| |
|---|
| Experiment No.6 |
| Predict disease risk from patient data |
| Date of Performance: |
| Date of Submission: |

Aim: To predict disease risk from patient data
Objective: Develop an artificial intelligence-based predictive model to assess the risk of Type 2 diabetes in adults using a dataset of patient demographics, medical history, and lifestyle factors, with the aim of early intervention and personalized preventive healthcare strategies.
Theory:
Predicting disease risk from patient data using artificial intelligence (AI) in healthcare involves the use of machine learning and data analysis techniques to analyze patient information and identify potential health risks. Here's a general framework for building a disease risk prediction model:

Data Collection:
Gather a comprehensive dataset that includes patient information relevant to the disease you want to predict. This data may include medical history, family history, demographics, lifestyle factors, clinical tests, and genetic information.

Data Preprocessing:
Clean and preprocess the data to handle missing values, outliers, and ensure it's in a suitable format for analysis. This may involve data normalization, feature engineering, and data encoding.

Feature Selection:
Identify which features (variables) are most relevant for predicting the disease. Feature selection helps in reducing noise in the data and improving model performance.

Data Splitting:
Split the dataset into training, validation, and test sets. The training set is used to train the AI model, the validation set is used for

hyperparameter tuning, and the test set is used to evaluate the model's performance.

Model Selection:
Choose an appropriate machine learning model for disease risk prediction. Common models include logistic regression, decision trees, random forests, support vector machines, and deep learning models (e.g., neural networks).

Model Training:
Train the selected model on the training data. The model learns to recognize patterns and relationships within the data.

Hyperparameter Tuning:
Optimize the model's hyperparameters to improve its performance. Techniques like grid search or random search can be used for this purpose.

Model Evaluation:
Evaluate the model's performance using the validation set. Common evaluation metrics for classification tasks include accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC).

Testing and Validation:
Assess the model's performance on the test set, which represents new, unseen data. This step helps determine how well the model generalizes to real-world cases.

Deployment:
If the model performs well, it can be deployed in a healthcare setting, such as a hospital or clinic, where it can analyze patient data and provide risk predictions in real-time.

Continuous Monitoring and Improvement:
Regularly update the model and its data to account for changes in patient profiles, medical guidelines, and the availability of new data. Monitoring the model's performance and retraining as needed is crucial for maintaining accuracy.

Ethical Considerations and Privacy:
Be mindful of ethical considerations and patient privacy when handling medical data. Compliance with relevant regulations and obtaining patient consent is essential.

Interpretability and Explainability:
Ensure that the AI model is interpretable and can provide explanations for its predictions, especially in the healthcare domain, where transparency is critical.

The choice of the specific AI algorithms and techniques will depend on the disease being predicted, the available data, and the goals of the prediction. Also, it's important to involve healthcare professionals and domain experts throughout the process to ensure that the predictions align with clinical knowledge and best practices.

Code: -

```
# Importing necessary libraries
```

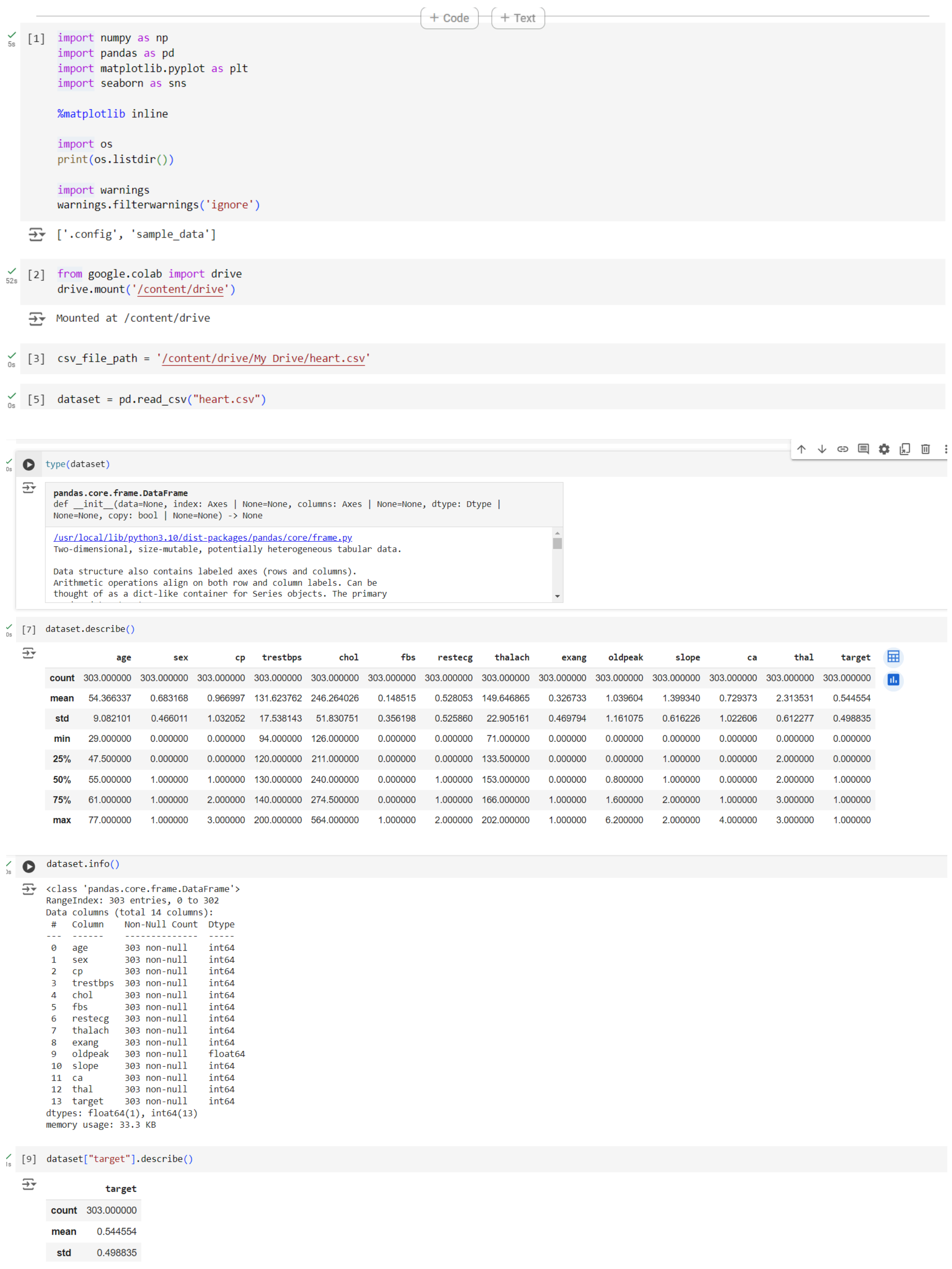
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Load the dataset
csv_file_path = '/content/drive/My Drive/heart.csv'
dataset = pd.read_csv(csv_file_path)

# Dataset Information
print(dataset.describe())
```

```
dataset.info()

# Target Distribution
print("Target Unique Values:", dataset["target"].unique())
print("Correlation with Target:")
print(dataset.corr()["target"].abs().sort_values(ascending=False))

# Plotting target distribution
sns.countplot(dataset["target"])
plt.show()

# Check percentage of patients with and without heart problems
target_temp = dataset["target"].value_counts()
print("Percentage of patients without heart problems: ", round(target_temp[0]*100/len(dataset), 2))
print("Percentage of patients with heart problems: ", round(target_temp[1]*100/len(dataset), 2))

# Bar plot for 'cp' (chest pain) and 'thal' against target
sns.barplot(x=dataset["cp"], y=dataset["target"])
plt.show()

sns.barplot(x=dataset["thal"], y=dataset["target"])
```

```
plt.show()

# Splitting dataset into predictors and target
predictors = dataset.drop("target", axis=1)
target = dataset["target"]

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(predictors, target,
test_size=0.20, random_state=0)

# Shapes of training and testing data
print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)

# Logistic Regression Model
lr = LogisticRegression()
lr.fit(X_train, Y_train)
Y_pred_lr = lr.predict(X_test)
score_lr = round(accuracy_score(Y_pred_lr, Y_test) * 100, 2)
print("Accuracy using Logistic Regression: ", score_lr, "%")

# Naive Bayes Model
nb = GaussianNB()
```

```
nb.fit(X_train, Y_train)
Y_pred_nb = nb.predict(X_test)
score_nb = round(accuracy_score(Y_pred_nb, Y_test) * 100, 2)
print("Accuracy using Naive Bayes: ", score_nb, "%")

# Neural Network Model
model = Sequential()
model.add(Dense(11, activation='relu', input_dim=13))    # Input
features = 13
model.add(Dense(1, activation='sigmoid'))  # Binary classification

# Compile the model
model.compile(loss='binary_crossentropy',              optimizer='adam',
metrics=['accuracy'])

# Train the model
model.fit(X_train, Y_train, epochs=100, batch_size=32, verbose=1)

# Evaluate Neural Network on test data
loss, accuracy = model.evaluate(X_test, Y_test)
print(f"Accuracy using Neural Network: {accuracy * 100:.2f} %")
```

Output:



```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns

    %matplotlib inline

    import os
    print(os.listdir())

    import warnings
    warnings.filterwarnings('ignore')
```

```
['.config', 'sample_data']
```

```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] csv_file_path = '/content/drive/My Drive/heart.csv'
```

```
[5] dataset = pd.read_csv("heart.csv")
```

```
type(dataset)
```

```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dtype |
None=None, copy: bool | None=None) -> None

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
```

```
[7] dataset.describe()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| mean | 54.366337 | 0.683168 | 0.966997 | 131.623762 | 246.264026 | 0.148515 | 0.528053 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 | 2.313531 | 0.544554 |
| std | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 | 0.612277 | 0.498835 |
| min | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.000000 | 0.000000 |
| 50% | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 | 2.000000 | 1.000000 |
| 75% | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 |
| max | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 | 3.000000 | 1.000000 |

```
dataset.info()
```

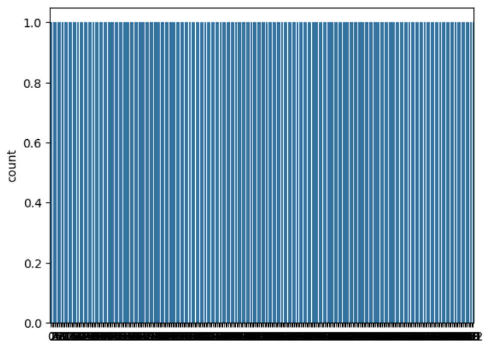
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
[9] dataset["target"].describe()
```

| | target |
|---|---|
| count | 303.000000 |
| mean | 0.544554 |
| std | 0.498835 |

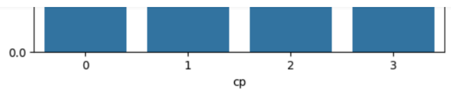
```
[10] dataset["target"].unique()

array([1, 0])

[ ] Start coding or generate with AI.

[11] print(dataset.corr()["target"].abs().sort_values(ascending=False))

target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: target, dtype: float64

[12] y = dataset["target"]

     sns.countplot(y)
```

```
[12] print(target_temp)

target
1    165
0    138
Name: count, dtype: int64
```

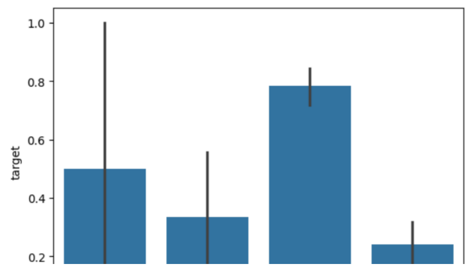


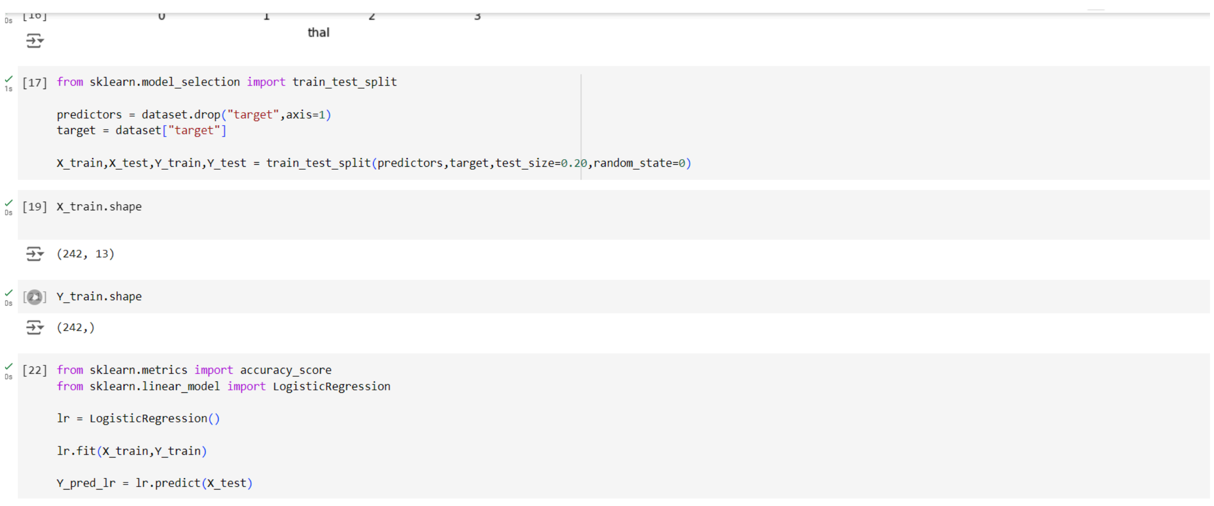
```
[13] print("Percentage of patience without heart problems: "+str(round(target_temp[0]*100/303,2)))
     print("Percentage of patience with heart problems: "+str(round(target_temp[1]*100/303,2)))
```

```
[14]
```

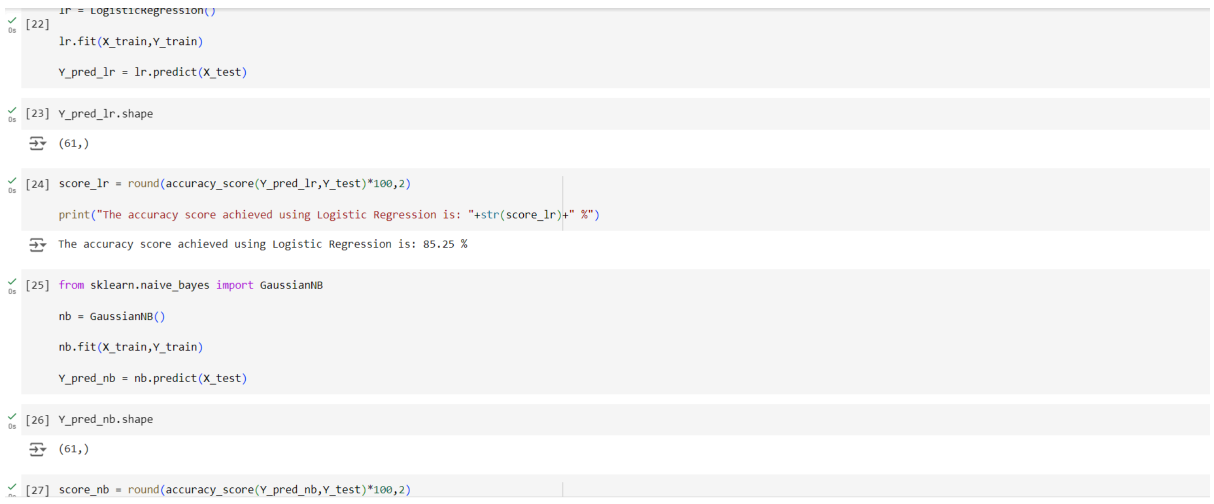


```
[15] dataset["thal"].unique()

array([1, 2, 3, 0])

[16] sns.barplot(x=dataset["thal"], y=y)

<Axes: xlabel='thal', ylabel='target'>
```

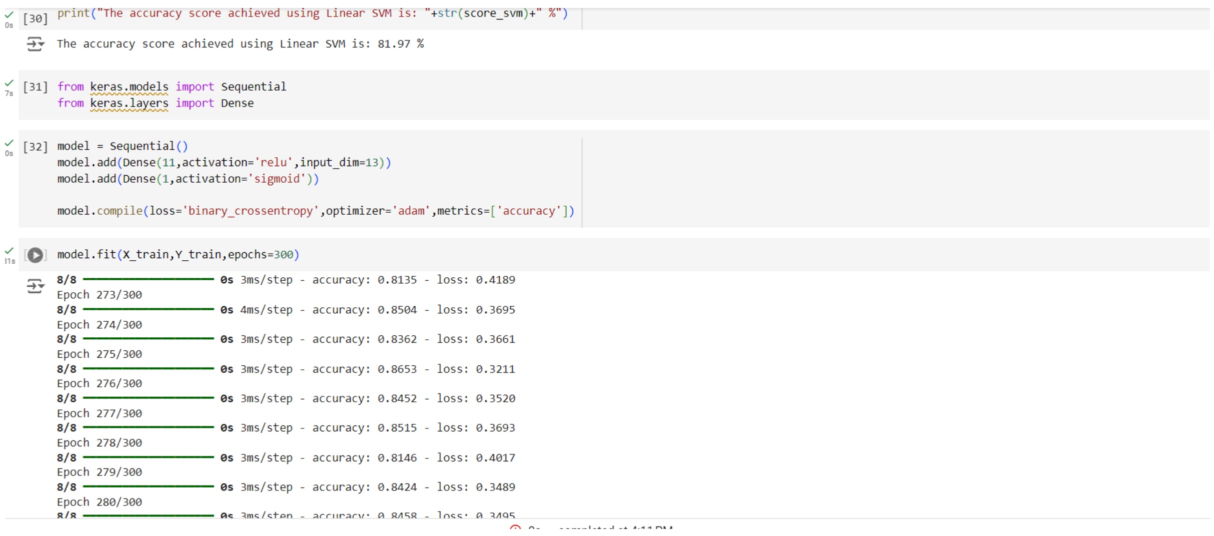




```
[16]          0          1          2          3
                         thal

[17] from sklearn.model_selection import train_test_split

     predictors = dataset.drop("target",axis=1)
     target = dataset["target"]

     X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.20,random_state=0)

[19] X_train.shape

     (242, 13)

     Y_train.shape

     (242,)

[22] from sklearn.metrics import accuracy_score
     from sklearn.linear_model import LogisticRegression

     lr = LogisticRegression()

     lr.fit(X_train,Y_train)

     Y_pred_lr = lr.predict(X_test)
```



```
     lr = LogisticRegression()
[22]
     lr.fit(X_train,Y_train)

     Y_pred_lr = lr.predict(X_test)

[23] Y_pred_lr.shape

     (61,)

[24] score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)

     print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")

     The accuracy score achieved using Logistic Regression is: 85.25 %

[25] from sklearn.naive_bayes import GaussianNB

     nb = GaussianNB()

     nb.fit(X_train,Y_train)

     Y_pred_nb = nb.predict(X_test)

[26] Y_pred_nb.shape

     (61,)

[27] score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)
```



```
[30] print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")

     The accuracy score achieved using Linear SVM is: 81.97 %

[31] from keras.models import Sequential
     from keras.layers import Dense

[32] model = Sequential()
     model.add(Dense(11,activation='relu',input_dim=13))
     model.add(Dense(1,activation='sigmoid'))

     model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

     model.fit(X_train,Y_train,epochs=300)

     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8135 - loss: 0.4189
     Epoch 273/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8504 - loss: 0.3695
     Epoch 274/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8362 - loss: 0.3661
     Epoch 275/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8653 - loss: 0.3211
     Epoch 276/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8452 - loss: 0.3520
     Epoch 277/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8515 - loss: 0.3693
     Epoch 278/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8146 - loss: 0.4017
     Epoch 279/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8424 - loss: 0.3489
     Epoch 280/300
     8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8458 - loss: 0.3495
```

Google Collaboratory Link: -

https://colab.research.google.com/drive/1MHB7IF7M0GefJ9jUsiSfT_RaYiMei8Vo#scrollTo=s6eihg_9AKpd

Conclusion: -

AI-based disease prediction from patient data plays a crucial role in modern healthcare by enhancing early detection and improving patient outcomes. By analyzing vast amounts of data, including medical history, genetic information, and lifestyle factors, AI can identify patterns that might be overlooked by human clinicians, leading to more accurate diagnoses. This capability enables healthcare providers to implement timely interventions, which can significantly improve the chances of successful treatment.