CA4

April 23, 2024

1 CA4

Anniken Rabben

Kaggle username: Anniken01

1.0.1 Imports

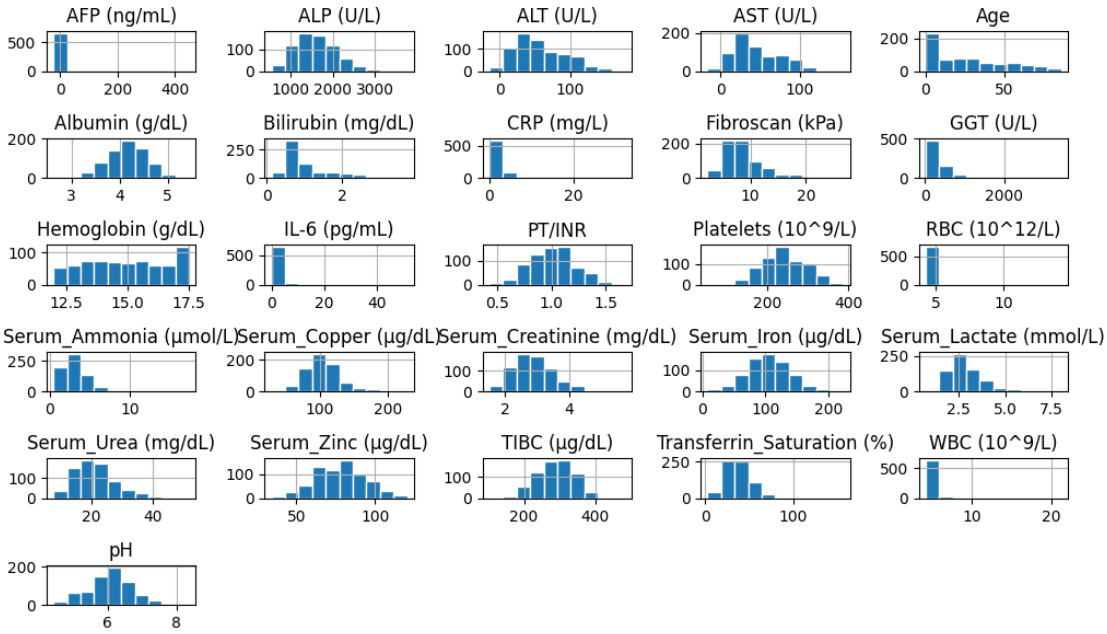
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.model_selection import StratifiedKFold
```

1.0.2 Reading data and indexing

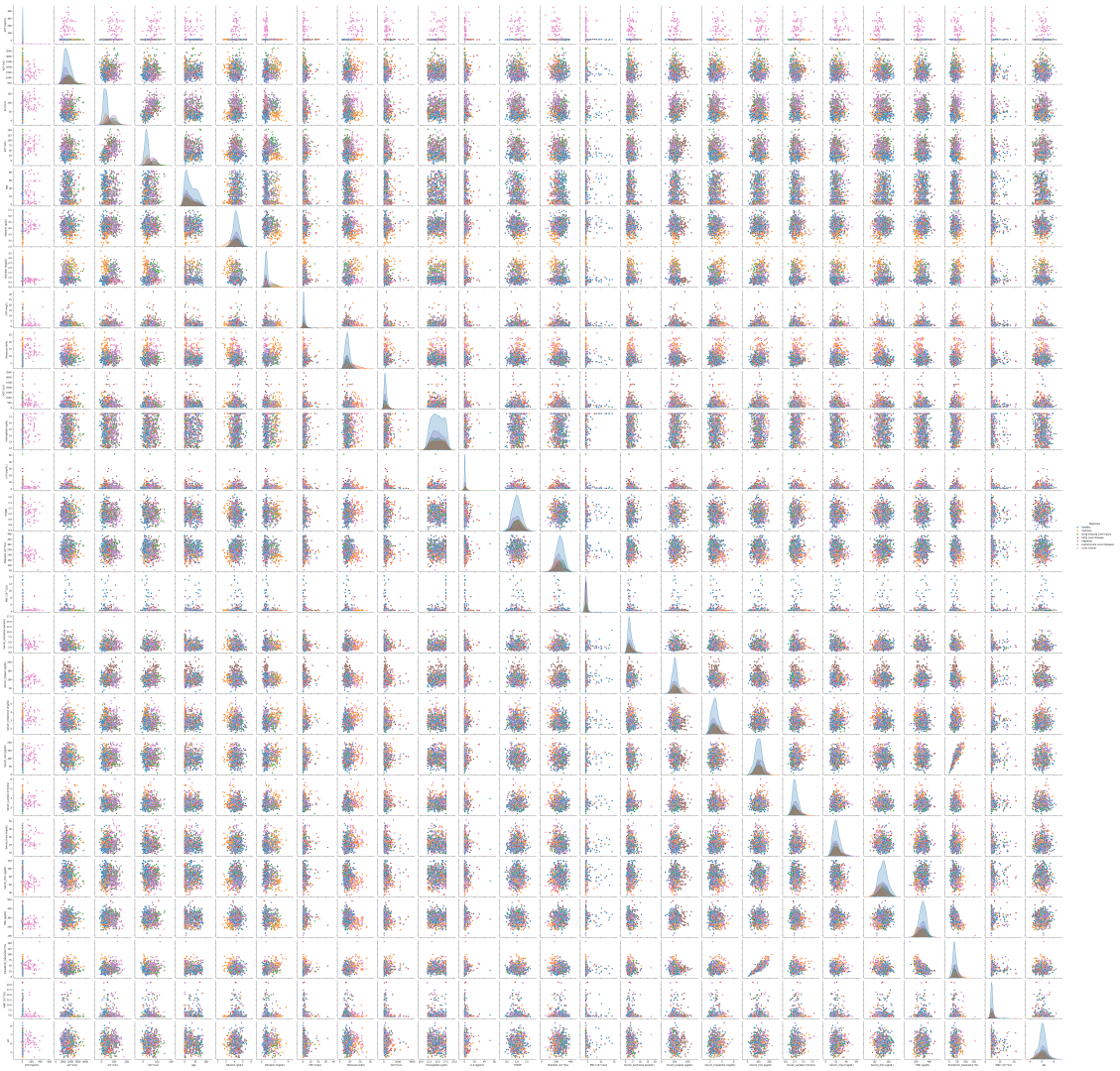
[ ]:
```
raw_train = pd.read_csv('train.csv', index_col=0)
raw_train.drop('index',inplace=True, axis=1) # removed one index column, since␣
↪it was two
test_data = pd.read_csv('test.csv', index_col=0)
test_data.drop('index',inplace=True, axis=1) # removed one index, since it was␣
↪two
```

1.0.3 Data exploration and visualisation before cleaning

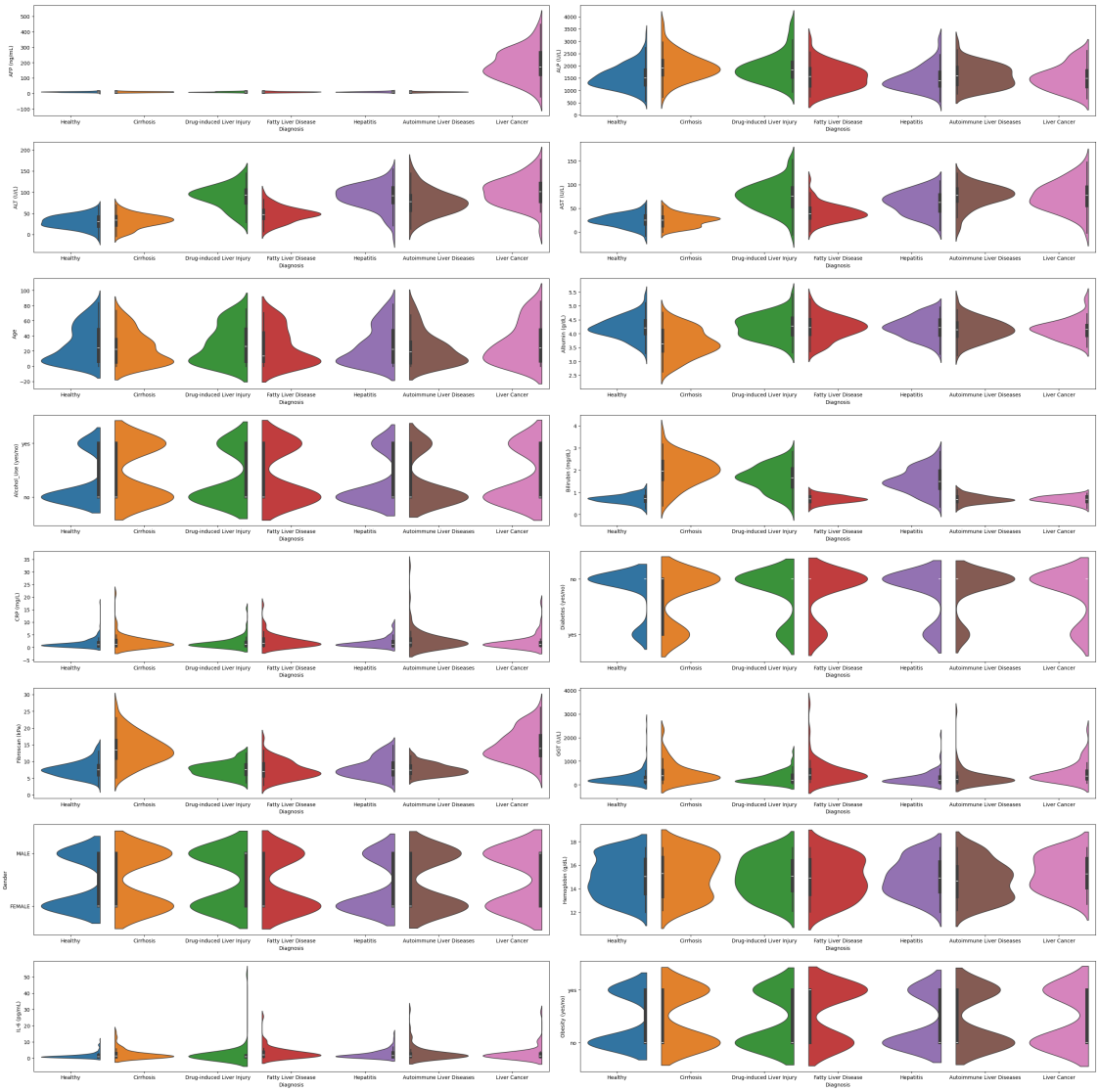
[ ]:
```
# Histogram
raw_train.hist(figsize=(10,6) , edgecolor="white") # edgecolor makes it look
↪nicer
plt.tight_layout()
plt.show()
```



[ ]:
```
# Pairplot
sns.pairplot(data=raw_train, hue = 'Diagnosis')
plt.show()
```



[ ]:
```
# violin plot
fig_violin, ax_violin = plt.subplots(nrows=8, ncols=2, figsize=(30,30))
plt.tight_layout(pad=3.0)

i = 0
for row in range(8):
    for col in range(2):
        sns.violinplot(ax=ax_violin[row, col], data=raw_train, x='Diagnosis',␣
 ↪y=raw_train.columns[i],
                       hue='Diagnosis', split=True)
        i += 1

plt.show()
```



[ ]:
```
# Check for missing values and shape of the data
print(raw_train.isnull().sum())
raw_train.shape
```

```
AFP (ng/mL)                0
ALP (U/L)                  0
ALT (U/L)                  0
AST (U/L)                  0
Age                        0
Albumin (g/dL)             0
Alcohol_Use (yes/no)       0
Bilirubin (mg/dL)          0
CRP (mg/L)                 0
Diabetes (yes/no)          0
```

```
Fibroscan (kPa)                0
GGT (U/L)                      0
Gender                         0
Hemoglobin (g/dL)              0
IL-6 (pg/mL)                   0
Obesity (yes/no)               0
PT/INR                         0
Platelets (10^9/L)             0
RBC (10^12/L)                  0
Serum_Ammonia ( mol/L)         0
Serum_Copper ( g/dL)           0
Serum_Creatinine (mg/dL)       0
Serum_Iron ( g/dL)             0
Serum_Lactate (mmol/L)         0
Serum_Urea (mg/dL)             0
Serum_Zinc ( g/dL)             0
TIBC ( g/dL)                   0
Transferrin_Saturation (%)     0
WBC (10^9/L)                   0
pH                             0
Diagnosis                      0
dtype: int64
```

[ ]:
```
(703, 31)
```

note: There are not missing any values

[ ]:
```
# counts the different diagnosis
print(raw_train['Diagnosis'].value_counts())
```

```
Diagnosis
Healthy                      250
Hepatitis                    113
Autoimmune Liver Diseases     79
Drug-induced Liver Injury     73
Cirrhosis                     71
Fatty Liver Disease           62
Liver Cancer                  55
Name: count, dtype: int64
```

1.0.4 Data cleaning

[ ]:
```
# convert the categorical train data to numerical data
data = [raw_train, test_data]
columns_to_encode = ['Alcohol_Use (yes/no)','Diabetes (yes/no)',␣
 ↪'Gender','Obesity (yes/no)']
le = LabelEncoder() # create a label encoder
```

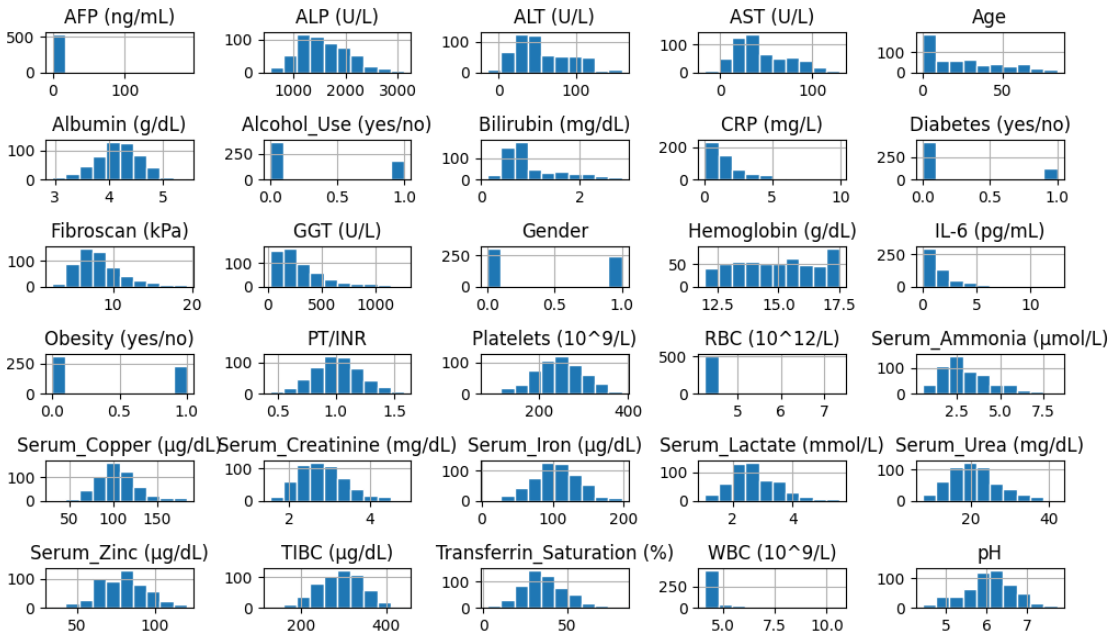
```
for df in data:
    for col in columns_to_encode:
        if df[col].dtype == 'object': # if the column is a string
            df[col] = le.fit_transform(df[col]) # convert the string to a␣
 ↪number
df = raw_train
```

[ ]:
```
# Using Z-scores to filter out the outliers. Z-score < |3|
print(f'Shape of dataset before removing outliers{df.shape}')
z_scores = stats.zscore(df.drop(columns=["Diagnosis"])) # calculates z score␣
 ↪for each column except the diagnosis column
abs_z_scores = np.abs(z_scores) # take the absolute value of the z-scores
not_outliers = (abs_z_scores < 3).all(axis=1) # if all the z-scores are less␣
 ↪than 3, then it is not an outlier
cleaned = df[not_outliers] # remove the outliers
print(f'Shape of dataset after removing outliers {cleaned.shape}')
```

```
Shape of dataset before removing outliers(703, 31)
Shape of dataset after removing outliers (548, 31)
```

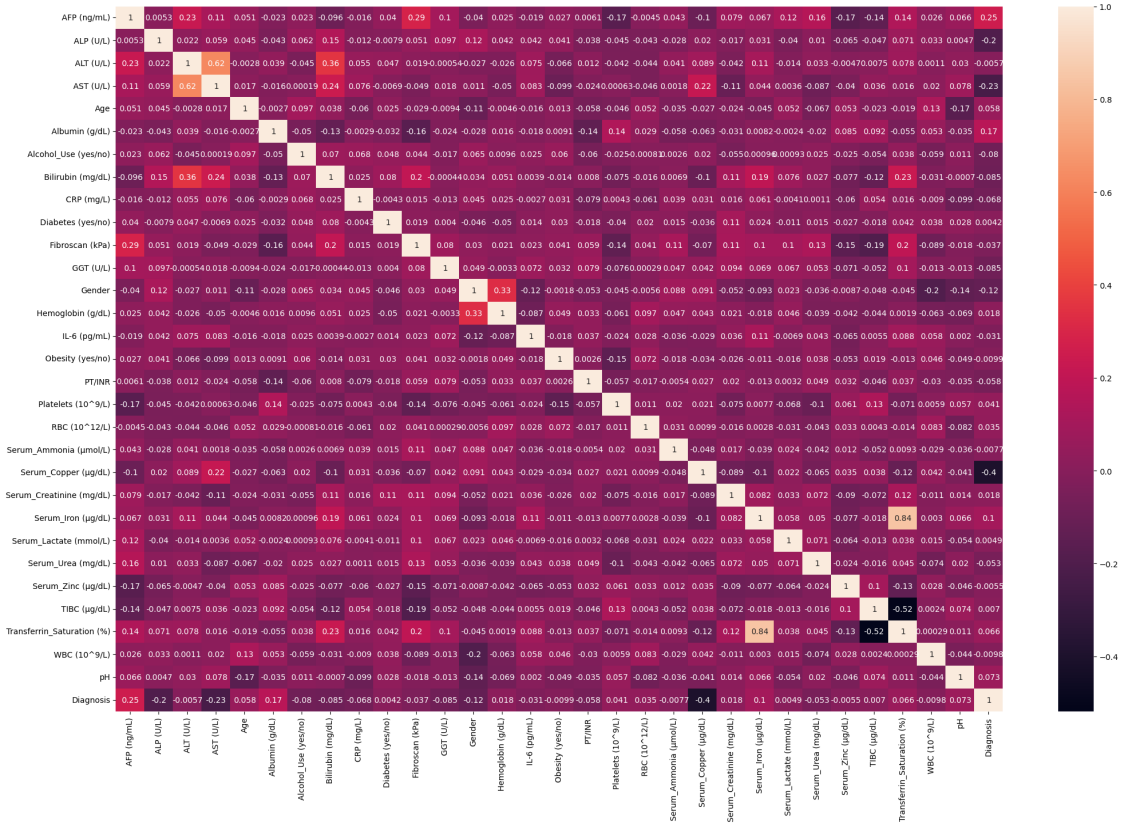
1.0.5 Data preprocessing and visualisation after cleaning

[ ]:
```
# Histogram
cleaned.hist(figsize=(10,6) , edgecolor="white") # edgecolor makes it look␣
 ↪nicer
plt.tight_layout()
plt.show()
```



```
[ ]: # convert Diagnosis column to numerical data using label encoder to make it␣
      ↪easier to plot the correlation matrix
     df = cleaned.copy()
     columns_to_encode = ['Diagnosis']
     le = LabelEncoder()
     for col in columns_to_encode:
         if df[col].dtype == 'object':
             df[col] = le.fit_transform(df[col])

     # Correlation matrix after cleaning
     corr_matrix = df.corr()
     fig_corr, ax_corr = plt.subplots(figsize=(25,16))
     sns.heatmap(data=corr_matrix,annot=True, ax=ax_corr)
     plt.show
```

```
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>
```



note: from the correatiom matrix, its shown that the amount of transferrin_Saturation and Iron has a high correlation of 0.84

```
[ ]: # convert the numeric data in the Diagnosis column to the original string values
Diagnosis = df['Diagnosis']
reverse_label_mapping = {
    0: 'Autoimmune Liver Diseases',
    1: 'Chirrosis',
    2: 'Drug-induced Liver Injury',
    3: 'Fatty Liver Disease',
    4: 'Healthy',
    5: 'Hepatitis',
    6: 'Liver Cancer'
}

df.loc[:, 'Diagnosis'] = [reverse_label_mapping[label] for label in␣
 ↪df['Diagnosis']]
```

Note: Uses the cleaned df, because the fit.transform convert the different disgnosis not as wanted, and makes the accuracy in piplines down below very low/wrong

```
[ ]: X = cleaned.drop(["Diagnosis"], axis=1) # contains all columns except traget␣
 ↪column Diagnosis
y = cleaned["Diagnosis"] # only contains the target column Diagnosis
```

```
[ ]: # Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42)

print("X_train: ",X_train.shape)
print("X_test: ",X_test.shape)
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
```

```
X_train:  (328, 30)
X_test:  (220, 30)
y_train:  (328,)
y_test:  (220,)
```

1.0.6 Modelling

pipline with regularization

```
[ ]: # Create Pipeline
pipeline_reg = Pipeline([
    ('scaler', StandardScaler()),  # # Standardize the features
    ('clf', LogisticRegression(max_iter=1000))  # Classifier with regularization
])

# Define Hyperparameters
parameters_reg = {
```

```
    'clf__C' : [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0] # Regularization␣
 ↪parameter
}

# Grid Search for Hyperparameters
grid_search_reg = GridSearchCV(estimator=pipeline_reg, # Use the pipeline as␣
 ↪the estimator
                        param_grid=parameters_reg,     # Search over the␣
 ↪hyperparameters defined in the dictionary
                        scoring='f1_macro',            # Use f1_macro as the␣
 ↪scoring metric
                        cv=6,                          # Perform 6-fold␣
 ↪cross-validation
                        n_jobs=-1)

grid_search_reg.fit(X, y) # fit the model

print(grid_search_reg.best_score_)
print(grid_search_reg.best_params_)

# source code:
# GridSearch : Lecture_15_cross_val_hyperpar_optim/scripts/grid_random_search.
 ↪ipynb
```

```
0.7186654717029679
{'clf__C': 0.1}
```

[ ]:
```
# Hyperparameter Tuning
best_estimator_reg = grid_search_reg.best_estimator_ # get the best estimator
best_estimator_reg.fit(X_train, y_train) # fit the best estimator
print('Accuracy: %.3f' % best_estimator_reg.score(X_test, y_test))

# source code: Lecture_15_cross_val_hyperpar_optim/scripts/grid_random_search.
 ↪ipynb
```

```
Accuracy: 0.714
```

[ ]:
```
y_pred = best_estimator_reg.predict(X_test) # predict the test data
conf_matrix = confusion_matrix(y_test, y_pred) #
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[16  0  1  0  3  0  0]
 [ 0 13  0  0  1  1  0]
 [ 1  0  7  0  1 12  0]
 [ 3  0  0  5 16  1  0]
 [ 0  0  0  1 83  0  0]
 [ 2  0 11  2  6 25  0]
```

```
 [ 0  0  0  0  0  1  8]]
```

Pipeline with kernel

[ ]:
```
# Create a pipeline
pipeline_svc = Pipeline([
    ('scaler', StandardScaler()), # Standardize the features
    ('reduce_dim', PCA()), # PCA as the dimensionality reduction technique
    ('clf', SVC())  # Classifier
])
# get parameters
pipeline_svc.get_params().keys()
# Define Hyperparameters
parameters_svc = {
    'reduce_dim__n_components': [2, 5, 10, 20, 30], # Number of components to␣
 ↪keep
    'clf__C' : [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0], # Regularization␣
 ↪parameter
    'clf__kernel': ['linear', 'rbf'], # Kernel type
    'clf__gamma': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0] # Kernel␣
 ↪coefficient
}

# Cross-Validation
grid_search_svc =GridSearchCV(estimator=pipeline_svc,  # Use the pipeline as␣
 ↪the estimator
                        param_grid=parameters_svc,     # Search over the␣
 ↪hyperparameters defined in the dictionary
                        scoring='f1_macro',            # Use f1_macro as the␣
 ↪scoring metric
                        cv=6,                          # Perform 6-fold␣
 ↪cross-validation
                        n_jobs=-1)

grid_search_svc.fit(X, y)

print(grid_search_svc.best_score_)
print(grid_search_svc.best_params_)

#  source code: Lecture_15_cross_val_hyperpar_optim/scripts/grid_random_search.
 ↪ipynb
```

```
0.7244510733164629
{'clf__C': 0.1, 'clf__gamma': 0.0001, 'clf__kernel': 'linear',
'reduce_dim__n_components': 30}
```

[ ]:
```
# Hyperparameter Tuning
best_estimator_svc = grid_search_svc.best_estimator_
```

```
best_estimator_svc.fit(X_train, y_train)
print('Accuracy: %.3f' % best_estimator_svc.score(X_test, y_test))

# source code: Lecture_15_cross_val_hyperpar_optim/scripts/grid_random_search.
 ↪ipynb
```

```
Accuracy: 0.695
```

[ ]:
```
#Confusion Matrix
y_pred = best_estimator_svc.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[16  0  0  0  4  0  0]
 [ 0 13  0  0  1  1  0]
 [ 3  1  6  0  1 10  0]
 [ 5  1  0  2 16  1  0]
 [ 0  0  0  1 83  0  0]
 [ 3  1 13  2  2 25  0]
 [ 0  0  0  0  0  1  8]]
```

1.0.7 Final evaluation

confusion matrix from the confusion matrices it shows that a lot of labels is missclassied as 5: 'Hepatitis'

accuracy the best pipeline was the knn, it had an accuracy at of 77,3 % (see kaggle submition below)

1.0.8 Kaggle submission

Note: the best results where when I only encoded the needed columns and did not remove outliers with z-score, and then ran the cell bellow.

[ ]:
```
X = raw_train.drop(["Diagnosis"], axis=1) # contains all columns except traget␣
 ↪column Diagnosis
y = raw_train["Diagnosis"] # only contains the traget column Diagnosis
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42)

# Create a pipeline
pipe_knn = Pipeline([('scaler', StandardScaler()),
                     ('lda', LDA()),
                     ('knn', KNeighborsClassifier())])

# Define a dictionary of hyperparameters to search over
param_grid_knn = {'knn__n_neighbors': [5, 10, 15, 20, 25],
                  'knn__leaf_size': [10, 20, 30, 40, 50],
```

```
                  'knn__weights': ['uniform', 'distance'],
                  'knn__p': [1, 2], # L1/L2
                  }

# Use GridSearchCV to search over hyperparameters for the KNeighborsClassifier
grid_search_knn = GridSearchCV(estimator=pipe_knn,        # Use the pipeline as␣
 ↪the estimator
                        param_grid=param_grid_knn, # Search over the␣
 ↪hyperparameters defined in the dictionary
                        scoring='f1_macro',        # Use f1_macro as the␣
 ↪scoring metric
                        cv=5,                      # Perform 6-fold␣
 ↪cross-validation
                        n_jobs=-1)

# Fit the grid search object on the training data
grid_search_knn.fit(X, y)

# Print the best score and best set of hyperparameters
print('%.3f'% grid_search_knn.best_score_)
print('the best parameters:',grid_search_knn.best_params_)

# Hyperparameter Tuning
best_estimator_knn = grid_search_knn.best_estimator_
best_estimator_knn.fit(X_train, y_train)
print('Accuracy: %.3f' % best_estimator_knn.score(X_test, y_test))
```

```
0.717
the best parameters: {'knn__leaf_size': 10, 'knn__n_neighbors': 15, 'knn__p': 2,
'knn__weights': 'uniform'}
Accuracy: 0.773
```

[ ]:
```
y_pred = best_estimator_knn.predict(test_data)
df_submission = pd.DataFrame({'Diagnosis': y_pred})
df_submission.reset_index(level=0, inplace=True)

df_submission.columns = ['index', 'Diagnosis']
df_submission.to_csv('submission_knn.csv', index=False)
```

Note: the best score i got on Kaggle was 0.7538 with a knn classifier, but i do not have the code now. the second best is with this code over, ang got the score 0.7424

1.0.9 confusion matrix of the best classifer

[ ]:
```
y_pred = best_estimator_knn.predict(X_test) # predict the test data
conf_matrix = confusion_matrix(y_test, y_pred) #
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[ 20   0   3   1   3   3   0]
 [  0  32   0   1   2   1   0]
 [  0   0   9   0   1  14   0]
 [  1   0   0   4  19   1   0]
 [  0   0   0   0 101   1   0]
 [  1   0   6   0   4  27   0]
 [  0   0   0   0   0   2  25]]
```

Note: as expected, this confusion matrix has less wrong guesses than the other classfiers, as it can bee seen here compared to the other cofusion matrices

1.0.10 ROC curve

[ ]:
```
# convert diagnosis to not healthy and healthy
raw_train['Diagnosis'] = np.where(raw_train['Diagnosis'] == 'Healthy', 0, 1)
```

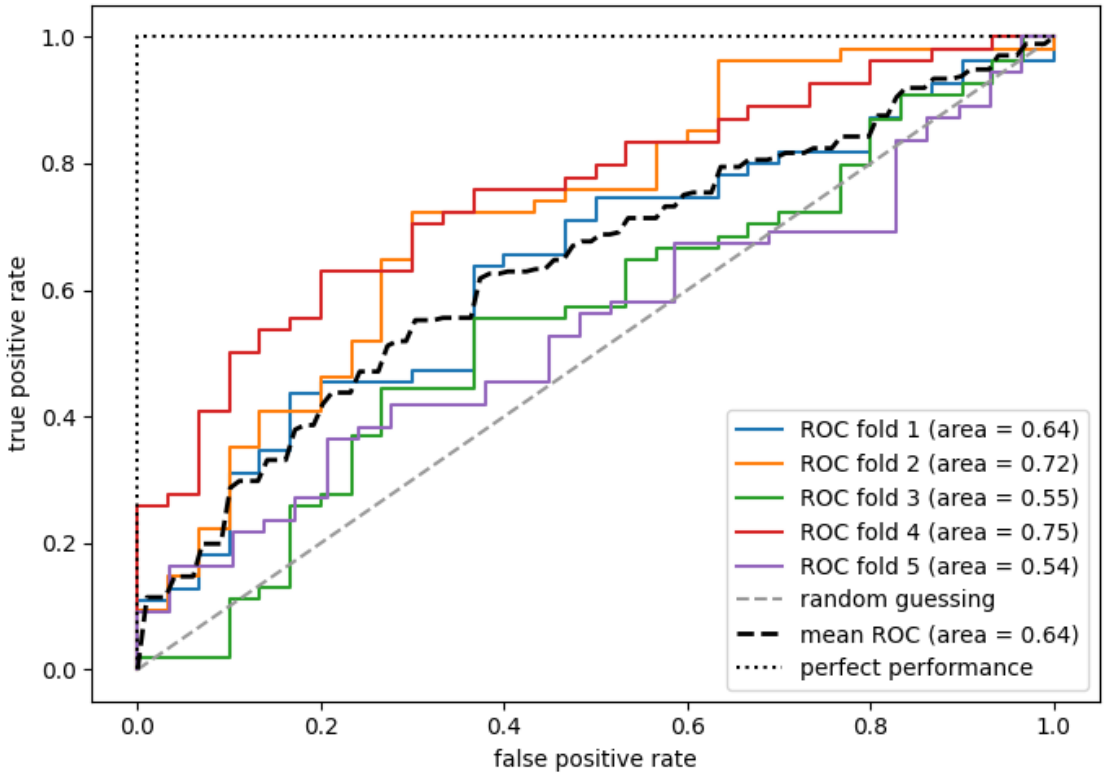
[ ]:
```
X = raw_train.drop(["Diagnosis"], axis=1) # contains all columns except traget␣
 ↪column Diagnosis
y = raw_train["Diagnosis"] # only contains the traget column Diagnosis

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=43)
```

[ ]:
```
# convert to numpy arrays
X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

[ ]:
```
# Create ROC curve


# Set up pipeline with scale, decomposer and classifyer
pipeline_reg = Pipeline([
    ('scaler', StandardScaler()),  # Standardize the features
    ('clf', LogisticRegression(max_iter=1000))  # Classifier with regularization
])

# Reduced variable set
X_train2 = X_train[:, [4, 11, 14]] # Selecting the 3rd, 10th and 13th columns

# Cross-validation specification
```

```
cv = list(StratifiedKFold(n_splits=5).split(X_train, y_train)) # 5 fold␣
 ↪cross-validation

fig = plt.figure(figsize=(7, 5))

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

# Loop through folds of CV
for i, (train, test) in enumerate(cv):
    probas = pipeline_reg.fit(X_train2[train],
                         y_train[train]).predict_proba(X_train2[test]) #␣
 ↪Predict probability of classes

    # False Positive and True Positive Rates (thresholds for the decision␣
 ↪function)
    fpr, tpr, thresholds = roc_curve(y_train[test], # true labels
                                     probas[:, 1], # probability of class 1
                                     pos_label=1) # positive class is 1
    # Add to mean True Predictive Rate in a smoothed variant (interpolated)
    mean_tpr += np.interp(mean_fpr, fpr, tpr) # Interpolate the true positive␣
 ↪rate at the mean false positive rate
    roc_auc = auc(fpr, tpr) # Area under the curve
    plt.plot(fpr,
             tpr,
             label='ROC fold %d (area = %0.2f)'
                   % (i+1, roc_auc))

plt.plot([0, 1],
         [0, 1],
         linestyle='--',
         color=(0.6, 0.6, 0.6),
         label='random guessing')

# Average True Positive Rate
mean_tpr /= len(cv)
mean_tpr[0] = 0.0
mean_tpr[-1] = 1.0

# Average AUC
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',
         label='mean ROC (area = %0.2f)' % mean_auc, lw=2)
plt.plot([0, 0, 1],
         [0, 1, 1],
         linestyle=':',
```

```
         color='black',
         label='perfect performance')

plt.xlim([-0.05, 1.05]) # x-axis limits
plt.ylim([-0.05, 1.05]) # y-axis limits
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend(loc="lower right")

plt.tight_layout()
plt.show()


# source code: Lecture_16_evaluation_metrics/scripts/ROC_AUC_with_LR.ipynb
```



Note: the best results is with 4 folds(the red line)