**SOURCE CODE**

import pandas as pd

d=pd.read\_csv('/content/drive/MyDrive/fetal\_health\_expanded.csv')

print(d)

d.head()

d.shape

d.describe()

d.info()

from matplotlib import pyplot as pt

d.hist(figsize=(20,20))

pt.show()

#Graphs

x=d['histogram\_tendency']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_variance']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_median']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_mean']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_mode']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_number\_of\_zeroes']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_number\_of\_peaks']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_max']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['histogram\_min']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['percentage\_of\_time\_with\_abnormal\_long\_term\_variability']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['mean\_value\_of\_short\_term\_variability']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['abnormal\_short\_term\_variability']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['prolongued\_decelerations']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['severe\_decelerations']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['light\_decelerations']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['uterine\_contractions']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['fetal\_movement']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['accelerations']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

x=d['baseline value']

y=d['fetal\_health']

pt.plot(x,y)

pt.scatter(x,y)

print(d.isnull())

print(d.describe())

d.columns

import matplotlib.pyplot as plt

category\_mapping = {1.0: "Normal", 2.0: "Suspect", 3.0: "Pathological"}

fetal\_health\_counts = d["fetal\_health"].value\_counts()

fetal\_health\_counts = fetal\_health\_counts.loc[[1.0, 2.0, 3.0]]

category\_labels = [category\_mapping[key] for key in fetal\_health\_counts.index]

plt.figure(figsize=(10, 5))

plt.bar(category\_labels, fetal\_health\_counts.values,color=["#5F93A0", "#B0E0E6", "#ADD8E6"])

plt.title("Fetal Health Count")

plt.xlabel("Fetal Health")

plt.ylabel("Count")

plt.show()

import matplotlib.pyplot as plt

fetal\_health\_counts = d["fetal\_health"].value\_counts().sort\_index()

category\_mapping = {1.0: "Normal", 2.0: "Suspect", 3.0: "Pathological"}

fetal\_health\_counts = fetal\_health\_counts.loc[[1.0, 2.0, 3.0]]

labels = [category\_mapping[key] for key in fetal\_health\_counts.index]

colors = ['lightblue', 'orange', 'red']

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

fetal\_health\_counts.plot(

    kind="pie",

    autopct="%1.0f%%",

    labels=labels,

    colors=colors,

    startangle=140,

    wedgeprops={'edgecolor': 'black'}

)

plt.legend(labels, title="Fetal Health Categories", bbox\_to\_anchor=(1.1, 0.5), loc="center left")

plt.show()

import seaborn as sns

sns.boxplot(data=d,x='baseline value')

sns.boxplot(data=d,x='accelerations')

sns.boxplot(data=d,x='fetal\_movement')

sns.boxplot(data=d,x='uterine\_contractions')

sns.boxplot(data=d,x='light\_decelerations')

sns.boxplot(data=d,x='severe\_decelerations')

sns.boxplot(data=d,x='prolongued\_decelerations')

sns.boxplot(data=d,x='abnormal\_short\_term\_variability')

sns.boxplot(data=d,x='mean\_value\_of\_short\_term\_variability')

sns.boxplot(data=d,x='percentage\_of\_time\_with\_abnormal\_long\_term\_variability')

sns.boxplot(data=d,x='mean\_value\_of\_long\_term\_variability')

sns.boxplot(data=d,x='histogram\_width')

sns.boxplot(data=d,x='histogram\_min')

sns.boxplot(data=d,x='histogram\_max')

sns.boxplot(data=d,x='histogram\_number\_of\_peaks')

sns.boxplot(data=d,x='histogram\_number\_of\_zeroes')

sns.boxplot(data=d,x='histogram\_mode')

sns.boxplot(data=d,x='histogram\_mean')

sns.boxplot(data=d,x='histogram\_median')

sns.boxplot(data=d,x='histogram\_variance')

sns.boxplot(data=d,x='histogram\_tendency')

sns.boxplot(data=d,x='fetal\_health')

corr\_matrix=d.corr()

print(corr\_matrix)

import seaborn as sn

import matplotlib.pyplot as plt

sn.heatmap(corr\_matrix,annot=True)

numeric\_data=d.select\_dtypes(exclude=['object'])

numeric\_corr=numeric\_data.corr()

f,ax=plt.subplots(figsize=(25,1))

sn.heatmap(numeric\_corr.sort\_values(by='fetal\_health',ascending=False).head(1),cmap='GnBu')

plt.title("Numerical Features with fetal health",weight='bold',fontsize=20,color="black")

plt.yticks(weight="bold",color="darkgreen",rotation=0)

plt.show()

cov\_matrix=d.cov()

print(cov\_matrix)

import seaborn as sn

import matplotlib.pyplot as plt

sn.heatmap(cov\_matrix,annot=True)

x=d.iloc[:,0:21]

y=d.iloc[:,21:22]

print(x)

print(y)

from sklearn.model\_selection import train\_test\_split,GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from imblearn.over\_sampling import SMOTE

print(x.shape)

X = d.drop(columns=["fetal\_health"])

y = d["fetal\_health"].astype(int)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42,stratify=y)

from sklearn.feature\_selection import mutual\_info\_regression, SelectKBest

mi\_score = SelectKBest(mutual\_info\_regression, k=10)

mi\_score.fit(x\_train, y\_train)

x\_train\_mi = mi\_score.transform(x\_train)

x\_test\_mi = mi\_score.transform(x\_test)

for i in range(len(mi\_score.scores\_)):

    print(f'Feature {i}: {round(mi\_score.scores\_[i], 4)}')

original\_columns=d.drop('fetal\_health',axis=1).columns

plt.bar([original\_columns[i] for i in range(len(mi\_score.scores\_))],mi\_score.scores\_,color=['green'])

plt.xticks(rotation=90) #prevent overlapping

plt.rcParams['figure.figsize']=[8,7]

ax=plt.gca()

ax.patch.set\_facecolor('white')

ax.yaxis.grid(True,color='blue')

ax.patch.set\_edgecolor('black')

ax.patch.set\_linewidth(1.5)

plt.xlabel('Features',fontsize=14)

plt.ylabel('Mutual Information Score',fontsize=14)

plt.title('Mutual Information Scores of Features',fontsize=16,weight='bold')

plt.show()

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

x\_train\_scaled = scaler.fit\_transform(x\_train)

x\_test\_scaled = scaler.transform(x\_test)

smote = SMOTE(random\_state=42)

x\_train\_resampled, y\_train\_resampled = smote.fit\_resample(x\_train\_scaled, y\_train)

from sklearn.linear\_model import LogisticRegression

lr=LogisticRegression()

param\_grid = {'C': [0.01, 0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}

grid\_search = GridSearchCV(LogisticRegression(), param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(x\_train\_resampled, y\_train\_resampled)

best\_lr = grid\_search.best\_estimator\_

test\_predictions = best\_lr.predict(x\_test\_scaled)

print(test\_predictions)

from sklearn.metrics import accuracy\_score

accuracy=accuracy\_score(test\_predictions,y\_test)\*100

print(f"Accuracy of logistic Regression = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.tree import DecisionTreeClassifier

classifier=DecisionTreeClassifier(criterion='entropy',random\_state=0)

mm=classifier.fit(x\_train,y\_train)

yp=mm.predict(x\_test)

print(yp)

from sklearn.metrics import accuracy\_score

accuracy=accuracy\_score(yp,y\_test)\*100

print(f"Accuracy of Decision Tree Classifier = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.neighbors import KNeighborsClassifier

KNN=KNeighborsClassifier(n\_neighbors=5,metric='minkowski',p=2)

KNN.fit(x\_train,y\_train)

yp=KNN.predict(x\_test)

print(yp)

from sklearn.metrics import accuracy\_score

accuracy=accuracy\_score(yp,y\_test)\*100

print(f"Accuracy of KNN = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.naive\_bayes import GaussianNB

gnb=GaussianNB()

gnb.fit(x\_train,y\_train)

yp=gnb.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test)\*100

print(f"Accuracy of Gaussian Naive Bayes = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.svm import SVC

svm\_model=SVC(kernel='linear')

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

yp=svm\_model.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test)\*100

print(f"Accuracy of SVM = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.ensemble import RandomForestClassifier

random\_forest=RandomForestClassifier()

random\_forest.fit(x\_train,y\_train)

yp=random\_forest.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test)\*100

print(f"Accuracy of Random Forest = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.ensemble import GradientBoostingClassifier

gbc=GradientBoostingClassifier()

gbc.fit(x\_train,y\_train)

yp=gbc.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test)\*100

print(f"Accuracy of Gradient Boosting Classifier = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

pip install xgboost lightgbm

from xgboost import XGBClassifier

y\_train\_adjusted = y\_train - 1

y\_test\_adjusted = y\_test - 1

xgb=XGBClassifier()

xgb.fit(x\_train,y\_train\_adjusted)

yp=xgb.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test\_adjusted)\*100

print(f"Accuracy of XGB Classifier = {round(accuracy,2)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test\_adjusted,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from lightgbm import LGBMClassifier

y\_train\_adjusted = y\_train - 1

y\_test\_adjusted = y\_test - 1

lgbm=LGBMClassifier()

lgbm.fit(x\_train,y\_train\_adjusted)

yp=lgbm.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test\_adjusted)\*100

print(f"Accuracy of LGBM Classifier = {round(accuracy,3)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test\_adjusted,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.ensemble import AdaBoostClassifier

y\_train\_adjusted = y\_train - 1

y\_test\_adjusted = y\_test - 1

AdaBoost=AdaBoostClassifier(n\_estimators=100)

AdaBoost.fit(x\_train,y\_train\_adjusted)

yp=AdaBoost.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test\_adjusted)\*100

print(f"Accuracy of AdaBoost Classifier = {round(accuracy,2)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test\_adjusted,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

from sklearn.neural\_network import MLPClassifier #Multi-Layer Perceptron

y\_train\_adjusted = y\_train - 1

y\_test\_adjusted = y\_test - 1

MLP=MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=500)

MLP.fit(x\_train,y\_train\_adjusted)

yp=MLP.predict(x\_test)

print(yp)

accuracy=accuracy\_score(yp,y\_test\_adjusted)\*100

print(f"Accuracy of MLP Classifier = {round(accuracy,2)} %")

from sklearn import metrics

confusion\_matrix=metrics.confusion\_matrix(y\_test\_adjusted,yp)

cm\_display=metrics.ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix,display\_labels=[3,2,1])

plt.figure(figsize=(6,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[1,2,3], yticklabels=[1,2,3])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

import matplotlib.pyplot as plt

import numpy as np

y\_train\_adjusted = y\_train - 1

y\_test\_adjusted = y\_test - 1

models = {

    'Logistic Regression': lr,

    'Decision Tree Classifier': classifier,

    'KNN': classifier,

    'Gaussian Naive Bayes': gnb,

    'SVM': svm\_model,

    'Random Forest': random\_forest,

    'Gradient Boost ':gbc,

    'XGB':xgb,

    'LGBM':lgbm,

    'AdaBoost':AdaBoost,

    'MLP':MLP

}

model\_accuracies = {}

for model\_name, model in models.items():

    model.fit(x\_train, y\_train\_adjusted)

    y\_pred\_adjusted = model.predict(x\_test)

    y\_pred=y\_pred\_adjusted+1

    accuracy = accuracy\_score(y\_test, y\_pred)\*100

    model\_accuracies[model\_name] = accuracy

    print(f"Accuracy of {model\_name}: {accuracy:.4f}")

    print(metrics.classification\_report(y\_test, y\_pred))

    print("-" \* 50)

sorted\_models = dict(sorted(model\_accuracies.items(), key=lambda item: item[1], reverse=True))

best\_model\_name = list(sorted\_models.keys())[0]

best\_model\_accuracy = list(sorted\_models.values())[0]

print(f"The best model is {best\_model\_name} with an accuracy of {best\_model\_accuracy:.2f}")

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

colors = ['blue', 'green', 'red', 'purple', 'orange', 'cyan', 'magenta', 'gray', 'yellow', 'pink', 'brown']

model\_names = list(sorted\_models.keys())

accuracies = list(sorted\_models.values())

plt.figure(figsize=(12, 6))

bars = plt.bar(model\_names, accuracies, color=colors)

plt.xlabel('Models')

plt.ylabel('Accuracy (%)')

plt.title('Comparison of Model Accuracies')

plt.ylim(0, 100)

plt.yticks(range(0, 110, 10))

plt.xticks(rotation=45)

legend\_patches = [mpatches.Patch(color=colors[i], label=model\_names[i]) for i in range(len(model\_names))]

plt.legend(handles=legend\_patches, bbox\_to\_anchor=(1.05, 1), loc='upper left', title="Algorithms")  # Legend box on side

plt.show()

plt.figure(figsize=(12, 6))

plt.plot(model\_names, accuracies, marker='o', linestyle='-', color='b', linewidth=1, label="Accuracy")  # Thin line

plt.xlabel('Models')

plt.ylabel('Accuracy (%)')

plt.title('Model Accuracy Trend')

plt.ylim(0, 100)

plt.yticks(range(0, 110, 10))

plt.xticks(rotation=45)

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left', title="Performance metrics")

plt.show()

import pandas as pd

from tabulate import tabulate

from prettytable import PrettyTable

columns=["Algorithm\_name","Accuracy"]

table=PrettyTable(columns)

for key,value in sorted\_models.items():

    table.add\_row([key,f"{value:.2f}"])

print(table)