Data Augmentation using GAN's

KJ Schmidt - KJS7715, Aristana Scourtas - ASP1723, Nayan Mehta - NMH3011

In our experiment, we worked with the Pima Indians Diabetes Database on Kaggle. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. We wanted to create an entirely new dataset based on this original dataset that retains important information from the original- which would be useful in solving the problem of restricted access to data due to data regulation and privacy concerns.

We based our approach on the paper Data Augmentation Using GANs by Fabio Henrique K. dos S. Tanaka. (https://github.com/fhtanaka/directed_research_CS_2018)(https://arxiv.org/pdf/1904.09135.pdf)

We experimented with different hyperparameters and architectures to build upon the results of the Tanaka paper. What we present here are our best results.

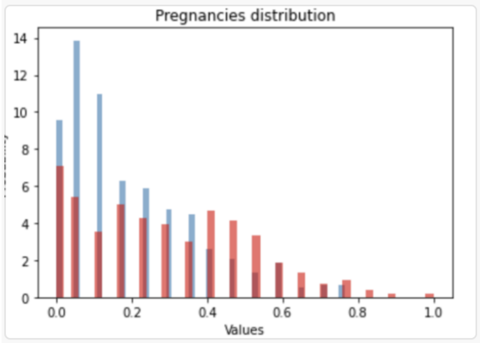
Using generative adversarial networks, or GANs, we can generate a dataset for training. We can solve those issues by:

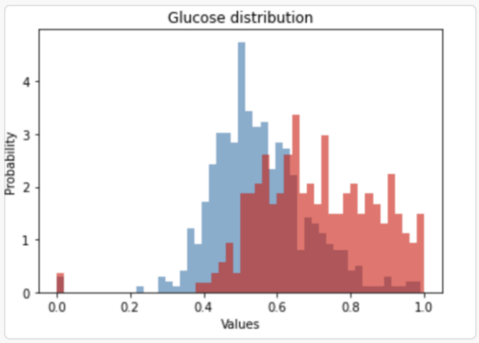
1. Create an entirely new dataset based on the original dataset that retains important information (which would help in instances where we can't use the original dataset because of data regulation and privacy concerns)
2. Balancing the dataset by oversampling the minority class (which would help in instances of detecting fraudulent payments identifying cancerous tumors)

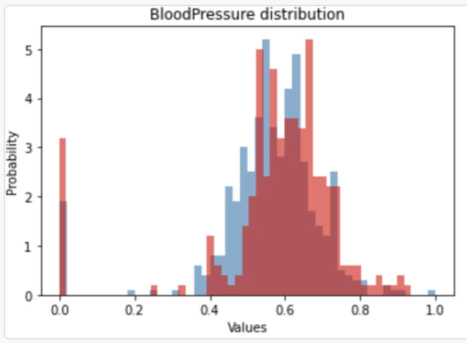
```
#1 Data Analysis
# We first plot our database to look at the distribution more closely (Check if dataset is balanced)
data = pd.read_csv(file_name)
print(dataAtts.message, "\n")
print(dataAtts.values_names[0], round(data[dataAtts.class_name].value_counts()[0]/len(data) * 100,2), '%  of the dataset')
print(dataAtts.values_names[1], round(data[dataAtts.class_name].value_counts()[1]/len(data) * 100,2), '%  of the dataset')
```

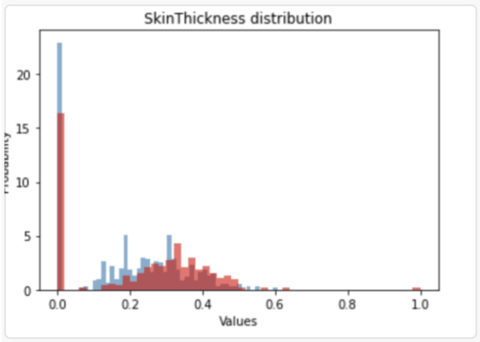
```
Pima Indians Diabetes Database eSCALONATED

Normal 65.1 %  of the dataset
Diabets 34.9 %  of the dataset
```

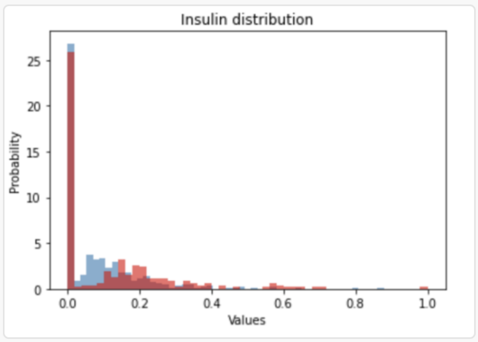
```
#As can be seen our dataset is imbalanced and likely
for name in classes:
    if name==dataAtts.class_name or name=="Unnamed: 32":
        continue
    plt.xlabel('Values')
    plt.ylabel('Probability')
    plt.title(name + " distribution")
    #Collecting corresponding class values where the Outcome is 0/1 (Color red being 1 and blue being 0)
    fraud_dist = data[name].loc[data[dataAtts.class_name] == 1].values
    common_dist = data[name].loc[data[dataAtts.class_name] == 0].values
    plt.hist(common_dist, 50, density=True, alpha=0.6)
    plt.hist(fraud_dist, 50, density=True, alpha=0.6, facecolor='r')
    plt.savefig('images/'+ dataAtts.fname + "/"+name+'_distribution.png')
    plt.show()
    plt.clf()
```

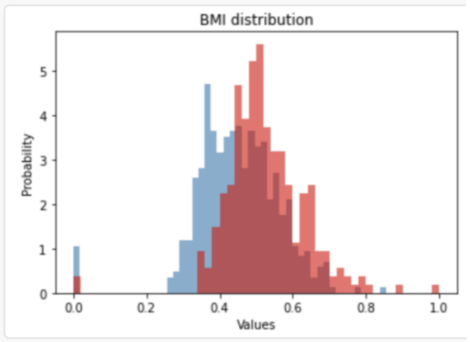


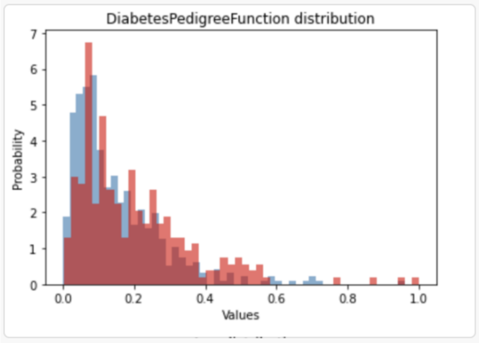


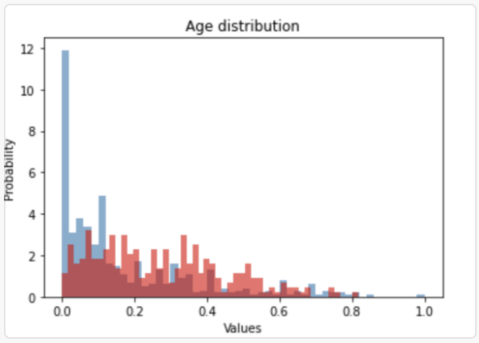






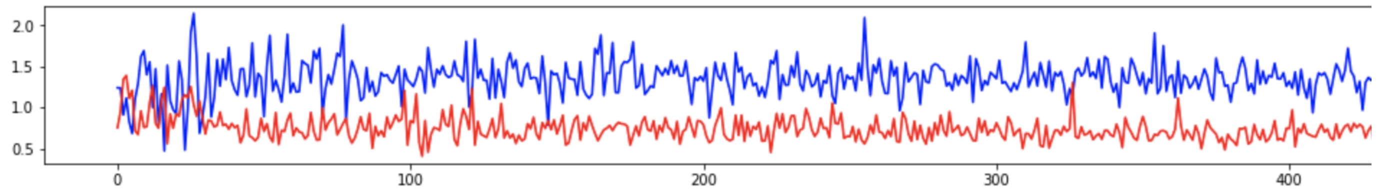




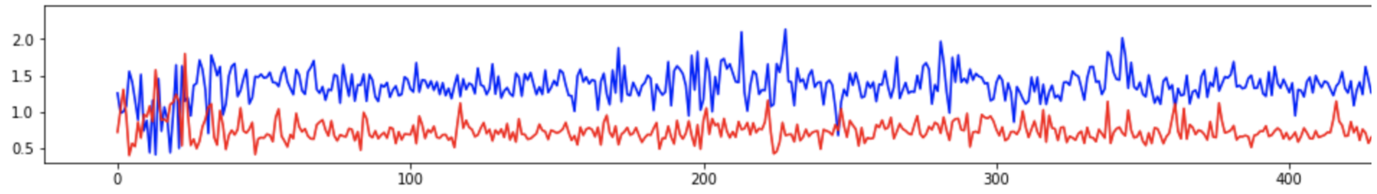


```
#2 Train Generator and Discriminator
# Here we train our generator and discriminator with different model params
# We experimented with lr= [0.002, 0.0002], batch_size=[5,15], hidden_layers=[[256, 512], [256], [128, 256], [128]]
# We found the best results for the lr=0.002, 256 Hidden Layer, batch_size = 5, with an Accuracy of 79.1%
# This was found to be higher than the result of 74.8% reported by the Tanaka paper
class Architecture():
    def __init__(self, learning_rate, batch_size, loss, hidden_layers, name):
        self.learning_rate=learning_rate
        self.batch_size=batch_size
        self.loss=loss
        self.hidden_layers=hidden_layers
        self.name=name

def save_model(name, epoch, attributes, dictionary, optimizer_dictionary, loss_function, db_name, arch_name):
    torch.save({
        'epoch': epoch,
        'model_attributes': attributes,
        'model_state_dict': dictionary,
        'optimizer_state_dict': optimizer_dictionary,
        'loss': loss_function
    }, "models/" + db_name + "/" + name + "_" + arch_name + ".pt")


# Check if creditcard.csv exists and if so, create a scalonated version of it
# escalonate_creditcard_db()
if not os.path.isfile('./original_data/diabetes_escalonated.csv'):
    print("Database creditcard.csv not found, exiting...")
    exit()

file_names=["original_data/diabetes_escalonated.csv"]
num_epochs=[500]
learning_rate=[0.0002]
batch_size=[5]
number_of_experiments = 5
hidden_layers=[[256, 512], [256], [128, 256], [128]]


#create the different architetures
#Only entity being changed here is the hidden_layer array and the number of experiment
architectures=[]
count=0
for lr in learning_rate:
    for b_size in batch_size:
        for hidden in hidden_layers:
            for i in range(number_of_experiments):
                name = "id-" + str(count)
                name += "_epochs-" + str(num_epochs[0])
                name += "_layer-" + str(len(hidden))
                name += "_lr-" + str(lr)
                name += "_batch-" + str(b_size)
                name += "_arc-" + ','.join(map(str, hidden))
                architectures.append( Architecture(
                        learning_rate=lr,
                        batch_size=b_size,
                        loss=nn.BCELoss(),
                        hidden_layers=hidden,
                        name=name
                    )
                )
                count+=1


#training process
for file_name, epochs in zip(file_names, num_epochs):
    dataAtts = DataAtts(file_name)
    database = DataSet (csv_file=file_name, root_dir=".", shuffle_db=False)

    for arc in architectures:
        if ("escalonated" in file_name):
            esc = torch.nn.Sigmoid()
        else:
            esc = False

        generatorAtts = {
            'out_features':dataAtts.class_len,
            'leakyRelu':0.2,
            'hidden_layers':arc.hidden_layers,
            'in_features':100,
            'escalonate':esc
        }
        generator = GeneratorNet(**generatorAtts)

        discriminatorAtts = {
            'in_features':dataAtts.class_len,
            'leakyRelu':0.2,
            'dropout':0.3,
            'hidden_layers':arc.hidden_layers[::-1]

        }
        discriminator = DiscriminatorNet(**discriminatorAtts)

        if torch.cuda.is_available():
            discriminator.cuda()
            generator.cuda()
        d_optimizer = optim.Adam(discriminator.parameters(), lr=arc.learning_rate)
        g_optimizer = optim.Adam(generator.parameters(), lr=arc.learning_rate)
        loss = arc.loss
        data_loader = torch.utils.data.DataLoader(database, batch_size=arc.batch_size, shuffle=True)
        num_batches = len(data_loader)

        print(dataAtts.fname)
        print(arc.name)

        d_error_plt = []
        g_error_plt = []

        for epoch in range(epochs):
            if (epoch % 100 == 0):
                print("Epoch ", epoch)

            for n_batch, real_batch in enumerate(data_loader):
                # 1. Train DdataAtts.fnameiscriminator
                real_data = Variable(real_batch).float()
                if torch.cuda.is_available():
                    real_data = real_data.cuda()
                # Generate fake data
                fake_data = generator(random_noise(real_data.size(0))).detach()
                # Train D
                d_error, d_pred_real, d_pred_fake = train_discriminator(d_optimizer, discriminator, loss, real_data, fake_data)

                # 2. Train Generator
                # Generate fake data
                fake_data = generator(random_noise(real_batch.size(0)))
                # Train G
                g_error = train_generator(g_optimizer, discriminator, loss, fake_data)

            d_error_plt.append(d_error)
            g_error_plt.append(g_error)

        # Display Plots
        fig = plt.figure(figsize=(20, 2))
        ax = fig.add_subplot(111)
        ax.plot(d_error_plt, 'b')
        ax.plot(g_error_plt, 'r')
        filename = "results/" + dataAtts.fname + "/" + arc.name +"_"+"error_growth.txt"
        file = open(filename, "w")
        file.write("Discriminator error: " + str(d_error_plt) + "\n")
        file.write("\n\n\n")
        file.write("Generator error: " + str(g_error_plt) + "\n")
        file.close()

        plt.savefig('images/'+ dataAtts.fname + "/"+ arc.name +"_"+'error.png')
        plt.show()
        plt.clf()


        # From this line on it's just the saving
        # save_model("generator", epoch, generatorAtts, generator.state_dict(), g_optimizer.state_dict(), loss, dataAtts.fname, arc.name)
        # save_model("discriminator", epoch, discriminatorAtts, discriminator.state_dict(), d_optimizer.state_dict(), loss, dataAtts.fname, arc.r

        torch.save({
            'epoch': epoch,
            'model_attributes': generatorAtts,
            'model_state_dict': generator.state_dict(),
            'optimizer_state_dict': g_optimizer.state_dict(),
            'loss': loss
            }, "models/" + dataAtts.fname + "/generator_" + arc.name + ".pt")

        torch.save({
            'epoch': epoch,
            'model_attributes': discriminatorAtts,
            'model_state_dict': discriminator.state_dict(),
            'optimizer_state_dict': d_optimizer.state_dict(),
            'loss': loss
        }, "models/" + dataAtts.fname + "/discriminator_" + arc.name + ".pt")
```

```
diabetes_escalonated
id-0_epochs-500_layer-2_lr-0.0002_batch-5_arc-256,512
Epoch  0
Epoch  100
Epoch  200
Epoch  300
Epoch  400
```



```
diabetes_escalonated
id-1_epochs-500_layer-2_lr-0.0002_batch-5_arc-256,512
Epoch  0
Epoch  100
Epoch  200
Epoch  300
Epoch  400
<Figure size 432x288 with 0 Axes>
```



```
#3 Creating Fake Data
original_db_name = folder.value[7:]
original_db_path = "original_data/" + original_db_name + ".csv"
original_db = pd.read_csv(original_db_path)
original_db_size=original_db.shape[0]

try:
    checkpoint= torch.load(model_widget.value, map_location='cuda')
except:
    checkpoint= torch.load(model_widget.value, map_location='cpu')
print(model_widget.value)
checkpoint['model_attributes']['out_features'] = len(original_db.columns)
generator = GeneratorNet(**checkpoint['model_attributes'])
generator.load_state_dict(checkpoint['model_state_dict'])
```

```
models/diabetes_escalonated/generator_id-9_epochs-500_layer-1_lr-0.0002_batch-5_arc-256.pt
<All keys matched successfully>
```

```
size = original_db_size
new_data = generator.create_data(size)
df = pd.DataFrame(new_data, columns=original_db.columns)
#Changes the name to be easier to read
name = model_widget.value.split("/")[-1][10:-4] + "_size-" + str(size)
df.to_csv( "fake_data/" + original_db_name + "/" + name + ".csv", index=False)
```

```
#4 Classification using CART
# Picked Run ID-6 with Big layer(256) as the best accuracy

file_name=files_dropdown.value
dataAtts = DataAtts(file_name)
data = pd.read_csv(file_name)

fake_data = pd.read_csv(fake_files_dropdown.value)
#Makes the outcome be 0 or 1
fake_data.loc[getattr(fake_data, dataAtts.class_name) >= 0.5, dataAtts.class_name] = 1
fake_data.loc[getattr(fake_data, dataAtts.class_name) < 0.5, dataAtts.class_name] = 0

data.head()
```

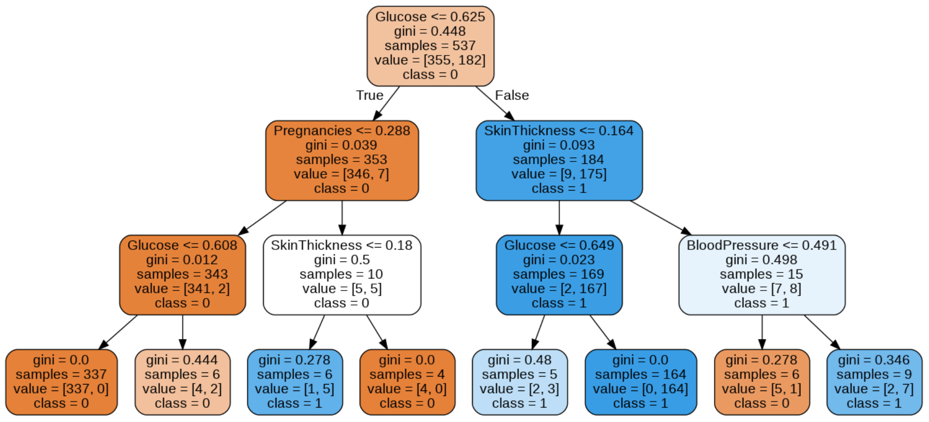
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.234415	0.483333	1
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.116567	0.166667	0
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.253629	0.183333	1
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.038002	0.000000	0
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.943638	0.200000	1

```
fake_data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.138297	0.626414	0.510178	0.236434	0.146556	0.524160	0.186706	0.092767	1.0
1	0.001360	0.471387	0.576830	0.210947	0.004377	0.483142	0.170741	0.005094	0.0
2	0.149213	0.554525	0.544073	0.270423	0.097864	0.466329	0.168845	0.109385	0.0
3	0.097698	0.402009	0.544234	0.300452	0.011518	0.460676	0.163571	0.079259	0.0
4	0.005359	0.471368	0.602342	0.223484	0.002330	0.421150	0.223785	0.015835	0.0

```
original_data_training_set = data.head(int(data.shape[0]*0.7))
fake_data_training_set  = fake_data.head(int(fake_data.shape[0]*0.7))
original_data_testing_set  = data.tail(int(data.shape[0]*0.3))
fake_data_testing_set  = fake_data.tail(int(fake_data.shape[0]*0.3))
mixed_data_training_set=pd.concat([original_data_training_set, fake_data_training_set])
mixed data testing set=pd.concat([original data testing set, fake data testing set])
```

```
mask_0 = original_data_testing_set[dataAtts.class_name] == 0
mask_1 = original_data_testing_set[dataAtts.class_name] == 1
original_1s = original_data_testing_set[mask_1]
head_0s = original_data_testing_set[mask_0].head(original_1s.shape[0])
tail_0s = original_data_testing_set[mask_0].tail(original_1s.shape[0])
sampeld_0s = original_data_testing_set[mask_0].sample(original_1s.shape[0])
balanced_test = pd.concat([original_1s, sampeld_0s])

train = fake_data_training_set
test = original_data_testing_set

trainX = train.drop(dataAtts.class_name, 1)
testX = test.drop(dataAtts.class_name, 1)
y_train = train[dataAtts.class_name]
y_test = test[dataAtts.class_name]

clf1 = DT(max_depth = 3, min_samples_leaf = 1)
clf1 = clf1.fit(trainX,y_train)

export_graphviz(clf1, out_file="models/tree.dot", feature_names=trainX.columns, class_names=["0","1"], filled=True, rounded=True)
g = pydotplus.graph_from_dot_file(path="models/tree.dot")
Image(g.create_png())
```



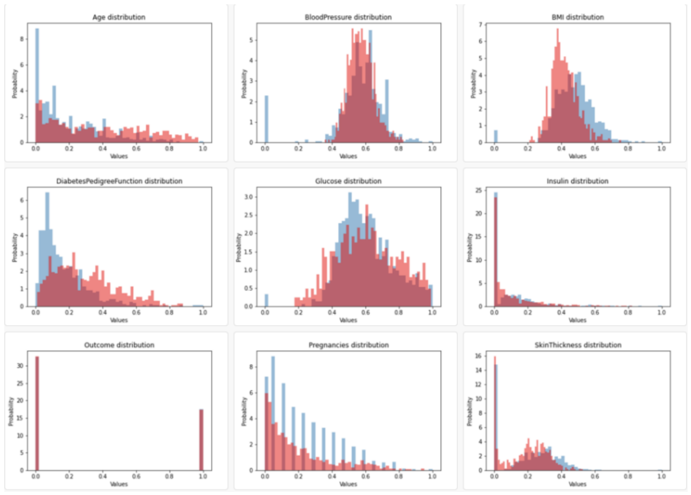
```
pred = clf1.predict_proba(testX)

# If there is only one possible class in the training_data the tree will predict all results to be 1
# The lines below are a little hack to avoid this problem
if pred.shape[1] > 1:
    pred = np.argmax(pred, axis=1)
else:
    pred = pred.reshape((pred.shape[0]))
    if negative=="0":
        pred = pred-1

mse = ((pred - y_test.values)**2).mean(axis=0)
mse
```

```
0.2956521739130435
```

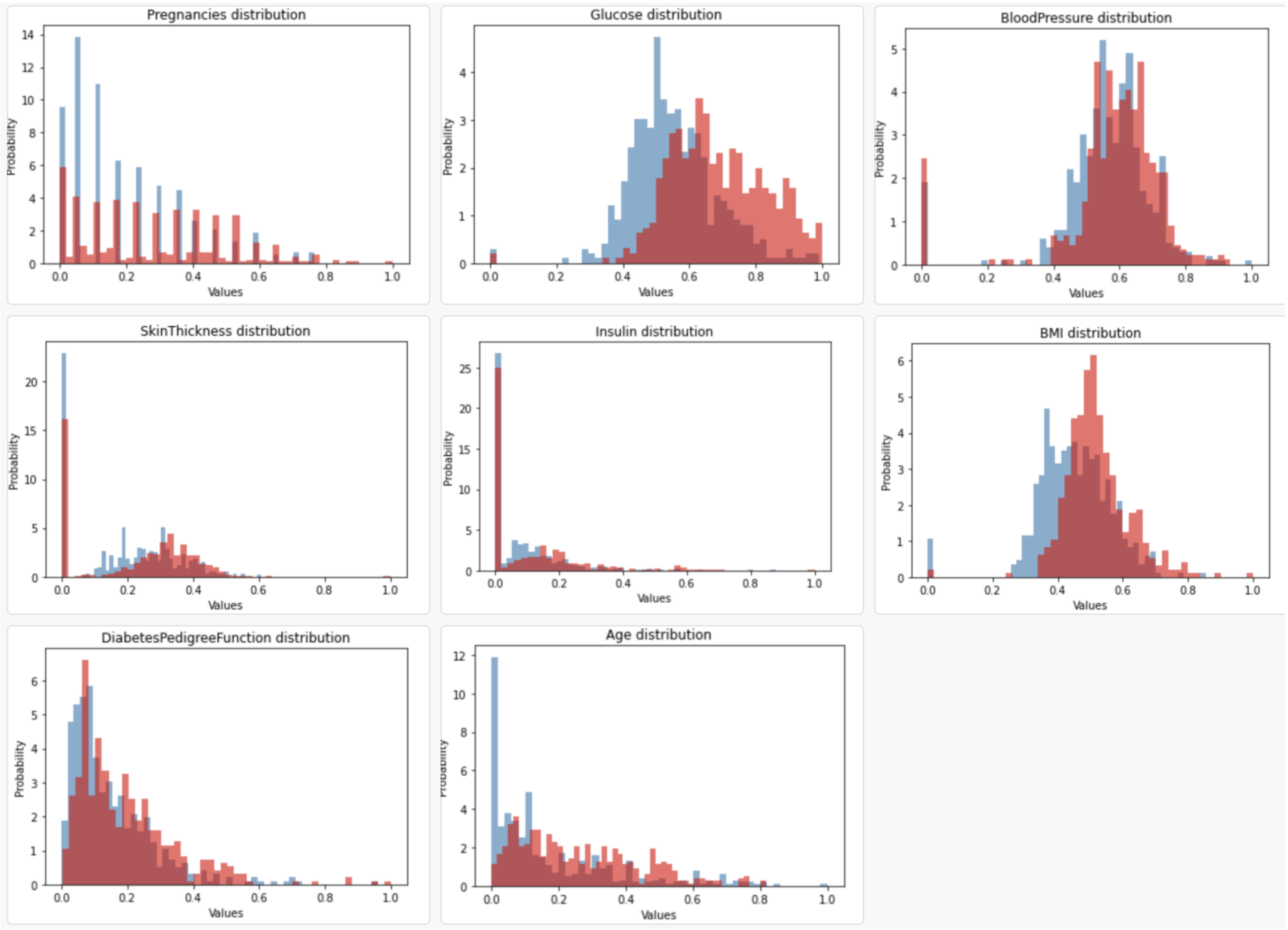
```
conf_matrix = confusion_matrix(y_test.values, pred)
TN, FN, TP, FP = conf_matrix[0][0], conf_matrix[1][0], conf_matrix[1][1], conf_matrix[0][1]
confusion_matrix_str = str(TN) + "/" + str(FN) + "/" + str(TP) + "/" + str(FP)
precision = round(TP/(TP+FP), 3)
recall = round(TP/(TP+FN), 3)
accuracy = round((TP+TN)/(TP+TN+FP+FN), 3)
f1_score = round(2*(precision*recall)/(precision+recall),3)
print("TN/FN/TP/FP: ", confusion_matrix_str)
print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("F-1 score: ", f1_score)
```

```
TN/FN/TP/FP:  105/22/57/46
Accuracy:  0.704
Precision:  0.553
Recall:  0.722
F-1 score:  0.626
```

```
#5 Fake Data Analysis
# Selected Model Parameters with the highest accuracy (Best Setting included data analysis of that model,id-3, lr-0.002, Accuracy- 79.1%)


classes = list(data)

for name in classes:
    if name=="Unnamed: 32":
        continue

    plt.xlabel('Values')
    plt.ylabel('Probability')
    plt.title(name + " distribution")
    real_dist = data[name].values
    fake_dist = fake_data[name].values
    plt.hist(real_dist, 50, density=True, alpha=0.5)
    plt.hist(fake_dist, 50, density=True, alpha=0.5, facecolor='r')
    plt.show()
    plt.clf()
```



```
# ADASYN (Oversampling minority classes)
data = pd.read_csv('original_data/diabetes_escalonated.csv')
print(data)
ada = ADASYN()
new_X, new_y = ada.fit_resample(data.iloc[:,:-1].values,data['Outcome'])
new_y = pd.DataFrame(new_y)
new_y.columns = ['Outcome']
new_data = pd.DataFrame({'Pregnancies': new_X[:, 0],'Glucose': new_X[:, 1] ,'BloodPressure': new_X[:, 2], 'SkinThickness': new_X[:, 3], 'Insulin'
new_data = new_data.join(new_y)
```


Pregnancies distribution
Glucose distribution
BloodPressure distribution
SkinThickness distribution
Insulin distribution
BMI distribution
DiabetesPedigreeFunction distribution
Age distribution
Probability
Values