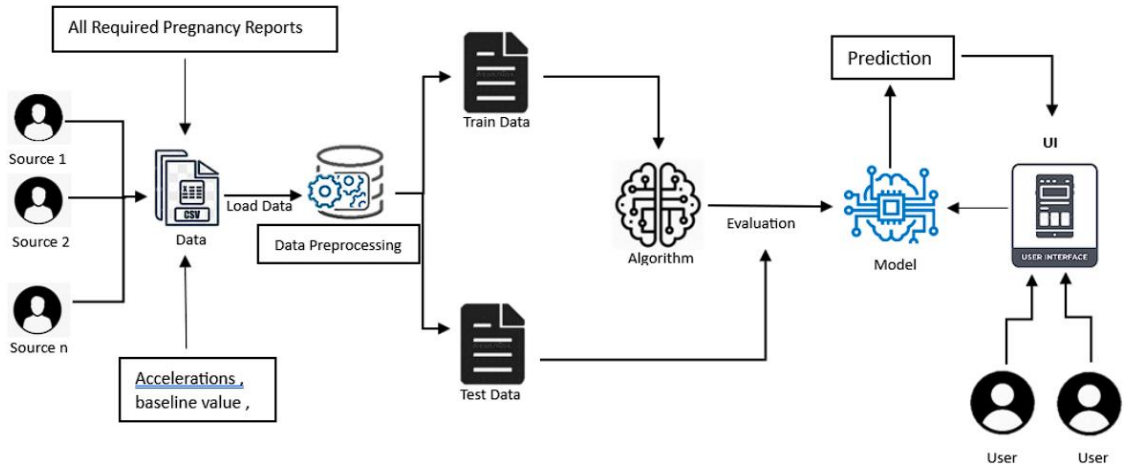
FetalAI: USING MACHINE LEARNING TO PREDICT AND MONITOR FETAL HEALTH

Project Description:

Reduction of child mortality is reflected in several of the United Nations' Sustainable Development Goals and is a key indicator of human progress.The UN expects that by 2030, countries end preventable deaths of newborns and children under 5 years of age, with all countries aiming to reduce under 5 mortality to at least as low as 25 per 1,000 live births. Parallel to the notion of child mortality is of course maternal mortality, which accounts for 295 000 deaths during and following pregnancy and childbirth (as of 2017). The vast majority of these deaths (94%) occurred in low-resource settings, and most could have been prevented.In light of what was mentioned above, Cardiotocograms (CTGs) are a simple and cost accessible option to assess fetal health, allowing healthcare professionals to take action in order to prevent child and maternal mortality. The equipment itself works by sending ultrasound pulses and reading its response, thus shedding light on fetal heart rate (FHR), fetal movements, uterine contractions and more. In this project, we have some characteristics of Fetal Health as a dataset. The target variable of this dataset is Fetal Health. Since it is a multi class classification, the classes are represented by ‘Normal’, ‘Pathological’ and ‘Suspect’.

Technical Architecture:



Project Workflow:

1. **Problem Definition / Understanding:**

- Clearly articulate the business problem.
- Outline business requirements.
- Conduct a literature survey to understand existing solutions.
- Assess the social or business impact of the problem.

2. **Data Collection & Preparation:**

- Gather the necessary dataset for analysis.

3. **Exploratory Data Analysis:**

- Utilize descriptive statistical techniques.
- Perform visual analysis of the dataset.
- Conduct feature selection for relevant variables.
- Apply scaling to ensure uniformity in data units.
- Check and address any imbalance in the dataset.

4. **Model Building:**

- Split the dataset into training and testing sets.
- Apply Synthetic Minority Over-sampling Technique (SMOTE) for balancing data.
- Train the model post-SMOTE application.
- Explore training the model using multiple algorithms.
- Test the model to assess its performance.

5. **Performance Testing**:**

- Generate a dataframe summarizing the model's performance metrics.
- Create a bar plot to visualize and compare model performance.

6. **Model Deployment**:**

1. Save the most effective model.
2. Integrate the model with a web framework for user interaction.

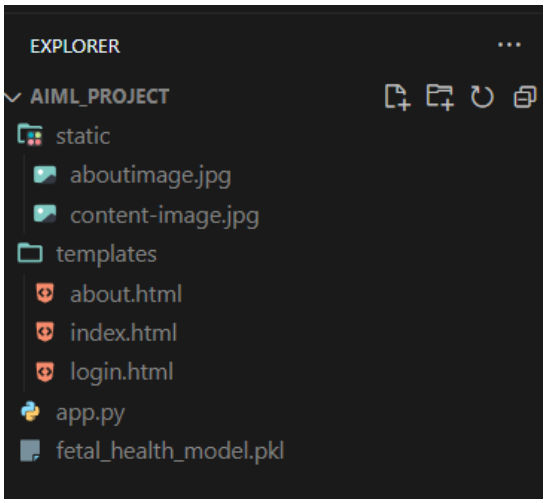
This structured approach ensures a comprehensive and systematic progression from understanding the problem to deploying the model in a practical setting.

Project Demonstration & Documentation

1. Create an explanatory video detailing the entire solution for the project.

2. Document the project comprehensively, outlining a step-by-step procedure for project development.

Project Structure:



1. We're constructing a Flask application that requires HTML pages stored within the templates folder, alongside a Python script named app.py for scripting.

2. Our saved model, fetal_health_model.pkl, will be utilized for integration with Flask.

3. The training folder encompasses a file dedicated to model training.

Milestone 1: Define Problem / Problem Understanding

Objective 1: Define the Business Problem

Addressing the reduction of child mortality aligns with various Sustainable Development Goals outlined by the United Nations and serves as a crucial indicator of human progress. The UN envisions that by 2030, nations should eliminate preventable deaths among newborns and children under 5, striving to reduce under-5 mortality to a minimum of 25 per 1,000 live births. In tandem, maternal mortality, accounting for 295,000 deaths during and after pregnancy and childbirth as of 2017, remains a significant concern. A striking 94% of these fatalities occurred in low-resource settings and were largely preventable.

In consideration of the aforementioned challenges, Cardiotocograms (CTGs) emerge as a straightforward and economically accessible solution for assessing fetal health. This technology empowers healthcare professionals to proactively address fetal and maternal mortality risks. Operating by emitting ultrasound pulses and analyzing their responses, CTGs provide insights into fetal heart rate (FHR), fetal movements, uterine contractions, and more.

Objective 2: Business Requirements

The requirements for a Fetal Health classification project within the healthcare sector vary based on the project's specific goals and objectives. Typically, the business need for fetal health classification is prevalent in the healthcare industry, particularly in the Obstetrics and Gynecology (OB/GYN) department. The classification of fetal health holds significance in ensuring the well-being of the unborn baby and facilitating informed decisions regarding pregnancy management.

Numerous reasons drive the necessity for healthcare providers to classify fetal health. For instance, it aids in identifying fetuses at risk of preterm labor, intrauterine growth restriction, or other medical conditions requiring early intervention or specialized care. Moreover, fetal health classification supports healthcare providers in monitoring fetal health during pregnancy, detecting abnormalities or complications, and making informed decisions regarding the timing of delivery.

Healthcare providers employ various tools and techniques, such as ultrasound, fetal monitoring, and diagnostic tests, to classify fetal health. Additionally, machine learning and artificial intelligence algorithms play a role in classifying fetal health based on parameters like heart rate, movement, and other physiological measures. These advanced techniques enhance the accuracy and timeliness of diagnosis and treatment decisions, ultimately contributing to improved health outcomes for both the mother and the baby.

Objective 3: Literature Review

Conducting a literature review for a Fetal Health classification project entails an in-depth exploration of existing studies, articles, and publications pertaining to the subject. The survey seeks to compile insights into current classification systems, their merits and shortcomings, and potential knowledge gaps that the project might fill. Additionally, the literature review delves into the methodologies and techniques applied in past classification projects, examining any pertinent data or discoveries that could guide the design and execution of the present undertaking.

Objective 4: Social or Business Impact

Social Impact:

Facilitating Informed Decision-Making: The provision of accurate and current information on Fetal Health empowers expectant parents to make well-informed decisions regarding their pregnancy and childbirth. For instance, upon detecting a significant health issue in the fetus, parents can make choices about continuing the pregnancy or exploring alternative options.

Mitigating Infant Mortality: Access to information about fetal health enables expectant parents to identify and address potential health issues before they pose life-threatening risks to the unborn child. This proactive approach contributes to reducing the infant mortality rate and ensuring a higher number of healthy births.

Enhancing Prenatal Care: Informed about the health status of their fetus, expectant parents can collaborate with their healthcare providers to develop a customized prenatal care plan that addresses any identified issues. This collaborative effort is poised to yield improved outcomes for both the mother and the child.

Business Model/Impact:

Enhanced Patient Outcomes: Early detection of potential health issues in fetuses enables healthcare providers to formulate treatment plans that contribute to improved outcomes for both the mother and the child. This, in turn, has the potential to elevate patient satisfaction and retention rates.

Revenue Growth: Healthcare providers offering fetal health testing and monitoring services stand to create additional revenue streams from expectant parents willing to invest in these services. Moreover, the identification of a health issue in the fetus may necessitate additional tests, procedures, and treatments, further augmenting the revenue potential for healthcare providers.

Research and Development Stimulus: Knowledge about fetal health serves as a catalyst for research and development within the healthcare industry. Insights gained from fetal health monitoring and testing can spur the creation of innovative diagnostic tests and treatments, contributing to advancements in healthcare.

Milestone 2: Data Collection & Preparation

Machine learning is inherently reliant on data, constituting the pivotal element that facilitates the training of algorithms. This section provides the means to acquire the necessary dataset.

Objective 1: Collect the Dataset

- Numerous widely recognized open sources, such as Kaggle.com and the UCI repository, offer data collection opportunities.

- For this particular project, we have employed .csv data, obtainable from Kaggle.com. To access the dataset, please utilize the following link: https://www.kaggle.com/datasets/andrewmvd/fetal-health-classification.

- Following the dataset download, it is imperative to thoroughly comprehend the data through the application of visualization and analysis techniques.

- Note: Various techniques exist for comprehending data, and in this instance, a selection of them has been utilized. It is advisable to explore multiple techniques for a more comprehensive understanding.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
[1] 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.model_selection import StratifiedKFold

    from sklearn.linear_model import LogisticRegression

    from sklearn.ensemble import RandomForestClassifier

    from sklearn.ensemble import GradientBoostingClassifier

    from xgboost import XGBClassifier
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import classification_report
    from sklearn.metrics import confusion_matrix
    from sklearn.model_selection import learning_curve
    from sklearn.metrics import accuracy_score
    from imblearn.over_sampling import SMOTE
    import warnings

    warnings.simplefilter(action="ignore")
```

Let's dive into Activity 1.2: Accessing the Dataset

Our dataset comes in various formats like .csv, excel files, .txt, .json, and more. To access and interpret this dataset, we can leverage the functionality of Pandas, a powerful Python library.

In Pandas, the method read_csv() proves useful for reading datasets stored in CSV format. It requires the directory path pointing to the specific CSV file as a parameter.

```
[2] # Load the dataset
    data = pd.read_csv('/content/fetal_health.csv')

    # Display the first few rows to get an overview of the data
    # print(data.head())
    print(data.shape)

    (2126, 22)
```

Let's tackle Activity 2: Data Preparation

Now that we've familiarized ourselves with the dataset, it's time to prep it for analysis.

Directly using the imported data for machine learning models isn't feasible as it might contain irregularities. We'll need to clean and shape the dataset appropriately. This activity involves a series of steps:

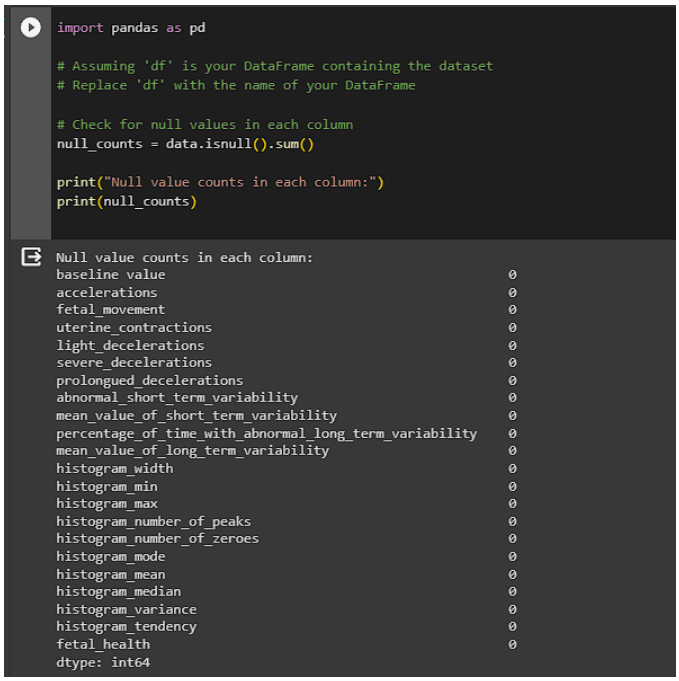
- Dealing with Missing Values

- Encoding Data

- Addressing Imbalanced Data

Keep in mind, these steps serve as general guidelines for data preprocessing in preparation for machine learning. Depending on your dataset's specific characteristics, you might not need to execute all of these steps or may need additional procedures to ensure the data is in optimal condition for analysis.

Missing values:



```
import pandas as pd

# Assuming 'df' is your DataFrame containing the dataset
# Replace 'df' with the name of your DataFrame

# Check for null values in each column
null_counts = data.isnull().sum()

print("Null value counts in each column:")
print(null_counts)
```

```
Null value counts in each column:
baseline value                                          0
accelerations                                           0
fetal_movement                                          0
uterine_contractions                                    0
light_decelerations                                     0
severe_decelerations                                    0
prolongued_decelerations                                0
abnormal_short_term_variability                         0
mean_value_of_short_term_variability                    0
percentage_of_time_with_abnormal_long_term_variability  0
mean_value_of_long_term_variability                     0
histogram_width                                         0
histogram_min                                           0
histogram_max                                           0
histogram_number_of_peaks                               0
histogram_number_of_zeroes                              0
histogram_mode                                          0
histogram_mean                                          0
histogram_median                                        0
histogram_variance                                      0
histogram_tendency                                      0
fetal_health                                            0
dtype: int64
```

No , null values time to move on

Handling Categorical Data.

There are no categorical values in the dataset. That is why we can skip this step.

Handling Imbalance Data

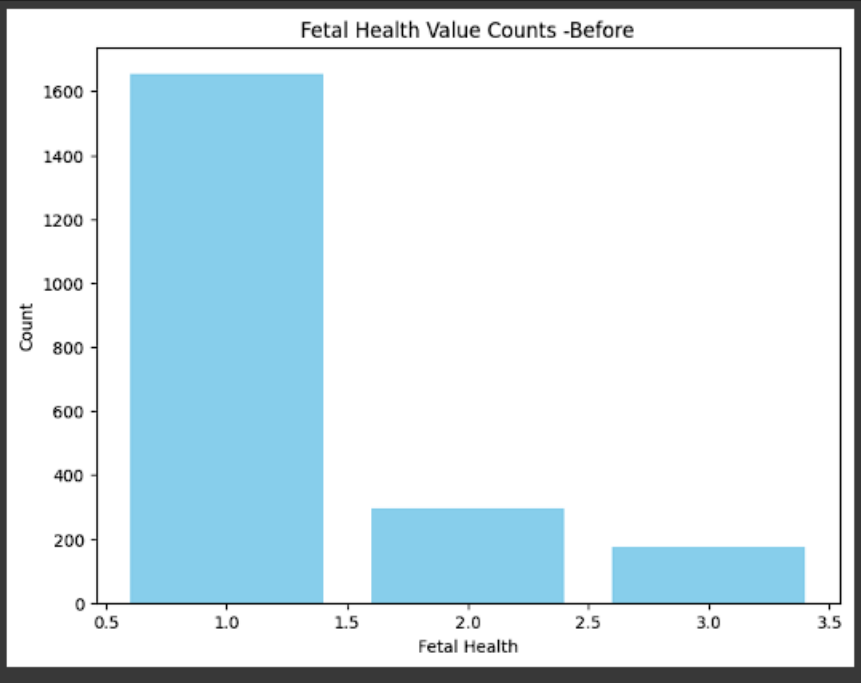
```
[ ] data['fetal_health'].unique()
```

```
array([2., 1., 3.])
```

```
[ ] from collections import Counter

    counter = Counter(y)
    for k, v in counter.items():
      dist = v/len(y)*100
      print(f"class={k}, n = {v} ({dist}%)")
```

```
class=2.0, n = 295 (13.8758231420508%)
class=1.0, n = 1655 (77.84571966133585%)
class=3.0, n = 176 (8.27845719661336%)
```

```
from collections import Counter
import matplotlib.pyplot as plt

# Assuming you have already calculated the counter
counter = Counter(y)

# Extract class labels and counts
classes = list(counter.keys())
counts = list(counter.values())

# Plot histogram
plt.figure(figsize=(8, 6))
plt.bar(classes, counts, color='skyblue')
plt.title('Fetal Health Value Counts -Before')
plt.xlabel('Fetal Health')
plt.ylabel('Count')
plt.show()
```



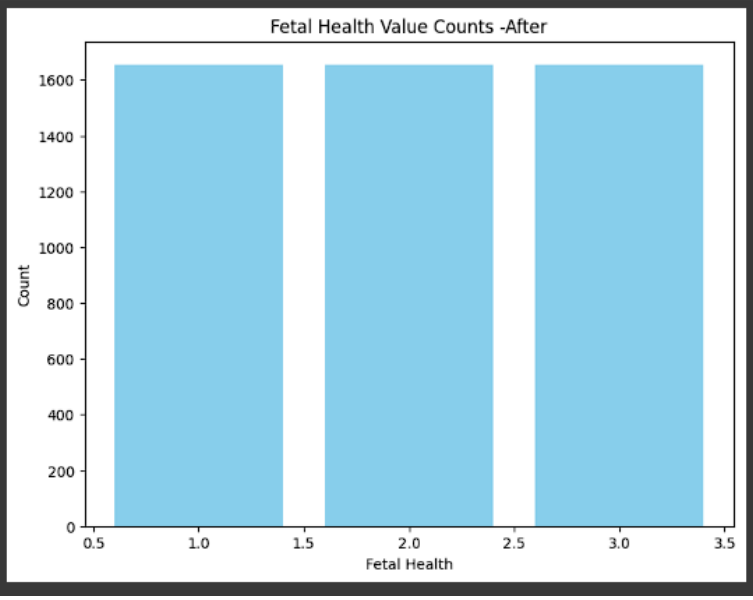
Applying SMOTE for balancing the data

```
[ ]  from imblearn.over_sampling import SMOTE
     oversample = SMOTE()
     X , y = oversample.fit_resample(X , y)
```

```
[ ]  from collections import Counter

     counter = Counter(y)
     for k, v in counter.items():
       dist = v/len(y)*100
       print(f"class={k}, n = {v} ({dist}%)")
```

```
class=2.0, n = 1655 (33.33333333333333%)
class=1.0, n = 1655 (33.33333333333333%)
class=3.0, n = 1655 (33.33333333333333%)
```

```
[ ]  from collections import Counter
     import matplotlib.pyplot as plt

     # Assuming you have already calculated the counter
     counter = Counter(y)

     # Extract class labels and counts
     classes = list(counter.keys())
     counts = list(counter.values())

     # Plot histogram
     plt.figure(figsize=(8, 6))
     plt.bar(classes, counts, color='skyblue')
     plt.title('Fetal Health Value Counts -After')
     plt.xlabel('Fetal Health')
     plt.ylabel('Count')
     plt.show()
```



Milestone 3: Exploratory Data Analysis

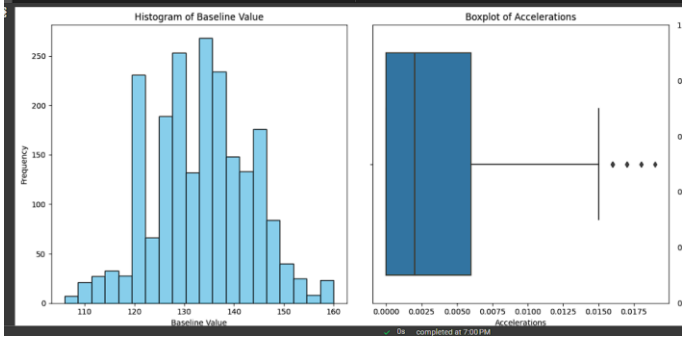
Activity 1: Descriptive statistical analysis

```
# Descriptive statistics
description = data.describe()

print("Descriptive statistics:")
print(description)
```

```
Descriptive statistics:
       baseline value  accelerations  fetal_movement  uterine_contractions  \
count     2126.000000    2126.000000     2126.000000           2126.000000
mean       133.303857       0.003178        0.009481              0.004366
std          9.840844       0.003866        0.046666              0.002946
min        106.000000       0.000000        0.000000              0.000000
25%        126.000000       0.000000        0.000000              0.002000
50%        133.000000       0.002000        0.000000              0.004000
75%        140.000000       0.006000        0.003000              0.007000
max        160.000000       0.019000        0.481000              0.015000

       light_decelerations  severe_decelerations  prolongued_decelerations  \
count          2126.000000           2126.000000               2126.000000
mean              0.001889              0.000003                  0.000159
std               0.002960              0.000057                  0.000590
min               0.000000              0.000000                  0.000000
25%               0.000000              0.000000                  0.000000
50%               0.000000              0.000000                  0.000000
75%               0.003000              0.000000                  0.000000
max               0.015000              0.001000                  0.005000

       abnormal_short_term_variability  mean_value_of_short_term_variability  \
count                      2126.000000                           2126.000000
mean                         46.990122                              1.332785
std                          17.192814                              0.883241
min                          12.000000                              0.200000
25%                          32.000000                              0.700000
50%                          49.000000                              1.200000
75%                          61.000000                              1.700000
max                          87.000000                              7.000000

       percentage_of_time_with_abnormal_long_term_variability  ...  \
count                                          2126.00000       ...
mean                                              9.84666       ...
std                                              18.39688       ...
min                                               0.00000       ...
25%                                               0.00000       ...
50%                                               0.00000       ...
75%                                              11.00000       ...
max                                              91.00000       ...

       histogram_min  histogram_max  histogram_number_of_peaks  \
count    2126.000000    2126.000000                2126.000000
mean       93.579492     164.025400                   4.068203
std        29.560212      17.944183                   2.949386
min        50.000000     122.000000                   0.000000
25%        67.000000     152.000000                   2.000000
50%        93.000000     162.000000                   3.000000
75%       120.000000     174.000000                   6.000000
max       159.000000     238.000000                  18.000000
```

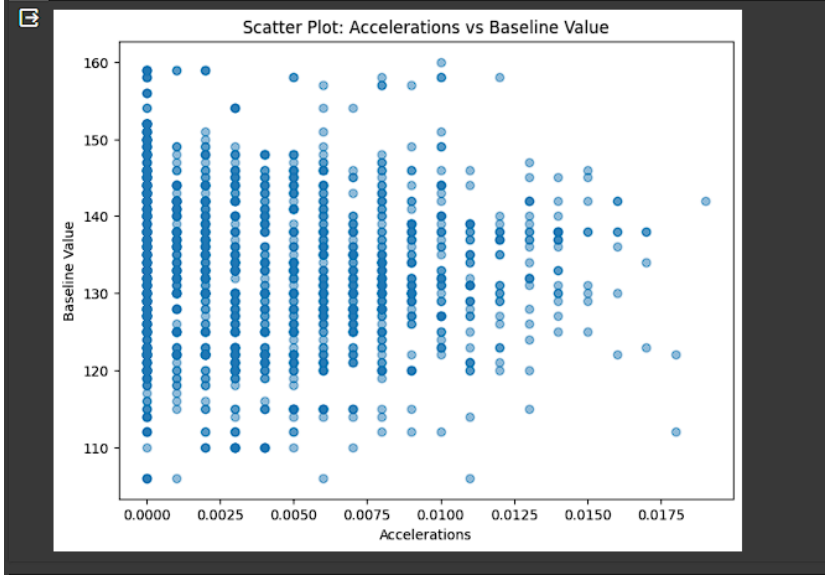
Activity 2.1: Univariate analysis

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a figure with subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Histogram
axes[0].hist(data['Baseline value'], bins=20, color='skyblue', edgecolor='black')
axes[0].set_title('Histogram of Baseline Value')
axes[0].set_xlabel('Baseline Value')
axes[0].set_ylabel('Frequency')

# Box plot
sns.boxplot(x=data['accelerations'], ax=axes[1])
axes[1].set_title('Boxplot of Accelerations')
axes[1].set_xlabel('Accelerations')


plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```

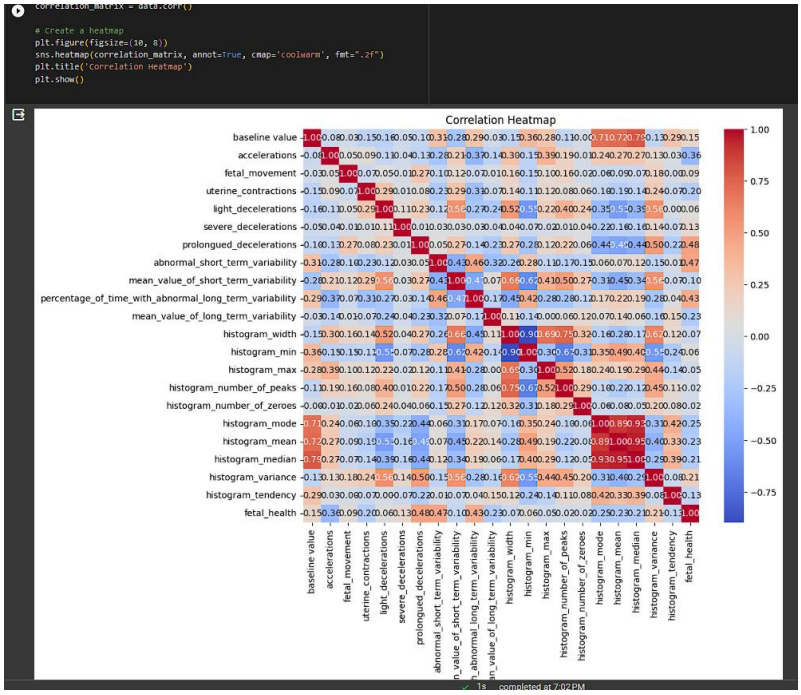


Bivariate Analysis:

```
import matplotlib.pyplot as plt

# Example: Scatter plot between 'accelerations' and 'baseline value'
plt.figure(figsize=(8, 6))
plt.scatter(data['accelerations'], data['baseline value'], alpha=0.5)
plt.title('Scatter Plot: Accelerations vs Baseline Value')
plt.xlabel('Accelerations')
plt.ylabel('Baseline Value')
plt.show()
```



Activity 2.2: Multivariate analysis



Activity 3: Feature Selection:

```
[ ]  #split the data

     X  = data.drop(['fetal_health'] , axis=1)
     y = data['fetal_health']
```

Activity 4: Scaling the data:

Our dataset has almost identical values so we proceeded with no scaling of values and we also got better results when compared to scaling the data.

Splitting data into train and test

```
[ ]  X_train , X_test , y_train , y_test = train_test_split(X , y , test_size =.25 , random_state = 42)
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Activity 4.1: Random forest model

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model on the resampled data
rf_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_rf = rf_classifier.predict(X_test)

# Calculate accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Confusion Matrix and Classification Report
confusion_rf = confusion_matrix(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Classifier Accuracy:", accuracy_rf)
print("\nRandom Forest Classifier Confusion Matrix:")
print(confusion_rf)
print("\nRandom Forest Classifier Classification Report:")
print(report_rf)
print("\n")
```

```
Random Forest Classifier Accuracy: 0.9710144927536232

Random Forest Classifier Confusion Matrix:
[[409  21   3]
 [  5 393   6]
 [  0   1 404]]

Random Forest Classifier Classification Report:
              precision    recall  f1-score   support

         1.0       0.99      0.94      0.97       433
         2.0       0.95      0.97      0.96       404
         3.0       0.98      1.00      0.99       405

    accuracy                           0.97      1242
   macro avg       0.97      0.97      0.97      1242
weighted avg       0.97      0.97      0.97      1242
```

Activity 4.2: Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Initialize the Logistic Regression model
lr_classifier = LogisticRegression(max_iter=1000, random_state=42)

# Train the model on the resampled data
lr_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_lr = lr_classifier.predict(X_test)

# Calculate accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# Confusion Matrix and Classification Report
confusion_lr = confusion_matrix(y_test, y_pred_lr)
report_lr = classification_report(y_test, y_pred_lr)

print("Logistic Regression Classifier Accuracy:", accuracy_lr)
print("\nLogistic Regression Classifier Confusion Matrix:")
print(confusion_lr)
print("\nLogistic Regression Classifier Classification Report:")
print(report_lr)
print("\n")
```

```
Logistic Regression Classifier Accuracy: 0.8252818035426731

Logistic Regression Classifier Confusion Matrix:
[[341  76  16]
 [ 44 333  27]
 [  6  48 351]]

Logistic Regression Classifier Classification Report:
              precision    recall  f1-score   support

         1.0       0.87      0.79      0.83       433
         2.0       0.73      0.82      0.77       404
         3.0       0.89      0.87      0.88       405

    accuracy                           0.83      1242
   macro avg       0.83      0.83      0.83      1242
weighted avg       0.83      0.83      0.83      1242
```

Activity 4.3: Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree model
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the model on the resampled data
dt_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred_dt = dt_classifier.predict(X_test)

# Calculate accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# Confusion Matrix and Classification Report
confusion_dt = confusion_matrix(y_test, y_pred_dt)
report_dt = classification_report(y_test, y_pred_dt)

print("Decision Tree Classifier Accuracy:", accuracy_dt)
print("\nDecision Tree Classifier Confusion Matrix:")
print(confusion_dt)
print("\nDecision Tree Classifier Classification Report:")
print(report_dt)
print("\n")
```

```
Decision Tree Classifier Accuracy: 0.962157809983897

Decision Tree Classifier Confusion Matrix:
[[412  19   2]
 [ 19 380   5]
 [  1   1 403]]

Decision Tree Classifier Classification Report:
              precision    recall  f1-score   support

         1.0       0.95      0.95      0.95       433
         2.0       0.95      0.94      0.95       404
         3.0       0.98      1.00      0.99       405

    accuracy                           0.96      1242
   macro avg       0.96      0.96      0.96      1242
weighted avg       0.96      0.96      0.96      1242
```

Activity 4.4: XG Boost classifier

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Encode target variable for XGBoost
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Initialize and train XGBoost model
xgb_model = XGBClassifier()
xgb_model.fit(X_train, y_train_encoded)

# Predict and evaluate
y_pred_xgb = xgb_model.predict(X_test)
xgb_accuracy = xgb_model.score(X_test, y_test_encoded)

# Confusion Matrix
xgb_confusion = confusion_matrix(y_test_encoded, y_pred_xgb)

# Classification Report
xgb_report = classification_report(y_test_encoded, y_pred_xgb)

# Print Results
print("XGBoost Accuracy:", xgb_accuracy)
print("\nXGBoost Confusion Matrix:")
print(xgb_confusion)
print("\nXGBoost Classification Report:")
print(xgb_report)
```

```
XGBoost Accuracy: 0.9766505636070854

XGBoost Confusion Matrix:
[[417  15   1]
 [  6 392   6]
 [  0   1 404]]

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97       433
           1       0.96      0.97      0.97       404
           2       0.98      1.00      0.99       405

    accuracy                           0.98      1242
   macro avg       0.98      0.98      0.98      1242
weighted avg       0.98      0.98      0.98      1242
```

Activity 4.5: Ada Boost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Encode target variable for AdaBoost
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Initialize and train AdaBoost model
ada_model = AdaBoostClassifier()
ada_model.fit(X_train, y_train_encoded)

# Predict and evaluate
y_pred_ada = ada_model.predict(X_test)
ada_accuracy = ada_model.score(X_test, y_test_encoded)

# Confusion Matrix
ada_confusion = confusion_matrix(y_test_encoded, y_pred_ada)

# Classification Report
ada_report = classification_report(y_test_encoded, y_pred_ada)

# Print Results
print("AdaBoost Accuracy:", ada_accuracy)
print("\nAdaBoost Confusion Matrix:")
print(ada_confusion)
print("\nAdaBoost Classification Report:")
print(ada_report)
```

```
AdaBoost Accuracy: 0.894524959742351

AdaBoost Confusion Matrix:
[[371  60   2]
 [ 29 371   4]
 [ 28   8 369]]

AdaBoost Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       433
           1       0.85      0.92      0.88       404
           2       0.98      0.91      0.95       405

    accuracy                           0.89      1242
   macro avg       0.90      0.90      0.90      1242
weighted avg       0.90      0.89      0.90      1242
```

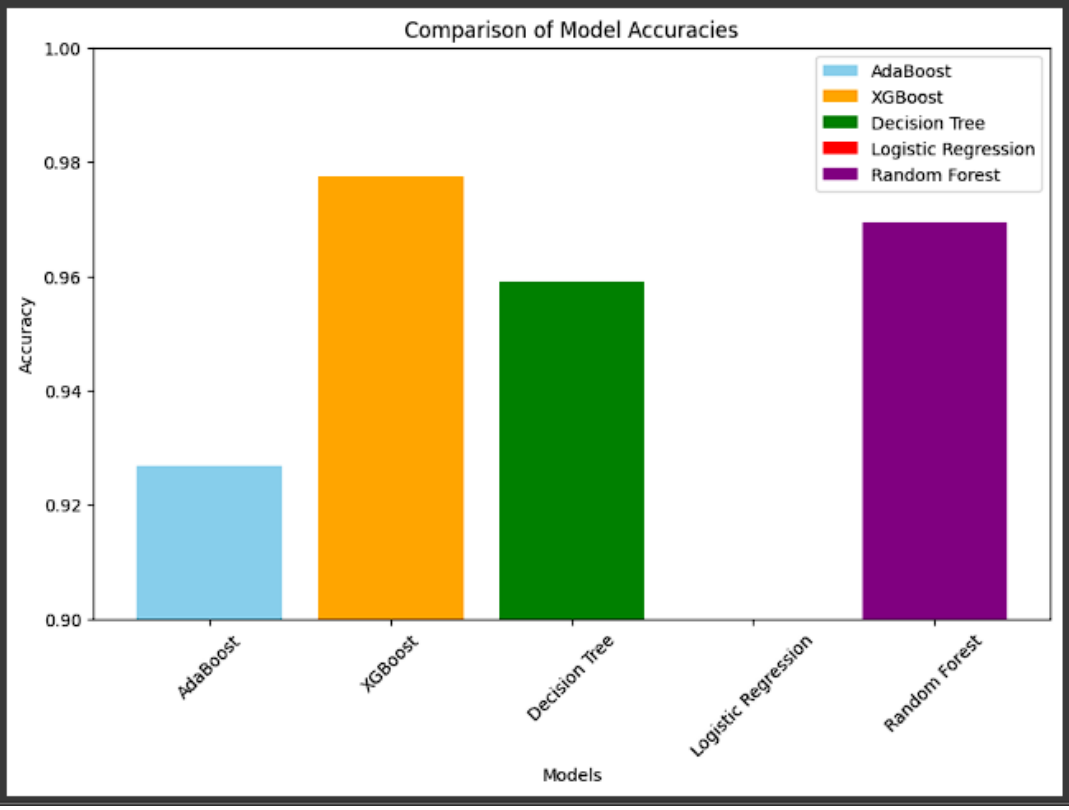
Activity 5: Testing the model

```
    [128, 0, 0, 0.003, 0, 0, 0, 86, 0.3, 79, 2.9, 16, 114, 130, 0, 0, 128, 126, 129, 0, 1],
    [124, 0, 0, 0, 0, 0, 0, 86, 0.3, 72, 4, 12, 110, 130, 1, 0, 124, 124, 125, 0, 0],
    [124, 0, 0, 0, 0, 0, 0, 86, 0.4, 14, 4.8, 24, 122, 146, 1, 0, 126, 126, 127, 0, -1],
    [124, 0, 0, 0, 0, 0, 0, 87, 0.2, 71, 3.4, 10, 118, 128, 0, 0, 124, 123, 125, 0, 0], # all 3's here
    #all 1's here
    [132, 0, 0.108, 0.002, 0.01, 0, 0, 26, 4.5, 0, 12.5, 149, 50, 199, 9, 0, 133, 120, 126, 56, 0],
    [132, 0, 0.112, 0.004, 0.014, 0, 0, 22, 6.9, 0, 6.3, 149, 50, 199, 10, 0, 123, 112, 115, 66, 0],
    [132, 0, 0.089, 0.001, 0.01, 0, 0, 29, 2.9, 0, 15.1, 144, 50, 194, 11, 1, 133, 124, 130, 35, 0],
    [120, 0.008, 0.103, 0.001, 0.001, 0, 0, 28, 3.4, 0, 21.7, 126, 55, 181, 13, 0, 121, 124, 126, 25, 0],
    [120, 0.009, 0.085, 0.002, 0.002, 0, 0, 28, 3.2, 0, 12.4, 128, 53, 181, 9, 1, 129, 125, 127, 25, 0],
    [120, 0.006, 0.109, 0.007, 0, 0, 0, 27, 3.7, 0, 24.2, 144, 51, 195, 11, 0, 125, 124, 126, 24, 0],
    [115, 0.005, 0.079, 0.005, 0.003, 0, 0, 23, 3.4, 0, 18.8, 130, 52, 182, 9, 0, 119, 116, 118, 21, 0],
    [114, 0.005, 0, 0.005, 0.003, 0, 0, 24, 3.2, 0, 16.2, 134, 52, 186, 8, 0, 117, 115, 117, 19, 0],
    [115, 0.006, 0.065, 0.004, 0.001, 0, 0, 22, 3.6, 0, 19.6, 138, 50, 188, 8, 0, 117, 117, 119, 21, 0],
    [115, 0.009, 0.055, 0.005, 0, 0, 0, 27, 2.3, 0, 12.4, 129, 53, 182, 7, 0, 119, 120, 120, 14, 0],
    [114, 0.008, 0.058, 0.007, 0.001, 0, 0, 28, 2.2, 0, 12.2, 98, 55, 153, 7, 1, 119, 119, 120, 13, 0],
    [114, 0.006, 0.047, 0.009, 0, 0, 0, 27, 2.4, 0, 13.5, 128, 54, 182, 6, 0, 119, 118, 119, 13, 0],
    #all 2's here
    [146, 0, 0.006, 0, 0, 0, 0, 69, 0.5, 45, 11, 42, 117, 159, 6, 0, 153, 150, 152, 3, 1],
    [146, 0, 0.004, 0, 0, 0, 0, 76, 0.3, 58, 4.4, 18, 135, 153, 2, 0, 150, 149, 151, 0, 1],
    [146, 0, 0.007, 0, 0, 0, 0, 65, 0.5, 32, 9.5, 33, 133, 166, 4, 0, 153, 151, 154, 1, 0],
    [146, 0, 0.014, 0, 0, 0, 0, 72, 0.3, 41, 6.9, 35, 120, 155, 4, 0, 147, 146, 148, 0, 1],
    [146, 0, 0.006, 0, 0, 0, 0, 77, 0.3, 61, 5.7, 29, 120, 149, 4, 0, 147, 146, 147, 0, 1],
    [146, 0, 0.021, 0, 0, 0, 0, 68, 0.3, 23, 7.2, 20, 136, 156, 1, 0, 147, 147, 148, 1, 0],
    [150, 0, 0.022, 0, 0, 0, 0, 74, 0.3, 56, 6.4, 25, 131, 156, 4, 0, 150, 150, 151, 0, 1]


]



# Loop through the custom inputs
for i, custom_input in enumerate(custom_inputs):
    # Reshape the input data into a 2D array
    custom_input = np.array([custom_input]).reshape(1, -1)

    # Make the prediction
    predictionrf =  xgb_model.predict(custom_input)
    print(f"Predicted Fetal Health for Data rf {i + 1}:", predictionrf[0])
```

```
Predicted Fetal Health for Data rf 1: 2
Predicted Fetal Health for Data rf 2: 2
Predicted Fetal Health for Data rf 3: 2
Predicted Fetal Health for Data rf 4: 2
Predicted Fetal Health for Data rf 5: 0
Predicted Fetal Health for Data rf 6: 0
Predicted Fetal Health for Data rf 7: 0
Predicted Fetal Health for Data rf 8: 0
Predicted Fetal Health for Data rf 9: 0
Predicted Fetal Health for Data rf 10: 0
Predicted Fetal Health for Data rf 11: 0
Predicted Fetal Health for Data rf 12: 0
Predicted Fetal Health for Data rf 13: 0
Predicted Fetal Health for Data rf 14: 0
Predicted Fetal Health for Data rf 15: 0
Predicted Fetal Health for Data rf 16: 0
```

Milestone 5: Performance Testing

```python
import matplotlib.pyplot as plt

# List of model names and their accuracies
model_names = ['AdaBoost', 'XGBoost', 'Decision Tree', 'Logistic Regression', 'Random Forest']
accuracies = [ada_accuracy, xgb_accuracy, accuracy_dt, accuracy_lr, accuracy_rf]

# Define colors for each model
colors = ['skyblue', 'orange', 'green', 'red', 'purple']

# Plotting the bar plot
plt.figure(figsize=(10, 6))

# Iterate through each model and plot its accuracy
for i, (model, acc, color) in enumerate(zip(model_names, accuracies, colors)):
    plt.bar(model, acc, color=color, label=model)

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim(0.9, 1)  # Set y-axis limits from 0.9 to 1 (90% to 100% accuracy)
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



The xg boost algorithm is working best among the rest so we choose and continue with it.

Milestone 6: Model Deployment:

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
[ ]  import joblib

     # Assuming `model` is your trained machine learning model
     joblib.dump(xgb_model, 'fetal_health_model.pkl')
```

```
['fetal_health_model.pkl']
```

Activity 2: Integrate with Web Framework

In this segment, we're developing a web application linked to our previously created model. A user interface (UI) is designed for users to input prediction values. The entered data is then fed into the saved model, and the resulting prediction is displayed on the UI. This phase involves the following steps:

- Creating HTML Pages

- Developing server-side scripts

- Launching and running the web application

Activity 2.1: Building Html Pages

For this project create two HTML files namely

• index.html

• login.html

• about.html

and save them in the templates folder.

Activity 2.2: Build Python code

- It begins by importing necessary modules like Flask, rendering templates, and handling requests.

- The Flask app is initialized.

- The saved machine learning model (presumably for fetal health prediction) is loaded using joblib.

- Various routes are defined:

  - `**/home_page**` renders the 'index.html' template.

  - `**/**` renders the 'login.html' template for the home route.

- **`/about`** renders the 'about.html' template.

- **`/predict`** is a POST route that processes form data sent by the user. It extracts the form values for fetal health features entered by the user, processes them as a feature array, and uses the loaded model to make a prediction.

- The prediction is converted to a human-readable label ('Normal', 'Suspect', 'Pathological') based on the predicted class.

- Finally, the prediction result is passed back to the 'index.html' template to display the predicted outcome on the user interface.

This script creates an interactive web application allowing users to input fetal health features, and based on these inputs, predicts the fetal health status using a pre-trained machine learning model. The result is displayed back to the user on the web page.

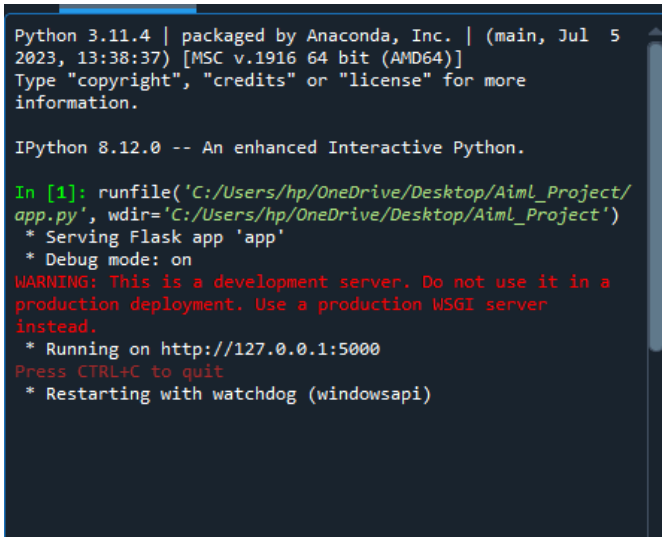
```
from flask import Flask, render_template, request
import joblib
import numpy as np

app = Flask(__name__)

# Load the saved model
model = joblib.load('fetal_health_model.pkl')

@app.route('/home_page')
def some_page():
    return render_template('index.html')


@app.route('/')
def home():
    return render_template('login.html')
@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Get the values from the form
    features = [
        float(request.form['baseline_value']),
        float(request.form['accelerations']),
        float(request.form['fetal_movement']),
        float(request.form['uterine_contractions']),
        float(request.form['light_decelerations']),
        float(request.form['severe_decelerations']),
        float(request.form['prolongued_decelerations']),
        float(request.form['abnormal_short_term_variability']),
        float(request.form['mean_value_of_short_term_variability']),
```

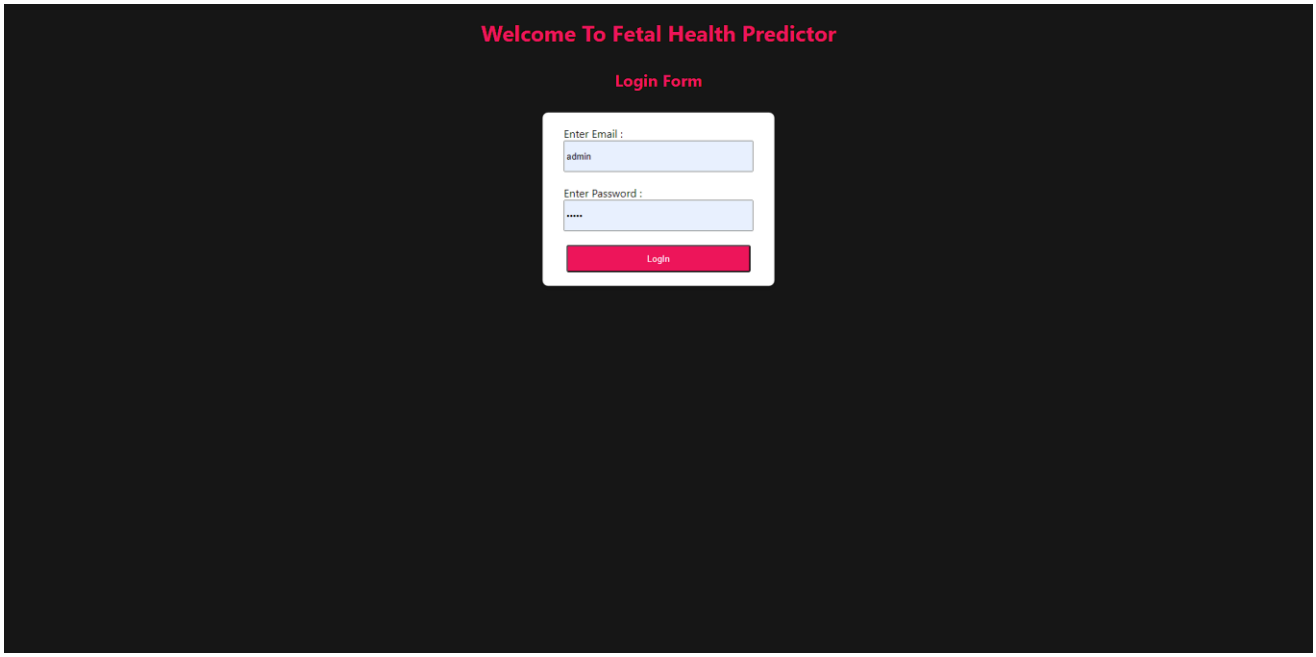
```
        float(request.form['percentage_of_time_with_abnormal_long_term_variabi
lity']),
        float(request.form['mean_value_of_long_term_variability']),
        float(request.form['histogram_width']),
        float(request.form['histogram_min']),
        float(request.form['histogram_max']),
        float(request.form['histogram_number_of_peaks']),
        float(request.form['histogram_number_of_zeroes']),
        float(request.form['histogram_mode']),
        float(request.form['histogram_mean']),
        float(request.form['histogram_median']),
        float(request.form['histogram_variance']),
        float(request.form['histogram_tendency'])
    ]

    # Convert the features into a numpy array
    features = np.array([features])

    # Make the prediction
    prediction = model.predict(features)

    # Convert the prediction to a human-readable label
    if prediction == 0:
        result = "Normal"
    elif prediction == 1:
        result = "Suspect"
    elif prediction == 2:
        result = "Pathological"

    return render_template('index.html', result=result)

if __name__ == '__main__':
    app.run(debug=True)
```

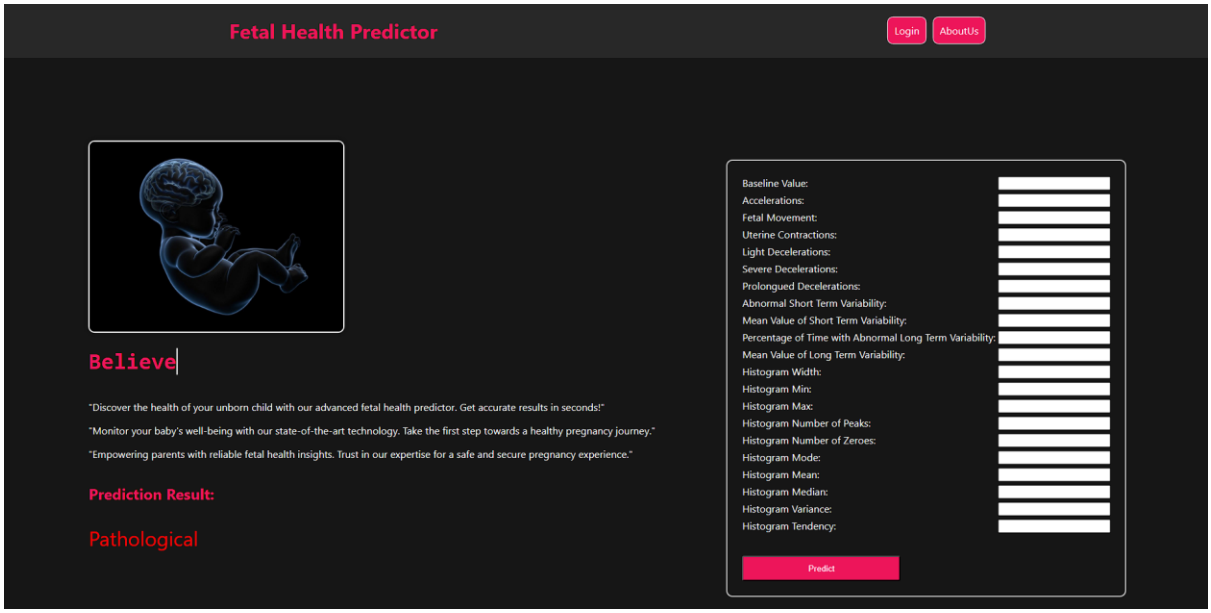
Activity 2.3: Run the web application

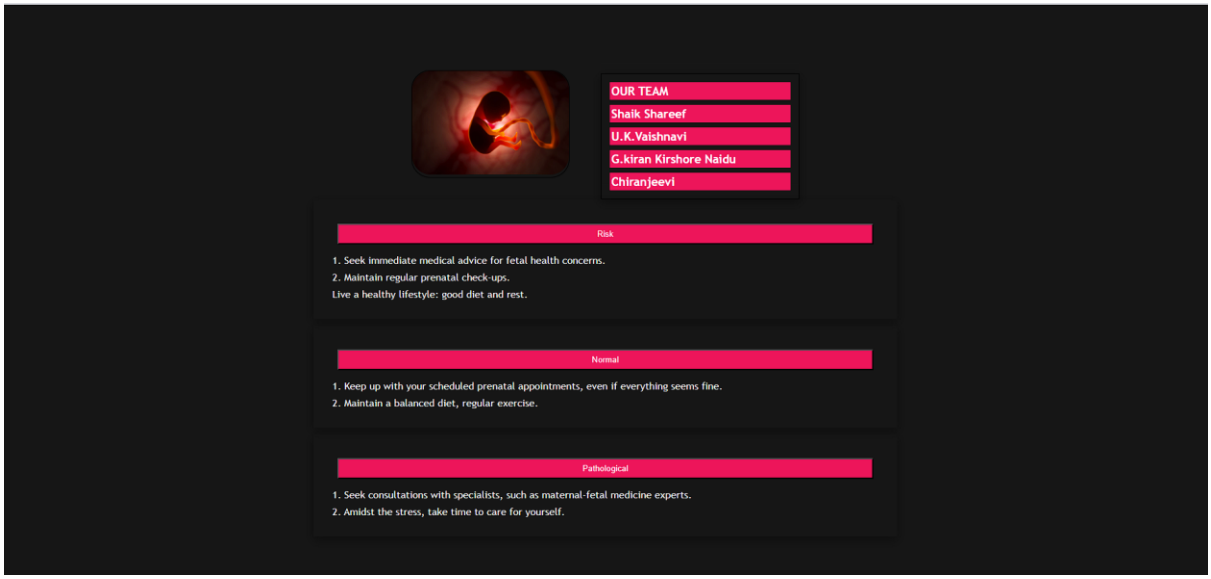
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command ‘
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button,

and see the result/prediction on the web.


Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 8.12.0 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/hp/OneDrive/Desktop/Aiml_Project/app.py', wdir='C:/Users/hp/OneDrive/Desktop/Aiml_Project')
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)


Welcome To Fetal Health Predictor
Login Form
Enter Email :
admin
Enter Password :
Login





Milestone 7: Project Demonstration & Documentation

Outlined below are the required submissions alongside other deliverables.

Activity 1: Produce a comprehensive video explaining the complete project solution from start to finish.

Activity 2: Compile project documentation detailing the step-by-step procedure of project development.

Generate the document in accordance with the provided template.