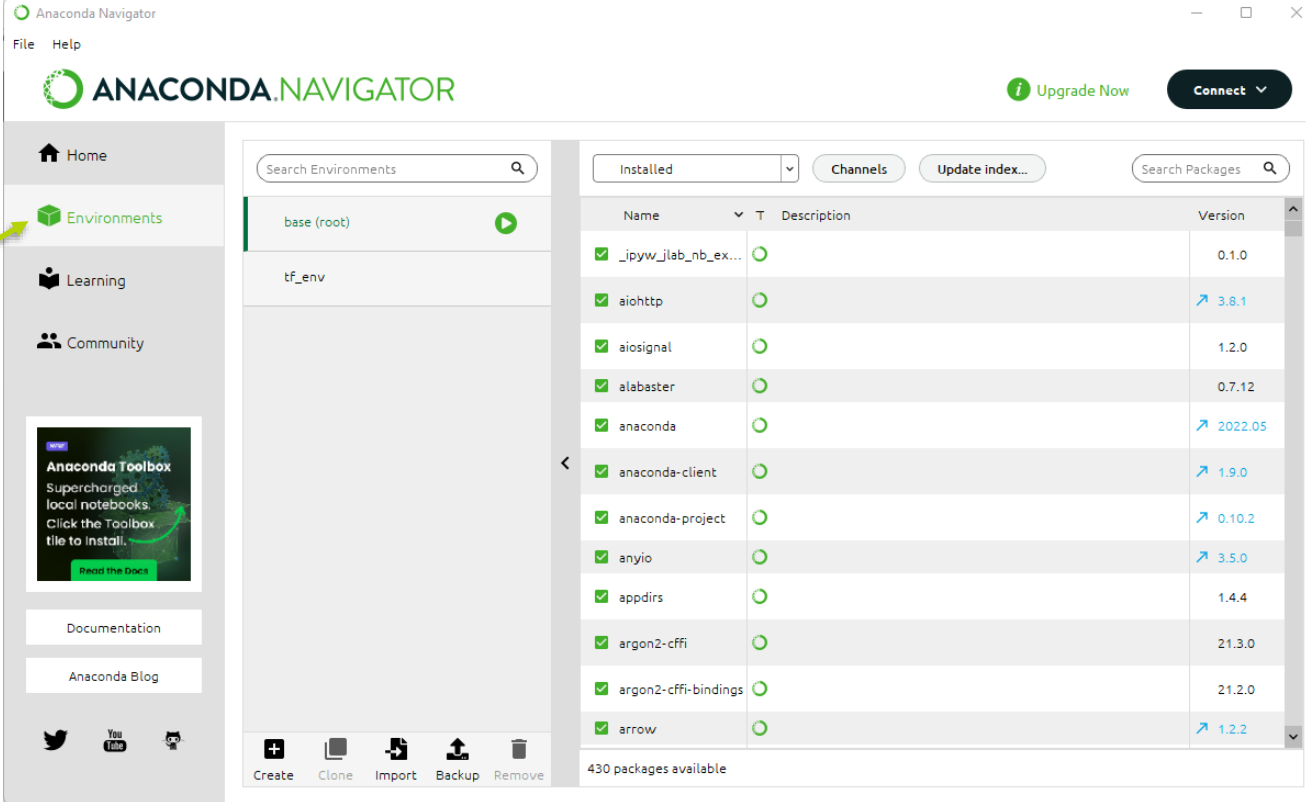


BE WHAT **YOU** WANT TO BE
cqu.edu.au

Managing environments

- On the **Environments** page, the left column displays your environments. At the bottom of the environments list are the Create, Clone, Import, Backup, and Remove buttons.

Click



The screenshot shows the Anaconda Navigator application window. The left sidebar has a menu with 'Home', 'Environments', 'Learning', and 'Community'. The 'Environments' button is highlighted with a yellow arrow and the word 'Click'. The main panel displays the 'Environments' page with a search bar, a list of environments ('base (root)' and 'tf_env'), and a table of installed packages. At the bottom of the environments list are buttons for 'Create', 'Clone', 'Import', 'Backup', and 'Remove'.

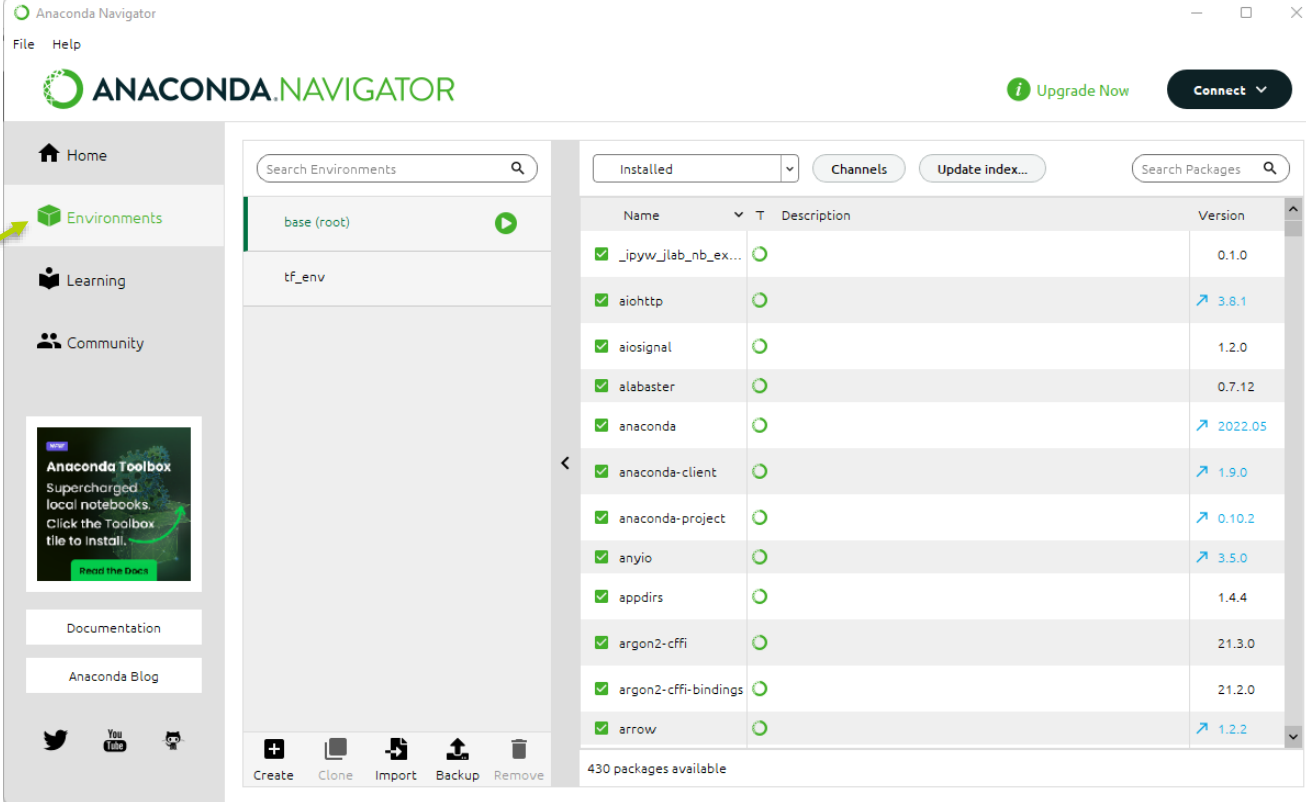
| Name | Description | Version |
|------------------------|-------------|---------|
| ✓ _ipyw_jlab_nb_ex... | | 0.1.0 |
| ✓ aiohttp | | 3.8.1 |
| ✓ aiosignal | | 1.2.0 |
| ✓ alabaster | | 0.7.12 |
| ✓ anaconda | | 2022.05 |
| ✓ anaconda-client | | 1.9.0 |
| ✓ anaconda-project | | 0.10.2 |
| ✓ anyio | | 3.5.0 |
| ✓ appdirs | | 1.4.4 |
| ✓ argon2-cffi | | 21.3.0 |
| ✓ argon2-cffi-bindings | | 21.2.0 |
| ✓ arrow | | 1.2.2 |

430 packages available

Creating a new environment

- At the bottom of the environments list, select **Create**.
- In the Create new environment dialog, enter **COIT20277** for the new environment.

Click



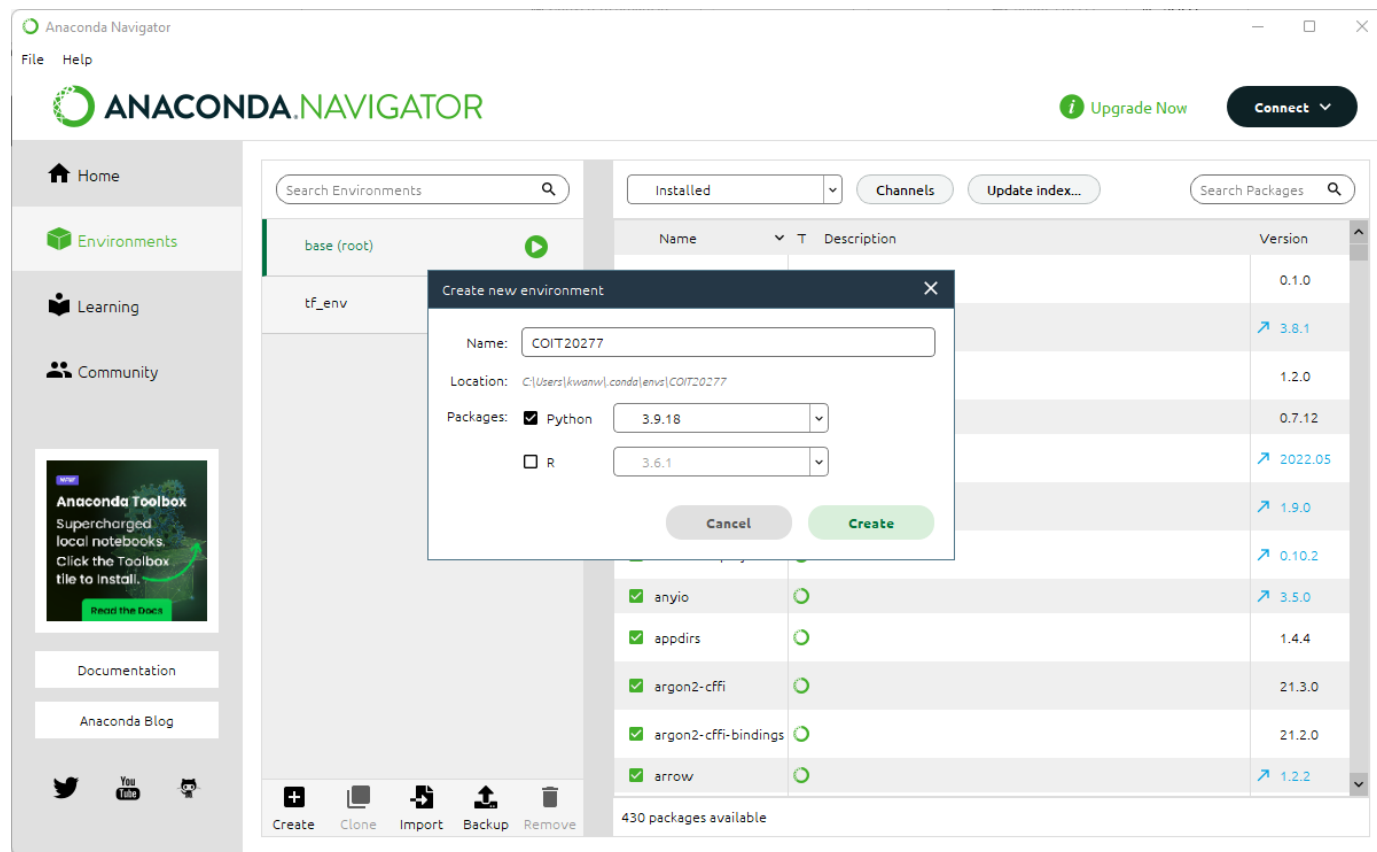
The screenshot shows the Anaconda Navigator application window. The left sidebar has a menu with 'Home', 'Environments', 'Learning', and 'Community'. The 'Environments' button is highlighted with a yellow arrow and the word 'Click'. The main panel displays the 'Environments' tab with a search bar and a list of environments: 'base (root)' and 'tf_env'. Below the list is a 'Create' button. The right panel shows a table of installed packages.

| Name | Description | Version |
|------------------------|-------------|---------|
| ✓ _ipyw_jlab_nb_ex... | | 0.1.0 |
| ✓ aiohttp | | 3.8.1 |
| ✓ aiosignal | | 1.2.0 |
| ✓ alabaster | | 0.7.12 |
| ✓ anaconda | | 2022.05 |
| ✓ anaconda-client | | 1.9.0 |
| ✓ anaconda-project | | 0.10.2 |
| ✓ anyio | | 3.5.0 |
| ✓ appdirs | | 1.4.4 |
| ✓ argon2-cffi | | 21.3.0 |
| ✓ argon2-cffi-bindings | | 21.2.0 |
| ✓ arrow | | 1.2.2 |

430 packages available

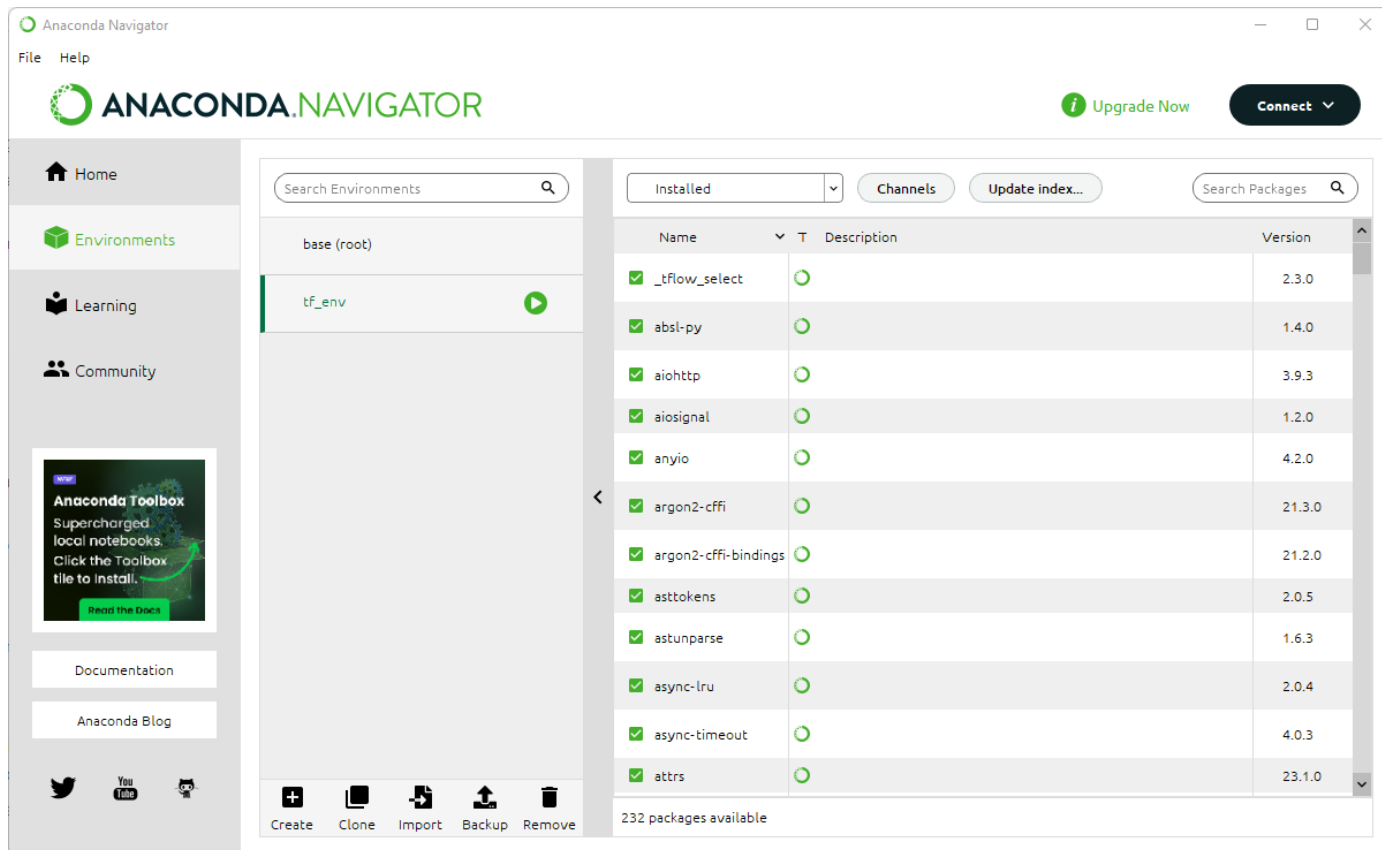
Creating a new environment (cont...)

- Select **Python (3.9.18)** to set the package type for the new environment.
- Click **Create**.



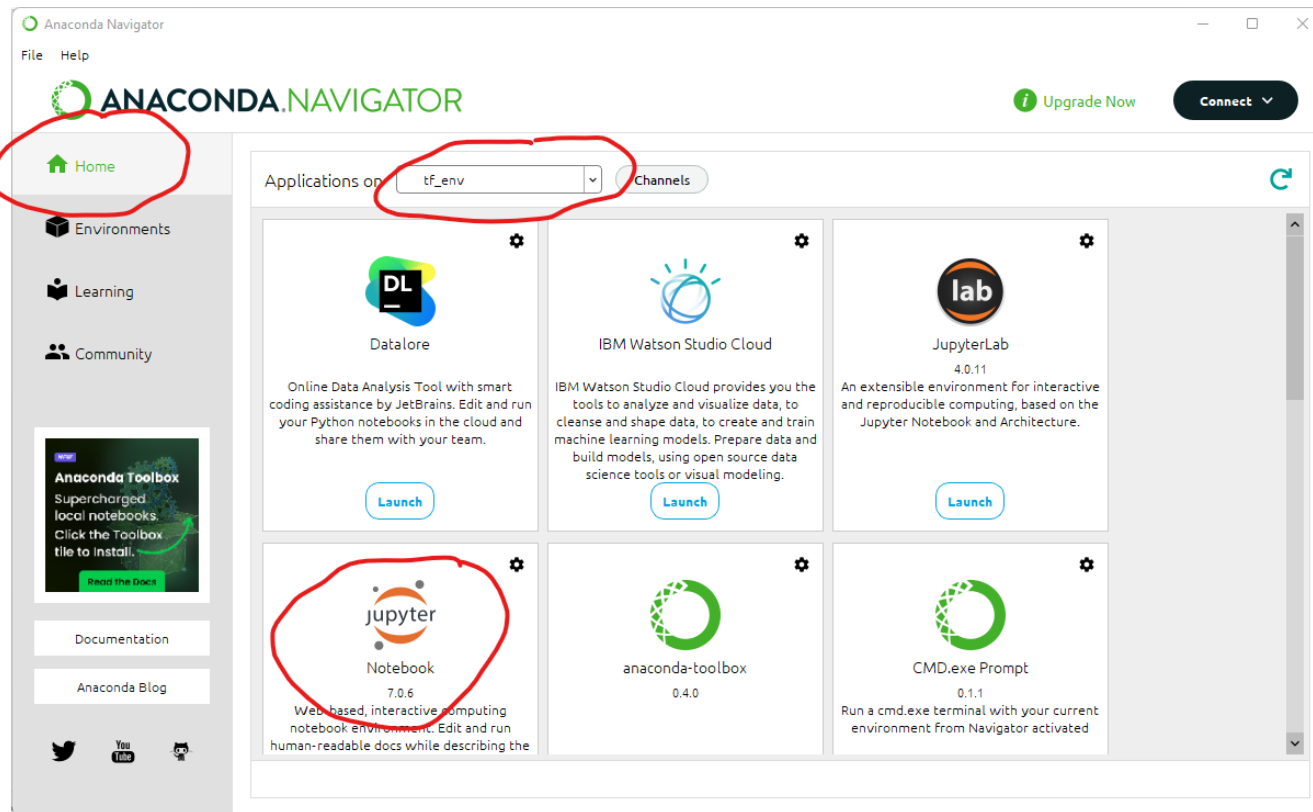
Using an environment

- In the environments list, select the environment name (in my case, `tf_env`) to activate it.
- Then, launch



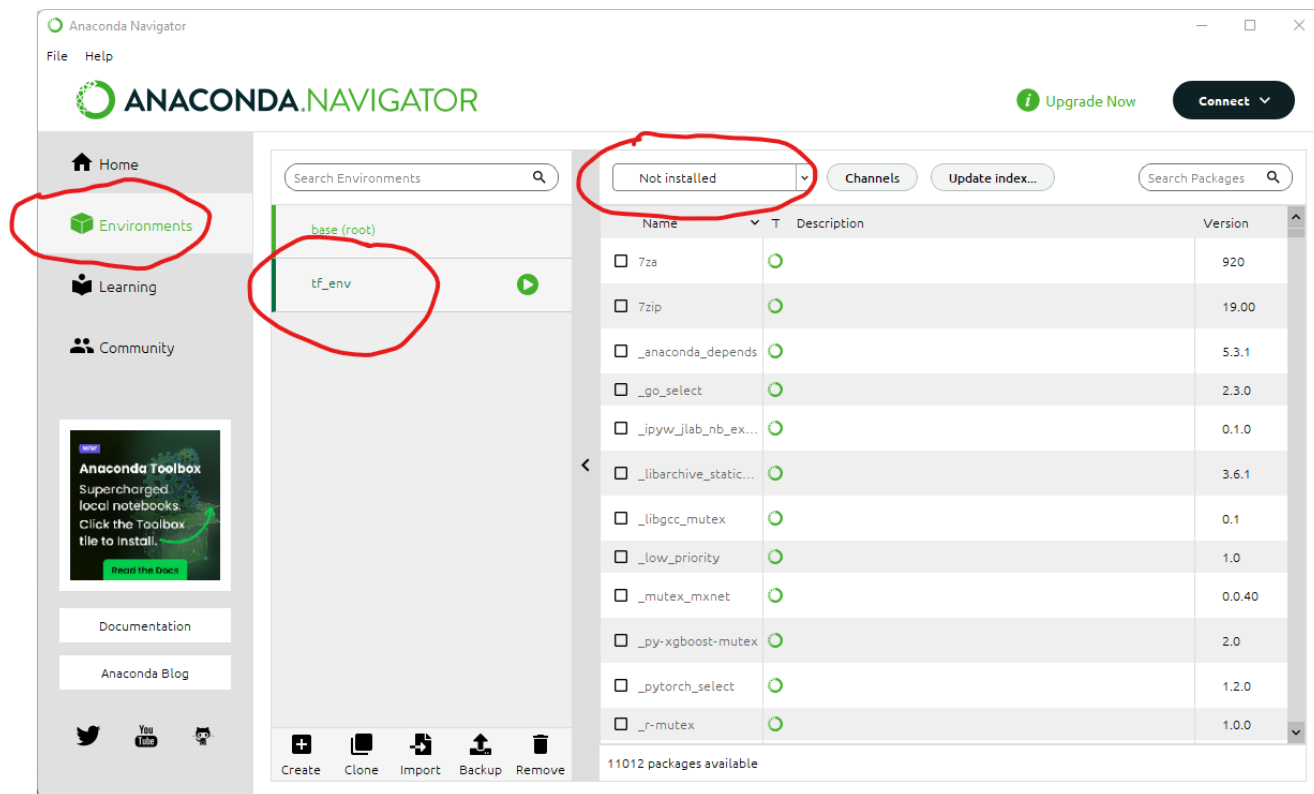
Using an environment (cont...)

- Select **Home** on the left column.
- When Jupyter Notebook is launched, it will use settings and packages installed in this environment.



Scikit-Learn Installation

- Scikit-Learn is just one of Python packages that you can install in your current environment.
- Install Scikit-Learn by following the instructions at <https://docs.anaconda.com/free/navigator/tutorials/manage-packages/>



Additional Packages Needed

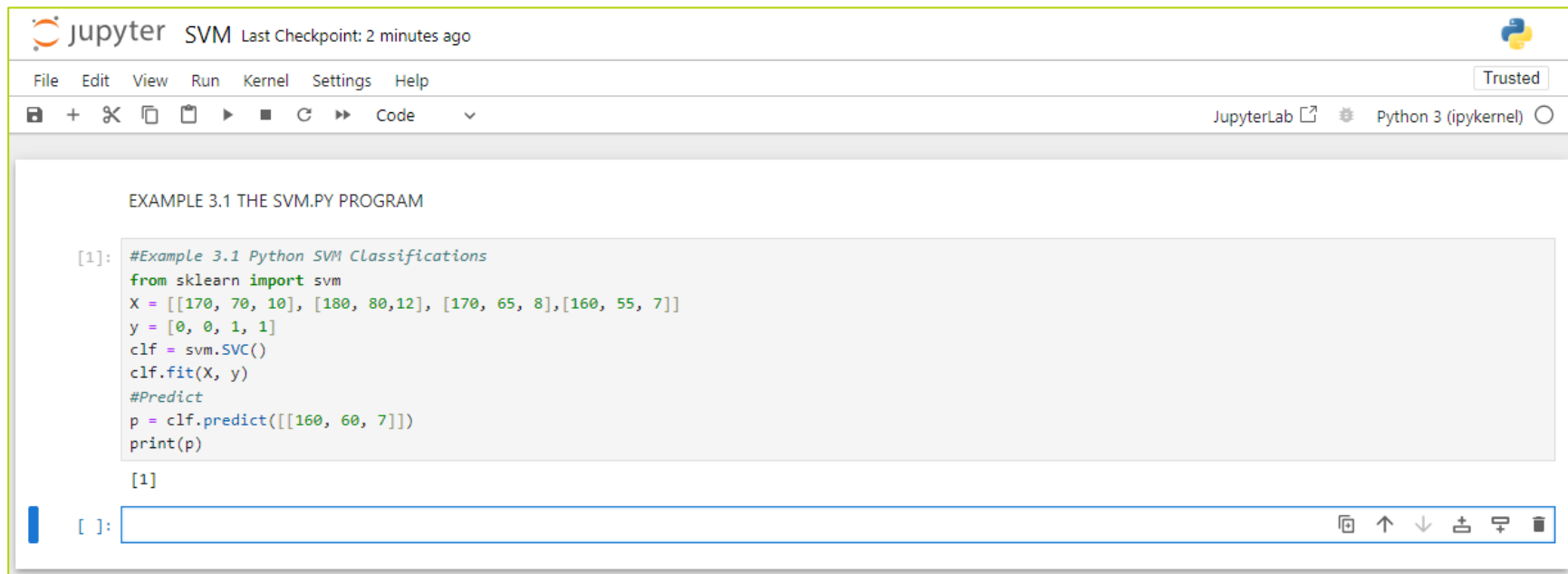
- Numpy: For dealing with N -dimensional array data
- Pandas: For reading data files and analyzing data
- Matplotlib: For plotting data
- SciPy: For scientific computing

Supervised Learning: Classification

- The supervised learning algorithms explored in this tutorial are:
 - Support vector machines
 - Naïve Bayes
 - Decision trees and Random forest
 - K-nearest neighbor
- You can learn more about these algorithms implemented in scikit-learn by accessing https://scikit-learn.org/stable/supervised_learning.html

Support Vector Machine (Prac #1)

- Example 3.1 shows a simple SVM gender classification example based on height, weight, and shoe size.
- It first uses `from sklearn import svm` to import the SVM library. It uses an array called `x` to store four sets of values of height in centimeters, weight in kilos, and shoe size in UK size.
- It uses an array named `y` to store four sets of known genders, 0 for Male, 1 for Female.
- It then trains the SVM classifier and makes a prediction for a given height, weight, and shoe size `[[160, 60, 7]]`.



The screenshot shows a JupyterLab window titled "SVM" with a last checkpoint of 2 minutes ago. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and execution. The code cell contains the following Python code:

```
EXAMPLE 3.1 THE SVM.PY PROGRAM

[1]: #Example 3.1 Python SVM Classifications
from sklearn import svm
X = [[170, 70, 10], [180, 80, 12], [170, 65, 8], [160, 55, 7]]
y = [0, 0, 1, 1]
clf = svm.SVC()
clf.fit(X, y)
#Predict
p = clf.predict([[160, 60, 7]])
print(p)

[1]
```

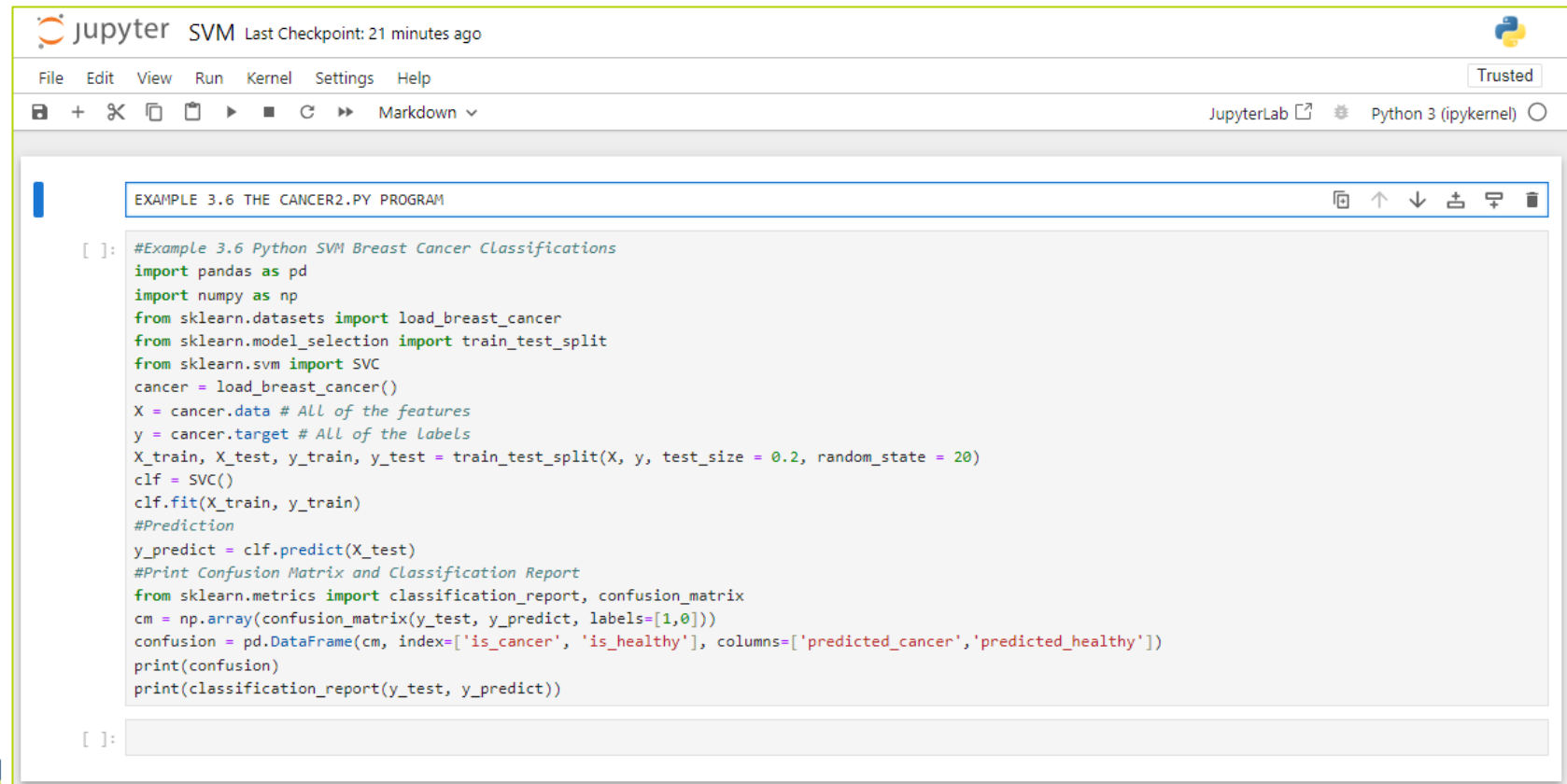
The output of the code cell is `[1]`, indicating the predicted gender for the input features is Female. The bottom of the interface shows a prompt `[]:` for the next input.

Prac #1

- Modify the Python program from the last slide to use twice the number of samples of x and y .
- Submit your Python program at the end of the tutorial.

Support Vector Machine (Prac #2)

- It first loads the breast cancer data's `load_breast_cancer()` function and then puts the data into a dataframe format, uses features data as `X` and target as `y`.
- It uses `train_test_split()` to split `X` and `y` into training and testing sets.
- Specifically, 80 percent of data is for training, and 20 percent is for testing.
- It uses the training data `X_train` and `y_train` to train an SVM and uses testing data `X_test` to make predictions. `w`



The screenshot shows a JupyterLab environment with a single code cell. The cell is titled "EXAMPLE 3.6 THE CANCER2.PY PROGRAM". The code within the cell is a Python script that demonstrates how to load breast cancer data, split it into training and testing sets, train an SVM model, and evaluate its performance using a confusion matrix and classification report.

```
[ ]: #Example 3.6 Python SVM Breast Cancer Classifications
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
cancer = load_breast_cancer()
X = cancer.data # All of the features
y = cancer.target # All of the Labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 20)
clf = SVC()
clf.fit(X_train, y_train)
#Prediction
y_predict = clf.predict(X_test)
#Print Confusion Matrix and Classification Report
from sklearn.metrics import classification_report, confusion_matrix
cm = np.array(confusion_matrix(y_test, y_predict, labels=[1,0]))
confusion = pd.DataFrame(cm, index=['is_cancer', 'is_healthy'], columns=['predicted_cancer', 'predicted_healthy'])
print(confusion)
print(classification_report(y_test, y_predict))

[ ]:
```

Prac #2

- Enter the Python program shown in the last slide in your Jupyter Notebook.
- Run the code to explore and make sure you understand the output.
- Experiment by varying the sizes of the training and the test sets. Plot the classification accuracy versus the size of the training set.
- Submit your Python program at the end of the tutorial.

Naïve Bayes

- In the Python code, it uses `X, y = load_iris(return_X_y=True)` to load the iris data and returns the data as `X` and `y`, where `X` includes all four features of that data.
- It then trains the naive Bayes classifier model and makes a prediction for a given sample with the sepal's length and width and the petal's length and width.



The screenshot displays a JupyterLab environment with a single code cell. The interface includes a top bar with the Jupyter logo, the name 'SVM', and the last checkpoint time '45 minutes ago'. Below this is a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Settings', and 'Help'. A toolbar with various icons for file operations and execution is visible. The code cell contains the following Python code:

```
EXAMPLE 3.7 THE NAIVEBAYES.PY PROGRAM

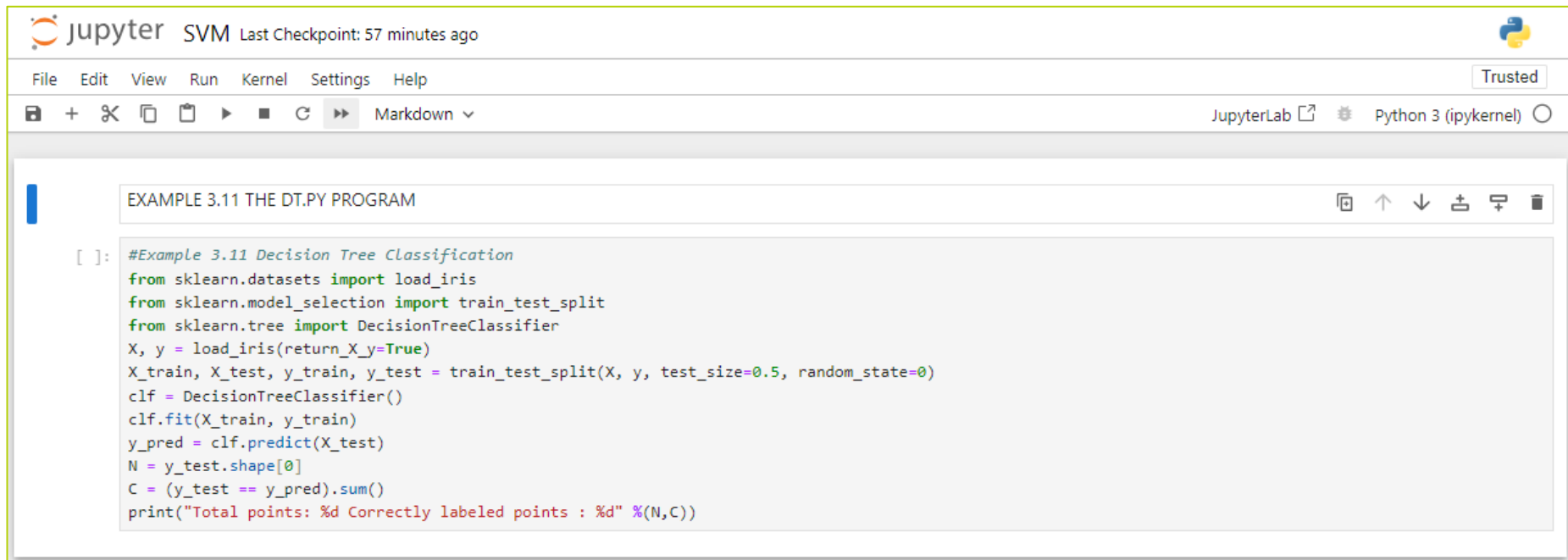
[ ]: #Example 3.7 Naive Bayes Iris
      from sklearn.datasets import load_iris
      from sklearn.model_selection import train_test_split
      from sklearn.naive_bayes import GaussianNB
      X, y = load_iris(return_X_y=True)
      print(X)
      clf = GaussianNB()
      clf.fit(X, y)
      p = clf.predict([[5.0, 3.4, 1.5, 0.4]])
      print(p)
```

Prac

- What does `GaussianNB` refer to in the Python code? Report on this in your submission.
- Modify the code so that it uses different training and test splits. Run the code and make sure you understand the output.
- Experiment by varying the sizes of the training and the test sets. Plot the classification accuracy versus the size of the training set.
- Submit your Python program at the end of the tutorial.

Decision Trees

- In the Python code, it uses `X, y = load_iris(return_X_y=True)` to load the Iris data and returns the data as `X` and `y`, using all four features of the data.
- It then splits the data into the training set and the testing set.
- It then trains the decision tree classifier model and makes predictions on the testing set.
- It also calculates and displays the total number of points, as well as the number of points that are correctly predicted.



The screenshot shows a JupyterLab environment with a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The main area displays a code cell titled "EXAMPLE 3.11 THE DT.PY PROGRAM". The code imports `load_iris` from `sklearn.datasets`, `train_test_split` from `sklearn.model_selection`, and `DecisionTreeClassifier` from `sklearn.tree`. It then loads the Iris data, splits it into training and testing sets, trains a `DecisionTreeClassifier` model, makes predictions on the test set, and prints the total number of points and the number of correctly labeled points.

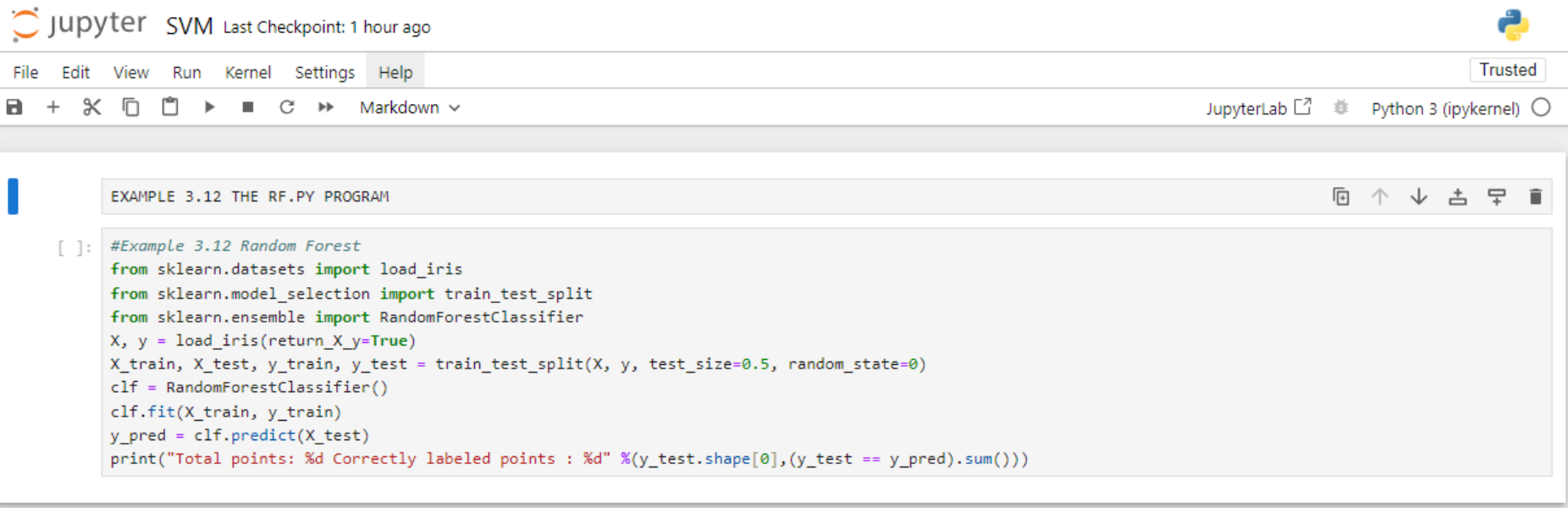
```
[ ]: #Example 3.11 Decision Tree Classification
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
N = y_test.shape[0]
C = (y_test == y_pred).sum()
print("Total points: %d Correctly labeled points : %d" %(N,C))
```

Prac

- Modify the Python program from the last slide so that it performs decision tree classification on Scikit-Learn's wine data.
- https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html
- Experiment by varying the sizes of the training and the test sets. Plot the classification accuracy versus the size of the training set.
- Submit your Python program at the end of the tutorial.

Random Forest

- In the Python code, it uses `X, y = load_iris(return_X_y=True)` to load the Iris data and returns the data as `X` and `y`, using all four features of the data.
- It then splits the data into the training set and the testing set.
- It then trains the random forest classifier model and makes predictions on the testing set.
- It calculates and displays the number of points that are correctly predicted.



The screenshot shows a JupyterLab environment with a file named 'SVM'. The interface includes a top bar with the Jupyter logo, the file name 'SVM', and the last checkpoint time 'Last Checkpoint: 1 hour ago'. Below this is a menu bar with 'File', 'Edit', 'View', 'Run', 'Kernel', 'Settings', and 'Help'. A toolbar with various icons for file operations and execution is visible. The main area displays a code cell with the following Python code:

```
EXAMPLE 3.12 THE RF.PY PROGRAM

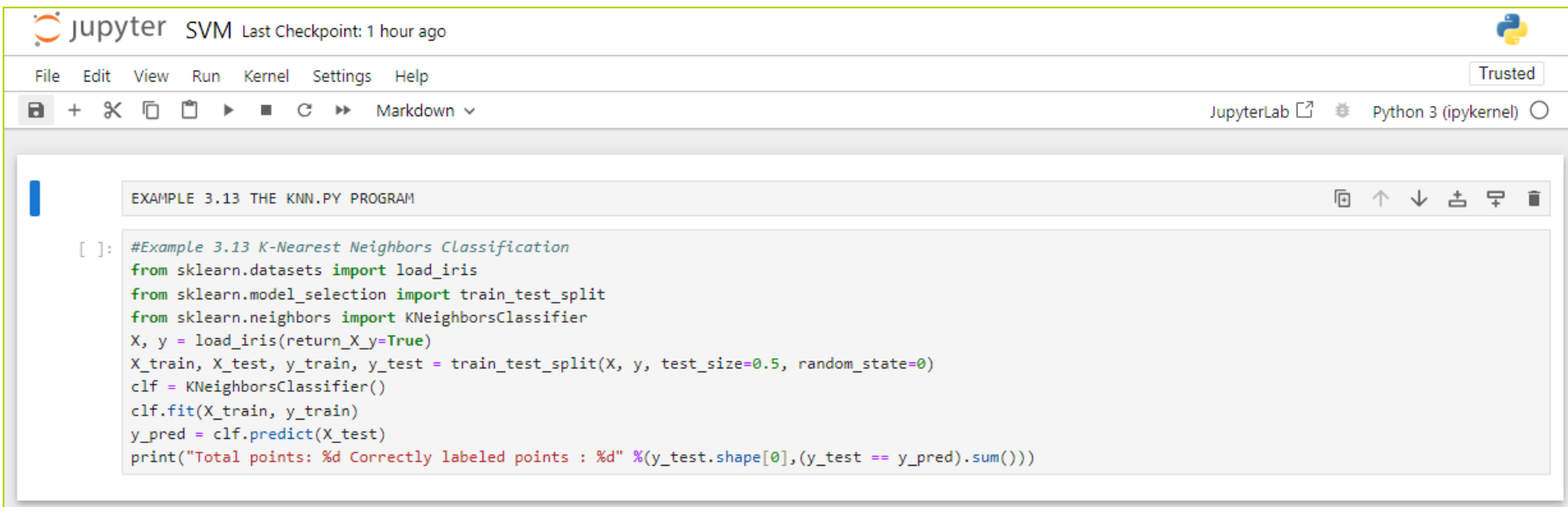
[ ]: #Example 3.12 Random Forest
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Total points: %d Correctly labeled points : %d" %(y_test.shape[0],(y_test == y_pred).sum()))
```

Prac

- Modify the Python program from the last slide so that it performs random forest classification on Scikit-Learn's diabetes data.
- https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes
- Plot the classification accuracy versus the size of the training set.
- Submit your Python program at the end of the tutorial.

K-Nearest Neighbors

- In the Python code, it uses `X, y = load_iris(return_X_y=True)` to load the Iris data and returns the data as `X` and `y`, using all four features of the data.
- It then splits the data into the training set and the testing set.
- It then trains the K-NN classifier model and makes predictions on the testing set.
- It calculates and displays the number of points that are correctly predicted.



The screenshot shows a JupyterLab environment with a single code cell. The interface includes a top bar with the JupyterLab logo, a 'Trusted' status indicator, and a menu bar with options like File, Edit, View, Run, Kernel, Settings, and Help. Below the menu is a toolbar with icons for file operations and execution. The code cell is titled 'EXAMPLE 3.13 THE KNN.PY PROGRAM' and contains the following Python code:

```
[ ]: #Example 3.13 K-Nearest Neighbors Classification
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Total points: %d Correctly labeled points : %d" %(y_test.shape[0],(y_test == y_pred).sum()))
```

Prac

- Modify the Python program from the last slide so that it performs K-Nearest Neighbors classification on Scikit-Learn's diabetes data.
- https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes
- Plot the classification accuracy versus the size of the training set.
- Submit your Python program at the end of the tutorial.



THANK YOU

TIME FOR DISCUSSION & QUESTIONS