Case 2: Predicting Student Success in Online Courses

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go

from sklearn.preprocessing import label_binarize

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc
```

In [2]:

```
df = pd.read_csv('student_course_data.csv')
df.head()
```

Out[2]:

	student_id	age	gender	major	year	region	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started
0	1	19	Female	IT	4	Rajasthan	9	10	8	94	4
1	2	18	Female	Bio-Technology	1	Telenagana	2	6	10	98	5
2	3	22	Female	Bio-Technology	3	Gujarat	2	14	5	71	7
3	4	21	Male	Civil Engineering	3	Tamil Nadu	4	5	6	59	7
4	5	21	Female	Electrical Engineering	2	Karnataka	9	14	5	98	7

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   student_id                1000 non-null   int64 
 1   age                       1000 non-null   int64 
 2   gender                    1000 non-null   object
 3   major                     1000 non-null   object
 4   year                      1000 non-null   int64 
 5   region                    1000 non-null   object
 6   logins_per_week           1000 non-null   int64 
 7   videos_watched            1000 non-null   int64 
 8   time_spent                1000 non-null   int64 
 9   avg_quiz_score            1000 non-null   int64 
 10  courses_started           1000 non-null   int64 
 11  courses_completed         1000 non-null   int64 
 12  avg_score_across_courses  1000 non-null   int64 
dtypes: int64(10), object(3)
memory usage: 101.7+ KB
```

In [4]:

```
#Check the missing values
df.isna().sum()
```

Out[4]:

```
student_id                  0
age                         0
gender                      0
major                       0
year                        0
region                      0
logins_per_week             0
videos_watched              0
time_spent                  0
avg_quiz_score              0
courses_started             0
courses_completed           0
avg_score_across_courses    0
dtype: int64
```

In [5]:
```
#Check for duplicates
df.duplicated().sum()
```

Out[5]:
```
0
```

In [6]:
```
#Unique values in the dataset
df.nunique()
```

Out[6]:
```
student_id                1000
age                         10
gender                       2
major                        8
year                         4
region                      11
logins_per_week             10
videos_watched              16
time_spent                  13
avg_quiz_score              86
courses_started              7
courses_completed            8
avg_score_across_courses    86
dtype: int64
```

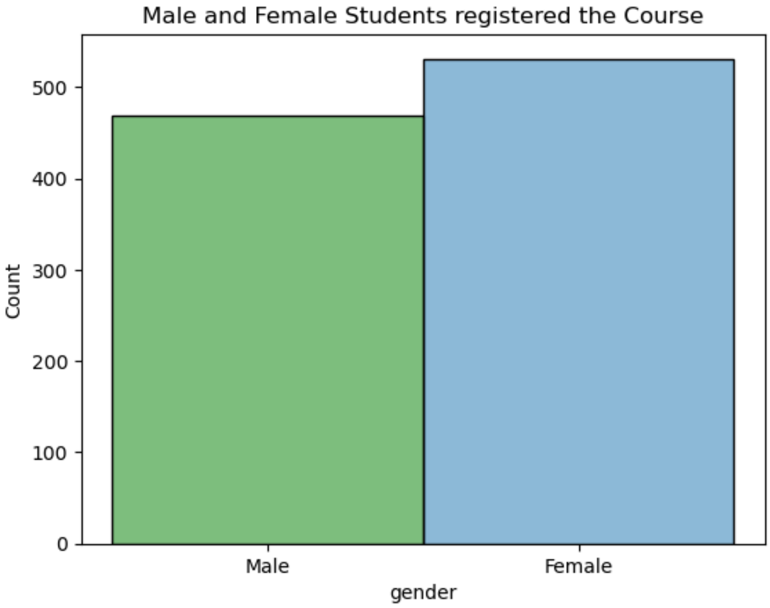
In [7]:
```
#Statistics of the data
df.describe()
```

Out[7]:

	student_id	age	year	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started	courses_comp
count	1000.000000	1000.00000	1000.00000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.0
mean	500.500000	22.51400	2.47600	5.708000	12.536000	9.07400	57.854000	4.047000	1.9
std	288.819436	2.84357	1.12191	2.923916	4.715394	3.75881	24.830047	2.022098	1.7
min	1.000000	18.00000	1.00000	1.000000	5.000000	3.00000	15.000000	1.000000	0.0
25%	250.750000	20.00000	1.00000	3.000000	8.000000	6.00000	37.000000	2.000000	0.0
50%	500.500000	22.00000	2.00000	6.000000	12.500000	9.00000	58.000000	4.000000	1.0
75%	750.250000	25.00000	3.00000	8.000000	17.000000	13.00000	80.000000	6.000000	3.0
max	1000.000000	27.00000	4.00000	10.000000	20.000000	15.00000	100.000000	7.000000	7.0

Data Visulaization

In [8]:
```
#Distribution of the gender
Female = df[df['gender'] == 'Female']
Male = df[df['gender'] == 'Male' ]

# Plot histogram for Male and Female
sns.histplot(data=Male, x='gender', kde=True, color='g')
sns.histplot(data=Female, x='gender', kde=True)

plt.title('Male and Female Students registered the Course')
plt.show()
```



In [9]:

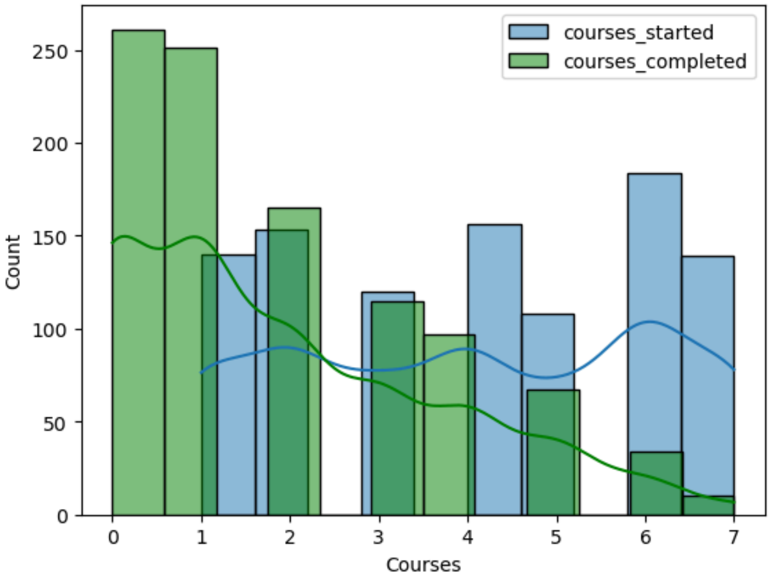
```
#Count of male and female stundets in the course
Female = df[df['gender'] == 'Female']['gender'].count()
Male = df[df['gender'] == 'Male']['gender'].count()

print('The number of female students in the course=', Female)
print('The number of male students in the course=', Male)
```

```
The number of female students in the course= 531
The number of male students in the course= 469
```

Historical Data

In [10]:

```
##Distribution of student course started and course completed
cs = df['courses_started']
cc = df['courses_completed']
sns.histplot(cs, kde=True, label = 'courses_started', stat='count', bins=10)
sns.histplot(cc, kde=True, label = 'courses_completed', stat='count', color='g')

plt.xlabel('Courses')
plt.legend()

plt.show()
```



In [11]:

```
cc.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 1000 entries, 0 to 999
Series name: courses_completed
Non-Null Count  Dtype
--------------  -----
1000 non-null   int64
dtypes: int64(1)
memory usage: 7.9 KB
```

Number of students started and comleted the courses

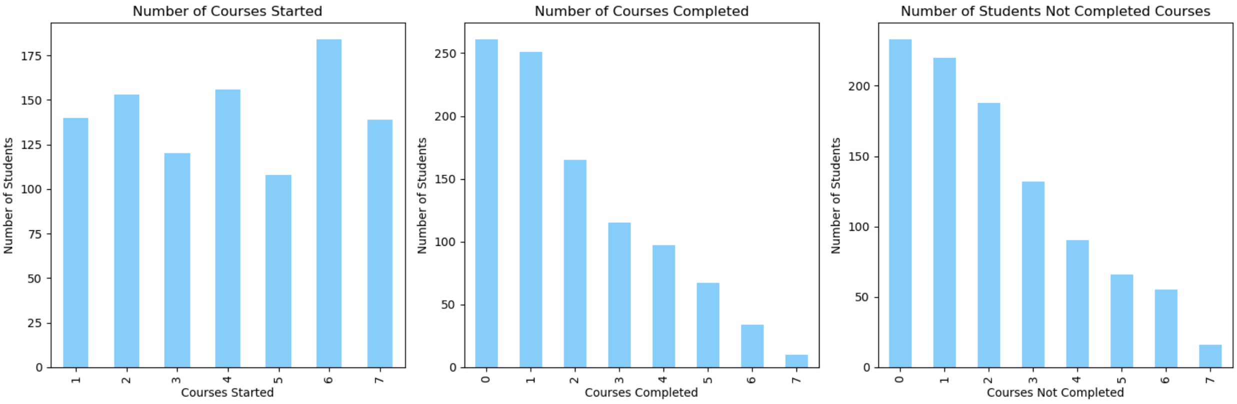
In [12]:

```
print('Number of courses started:')
cs = df['courses_started'].value_counts().sort_index()
print(cs)

print('Number of course completed after starting:')
cc = df['courses_completed'].value_counts().sort_index()
print(cc)


#Adding a column for courses not completed (i.e: courses_started = courses_completed)
df['courses_not_completed'] = df['courses_started'] - df['courses_completed']

cnc = df['courses_not_completed'].value_counts().sort_index()
print(cnc)
```

```
Number of courses started:
courses_started
1    140
2    153
3    120
4    156
5    108
6    184
7    139
Name: count, dtype: int64
Number of course completed after starting:
courses_completed
0    261
1    251
2    165
3    115
4     97
5     67
6     34
7     10
Name: count, dtype: int64
courses_not_completed
0    233
1    220
2    188
3    132
4     90
5     66
6     55
7     16
Name: count, dtype: int64
```

In [13]:

```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))  # Adjust figsize as needed

# Plot for courses started
cs.plot(kind='bar', color='lightskyblue', ax=axs[0])
axs[0].set_title('Number of Courses Started')
axs[0].set_xlabel('Courses Started')
axs[0].set_ylabel('Number of Students')

# Plot for courses completed
cc.plot(kind='bar', color='lightskyblue', ax=axs[1])
axs[1].set_title('Number of Courses Completed')
axs[1].set_xlabel('Courses Completed')
axs[1].set_ylabel('Number of Students')

# Plot for students not completed courses
cnc.plot(kind='bar', color='lightskyblue', ax=axs[2])
axs[2].set_title('Number of Students Not Completed Courses')
axs[2].set_xlabel('Courses Not Completed')
axs[2].set_ylabel('Number of Students')

plt.tight_layout()

# Show the plots
plt.show()
```



Course Engagement

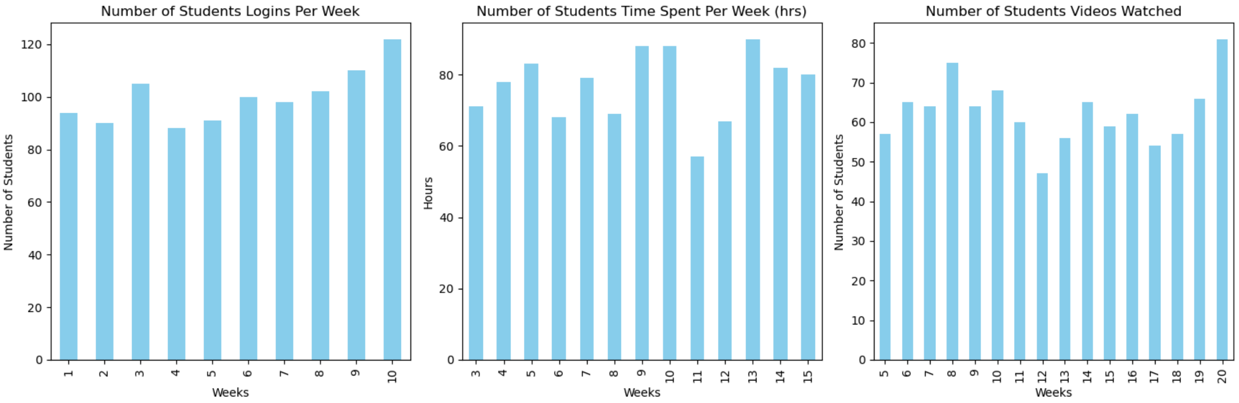
In [14]:

```
logins = df['logins_per_week'].value_counts().sort_index()
print('Number of students logins per week:', logins)

videos = df['videos_watched'].value_counts().sort_index()
print('Number of students videos watch per week:',videos)

time_spent = df['time_spent'].value_counts().sort_index()
print('Number of students time spent per week:',time_spent)
```

```
Number of students logins per week: logins_per_week
1      94
2      90
3     105
4      88
5      91
6     100
7      98
8     102
9     110
10    122
Name: count, dtype: int64
Number of students videos watch per week: videos_watched
5     57
6     65
7     64
8     75
9     64
10    68
11    60
12    47
13    56
14    65
15    59
16    62
17    54
18    57
19    66
20    81
Name: count, dtype: int64
Number of students time spent per week: time_spent
3     71
4     78
5     83
6     68
7     79
8     69
9     88
10    88
11    57
12    67
13    90
14    82
15    80
Name: count, dtype: int64
```

In [15]:

```python
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

logins.plot(kind='bar', color='skyblue', ax=ax[0])
ax[0].set_title('Number of Students Logins Per Week')
ax[0].set_ylabel('Number of Students')
ax[0].set_xlabel('Weeks')

time_spent.plot(kind='bar', color='skyblue', ax=ax[1])
ax[1].set_title('Number of Students Time Spent Per Week (hrs)')
ax[1].set_ylabel('Hours')
ax[1].set_xlabel('Weeks')

videos.plot(kind='bar', color='skyblue', ax=ax[2])
ax[2].set_title('Number of Students Videos Watched')
ax[2].set_ylabel('Number of Students')
ax[2].set_xlabel('Weeks')

plt.tight_layout()
plt.show()
```



In [16]:

```python
## adding the completion rate of the courses
df['completion_rate'] = df['courses_completed'] / df['courses_started']
df.head()
```

Out[16]:

	student_id	age	gender	major	year	region	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started
0	1	19	Female	IT	4	Rajasthan	9	10	8	94	4
1	2	18	Female	Bio-Technology	1	Telenagana	2	6	10	98	5
2	3	22	Female	Bio-Technology	3	Gujarat	2	14	5	71	7
3	4	21	Male	Civil Engineering	3	Tamil Nadu	4	5	6	59	7
4	5	21	Female	Electrical Engineering	2	Karnataka	9	14	5	98	7

In [17]:

```
data = df
data.describe()
```

Out[17]:

	student_id	age	year	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started	courses_comp
count	1000.000000	1000.00000	1000.00000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.0
mean	500.500000	22.51400	2.47600	5.708000	12.536000	9.07400	57.854000	4.047000	1.9
std	288.819436	2.84357	1.12191	2.923916	4.715394	3.75881	24.830047	2.022098	1.7
min	1.000000	18.00000	1.00000	1.000000	5.000000	3.00000	15.000000	1.000000	0.0
25%	250.750000	20.00000	1.00000	3.000000	8.000000	6.00000	37.000000	2.000000	0.0
50%	500.500000	22.00000	2.00000	6.000000	12.500000	9.00000	58.000000	4.000000	1.0
75%	750.250000	25.00000	3.00000	8.000000	17.000000	13.00000	80.000000	6.000000	3.0
max	1000.000000	27.00000	4.00000	10.000000	20.000000	15.00000	100.000000	7.000000	7.0

Feature Engineering

In [18]:

```
#Manually defining a threshold for the status and
data['course_completed'] = (
    (data['avg_quiz_score'] >= 55) &
    (data['avg_score_across_courses'] >= 25) &
    (data['courses_completed'] >= 2) &

    (data['completion_rate'] >= 0.50)

).astype(int)
```

In [19]:

```
data.head()
```

Out[19]:

	student_id	age	gender	major	year	region	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started
0	1	19	Female	IT	4	Rajasthan	9	10	8	94	4
1	2	18	Female	Bio-Technology	1	Telenagana	2	6	10	98	5
2	3	22	Female	Bio-Technology	3	Gujarat	2	14	5	71	7
3	4	21	Male	Civil Engineering	3	Tamil Nadu	4	5	6	59	7
4	5	21	Female	Electrical Engineering	2	Karnataka	9	14	5	98	7

In [20]:

```
data['course_completed'].value_counts()
```

Out[20]:

```
course_completed
0    782
1    218
Name: count, dtype: int64
```

Data Processing

In [21]:

```
#Drop the non numeric columns and define the features
X = data.drop(['student_id', 'course_completed'], axis=1)
y = data['course_completed']
```

In [22]:

```
X.head()
```

Out[22]:

	age	gender	major	year	region	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started	courses_cc
0	19	Female	IT	4	Rajasthan	9	10	8	94	4	
1	18	Female	Bio-Technology	1	Telenagana	2	6	10	98	5	
2	22	Female	Bio-Technology	3	Gujarat	2	14	5	71	7	
3	21	Male	Civil Engineering	3	Tamil Nadu	4	5	6	59	7	
4	21	Female	Electrical Engineering	2	Karnataka	9	14	5	98	7	

In [23]:

```
print("Categories in 'gender' variable:     ",end=" " )
print(df['gender'].unique())

print("Categories in 'major' variable:  ",end=" ")
print(df['major'].unique())

print("Categories in'region' variable:",end=" " )
print(df['region'].unique())
```

```
Categories in 'gender' variable:      ['Female' 'Male']
Categories in 'major' variable:   ['IT' 'Bio-Technology' 'Civil Engineering' 'Electrical Engineering'
 'Chemical Engineering' 'Computer Science' 'Mechanical Engineering'
 'Environmental Science']
Categories in'region' variable: ['Rajasthan' 'Telenagana' 'Gujarat' 'Tamil Nadu' 'Karnataka'
 'Andra Pradesh' 'Kerala' 'Delhi' 'UP' 'Maharashtra' 'West Bengal']
```

In [24]:

```
# Create Column Transformer with 3 types of transformers
num_features = X.select_dtypes(exclude="object").columns
cat_features = X.select_dtypes(include="object").columns

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder()

preprocessor = ColumnTransformer(
    [
        ("OneHotEncoder", oh_transformer, cat_features),
         ("StandardScaler", numeric_transformer, num_features),
    ]
)
```

In [25]:

```
X = preprocessor.fit_transform(X)
```

In [26]:

```
X.shape
```

Out[26]:

```
(1000, 32)
```

In [27]:

```
feature_names = preprocessor.get_feature_names_out()
```

In [28]:

```
feature_names
```

Out[28]:

```
array(['OneHotEncoder__gender_Female', 'OneHotEncoder__gender_Male',
       'OneHotEncoder__major_Bio-Technology',
       'OneHotEncoder__major_Chemical Engineering',
       'OneHotEncoder__major_Civil Engineering',
       'OneHotEncoder__major_Computer Science',
       'OneHotEncoder__major_Electrical Engineering',
       'OneHotEncoder__major_Environmental Science',
       'OneHotEncoder__major_IT',
       'OneHotEncoder__major_Mechanical Engineering',
       'OneHotEncoder__region_Andra Pradesh',
       'OneHotEncoder__region_Delhi', 'OneHotEncoder__region_Gujarat',
       'OneHotEncoder__region_Karnataka', 'OneHotEncoder__region_Kerala',
       'OneHotEncoder__region_Maharashtra',
       'OneHotEncoder__region_Rajasthan',
       'OneHotEncoder__region_Tamil Nadu',
       'OneHotEncoder__region_Telenagana', 'OneHotEncoder__region_UP',
       'OneHotEncoder__region_West Bengal', 'StandardScaler__age',
       'StandardScaler__year', 'StandardScaler__logins_per_week',
       'StandardScaler__videos_watched', 'StandardScaler__time_spent',
       'StandardScaler__avg_quiz_score',
       'StandardScaler__courses_started',
       'StandardScaler__courses_completed',
       'StandardScaler__avg_score_across_courses',
       'StandardScaler__courses_not_completed',
       'StandardScaler__completion_rate'], dtype=object)
```

In [29]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

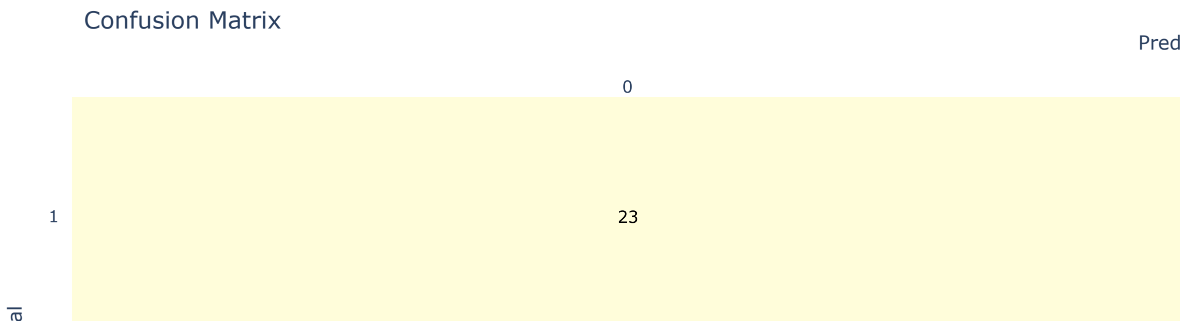
Logistic Regression

In [30]:
```python
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train, y_train)

# Make predictions
y_log_pred = logistic_model.predict(X_test)
```

In [31]:
```python
print("Model Accuracy:", accuracy_score(y_test, y_log_pred))
print("\nClassification Report:\n", classification_report(y_test, y_log_pred))
```

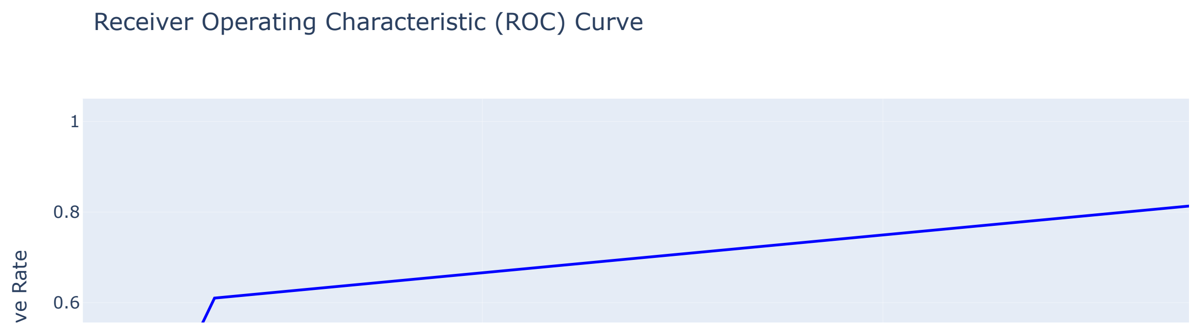
```
Model Accuracy: 0.87

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.93      0.92       241
           1       0.69      0.61      0.65        59

    accuracy                           0.87       300
   macro avg       0.80      0.77      0.78       300
weighted avg       0.86      0.87      0.87       300
```

In [32]:
```python
cm = confusion_matrix(y_test, y_log_pred)
fig = ff.create_annotated_heatmap(z=cm, x=list(range(len(cm))), y=list(range(len(cm))), colorscale='ylorbr')
fig.update_layout(title='Confusion Matrix', xaxis_title='Predicted', yaxis_title='Actual')
fig.show()
```



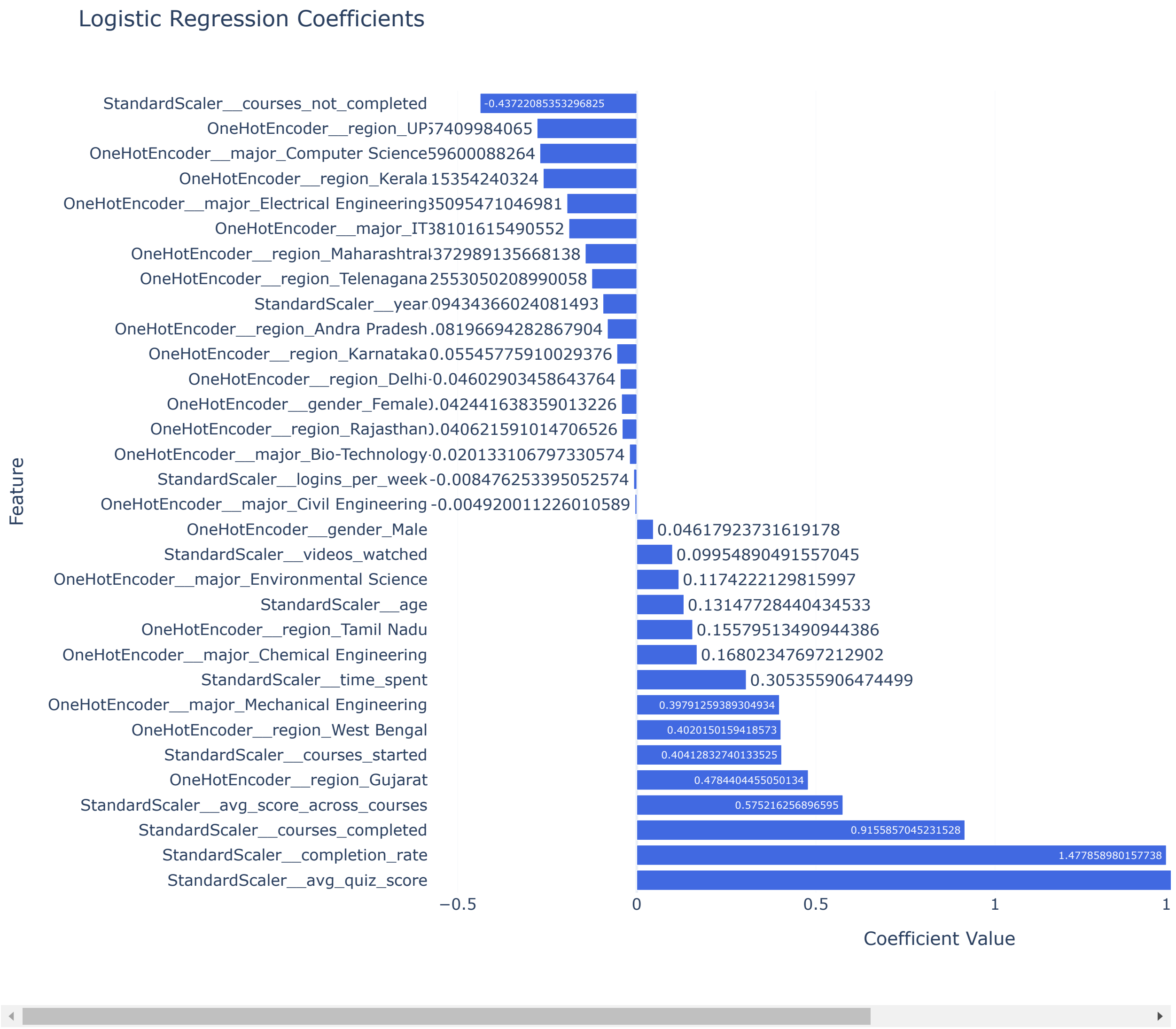
In [33]:
```python
fpr, tpr, thresholds = roc_curve(y_test, y_log_pred)
roc_auc_log = auc(fpr, tpr)
```

In [34]:
```python
# Create a Plotly figure for the ROC curve
fig = go.Figure()

# Add the ROC curve trace
fig.add_trace(go.Scatter(x=fpr,
                          y=tpr,
                          mode='lines',
                          name='ROC Curve (AUC = {:.2f})'.format(roc_auc_log),
                          line=dict(color='blue')))

# Add a diagonal line for random guessing
fig.add_trace(go.Scatter(x=[0, 1],
                          y=[0, 1],
                          mode='lines',
                          name='Random Guessing',
                          line=dict(color='red', dash='dash')))

# Update layout of the figure
fig.update_layout(title='Receiver Operating Characteristic (ROC) Curve',
                  xaxis_title='False Positive Rate',
                  yaxis_title='True Positive Rate',
                  xaxis=dict(range=[0.0, 1.0]),
                  yaxis=dict(range=[0.0, 1.05]),
                  showlegend=True)

# Show the figure
fig.show()
```



In [35]:

```
# Analyze coefficients of logistic regression
coefficients = pd.Series(logistic_model.coef_[0], index=feature_names).sort_values(ascending=False)

fig = go.Figure()

# Add bars for each coefficient
fig.add_trace(go.Bar(
    x=coefficients.values,
    y=coefficients.index,
    orientation='h',
    marker=dict(color='royalblue'),
    text=coefficients.values,
    textposition='auto'
))

# Update layout for better visualization
fig.update_layout(
    title='Logistic Regression Coefficients',
    xaxis_title='Coefficient Value',
    yaxis_title='Feature',
    template='plotly_white',
    height=800,
    width=1200
)

# Show the interactive plot
fig.show()
```



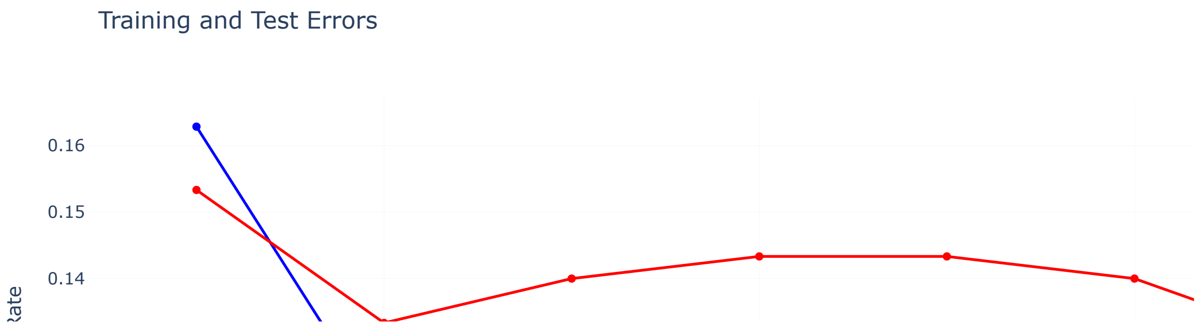
In [36]:

```
#Train and Test Error
train_errors = []
test_errors = []

for i in range(1, 12):  # training for a number of iterations
    logistic_model.fit(X_train[:i * int(len(X_train) / 10)], y_train[:i * int(len(y_train) / 10)]) # Incrementally fit on mo

    # Calculate training accuracy
    train_pred = logistic_model.predict(X_train)
    train_accuracy = accuracy_score(y_train, train_pred)
    train_errors.append(1 - train_accuracy) # Store error (1 - accuracy)

    # Calculate test accuracy
    test_pred = logistic_model.predict(X_test)
    test_accuracy = accuracy_score(y_test, test_pred)
    test_errors.append(1 - test_accuracy) # Store error (1 - accuracy)
```

In [37]:

```
fig = go.Figure()

# Add training error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(train_errors) + 1)),
    y=train_errors,
    mode='lines+markers',
    name='Training Error',
    line=dict(color='blue')
))

# Add test error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(test_errors) + 1)),
    y=test_errors,
    mode='lines+markers',
    name='Test Error',
    line=dict(color='red')
))

# Update layout for better visualization
```

```
fig.update_layout(
    title='Training and Test Errors',
    xaxis_title='Iterations',
    yaxis_title='Error Rate',
    legend_title='Error Type',
    template='plotly_white',
)

fig.show()
```



Random Forest Model

In [38]:

```
# Initialize and train the Random Forest classifier
model_forest = RandomForestClassifier(random_state=42)
model_forest.fit(X_train, y_train)
```

Out[38]:

```
▾ RandomForestClassifier

RandomForestClassifier(random_state=42)
```

In [39]:

```
# Predict on the test set
y_pred = model_forest.predict(X_test)
```

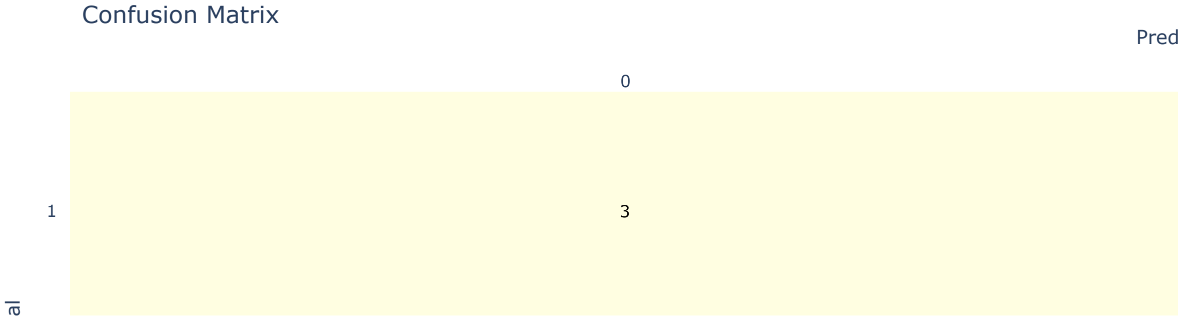
In [40]:

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
```

```
Model Accuracy: 0.99
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       241
           1       1.00      0.95      0.97        59

    accuracy                           0.99       300
   macro avg       0.99      0.97      0.98       300
weighted avg       0.99      0.99      0.99       300
```

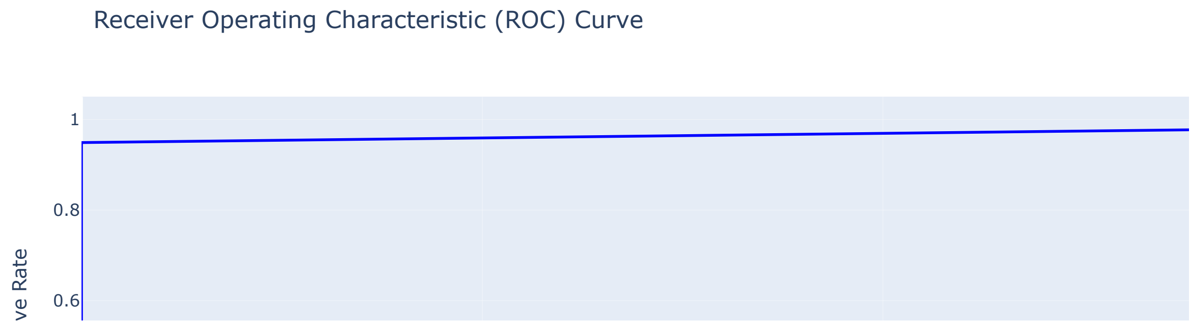
In [41]:

```
cm = confusion_matrix(y_test, y_pred)
fig = ff.create_annotated_heatmap(z=cm, x=list(range(len(cm))), y=list(range(len(cm))), colorscale='ylorbr')
fig.update_layout(title='Confusion Matrix', xaxis_title='Predicted', yaxis_title='Actual')
fig.show()
```



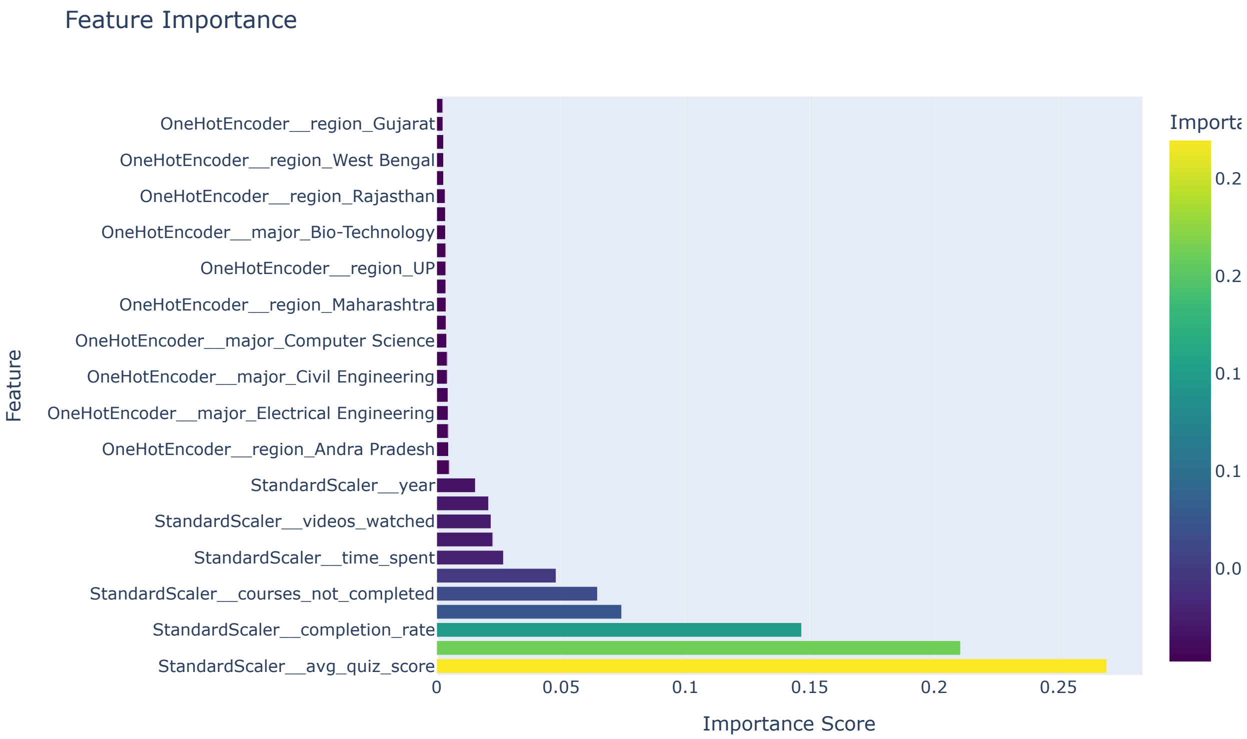
In [42]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc_forest = auc(fpr, tpr)
```

In [43]:
```
# Create a Plotly figure for the ROC curve
fig = go.Figure()

# Add the ROC curve trace
fig.add_trace(go.Scatter(x=fpr,
                         y=tpr,
                         mode='lines',
                         name='ROC Curve (AUC = {:.2f})'.format(roc_auc_forest),
                         line=dict(color='blue')))

# Add a diagonal line for random guessing
fig.add_trace(go.Scatter(x=[0, 1],
                         y=[0, 1],
                         mode='lines',
                         name='Random Guessing',
                         line=dict(color='red', dash='dash')))

# Update layout of the figure
fig.update_layout(title='Receiver Operating Characteristic (ROC) Curve',
                  xaxis_title='False Positive Rate',
                  yaxis_title='True Positive Rate',
                  xaxis=dict(range=[0.0, 1.0]),
                  yaxis=dict(range=[0.0, 1.05]),
                  showlegend=True)

# Show the figure
fig.show()
```



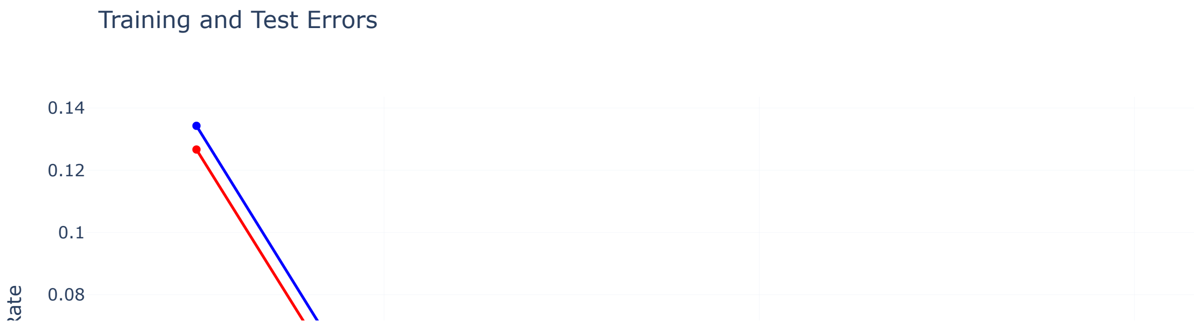
In [44]:

```
feature_importance = pd.Series(model_forest.feature_importances_, index=feature_names).sort_values(ascending=False)

# Create a DataFrame for plotting
importance_df = pd.DataFrame({
    'Feature': feature_importance.index,
    'Importance Score': feature_importance.values
})

# Create the Plotly bar chart with specified height and width
fig = px.bar(importance_df,
             x='Importance Score',
             y='Feature',
             orientation='h',
             title='Feature Importance',
             labels={'Importance Score': 'Importance Score', 'Feature': 'Feature'},
             color='Importance Score',
             color_continuous_scale='Viridis',
             height=600,
             width=1000)

fig.show()
```



In [45]:
```
#Train and Test Error
train_errors = []
test_errors = []

for i in range(1, 12):  # training for a number of iterations
    model_forest.fit(X_train[:i * int(len(X_train) / 10)], y_train[:i * int(len(y_train) / 10)]) # Incrementally fit on more

    # Calculate training accuracy
    train_pred = model_forest.predict(X_train)
    train_accuracy = accuracy_score(y_train, train_pred)
    train_errors.append(1 - train_accuracy) # Store error (1 - accuracy)

    # Calculate test accuracy
    test_pred = model_forest.predict(X_test)
    test_accuracy = accuracy_score(y_test, test_pred)
    test_errors.append(1 - test_accuracy) # Store error (1 - accuracy)
```

In [46]:
```
fig = go.Figure()

# Add training error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(train_errors) + 1)),
    y=train_errors,
    mode='lines+markers',
    name='Training Error',
    line=dict(color='blue')
))

# Add test error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(test_errors) + 1)),
    y=test_errors,
    mode='lines+markers',
    name='Test Error',
    line=dict(color='red')
))

# Update layout for better visualization
fig.update_layout(
    title='Training and Test Errors',
    xaxis_title='Iterations',
    yaxis_title='Error Rate',
    legend_title='Error Type',
    template='plotly_white',
)

fig.show()
```



Support Vector Machine

In [47]:
```
from sklearn.svm import SVC

svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train, y_train)
```

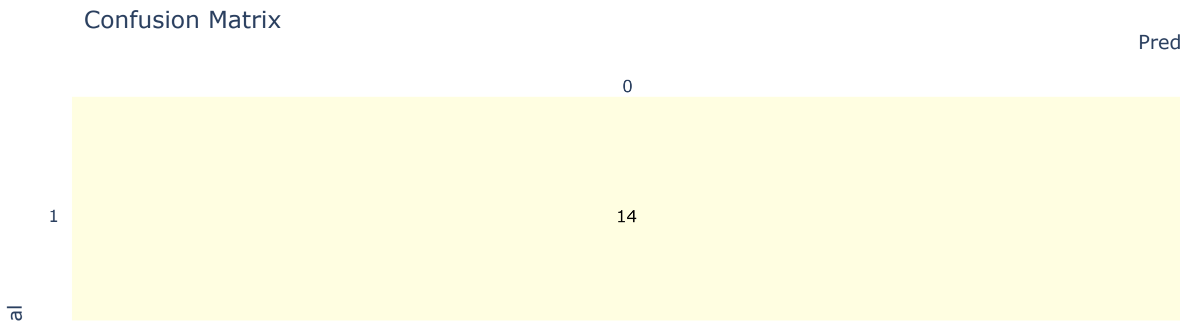
Out[47]:
```
SVC
SVC(random_state=42)
```

In [48]:
```
y_pred_svm = svm_model.predict(X_test)
```

In [49]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_svm)
print(f"Model Accuracy: {accuracy}")
print(classification_report(y_test, y_pred_svm))
```

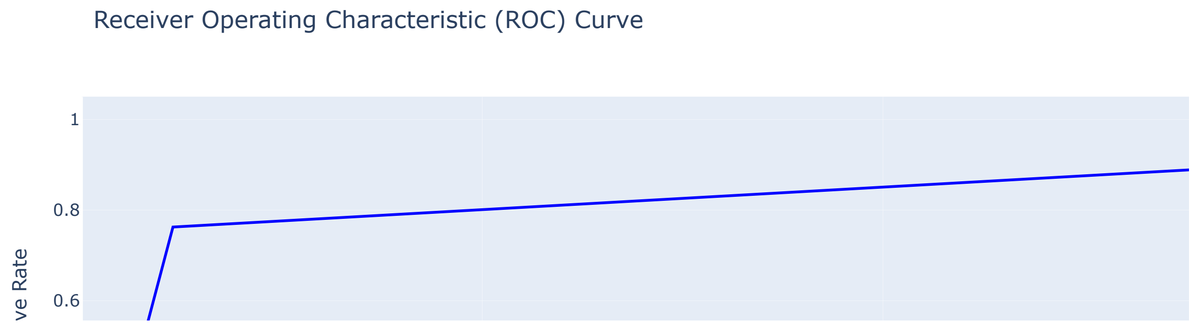
```
Model Accuracy: 0.9166666666666666
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       241
           1       0.80      0.76      0.78        59

    accuracy                           0.92       300
   macro avg       0.87      0.86      0.87       300
weighted avg       0.92      0.92      0.92       300
```

In [50]:
```
cm = confusion_matrix(y_test, y_pred_svm)
fig = ff.create_annotated_heatmap(z=cm, x=list(range(len(cm))), y=list(range(len(cm))), colorscale='ylorbr')
fig.update_layout(title='Confusion Matrix', xaxis_title='Predicted', yaxis_title='Actual')
fig.show()
```

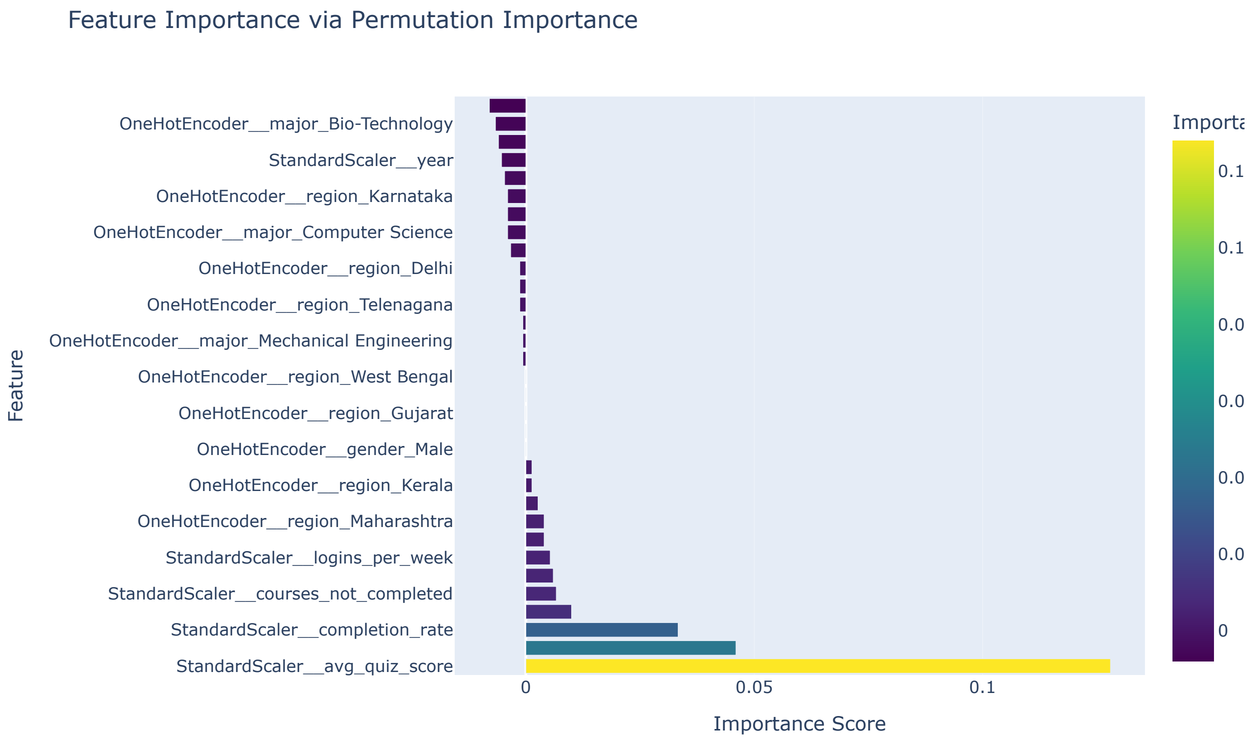


In [51]:

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_svm)
roc_auc_svm = auc(fpr, tpr)

fig = go.Figure()
# Add the ROC curve trace
fig.add_trace(go.Scatter(x=fpr,
                          y=tpr,
                          mode='lines',
                          name='ROC Curve (AUC = {:.2f})'.format(roc_auc_svm),
                          line=dict(color='blue')))

# Add a diagonal line for random guessing
fig.add_trace(go.Scatter(x=[0, 1],
                          y=[0, 1],
                          mode='lines',
                          name='Random Guessing',
                          line=dict(color='red', dash='dash')))

# Update layout of the figure
fig.update_layout(title='Receiver Operating Characteristic (ROC) Curve',
                  xaxis_title='False Positive Rate',
                  yaxis_title='True Positive Rate',
                  xaxis=dict(range=[0.0, 1.0]),
                  yaxis=dict(range=[0.0, 1.05]),
                  showlegend=True)

fig.show()
```

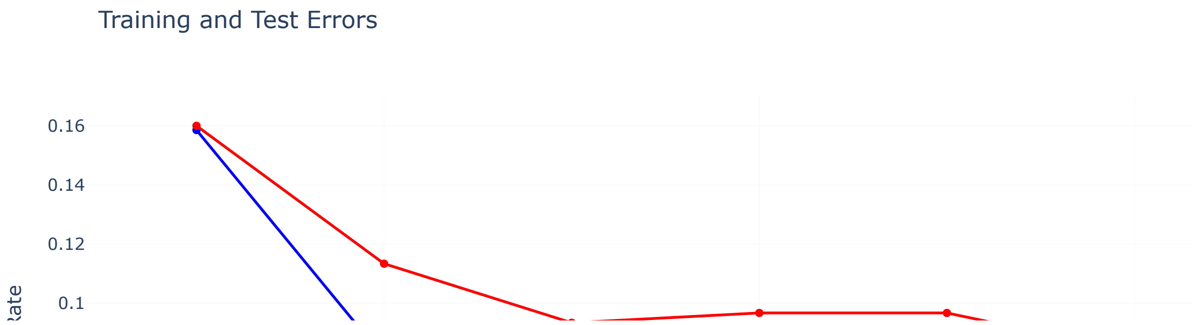


In [52]:
```
from sklearn.inspection import permutation_importance

# Calculate permutation importance
perm_importance = permutation_importance(svm_model, X_test, y_test)

# Create a DataFrame for plotting
importance_df = pd.DataFrame({
    'Feature': preprocessor.get_feature_names_out(),
    'Importance Score': perm_importance.importances_mean
})

# Create the Plotly bar chart with specified height and width
fig = px.bar(importance_df.sort_values(by='Importance Score', ascending=False),
             x='Importance Score',
             y='Feature',
             orientation='h',
             title='Feature Importance via Permutation Importance',
             labels={'Importance Score': 'Importance Score', 'Feature': 'Feature'},
             color='Importance Score',
             color_continuous_scale='Viridis',
             height=600,
             width=1000)

fig.show()
```



In [53]:

```
#Train and Test Error
train_errors = []
test_errors = []

for i in range(1, 12):  # training for a number of iterations
    svm_model.fit(X_train[:i * int(len(X_train) / 10)], y_train[:i * int(len(y_train) / 10)]) # Incrementally fit on more da

    # Calculate training accuracy
    train_pred = svm_model.predict(X_train)
    train_accuracy = accuracy_score(y_train, train_pred)
    train_errors.append(1 - train_accuracy) # Store error (1 - accuracy)

    # Calculate test accuracy
    test_pred = svm_model.predict(X_test)
    test_accuracy = accuracy_score(y_test, test_pred)
    test_errors.append(1 - test_accuracy) # Store error (1 - accuracy)

fig = go.Figure()

# Add training error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(train_errors) + 1)),
    y=train_errors,
    mode='lines+markers',
    name='Training Error',
    line=dict(color='blue')
))

# Add test error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(test_errors) + 1)),
    y=test_errors,
    mode='lines+markers',
    name='Test Error',
    line=dict(color='red')
))

# Update layout for better visualization
fig.update_layout(
    title='Training and Test Errors',
    xaxis_title='Iterations',
    yaxis_title='Error Rate',
    legend_title='Error Type',
    template='plotly_white',
)

fig.show()
```



XG boost Classifier

```
In [54]: import xgboost as xgb
```

```
In [55]: xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
         xgb_model.fit(X_train, y_train)
```

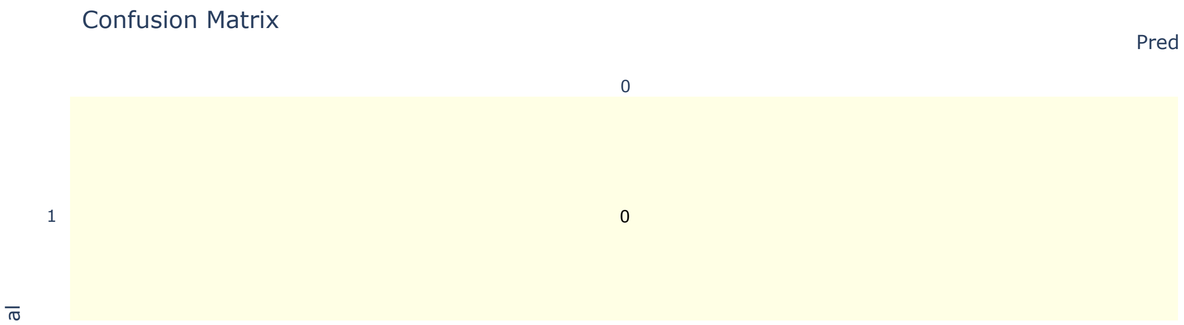
```
Out[55]: ▾                          XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
```

```
In [56]: y_pred_xgb = xgb_model.predict(X_test)
```

```
In [57]: # Evaluate the model
         accuracy = accuracy_score(y_test, y_pred_xgb)
         print(f"Model Accuracy: {accuracy}")
         print(classification_report(y_test, y_pred_xgb))
```

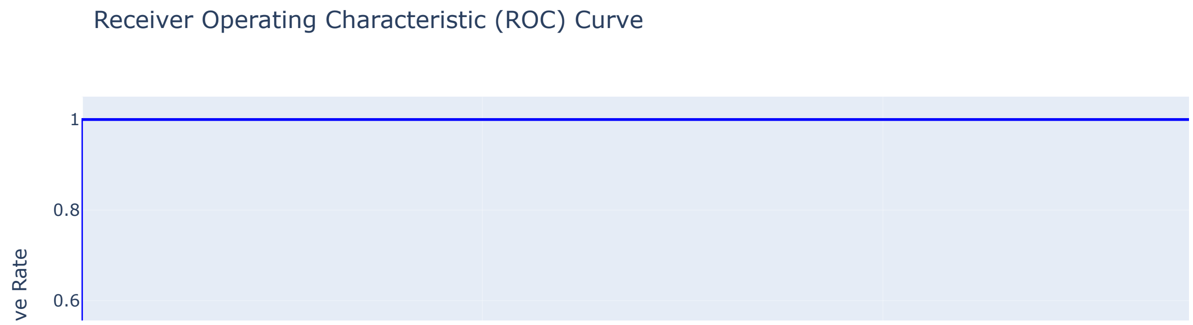
```
Model Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       241
           1       1.00      1.00      1.00        59

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

```
In [58]: cm = confusion_matrix(y_test, y_pred_xgb)
         fig = ff.create_annotated_heatmap(z=cm, x=list(range(len(cm))), y=list(range(len(cm))), colorscale='ylorbr')
         fig.update_layout(title='Confusion Matrix', xaxis_title='Predicted', yaxis_title='Actual')
         fig.show()
```

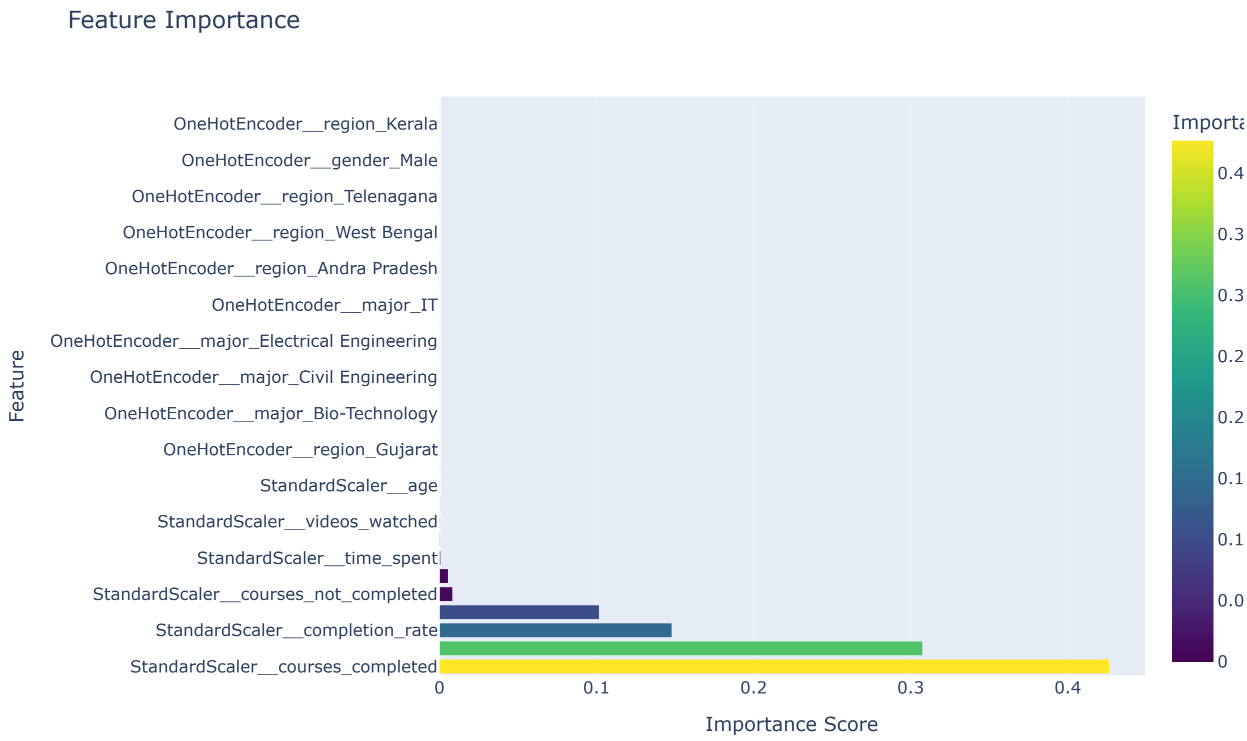


In [59]:

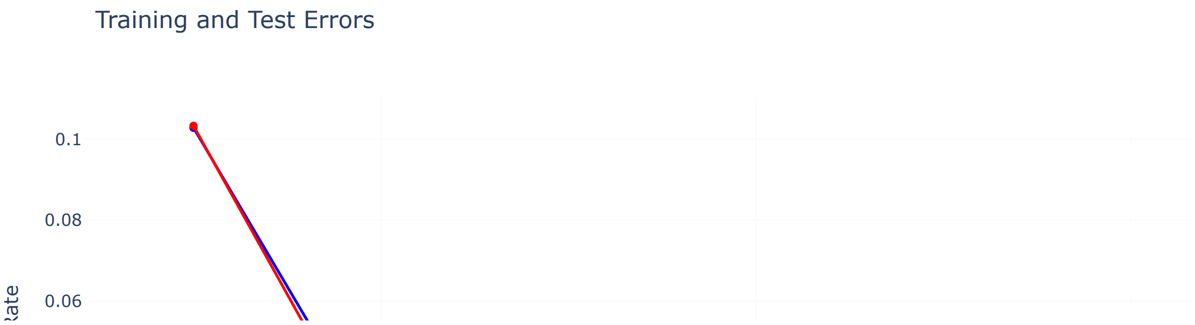
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_xgb)
roc_auc_xgb = auc(fpr, tpr)

fig = go.Figure()
# Add the ROC curve trace
fig.add_trace(go.Scatter(x=fpr,
                          y=tpr,
                          mode='lines',
                          name='ROC Curve (AUC = {:.2f})'.format(roc_auc_xgb),
                          line=dict(color='blue')))

# Add a diagonal line for random guessing
fig.add_trace(go.Scatter(x=[0, 1],
                          y=[0, 1],
                          mode='lines',
                          name='Random Guessing',
                          line=dict(color='red', dash='dash')))

# Update layout of the figure
fig.update_layout(title='Receiver Operating Characteristic (ROC) Curve',
                  xaxis_title='False Positive Rate',
                  yaxis_title='True Positive Rate',
                  xaxis=dict(range=[0.0, 1.0]),
                  yaxis=dict(range=[0.0, 1.05]),
                  showlegend=True)

fig.show()
```



In [60]:
```
feature_importance = pd.Series(xgb_model.feature_importances_, index=feature_names).sort_values(ascending=False)

# Create a DataFrame for plotting
importance_df = pd.DataFrame({
    'Feature': feature_importance.index,
    'Importance Score': feature_importance.values
})

# Create the Plotly bar chart with specified height and width
fig = px.bar(importance_df,
             x='Importance Score',
             y='Feature',
             orientation='h',
             title='Feature Importance',
             labels={'Importance Score': 'Importance Score', 'Feature': 'Feature'},
             color='Importance Score',
             color_continuous_scale='Viridis',
             height=600,
             width=1000)

fig.show()
```


Feature Importance

In [61]:
```
#Train and Test Error
train_errors = []
test_errors = []

for i in range(1, 12):  # training for a number of iterations
    xgb_model.fit(X_train[:i * int(len(X_train) / 10)], y_train[:i * int(len(y_train) / 10)]) # Incrementally fit on more da

    # Calculate training accuracy
    train_pred = xgb_model.predict(X_train)
    train_accuracy = accuracy_score(y_train, train_pred)
    train_errors.append(1 - train_accuracy) # Store error (1 - accuracy)

    # Calculate test accuracy
    test_pred = xgb_model.predict(X_test)
    test_accuracy = accuracy_score(y_test, test_pred)
    test_errors.append(1 - test_accuracy) # Store error (1 - accuracy)

fig = go.Figure()

# Add training error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(train_errors) + 1)),
    y=train_errors,
    mode='lines+markers',
    name='Training Error',
    line=dict(color='blue')
))

# Add test error trace
fig.add_trace(go.Scatter(
    x=list(range(1, len(test_errors) + 1)),
    y=test_errors,
    mode='lines+markers',
    name='Test Error',
    line=dict(color='red')
))

# Update layout for better visualization
fig.update_layout(
    title='Training and Test Errors',
    xaxis_title='Iterations',
    yaxis_title='Error Rate',
    legend_title='Error Type',
    template='plotly_white',
)

fig.show()
```



In [62]:

```
data.head()
```

Out[62]:

	student_id	age	gender	major	year	region	logins_per_week	videos_watched	time_spent	avg_quiz_score	courses_started
0	1	19	Female	IT	4	Rajasthan	9	10	8	94	4
1	2	18	Female	Bio-Technology	1	Telenagana	2	6	10	98	5
2	3	22	Female	Bio-Technology	3	Gujarat	2	14	5	71	7
3	4	21	Male	Civil Engineering	3	Tamil Nadu	4	5	6	59	7
4	5	21	Female	Electrical Engineering	2	Karnataka	9	14	5	98	7

Correlation among features

In [63]:

```
df['region'].unique()
```

Out[63]:

```
array(['Rajasthan', 'Telenagana', 'Gujarat', 'Tamil Nadu', 'Karnataka',
       'Andra Pradesh', 'Kerala', 'Delhi', 'UP', 'Maharashtra',
       'West Bengal'], dtype=object)
```

In [64]:

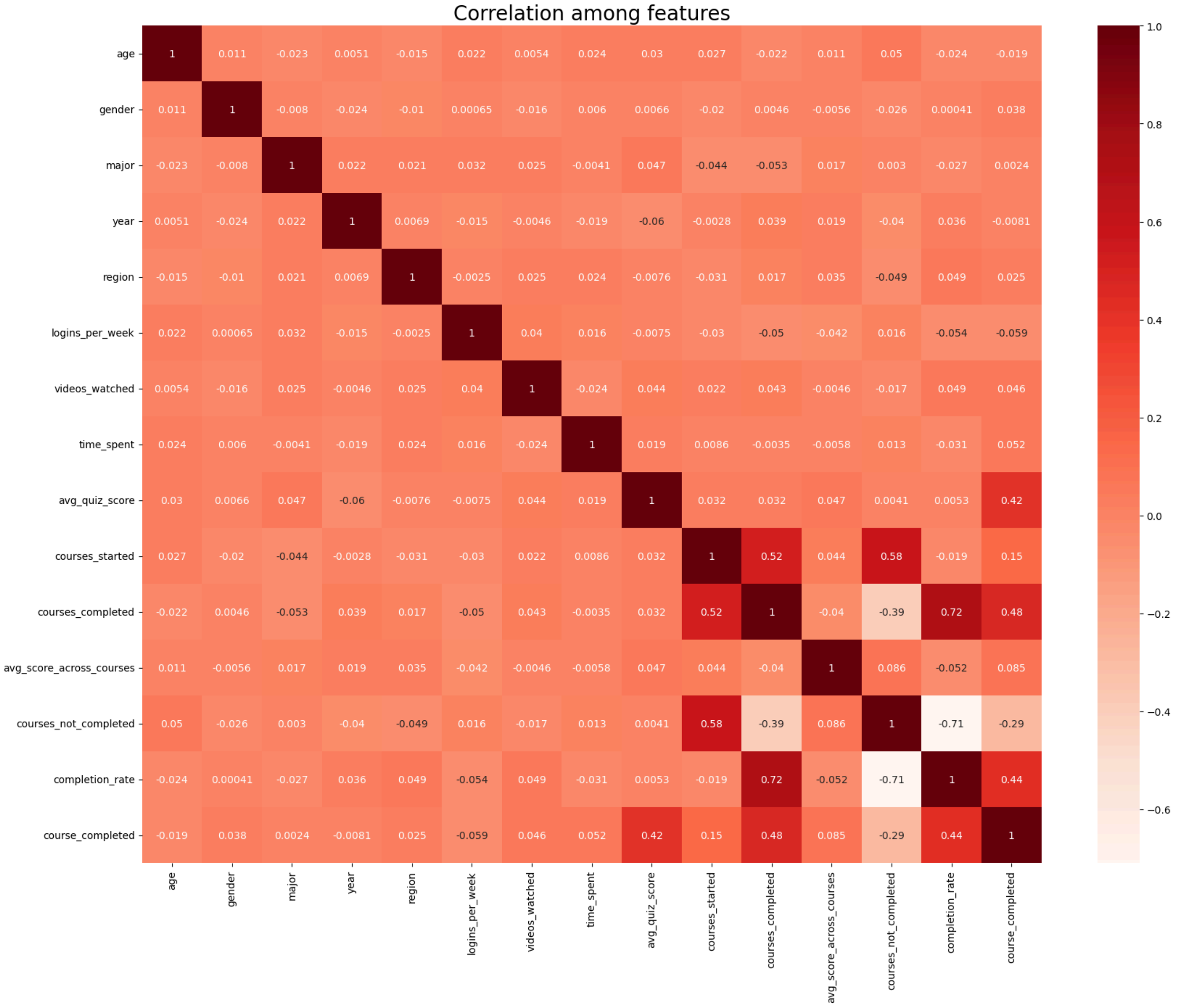
```
#map age, gender, major and region to numeric data

def numerical_data():
    data['gender'] = data['gender'].map({'Female': 0,
                                         'Male': 1})

    data['major'] = data['major'].map({'IT':0, 'Bio-Technology':1,
                                       'Civil Engineering':2,
                                       'Electrical Engineering':3,
                                       'Chemical Engineering':4,
                                       'Computer Science':4,
                                       'Mechanical Engineering':6,
                                       'Environmental Science':8})

    data['region'] = data['region'].map({'Rajasthan':0,
                                         'Telenagana':1,
                                         'Gujarat':2,
                                         'Tamil Nadu':3,
                                         'Karnataka':4,
                                         'Andra Pradesh':5,
                                         'Kerala':6,
                                         'Delhi':7,
                                         'UP':8,
                                         'Maharashtra':9,
                                         'West Bengal':10 })
```

In [65]:
```
numerical_data()
data.head()
```

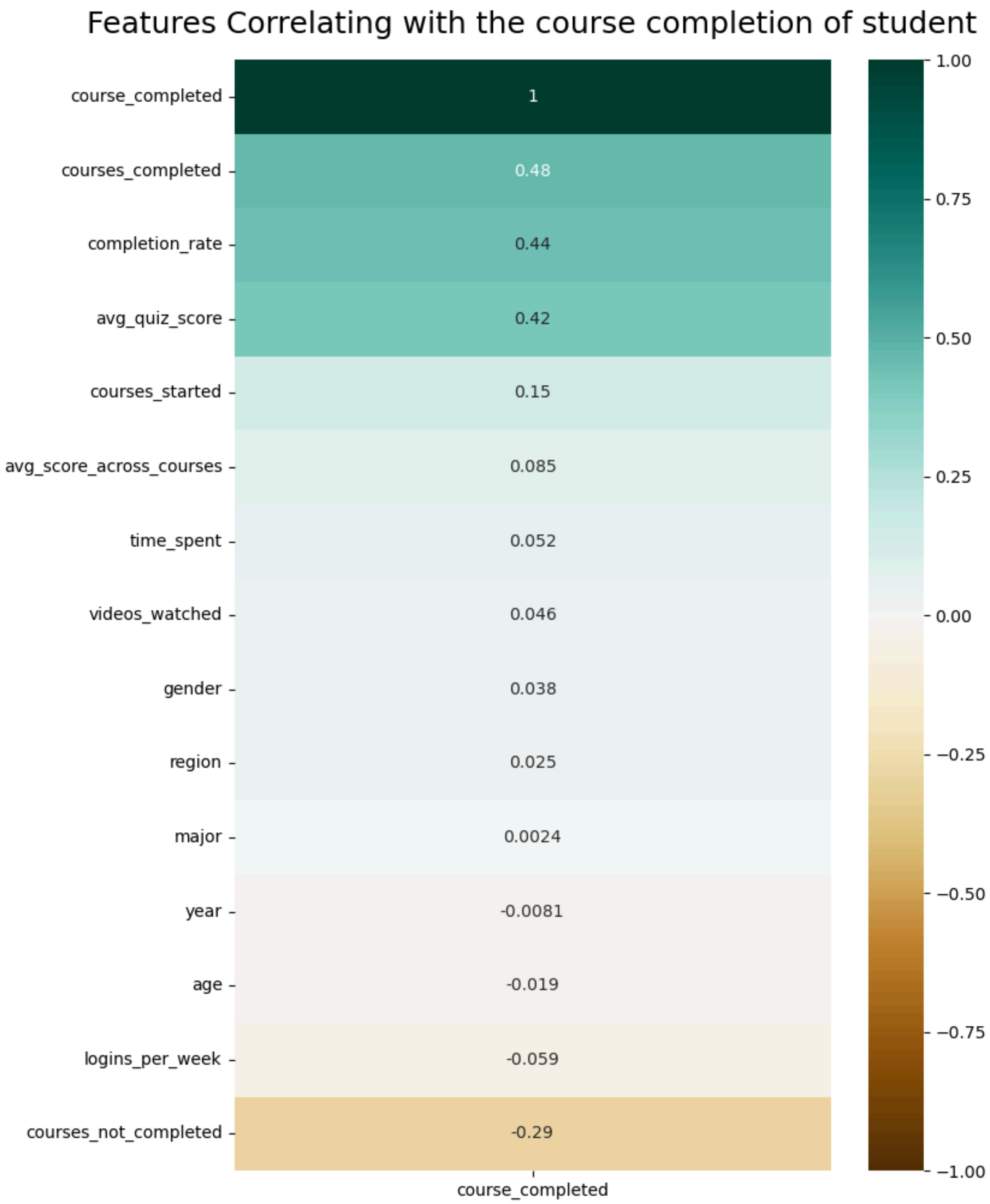
Out[65]:
```
   student_id  age  gender  major  year  region  logins_per_week  videos_watched  time_spent  avg_quiz_score  courses_started  courses
0           1   19       0      0     4       0                9              10           8              94                4
1           2   18       0      1     1       1                2               6          10              98                5
2           3   22       0      1     3       2                2              14           5              71                7
3           4   21       1      2     3       3                4               5           6              59                7
4           5   21       0      3     2       4                9              14           5              98                7
```

In [66]:
```
data = data.drop('student_id', axis=1)
```

In [67]:
```
corr = data.corr()
plt.figure(figsize=(20,15))
sns.heatmap(corr, annot=True, cmap="Reds")
plt.title('Correlation among features', fontsize=20)
plt.show()
```



In [68]:
```
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(data.corr()[['course_completed']].sort_values(by='course_completed', ascending=False), vmin=-1, vmax=1
heatmap.set_title('Features Correlating with the course completion of student', fontdict={'fontsize':18}, pad=16);
plt.show()
```



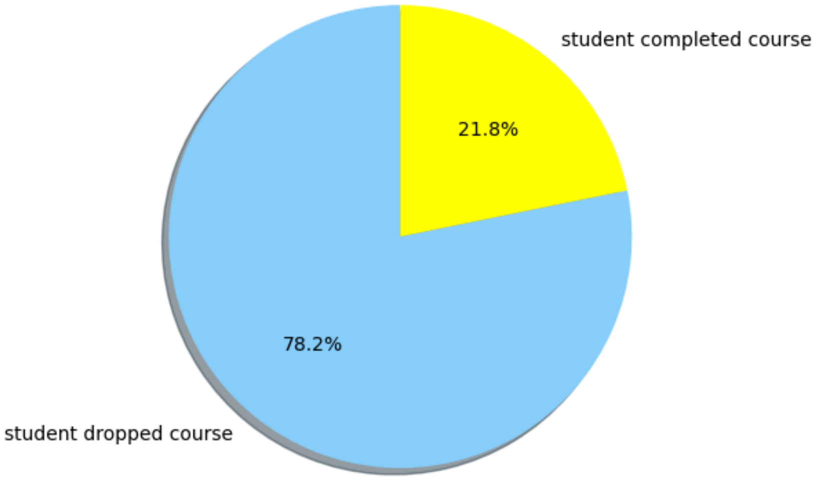
```
In [69]: data.columns
```

```
Out[69]: Index(['age', 'gender', 'major', 'year', 'region', 'logins_per_week',
                'videos_watched', 'time_spent', 'avg_quiz_score', 'courses_started',
                'courses_completed', 'avg_score_across_courses',
                'courses_not_completed', 'completion_rate', 'course_completed'],
               dtype='object')
```

```
In [70]: features = ['age', 'gender', 'major', 'year', 'region', 'logins_per_week',
                'videos_watched', 'time_spent', 'avg_quiz_score', 'courses_started',
                'courses_completed', 'avg_score_across_courses',
                'courses_not_completed', 'completion_rate', 'course_completed']
```

Feature Visualization

```
In [71]: labels = 'student dropped course ', 'student completed course'
sizes = [782, 218]
colors=['lightskyblue','yellow']
fig1, ax1 = plt.subplots()
ax1.pie(sizes,  labels=labels, autopct='%1.1f%%',colors=colors,
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



In [73]:

```python
from sklearn.tree import export_graphviz
import graphviz

tree = model_forest.estimators_[0]  # You can change the index to visualize other trees

# Export the tree in DOT format using export_graphviz
export_graphviz(
    tree,
    out_file='tree.dot',
    feature_names=feature_names,
    class_names=['Dropout', 'Complete'],
    rounded=True,
    filled=True,
    impurity=True,
    proportion=True
)

# Read the generated DOT file and create a Graphviz object
with open('tree.dot') as f:
    dot_graph = f.read()

# Visualize the tree using Graphviz
graph = graphviz.Source(dot_graph)
graph.render("random_forest_tree")  # This will save the visualization as a PDF file
graph.view()
```

Out[73]:

```
'random_forest_tree.pdf'
```