Created on Mon Jun 12 13:04:36 2023

@author: btb5103

Questions Part 1: Data Exploration and Preprocessing

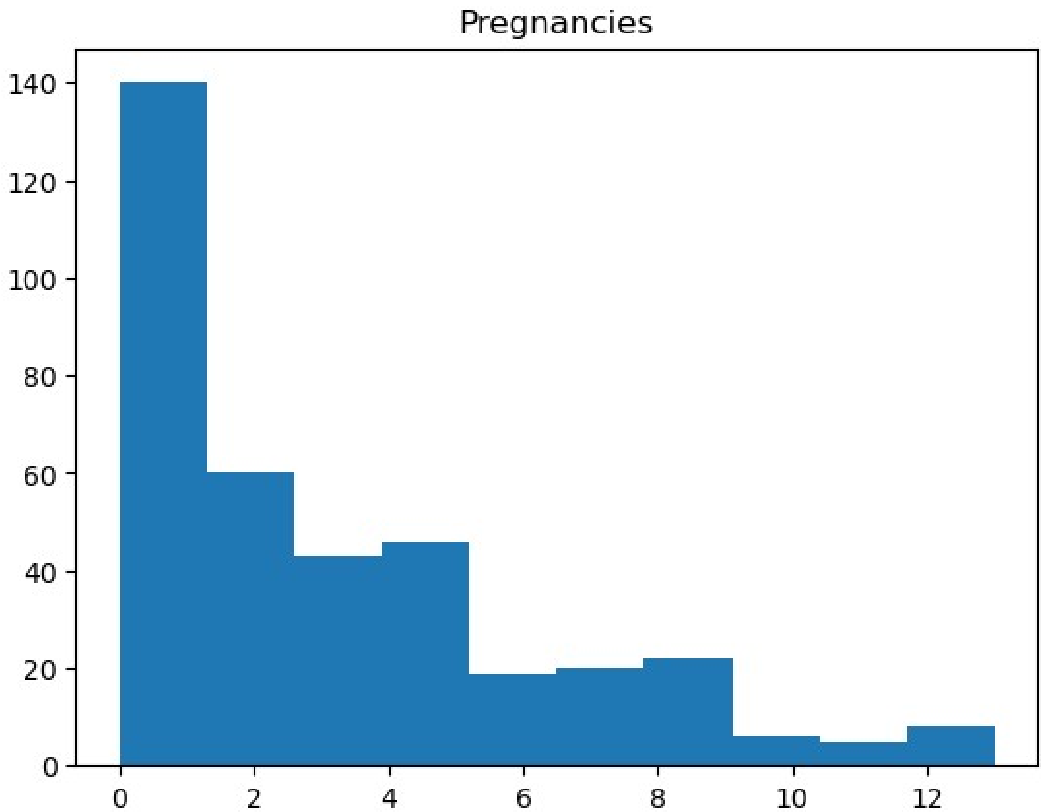
```
Read and load data into Python
Explore and pre-process the dataset. For examples;
    Handle Missing values
    Check Duplicate values
    Outliers detection
    Check correlation
    Check imbalanced data
    Scale or Normalize data
    Plots: Histograms, Boxplots, pairplot, etc.
```

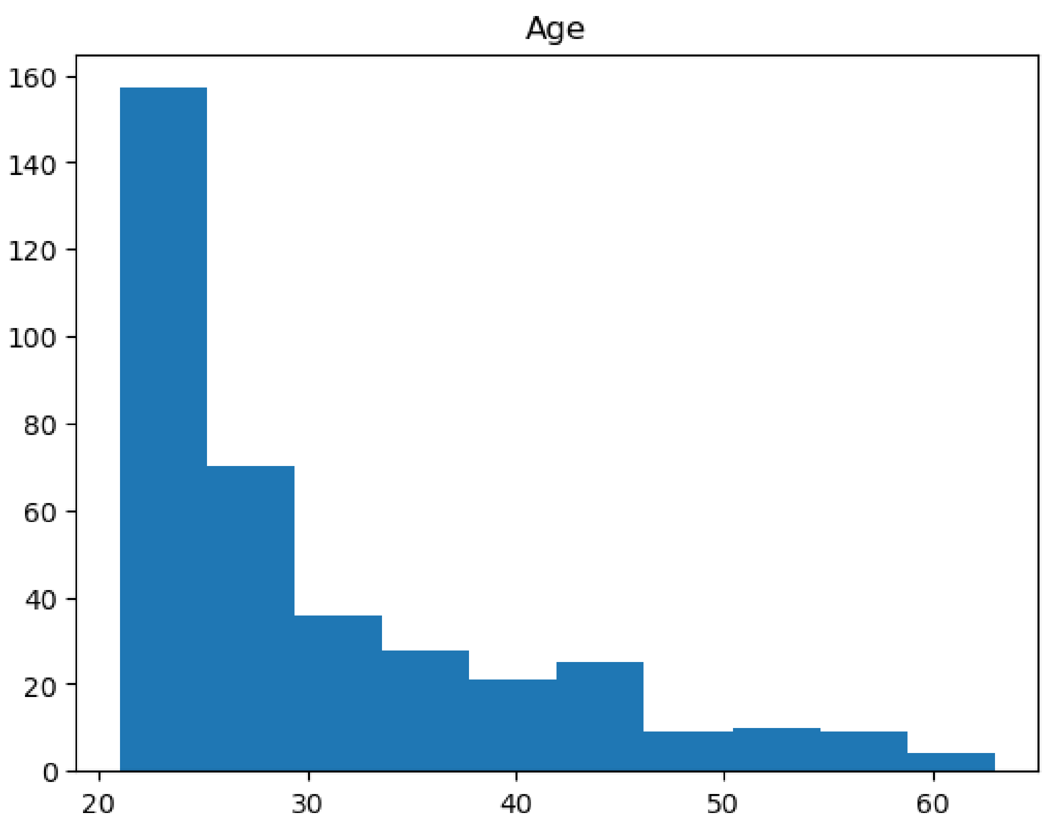
Part 1 - Preprocessing

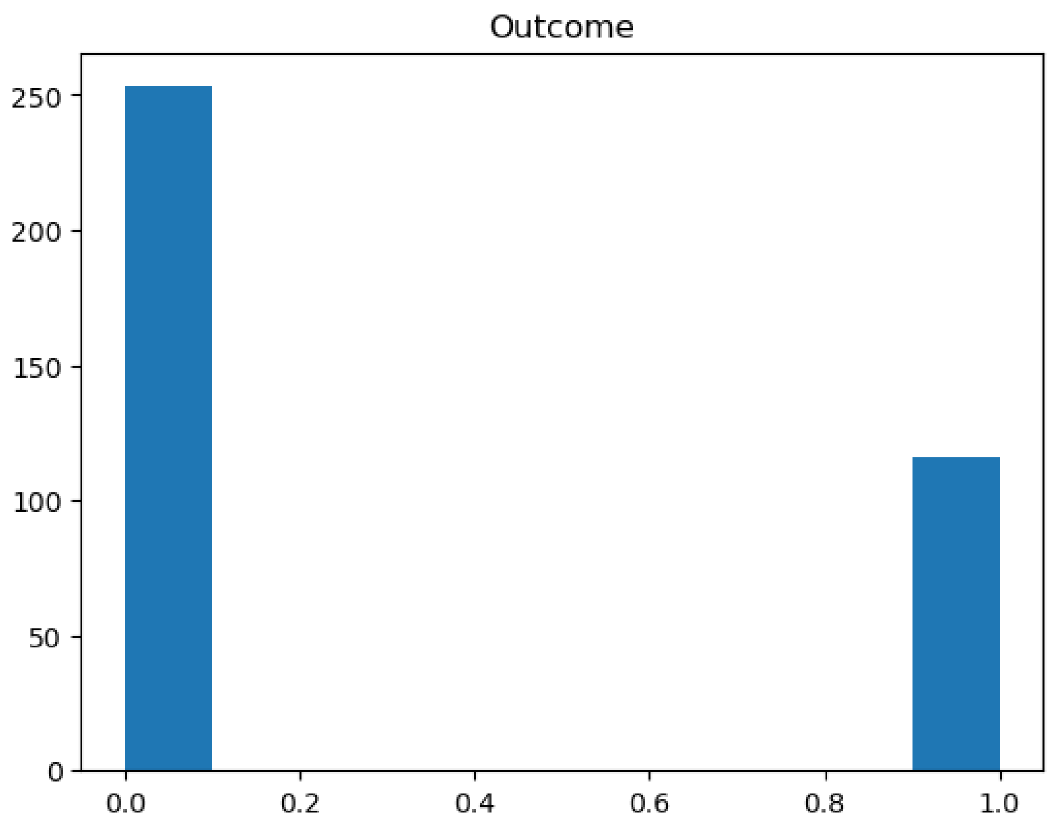
```
#PART 1:

#from dl import tensorflow as dl

import pandas as pd
import numpy as np
import requests
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler


#I don't think the Pima Indians Diabetes dataset is hosted at the
#UCI ML Repository anymore.
#I was able to pull it from Kaggle and use it locally
url = 'https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-
database/download?datasetVersionNumber=1'

file = 'C:\\Users\\btb51\\Documents\\GitHub\\DeepLearning_DAAN570\\
DAAN570_Instructor_Sample_Codes\\Lesson_06_Code\\archive\\
diabetes.csv'

data = pd.read_csv(file)

#Turn missing values to NANs with the exception of pregnacies
data["BloodPressure"].replace(to_replace=0, value=np.NAN,
inplace=True)
data["SkinThickness"].replace(to_replace=0, value=np.NAN,
inplace=True)
data["Insulin"].replace(to_replace=0, value=np.NAN, inplace=True)
```

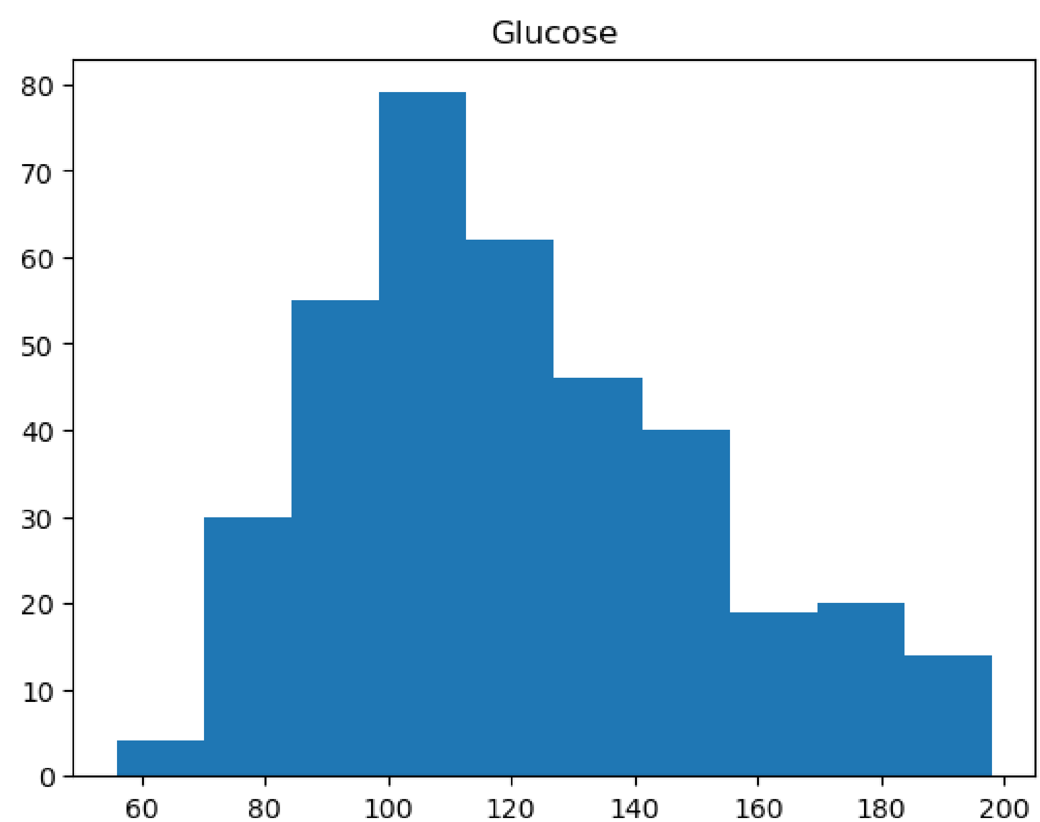
```
#It may be beneficial to_replace with the average of the column if the zeros
#push values

#drop the duplicates keeping the first instance of any dups
data = data.drop_duplicates(keep='first')

#Check for outliers (keep anything where all data cols are within 3 std dev)
data = data[(np.abs(stats.zscore(data, nan_policy='omit')) < 3).all(axis=1)]

#Make some quick plots to see if there are any possible imbalances

#It looks like zeros dominate here
fig1, ax1 = plt.subplots()
ax1.hist(data["Pregnancies"])
ax1.set_title("Pregnancies")

#It looks like the age of 20-30 dominates here
fig2, ax2 = plt.subplots()
ax2.hist(data['Age'], bins = 10)
ax2.set_title("Age")

#The outcome looks fair enough, I don't think I need to work on imbalances
fig3, ax3 = plt.subplots()
ax3.hist(data['Outcome'])
ax3.set_title("Outcome")

fig4, ax4 = plt.subplots()
ax4.hist(data['Glucose'])
ax4.set_title('Glucose')
```

```
Text(0.5, 1.0, 'Glucose')
```


Pregnancies
0
20
40
60
80
100
120
140
0
2
4
6
8
10
12


Age
0
20
40
60
80
100
120
140
160
20
30
40
50
60


Outcome
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0

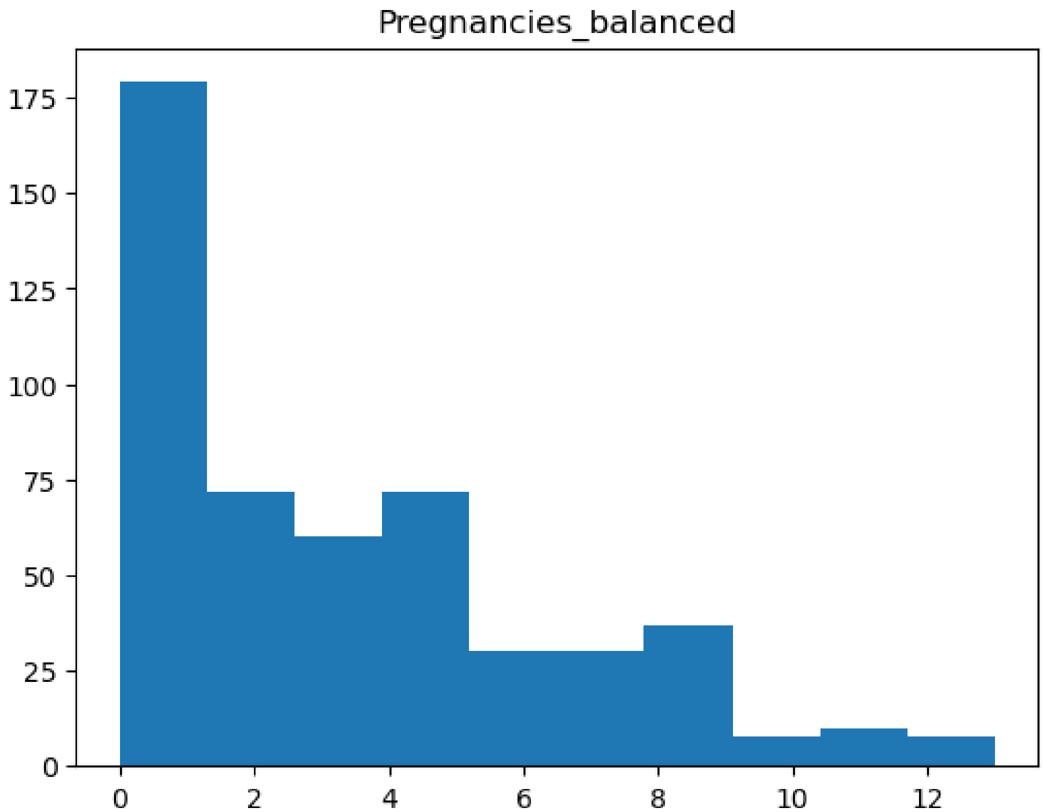

Glucose
80
70
60
50
40
30
20
10
0
60
80
100
120
140
160
180
200

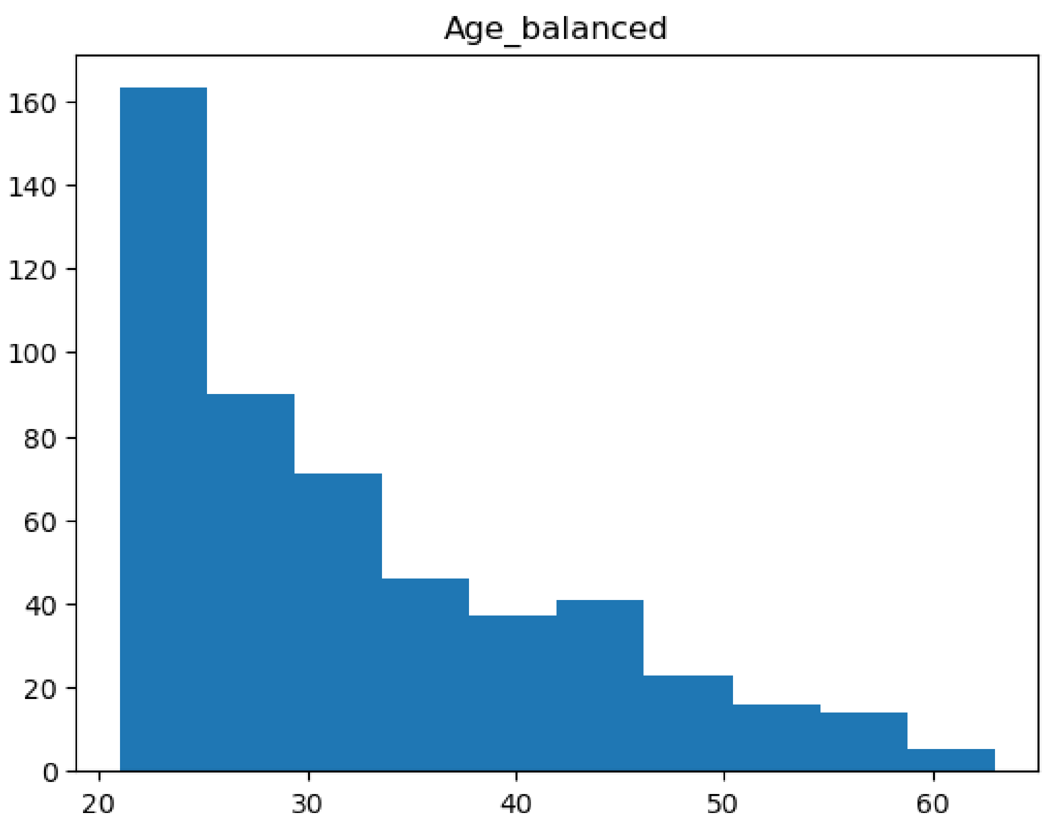
```python
#Deal with the class imbalance
from imblearn.over_sampling import SMOTE

#splice data
y = data.iloc[:, 8]
x = data.iloc[:,:8]

#make the SMOTE object
oversample = SMOTE()

#Restore balance
x, y = oversample.fit_resample(x,y)

#Check the balance
print("Length of x: " + str(len(x)))
print("Length of y: " + str(len(y)))
```

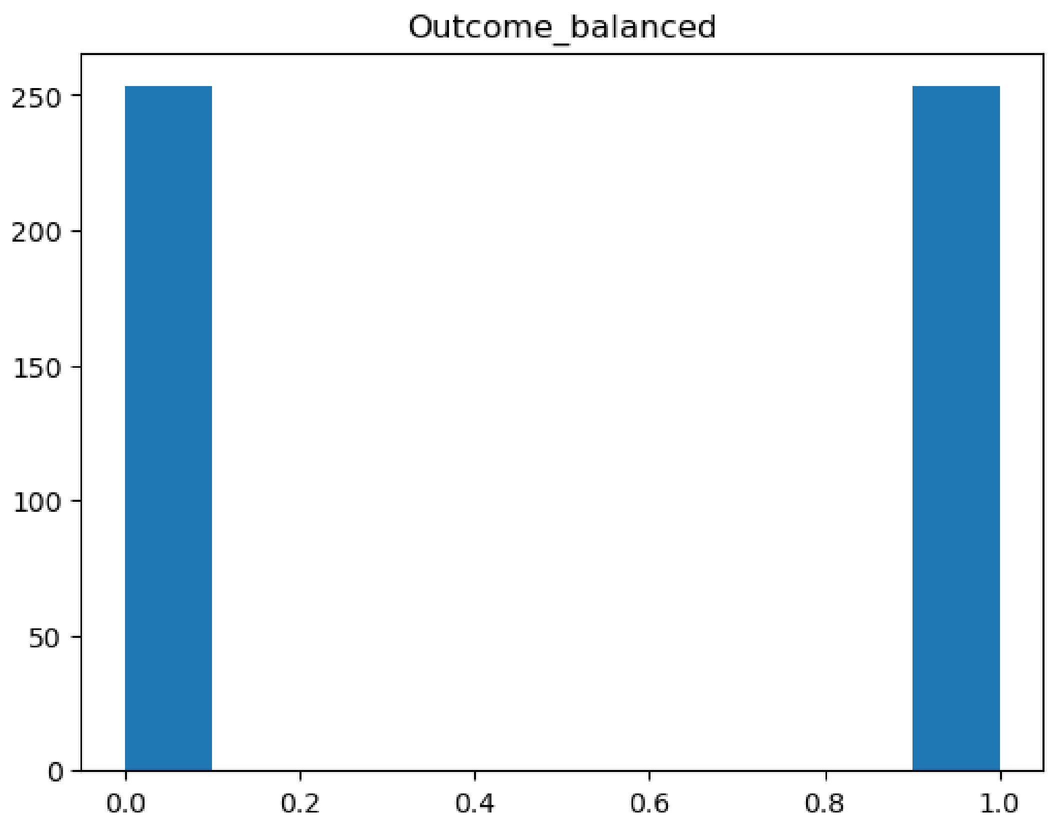
```
Length of x: 506
Length of y: 506
```

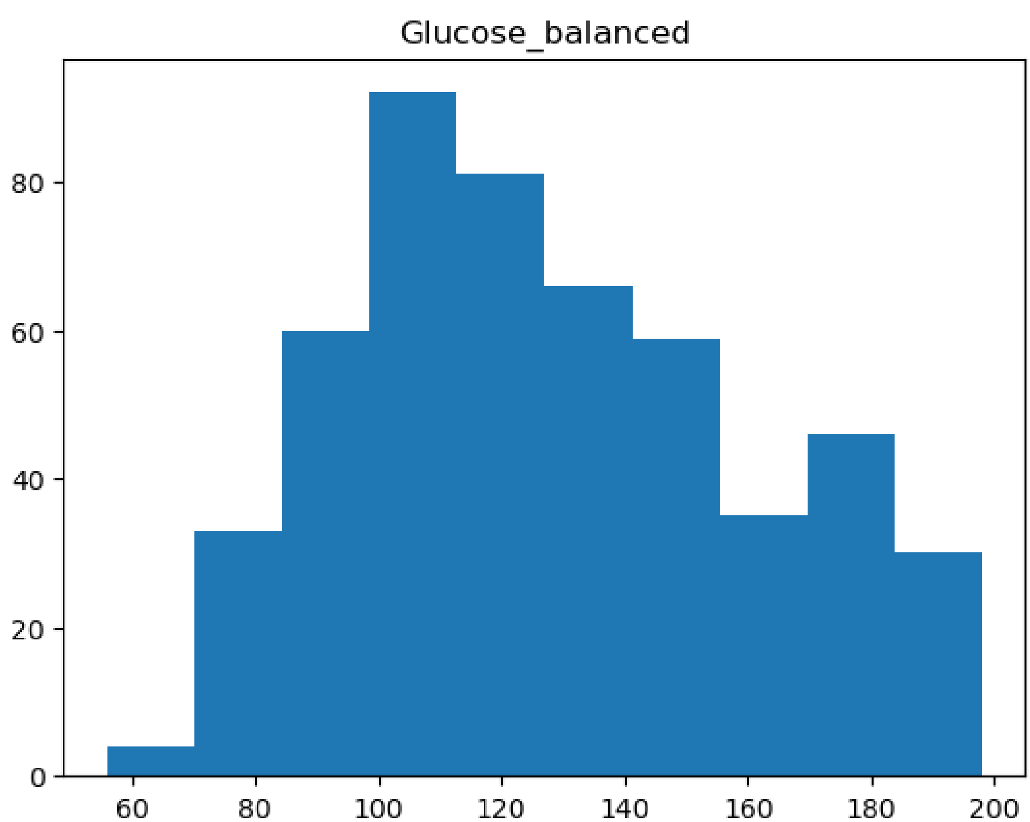
```python
#Check balances again...

#It looks like zeros dominate here
fig1, ax1 = plt.subplots()
ax1.hist([x["Pregnancies"]])
ax1.set_title("Pregnancies_balanced")

#It looks like the age of 20-30 dominates here
fig2, ax2 = plt.subplots()
ax2.hist(x['Age'], bins = 10)
ax2.set_title("Age_balanced")

#The outcome looks fair enough, I don't think I need to work on
imbalances
fig3, ax3 = plt.subplots()
ax3.hist(y)
ax3.set_title("Outcome_balanced")

fig4, ax4 = plt.subplots()
ax4.hist(x['Glucose'])
ax4.set_title('Glucose_balanced')
```

```
Text(0.5, 1.0, 'Glucose_balanced')
```


Pregnancies_balanced
175
150
125
100
75
50
25
0
0
2
4
6
8
10
12

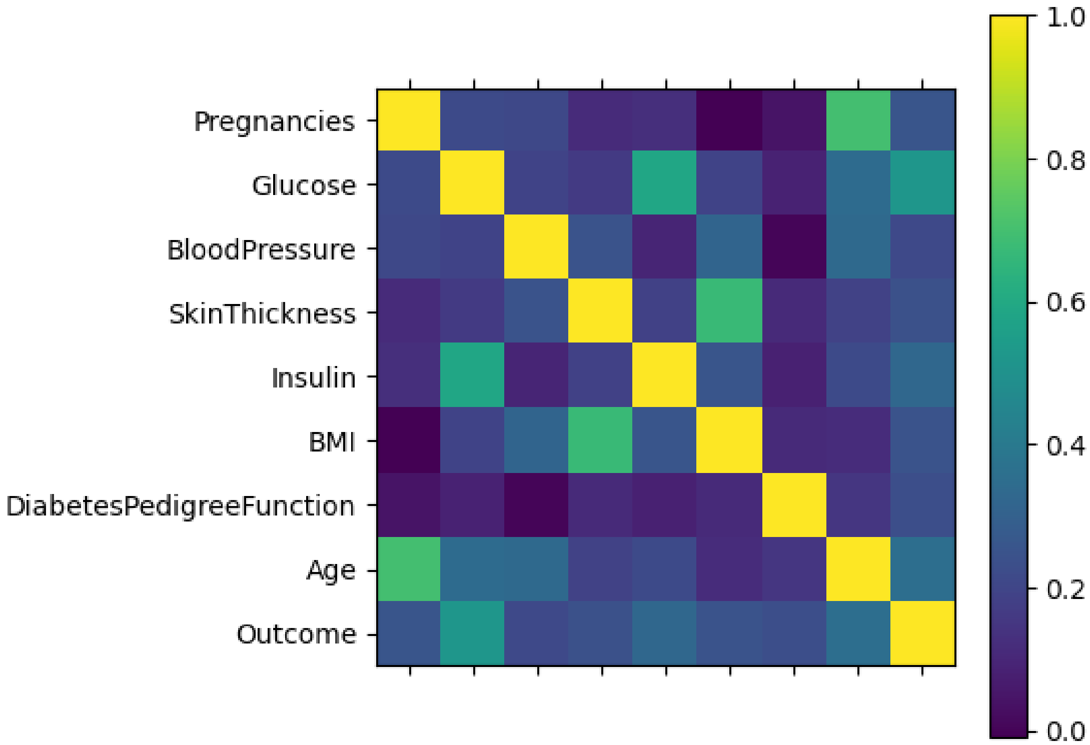

Age_balanced
160
140
120
100
80
60
40
20
0
20
30
40
50
60

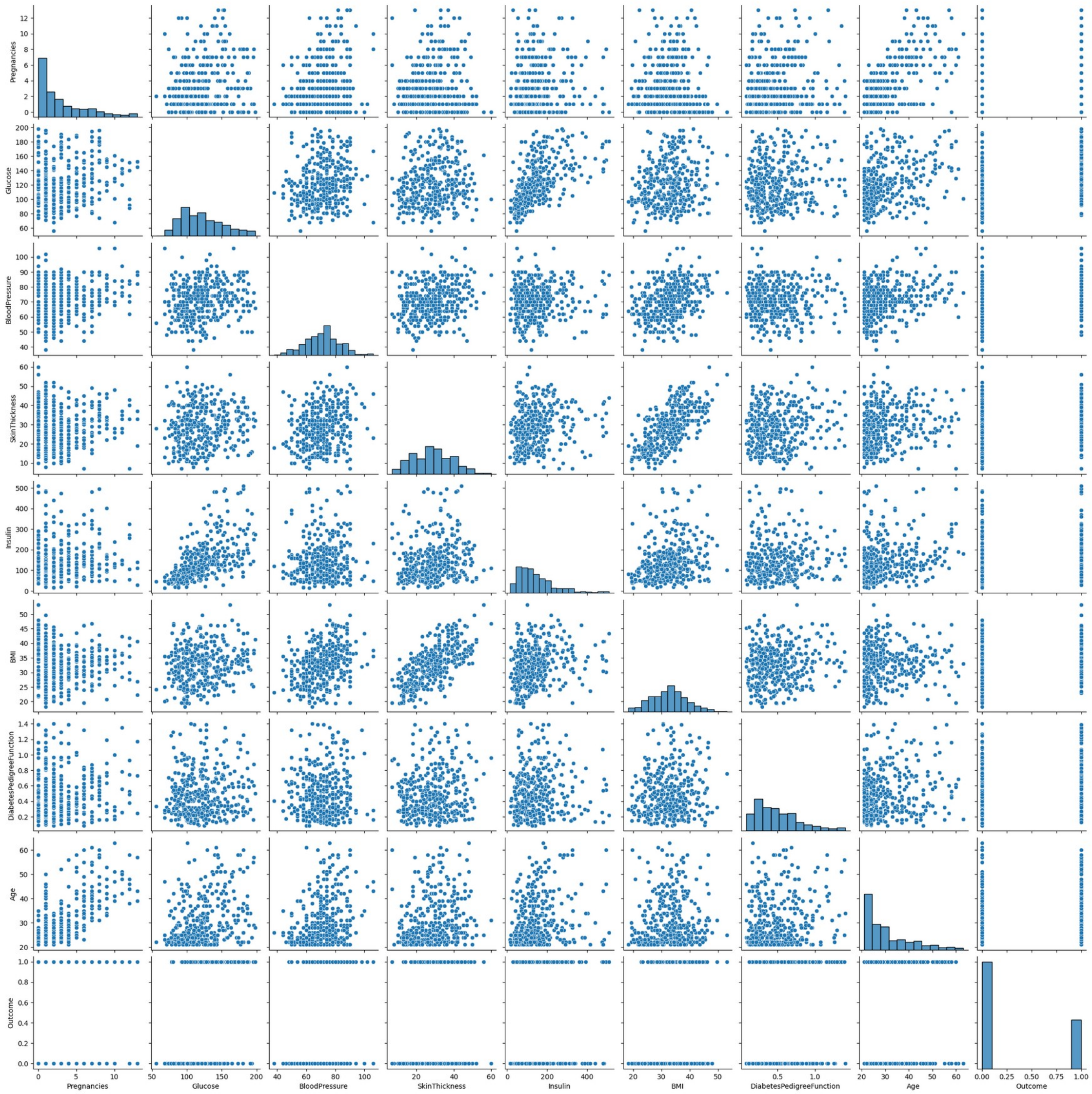

Outcome_balanced
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0


Glucose_balanced
80
60
40
20
0
60
80
100
120
140
160
180
200

```
#Check correlation
cor = data.corr()

# plotting
plt.matshow(cor)
plt.colorbar()
plt.xticks(ticks=range(9),labels="")
plt.yticks(ticks=range(9),labels = data.columns)

#Pairplot
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x131abc05700>
```






```
#Scale/Normalize the data

#use minmaxscaler
scaler = MinMaxScaler()


data_x = scaler.fit_transform(x)
```

Part 2: Baseline Model

#PART 2: Part 2: Build a Baseline Model

Use the Sequential model to quickly build a baseline neural network with one single hidden layer with 12 nodes.

Split the data to training and testing dataset (75%, 25%)
Build the baseline model and find how many parameters does your model have?
Train you model with 20 epochs with RMSProp at a learning rate of .001 and a batch size of 128
Graph the trajectory of the loss functions, accuracy on both train and test set.
Evaluate and interpret the accuracy and loss performance during training, and testing.

```
#Model imports
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.utils import plot_model
from keras.layers import concatenate
from keras.metrics import AUC

from sklearn.model_selection import train_test_split

#train test split
X_train, X_test, y_train, y_test = train_test_split(data_x, y,
test_size=0.25,
                                                    random_state=570)

# Model Generation

def build_model():

    #single hidden layer of 12 nodes
    model_input = Input(shape=(8,), name='data_in')
    hidden_layer_1 = Dense(units=12, activation='relu', name='HL_1')
(model_input)
    model_out = Dense(1, activation='sigmoid', name='data_out')
(hidden_layer_1)

    #create the model by linking inputs and outputs through Keras
functional API
    model = Model(inputs=model_input, outputs=model_out,
name='Diabetes')

    return model

#%%
# Plotting function to use repeatedly

def quick_plot(values, keys,title = "You Need A Title"):

    # Plot loss function of the training
    fig, axs = plt.subplots(2,2)
    fig.suptitle(title)
```

```
    fig.tight_layout()

    axs[0,0].plot(values.history[keys[0]])
    axs[0,0].set_ylabel('loss')
    axs[0,0].set_xlabel('epoch')

    axs[0,1].plot(values.history[keys[1]])
    axs[0,1].set_ylabel(keys[1])
    axs[0,1].set_xlabel('epoch')

    axs[1,0].plot(values.history[keys[2]])
    axs[1,0].set_ylabel(keys[2])
    axs[1,0].set_xlabel('epoch')

    axs[1,1].plot(values.history[keys[3]])
    axs[1,1].set_ylabel(keys[3])
    axs[1,1].set_xlabel('epoch')

##Compile the model
model = build_model()

#USING RMSProp
optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)

bi_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)

metric = [tf.keras.metrics.BinaryAccuracy(),
          tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.AUC(curve='ROC')]

model.compile(optimizer=optimizer, loss=bi_loss, metrics=metric)

# Train the model

values = model.fit(X_train, y_train, batch_size=128, epochs=20,
verbose=1)


#Evaluate the model
loss, accuracy, false_pos, auc = model.evaluate(X_test, y_test)
```

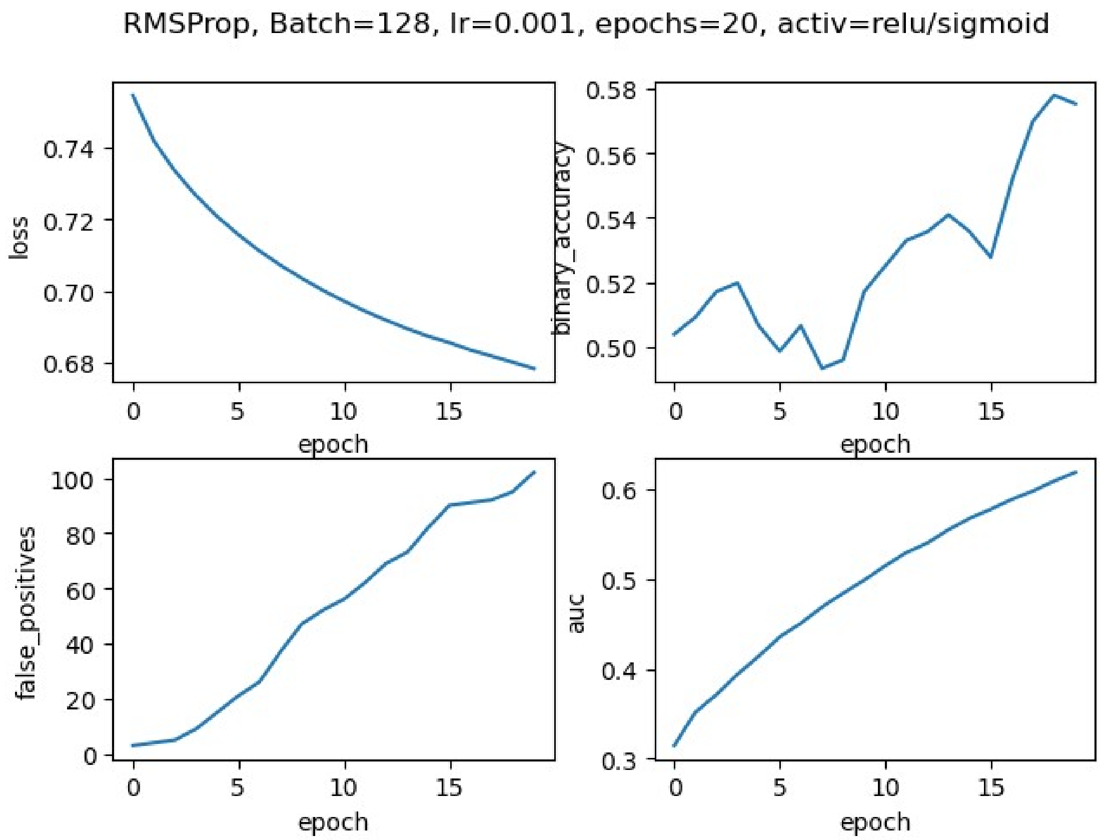
```
Epoch 1/20

C:\Users\btb51\anaconda3\envs\tf_LAtest\lib\site-packages\keras\
backend.py:5673: UserWarning: "`binary_crossentropy` received
`from_logits=True`, but the `output` argument was produced by a
Sigmoid activation and thus does not represent logits. Was this
intended?
  output, from_logits = _get_logits(
```

```
3/3 [==============================] - 1s 11ms/step - loss: 0.7545 -
binary_accuracy: 0.5040 - false_positives: 3.0000 - auc: 0.3145
Epoch 2/20
3/3 [==============================] - 0s 9ms/step - loss: 0.7418 -
binary_accuracy: 0.5092 - false_positives: 4.0000 - auc: 0.3514
Epoch 3/20
3/3 [==============================] - 0s 11ms/step - loss: 0.7334 -
binary_accuracy: 0.5172 - false_positives: 5.0000 - auc: 0.3711
Epoch 4/20
3/3 [==============================] - 0s 10ms/step - loss: 0.7265 -
binary_accuracy: 0.5198 - false_positives: 9.0000 - auc: 0.3939
Epoch 5/20
3/3 [==============================] - 0s 9ms/step - loss: 0.7207 -
binary_accuracy: 0.5066 - false_positives: 15.0000 - auc: 0.4138
Epoch 6/20
3/3 [==============================] - 0s 8ms/step - loss: 0.7157 -
binary_accuracy: 0.4987 - false_positives: 21.0000 - auc: 0.4355
Epoch 7/20
3/3 [==============================] - 0s 10ms/step - loss: 0.7111 -
binary_accuracy: 0.5066 - false_positives: 26.0000 - auc: 0.4507
Epoch 8/20
3/3 [==============================] - 0s 10ms/step - loss: 0.7071 -
binary_accuracy: 0.4934 - false_positives: 37.0000 - auc: 0.4686
Epoch 9/20
3/3 [==============================] - 0s 8ms/step - loss: 0.7035 -
binary_accuracy: 0.4960 - false_positives: 47.0000 - auc: 0.4841
Epoch 10/20
3/3 [==============================] - 0s 10ms/step - loss: 0.7002 -
binary_accuracy: 0.5172 - false_positives: 52.0000 - auc: 0.4987
Epoch 11/20
3/3 [==============================] - 0s 10ms/step - loss: 0.6971 -
binary_accuracy: 0.5251 - false_positives: 56.0000 - auc: 0.5148
Epoch 12/20
3/3 [==============================] - 0s 8ms/step - loss: 0.6943 -
binary_accuracy: 0.5330 - false_positives: 62.0000 - auc: 0.5294
Epoch 13/20
3/3 [==============================] - 0s 9ms/step - loss: 0.6918 -
binary_accuracy: 0.5356 - false_positives: 69.0000 - auc: 0.5402
Epoch 14/20
3/3 [==============================] - 0s 9ms/step - loss: 0.6894 -
binary_accuracy: 0.5409 - false_positives: 73.0000 - auc: 0.5549
Epoch 15/20
3/3 [==============================] - 0s 10ms/step - loss: 0.6873 -
binary_accuracy: 0.5356 - false_positives: 82.0000 - auc: 0.5677
Epoch 16/20
3/3 [==============================] - 0s 12ms/step - loss: 0.6856 -
binary_accuracy: 0.5277 - false_positives: 90.0000 - auc: 0.5777
Epoch 17/20
3/3 [==============================] - 0s 11ms/step - loss: 0.6835 -
binary_accuracy: 0.5515 - false_positives: 91.0000 - auc: 0.5887
```

```
Epoch 18/20
3/3 [==============================] - 0s 9ms/step - loss: 0.6818 -
binary_accuracy: 0.5699 - false_positives: 92.0000 - auc: 0.5980
Epoch 19/20
3/3 [==============================] - 0s 10ms/step - loss: 0.6801 -
binary_accuracy: 0.5778 - false_positives: 95.0000 - auc: 0.6090
Epoch 20/20
3/3 [==============================] - 0s 9ms/step - loss: 0.6784 -
binary_accuracy: 0.5752 - false_positives: 102.0000 - auc: 0.6188
4/4 [==============================] - 0s 6ms/step - loss: 0.6601 -
binary_accuracy: 0.5827 - false_positives: 33.0000 - auc: 0.6809
```

```python
# Model Summary statistics and figure
print(model.summary())

plot_model(model, to_file='Diabetes_Model_V1.png')

quick_plot(values, list(values.history.keys()),"RMSProp, Batch=128,
lr=0.001, epochs=20, activ=relu/sigmoid")
```

```
Model: "Diabetes"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 data_in (InputLayer)        [(None, 8)]               0

 HL_1 (Dense)                (None, 12)                108

 data_out (Dense)            (None, 1)                 13

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
None
```



Part 3: Find the Best Model

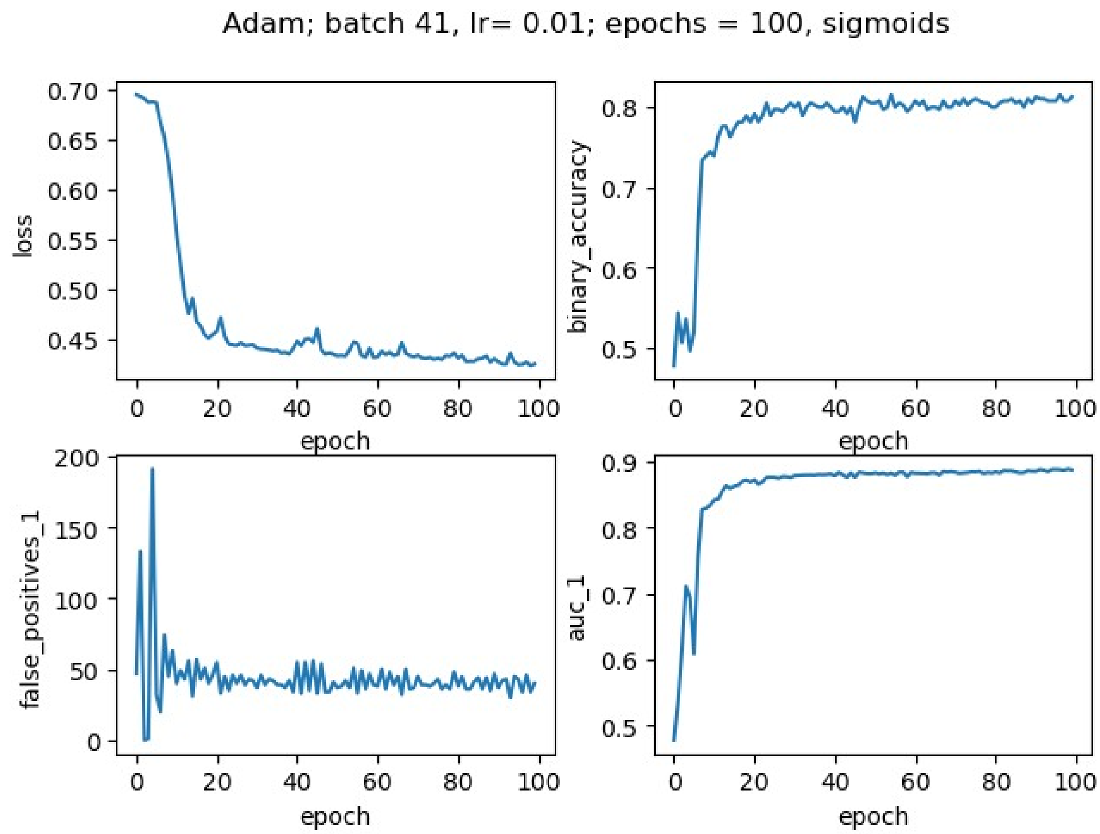
Model v2:

Change to the model architecture as well as
Using Adam as Optimizer; lr is 0.01; batchsize = 41; epochs = 100;
replaced ReLUs with more sigmoids

```
def build_model_v2():

    model_input = Input(shape=(8,), name='data_in')
    hidden_layer_1 = Dense(units=12, activation='sigmoid',
name='HL_1')(model_input)
    hidden_layer_2 = Dense(units=12, activation='sigmoid',
name='HL_2')(hidden_layer_1)
    hidden_layer_3 = Dense(units=12, activation='sigmoid',
name='HL_3')(hidden_layer_2)
    model_out = Dense(1, activation='sigmoid', name='data_out')
(hidden_layer_3)

    model = Model(inputs=model_input, outputs=model_out,
name='Diabetes')

    return model
```

```
##Compile the model
model_v2 = build_model_v2()

#Using ADAM
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)

bi_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)

metric = [tf.keras.metrics.BinaryAccuracy(),
          tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.AUC(curve='ROC')]

model_v2.compile(optimizer=optimizer, loss=bi_loss, metrics=metric)

# Train the model
values_v2 = model_v2.fit(X_train, y_train, batch_size=41, epochs=100,
verbose=1)


#Evaluate the model
loss, accuracy, false_pos, auc = model_v2.evaluate(X_test, y_test)

# Model Summary statistics and figure
print(model_v2.summary())

plot_model(model_v2, to_file='Diabetes_Model_v2.png')

quick_plot(values_v2, list(values_v2.history.keys()),"Adam; batch 41,
lr= 0.01; epochs = 100, sigmoids")
```

```
Model: "Diabetes"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 data_in (InputLayer)        [(None, 8)]               0

 HL_1 (Dense)                (None, 12)                108

 HL_2 (Dense)                (None, 12)                156

 HL_3 (Dense)                (None, 12)                156

 data_out (Dense)            (None, 1)                 13

=================================================================
Total params: 433
Trainable params: 433
Non-trainable params: 0
_________________________________________________________________
None
```



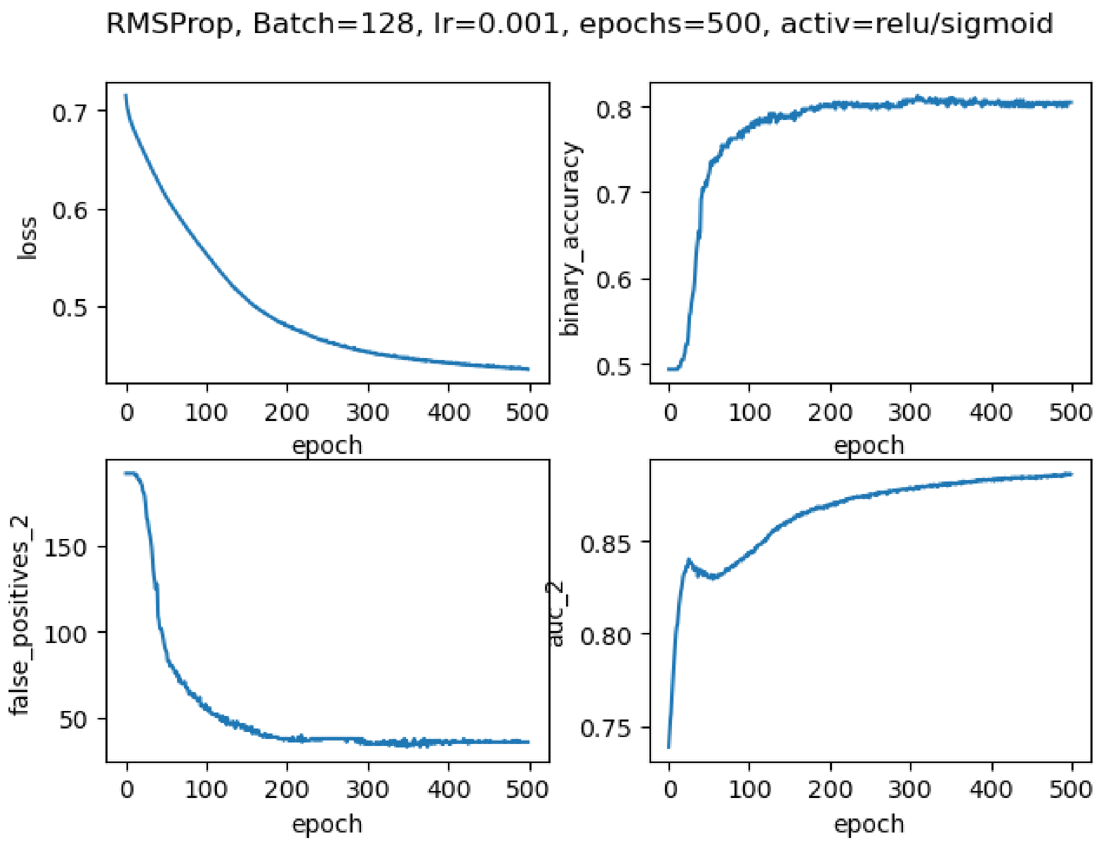
Model v3:

Origional Model architecture
Epochs increased to 500

```
# Model Generation

def build_model_v3():

    model_input = Input(shape=(8,), name='data_in')
    hidden_layer_1 = Dense(units=12, activation='relu', name='HL_1')
(model_input)
    model_out = Dense(1, activation='sigmoid', name='data_out')
(hidden_layer_1)

    model = Model(inputs=model_input, outputs=model_out,
name='Diabetes')

    return model

##Compile the model
model_v3 = build_model()

#USING RMSProp
optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
```

```
bi_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)

metric = [tf.keras.metrics.BinaryAccuracy(),
          tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.AUC(curve='ROC')]

model_v3.compile(optimizer=optimizer, loss=bi_loss, metrics=metric)

# Train the model
values_v3 = model_v3.fit(X_train, y_train, batch_size=128, epochs=500,
verbose=1)


#Evaluate the model
loss, accuracy, false_pos, auc = model_v3.evaluate(X_test, y_test)

# Model Summary statistics and figure
print(model_v3.summary())

plot_model(model_v3, to_file='Diabetes_Model_V3.png')

quick_plot(values_v3, list(values_v3.history.keys()),"RMSProp,
Batch=128, lr=0.001, epochs=500, activ=relu/sigmoid")
```

```
Model: "Diabetes"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 data_in (InputLayer)        [(None, 8)]               0

 HL_1 (Dense)                (None, 12)                108

 data_out (Dense)            (None, 1)                 13

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
None
```



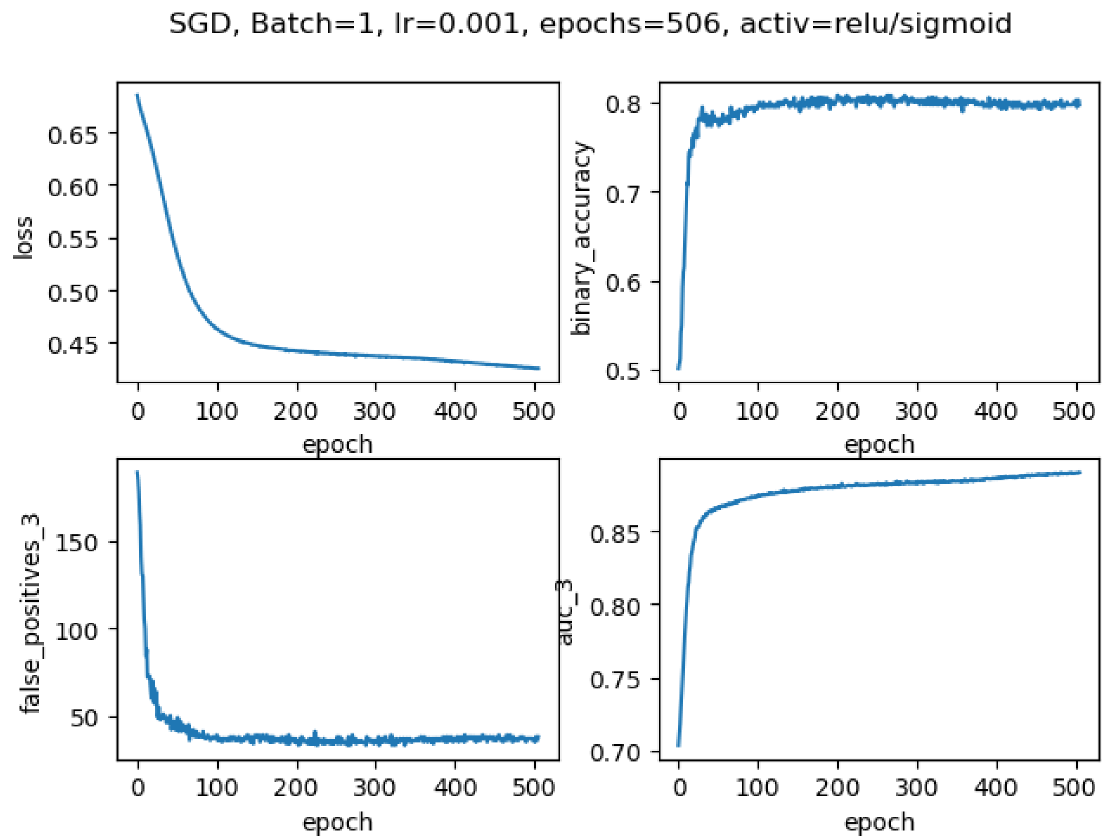
Model v4

```
Origional Architecture
Optimizer=SGD; Batch = 1 (true SGD)
```

```
# Model Generation

def build_model_v4():

    model_input = Input(shape=(8,), name='data_in')
    hidden_layer_1 = Dense(units=12, activation='relu', name='HL_1')
(model_input)
    model_out = Dense(1, activation='sigmoid', name='data_out')
(hidden_layer_1)

    model = Model(inputs=model_input, outputs=model_out,
name='Diabetes')

    return model

##Compile the model
model_v4 = build_model()

#USING RMSProp
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
```

```python
bi_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)

metric = [tf.keras.metrics.BinaryAccuracy(),
          tf.keras.metrics.FalsePositives(),
          tf.keras.metrics.AUC(curve='ROC')]

model_v4.compile(optimizer=optimizer, loss=bi_loss, metrics=metric)

# Train the model
values_v4 = model_v4.fit(X_train, y_train, batch_size=1, epochs=506,
verbose=1)


#Evaluate the model
loss, accuracy, false_pos, auc = model_v4.evaluate(X_test, y_test)

# Model Summary statistics and figure
print(model_v4.summary())

plot_model(model_v4, to_file='Diabetes_Model_V4.png')

quick_plot(values_v4, list(values_v4.history.keys()),"SGD, Batch=1,
lr=0.001, epochs=506, activ=relu/sigmoid")


#Model 4 runs SLOWLY!  ~1 to 2 seconds per epoch
```

```
Model: "Diabetes"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 data_in (InputLayer)        [(None, 8)]               0

 HL_1 (Dense)                (None, 12)                108

 data_out (Dense)            (None, 1)                 13

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
None
```



Takeaways from the four model runs

Which model has best performance, why? Save your best model weights into a binary file.

Submit two files: the Jupyter notebook with your code and answers and its print out PDF.

Last display from each model:

Model 1: Epoch 20/20 3/3 [==============================] - 0s 9ms/step - loss: 0.6784 - binary_accuracy: 0.5752 - false_positives: 102.0000 - auc: 0.6188

Model 2: Epoch 100/100 10/10 [==============================] - 0s 7ms/step - loss: 0.4247 - binary_accuracy: 0.8127 - false_positives_1: 40.0000 - auc_1: 0.8864

Model 3: Epoch 500/500 3/3 [==============================] - 0s 8ms/step - loss: 0.4351 - binary_accuracy: 0.8047 - false_positives_2: 36.0000 - auc_2: 0.8861

Model 4: Epoch 506/506 379/379 [==============================] - **2s** 5ms/step - loss: 0.4254 - binary_accuracy: 0.7968 - false_positives_3: 38.0000 - auc_3: 0.8899

While this is just a snapshot of each model, it gives a good comparison that three of the four models have converged to a similar metric values. Model 1 failed to do this but as given in model 3 (which was model 1 with more epochs) we see that it can achieve better performace if given enough epochs and runtime.

What is also shown here is that in model 4, the true SGD, each single sample took approximatly 2 seconds to run. This was much, much longer than the other models.

The primary advantage that the second model had was that it converged in approximatly 30 epochs. The first model (which was also the third) tood a little over 100 epochs. The last model (SGD) took around 50 but at a much, much slower pace.

From this, the second model, which additional layers, was the best model. The additional layers allowed for additional weights to be used within the model. The next best was the third model (origional with 500 epochs).

As a note to the reader, and to my future self...

On my output layer, I was origionally using the 'softmax' activation instead of 'sigmoid'. I was expecting the softmax to create the probability of each of the two categories. However, when running, it failed to increase the binary_accuracy or ROC. Instead, it returned a binary_accuracy and ROC of a constant 0.5. The false positives were stuck at a constant as well (I believe 168?). All in all I was surprised by this and after some tinkering, switched to the sigmoid and obtained the results presented here.

Please also note, I cleared the outputs on Models 2-4 as the epochs were long and it created an 88 page .pdf

```
#Serialize the model weights for loading

model.save('model_v1')
model_v2.save('model_v2')
model_v3.save('model_v3')
model_v4.save('model_v4')
```

```
INFO:tensorflow:Assets written to: model_v1\assets
INFO:tensorflow:Assets written to: model_v2\assets
INFO:tensorflow:Assets written to: model_v3\assets
INFO:tensorflow:Assets written to: model_v4\assets
```