Machine learning Group 2 Final Project

Problem (a)

Classifying for depression and gender. Use a machine learning algorithm of your choice to conduct two types of classification tasks based on the original set of acoustic features.

Problem (a.i)

Depression classification. In the first task, you will classify between speakers who have depression and speakers who do not have depression using the original set of 88 acoustic features. Please report the simple classification accuracy A and balanced classification accuracy BA on the test set for all participants included in the test set, as well as for the female and male participants in the test set separately. In addition, please report the equality of opportunity (EO) that computes the difference in true positive rate TPR (i.e., ratio of correctly classified participants with depression) between female and male participants, i.e., EO = 1 – |TPR(male) – TPR(female)|, quantifying to what extent the same proportion of female and male participants receive a true positive outcome. Please discuss your findings. Note: Each turn can be used as a sample for the train and test data, thus, you will be obtaining a decision on whether an turn comes from a participant with depression or without depression. However, the accuracy and EO measures should be computed at the participant level. This means that you will need to aggregate the turn-based decisions on depression classification at the participant level (e.g., via averaging those decisions).

File management

Description of the files and folders

The folders 'features_test' and 'features_train' contain .csv files. Each .csv file corresponds to one participant - the name of the file is that participant's ID.

Each participant is either male or female, and either has depression or does not have depression. These details are available in the 'labels.csv' file.

Setting up files to be used

For problem (a.i), each row of each partipant's file is treated as a sample. So we would like to add two columns to each participant's file - one for their gender and one for whether they have depression or not. In each file, these two columns will have the same value for all the rows since they all correspond to the same participant. we would like to add yet another column which contains the participant ID.

We would then like to combine all the files of the training participants and make one single .csv file (similarly for the testing participants). These are then used for training and testing the ML algorithm.

In [ ]:
```
## Loading libraries

import pandas as pd
import numpy as np
import os

# Loading the labels
df_labels = pd.read_csv('labels.csv')

# Assign directory to be the features_train folder
directory = 'features_train'

# Creating an empty master array to store the combined data
master_train = np.empty((0, 91))


## The files are of the form 'spk_XYZ.csv'
## where XYZ is the participantID
## Using this fact, we can get the participantID
## as index 4, 5, 6 of the file name


# Iterating over the files in the directory
for filename in os.listdir(directory):
    if filename != '.DS_Store':
        fullname = 'features_train/' + filename
        df1 = pd.read_csv(fullname)         # Storing the data in the csv fi
        np1 = df1.to_numpy()                                    # Converting

        id = float(filename[4:7])                               # Gives the
        dep = df_labels[df_labels['Participant_ID'] == id]['Depression']
        gender = df_labels[df_labels['Participant_ID'] == id]['Gender']

        np1 = np.insert(np1, 88, dep, axis = 1)         # Storing the depress
        np1 = np.insert(np1, 89, gender, axis = 1)      # Storing the gender
        np1 = np.insert(np1, 0, id, axis = 1)           # Storing the id in c

        master_train = np.append(master_train, np1, axis = 0)


train_df = pd.DataFrame(master_train)

# Storing the dataframe in a csv file
train_df.to_csv('fulltrain.csv', index = False)

# Doing the same process on the test data
```

```
directory = 'features_test'
master_test = np.empty((0, 91))

for filename in os.listdir(directory):
    if filename != '.DS_Store':
        fullname = 'features_test/' + filename
        df1 = pd.read_csv(fullname)        # Storing the data in the csv fi
        np1 = df1.to_numpy()                                    # Converting

        id = float(filename[4:7])                               # Gives the
        dep = df_labels[df_labels['Participant_ID'] == id]['Depression']
        gender = df_labels[df_labels['Participant_ID'] == id]['Gender']

        np1 = np.insert(np1, 88, dep, axis = 1)        # Storing the depress
        np1 = np.insert(np1, 89, gender, axis = 1)     # Storing the gender
        np1 = np.insert(np1, 0, id, axis = 1)          # Storing the id in c

        master_test = np.append(master_test, np1, axis = 0)


test_df = pd.DataFrame(master_test)
test_df.to_csv('fulltest.csv', index = False)
```

Data handling: The training data from all the participants (87 in total) was combined using Pandas. Each participant's data consists of multiple turns, which were treated as individual examples. Each participant's gender and depression status was appended as the label for all turns. Their participant_id was also appended to be able to aggregate the outputs later. The resulting data consisted of 13625 rows. The same was done with testing data (20 participants, 3280 rows, 19.4% split) and stored as a separate structure. There was only one row with NAN values which was removed. For the logistic regression and SVM models, we so do standard scaling of the features.

Model: The model pipeline consists of training on gender classification for each turn of all participants followed by evaluating on test data. A few different models were considered and the following logistic regression model was finalised.

In [ ]:

```
## Importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import recall_score
from sklearn.preprocessing import StandardScaler
import seaborn as sns
```

In [ ]:

```
## Reading the data and setting up X and y
## For this problem, Depression is the outcome
```

```
df_train = pd.read_csv('fulltrain.csv')
df_train.dropna(inplace = True)
X_train = df_train.iloc[:, 1:-2]
y_train = df_train.iloc[:, -2]

df_test = pd.read_csv('fulltest.csv')
df_test.dropna(inplace = True)
X_test = df_test.iloc[:, 1:-2]
y_test = df_test.iloc[:, -2]
```

Logistic regression classifier

In [ ]:
```
X_train_norm = StandardScaler().fit_transform(X_train)
X_test_norm = StandardScaler().fit_transform(X_test)


logr = LogisticRegression(max_iter = 1000)

model = logr.fit(X_train_norm, y_train)

y_pred = model.predict(X_test_norm)
# y_pred gives a prediction on each row of the partcipant data
```

Accuracies and EO need to be calculated at the participant level, so the decisions need to be aggregated for each participant.

Aggregating predictions

Since accuracies need to be calculated participant wise, I need to group the data according to participant ID:

In [ ]:
```
## Combining the predictions, true y-values, participant ID and gender into

y_train_total = np.empty((np.shape(y_train)[0], 4))

y_train_total[:, 0] = df_train.iloc[:, 0]  # ID
y_train_total[:, 1] = df_train.iloc[:, -2]  # Depression true value
y_train_total[:, 2] = df_train.iloc[:, -1]  # Gender true value
y_train_total[:, 3] = model.predict(X_train_norm)   # For logistic regressic

y_train_total_df = pd.DataFrame(y_train_total)

y_test_total = np.empty((np.shape(y_test)[0], 4))
y_test_total[:, 0] = df_test.iloc[:, 0]
y_test_total[:, 1] = df_test.iloc[:, -2]
y_test_total[:, 2] = df_test.iloc[:, -1]
y_test_total[:, 3] = y_pred
y_test_total_df = pd.DataFrame(y_test_total)

train_groups = y_train_total_df.groupby(y_train_total_df[0])   # Grouping by
test_groups = y_test_total_df.groupby(y_test_total_df[0])
```

```
train_true = train_groups[1].aggregate(pd.Series.mode)
train_prediction = train_groups[3].aggregate(pd.Series.mode)

test_true = test_groups[1].aggregate(pd.Series.mode)
test_prediction = test_groups[3].aggregate(pd.Series.mode)

print("Simple accuracy for all participants in training set:", accuracy_scor
print("Balanced accuracy for all partipants in training set:", balanced_accu


print("Simple accuracy for all participants in test set:", accuracy_score(te
print("Balanced accuracy for all partipants in test set:", balanced_accuracy


### Splitting the data into female and male

train_males_df = pd.DataFrame(y_train_total[np.argwhere(y_train_total[:, 2]
train_females_df = pd.DataFrame(y_train_total[np.argwhere(y_train_total[:, 2

test_males_df = pd.DataFrame(y_test_total[np.argwhere(y_test_total[:, 2] ==
test_females_df = pd.DataFrame(y_test_total[np.argwhere(y_test_total[:, 2] =

train_males_groups = train_males_df.groupby(train_males_df[0])     # Group b
train_females_groups = train_females_df.groupby(train_females_df[0])     # G

test_males_groups = test_males_df.groupby(test_males_df[0])
test_females_groups = test_females_df.groupby(test_females_df[0])

train_true_males = train_males_groups[1].aggregate(pd.Series.mode)
train_prediction_males = train_males_groups[3].aggregate(pd.Series.mode)

train_true_females = train_females_groups[1].aggregate(pd.Series.mode)
train_prediction_females = train_females_groups[3].aggregate(pd.Series.mode)

test_true_males = train_males_groups[1].aggregate(pd.Series.mode)
test_prediction_males = train_males_groups[3].aggregate(pd.Series.mode)

test_true_females = test_females_groups[1].aggregate(pd.Series.mode)
test_prediction_females = test_females_groups[3].aggregate(pd.Series.mode)

print("Simple accuracy for males in training set:", accuracy_score(train_tru
print("Balanced accuracy for males in training set:", balanced_accuracy_scor


print("Simple accuracy for females in training set:", accuracy_score(train_t
print("Balanced accuracy for females in training set:", balanced_accuracy_sc

print("Simple accuracy for males in test set:", accuracy_score(test_true_mal
print("Balanced accuracy for males in test set:", balanced_accuracy_score(te


print("Simple accuracy for females in test set:", accuracy_score(test_true_f
print("Balanced accuracy for females in test set:", balanced_accuracy_score(
```

```
tpr_males = recall_score(test_true_males, test_prediction_males)
tpr_females = recall_score(test_true_females, test_prediction_females)
eo = 1 - np.abs(tpr_males - tpr_females)

print("Equality of Opportunity (EO) on test set:", eo)
```

```
Simple accuracy for all participants in training set: 0.7701149425287356
Balanced accuracy for all partipants in training set: 0.5833333333333334
Simple accuracy for all participants in test set: 0.7
Balanced accuracy for all partipants in test set: 0.5
Simple accuracy for males in training set: 0.803921568627451
Balanced accuracy for males in training set: 0.5833333333333334
Simple accuracy for females in training set: 0.7222222222222222
Balanced accuracy for females in training set: 0.5833333333333334
Simple accuracy for males in test set: 0.803921568627451
Balanced accuracy for males in test set: 0.5833333333333334
Simple accuracy for females in test set: 0.375
Balanced accuracy for females in test set: 0.5
Equality of Opportunity (EO) on test set: 0.8333333333333334
```

These are final results for problem a (i) as specified in the question. We see that the simple classification accuracy is around 70 % for the test set for all participants. For males, the accuracy is higher at 80%, because of the higher amount of data for males, and higher number of males with depression in the train set. The EO value obtained was around 83.33%.

Problem (a.ii) Gender Classification

Data handling: The training data from all the participants (87 in total) was combined using Pandas. Each participant's data consists of multiple turns, which were treated as individual examples. Each participant's gender was appended as the label for all turns. Their participant_id was also appended to be able to aggregate the outputs later. The resulting data consisted of 13625 rows. The same was done with testing data (20 participants, 3280 rows, 19.4% split) and stored as a separate structure. There was only one row with NAN values which was removed. For the logistic regression and SVM models, we so do standard scaling of the features.

Model: The model pipeline consists of training on gender classification for each turn of all participants followed by evaluating on test data. There is also a cross validation score reported for 5 folds of the data for each model. This is to see which is generally a better model. For the test data, we predict gender on all turns as during training, but follow it by choosing the mode class (majority voting) as the final output for each participant. This is used to get simple and balanced accuracy for the test participants.

In this notebook, we train models to detect gender from the given dataset. The final model we choose is Random Forest which gives perfect scores in all cases. This is after doing cross validation and measuring average simple accuracy for each model. We also get the idea of the importance of features for gender using this Random Forest Model.

Additionally, we use the top 5, 10, 15, 20, 25 and 30 features using part c) and check the updated performance of this model. The model performance remains the same (1.0 simple and balanced accuracy)

Load training data and associated labels

In [ ]:
```
import os
import pandas as pd
import numpy as np
import warnings

warnings.filterwarnings("ignore")
```

In [ ]:
```
def load_data(dirname):
    df = pd.DataFrame()
    count = 0
    labels = pd.read_csv("labels.csv")

    # for each training file
    for filename in os.listdir(dirname):

        # get corresponding gender for the sample as the true label
```

```
        sample_id = int(filename.split(".")[0].split("_")[1])
        sample_label = list(labels.loc[labels['Participant_ID']== sample_id,

        # load features for that sample
        currentfile = pd.read_csv(os.path.join(dirname,filename),header=None
        currentfile["gender"] = sample_label*len(currentfile)
        currentfile["participant_ID"] = [sample_id]*len(currentfile)
        # print(filename,len(currentfile))

        # concatenate to the whole data
        df=pd.concat([df, currentfile], axis = 0)
        # print(len(df.columns))
    return df
```

In [ ]:

```
traindata = load_data("features_train").reset_index()
traindata.dropna(inplace=True)

testdata = load_data("features_test").reset_index()
testdata.dropna(inplace=True)
display(testdata)
```

	index	0	1	2	3	4	5	
0	0	32.160255	0.200581	23.145561	35.632530	36.815937	13.670376	-65.04
1	1	28.780031	0.074786	27.129395	28.150295	31.058764	3.929369	52.7
2	2	29.038708	0.144522	25.411283	25.819115	34.090847	8.679564	65.1
3	3	24.198637	0.077389	22.477812	24.032180	25.971500	3.493689	106.8
4	4	23.637993	0.130217	18.551594	25.037369	26.020950	7.469356	40.8
...	...	...	...	...	...	...	...	
3275	146	19.820496	0.022153	19.758846	19.922794	20.141985	0.383139	14.8
3276	147	22.432129	0.060207	21.505978	22.008904	23.922071	2.416094	49.2
3277	148	20.474200	0.027035	20.055449	20.319780	20.805794	0.750345	13.84
3278	149	21.627142	0.027161	21.076570	21.605469	22.160873	1.084303	14.4
3279	150	22.420483	0.054707	21.510702	22.143350	22.968153	1.457451	17.34

3280 rows × 91 columns

In [ ]:

```
output = 'gender'
columns = list(traindata.columns)
columns.remove(output)
columns.remove('index')
columns.remove('participant_ID')

X_train = traindata[columns]
y_train = traindata[output]

print(set(y_train.values))
```

```
{0, 1}
```

Define Model pipeline

```
In [ ]: from sklearn.metrics import accuracy_score, balanced_accuracy_score
        from sklearn.model_selection import cross_val_score

        def evaluate_model(model,test):
            y_preds = model.predict(test)
            testdf = testdata[["participant_ID",output]]
            testdf["preds"] = y_preds
            testdf = testdf.groupby(testdf["participant_ID"]).agg(pd.Series.mode)

            simple_acc = accuracy_score(np.array(testdf['preds']).reshape(-1,1),np.a
            balanced_acc = balanced_accuracy_score(np.array(testdf['preds']).reshape
            return simple_acc, balanced_acc

        def model_pipeline(Classifier, train, test=testdata[columns], **params):

            model = Classifier(**params)
            scores = cross_val_score(model, train, y_train, cv=5)
            print("CV accuracies: ",scores)
            print("Average CV accuracy", np.mean(scores))
            model.fit(train,y_train)
            print("Train accuracy: ",model.score(train,y_train))

            test_acc = evaluate_model(model,test)
            print("Test accuracy: ", test_acc[0])
            print("Test balanced accuracy: ", test_acc[1])
            return model
```

Trying Scikit learn models out of the box

```
In [ ]: from sklearn.linear_model import LogisticRegression
        from sklearn.linear_model import SGDClassifier

        print("Logistic Regression")
        model_pipeline(LogisticRegression, X_train)

        print()
        print("SGD")
        model_pipeline(SGDClassifier, X_train, max_iter=1000, loss="log_loss")
```

```
Logistic Regression
CV accuracies:  [0.9266055  0.92440367 0.92623853 0.78055046 0.88880734]
Average CV accuracy 0.8893211009174312
Train accuracy:  0.8907155963302752
Test accuracy:  [0.9]
Test balanced accuracy:  [0.89583333]

SGD
CV accuracies:  [0.87009174 0.92256881 0.72844037 0.76733945 0.85284404]
Average CV accuracy 0.828256880733945
Train accuracy:  0.8455045871559633
Test accuracy:  [0.85]
Test balanced accuracy:  [0.9]
```

Out[ ]:

```
▾ SGDClassifier
SGDClassifier(loss='log_loss')
```

In [ ]:

```
from sklearn.svm import SVC

print("Support Vector Machine")
model_pipeline(SVC, X_train)
```

```
Support Vector Machine
CV accuracies:  [0.75412844 0.82972477 0.85247706 0.67376147 0.77981651]
Average CV accuracy 0.7779816513761468
Train accuracy:  0.8
Test accuracy:  [0.75]
Test balanced accuracy:  [0.85294118]
```

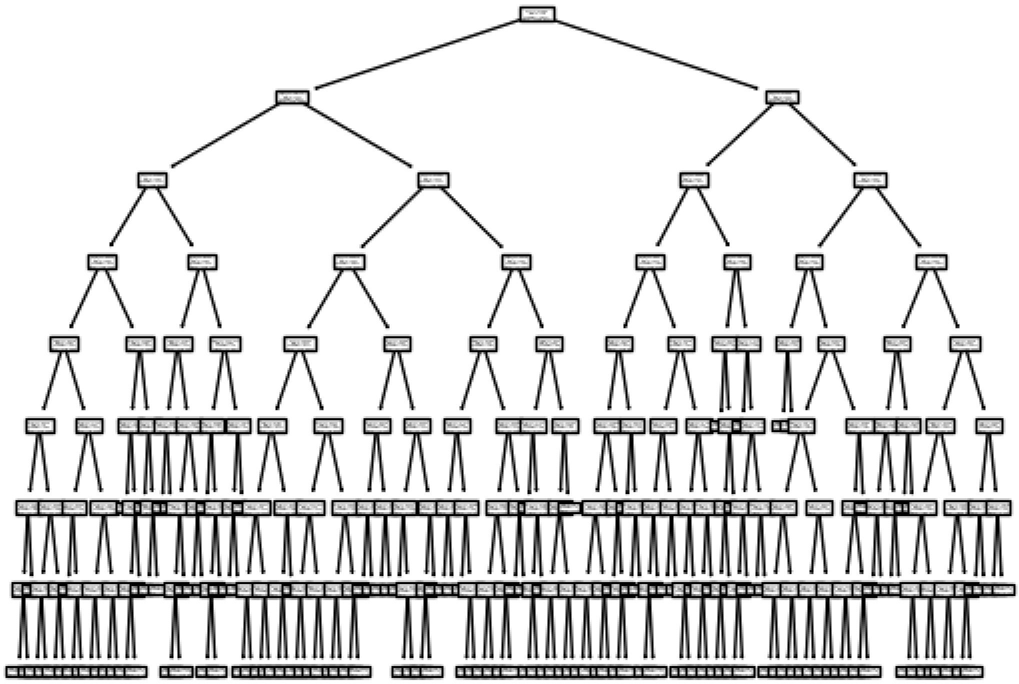
Out[ ]:

```
▾ SVC
SVC()
```

In [ ]:

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier as DTC

print("Decision Trees")
model = model_pipeline(DTC, X_train, max_depth=8)

tree.plot_tree(model)
print(model)
```

```
Decision Trees
CV accuracies:  [0.92623853 0.91045872 0.95779817 0.83669725 0.89981651]
Average CV accuracy 0.9062018348623854
Train accuracy:  0.9699816513761468
Test accuracy:  [1.]
Test balanced accuracy:  [1.]
DecisionTreeClassifier(max_depth=8)
```



In [ ]:
```
from sklearn.ensemble import RandomForestClassifier as RFC

print("Random Forest")
rf_model = model_pipeline(RFC, X_train)
```

```
Random Forest
CV accuracies:  [0.95853211 0.93577982 0.97614679 0.8546789  0.9240367 ]
Average CV accuracy 0.9298348623853212
Train accuracy:  1.0
Test accuracy:  [1.]
Test balanced accuracy:  [1.]
```

With standardized data

In [ ]:
```
from sklearn.preprocessing import StandardScaler as SSC
scaler = SSC()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.fit_transform(testdata[columns])
```

In [ ]:
```
print("Logistic Regression")
model_pipeline(LogisticRegression, X_train_std, test = X_test_std )

print()
print("SGD")
model_pipeline(SGDClassifier, X_train_std, test = X_test_std, max_iter=1000,

print()
print("Support Vector Machine")
model_pipeline(SVC, X_train_std,test = X_test_std)
```

```
Logistic Regression
CV accuracies:  [0.94238532 0.92146789 0.97394495 0.90899083 0.92183486]
Average CV accuracy 0.9337247706422019
Train accuracy:  0.9505321100917431
Test accuracy:  [0.9]
Test balanced accuracy:  [0.89583333]

SGD
CV accuracies:  [0.92036697 0.92146789 0.96550459 0.8987156  0.90091743]
Average CV accuracy 0.921394495412844
Train accuracy:  0.9442935779816514
Test accuracy:  [1.]
Test balanced accuracy:  [1.]

Support Vector Machine
CV accuracies:  [0.94568807 0.93688073 0.97724771 0.91633028 0.91633028]
Average CV accuracy 0.9384954128440366
Train accuracy:  0.9745321100917431
Test accuracy:  [0.95]
Test balanced accuracy:  [0.96153846]
```

Out[ ]:

```
▾ SVC
SVC()
```

Model	Cross Validation average accuracy (5 fold)	Simple Accuracy		Test Balanced Accuracy
		Train	Test	
Logistic Regression(scaled)	0.93	0.89	0.9	0.89
SVM (scaled)	0.94	0.97	0.95	0.96
Decision Tree (depth 8)	0.91	0.97	1.0	1.0
Random Forest (100 estimators)	0.93	1.0	1.0	1.0

Exploring correlations for problems (b) and (c)

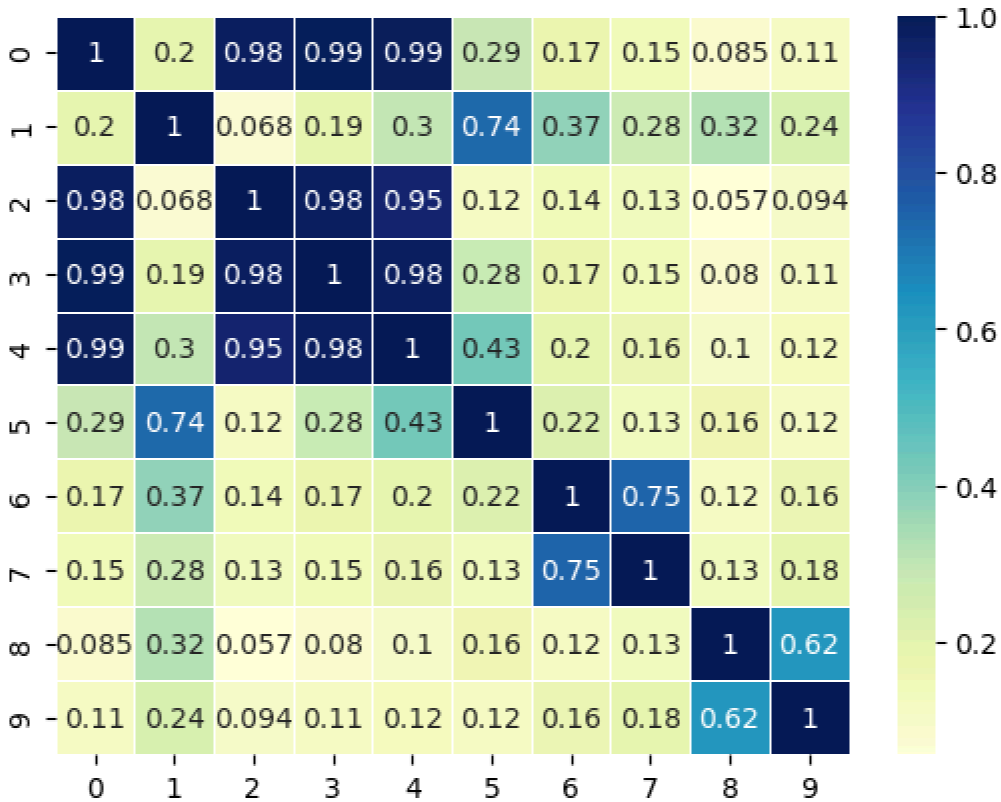
In [ ]:
```
## Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
## Reading the data and setting up X and y
## For this problem, Depression is the outcome

df_train = pd.read_csv('fulltrain.csv')
df_train.dropna(inplace = True)
X_train = df_train.iloc[:, 1:-2]
y_train = df_train.iloc[:, -2]

df_test = pd.read_csv('fulltest.csv')
df_test.dropna(inplace = True)
X_test = df_test.iloc[:, 1:-2]
y_test = df_test.iloc[:, -2]
```

In [ ]:
```
## Converting the feature data from the dataframe to numpy array

np_train = X_train.to_numpy()
```

In [ ]:
```
## Finding the correlation between the first 10 features
## which correspond to F0semitone

corr1 = np.empty((10, 10))
for i in np.arange(10):
    for j in np.arange(10):
        corr1[i, j] = np.corrcoef(np_train[:, i], np_train[:, j])[0, 1]

sns.heatmap(corr1, linewidths = 0.5, cmap="YlGnBu", annot = True)
```

Out[ ]:
```
<Axes: >
```



From the above correlation plot, we see that there is strong correlation in the following groups :

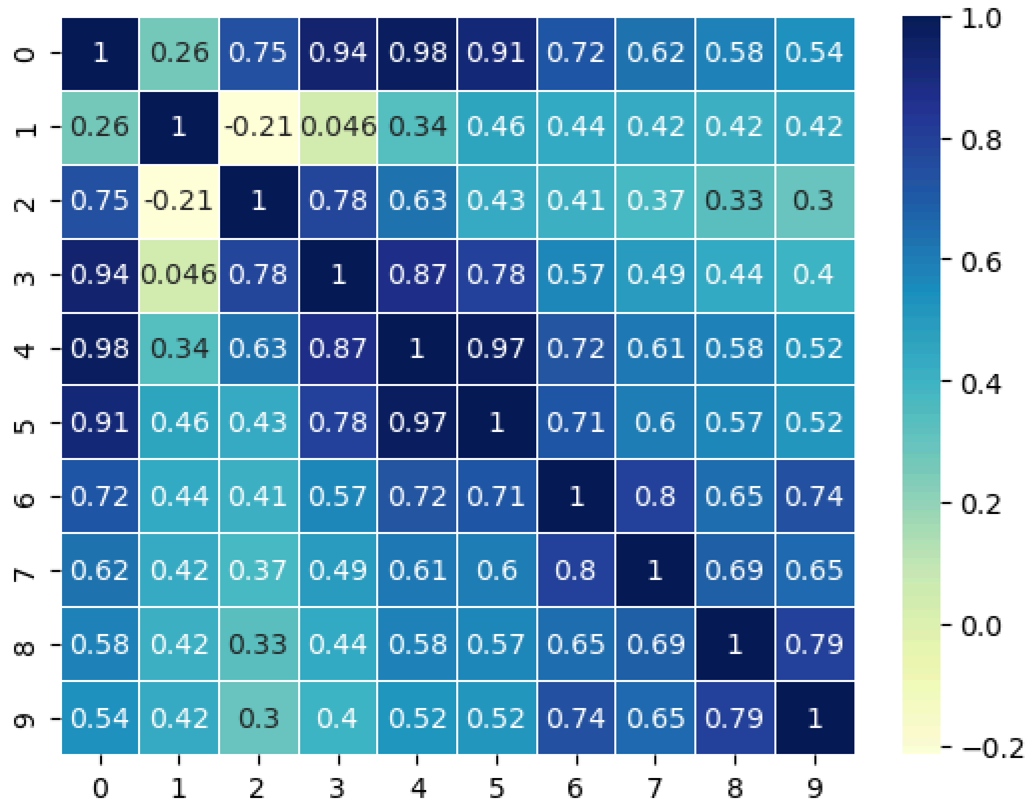
- 0, 2, 3, 4
- 1 and 5
- 6 and 7
- 8 and 9

Thus, we can select one feature from each of these feature groups.

In [ ]:
```
chosen = np.array([0, 1, 6, 7])
```

In [ ]:
```
## Finding the correlation between next 10 features
## which correspond to loudness

corr2 = np.empty((10, 10))
for i in np.arange(10, 20):
    for j in np.arange(10, 20):
        corr2[i - 10, j - 10] = np.corrcoef(np_train[:, i], np_train[:, j])[

sns.heatmap(corr2, linewidths = 0.5, cmap="YlGnBu", annot = True)
```

Out[ ]: <Axes: >



From the above correlation plot, we see that there is strong correlation in the following groups :

- 0, 2, 3, 4, 5
- 1
- 6 and 7
- 8 and 9

Thus, we can select one feature from each of these feature groups.

In [ ]:
```
chosen = np.append(chosen, [10, 11, 16, 18])
```

In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to spectralFlux

corr3 = np.corrcoef(np_train[:, 20], np_train[:, 21])
corr3[0, 1]
```

Out[ ]:
```
0.2918258250320712
```

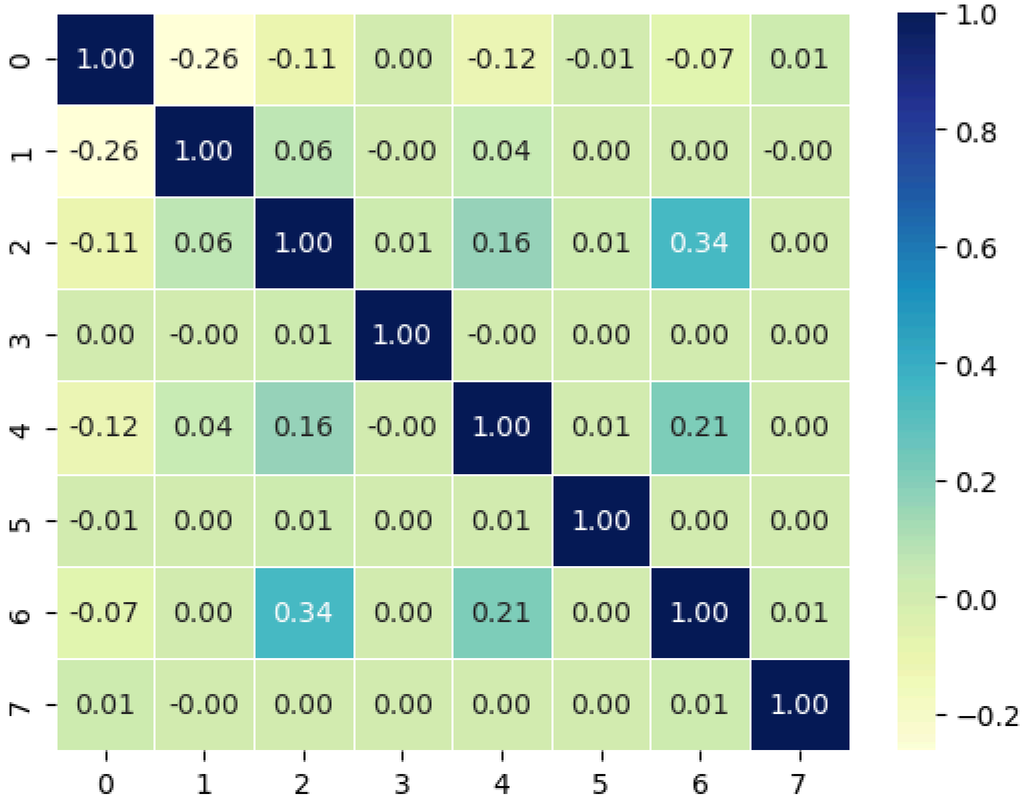
These two are not heavily correlated.

In [ ]:
```
chosen = np.append(chosen, [20, 21])
```

In [ ]:
```
## Finding the correlation between next 8 features
## which correspond to mfcc
```

```
corr4 = np.empty((8, 8))
for i in np.arange(22, 30):
    for j in np.arange(22, 30):
        corr4[i - 22, j - 22] = np.corrcoef(np_train[:, i], np_train[:, j])[

sns.heatmap(corr4, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.2
```

Out[ ]: `<Axes: >`



None of these show any correlation, so none of them can be dropped.

In [ ]:
```
chosen = np.append(chosen, np.arange(22, 30))
```

In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to jitter
corr41 = np.corrcoef(np_train[:, 30], np_train[:, 31])
corr41[0, 1]
```

Out[ ]: `0.1609483849831289`

No correlation between these two features.

In [ ]:
```
chosen = np.append(chosen, [30, 31])
```

In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to shimmer
corr5 = np.corrcoef(np_train[:, 32], np_train[:, 33])
corr5[0, 1]
```

Out[ ]: 0.0011673537442088168

In [ ]:
```
chosen = np.append(chosen, [32, 33])
```

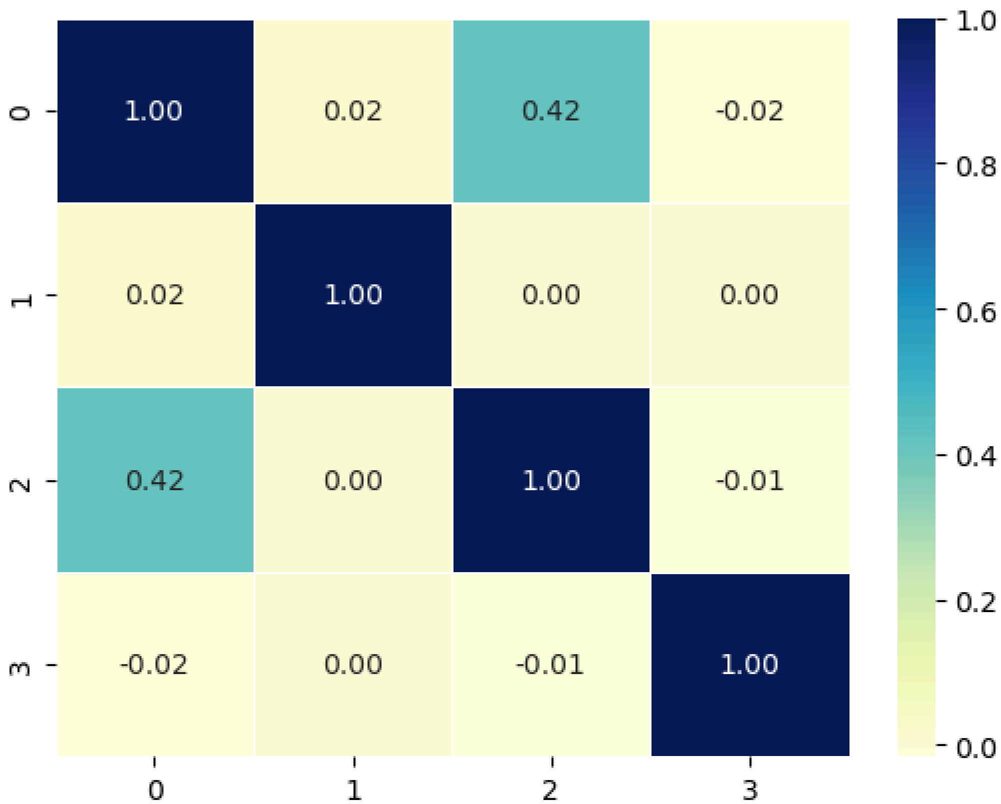
In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to HNRdBACF
corr6 = np.corrcoef(np_train[:, 34], np_train[:, 35])
corr6[0, 1]
```

Out[ ]: -0.07632523314132521

In [ ]:
```
chosen = np.append(chosen, [34, 35])
```

In [ ]:
```
## Finding the correlation between next 4 features
## which correspond to logRelF0
corr7 = np.empty((4, 4))
for i in np.arange(36, 40):
    for j in np.arange(36, 40):
        corr7[i - 36, j - 36] = np.corrcoef(np_train[:, i], np_train[:, j])[

sns.heatmap(corr7, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.2
```

Out[ ]: <Axes: >

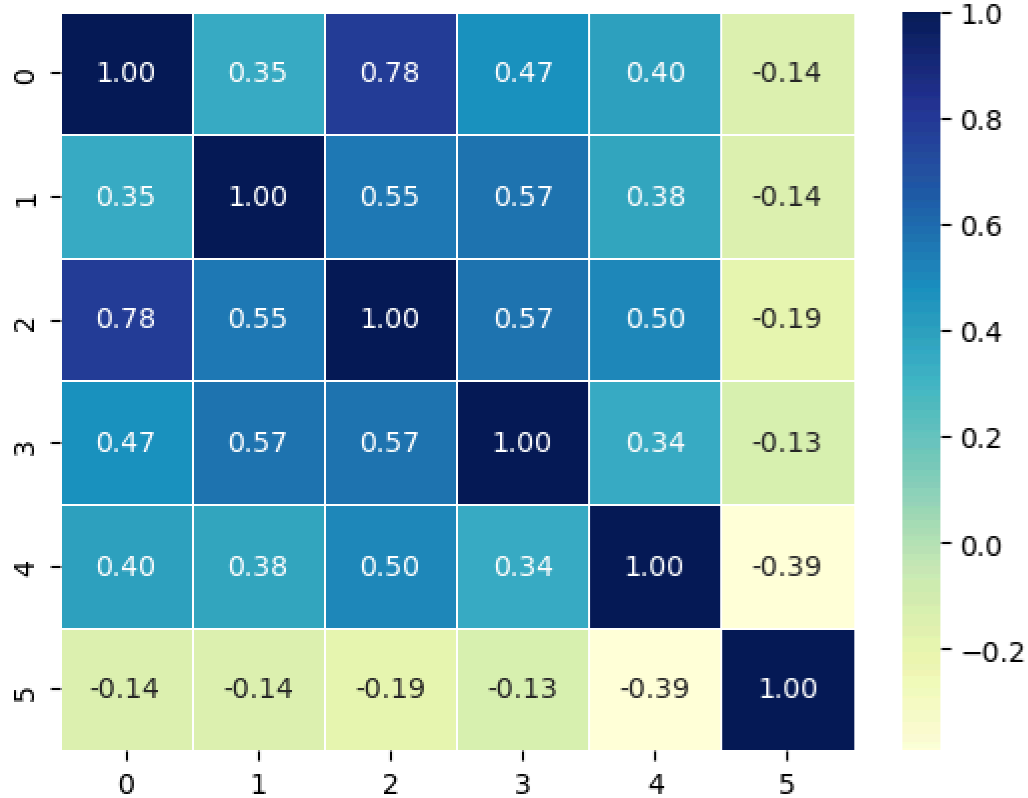


In [ ]:
```
chosen = np.append(chosen, np.arange(36, 40))
```

In [ ]:
```
## Finding the correlation between next 6 features
## which correspond to F1
corr8 = np.empty((6, 6))
```

```
for i in np.arange(40, 46):
    for j in np.arange(40, 46):
        corr8[i - 40, j - 40] = np.corrcoef(np_train[:, i], np_train[:, j])[

sns.heatmap(corr8, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.2
```

Out[ ]: <Axes: >



From the above correlation plot, we see that there is strong correlation in the following groups :

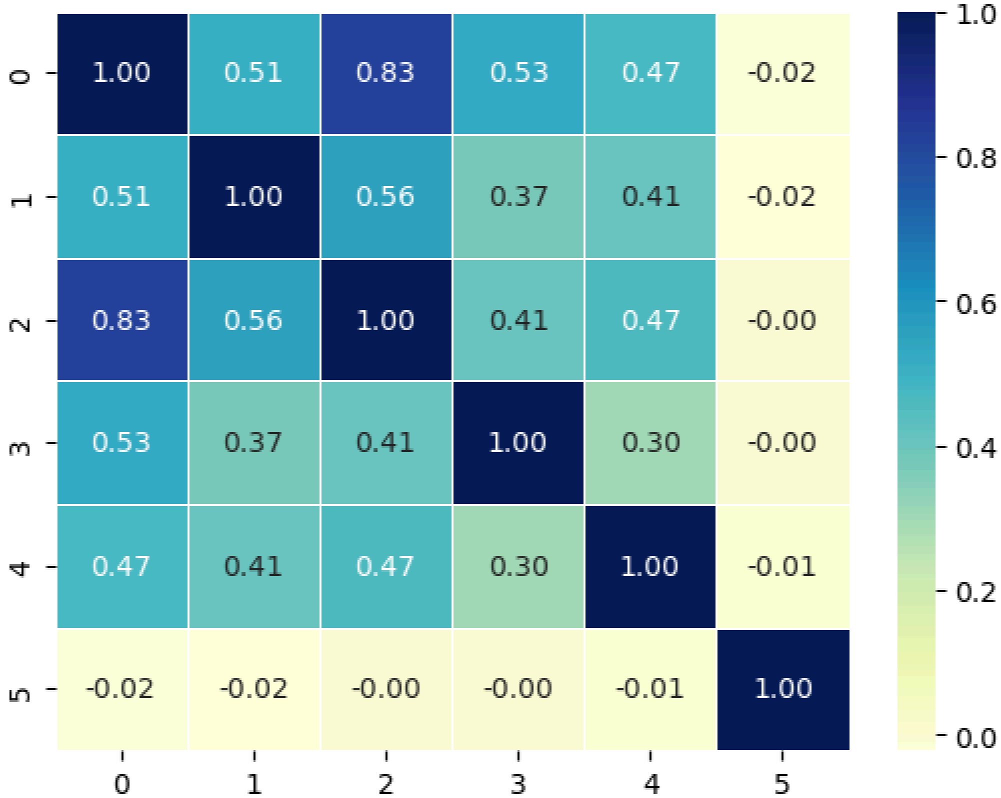
- 0, 1, 2, 3, 4
- 5

Thus, we can select one feature from each of these feature groups.

In [ ]:
```
chosen = np.append(chosen, [40, 45])
```

In [ ]:
```
## Finding the correlation between next 6 features
## which correspond to F2
corr9 = np.empty((6, 6))
for i in np.arange(46, 52):
    for j in np.arange(46, 52):
        corr9[i - 46, j - 46] = np.corrcoef(np_train[:, i], np_train[:, j])[

sns.heatmap(corr9, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.2
```

Out[ ]: <Axes: >



From the above correlation plot, we see that there is strong correlation in the following groups :

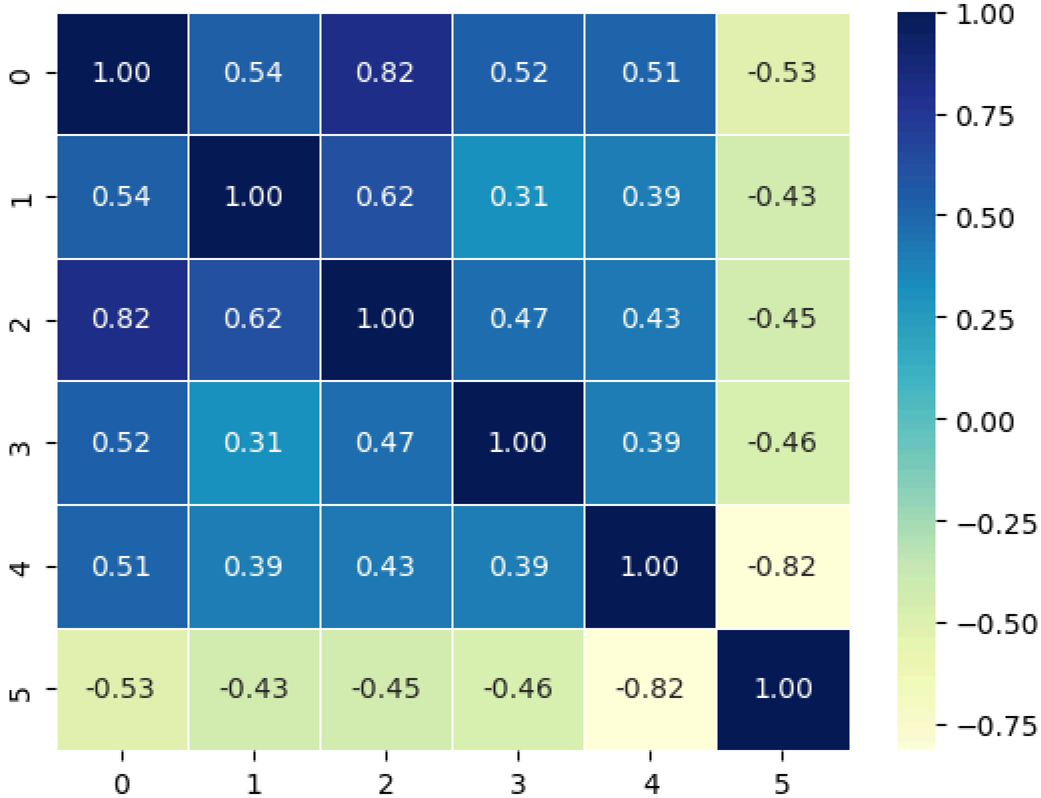
- 0, 1, 2
- 3
- 4
- 5

Thus, we can select one feature from each of these feature groups.

In [ ]:
```
chosen = np.append(chosen, [46, 49, 50, 51])
```

In [ ]:
```
## Finding the correlation between next 6 features
## which correspond to F3
corr10 = np.empty((6, 6))
for i in np.arange(52, 58):
    for j in np.arange(52, 58):
        corr10[i - 52, j - 52] = np.corrcoef(np_train[:, i], np_train[:, j])

sns.heatmap(corr10, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.
```

Out[ ]: `<Axes: >`



From the above correlation plot, we see that there is strong correlation in the following groups :

- 0, 1, 2, 3
- 4, 5

Thus, we can select one feature from each of these feature groups.

In [ ]:
```
chosen = np.append(chosen, [52, 56])
```

In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to alphaRatio
corr11 = np.corrcoef(np_train[:, 58], np_train[:, 59])
corr11[0, 1]
```

Out[ ]: `-0.11379120043860923`

In [ ]:
```
chosen = np.append(chosen, [58, 59])
```

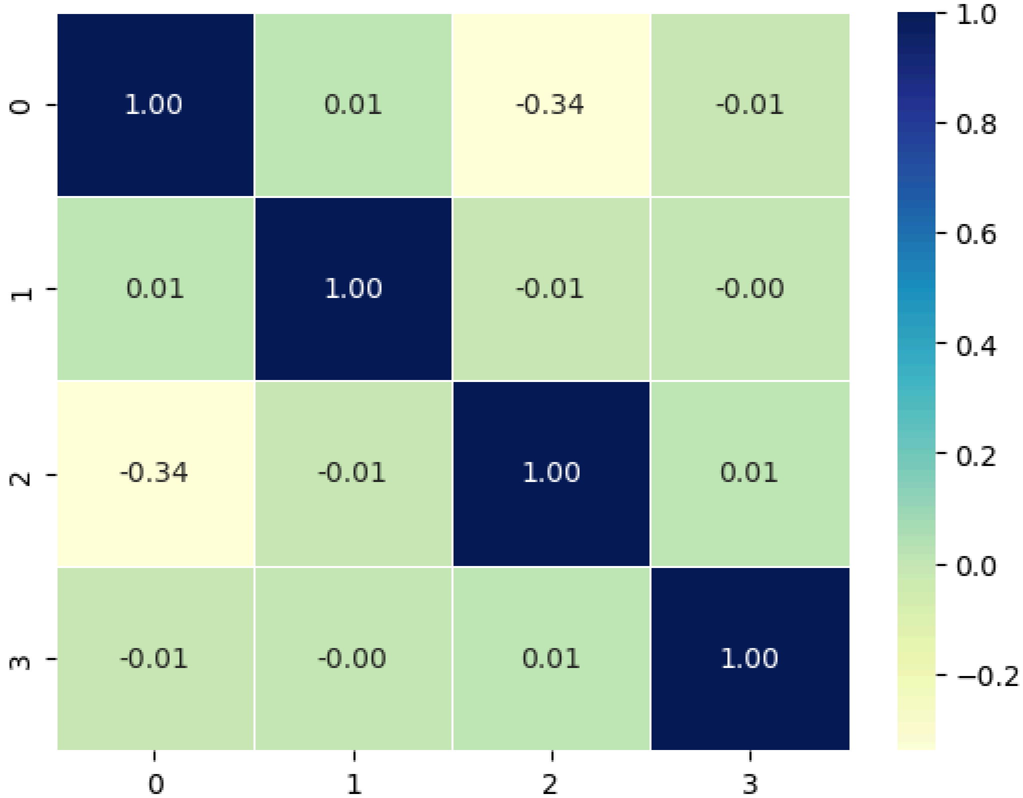
In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to hammarbergIndex
corr12 = np.corrcoef(np_train[:, 60], np_train[:, 61])
corr12[0, 1]
```

Out[ ]: `0.06469415227844454`

In [ ]:
```
chosen = np.append(chosen, [60, 61])
```

In [ ]:
```
## Finding the correlation between next 4 features
## which correspond to slopeV
corr13 = np.empty((4, 4))
for i in np.arange(62, 66):
    for j in np.arange(62, 66):
        corr13[i - 62, j - 62] = np.corrcoef(np_train[:, i], np_train[:, j])

sns.heatmap(corr13, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.
```

Out[ ]: <Axes: >



In [ ]:
```
chosen = np.append(chosen, np.arange(62, 66))
```

In [ ]:
```
## Finding the correlation between next 2 features
## which correspond to spectralFluxV
corr14 = np.corrcoef(np_train[:, 66], np_train[:, 67])
corr14[0, 1]
```

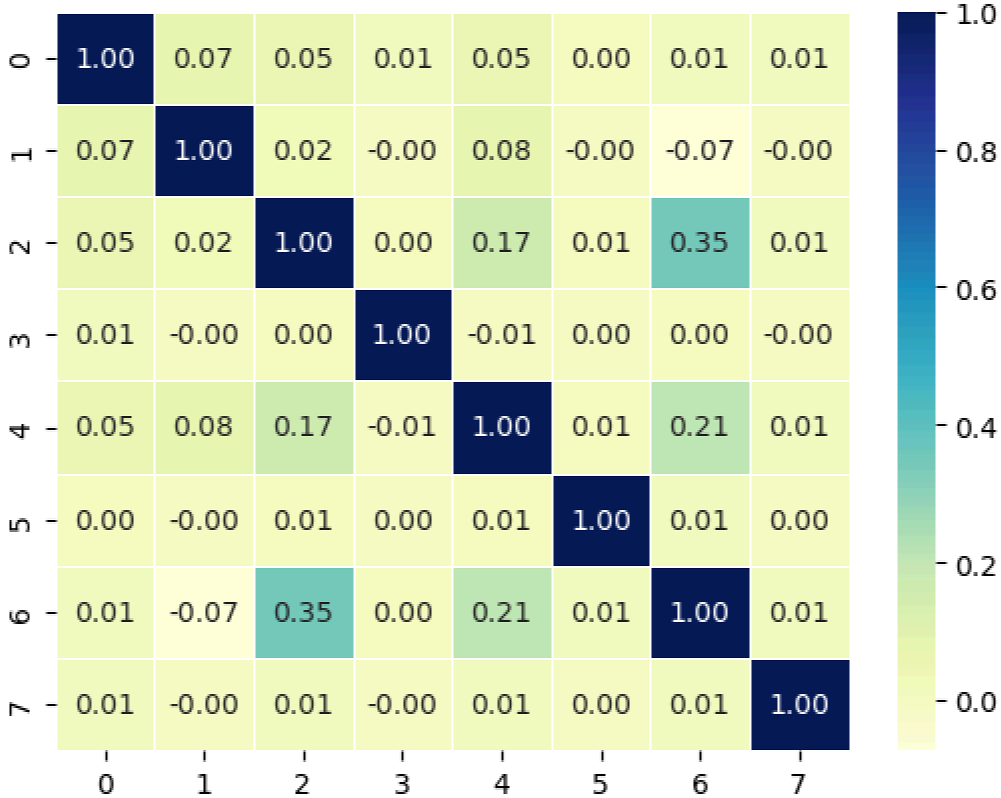
Out[ ]: 0.2681690038025103

In [ ]:
```
chosen = np.append(chosen, [66, 67])
```

In [ ]:
```
## Finding the correlation between next 8 features
## which correspond to mfcc
corr15 = np.empty((8, 8))
for i in np.arange(68,76):
    for j in np.arange(68, 76):
        corr15[i - 68, j - 68] = np.corrcoef(np_train[:, i], np_train[:, j])
```

```
sns.heatmap(corr15, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.
```

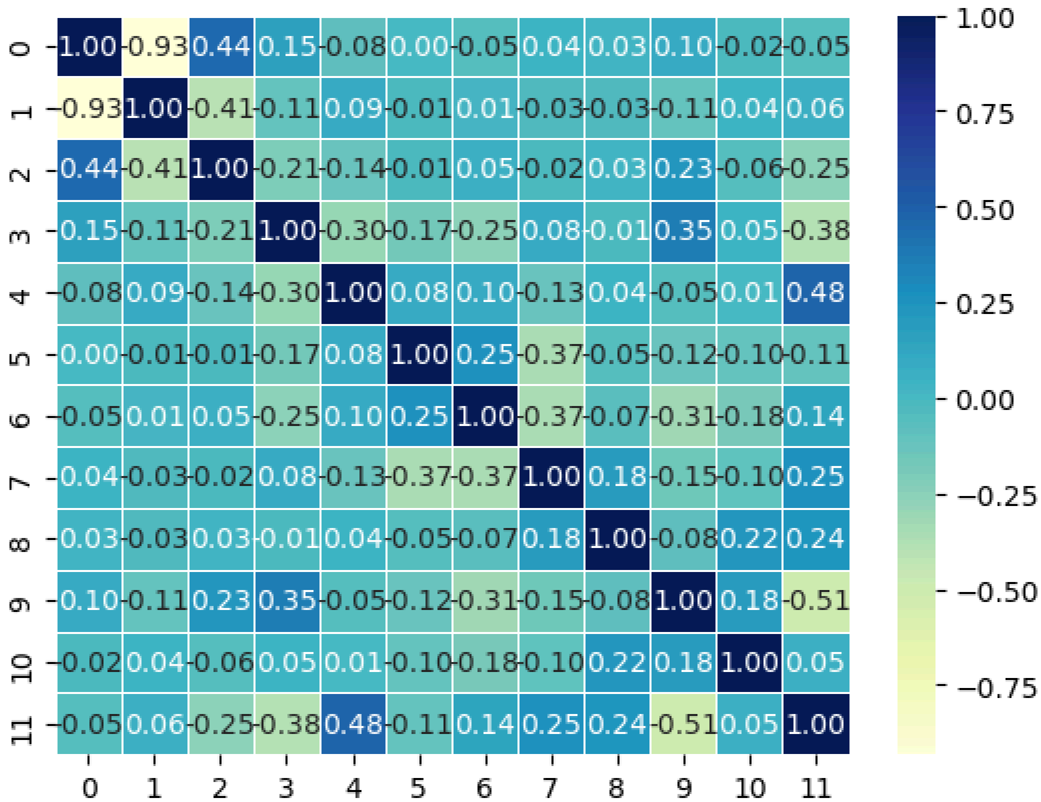
Out[ ]: <Axes: >



In [ ]:
```
chosen = np.append(chosen, np.arange(68,76))
```

In [ ]:
```
corr16 = np.empty((12, 12))
for i in np.arange(76,88):
    for j in np.arange(76, 88):
        corr16[i - 76, j - 76] = np.corrcoef(np_train[:, i], np_train[:, j])

sns.heatmap(corr16, linewidths = 0.5, cmap="YlGnBu", annot = True, fmt = '0.
```

Out[ ]: <Axes: >



In [ ]:
```
chosen = np.append(chosen, [76])
chosen = np.append(chosen, np.arange(78, 88))
```

In [ ]:
```
chosen
```

Out[ ]:
```
array([ 0,  1,  6,  7, 10, 11, 16, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28,
       29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 45, 46, 49, 50, 51,
       52, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
       73, 74, 75, 76, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87])
```

In [ ]:
```
chosen.shape
```

Out[ ]:
```
(65,)
```

Thus, from correlations within the features, we have brought the number of features down from 88 to 65. Now, to showcase the m features that are most informative of gender on the training data, we check their correlations with gender :

In [ ]:
```
gender = df_train.iloc[:, -1]
corr_with_gender = np.zeros((65, 1))
j = 0

for i in chosen:
    corr_with_gender[j] = np.corrcoef(gender, np_train[:, i])[0, 1]
    j = j + 1

sorted_indices = np.flip(np.argsort(np.ndarray.flatten(np.abs(corr_with_genc
```

In [ ]:
```
for m in np.arange(5, 31, 5):
    print("Best ", m, "features based on filtering : " )
    print(sorted_indices[0:m])
```

```
Best  5 features based on filtering :
[22  0 40 52 16]
Best  10 features based on filtering :
[22  0 40 52 16 19 50 14 55 21]
Best  15 features based on filtering :
[22  0 40 52 16 19 50 14 55 21 28 45 39 48 20]
Best  20 features based on filtering :
[22  0 40 52 16 19 50 14 55 21 28 45 39 48 20 31 56 30 37 46]
Best  25 features based on filtering :
[22  0 40 52 16 19 50 14 55 21 28 45 39 48 20 31 56 30 37 46  9  8  2 12
 57]
Best  30 features based on filtering :
[22  0 40 52 16 19 50 14 55 21 28 45 39 48 20 31 56 30 37 46  9  8  2 12
 57 44 27  3 34 58]
```

In [ ]:
```
sorted_indices
# Decreasing order of importance on gender
```

Out[ ]:
```
array([22,  0, 40, 52, 16, 19, 50, 14, 55, 21, 28, 45, 39, 48, 20, 31, 56,
       30, 37, 46,  9,  8,  2, 12, 57, 44, 27,  3, 34, 58, 35, 32, 64, 36,
       10, 47, 26, 42, 54, 59,  7, 23, 61,  5, 29, 62, 18, 24, 63, 11, 53,
       60, 43,  4, 15, 41, 49, 33,  1, 25, 51, 13,  6, 38, 17])
```

Thus, we have used filter feature selection method to get a list of indices of features in the descending order of their significance in conveying gender - the feature with highest correlation with gender is 22, then 0 and so on.

In [ ]:
```
dep = df_train.iloc[:, -2]
corr_with_dep = np.zeros((65, 1))
j = 0

for i in chosen:
    corr_with_dep[j] = np.corrcoef(dep, np_train[:, i])[0, 1]
    j = j + 1

sorted_indices = np.flip(np.argsort(np.ndarray.flatten(np.abs(corr_with_dep)
```

In [ ]:
```
sorted_indices
# Decreasing order of importance on depression
```

Out[ ]:
```
array([ 4, 44,  8, 22,  6,  0,  7, 45, 64, 14, 28, 24, 52, 16, 50, 19, 40,
       21,  5, 57, 34, 30,  3,  9,  2, 31, 27, 32, 35, 58, 12, 46, 38, 62,
       20, 10, 39, 54, 47, 61, 56, 37, 60, 23, 11, 51, 48, 18, 49, 53, 13,
       55, 29, 36, 33, 42,  1, 59, 26, 43, 25, 15, 17, 63, 41])
```

Problem (b)

List of features in decreasing order of significance on depression:

```
In [ ]: sorted_indices = np.array([ 4, 44,  8, 22,  6,  0,  7, 45, 64, 14, 28, 24, 5
               21,  5, 57, 34, 30,  3,  9,  2, 31, 27, 32, 35, 58, 12, 46, 38, 62,
               20, 10, 39, 54, 47, 61, 56, 37, 60, 23, 11, 51, 48, 18, 49, 53, 13,
               55, 29, 36, 33, 42,  1, 59, 26, 43, 25, 15, 17, 63, 41])


        dacc = np.zeros(np.shape(np.arange(10, 51, 5))[0])
        dbal = np.zeros(np.shape(np.arange(10, 51, 5))[0])

        dacc_males = np.zeros(np.shape(np.arange(10, 51, 5))[0])
        dbal_males = np.zeros(np.shape(np.arange(10, 51, 5))[0])

        dacc_females = np.zeros(np.shape(np.arange(10, 51, 5))[0])
        dbal_females = np.zeros(np.shape(np.arange(10, 51, 5))[0])

        deo = np.zeros(np.shape(np.arange(10, 51, 5))[0])
        index = 0

        logr = LogisticRegression(max_iter = 1000)

        for m in np.arange(10, 51, 5):

            X_train_norm = StandardScaler().fit_transform(X_train.iloc[:, sorted_inc
```

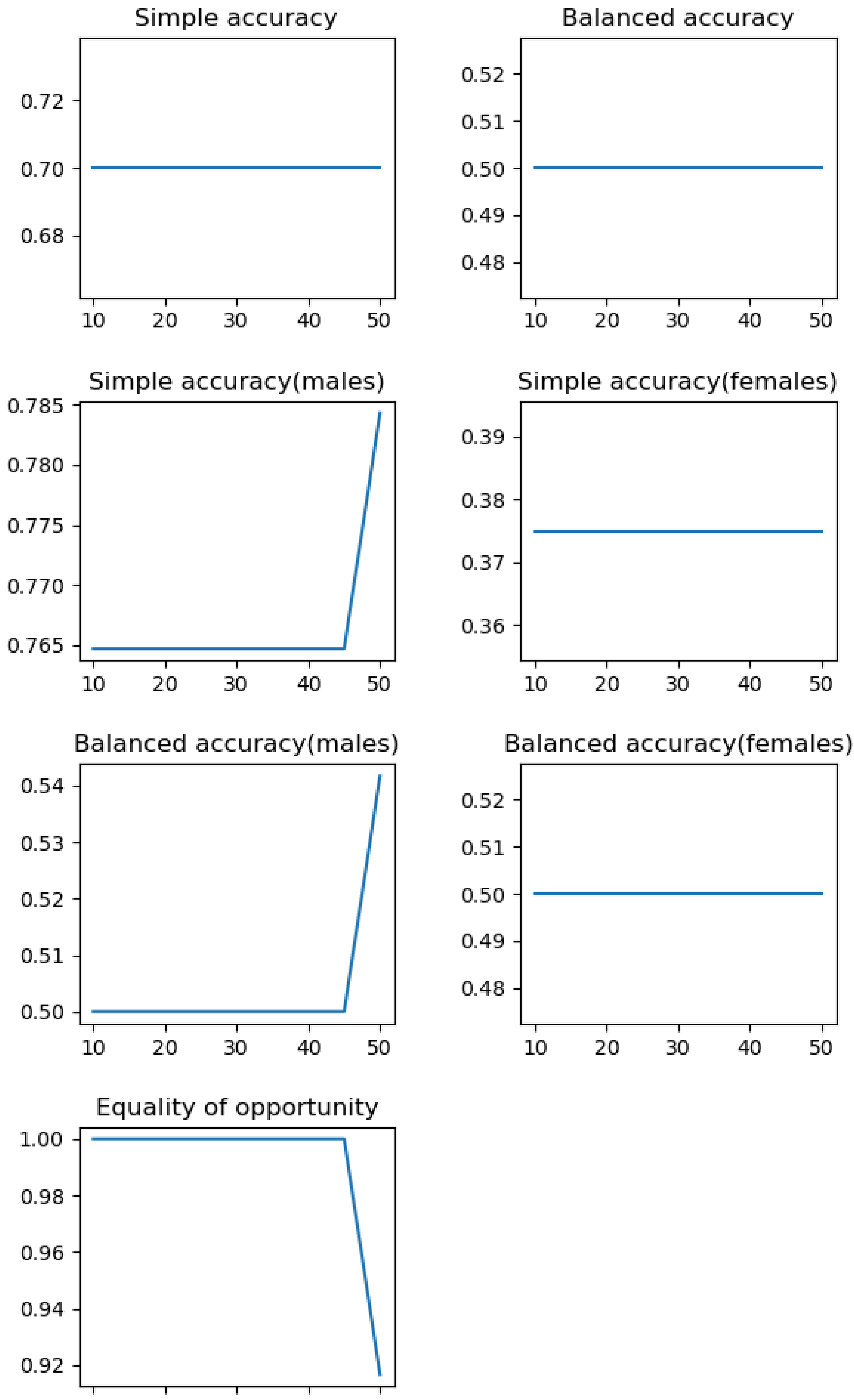
```
    X_test_norm = StandardScaler().fit_transform(X_test.iloc[:, sorted_indic

    model = logr.fit(X_train_norm, y_train)
    #model = rfclassifier.fit(X_train, y_train)

    y_pred = model.predict(X_test_norm)
    #y_pred = model.predict(X_test)
    y_train_total = np.empty((np.shape(y_train)[0], 4))

    y_train_total[:, 0] = df_train.iloc[:, 0]  # ID
    y_train_total[:, 1] = df_train.iloc[:, -2]  # Depression true value
    y_train_total[:, 2] = df_train.iloc[:, -1]  # Gender true value
    y_train_total[:, 3] = model.predict(X_train_norm)   # For logistic regre
    #y_train_total[:, 3] = model.predict(X_train)  # Depression predicted va

    y_train_total_df = pd.DataFrame(y_train_total)

    y_test_total = np.empty((np.shape(y_test)[0], 4))
    y_test_total[:, 0] = df_test.iloc[:, 0]
    y_test_total[:, 1] = df_test.iloc[:, -2]
    y_test_total[:, 2] = df_test.iloc[:, -1]
    y_test_total[:, 3] = y_pred
    y_test_total_df = pd.DataFrame(y_test_total)

    train_groups = y_train_total_df.groupby(y_train_total_df[0])   # Groupir
    test_groups = y_test_total_df.groupby(y_test_total_df[0])

    train_true = train_groups[1].aggregate(pd.Series.mode)
    train_prediction = train_groups[3].aggregate(pd.Series.mode)

    test_true = test_groups[1].aggregate(pd.Series.mode)
    test_prediction = test_groups[3].aggregate(pd.Series.mode)

    #print("Simple accuracy for all participants in training set:", accuracy
    #print("Balanced accuracy for all partipants in training set:", balancec


    #print("Simple accuracy for all participants in test set:", accuracy_scc
    #print("Balanced accuracy for all partipants in test set:", balanced_acc

    dacc[index] = accuracy_score(test_true, test_prediction)
    dbal[index] = balanced_accuracy_score(test_true, test_prediction)

    ### Splitting the data into female and male

    train_males_df = pd.DataFrame(y_train_total[np.argwhere(y_train_total[:,
    train_females_df = pd.DataFrame(y_train_total[np.argwhere(y_train_total[

    test_males_df = pd.DataFrame(y_test_total[np.argwhere(y_test_total[:, 2]
    test_females_df = pd.DataFrame(y_test_total[np.argwhere(y_test_total[:,

    train_males_groups = train_males_df.groupby(train_males_df[0])     # Grc
    train_females_groups = train_females_df.groupby(train_females_df[0])

    test_males_groups = test_males_df.groupby(test_males_df[0])
    test_females_groups = test_females_df.groupby(test_females_df[0])
```

```
    train_true_males = train_males_groups[1].aggregate(pd.Series.mode)
    train_prediction_males = train_males_groups[3].aggregate(pd.Series.mode)

    train_true_females = train_females_groups[1].aggregate(pd.Series.mode)
    train_prediction_females = train_females_groups[3].aggregate(pd.Series.m

    test_true_males = train_males_groups[1].aggregate(pd.Series.mode)
    test_prediction_males = train_males_groups[3].aggregate(pd.Series.mode)

    test_true_females = test_females_groups[1].aggregate(pd.Series.mode)
    test_prediction_females = test_females_groups[3].aggregate(pd.Series.moc

    #print("Simple accuracy for males in training set:", accuracy_score(trai
    #print("Balanced accuracy for males in training set:", balanced_accuracy


    #print("Simple accuracy for females in training set:", accuracy_score(tr
    #print("Balanced accuracy for females in training set:", balanced_accura

    #print("Simple accuracy for males in test set:", accuracy_score(test_tru
    #print("Balanced accuracy for males in test set:", balanced_accuracy_scc

    dacc_males[index] = accuracy_score(test_true_males, test_prediction_male
    dbal_males[index] = balanced_accuracy_score(test_true_males, test_predic

    #print("Simple accuracy for females in test set:", accuracy_score(test_t
    #print("Balanced accuracy for females in test set:", balanced_accuracy_s

    dacc_females[index] = accuracy_score(test_true_females, test_prediction_
    dbal_females[index] = balanced_accuracy_score(test_true_females, test_pr



    tpr_males = recall_score(test_true_males, test_prediction_males)
    tpr_females = recall_score(test_true_females, test_prediction_females)
    deo[index] = 1 - np.abs(tpr_males - tpr_females)
    index = index + 1
```

In [ ]:

```
plt.subplot(4, 2, 1)
plt.plot(np.arange(10, 51, 5), dacc)
plt.title("Simple accuracy")

plt.subplot(4, 2, 2)
plt.plot(np.arange(10, 51, 5), dbal)
plt.title("Balanced accuracy")

plt.subplot(4, 2, 3)
plt.plot(np.arange(10, 51, 5), dacc_males)
plt.title("Simple accuracy(males)")

plt.subplot(4, 2, 4)
```

```
plt.plot(np.arange(10, 51, 5), dacc_females)
plt.title("Simple accuracy(females)")

plt.subplot(4, 2, 5)
plt.plot(np.arange(10, 51, 5), dbal_males)
plt.title("Balanced accuracy(males)")

plt.subplot(4, 2, 6)
plt.plot(np.arange(10, 51, 5), dbal_females)
plt.title("Balanced accuracy(females)")

plt.subplot(4, 2, 7)
plt.plot(np.arange(10, 51, 5), deo)
plt.title("Equality of opportunity")

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=2,
                    wspace=0.4,
                    hspace=0.4)

plt.show()
```


Simple accuracy
0.72
0.70
0.68
10 20 30 40 50
Balanced accuracy
0.52
0.51
0.50
0.49
0.48
10 20 30 40 50
Simple accuracy(males)
0.785
0.780
0.775
0.770
0.765
10 20 30 40 50
Simple accuracy(females)
0.39
0.38
0.37
0.36
10 20 30 40 50
Balanced accuracy(males)
0.54
0.53
0.52
0.51
0.50
10 20 30 40 50
Balanced accuracy(females)
0.52
0.51
0.50
0.49
0.48
10 20 30 40 50
Equality of opportunity
1.00
0.98
0.96
0.94
0.92
10 20 30 40 50

In the above plots, the x-axis represents the number of most important features considered. We see that the simple and balanced accuracy for all participants remain the same when considering only a a subset of the features, but for males the accuracies increase with number of features. This is expected as increase in features typically leads to increase in information. There is barely any change in the other accuracies.

Problem (c)

Computing importance of features from the model:

In [ ]:
```
importances = rf_model.feature_importances_

imp_dict = {}
for id,imp in enumerate(importances):
    imp_dict[id] = imp
    # print('Feature: %0d, Importance Score: %.5f' %(id, imp))
for feature in sorted(imp_dict,key=imp_dict.get):
    print('Feature: %0d, Importance Score: %.5f' %(feature, imp_dict[feature
```

```
Feature: 7, Importance Score: 0.00110
Feature: 9, Importance Score: 0.00122
Feature: 57, Importance Score: 0.00132
Feature: 86, Importance Score: 0.00138
Feature: 51, Importance Score: 0.00159
Feature: 65, Importance Score: 0.00163
Feature: 84, Importance Score: 0.00165
Feature: 45, Importance Score: 0.00168
Feature: 36, Importance Score: 0.00177
Feature: 6, Importance Score: 0.00181
Feature: 67, Importance Score: 0.00182
Feature: 71, Importance Score: 0.00185
Feature: 50, Importance Score: 0.00187
Feature: 48, Importance Score: 0.00191
Feature: 38, Importance Score: 0.00192
Feature: 73, Importance Score: 0.00202
Feature: 44, Importance Score: 0.00205
Feature: 64, Importance Score: 0.00211
Feature: 54, Importance Score: 0.00214
Feature: 56, Importance Score: 0.00220
Feature: 43, Importance Score: 0.00223
Feature: 19, Importance Score: 0.00233
Feature: 30, Importance Score: 0.00241
Feature: 82, Importance Score: 0.00251
Feature: 35, Importance Score: 0.00254
Feature: 49, Importance Score: 0.00257
Feature: 8, Importance Score: 0.00263
Feature: 21, Importance Score: 0.00273
Feature: 17, Importance Score: 0.00277
Feature: 11, Importance Score: 0.00277
Feature: 60, Importance Score: 0.00284
Feature: 69, Importance Score: 0.00285
Feature: 23, Importance Score: 0.00293
Feature: 32, Importance Score: 0.00293
Feature: 25, Importance Score: 0.00295
Feature: 39, Importance Score: 0.00296
Feature: 70, Importance Score: 0.00306
Feature: 58, Importance Score: 0.00310
Feature: 81, Importance Score: 0.00313
Feature: 12, Importance Score: 0.00313
Feature: 72, Importance Score: 0.00316
Feature: 33, Importance Score: 0.00330
Feature: 27, Importance Score: 0.00335
Feature: 59, Importance Score: 0.00341
Feature: 83, Importance Score: 0.00342
Feature: 77, Importance Score: 0.00361
Feature: 16, Importance Score: 0.00367
Feature: 22, Importance Score: 0.00380
Feature: 37, Importance Score: 0.00383
Feature: 24, Importance Score: 0.00383
Feature: 68, Importance Score: 0.00384
Feature: 18, Importance Score: 0.00386
Feature: 76, Importance Score: 0.00390
Feature: 1, Importance Score: 0.00417
Feature: 15, Importance Score: 0.00424
Feature: 55, Importance Score: 0.00472
```

```
Feature: 85, Importance Score: 0.00480
Feature: 31, Importance Score: 0.00483
Feature: 53, Importance Score: 0.00501
Feature: 80, Importance Score: 0.00512
Feature: 14, Importance Score: 0.00517
Feature: 26, Importance Score: 0.00533
Feature: 78, Importance Score: 0.00537
Feature: 29, Importance Score: 0.00538
Feature: 13, Importance Score: 0.00576
Feature: 10, Importance Score: 0.00585
Feature: 61, Importance Score: 0.00597
Feature: 79, Importance Score: 0.00601
Feature: 5, Importance Score: 0.00623
Feature: 66, Importance Score: 0.00631
Feature: 40, Importance Score: 0.00662
Feature: 75, Importance Score: 0.00790
Feature: 20, Importance Score: 0.00798
Feature: 63, Importance Score: 0.00917
Feature: 42, Importance Score: 0.00994
Feature: 52, Importance Score: 0.01065
Feature: 28, Importance Score: 0.01172
Feature: 87, Importance Score: 0.01177
Feature: 47, Importance Score: 0.01296
Feature: 46, Importance Score: 0.01374
Feature: 74, Importance Score: 0.01869
Feature: 41, Importance Score: 0.02999
Feature: 62, Importance Score: 0.04516
Feature: 34, Importance Score: 0.07159
Feature: 4, Importance Score: 0.09303
Feature: 2, Importance Score: 0.10566
Feature: 0, Importance Score: 0.14902
Feature: 3, Importance Score: 0.15676
```

keeping top 20 important features

In [ ]:
```
updated_columns = [5, 66, 40, 75, 20, 63, 42, 52, 28, 87, 47, 46, 74, 41, 62

updated_rf_model = model_pipeline(RFC, X_train[updated_columns],test = testd
```

```
CV accuracies:  [0.95009174 0.92587156 0.97724771 0.86348624 0.92036697]
Average CV accuracy 0.9274128440366972
Train accuracy:  0.9992660550458715
Test accuracy:  [1.]
Test balanced accuracy:  [1.]
```

In [ ]:
```
# based on correlation
selected_columns= [22,0,38,50,16]
top5_rf_model = model_pipeline(RFC, X_train[selected_columns],test = testdat
print()
selected_columns = [22, 0, 38, 50, 16, 19, 48, 14, 53, 21]
top10_rf_model = model_pipeline(RFC, X_train[selected_columns],test = testda
print()
selected_columns=[22, 0, 38, 50, 16, 19, 48, 14, 53, 21, 28, 43, 46, 20, 31]
top15_rf_model = model_pipeline(RFC, X_train[selected_columns],test = testda
```

```
CV accuracies:  [0.92844037 0.90899083 0.96183486 0.82238532 0.89357798]
Average CV accuracy 0.903045871559633
Train accuracy:  0.9991192660550459
Test accuracy:  [1.]
Test balanced accuracy:  [1.]

CV accuracies:  [0.94165138 0.90972477 0.96256881 0.83522936 0.90201835]
Average CV accuracy 0.9102385321100919
Train accuracy:  0.9992660550458715
Test accuracy:  [1.]
Test balanced accuracy:  [1.]

CV accuracies:  [0.94385321 0.91302752 0.96880734 0.8440367  0.9133945 ]
Average CV accuracy 0.9166238532110093
Train accuracy:  0.9992660550458715
Test accuracy:  [1.]
Test balanced accuracy:  [1.]
```

In [ ]:

```python
selected_columns=[22, 0, 38, 50, 16, 19, 48, 14, 53, 21, 28, 43, 46, 20, 31,
top20_rf_model = model_pipeline(RFC, X_train[selected_columns],test = testda
print()
selected_columns=[22, 0, 38, 50, 16, 19, 48, 14, 53, 21, 28, 43, 46, 20, 31,
top25_rf_model = model_pipeline(RFC, X_train[selected_columns],test = testda
```

```
CV accuracies:  [0.94605505 0.91376147 0.9666055  0.84587156 0.91082569]
Average CV accuracy 0.9166238532110093
Train accuracy:  0.9992660550458715
Test accuracy:  [1.]
Test balanced accuracy:  [1.]

CV accuracies:  [0.95009174 0.92733945 0.97247706 0.84770642 0.91706422]
Average CV accuracy 0.9229357798165138
Train accuracy:  0.9992660550458715
Test accuracy:  [1.]
Test balanced accuracy:  [1.]
```

In [ ]:

```python
selected_columns=[22, 0, 38, 50, 16, 19, 48, 14, 53, 21, 28, 43, 46, 20, 31,
top25_rf_model = model_pipeline(RFC, X_train[selected_columns],test = testda
```

```
CV accuracies:  [0.95229358 0.92733945 0.97357798 0.84587156 0.92550459]
Average CV accuracy 0.9249174311926606
Train accuracy:  0.9992660550458715
Test accuracy:  [1.]
Test balanced accuracy:  [1.]
```

Problem (d)

All the feature variables' correlation with gender is computed and sorted(descending) to obtain the following array. Thus, column index 22 has the highest Pearson's correlation coefficient with gender, and so on. Removing these features 5 at a time, the change in the performance of the model is studied.

In [ ]:
```
sorted_indices = np.array([22,  0, 40, 52, 16, 19, 50, 14, 55, 21, 28, 45, 3
       30, 37, 46,  9,  8,  2, 12, 57, 44, 27,  3, 34, 58, 35, 32, 64, 36,
       10, 47, 26, 42, 54, 59,  7, 23, 61,  5, 29, 62, 18, 24, 63, 11, 53,
       60, 43,  4, 15, 41, 49, 33,  1, 25, 51, 13,  6, 38, 17])
```

In [ ]:
```
dacc = np.zeros(np.shape(np.arange(0, 65, 5))[0])
dbal = np.zeros(np.shape(np.arange(0, 65, 5))[0])

dacc_males = np.zeros(np.shape(np.arange(0, 65, 5))[0])
dbal_males = np.zeros(np.shape(np.arange(0, 65, 5))[0])

dacc_females = np.zeros(np.shape(np.arange(0, 65, 5))[0])
dbal_females = np.zeros(np.shape(np.arange(0, 65, 5))[0])

deo = np.zeros(np.shape(np.arange(0, 65, 5))[0])
index = 0

logr = LogisticRegression(max_iter = 1000)

for m in np.arange(0, 65, 5):

    X_train_norm = StandardScaler().fit_transform(X_train.iloc[:, sorted_inc
    X_test_norm = StandardScaler().fit_transform(X_test.iloc[:, sorted_indic

    model = logr.fit(X_train_norm, y_train)
    #model = rfclassifier.fit(X_train, y_train)

    y_pred = model.predict(X_test_norm)
    #y_pred = model.predict(X_test)
    y_train_total = np.empty((np.shape(y_train)[0], 4))

    y_train_total[:, 0] = df_train.iloc[:, 0]  # ID
    y_train_total[:, 1] = df_train.iloc[:, -2]  # Depression true value
    y_train_total[:, 2] = df_train.iloc[:, -1]  # Gender true value
    y_train_total[:, 3] = model.predict(X_train_norm)   # For logistic regre
    #y_train_total[:, 3] = model.predict(X_train)  # Depression predicted
```

Screensh

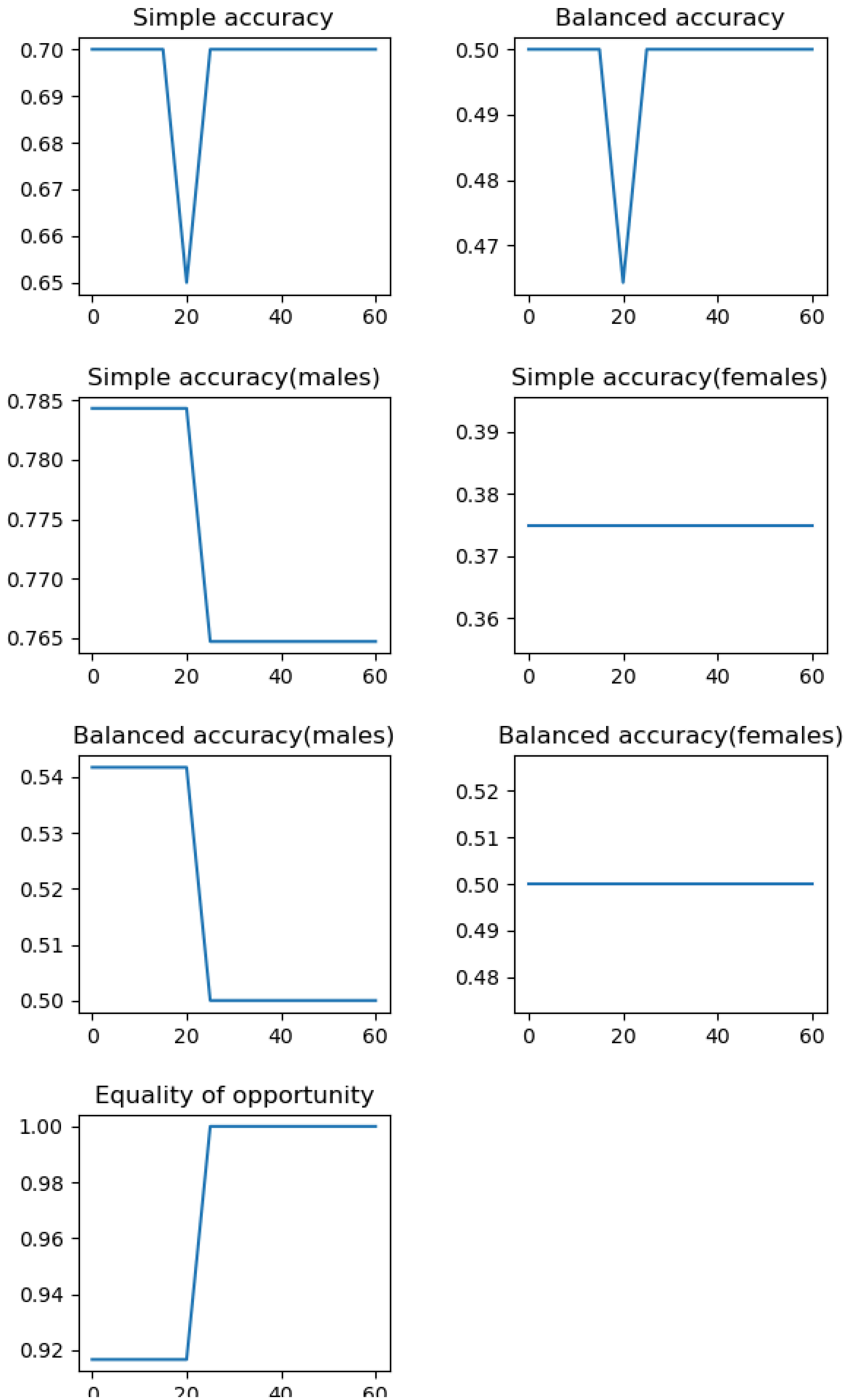
```
    y_train_total_df = pd.DataFrame(y_train_total)

    y_test_total = np.empty((np.shape(y_test)[0], 4))
    y_test_total[:, 0] = df_test.iloc[:, 0]
    y_test_total[:, 1] = df_test.iloc[:, -2]
    y_test_total[:, 2] = df_test.iloc[:, -1]
    y_test_total[:, 3] = y_pred
    y_test_total_df = pd.DataFrame(y_test_total)

    train_groups = y_train_total_df.groupby(y_train_total_df[0])   # Groupir
    test_groups = y_test_total_df.groupby(y_test_total_df[0])

    train_true = train_groups[1].aggregate(pd.Series.mode)
    train_prediction = train_groups[3].aggregate(pd.Series.mode)

    test_true = test_groups[1].aggregate(pd.Series.mode)
    test_prediction = test_groups[3].aggregate(pd.Series.mode)

    #print("Simple accuracy for all participants in training set:", accuracy
    #print("Balanced accuracy for all partipants in training set:", balancec


    #print("Simple accuracy for all participants in test set:", accuracy_scc
    #print("Balanced accuracy for all partipants in test set:", balanced_acc

    dacc[index] = accuracy_score(test_true, test_prediction)
    dbal[index] = balanced_accuracy_score(test_true, test_prediction)

    ### Splitting the data into female and male

    train_males_df = pd.DataFrame(y_train_total[np.argwhere(y_train_total[:,
    train_females_df = pd.DataFrame(y_train_total[np.argwhere(y_train_total[

    test_males_df = pd.DataFrame(y_test_total[np.argwhere(y_test_total[:, 2]
    test_females_df = pd.DataFrame(y_test_total[np.argwhere(y_test_total[:,

    train_males_groups = train_males_df.groupby(train_males_df[0])     # Grc
    train_females_groups = train_females_df.groupby(train_females_df[0])

    test_males_groups = test_males_df.groupby(test_males_df[0])
    test_females_groups = test_females_df.groupby(test_females_df[0])

    train_true_males = train_males_groups[1].aggregate(pd.Series.mode)
    train_prediction_males = train_males_groups[3].aggregate(pd.Series.mode)

    train_true_females = train_females_groups[1].aggregate(pd.Series.mode)
    train_prediction_females = train_females_groups[3].aggregate(pd.Series.m

    test_true_males = train_males_groups[1].aggregate(pd.Series.mode)
    test_prediction_males = train_males_groups[3].aggregate(pd.Series.mode)

    test_true_females = test_females_groups[1].aggregate(pd.Series.mode)
    test_prediction_females = test_females_groups[3].aggregate(pd.Series.moc

    #print("Simple accuracy for males in training set:", accuracy_score(trai
```

```
    #print("Balanced accuracy for males in training set:", balanced_accuracy


    #print("Simple accuracy for females in training set:", accuracy_score(tr
    #print("Balanced accuracy for females in training set:", balanced_accura

    #print("Simple accuracy for males in test set:", accuracy_score(test_tru
    #print("Balanced accuracy for males in test set:", balanced_accuracy_sco

    dacc_males[index] = accuracy_score(test_true_males, test_prediction_male
    dbal_males[index] = balanced_accuracy_score(test_true_males, test_predic

    #print("Simple accuracy for females in test set:", accuracy_score(test_t
    #print("Balanced accuracy for females in test set:", balanced_accuracy_s

    dacc_females[index] = accuracy_score(test_true_females, test_prediction_
    dbal_females[index] = balanced_accuracy_score(test_true_females, test_pr



    tpr_males = recall_score(test_true_males, test_prediction_males)
    tpr_females = recall_score(test_true_females, test_prediction_females)
    deo[index] = 1 - np.abs(tpr_males - tpr_females)
    index = index + 1
```

In [ ]:

```
plt.subplot(4, 2, 1)
plt.plot(np.arange(0, 65, 5), dacc)
plt.title("Simple accuracy")

plt.subplot(4, 2, 2)
plt.plot(np.arange(0, 65, 5), dbal)
plt.title("Balanced accuracy")

plt.subplot(4, 2, 3)
plt.plot(np.arange(0, 65, 5), dacc_males)
plt.title("Simple accuracy(males)")

plt.subplot(4, 2, 4)
plt.plot(np.arange(0, 65, 5), dacc_females)
plt.title("Simple accuracy(females)")

plt.subplot(4, 2, 5)
plt.plot(np.arange(0, 65, 5), dbal_males)
plt.title("Balanced accuracy(males)")

plt.subplot(4, 2, 6)
plt.plot(np.arange(0, 65, 5), dbal_females)
plt.title("Balanced accuracy(females)")

plt.subplot(4, 2, 7)
plt.plot(np.arange(0, 65, 5), deo)
plt.title("Equality of opportunity")
```

```
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=2,
                    wspace=0.4,
                    hspace=0.4)

plt.show()
```


Simple accuracy
0.70
0.69
0.68
0.67
0.66
0.65
0
20
40
60
Balanced accuracy
0.50
0.49
0.48
0.47
0
20
40
60
Simple accuracy(males)
0.785
0.780
0.775
0.770
0.765
0
20
40
60
Simple accuracy(females)
0.39
0.38
0.37
0.36
0
20
40
60
Balanced accuracy(males)
0.54
0.53
0.52
0.51
0.50
0
20
40
60
Balanced accuracy(females)
0.52
0.51
0.50
0.49
0.48
0
20
40
60
Equality of opportunity
1.00
0.98
0.96
0.94
0.92

On removing the most informative features of gender, we see that the equality of opportunity increases, with very little difference (order of 0.01 difference) in the other quantities. Thus, such a practice of removing the most gender informative features seems like a good(in terms of ethics) thing to do, as the EO value increases without much change in the accuracies.