In [1]:

```
#Importing the libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Ignore harmless warnings

import warnings
warnings.filterwarnings("ignore")

# Set to display all the columns in dataset

pd.set_option("display.max_columns", None)

# Import psql to run queries

import pandasql as psql
```

In [2]:

```
#load the disease_testing datset
disease=pd.read_csv(r"C:\Users\harsh\Downloads\disease_Testing.csv",header=0)
#copy file to backup file
disease_BK=disease.copy()
#display the first 5 records
disease.head()
```

Out[2]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting	vomiti
0	1	1	1	0	0	0	0	0	0	0	0	
1	0	0	0	1	1	1	0	0	0	0	0	
2	0	0	0	0	0	0	0	1	1	1	0	
3	1	0	0	0	0	0	0	0	0	0	0	
4	1	1	0	0	0	0	0	1	0	0	0	

In [4]:

```
disease.shape
```

Out[4]: (42, 133)

In [3]:

```
disease.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Columns: 133 entries, itching to prognosis
dtypes: int64(132), object(1)
memory usage: 43.8+ KB
```

In [5]:

```
#displaying the duplicate values in dataset
disease_dup = disease[disease.duplicated(keep='last')]
disease_dup
```

Out[5]:

itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_p

In [6]:
```
1 disease.nunique()
```

Out[6]:
```
itching                 2
skin_rash               2
nodal_skin_eruptions    2
continuous_sneezing     2
shivering               2
                      ..
inflammatory_nails      2
blister                 2
red_sore_around_nose    2
yellow_crust_ooze       2
prognosis              41
Length: 133, dtype: int64
```

In [8]:
```
1 disease.columns
```

Out[8]:
```
Index(['itching', 'skin_rash', 'nodal_skin_eruptions', 'continuous_sneezing',
       'shivering', 'chills', 'joint_pain', 'stomach_pain', 'acidity',
       'ulcers_on_tongue',
       ...
       'blackheads', 'scurring', 'skin_peeling', 'silver_like_dusting',
       'small_dents_in_nails', 'inflammatory_nails', 'blister',
       'red_sore_around_nose', 'yellow_crust_ooze', 'prognosis'],
      dtype='object', length=133)
```

In [7]:
```
1 disease.isnull().sum()
```

Out[7]:
```
itching                 0
skin_rash               0
nodal_skin_eruptions    0
continuous_sneezing     0
shivering               0
                       ..
inflammatory_nails      0
blister                 0
red_sore_around_nose    0
yellow_crust_ooze       0
prognosis               0
Length: 133, dtype: int64
```

In [9]:
```
1 disease['prognosis'].value_counts()
```

Out[9]:
```
prognosis
Fungal infection                          2
Hepatitis C                               1
Hepatitis E                               1
Alcoholic hepatitis                       1
Tuberculosis                              1
Common Cold                               1
Pneumonia                                 1
Dimorphic hemmorhoids(piles)              1
Heart attack                              1
Varicose veins                            1
Hypothyroidism                            1
Hyperthyroidism                           1
Hypoglycemia                              1
Osteoarthristis                           1
Arthritis                                 1
(vertigo) Paroymsal  Positional Vertigo   1
Acne                                      1
Urinary tract infection                   1
Psoriasis                                 1
Hepatitis D                               1
Hepatitis B                               1
Allergy                                   1
hepatitis A                               1
GERD                                      1
Chronic cholestasis                       1
Drug Reaction                             1
Peptic ulcer diseae                       1
AIDS                                      1
Diabetes                                  1
Gastroenteritis                           1
Bronchial Asthma                          1
Hypertension                              1
Migraine                                  1
Cervical spondylosis                      1
```

In [10]:
```
1 #use label encoder for target variables
2 from sklearn.preprocessing import LabelEncoder
3 LE=LabelEncoder()
4 disease['prognosis']=LE.fit_transform(disease['prognosis'])
```

In [11]:
```
1 disease.head()
```

Out[11]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting	vomiti
0	1	1	1	0	0	0	0	0	0	0	0	
1	0	0	0	1	1	1	0	0	0	0	0	
2	0	0	0	0	0	0	0	1	1	1	0	
3	1	0	0	0	0	0	0	0	0	0	0	
4	1	1	0	0	0	0	0	1	0	0	0	

In [12]:
```
# Identify the independent and Target (dependent) variables

IndepVar = []
for col in disease.columns:
    if col != 'prognosis':
        IndepVar.append(col)

TargetVar = 'prognosis'

x = disease[IndepVar]
y = disease[TargetVar]
```

In [13]:
```
# Split the data into train and test (random sampling)

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

# Display the shape for train & test data

x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[13]: ((29, 132), (13, 132), (29,), (13,))

In [14]:
```
x_train.head()
```

Out[14]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting	vomi
9	0	0	0	0	0	0	0	0	0	0	0	
15	0	0	0	0	0	1	0	0	0	0	0	
33	0	0	0	0	0	0	0	0	0	0	0	
16	1	1	0	0	0	0	0	0	0	0	0	
36	0	0	0	0	0	0	0	0	0	0	0	

In [15]:
```
y_train.head()
```

Out[15]:
```
9      6
15    29
33    25
16     8
36     0
Name: prognosis, dtype: int32
```

In [16]:
```
from sklearn.preprocessing import MinMaxScaler

mmscaler = MinMaxScaler(feature_range=(0, 1))

x_train = mmscaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train)

x_test = mmscaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test)
```

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report

```
from imblearn.over_sampling import SMOTE
import xgboost as xgb
# Load the dataset
file_path = r"C:\Users\ratho\OneDrive\Desktop\Medical_dataset(1).xlsx"
data = pd.read_excel(file_path, header=0)
# Preprocessing
le = LabelEncoder()
data['Gender'] = le.fit_transform(data['Gender'])
data['Family Medical History'] = le.fit_transform(data['Family Medical History'])

# Splitting features and target variable
X = data.drop('Family Medical History', axis=1)
y = data['Family Medical History']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
# Hyperparameter tuning for Random Forest, XGBoost, and Gradient Boosting
param_grid_rf = {
    'n_estimators': [100, 200, 300],
```

```
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
param_grid_xgb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 6, 9],
    'subsample': [0.8, 1.0]
}
# Models initialization with refined hyperparameter tuning
models = {
    'KNN': KNeighborsClassifier(n_neighbors=5),    # Adjust KNN parameters
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=3, n_jobs=-1, verbose=2),
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'XGBoost': GridSearchCV(xgb.XGBClassifier(random_state=42), param_grid_xgb, cv=3, n_jobs=-1, verbose=2),
    'Gradient Boosting': GridSearchCV(GradientBoostingClassifier(random_state=42), param_grid_gb, cv=3, n_jobs=-1, verbose=2)
}
```

```
# Function to evaluate models
def evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    return accuracy, report
# Evaluating each model
results = {}
for model_name, model in models.items():
    print(f"Training {model_name}...")
    accuracy, report = evaluate_model(model, X_train_res, y_train_res, X_test, y_test)
    results[model_name] = {'Accuracy': accuracy, 'Classification Report': report}
# Displaying results
for model_name, result in results.items():
    print(f"Model: {model_name}")
    print(f"Accuracy: {result['Accuracy']:.2f}")
    print("Classification Report:")
    print(result['Classification Report'])
    print("-" * 50)
# Plotting bar graph for accuracies
model_names = list(results.keys())
accuracies = [results[model]['Accuracy'] for model in model_names]
plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracies, color='skyblue')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Models')
```

```
plt.ylim([0, 1])

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Model: KNN
Accuracy: 0.24
Classification Report:
              precision    recall  f1-score   support

           0       0.30      0.38      0.33        21
           1       0.21      0.29      0.24        17
           2       0.19      0.25      0.22        16
           3       0.35      0.25      0.29        24
           4       0.09      0.05      0.06        22

    accuracy                           0.24       100
   macro avg       0.23      0.24      0.23       100
weighted avg       0.23      0.24      0.23       100

--------------------------------------------------
Model: Decision Tree
Accuracy: 0.22
Classification Report:
              precision    recall  f1-score   support

           0       0.32      0.29      0.30        21
           1       0.29      0.29      0.29        17
           2       0.12      0.19      0.14        16
           3       0.39      0.29      0.33        24
           4       0.05      0.05      0.05        22

    accuracy                           0.22       100
   macro avg       0.23      0.22      0.22       100
weighted avg       0.24      0.22      0.23       100

--------------------------------------------------
Model: Random Forest
Accuracy: 0.25
Classification Report:
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.33      0.29      0.31        21
           1       0.14      0.18      0.16        17
           2       0.27      0.44      0.33        16
           3       0.31      0.21      0.25        24
           4       0.21      0.18      0.20        22

    accuracy                           0.25       100
   macro avg       0.25      0.26      0.25       100
weighted avg       0.26      0.25      0.25       100

--------------------------------------------------
Model: Logistic Regression
Accuracy: 0.18
Classification Report:
              precision    recall  f1-score   support

           0       0.15      0.10      0.12        21
           1       0.19      0.35      0.25        17
           2       0.15      0.31      0.20        16
           3       0.11      0.04      0.06        24
           4       0.29      0.18      0.22        22

    accuracy                           0.18       100
   macro avg       0.18      0.20      0.17       100
weighted avg       0.18      0.18      0.16       100

--------------------------------------------------
Model: XGBoost
Accuracy: 0.26
Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.38      0.36        21
           1       0.18      0.24      0.21        17
           2       0.22      0.25      0.24        16
           3       0.37      0.29      0.33        24
```

4	0.18	0.14	0.15	22
accuracy			0.26	100
macro avg	0.26	0.26	0.26	100
weighted avg	0.26	0.26	0.26	100

Model: Gradient Boosting
Accuracy: 0.23
Classification Report:

	precision	recall	f1-score	support
0	0.33	0.33	0.33	21
1	0.09	0.12	0.10	17
2	0.24	0.38	0.29	16
3	0.27	0.12	0.17	24
4	0.25	0.23	0.24	22
accuracy			0.23	100
macro avg	0.24	0.24	0.23	100
weighted avg	0.24	0.23	0.23	100