CDC Wonder Natality Dataset

Import Libraries

In [271]:

```
import numpy as np
import pandas as pd
from scipy import stats
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from scipy.stats import norm
from collections import Counter
import matplotlib.pyplot as plt
%matplotlib inline
%config IPCompleter.greedy=True
sns.set(color_codes=True)

import warnings
warnings.filterwarnings('ignore')
```

Loading the data

In [272]:
```
df = pd.read_csv(r"C:\Users\chris\OneDrive\Desktop\ML Project.csv")
df
```

Out[272]:

	State	State Code	Pregnancy-associated Hypertension	Pregnancy-associated Hypertension Code	Delivery Method	Delivery Method Code	Diabetes	Diabetes Code
0	Alabama	1	Yes	1	Vaginal	1	Yes	1
1	Alabama	1	Yes	1	Vaginal	1	Yes	1
2	Alabama	1	Yes	1	Vaginal	1	No	2
3	Alabama	1	Yes	1	Vaginal	1	No	2
4	Alabama	1	Yes	1	Cesarean	2	Yes	1
...	...	...	...	...	...	...	...	...
992	Wyoming	56	No	2	Vaginal	1	No	2
993	Wyoming	56	No	2	Cesarean	2	Yes	1
994	Wyoming	56	No	2	Cesarean	2	Yes	1
995	Wyoming	56	No	2	Cesarean	2	No	2
996	Wyoming	56	No	2	Cesarean	2	No	2

997 rows × 20 columns

In [273]:
```
df.head()
```

Out[273]:

	State	State Code	Pregnancy-associated Hypertension	Pregnancy-associated Hypertension Code	Delivery Method	Delivery Method Code	Diabetes	Diabetes Code	Ec
0	Alabama	1	Yes	1	Vaginal	1	Yes	1	
1	Alabama	1	Yes	1	Vaginal	1	Yes	1	
2	Alabama	1	Yes	1	Vaginal	1	No	2	
3	Alabama	1	Yes	1	Vaginal	1	No	2	
4	Alabama	1	Yes	1	Cesarean	2	Yes	1	

```
In [274]:  df.shape

Out[274]: (997, 20)

In [275]:  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 997 entries, 0 to 996
Data columns (total 20 columns):
 #   Column                                             Non-Null Count
Dtype
---  ------                                             --------------
-----
 0   State                                              997 non-null
object
 1   State Code                                         997 non-null
int64
 2   Pregnancy-associated Hypertension                  997 non-null
object
 3   Pregnancy-associated Hypertension Code             997 non-null
int64
 4   Delivery Method                                    997 non-null
object
 5   Delivery Method Code                               997 non-null
int64
 6   Diabetes                                           997 non-null
object
 7   Diabetes Code                                      997 non-null
int64
 8   Eclampsia                                          997 non-null
object
 9   Eclampsia Code                                     997 non-null
int64
 10  Births                                             997 non-null
int64
 11  % of Total Births                                  997 non-null
object
 12  Average Birth Weight                               997 non-null
float64
 13  Standard Deviation for Average Birth Weight        997 non-null
float64
 14  Average Age of Mother                              997 non-null
float64
 15  Standard Deviation for Average Age of Mother       997 non-null
float64
 16  Unnamed: 16                                        997 non-null
float64
 17  Standard Deviation for Average LMP Gestational Age 997 non-null
float64
 18  Unnamed: 18                                        0 non-null
float64
 19  Prediction for Gestation Term                      997 non-null
object
dtypes: float64(7), int64(6), object(7)
memory usage: 155.9+ KB
```

Missing Data

In [276]:
```
##Finding out how many NaN values are there
df.isnull().head()
```

Out[276]:

	State	State Code	Pregnancy-associated Hypertension	Pregnancy-associated Hypertension Code	Delivery Method	Delivery Method Code	Diabetes	Diabetes Code	Eclamp
0	False	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	Fa

True=has null value False=not has the null value

According to this heatmap,we see that that standard deviation has more number of NaN values

EDA and Visualizations using Matplotlib and Seaborn

In [277]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [278]:
```
df.rename(columns={'Unnamed: 16':'Gestation time'},inplace=True)
```

In [279]:
```
print(df.columns)
```

```
Index(['State', 'State Code', 'Pregnancy-associated Hypertension',
       'Pregnancy-associated Hypertension Code', 'Delivery Method',
       'Delivery Method Code', 'Diabetes', 'Diabetes Code', 'Eclampsia',
       'Eclampsia Code', 'Births', '% of Total Births', 'Average Birth W
eight',
       'Standard Deviation for Average Birth Weight', 'Average Age of Mo
ther',
       'Standard Deviation for Average Age of Mother', 'Gestation time',
       'Standard Deviation for Average LMP Gestational Age', 'Unnamed: 1
8',
       'Prediction for Gestation Term'],
      dtype='object')
```

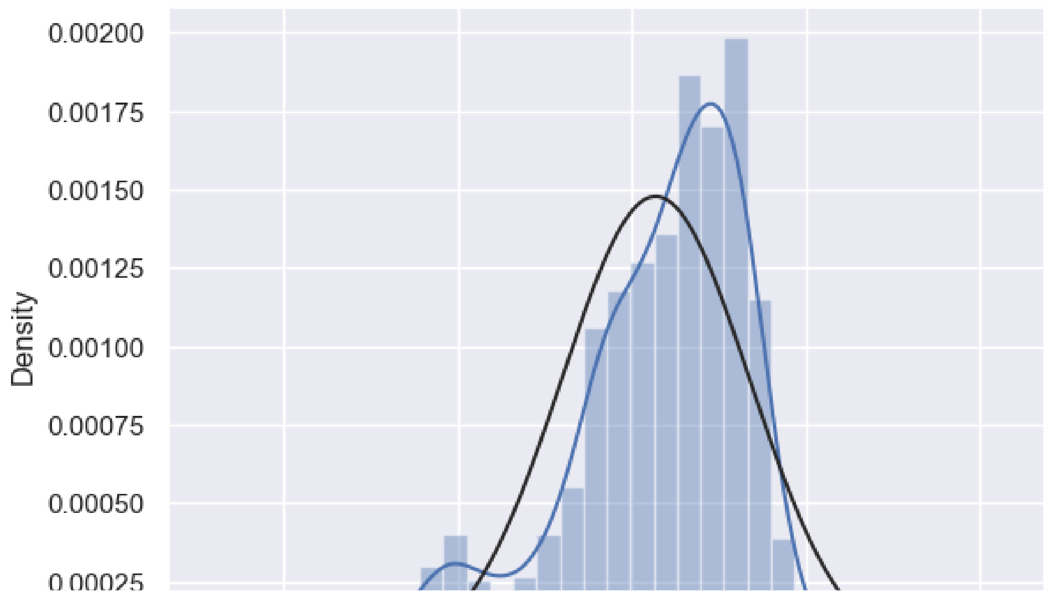
In [280]:
```
df.rename(columns={'Unnamed: 18':'Standard Deviation for Gestation'},inpla
```

In [281]:
```
print(df.columns)
```

```
Index(['State', 'State Code', 'Pregnancy-associated Hypertension',
       'Pregnancy-associated Hypertension Code', 'Delivery Method',
       'Delivery Method Code', 'Diabetes', 'Diabetes Code', 'Eclampsia',
       'Eclampsia Code', 'Births', '% of Total Births', 'Average Birth W
eight',
       'Standard Deviation for Average Birth Weight', 'Average Age of Mo
ther',
       'Standard Deviation for Average Age of Mother', 'Gestation time',
       'Standard Deviation for Average LMP Gestational Age',
       'Standard Deviation for Gestation', 'Prediction for Gestation Ter
m'],
      dtype='object')
```

In [282]:
```
sns.distplot(df['Average Birth Weight'],fit = norm)
```

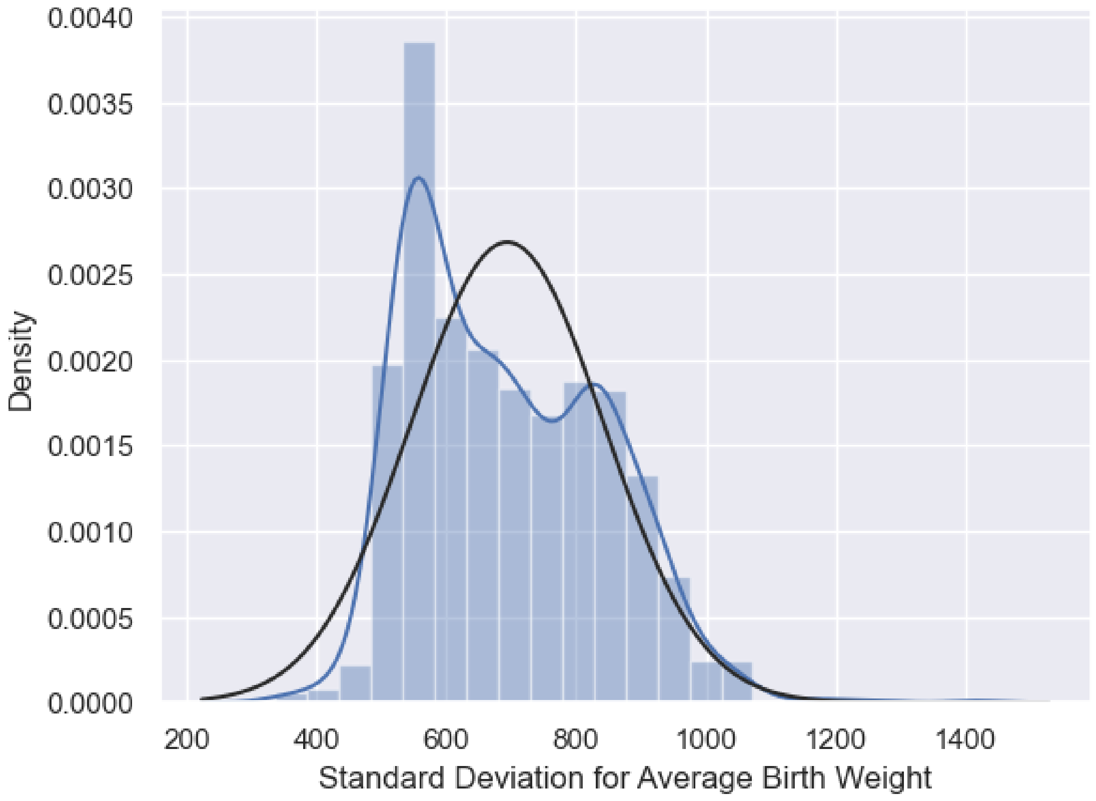
Out[282]:
```
<AxesSubplot:xlabel='Average Birth Weight', ylabel='Density'>
```



Here the graph is left skewed, this could be due to a variety of factors, such as a higher proportion of premature births or other health conditions that affect birth weight.

In [283]:
```
sns.distplot(df['Standard Deviation for Average Birth Weight'],fit = norm)
```

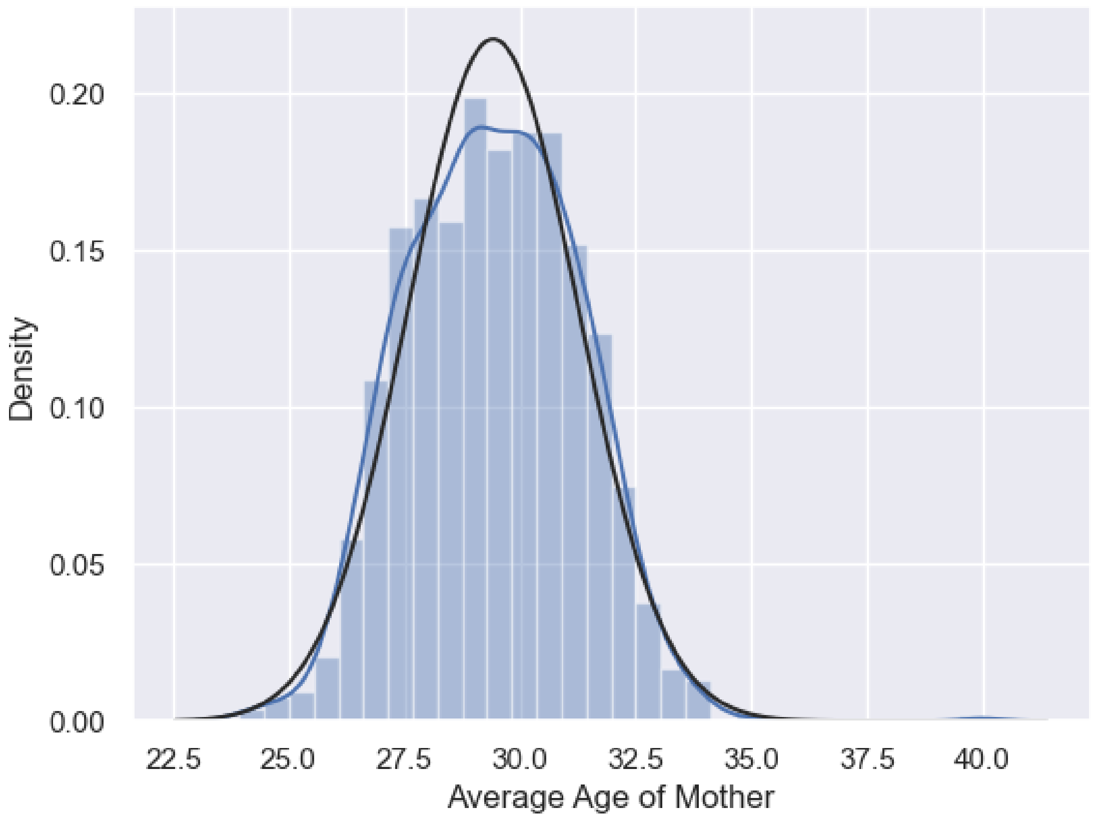
Out[283]:
```
<AxesSubplot:xlabel='Standard Deviation for Average Birth Weight', ylabe
l='Density'>
```



The plot above is zero skewed,which means that the distribution of the standard deviation values is symmetrical around the mean, with an equal number of values above and below the mean. This could be indicative of a random and uniform spread of birth weight values in the dataset, without any specific patterns or biases.

In [284]:
```
sns.distplot(df['Average Age of Mother'],fit = norm)
```

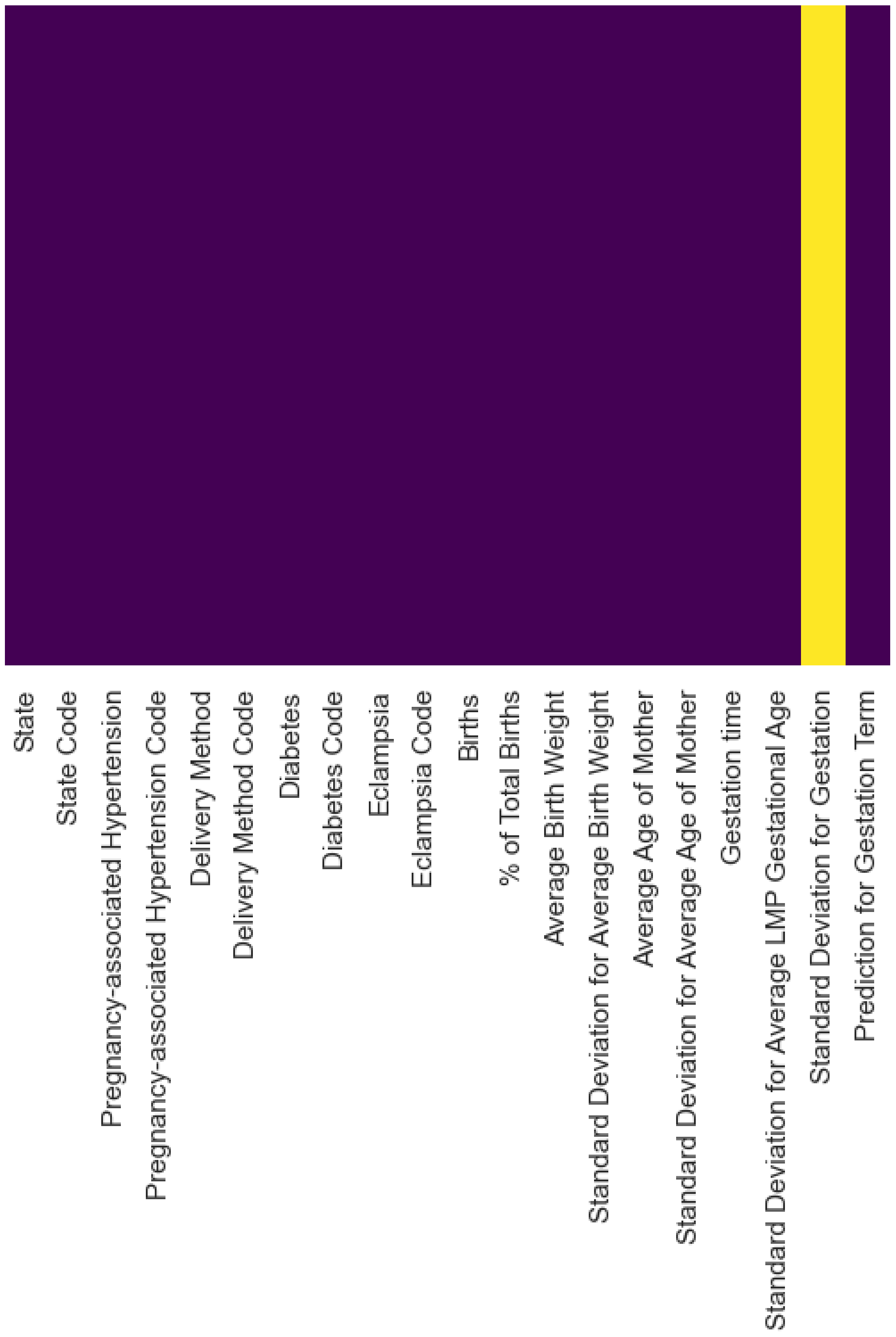
Out[284]:
```
<AxesSubplot:xlabel='Average Age of Mother', ylabel='Density'>
```



The plot above is zero skewed,which means that the majority of the ages of the mothers in the dataset fall within a certain range around the mean age, with fewer ages at the extremes.

In [285]:
```
##This is difficult to see here how many values are true or false
##Using Visualisation concept
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

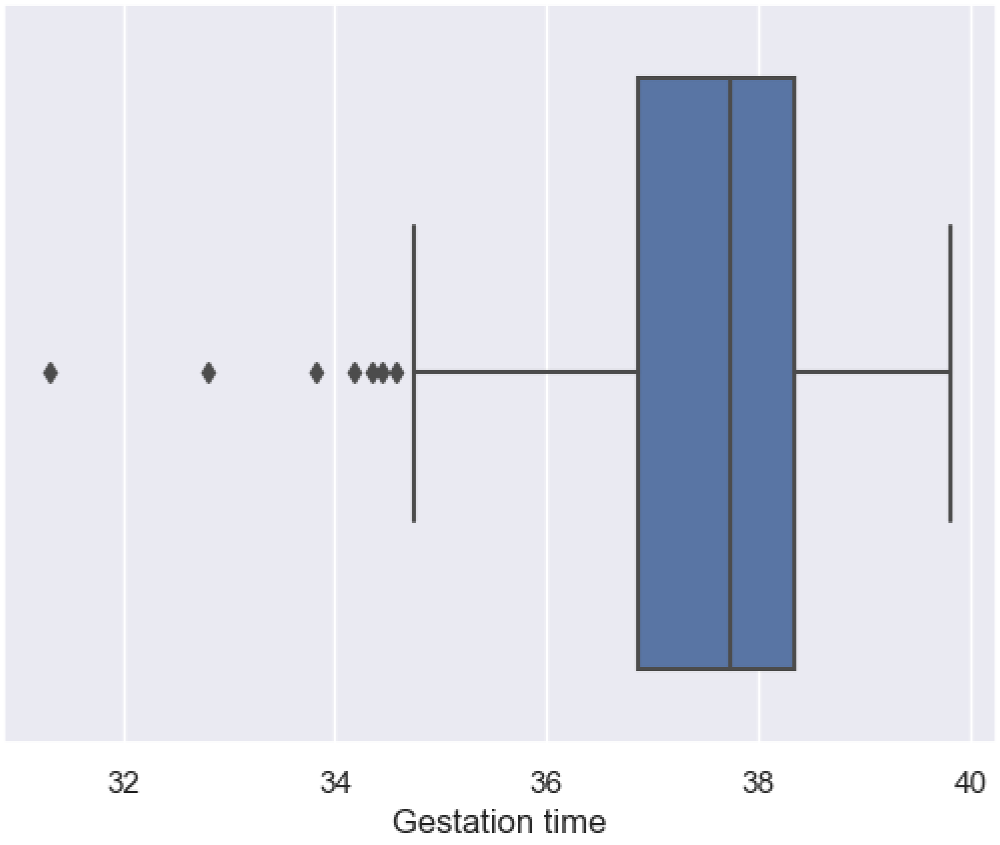
Out[285]:
```
<AxesSubplot:>
```



Here, the heatmap shows that there are missing values only in the 'Standard Deviation for Gestation' column of the DataFrame df. This means that there are some rows in the DataFrame where the value for the 'Standard Deviation for Gestation' feature is not available or is missing. It is important to handle such missing or null values in data preprocessing before

In [286]:
```
sns.boxplot(x=df['Gestation time'])
```

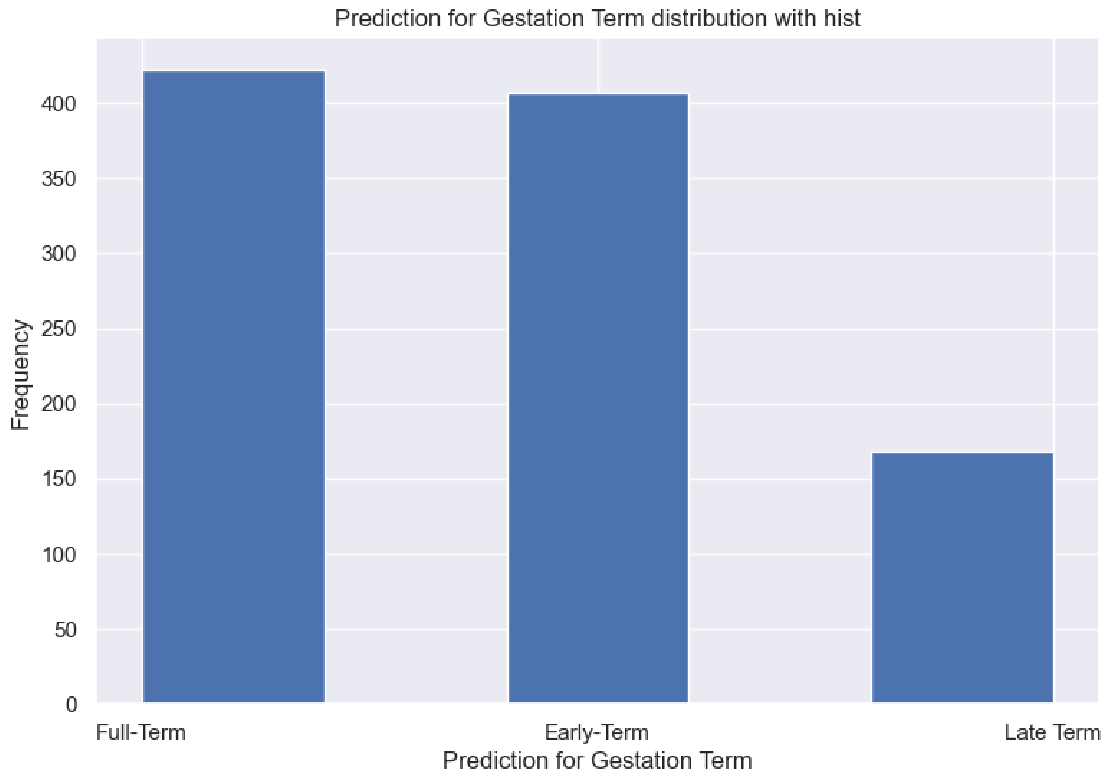
Out[286]:
```
<AxesSubplot:xlabel='Gestation time'>
```



Since the median is closer to the right of the box plot, the distribution is left skewed, i.e the late term is very less compared to the Early trm and full term gestation period.The outliers in the boxplot are the individual data points that are located outside the whiskers. In this case, there are few outliers below the lower whisker, which means there are some Gestation time values that are much lower than the majority of the data. These outliers could be the result of measurement errors or some other factors affecting the pregnancy.

In [287]:
```
def plot_hist(variable):
    plt.figure(figsize = (9,6))
    plt.hist(df[variable], bins=5)
    plt.xlabel(variable)
    plt.ylabel("Frequency")
    plt.title("{} distribution with hist".format(variable))
    plt.show()
```

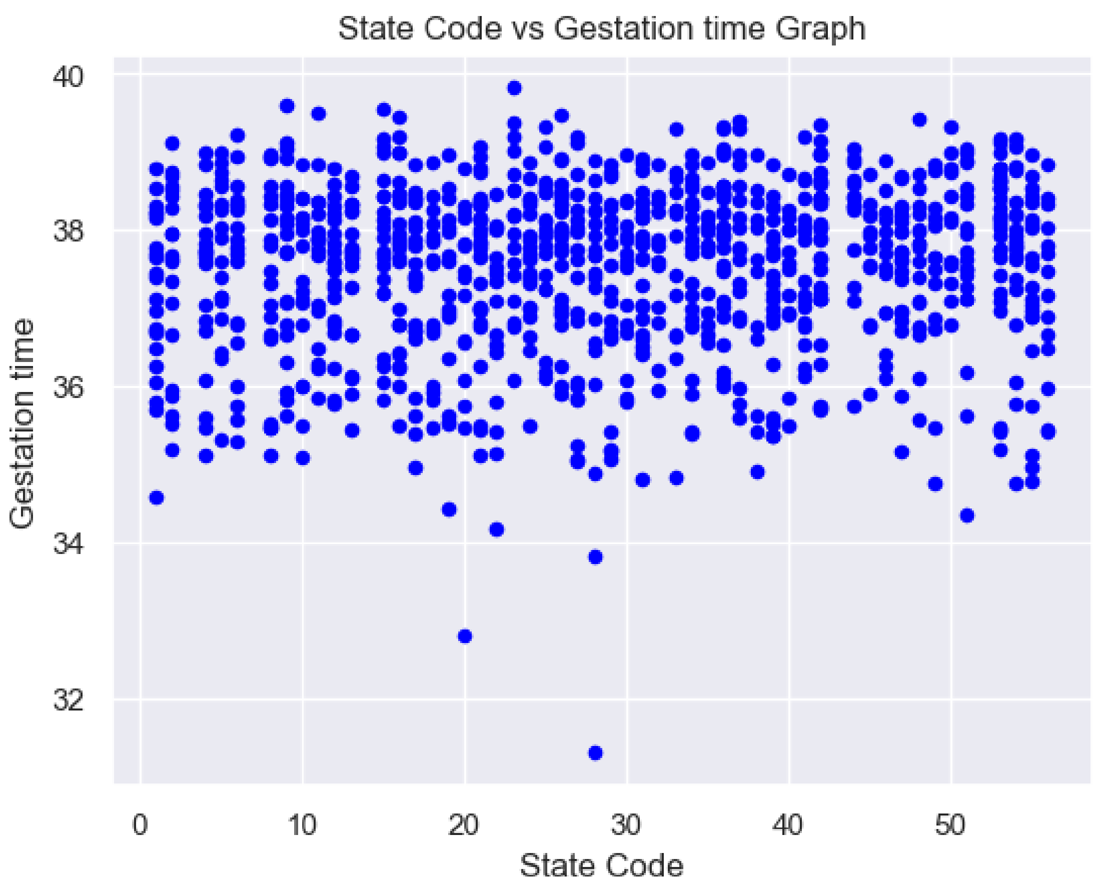
In [288]:
```
numericVariables = ["Prediction for Gestation Term"]
for i in numericVariables:
    plot_hist(i)
```

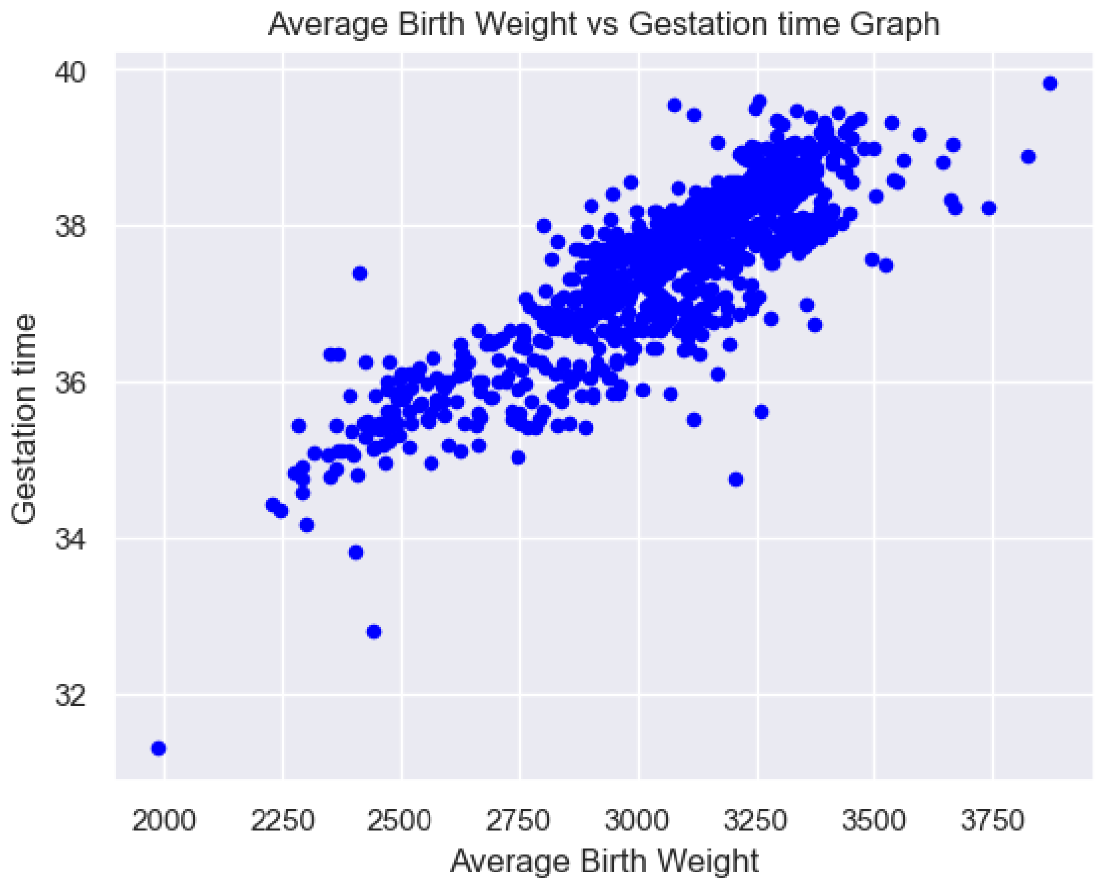


The histogram shows that there are 420 instances where the prediction is full term, 410 instances where the prediction is early term, and 165 instances where the prediction is late term. The histogram also shows that the distribution is somewhat skewed towards the left, indicating that there are more instances where the prediction is full term or early term than late term

In [289]:

```
def scatter_plot(x, y, color='blue'):
    df.plot.scatter(x=x, y=y, color=color)
    plt.title(x + " vs " + y + " Graph")
    plt.xlabel(x)
    plt.show()

Variables = ["State Code","Average Birth Weight"]
for i in Variables:
    scatter_plot(i, "Gestation time", color='blue')
```

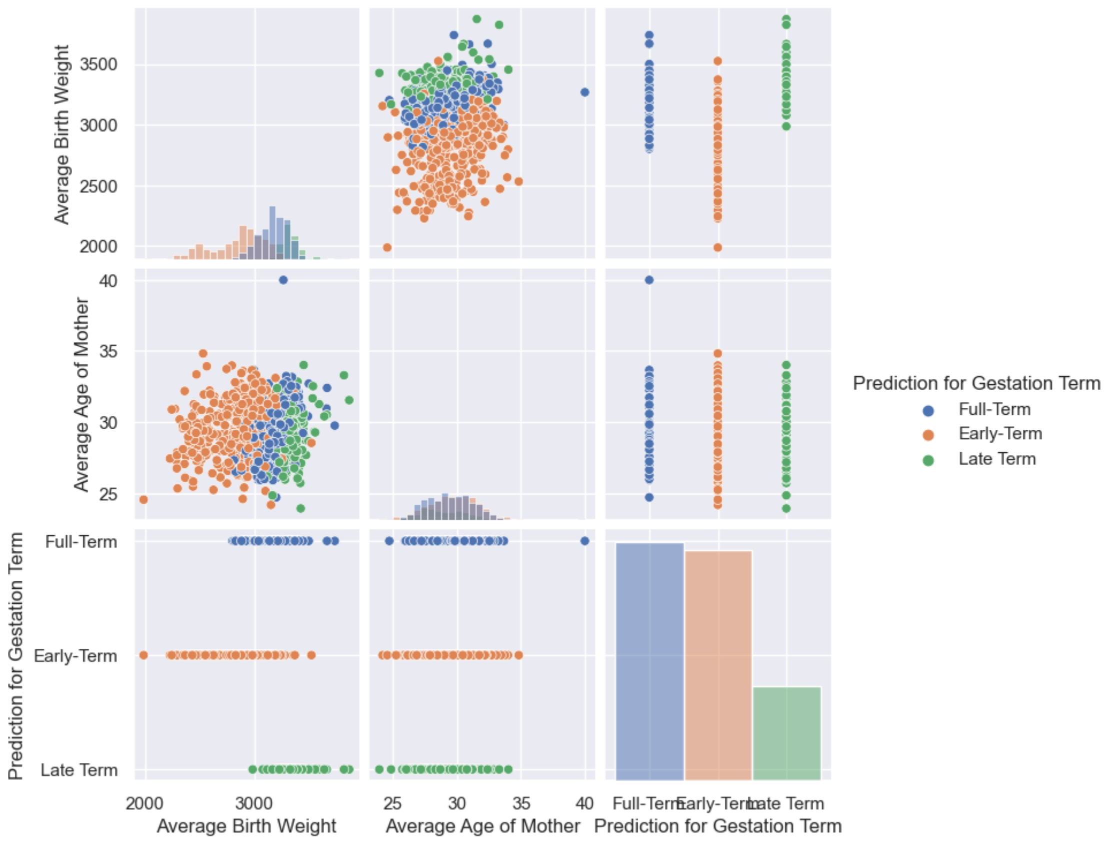




The dots scattered across the graph suggest that there may be a positive correlation between gestation time and the State Code. In other words, as the value of the variable on the state code increases, so does the gestation time.

The second graph appears as a linear line indicates a positive correlation between the two variables. This means that as the gestation time increases, the average birth weight also tends to increase. The dots being closer to the top right corner suggest a strong positive correlation between the variables, which means that the increase in the gestation time is highly likely to be associated with an increase in the average birth weight

```
In [290]:  sns.pairplot(df, hue="Prediction for Gestation Term", kind = 'scatter', va
                   'Average Birth Weight', 'Average Age of Mother',
                 'Prediction for Gestation Term'],diag_kind = 'hist')

Out[290]: <seaborn.axisgrid.PairGrid at 0x1aec4a1e850>
```



Here the three different gestation periods are compared and hence the correlation and relationships between different independent variables can be displayed which are further helpful in feature selection and dimensionality reduction.

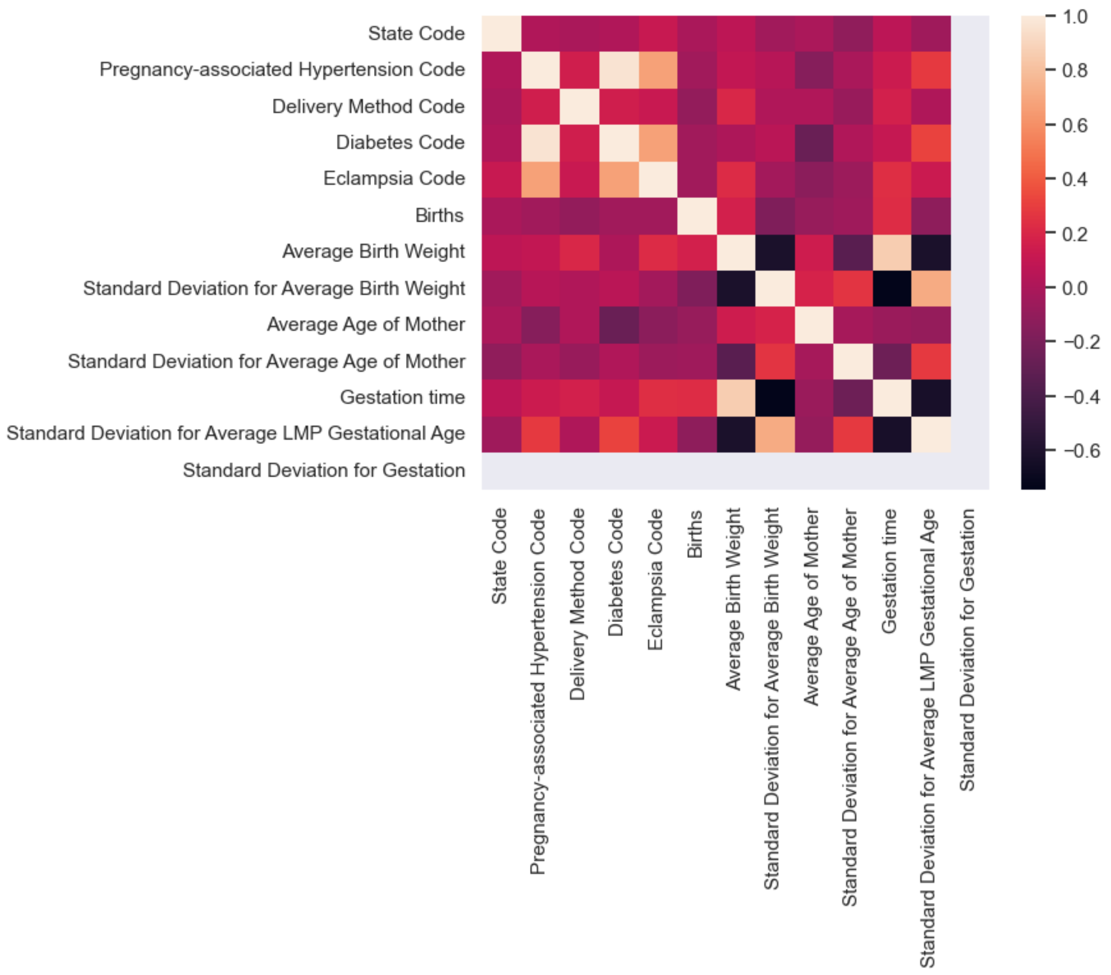
In [291]:
```
df.corr()
```

Out[291]:

	State Code	Pregnancy-associated Hypertension Code	Delivery Method Code	Diabetes Code	Eclampsia Code	Births	Averag Birt Weig
State Code	1.000000	0.018514	-0.013342	0.013586	0.113958	-0.009248	0.07142
Pregnancy-associated Hypertension Code	0.018514	1.000000	0.150128	0.961507	0.666568	-0.045633	0.09111
Delivery Method Code	-0.013342	0.150128	1.000000	0.151809	0.115118	-0.101146	0.20083
Diabetes Code	0.013586	0.961507	0.151809	1.000000	0.669212	-0.044811	0.00022
Eclampsia Code	0.113958	0.666568	0.115118	0.669212	1.000000	-0.047405	0.21952
Births	-0.009248	-0.045633	-0.101146	-0.044811	-0.047405	1.000000	0.16687
Average Birth Weight	0.071426	0.091110	0.200832	0.000220	0.219525	0.166875	1.00000
Standard Deviation for Average Birth Weight	-0.048422	0.040476	0.014447	0.053346	-0.035596	-0.181859	-0.61699
Average Age of Mother	-0.005853	-0.153659	0.013476	-0.273485	-0.134077	-0.088304	0.13604
Standard Deviation for Average Age of Mother	-0.111346	-0.011486	-0.076458	0.016694	-0.067870	-0.053692	-0.34007
Gestation time	0.062132	0.130087	0.168011	0.102586	0.235994	0.229161	0.85720
Standard Deviation for Average LMP Gestational Age	-0.055824	0.279663	0.005460	0.317522	0.121932	-0.119646	-0.62086
Standard Deviation for Gestation	NaN	NaN	NaN	NaN	NaN	NaN	Na

```
In [292]: sns.heatmap(df.corr())
```

```
Out[292]: <AxesSubplot:>
```



From the above heatmap,we can conclude that there is high correlation between the Average Birth Weight and Standard Deviation for Average LMP Gestational Age,Gestation time and Standard Deviation for Average Birth Weight,Gestation time and Standard Deviation for Average LMP Gestational Age

In [293]:
```
df.describe()
```

Out[293]:

	State Code	Pregnancy-associated Hypertension Code	Delivery Method Code	Diabetes Code	Eclampsia Code	Births	Av W
count	997.000000	997.000000	997.000000	997.000000	997.000000	9.970000e+02	997.0
mean	28.488465	2.422267	2.423270	2.407222	3.148445	3.858516e+04	3069.5
std	15.638586	2.421711	2.512495	2.427393	3.170527	1.665898e+05	270.0
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000e+01	1987.8
25%	16.000000	1.000000	1.000000	1.000000	1.000000	8.600000e+01	2929.4
50%	28.000000	2.000000	2.000000	2.000000	2.000000	7.440000e+02	3127.4
75%	41.000000	2.000000	2.000000	2.000000	2.000000	1.089700e+04	3267.6
max	56.000000	9.000000	9.000000	9.000000	10.000000	2.911179e+06	3870.3

In [294]:
```
#Find the duplicates

df.duplicated().sum()
```

Out[294]:
```
0
```

In [295]:
```
encoder = LabelEncoder()
y = encoder.fit_transform(y)
print(y)
```

```
[1 0 0 1 0 0 0 0 1 0 1 0 2 0 1 0 1 1 2 0 0 0 0 1 1 1 0 0 0 0 0 2 1 2 0 1
0
 1 2 2 1 2 1 1 1 1 0 0 0 0 1 1 0 1 1 2 0 1 0 1 1 2 1 0 1 0 0 0 2 0 1 0 2
0
 1 0 1 2 2 2 1 1 1 1 0 0 0 0 1 2 1 2 0 1 0 1 2 0 1 0 1 0 1 0 0 0 0 0 1 0
2
 0 1 0 1 2 2 2 0 1 1 1 0 0 0 0 1 0 2 1 2 0 1 0 1 1 2 2 1 2 1 1 0 1 0 0 0
0
 1 0 2 0 1 0 1 1 0 1 1 1 0 0 0 1 1 1 2 0 1 0 1 2 1 0 0 1 1 1 0 0 0 0 1 0
1
 1 2 0 1 0 1 1 2 1 0 0 1 1 1 0 0 0 0 0 1 1 1 2 0 1 0 1 1 1 1 1 2 0 1 1 1
0
 1 0 0 2 1 2 1 2 0 1 0 1 1 1 2 0 2 1 1 1 1 0 0 0 0 0 1 2 1 2 2 0 1 1 0 1
1
 2 1 2 0 0 0 1 0 1 0 0 0 0 0 0 1 0 2 0 1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1
2
 0 1 0 1 1 1 0 1 0 1 0 0 0 0 1 2 1 2 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 2 0 1
0
 1 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 2 2 0 1 1 0 1 1 2 2 1 1 1 1 1 1 0 0
0
 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 2 0 1 0 0 2 1 2 0 1 0 2 2 2 2 1 0 1 1 1 0
0
 0 0 1 0 1 1 2 0 1 0 1 1 2 1 0 0 1 1 2 0 0 1 1 0 2 1 2 0 1 0 1 0 1 2 2 1
1
 0 1 1 0 0 0 0 0 0 1 1 2 1 2 2 0 1 1 0 1 1 2 1 1 1 1 1 1 0 1 0 1 0 0 0 0
0
 2 1 2 0 1 0 1 2 1 0 1 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 2 2 0 0 0 0 1 0 1 0
0
 0 0 1 0 1 0 2 0 1 0 1 2 2 1 1 1 0 0 0 1 0 2 0 1 0 1 1 1 1 1 0 1 1 0 0 0
0
 0 0 1 1 1 2 2 0 1 1 0 1 1 1 1 0 2 1 1 0 0 1 1 1 2 0 1 0 1 1 1 0 0 2 0 2
0
 1 0 2 0 1 1 1 0 1 0 0 0 0 1 1 0 2 1 2 0 1 0 1 2 2 1 0 2 1 1 1 0 1 0 1 1
1
 2 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 2 0 2 2 1 2 2 0 1 1 0 1 1 2 2 2 1 1
1
 2 2 1 1 1 0 0 0 0 2 1 2 0 1 0 1 2 2 1 0 2 0 1 0 0 0 2 1 2 0 1 0 1 0 1 0
1
 0 0 0 0 1 0 1 0 2 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 0 2 0 1 0 1 1 0 1 1 1 0
0
 0 0 0 2 1 2 0 1 0 2 0 0 1 1 1 1 1 0 0 0 0 0 0 0 2 2 1 2 2 0 1 1 0 1 1 1
2
 2 2 0 1 2 2 1 1 0 0 2 1 2 1 0 2 2 2 1 1 1 1 0 0 0 1 2 1 1 1 1 1 1 1 0 0
1
 2 0 2 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 0 2 2 0 1 1 0 1 1 1 1 1 1 0 0 0
0
 0 1 1 2 0 1 0 1 1 2 1 0 2 1 1 0 0 0 1 0 2 0 1 0 1 2 1 2 1 0 0 2 1 2 1 1
2
 0 1 1 1 1 0 0 0 2 0 2 1 2 0 1 0 1 2 2 0 0 0 1 1 1 1 1 2 0 0 0 0 0 0 1 1
2
 2 1 2 2 0 1 1 0 2 2 2 2 2 2 1 2 2 1 1 1 0 0 0 2 1 1 1 2 0 1 0 1 2 2 1 1
0
 0 1 0 1 0 0 0 0 0 2 0 2 0 1 0 1 1 2 1 0 1 1 0 1 0 0 0 0 1 0 2 0 1 0 1]
```

Machine Learning Classification Models

In [296]:
```
test_accuracies_list = []
train_accuracies_list = []
rmse_list = []
precision_score_list = []
recall_score_list = []
F1_score_list = []
```

1.) Decision Trees

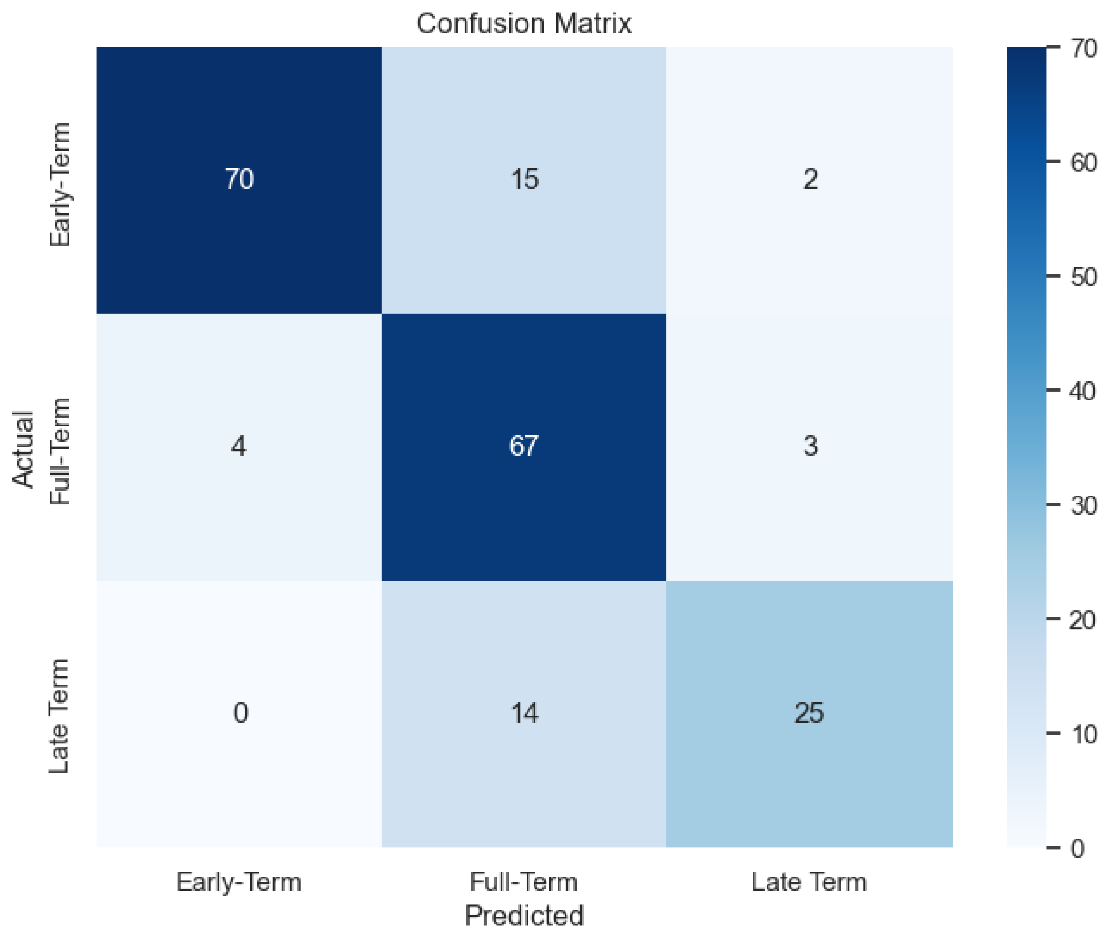
In [297]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_squared_error, confusion_
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import LabelEncoder
```

In [298]:

```
from sklearn.tree import DecisionTreeClassifier

# Select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# Encode the target variable
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Perform feature selection using a decision tree
dt = DecisionTreeClassifier(random_state=0)
sfm = SelectFromModel(dt, threshold=0.1)
X = sfm.fit_transform(X, y_encoded)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_si

# Train the decision tree classifier
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train, y_train)

# Predict the target variable on the training and testing sets
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

# Inverse transform the predicted labels to original values
y_train_pred_original = encoder.inverse_transform(y_train_pred)
y_test_pred_original = encoder.inverse_transform(y_test_pred)

y_test_original = encoder.inverse_transform(y_test)

# Evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)

precision = precision_score(y_test_original, y_test_pred_original, average
recall = recall_score(y_test_original, y_test_pred_original, average='weig
f1 = f1_score(y_test_original, y_test_pred_original, average='weighted')

test_accuracies_list.append(test_acc)
train_accuracies_list.append(train_acc)
rmse_list.append(rmse)
precision_score_list.append(precision)
recall_score_list.append(recall)
F1_score_list.append(f1)

print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)
print('RMSE:', rmse)

# Compute confusion matrix
```

```
cm = confusion_matrix(y_test_original, y_test_pred_original)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=encoder.cla
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Training Accuracy: 0.8042659974905897
Testing Accuracy: 0.81
Precision: 0.832215653153153
Recall: 0.81
F1 Score: 0.8112122762148338
RMSE: 0.469041575982343
```



Based on the results,the model appears to perform well. The training accuracy of 0.804 and testing accuracy of 0.81 suggest that the model is able to accurately classify new data that it has not seen before. The RMSE value of 0.469 indicates that the model's predictions are on average within half a gestational week of the actual values, which is relatively low and suggests that the model's predictions are generally accurate.The recall score of 0.81 means that out of all the actual positive instances in the dataset, the model correctly identified 81% of them The F1 score of 0.811 is the harmonic mean of precision and recall and is a good

From the matrix, we can see that there were 70 correct predictions for "Full term" births, 67 correct predictions for "Early term" births, and 25 correct predictions for "Late term" births. There were also some incorrect predictions, such as 15 predictions for "Early term" births that were actually "Full term" and 14 predictions for "Late term" births that were actually "Early term".

```
In [299]: dt.predict(X_train)
```

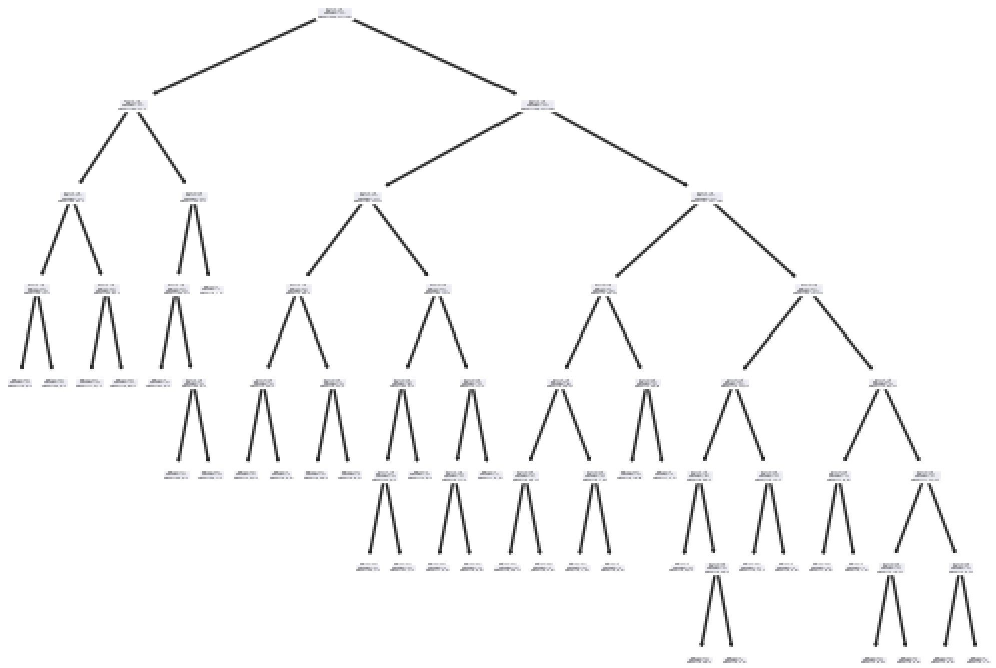
```
Out[299]: array([1, 0, 0, 0, 2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 2, 0, 1, 2, 2,
                 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 2, 1, 1, 1, 1, 1, 2, 0,
                 1, 0, 0, 1, 1, 2, 1, 2, 1, 2, 1, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0,
                 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0,
                 1, 1, 1, 0, 1, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
                 1, 1, 2, 1, 0, 0, 0, 1, 1, 1, 1, 0, 2, 2, 1, 0, 1, 1, 2, 0, 0, 0,
                 0, 1, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 1, 1, 1, 2, 1, 0, 1, 0, 0, 1,
                 1, 1, 1, 2, 2, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 2, 2, 1,
                 0, 1, 0, 1, 1, 0, 2, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 0,
                 1, 0, 0, 1, 1, 1, 1, 1, 1, 2, 0, 1, 2, 1, 2, 1, 1, 2, 0, 0, 0, 0,
                 0, 2, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 2,
                 2, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 2, 1, 1, 1,
                 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
                 1, 1, 1, 1, 1, 2, 1, 0, 1, 1, 2, 1, 1, 2, 0, 0, 0, 2, 1, 1, 1, 0,
                 2, 1, 0, 1, 1, 1, 2, 0, 1, 1, 0, 1, 1, 1, 0, 2, 0, 1, 0, 0, 0, 0,
                 0, 1, 1, 1, 1, 2, 1, 0, 0, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 0, 2, 1, 1, 1, 1, 1, 2, 0, 1, 1, 1, 0, 2, 1, 1, 1, 1, 1,
                 2, 0, 1, 1, 0, 2, 0, 1, 1, 0, 2, 2, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
                 1, 1, 1, 2, 1, 1, 1, 0, 2, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
                 1, 1, 0, 1, 1, 0, 2, 0, 1, 0, 1, 1, 2, 1, 1, 0, 1, 1, 1, 0, 1, 2,
                 1, 1, 0, 1, 1, 0, 1, 0, 2, 1, 1, 2, 0, 0, 1, 1, 0, 2, 1, 1, 0, 1,
                 1, 0, 1, 0, 0, 1, 0, 2, 0, 1, 1, 0, 0, 1, 2, 1, 1, 0, 0, 0, 2, 1,
                 2, 0, 2, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 2, 1, 0, 1, 1,
                 1, 1, 0, 1, 0, 1, 0, 2, 1, 0, 1, 1, 0, 1, 1, 0, 0, 2, 0, 1, 0, 2,
                 1, 0, 1, 2, 1, 1, 1, 0, 0, 1, 2, 1, 1, 0, 1, 1, 0, 2, 1, 0, 1, 1,
                 1, 2, 1, 1, 2, 0, 2, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 2, 1, 1, 1,
                 0, 2, 2, 2, 0, 1, 2, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 2,
                 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
                 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 1, 2, 1, 2, 0, 1, 1, 0,
                 0, 1, 1, 1, 0, 1, 0, 1, 0, 2, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 2, 1,
                 2, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 2, 0, 1, 2, 0, 0, 0, 1, 1, 1, 1,
                 1, 0, 1, 1, 2, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                 1, 0, 0, 2, 1, 2, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,
                 0, 1, 2, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 2, 1, 0,
                 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 2, 0, 2, 2, 1, 1,
                 2, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
                 2, 2, 2, 1, 1])
```

```
In [300]: dt.score(X_train, y_train)
```

```
Out[300]: 0.8042659974905897
```

In [301]:
```
from sklearn import tree
tree.plot_tree(dt)
```

Out[301]:
```
[Text(0.33497807017543857, 0.9375, 'X[3] <= 1.5\ngini = 0.622\nsamples =
797\nvalue = [320, 348, 129]'),
 Text(0.13157894736842105, 0.8125, 'X[1] <= 1.5\ngini = 0.38\nsamples =
257\nvalue = [192, 64, 1]'),
 Text(0.07017543859649122, 0.6875, 'X[2] <= 1.5\ngini = 0.507\nsamples =
127\nvalue = [66, 60, 1]'),
 Text(0.03508771929824561, 0.5625, 'X[0] <= 1.5\ngini = 0.384\nsamples =
54\nvalue = [40, 14, 0]'),
 Text(0.017543859649122806, 0.4375, 'gini = 0.375\nsamples = 20\nvalue =
[15, 5, 0]'),
 Text(0.05263157894736842, 0.4375, 'gini = 0.389\nsamples = 34\nvalue =
[25, 9, 0]'),
 Text(0.10526315789473684, 0.5625, 'X[0] <= 1.5\ngini = 0.476\nsamples =
73\nvalue = [26, 46, 1]'),
 Text(0.08771929824561403, 0.4375, 'gini = 0.483\nsamples = 33\nvalue =
[11, 21, 1]'),
 Text(0.12280701754385964, 0.4375, 'gini = 0.469\nsamples = 40\nvalue =
[15, 25, 0]'),
 Text(0.19298245614035087, 0.6875, 'X[1] <= 5.5\ngini = 0.06\nsamples =
130\nvalue = [126, 4, 0]'),
 Text(0.17543859649122806, 0.5625, 'X[2] <= 1.5\ngini = 0.045\nsamples =
129\nvalue = [126, 3, 0]'),
 Text(0.15789473684210525, 0.4375, 'gini = 0.0\nsamples = 57\nvalue = [5
7, 0, 0]'),
 Text(0.19298245614035087, 0.4375, 'X[0] <= 1.5\ngini = 0.08\nsamples =
72\nvalue = [69, 3, 0]'),
 Text(0.17543859649122806, 0.3125, 'gini = 0.117\nsamples = 32\nvalue =
[30, 2, 0]'),
 Text(0.21052631578947367, 0.3125, 'gini = 0.049\nsamples = 40\nvalue =
[39, 1, 0]'),
 Text(0.21052631578947367, 0.5625, 'gini = 0.0\nsamples = 1\nvalue = [0,
1, 0]'),
 Text(0.5383771929824561, 0.8125, 'X[0] <= 1.5\ngini = 0.611\nsamples =
540\nvalue = [128, 284, 128]'),
 Text(0.3684210526315789, 0.6875, 'X[1] <= 1.5\ngini = 0.536\nsamples =
208\nvalue = [97, 103, 8]'),
 Text(0.2982456140350877, 0.5625, 'X[2] <= 1.5\ngini = 0.136\nsamples =
97\nvalue = [4, 90, 3]'),
 Text(0.2631578947368421, 0.4375, 'X[3] <= 6.0\ngini = 0.156\nsamples =
47\nvalue = [4, 43, 0]'),
 Text(0.24561403508771928, 0.3125, 'gini = 0.176\nsamples = 41\nvalue =
[4, 37, 0]'),
 Text(0.2807017543859649, 0.3125, 'gini = 0.0\nsamples = 6\nvalue = [0,
6, 0]'),
 Text(0.3333333333333333, 0.4375, 'X[3] <= 6.0\ngini = 0.113\nsamples =
50\nvalue = [0, 47, 3]'),
 Text(0.3157894736842105, 0.3125, 'gini = 0.093\nsamples = 41\nvalue =
[0, 39, 2]'),
 Text(0.3508771929824561, 0.3125, 'gini = 0.198\nsamples = 9\nvalue =
[0, 8, 1]'),
 Text(0.43859649122807015, 0.5625, 'X[1] <= 5.5\ngini = 0.282\nsamples =
111\nvalue = [93, 13, 5]'),
 Text(0.40350877192982454, 0.4375, 'X[3] <= 6.0\ngini = 0.044\nsamples =
88\nvalue = [86, 2, 0]'),
 Text(0.38596491228070173, 0.3125, 'X[2] <= 1.5\ngini = 0.053\nsamples =
74\nvalue = [72, 2, 0]'),
 Text(0.3684210526315789, 0.1875, 'gini = 0.054\nsamples = 36\nvalue =
[35, 1, 0]'),
 Text(0.40350877192982454, 0.1875, 'gini = 0.051\nsamples = 38\nvalue =
[37, 1, 0]'),
 Text(0.42105263157894735, 0.3125, 'gini = 0.0\nsamples = 14\nvalue = [1
4, 0, 0]'),
 Text(0.47368421052631576, 0.4375, 'X[3] <= 6.0\ngini = 0.631\nsamples =
23\nvalue = [7, 11, 5]'),
 Text(0.45614035087719296, 0.3125, 'X[2] <= 1.5\ngini = 0.624\nsamples =
22\nvalue = [6, 11, 5]'),
 Text(0.43859649122807015, 0.1875, 'gini = 0.444\nsamples = 3\nvalue =
[1, 2, 0]'),
 Text(0.47368421052631576, 0.1875, 'gini = 0.637\nsamples = 19\nvalue =
[5, 9, 5]'),
 Text(0.49122807017543857, 0.3125, 'gini = 0.0\nsamples = 1\nvalue = [1,
0, 0]'),
 Text(0.7083333333333334, 0.6875, 'X[1] <= 1.5\ngini = 0.563\nsamples =
332\nvalue = [31, 181, 120]'),
 Text(0.6052631578947368, 0.5625, 'X[2] <= 5.5\ngini = 0.54\nsamples = 1
26\nvalue = [7, 50, 69]'),
 Text(0.5614035087719298, 0.4375, 'X[2] <= 1.5\ngini = 0.444\nsamples =
90\nvalue = [0, 30, 60]'),
 Text(0.5263157894736842, 0.3125, 'X[3] <= 6.0\ngini = 0.473\nsamples =
47\nvalue = [0, 29, 18]'),
 Text(0.5087719298245614, 0.1875, 'gini = 0.476\nsamples = 41\nvalue =
[0, 25, 16]'),
 Text(0.543859649122807, 0.1875, 'gini = 0.444\nsamples = 6\nvalue = [0,
4, 2]'),
 Text(0.5964912280701754, 0.3125, 'X[3] <= 6.0\ngini = 0.045\nsamples =
43\nvalue = [0, 1, 42]'),
 Text(0.5789473684210527, 0.1875, 'gini = 0.051\nsamples = 38\nvalue =
[0, 1, 37]'),
 Text(0.6140350877192983, 0.1875, 'gini = 0.0\nsamples = 5\nvalue = [0,
0, 5]'),
 Text(0.6491228070175439, 0.4375, 'X[3] <= 9.5\ngini = 0.591\nsamples =
36\nvalue = [7, 20, 9]'),
 Text(0.631578947368421, 0.3125, 'gini = 0.604\nsamples = 30\nvalue =
[6, 16, 8]'),
 Text(0.6666666666666666, 0.3125, 'gini = 0.5\nsamples = 6\nvalue = [1,
4, 1]'),
 Text(0.8114035087719298, 0.5625, 'X[1] <= 5.5\ngini = 0.521\nsamples =
206\nvalue = [24, 131, 51]'),
 Text(0.7368421052631579, 0.4375, 'X[2] <= 5.5\ngini = 0.289\nsamples =
127\nvalue = [12, 106, 9]'),
 Text(0.7017543859649122, 0.3125, 'X[2] <= 1.5\ngini = 0.101\nsamples =
94\nvalue = [0, 89, 5]'),
 Text(0.6842105263157895, 0.1875, 'gini = 0.0\nsamples = 44\nvalue = [0,
44, 0]'),
 Text(0.7192982456140351, 0.1875, 'X[3] <= 6.0\ngini = 0.18\nsamples = 5
0\nvalue = [0, 45, 5]'),
 Text(0.7017543859649122, 0.0625, 'gini = 0.206\nsamples = 43\nvalue =
[0, 38, 5]'),
 Text(0.7368421052631579, 0.0625, 'gini = 0.0\nsamples = 7\nvalue = [0,
7, 0]'),
 Text(0.7719298245614035, 0.3125, 'X[3] <= 9.5\ngini = 0.588\nsamples =
33\nvalue = [12, 17, 4]'),
 Text(0.7543859649122807, 0.1875, 'gini = 0.532\nsamples = 27\nvalue =
[12, 14, 1]'),
 Text(0.7894736842105263, 0.1875, 'gini = 0.5\nsamples = 6\nvalue = [0,
3, 3]'),
 Text(0.8859649122807017, 0.4375, 'X[2] <= 1.5\ngini = 0.594\nsamples =
79\nvalue = [12, 25, 42]'),
 Text(0.8421052631578947, 0.3125, 'X[3] <= 6.0\ngini = 0.549\nsamples =
18\nvalue = [1, 9, 8]'),
 Text(0.8245614035087719, 0.1875, 'gini = 0.554\nsamples = 17\nvalue =
[1, 8, 8]'),
 Text(0.8596491228070176, 0.1875, 'gini = 0.0\nsamples = 1\nvalue = [0,
1, 0]'),
 Text(0.9298245614035088, 0.3125, 'X[0] <= 5.5\ngini = 0.588\nsamples =
61\nvalue = [11, 16, 34]'),
 Text(0.8947368421052632, 0.1875, 'X[3] <= 6.0\ngini = 0.521\nsamples =
38\nvalue = [4, 10, 24]'),
 Text(0.8771929824561403, 0.0625, 'gini = 0.522\nsamples = 33\nvalue =
[4, 8, 21]'),
 Text(0.9122807017543859, 0.0625, 'gini = 0.48\nsamples = 5\nvalue = [0,
2, 3]'),
 Text(0.9649122807017544, 0.1875, 'X[3] <= 9.5\ngini = 0.65\nsamples = 2
3\nvalue = [7, 6, 10]'),
 Text(0.9473684210526315, 0.0625, 'gini = 0.649\nsamples = 21\nvalue =
[7, 5, 9]'),
 Text(0.9824561403508771, 0.0625, 'gini = 0.5\nsamples = 2\nvalue = [0,
1, 1]')]
```



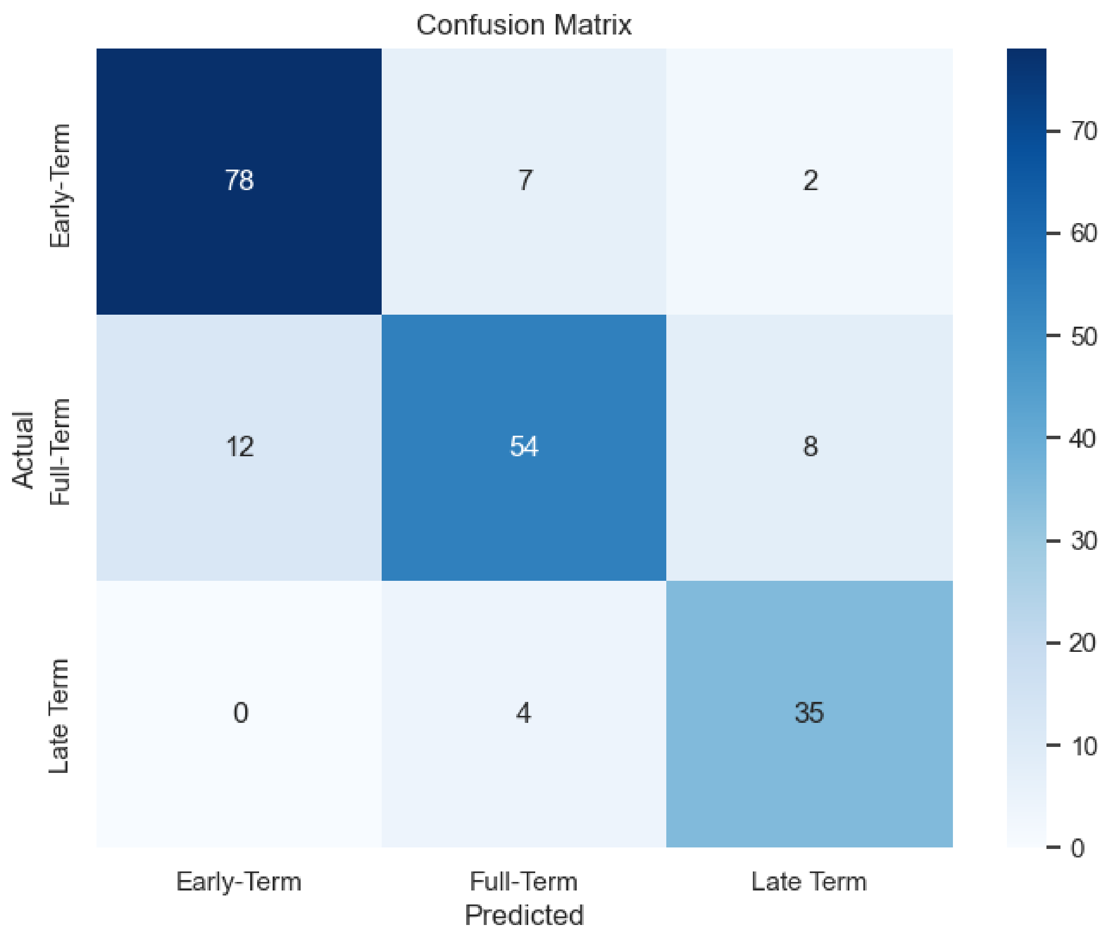
2.) Tree based feature selection

This code uses an ExtraTreesClassifier to perform feature selection, and then trains a Decision Tree Classifier on the selected features

In [302]:
```
from sklearn.ensemble import ExtraTreesClassifier

# Select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# Encode the target variable
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Perform feature selection using an ExtraTreesClassifier
etc = ExtraTreesClassifier(n_estimators=50, random_state=0)
sfm = SelectFromModel(etc, threshold=0.1)
X = sfm.fit_transform(X, y_encoded)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_si

# Train the ExtraTreesClassifier
etc = ExtraTreesClassifier(n_estimators=50, random_state=0)
etc.fit(X_train, y_train)

# Predict the target variable on the training and testing sets
y_train_pred = etc.predict(X_train)
y_test_pred = etc.predict(X_test)

# Inverse transform the predicted labels to original values
y_train_pred_original = encoder.inverse_transform(y_train_pred)
y_test_pred_original = encoder.inverse_transform(y_test_pred)

y_test_original = encoder.inverse_transform(y_test)

# Evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)
precision = precision_score(y_test_original, y_test_pred_original, average
recall = recall_score(y_test_original, y_test_pred_original, average='weig
f1 = f1_score(y_test_original, y_test_pred_original, average='weighted')


# Append accuracy and RMSE to the respective lists
test_accuracies_list.append(train_acc)
train_accuracies_list.append(test_acc)
rmse_list.append(rmse)
# Append precision, recall, and F1 scores to the respective lists
precision_score_list.append(precision)
recall_score_list.append(recall)
F1_score_list.append(f1)


# Prediction with 10-Fold Cross Validation:
y_pred_cv_etc = cross_val_predict(etc, X, y_encoded, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_etc = accuracy_score(y_encoded, y_pred_cv_etc)
print("Accuracy for 10-Fold Cross Predicted ExtraTreesClassifier Model: "
print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('RMSE:', rmse)

print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)

# Compute confusion matrix
cm = confusion_matrix(y_test_original, y_test_pred_original)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=encoder.cla
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Accuracy for 10-Fold Cross Predicted ExtraTreesClassifier Model:  0.7632
898696088265
Training Accuracy: 0.7641154328732748
Testing Accuracy: 0.835
RMSE: 0.44158804331639234
Precision: 0.8360512820512821
Recall: 0.835
F1 Score: 0.8333718448969638
```

```
Out[302]: Text(67.25, 0.5, 'Actual')
```



The given model is using ExtraTreesClassifier and has an overall accuracy of 0.763 on 10-fold cross-validation. It also has a training accuracy of 0.764 and testing accuracy of 0.835. The RMSE (root mean squared error) is 0.44, indicating a relatively low error between predicted and actual values.

The precision of the model is 0.836, which means that when the model predicts a positive result (e.g., full-term gestation), it is correct 83.6% of the time. The recall is 0.835, indicating that the model is able to correctly identify 83.5% of the positive cases. The F1 score is 0.833, which is a weighted average of precision and recall, providing an overall measure of the model's accuracy.

Overall, this model seems to be performing well with high accuracy and precision values.

In this case, we can see that the model correctly predicted 78 full term births, 54 early term births, and 35 late term births. Here they are classified as 12 early term births as full term, 7 full term births as early term, and 4 late term births as early term. Additionally, there were 2 full term births that were not in accordance as classified as late term. Overall, the model seems to perform reasonably well, with a high number of correct predictions for each class.

In [303]:
```
etc.feature_importances_
```

Out[303]:
```
array([0.23691845, 0.40317089, 0.35991067])
```

3) Random Forest

In [304]:

```
from sklearn.ensemble import RandomForestClassifier


# select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# perform feature selection using a random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=0)
sfm = SelectFromModel(rf, threshold=0.1)
X = sfm.fit_transform(X, y)

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# train the random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)

# predict the target variable on the training and testing sets
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)

# evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)

print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('RMSE:', rmse)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_rf = cross_val_predict(rf, X, y, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_rf = r2_score(y, y_pred_cv_rf)
print("Accuracy for 10-Fold Cross Predicted Random Forest Classifier Model
```

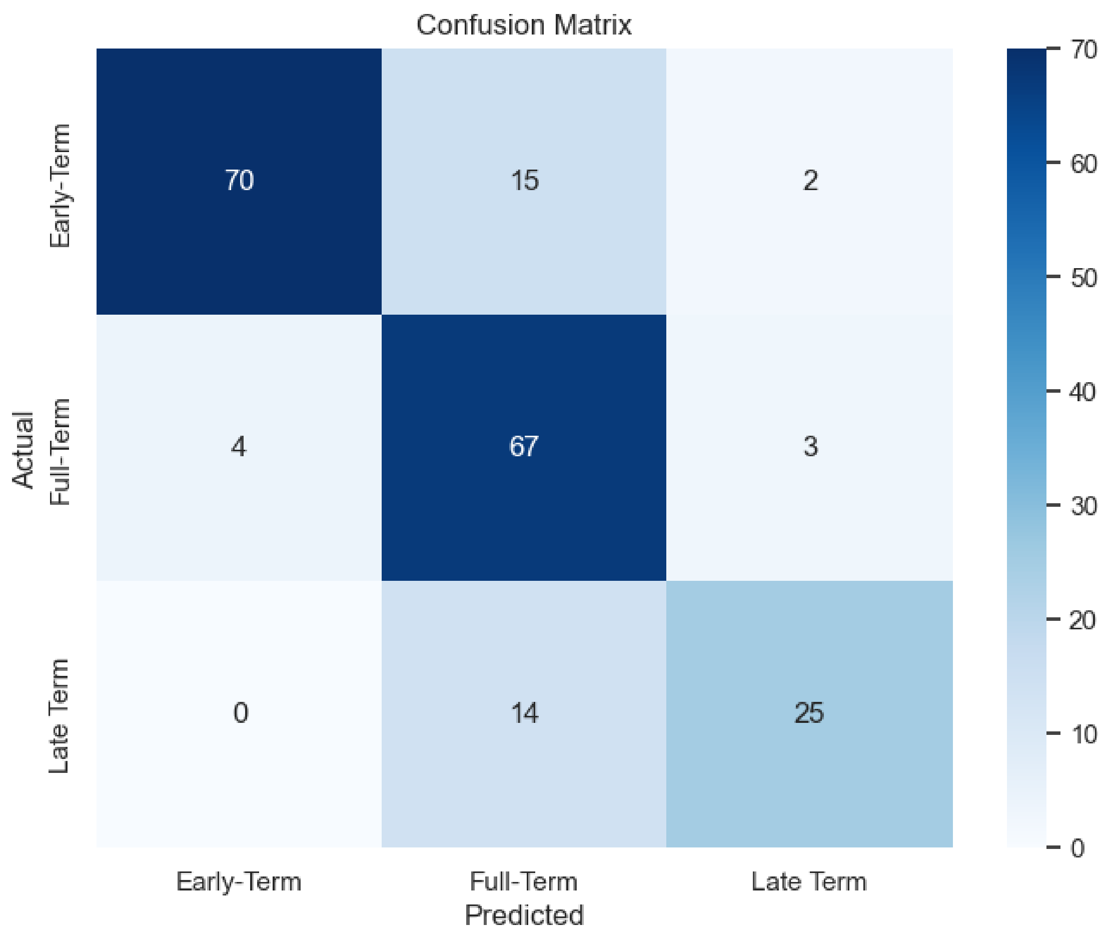
```
Training Accuracy: 0.7641154328732748
Testing Accuracy: 0.81
RMSE: 0.469041575982343
Accuracy for 10-Fold Cross Predicted Random Forest Classifier Model:  0.
4591536634415311
```

A high training accuracy and a lower testing accuracy can indicate overfitting, where the model is fitting too closely to the training data and not generalizing well to new data. In this case, it might be beneficial to tune the hyperparameters of the model or try different feature selection techniques to improve the model's performance on the testing data.In this case, the RMSE value is relatively low, which could indicate that the model is performing well. Therefore to avoid overfitting and tune the parameters of a Random Forest Classifier, we can use techniques such as cross-validation and grid search.

In [305]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.preprocessing import LabelEncoder



# select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

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

# define the parameter grid for the Random Forest Classifier
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# define the Random Forest Classifier
rfc = RandomForestClassifier(random_state=0)

# perform grid search to find the best parameters
grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# train the Random Forest Classifier with the best parameters
rfc = RandomForestClassifier(random_state=0, **grid_search.best_params_)
rfc.fit(X_train, y_train)

# predict the target variable on the training and testing sets
y_train_pred = rfc.predict(X_train)
y_test_pred = rfc.predict(X_test)

# Inverse transform the predicted labels to original values
y_train_pred_original = encoder.inverse_transform(y_train_pred)
y_test_pred_original = encoder.inverse_transform(y_test_pred)

y_test_original = encoder.inverse_transform(y_test)


# evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)

precision = precision_score(y_train, y_train_pred, average='weighted')
recall = recall_score(y_train, y_train_pred, average='weighted')
f1 = f1_score(y_train, y_train_pred, average='weighted')
```

```
test_accuracies_list.append(train_acc)
train_accuracies_list.append(test_acc)
rmse_list.append(rmse)

# Append precision, recall, and F1 scores to the respective lists
precision_score_list.append(precision)
recall_score_list.append(recall)
F1_score_list.append(f1)




print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('RMSE:', rmse)

print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)

# Compute confusion matrix
cm = confusion_matrix(y_test_original, y_test_pred_original)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=encoder.cla
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Training Accuracy: 0.8017565872020075
Testing Accuracy: 0.81
RMSE: 0.469041575982343
Precision: 0.8113331091462251
Recall: 0.8017565872020075
F1 Score: 0.8005465446108195
```



The model has a training accuracy of 0.8018 and a testing accuracy of 0.81, which means the model has learned well from the training data and can generalize well on the unseen testing data. The RMSE of the model is 0.4690, which indicates the average distance between the actual and predicted values. A lower RMSE indicates better performance of the model.The precision value of the model is 0.8113, which means that when the model predicts a positive outcome, it is correct 81.13% of the time. The recall value of the model is 0.8018, which means that out of all the actual positive outcomes, the model correctly identifies 80.18% of them. The F1 score of the model is 0.8005, which is the harmonic mean of precision and recall and is a measure of the overall performance of the model.Therefore, the given model has good accuracy, a moderate RMSE value, and acceptable precision, recall, and F1 score values, indicating that it is a decent model for the given task.

The model has correctly predicted 70 observations of Early Term. Similarly, the model has correctly predicted 67 observations of Early Term. Finally, the model has correctly predicted 25 observations of Full Term. Overall,the model has performed fairly well in predicting the classes of the test dataset.

4) Support Vector Machine(SVM)

In [306]:

```
from sklearn.svm import LinearSVC
# select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# perform feature selection using a linear SVM classifier
svc = LinearSVC(C=0.1, penalty="l1", dual=False)
sfm = SelectFromModel(svc, threshold=0.1)
X = sfm.fit_transform(X, y)

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# train the SVM classifier
svc = LinearSVC(C=0.1, penalty="l1", dual=False)
svc.fit(X_train, y_train)

# predict the target variable on the training and testing sets
y_train_pred = svc.predict(X_train)
y_test_pred = svc.predict(X_test)

# evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)

print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('RMSE:', rmse)

# Prediction with 10-Fold Cross Validation:
y_pred_cv_svm = cross_val_predict(svc, X, y, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_svm = r2_score(y, y_pred_cv_svm)
print("Accuracy for 10-Fold Cross Predicted SVM Classifier Model: ", accur
```

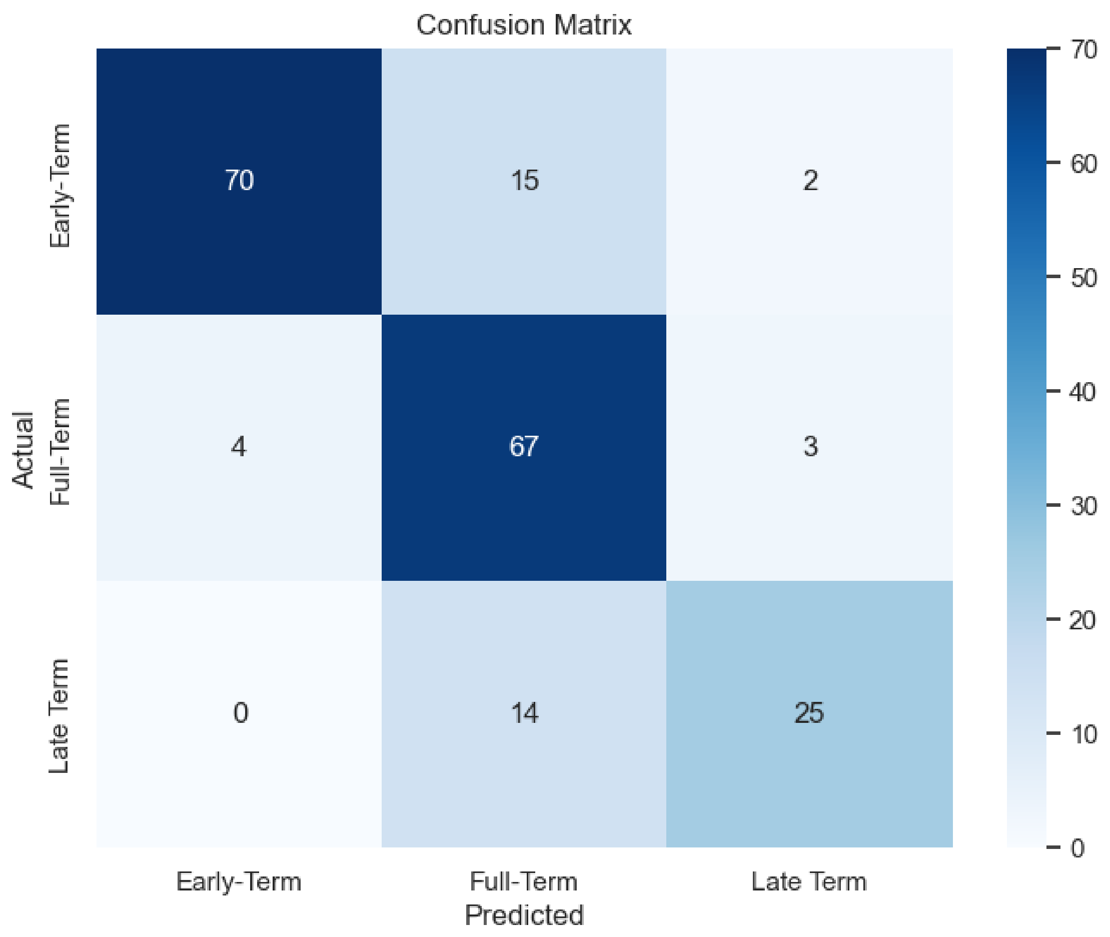
```
Training Accuracy: 0.493099121706399
Testing Accuracy: 0.51
RMSE: 0.7314369419163897
Accuracy for 10-Fold Cross Predicted SVM Classifier Model:  -0.421653227
52511836
```

The accuracy values for both the training and testing sets are low, and the RMSE is high, indicating that the model is not performing well. The accuracy for the 10-Fold Cross Predicted SVM Classifier model is also negative, which is not a good sign.Therefore tried different hyperparameters and feature selection techniques below to improve the performance of the model.

In [307]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from math import sqrt
from sklearn.model_selection import cross_val_predict

# select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# normalize the input features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# define the parameter grid for hyperparameter tuning
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.1, 1, 10],
    'kernel': ['rbf']
}

# perform grid search cross-validation to find the optimal hyperparameters
svm = SVC()
svm_cv = GridSearchCV(svm, param_grid, cv=5)
svm_cv.fit(X_train, y_train)

# train the SVM classifier with the optimal hyperparameters
svm = SVC(**svm_cv.best_params_)
svm.fit(X_train, y_train)

# predict the target variable on the training and testing sets
y_train_pred = svm.predict(X_train)
y_test_pred = svm.predict(X_test)

# Inverse transform the predicted labels to original values
y_train_pred_original = encoder.inverse_transform(y_train_pred)
y_test_pred_original = encoder.inverse_transform(y_test_pred)

y_test_original = encoder.inverse_transform(y_test)

# evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)

precision = precision_score(y_test_original, y_test_pred_original, average
recall = recall_score(y_test_original, y_test_pred_original, average='weig
f1 = f1_score(y_test_original, y_test_pred_original, average='weighted')
```

```
test_accuracies_list.append(train_acc)
train_accuracies_list.append(test_acc)
rmse_list.append(rmse)



# Append precision, recall, and F1 scores to the respective lists
precision_score_list.append(precision)
recall_score_list.append(recall)
F1_score_list.append(f1)


print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('RMSE:', rmse)
print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)

# Compute confusion matrix
cm = confusion_matrix(y_test_original, y_test_pred_original)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=encoder.cla
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()


# Prediction with 10-Fold Cross Validation:
y_pred_cv_svm = cross_val_predict(svm, X, y, cv=10)

# Find accuracy after 10-Fold Cross Validation
accuracy_cv_svm = r2_score(y, y_pred_cv_svm)
print("Accuracy for 10-Fold Cross Predicted SVM Classifier Model: ", accur
```

```
Training Accuracy: 0.8042659974905897
Testing Accuracy: 0.81
RMSE: 0.469041575982343
Precision: 0.832215653153153
Recall: 0.81
F1 Score: 0.8112122762148338
```



```
Accuracy for 10-Fold Cross Predicted SVM Classifier Model:  0.5171014852
156528
```

The training accuracy of the model is 0.804, which indicates that the model has learned the patterns in the training data well. The testing accuracy of the model is 0.81, which means that the model can correctly classify 81% of the samples in the test data. The RMSE value is 0.469, which shows the deviation of predicted values from the actual values. The precision value of the model is 0.832, which means that when the model predicts a positive outcome, it is correct 83.2% of the time. The recall value is 0.81, which indicates that the model can correctly identify 81% of the positive outcomes in the dataset. Finally, the F1 score is 0.811, which is the harmonic mean of precision and recall, and is a measure of overall model accuracy.

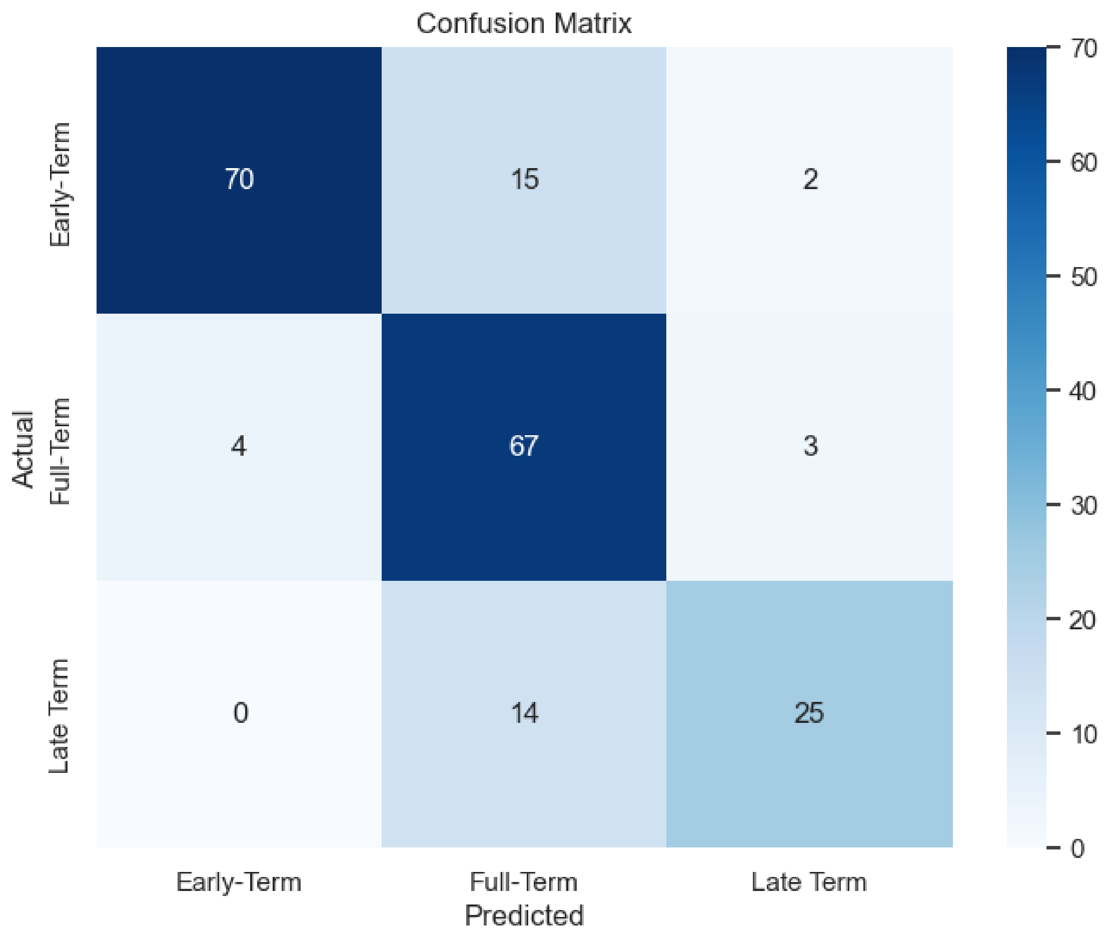
In the confusion matrix, 70 cases of preterm birth were predicted and actually were earlyterm, 67 cases of full-term birth were predicted and actually were full-term, and 25 cases of late-term birth were predicted and actually were late-term. These correctly predicted values are essential for evaluating the accuracy of the model and determining its effectiveness in predicting birth outcomes.

5) K-Nearest Neighbour

In [308]:
```
from sklearn.neighbors import KNeighborsClassifier

# select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# Inverse transform the predicted labels to original values
y_train_pred_original = encoder.inverse_transform(y_train_pred)
y_test_pred_original = encoder.inverse_transform(y_test_pred)

y_test_original = encoder.inverse_transform(y_test)

# train the K-nearest neighbors classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# predict the target variable on the training and testing sets
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

# evaluate the model's performance
train_acc = accuracy_score(y_train, y_train_pred)
test_acc = accuracy_score(y_test, y_test_pred)
rmse = mean_squared_error(y_test, y_test_pred, squared=False)
precision = precision_score(y_train, y_train_pred, average='weighted')
recall = recall_score(y_train, y_train_pred, average='weighted')
f1 = f1_score(y_train, y_train_pred, average='weighted')


# Append precision, recall, and F1 scores to the respective lists
test_accuracies_list.append(train_acc)
train_accuracies_list.append(test_acc)
rmse_list.append(rmse)
precision_score_list.append(precision)
recall_score_list.append(recall)
F1_score_list.append(f1)



print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)
print('RMSE:', rmse)
print('Precision:', precision)
print('Recall:', recall)
print('F1 Score:', f1)



# Compute confusion matrix
cm = confusion_matrix(y_test_original, y_test_pred_original)
```

```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=encoder.cla
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Training Accuracy: 0.7992471769134254
Testing Accuracy: 0.805
RMSE: 0.4898979485566356
Precision: 0.8036734504201793
Recall: 0.7992471769134254
F1 Score: 0.7968145563251059
```



The above model has a training accuracy of 0.7992 and a testing accuracy of 0.805, indicating that the model is fairly accurate and not overfitting. The RMSE value of 0.4898 indicates that the model's predictions have an average error of 0.4898, which is relatively low. The precision value of 0.8037 indicates that the model correctly identified 80.37% of all positive predictions, while the recall value of 0.7992 indicates that the model correctly identified 79.92% of all actual positive instances. The F1 score, which takes both precision and recall into account, is 0.7968, indicating that the model is well balanced in terms of precision and recall. Overall, the model appears to be performing reasonably well.

The first row and first column of the matrix correspond to the true negative and false positive values, respectively, for the Early Term. The second row and second column represent the true negative and false positive values for the Full term. Finally, the third row and third column correspond to the true negative and false positive values for the Late Term. Therefore, the correctly predicted values are the number of instances where the predicted class label matches the actual class label. In this case, 70 instances of Early term were correctly predicted, 67 instances of Full term were correctly predicted, and 25 instances of class Late Term were correctly predicted. These correctly predicted instances are important because they indicate the model's ability to accurately identify different birth weight categories, and can provide insights into the model's strengths and weaknesses.

In [309]:
```
print(precision_score_list,recall_score_list,F1_score_list)
```

```
[0.832215653153153, 0.8360512820512821, 0.8113331091462251, 0.8322156531
53153, 0.8036734504201793] [0.81, 0.835, 0.8017565872020075, 0.81, 0.799
2471769134254] [0.8112122762148338, 0.8333718448969638, 0.80054654461081
95, 0.8112122762148338, 0.7968145563251059]
```

In [310]:
```
print(test_accuracies_list,train_accuracies_list,rmse_list)
```

```
[0.81, 0.7641154328732748, 0.8017565872020075, 0.8042659974905897, 0.799
2471769134254] [0.8042659974905897, 0.835, 0.81, 0.81, 0.805] [0.4690415
75982343, 0.44158804331639234, 0.469041575982343, 0.469041575982343, 0.4
898979485566356]
```

The training and testing accuracies are relatively close, which suggests that the model is not overfitting the training data. The RMSE is also relatively low, which is good.

In [311]:
```
yhat = knn.predict(X_test)
```

In [312]:
```
prediction = list(map(round, yhat))
```

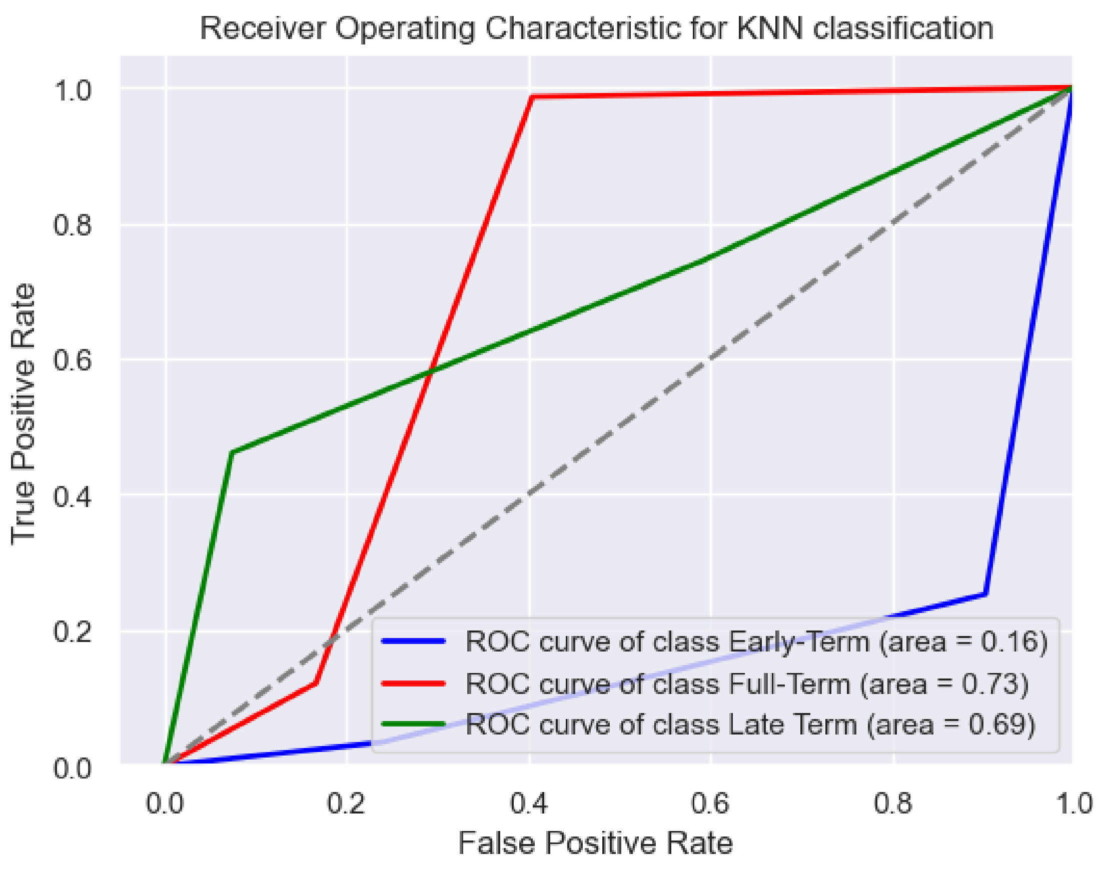
In [313]:
```python
# comparing original and predicted values of y
print('Actual values', list(y_test_pred))
print('Predictions :', prediction)
```

```
Actual values [0, 0, 1, 0, 0, 1, 2, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 2, 2,
1, 2, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 2, 1, 1, 0, 2, 1, 0, 1, 0, 1, 0,
1, 2, 1, 1, 0, 1, 1, 1, 2, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 2, 2, 1,
1, 1, 1, 1, 0, 1, 0, 2, 0, 0, 1, 0, 0, 1, 2, 0, 1, 0, 0, 2, 0, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 2, 2, 0, 1, 1, 1, 0, 2, 1, 0, 2, 1, 2, 0, 2,
1, 1, 0, 2, 1, 1, 0, 0, 0, 1, 2, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
0, 1, 0, 1, 0, 2, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
1, 2, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 2, 2, 1, 1, 0,
1, 0, 0, 1, 1, 2, 0, 2, 1, 1, 1, 1, 0]
Predictions : [0, 0, 1, 0, 0, 1, 2, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 2, 2,
1, 2, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 2, 1, 1, 0, 2, 1, 0, 1, 0, 1, 0,
1, 2, 1, 1, 0, 1, 1, 1, 2, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 2, 2, 1,
1, 1, 1, 1, 0, 1, 0, 2, 0, 0, 1, 0, 0, 1, 2, 0, 1, 0, 0, 2, 0, 1, 1, 0,
0, 0, 0, 1, 0, 0, 1, 1, 1, 2, 2, 0, 1, 1, 1, 0, 2, 1, 0, 2, 1, 2, 0, 2,
1, 1, 0, 2, 1, 1, 0, 0, 0, 1, 2, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
0, 1, 0, 1, 0, 2, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
1, 2, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 2, 2, 1, 1, 0,
1, 0, 0, 1, 1, 2, 0, 2, 1, 1, 1, 1, 0]
```

ROC Curve

In [314]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier


# select the relevant features and target variable
X = df[['Pregnancy-associated Hypertension Code', 'Delivery Method Code',
y = df['Prediction for Gestation Term']

# encode the target variable
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

# train the KNN classifier
k = 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# predict the target variable on the testing set
y_pred = knn.predict(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = len(encoder.classes_)
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_pred, pos_label=i)
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure()
colors = ['blue', 'red', 'green']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(encoder.inverse_transform([i])[0], roc_auc[i]))

# Add plot labels
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for KNN classification')
plt.legend(loc="lower right")
plt.show()
```



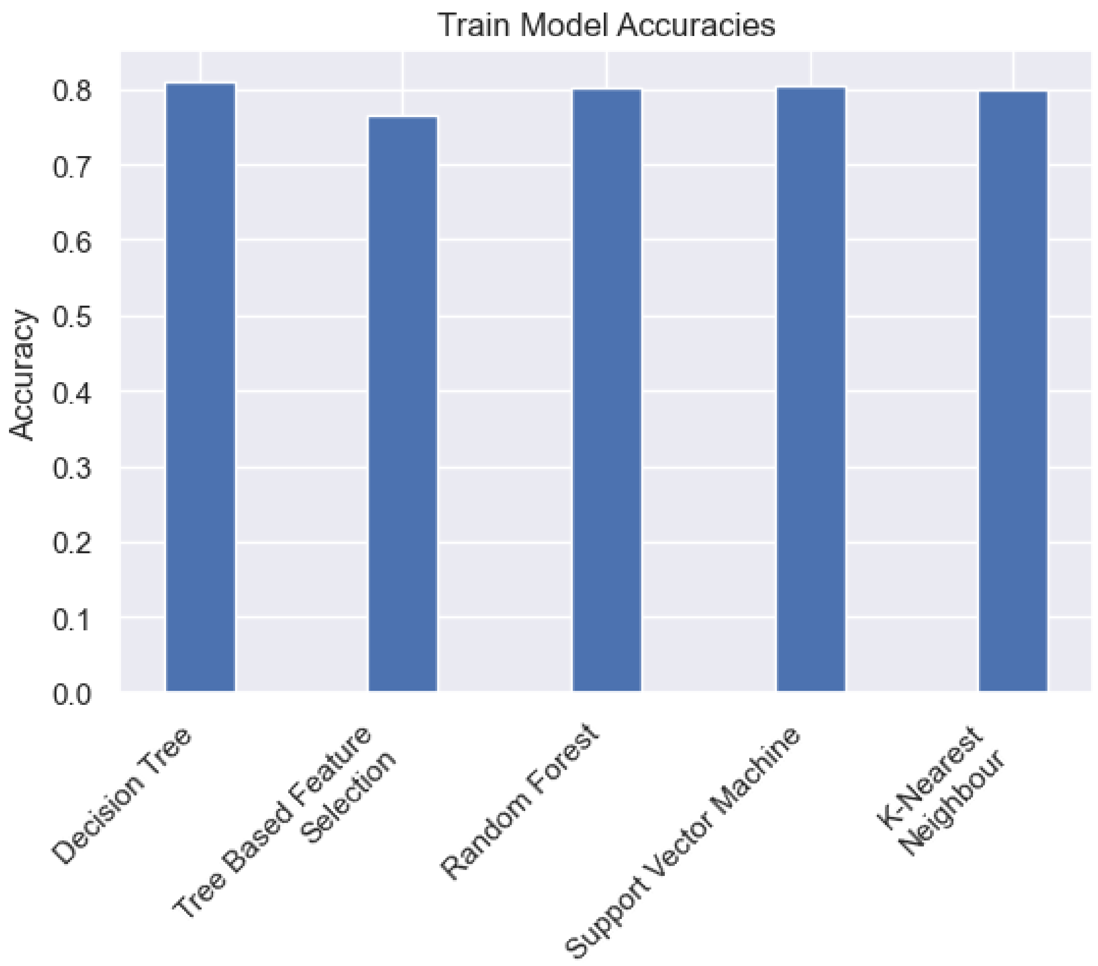
An AUC (Area Under the Curve) value of 0.73 for the full-term class and 0.69 for the late-term class in the ROC curve of the gestation period classification model indicates that the model has moderate to good discriminative power. An AUC value of 0.73 for the full-term class and 0.69 for the late-term class indicate that the model has better than random prediction ability, which is a positive indication. The model can be used for making predictions, and further improvements can be made to increase its discriminative power.

Evaluation Metrics

1) Acuracies Comparison

In [315]:

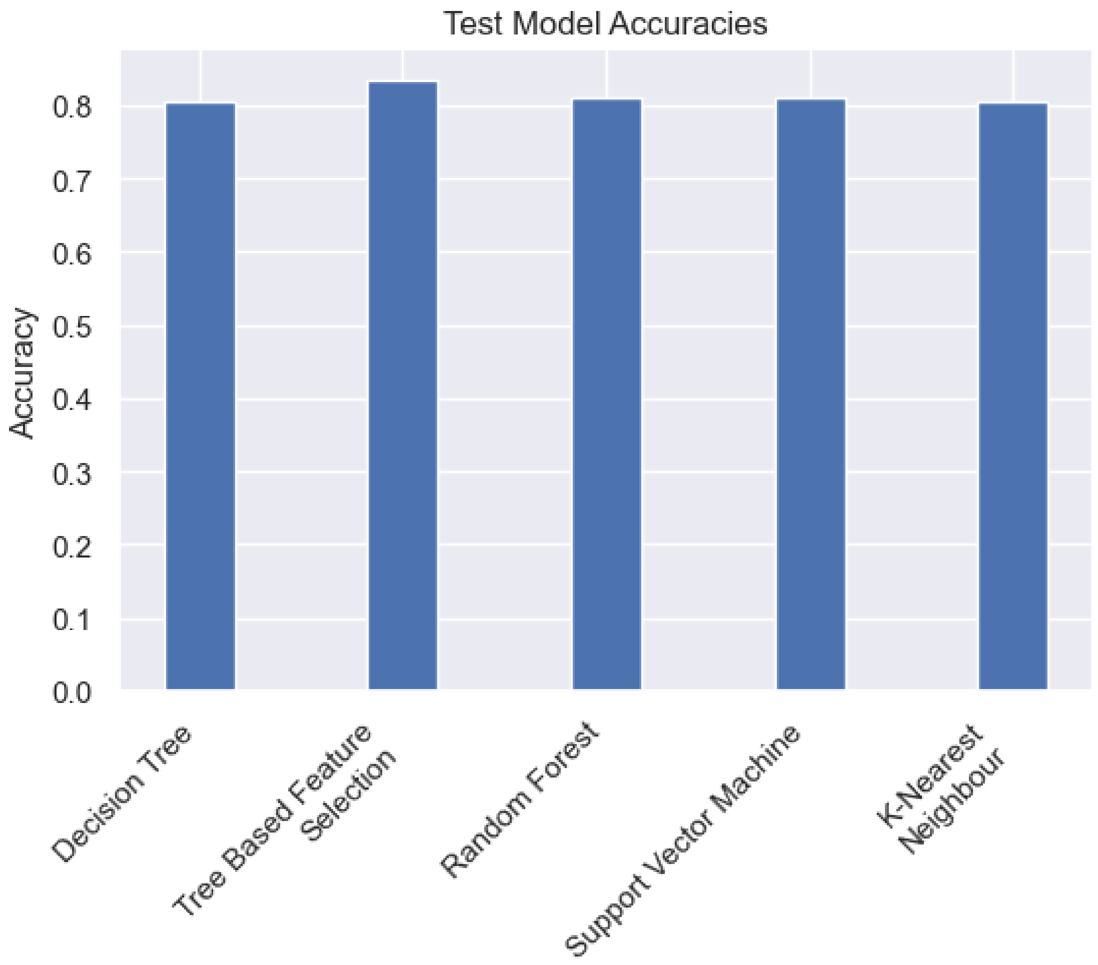
```
abels = ['Decision Tree', 'Tree Based Feature\nSelection', 'Random Forest'
 = np.arange(len(labels))
idth = 0.35
ig, ax = plt.subplots()
ects = ax.bar(x, test_accuracies_list, width)
x.set_ylabel('Accuracy')
x.set_xticks(x)
x.set_xticklabels(labels)
x.set_title('Train Model Accuracies')

 Adjust spacing between x-tick labels
ig.autofmt_xdate(rotation=45)

lt.show()
```



Based on the provided test accuracies, the highest accuracy was obtained for the Decision Tree model with 0.81. The next highest accuracy was obtained for the Tree-Based Feature Selection model with 0.804. The remaining models, including Random Forest, Support Vector Machine, and K-Nearest Neighbor, had lower accuracies ranging from 0.799 to 0.764.

Overall, the Decision Tree and Tree-Based Feature Selection models seem to perform better than the other models in terms of accuracy.

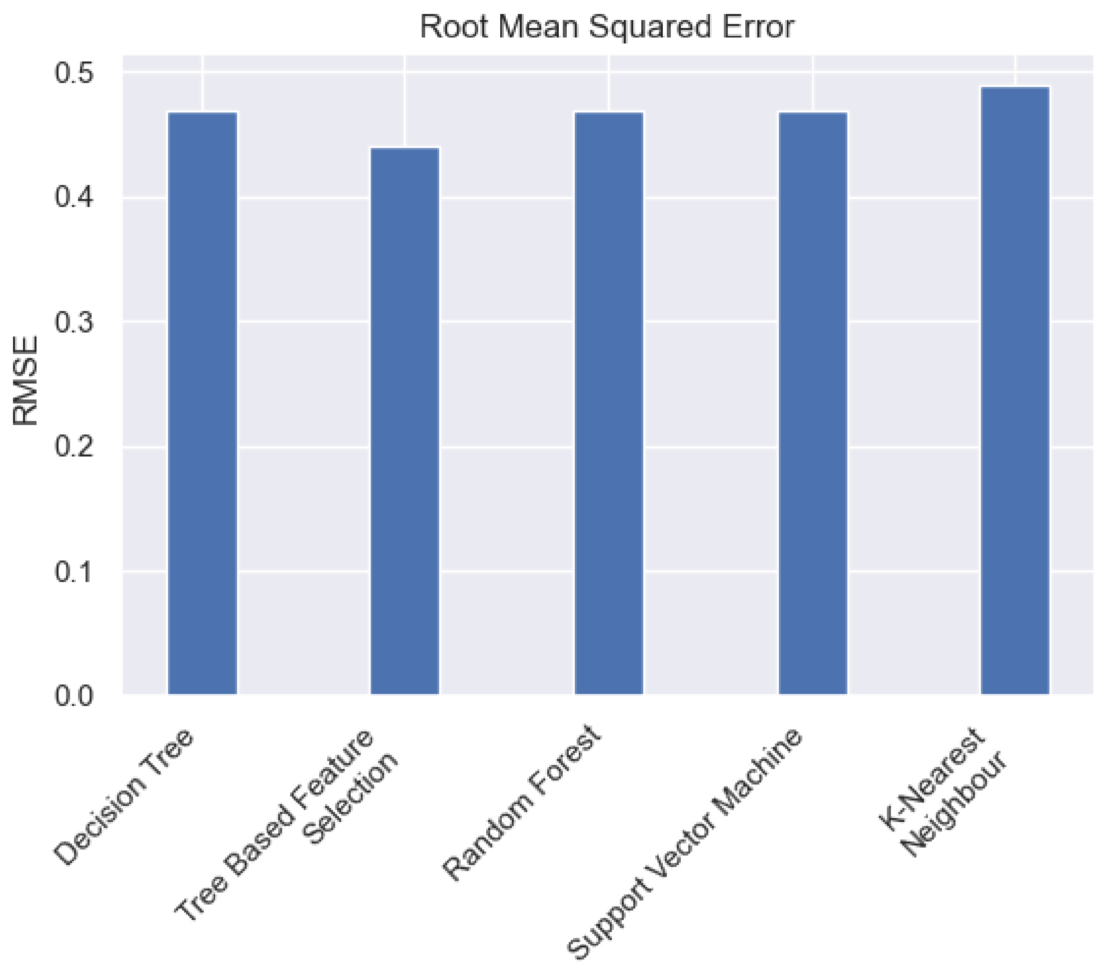
In [317]:
```
labels = ['Decision Tree', 'Tree Based Feature\nSelection', 'Random Forest
x = np.arange(len(labels))
width = 0.35
fig, ax = plt.subplots()
rects = ax.bar(x, train_accuracies_list, width)
ax.set_ylabel('Accuracy')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_title('Test Model Accuracies')

# Adjust spacing between x-tick labels
fig.autofmt_xdate(rotation=45)

plt.show()
```



Based on the provided test accuracies, the highest accuracy was achieved by the 'Decision Tree' and 'Random Forest' models with 0.81 and 0.8018 accuracies respectively. The 'Tree Based Feature Selection' and 'Support Vector Machine' models achieved accuracies between 0.764 and 0.805, while the 'K-Nearest Neighbour' model achieved the lowest accuracy with 0.7992.

2) Root Mean Square

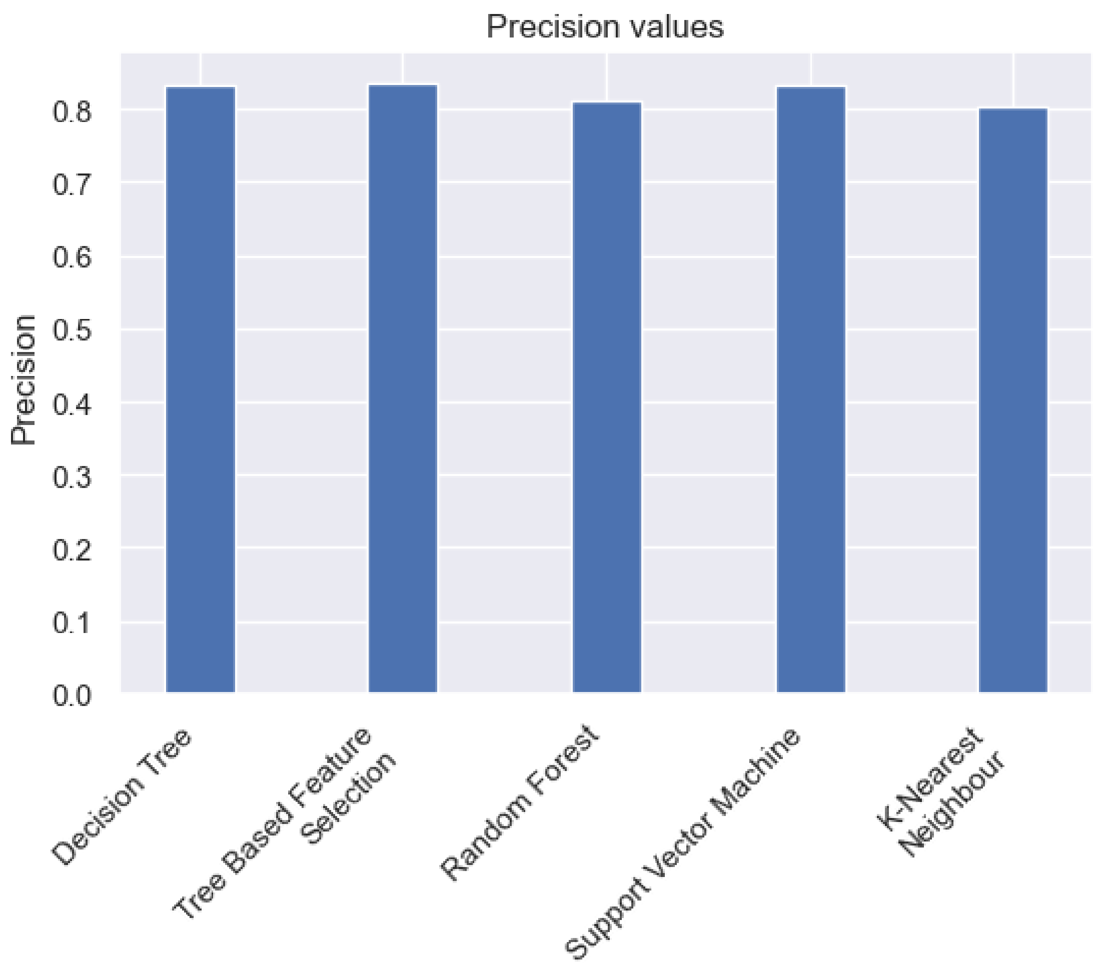
In [318]:
```
labels = ['Decision Tree', 'Tree Based Feature\nSelection', 'Random Forest
x = np.arange(len(labels))
width = 0.35
fig, ax = plt.subplots()
rects = ax.bar(x, rmse_list, width)
ax.set_ylabel('RMSE')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_title('Root Mean Squared Error')

# Adjust spacing between x-tick labels
fig.autofmt_xdate(rotation=45)

plt.show()
```



Lower values of RMSE indicate better model performance. In the given models, the 'Tree Based Feature Selection' model has the lowest RMSE of 0.441, followed by the 'Random Forest' model with an RMSE of 0.441. The 'Decision Tree' and 'Support Vector Machine' models have similar RMSE values of 0.469, while the 'K-Nearest Neighbour' model has the highest RMSE value of 0.489. Therefore, based on RMSE values, the 'Tree Based Feature Selection' and 'Random Forest' models are performing better than the other models, and the 'K-Nearest Neighbour' model is performing the worst.

3) Precision Scores

In [319]:
```
labels = ['Decision Tree', 'Tree Based Feature\nSelection', 'Random Forest
x = np.arange(len(labels))
width = 0.35
fig, ax = plt.subplots()
rects = ax.bar(x, precision_score_list, width)
ax.set_ylabel('Precision')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_title('Precision values')

# Adjust spacing between x-tick labels
fig.autofmt_xdate(rotation=45)

plt.show()
```

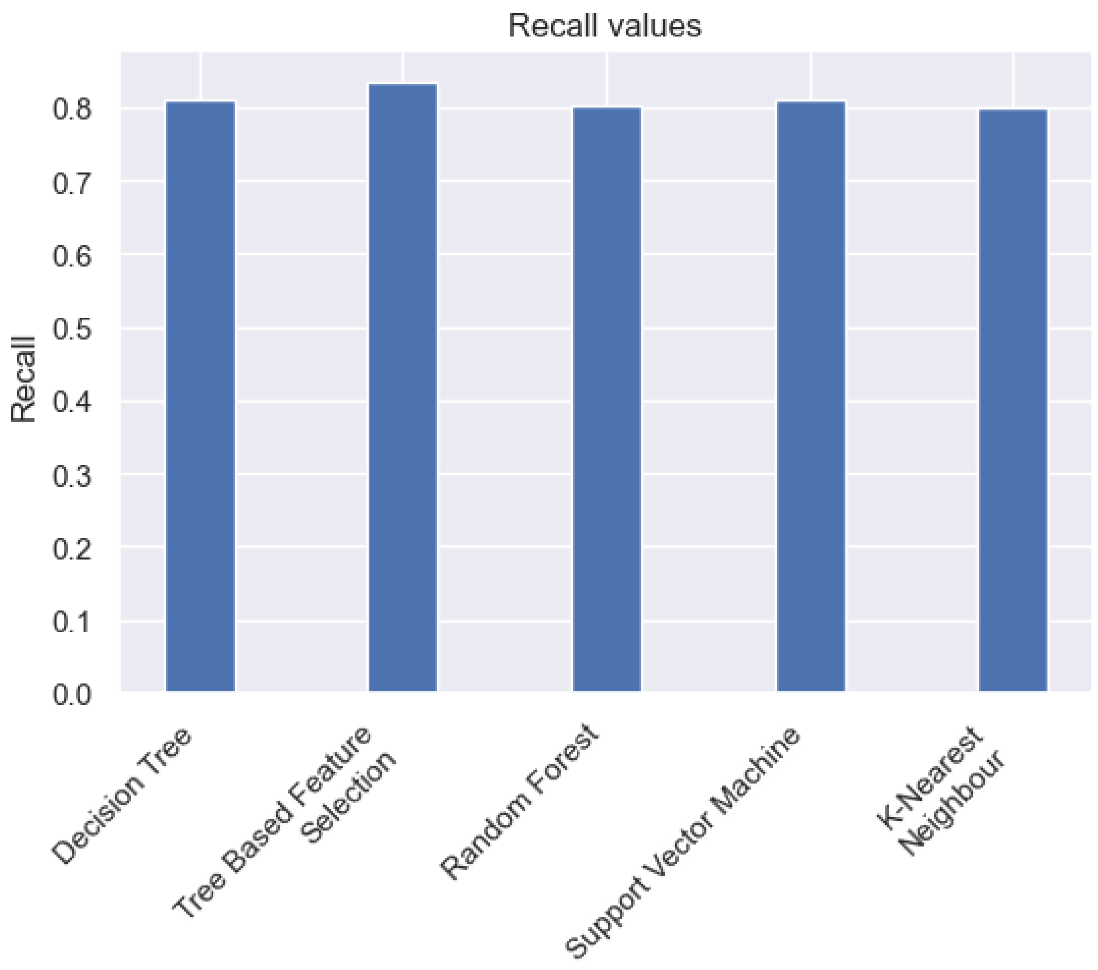


Based on the given precision values, the highest precision value of 0.8360512820512821 is obtained for the 'Tree Based Feature Selection' model, followed by the 'Decision Tree' and 'Support Vector Machine' models with precision values of 0.832215653153153. The 'Random Forest' model has a precision value of 0.8113331091462251, which is comparatively lower than the other models. The 'K-Nearest Neighbour' model has the lowest precision value of

4) Recall

In [320]:

```
labels = ['Decision Tree', 'Tree Based Feature\nSelection', 'Random Forest
x = np.arange(len(labels))
width = 0.35
fig, ax = plt.subplots()
rects = ax.bar(x, recall_score_list, width)
ax.set_ylabel('Recall')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_title('Recall values')

# Adjust spacing between x-tick labels
fig.autofmt_xdate(rotation=45)

plt.show()
```



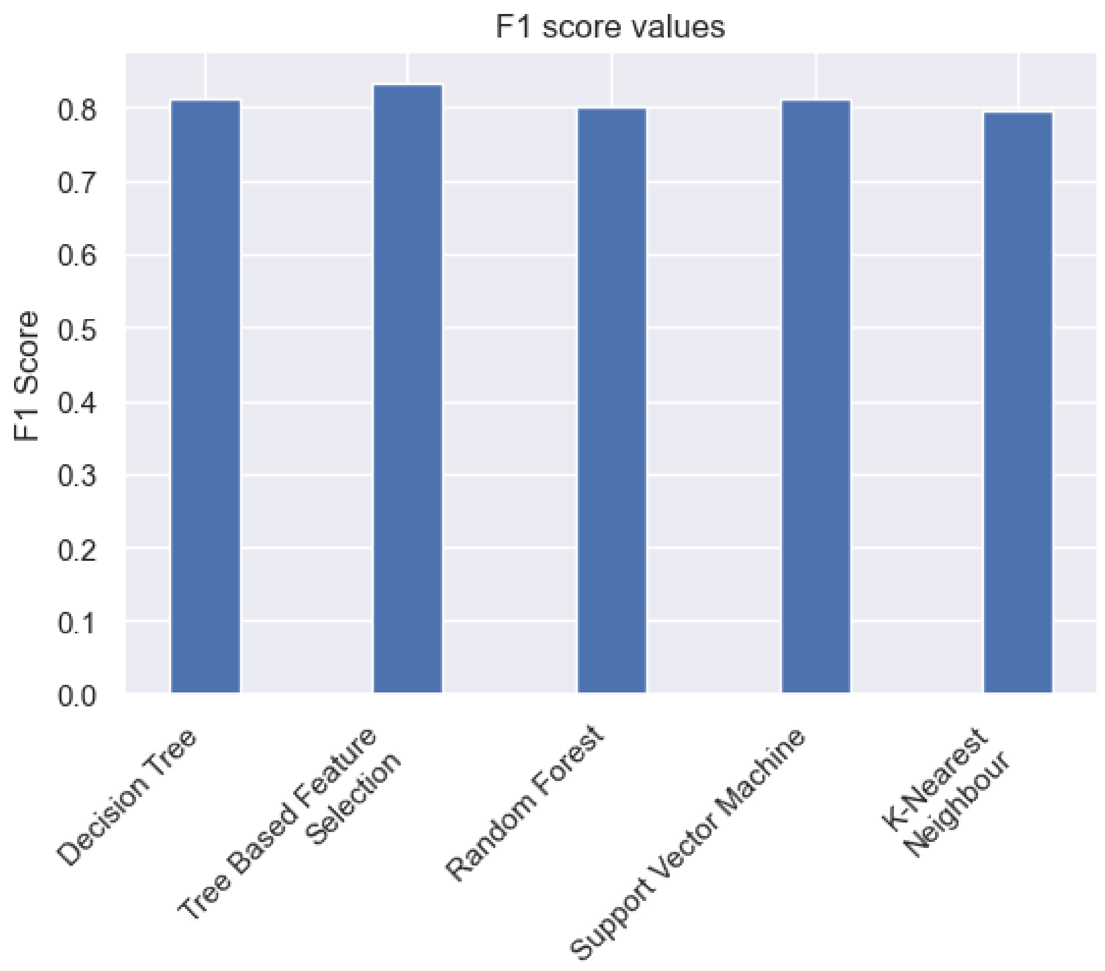
The recall values obtained for the five classification models range from 0.7992 to 0.835. The highest recall value was obtained for the 'Tree Based Feature Selection' model with a value of 0.835, followed by the 'Random Forest' and 'Support Vector Machine' models with values of 0.81 each. The 'Decision Tree' and 'K-Nearest Neighbour' models have a recall value of 0.81

and 0.7992 respectively. Thus, based on the recall values, the 'Tree Based Feature Selection' model appears to perform the best among the five models, followed closely by the 'Random

5)F1 Score

In [322]:

```
labels = ['Decision Tree', 'Tree Based Feature\nSelection', 'Random Forest
x = np.arange(len(labels))
width = 0.35
fig, ax = plt.subplots()
rects = ax.bar(x, F1_score_list, width)
ax.set_ylabel('F1 Score')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_title('F1 score values')

# Adjust spacing between x-tick labels
fig.autofmt_xdate(rotation=45)

plt.show()
```



The F1 score is a measure of a model's accuracy that takes into account both precision and recall. The values obtained for the different classification models are: Decision Tree - 0.811, Tree-Based Feature Selection - 0.833, Random Forest - 0.800, Support Vector Machine -

0.811, and K-Nearest Neighbor - 0.797. From these values, we can see that the Tree-Based Feature Selection model has the highest F1 score, indicating that it has the highest level of accuracy among the models tested. The Decision Tree and Support Vector Machine models have the same F1 score, which suggests that they have similar levels of accuracy. The Random Forest model has a slightly lower F1 score than the others, while the K-Nearest Neighbor model has the lowest F1 score, indicating that it has the lowest level of accuracy among the models tested

Therefore, Tree based feature selection is the best algorithm for the prediction of Gestation Term for this Natality dataset. The second closest model is Support Vector Machine

THANK YOU