In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:
```python
df = pd.read_csv("/content/drive/MyDrive/DL/diabetes.csv")
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [3]:
```python
df.shape
```

Out[3]:
```
(768, 9)
```

Exploratory Data Analysis (EDA) and Preprocessing

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:
```python
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Missing Values

Here we can see certain columns have minimum value as 0 which is clearly not logical. The columns are:

1. Glucose
2. Blood Pressure
3. Skin Thickness
4. Insulin
5. BMI

Next we need to check the amount of missing information in these columns.

In [6]:
```python
# Checking 0 value rows in specific columns

x = df[["Glucose","BloodPressure","SkinThickness","Insulin","BMI"]] == 0
x = x.sum()
print(x)
```

```
Glucose            5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI               11
dtype: int64
```

Treating missing data

In [7]:
```python
# change 0 values to NAN for fewer missing rows
import numpy as np

# Mark as Nan
df[["BloodPressure","Glucose","BMI"]] = df[["BloodPressure","Glucose","BMI"]].replace(0, np.
NaN)
# fill missing values with mean column values
df.fillna(df.mean(), inplace=True)
```

In [8]:
```python
df.describe()
```

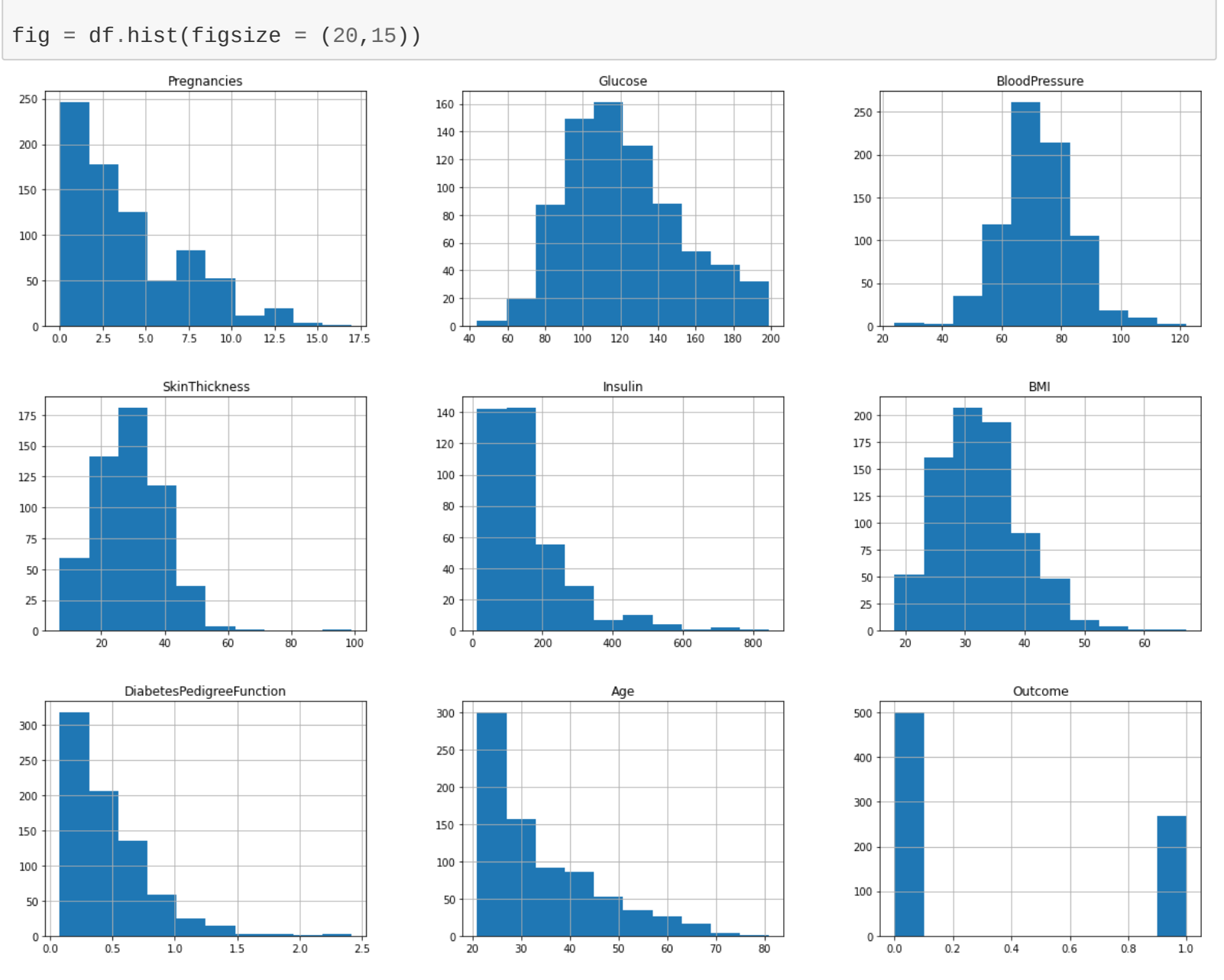
Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	20.536458	79.799479	32.457464	0.471876	33.240885
std	3.369578	30.435949	12.096346	15.952218	115.244002	6.875151	0.331329	11.760232
min	0.000000	44.000000	24.000000	0.000000	0.000000	18.200000	0.078000	21.000000
25%	1.000000	99.750000	64.000000	0.000000	0.000000	27.500000	0.243750	24.000000
50%	3.000000	117.000000	72.202592	23.000000	30.500000	32.400000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Checking the data distribution to decide how to treat Missing values for Insulin and SkinThickness

In [9]:
```python
df[["SkinThickness","Insulin"]] = df[["SkinThickness","Insulin"]].replace(0, np.NaN)

fig = df.hist(figsize = (20,15))
```



From distribution we will treat missing values of Skin Thickness and Insulin with median values

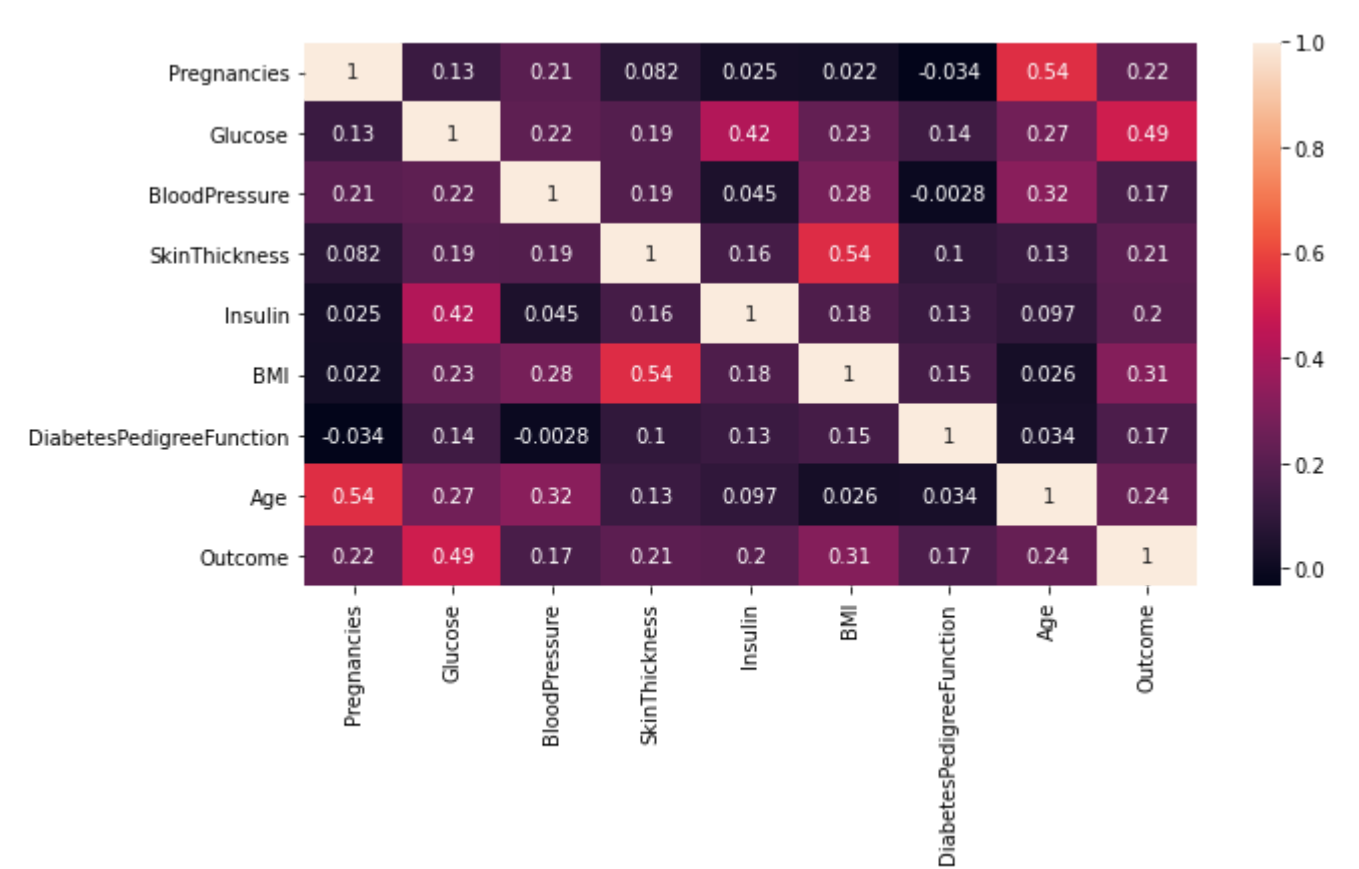
In [10]:
```python
df['SkinThickness'].fillna(df['SkinThickness'].median(), inplace=True)
df['Insulin'].fillna(df['Insulin'].median(), inplace=True)
```

In [11]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

In [12]:
```python
import seaborn as sns
plt.figure(figsize = (10,5))
sns.heatmap(df.corr(),annot = True)
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5443885c10>
```



From the correlation matrices, a few Clear correlations that are standing out are:

- Glucose and Insulin seem to have a high correlation : 0.42
- Body Mass Index and Skin Thcikness seem to have a high correlation: 0.54
- Pregnancies and Age seem to have a high correlation : 0.54

Therefore we will drop few columns in order to remove multicollinearity

In [13]:
```python
df=df.drop(['Insulin','SkinThickness','Pregnancies'],axis=1)
```

In [14]:
```python
df.head()
```

Out[14]:

	Glucose	BloodPressure	BMI	DiabetesPedigreeFunction	Age	Outcome
0	148.0	72.0	33.6	0.627	50	1
1	85.0	66.0	26.6	0.351	31	0
2	183.0	64.0	23.3	0.672	32	1
3	89.0	66.0	28.1	0.167	21	0
4	137.0	40.0	43.1	2.288	33	1

In [15]:
```python
x = df.drop('Outcome',axis=1)
y = df['Outcome']
```

Dividing into training and testing data

In [16]:
```python
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=1,test_size=0.3)
```

In [17]:
```python
ss=StandardScaler()
xtrain=ss.fit_transform(xtrain)
xtest=ss.transform(xtest)
```

In [18]:
```python
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_dim=5, activation="sigmoid")
])
```

In [19]:
```python
model.compile(optimizer="sgd", loss="binary_crossentropy")
```

In [20]:
```python
trained_model = model.fit(xtrain,ytrain,epochs=50)
```

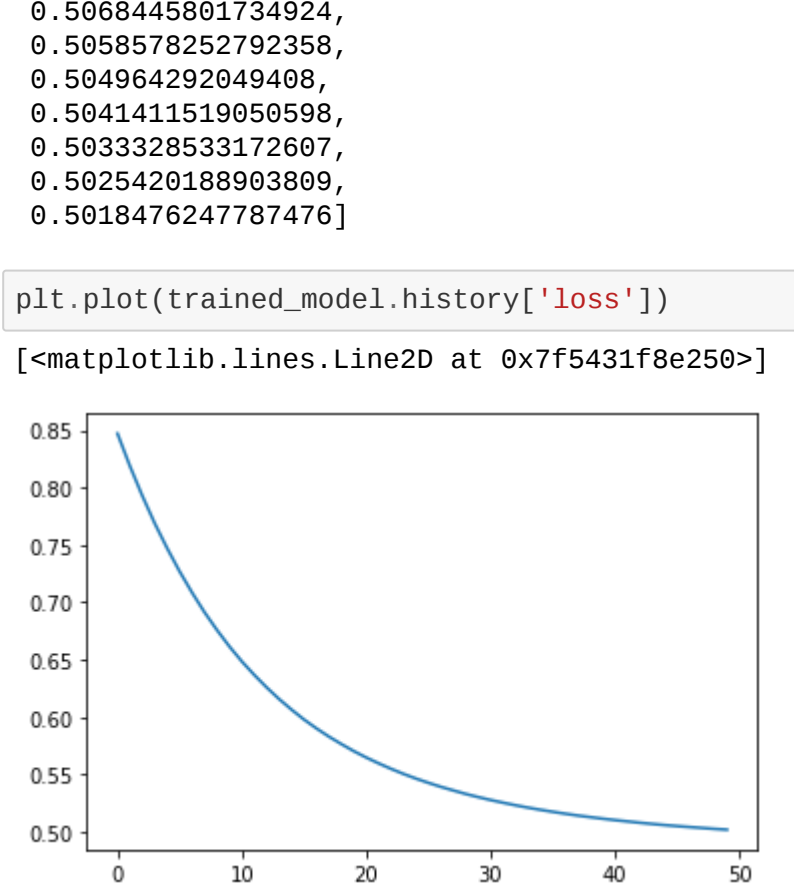
```
Epoch 1/50
17/17 [==============================] - 0s 1ms/step - loss: 0.8471
Epoch 2/50
17/17 [==============================] - 0s 1ms/step - loss: 0.8188
Epoch 3/50
17/17 [==============================] - 0s 1ms/step - loss: 0.7928
Epoch 4/50
17/17 [==============================] - 0s 1ms/step - loss: 0.7688
Epoch 5/50
17/17 [==============================] - 0s 1ms/step - loss: 0.7470
Epoch 6/50
17/17 [==============================] - 0s 1ms/step - loss: 0.7269
Epoch 7/50
17/17 [==============================] - 0s 1ms/step - loss: 0.7084
Epoch 8/50
17/17 [==============================] - 0s 1ms/step - loss: 0.6913
Epoch 9/50
17/17 [==============================] - 0s 2ms/step - loss: 0.6759
Epoch 10/50
17/17 [==============================] - 0s 2ms/step - loss: 0.6615
Epoch 11/50
17/17 [==============================] - 0s 2ms/step - loss: 0.6485
Epoch 12/50
17/17 [==============================] - 0s 1ms/step - loss: 0.6367
Epoch 13/50
17/17 [==============================] - 0s 1ms/step - loss: 0.6258
Epoch 14/50
17/17 [==============================] - 0s 1ms/step - loss: 0.6157
Epoch 15/50
17/17 [==============================] - 0s 1ms/step - loss: 0.6064
Epoch 16/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5978
Epoch 17/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5901
Epoch 18/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5830
Epoch 19/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5765
Epoch 20/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5703
Epoch 21/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5647
Epoch 22/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5596
Epoch 23/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5549
Epoch 24/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5504
Epoch 25/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5464
Epoch 26/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5427
Epoch 27/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5392
Epoch 28/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5361
Epoch 29/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5330
Epoch 30/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5303
Epoch 31/50
17/17 [==============================] - 0s 2ms/step - loss: 0.5277
Epoch 32/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5254
Epoch 33/50
17/17 [==============================] - 0s 2ms/step - loss: 0.5231
Epoch 34/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5211
Epoch 35/50
17/17 [==============================] - 0s 2ms/step - loss: 0.5192
Epoch 36/50
17/17 [==============================] - 0s 2ms/step - loss: 0.5174
Epoch 37/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5158
Epoch 38/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5142
Epoch 39/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5128
Epoch 40/50
17/17 [==============================] - 0s 2ms/step - loss: 0.5114
Epoch 41/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5102
Epoch 42/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5090
Epoch 43/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5079
Epoch 44/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5068
Epoch 45/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5059
Epoch 46/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5050
Epoch 47/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5041
Epoch 48/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5033
Epoch 49/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5025
Epoch 50/50
17/17 [==============================] - 0s 1ms/step - loss: 0.5018
```

In [21]:
```python
trained_model.history['loss']
```

Out[21]:
```
[0.847138524055481,
 0.818816602230072,
 0.7927969098091125,
 0.7687559723854065,
 0.7470459938049316,
 0.7268669605255127,
 0.7084356546401978,
 0.691319465637207,
 0.6758769750595093,
 0.6615059971809387,
 0.6484941244125366,
 0.6367011666297913,
 0.6257579326629639,
 0.615656316280365,
 0.6064385175704956,
 0.5978204607963562,
 0.5900984406471252,
 0.5829575657844543,
 0.57646244764328,
 0.5703063011169434,
 0.5647388100624084,
 0.559632420539856,
 0.5548587441444397,
 0.5504354238510132,
 0.5463824272155762,
 0.5426715612411499,
 0.5391868948936462,
 0.5360558032989502,
 0.5329921245574951,
 0.5302549004554749,
 0.5277054905891418,
 0.5253647565841675,
 0.5231305360794067,
 0.5210687518119812,
 0.519173264503479,
 0.5174192190170288,
 0.51575767993927,
 0.5141831040382385,
 0.5127838850021362,
 0.5114448666572571,
 0.5101888179779053,
 0.5090417861938477,
 0.5078962445259094,
 0.5068445801734924,
 0.5058578252792358,
 0.504964292049408,
 0.5041411519050598,
 0.5033328533172607,
 0.5025420188903809,
 0.5018476247787476]
```

In [22]:
```python
plt.plot(trained_model.history['loss'])
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x7f5431f8e250>]
```



In [23]:
```python
ypred=model.predict(xtest)
```

In [24]:
```python
ypred = np.where(ypred >= 0.5,1,0)
```

In [25]:
```python
from sklearn.metrics import classification_report
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.88      0.84       146
           1       0.76      0.66      0.70        85

    accuracy                           0.80       231
   macro avg       0.79      0.77      0.77       231
weighted avg       0.79      0.80      0.79       231
```