

Notebook 4 - Modelling of data derived from network analysis of metabolomic measurements from plasma samples from COVID patients.

This notebook is a part of a project with the Center for Systems Biology at the University of Iceland. The goal here is to:

- Take the results from a correlation network analysis of the untargeted metabolomic measurements of the plasma samples (which includes the module eigenvectors and the assignment of features into different modules) along with the original metabolomic dataframe and the clinical data, and create a model that is able to predict the composite outcome of the patients.
- After finding the modules that are most important for the prediction of composite outcomes, we look into these modules, what metabolic pathways they represent and what features are most representative of them.
- The most representative features (who have now been putatively annotated) are tried out as predictors of composite outcome. The results of that feature selection-based model is then compared to a model based on all >5000 features present in the untargeted metabolomic measurement.

Import modules:

```
In [1]: import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, to_rgb
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import stratified_fold
from sklearn.metrics import mean_squared_error
from sklearn.metrics import pearsonr
```

Define functions

```
In [2]: def impute_df(dataframe):
    """
    A function to impute dataframe using KNN while still keeping the dataframe structure
    """
    df_columns = dataframe.columns
    dat_index = dataframe.index.tolist()
    imputah = KNNImputer()
    dat_imputed = StandardScaler()
    dat_scaled = scaler.fit_transform(dataframe)
    dat_imputed = imputah.fit_transform(dat_scaled)
    dat_orig_transformed = scaler.inverse_transform(dat_imputed)
    return_df = pd.DataFrame(dat_orig_transformed, columns = df_columns, index = dat_index)
    return return_df

def remove_columns_with_missing_data(dataframe, threshold_percentage):
    """
    A function to remove columns from a dataframe that have more than threshold_percentage missing values
    """
    # First remove rows that have NaN for all values:
    idx_remove = []
    for column_val in dataframe.columns:
        if dataframe[column_val].isna().sum() > threshold:
            idx_remove.append(column_val)
    df_out = dataframe.drop(idx_remove, axis=1)
    return df_out

def indices(lst, element):
    """
    result = [i for i, x in enumerate(lst) if x == element]
    """
    result = []
    offset = -1
    while True:
        try:
            offset = lst.index(element, offset+1)
        except ValueError:
            return result
        result.append(offset)

def t_test(X_dat, y_dat):
    """
    A function to perform two sample t-test on all features in X_dat, where the rows are grouped based on y_dat
    coefficient for multiple testing takes place here, but could easily be added.
    """
    list_response_var = list(y_dat)
    p_vals = []
    t_vals = []
    id_group1 = indices(list_response_var, 1) # Get ids of all binaries of 1
    id_group0 = indices(list_response_var, 0) # Get ids of all binaries of 0
    for col in X_dat.columns:
        sample1 = X_dat.loc[id_group1, col].tolist()
        sample0 = X_dat.loc[id_group0, col].tolist()
        mean1, mean0 = np.mean(sample1), np.mean(sample0)
        std1, std0 = np.std(sample1), np.std(sample0)
        n1, n0 = len(sample1), len(sample0)
        var_sample1, var_sample0 = np.var(sample1, ddof=1), np.var(sample0, ddof=1)
        var = ((n1-1)*var_sample1 + (n0-1)*var_sample0) / (n1+n0-2)
        std_error = np.sqrt(var * (1.0 / n1 + 1.0 / n0))
        t = abs(mean1 - mean0) / std_error
        p_val = 2 * (1 - stats.t.cdf(abs(t), df=n1+n0-2))
        p_vals.append(p_val)
        t_vals.append(t)
    return t_vals, p_vals
```

Load data and make some adjustments to the clinical data:

```
In [3]: mes = pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/Mes_data/patient_mes')
pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/patient_mes')
pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/patient_mes')
clin = pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/Pinal_data/clin')
clin.reset_index(inplace=True)

# Begin by getting only the clinical data with the non-outlier patient samples from the metabolomics network an
clin['sex_binary'] = clin['Sex']
clin['sex_binary'].replace('Male', 0, inplace=True)
clin['sex_binary'].replace('Female', 1, inplace=True)
clin['HTN_binary'] = clin['HTN']
clin['HTN_binary'].replace('Yes', 1, inplace=True)
clin['HTN_binary'].replace('No', 0, inplace=True)
clin['diabetes_binary'] = clin['Diabetes']
clin['diabetes_binary'].replace('Yes', 1, inplace=True)
clin['diabetes_binary'].replace('No', 0, inplace=True)
clin = clin.loc[:, clin.columns != 'Sex']
clin = clin.loc[:, clin.columns != 'HTN']
clin = clin.loc[:, clin.columns != 'Diabetes']

ttypes = clin.dtypes
for column_val in clin.columns[6:]:
    if ttypes[column_val] == 'O':
        clin[column_val] = clin[column_val].str.replace(' ', '-')
        clin[column_val] = pd.to_numeric(clin[column_val], errors='coerce')

# Remove clinical data with more than 10% missing data (clin_20)
clin = remove_columns_with_missing_data(clin_20)

# Select appropriate variables to keep in data:
clin_orig = clin.copy()
clin = impute_df(clin)
clin = impute_df(clin)

# Round clin_orig subject identifier to integers:
clin_orig['Subject Identifier for the Study'] = round(clin_orig['Subject Identifier for the Study']).astype(int)

# Set response variable as integers:
clin['Composite'] = clin['Composite'].astype(int)
clin_orig['Composite'] = clin_orig['Composite'].astype(int)

# Load data from the network analysis in R:
merged_columns = pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/R_data/merged_columns')
findat = pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/Pinal_data/findat')
datkme = pd.read_csv('C:/Users/sigurdur.karvelsson/OneDrive - Alvotech/Documents/Other/COVID_project_CSB/datkme')

# Prepare R and y data:
new_mes = pd.concat([mes, clin.loc[:, clin.columns != 'Composite']], axis=1)
X = new_mes
y = clin['Composite']
```

Use a GRID-search to search for optimal parameters for each model, then run the LOO-fold CV to get an estimate of predictive capabilities of top modules vs all modules:

```
In [4]: # Run classifier with cross-validation and plot ROC curves
X_to_use = np.array(mes)
y_to_use = y.values

classifier = RandomForestClassifier()
# Try out a grid of parameters:
parameters = {'max_features':np.arange(2,4), 'n_estimators':[1000,2000], 'max_depth':[2,3,4], 'random_state':[0]}
classifier_opt = GridSearchCV(classifier, parameters, cv=5)

# Fit on entire dataset to get optimal parameters:
classifier_opt.fit(X_to_use, y_to_use)
params_to_use = classifier_opt.best_estimator_.get_params()

cv = StratifiedFold(n_splits=5)
classifier_for_cv = RandomForestClassifier(*params_to_use)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)
importance_list = []

fig, ax = plt.subplots(figsize=(8,6))
for i, (train, test) in enumerate(cv.split(X_to_use, y_to_use)):
    classifier_for_cv.fit(X_to_use[train], y_to_use[train])
    tprs.append(interpr(classifier_for_cv.feature_importances_))
    # Scores = classifier_for_cv.predict_proba(X_to_use[test])[:,1]
    fpr, roc_auc_score(y_to_use[test], scores)
    viz = RocCurveDisplay.from_estimator(classifier_for_cv, X_to_use[test], y_to_use[test], name="ROC fold {}".format(i+1), alpha=0.3, ax=ax, lw=1, lw2=1, lw3=1, lw4=1, lw5=1, lw6=1, lw7=1, lw8=1, lw9=1, lw10=1, lw11=1, lw12=1, lw13=1, lw14=1, lw15=1, lw16=1, lw17=1, lw18=1, lw19=1, lw20=1, lw21=1, lw22=1, lw23=1, lw24=1, lw25=1, lw26=1, lw27=1, lw28=1, lw29=1, lw30=1, lw31=1, lw32=1, lw33=1, lw34=1, lw35=1, lw36=1, lw37=1, lw38=1, lw39=1, lw40=1, lw41=1, lw42=1, lw43=1, lw44=1, lw45=1, lw46=1, lw47=1, lw48=1, lw49=1, lw50=1, lw51=1, lw52=1, lw53=1, lw54=1, lw55=1, lw56=1, lw57=1, lw58=1, lw59=1, lw60=1, lw61=1, lw62=1, lw63=1, lw64=1, lw65=1, lw66=1, lw67=1, lw68=1, lw69=1, lw70=1, lw71=1, lw72=1, lw73=1, lw74=1, lw75=1, lw76=1, lw77=1, lw78=1, lw79=1, lw80=1, lw81=1, lw82=1, lw83=1, lw84=1, lw85=1, lw86=1, lw87=1, lw88=1, lw89=1, lw90=1, lw91=1, lw92=1, lw93=1, lw94=1, lw95=1, lw96=1, lw97=1, lw98=1, lw99=1, lw100=1, lw101=1, lw102=1, lw103=1, lw104=1, lw105=1, lw106=1, lw107=1, lw108=1, lw109=1, lw110=1, lw111=1, lw112=1, lw113=1, lw114=1, lw115=1, lw116=1, lw117=1, lw118=1, lw119=1, lw120=1, lw121=1, lw122=1, lw123=1, lw124=1, lw125=1, lw126=1, lw127=1, lw128=1, lw129=1, lw130=1, lw131=1, lw132=1, lw133=1, lw134=1, lw135=1, lw136=1, lw137=1, lw138=1, lw139=1, lw140=1, lw141=1, lw142=1, lw143=1, lw144=1, lw145=1, lw146=1, lw147=1, lw148=1, lw149=1, lw150=1, lw151=1, lw152=1, lw153=1, lw154=1, lw155=1, lw156=1, lw157=1, lw158=1, lw159=1, lw160=1, lw161=1, lw162=1, lw163=1, lw164=1, lw165=1, lw166=1, lw167=1, lw168=1, lw169=1, lw170=1, lw171=1, lw172=1, lw173=1, lw174=1, lw175=1, lw176=1, lw177=1, lw178=1, lw179=1, lw180=1, lw181=1, lw182=1, lw183=1, lw184=1, lw185=1, lw186=1, lw187=1, lw188=1, lw189=1, lw190=1, lw191=1, lw192=1, lw193=1, lw194=1, lw195=1, lw196=1, lw197=1, lw198=1, lw199=1, lw200=1, lw201=1, lw202=1, lw203=1, lw204=1, lw205=1, lw206=1, lw207=1, lw208=1, lw209=1, lw210=1, lw211=1, lw212=1, lw213=1, lw214=1, lw215=1, lw216=1, lw217=1, lw218=1, lw219=1, lw220=1, lw221=1, lw222=1, lw223=1, lw224=1, lw225=1, lw226=1, lw227=1, lw228=1, lw229=1, lw230=1, lw231=1, lw232=1, lw233=1, lw234=1, lw235=1, lw236=1, lw237=1, lw238=1, lw239=1, lw240=1, lw241=1, lw242=1, lw243=1, lw244=1, lw245=1, lw246=1, lw247=1, lw248=1, lw249=1, lw250=1, lw251=1, lw252=1, lw253=1, lw254=1, lw255=1, lw256=1, lw257=1, lw258=1, lw259=1, lw260=1, lw261=1, lw262=1, lw263=1, lw264=1, lw265=1, lw266=1, lw267=1, lw268=1, lw269=1, lw270=1, lw271=1, lw272=1, lw273=1, lw274=1, lw275=1, lw276=1, lw277=1, lw278=1, lw279=1, lw280=1, lw281=1, lw282=1, lw283=1, lw284=1, lw285=1, lw286=1, lw287=1, lw288=1, lw289=1, lw290=1, lw291=1, lw292=1, lw293=1, lw294=1, lw295=1, lw296=1, lw297=1, lw298=1, lw299=1, lw300=1, lw301=1, lw302=1, lw303=1, lw304=1, lw305=1, lw306=1, lw307=1, lw308=1, lw309=1, lw310=1, lw311=1, lw312=1, lw313=1, lw314=1, lw315=1, lw316=1, lw317=1, lw318=1, lw319=1, lw320=1, lw321=1, lw322=1, lw323=1, lw324=1, lw325=1, lw326=1, lw327=1, lw328=1, lw329=1, lw330=1, lw331=1, lw332=1, lw333=1, lw334=1, lw335=1, lw336=1, lw337=1, lw338=1, lw339=1, lw340=1, lw341=1, lw342=1, lw343=1, lw344=1, lw345=1, lw346=1, lw347=1, lw348=1, lw349=1, lw350=1, lw351=1, lw352=1, lw353=1, lw354=1, lw355=1, lw356=1, lw357=1, lw358=1, lw359=1, lw360=1, lw361=1, lw362=1, lw363=1, lw364=1, lw365=1, lw366=1, lw367=1, lw368=1, lw369=1, lw370=1, lw371=1, lw372=1, lw373=1, lw374=1, lw375=1, lw376=1, lw377=1, lw378=1, lw379=1, lw380=1, lw381=1, lw382=1, lw383=1, lw384=1, lw385=1, lw386=1, lw387=1, lw388=1, lw389=1, lw390=1, lw391=1, lw392=1, lw393=1, lw394=1, lw395=1, lw396=1, lw397=1, lw398=1, lw399=1, lw400=1, lw401=1, lw402=1, lw403=1, lw404=1, lw405=1, lw406=1, lw407=1, lw408=1, lw409=1, lw410=1, lw411=1, lw412=1, lw413=1, lw414=1, lw415=1, lw416=1, lw417=1, lw418=1, lw419=1, lw420=1, lw421=1, lw422=1, lw423=1, lw424=1, lw425=1, lw426=1, lw427=1, lw428=1, lw429=1, lw430=1, lw431=1, lw432=1, lw433=1, lw434=1, lw435=1, lw436=1, lw437=1, lw438=1, lw439=1, lw440=1, lw441=1, lw442=1, lw443=1, lw444=1, lw445=1, lw446=1, lw447=1, lw448=1, lw449=1, lw450=1, lw451=1, lw452=1, lw453=1, lw454=1, lw455=1, lw456=1, lw457=1, lw458=1, lw459=1, lw460=1, lw461=1, lw462=1, lw463=1, lw464=1, lw465=1, lw466=1, lw467=1, lw468=1, lw469=1, lw470=1, lw471=1, lw472=1, lw473=1, lw474=1, lw475=1, lw476=1, lw477=1, lw478=1, lw479=1, lw480=1, lw481=1, lw482=1, lw483=1, lw484=1, lw485=1, lw486=1, lw487=1, lw488=1, lw489=1, lw490=1, lw491=1, lw492=1, lw493=1, lw494=1, lw495=1, lw496=1, lw497=1, lw498=1, lw499=1, lw500=1, lw501=1, lw502=1, lw503=1, lw504=1, lw505=1, lw506=1, lw507=1, lw508=1, lw509=1, lw510=1, lw511=1, lw512=1, lw513=1, lw514=1, lw515=1, lw516=1, lw517=1, lw518=1, lw519=1, lw520=1, lw521=1, lw522=1, lw523=1, lw524=1, lw525=1, lw526=1, lw527=1, lw528=1, lw529=1, lw530=1, lw531=1, lw532=1, lw533=1, lw534=1, lw535=1, lw536=1, lw537=1, lw538=1, lw539=1, lw540=1, lw541=1, lw542=1, lw543=1, lw544=1, lw545=1, lw546=1, lw547=1, lw548=1, lw549=1, lw550=1, lw551=1, lw552=1, lw553=1, lw554=1, lw555=1, lw556=1, lw557=1, lw558=1, lw559=1, lw560=1, lw561=1, lw562=1, lw563=1, lw564=1, lw565=1, lw566=1, lw567=1, lw568=1, lw569=1, lw570=1, lw571=1, lw572=1, lw573=1, lw574=1, lw575=1, lw576=1, lw577=1, lw578=1, lw579=1, lw580=1, lw581=1, lw582=1, lw583=1, lw584=1, lw585=1, lw586=1, lw587=1, lw588=1, lw589=1, lw590=1, lw591=1, lw592=1, lw593=1, lw594=1, lw595=1, lw596=1, lw597=1, lw598=1, lw599=1, lw600=1, lw601=1, lw602=1, lw603=1, lw604=1, lw605=1, lw606=1, lw607=1, lw608=1, lw609=1, lw610=1, lw611=1, lw612=1, lw613=1, lw614=1, lw615=1, lw616=1, lw617=1, lw618=1, lw619=1, lw620=1, lw621=1, lw622=1, lw623=1, lw624=1, lw625=1, lw626=1, lw627=1, lw628=1, lw629=1, lw630=1, lw631=1, lw632=1, lw633=1, lw634=1, lw635=1, lw636=1, lw637=1, lw638=1, lw639=1, lw640=1, lw641=1, lw642=1, lw643=1, lw644=1, lw645=1, lw646=1, lw647=1, lw648=1, lw649=1, lw650=1, lw651=1, lw652=1, lw653=1, lw654=1, lw655=1, lw656=1, lw657=1, lw658=1, lw659=1, lw660=1, lw661=1, lw662=1, lw663=1, lw664=1, lw665=1, lw666=1, lw667=1, lw668=1, lw669=1, lw670=1, lw671=1, lw672=1, lw673=1, lw674=1, lw675=1, lw676=1, lw677=1, lw678=1, lw679=1, lw680=1, lw681=1, lw682=1, lw683=1, lw684=1, lw685=1, lw686=1, lw687=1, lw688=1, lw689=1, lw690=1, lw691=1, lw692=1, lw693=1, lw694=1, lw695=1, lw696=1, lw697=1, lw698=1, lw699=1, lw700=1, lw701=1, lw702=1, lw703=1, lw704=1, lw705=1, lw706=1, lw707=1, lw708=1, lw709=1, lw710=1, lw711=1, lw712=1, lw713=1, lw714=1, lw715=1, lw716=1, lw717=1, lw718=1, lw719=1, lw720=1, lw721=1, lw722=1, lw723=1, lw724=1, lw725=1, lw726=1, lw727=1, lw728=1, lw729=1, lw730=1, lw731=1, lw732=1, lw733=1, lw734=1, lw735=1, lw736=1, lw737=1, lw738=1, lw739=1, lw740=1, lw741=1, lw742=1, lw743=1, lw744=1, lw745=1, lw746=1, lw747=1, lw748=1, lw749=1, lw750=1, lw751=1, lw752=1, lw753=1, lw754=1, lw755=1, lw756=1, lw757=1, lw758=1, lw759=1, lw760=1, lw761=1, lw762=1, lw763=1, lw764=1, lw765=1, lw766=1, lw767=1, lw768=1, lw769=1, lw770=1, lw771=1, lw772=1, lw773=1, lw774=1, lw775=1, lw776=1, lw777=1, lw778=1, lw779=1, lw780=1, lw781=1, lw782=1, lw783=1, lw784=1, lw785=1, lw786=1, lw787=1, lw788=1, lw789=1, lw790=1, lw791=1, lw792=1, lw793=1, lw794=1, lw795=1, lw796=1, lw797=1, lw798=1, lw799=1, lw800=1, lw801=1, lw802=1, lw803=1, lw804=1, lw805=1, lw806=1, lw807=1, lw808=1, lw809=1, lw810=1, lw811=1, lw812=1, lw813=1, lw814=1, lw815=1, lw816=1, lw817=1, lw818=1, lw819=1, lw820=1, lw821=1, lw822=1, lw823=1, lw824=1, lw825=1, lw826=1, lw827=1, lw828=1, lw829=1, lw830=1, lw831=1, lw832=1, lw833=1, lw834=1, lw835=1, lw836=1, lw837=1, lw838=1, lw839=1, lw840=1, lw841=1, lw842=1, lw843=1, lw844=1, lw845=1, lw846=1, lw847=1, lw848=1, lw849=1, lw850=1, lw851=1, lw852=1, lw853=1, lw854=1, lw855=1, lw856=1, lw857=1, lw858=1, lw859=1, lw860=1, lw861=1, lw862=1, lw863=1, lw864=1, lw865=1, lw866=1, lw867=1, lw868=1, lw869=1, lw870=1, lw871=1, lw872=1, lw873=1, lw874=1, lw875=1, lw876=1, lw877=1, lw878=1, lw879=1, lw880=1, lw881=1, lw882=1, lw883=1, lw884=1, lw885=1, lw886=1, lw887=1, lw888=1, lw889=1, lw890=1, lw891=1, lw892=1, lw893=1, lw894=1, lw895=1, lw896=1, lw897=1, lw898=1, lw899=1, lw900=1, lw901=1, lw902=1, lw903=1, lw904=1, lw905=1, lw906=1, lw907=1, lw908=1, lw909=1, lw910=1, lw911=1, lw912=1, lw913=1, lw914=1, lw915=1, lw916=1, lw917=1, lw918=1, lw919=1, lw920=1, lw921=1, lw922=1, lw923=1, lw924=1, lw925=1, lw926=1, lw927=1, lw928=1, lw929=1, lw930=1, lw931=1, lw932=1, lw933=1, lw934=1, lw935=1, lw936=1, lw937=1, lw938=1, lw939=1, lw940=1, lw941=1, lw942=1, lw943=1, lw944=1, lw945=1, lw946=1, lw947=1, lw948=1, lw949=1, lw950=1, lw951=1, lw952=1, lw953=1, lw954=1, lw955=1, lw956=1, lw957=1, lw958=1, lw959=1, lw960=1, lw961=1, lw962=1, lw963=1, lw964=1, lw965=1, lw966=1, lw967=1, lw968=1, lw969=1, lw970=1, lw971=1, lw972=1, lw973=1, lw974=1, lw975=1, lw976=1, lw977=1, lw978=1, lw979=1, lw980=1, lw981=1, lw982=1, lw983=1, lw984=1, lw985=1, lw986=1, lw987=1, lw988=1, lw989=1, lw990=1, lw991=1, lw992=1, lw993=1, lw994=1, lw995=1, lw996=1, lw997=1, lw998=1, lw999=1, lw1000=1, lw1001=1, lw1002=1, lw1003=1, lw1004=1, lw1005=1, lw1006=1, lw1007=1, lw1008=1, lw1009=1, lw1010=1, lw1011=1, lw1012=1, lw1013=1, lw1014=1, lw1015=1, lw1016=1, lw1017=1, lw1018=1, lw1019=1, lw1020=1, lw1021=1, lw1022=1, lw1023=1, lw1024=1, lw1025=1, lw1026=1, lw1027=1, lw1028=1, lw1029=1, lw1030=1, lw1031=1, lw1032=1, lw1033=1, lw1034=1, lw1035=1, lw1036=1, lw1037=1, lw1038=1, lw1039=1, lw1040=1, lw1041=1, lw1042=1, lw1043=1, lw1044=1, lw1045=1, lw1046=1, lw1047=1, lw1048=1, lw1049=1, lw1050=1, lw1051=1, lw1052=1, lw1053=1, lw1054=1, lw1055=1, lw1056=1, lw1057=1, lw1058=1, lw1059=1, lw1060=1, lw1061=1, lw1062=1, lw1063=1, lw1064=1, lw1065=1, lw1066=1, lw1067=1, lw1068=1, lw1069=1, lw1070=1, lw1071=1, lw1072=1, lw1073=1, lw1074=1, lw1075=1, lw1076=1, lw1077=1, lw1078=1, lw1079=1, lw1080=1, lw1081=1, lw1082=1, lw1083=1, lw1084=1, lw1085=1, lw1086=1, lw1087=1, lw1088=1, lw1089=1, lw1090=1, lw1091=1, lw1092=1, lw1093=1, lw1094=1, lw1095=1, lw1096=1, lw1097=1, lw1098=1, lw1099=1, lw1100=1, lw1101=1, lw1102=1, lw1103=1, lw1104=1, lw1105=1, lw1106=1, lw1107=1, lw1108=1, lw1109=1, lw1110=1, lw1111=1, lw1112=1, lw1113=1, lw1114=1, lw1115=1, lw1116=1, lw1117=1, lw1118=1, lw1119=1, lw1120=1, lw1121=1, lw1122=1, lw1123=1, lw1124=1, lw1125=1, lw1126=1, lw1127=1, lw1128=1, lw1129=1, lw1130=1, lw1131=1, lw1132=1, lw1133=1, lw1134=1, lw1135=1, lw1136=1, lw1137=1, lw1138=1, lw1139=1, lw1140=1, lw1141=1, lw1142=1, lw1143=1, lw1144=1, lw1145=1, lw1146=1, lw1147=1, lw1148=1, lw1149=1, lw1150=1, lw1151=1, lw1152=1, lw1153=1, lw1154=1, lw1155=1, lw1156=1, lw1157=1, lw1158=1, lw1159=1, lw1160=1, lw1161=1, lw1162=1, lw1163=1, lw1164=1, lw1165=1, lw1166=1, lw1167=1, lw1168=1, lw1169=1, lw1170=1, lw1171=1, lw1172=1, lw1173=1, lw1174=1, lw1175=1, lw1176=1, lw1177=1, lw1178=1, lw1179=1, lw1180=1, lw1181=1, lw1182=1, lw1183=1, lw1184=1, lw1185=1, lw1186=1, lw1187=1, lw1188=1, lw1189=1, lw1190=1, lw1191=1, lw1192=1, lw1193=1, lw1194=1, lw1195=1, lw1196=1, lw1197=1, lw1198=1, lw1199=1, lw1200=1, lw1201=1, lw1202=1, lw1203=1, lw1204=1, lw1205=1, lw1206=1, lw1207=1, lw1208=1, lw1209=1, lw1210=1, lw1211=1, lw1212=1, lw1213=1, lw1214=1, lw1215=1, lw1216=1, lw1217=1, lw1218=1, lw1219=1, lw1220=1, lw1221=1, lw1222=1, lw1223=1, lw1224=1, lw1225=1, lw1226=1, lw1227=1, lw1228=1, lw1229=1, lw1230=1, lw1231=1, lw1232=1, lw1233=1, lw1234=1, lw1235=1, lw1236=1, lw1237=1, lw1238=1, lw1239=1, lw1240=1, lw1241=1, lw1242=1, lw1243=1, lw1244=1, lw1245=1, lw1246=1, lw1247=1, lw1248=1, lw1249=1, lw1250=1, lw1251=1, lw1252=1, lw1253=1, lw1254=1, lw1255=1, lw1256=1, lw1257=1, lw1258=1, lw1259=1, lw1260=1, lw1261=1, lw1262=1, lw1263=1, lw1264=1, lw1265=1, lw1266=1, lw1267=1, lw1268=1, lw1269=1, lw1270=1, lw1271=1, lw1272=1, lw1273=1, lw1274=1, lw1275=1, lw1276=1, lw1277=1, lw1278=1, lw1279=1, lw1280=1, lw1281=1, lw1282=1, lw1283=1, lw1284=1, lw1285=1, lw1286=1, lw1287=1, lw1288=1, lw1289=1, lw1290=1, lw1291=1, lw1292=1, lw1293=1, lw1294=1, lw1295=1, lw1296=1, lw1297=1, lw1298=1, lw1299=1, lw1300=1, lw1301=1, lw1302=1, lw1303=1, lw1304=1, lw1305=1, lw1306=1, lw1307=1, lw1308=1, lw1309=1, lw1310=1, lw1311=1, lw1312=1, lw1313=1, lw1314=1, lw1315=1, lw1316=1, lw1317=1, lw1318=1, lw1319=1, lw1320=1, lw1321=1, lw1322=1, lw1323=1, lw1324=1, lw1325=1, lw1326=1, lw1327=1, lw1328=1, lw1329=1, lw1330=1, lw1331=1, lw1332=1, lw1333=1, lw1334=1, lw1335=1, lw1336=1, lw1337=1, lw1338=1, lw1339=1, lw1340=1, lw1341=1, lw1342=1, lw1343=1, lw1344=1, lw1345=1, lw1346=1, lw1347=1, lw1348=1, lw1349=1, lw1350=1, lw1351=1, lw1352=1, lw1353=1, lw1354=1, lw1355=1, lw1356=1, lw1357=1, lw1358=1, lw1359=1, lw1360=1, lw1361=1, lw1362=1, lw1363=1, lw1364=1, lw1365=1, lw1366=1, lw1367=1, lw1368=1, lw1369=1, lw1370=1,
```



```
[15]: X_to_use = np.array(findat('SAPS2','SOFA','NT-pro-BNP','Urea','Creatinine','Troponin','Platelets'))
      y_to_use = y.values

classifier = RandomForestClassifier()
# Try out a grid of parameters:
parameters = {'max_features':np.arange(2,4), 'n_estimators':[1000,2000], 'max_depth':[2,3,4], 'random_state':[0]}
classifier_opt = GridSearchCV(classifier, parameters, cv = 5)

# Fit on entire dataset to get optimal parameters:
classifier_opt.fit(X_to_use,y_to_use)
params_to_use = classifier_opt.best_estimator_.get_params()

cv = StratifiedFold(n_splits=5)
classifier_for_cv = RandomForestClassifier(**params_to_use)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots(figsize=(8,6))
for i, (train, test) in enumerate(cv.split(X_to_use, y_to_use)):
    classifier_for_cv.fit(X_to_use[train], y_to_use[train])
    viz = RocCurveDisplay.from_estimator(
        classifier_for_cv,
        X_to_use[test],
        y_to_use[test],
        name="ROC fold {}".format(i+1),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

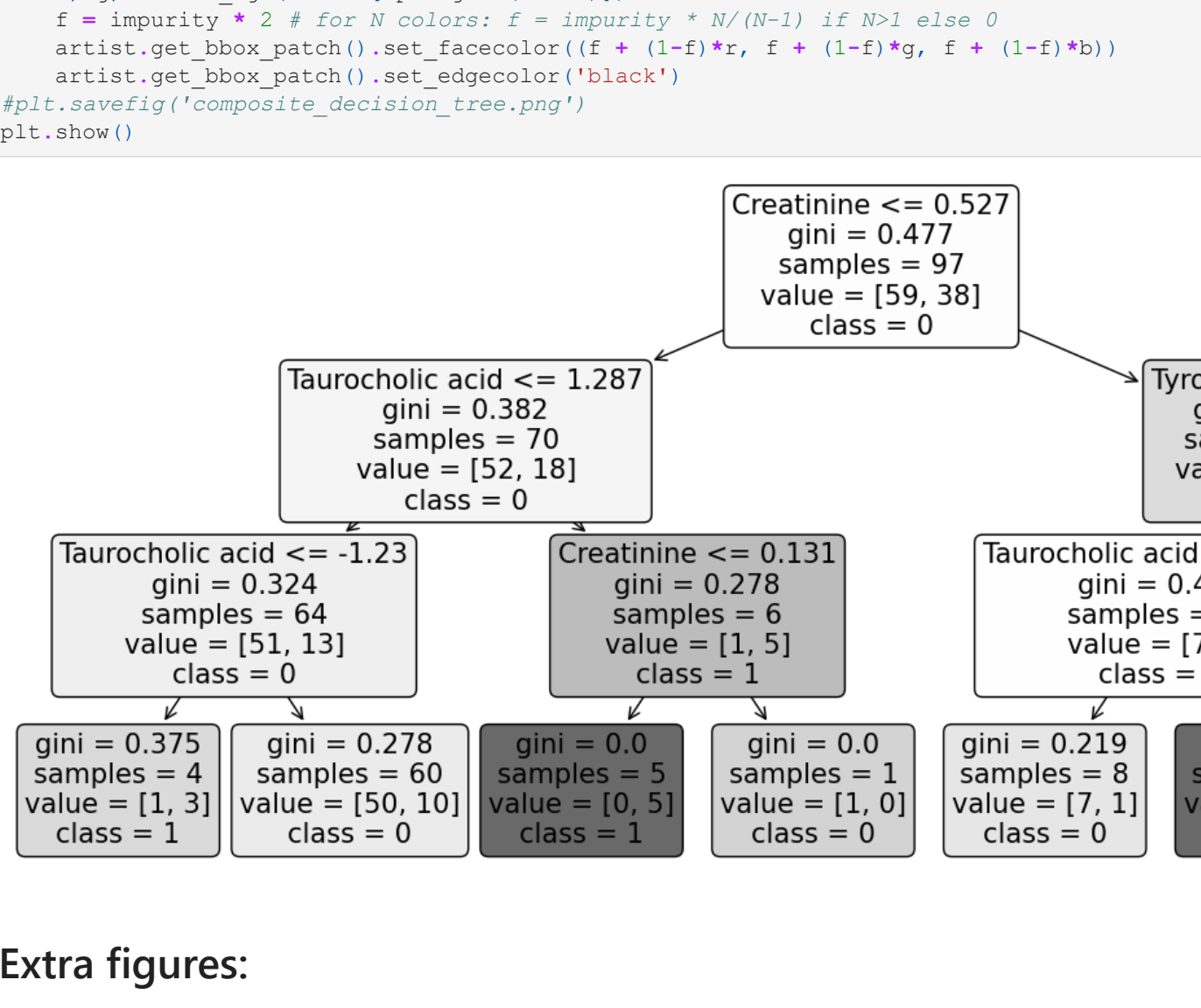
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label="Chance", alpha=0.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color='b',
    label="Mean ROC (AUC = %0.2f ± %0.2f) % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label="%0.2f ± 1 std. dev.",
)

ax.set(
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="ROC for 5-fold CV of all features",
)
ax.legend(loc="lower right")
plt.show()
```

ROC for 5-fold CV of all features



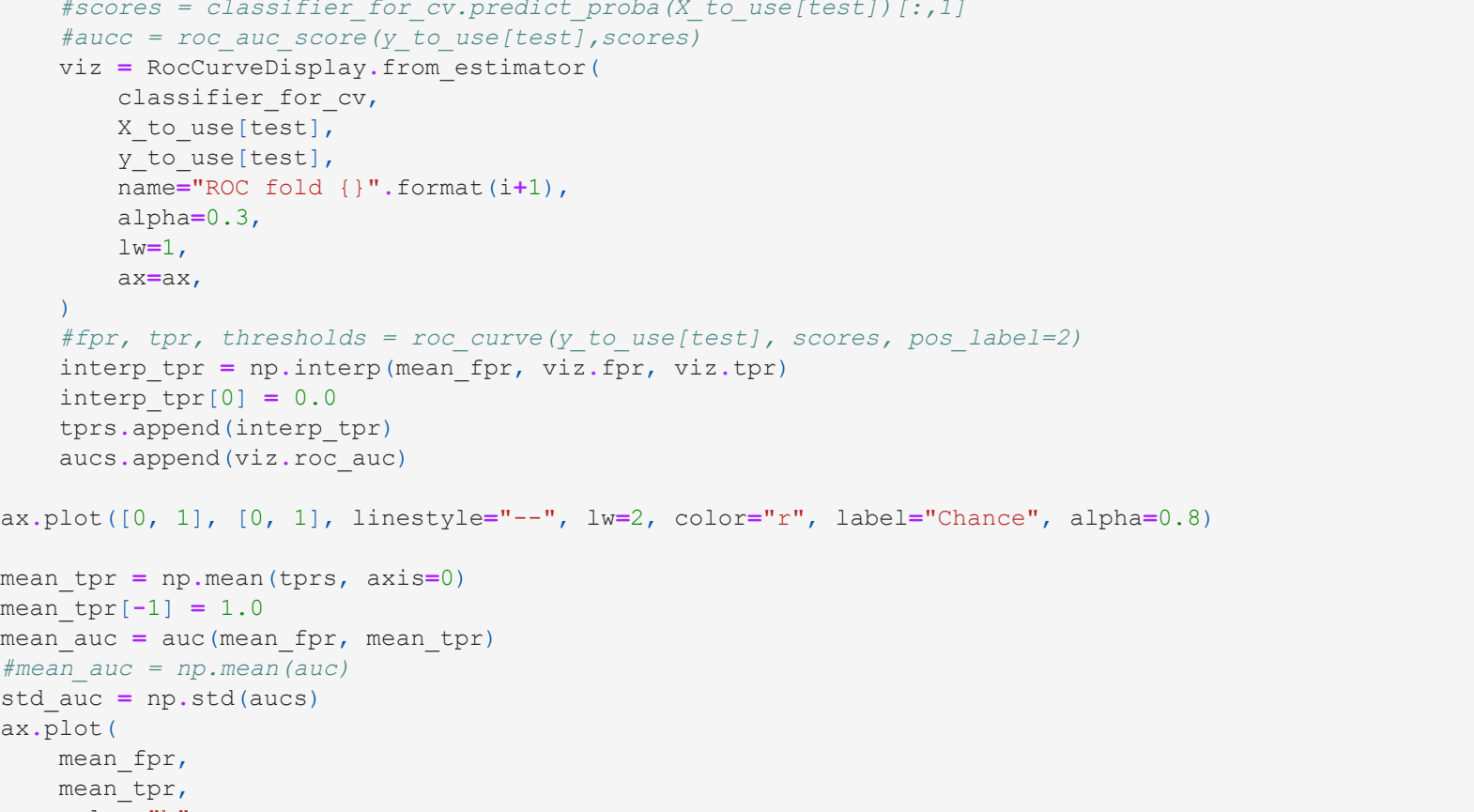
Generate a decision tree from our data:

```
In [16]: X_to_use = np.array(findat.iloc[:,my_top_feature_id])
      y_to_use = y.values

clf = tree.DecisionTreeClassifier(max_depth=3, max_leaf_nodes=None)
clf = clf.fit(X_to_use, y_to_use)
y_pred = clf.predict(X_to_use)
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y, y_pred)))

Root Mean Squared Error: 0.3517262290563295

In [17]: plt.figure(figsize=(15,7))
artists = tree.plot_tree(clf, feature_names = ['Taurocholic acid', 'Creatinine', 'Tyrosine'],
                        filled=True, fontsize=15, rounded = True, class_names=['0', '1'])
for artist, impurity, value in zip(artists, clf.tree_.impurity, clf.tree_.value):
    # let the max value decide the color; whiten the color depending on impurity (gini)
    r, g, b = to_rgb(colors[top_argmax(value)])
    f = impurity * 2 # for N colors: f = impurity * N/(N-1) if N>1 else 0
    artist.get_box_patch().set_facecolor((f + (1-f)*r, f + (1-f)*g, f + (1-f)*b))
    artist.get_box_patch().set_edgecolor('black')
plt.savefig('composite_decision_tree.png')
plt.show()
```



Extra figures:

ROC curve for random forest model based on clinical variables only:

```
In [18]: X_to_use = np.array(clin_orig[['SAPS2','SOFA','NT-pro-BNP','Urea','Creatinine','Troponin','Platelets']])
      y_to_use = y.values

classifier = RandomForestClassifier()
# Try out a grid of parameters:
parameters = {'max_features':np.arange(2,4), 'n_estimators':[1000,2000], 'max_depth':[2,3,4], 'random_state':[0]}
classifier_opt = GridSearchCV(classifier, parameters, cv = 5)

# Fit on entire dataset to get optimal parameters:
classifier_opt.fit(X_to_use,y_to_use)
params_to_use = classifier_opt.best_estimator_.get_params()

cv = StratifiedFold(n_splits=5)
classifier_for_cv = RandomForestClassifier(**params_to_use)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)
importance_list = []

fig, ax = plt.subplots(figsize=(8,6))
for i, (train, test) in enumerate(cv.split(X_to_use, y_to_use)):
    classifier_for_cv.fit(X_to_use[train], y_to_use[train])
    importance_list.append(classifier_for_cv.feature_importances_)
    fscores = classifier_for_cv.predict_proba(X_to_use[test])[:,1]
    fscore = roc_auc_score(y_to_use[test], fscores)
    viz = RocCurveDisplay.from_estimator(
        classifier_for_cv,
        X_to_use[test],
        y_to_use[test],
        name="ROC fold {}".format(i+1),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    fpr, tpr, thresholds = roc_curve(y_to_use[test], fscores, pos_label=2)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

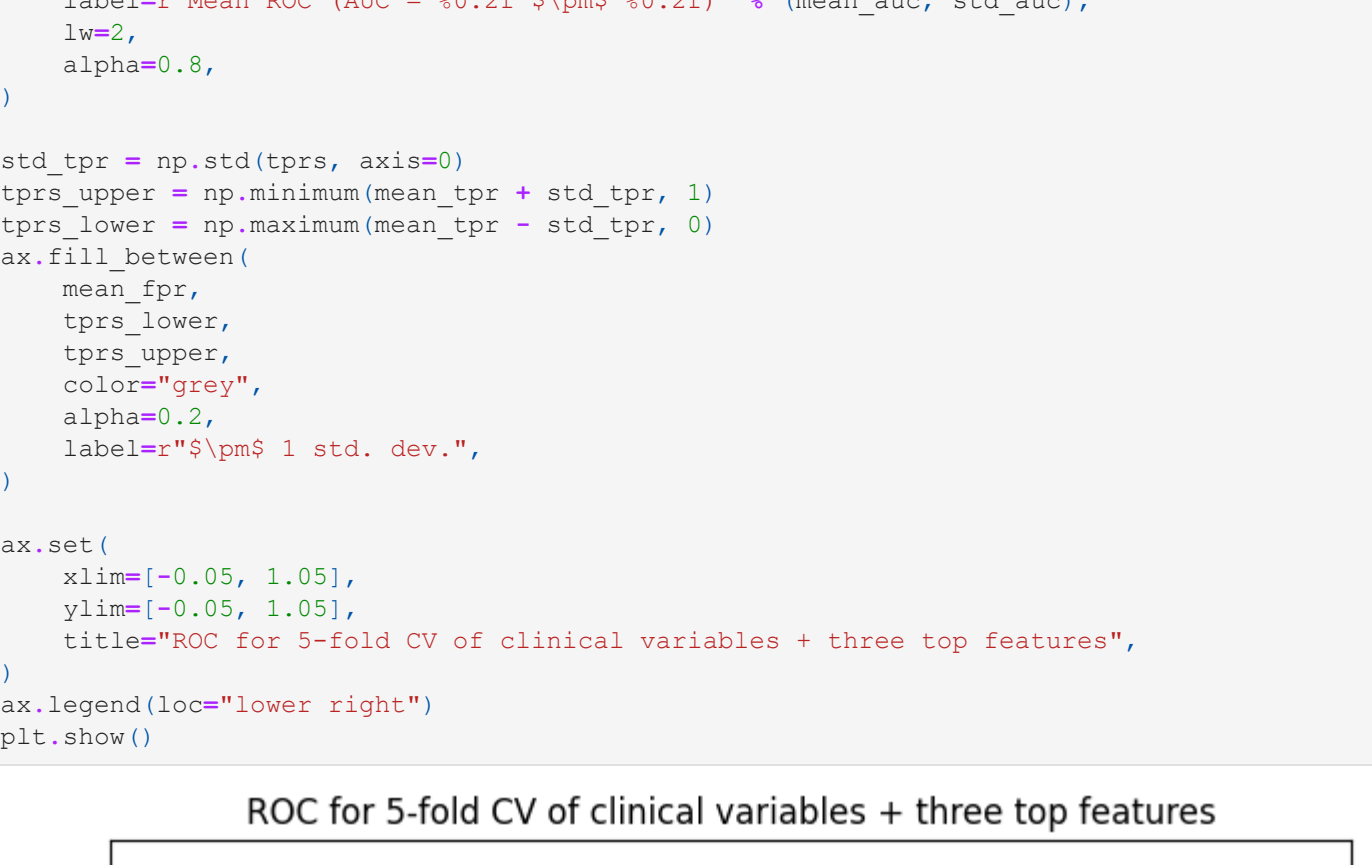
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label="Chance", alpha=0.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
mean_auc = np.mean(aucs)
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color='b',
    label="Mean ROC (AUC = %0.2f ± %0.2f) % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label="%0.2f ± 1 std. dev.",
)

ax.set(
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="ROC for 5-fold CV of clinical variables",
)
ax.legend(loc="lower right")
plt.show()
```

ROC for 5-fold CV of clinical variables



Use clinical variables with top 3 features:

```
In [31]: X_to_use = np.array(pd.concat([findat[['pos_mz_516.3008798','pos_mz_114.066238','pos_mz_182.0808902']],
      clin_orig[['SAPS2','SOFA','NT-pro-BNP','Urea','Creatinine','Troponin','Platelets']]]), axis=1)
      y_to_use = y.values

classifier = RandomForestClassifier()
# Try out a grid of parameters:
parameters = {'max_features':np.arange(2,4), 'n_estimators':[1000,2000], 'max_depth':[2,3,4], 'random_state':[0]}
classifier_opt = GridSearchCV(classifier, parameters, cv = 5)

# Fit on entire dataset to get optimal parameters:
classifier_opt.fit(X_to_use,y_to_use)
params_to_use = classifier_opt.best_estimator_.get_params()

cv = StratifiedFold(n_splits=5)
classifier_for_cv = RandomForestClassifier(**params_to_use)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)
importance_list = []

fig, ax = plt.subplots(figsize=(8,6))
for i, (train, test) in enumerate(cv.split(X_to_use, y_to_use)):
    classifier_for_cv.fit(X_to_use[train], y_to_use[train])
    importance_list.append(classifier_for_cv.feature_importances_)
    fscores = classifier_for_cv.predict_proba(X_to_use[test])[:,1]
    fscore = roc_auc_score(y_to_use[test], fscores)
    viz = RocCurveDisplay.from_estimator(
        classifier_for_cv,
        X_to_use[test],
        y_to_use[test],
        name="ROC fold {}".format(i+1),
        alpha=0.3,
        lw=1,
        ax=ax,
    )
    fpr, tpr, thresholds = roc_curve(y_to_use[test], fscores, pos_label=2)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

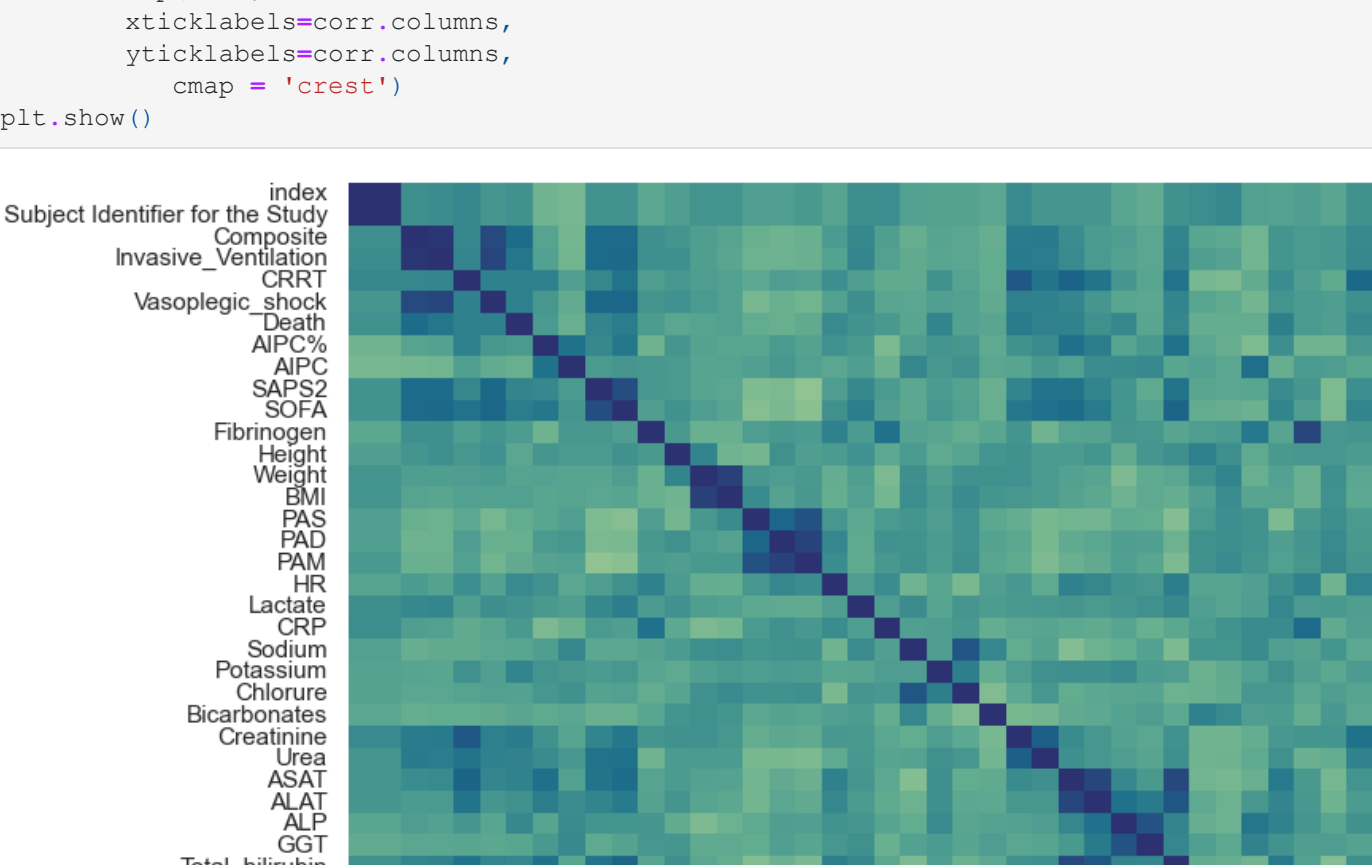
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label="Chance", alpha=0.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
mean_auc = np.mean(aucs)
std_auc = np.std(aucs)
ax.plot(
    mean_fpr,
    mean_tpr,
    color='b',
    label="Mean ROC (AUC = %0.2f ± %0.2f) % (mean_auc, std_auc),
    lw=2,
    alpha=0.8,
)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(
    mean_fpr,
    tprs_lower,
    tprs_upper,
    color="grey",
    alpha=0.2,
    label="%0.2f ± 1 std. dev.",
)

ax.set(
    xlim=[-0.05, 1.05],
    ylim=[-0.05, 1.05],
    title="ROC for 5-fold CV of clinical variables + three top features",
)
ax.legend(loc="lower right")
plt.show()
```

ROC for 5-fold CV of clinical variables + three top features

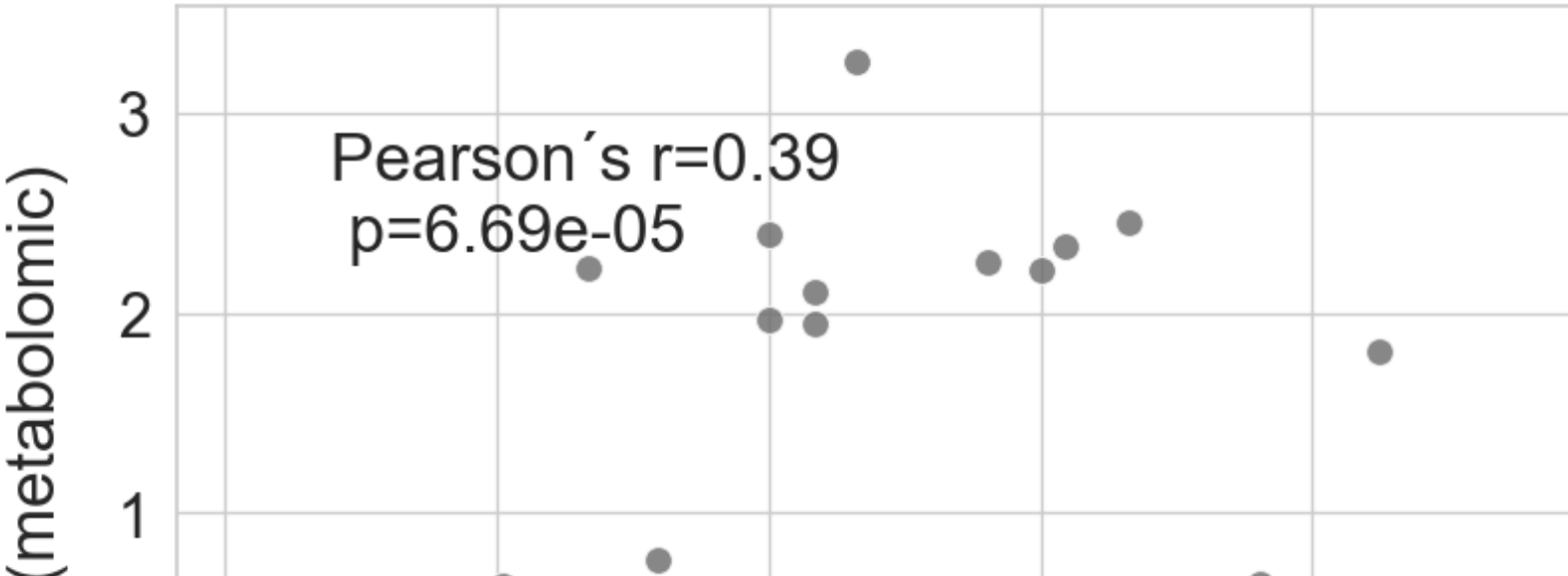


Comparison of creatinine levels and metabolomic creatinine levels (mz 114.0662):

```
In [20]: sns.set(rc={"figure.figsize":(10, 8)})
sns.set(font_scale=2)
sns.set_style("whitegrid")
colors = "darkgray"
sns.set_palette(sns.color_palette(colors))

glog = lambda x: np.log2((x*np.sqrt(x**2+1))/2) # Same transformation as was made to the metabolomic data
new_dat = pd.DataFrame([findat[['pos_mz_114.066238']],glog(clin_orig['Creatinine'])]).T
new_dat.columns = ['Creatinine (metabolomic)', 'Creatinine (clinical)']

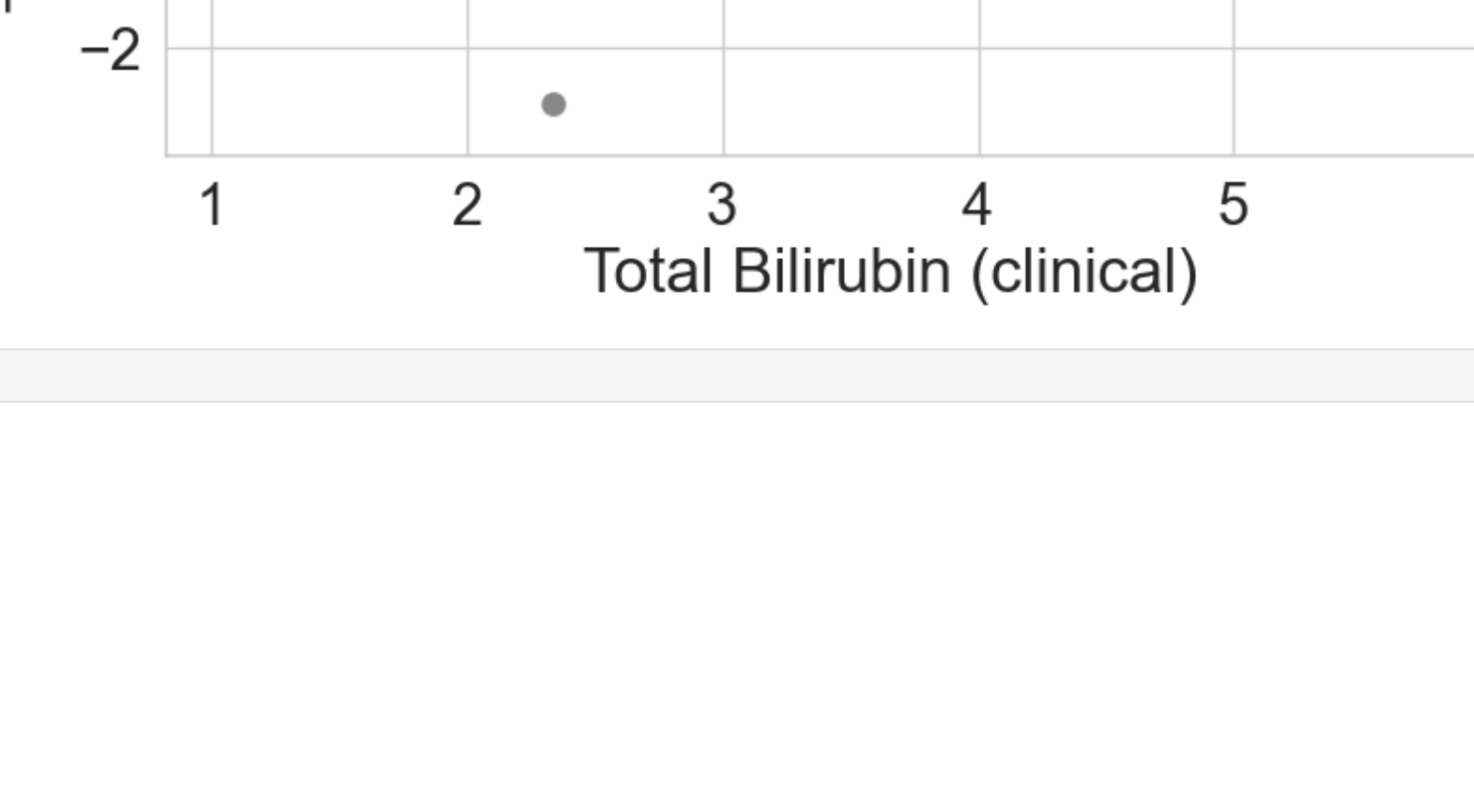
ax = sns.scatterplot(data=new_dat, x = 'Creatinine (clinical)', y = 'Creatinine (metabolomic)', alpha=0.8, s = 10)
r, p = pearsonr(new_dat['Creatinine (clinical)'], new_dat['Creatinine (metabolomic)'])
ax.text(1.1, .8, 'Pearson's r=(%.2f)\n p=(%.2e)' % (r,p), transform=ax.transAxes)
plt.show()
```



Check correlation of putative taurocholic acid feature with clinical variables:

```
In [24]: sns.set(font_scale=1)
test_dat = pd.concat([clin_orig, findat[['pos_mz_516.3008798']], axis = 1)
corr_test_dat = corr(method = 'pearson')

# plot the heatmap
plt.figure(figsize = (12,8))
xticklabels=corr.columns,
yticklabels=corr.columns,
cmap = 'cmap'
plt.show()
```



```
In [28]: sns.set(rc={"figure.figsize":(10, 8)})
sns.set(font_scale=2)
sns.set_style("whitegrid")
colors = "darkgray"
sns.set_palette(sns.color_palette(colors))
pearsonr(glog(clin_orig['Total Bilirubin']), findat[['pos_mz_516.3008798']])

new_dat = pd.DataFrame([findat[['pos_mz_516.3008798']],glog(clin_orig['Total Bilirubin'])]).T
new_dat.columns = ['Taurocholic acid (metabolomic)', 'Total Bilirubin (clinical)']

ax = sns.scatterplot(data=new_dat, x = 'Total Bilirubin (clinical)', y = 'Taurocholic acid (metabolomic)', alpha=0.8, s = 10)
r, p = pearsonr(new_dat['Total Bilirubin (clinical)'], new_dat['Taurocholic acid (metabolomic)'])
ax.text(1.1, .8, 'Pearson's r=(%.2f)\n p=(%.2e)' % (r,p), transform=ax.transAxes)
plt.show()
```



In []: