In [1]:

```python
# Import necessary libraries
import pandas as pd

# Load the dataset
data = pd.read_csv("churn.csv")

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())
```

```
C:\Users\Vyshnavi\anaconda\lib\site-packages\pandas\core\computation\expressions.py:20: UserWarn
ing: Pandas requires version '2.7.3' or newer of 'numexpr' (version '2.7.1' currently installe
d).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
First few rows of the dataset:
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

In [2]:

```python
# Check for missing values
print("Missing values in the dataset:")
print(data.isnull().sum())
```

```
Missing values in the dataset:
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [3]:

```python
# Perform one-hot encoding for 'Geography' column
data = pd.get_dummies(data, columns=['Geography'], drop_first=True)

# Binary encode 'Gender' column (Female: 0, Male: 1)
data['Gender'] = data['Gender'].map({'Female': 0, 'Male': 1})

# Display the first few rows of the updated dataset
print("First few rows of the updated dataset:")
print(data.head())
```

```
First few rows of the updated dataset:
   RowNumber  CustomerId   Surname  CreditScore  Gender  Age  Tenure  \
0          1    15634602  Hargrave          619       0   42       2
1          2    15647311      Hill          608       0   41       1
2          3    15619304      Onio          502       0   42       8
3          4    15701354      Boni          699       0   39       1
4          5    15737888  Mitchell          850       0   43       2
```

```
     Balance  NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary  \
0       0.00              1          1               1        101348.88   
1   83807.86              1          0               1        112542.58   
2  159660.80              3          1               0        113931.57   
3       0.00              2          0               0         93826.63   
4  125510.82              1          1               1         79084.10   

   Exited  Geography_Germany  Geography_Spain  
0       1                  0                0  
1       0                  0                1  
2       1                  0                0  
3       0                  0                0  
4       0                  0                1  
```

In [4]:
```
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler
scaler = StandardScaler()

# Select numerical features for scaling
numerical_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSal

# Scale the numerical features
data[numerical_features] = scaler.fit_transform(data[numerical_features])

# Display the first few rows of the scaled dataset
print("First few rows of the scaled dataset:")
print(data.head())
```

```
First few rows of the scaled dataset:
   RowNumber  CustomerId   Surname  CreditScore  Gender       Age    Tenure  \
0          1    15634602  Hargrave    -0.326221       0  0.293517 -1.041760   
1          2    15647311      Hill    -0.440036       0  0.198164 -1.387538   
2          3    15619304      Onio    -1.536794       0  0.293517  1.032908   
3          4    15701354      Boni     0.501521       0  0.007457 -1.387538   
4          5    15737888  Mitchell     2.063884       0  0.388871 -1.041760   

    Balance  NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary  \
0 -1.225848      -0.911583          1               1         0.021886   
1  0.117350      -0.911583          0               1         0.216534   
2  1.333053       2.527057          1               0         0.240687   
3 -1.225848       0.807737          0               0        -0.108918   
4  0.785728      -0.911583          1               1        -0.365276   

   Exited  Geography_Germany  Geography_Spain  
0       1                  0                0  
1       0                  0                1  
2       1                  0                0  
3       0                  0                0  
4       0                  0                1  
```

In [5]:
```
from sklearn.model_selection import train_test_split

# Split the dataset into features (X) and target variable (y)
X = data.drop(['RowNumber', 'CustomerId', 'Surname', 'Exited'], axis=1)
y = data['Exited']

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

# Display the shapes of the training and testing sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (8000, 11)
Shape of X_test: (2000, 11)
Shape of y_train: (8000,)
Shape of y_test: (2000,)
```

In [6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Initialize and train the logistic regression model
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = logistic_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8115

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      1607
           1       0.56      0.20      0.30       393

    accuracy                           0.81      2000
   macro avg       0.69      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000


Confusion Matrix:
[[1544   63]
 [ 314   79]]
```

In [7]:

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameters to tune
param_grid = {
    'penalty': ['l1', 'l2'],  # Regularization penalty (L1 or L2)
    'C': [0.001, 0.01, 0.1, 1, 10, 100]  # Inverse of regularization strength
}

# Initialize grid search
grid_search = GridSearchCV(LogisticRegression(random_state=42), param_grid, cv=5, scoring='accu

# Perform grid search
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Initialize logistic regression model with the best hyperparameters
best_logistic_model = LogisticRegression(**best_params, random_state=42)

# Train the model with the best hyperparameters
best_logistic_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred_best = best_logistic_model.predict(X_test)

# Evaluate the model
accuracy_best = accuracy_score(y_test, y_pred_best)
print("\nAccuracy with Best Hyperparameters:", accuracy_best)
print("\nClassification Report with Best Hyperparameters:")
print(classification_report(y_test, y_pred_best))
print("\nConfusion Matrix with Best Hyperparameters:")
print(confusion_matrix(y_test, y_pred_best))
```

```
Best Hyperparameters: {'C': 10, 'penalty': 'l2'}

Accuracy with Best Hyperparameters: 0.811

Classification Report with Best Hyperparameters:
              precision    recall  f1-score   support
```

```
           0       0.83      0.96      0.89      1607
           1       0.55      0.20      0.29       393

    accuracy                           0.81      2000
   macro avg       0.69      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000


Confusion Matrix with Best Hyperparameters:
[[1543   64]
 [ 314   79]]
```

```
C:\Users\Vyshnavi\anaconda\lib\site-packages\sklearn\model_selection\_validation.py:425: FitFail
edWarning:
30 fits failed out of a total of 60.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Vyshnavi\anaconda\lib\site-packages\sklearn\model_selection\_validation.py", li
ne 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Vyshnavi\anaconda\lib\site-packages\sklearn\base.py", line 1152, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "C:\Users\Vyshnavi\anaconda\lib\site-packages\sklearn\linear_model\_logistic.py", line 11
69, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\Vyshnavi\anaconda\lib\site-packages\sklearn\linear_model\_logistic.py", line 5
6, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Vyshnavi\anaconda\lib\site-packages\sklearn\model_selection\_search.py:979: UserWarnin
g: One or more of the test scores are non-finite: [     nan 0.794          nan 0.804          nan
0.81           nan 0.810125
      nan 0.810625      nan 0.810625]
  warnings.warn(
```

In [8]:

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the random forest classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy with Random Forest:", accuracy_rf)
print("\nClassification Report with Random Forest:")
print(classification_report(y_test, y_pred_rf))
print("\nConfusion Matrix with Random Forest:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Accuracy with Random Forest: 0.866

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

           0       0.88      0.97      0.92      1607
           1       0.77      0.46      0.57       393

    accuracy                           0.87      2000
   macro avg       0.82      0.71      0.75      2000
weighted avg       0.86      0.87      0.85      2000


Confusion Matrix with Random Forest:
[[1552   55]
 [ 213  180]]
```

In [11]:

```
# Define hyperparameters to tune
param_grid_rf = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Initialize grid search for random forest
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, sco

# Perform grid search
grid_search_rf.fit(X_train, y_train)

# Get the best hyperparameters
best_params_rf = grid_search_rf.best_params_
print("Best Hyperparameters for Random Forest:", best_params_rf)

# Initialize random forest model with the best hyperparameters
best_rf_model = RandomForestClassifier(**best_params_rf, random_state=42)

# Train the model with the best hyperparameters
best_rf_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred_best_rf = best_rf_model.predict(X_test)

# Evaluate the model
accuracy_best_rf = accuracy_score(y_test, y_pred_best_rf)
print("\nAccuracy with Best Hyperparameters for Random Forest:", accuracy_best_rf)
print("\nClassification Report with Best Hyperparameters for Random Forest:")
print(classification_report(y_test, y_pred_best_rf))
print("\nConfusion Matrix with Best Hyperparameters for Random Forest:")
print(confusion_matrix(y_test, y_pred_best_rf))
```

```
Best Hyperparameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_sp
lit': 2, 'n_estimators': 100}

Accuracy with Best Hyperparameters for Random Forest: 0.8645

Classification Report with Best Hyperparameters for Random Forest:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1607
           1       0.76      0.45      0.57       393

    accuracy                           0.86      2000
   macro avg       0.82      0.71      0.74      2000
weighted avg       0.86      0.86      0.85      2000


Confusion Matrix with Best Hyperparameters for Random Forest:
[[1552   55]
 [ 216  177]]
```

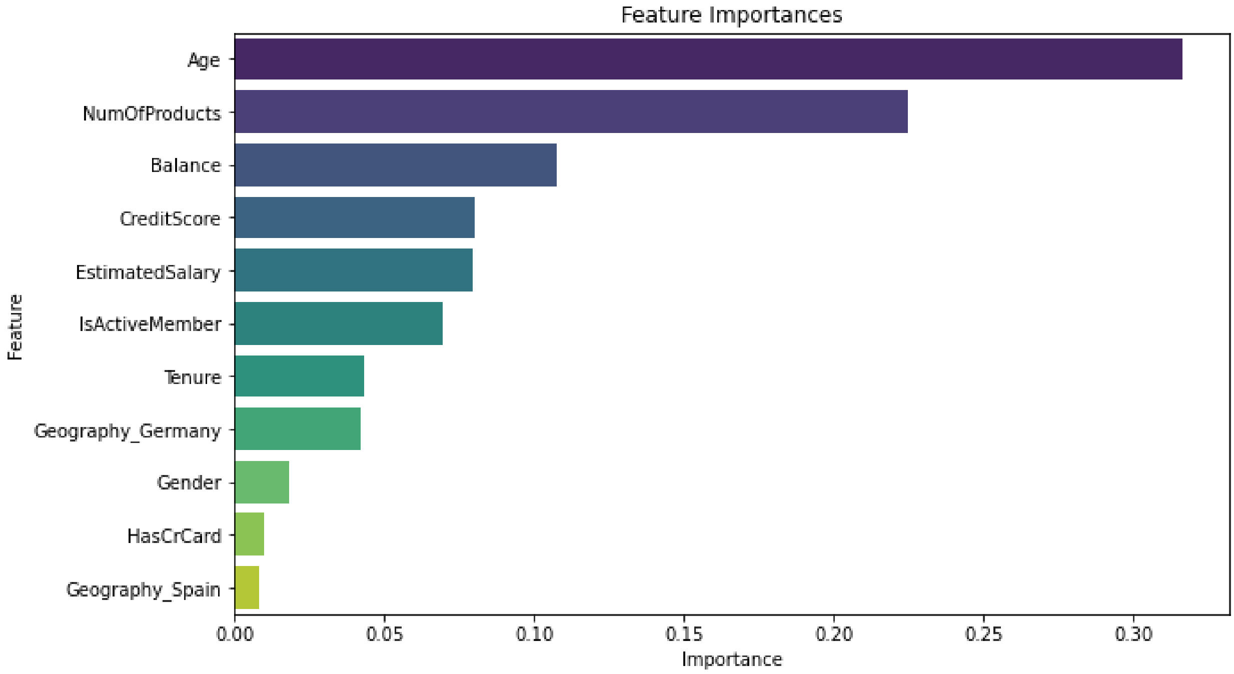
In [13]:

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

# Extract feature importances from the trained random forest model
feature_importances = best_rf_model.feature_importances_

# Create a DataFrame to display feature importances
importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [14]:
```
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline

# Define SMOTE and random forest pipeline
smote = SMOTE(random_state=42)
rf_pipeline = Pipeline([('smote', smote), ('rf', best_rf_model)])

# Train the model with SMOTE for oversampling
rf_pipeline.fit(X_train, y_train)

# Make predictions on the testing set
y_pred_rf_smote = rf_pipeline.predict(X_test)

# Evaluate the model with SMOTE
accuracy_rf_smote = accuracy_score(y_test, y_pred_rf_smote)
print("Accuracy with SMOTE:", accuracy_rf_smote)
print("\nClassification Report with SMOTE:")
print(classification_report(y_test, y_pred_rf_smote))
print("\nConfusion Matrix with SMOTE:")
print(confusion_matrix(y_test, y_pred_rf_smote))
```

```
Accuracy with SMOTE: 0.827

Classification Report with SMOTE:
              precision    recall  f1-score   support

           0       0.92      0.86      0.89      1607
           1       0.55      0.70      0.61       393

    accuracy                           0.83      2000
   macro avg       0.73      0.78      0.75      2000
weighted avg       0.85      0.83      0.83      2000


Confusion Matrix with SMOTE:
[[1378  229]
 [ 117  276]]
```

In [33]:
```
pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-2.0.3-py3-none-win_amd64.whl (99.8 MB)
Requirement already satisfied: numpy in c:\users\vyshnavi\anaconda\lib\site-packages (from xgboo
st) (1.24.4)
Requirement already satisfied: scipy in c:\users\vyshnavi\anaconda\lib\site-packages (from xgboo
st) (1.5.2)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.3
Note: you may need to restart the kernel to use updated packages.
```

In [34]:

```python
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Train XGBoost model
xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train)

# Predictions
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate XGBoost model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print("Accuracy with XGBoost:", accuracy_xgb)

# Classification report
print("\nClassification Report with XGBoost:")
print(classification_report(y_test, y_pred_xgb))

# Confusion matrix
print("\nConfusion Matrix with XGBoost:")
print(confusion_matrix(y_test, y_pred_xgb))

# Visualize feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x=xgb_model.feature_importances_, y=X.columns)
plt.title('Feature Importances with XGBoost')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

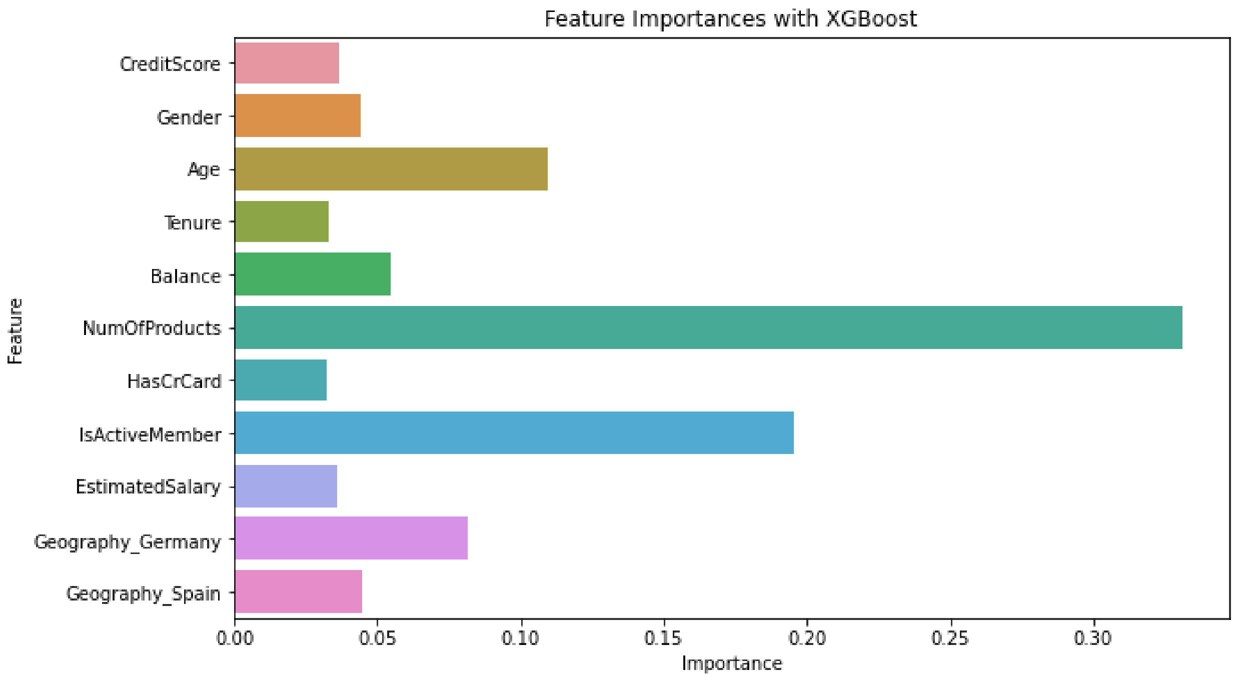
```
Accuracy with XGBoost: 0.864

Classification Report with XGBoost:
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      1607
           1       0.71      0.51      0.60       393

    accuracy                           0.86      2000
   macro avg       0.80      0.73      0.76      2000
weighted avg       0.85      0.86      0.86      2000


Confusion Matrix with XGBoost:
[[1526   81]
 [ 191  202]]
```



In [ ]:

```

```