Cervical Cancer Detection Using Cytology Data

This notebook demonstrates the process of detecting cervical cancer using cytology data. It includes data preprocessing, handling missing values, feature scaling, addressing class imbalance, model training, evaluation, and saving the trained model for future use.

1. Import Necessary Libraries

In [1]:
```python
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

from imblearn.over_sampling import SMOTE

import joblib

import warnings
warnings.filterwarnings("ignore")

# Set seaborn style for better aesthetics
sns.set(style='whitegrid', palette='muted', font_scale=1.1)
plt.rcParams['figure.figsize'] = (10, 6)
```

2: Load and Inspect the Dataset

Load the dataset and perform initial inspection to understand its structure and identify any missing values.

In [2]:
```python
# Load the dataset
data = pd.read_csv('cervical_cancer_dataset.csv')
```

In [3]:
```python
# Display the shape of the dataset
print(f"Dataset Shape: {data.shape}")
```

```
Dataset Shape: (858, 36)
```

In [4]:
```python
# Show the first five rows of the dataset
data.head()
```

Out[4]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	...	STDs: Time since first diagnosis	STDs: Time since last diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselmann	Schiller	Citol
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	?	?	0	0	0	0	0	0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	?	?	0	0	0	0	0	0	
2	34	1.0	?	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	?	?	0	0	0	0	0	0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	3.0	0.0	...	?	?	1	0	1	0	0	0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	15.0	0.0	...	?	?	0	0	0	0	0	0	

5 rows × 36 columns

In [5]:
```python
# Check for missing values represented by '?'
missing_values = data.isin(['?']).sum()
print("Missing values in each column:")
print(missing_values[missing_values > 0])
```

```
Missing values in each column:
Number of sexual partners              26
First sexual intercourse                7
Num of pregnancies                     56
Smokes                                 13
Smokes (years)                         13
Smokes (packs/year)                    13
Hormonal Contraceptives               108
Hormonal Contraceptives (years)       108
IUD                                   117
IUD (years)                           117
STDs                                  105
STDs (number)                         105
STDs:condylomatosis                   105
STDs:cervical condylomatosis          105
STDs:vaginal condylomatosis           105
STDs:vulvo-perineal condylomatosis    105
STDs:syphilis                         105
STDs:pelvic inflammatory disease      105
STDs:genital herpes                   105
STDs:molluscum contagiosum            105
STDs:AIDS                             105
STDs:HIV                              105
STDs:Hepatitis B                      105
STDs:HPV                              105
STDs: Time since first diagnosis      787
STDs: Time since last diagnosis       787
dtype: int64
```

3: Data Preprocessing

Replace placeholder missing values, convert data types, and select relevant features for analysis.

In [6]:
```python
# Replace '?' with NaN and convert all data to numeric types
data.replace('?', np.nan, inplace=True)
data = data.apply(pd.to_numeric, errors='coerce')
```

In [7]:
```python
# Define feature columns and target variable
user_features = [
    'Age',
    'Number of sexual partners',
    'First sexual intercourse',
    'Num of pregnancies',
    'Smokes',
    'Smokes (years)',
    'Smokes (packs/year)',
    'STDs',
    'STDs (number)',
    'Hormonal Contraceptives',
    'Hormonal Contraceptives (years)',
    'IUD',
    'IUD (years)',
]

target = 'Dx:Cancer'
```

In [8]:
```python
# Exclude irrelevant or redundant features
features_to_exclude = [
    'STDs: Time since first diagnosis',
    'STDs: Time since last diagnosis',
]
data.drop(columns=features_to_exclude, inplace=True, errors='ignore')
```

In [9]:
```python
# Separate features and target variable
X = data[user_features]
y = data[target]
```

4: Handle Missing Values

Identify and impute missing values using K-Nearest Neighbors imputation.

In [10]:
```python
print("Number of missing values per column:")
print(X.isnull().sum())
```

```
Number of missing values per column:
Age                                  0
Number of sexual partners           26
First sexual intercourse             7
Num of pregnancies                  56
Smokes                              13
Smokes (years)                      13
Smokes (packs/year)                 13
STDs                               105
STDs (number)                      105
Hormonal Contraceptives            108
Hormonal Contraceptives (years)    108
IUD                                117
IUD (years)                        117
dtype: int64
```

In [11]:
```python
# Initialize KNN Imputer with 5 neighbors
imputer = KNNImputer(n_neighbors=5)

# Impute missing values
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

5: Feature Scaling

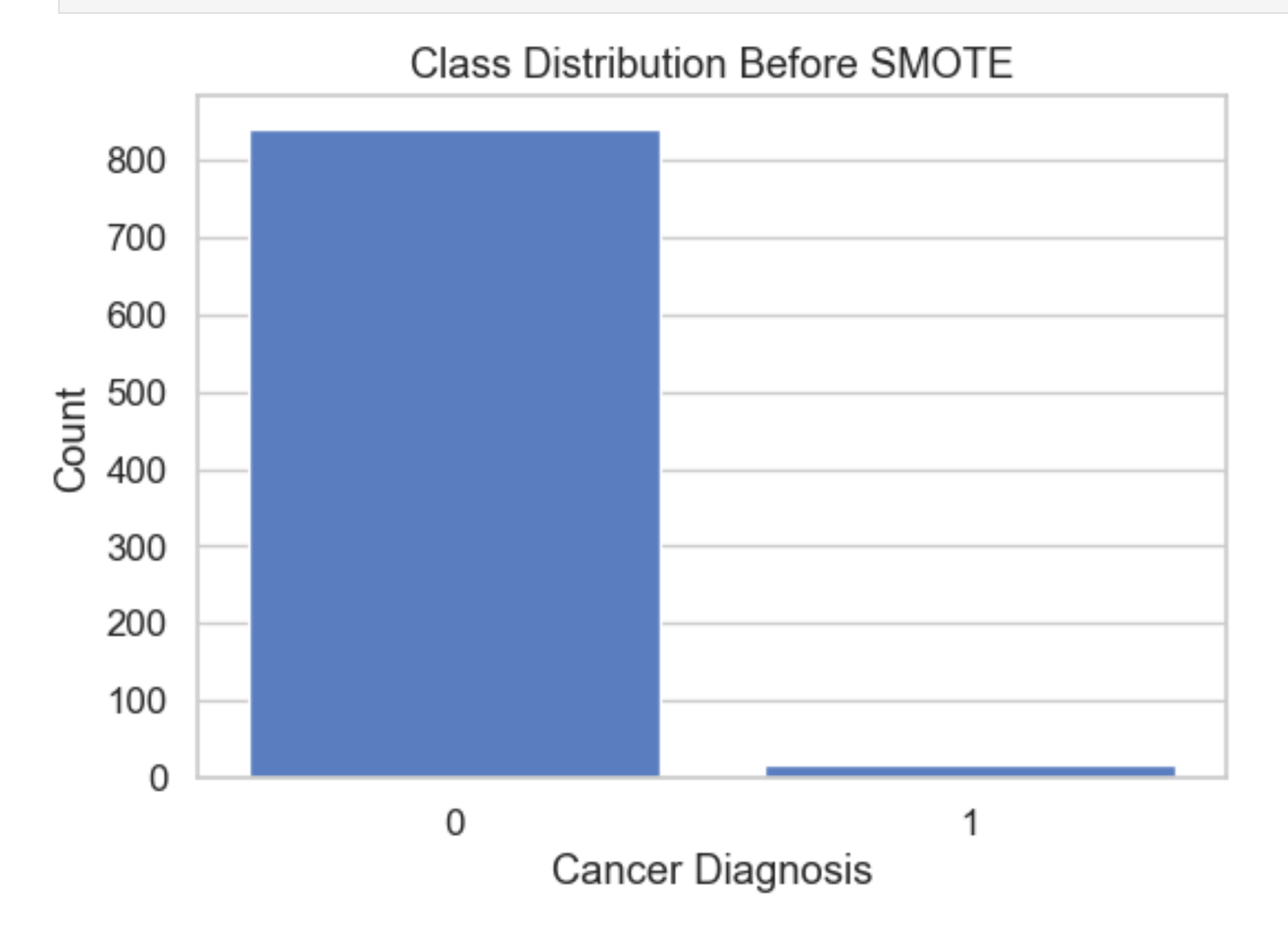
Standardize numerical features to ensure they contribute equally to the model training.

In [12]:
```python
# Define numerical features for scaling
numerical_features = [
    'Age',
    'Number of sexual partners',
    'First sexual intercourse',
    'Num of pregnancies',
    'Smokes (years)',
    'Smokes (packs/year)',
    'Hormonal Contraceptives (years)',
    'IUD (years)',
    'STDs (number)',
]

# Initialize StandardScaler
scaler = StandardScaler()

# Apply scaling to numerical features
X_imputed[numerical_features] = scaler.fit_transform(X_imputed[numerical_features])
```

6: Address Class Imbalance

Visualize class distribution and apply SMOTE to balance the classes.

In [13]:
```python
# Visualize class distribution before applying SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y)
plt.title('Class Distribution Before SMOTE')
plt.xlabel('Cancer Diagnosis')
plt.ylabel('Count')
plt.show()
```

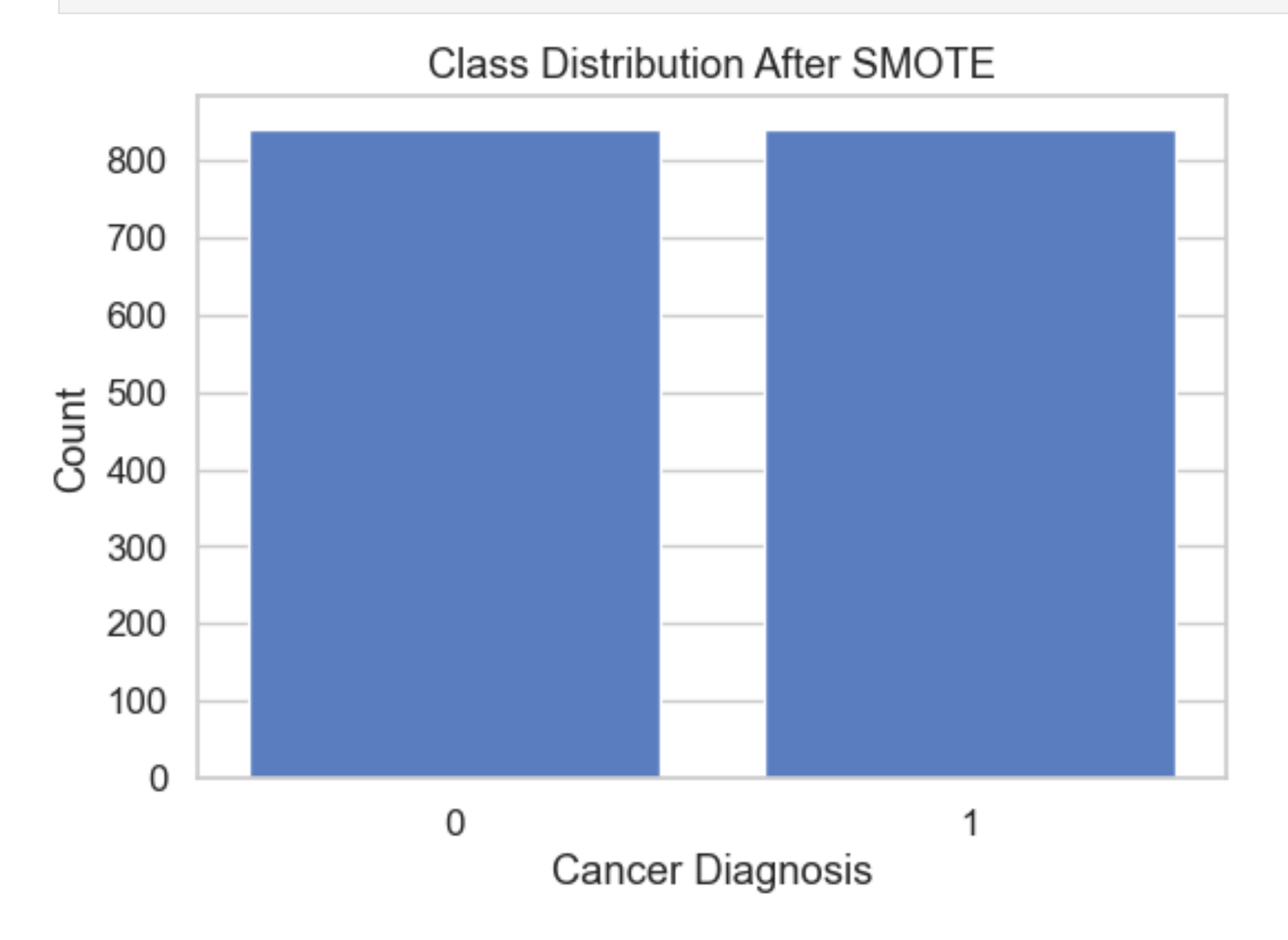


In [14]:
```python
print("Class distribution before SMOTE:")
print(y.value_counts())
```

```
Class distribution before SMOTE:
Dx:Cancer
0    840
1     18
Name: count, dtype: int64
```

In [15]:
```python
# Initialize SMOTE with a fixed random state for reproducibility
sm = SMOTE(random_state=42)

# Apply SMOTE to balance the classes
X_resampled, y_resampled = sm.fit_resample(X_imputed, y)
```

In [16]:
```python
# Visualize class distribution after applying SMOTE
plt.figure(figsize=(6, 4))
sns.countplot(x=y_resampled)
plt.title('Class Distribution After SMOTE')
plt.xlabel('Cancer Diagnosis')
plt.ylabel('Count')
plt.show()
```



In [17]:
```python
print("Class distribution after SMOTE:")
print(y_resampled.value_counts())
```

```
Class distribution after SMOTE:
Dx:Cancer
0    840
1    840
Name: count, dtype: int64
```

7: Split the Data into Training and Testing Sets

Divide the dataset into training and testing subsets to evaluate model performance.

In [18]:
```python
X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled,
    test_size=0.2,
    random_state=42,
    stratify=y_resampled
)
```

8: Train the Random Forest Classifier

Initialize and train the Random Forest model with specified hyperparameters.

In [19]:
```python
# Initialize the Random Forest Classifier with balanced class weights
model = RandomForestClassifier(
    n_estimators=200,
    max_depth=6,
    min_samples_split=2,
    min_samples_leaf=1,
    class_weight='balanced',
    random_state=42
)
```

In [20]:
```python
# Train the model on the training data
model.fit(X_train, y_train)
```

Out[20]:

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=6, n_estimators=200,
                       random_state=42)
```

9: Evaluate the Model

Assess the trained model's performance using classification metrics.

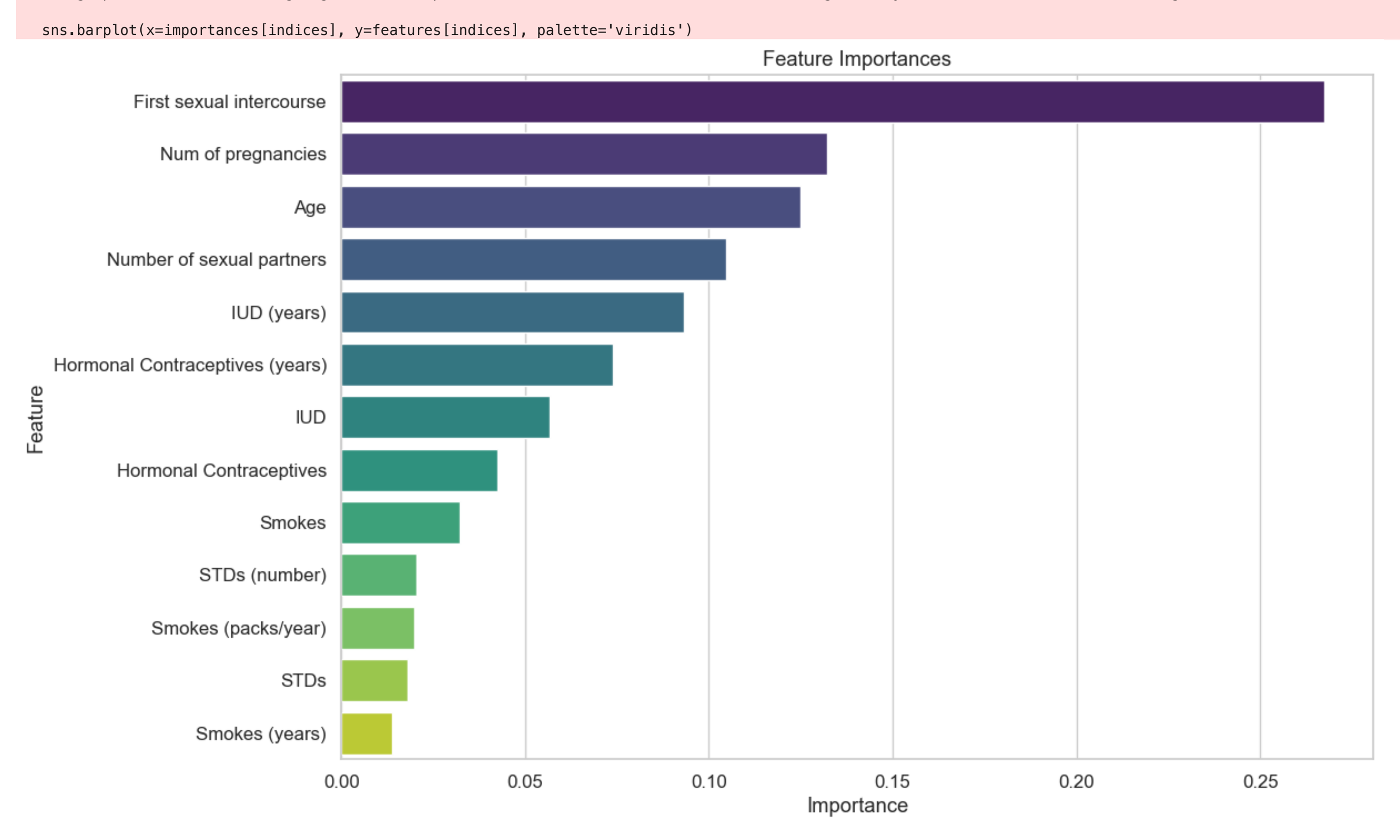
In [21]:
```python
# Make predictions on the test set
y_pred = model.predict(X_test)

# Obtain prediction probabilities for the positive class
y_proba = model.predict_proba(X_test)[:, 1]
```

In [22]:
```python
# Display the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.94       168
           1       0.93      0.94      0.94       168

    accuracy                           0.94       336
   macro avg       0.94      0.94      0.94       336
weighted avg       0.94      0.94      0.94       336
```

In [23]:
```python
# Visualize feature importance
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]
features = X_train.columns

plt.figure(figsize=(12, 8))
sns.barplot(x=importances[indices], y=features[indices], palette='viridis')
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

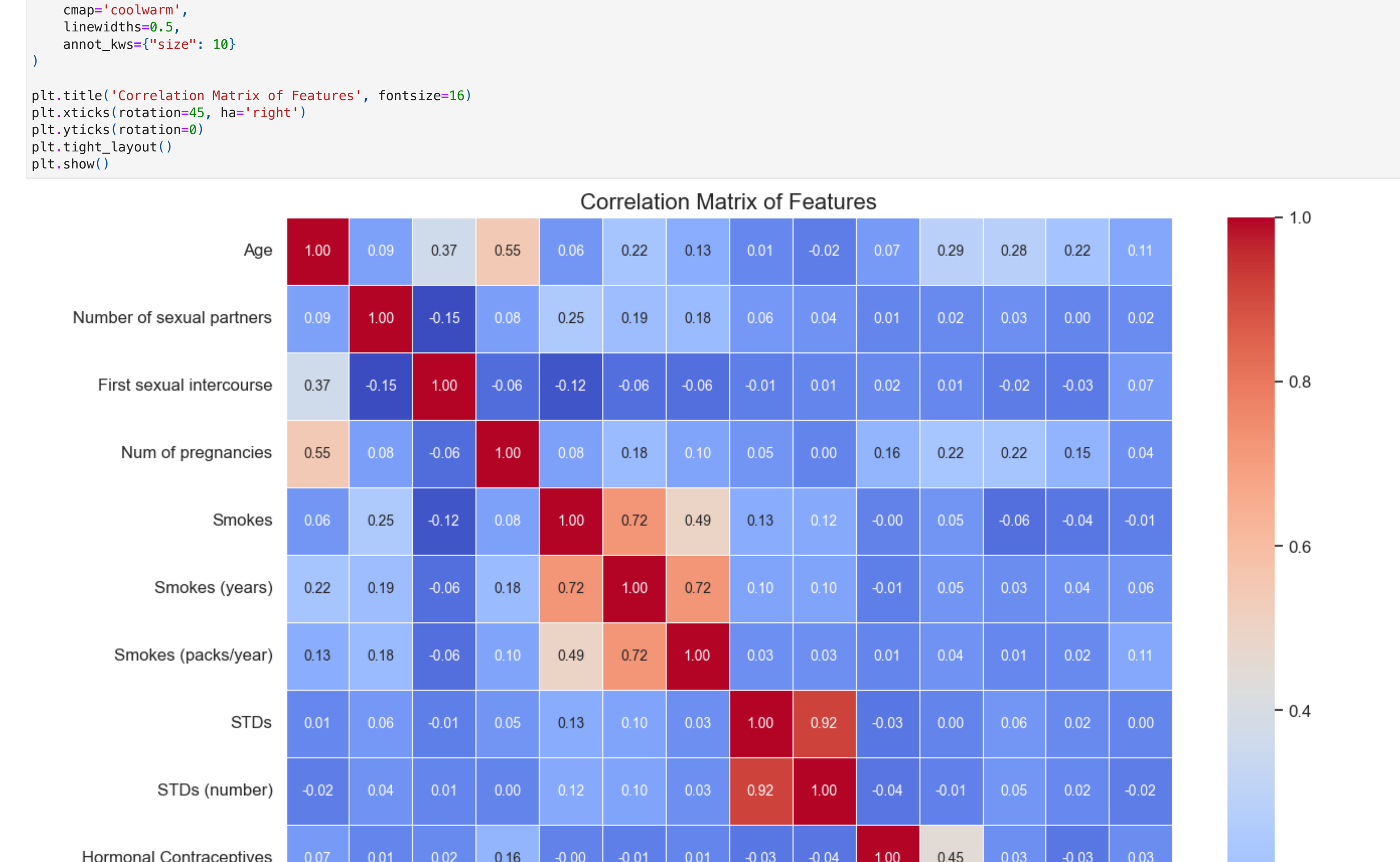
```
/var/folders/ff/sds78knx6lv3kyl_xckqx8cc0000gn/T/ipykernel_8023/1891037928.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=importances[indices], y=features[indices], palette='viridis')
```



10. Correlation Matrix

Generate and visualize the correlation matrix to understand relationships between features.

In [24]:
```python
# Compute the correlation matrix
corr_matrix = data[user_features + [target]].corr()

# Plot the heatmap
plt.figure(figsize=(14, 12))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    linewidths=0.5,
    annot_kws={"size": 10}
)

plt.title('Correlation Matrix of Features', fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



11: Define the Prediction Function

Create a function to predict the risk of cervical cancer based on user input.

In [25]:
```python
def predict_risk(input_data):
    """
    Predicts the risk probability of cervical cancer based on input features.

    Parameters:
    input_data (dict): A dictionary containing feature values.

    Returns:
    dict: A dictionary with the risk probability.
    """
    # Convert input data to DataFrame
    input_df = pd.DataFrame([input_data])

    # Ensure all required features are present
    for feature in X.columns:
        if feature not in input_df.columns:
            input_df[feature] = np.nan

    # Impute missing values
    input_df_imputed = pd.DataFrame(imputer.transform(input_df), columns=input_df.columns)

    # Scale numerical features
    input_df_imputed[numerical_features] = scaler.transform(input_df_imputed[numerical_features])

    # Predict risk probability
    risk_proba = model.predict_proba(input_df_imputed)[0][1]

    return {'Risk Probability': risk_proba}
```

12: Example User Input and Prediction

Provide an example of how to use the prediction function with sample user data.

In [26]:
```python
# Example user input
user_input = {
    'Age': 25,
    'Number of sexual partners': 0,
    'First sexual intercourse': 0,
    'Num of pregnancies': 0,
    'Smokes': 0,
    'Smokes (years)': 0,
    'Smokes (packs/year)': 0,
    'STDs': 0,
    'STDs (number)': 0,
    'Hormonal Contraceptives': 0,
    'Hormonal Contraceptives (years)': 0,
    'IUD': 0,
    'IUD (years)': 0,
}

# Make a prediction
prediction = predict_risk(user_input)

# Display the prediction result
print("\nRisk Prediction for the User:")
print(f"Risk Probability: {prediction['Risk Probability']*100:.2f}%")
```

```
Risk Prediction for the User:
Risk Probability: 1.87%
```

13: Save the Model for Future Use

Persist the trained model to disk for later deployment or inference.

In [27]:
```python
# Save the trained model to a file
joblib.dump(model, "ML_model.pkl")
print("Model saved as 'ML_model.pkl'")
```

```
Model saved as 'ML_model.pkl'
```

Summary

This notebook provides a complete workflow for detecting cervical cancer based on cytology data using traditional machine learning techniques. It encompasses data loading and inspection, preprocessing steps such as handling missing values and feature scaling, and addressing class imbalance with SMOTE. A Random Forest Classifier is trained and evaluated with detailed classification metrics. The notebook includes insightful visualizations like feature importance and a correlation matrix to understand feature relationships. Additionally, it offers a user-friendly prediction function for assessing individual risk and demonstrates how to save the trained model for future use. The well-structured and commented code ensures ease of understanding and adaptability for similar datasets and classification tasks.