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
```

```
Requirement already satisfied: import-ipynb in /opt/anaconda3/lib/python3.1
1/site-packages (0.1.4)
Requirement already satisfied: IPython in /opt/anaconda3/lib/python3.11/site
-packages (from import-ipynb) (8.20.0)
Requirement already satisfied: nbformat in /opt/anaconda3/lib/python3.11/sit
e-packages (from import-ipynb) (5.9.2)
Requirement already satisfied: decorator in /opt/anaconda3/lib/python3.11/si
te-packages (from IPython->import-ipynb) (5.1.1)
Requirement already satisfied: jedi>=0.16 in /opt/anaconda3/lib/python3.11/s
ite-packages (from IPython->import-ipynb) (0.18.1)
Requirement already satisfied: matplotlib-inline in /opt/anaconda3/lib/pytho
n3.11/site-packages (from IPython->import-ipynb) (0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in /opt/anacond
a3/lib/python3.11/site-packages (from IPython->import-ipynb) (3.0.43)
Requirement already satisfied: pygments>=2.4.0 in /opt/anaconda3/lib/python
3.11/site-packages (from IPython->import-ipynb) (2.15.1)
Requirement already satisfied: stack-data in /opt/anaconda3/lib/python3.11/s
ite-packages (from IPython->import-ipynb) (0.2.0)
Requirement already satisfied: traitlets>=5 in /opt/anaconda3/lib/python3.1
1/site-packages (from IPython->import-ipynb) (5.7.1)
Requirement already satisfied: pexpect>4.3 in /opt/anaconda3/lib/python3.11/
site-packages (from IPython->import-ipynb) (4.8.0)
Requirement already satisfied: fastjsonschema in /opt/anaconda3/lib/python3.
11/site-packages (from nbformat->import-ipynb) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in /opt/anaconda3/lib/python
3.11/site-packages (from nbformat->import-ipynb) (4.19.2)
Requirement already satisfied: jupyter-core in /opt/anaconda3/lib/python3.1
1/site-packages (from nbformat->import-ipynb) (5.5.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /opt/anaconda3/lib/pyt
hon3.11/site-packages (from jedi>=0.16->IPython->import-ipynb) (0.8.3)
Requirement already satisfied: attrs>=22.2.0 in /opt/anaconda3/lib/python3.1
1/site-packages (from jsonschema>=2.6->nbformat->import-ipynb) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /opt/
anaconda3/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat->impo
rt-ipynb) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in /opt/anaconda3/lib/pyt
hon3.11/site-packages (from jsonschema>=2.6->nbformat->import-ipynb) (0.30.
2)
Requirement already satisfied: rpds-py>=0.7.1 in /opt/anaconda3/lib/python3.
11/site-packages (from jsonschema>=2.6->nbformat->import-ipynb) (0.10.6)
Requirement already satisfied: ptyprocess>=0.5 in /opt/anaconda3/lib/python
3.11/site-packages (from pexpect>4.3->IPython->import-ipynb) (0.7.0)
Requirement already satisfied: wcwidth in /opt/anaconda3/lib/python3.11/site
-packages (from prompt-toolkit<3.1.0,>=3.0.41->IPython->import-ipynb) (0.2.
5)
Requirement already satisfied: platformdirs>=2.5 in /opt/anaconda3/lib/pytho
n3.11/site-packages (from jupyter-core->nbformat->import-ipynb) (3.10.0)
Requirement already satisfied: executing in /opt/anaconda3/lib/python3.11/si
te-packages (from stack-data->IPython->import-ipynb) (0.8.3)
Requirement already satisfied: asttokens in /opt/anaconda3/lib/python3.11/si
te-packages (from stack-data->IPython->import-ipynb) (2.0.5)
Requirement already satisfied: pure-eval in /opt/anaconda3/lib/python3.11/si
te-packages (from stack-data->IPython->import-ipynb) (0.2.2)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.11/site-pac
kages (from asttokens->stack-data->IPython->import-ipynb) (1.16.0)
Requirement already satisfied: pandoc in /opt/anaconda3/lib/python3.11/site-
```

```
packages (2.3)
Requirement already satisfied: plumbum in /opt/anaconda3/lib/python3.11/site
-packages (from pandoc) (1.8.3)
Requirement already satisfied: ply in /opt/anaconda3/lib/python3.11/site-pac
kages (from pandoc) (3.11)
Requirement already satisfied: nbconvert in /opt/anaconda3/lib/python3.11/si
te-packages (7.10.0)
Requirement already satisfied: beautifulsoup4 in /opt/anaconda3/lib/python3.
11/site-packages (from nbconvert) (4.12.2)
Requirement already satisfied: bleach!=5.0.0 in /opt/anaconda3/lib/python3.1
1/site-packages (from nbconvert) (4.1.0)
Requirement already satisfied: defusedxml in /opt/anaconda3/lib/python3.11/s
ite-packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in /opt/anaconda3/lib/python3.11/
site-packages (from nbconvert) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in /opt/anaconda3/lib/pytho
n3.11/site-packages (from nbconvert) (5.5.0)
Requirement already satisfied: jupyterlab-pygments in /opt/anaconda3/lib/pyt
hon3.11/site-packages (from nbconvert) (0.1.2)
Requirement already satisfied: markupsafe>=2.0 in /opt/anaconda3/lib/python
3.11/site-packages (from nbconvert) (2.1.3)
Requirement already satisfied: mistune<4,>=2.0.3 in /opt/anaconda3/lib/pytho
n3.11/site-packages (from nbconvert) (2.0.4)
Requirement already satisfied: nbclient>=0.5.0 in /opt/anaconda3/lib/python
3.11/site-packages (from nbconvert) (0.8.0)
Requirement already satisfied: nbformat>=5.7 in /opt/anaconda3/lib/python3.1
1/site-packages (from nbconvert) (5.9.2)
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.11/si
te-packages (from nbconvert) (23.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /opt/anaconda3/lib/py
thon3.11/site-packages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in /opt/anaconda3/lib/python
3.11/site-packages (from nbconvert) (2.15.1)
Requirement already satisfied: tinycss2 in /opt/anaconda3/lib/python3.11/sit
e-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/s
ite-packages (from bleach!=5.0.0->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /opt/anaconda3/lib/python3.1
1/site-packages (from bleach!=5.0.0->nbconvert) (0.5.1)
Requirement already satisfied: platformdirs>=2.5 in /opt/anaconda3/lib/pytho
n3.11/site-packages (from jupyter-core>=4.7->nbconvert) (3.10.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /opt/anaconda3/lib/
python3.11/site-packages (from nbclient>=0.5.0->nbconvert) (7.4.9)
Requirement already satisfied: fastjsonschema in /opt/anaconda3/lib/python3.
11/site-packages (from nbformat>=5.7->nbconvert) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in /opt/anaconda3/lib/python
3.11/site-packages (from nbformat>=5.7->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in /opt/anaconda3/lib/python3.1
1/site-packages (from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: attrs>=22.2.0 in /opt/anaconda3/lib/python3.1
1/site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /opt/
anaconda3/lib/python3.11/site-packages (from jsonschema>=2.6->nbformat>=5.7-
>nbconvert) (2023.7.1)
```

```
Requirement already satisfied: referencing>=0.28.4 in /opt/anaconda3/lib/pyt
hon3.11/site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.3
0.2)
Requirement already satisfied: rpds-py>=0.7.1 in /opt/anaconda3/lib/python3.
11/site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.10.6)
Requirement already satisfied: entrypoints in /opt/anaconda3/lib/python3.11/
site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (0.
4)
Requirement already satisfied: nest-asyncio>=1.5.4 in /opt/anaconda3/lib/pyt
hon3.11/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconve
rt) (1.6.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/
python3.11/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbco
nvert) (2.8.2)
Requirement already satisfied: pyzmq>=23.0 in /opt/anaconda3/lib/python3.11/
site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (24.
0.1)
Requirement already satisfied: tornado>=6.2 in /opt/anaconda3/lib/python3.1
1/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert)
(6.3.3)
Requirement already satisfied: lightgbm in /opt/anaconda3/lib/python3.11/sit
e-packages (4.4.0)
Requirement already satisfied: numpy>=1.17.0 in /opt/anaconda3/lib/python3.1
1/site-packages (from lightgbm) (1.26.4)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.11/site-p
ackages (from lightgbm) (1.11.4)
Requirement already satisfied: xgboost in /opt/anaconda3/lib/python3.11/site
-packages (2.0.3)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.11/site-p
ackages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.11/site-p
ackages (from xgboost) (1.11.4)
```

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
    - `corr()`
    - `drop()`
- **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()`
        - `ConfusionMatrixDisplay()`
    - model_selection
        - `train_test_split()`. Use `random_state` parameter to randomize the split selection.
    - preprocessing
        - `MinMaxScaler`
- **seaborn**. Useful for high level data visualization
    - `heatmap()`
    - `pairplot()`
- **`matplotlib.pyplot`**: useful for plotting graphs in Pythong
    - `subplots()`

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, RandomForestClassifier, G
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler
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

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	BM
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	19.78225
3	6003	21	1	1	1	2	32.37688
4	6004	27	1	0	1	3	16.80860

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  
 14  PolycysticOvarySyndrome         1879 non-null   int64  
 15  PreviousPreDiabetes             1879 non-null   int64  
 16  Hypertension                    1879 non-null   int64  
 17  SystolicBP                      1879 non-null   int64  
 18  DiastolicBP                     1879 non-null   int64  
 19  FastingBloodSugar               1879 non-null   float64
 20  HbA1c                           1879 non-null   float64
 21  SerumCreatinine                 1879 non-null   float64
 22  BUNLevels                       1879 non-null   float64
 23  CholesterolTotal                1879 non-null   float64
 24  CholesterolLDL                  1879 non-null   float64
 25  CholesterolHDL                  1879 non-null   float64
 26  CholesterolTriglycerides        1879 non-null   float64
 27  AntihypertensiveMedications     1879 non-null   int64  
 28  Statins                         1879 non-null   int64  
 29  AntidiabeticMedications         1879 non-null   int64  
 30  FrequentUrination               1879 non-null   int64  
 31  ExcessiveThirst                 1879 non-null   int64  
 32  UnexplainedWeightLoss           1879 non-null   int64  
 33  FatigueLevels                   1879 non-null   float64
 34  BlurredVision                   1879 non-null   int64  
 35  SlowHealingSores                1879 non-null   int64  
 36  TinglingHandsFeet               1879 non-null   int64  
 37  QualityOfLifeScore              1879 non-null   float64
 38  HeavyMetalsExposure             1879 non-null   int64  
 39  OccupationalExposureChemicals   1879 non-null   int64  
 40  WaterQuality                    1879 non-null   int64  
 41  MedicalCheckupsFrequency        1879 non-null   float64
 42  MedicationAdherence             1879 non-null   float64
 43  HealthLiteracy                  1879 non-null   float64
 44  Diagnosis                       1879 non-null   int64  
 45  DoctorInCharge                  1879 non-null   object 
dtypes: float64(18), int64(27), object(1)
memory usage: 675.4+ KB
```

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 each feature.

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

Out[6]:

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

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.

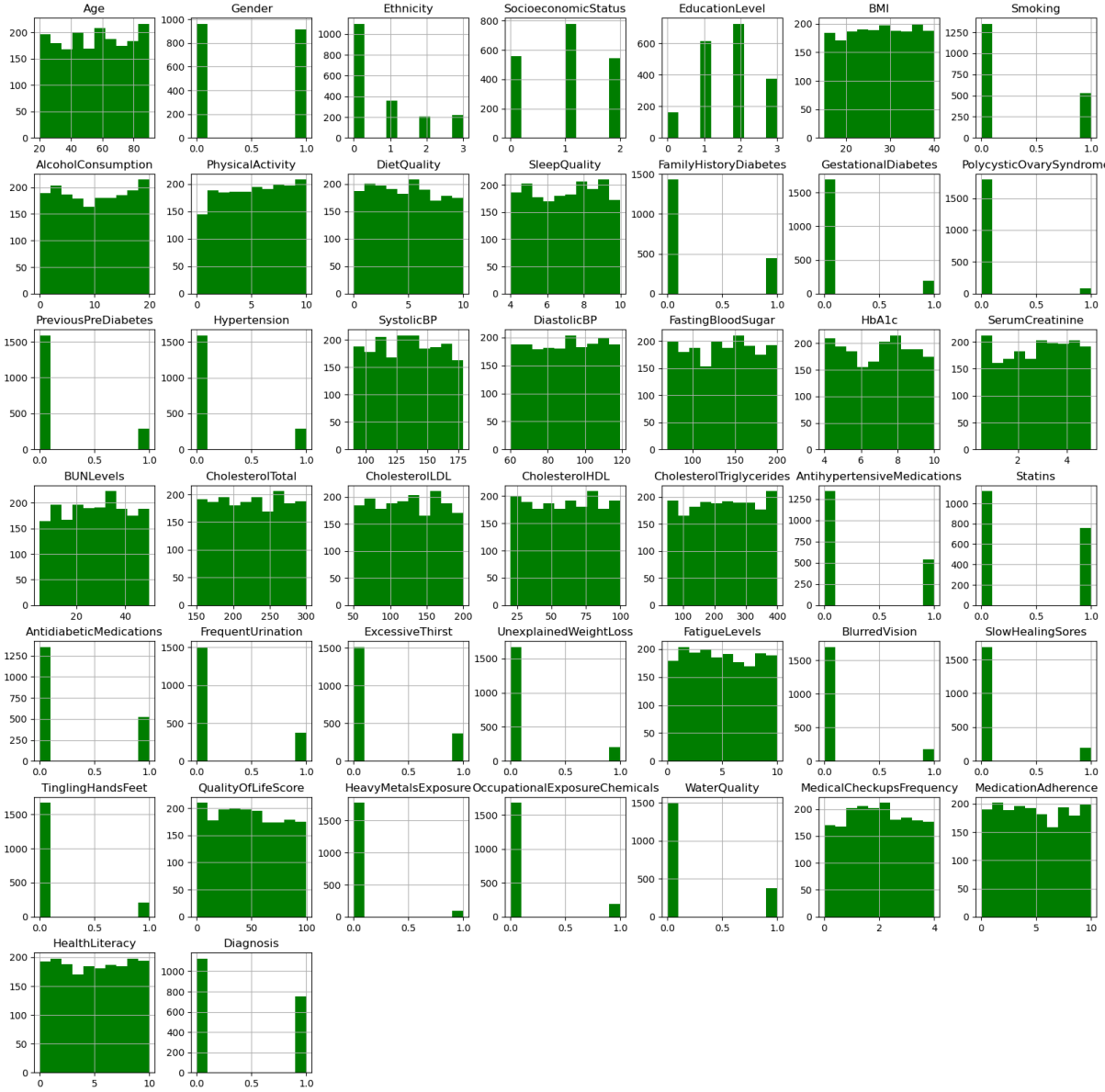
In [10]:
```
selected_features = [
                    'Age',
                    'BMI',
#                     'AlcoholConsumption',
#                     'PhysicalActivity',
```

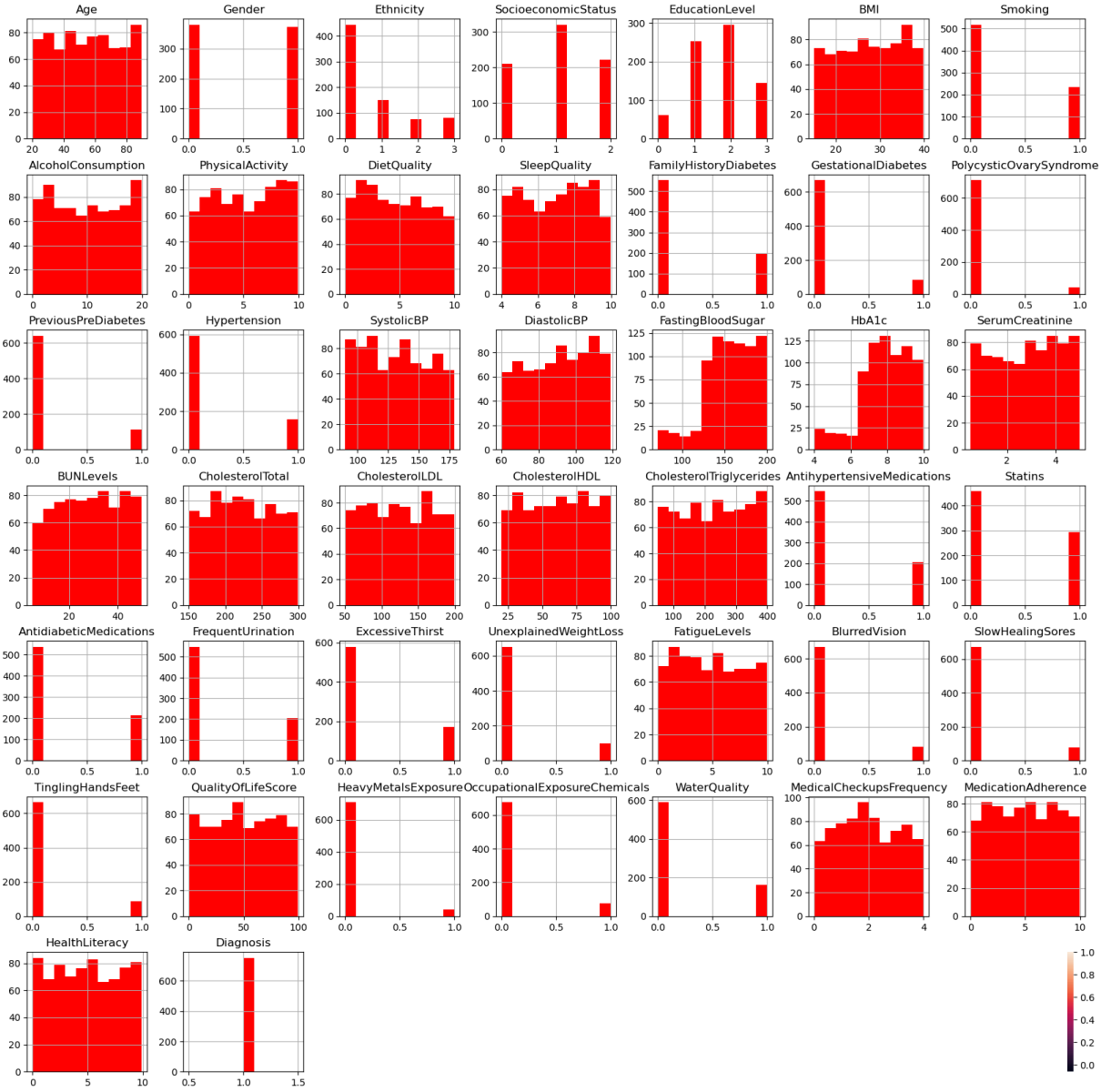
```
#                    'DietQuality',
#                    'SleepQuality',
                   'SystolicBP',
                   'DiastolicBP',
                   'FastingBloodSugar',
                   'HbA1c',
                   'SerumCreatinine',
#                    'BUNLevels',
#                    'CholesterolTotal',
#                    'CholesterolLDL',
#                    'CholesterolHDL',
#                    'CholesterolTriglycerides',
#                    'AntidiabeticMedications',
                    ]
```

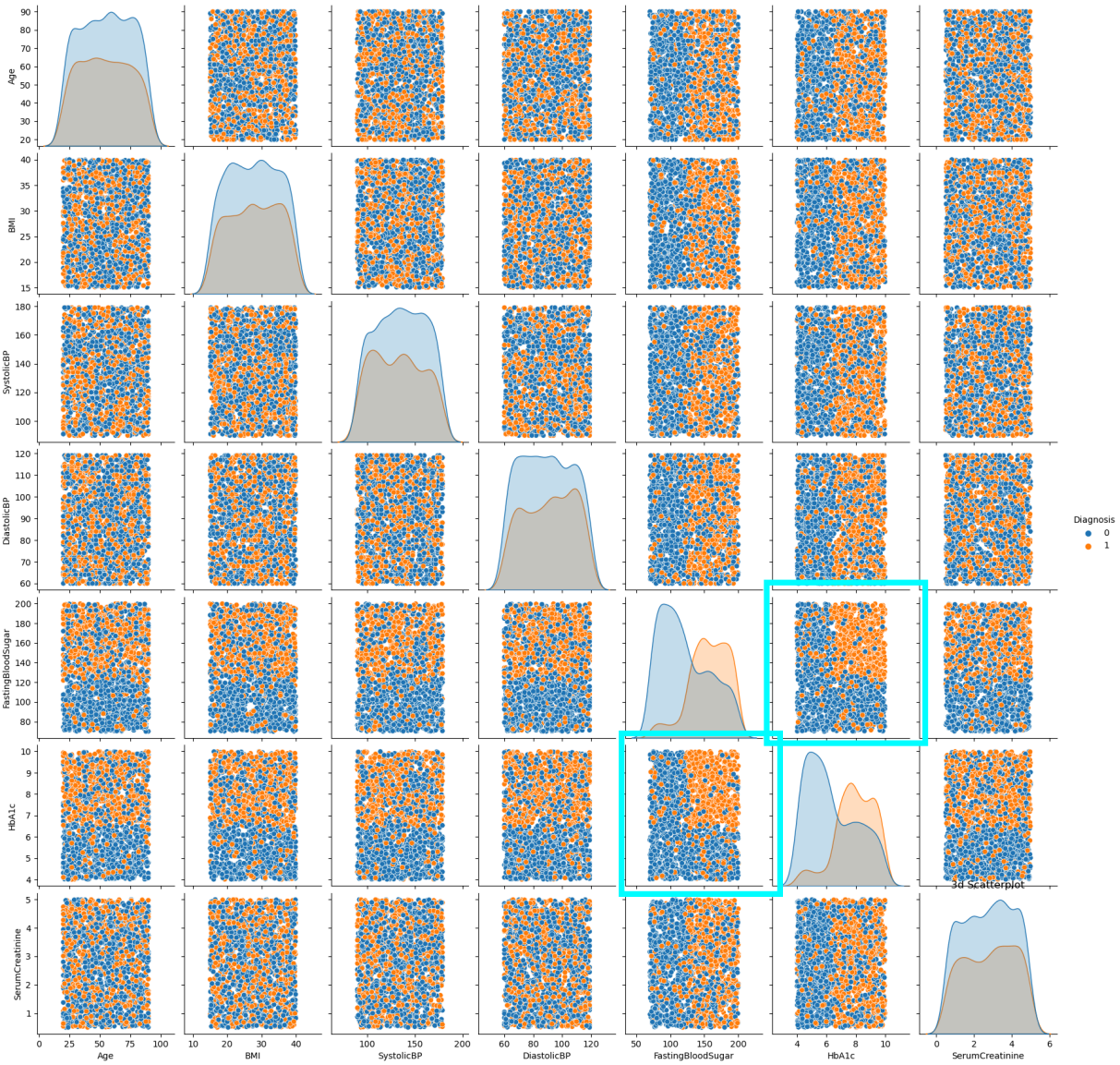
Run this line if you want to visualize the diabetes data in histogram, heatmap, pairplot and 3D scatterplot.

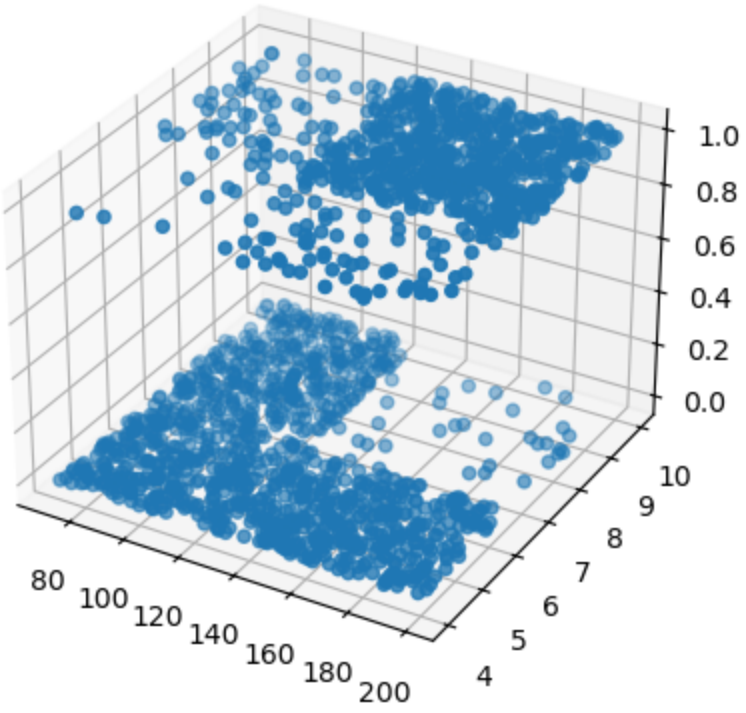
In [11]:
```
visualize_data(diabetes_df, selected_features)
```

```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: Future
Warning: 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: Future
Warning: 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: Future
Warning: 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: Future
Warning: 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: Future
Warning: 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: Future
Warning: 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: Future
Warning: 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):
```


Age
Gender
Ethnicity
SocioeconomicStatus
EducationLevel
BMI
Smoking
AlcoholConsumption
PhysicalActivity
DietQuality
SleepQuality
FamilyHistoryDiabetes
GestationalDiabetes
PolycysticOvarySyndrome
PreviousPreDiabetes
Hypertension
SystolicBP
DiastolicBP
FastingBloodSugar
HbA1c
SerumCreatinine
BUNLevels
CholesterolTotal
CholesterolLDL
CholesterolHDL
CholesterolTriglycerides
AntihypertensiveMedications
Statins
AntidiabeticMedications
FrequentUrination
ExcessiveThirst
UnexplainedWeightLoss
FatigueLevels
BlurredVision
SlowHealingSores
TinglingHandsFeet
QualityOfLifeScore
HeavyMetalsExposure
OccupationalExposureChemicals
WaterQuality
MedicalCheckupsFrequency
MedicationAdherence
HealthLiteracy
Diagnosis


Age
Gender
Ethnicity
SocioeconomicStatus
EducationLevel
BMI
Smoking
AlcoholConsumption
PhysicalActivity
DietQuality
SleepQuality
FamilyHistoryDiabetes
GestationalDiabetes
PolycysticOvarySyndrome
PreviousPreDiabetes
Hypertension
SystolicBP
DiastolicBP
FastingBloodSugar
HbA1c
SerumCreatinine
BUNLevels
CholesterolTotal
CholesterolLDL
CholesterolHDL
CholesterolTriglycerides
AntihypertensiveMedications
Statins
AntidiabeticMedications
FrequentUrination
ExcessiveThirst
UnexplainedWeightLoss
FatigueLevels
BlurredVision
SlowHealingSores
TinglingHandsFeet
QualityOfLifeScore
HeavyMetalsExposure
OccupationalExposureChemicals
WaterQuality
MedicalCheckupsFrequency
MedicationAdherence
HealthLiteracy
Diagnosis


Age
BMI
SystolicBP
DiastolicBP
FastingBloodSugar
HbA1c
SerumCreatinine
Diagnosis
0
1
3d Scatterplot



Feature Engineering

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

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.

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

Out[15]:

	Age	BMI	SystolicBP	DiastolicBP	FastingBloodSugar	HbA1c	SerumC
0	44	32.985284	93	73	163.687162	9.283631	
1	51	39.916764	165	99	188.347070	7.326870	
2	89	19.782251	119	91	127.703653	4.083426	
3	21	32.376881	169	87	82.688415	6.516645	
4	27	16.808600	165	69	90.743395	5.607222	
...	...	...	...	...	...	...	
1874	37	20.811137	104	74	109.832032	5.920723	
1875	80	27.694312	166	115	90.729361	7.332397	
1876	38	35.640824	128	70	149.366801	4.907208	
1877	43	32.423016	124	91	162.027044	8.820613	
1878	85	33.145119	134	86	175.011749	7.814477	

1879 rows × 7 columns

In [16]:
```
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	Ser
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	
1878	0.928571	0.725555	0.494382	0.440678	0.808073	0.636493	

1879 rows × 7 columns

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_scaled, Y)
X_train.shape, y_train.shape, X_cv.shape, y_cv.shape, X_test.shape, y_test.s
```

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 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*

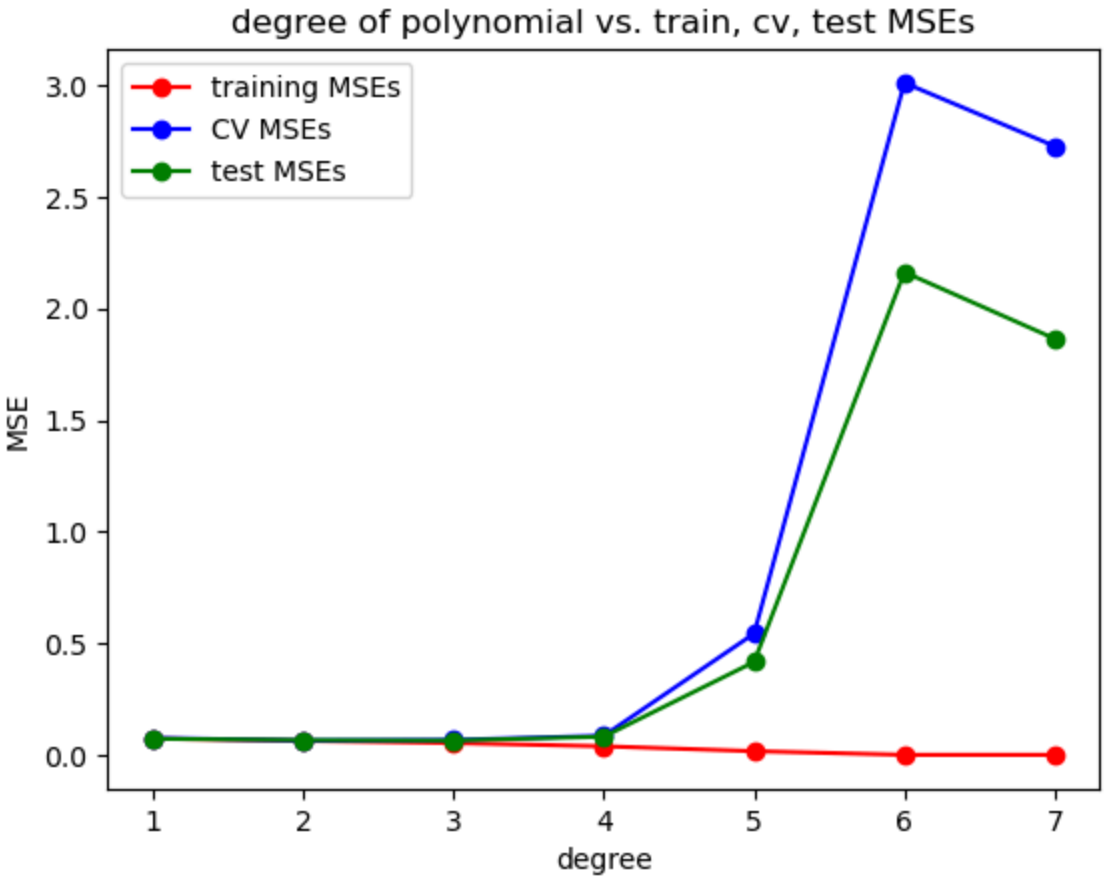
fit() vs transform() vs fit_transform()

- *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)
```

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



Choosing the Polynomial Feature to add to our data

In [21]:
```
# Get the model with the lowest CV MSE (add 1 because list indices start at
# 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} with {cv_m
```

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

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]:
```
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()
```

```
build_models
```

Model: "Model_1"

Layer (type)	Output Shape	Par
dense (Dense)	(None, 14)	
dense_1 (Dense)	(None, 12)	
dense_2 (Dense)	(None, 1)	

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	Par
dense_3 (Dense)	(None, 20)	
dense_4 (Dense)	(None, 12)	
dense_5 (Dense)	(None, 12)	
dense_6 (Dense)	(None, 20)	
dense_7 (Dense)	(None, 1)	

Total params: 1,409 (5.50 KB)
Trainable params: 1,409 (5.50 KB)
Non-trainable params: 0 (0.00 B)

Model: "Model_3"

Layer (type)	Output Shape	Par
dense_8 (Dense)	(None, 32)	1
dense_9 (Dense)	(None, 16)	
dense_10 (Dense)	(None, 8)	
dense_11 (Dense)	(None, 4)	
dense_12 (Dense)	(None, 12)	
dense_13 (Dense)	(None, 1)	

Total params: 1,925 (7.52 KB)
Trainable params: 1,925 (7.52 KB)
Non-trainable params: 0 (0.00 B)

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.

In [23]:
```
histories = []
for model_name, model in models.items():
    print(f"Training with {model_name}")
    model.compile(optimizer='Adam', loss=BinaryCrossentropy(from_logits=Fals
    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 - loss: 0.6922 - m
se: 0.2494 - val_accuracy: 0.6383 - val_loss: 0.6644 - val_mse: 0.2356
Epoch 2/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step - accuracy: 0.6671 - loss: 0.6580 -
mse: 0.2324 - val_accuracy: 0.7074 - val_loss: 0.6315 - val_mse: 0.2195
Epoch 3/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 915us/step - accuracy: 0.7272 - loss: 0.6232 -
mse: 0.2155 - val_accuracy: 0.7473 - val_loss: 0.5970 - val_mse: 0.2030
Epoch 4/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step - accuracy: 0.7537 - loss: 0.5878 -
mse: 0.1987 - val_accuracy: 0.7500 - val_loss: 0.5624 - val_mse: 0.1873
Epoch 5/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - accuracy: 0.7749 - loss: 0.5518 -
mse: 0.1825 - val_accuracy: 0.7500 - val_loss: 0.5303 - val_mse: 0.1740
Epoch 6/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 699us/step - accuracy: 0.7847 - loss: 0.5187 -
mse: 0.1685 - val_accuracy: 0.7633 - val_loss: 0.5033 - val_mse: 0.1640
Epoch 7/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - accuracy: 0.7995 - loss: 0.4894 -
mse: 0.1569 - val_accuracy: 0.7500 - val_loss: 0.4825 - val_mse: 0.1570
Epoch 8/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 676us/step - accuracy: 0.8049 - loss: 0.4659 -
mse: 0.1480 - val_accuracy: 0.7580 - val_loss: 0.4668 - val_mse: 0.1519
Epoch 9/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - accuracy: 0.8209 - loss: 0.4483 -
mse: 0.1415 - val_accuracy: 0.7766 - val_loss: 0.4549 - val_mse: 0.1478
Epoch 10/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - accuracy: 0.8274 - loss: 0.4349 -
mse: 0.1365 - val_accuracy: 0.7846 - val_loss: 0.4458 - val_mse: 0.1445
Epoch 11/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.8207 - loss: 0.4246 -
mse: 0.1327 - val_accuracy: 0.7952 - val_loss: 0.4391 - val_mse: 0.1421
Epoch 12/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - accuracy: 0.8218 - loss: 0.4171 -
mse: 0.1300 - val_accuracy: 0.8005 - val_loss: 0.4339 - val_mse: 0.1402
Epoch 13/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 745us/step - accuracy: 0.8197 - loss: 0.4117 -
mse: 0.1281 - val_accuracy: 0.8059 - val_loss: 0.4299 - val_mse: 0.1386
Epoch 14/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - accuracy: 0.8215 - loss: 0.4074 -
mse: 0.1265 - val_accuracy: 0.8085 - val_loss: 0.4264 - val_mse: 0.1372
Epoch 15/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - accuracy: 0.8308 - loss: 0.4039 -
mse: 0.1252 - val_accuracy: 0.8085 - val_loss: 0.4235 - val_mse: 0.1360
Epoch 16/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - accuracy: 0.8311 - loss: 0.4011 -
mse: 0.1242 - val_accuracy: 0.8112 - val_loss: 0.4206 - val_mse: 0.1348
Epoch 17/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - accuracy: 0.8330 - loss: 0.3987 -
mse: 0.1233 - val_accuracy: 0.8112 - val_loss: 0.4184 - val_mse: 0.1338
Epoch 18/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.8357 - loss: 0.3964 -
mse: 0.1224 - val_accuracy: 0.8112 - val_loss: 0.4162 - val_mse: 0.1328
Epoch 19/100
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - accuracy: 0.8427 - loss: 0.3946 -
mse: 0.1216 - val_accuracy: 0.8191 - val_loss: 0.4138 - val_mse: 0.1317
Epoch 20/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - accuracy: 0.8459 - loss: 0.3930 -
mse: 0.1209 - val_accuracy: 0.8191 - val_loss: 0.4118 - val_mse: 0.1307
Epoch 21/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - accuracy: 0.8459 - loss: 0.3916 -
mse: 0.1203 - val_accuracy: 0.8245 - val_loss: 0.4099 - val_mse: 0.1299
Epoch 22/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - accuracy: 0.8483 - loss: 0.3903 -
mse: 0.1197 - val_accuracy: 0.8298 - val_loss: 0.4082 - val_mse: 0.1290
Epoch 23/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - accuracy: 0.8484 - loss: 0.3890 -
mse: 0.1191 - val_accuracy: 0.8298 - val_loss: 0.4068 - val_mse: 0.1283
Epoch 24/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - accuracy: 0.8480 - loss: 0.3877 -
mse: 0.1186 - val_accuracy: 0.8298 - val_loss: 0.4054 - val_mse: 0.1277
Epoch 25/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - accuracy: 0.8480 - loss: 0.3866 -
mse: 0.1181 - val_accuracy: 0.8324 - val_loss: 0.4042 - val_mse: 0.1271
Epoch 26/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - accuracy: 0.8535 - loss: 0.3855 -
mse: 0.1176 - val_accuracy: 0.8324 - val_loss: 0.4028 - val_mse: 0.1264
Epoch 27/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - accuracy: 0.8573 - loss: 0.3844 -
mse: 0.1171 - val_accuracy: 0.8324 - val_loss: 0.4017 - val_mse: 0.1258
Epoch 28/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - accuracy: 0.8578 - loss: 0.3833 -
mse: 0.1166 - val_accuracy: 0.8324 - val_loss: 0.4005 - val_mse: 0.1252
Epoch 29/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 662us/step - accuracy: 0.8581 - loss: 0.3822 -
mse: 0.1161 - val_accuracy: 0.8324 - val_loss: 0.3996 - val_mse: 0.1247
Epoch 30/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8580 - loss: 0.3814 -
mse: 0.1158 - val_accuracy: 0.8324 - val_loss: 0.3985 - val_mse: 0.1241
Epoch 31/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.8617 - loss: 0.3803 -
mse: 0.1153 - val_accuracy: 0.8351 - val_loss: 0.3975 - val_mse: 0.1236
Epoch 32/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - accuracy: 0.8617 - loss: 0.3793 -
mse: 0.1148 - val_accuracy: 0.8351 - val_loss: 0.3968 - val_mse: 0.1232
Epoch 33/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - accuracy: 0.8617 - loss: 0.3785 -
mse: 0.1145 - val_accuracy: 0.8324 - val_loss: 0.3960 - val_mse: 0.1227
Epoch 34/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - accuracy: 0.8621 - loss: 0.3774 -
mse: 0.1140 - val_accuracy: 0.8324 - val_loss: 0.3953 - val_mse: 0.1223
Epoch 35/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - accuracy: 0.8632 - loss: 0.3769 -
mse: 0.1138 - val_accuracy: 0.8324 - val_loss: 0.3946 - val_mse: 0.1219
Epoch 36/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - accuracy: 0.8650 - loss: 0.3761 -
mse: 0.1134 - val_accuracy: 0.8324 - val_loss: 0.3939 - val_mse: 0.1216
Epoch 37/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - accuracy: 0.8636 - loss: 0.3752 -
mse: 0.1130 - val_accuracy: 0.8324 - val_loss: 0.3932 - val_mse: 0.1211
```

```
Epoch 38/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - accuracy: 0.8638 - loss: 0.3745 -
mse: 0.1127 - val_accuracy: 0.8324 - val_loss: 0.3925 - val_mse: 0.1207
Epoch 39/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - accuracy: 0.8632 - loss: 0.3738 -
mse: 0.1123 - val_accuracy: 0.8351 - val_loss: 0.3920 - val_mse: 0.1205
Epoch 40/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 611us/step - accuracy: 0.8632 - loss: 0.3732 -
mse: 0.1121 - val_accuracy: 0.8378 - val_loss: 0.3914 - val_mse: 0.1201
Epoch 41/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - accuracy: 0.8632 - loss: 0.3723 -
mse: 0.1118 - val_accuracy: 0.8378 - val_loss: 0.3908 - val_mse: 0.1198
Epoch 42/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - accuracy: 0.8657 - loss: 0.3717 -
mse: 0.1115 - val_accuracy: 0.8378 - val_loss: 0.3904 - val_mse: 0.1195
Epoch 43/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - accuracy: 0.8657 - loss: 0.3710 -
mse: 0.1111 - val_accuracy: 0.8351 - val_loss: 0.3899 - val_mse: 0.1192
Epoch 44/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - accuracy: 0.8657 - loss: 0.3704 -
mse: 0.1109 - val_accuracy: 0.8378 - val_loss: 0.3894 - val_mse: 0.1190
Epoch 45/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 588us/step - accuracy: 0.8657 - loss: 0.3698 -
mse: 0.1106 - val_accuracy: 0.8378 - val_loss: 0.3888 - val_mse: 0.1186
Epoch 46/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - accuracy: 0.8657 - loss: 0.3691 -
mse: 0.1103 - val_accuracy: 0.8404 - val_loss: 0.3885 - val_mse: 0.1184
Epoch 47/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.8657 - loss: 0.3684 -
mse: 0.1100 - val_accuracy: 0.8404 - val_loss: 0.3882 - val_mse: 0.1182
Epoch 48/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - accuracy: 0.8672 - loss: 0.3678 -
mse: 0.1097 - val_accuracy: 0.8404 - val_loss: 0.3882 - val_mse: 0.1182
Epoch 49/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - accuracy: 0.8650 - loss: 0.3673 -
mse: 0.1096 - val_accuracy: 0.8404 - val_loss: 0.3879 - val_mse: 0.1180
Epoch 50/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - accuracy: 0.8690 - loss: 0.3667 -
mse: 0.1093 - val_accuracy: 0.8404 - val_loss: 0.3878 - val_mse: 0.1178
Epoch 51/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 598us/step - accuracy: 0.8671 - loss: 0.3664 -
mse: 0.1092 - val_accuracy: 0.8404 - val_loss: 0.3877 - val_mse: 0.1177
Epoch 52/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 594us/step - accuracy: 0.8682 - loss: 0.3657 -
mse: 0.1089 - val_accuracy: 0.8404 - val_loss: 0.3874 - val_mse: 0.1175
Epoch 53/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 589us/step - accuracy: 0.8682 - loss: 0.3652 -
mse: 0.1087 - val_accuracy: 0.8431 - val_loss: 0.3874 - val_mse: 0.1175
Epoch 54/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 580us/step - accuracy: 0.8684 - loss: 0.3651 -
mse: 0.1087 - val_accuracy: 0.8431 - val_loss: 0.3870 - val_mse: 0.1173
Epoch 55/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 579us/step - accuracy: 0.8687 - loss: 0.3643 -
mse: 0.1084 - val_accuracy: 0.8431 - val_loss: 0.3870 - val_mse: 0.1172
Epoch 56/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 565us/step - accuracy: 0.8701 - loss: 0.3638 -
```

```
mse: 0.1081 - val_accuracy: 0.8431 - val_loss: 0.3871 - val_mse: 0.1172
Epoch 57/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 568us/step - accuracy: 0.8701 - loss: 0.3635 -
mse: 0.1081 - val_accuracy: 0.8431 - val_loss: 0.3868 - val_mse: 0.1170
Epoch 58/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 573us/step - accuracy: 0.8669 - loss: 0.3632 -
mse: 0.1080 - val_accuracy: 0.8457 - val_loss: 0.3864 - val_mse: 0.1168
Epoch 59/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - accuracy: 0.8665 - loss: 0.3627 -
mse: 0.1077 - val_accuracy: 0.8457 - val_loss: 0.3863 - val_mse: 0.1167
Epoch 60/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 605us/step - accuracy: 0.8677 - loss: 0.3622 -
mse: 0.1075 - val_accuracy: 0.8457 - val_loss: 0.3861 - val_mse: 0.1166
Epoch 61/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 574us/step - accuracy: 0.8691 - loss: 0.3617 -
mse: 0.1074 - val_accuracy: 0.8431 - val_loss: 0.3860 - val_mse: 0.1165
Epoch 62/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 576us/step - accuracy: 0.8691 - loss: 0.3613 -
mse: 0.1072 - val_accuracy: 0.8431 - val_loss: 0.3858 - val_mse: 0.1163
Epoch 63/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 568us/step - accuracy: 0.8698 - loss: 0.3607 -
mse: 0.1069 - val_accuracy: 0.8457 - val_loss: 0.3859 - val_mse: 0.1163
Epoch 64/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - accuracy: 0.8698 - loss: 0.3602 -
mse: 0.1067 - val_accuracy: 0.8431 - val_loss: 0.3860 - val_mse: 0.1164
Epoch 65/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 600us/step - accuracy: 0.8711 - loss: 0.3601 -
mse: 0.1068 - val_accuracy: 0.8457 - val_loss: 0.3858 - val_mse: 0.1162
Epoch 66/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 590us/step - accuracy: 0.8707 - loss: 0.3596 -
mse: 0.1065 - val_accuracy: 0.8457 - val_loss: 0.3859 - val_mse: 0.1162
Epoch 67/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 604us/step - accuracy: 0.8689 - loss: 0.3593 -
mse: 0.1065 - val_accuracy: 0.8457 - val_loss: 0.3857 - val_mse: 0.1160
Epoch 68/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - accuracy: 0.8684 - loss: 0.3588 -
mse: 0.1062 - val_accuracy: 0.8457 - val_loss: 0.3858 - val_mse: 0.1161
Epoch 69/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - accuracy: 0.8689 - loss: 0.3584 -
mse: 0.1061 - val_accuracy: 0.8457 - val_loss: 0.3857 - val_mse: 0.1160
Epoch 70/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 607us/step - accuracy: 0.8689 - loss: 0.3579 -
mse: 0.1060 - val_accuracy: 0.8457 - val_loss: 0.3858 - val_mse: 0.1161
Epoch 71/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - accuracy: 0.8689 - loss: 0.3577 -
mse: 0.1059 - val_accuracy: 0.8457 - val_loss: 0.3853 - val_mse: 0.1158
Epoch 72/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 595us/step - accuracy: 0.8689 - loss: 0.3573 -
mse: 0.1057 - val_accuracy: 0.8457 - val_loss: 0.3855 - val_mse: 0.1159
Epoch 73/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - accuracy: 0.8689 - loss: 0.3569 -
mse: 0.1057 - val_accuracy: 0.8431 - val_loss: 0.3856 - val_mse: 0.1159
Epoch 74/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 592us/step - accuracy: 0.8689 - loss: 0.3565 -
mse: 0.1056 - val_accuracy: 0.8431 - val_loss: 0.3855 - val_mse: 0.1158
Epoch 75/100
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 606us/step - accuracy: 0.8699 - loss: 0.3563 -
mse: 0.1055 - val_accuracy: 0.8457 - val_loss: 0.3856 - val_mse: 0.1158
Epoch 76/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - accuracy: 0.8687 - loss: 0.3556 -
mse: 0.1052 - val_accuracy: 0.8457 - val_loss: 0.3859 - val_mse: 0.1160
Epoch 77/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - accuracy: 0.8687 - loss: 0.3553 -
mse: 0.1053 - val_accuracy: 0.8457 - val_loss: 0.3858 - val_mse: 0.1159
Epoch 78/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 612us/step - accuracy: 0.8687 - loss: 0.3551 -
mse: 0.1052 - val_accuracy: 0.8431 - val_loss: 0.3856 - val_mse: 0.1158
Epoch 79/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 603us/step - accuracy: 0.8687 - loss: 0.3545 -
mse: 0.1050 - val_accuracy: 0.8431 - val_loss: 0.3856 - val_mse: 0.1157
Epoch 80/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - accuracy: 0.8687 - loss: 0.3544 -
mse: 0.1049 - val_accuracy: 0.8431 - val_loss: 0.3859 - val_mse: 0.1159
Epoch 81/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 606us/step - accuracy: 0.8683 - loss: 0.3540 -
mse: 0.1048 - val_accuracy: 0.8431 - val_loss: 0.3860 - val_mse: 0.1160
Epoch 82/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - accuracy: 0.8687 - loss: 0.3540 -
mse: 0.1049 - val_accuracy: 0.8431 - val_loss: 0.3858 - val_mse: 0.1157
Epoch 83/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 612us/step - accuracy: 0.8684 - loss: 0.3532 -
mse: 0.1046 - val_accuracy: 0.8404 - val_loss: 0.3861 - val_mse: 0.1160
Epoch 84/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - accuracy: 0.8689 - loss: 0.3533 -
mse: 0.1047 - val_accuracy: 0.8431 - val_loss: 0.3859 - val_mse: 0.1158
Epoch 85/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 587us/step - accuracy: 0.8684 - loss: 0.3523 -
mse: 0.1043 - val_accuracy: 0.8404 - val_loss: 0.3861 - val_mse: 0.1158
Epoch 86/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 692us/step - accuracy: 0.8689 - loss: 0.3526 -
mse: 0.1044 - val_accuracy: 0.8404 - val_loss: 0.3860 - val_mse: 0.1157
Epoch 87/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - accuracy: 0.8684 - loss: 0.3515 -
mse: 0.1040 - val_accuracy: 0.8378 - val_loss: 0.3864 - val_mse: 0.1159
Epoch 88/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 593us/step - accuracy: 0.8688 - loss: 0.3516 -
mse: 0.1041 - val_accuracy: 0.8431 - val_loss: 0.3861 - val_mse: 0.1157
Epoch 89/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - accuracy: 0.8676 - loss: 0.3507 -
mse: 0.1038 - val_accuracy: 0.8378 - val_loss: 0.3862 - val_mse: 0.1158
Epoch 90/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - accuracy: 0.8678 - loss: 0.3504 -
mse: 0.1037 - val_accuracy: 0.8431 - val_loss: 0.3860 - val_mse: 0.1157
Epoch 91/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 599us/step - accuracy: 0.8726 - loss: 0.3499 -
mse: 0.1036 - val_accuracy: 0.8431 - val_loss: 0.3862 - val_mse: 0.1157
Epoch 92/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 582us/step - accuracy: 0.8726 - loss: 0.3499 -
mse: 0.1036 - val_accuracy: 0.8431 - val_loss: 0.3864 - val_mse: 0.1157
Epoch 93/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 569us/step - accuracy: 0.8718 - loss: 0.3492 -
mse: 0.1034 - val_accuracy: 0.8431 - val_loss: 0.3865 - val_mse: 0.1157
```

```
Epoch 94/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 599us/step - accuracy: 0.8712 - loss: 0.3488 -
mse: 0.1033 - val_accuracy: 0.8457 - val_loss: 0.3866 - val_mse: 0.1157
Epoch 95/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 589us/step - accuracy: 0.8718 - loss: 0.3482 -
mse: 0.1031 - val_accuracy: 0.8457 - val_loss: 0.3865 - val_mse: 0.1156
Epoch 96/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 593us/step - accuracy: 0.8718 - loss: 0.3481 -
mse: 0.1030 - val_accuracy: 0.8457 - val_loss: 0.3870 - val_mse: 0.1158
Epoch 97/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 594us/step - accuracy: 0.8729 - loss: 0.3474 -
mse: 0.1029 - val_accuracy: 0.8457 - val_loss: 0.3870 - val_mse: 0.1157
Epoch 98/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 573us/step - accuracy: 0.8740 - loss: 0.3471 -
mse: 0.1027 - val_accuracy: 0.8457 - val_loss: 0.3871 - val_mse: 0.1158
Epoch 99/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 570us/step - accuracy: 0.8729 - loss: 0.3461 -
mse: 0.1025 - val_accuracy: 0.8457 - val_loss: 0.3874 - val_mse: 0.1159
Epoch 100/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 569us/step - accuracy: 0.8743 - loss: 0.3460 -
mse: 0.1024 - val_accuracy: 0.8457 - val_loss: 0.3872 - val_mse: 0.1159
Training with Model_2
Epoch 1/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5281 - loss: 0.6908 - m
se: 0.2485 - val_accuracy: 0.6356 - val_loss: 0.6805 - val_mse: 0.2433
Epoch 2/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6541 - loss: 0.6737 - m
se: 0.2400 - val_accuracy: 0.6968 - val_loss: 0.6481 - val_mse: 0.2273
Epoch 3/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7355 - loss: 0.6303 - m
se: 0.2186 - val_accuracy: 0.7340 - val_loss: 0.5837 - val_mse: 0.1966
Epoch 4/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 989us/step - accuracy: 0.7938 - loss: 0.5584 -
mse: 0.1849 - val_accuracy: 0.7606 - val_loss: 0.5106 - val_mse: 0.1660
Epoch 5/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - accuracy: 0.8173 - loss: 0.4845 -
mse: 0.1536 - val_accuracy: 0.7952 - val_loss: 0.4578 - val_mse: 0.1469
Epoch 6/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 700us/step - accuracy: 0.8344 - loss: 0.4383 -
mse: 0.1350 - val_accuracy: 0.8085 - val_loss: 0.4342 - val_mse: 0.1385
Epoch 7/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - accuracy: 0.8419 - loss: 0.4171 -
mse: 0.1265 - val_accuracy: 0.8085 - val_loss: 0.4200 - val_mse: 0.1334
Epoch 8/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - accuracy: 0.8467 - loss: 0.4063 -
mse: 0.1222 - val_accuracy: 0.8165 - val_loss: 0.4106 - val_mse: 0.1300
Epoch 9/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - accuracy: 0.8501 - loss: 0.3996 -
mse: 0.1197 - val_accuracy: 0.8165 - val_loss: 0.4052 - val_mse: 0.1281
Epoch 10/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 670us/step - accuracy: 0.8516 - loss: 0.3952 -
mse: 0.1182 - val_accuracy: 0.8218 - val_loss: 0.4008 - val_mse: 0.1261
Epoch 11/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - accuracy: 0.8507 - loss: 0.3906 -
mse: 0.1167 - val_accuracy: 0.8245 - val_loss: 0.3980 - val_mse: 0.1249
Epoch 12/100
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - accuracy: 0.8520 - loss: 0.3881 -
mse: 0.1160 - val_accuracy: 0.8271 - val_loss: 0.3959 - val_mse: 0.1238
Epoch 13/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - accuracy: 0.8528 - loss: 0.3850 -
mse: 0.1153 - val_accuracy: 0.8298 - val_loss: 0.3931 - val_mse: 0.1225
Epoch 14/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8540 - loss: 0.3820 -
mse: 0.1144 - val_accuracy: 0.8324 - val_loss: 0.3911 - val_mse: 0.1215
Epoch 15/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8518 - loss: 0.3784 -
mse: 0.1134 - val_accuracy: 0.8378 - val_loss: 0.3861 - val_mse: 0.1194
Epoch 16/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - accuracy: 0.8530 - loss: 0.3741 -
mse: 0.1119 - val_accuracy: 0.8404 - val_loss: 0.3820 - val_mse: 0.1177
Epoch 17/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 670us/step - accuracy: 0.8548 - loss: 0.3702 -
mse: 0.1105 - val_accuracy: 0.8404 - val_loss: 0.3779 - val_mse: 0.1160
Epoch 18/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 650us/step - accuracy: 0.8585 - loss: 0.3673 -
mse: 0.1095 - val_accuracy: 0.8484 - val_loss: 0.3728 - val_mse: 0.1138
Epoch 19/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - accuracy: 0.8526 - loss: 0.3633 -
mse: 0.1080 - val_accuracy: 0.8590 - val_loss: 0.3683 - val_mse: 0.1117
Epoch 20/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8571 - loss: 0.3594 -
mse: 0.1064 - val_accuracy: 0.8590 - val_loss: 0.3647 - val_mse: 0.1101
Epoch 21/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - accuracy: 0.8562 - loss: 0.3566 -
mse: 0.1054 - val_accuracy: 0.8590 - val_loss: 0.3618 - val_mse: 0.1087
Epoch 22/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8562 - loss: 0.3529 -
mse: 0.1041 - val_accuracy: 0.8617 - val_loss: 0.3599 - val_mse: 0.1078
Epoch 23/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - accuracy: 0.8550 - loss: 0.3504 -
mse: 0.1031 - val_accuracy: 0.8590 - val_loss: 0.3591 - val_mse: 0.1076
Epoch 24/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.8584 - loss: 0.3489 -
mse: 0.1027 - val_accuracy: 0.8537 - val_loss: 0.3587 - val_mse: 0.1073
Epoch 25/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.8609 - loss: 0.3469 -
mse: 0.1020 - val_accuracy: 0.8564 - val_loss: 0.3577 - val_mse: 0.1069
Epoch 26/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - accuracy: 0.8627 - loss: 0.3445 -
mse: 0.1012 - val_accuracy: 0.8590 - val_loss: 0.3567 - val_mse: 0.1064
Epoch 27/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - accuracy: 0.8627 - loss: 0.3433 -
mse: 0.1009 - val_accuracy: 0.8590 - val_loss: 0.3563 - val_mse: 0.1062
Epoch 28/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - accuracy: 0.8648 - loss: 0.3410 -
mse: 0.1001 - val_accuracy: 0.8590 - val_loss: 0.3556 - val_mse: 0.1060
Epoch 29/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 678us/step - accuracy: 0.8652 - loss: 0.3392 -
mse: 0.0995 - val_accuracy: 0.8590 - val_loss: 0.3548 - val_mse: 0.1057
Epoch 30/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 631us/step - accuracy: 0.8703 - loss: 0.3380 -
mse: 0.0991 - val_accuracy: 0.8590 - val_loss: 0.3543 - val_mse: 0.1055
```

```
Epoch 31/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 646us/step - accuracy: 0.8687 - loss: 0.3367 -
mse: 0.0987 - val_accuracy: 0.8590 - val_loss: 0.3532 - val_mse: 0.1051
Epoch 32/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8680 - loss: 0.3357 -
mse: 0.0985 - val_accuracy: 0.8590 - val_loss: 0.3530 - val_mse: 0.1050
Epoch 33/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 626us/step - accuracy: 0.8737 - loss: 0.3340 -
mse: 0.0980 - val_accuracy: 0.8617 - val_loss: 0.3522 - val_mse: 0.1048
Epoch 34/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - accuracy: 0.8714 - loss: 0.3330 -
mse: 0.0977 - val_accuracy: 0.8590 - val_loss: 0.3522 - val_mse: 0.1049
Epoch 35/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8728 - loss: 0.3318 -
mse: 0.0974 - val_accuracy: 0.8590 - val_loss: 0.3514 - val_mse: 0.1045
Epoch 36/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - accuracy: 0.8733 - loss: 0.3305 -
mse: 0.0970 - val_accuracy: 0.8644 - val_loss: 0.3505 - val_mse: 0.1042
Epoch 37/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - accuracy: 0.8732 - loss: 0.3280 -
mse: 0.0960 - val_accuracy: 0.8697 - val_loss: 0.3501 - val_mse: 0.1040
Epoch 38/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - accuracy: 0.8742 - loss: 0.3260 -
mse: 0.0954 - val_accuracy: 0.8697 - val_loss: 0.3498 - val_mse: 0.1039
Epoch 39/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - accuracy: 0.8796 - loss: 0.3234 -
mse: 0.0945 - val_accuracy: 0.8670 - val_loss: 0.3497 - val_mse: 0.1039
Epoch 40/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - accuracy: 0.8793 - loss: 0.3225 -
mse: 0.0944 - val_accuracy: 0.8670 - val_loss: 0.3493 - val_mse: 0.1038
Epoch 41/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 615us/step - accuracy: 0.8807 - loss: 0.3208 -
mse: 0.0939 - val_accuracy: 0.8644 - val_loss: 0.3486 - val_mse: 0.1035
Epoch 42/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - accuracy: 0.8805 - loss: 0.3195 -
mse: 0.0935 - val_accuracy: 0.8644 - val_loss: 0.3479 - val_mse: 0.1032
Epoch 43/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - accuracy: 0.8813 - loss: 0.3169 -
mse: 0.0926 - val_accuracy: 0.8617 - val_loss: 0.3474 - val_mse: 0.1029
Epoch 44/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - accuracy: 0.8845 - loss: 0.3153 -
mse: 0.0920 - val_accuracy: 0.8644 - val_loss: 0.3471 - val_mse: 0.1028
Epoch 45/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 652us/step - accuracy: 0.8819 - loss: 0.3141 -
mse: 0.0917 - val_accuracy: 0.8617 - val_loss: 0.3475 - val_mse: 0.1029
Epoch 46/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - accuracy: 0.8834 - loss: 0.3125 -
mse: 0.0912 - val_accuracy: 0.8670 - val_loss: 0.3482 - val_mse: 0.1032
Epoch 47/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - accuracy: 0.8834 - loss: 0.3112 -
mse: 0.0909 - val_accuracy: 0.8670 - val_loss: 0.3490 - val_mse: 0.1035
Epoch 48/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - accuracy: 0.8837 - loss: 0.3102 -
mse: 0.0907 - val_accuracy: 0.8670 - val_loss: 0.3484 - val_mse: 0.1032
Epoch 49/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - accuracy: 0.8823 - loss: 0.3090 -
```

```
mse: 0.0904 - val_accuracy: 0.8670 - val_loss: 0.3484 - val_mse: 0.1032
Epoch 50/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.8828 - loss: 0.3081 -
mse: 0.0901 - val_accuracy: 0.8697 - val_loss: 0.3488 - val_mse: 0.1034
Epoch 51/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - accuracy: 0.8828 - loss: 0.3082 -
mse: 0.0903 - val_accuracy: 0.8644 - val_loss: 0.3501 - val_mse: 0.1038
Epoch 52/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.8824 - loss: 0.3063 -
mse: 0.0898 - val_accuracy: 0.8670 - val_loss: 0.3495 - val_mse: 0.1036
Epoch 53/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - accuracy: 0.8824 - loss: 0.3056 -
mse: 0.0897 - val_accuracy: 0.8670 - val_loss: 0.3493 - val_mse: 0.1034
Epoch 54/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.8814 - loss: 0.3040 -
mse: 0.0891 - val_accuracy: 0.8670 - val_loss: 0.3493 - val_mse: 0.1033
Epoch 55/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - accuracy: 0.8814 - loss: 0.3020 -
mse: 0.0885 - val_accuracy: 0.8670 - val_loss: 0.3493 - val_mse: 0.1033
Epoch 56/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8814 - loss: 0.3014 -
mse: 0.0883 - val_accuracy: 0.8670 - val_loss: 0.3491 - val_mse: 0.1032
Epoch 57/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - accuracy: 0.8835 - loss: 0.3001 -
mse: 0.0880 - val_accuracy: 0.8670 - val_loss: 0.3492 - val_mse: 0.1031
Epoch 58/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - accuracy: 0.8835 - loss: 0.2990 -
mse: 0.0876 - val_accuracy: 0.8670 - val_loss: 0.3499 - val_mse: 0.1034
Epoch 59/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - accuracy: 0.8872 - loss: 0.2980 -
mse: 0.0874 - val_accuracy: 0.8670 - val_loss: 0.3506 - val_mse: 0.1036
Epoch 60/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 671us/step - accuracy: 0.8890 - loss: 0.2963 -
mse: 0.0868 - val_accuracy: 0.8670 - val_loss: 0.3496 - val_mse: 0.1032
Epoch 61/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - accuracy: 0.8895 - loss: 0.2956 -
mse: 0.0866 - val_accuracy: 0.8670 - val_loss: 0.3499 - val_mse: 0.1033
Epoch 62/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - accuracy: 0.8878 - loss: 0.2950 -
mse: 0.0866 - val_accuracy: 0.8670 - val_loss: 0.3500 - val_mse: 0.1033
Epoch 63/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - accuracy: 0.8879 - loss: 0.2945 -
mse: 0.0864 - val_accuracy: 0.8670 - val_loss: 0.3508 - val_mse: 0.1036
Epoch 64/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 650us/step - accuracy: 0.8878 - loss: 0.2935 -
mse: 0.0863 - val_accuracy: 0.8670 - val_loss: 0.3520 - val_mse: 0.1039
Epoch 65/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - accuracy: 0.8885 - loss: 0.2923 -
mse: 0.0859 - val_accuracy: 0.8697 - val_loss: 0.3519 - val_mse: 0.1038
Epoch 66/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - accuracy: 0.8913 - loss: 0.2909 -
mse: 0.0853 - val_accuracy: 0.8697 - val_loss: 0.3520 - val_mse: 0.1038
Epoch 67/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 652us/step - accuracy: 0.8888 - loss: 0.2902 -
mse: 0.0852 - val_accuracy: 0.8697 - val_loss: 0.3523 - val_mse: 0.1038
Epoch 68/100
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - accuracy: 0.8930 - loss: 0.2887 -
mse: 0.0847 - val_accuracy: 0.8697 - val_loss: 0.3524 - val_mse: 0.1038
Epoch 69/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 652us/step - accuracy: 0.8921 - loss: 0.2882 -
mse: 0.0846 - val_accuracy: 0.8697 - val_loss: 0.3527 - val_mse: 0.1039
Epoch 70/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - accuracy: 0.8917 - loss: 0.2869 -
mse: 0.0843 - val_accuracy: 0.8697 - val_loss: 0.3528 - val_mse: 0.1038
Epoch 71/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 631us/step - accuracy: 0.8926 - loss: 0.2856 -
mse: 0.0838 - val_accuracy: 0.8697 - val_loss: 0.3542 - val_mse: 0.1043
Epoch 72/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - accuracy: 0.8917 - loss: 0.2852 -
mse: 0.0838 - val_accuracy: 0.8697 - val_loss: 0.3545 - val_mse: 0.1043
Epoch 73/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - accuracy: 0.8917 - loss: 0.2845 -
mse: 0.0836 - val_accuracy: 0.8670 - val_loss: 0.3544 - val_mse: 0.1042
Epoch 74/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - accuracy: 0.8917 - loss: 0.2827 -
mse: 0.0831 - val_accuracy: 0.8670 - val_loss: 0.3547 - val_mse: 0.1042
Epoch 75/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - accuracy: 0.8926 - loss: 0.2810 -
mse: 0.0826 - val_accuracy: 0.8644 - val_loss: 0.3535 - val_mse: 0.1038
Epoch 76/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - accuracy: 0.8926 - loss: 0.2802 -
mse: 0.0822 - val_accuracy: 0.8644 - val_loss: 0.3555 - val_mse: 0.1043
Epoch 77/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 620us/step - accuracy: 0.8926 - loss: 0.2783 -
mse: 0.0817 - val_accuracy: 0.8644 - val_loss: 0.3544 - val_mse: 0.1038
Epoch 78/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - accuracy: 0.8926 - loss: 0.2766 -
mse: 0.0811 - val_accuracy: 0.8644 - val_loss: 0.3554 - val_mse: 0.1041
Epoch 79/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - accuracy: 0.8928 - loss: 0.2754 -
mse: 0.0808 - val_accuracy: 0.8617 - val_loss: 0.3560 - val_mse: 0.1042
Epoch 80/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 604us/step - accuracy: 0.8912 - loss: 0.2742 -
mse: 0.0805 - val_accuracy: 0.8617 - val_loss: 0.3577 - val_mse: 0.1048
Epoch 81/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - accuracy: 0.8928 - loss: 0.2723 -
mse: 0.0799 - val_accuracy: 0.8644 - val_loss: 0.3577 - val_mse: 0.1047
Epoch 82/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - accuracy: 0.8960 - loss: 0.2704 -
mse: 0.0793 - val_accuracy: 0.8644 - val_loss: 0.3578 - val_mse: 0.1046
Epoch 83/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 646us/step - accuracy: 0.8946 - loss: 0.2688 -
mse: 0.0788 - val_accuracy: 0.8644 - val_loss: 0.3572 - val_mse: 0.1043
Epoch 84/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - accuracy: 0.8946 - loss: 0.2677 -
mse: 0.0783 - val_accuracy: 0.8617 - val_loss: 0.3588 - val_mse: 0.1048
Epoch 85/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - accuracy: 0.8950 - loss: 0.2664 -
mse: 0.0781 - val_accuracy: 0.8617 - val_loss: 0.3597 - val_mse: 0.1052
Epoch 86/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - accuracy: 0.8936 - loss: 0.2653 -
mse: 0.0777 - val_accuracy: 0.8644 - val_loss: 0.3601 - val_mse: 0.1051
```

```
Epoch 87/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - accuracy: 0.8936 - loss: 0.2628 -
mse: 0.0769 - val_accuracy: 0.8697 - val_loss: 0.3594 - val_mse: 0.1047
Epoch 88/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - accuracy: 0.8936 - loss: 0.2615 -
mse: 0.0765 - val_accuracy: 0.8670 - val_loss: 0.3611 - val_mse: 0.1051
Epoch 89/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - accuracy: 0.8940 - loss: 0.2606 -
mse: 0.0763 - val_accuracy: 0.8670 - val_loss: 0.3627 - val_mse: 0.1056
Epoch 90/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - accuracy: 0.8941 - loss: 0.2591 -
mse: 0.0759 - val_accuracy: 0.8670 - val_loss: 0.3633 - val_mse: 0.1057
Epoch 91/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - accuracy: 0.8935 - loss: 0.2579 -
mse: 0.0754 - val_accuracy: 0.8697 - val_loss: 0.3636 - val_mse: 0.1057
Epoch 92/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - accuracy: 0.8965 - loss: 0.2559 -
mse: 0.0747 - val_accuracy: 0.8670 - val_loss: 0.3645 - val_mse: 0.1060
Epoch 93/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - accuracy: 0.8936 - loss: 0.2562 -
mse: 0.0750 - val_accuracy: 0.8670 - val_loss: 0.3661 - val_mse: 0.1066
Epoch 94/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.8974 - loss: 0.2542 -
mse: 0.0743 - val_accuracy: 0.8697 - val_loss: 0.3669 - val_mse: 0.1066
Epoch 95/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 664us/step - accuracy: 0.8978 - loss: 0.2519 -
mse: 0.0735 - val_accuracy: 0.8697 - val_loss: 0.3670 - val_mse: 0.1065
Epoch 96/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - accuracy: 0.8983 - loss: 0.2526 -
mse: 0.0739 - val_accuracy: 0.8697 - val_loss: 0.3670 - val_mse: 0.1065
Epoch 97/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - accuracy: 0.8987 - loss: 0.2502 -
mse: 0.0730 - val_accuracy: 0.8723 - val_loss: 0.3671 - val_mse: 0.1063
Epoch 98/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - accuracy: 0.9005 - loss: 0.2482 -
mse: 0.0724 - val_accuracy: 0.8777 - val_loss: 0.3685 - val_mse: 0.1063
Epoch 99/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - accuracy: 0.8995 - loss: 0.2476 -
mse: 0.0723 - val_accuracy: 0.8777 - val_loss: 0.3699 - val_mse: 0.1065
Epoch 100/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.9047 - loss: 0.2456 -
mse: 0.0717 - val_accuracy: 0.8750 - val_loss: 0.3709 - val_mse: 0.1065
Training with Model_3
Epoch 1/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6186 - loss: 0.6771 - m
se: 0.2417 - val_accuracy: 0.6011 - val_loss: 0.6498 - val_mse: 0.2284
Epoch 2/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 986us/step - accuracy: 0.6186 - loss: 0.6387 -
mse: 0.2231 - val_accuracy: 0.6011 - val_loss: 0.6118 - val_mse: 0.2107
Epoch 3/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6186 - loss: 0.5967 - m
se: 0.2041 - val_accuracy: 0.6011 - val_loss: 0.5769 - val_mse: 0.1959
Epoch 4/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 934us/step - accuracy: 0.6227 - loss: 0.5584 -
mse: 0.1879 - val_accuracy: 0.7686 - val_loss: 0.5521 - val_mse: 0.1860
Epoch 5/100
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 743us/step - accuracy: 0.7844 - loss: 0.5330 -
mse: 0.1773 - val_accuracy: 0.8191 - val_loss: 0.5387 - val_mse: 0.1811
Epoch 6/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 711us/step - accuracy: 0.8305 - loss: 0.5203 -
mse: 0.1722 - val_accuracy: 0.8324 - val_loss: 0.5279 - val_mse: 0.1767
Epoch 7/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 697us/step - accuracy: 0.8491 - loss: 0.5117 -
mse: 0.1686 - val_accuracy: 0.8431 - val_loss: 0.5142 - val_mse: 0.1709
Epoch 8/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 756us/step - accuracy: 0.8498 - loss: 0.5008 -
mse: 0.1639 - val_accuracy: 0.8431 - val_loss: 0.5025 - val_mse: 0.1658
Epoch 9/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - accuracy: 0.8601 - loss: 0.4889 -
mse: 0.1586 - val_accuracy: 0.8431 - val_loss: 0.4945 - val_mse: 0.1622
Epoch 10/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - accuracy: 0.8608 - loss: 0.4808 -
mse: 0.1549 - val_accuracy: 0.8484 - val_loss: 0.4890 - val_mse: 0.1596
Epoch 11/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - accuracy: 0.8649 - loss: 0.4754 -
mse: 0.1524 - val_accuracy: 0.8431 - val_loss: 0.4830 - val_mse: 0.1567
Epoch 12/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 715us/step - accuracy: 0.8632 - loss: 0.4699 -
mse: 0.1499 - val_accuracy: 0.8511 - val_loss: 0.4763 - val_mse: 0.1535
Epoch 13/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 693us/step - accuracy: 0.8644 - loss: 0.4631 -
mse: 0.1467 - val_accuracy: 0.8564 - val_loss: 0.4699 - val_mse: 0.1503
Epoch 14/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 719us/step - accuracy: 0.8652 - loss: 0.4565 -
mse: 0.1437 - val_accuracy: 0.8590 - val_loss: 0.4634 - val_mse: 0.1472
Epoch 15/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.8637 - loss: 0.4495 -
mse: 0.1405 - val_accuracy: 0.8617 - val_loss: 0.4577 - val_mse: 0.1445
Epoch 16/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - accuracy: 0.8663 - loss: 0.4435 -
mse: 0.1377 - val_accuracy: 0.8617 - val_loss: 0.4524 - val_mse: 0.1419
Epoch 17/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - accuracy: 0.8680 - loss: 0.4377 -
mse: 0.1351 - val_accuracy: 0.8590 - val_loss: 0.4478 - val_mse: 0.1398
Epoch 18/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - accuracy: 0.8665 - loss: 0.4331 -
mse: 0.1331 - val_accuracy: 0.8564 - val_loss: 0.4421 - val_mse: 0.1372
Epoch 19/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - accuracy: 0.8685 - loss: 0.4269 -
mse: 0.1304 - val_accuracy: 0.8537 - val_loss: 0.4361 - val_mse: 0.1344
Epoch 20/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.8682 - loss: 0.4195 -
mse: 0.1272 - val_accuracy: 0.8511 - val_loss: 0.4269 - val_mse: 0.1305
Epoch 21/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8679 - loss: 0.4083 -
mse: 0.1222 - val_accuracy: 0.8484 - val_loss: 0.4170 - val_mse: 0.1269
Epoch 22/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - accuracy: 0.8548 - loss: 0.3972 -
mse: 0.1179 - val_accuracy: 0.8404 - val_loss: 0.4087 - val_mse: 0.1244
Epoch 23/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - accuracy: 0.8528 - loss: 0.3871 -
mse: 0.1146 - val_accuracy: 0.8404 - val_loss: 0.3992 - val_mse: 0.1221
```

```
Epoch 24/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 650us/step - accuracy: 0.8507 - loss: 0.3744 -
mse: 0.1105 - val_accuracy: 0.8378 - val_loss: 0.3924 - val_mse: 0.1212
Epoch 25/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8503 - loss: 0.3654 -
mse: 0.1078 - val_accuracy: 0.8378 - val_loss: 0.3867 - val_mse: 0.1196
Epoch 26/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - accuracy: 0.8535 - loss: 0.3582 -
mse: 0.1057 - val_accuracy: 0.8404 - val_loss: 0.3830 - val_mse: 0.1183
Epoch 27/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8603 - loss: 0.3515 -
mse: 0.1035 - val_accuracy: 0.8431 - val_loss: 0.3808 - val_mse: 0.1176
Epoch 28/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 650us/step - accuracy: 0.8613 - loss: 0.3472 -
mse: 0.1022 - val_accuracy: 0.8431 - val_loss: 0.3798 - val_mse: 0.1172
Epoch 29/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - accuracy: 0.8634 - loss: 0.3443 -
mse: 0.1012 - val_accuracy: 0.8431 - val_loss: 0.3805 - val_mse: 0.1177
Epoch 30/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.8662 - loss: 0.3425 -
mse: 0.1008 - val_accuracy: 0.8431 - val_loss: 0.3791 - val_mse: 0.1170
Epoch 31/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - accuracy: 0.8640 - loss: 0.3392 -
mse: 0.0998 - val_accuracy: 0.8431 - val_loss: 0.3806 - val_mse: 0.1178
Epoch 32/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - accuracy: 0.8612 - loss: 0.3383 -
mse: 0.0998 - val_accuracy: 0.8431 - val_loss: 0.3805 - val_mse: 0.1178
Epoch 33/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.8619 - loss: 0.3368 -
mse: 0.0995 - val_accuracy: 0.8431 - val_loss: 0.3797 - val_mse: 0.1177
Epoch 34/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 684us/step - accuracy: 0.8634 - loss: 0.3331 -
mse: 0.0983 - val_accuracy: 0.8404 - val_loss: 0.3809 - val_mse: 0.1183
Epoch 35/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 659us/step - accuracy: 0.8666 - loss: 0.3300 -
mse: 0.0971 - val_accuracy: 0.8431 - val_loss: 0.3793 - val_mse: 0.1174
Epoch 36/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - accuracy: 0.8638 - loss: 0.3286 -
mse: 0.0968 - val_accuracy: 0.8431 - val_loss: 0.3797 - val_mse: 0.1175
Epoch 37/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 664us/step - accuracy: 0.8652 - loss: 0.3256 -
mse: 0.0958 - val_accuracy: 0.8431 - val_loss: 0.3811 - val_mse: 0.1179
Epoch 38/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - accuracy: 0.8698 - loss: 0.3246 -
mse: 0.0957 - val_accuracy: 0.8457 - val_loss: 0.3818 - val_mse: 0.1180
Epoch 39/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.8727 - loss: 0.3221 -
mse: 0.0948 - val_accuracy: 0.8484 - val_loss: 0.3826 - val_mse: 0.1182
Epoch 40/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - accuracy: 0.8721 - loss: 0.3201 -
mse: 0.0943 - val_accuracy: 0.8484 - val_loss: 0.3825 - val_mse: 0.1180
Epoch 41/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.8762 - loss: 0.3186 -
mse: 0.0938 - val_accuracy: 0.8484 - val_loss: 0.3839 - val_mse: 0.1184
Epoch 42/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - accuracy: 0.8775 - loss: 0.3177 -
```

```
mse: 0.0937 - val_accuracy: 0.8484 - val_loss: 0.3851 - val_mse: 0.1187
Epoch 43/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - accuracy: 0.8757 - loss: 0.3159 -
mse: 0.0931 - val_accuracy: 0.8484 - val_loss: 0.3868 - val_mse: 0.1192
Epoch 44/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8761 - loss: 0.3159 -
mse: 0.0935 - val_accuracy: 0.8484 - val_loss: 0.3872 - val_mse: 0.1192
Epoch 45/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 656us/step - accuracy: 0.8775 - loss: 0.3147 -
mse: 0.0930 - val_accuracy: 0.8511 - val_loss: 0.3875 - val_mse: 0.1192
Epoch 46/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 670us/step - accuracy: 0.8755 - loss: 0.3136 -
mse: 0.0929 - val_accuracy: 0.8511 - val_loss: 0.3864 - val_mse: 0.1186
Epoch 47/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 675us/step - accuracy: 0.8768 - loss: 0.3110 -
mse: 0.0920 - val_accuracy: 0.8511 - val_loss: 0.3866 - val_mse: 0.1186
Epoch 48/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.8766 - loss: 0.3087 -
mse: 0.0913 - val_accuracy: 0.8511 - val_loss: 0.3863 - val_mse: 0.1181
Epoch 49/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - accuracy: 0.8755 - loss: 0.3069 -
mse: 0.0908 - val_accuracy: 0.8511 - val_loss: 0.3860 - val_mse: 0.1179
Epoch 50/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - accuracy: 0.8789 - loss: 0.3053 -
mse: 0.0902 - val_accuracy: 0.8511 - val_loss: 0.3854 - val_mse: 0.1174
Epoch 51/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - accuracy: 0.8812 - loss: 0.3026 -
mse: 0.0893 - val_accuracy: 0.8484 - val_loss: 0.3867 - val_mse: 0.1176
Epoch 52/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - accuracy: 0.8844 - loss: 0.2998 -
mse: 0.0884 - val_accuracy: 0.8457 - val_loss: 0.3867 - val_mse: 0.1175
Epoch 53/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - accuracy: 0.8810 - loss: 0.2996 -
mse: 0.0884 - val_accuracy: 0.8457 - val_loss: 0.3874 - val_mse: 0.1176
Epoch 54/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - accuracy: 0.8883 - loss: 0.2968 -
mse: 0.0873 - val_accuracy: 0.8484 - val_loss: 0.3871 - val_mse: 0.1171
Epoch 55/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - accuracy: 0.8850 - loss: 0.2957 -
mse: 0.0872 - val_accuracy: 0.8484 - val_loss: 0.3873 - val_mse: 0.1172
Epoch 56/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 646us/step - accuracy: 0.8913 - loss: 0.2943 -
mse: 0.0866 - val_accuracy: 0.8484 - val_loss: 0.3886 - val_mse: 0.1173
Epoch 57/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.8896 - loss: 0.2930 -
mse: 0.0864 - val_accuracy: 0.8511 - val_loss: 0.3887 - val_mse: 0.1171
Epoch 58/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.8901 - loss: 0.2906 -
mse: 0.0854 - val_accuracy: 0.8484 - val_loss: 0.3886 - val_mse: 0.1169
Epoch 59/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8927 - loss: 0.2885 -
mse: 0.0846 - val_accuracy: 0.8537 - val_loss: 0.3883 - val_mse: 0.1161
Epoch 60/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - accuracy: 0.8947 - loss: 0.2874 -
mse: 0.0844 - val_accuracy: 0.8537 - val_loss: 0.3884 - val_mse: 0.1162
Epoch 61/100
```

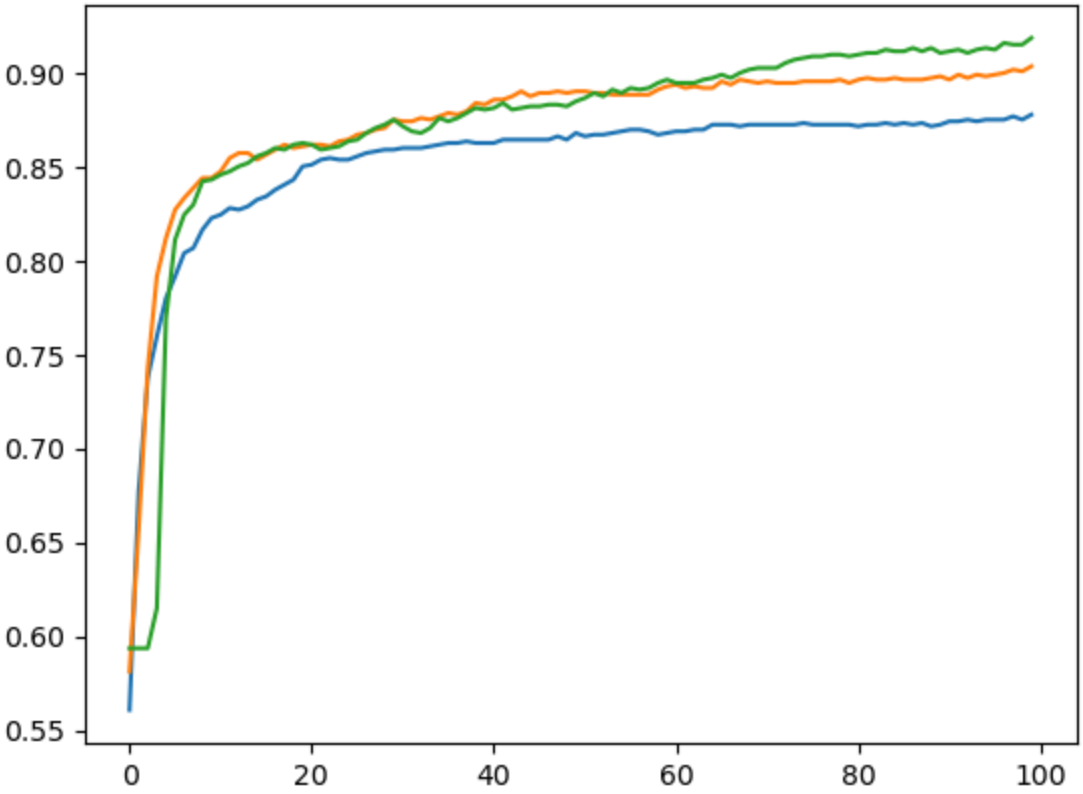
```
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - accuracy: 0.8934 - loss: 0.2841 -
mse: 0.0831 - val_accuracy: 0.8537 - val_loss: 0.3898 - val_mse: 0.1165
Epoch 62/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 620us/step - accuracy: 0.8927 - loss: 0.2833 -
mse: 0.0829 - val_accuracy: 0.8590 - val_loss: 0.3901 - val_mse: 0.1164
Epoch 63/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - accuracy: 0.8912 - loss: 0.2824 -
mse: 0.0826 - val_accuracy: 0.8564 - val_loss: 0.3918 - val_mse: 0.1164
Epoch 64/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - accuracy: 0.8949 - loss: 0.2798 -
mse: 0.0818 - val_accuracy: 0.8564 - val_loss: 0.3913 - val_mse: 0.1164
Epoch 65/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - accuracy: 0.8946 - loss: 0.2771 -
mse: 0.0809 - val_accuracy: 0.8590 - val_loss: 0.3909 - val_mse: 0.1155
Epoch 66/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - accuracy: 0.8962 - loss: 0.2757 -
mse: 0.0806 - val_accuracy: 0.8564 - val_loss: 0.3919 - val_mse: 0.1160
Epoch 67/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - accuracy: 0.8948 - loss: 0.2744 -
mse: 0.0801 - val_accuracy: 0.8590 - val_loss: 0.3938 - val_mse: 0.1164
Epoch 68/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 659us/step - accuracy: 0.8972 - loss: 0.2728 -
mse: 0.0798 - val_accuracy: 0.8564 - val_loss: 0.3936 - val_mse: 0.1157
Epoch 69/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - accuracy: 0.8971 - loss: 0.2703 -
mse: 0.0790 - val_accuracy: 0.8590 - val_loss: 0.3938 - val_mse: 0.1158
Epoch 70/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8972 - loss: 0.2691 -
mse: 0.0786 - val_accuracy: 0.8590 - val_loss: 0.3951 - val_mse: 0.1154
Epoch 71/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - accuracy: 0.8973 - loss: 0.2682 -
mse: 0.0785 - val_accuracy: 0.8590 - val_loss: 0.3951 - val_mse: 0.1159
Epoch 72/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 631us/step - accuracy: 0.8973 - loss: 0.2663 -
mse: 0.0777 - val_accuracy: 0.8564 - val_loss: 0.3966 - val_mse: 0.1158
Epoch 73/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.9010 - loss: 0.2628 -
mse: 0.0765 - val_accuracy: 0.8564 - val_loss: 0.3982 - val_mse: 0.1158
Epoch 74/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.9030 - loss: 0.2641 -
mse: 0.0771 - val_accuracy: 0.8537 - val_loss: 0.3992 - val_mse: 0.1162
Epoch 75/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - accuracy: 0.9020 - loss: 0.2596 -
mse: 0.0755 - val_accuracy: 0.8511 - val_loss: 0.3994 - val_mse: 0.1156
Epoch 76/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - accuracy: 0.9033 - loss: 0.2581 -
mse: 0.0752 - val_accuracy: 0.8511 - val_loss: 0.3988 - val_mse: 0.1159
Epoch 77/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - accuracy: 0.9041 - loss: 0.2570 -
mse: 0.0746 - val_accuracy: 0.8511 - val_loss: 0.4022 - val_mse: 0.1160
Epoch 78/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - accuracy: 0.9044 - loss: 0.2553 -
mse: 0.0743 - val_accuracy: 0.8511 - val_loss: 0.4024 - val_mse: 0.1165
Epoch 79/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - accuracy: 0.9055 - loss: 0.2545 -
mse: 0.0740 - val_accuracy: 0.8511 - val_loss: 0.4061 - val_mse: 0.1165
```

```
Epoch 80/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - accuracy: 0.9032 - loss: 0.2532 -
mse: 0.0738 - val_accuracy: 0.8511 - val_loss: 0.4055 - val_mse: 0.1165
Epoch 81/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 674us/step - accuracy: 0.9055 - loss: 0.2505 -
mse: 0.0729 - val_accuracy: 0.8511 - val_loss: 0.4048 - val_mse: 0.1167
Epoch 82/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - accuracy: 0.9059 - loss: 0.2489 -
mse: 0.0724 - val_accuracy: 0.8511 - val_loss: 0.4076 - val_mse: 0.1167
Epoch 83/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - accuracy: 0.9059 - loss: 0.2484 -
mse: 0.0724 - val_accuracy: 0.8511 - val_loss: 0.4102 - val_mse: 0.1167
Epoch 84/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.9086 - loss: 0.2473 -
mse: 0.0722 - val_accuracy: 0.8484 - val_loss: 0.4141 - val_mse: 0.1174
Epoch 85/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - accuracy: 0.9066 - loss: 0.2462 -
mse: 0.0719 - val_accuracy: 0.8484 - val_loss: 0.4132 - val_mse: 0.1176
Epoch 86/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.9074 - loss: 0.2447 -
mse: 0.0713 - val_accuracy: 0.8590 - val_loss: 0.4139 - val_mse: 0.1167
Epoch 87/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - accuracy: 0.9081 - loss: 0.2433 -
mse: 0.0712 - val_accuracy: 0.8564 - val_loss: 0.4173 - val_mse: 0.1182
Epoch 88/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 656us/step - accuracy: 0.9088 - loss: 0.2407 -
mse: 0.0702 - val_accuracy: 0.8590 - val_loss: 0.4204 - val_mse: 0.1184
Epoch 89/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 664us/step - accuracy: 0.9095 - loss: 0.2386 -
mse: 0.0696 - val_accuracy: 0.8617 - val_loss: 0.4176 - val_mse: 0.1166
Epoch 90/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - accuracy: 0.9066 - loss: 0.2384 -
mse: 0.0695 - val_accuracy: 0.8617 - val_loss: 0.4225 - val_mse: 0.1179
Epoch 91/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - accuracy: 0.9082 - loss: 0.2365 -
mse: 0.0691 - val_accuracy: 0.8590 - val_loss: 0.4217 - val_mse: 0.1189
Epoch 92/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - accuracy: 0.9083 - loss: 0.2343 -
mse: 0.0682 - val_accuracy: 0.8617 - val_loss: 0.4253 - val_mse: 0.1180
Epoch 93/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - accuracy: 0.9071 - loss: 0.2333 -
mse: 0.0680 - val_accuracy: 0.8590 - val_loss: 0.4271 - val_mse: 0.1188
Epoch 94/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 656us/step - accuracy: 0.9088 - loss: 0.2308 -
mse: 0.0672 - val_accuracy: 0.8590 - val_loss: 0.4240 - val_mse: 0.1184
Epoch 95/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.9101 - loss: 0.2281 -
mse: 0.0662 - val_accuracy: 0.8590 - val_loss: 0.4301 - val_mse: 0.1182
Epoch 96/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - accuracy: 0.9094 - loss: 0.2275 -
mse: 0.0660 - val_accuracy: 0.8617 - val_loss: 0.4292 - val_mse: 0.1175
Epoch 97/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - accuracy: 0.9131 - loss: 0.2248 -
mse: 0.0653 - val_accuracy: 0.8564 - val_loss: 0.4302 - val_mse: 0.1184
Epoch 98/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - accuracy: 0.9102 - loss: 0.2241 -
```

```
mse: 0.0649 - val_accuracy: 0.8617 - val_loss: 0.4386 - val_mse: 0.1190
Epoch 99/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - accuracy: 0.9107 - loss: 0.2260 -
mse: 0.0656 - val_accuracy: 0.8670 - val_loss: 0.4381 - val_mse: 0.1189
Epoch 100/100
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - accuracy: 0.9134 - loss: 0.2221 -
mse: 0.0645 - val_accuracy: 0.8590 - val_loss: 0.4366 - val_mse: 0.1183
```

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

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



VII. Train with pretrained models

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

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
```

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:.2f}%")
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.

HOW

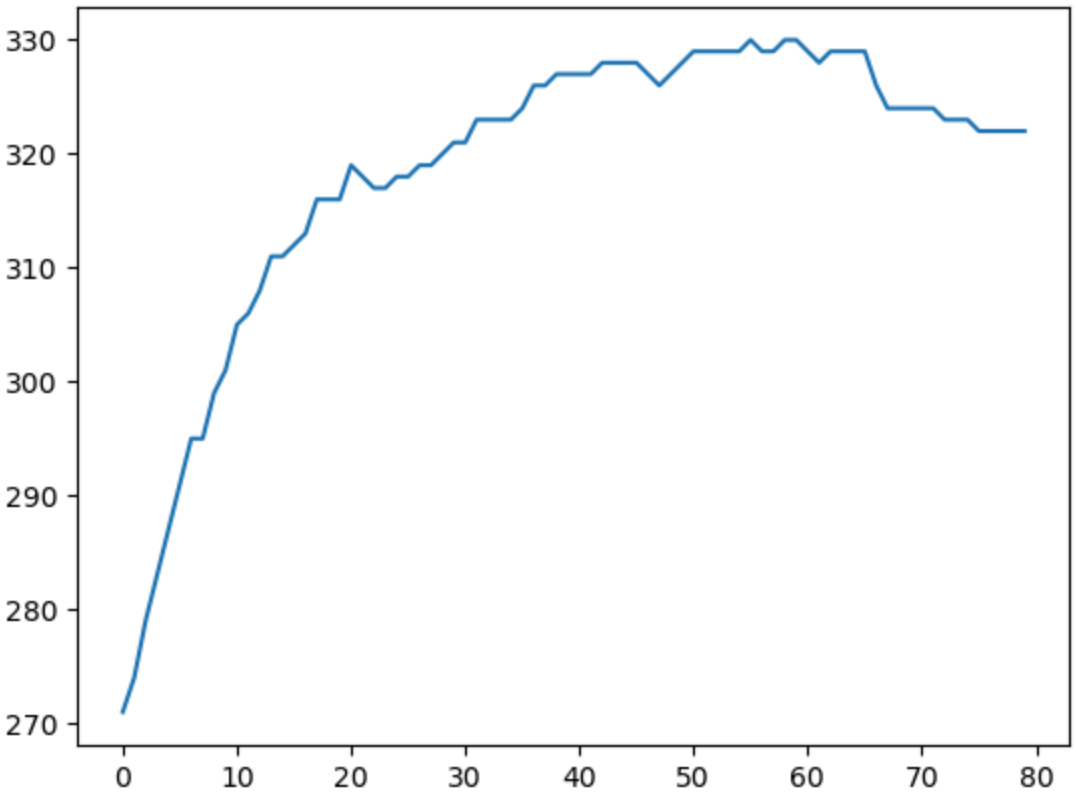
In [31]:
```
val, max_correct, y_pred, result = search_boundary(model, X_test, y_test)

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

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

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

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

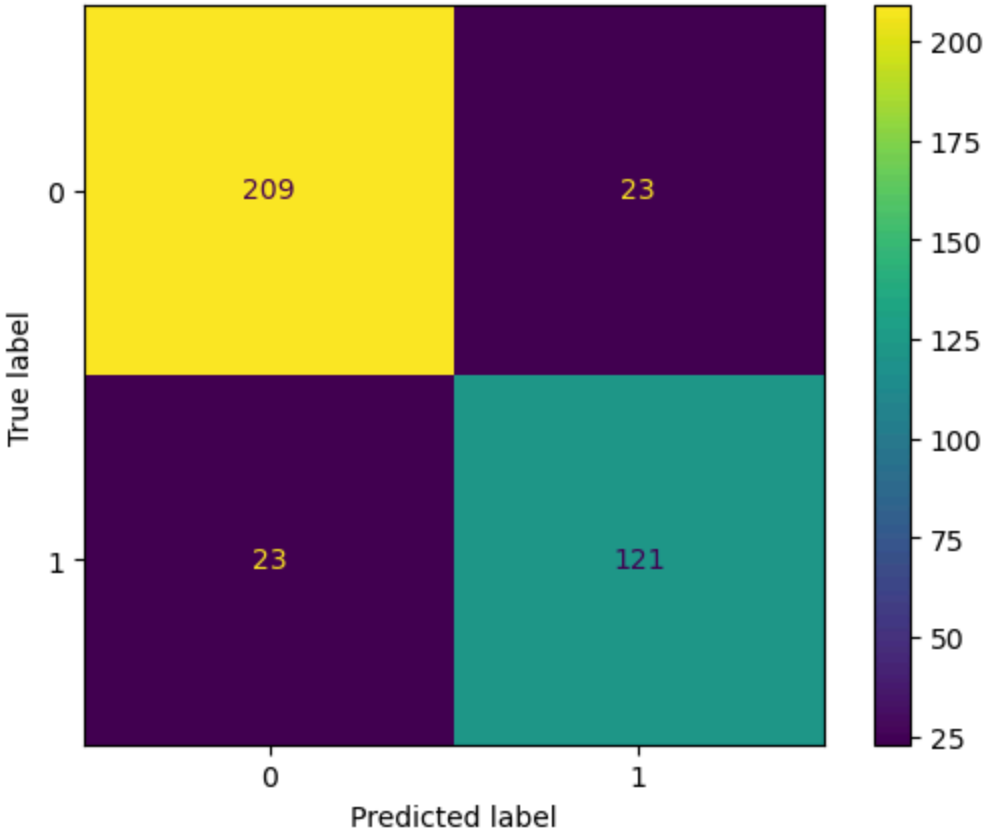


In [33]:
```
y_predict = [1 if x >= val else 0 for x in y_pred]
```

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:.2f}%")

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.
- Apply existing models and evaluate their performance.