Diabetes Binary Classification

Objective: classify a diabetes diagnosis based on patients' data. We want to gain insights on different factors - such as demographics, lifestyle, medical history, medical measurements, medication, symtoms - that are associated to diabetes. Then select features that have strong correlation with the diagnosis and use it to train a classification model.

In [1]:
```
!pip install import-ipynb
!pip install pandoc
!pip install nbconvert
!pip install lightgbm
!pip install xgboost
```

```
3/lib/python3.11/site-packages (from nbconvert) (2.15.1)
Requirement already satisfied: tinycss2 in /opt/anaconda3/lib/py
thon3.11/site-packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.1 in /opt/anaconda3/
lib/python3.11/site-packages (from nbconvert) (5.7.1)
Requirement already satisfied: six>=1.9.0 in /opt/anaconda3/lib/
python3.11/site-packages (from bleach!=5.0.0->nbconvert) (1.16.
0)
Requirement already satisfied: webencodings in /opt/anaconda3/li
b/python3.11/site-packages (from bleach!=5.0.0->nbconvert) (0.5.
1)
Requirement already satisfied: platformdirs>=2.5 in /opt/anacond
a3/lib/python3.11/site-packages (from jupyter-core>=4.7->nbconve
rt) (3.10.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /opt/an
aconda3/lib/python3.11/site-packages (from nbclient>=0.5.0->nbco
nvert) (7.4.9)
Requirement already satisfied: fastjsonschema in /opt/anaconda3/
lib/python3.11/site-packages (from nbformat>=5.7->nbconvert) (2.
16.2)
```

I. Import libraries and datasets

These are the libraries to import and common methods to use for the project.

- **pandas**. Useful for:
  - visualization
    - `hist()`
  - normalization,
  - merge and join,
  - loading and savings
    - `read_csv()`
  - data inspection
  - other

Press Option + A to listen

        - `corr()` (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html)
        - `drop()` (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html)
- **numpy**: fundamental package for scientific computing in Python.
    - `array()`
    - `squeeze()`
- **tensorflow** provides the model we can use to make the predication on our diabetes dataset
- **sklearn** basic library for data mining
    - metrics
        - `confusion_matrix()` (https://www.tensorflow.org/api_docs/python/tf/math/confusion_matrix)
        - `ConfusionMatrixDisplay()`
    - model_selection
        - `train_test_split()` (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.htr
          Use `random_state` parameter to randomize the split selection.
    - preprocessing
        - `MinMaxScaler` (https://scikit-learn.org/stable/modules/preprocessing.html)
- **seaborn**. Useful for high level data visualization
    - `heatmap()`
    - `pairplot()` (https://seaborn.pydata.org/generated/seaborn.pairplot.html)
- **`matplotlib.pyplot`** : useful for plotting graphs in Pythong

Press Option + A to listen

In [2]:
```
import import_ipynb
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn as sk
import tensorflow as tf

import matplotlib.pyplot as plt
import tensorflow.keras as keras

from keras.layers import Dense
from keras.losses import BinaryCrossentropy
from keras.models import Sequential

from lightgbm import LGBMClassifier

from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassi
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_s
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, RobustScaler, Standar
from sklearn.tree import DecisionTreeClassifier

from xgboost import XGBClassifier

from utils_custom import *
```

```
importing Jupyter notebook from utils_custom.ipynb
```

II. Describe and display the datasets

This dataset contains comprehensive health data for 1,879 patients, uniquely identified with IDs ranging from 6000 to 7878. The data includes demographic details, lifestyle factors, medical history, clinical measurements, medication usage, symptoms, quality of life scores, environmental exposures, and health behaviors. Each patient is associated with a confidential doctor in charge, ensuring privacy and confidentiality.

[Data source: Diabetes Health Dataset Analysis (https://www.kaggle.com/datasets/rabieelkharoua/diabetes-health-dataset-analysis/data)](https://www.kaggle.com/datasets/rabieelkharoua/diabetes-health-dataset-analysis/data)

We will read the diabetes medical information from a .csv file

In [3]:
```
diabetes_df = pd.read_csv('diabetes_data.csv')
```

We will view the firs five rows of the dataset. We also want to display the info to show the column labels, non-empty cells count and the datatypes.

In [4]:
```
diabetes_df.head(5)
```

Out[4]:

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationLevel	BMI	Smok
0	6000	44	0	1	2	1	32.985284	
1	6001	51	1	0	1	2	39.916764	
2	6002	89	1	0	1	3	19.782251	
3	6003	21	1	1	1	2	32.376881	
4	6004	27	1	0	1	3	16.808600	

5 rows × 46 columns

Let's print out the summary of the diabetes dataframe. This summary will include the column labels, non-null count and the datatype included in the diabetes dataset. We can see that there are 46 labels or features. We can also note that every feauture does not contain an empty data. We can verify that by looking at the non-null count column.

In [5]:
```
diabetes_df.info();
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1879 entries, 0 to 1878
Data columns (total 46 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   PatientID                     1879 non-null   int64  
 1   Age                           1879 non-null   int64  
 2   Gender                        1879 non-null   int64  
 3   Ethnicity                     1879 non-null   int64  
 4   SocioeconomicStatus           1879 non-null   int64  
 5   EducationLevel                1879 non-null   int64  
 6   BMI                           1879 non-null   float64
 7   Smoking                       1879 non-null   int64  
 8   AlcoholConsumption            1879 non-null   float64
 9   PhysicalActivity              1879 non-null   float64
 10  DietQuality                   1879 non-null   float64
 11  SleepQuality                  1879 non-null   float64
 12  FamilyHistoryDiabetes         1879 non-null   int64  
 13  GestationalDiabetes           1879 non-null   int64  
```

We can also use `Dataframe.notna()` to check if every patient's data is completely filled. In the case that we detect an incomplete data, we can do either of these two things:

1. We can remove that data from the set, OR
2. We have to go through another step of collaborative filtering to guess the closest possible value

In this step, since we verified that every data is complete, we can just proceed to the next process.

We can use `describe()` method to highlight the average, lowest and highest value in

In [6]:
```
diabetes_df.describe()
```

Out[6]:

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationL
count	1879.000000	1879.000000	1879.000000	1879.000000	1879.000000	1879.00
mean	6939.000000	55.043108	0.487493	0.755721	0.992017	1.69
std	542.564896	20.515839	0.499977	1.047558	0.764940	0.88
min	6000.000000	20.000000	0.000000	0.000000	0.000000	0.00
25%	6469.500000	38.000000	0.000000	0.000000	0.000000	1.00
50%	6939.000000	55.000000	0.000000	0.000000	1.000000	2.00
75%	7408.500000	73.000000	1.000000	1.000000	2.000000	2.00
max	7878.000000	90.000000	1.000000	3.000000	2.000000	3.00

8 rows × 45 columns

In [7]:
```
ave_age = diabetes_df['Age'].mean()
diag_ct = diabetes_df['Diagnosis'].sum()
pos_per = diag_ct.sum()/len(diabetes_df) * 100
female_ct = diabetes_df['Gender'].sum()
female_per = female_ct.sum()/len(diabetes_df) * 100
male_per = 100 - female_per
```

In [8]:
```
print(f"The average age is {ave_age:.2f} years old.")
print(f"There are {female_per:.2f}% females and {male_per:.2f}% males
print(f"There are {pos_per:.2f}% diagnosed as diabetic.")
```

```
The average age is 55.04 years old.
There are 48.75% females and 51.25% males.
There are 40.02% diagnosed as diabetic.
```

Before we visualize our data, we noticed that we do not need some columns like PatienID and DoctorInCharge. We can drop these columns from the original data.

Note: If you are getting a Keyerror because of the labels, PatientID and DoctorInCharge are not in axis, this means that you probably have removed the columns already from your run previously. In this case, you can ignore and go to the next step.

In [9]:
```
diabetes_df = diabetes_df.drop(columns=['PatientID', 'DoctorInCharge'
```

III. Visualize data

We will be using data visualization library, `seaborn` to help us identify which features in our dataset can affect greatly with the diabetes classification.

1. In this data, we can see the distribution in each features. For example in the `Diagnosis` chart (the last chart at the bottom), we can see graphically that there are more patients in our dataset that do not have diabetes. We can also see that there are ~750 patients that are diagnosed with diabetes. We can also take note that more than 50% are male. Majority of the patients in this dataset are Caucasion. There are also high number of patients belonging in the middle class. Majority have high school and undergraduate degrees. Most are non-smokers, no family history of diabetes,, no gestationsl diabetes, no polycystic ovary syndrome, no previous predisposed diabetes, no hypertension.

Here, we want to see the profile of patients with diabetes. We notice that majority of patients with diabetes has high readings of FastingBloodSugar and HbA1c.

2. Another way of visualizing out data is by using the correlation function. `corr()` is another way of visualizing correlationship between a pair of features and the `Diagnosis` results. We will ignore the white diagonal line because it's just trying to find similarity with itself. But we can look at lighter and brighter colors to find the most similar features.
3. The blue dot represents the patients who were not diagnosed with diabetes. The orange dot represents patients who are diagnosed with diabetes.
4. Let's take a look at how our dataset looks on a 3d plot using *FastingBloodSugar* , *HbA1c* and *Diagnosis*. We can observe that the dataset on the top layer are patients diagnosed with diabetes. We can observe that there is a clump forming on the top right quadrant with many outliers sparsely spread out.

We narrow down the features and join them in strong_features. Here we can notice that `FastingBloodSugar` and `HbA1c` has strong influence on the diabetic results.

Press Option + A to listen

In [10]:
```
selected_features = [
                    'Age',
                    'BMI',
#                     'AlcoholConsumption',
#                     'PhysicalActivity',
#                     'DietQuality',
#                     'SleepQuality',
                    'SystolicBP',
                    'DiastolicBP',
                    'FastingBloodSugar',
                    'HbA1c',
                    'SerumCreatinine',
#                     'BUNLevels',
#                     'CholesterolTotal',
#                     'CholesterolLDL',
#                     'CholesterolHDL',
#                     'CholesterolTriglycerides',
#                     'AntidiabeticMedications',
                     ]
```

Run this line if you want to visualize the diabetes data in histogram, heatmap, pairplot and 3D scatterplot.

In [11]:
```
visualize_data(diabetes_df, selected_features)
```

```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:
1119: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN before
operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:
1119: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN before
operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:
1119: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN before
operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:
1119: FutureWarning: use_inf_as_na option is deprecated and will
be removed in a future version. Convert inf values to NaN before
operating instead.
```

Feature Engineering

1. Possible Engineered feature we can use is the ratio of Systolic and Diastolic Blood Pressure.

Press Option + A to listen

In [12]:
```
systolic = diabetes_df['SystolicBP']
diastolic = diabetes_df['DiastolicBP']
bloodPressure = (systolic / diastolic)
bloodPressure = pd.DataFrame({'BloodPressure': bloodPressure})
```

In [13]:
```
# diabetes_df = diabetes_df.drop(columns='BloodPressure')
# diabetes_df.insert(43, 'BloodPressure', bloodPressure)
```

Then, we will add the new features in our selected feature list.

In [14]:
```
# new_features = ['BloodPressure']
# selected_features.extend(new_features)
selected_features
```

Out[14]:
```
['Age',
 'BMI',
 'SystolicBP',
 'DiastolicBP',
 'FastingBloodSugar',
 'HbA1c',
 'SerumCreatinine']
```

We found out that adding this feature did not improve our prediction model.

IV. Preprocessing

Scaling the data

We need to separate the inputs and the output. The X input should contain all of the selected features, except for the `Diagnosis`. The output should contain the `Diagnosis` results, 0 for not diabetic and 1 for diabetic. The output *y* will be the groundtruth for our model later.

We scale the data because we want our model to converge faster. This helps with the time efficiency.

In [15]:
```python
X = diabetes_df[selected_features]
X
```

Out[15]:

	Age	BMI	SystolicBP	DiastolicBP	FastingBloodSugar	HbA1c	SerumCreat
0	44	32.985284	93	73	163.687162	9.283631	2.66
1	51	39.916764	165	99	188.347070	7.326870	4.17
2	89	19.782251	119	91	127.703653	4.083426	1.97
3	21	32.376881	169	87	82.688415	6.516645	3.05
4	27	16.808600	165	69	90.743395	5.607222	4.15
...	...	...	...	...	...	...	
1874	37	20.811137	104	74	109.832032	5.920723	3.98
1875	80	27.694312	166	115	90.729361	7.332397	2.13
1876	38	35.640824	128	70	149.366801	4.907208	2.19
1877	43	32.423016	124	91	162.027044	8.820613	0.89

In [16]:
```python
Y = diabetes_df['Diagnosis']
```

To scale, we will be using the `MinMaxScaler` from sci-kit learn. `MinMaxScaler()` will narrow down the range of values in each feature.

`MinMaxScaler()` transforms features by scaling each feature to a given range.

`fit_transform()` learns the parameters and apply the transformation to new data

In [17]:
```
scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled
```

Out[17]:

	Age	BMI	SystolicBP	DiastolicBP	FastingBloodSugar	HbA1c	SerumC
0	0.342857	0.719155	0.033708	0.220339	0.720868	0.881839	
1	0.442857	0.996715	0.842697	0.661017	0.910763	0.555064	
2	0.985714	0.190460	0.325843	0.525424	0.443775	0.013416	
3	0.014286	0.694792	0.887640	0.457627	0.097133	0.419758	
4	0.100000	0.071385	0.842697	0.152542	0.159161	0.267887	
...	...	...	...	...	...	...	
1874	0.242857	0.231661	0.157303	0.237288	0.306154	0.320241	
1875	0.857143	0.507286	0.853933	0.932203	0.159053	0.555987	
1876	0.257143	0.825491	0.426966	0.169492	0.610593	0.150986	
1877	0.328571	0.696640	0.382022	0.525424	0.708084	0.804516	

Splitting data

We want to split our datasets into three:

data	% of total	Description
training	60	Data used to tune model parameters w and b in training or fitting
cross-validation	20	Data used to tune other model parameters like degree of polynomial, regularization or the architecture of a neural network.
test	20	Data used to test the model after tuning to gauge performance on new data

In [18]:
```
X_train, y_train, X_cv, y_cv, X_test, y_test = split_dataset(X_scalec
X_train.shape, y_train.shape, X_cv.shape, y_cv.shape, X_test.shape, y
```

Out[18]:
```
((1127, 7), (1127,), (376, 7), (376,), (376, 7), (376,))
```

V. Adding Polynomial feature (Optional)

NOTE: Do not run part V if you don't want to add polynomial feature on your selected features. We may not need polynomial feature when we are using neural network because neural networks can learn non-linear relationship.

Press Option + A to listen

We will generate polynomial features from our training set using `PolynomialFeatures()`

Training set

1. Initiate Polynomial Features
2. Add the polynomial features in the training set using *fit_transform()*
3. Scale the training set
4. Create and train model. We will use *LinearRegression* for the model and we will use *fit()* to train the set with the model.
5. Compute the *Mean Square Error*

Cross-Validation and Testing set

1. Add polynomial features and scale using *transform()*
2. Compute the *Mean Square Error*

All these steps are done inside the `poly_optizer()`.

fit() vs transform() vs fit_transform()

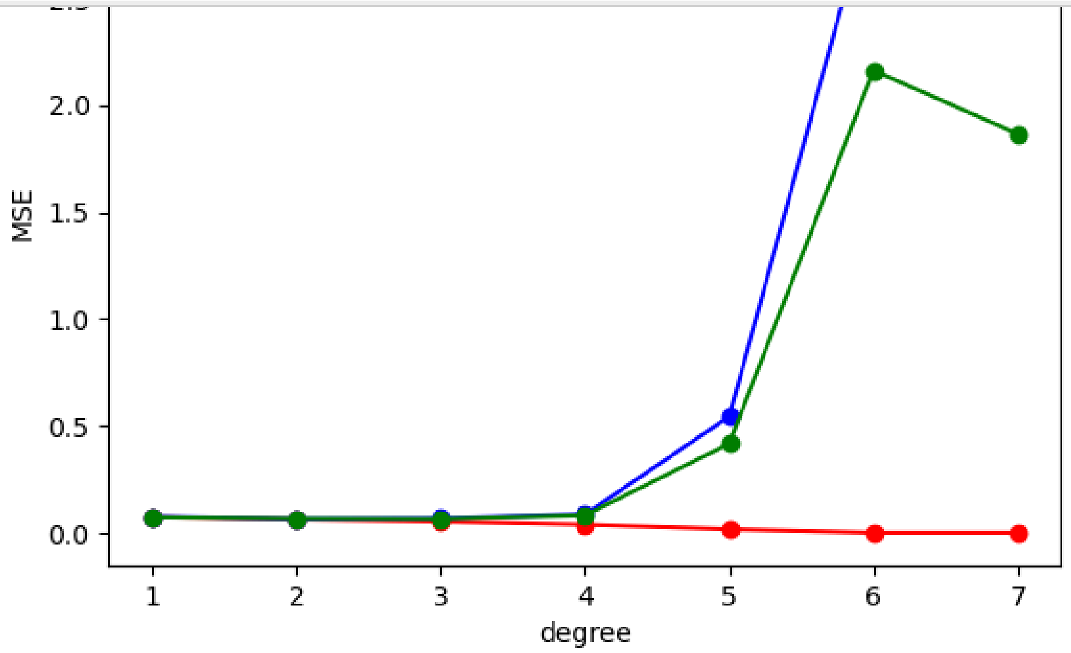
- *fit()* learn and estimate the parameters fo the transformation
- *transform()* apply the learned transformation to new data
- *fit_transform()* learn the parameters and apply the transformation to new data.

In [19]:
```
degrees = range(1, 8)
datasets = [X_train, y_train, X_cv, y_cv, X_test, y_test]
maps, mses = poly_optimizer(degrees, datasets)
```

```
1 (1127, 7) (376, 7) (376, 7)
2 (1127, 35) (376, 35) (376, 35)
3 (1127, 119) (376, 119) (376, 119)
4 (1127, 329) (376, 329) (376, 329)
5 (1127, 791) (376, 791) (376, 791)
6 (1127, 1715) (376, 1715) (376, 1715)
7 (1127, 3431) (376, 3431) (376, 3431)
```

Press Option + A to listen

In [20]:
```
# Plot the results
plot_train_cv_test_mses(degrees, mses['train'], mses['cv'], mses['tes
```



In the plot above, we ran all our three datasets and measure the performance of each degrees in the polynomial. We can see that all of our sets unanimously perform better when we add 2nd order polynomial because it has the lowest MSE (Mean Square Error). This implies that the model is able to learn the patterns from the training set without overfitting.

Choosing the Polynomial Feature to add to our data

In [21]:
```
# Get the model with the lowest CV MSE (add 1 because list indices st
# This also corresponds to the degree of the polynomial added

deg = np.argmin(mses['cv'])

X_train = maps['train'][deg]
X_cv = maps['cv'][deg]
X_test = maps['test'][deg]

train_mse = mses['train'][deg]
cv_mse = mses['cv'][deg]
test_mse = mses['test'][deg]

print(f"Lowest CV MSE is found in the model with degree={deg + 1} wit
```

```
Lowest CV MSE is found in the model with degree=2 with 0.06 mse
```

Press Option + A to listen

VI. Training the classification models

Here, we will be looking at 3 different architectures of neural network. This is how we build the models in `build_models()` :

Model Name	# of Hidden Layers	Activation layers
Model_1	3	(relu, relu, sigmoid)
Model_2	5	(relu, relu, relu, relu, sigmoid)
Model_3	6	(relu, relu, relu, relu, relu, sigmoid)

We want the input to be the same length as our features. For the output, we want the size to be 1 since we just want a binary output, either 0 or 1. For binary classification, we want to use 'sigmoid' rather than 'linear' for the final neuron.

In [22]:
```python
rows_len = X_train.shape[1]
rows_len = (rows_len, )
regularizer = 1e-5 # best
# regularizer = 1e3

models = build_models(rows_len, regularizer)

for model_name, model in models.items():
    model.summary()
```

```
 Total params: 697 (2.72 KB)

 Trainable params: 697 (2.72 KB)

 Non-trainable params: 0 (0.00 B)

Model: "Model_2"
```

Layer (type)	Output Shape	
dense_3 (Dense)	(None, 20)	
dense_4 (Dense)	(None, 12)	
dense_5 (Dense)	(None, 12)	

For the loss function, we want to use binary entropy rather than 'mean square error'). We also want to monitor the accuracy of our model for each epoch. The reason why we used *Binary Cross Entropy (Log Loss)* for the loss function because BCE is designed for problems whose output is probability value between 0 or 1. BCE also take into account the probability for actual class and penalize wrong predictions.

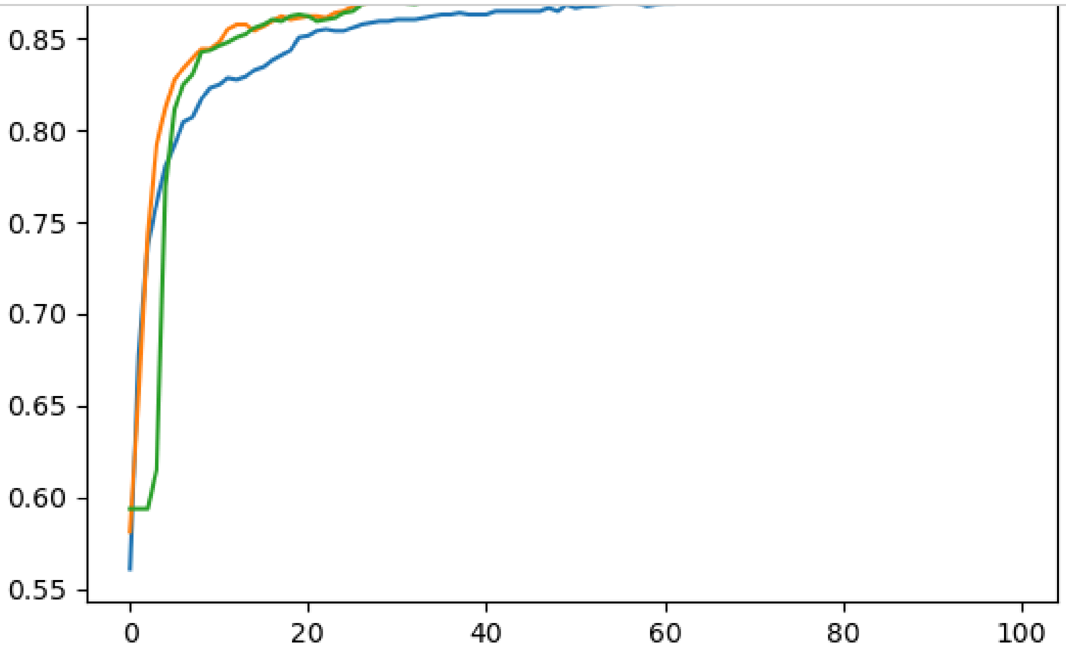
In [23]:
```
histories = []
for model_name, model in models.items():
    print(f"Training with {model_name}")
    model.compile(optimizer='Adam', loss=BinaryCrossentropy(from_logi
    history = model.fit(X_train,
                        y_train,
                        epochs = 100,
                        batch_size = 50,
                        validation_data = (X_cv, y_cv) )
    histories.append(history)
```

```
Training with Model_1
Epoch 1/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4998 - los
s: 0.6922 - mse: 0.2494 - val_accuracy: 0.6383 - val_loss: 0.664
4 - val_mse: 0.2356
Epoch 2/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step - accuracy: 0.6671 - lo
ss: 0.6580 - mse: 0.2324 - val_accuracy: 0.7074 - val_loss: 0.63
15 - val_mse: 0.2195
Epoch 3/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 915us/step - accuracy: 0.7272 - lo
ss: 0.6232 - mse: 0.2155 - val_accuracy: 0.7473 - val_loss: 0.59
70 - val_mse: 0.2030
Epoch 4/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step - accuracy: 0.7537 - lo
ss: 0.5878 - mse: 0.1987 - val_accuracy: 0.7500 - val_loss: 0.56
24 - val_mse: 0.1873
Epoch 5/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - accuracy: 0.7749 - lo
ss: 0.5518 - mse: 0.1825 - val_accuracy: 0.7500 - val_loss: 0.53
```

We can plot the model's accuracy on each epoch.

Press Option + A to listen

In [24]:
```python
for hist in histories:
    plt.plot(hist.history['accuracy'])
```



VII. Train with pretrained models

In [25]:
```python
model_libs = {
    'LogisticRegression': LogisticRegression(),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'RandomForestClassifier': RandomForestClassifier(),
    'GradientBoostingClassifier': GradientBoostingClassifier(),
    'AdaBoostClassifier': AdaBoostClassifier(),
    'ExtraTreesClassifier': ExtraTreesClassifier(),
    'XGBClassifier': XGBClassifier(use_label_encoder=False, eval_metr
    'LGBMClassifier': LGBMClassifier(verbosity = 0)
    }
```

Press Option + A to listen

In [26]:
```
accuracies = dict()
for model_name, model in model_libs.items():
    model.fit(X_train, y_train)

    y_pred = model.predict(X_cv)
    accuracy = accuracy_score(y_cv, y_pred)
    accuracies[model_name] = (model, accuracy)


    print(f"{model_name} - Accuracy: {accuracy*100:.2f}%")
```

```
LogisticRegression - Accuracy: 82.71%
DecisionTreeClassifier - Accuracy: 84.57%
RandomForestClassifier - Accuracy: 88.56%
GradientBoostingClassifier - Accuracy: 88.56%
AdaBoostClassifier - Accuracy: 86.44%
ExtraTreesClassifier - Accuracy: 87.23%
XGBClassifier - Accuracy: 88.03%
LGBMClassifier - Accuracy: 88.56%
```

VIII. Evaluate and select the optimal model

To evaluate the performance, we want to use `evaluate()`

In [27]:
```
for model_name, model in models.items():
    loss, accuracy, mse = model.evaluate(X_test, y_test)
    accuracies[model_name] = (model, accuracy)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 384us/step - accuracy: 0.8870 - loss:
0.3727 - mse: 0.1053
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 329us/step - accuracy: 0.8739 - loss:
0.3725 - mse: 0.1037
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 339us/step - accuracy: 0.8757 - loss:
0.4068 - mse: 0.1101
```

Press Option + A to listen

In [28]:
```
opt_model = None
highest = 0
for model_name, model_acc in accuracies.items():
    model, acc = model_acc
    if acc > highest:
        opt_model = model
        highest = acc

    print(f"The accuracy in {model_name} using cv set is {acc * 100:.
opt_model
```

```
The accuracy in LogisticRegression using cv set is 82.71%
The accuracy in DecisionTreeClassifier using cv set is 84.57%
The accuracy in RandomForestClassifier using cv set is 88.56%
The accuracy in GradientBoostingClassifier using cv set is 88.56%
The accuracy in AdaBoostClassifier using cv set is 86.44%
The accuracy in ExtraTreesClassifier using cv set is 87.23%
The accuracy in XGBClassifier using cv set is 88.03%
The accuracy in LGBMClassifier using cv set is 88.56%
The accuracy in Model_1 using cv set is 87.77%
The accuracy in Model_2 using cv set is 86.70%
The accuracy in Model_3 using cv set is 86.97%
```

Out[28]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [29]:
```
model_name = opt_model
print(f"The model chosen is Model {model_name}")
```

```
The model chosen is Model RandomForestClassifier()
```

In [30]:
```
model.evaluate(X_train, y_train)
model.evaluate(X_cv, y_cv)
model.evaluate(X_test, y_test)
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 283us/step - accuracy: 0.9157 - loss:
0.2128 - mse: 0.0597
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 348us/step - accuracy: 0.8628 - loss:
0.4277 - mse: 0.1168
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 321us/step - accuracy: 0.8757 - loss:
0.4068 - mse: 0.1101
```

Out[30]:
```
[0.40667182207107544, 0.8696808218955994, 0.1129303053021431]
```

###IX. Search the optimal decision boundary

We want to choose the right decision boundary. By default, we can use 0.50 as the boundary point but we want see if we can improve our model's performance by optimizing the boundary point.

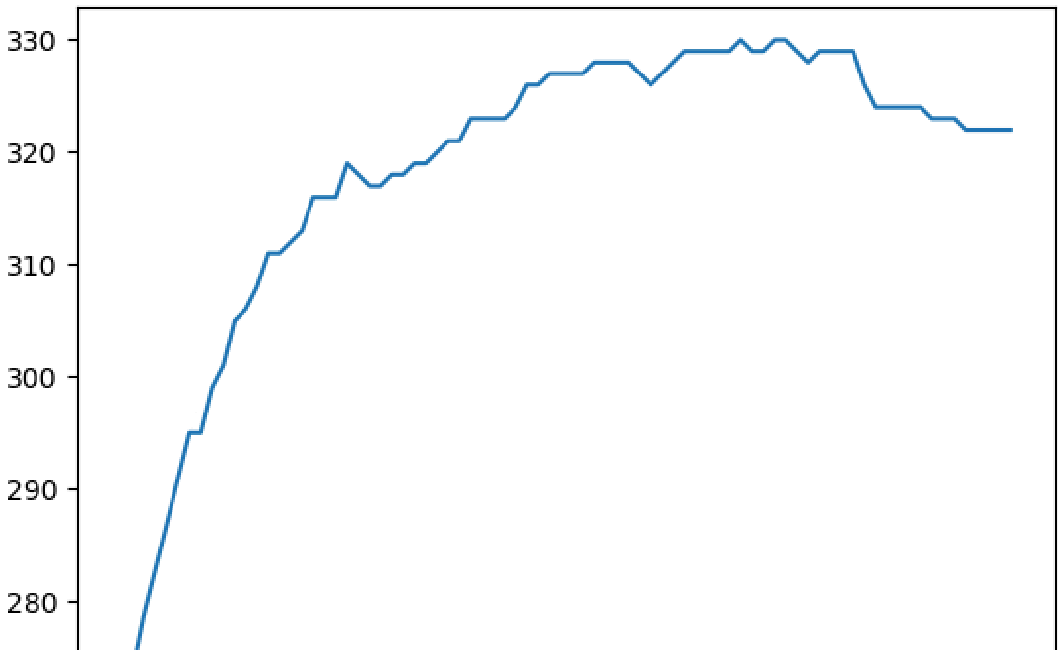
In [31]:
```
val, max_correct, y_pred, result = search_boundary(model, X_test, y_t

print(f"The optimal decision boundary for {model_name}: {val} with {m
```

```
<Sequential name=Model_3, built=True> (376, 35) (376,)
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
The optimal decision boundary for RandomForestClassifier(): 0.65 wit
h 330 correct predictons.
```

We plot the number of correct predictions vs. the possible decision value (in decimal).

In [32]:
```
plt.plot(result);
```

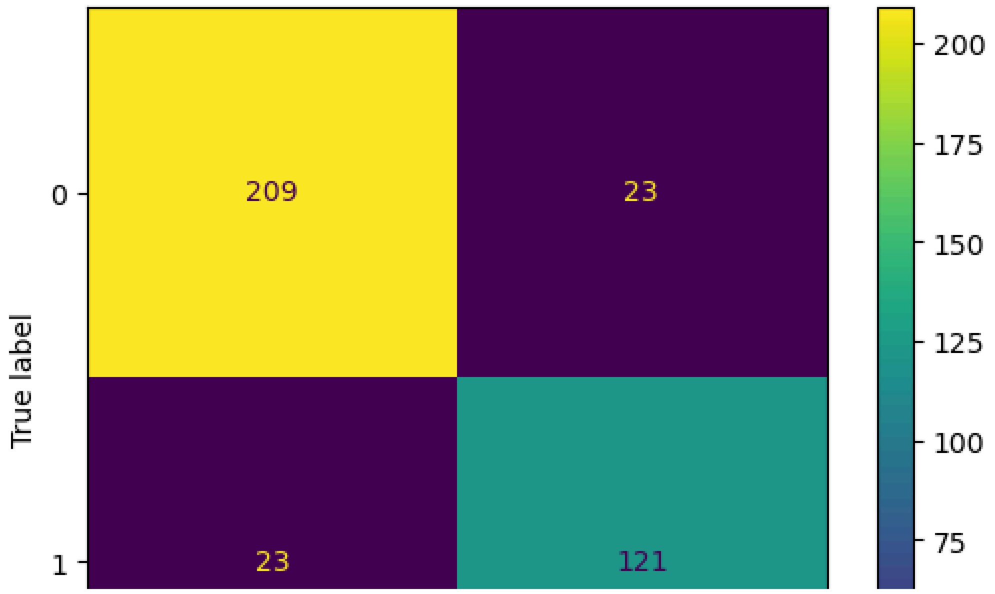


In [33]:
```
y_predict = [1 if x >= val else 0 for x in y_pred]
```

In [34]:
```
y_actual = np.array(y_test)
mat = confusion_matrix(y_actual, y_predict)
print(f"Accuracy from the prediction: {(mat.trace()/ mat.sum())*100:.

disp = ConfusionMatrixDisplay(confusion_matrix = mat)
disp.plot();
```

```
Accuracy from the prediction: 87.77%
```



In the confusion matrix, the diagonals are the number of correct predictions. We want to sum the total correct predictions using the `trace()` .

Modifications

- During the training, we reduce the hidden layer to 2 with fewer units.
- During the prediction, we used the optimal decision value by adjusting the step increments by 0.01. We found that the the sweet spot is at `{val}` instead of 0.05
- Feature reduction. We also reduce the number of features from 17 to 3 features. By reducing the number of features, our model's accuracy increased by 9%.

Things to improve

- Use polynomial to see if features needed to be changed. We found out that not adding the polynomical features slightly peform better.
- Apply existing models and evaluate their performance.

In [ ]:

Press Option + A to listen

Press Option + A to listen