PYTHON DATA MINING

COMP7103 Tutorial 3
Dr Kevin Wu
thwu@cs.hku.hk

(Some material from Kevin Lam)

OUTLINE

Introduction: Python Installation, Environment Setup

Data Preparation

Classification

Explainable AI (self-study)

PYTHON & JUPYTER NOTEBOOK

In this tutorial, we will run python code with **Jupyter Notebook** (formerly known as the *IPython Notebook*).

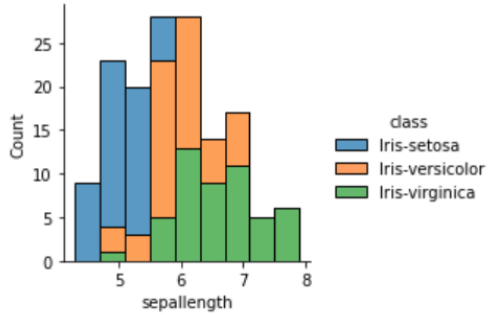
Advantages:

- Great for organizing and displaying the code and outputs
- Easy for data exploration

Code

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


plt.show()
iris_df = pd.read_csv("iris.csv")
sns.displot(iris_df, x="sepallength", hue="class", multiple="stack", height=3)
plt.show()
```



Formula

The formula of entropy $H(S) = \sum_i p_i \log_2 \frac{1}{p_i}$ can be generated by the LaTeX code:

```
$H(S) = \sum_i p_i \log_2 \frac {1}{p_i}$
```

INSTALL ANACONDA

Anaconda contains **Python** and some useful packages (including the Jupyter notebook)

Download and install Anacoda: https://www.anaconda.com/products/individual-d

To run Jupyter after installing anaconda, go to **Terminal** (Mac/Linux) or **Anaconda Prompt** (Windows) and run:

```
jupyter-lab
```

You should see the notebook open in your browser. Do not close this terminal.

Alternatively, you may run the code on Google Colab:
https://colab.research.google.com/

SET UP CONDA ENVIRONMENT

One good practice is to create a conda environment for each project such that python packages can be managed independently.

To create and activate a conda environment, open another terminal / anaconda prompt (don't use the one running jupyter lab) and run:

```
conda create -n comp7103-venv
conda activate comp7103-venv
```

To check if the conda environment is in effect:

```
conda env list
```

The output should be similar to:

```
base                     /home/username/anaconda3
comp7103-venv         *  /home/username/anaconda3/envs/comp7103-venv
```

An asterisk(*) should be next to comp7103-venv

INSTALL REQUIRED PACKAGES

In this tutorial, we will be using

- NumPy - https://numpy.org/
  - For multidimension array manipulation
- Pandas - https://pandas.pydata.org/
  - Data analysis and manipulation
- Matplotlib - https://matplotlib.org/
  - Generating figures with python
- Seaborn - http://seaborn.pydata.org/
  - Statistical data visualization
- Scikit-learn - https://scikit-learn.org/
  - Machine learning tools

To install these packages, run the following command in the terminal with `comp7103-venv` activated:

```
pip install -U numpy pandas matplotlib seaborn scikit-learn
```

You also need to know the following in Python:

- Import modules - https://docs.python.org/3/reference/import.html
- Use of generators - https://wiki.python.org/moin/Generators

ADD THE CONDA ENVIRONMENT TO JUPYTER NOTEBOOK

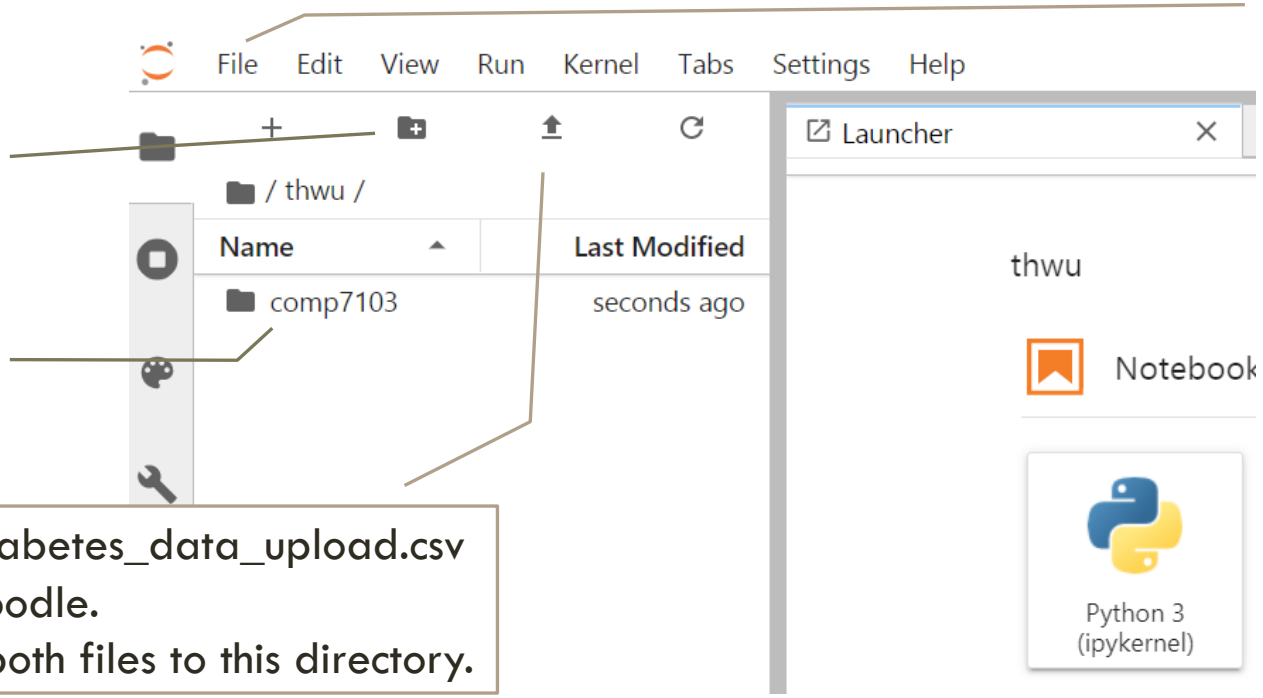
In the terminal with comp7103-venv activated, run:

```
pip install --user ipykernel
python -m ipykernel install --user --name=comp7103-venv
```

Go to Jupyter:

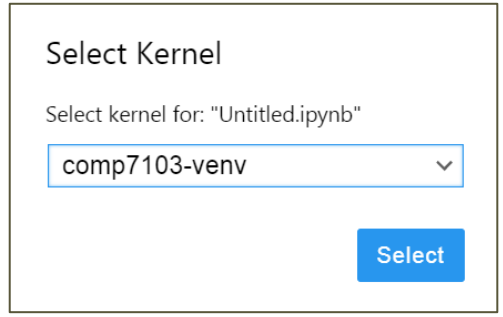
1. Click to create a directory comp7103

2. Double-click to enter the directory

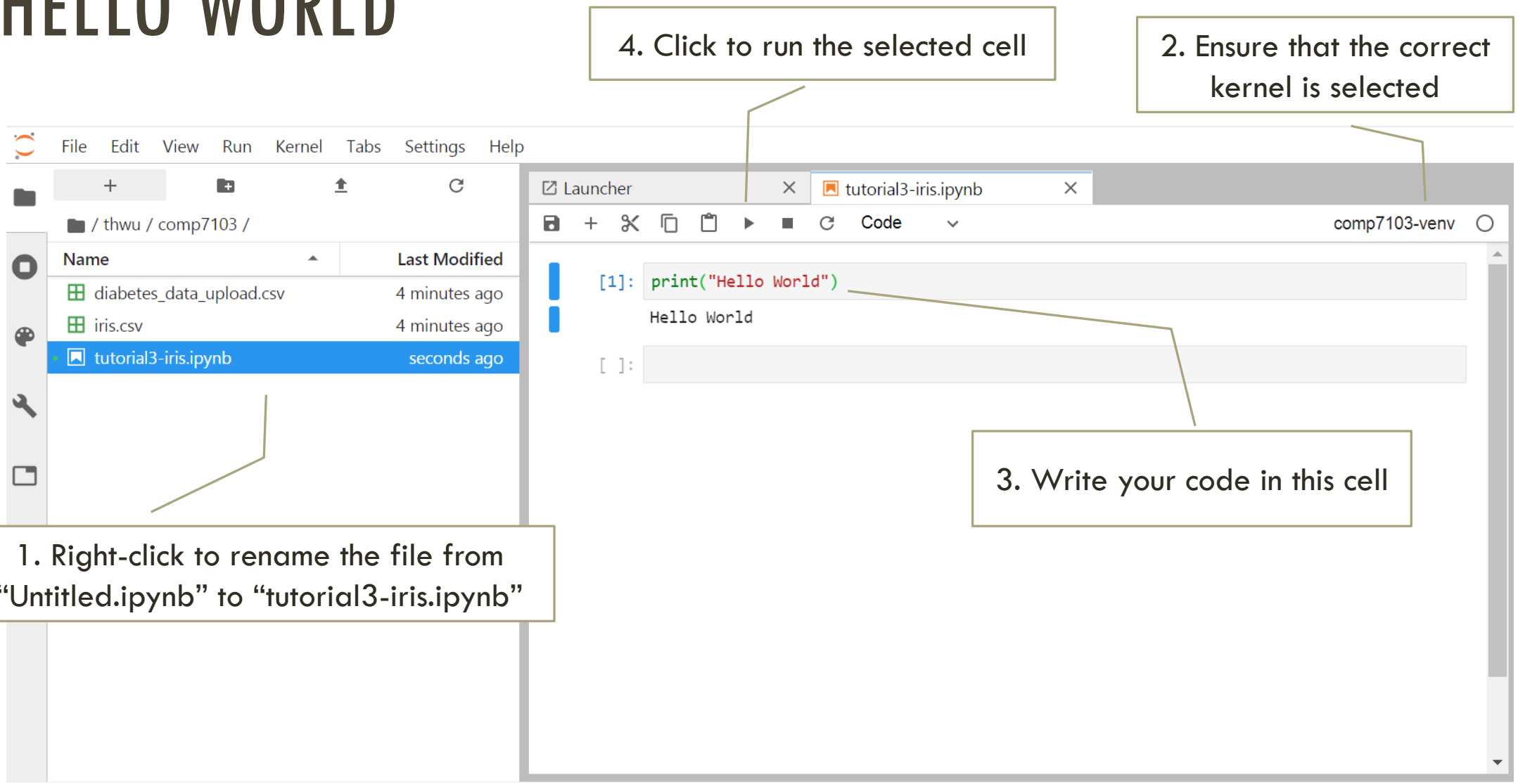


3. Download iris.csv and diabetes_data_upload.csv from moodle.
Click this button to upload both files to this directory.

4. Click [File] → [New] → [Notebook], and select comp7103-venv as the kernel



HELLO WORLD


4. Click to run the selected cell
2. Ensure that the correct kernel is selected
File Edit View Run Kernel Tabs Settings Help
/ thwu / comp7103 /
Name
Last Modified
diabetes_data_upload.csv 4 minutes ago
iris.csv 4 minutes ago
tutorial3-iris.ipynb seconds ago
Launcher
tutorial3-iris.ipynb
Code
comp7103-venv
[1]: print("Hello World")
Hello World
[ ]:
1. Right-click to rename the file from "Untitled.ipynb" to "tutorial3-iris.ipynb"
3. Write your code in this cell

DATA PREPARATION

DataFrame (pandas): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
read_csv (pandas): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

LOAD DATA FROM CSV

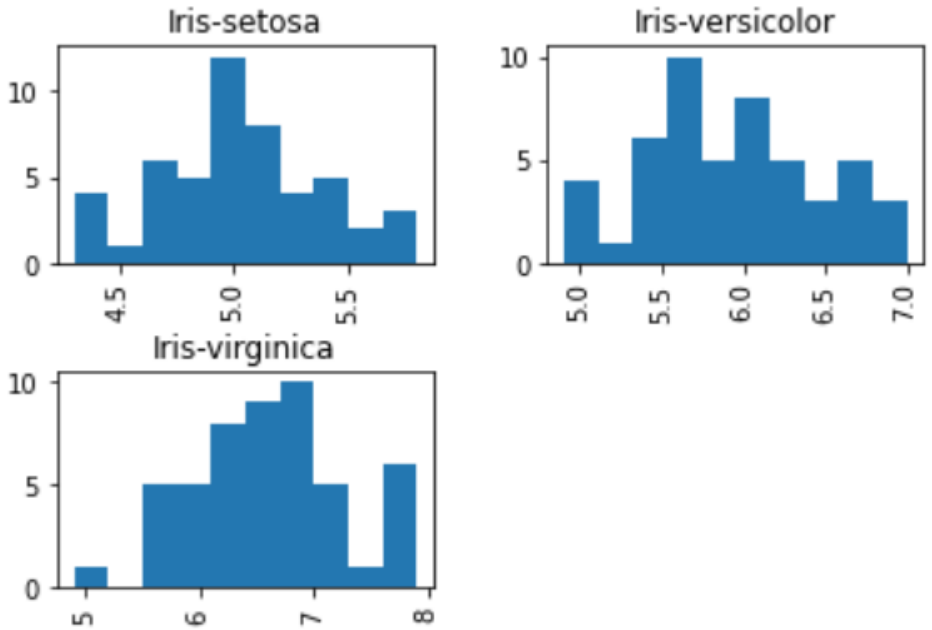
[2]:
```
import pandas as pd

# Load CSV
iris_df = pd.read_csv("iris.csv")

print(iris_df)
```

```
      id  sepallength  sepalwidth  petallength  petalwidth           class
0      1          5.1         3.5          1.4         0.2     Iris-setosa
1      2          4.9         3.0          1.4         0.2     Iris-setosa
2      3          4.7         3.2          1.3         0.2     Iris-setosa
3      4          4.6         3.1          1.5         0.2     Iris-setosa
4      5          5.0         3.6          1.4         0.2     Iris-setosa
..   ...          ...         ...          ...         ...             ...
145  146          6.7         3.0          5.2         2.3  Iris-virginica
146  147          6.3         2.5          5.0         1.9  Iris-virginica
147  148          6.5         3.0          5.2         2.0  Iris-virginica
148  149          6.2         3.4          5.4         2.3  Iris-virginica
149  150          5.9         3.0          5.1         1.8  Iris-virginica

[150 rows x 6 columns]
```

DataFrame.hist (pandas): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.hist.html

HISTOGRAM

[3]:
```
import matplotlib.pyplot as plt

# Group data by "class" label
# Select the "sepallength" column
iris_df["sepallength"].hist(by=iris_df["class"])

# Show plot
plt.show()
```



Selecting data from DataFrame (pandas): https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
Concat (pandas): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html

STACKED HISTOGRAM (PREPARE DATA)

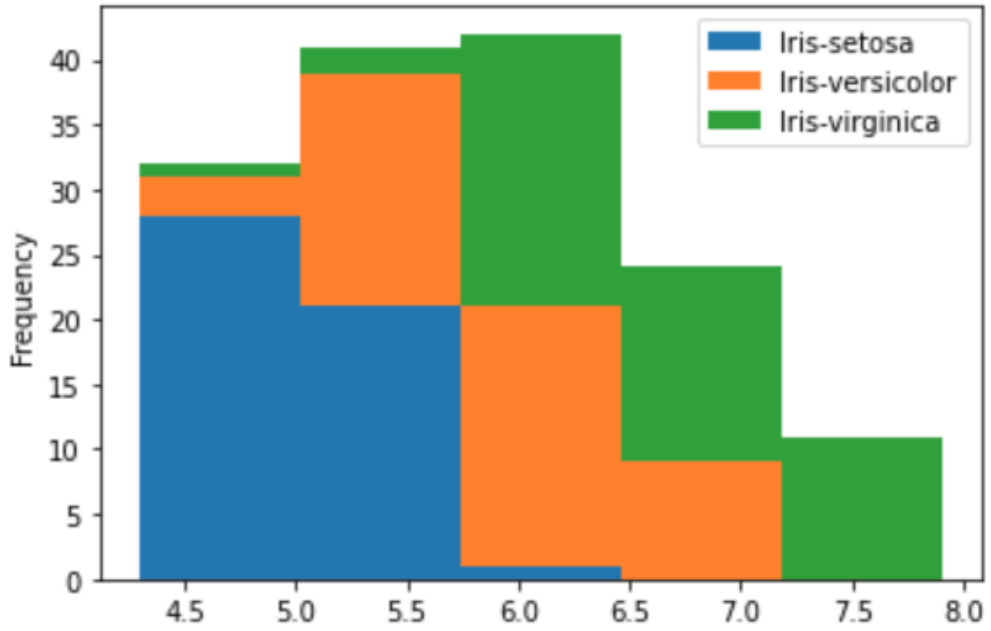
[4]:
```
# Get all class labels
labels = iris_df["class"].unique()

# Extract "attribute" values from dataframe ("input_df")
# with the specific "label"
def filterData(input_df, attribute, label):
    values = input_df[input_df["class"] == label][[attribute]].values
    return pd.DataFrame(data=values,  columns=[label])

# Combine data of different class labels into a table
df = pd.concat([filterData(iris_df, "sepallength", label)
                for label in labels], axis="columns")

# Show the first 5 rows of df
print(df.head(n=5))
```

```
   Iris-setosa  Iris-versicolor  Iris-virginica
0          5.1              7.0             6.3
1          4.9              6.4             5.8
2          4.7              6.9             7.1
3          4.6              5.5             6.3
4          5.0              6.5             6.5
```

Each column contains the sepal lengths of an iris type.

STACKED HISTOGRAM (PLOT)

[5]:
```
# Plot the stacked histogram
df.plot.hist(stacked=True, bins=5)

plt.show()
```



histplot (Seaborn): https://seaborn.pydata.org/generated/seaborn.histplot.html

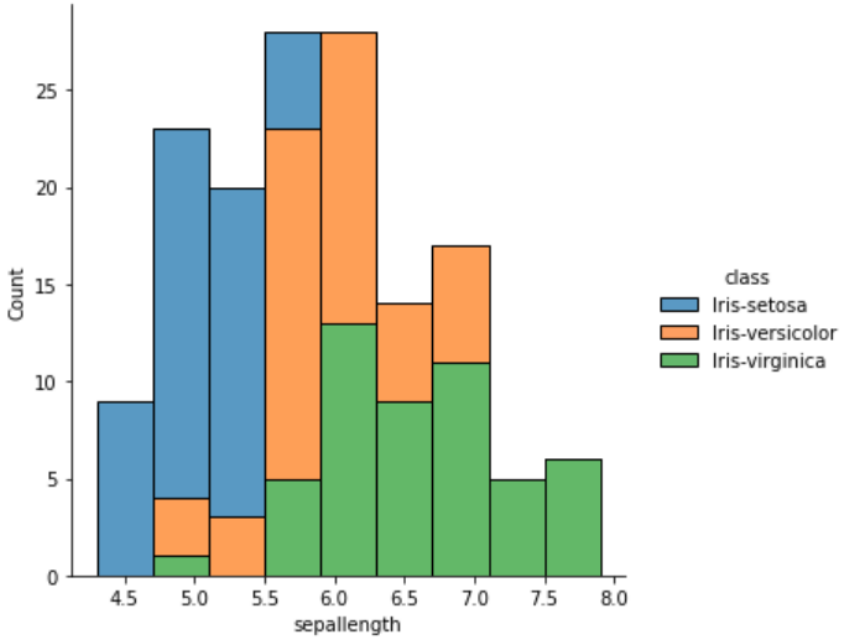
STACKED HISTOGRAM (SEABORN)

[6]:
```
# Using seaborn to plot the stacked histogram
import seaborn as sns

sns.displot(iris_df, x="sepallength", hue="class", multiple="stack")

plt.show()
```

Plot the "sepallength" attribute

Stacked histogram (read the documentation for more options)

Color the bins according to the "class"



Subplots (matplotlib.pyplot): https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.subplots.html

ALL HISTOGRAMS

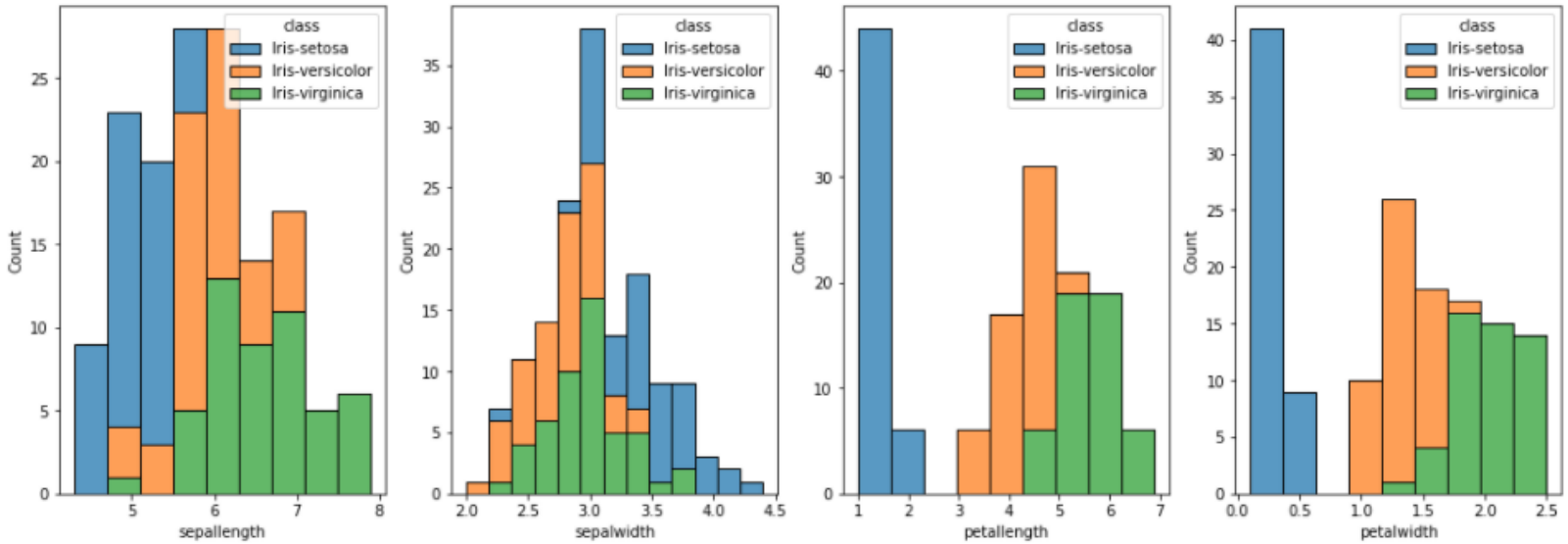
iris_df.columns

~~id~~	sepallength	sepalwidth	petallength	petalwidth	~~class~~

[7]:
```
# Get all attributes except "id" and class labels ("class")
columns = [column for column in iris_df.columns
                  if column not in ['id', 'class']]

# Prepare plot areas
_, axes = plt.subplots(1, len(columns), figsize=(18,6))
for c in range(len(columns)):
    # Plot each histogram at the correct area
    sns.histplot(iris_df, x=columns[c], hue="class", multiple="stack", ax=axes[c])

plt.show()
```

The histograms are organized into 1 row, 4 columns

Specify the location (subplot) of each histogram



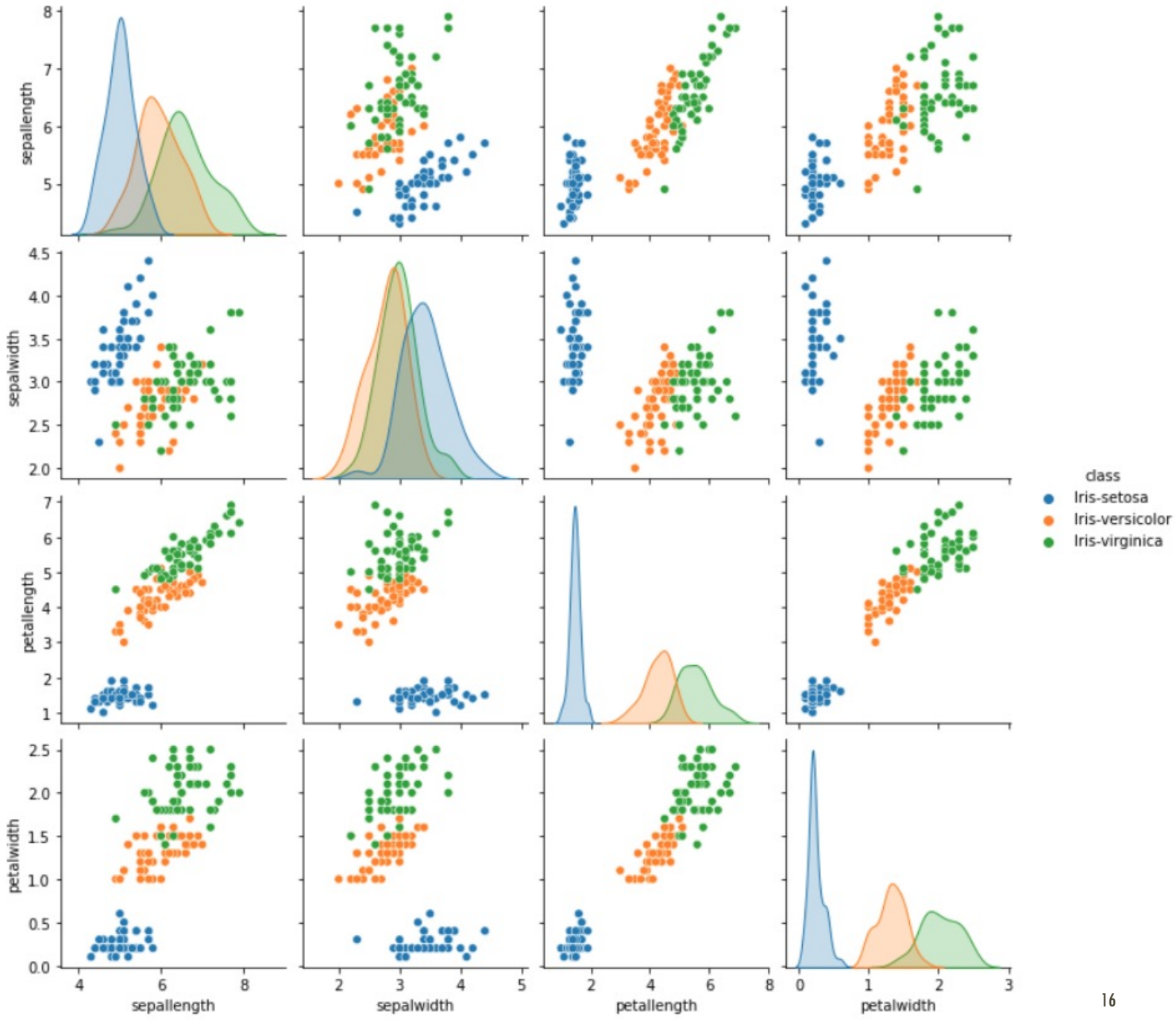
Pairplot (Seaborn): http://seaborn.pydata.org/generated/seaborn.pairplot.html#seaborn.pairplot

SCATTER MATRIX

Drop id

```
[8]: sns.pairplot(iris_df.drop(columns="id"),
                  hue="class")
     plt.show()
```

Plot scatter matrix and color the instances based on the class labels



CLASSIFICATION

DIABETES DATASET

Create a new notebook and name it `tutorial3-diabetes.ipynb`

Load the diabetes dataset:

[1]:
```
import pandas as pd

df = pd.read_csv("diabetes_data_upload.csv")
df
```

[1]:

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia
0	40	Male	No	Yes	No	Yes	No
1	58	Male	No	No	No	Yes	No
2	41	Male	Yes	No	No	Yes	Yes
3	45	Male	No	No	Yes	Yes	Yes
4	60	Male	Yes	Yes	Yes	Yes	Yes
...	...	...	...	...	...	...	...

Get_dummies (pandas): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
Replace (pandas): https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html

CONVERT CATEGORICAL ATTRIBUTES TO BINARY ATTRIBUTES

[2]:
```
# scikit-learn's decision tree (CART) does not support categorical attributes
# Convert categorical attributes to binary attributes using get_dummies()

X = pd.get_dummies(df.drop(columns="class"))
y = df["class"]

X
```

[2]:

	Age	Gender_Female	Gender_Male	Polyuria_No	Polyuria_Yes	Polydipsia_No	Polydipsia_Yes
0	40	0	1	1	0	0	1
1	58	0	1	1	0	1	0
2	41	0	1	0	1	1	0
3	45	0	1	1	0	1	0
4	60	0	1	0	1	0	1
...	...	...	...	...	...	...	...

Can we get the dataframe X without redundancy?

Tree (scikit-learn): https://scikit-learn.org/stable/modules/tree.html#tree

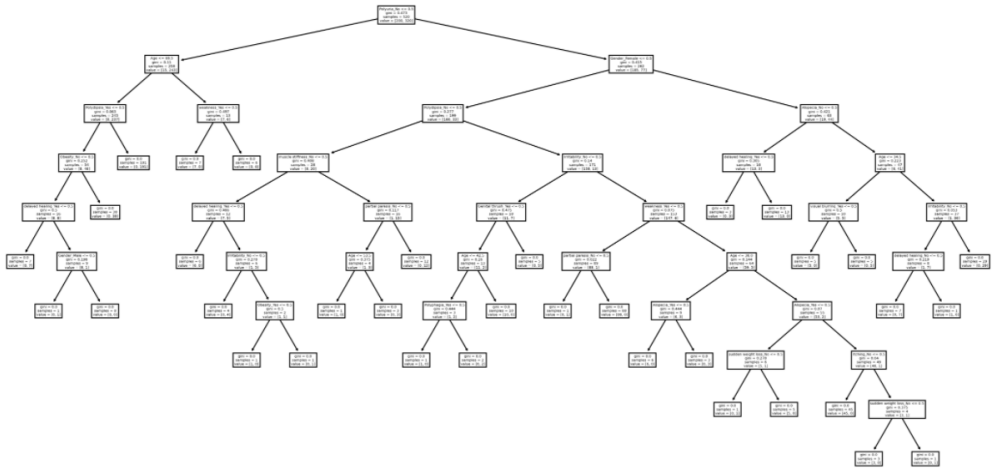
BUILDING DECISION TREE

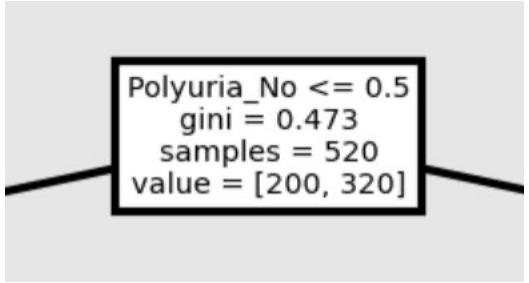
[3]:
```
from sklearn import tree
# Build decision tree
dtc = tree.DecisionTreeClassifier().fit(X, y)

# Export the tree in text form
# Use column label as names in the tree
print(tree.export_text(dtc, feature_names=X.columns.tolist()))
```

```
|--- Polyuria_No <= 0.50
|   |--- Age <= 69.50
|   |   |--- Polydipsia_Yes <= 0.50
|   |   |   |--- Obesity_No <= 0.50
|   |   |   |   |--- delayed healing_Yes <= 0.50
|   |   |   |   |   |--- class: Positive
|   |   |   |   |--- delayed healing_Yes >  0.50
|   |   |   |   |   |--- Gender_Male <= 0.50
|   |   |   |   |   |   |--- class: Positive
|   |   |   |   |   |--- Gender_Male >  0.50
|   |   |   |   |   |   |--- class: Negative
|   |   |   |--- Obesity_No >  0.50
|   |   |   |   |--- class: Positive
|   |   |--- Polydipsia_Yes >  0.50
|   |   |   |--- class: Positive
|   |--- Age >  69.50
|   |   |--- weakness_Yes <= 0.50
```

TREE VISUALIZATION

[4]:
```
# Draw the tree
# To enlarge the generated tree: Shift + Right-click on the image
#                                   -> Open image in new tab
import matplotlib.pyplot as plt
plt.figure(figsize=(18,9), dpi=300)
tree.plot_tree(dtc, feature_names=X.columns.tolist())
plt.show()
```





Train_test_split (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
Model evaluation (scikit-learn): https://scikit-learn.org/stable/modules/model_evaluation.html

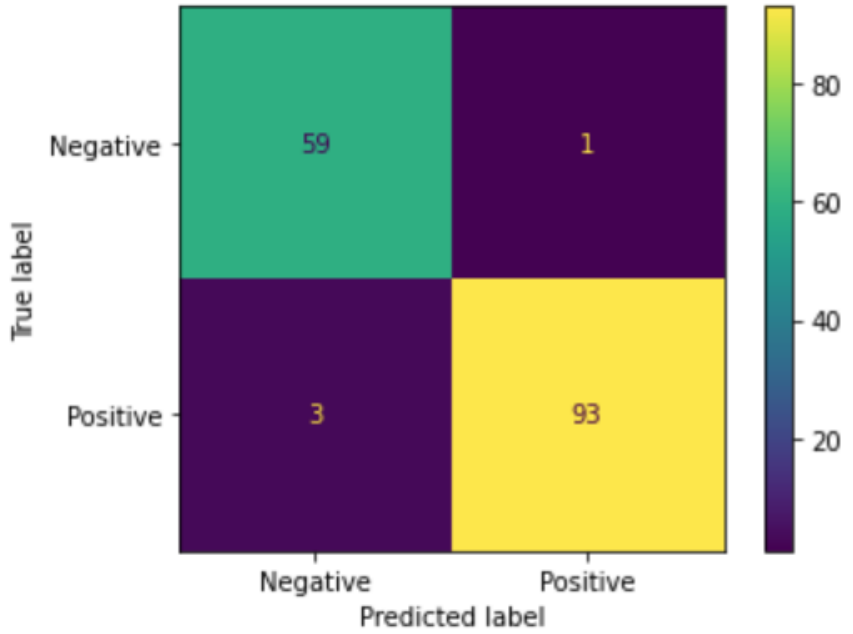
HOLDOUT EVALUATION

[5]:
```
from sklearn.model_selection import train_test_split
import sklearn.metrics as mt

# Split data into training/testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

dtc = tree.DecisionTreeClassifier().fit(X_train, y_train)

# Use test data to generate predictions
y_pred = dtc.predict(X_test)

# Compare predictions with ground truth
print(mt.accuracy_score(y_test, y_pred))
print(mt.classification_report(y_test, y_pred))
print(mt.confusion_matrix(y_test, y_pred))
```

```
0.9743589743589743

              precision    recall  f1-score   support

    Negative       0.95      0.98      0.97        60
    Positive       0.99      0.97      0.98        96

    accuracy                           0.97       156
   macro avg       0.97      0.98      0.97       156
weighted avg       0.97      0.97      0.97       156


[[59  1]
 [ 3 93]]
```

CONFUSION MATRIX (PLOT)

[6]:
```
# Plot the confusion matrix
import matplotlib.pyplot as plt
mt.plot_confusion_matrix(dtc, X_test, y_test)
plt.show()
```



Make_scorer (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html
Classification metrics (scikit-learn): https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics
Cross-validation (scikit-learn): https://scikit-learn.org/stable/modules/cross_validation.html

CROSS VALIDATION

[7]:
```
from sklearn.model_selection import cross_validate

# Define different scoring metric to be used
scorer = {
    # Define "positive" label for F-measure
    'f1': mt.make_scorer(mt.f1_score, pos_label="Positive"),
    'accuracy': 'accuracy'
    }

dtc = tree.DecisionTreeClassifier()

# Cross-validate with a decision tree classifier
scores = cross_validate(dtc, X, y, cv=10, scoring=scorer)

print(scores)
```

```
{'fit_time': array([0.00327539, 0.00288105, 0.00285363, 0.00267529, 0.00274968,
       0.00272894, 0.00272512, 0.00271916, 0.00260472, 0.00273967]), 'score_time': array
([0.00269461, 0.00218034, 0.00211716, 0.00224662, 0.00221848,
       0.00212145, 0.0023036 , 0.0021317 , 0.00210667, 0.0021112 ]), 'test_f1': array
([0.90625   , 0.9375    , 0.96774194, 1.        , 0.91803279,
       1.        , 0.98412698, 1.        , 1.        , 0.98461538]), 'test_accuracy': ar
ray([0.88461538, 0.92307692, 0.96153846, 1.        , 0.90384615,
       1.        , 0.98076923, 1.        , 1.        , 0.98076923])}
```

F1 Score of each fold

We can take an average as an overall result (try it as an exercise ☺)

Cross_validate (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html

K-FOLD… STRATIFIED!

```
dtc = tree.DecisionTreeClassifier()

# Cross-validate with a decision tree classifier
scores = cross_validate(dtc, X, y, cv=10, scoring=scorer)

print(scores)
```

cv : *int, cross-validation generator or an iterable, default=None*

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 5-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

For integer/None inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. In all other cases, `KFold` is used. These splitters are instantiated with `shuffle=False` so the splits will be the same across calls.

In "this" cross validation, we are using **stratified** 10-fold.

Grid search (scikit-learn): https://scikit-learn.org/stable/modules/grid_search.html
GridSearchCV (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

PARAMETER TUNING – GridSearchCV

[8]:
```
from sklearn.model_selection import GridSearchCV

# Setup the list of parameters to be tested
parameters = {'min_impurity_decrease': [0.05*i for i in range(3)],
              'criterion': ["gini", "entropy"]}

dtc = tree.DecisionTreeClassifier()

# Using accuracy as the score, 10-fold validation
gs_dtc = GridSearchCV(dtc, parameters, scoring="accuracy", cv=10)

# Specify the data
gs_dtc.fit(X_train, y_train)

print(gs_dtc.cv_results_)
```

This is also stratified

We are tuning parameters here only. We need to train the classifier using best parameters, and then evaluate **using a separate test set.**

GRID SEARCH RESULTS

6 sets of parameters, corresponding to the 6 values of each fold

```
{… 'params': [
{'criterion': 'gini', 'min_impurity_decrease': 0.0},
{'criterion': 'gini', 'min_impurity_decrease': 0.05},
{'criterion': 'gini', 'min_impurity_decrease': 0.1},
{'criterion': 'entropy', 'min_impurity_decrease': 0.0},
{'criterion': 'entropy', 'min_impurity_decrease': 0.05},
{'criterion': 'entropy', 'min_impurity_decrease': 0.1}
],
'split0_test_score': array([0.94594595, 0.83783784, 0.83783784, 0.91891892, 0.86486486, 0.83783784]),
'split1_test_score': array([0.94594595, 0.75675676, 0.81081081, 0.94594595, 0.75675676, 0.81081081]),
'split2_test_score': array([0.97297297, 0.75675676, 0.75675676, 0.97297297, 0.91891892, 0.75675676]),
'split3_test_score': array([0.81081081, 0.83783784, 0.83783784, 0.81081081, 0.81081081, 0.72972973]),
'split4_test_score': array([0.94444444, 0.77777778, 0.77777778, 0.94444444, 0.88888889, 0.77777778]),
'split5_test_score': array([0.91666667, 0.75      , 0.75      , 0.94444444, 0.86111111, 0.75      ]),
'split6_test_score': array([0.94444444, 0.83333333, 0.83333333, 0.91666667, 0.86111111, 0.83333333]),
'split7_test_score': array([0.97222222, 0.86111111, 0.86111111, 0.97222222, 0.86111111, 0.83333333]),
'split8_test_score': array([0.97222222, 0.88888889, 0.88888889, 0.97222222, 0.91666667, 0.88888889]),
'split9_test_score': array([0.97222222, 0.83333333, 0.83333333, 0.97222222, 0.88888889, 0.83333333]),
'mean_test_score': array([0.93978979, 0.81336336, 0.81876877, 0.93708709, 0.86291291, 0.80518018]),
'std_test_score': array([0.04641777, 0.04655042, 0.04263736, 0.04676617, 0.04623089, 0.04724036]),
'rank_test_score': array([1, 5, 4, 2, 3, 6], dtype=int32)}
```

Argmin (NumPy): https://numpy.org/doc/stable/reference/generated/numpy.argmin.html

GRID SEARCH RESULTS

```
'mean_test_score': array([0.93978979, 0.81336336, 0.81876877, 0.93708709, 0.86291291, 0.80518018]),
'std_test_score': array([0.04641777, 0.04655042, 0.04263736, 0.04676617, 0.04623089, 0.04724036]),
'rank_test_score': array([1, 5, 4, 2, 3, 6], dtype=int32)
```

The model using the first set of parameters have the best performance. We can build a model using this parameter set and evaluate with the test data.

To get the best parameters:

```
[9]: # Get the best parameters
     best_index = gs_dtc.cv_results_['rank_test_score'].argmin()
     best_param = gs_dtc.cv_results_['params'][best_index]
     print(best_param)
```

```
{'criterion': 'gini', 'min_impurity_decrease': 0.0}
```

To build the model using the best parameters:

```
[10]: dtc = tree.DecisionTreeClassifier(**best_param)
      dtc.fit(X_train, y_train)
      y_pred = dtc.predict(X_test)
      print(mt.confusion_matrix(y_test, y_pred))
```

```
[[47  3]
 [ 8 98]]
```

GridSearchCV (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

GRIDSEARCHCV – REFIT

In the previous slides, we have seen these typical steps:

- Use GridSearchCV to tune the parameters
- Obtain the best parameters
- Rebuild the model (decision tree) with the best parameters

For convenience, scikit-learn automatically refits the model (dtc) with the best parameters and stores the model in the GridSearchCV instance (gs_dtc).

[11]:
```
y_pred = gs_dtc.predict(X_test)
print(mt.confusion_matrix(y_test, y_pred))
```

```
[[47  3]
 [ 8 98]]
```

You can consider gs_dtc as the decision tree model (dtc) trained with the best parameters using X_train and y_train

Note that the result may be different due to the randomness in building the decision tree.

Nested cross validation: https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

PARAMETER TUNING + CROSS VALIDATION

[12]:
```
import numpy as np

dtc = tree.DecisionTreeClassifier()

gs_dtc = GridSearchCV(dtc, parameters, scoring="accuracy", cv=10, refit=True)
scores = cross_validate(gs_dtc, X, y, cv=10, scoring=scorer)

print("Accuracy:", np.mean(scores['test_accuracy']))
print("F1:", np.mean(scores['test_f1']))
print()
print(scores)
```

We are nesting the GridSearchCV into the cross_validate function, i.e., running cross validation on the decision tree whose parameters are tuned using GridSearchCV.

```
Accuracy: 0.9615384615384615
F1: 0.968239407523847

{'fit_time': array([0.24008036, 0.23771119, 0.23809075, 0.23865676, 0.23849177,
       0.2382319 , 0.23816228, 0.23904943, 0.23854494, 0.23868084]), 'score_time': array([0.0
0218964, 0.00215292, 0.00213742, 0.00212693, 0.00214124,
       0.00213552, 0.00213766, 0.00213099, 0.002141  , 0.00211787]), 'test_f1': array([0.9062
5   , 0.9375    , 0.96774194, 0.98412698, 0.91803279,
       1.        , 0.98412698, 1.        , 1.        , 0.98461538]), 'test_accuracy': array
([0.88461538, 0.92307692, 0.96153846, 0.98076923, 0.90384615,
       1.        , 0.98076923, 1.        , 1.        , 0.98076923])}
```

OTHER CLASSIFIERS

Nearest Neighbors

- https://scikit-learn.org/stable/modules/neighbors.html

Naïve Bayes

- https://scikit-learn.org/stable/modules/naive_bayes.html

SVM

- https://scikit-learn.org/stable/modules/svm.html

EXPLAINABLE AI

(Self-study)

EXPLAINABLE AI

Explainable AI provides some explanation on how the machine learning models perform reasoning. It allows humans to interpret the results and make the results more trustworthy. For example, if a patient is diagnosed of a disease by the machine, the machine should also provide the reasons as the supporting evidence, such as the symptoms of the patient or the irregularities in the medical examination report, to make the prediction more convincing.

A single decision tree by itself provides some degree of explainability as the decision process is clearly specified. For a more complex model, such as random forest, it is unclear to know to what extent each feature contributes to the final prediction.

Some measures and tools that can help explain a model:

- Feature importance scores of a decision tree
- LIME: Local interpretable model-agnostic explanations
- SHAP: Shapley additive explanations (this tutorial)

Lundberg & Lee. "A unified approach to interpreting model predictions." In NeurIPS, 2017.

SHAP: SHAPLEY ADDITIVE EXPLANATIONS

SHAP computes feature attributions based on Shapley regression values in game theory.

To compute the effect of a feature f_i from the feature set $\mathcal{F}$, for each subset of features $S \subseteq \mathcal{F} \setminus \{f_i\}$, we train two models M_S and $M_{S \cup \{f_i\}}$. The former is trained with features in S and the latter includes feature f_i as well. Given an input x, the feature attribution score (ϕ_i) of the feature f_i is computed by

$$\phi_i = \sum_{S \subseteq \mathcal{F} \setminus \{f_i\}} \frac{|S|!\,(|\mathcal{F}| - |S| - 1)!}{|\mathcal{F}|} \left[M_{S \cup \{f_i\}}\left(x_{S \cup \{f_i\}}\right) - M_S(x_S) \right]$$

Values of the input features in the set S

Intuitively, the feature attribution ϕ_i is the weighted sum of the differences in the prediction results for the input x given by the classifiers constructed with and without using feature f_i.

INSTALL PACKAGES

In the terminal with comp7103-venv activated, run:

```
pip install -U shap
```

The documentation of the SHAP package can be found here:

https://shap.readthedocs.io/en/latest/

LOAD DATA & DECISION TREE BUILDING

In this tutorial, we demonstrate SHAP with the decision tree built earlier. It can also be applied to other classification/regression models.

Create a new notebook and name it `tutorial3-shap.ipynb`

Load the diabetes dataset and build the decision tree:

[1]:
```
from sklearn.model_selection import train_test_split
from sklearn import tree
import sklearn.metrics as mt
import pandas as pd
import numpy as np

df = pd.read_csv("diabetes_data_upload.csv")

X = pd.get_dummies(df.drop(columns="class"))
y = df["class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

dtc = tree.DecisionTreeClassifier().fit(X_train, y_train)
```

INITIALIZE

For SHAP to display properly in Jupyter notebook, you need to initialize javascript:

```
[2]: import shap
     shap.initjs()
```

The js icon will appear upon successful initialization:

You may ignore tqdm-related messages.

EXPLAINER

Create the explainer. The `shap.explainers.Tree` uses SHAP to explain the output of tree models (including tree ensembles).

```
[3]: explainer = shap.explainers.Tree(dtc, X_train)
```

The model that we want to explain

The background dataset required by SHAP to approximate feature attribution scores

Note that computing the exact Shapley values with the feature set $\mathcal{F}$ requires building $2^{|\mathcal{F}|}$ models (one for each subset of features), which is computationally expensive. The SHAP algorithm implements some approximation methods to increase the efficiency (details omitted here).

SHAP VALUES

Let's get the first example in the test set and see the SHAP values:

```
[4]: input_X = np.array(X_test.iloc[0])
     output_y = y_test.iloc[0]
     print(input_X)
     print(output_y)
```

input_X

```
[54  0  1  0  1  0  1  1  0  1  0  1  0  1  0  0  1  1  0  1  0  1  0  1
  0  1  0  1  0  1  0]
Positive
```

output_y

```
[5]: shap_values = explainer.shap_values(input_X)
```

SHAP VALUES

```
X = pd.get_dummies(df.drop(columns="class"))
```

	Age	Gender_Female	Gender_Male	Polyuria_No	Polyuria_Yes	Polydipsia_No	Polydip
0	40	0	1	1	0	0	
1	58	0	1	1	0	1	

SHAP value of input_X w.r.t the first feature

```
[6]: shap_values
```

```
[6]: [array([-0.001     ,  0.        ,  0.01657143,  0.        , -0.19192857,
             -0.15759524, -0.038     ,  0.        ,  0.0025    ,  0.0047381 ,
              0.        ,  0.        ,  0.        ,  0.003     ,  0.00783333,
             -0.0047619 ,  0.00983333, -0.038     ,  0.        ,  0.00566667,
              0.01      ,  0.        , -0.00542857,  0.        ,  0.        ,
              0.        , -0.03416667, -0.0072619 ,  0.        ,  0.        ,
             -0.002     ]),
      array([ 0.001     ,  0.        , -0.01657143,  0.        ,  0.19192857,
              0.15759524,  0.038     ,  0.        , -0.0025    , -0.0047381 ,
              0.        ,  0.        ,  0.        , -0.003     , -0.00783333,
              0.0047619 , -0.00983333,  0.038     ,  0.        , -0.00566667,
             -0.01      ,  0.        ,  0.00542857,  0.        ,  0.        ,
              0.        ,  0.03416667,  0.0072619 ,  0.        ,  0.        ,
              0.002     ])]
```

Why is there a list of two arrays?

(Answered in the next slide)

SHAP VALUES

```
[7]: dtc.classes_
```

```
[7]: array(['Negative', 'Positive'], dtype=object)
```

Recall that the decision tree classifier outputs one of the two classes: “Negative” or “Positive”. Hence, the first array (index [0]) in the list [-0.001, 0, 0.01657, …] corresponds to the first (i.e., Negative) class, while the second array (index [1]) [0.001, 0, -0.1657] corresponds to the second (i.e., Positive) class.

VISUALIZATION: FORCE PLOT

The force plot help visualize the features that makes the input (input_X) to be predicted "Positive". In other words, we want to visualize what features makes the predictor to assign the value 1 to the "Positive" class:

The "base" value (classifier's output without considering the input)

SHAP value of input_X for the "Positive" (index [1]) class

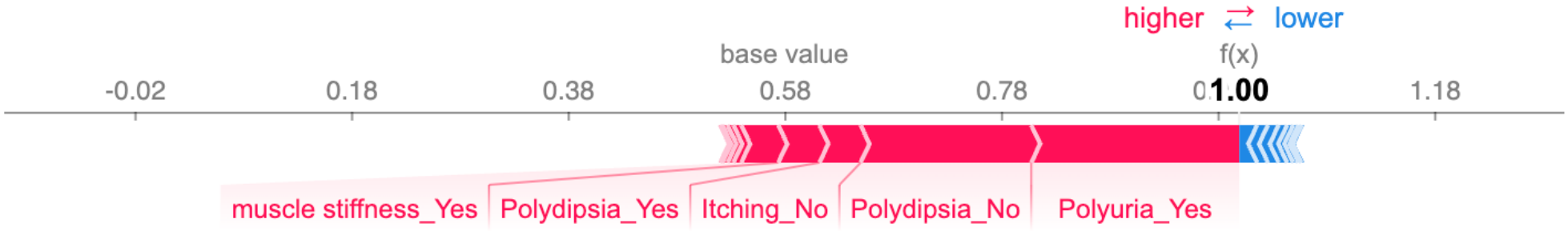
```
[8]: shap.force_plot(explainer.expected_value[1], shap_values[1],
                    feature_names=X_train.columns)
```

Feature names (for better visualization)

VISUALIZATION: FORCE PLOT

```
[8]: shap.force_plot(explainer.expected_value[1], shap_values[1],
                     feature_names=X_train.columns)
```

[8]:



The plot shows what features make the prediction positive. For example, Polyuria_Yes has the highest positive effect on the prediction, followed by Polydispia_No.

Slicing (numPy): https://numpy.org/doc/stable/user/basics.indexing.html#slicing-and-striding

VISUALIZATION: BEE SWARM PLOT

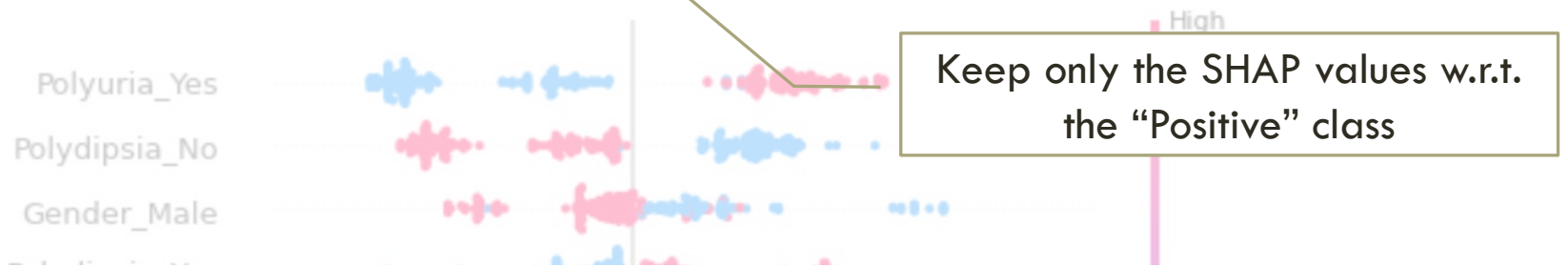
The bee swarm plot visualizes the SHAP values w.r.t. each feature in the dataset. In this example, we visualize the training set. (You can try it with the test set or the entire dataset.)

Get the SHAP values for all instances in the training set

Keep the SHAP values for all instances and all features

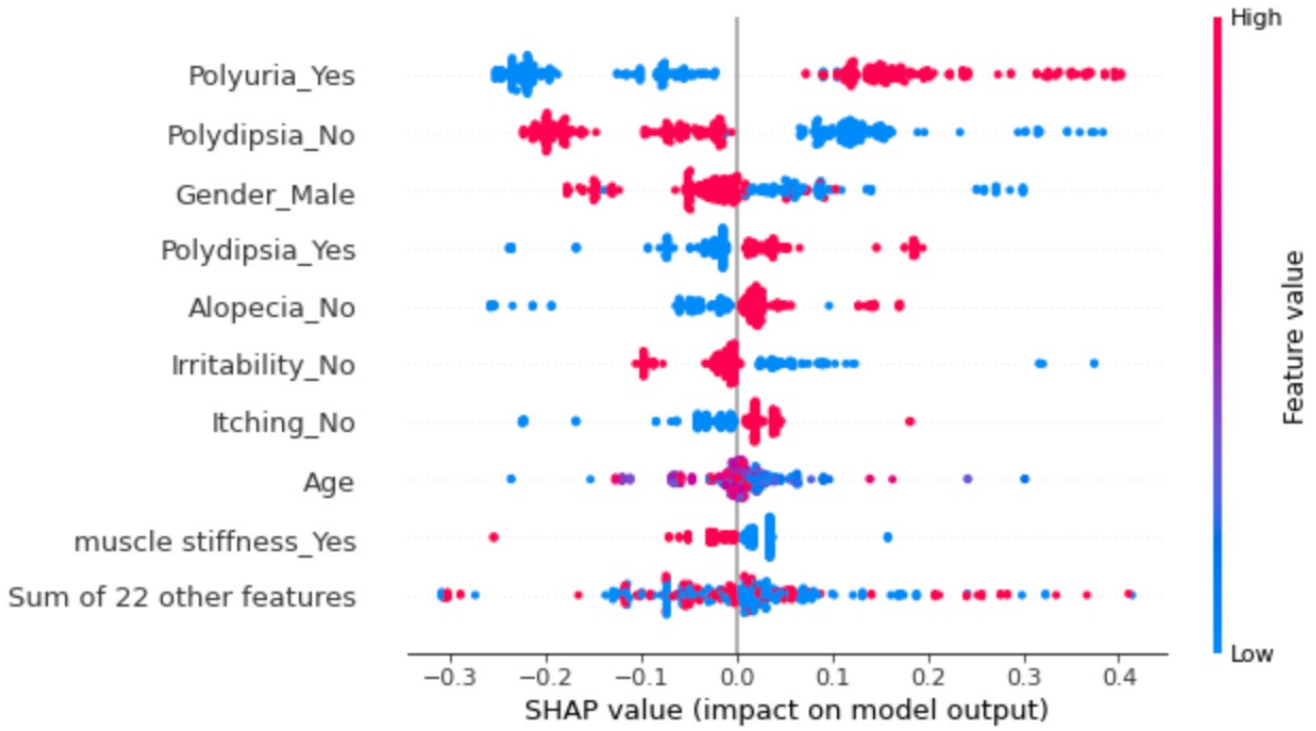
```
[9]: shap.plots.beeswarm(explainer(X_train)[:, :, 1])
```

Keep only the SHAP values w.r.t. the "Positive" class



VISUALIZATION: BEE SWARM PLOT

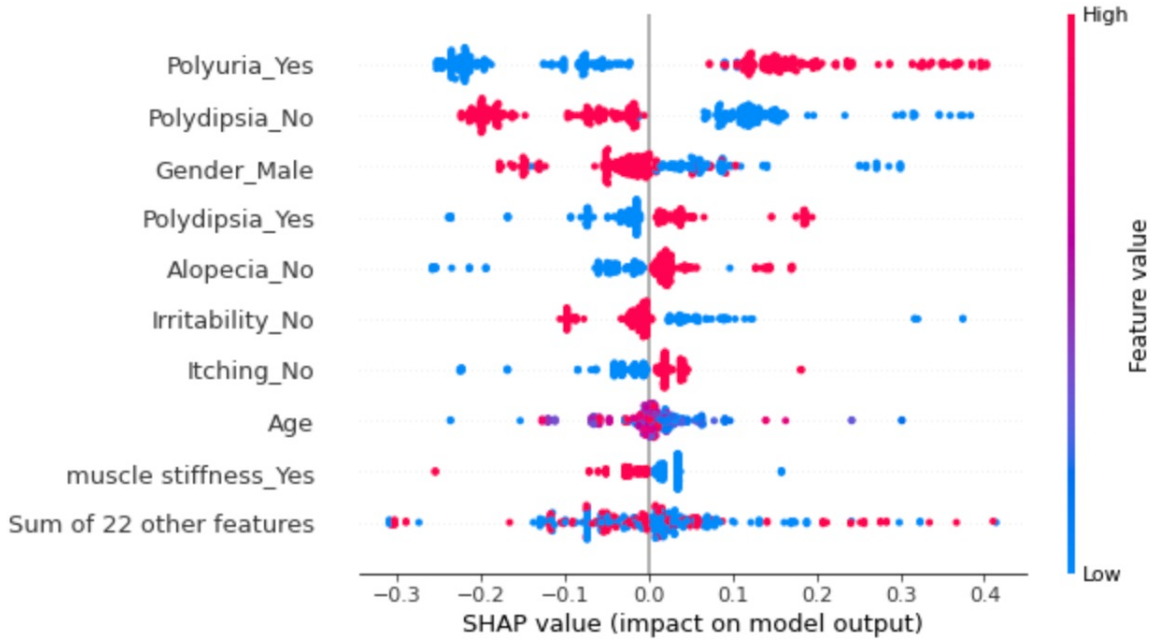
```
[9]: shap.plots.beeswarm(explainer(X_train)[:, :, 1])
```



In each row, each dot represents an input instance.

- The color shows the feature value. For example, a red dot in the first row shows that the input instance has a high value (for a binary feature, it would be the value of 1) in the Polyuria_Yes feature.
- The position of the dot shows the SHAP value of the instance w.r.t. the feature. For example, the right-most dot in the first row shows that the input instance has a high (roughly 0.4) SHAP value on the Polyuria_Yes feature.

VISUALIZATION: BEE SWARM PLOT



What can we conclude from the figure?

- A positive Polyuria_Yes has a high impact on the model for making a positive prediction as the red dots are on the right. Likewise, a negative Polyuria_Yes has a high impact on the model for making a negative prediction as the blue dots concentrate on the left.
- What else?