Import Libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns #for plotting
from sklearn.ensemble import RandomForestClassifier #for the model

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.tree import export_graphviz #plot tree
from sklearn.metrics import roc_curve, auc #for model evaluation
from sklearn.metrics import classification_report #for model evaluation
from sklearn.metrics import confusion_matrix #for model evaluation
from sklearn.model_selection import train_test_split #for data splitting
import eli5 #for purmutation importance
from eli5.sklearn import PermutationImportance
import shap #for SHAP values
from pdpbox import pdp, info_plots #for partial plots
np.random.seed(123) #ensure reproducibility
import pickle
from sklearn.model_selection import KFold, train_test_split, RandomizedSearchCV

from sklearn.metrics import auc, accuracy_score, recall_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn import svm, metrics, clone
import scipy



pd.options.mode.chained_assignment = None  #hide any pandas warnings
```

Preprocessing data

In [ ]:
```
dt = pd.read_csv("raw-heart-data.csv")
```

Watch 10 first observations

In [ ]:
```
dt.head(10)
```

Out[ ]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope
0	1	63	1	typical	145	233	1	2	150	0	2.3	3
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	3
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	1
5	6	56	1	nontypical	120	236	0	0	178	0	0.8	1
6	7	62	0	asymptomatic	140	268	0	2	160	0	3.6	3
7	8	57	0	asymptomatic	120	354	0	0	163	1	0.6	1
8	9	63	1	asymptomatic	130	254	0	2	147	0	1.4	2
9	10	53	1	asymptomatic	140	203	1	2	155	1	3.1	3

Drop a column

In [ ]:
```
dt.drop('Unnamed: 0', axis=1, inplace=True)
```

In [ ]:
```
dt
```

Out[ ]:

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	
0	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	
1	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	
2	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	re
3	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	
4	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
298	45	1	typical	110	264	0	0	132	0	1.2	2	0.0	re
299	68	1	asymptomatic	144	193	1	0	141	0	3.4	2	2.0	re
300	57	1	asymptomatic	130	131	0	0	115	1	1.2	2	1.0	re
301	57	0	nontypical	130	236	0	2	174	0	0.0	2	1.0	
302	38	1	nonanginal	138	175	0	0	173	0	0.0	1	NaN	

303 rows × 14 columns

Rename the column

In [ ]:
```
dt.columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_pressure', 'cholesterol'
       'exercise_induced_angina', 'st_depression', 'st_slope', 'num_major_vessels', 't
dt
```

Out[ ]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_ecg	ma
0	63	1	typical	145	233	1	2	
1	67	1	asymptomatic	160	286	0	2	
2	67	1	asymptomatic	120	229	0	2	
3	37	1	nonanginal	130	250	0	0	
4	41	0	nontypical	130	204	0	2	
...	...	...	...	...	...	...	...	
298	45	1	typical	110	264	0	0	
299	68	1	asymptomatic	144	193	1	0	
300	57	1	asymptomatic	130	131	0	0	
301	57	0	nontypical	130	236	0	2	
302	38	1	nonanginal	138	175	0	0	

303 rows × 14 columns

About the data

It's a clean, easy to understand set of data. However, the meaning of some of the column headers are not obvious. Here's what they mean,

- age: The person's age in years
- sex: The person's sex (1 = male, 0 = female)
- cp: The chest pain experienced (Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic)
- trestbps: The person's resting blood pressure (mm Hg on admission to the hospital)
- chol: The person's cholesterol measurement in mg/dl
- fbs: The person's fasting blood sugar (> 120 mg/dl, 1 = true; 0 = false)
- restecg: Resting electrocardiographic measurement (0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)
- thalach: The person's maximum heart rate achieved
- exang: Exercise induced angina (1 = yes; 0 = no)
- oldpeak: ST depression induced by exercise relative to rest ('ST' relates to positions on the ECG plot. See more here)
- slope: the slope of the peak exercise ST segment (Value 1: upsloping, Value 2: flat, Value 3: downsloping)
- ca: The number of major vessels (0-3)

- thal: A blood disorder called thalassemia (3 = normal; 6 = fixed defect; 7 = reversable defect)
- target: Heart disease (0 = no, 1 = yes)

Change the values of the categorical variables, to improve the interpretation later on

In [ ]:
```
dt['sex'][dt['sex'] == 0] = 'female'
dt['sex'][dt['sex'] == 1] = 'male'

dt['chest_pain_type'][dt['chest_pain_type'] == 1] = 'typical angina'
dt['chest_pain_type'][dt['chest_pain_type'] == 2] = 'atypical angina'
dt['chest_pain_type'][dt['chest_pain_type'] == 3] = 'non-anginal pain'
dt['chest_pain_type'][dt['chest_pain_type'] == 4] = 'asymptomatic'

dt['fasting_blood_sugar'][dt['fasting_blood_sugar'] == 0] = 'lower than 120mg/ml'
dt['fasting_blood_sugar'][dt['fasting_blood_sugar'] == 1] = 'greater than 120mg/ml'

dt['rest_ecg'][dt['rest_ecg'] == 0] = 'normal'
dt['rest_ecg'][dt['rest_ecg'] == 1] = 'ST-T wave abnormality'
dt['rest_ecg'][dt['rest_ecg'] == 2] = 'left ventricular hypertrophy'

dt['exercise_induced_angina'][dt['exercise_induced_angina'] == 0] = 'no'
dt['exercise_induced_angina'][dt['exercise_induced_angina'] == 1] = 'yes'

dt['st_slope'][dt['st_slope'] == 1] = 'upsloping'
dt['st_slope'][dt['st_slope'] == 2] = 'flat'
dt['st_slope'][dt['st_slope'] == 3] = 'downsloping'

dt['thalassemia'][dt['thalassemia'] == 'fixed'] = 'fixed defect'
dt['thalassemia'][dt['thalassemia'] == 'reversable'] = 'reversable defect'

dt['target'][dt['target'] == 'Yes'] = 1
dt['target'][dt['target'] == 'No'] = 0
```

In [ ]:
```
dt
```

Out[ ]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_ec
0	63	male	typical	145	233	greater than 120mg/ml	le ventricula hypertroph
1	67	male	asymptomatic	160	286	lower than 120mg/ml	le ventricula hypertroph
2	67	male	asymptomatic	120	229	lower than 120mg/ml	le ventricula hypertroph
3	37	male	nonanginal	130	250	lower than 120mg/ml	norma
4	41	female	nontypical	130	204	lower than 120mg/ml	le ventricula hypertroph
...	...	...	...	...	...	...	
298	45	male	typical	110	264	lower than 120mg/ml	norma
299	68	male	asymptomatic	144	193	greater than 120mg/ml	norma
300	57	male	asymptomatic	130	131	lower than 120mg/ml	norma
301	57	female	nontypical	130	236	lower than 120mg/ml	le ventricula hypertroph
302	38	male	nonanginal	138	175	lower than 120mg/ml	norma

303 rows × 14 columns

Eliminate NA values

In [ ]:
```
dt = dt.dropna()
```

In [ ]:
```
dt
```

Out[ ]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	rest_ec
0	63	male	typical	145	233	greater than 120mg/ml	le ventricula hypertroph
1	67	male	asymptomatic	160	286	lower than 120mg/ml	le ventricula hypertroph
2	67	male	asymptomatic	120	229	lower than 120mg/ml	le ventricula hypertroph
3	37	male	nonanginal	130	250	lower than 120mg/ml	norma
4	41	female	nontypical	130	204	lower than 120mg/ml	le ventricula hypertroph
...	...	...	...	...	...	...	
297	57	female	asymptomatic	140	241	lower than 120mg/ml	norma
298	45	male	typical	110	264	lower than 120mg/ml	norma
299	68	male	asymptomatic	144	193	greater than 120mg/ml	norma
300	57	male	asymptomatic	130	131	lower than 120mg/ml	norma
301	57	female	nontypical	130	236	lower than 120mg/ml	le ventricula hypertroph

297 rows × 14 columns

Check data types

In [ ]:
```
dt.dtypes
```

Out[ ]:
```
age                          int64
sex                         object
chest_pain_type             object
resting_blood_pressure       int64
cholesterol                  int64
fasting_blood_sugar         object
rest_ecg                    object
max_heart_rate_achieved      int64
exercise_induced_angina     object
st_depression              float64
st_slope                    object
num_major_vessels          float64
thalassemia                 object
target                      object
dtype: object
```

Change Data type of 2 columns: num_major_vessels & target

In [ ]:
```
dt['num_major_vessels'] = dt['num_major_vessels'].astype('int64')
dt['target'] = dt['target'].astype('int64')
```

In [ ]:
```
dt.dtypes
```

Out[ ]:
```
age                          int64
sex                         object
chest_pain_type             object
resting_blood_pressure       int64
cholesterol                  int64
fasting_blood_sugar         object
rest_ecg                    object
max_heart_rate_achieved      int64
exercise_induced_angina     object
st_depression              float64
st_slope                    object
num_major_vessels            int64
thalassemia                 object
target                       int64
dtype: object
```

In [ ]:
```
dt['chest_pain_type'].unique()
```

Out[ ]:
```
array(['typical', 'asymptomatic', 'nonanginal', 'nontypical'],
      dtype=object)
```

For the categorical varibles, we need to create dummy variables. I'm also going to drop the first category of each. For example, rather than having 'male' and 'female', we'll have 'male' with values of 0 or 1 (1 being male, and 0 therefore being female).

In [ ]:
```
dt = pd.get_dummies(dt, drop_first=True)
```

In [ ]:
```
dt
```

Out[ ]:

	age	resting_blood_pressure	cholesterol	max_heart_rate_achieved	st_depression	num_major_vess
0	63	145	233	150	2.3	
1	67	160	286	108	1.5	
2	67	120	229	129	2.6	
3	37	130	250	187	3.5	
4	41	130	204	172	1.4	
...	...	...	...	...	...	
297	57	140	241	123	0.2	
298	45	110	264	132	1.2	
299	68	144	193	141	3.4	
300	57	130	131	115	1.2	
301	57	130	236	174	0.0	

297 rows × 19 columns

Rename columns name: Chest pain type

In [ ]:
```
dt.rename(columns = {'chest_pain_type_typical':'chest_pain_type_typical angina', 'ches
                     'chest_pain_type_nonanginal':'chest_pain_type_non-angina
```

In [ ]:
```
dt.head()
```

Out[ ]:

	age	resting_blood_pressure	cholesterol	max_heart_rate_achieved	st_depression	num_major_vessels
0	63	145	233	150	2.3	0
1	67	160	286	108	1.5	3
2	67	120	229	129	2.6	2
3	37	130	250	187	3.5	0
4	41	130	204	172	1.4	0

In [ ]:
```
dt
```

Out[ ]:

	age	resting_blood_pressure	cholesterol	max_heart_rate_achieved	st_depression	num_major_vess
0	63	145	233	150	2.3	
1	67	160	286	108	1.5	
2	67	120	229	129	2.6	
3	37	130	250	187	3.5	
4	41	130	204	172	1.4	
...	...	...	...	...	...	
297	57	140	241	123	0.2	
298	45	110	264	132	1.2	
299	68	144	193	141	3.4	
300	57	130	131	115	1.2	
301	57	130	236	174	0.0	

297 rows × 19 columns

Save processed data

In [ ]:
```
dt.to_csv('processed-heart-data.csv', sep=',', index=False, encoding='utf-8')
```

The Model

Apply Random Forest model

Split the data into training and testing sets

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(dt.drop('target', 1), dt['target']
splits = [X_train, X_test, y_train, y_test]
```

In [ ]:
```
print("Training data size:", len(X_train))
print("Test data size:", len(X_test))
```

```
Training data size: 237
Test data size: 60
```

Tunning hyperparameters

In [ ]:
```
rf_tuning = RandomForestClassifier()

param_dist = {
    'bootstrap': [True, False],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
    'max_features': ['auto', 'sqrt'],
```

```
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [130, 180, 230]
}

random_search_rf = RandomizedSearchCV(estimator=rf_tuning,
                                      param_distributions=param_dist,
                                      cv=5,
                                      n_iter=100,
                                      verbose=2,
                                      n_jobs=-1)

random_search_rf.fit(X_train, y_train)
random_search_rf.best_params_
```

Out[ ]:
```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
{'n_estimators': 230,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 70,
 'bootstrap': True}
```

In [ ]:
```
save_model_path = 'D:\Data\Life Sciences\model'
```

In [ ]:
```
def save_models(models_dict, save_folder_path, running_count):
  for model in models_dict:
    filename = model['label'] + '_{}.pkl'.format(str(running_count))
    with open(save_model_path + '/' + filename, 'wb') as f:
      pickle.dump(model, f)
```

In [ ]:
```
model = RandomForestClassifier(**random_search_rf.best_params_)
model.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier(max_depth=70, min_samples_split=5, n_estimators=230)
```

In [ ]:
```
save_models(models_dict=[{"label": "RF", "model": model}], save_folder_path=save_model
```

Plot the consequent decision tree, to see what it's doing

In [ ]:
```
estimator = model.estimators_[0]
feature_names = [i for i in X_train.columns]

y_train_str = y_train.astype('str')
y_train_str[y_train_str == '0'] = 'no disease'
y_train_str[y_train_str == '1'] = 'disease'
y_train_str = y_train_str.values
```

Add Graphwiz to PATH for current user

In [ ]:
```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'
```

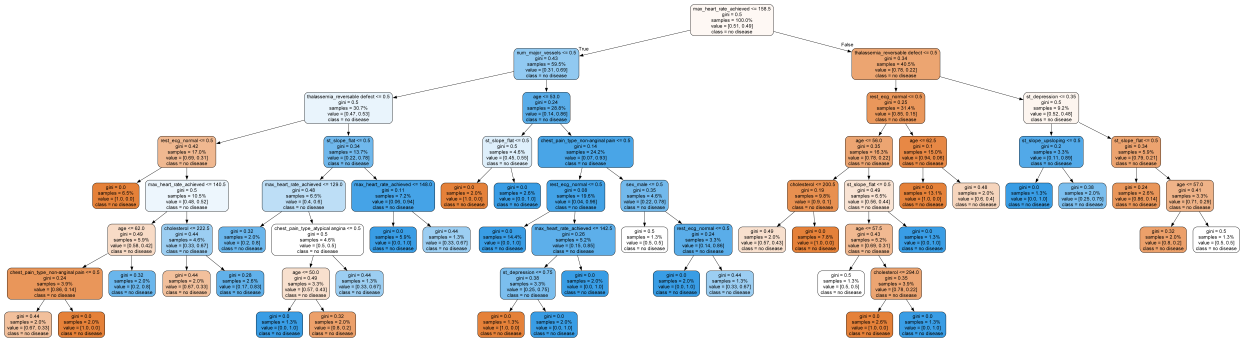
In [ ]:
```
export_graphviz(estimator,
                out_file='tree.dot',
                max_depth = 100000,
```

```
                feature_names = feature_names,
                class_names = y_train_str,
                rounded = True,
                proportion = True,
                label='all',
                precision = 2,
                filled = True)

from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

from IPython.display import Image
Image(filename = 'tree.png')
```

Out[ ]:



Evaluate the model RF model

Diagnostic tests are often sold, marketed, cited and used with **sensitivity** and **specificity** as the headline metrics. Sensitivity and specificity are defined as

$$Sensitivity = \frac{TruePositives}{TruePositives + FalseNegatives}$$

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives}$$

Another common metric is the Area Under the Curve, or AUC. This is a convenient way to capture the performance of a model in a single number, although it's not without certain issues. As a rule of thumb, an AUC can be classed as follows,

- 0.90 - 1.00 = excellent
- 0.80 - 0.90 = good
- 0.70 - 0.80 = fair
- 0.60 - 0.70 = poor
- 0.50 - 0.60 = fail

Let's see what the above ROC gives us,

In [ ]:
```
def model_performance(ml_model, test_x, test_y, verbose=True):
    """
    Helper function to calculate model performance

    Parameters
```

```
    ----------
    ml_model: sklearn model object
        The machine learning model to train.
    test_x: list
        Molecular fingerprints for test set.
    test_y: list
        Associated activity labels for test set.
    verbose: bool
        Print performance measure (default = True)

    Returns
    -------
    tuple:
        Accuracy, sensitivity, specificity, auc on test set.
    """

    # Prediction probability on test set
    test_prob = ml_model.predict_proba(test_x)[:, 1]

    # Prediction class on test set
    test_pred = ml_model.predict(test_x)

    # Performance of model on test set
    accuracy = accuracy_score(test_y, test_pred)
    sens = recall_score(test_y, test_pred)
    spec = recall_score(test_y, test_pred, pos_label=0)
    auc = roc_auc_score(test_y, test_prob)

    if verbose:
        # Print performance results
        # NBVAL_CHECK_OUTPUT        print(f"Accuracy: {accuracy:.2}")
        print(f"Sensitivity: {sens:.2f}")
        print(f"Specificity: {spec:.2f}")
        print(f"AUC: {auc:.2f}")

    return accuracy, sens, spec, auc
```

In [ ]:

```
def model_training_and_validation(ml_model, name, splits, verbose=True):
    """
    Fit a machine learning model on a random train-test split of the data
    and return the performance measures.

    Parameters
    ----------
    ml_model: sklearn model object
        The machine learning model to train.
    name: str
        Name of machine learning algorithm: RF, SVM, ANN
    splits: list
        List of desciptor and label data: train_x, test_x, train_y, test_y.
    verbose: bool
        Print performance info (default = True)

    Returns
    -------
    tuple:
        Accuracy, sensitivity, specificity, auc on test set.

    """
```

```
    train_x, test_x, train_y, test_y = splits

    # Fit the model
    ml_model.fit(train_x, train_y)

    # Calculate model performance results
    accuracy, sens, spec, auc = model_performance(ml_model, test_x, test_y, verbose)

    return accuracy, sens, spec, auc
```

In [ ]:
```
performance_measures = model_training_and_validation(model, "RF", splits)
```

```
Sensitivity: 0.84
Specificity: 0.89
AUC: 0.94
```

Plot feature importance

In [ ]:
```
importance = pd.DataFrame({'feature':X_train.columns, 'importance': model.feature_impc
print(importance.sort_values('importance', ascending=False))
```

```
                                  feature  importance
3                   max_heart_rate_achieved    0.144860
4                             st_depression    0.112260
5                          num_major_vessels    0.112173
16                        thalassemia_normal    0.103377
0                                      age    0.085238
17            thalassemia_reversable defect    0.071351
1                    resting_blood_pressure    0.066968
2                               cholesterol    0.063467
13              exercise_induced_angina_yes    0.062114
15                        st_slope_upsloping    0.031283
6                                 sex_male    0.030603
14                             st_slope_flat    0.028976
7          chest_pain_type_non-anginal pain    0.028625
9            chest_pain_type_typical angina    0.014429
12                           rest_ecg_normal    0.013069
11    rest_ecg_left ventricular hypertrophy    0.011956
8           chest_pain_type_atypical angina    0.011058
10  fasting_blood_sugar_lower than 120mg/ml    0.008193
```

Plot permutation importance

In [ ]:
```
perm = PermutationImportance(model, random_state=1).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Out[ ]:

Weight	Feature
0.0600 ± 0.0340	num_major_vessels
0.0367 ± 0.0389	thalassemia_normal
0.0267 ± 0.0267	st_depression
0.0233 ± 0.0163	rest_ecg_normal
0.0233 ± 0.0400	st_slope_upsloping
0.0200 ± 0.0133	fasting_blood_sugar_lower than 120mg/ml
0.0200 ± 0.0573	exercise_induced_angina_yes
0.0167 ± 0.0365	st_slope_flat
0.0167 ± 0.0211	rest_ecg_left ventricular hypertrophy
0.0167 ± 0.0298	thalassemia_reversable defect
0.0167 ± 0.0298	sex_male
0.0067 ± 0.0267	resting_blood_pressure
0.0067 ± 0.0267	max_heart_rate_achieved
0.0033 ± 0.0133	chest_pain_type_typical angina
0.0000 ± 0.0298	chest_pain_type_non-anginal pain
0 ± 0.0000	chest_pain_type_atypical angina
-0.0033 ± 0.0133	cholesterol
-0.0200 ± 0.0249	age

SVM Model

Tuning hyperparameters

In [ ]:
```
# Create a base model
svm_tuning = svm.SVC(random_state=22, probability=True)

param_dist = {
    'C': scipy.stats.expon(scale=100),
    'gamma': scipy.stats.expon(scale=.1),
    'kernel': ['rbf']
}

# Instantiate the grid search model
random_search_SVM = RandomizedSearchCV(svm_tuning,
                                       param_distributions=param_dist,
                                       cv=5,
                                       verbose=2,
                                       n_jobs=-1,
                                       n_iter=100,
                                       random_state=1)

# Fit the grid search to the data
random_search_SVM.fit(X_train, y_train)
random_search_SVM.best_params_
```

Out[ ]:
```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
{'C': 2.0080402388700334, 'gamma': 0.002656061777584434, 'kernel': 'rbf'}
```

In [ ]:
```
model_SVM = svm.SVC(random_state=22, probability=True, **random_search_SVM.best_params
```

In [ ]:
```
performance_measures = model_training_and_validation(model_SVM, "SVM", splits)
```

```
Sensitivity: 0.68
Specificity: 0.54
AUC: 0.65
```