**DIESEASE PREDICTION**

**1.Importing libraries:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split,cross\_val\_score

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from scipy.stats import mode

from collections import Counter

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")

**2.Load the Dataset:**

Reading the train.csv by removing the last column since it an empty column

df = pd.read\_csv("/content/Training.csv").dropna(*axis* = 1)

df.head()

**3. Checking whether the dataset is balanced or not:**

disease\_counts = df["prognosis"].value\_counts()

disease\_counts

temp\_df = pd.DataFrame({

    "Disease": disease\_counts.index,

    "Counts": disease\_counts.values

})

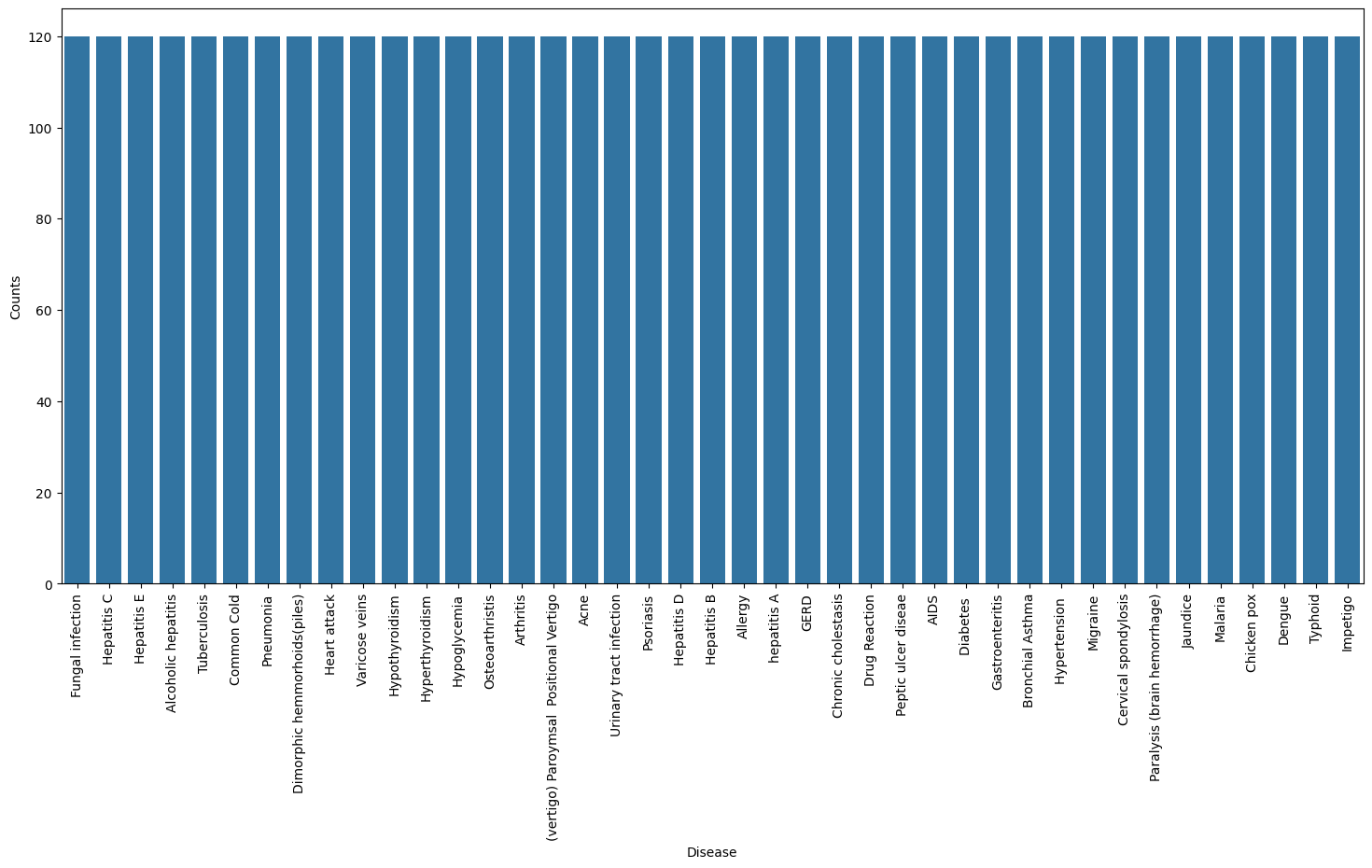
plt.figure(*figsize* = (18,8))

sns.barplot(*x* = "Disease", *y* = "Counts", *data* = temp\_df)

plt.xticks(*rotation*=90)

plt.show()

**Output:**

****

**4.** **Encoding the target value into numerical:**

encoder = LabelEncoder()

df["prognosis"] = encoder.fit\_transform(df["prognosis"])

**5.Splitting the data for training and testing the model:**

X = df.drop("prognosis",*axis* = 1)

y = df["prognosis"]

print(X.shape)

print(y.shape)

Output:

(4920, 132)

(4920,)

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,*test\_size* = 0.2,*random\_state* = 24)

print(f"Train: {X\_train.shape}, {y\_train.shape}")

print(f"Test: {X\_test.shape}, {y\_test.shape}")

Output:

Train: (3936, 132), (3936,)

Test: (984, 132), (984,)

**6.Defining scoring metric for k-fold cross validation:**

def *cv\_scoring*(*estimator*, *X*, *y*):

  return accuracy\_score(y, estimator.predict(X))

models = {

    "SVC":SVC(),

    "Guassian NB":GaussianNB(),

    "Random Forest":RandomForestClassifier(*random\_state* = 18)

}

**7.Producing cross validation score for the models:**

**for** model\_name **in** models:

    model =models[model\_name]

    scores =cross\_val\_score(model, X, y, *cv* =10,

*n\_jobs* =-1,

*scoring* =cv\_scoring)

print("=="\*30)

print(model\_name)

print(f"Scores: {scores}")

print(f"Mean Score: {np.mean(scores)}")

Output:

============================================================

Random Forest

Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Mean Score: 1.0

**8. Training and testing SVM Classifier:**

svm\_model =SVC()

svm\_model.fit(X\_train, y\_train)

preds =svm\_model.predict(X\_test)

print(f"Accuracy on test data by SVM Classifier: {accuracy\_score(y\_test, preds)\*100}")

cf\_matrix =confusion\_matrix(y\_test, preds)

plt.figure(*figsize*=(12,8))

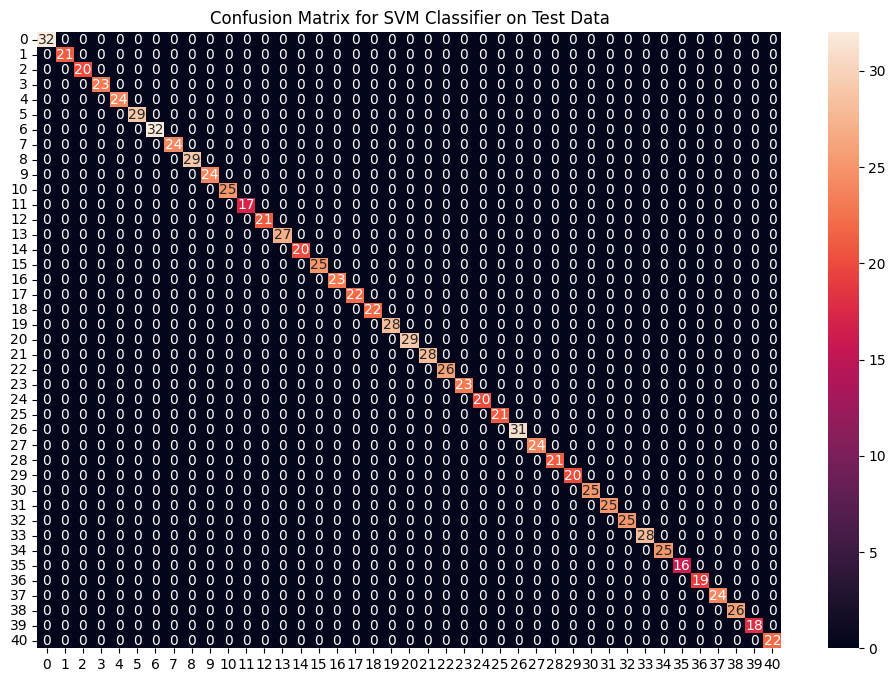
sns.heatmap(cf\_matrix, *annot*=True)

plt.title("Confusion Matrix for SVM Classifier on Test Data")

plt.show()

Output:

Accuracy on test data by SVM Classifier: 100.0



**9.Training and testing Naive Bayes Classifier:**

nb\_model =GaussianNB()

nb\_model.fit(X\_train, y\_train)

preds =nb\_model.predict(X\_test)

print(f"Accuracy on train data by Naive Bayes Classifier: {accuracy\_score(y\_train, nb\_model.predict(X\_train))\*100}")

print(f"Accuracy on test data by Naive Bayes Classifier: {accuracy\_score(y\_test, preds)\*100}")

cf\_matrix =confusion\_matrix(y\_test, preds)

plt.figure(*figsize*=(12,8))

sns.heatmap(cf\_matrix, *annot*=True)

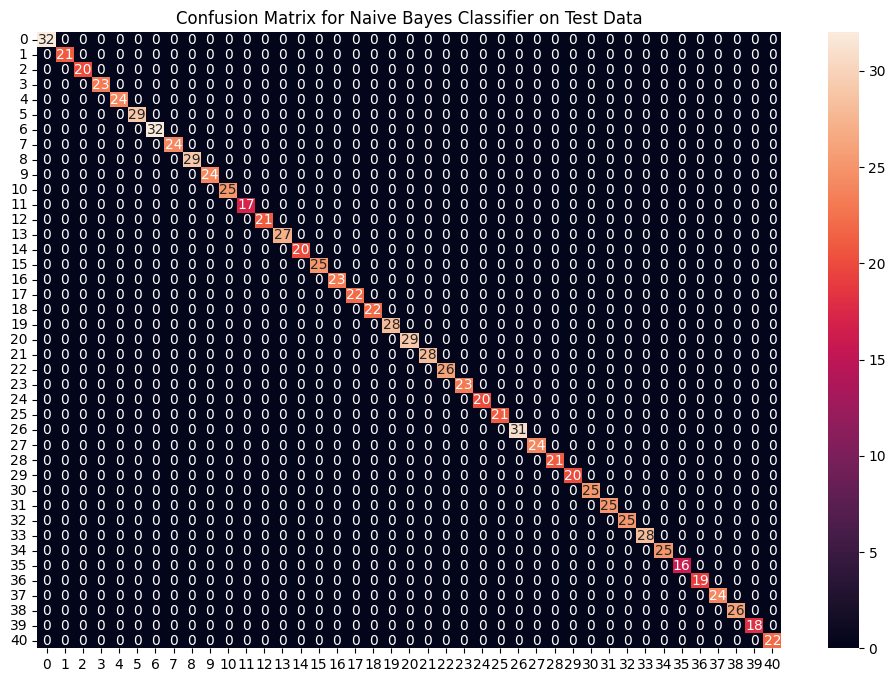
plt.title("Confusion Matrix for Naive Bayes Classifier on Test Data")

plt.show()

**Output:**

Accuracy on train data by Naive Bayes Classifier: 100.0

Accuracy on test data by Naive Bayes Classifier: 100.0

****

**10.Training and testing Random Forest Classifier**

rf\_model =RandomForestClassifier(*random\_state*=18)

rf\_model.fit(X\_train, y\_train)

preds =rf\_model.predict(X\_test)

print(f"Accuracy on train data by Random Forest Classifier: {accuracy\_score(y\_train, rf\_model.predict(X\_train))\*100}")

print(f"Accuracy on test data by Random Forest Classifier: {accuracy\_score(y\_test, preds)\*100}")

cf\_matrix =confusion\_matrix(y\_test, preds)

plt.figure(*figsize*=(12,8))

sns.heatmap(cf\_matrix, *annot*=True)

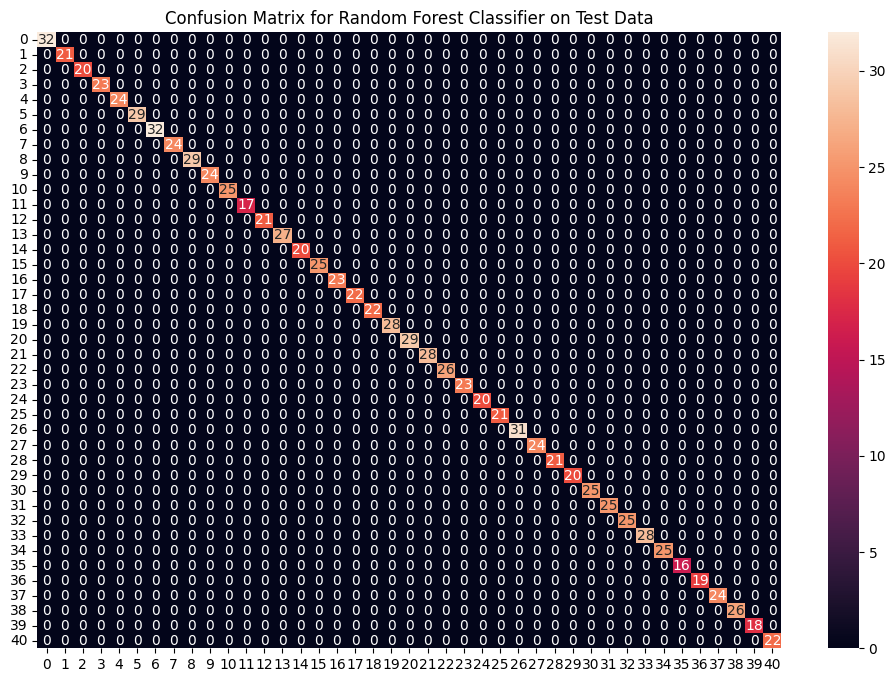
plt.title("Confusion Matrix for Random Forest Classifier on Test Data")

plt.show()

Output:

Accuracy on train data by Random Forest Classifier: 100.0

Accuracy on test data by Random Forest Classifier: 100.0



**11.Fitting the model on whole data and validating on the Test dataset**:

final\_svm\_model =SVC()

final\_nb\_model =GaussianNB()

final\_rf\_model =RandomForestClassifier(*random\_state*=18)

final\_svm\_model.fit(X, y)

final\_nb\_model.fit(X, y)

final\_rf\_model.fit(X, y)

**12. Reading the test data:**

test\_data = pd.read\_csv(r"D:\Data\_science\Practice\_files\Diseases\_Training\_data.csv").dropna(*axis*=1)

# Separate features (test\_X) and target labels (test\_Y)

test\_X = test\_data.iloc[:, :-1]

test\_Y = encoder.transform(test\_data.iloc[:, -1])

**13.Making prediction by take mode of predictions:**

svm\_preds = final\_svm\_model.predict(test\_X)

nb\_preds = final\_nb\_model.predict(test\_X)

rf\_preds = final\_rf\_model.predict(test\_X)

# Stack the predictions vertically

all\_preds = np.vstack((svm\_preds, nb\_preds, rf\_preds))

**14.Compute the mode along the first axis (axis=0)**

mode\_preds, \_ = mode(all\_preds, *axis*=0)

y\_preds = mode\_preds.flatten()

print(f"Accuracy on Test dataset by the combined model",{accuracy\_score(test\_Y, y\_preds)\*100})

cf\_matrix = confusion\_matrix(test\_Y, y\_preds)

plt.figure(*figsize*=(12,8))

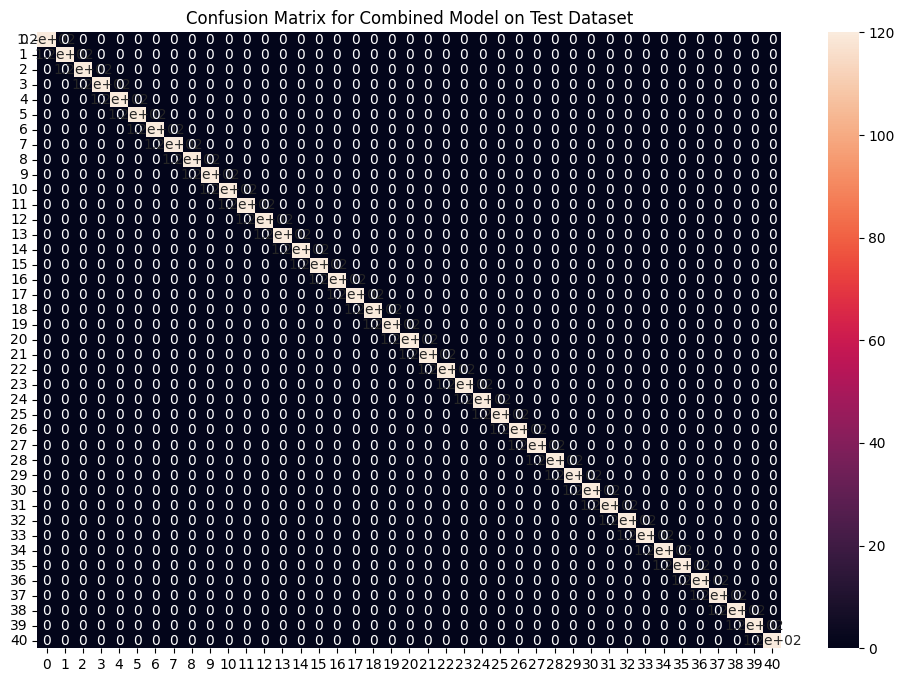
sns.heatmap(cf\_matrix, *annot* = True)

plt.title("Confusion Matrix for Combined Model on Test Dataset")

plt.show()

Output:

Accuracy on Test dataset by the combined model {100.0}



# Symptoms columns and encoding dictionary

symptoms = X.columns.values

# Creating a symptom index dictionary to encode the input symptoms into numerical form

symptom\_index = {}

**for** index, value **in** enumerate(symptoms):

    symptom = " ".join([i.capitalize() **for** i **in** value.split("\_")])

    symptom\_index[symptom] = index

data\_dict = {

    "symptom\_index": symptom\_index,

    "predictions\_classes": encoder.classes\_

}

# Defining the Function

# Input: string containing symptoms separated by commas

# Output: Generated predictions by models

def *predictDisease*(*symptoms*):

    symptoms = symptoms.split(",")

    # Creating input data for the models

    input\_data = [0] \* len(data\_dict["symptom\_index"])

**for** symptom **in** symptoms:

        symptom = symptom.strip()  # Ensure there's no leading/trailing whitespace

        index = data\_dict["symptom\_index"].get(symptom, None)

**if** index is not None:

            input\_data[index] = 1

    # Reshaping the input data and converting it into a suitable format for model predictions

    input\_data = np.array(input\_data).reshape(1, -1)

    # Generating individual outputs

    rf\_prediction = data\_dict["predictions\_classes"][final\_rf\_model.predict(input\_data)[0]]

    nb\_prediction = data\_dict["predictions\_classes"][final\_nb\_model.predict(input\_data)[0]]

    svm\_prediction = data\_dict["predictions\_classes"][final\_svm\_model.predict(input\_data)[0]]

    # Making final prediction by taking mode of all predictions

    final\_prediction = Counter([rf\_prediction, nb\_prediction, svm\_prediction]).most\_common(1)[0][0]

    predictions = {

        "rf\_model\_prediction": rf\_prediction,

        "naive\_bayes\_prediction": nb\_prediction,

        "svm\_model\_prediction": svm\_prediction,

        "final\_prediction": final\_prediction

    }

**return** predictions

# Testing the function

print(predictDisease("Itching,Skin Rash,Nodal Skin Eruptions"))

Output:

{'rf\_model\_prediction': 'Fungal infection', 'naive\_bayes\_prediction': 'Fungal infection', 'svm\_model\_prediction': 'Fungal infection', 'final\_prediction': 'Fungal infection'}