

Project 3a

The final part of the project will ask you to perform your own data science project to classify a new dataset.

Submission Details

Project is due June 14th at 11:59 pm (Friday Midnight). To submit the project, please save the notebook as a pdf file and submit the assignment via Gradescope. In addition, make sure that all figures are legible and sufficiently large. For best pdf results, we recommend printing the notebook using *LATEX*

Loading Essentials and Helper Functions

```
In [ ]: # fix for windows memory leak with MKL
import os
import platform

if platform.system() == "Windows":
    os.environ["OMP_NUM_THREADS"] = "2"
```

```
In [ ]: # import libraries
import time
import random
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph

# Sklearn classes
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    GridSearchCV,
    KFold,
)
from sklearn import metrics
from sklearn.metrics import confusion_matrix, silhouette_score
import sklearn.metrics.cluster as smc
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import (
    StandardScaler,
    OneHotEncoder,
    LabelEncoder,
    MinMaxScaler,
)
```

```

from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn import tree
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_blobs

from helper import (
    draw_confusion_matrix,
    heatmap,
    make_meshgrid,
    plot_contours,
    draw_contour,
)

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Sets random seed for reproducibility
SEED = 42
random.seed(SEED)

```

Background: Dataset Information (Recap)

For this exercise we will be using a subset of the UCI Heart Disease dataset, leveraging the fourteen most commonly used attributes. All identifying information about the patient has been scrubbed. You will be asked to classify whether a **patient is suffering from heart disease** based on a host of potential medical factors.

The dataset includes 14 columns. The information provided by each column is as follows:

- **age:** Age in years
- **sex:** (male/female)
- **cp:** Chest pain type (0 = asymptomatic; 1 = atypical angina; 2 = non-anginal pain; 3 = typical angina)
- **trestbps:** Resting blood pressure (in mm Hg on admission to the hospital)
- **chol:** cholesterol in mg/dl
- **fbs** Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- **restecg:** Resting electrocardiographic results (0= showing probable or definite left ventricular hypertrophy by Estes' criteria; 1 = normal; 2 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV))
- **thalach:** Maximum heart rate achieved
- **exang:** Exercise induced angina (1 = yes; 0 = no)
- **oldpeak:** Depression induced by exercise relative to rest
- **slope:** The slope of the peak exercise ST segment (0 = downsloping; 1 = flat; 2 = upsloping)
- **ca:** Number of major vessels (0-3) colored by flourosopy
- **thal:** 1 = normal; 2 = fixed defect; 7 = reversable defect

- sick: Indicates the presence of Heart disease (True = Disease; False = No disease)

Preprocess Data

This part is done for you since you would have already completed it in project 2. Use the train, target, test, and target_test for all future parts. We also provide the column names for each transformed column for future use.

```
In [ ]: # Preprocess Data

# Load Data
data = pd.read_csv("datasets/heartdisease.csv")

# Transform target feature into numerical
le = LabelEncoder()
data["target"] = le.fit_transform(data["sick"])
data["sex"] = le.fit_transform(data["sex"])
data = data.drop(["sick"], axis=1)

# Split target and data
y = data["target"]
x = data.drop(["target"], axis=1)

# Train test split
# 40% in test data as was in project 2
train_raw, test_raw, target, target_test = train_test_split(
    x, y, test_size=0.4, stratify=y, random_state=0
)

# Feature Transformation
# This is the only change from project 2 since we replaced standard scaler to
# This was done to ensure that the numerical features were still of the same
# as the one hot encoded features
num_pipeline = Pipeline([("minmax", MinMaxScaler())])

heart_num = train_raw.drop(
    ["sex", "cp", "fbs", "restecg", "exang", "slope", "ca", "thal"], axis=1
)
numerical_features = list(heart_num)
categorical_features = ["sex", "cp", "fbs", "restecg", "exang", "slope", "ca", "thal"]

full_pipeline = ColumnTransformer(
    [
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(categories="auto"), categorical_features),
    ]
)

# Transform raw data
train = full_pipeline.fit_transform(train_raw)
test = full_pipeline.transform(test_raw) # Note that there is no fit calls
```

```
# Extracts features names for each transformed column
feature_names = full_pipeline.get_feature_names_out(list(x.columns))
```

In []: `print("Column names after transformation by pipeline: ", feature_names)`

```
Column names after transformation by pipeline: ['num_age' 'num_trestbps'
 'num_chol' 'num_thalach' 'num_oldpeak'
 'cat_sex_0' 'cat_sex_1' 'cat_cp_0' 'cat_cp_1' 'cat_cp_2' 'cat_cp_3'
 'cat_fbs_0' 'cat_fbs_1' 'cat_restecg_0' 'cat_restecg_1'
 'cat_restecg_2' 'cat_exang_0' 'cat_exang_1' 'cat_slope_0'
 'cat_slope_1' 'cat_slope_2' 'cat_ca_0' 'cat_ca_1' 'cat_ca_2'
 'cat_ca_3' 'cat_ca_4' 'cat_thal_0' 'cat_thal_1' 'cat_thal_2'
 'cat_thal_3']
```

The following shows the baseline accuracy of simply classifying every sample as the majority class.

In []: `# Baseline accuracy of using the majority class`

```
ct = target_test.value_counts()
print("Counts of each class in target_test: ")
print(ct)
print(
    "=====",
    "\nBaseline Accuracy of using Majority Class:",
    np.round(np.max(ct) / np.sum(ct), 3),
)
```

```
Counts of each class in target_test:
target
0      66
1      56
Name: count, dtype: int64
=====
Baseline Accuracy of using Majority Class: 0.541
```

1. (25 pts) Decision Trees

1.1. [5 pts] Apply Decision Tree on Train Data

Apply the decision tree on the **train data** with default parameters of the `DecisionTreeClassifier`. **Report the accuracy and print the confusion matrix**. Make sure to use `random_state = SEED` so that your results match ours.

In []: `# TODO`

```
from sklearn import tree

clf = DecisionTreeClassifier(random_state=SEED)
clf.fit(train,target)
predicted = clf.predict(test)

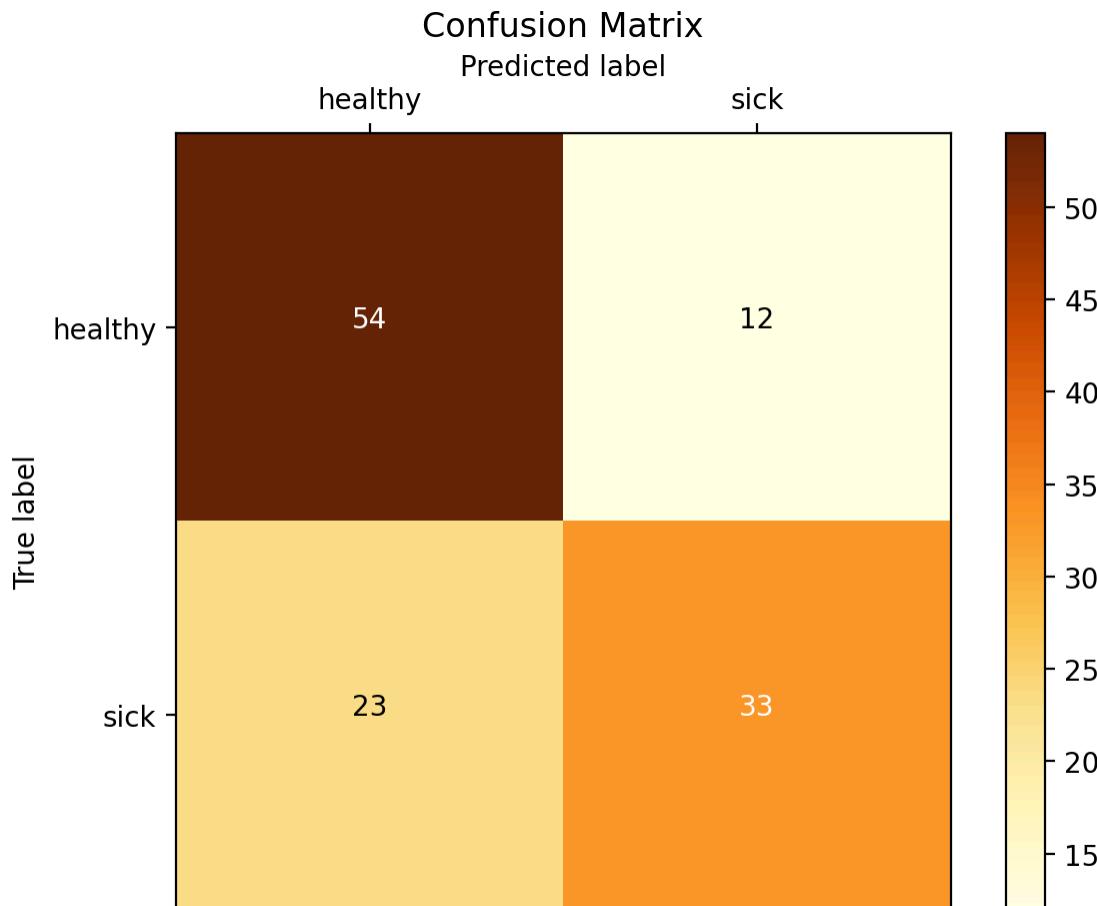
print("%-12s %f" % ("Accuracy:", metrics.accuracy_score(target_test, predict
```

```
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test, predicted))
draw_confusion_matrix(target_test, predicted, ["healthy", "sick"])
```

Accuracy: 0.713115

Confusion Matrix:

```
[[54 12]
 [23 33]]
```

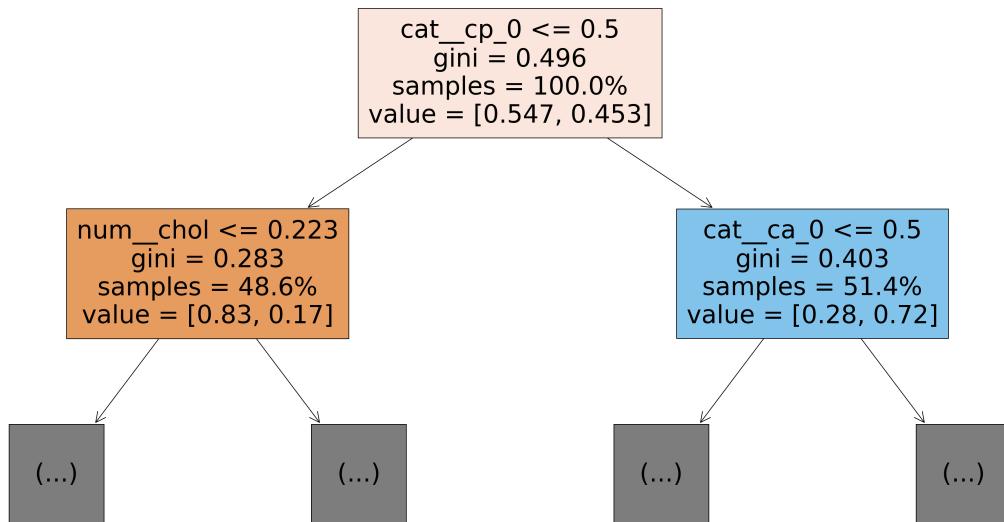


1.2. [5 pts] Visualize the Decision Tree

Visualize the first two layers of the decision tree that you trained.

```
In [ ]: # TODO
plt.figure(figsize=(30,15))
tree.plot_tree(clf, max_depth=1, proportion=True, feature_names=feature_names)
```

```
Out[ ]: [Text(0.5, 0.8333333333333334, 'cat_cp_0 <= 0.5\nngini = 0.496\nsamples = 1
00.0%\nvalue = [0.547, 0.453]'),
Text(0.25, 0.5, 'num_chol <= 0.223\nngini = 0.283\nsamples = 48.6%\nvalue
= [0.83, 0.17]'),
Text(0.125, 0.1666666666666666, '\n (...)\n'),
Text(0.375, 0.1666666666666666, '\n (...)\n'),
Text(0.75, 0.5, 'cat_ca_0 <= 0.5\nngini = 0.403\nsamples = 51.4%\nvalue =
[0.28, 0.72]'),
Text(0.625, 0.1666666666666666, '\n (...)\n'),
Text(0.875, 0.1666666666666666, '\n (...)\n')]
```



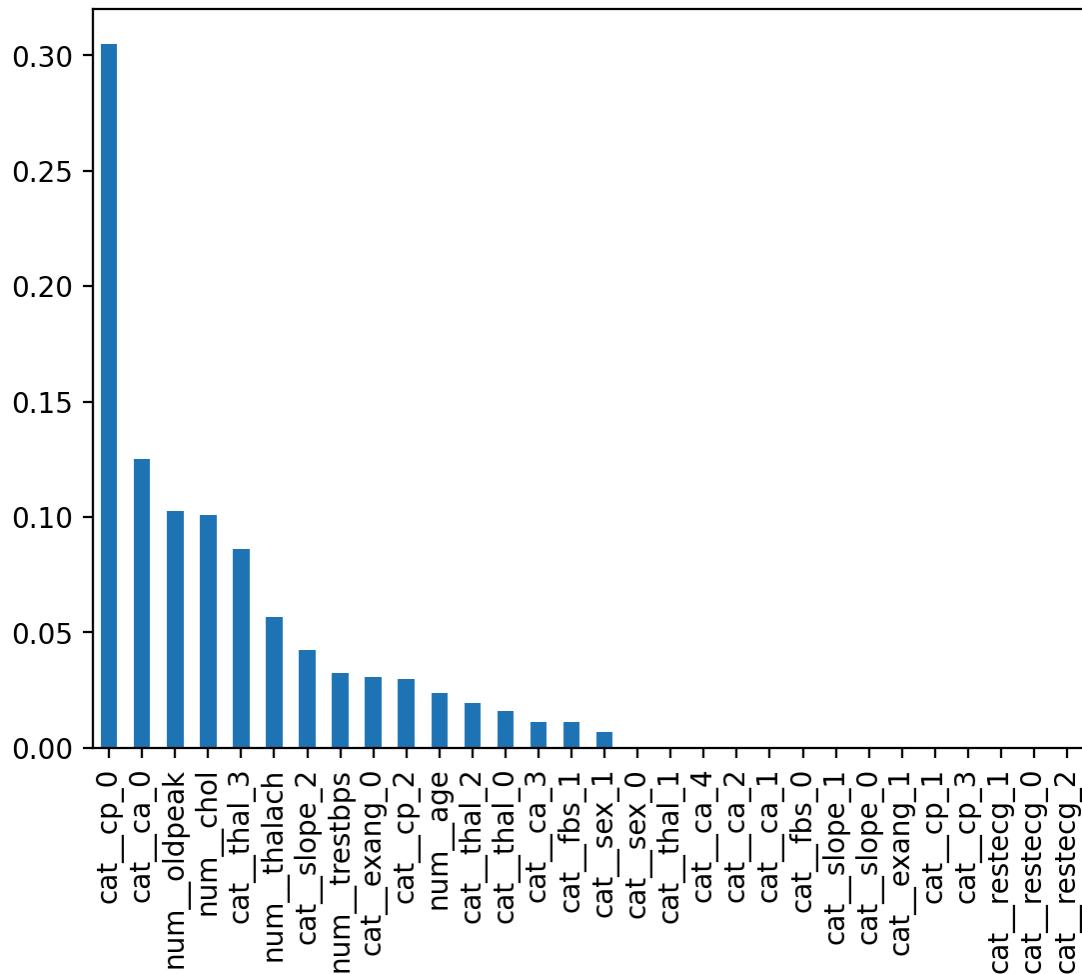
What is the gini index improvement of the first split?

Response: The first split produces an index improvoment of 0.189, New total Gini= (0.403+0.282) - Old Gini=(0.4946) = 0.189

1.3 [5 pts] Plot the importance of each feature for the Decision Tree

```
In [ ]: # TODO
imp_pd = pd.Series(data=clf.feature_importances_, index=feature_names)
imp_pd = imp_pd.sort_values(ascending=False)
imp_pd.plot.bar()
```

Out[]: <Axes: >



How many features have non-zero importance for the Decision Tree? If we remove the features with zero importance, will it change the decision tree for the same sampled dataset?

Response: In total there are 16 features which have non-zero importance. Features with zero importance do not change the decision tree at all and can be removed without altering the decision tree, since as stated, they have zero importance and thus have no affect on the algorithm.

1.4 [10 pts] Optimize Decision Tree

While the default Decision Tree performs fairly well on the data, lets see if we can improve performance by optimizing the parameters.

Run a `GridSearchCV` with 5-Fold Cross Validation for the Decision Tree. Find the best model parameters for accuracy amongst the following:

- `max_depth = [2, 4, 8, 16, 32]`
- `min_samples_split = [2, 4, 8, 16]`
- `criterion = [gini, entropy]`

After using `GridSearchCV`, Print the **best 5 models** with the following parameters:

`rank_test_score`, `param_max_depth`, `param_min_samples_split`,
`param_criterion`, `mean_test_score`, `std_test_score`.

```
In [ ]: # TODO
parameters = [
    {
        "max_depth": [2,4,8,16,32],
        "min_samples_split": [2,4,8,16],
        "criterion": ["gini", "entropy"],
    },
]

kf = KFold(n_splits=5, random_state=None)
grid = GridSearchCV(clf, parameters, cv=kf, scoring="accuracy")
grid.fit(train,target)

target_params = ['rank_test_score', 'param_max_depth', 'param_min_samples_split', 'param_criterion']
res = pd.DataFrame(grid.cv_results_).sort_values("rank_test_score", ascending=True)
res[target_params].head(5)
```

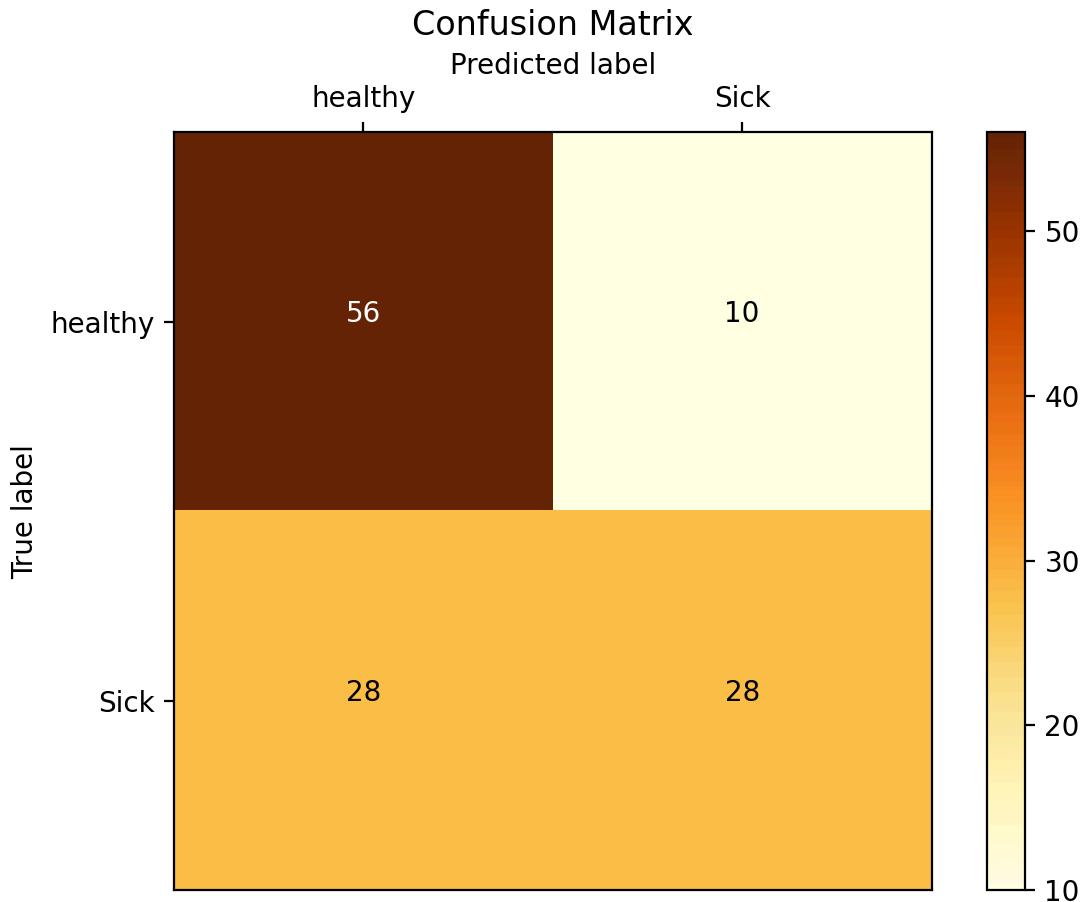
	rank_test_score	param_max_depth	param_min_samples_split	param_criterion	
17	1	32	4	gini	
13	1	16	4	gini	
9	1	8	4	gini	
4	4	4	2	gini	
12	5	16	2	gini	

Using the best model you have, report the test accuracy and print out the confusion matrix

```
In [ ]: # TODO
best = grid.best_estimator_
best.fit(train, target)
predicted = best.predict(test)

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test, predicted)))
draw_confusion_matrix(target_test, predicted, ["healthy", "Sick"])
```

Accuracy: 0.688525



2. (20 pts) Multi-Layer Perceptron

2.1 [5 pts] Applying a Multi-Layer Perceptron

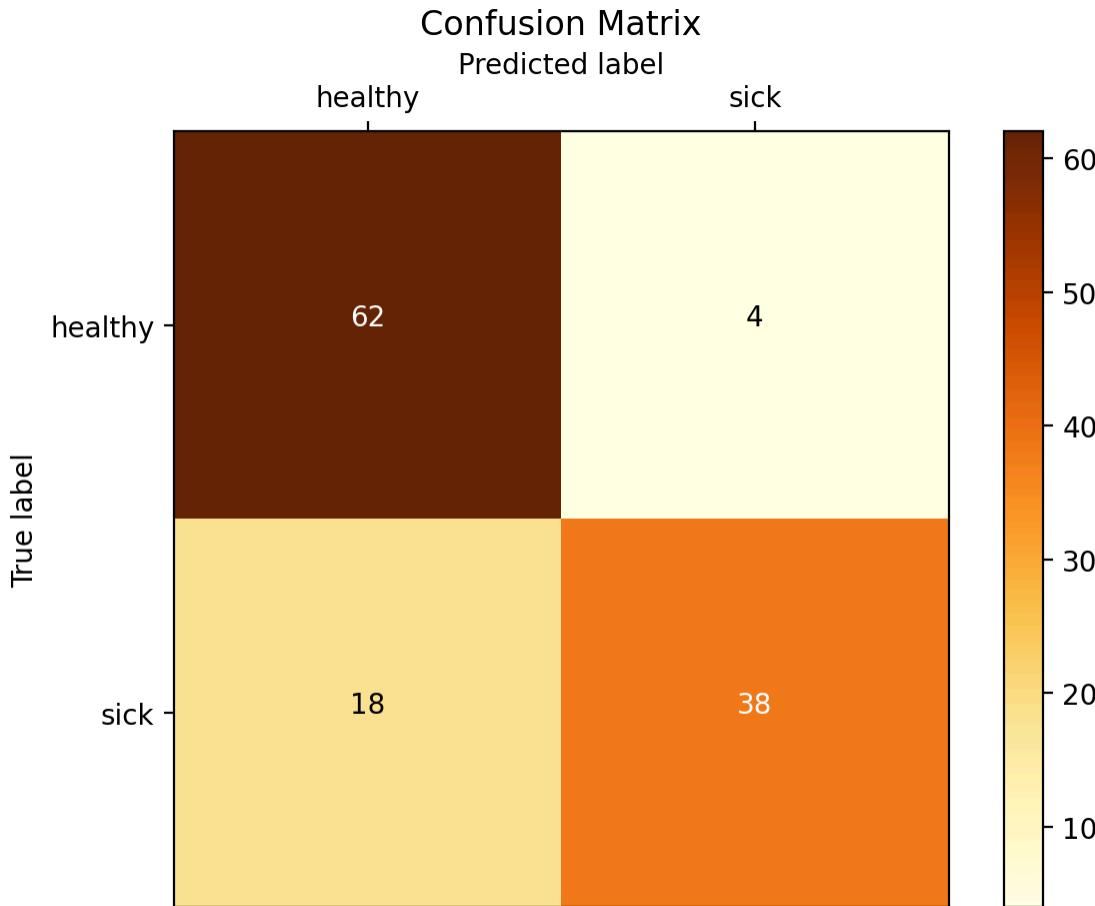
Apply the MLP on the **train data** with `hidden_layer_sizes=(50, 50)` and `max_iter = 1000`.

Report the accuracy and print the confusion matrix. Make sure to set
`random_state=SEED`.

```
In [ ]: # TODO
clf = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=1000, random_state=SEED)
clf.fit(train,target)
predicted = clf.predict(test)

print("%-12s %f" % ("Accuracy:", metrics.accuracy_score(target_test, predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test, predicted))
draw_confusion_matrix(target_test, predicted, ["healthy", "sick"]))

Accuracy: 0.819672
Confusion Matrix:
[[62  4]
 [18 38]]
```



2.2 [10 pts] Speedtest between Decision Tree and MLP

Let us compare the training times and prediction times of a Decision Tree and an MLP.

Time how long it takes for a Decision Tree and an MLP to perform a .fit operation (i.e. training the model). Then, time how long it takes for a Decision Tree and an MLP to perform a .predict operation (i.e. predicting the testing data). Print out the timings and specify which model was quicker for each operation. We recommend using the `time` python module to time your code. An example of the time module was shown in project 2. Use the default Decision Tree Classifier and the MLP with the previously mentioned parameters.

```
In [ ]: # TODO

dt = DecisionTreeClassifier(random_state=SEED)
mlp = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=1000, random_state=SEED)

t0 = time.time()
dt.fit(train, target)
t1 = time.time()
print("DT Training Time : ", t1 - t0)

t0 = time.time()
mlp.fit(train, target)
```

```
t1 = time.time()
print("MLP Training Time : ", t1 - t0)
```

DT Training Time : 0.005543231964111328
MLP Training Time : 0.6761059761047363

Decision Trees were much quicker than the MLP.

2.3 [5 pts] Compare and contrast Decision Trees and MLPs.

Describe at least one advantage and disadvantage of using an MLP over a Decision Tree.

Response: MLP's take longer to train and require more compute, but can handle more complex, non-linear relationships and patterns in the data.

3 (35 pts) PCA

3.1 [5 pts] Transform the train data using PCA

Train a PCA model to project the train data on the top 10 components. **Print out the 10 principal components.** Look at the documentation of [PCA](#) for reference.

In []:

```
# TODO
pca = PCA(n_components=10)
pca_pipe = Pipeline(
    [
        (
            "scaler",
            StandardScaler(),
        ), # Scikit learn PCA does not standardize so we need to add
        ("pca", pca),
    ]
)
pca_pipe.fit(train)
print(pca.components_)
```

```
[ [ 0.18611809  0.14723377  0.0827966 -0.26958547  0.28724158 -0.08876141
  0.08876141  0.29373271 -0.18880161 -0.18588046  0.00547491 -0.03616204
  0.03616204  0.17165538 -0.18229389  0.04263397 -0.29674427  0.29674427
  0.12038936  0.23640547 -0.29624895 -0.19017656  0.06708592  0.13205351
  0.14006446 -0.08106023 -0.00461948  0.08832725 -0.26014306  0.21954194]
[ 0.17782264  0.12934087  0.21781761  0.00085997 -0.00819057  0.47896852
 -0.47896852  0.01761698 -0.04673436  0.02638517 -0.01520276 -0.07185905
  0.07185905  0.32618016 -0.35610124  0.11900946  0.08021001 -0.08021001
 -0.05087735  0.09295655 -0.06752443  0.04033609 -0.06904782  0.0610647
  0.03216393 -0.13364728  0.02210197 -0.01976779  0.25112614 -0.25016493]
[ 0.1113784   0.20923116 -0.08340017  0.07921133 -0.03404349 -0.2018614
  0.2018614   -0.17797524 -0.05583133  0.16476042  0.1340265  -0.5511467
  0.5511467   0.16551362 -0.14622474 -0.07456177  0.10682938 -0.10682938
  0.02514107 -0.12564882  0.11302289 -0.1552864   0.10431192  0.05430955
  0.04779555  0.03009288  0.05778072  0.08887376 -0.02385473 -0.03478466]
[-0.0354728   0.25902106  0.07384051  0.19917574  0.21672998 -0.05117849
  0.05117849 -0.05916722  0.08173444 -0.16327904  0.27319757  0.09117566
 -0.09117566  0.07437116 -0.10230469  0.10978207  0.2768   -0.2768
  0.27081553 -0.10187863 -0.03320523  0.2913109  -0.46253951  0.0516882
  0.13021605 -0.03789029 -0.05916426  0.0925144  -0.24774416  0.21632863]
[-0.09120263  0.04867001 -0.16181035 -0.16313074  0.05191553  0.13154099
 -0.13154099 -0.07616742  0.01404994  0.03585138  0.06481676 -0.27743322
  0.27743322 -0.33678475  0.30039026  0.14054336 -0.1202493  0.1202493
  0.03773732  0.28359868 -0.30220316  0.37441033 -0.24584018 -0.12520515
 -0.12582519 -0.08216671  0.22809992  0.09364128 -0.00517024 -0.09262433]
[ 0.23307082  0.13298481  0.06645393 -0.14367125  0.22796243  0.14056155
 -0.14056155 -0.09547381 -0.18403626  0.36960395 -0.19049974  0.02293687
 -0.02293687 -0.32872864  0.24085064  0.34213852  0.13450707 -0.13450707
  0.22217753 -0.14046632  0.02960144 -0.25716438  0.07215712  0.00252155
  0.26635924  0.19015126 -0.06802427 -0.10381656 -0.03201311  0.10175308]
[ 0.13085041 -0.04575547 -0.22681007 -0.12219539 -0.03489723 -0.13260932
  0.13260932 -0.03455588 -0.01999381 -0.02700035  0.13552365  0.13776979
 -0.13776979 -0.02944488 -0.02098806  0.19728225  0.25071585 -0.25071585
 -0.12632303  0.29187912 -0.22868895 -0.17759453  0.03986722  0.22017285
 -0.09627945  0.17524083 -0.12280017  0.49558508  0.09362784 -0.33006298]
[-0.06057854 -0.04762676 -0.25470518 -0.066121   0.08963564 -0.04715481
  0.04715481  0.26192019  0.09891764 -0.16844768 -0.34155758 -0.03102717
  0.03102717  0.03440101 -0.09266925  0.2283075  -0.11375503  0.11375503
  0.4027465   -0.40278097  0.20170459  0.08092845 -0.06956298 -0.02784671
  0.03537828 -0.08045318  0.05552359  0.29333321  0.15082035 -0.31988296]
[-0.02460261 -0.02264009 -0.05736137  0.02458658 -0.04154146  0.18141881
 -0.18141881  0.2411226  0.10906297 -0.20323966 -0.25847479 -0.1271958
  0.1271958  -0.14610735  0.2077026  -0.24197382  0.09125688 -0.09125688
 -0.24611375 -0.03527912  0.15794758 -0.19074927 -0.2596861  0.56539207
  0.0541471   0.03303556  0.08050687  0.09757573 -0.18226449  0.11718802]
[ 0.08800402 -0.21217329  0.14694927 -0.11066296 -0.14357307 -0.05776048
  0.05776048  0.05206411  0.41875816 -0.2528875  -0.22931487 -0.11993696
  0.11993696 -0.02847204 -0.00386701  0.12643811  0.191856  -0.191856
 -0.13052738  0.16279614 -0.09760494 -0.0266236  0.08595074 -0.37887698
  0.47800008 -0.16289395 -0.0602162   0.01667568 -0.06317466  0.06842021]]
```

3.2 [5 pts] Percentage of variance explained by top 10 principal components

Using PCA's "explained_variance_ratio_", print the percentage of variance explained by the top 10 principal components.

```
In [ ]: # TODO  
print(pca.explained_variance_ratio_)
```

```
[0.17626116 0.08915634 0.07963692 0.06307261 0.05597995 0.05167126  
0.04939192 0.04876524 0.0428602 0.04058251]
```

3.3 [5 pts] Transform the train and test data into train_pca and test_pca using PCA

Note: Use fit_transform for train and transform for test

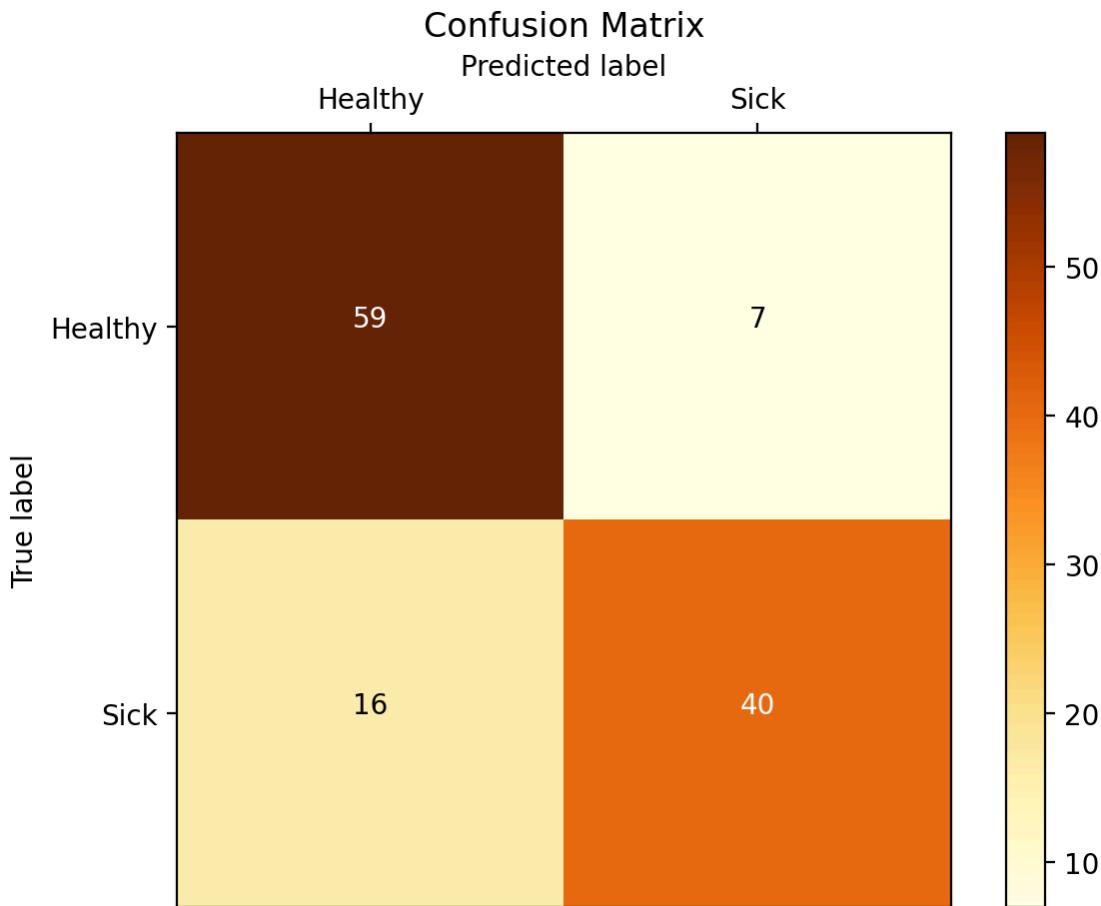
```
In [ ]: # TODO  
  
train_pca = pca.fit_transform(train)  
test_pca = pca.transform(test)
```

3.4 [5 pts] PCA+Decision Tree

Train the default Decision Tree Classifier using train_pca. **Report the accuracy using test_pca and print the confusion matrix.**

```
In [ ]: # TODO  
clf_pca = DecisionTreeClassifier(random_state=SEED)  
clf_pca.fit(train_pca, target)  
predicted = clf_pca.predict(test_pca)  
  
print("Accuracy with PCA")  
print("%-12s %f" % ("Accuracy:", metrics.accuracy_score(target_test, predicted))  
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test, predicted))  
draw_confusion_matrix(target_test, predicted, ["Healthy", "Sick"])
```

```
Accuracy with PCA  
Accuracy: 0.811475  
Confusion Matrix:  
[[59  7]  
[16 40]]
```



Does the model perform better with or without PCA?

Response: The model performs much better with PCA with an over 10% boost in accuracy after applying PCA.

3.5 [5 pts] PCA+MLP

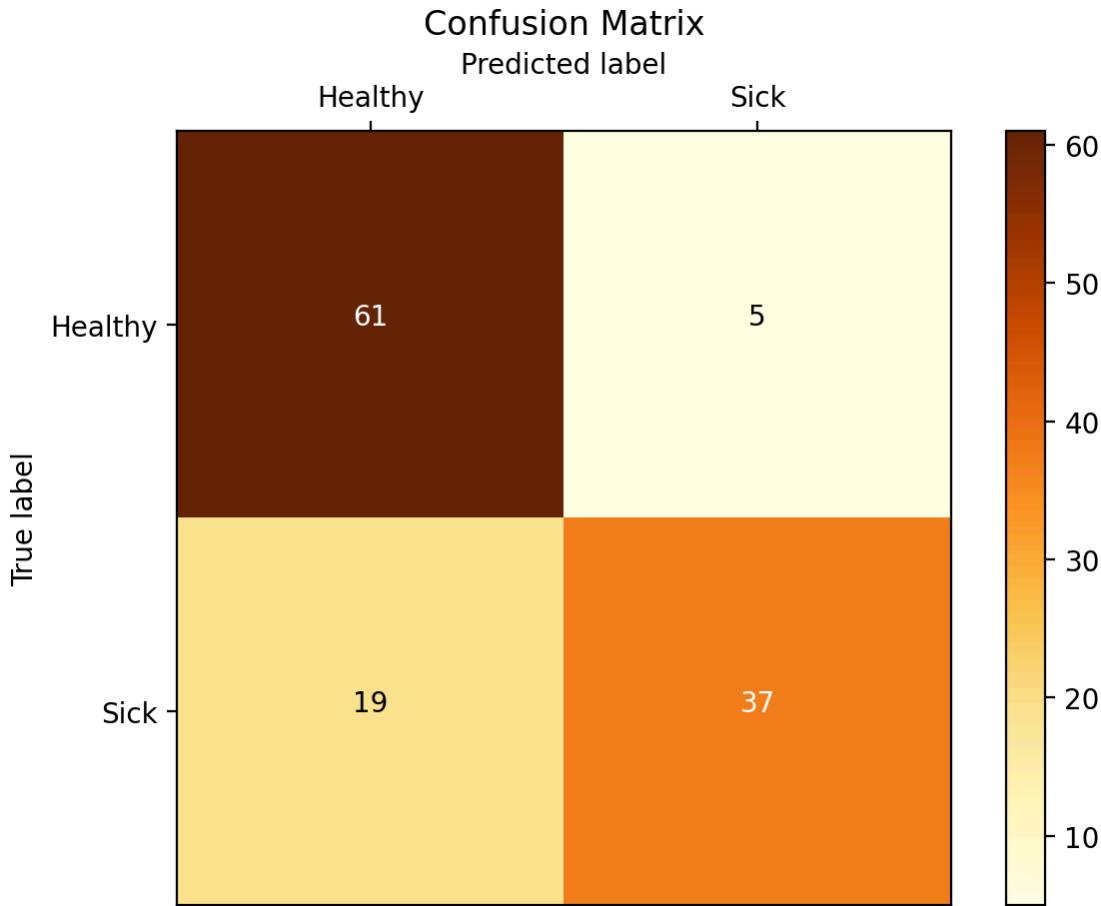
Train the MLP classifier with the same parameters as before using `train_pca`. **Report the accuracy using `test_pca` and print the confusion matrix.**

```
In [ ]: # TODO
mlp_pca = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=1000, random_state=42)
mlp_pca.fit(train_pca, target)
predicted = mlp_pca.predict(test_pca)

print("Accuracy with PCA")
print("%-12s %f" % ("Accuracy:", metrics.accuracy_score(target_test, predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test, predicted))
draw_confusion_matrix(target_test, predicted, ["Healthy", "Sick"])
```

Accuracy with PCA
 Accuracy: 0.803279
 Confusion Matrix:

$$\begin{bmatrix} 61 & 5 \\ 19 & 37 \end{bmatrix}$$



Does the model perform better with or without PCA?

Response: This model performed slightly worse after using PCA with around a 1% loss in accuracy.

3.6 [10 pts] Pros and Cons of PCA

In your own words, provide at least two pros and at least two cons for using PCA

Response: PCA is great for dimensionality reduction which can lead to reducing noise in the data and helps models focus on the main features leading to trends in the data. Also, PCA can remove multicollinearity, a key part in helping model data better. But, PCA can remove too much of the data, reducing models performance by removing components key to the model, removed for having low variance. Also, this a linear reduction technique so if the dataset does not have a linear decision boundary, PCA will not help at all.

4. (20 pts) K-Means Clustering

4.1 [5 pts] Apply K-means to the train data and print out the Inertia score

Use `n_cluster = 5` and `random_state = SEED`.

In []:

```
# TODO
kmeans = KMeans(n_clusters=5, random_state=SEED)
kmeans.fit(train)
print(f" Inertia Score:{kmeans.inertia_}")
```

Inertia Score:489.0488065162732

4.2 [10 pts] Find the optimal cluster size using the elbow method.

Use the elbow method to find the best cluster size or range of best cluster sizes for the train data. Check the cluster sizes from 2 to 25. Make sure to plot the Inertia and state where you think the elbow starts. Make sure to use `random_state = SEED`.

In []:

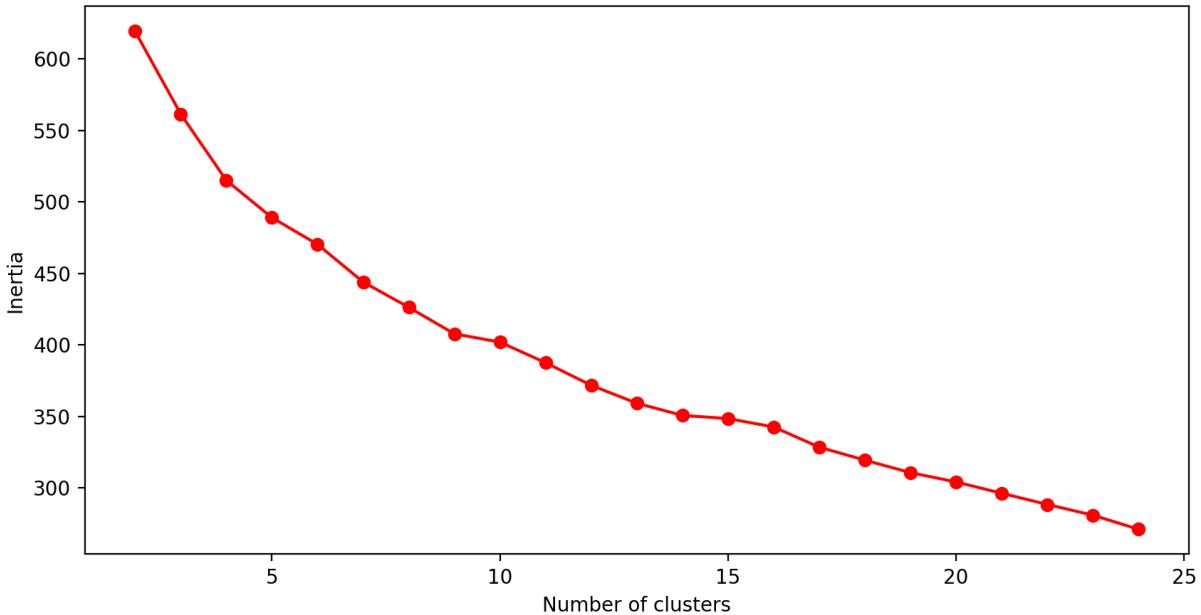
```
# TODO
ks = list(range(2, 25))

inertia = []
for k in ks:
    kmeans = KMeans(n_clusters=k, init="k-means++", random_state=SEED)
    kmeans.fit(train)
    # inertia method returns wcss for that model
    inertia.append(kmeans.inertia_)
    print(f" Inertia for K = {k}: {kmeans.inertia_}")

plt.figure(figsize=(10, 5))
plt.plot(ks, inertia, marker="o", color="red")
plt.title("The Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()
```

```
Inertia for K = 2: 619.2954019046072
Inertia for K = 3: 561.5278841368324
Inertia for K = 4: 515.0859158871034
Inertia for K = 5: 489.04880651627326
Inertia for K = 6: 470.33014819814787
Inertia for K = 7: 444.00021583021544
Inertia for K = 8: 426.3013305650318
Inertia for K = 9: 407.75246588206994
Inertia for K = 10: 401.96228381959475
Inertia for K = 11: 387.55286376836426
Inertia for K = 12: 371.7447692166255
Inertia for K = 13: 359.21875024779405
Inertia for K = 14: 350.70140597033185
Inertia for K = 15: 348.5128339576984
Inertia for K = 16: 342.60224115848393
Inertia for K = 17: 328.4880145733202
Inertia for K = 18: 319.4091985374208
Inertia for K = 19: 310.6815040643602
Inertia for K = 20: 304.1670627708047
Inertia for K = 21: 296.2499375298579
Inertia for K = 22: 288.42807171722495
Inertia for K = 23: 280.9185553969854
Inertia for K = 24: 271.0612652241275
```

The Elbow Method



Although K-Means clustering does not seem like good fit for our data, the elbow seems to be at around 6 clusters since this is when the slope started to decrease the most and points become closer together.

4.3 [5 pts] Find the optimal cluster size for the train_pca data

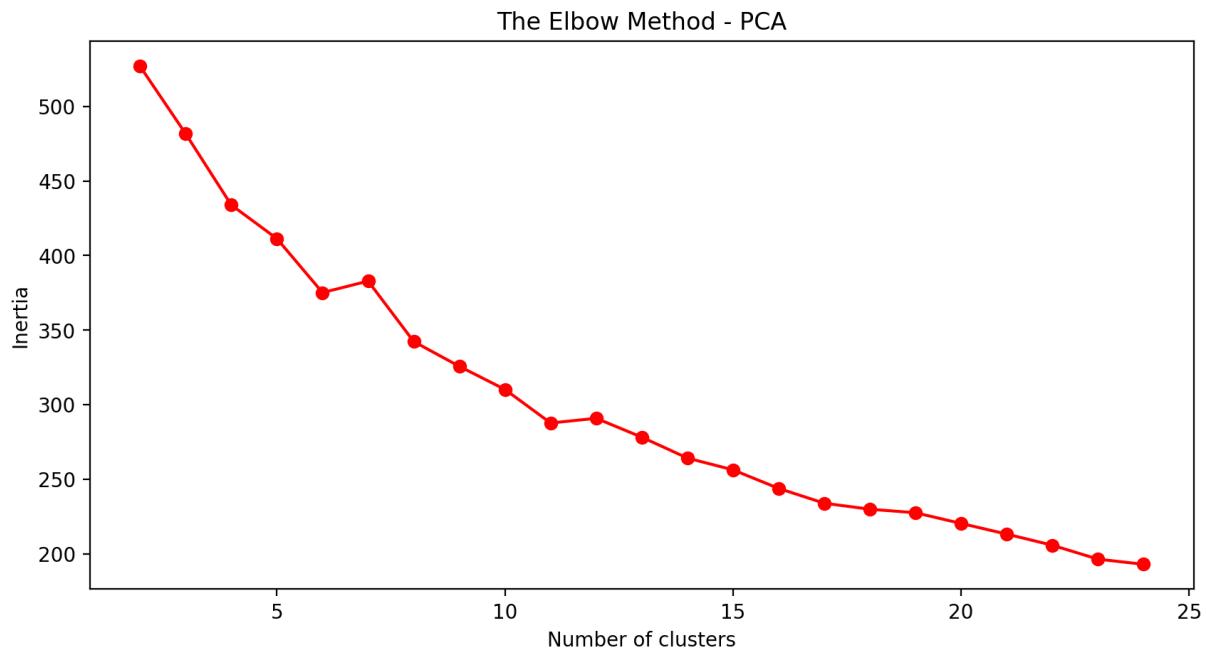
Repeat the same experiment but use train_pca instead of train.

```
In [ ]: # TODO
ks = list(range(2, 25))
```

```
inertia = []
for k in ks:
    kmeans = KMeans(n_clusters=k, init="k-means++", random_state=SEED)
    kmeans.fit(train_pca)
    # inertia method returns wcss for that model
    inertia.append(kmeans.inertia_)
    print(f"Inertia for K = {k}: {kmeans.inertia_}")

plt.figure(figsize=(10, 5))
plt.plot(ks, inertia, marker="o", color="red")
plt.title("The Elbow Method – PCA")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()
```

Inertia for K = 2: 526.8986473659902
Inertia for K = 3: 481.68331476211074
Inertia for K = 4: 433.93185987666993
Inertia for K = 5: 411.36968943504326
Inertia for K = 6: 375.1867871970702
Inertia for K = 7: 382.89919979252363
Inertia for K = 8: 342.3458233480021
Inertia for K = 9: 325.73232432730373
Inertia for K = 10: 310.2418151821646
Inertia for K = 11: 287.7537700262638
Inertia for K = 12: 290.9020452246428
Inertia for K = 13: 278.22459380391615
Inertia for K = 14: 264.2275350235982
Inertia for K = 15: 256.3164359685783
Inertia for K = 16: 243.8877943092183
Inertia for K = 17: 233.951678383709
Inertia for K = 18: 229.9404407582009
Inertia for K = 19: 227.5745255599991
Inertia for K = 20: 220.3989336720402
Inertia for K = 21: 213.25908756269746
Inertia for K = 22: 205.80054892550046
Inertia for K = 23: 196.45100530250517
Inertia for K = 24: 193.0105928950394



Similar to the previous experiment, we can guess that the best cluster size is somewhere between 5 and 10. Additionally, we see that the inertia is much smaller for every cluster size when using PCA features.

Response: The best cluster size seems to be around 7 clusters, since this is where the inertia slows down the most.

Project 3b

The final part of the project will ask you to perform your own data science project to classify a new dataset.

Submission Details

Project is due June 14th at 11:59 pm (Friday Midnight). To submit the project, please save the notebook as a pdf file and submit the assignment via Gradescope. In addition, make sure that all figures are legible and sufficiently large. For best pdf results, we recommend printing the notebook using *LATEX*

Loading Essentials and Helper Functions

```
In [ ]: # fix for windows memory leak with MKL
import os
import platform

if platform.system() == "Windows":
    os.environ["OMP_NUM_THREADS"] = "2"
```

```
In [ ]: # import libraries
import time
import random
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # this is used for the plot the graph

# Sklearn classes
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    GridSearchCV,
    KFold,
    RandomizedSearchCV
)
from sklearn import metrics
from sklearn.metrics import confusion_matrix, silhouette_score
import sklearn.metrics.cluster as smc
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import (
    StandardScaler,
    OneHotEncoder,
    LabelEncoder,
    MinMaxScaler,
```

```
)  
from sklearn.compose import ColumnTransformer, make_column_transformer  
from sklearn import tree  
from sklearn import datasets  
from sklearn.decomposition import PCA  
from sklearn.neural_network import MLPClassifier  
from sklearn.linear_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.datasets import make_blobs  
  
from helper import (  
    draw_confusion_matrix,  
    heatmap,  
    make_meshgrid,  
    plot_contours,  
    draw_contour,  
)  
  
%matplotlib inline  
%config InlineBackend.figure_format = 'retina'  
  
# Sets random seed for reproducibility  
SEED = 42  
random.seed(SEED)
```

(100 pts) Putting it all together: Classify your own data

Through the course of this program, you have acquired knowledge and skills in applying various models to tackle supervised learning tasks. Now, we challenge you to harness your cumulative learning and create a model capable of predicting whether a hotel reservation will be canceled or not.

Context

Hotels welcome millions of guests every year, and their primary objective is to keep rooms occupied and paid for. Cancellations can be detrimental to the business, as it may become challenging to rebook a room on short notice. Consequently, it is beneficial for hotels to anticipate which reservations are likely to be canceled. The provided dataset offers a diverse range of information about bookings, which you will utilize to predict cancellations.

Challenge

The goal of this project is to develop a predictive model that can determine whether a reservation will be canceled based on the available input parameters.

While we will provide specific instructions to guide you in the right direction, you have the freedom to choose the models and preprocessing techniques that you deem most appropriate. Upon completion, we request that you provide a detailed description outlining the models you selected and the rationale behind your choices.

Data Description

Refer to <https://www.kaggle.com/competitions/m-148-spring-2024-project-3/data> for information

(50 pts) Preprocessing

For the dataset, the following are mandatory pre-processing steps for your data:

- **Use One-Hot Encoding on all categorical features** (specify whether you keep the extra feature or not for features with multiple values)
- Determine which fields need to be dropped
- **Handle missing values** (Specify your strategy)
- **Rescale the real valued features using any strategy you choose** (StandardScaler, MinMaxScaler, Normalizer, etc)
- **Augment at least one feature**
- **Implement a train-test split with 20% of the data going to the test data.** Make sure that the test and train data are balanced in terms of the desired class.

After writing your preprocessing code, write out a description of what you did for each step and provide a justification for your choices. All descriptions should be written in the markdown cells of the jupyter notebook. Make sure your writing is clear and professional.

We highly recommend reading through the [scikit-learn documentation](#) to make this part easier.

```
In [ ]: # TODO
hotels = pd.read_csv("datasets/hotel_booking.csv")
hotels.info()
hotels.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69591 entries, 0 to 69590
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   hotel            69591 non-null   object  
 1   is_canceled      69591 non-null   int64  
 2   lead_time         69591 non-null   int64  
 3   arrival_date_month 69591 non-null   object  
 4   stays_in_weekend_nights 69591 non-null   int64  
 5   stays_in_week_nights 69591 non-null   int64  
 6   adults            69591 non-null   int64  
 7   children          69588 non-null   float64 
 8   babies             69591 non-null   int64  
 9   meal               69591 non-null   object  
 10  country            69591 non-null   object  
 11  previous_cancellations 69591 non-null   int64  
 12  previous_bookings_not_canceled 69591 non-null   int64  
 13  reserved_room_type 69591 non-null   object  
 14  booking_changes    69591 non-null   int64  
 15  deposit_type       69591 non-null   object  
 16  days_in_waiting_list 69591 non-null   int64  
 17  customer_type      69591 non-null   object  
 18  adr                69591 non-null   float64 
 19  required_car_parking_spaces 69591 non-null   int64  
 20  total_of_special_requests 69591 non-null   int64  
 21  name               69591 non-null   object  
 22  email              69591 non-null   object  
 23  phone-number       69591 non-null   object  
dtypes: float64(2), int64(12), object(10)
memory usage: 12.7+ MB
```

	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights
count	69591.000000	69591.000000	69591.000000	69591.000000
mean	0.405814	109.181546	0.880746	2.434280
std	0.491052	113.714559	0.983784	1.852226
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	17.000000	0.000000	1.000000
50%	0.000000	71.000000	1.000000	2.000000
75%	1.000000	169.000000	2.000000	3.000000
max	1.000000	709.000000	16.000000	41.000000

Here I visualized the data to see what I am working with in terms of types and features.

```
In [ ]: hotels = hotels.drop(["name", "email", "phone-number"], axis=1)
y = hotels["is_canceled"]
x = hotels.drop(["is_canceled"], axis=1)

train_raw, test_raw, target, target_test = train_test_split(x, y, test_size=
```

I dropped the unneccasary personal information of people booking since my model should not take this into account at all.

```
In [ ]: from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin

#remove personal information

weekend_ix, week_ix, month_ix = 4, 5, 3

class AugmentFeatures(BaseEstimator, TransformerMixin):

    def __init__(self, add_total_nights=True):
        self.add_total_nights = add_total_nights

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        if self.add_total_nights:
            total_nights = X[:, weekend_ix] + X[:, week_ix]
            return np.c_[X, total_nights]
        else:
            return X

num_pipeline = Pipeline(
    [
        ("imputer", SimpleImputer(strategy="median")),
        ("attribs_adder", AugmentFeatures(add_total_nights=True)),
        ("std_scaler", StandardScaler()),
    ]
)

categorical_features = ["hotel", "arrival_date_month", "meal", "country", "region"]
numerical_features = ["lead_time", "stays_in_weekend_nights", "stays_in_week"]

full_pipeline = ColumnTransformer(
    [
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ]
)

train = full_pipeline.fit_transform(train_raw)
test = full_pipeline.transform(test_raw)
```

Here I added in a numerical feature which counts the total time that person books instead of two splits for weekdays and weekend days. I also used the OneHotEncoder on all of my categorical data and handled null values by replacing them with the median value for these data points. The null points were numerical values so this was easy to replace and fix.

```
In [ ]: ct = target_test.value_counts()
print("Counts of each class in target_test: ")
print(ct)
print(
    "=====",
    "\nBaseline Accuracy of using Majority Class:",
    np.round(np.max(ct) / np.sum(ct), 3),
)
```

```
Counts of each class in target_test:
is_canceled
0      8270
1      5649
Name: count, dtype: int64
=====
Baseline Accuracy of using Majority Class: 0.594
```

```
In [ ]: pca = PCA(n_components=0.9)
pca_pipe = Pipeline(
    [
        (
            "scaler",
            StandardScaler(),
        ),
        ("pca", pca),
    ]
)
pca_pipe.fit(train)

train_pca = pca.fit_transform(train)
test_pca = pca.transform(test)
```

(50 pts) Try out a few models

Now that you have pre-processed your data, you are ready to try out different models.

For this part of the project, we want you to experiment with all the different models demonstrated in the course to determine which one performs best on the dataset.

You must perform classification using at least 3 of the following models:

- Logistic Regression
- K-nearest neighbors
- SVM
- Decision Tree
- Multi-Layer Perceptron

Due to the size of the dataset, be careful which models you use and look at their documentation to see how you should tackle this size issue for each model.

For full credit, you must perform some hyperparameter optimization on your models of choice. You may find the following scikit-learn library on [hyperparameter optimization](#) useful.

For each model chosen, write a description of which models were chosen, which parameters you optimized, and which parameters you choose for your best model. While the previous part of the project asked you to pre-process the data in a specific manner, you may alter pre-processing step as you wish to adjust for your chosen classification models.

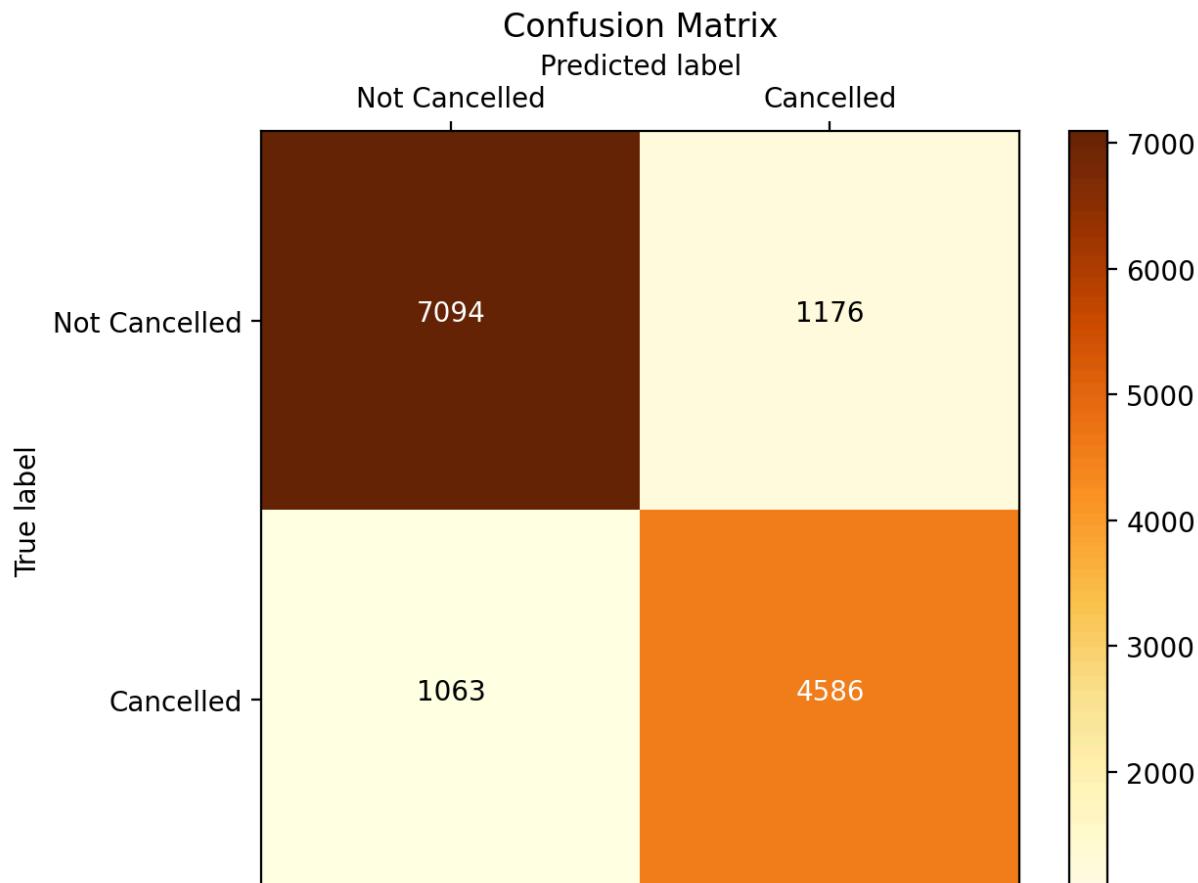
```
In [ ]: # TODO
dt = DecisionTreeClassifier(random_state=SEED)
dt.fit(train, target)
predicted = dt.predict(test)

print("%-12s %f" % ("Accuracy:", metrics.accuracy_score(target_test, predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test, predicted))
draw_confusion_matrix(target_test, predicted, ["Not Cancelled", "Cancelled"])
```

Accuracy: 0.839141

Confusion Matrix:

```
[[7094 1176]
 [1063 4586]]
```



```
In [ ]: parameters = [
    {
        "max_depth": [2, 4, 8, 16, 32],
```

```

        "min_samples_split": [2,4,8,16, 32],
        "criterion": ["gini", "entropy"],
    },
]

kf = KFold(n_splits=5, random_state=None)
grid = GridSearchCV(dt, parameters, cv=kf, scoring="accuracy")
grid.fit(train,target)

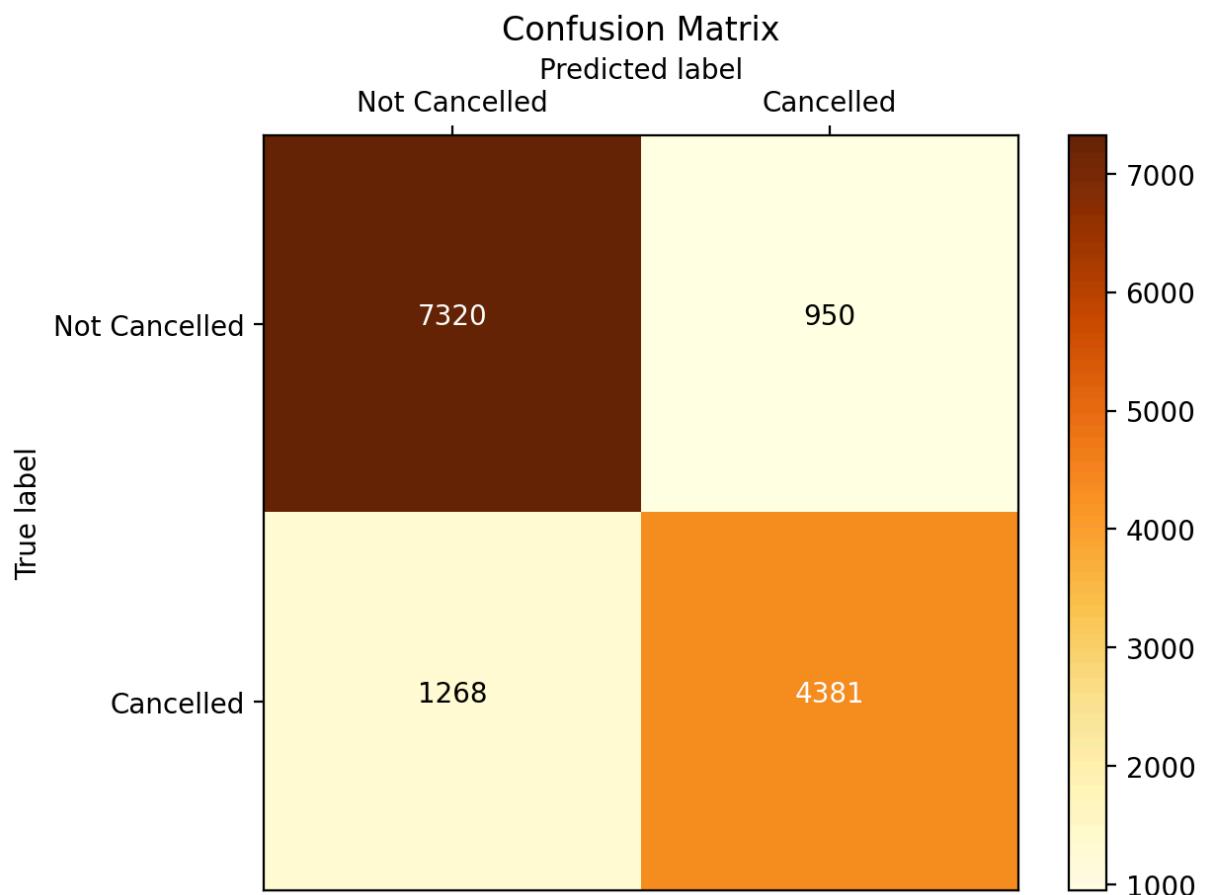
best_dt = grid.best_estimator_
best_dt.fit(train, target)

predicted = best_dt.predict(test)

print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test, predicted)))
draw_confusion_matrix(target_test, predicted, ["Not Cancelled", "Cancelled"])

```

Accuracy: 0.840649



In []: `grid.best_params_`

Out[]: {'criterion': 'entropy', 'max_depth': 32, 'min_samples_split': 32}

Performed hyperparameter adjustment with max depth of 32, min samples split of 32, and use of entropy loss instead gini after performed K Fold cross validation and retrieving the best parameter performance.

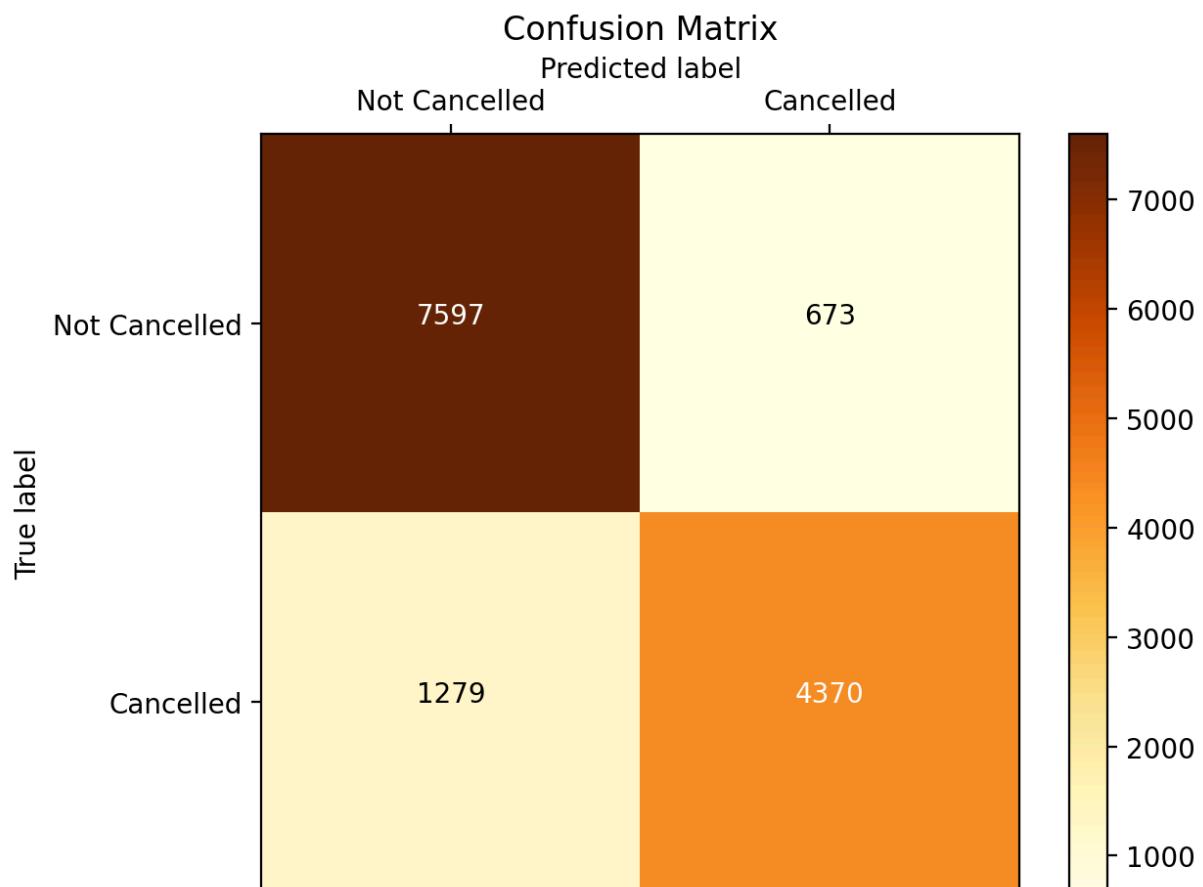
```
In [ ]: clf = MLPClassifier(hidden_layer_sizes=(15,25), max_iter=1000, random_state=42)
clf.fit(train,target)
predicted = clf.predict(test)

print("%-12s %f" % ("Accuracy:", metrics.accuracy_score(target_test, predicted)))
print("Confusion Matrix: \n", metrics.confusion_matrix(target_test, predicted))
draw_confusion_matrix(target_test, predicted, ["Not Cancelled", "Cancelled"])
```

Accuracy: 0.859760

Confusion Matrix:

```
[[7597 673]
 [1279 4370]]
```



Adjusted hidden layer size to (15,25) which I found to produce the best accuracy results. I could not use KFold in this instance due to extended computation runtime and had to manually perform the hyperparameter adjustments. This was the best model I was able to train with an 86% accuracy percentage and used max iterations of 1000 so convergence was found as well as SEED for random seed to keep things constant.

```
In [ ]: svm = SVC(kernel="rbf", C=50)
parameters = {
    "kernel" : ["rbf","linear"],
    "C" : [0.1, 10]
}

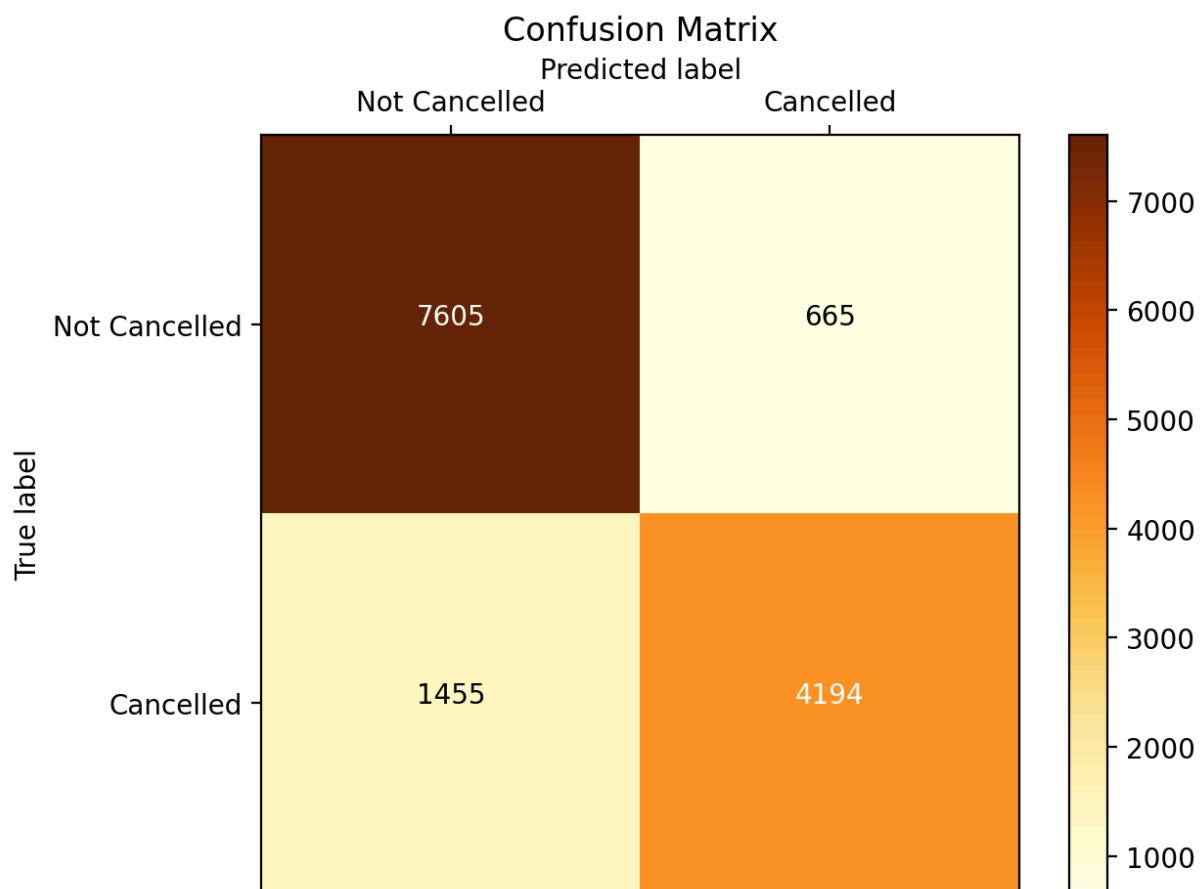
#kf = KFold(n_splits=2, random_state=None)
#grid = RandomizedSearchCV(svm, parameters, cv=kf, scoring = "accuracy")
```

```
#grid.fit(train_pca,target)

#best_lr = grid.best_estimator_
#best_lr.fit(train_pca, target)

#predicted = best_lr.predict(test_pca)
svm.fit(train_pca, target)
predicted = svm.predict(test_pca)
print("%-12s %f" % ('Accuracy:', metrics.accuracy_score(target_test, predicted)))
draw_confusion_matrix(target_test, predicted, ["Not Cancelled", "Cancelled"])
```

Accuracy: 0.847690



For SVC, I also could not use KFold due to extended runtime but used the rbf kernel and C value of 50 to produce the best accuracy results.

Extra Credit

We have provided an extra test dataset named `hotel_booking_test.csv` that does not have the target labels. Classify the samples in the dataset with any method of your choosing and save the predictions into a csv file. Submit the file to our [Kaggle](#) contest. The website will specify your classification accuracy on the test set. We will award a bonus point for the project for every percentage point over 75% that you get on your kaggle test accuracy.

To get the bonus points, you must also write out a summary of the model that you submit including any changes you made to the pre-processing steps. The summary must be written in a markdown cell of the jupyter notebook. Note that you should not change earlier parts of the project to complete the extra credit.

Please refer to *Submission and evaluation* section on the contest page for the `csv` file formatting

Model description

In []:

```
#TODO
unlabeled = pd.read_csv("datasets/hotel_booking_test.csv")
unlabeled.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17397 entries, 0 to 17396
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   hotel            17397 non-null   object 
 1   lead_time         17397 non-null   int64  
 2   arrival_date_month 17397 non-null   object 
 3   stays_in_weekend_nights 17397 non-null   int64  
 4   stays_in_week_nights 17397 non-null   int64  
 5   adults            17397 non-null   int64  
 6   children          17397 non-null   float64
 7   babies             17397 non-null   int64  
 8   meal               17397 non-null   object 
 9   country            17397 non-null   object 
 10  previous_cancellations 17397 non-null   int64  
 11  previous_bookings_not_canceled 17397 non-null   int64  
 12  reserved_room_type 17397 non-null   object 
 13  booking_changes    17397 non-null   int64  
 14  deposit_type       17397 non-null   object 
 15  days_in_waiting_list 17397 non-null   int64  
 16  customer_type      17397 non-null   object 
 17  adr                17397 non-null   float64
 18  required_car_parking_spaces 17397 non-null   int64  
 19  total_of_special_requests 17397 non-null   int64  
 20  name               17397 non-null   object 
 21  email              17397 non-null   object 
 22  phone-number        17397 non-null   object 

dtypes: float64(2), int64(11), object(10)
memory usage: 3.1+ MB
```