

**STEP 1: Install Libraries (Run only if needed)**

```
# Install libraries (optional in Colab)
```

```
!pip install pandas numpy scikit-learn tensorflow keras matplotlib
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages (3.8.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!0.5.1,!0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!4.21.1,!4.21.2,!4.21.3,!4.21.4,!4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.21.5)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.14.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.73.1)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.0)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras) (0.16.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.5)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.10.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.8.2)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (0.20.0)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.1.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras) (2.19.2)
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras) (0.1.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,
```

**STEP 2: Import All Required Libraries**

```
# Importing all necessary Python libraries
```

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.preprocessing import StandardScaler
```

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```


```
import joblib # for saving ML model
```

**STEP 3: Upload and Load the Dataset**

```
# Upload the fetal_health.csv file from your computer
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

 Choose Files archive (1).zip

- **archive (1).zip**(application/x-zip-compressed) - 46932 bytes, last modified: 7/10/2025 - 100% done


Saving archive (1).zip to archive (1) (2).zip

# Load the dataset into a pandas DataFrame

```
df = pd.read_csv("/content/archive (1).zip")
```

# Show first 5 rows of the data

```
df.head()
```



	baseline value	accelerations	fetal_movement	uterine_contractions	light_decelerations	severe_decelerations	prolongued_deceleration
0	120.0	0.000	0.0	0.000	0.000	0.0	0
1	132.0	0.006	0.0	0.006	0.003	0.0	0
2	133.0	0.003	0.0	0.008	0.003	0.0	0
3	134.0	0.003	0.0	0.008	0.003	0.0	0
4	132.0	0.007	0.0	0.008	0.000	0.0	0


5 rows × 22 columns

#### STEP 4: Understand and Prepare the Data

```
# Check the structure and classes in the target column
print("Unique fetal health values:", df['fetal_health'].unique())
print("Value Counts:\n", df['fetal_health'].value_counts())

# We convert it into binary: 1 = Normal (Low Risk), 2 or 3 = High Risk
df['fetal_health'] = df['fetal_health'].apply(lambda x: 0 if x == 1 else 1)

# Split the data into inputs (X) and output (y)
X = df.drop("fetal_health", axis=1)
y = df["fetal_health"]
```

 Unique fetal health values: [2. 1. 3.]

Value Counts:

fetal_health	count
1.0	1655
2.0	295
3.0	176

Name: count, dtype: int64


#### STEP 5: Train/Test Split

```
# Split the dataset into 80% training and 20% testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

#### STEP 6: Linear Regression Model

```
#Training a Linear Regression model for basic risk prediction
lr = LinearRegression()
lr.fit(X_train, y_train)

# Evaluate accuracy (R^2 score)
accuracy = lr.score(X_test, y_test)
print("Linear Regression Accuracy:", accuracy)
```

 Linear Regression Accuracy: 0.49729273173944677

#### STEP 7: Deep Neural Network (DNN) Model

```
# Scale input features to improve performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Build the DNN model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_scaled.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid')) # sigmoid for binary output

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

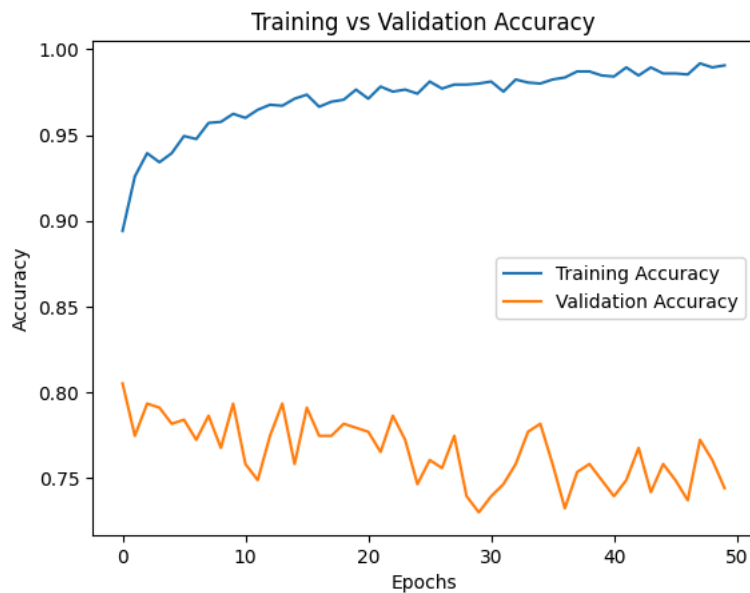
```
# Train the model
```

```
history = model.fit(X_scaled, y, epochs=50, batch_size=8, validation_split=0.2)
```

```
Epoch 23/50
213/213 ————— 1s 3ms/step - accuracy: 0.9795 - loss: 0.0528 - val_accuracy: 0.7653 - val_loss: 1.3198
Epoch 24/50
213/213 ————— 1s 3ms/step - accuracy: 0.9774 - loss: 0.0546 - val_accuracy: 0.7864 - val_loss: 1.1692
Epoch 25/50
213/213 ————— 1s 3ms/step - accuracy: 0.9688 - loss: 0.0612 - val_accuracy: 0.7723 - val_loss: 1.3782
Epoch 26/50
213/213 ————— 1s 2ms/step - accuracy: 0.9809 - loss: 0.0486 - val_accuracy: 0.7465 - val_loss: 1.6530
Epoch 27/50
213/213 ————— 1s 3ms/step - accuracy: 0.9830 - loss: 0.0483 - val_accuracy: 0.7606 - val_loss: 1.5934
Epoch 28/50
213/213 ————— 1s 3ms/step - accuracy: 0.9724 - loss: 0.0585 - val_accuracy: 0.7559 - val_loss: 1.7477
Epoch 29/50
213/213 ————— 1s 3ms/step - accuracy: 0.9814 - loss: 0.0520 - val_accuracy: 0.7746 - val_loss: 1.4307
Epoch 30/50
213/213 ————— 1s 3ms/step - accuracy: 0.9789 - loss: 0.0461 - val_accuracy: 0.7394 - val_loss: 1.7741
Epoch 31/50
213/213 ————— 1s 2ms/step - accuracy: 0.9839 - loss: 0.0439 - val_accuracy: 0.7300 - val_loss: 1.9072
Epoch 32/50
213/213 ————— 1s 3ms/step - accuracy: 0.9840 - loss: 0.0397 - val_accuracy: 0.7394 - val_loss: 1.8370
Epoch 33/50
213/213 ————— 1s 3ms/step - accuracy: 0.9752 - loss: 0.0459 - val_accuracy: 0.7465 - val_loss: 1.7568
Epoch 34/50
213/213 ————— 1s 3ms/step - accuracy: 0.9845 - loss: 0.0345 - val_accuracy: 0.7582 - val_loss: 1.7157
Epoch 35/50
213/213 ————— 1s 3ms/step - accuracy: 0.9835 - loss: 0.0369 - val_accuracy: 0.7770 - val_loss: 1.7045
Epoch 36/50
213/213 ————— 2s 4ms/step - accuracy: 0.9802 - loss: 0.0533 - val_accuracy: 0.7817 - val_loss: 1.7275
Epoch 37/50
213/213 ————— 1s 5ms/step - accuracy: 0.9777 - loss: 0.0680 - val_accuracy: 0.7582 - val_loss: 1.7087
Epoch 38/50
213/213 ————— 1s 3ms/step - accuracy: 0.9877 - loss: 0.0358 - val_accuracy: 0.7324 - val_loss: 1.9108
Epoch 39/50
213/213 ————— 1s 3ms/step - accuracy: 0.9946 - loss: 0.0286 - val_accuracy: 0.7535 - val_loss: 1.8109
Epoch 40/50
213/213 ————— 1s 3ms/step - accuracy: 0.9889 - loss: 0.0340 - val_accuracy: 0.7582 - val_loss: 1.8501
Epoch 41/50
213/213 ————— 1s 3ms/step - accuracy: 0.9867 - loss: 0.0322 - val_accuracy: 0.7488 - val_loss: 1.9253
Epoch 42/50
213/213 ————— 1s 3ms/step - accuracy: 0.9867 - loss: 0.0338 - val_accuracy: 0.7394 - val_loss: 2.1718
Epoch 43/50
213/213 ————— 1s 3ms/step - accuracy: 0.9879 - loss: 0.0358 - val_accuracy: 0.7488 - val_loss: 2.0164
Epoch 44/50
213/213 ————— 1s 3ms/step - accuracy: 0.9819 - loss: 0.0500 - val_accuracy: 0.7676 - val_loss: 1.8526
Epoch 45/50
213/213 ————— 1s 3ms/step - accuracy: 0.9919 - loss: 0.0241 - val_accuracy: 0.7418 - val_loss: 2.3175
Epoch 46/50
213/213 ————— 1s 3ms/step - accuracy: 0.9876 - loss: 0.0280 - val_accuracy: 0.7582 - val_loss: 2.1196
Epoch 47/50
213/213 ————— 1s 3ms/step - accuracy: 0.9935 - loss: 0.0245 - val_accuracy: 0.7488 - val_loss: 2.2841
Epoch 48/50
213/213 ————— 1s 3ms/step - accuracy: 0.9839 - loss: 0.0405 - val_accuracy: 0.7371 - val_loss: 2.3592
Epoch 49/50
213/213 ————— 1s 3ms/step - accuracy: 0.9962 - loss: 0.0186 - val_accuracy: 0.7723 - val_loss: 1.9704
Epoch 50/50
213/213 ————— 1s 3ms/step - accuracy: 0.9896 - loss: 0.0266 - val_accuracy: 0.7606 - val_loss: 2.1942
Epoch 50/50
213/213 ————— 1s 3ms/step - accuracy: 0.9909 - loss: 0.0250 - val_accuracy: 0.7441 - val_loss: 2.3639
```

#### STEP 8: Visualize Accuracy and Loss

```
# Plotting training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')
plt.show()
```



#### STEP 9: Test with Dummy Input (Simulate Prediction)

```
# Use correct feature names from your original dataset
feature_names = X.columns.tolist()

# Create a dummy input with 21 values (make sure these are just example values)
dummy_values = [120, 0.003, 0.5, 0.2, 0.003, 120, 0.002, 0.6, 2.5, 1.2, 0.005, 0.4, 0.1, 0.003, 0.002, 0.003, 0.003, 0.4, 0.1, 0.6, 0.2]

# Convert to DataFrame with correct column names
test_input_df = pd.DataFrame([dummy_values], columns=feature_names)

# Scale using the same scaler
test_input_scaled = scaler.transform(test_input_df)

# Predict
risk_score = model.predict(test_input_scaled)[0][0]
print("Risk Score:", risk_score)
print("Prediction:", "High Risk" if risk_score > 0.5 else "Low Risk")
```



1/1 — 0s 71ms/step  
Risk Score: 1.0  
Prediction: High Risk

#### Step 10: Save Models and Scaler in Google Colab

```
# Save Linear Regression Model
import joblib
joblib.dump(lr, "linear_model.pkl")

# Save Scaler used for DNN model
joblib.dump(scaler, "scaler.pkl")

# Save DNN Model
model.save("dnn_model.h5")

print("✅ Models and scaler saved successfully!")
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is deprecated. Use `model.save(format='tf')` to save as a SavedModel instead.

✅ Models and scaler saved successfully!

```
from google.colab import files
```

```
# Download all saved files
files.download("linear_model.pkl")
files.download("scaler.pkl")
files.download("dnn_model.h5")
```

