In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv(r'C:\Users\Ritik\Downloads\MBA.csv')
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6194 entries, 0 to 6193
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   application_id  6194 non-null   int64  
 1   gender          6194 non-null   object 
 2   international   6194 non-null   bool   
 3   gpa             6194 non-null   float64
 4   major           6194 non-null   object 
 5   race            4352 non-null   object 
 6   gmat            6194 non-null   int64  
 7   work_exp        6194 non-null   int64  
 8   work_industry   6194 non-null   object 
 9   admission       1000 non-null   object 
dtypes: bool(1), float64(1), int64(3), object(5)
memory usage: 441.7+ KB
```

In [4]:
```
df.describe()
```

Out[4]:

	application_id	gpa	gmat	work_exp
count	6194.000000	6194.000000	6194.000000	6194.000000
mean	3097.500000	3.250714	651.092993	5.016952
std	1788.198115	0.151541	49.294883	1.032432
min	1.000000	2.650000	570.000000	1.000000
25%	1549.250000	3.150000	610.000000	4.000000
50%	3097.500000	3.250000	650.000000	5.000000
75%	4645.750000	3.350000	680.000000	6.000000
max	6194.000000	3.770000	780.000000	9.000000

In [5]:

```
df.head()
```

Out[5]:

	application_id	gender	international	gpa	major	race	gmat	work_exp	work_industry	admission
0	1	Female	False	3.30	Business	Asian	620	3	Financial Services	Admit
1	2	Male	False	3.28	Humanities	Black	680	5	Investment Management	NaN
2	3	Female	True	3.30	Business	NaN	710	5	Technology	Admit
3	4	Male	False	3.47	STEM	Black	690	6	Technology	NaN
4	5	Male	False	3.35	STEM	Hispanic	590	5	Consulting	NaN

In [6]:

```
df.shape
```

Out[6]: (6194, 10)

In [7]:

```
df.isna()
```

Out[7]:

	application_id	gender	international	gpa	major	race	gmat	work_exp	work_industry	admission
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	True
2	False	False	False	False	False	True	False	False	False	False
3	False	False	False	False	False	False	False	False	False	True
4	False	False	False	False	False	False	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...
6189	False	False	False	False	False	False	False	False	False	True
6190	False	False	False	False	False	False	False	False	False	True
6191	False	False	False	False	False	True	False	False	False	False
6192	False	False	False	False	False	True	False	False	False	True
6193	False	False	False	False	False	False	False	False	False	True

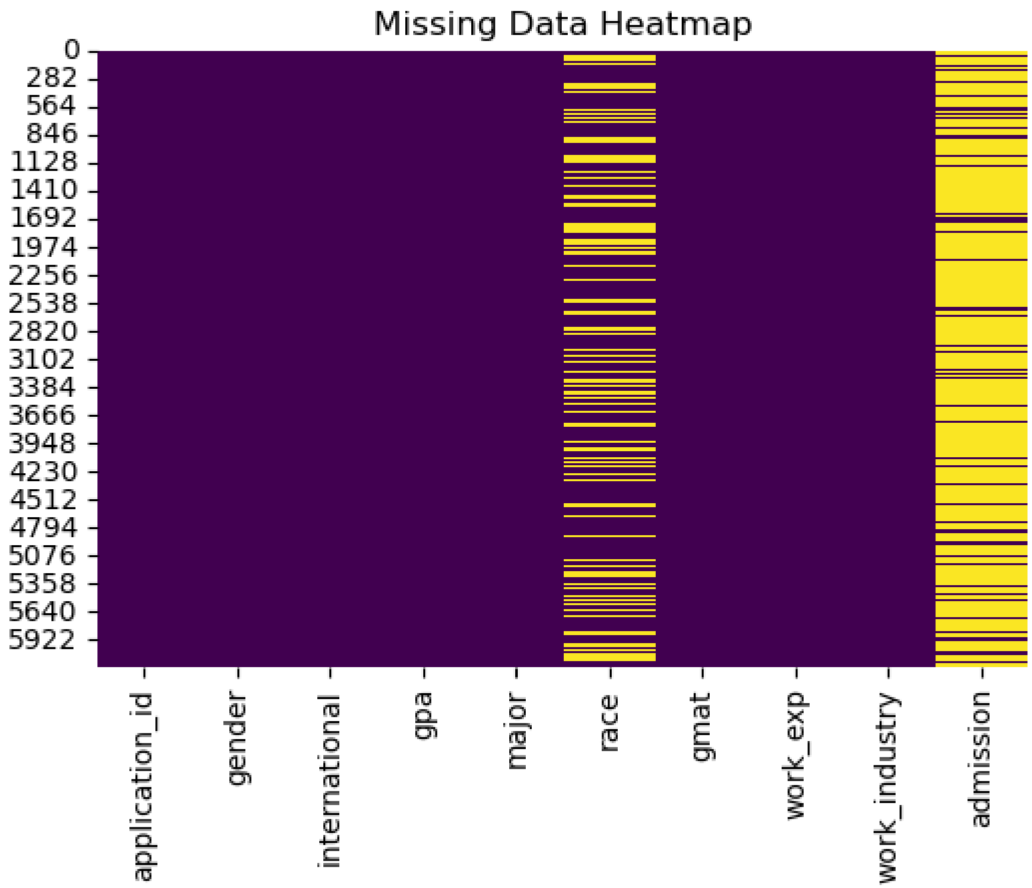
6194 rows × 10 columns

In [8]:
```
df.isna().sum()
```

Out[8]:
```
application_id       0
gender               0
international        0
gpa                  0
major                0
race              1842
gmat                 0
work_exp             0
work_industry        0
admission         5194
dtype: int64
```

In [9]:
```
# Missing Data Heatmap
plt.figure(figsize=(6, 4))
```

```
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Data Heatmap")
plt.show()
```



In [10]:

```
# Initialize an empty dictionary to store results
distinct_categories = {}

# Iterate through each column
for column in df.columns:
    unique_values = df[column].unique()  # Get unique values
    distinct_categories[column] = unique_values

# Display all distinct values for each column
```

```
for column, values in distinct_categories.items():
    print(f"Column: {column}")
    print(f"Distinct Values: {values}")
    print("-" * 50)
```

```
Column: application_id
Distinct Values: [   1    2    3 ... 6192 6193 6194]
--------------------------------------------------
Column: gender
Distinct Values: ['Female' 'Male']
--------------------------------------------------
Column: international
Distinct Values: [False  True]
--------------------------------------------------
Column: gpa
Distinct Values: [3.3  3.28 3.47 3.35 3.18 2.93 3.02 3.24 3.27 3.05 2.85 3.39 3.03 3.32
 3.23 3.13 3.09 3.46 3.64 3.42 3.4  3.26 2.99 3.08 3.65 3.04 3.19 3.33
 3.53 3.5  3.22 3.16 3.45 3.12 3.41 3.38 3.43 2.96 3.44 3.01 3.   3.36
 3.31 3.07 3.49 3.34 2.89 3.2  3.17 3.1  3.52 3.15 3.21 3.48 3.14 2.97
 3.11 3.29 3.25 3.51 3.06 2.95 3.37 3.55 3.54 3.6  3.61 3.71 3.77 3.58
 2.98 3.56 3.69 2.79 2.87 2.88 3.63 2.9  3.74 2.91 2.92 2.78 3.57 3.66
 2.81 3.59 2.82 3.62 2.73 3.68 2.84 2.83 2.86 3.67 2.94 2.72 2.8  3.76
 3.7  3.73 2.65]
--------------------------------------------------
Column: major
Distinct Values: ['Business' 'Humanities' 'STEM']
--------------------------------------------------
Column: race
Distinct Values: ['Asian' 'Black' nan 'Hispanic' 'White' 'Other']
--------------------------------------------------
Column: gmat
Distinct Values: [620 680 710 690 590 610 630 580 640 600 700 670 760 730 570 650 720 740
 660 780 750 770]
--------------------------------------------------
Column: work_exp
Distinct Values: [3 5 6 2 4 8 7 9 1]
--------------------------------------------------
Column: work_industry
Distinct Values: ['Financial Services' 'Investment Management' 'Technology' 'Consulting'
 'Nonprofit/Gov' 'PE/VC' 'Health Care' 'Investment Banking' 'Other'
 'Retail' 'Energy' 'CPG' 'Real Estate' 'Media/Entertainment']
--------------------------------------------------
Column: admission
Distinct Values: ['Admit' nan 'Waitlist']
--------------------------------------------------
```

In [11]:
```
distinct_gender = df['gender'].value_counts(dropna=False)
distinct_gender
```

Out[11]:
```
gender
Male      3943
Female    2251
Name: count, dtype: int64
```

In [12]:
```
distinct_international = df['international'].value_counts(dropna=False)
distinct_international
```

Out[12]:
```
international
False    4352
True     1842
Name: count, dtype: int64
```

In [13]:
```
distinct_major = df['major'].value_counts(dropna=False)
distinct_major
```

Out[13]:
```
major
Humanities    2481
STEM          1875
Business      1838
Name: count, dtype: int64
```

In [14]:
```
distinct_race = df['race'].value_counts(dropna=False)
distinct_race
```

Out[14]:
```
race
NaN         1842
White       1456
Asian       1147
Black        916
Hispanic     596
Other        237
Name: count, dtype: int64
```

In [15]:
```
distinct_gmat = df['gmat'].value_counts(dropna=False)
distinct_gmat
```

Out[15]:
```
gmat
660    483
670    454
650    451
640    444
620    439
570    422
630    417
680    399
610    381
690    329
600    313
700    280
590    260
710    251
580    212
720    193
730    125
740    107
750     78
780     65
760     60
770     31
Name: count, dtype: int64
```

In [16]:
```
distinct_work_exp = df['work_exp'].value_counts(dropna=False)
distinct_work_exp
```

Out[16]:
```
work_exp
5    2419
6    1528
4    1437
3     369
7     367
8      38
2      32
9       2
1       2
Name: count, dtype: int64
```

In [17]:
```
distinct_work_industry = df['work_industry'].value_counts(dropna=False)
distinct_work_industry
```

Out[17]:
```
work_industry
Consulting               1619
PE/VC                     907
Technology                716
Nonprofit/Gov             651
Investment Banking        580
Financial Services        451
Other                     421
Health Care               334
Investment Management     166
CPG                       114
Real Estate               111
Media/Entertainment        59
Retail                     33
Energy                     32
Name: count, dtype: int64
```

In [18]:
```
distinct_admission = df['admission'].value_counts(dropna=False)
distinct_admission
```

Out[18]:
```
admission
NaN         5194
Admit        900
Waitlist     100
Name: count, dtype: int64
```

Replacing NAN values with respective Values according to metadata

In [19]:
```
# Fill missing values in admission
df['admission'] = df['admission'].fillna('Deny')

# Fill missing values in race
df['race'] = df['race'].fillna('International')
```

In [20]:
```
distinct_admission1 = df['admission'].value_counts(dropna=False)
distinct_admission1
```

Out[20]:
```
admission
Deny        5194
Admit        900
Waitlist     100
Name: count, dtype: int64
```

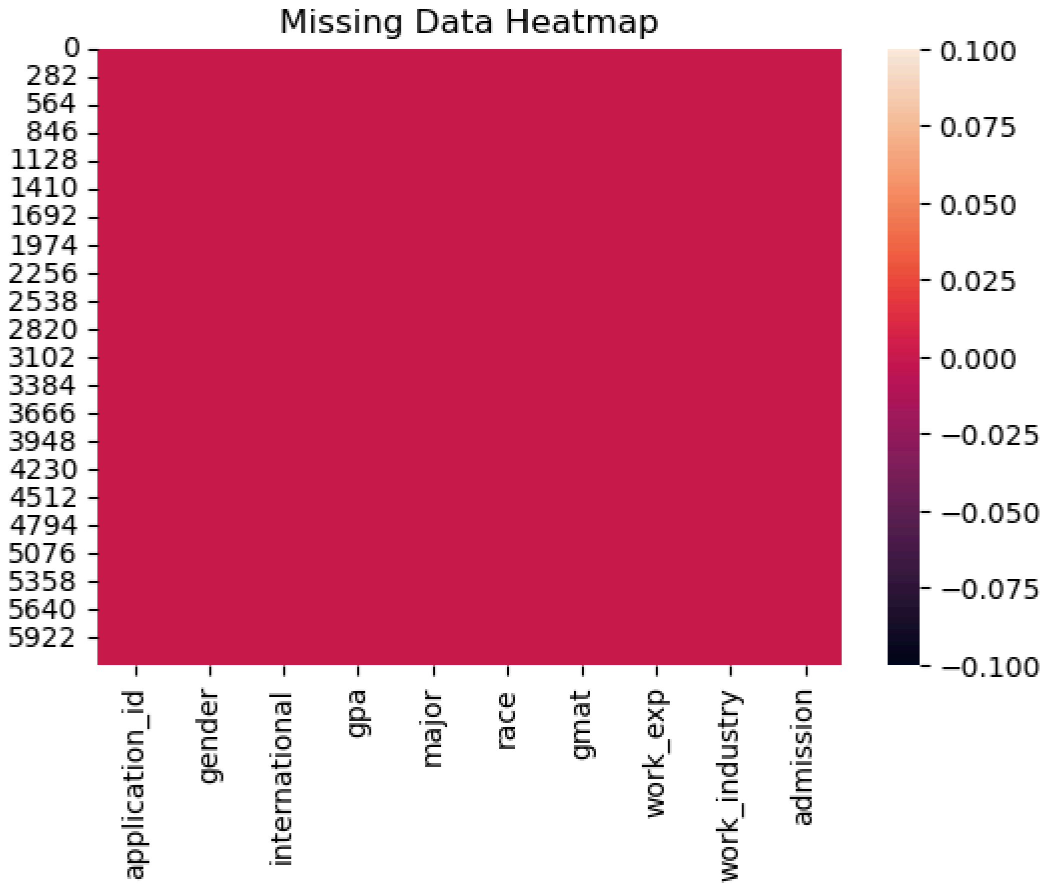
In [21]:
```
distinct_race1 = df['race'].value_counts(dropna=False)
distinct_race1
```

Out[21]:
```
race
International    1842
White            1456
Asian            1147
Black             916
Hispanic          596
Other             237
Name: count, dtype: int64
```

In [22]:
```
df.isna().sum()
```

Out[22]:
```
application_id    0
gender            0
international     0
gpa               0
major             0
race              0
gmat              0
work_exp          0
work_industry     0
admission         0
dtype: int64
```

In [23]:
```
# Missing Data Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(df.isnull())
plt.title("Missing Data Heatmap")
plt.show()
```



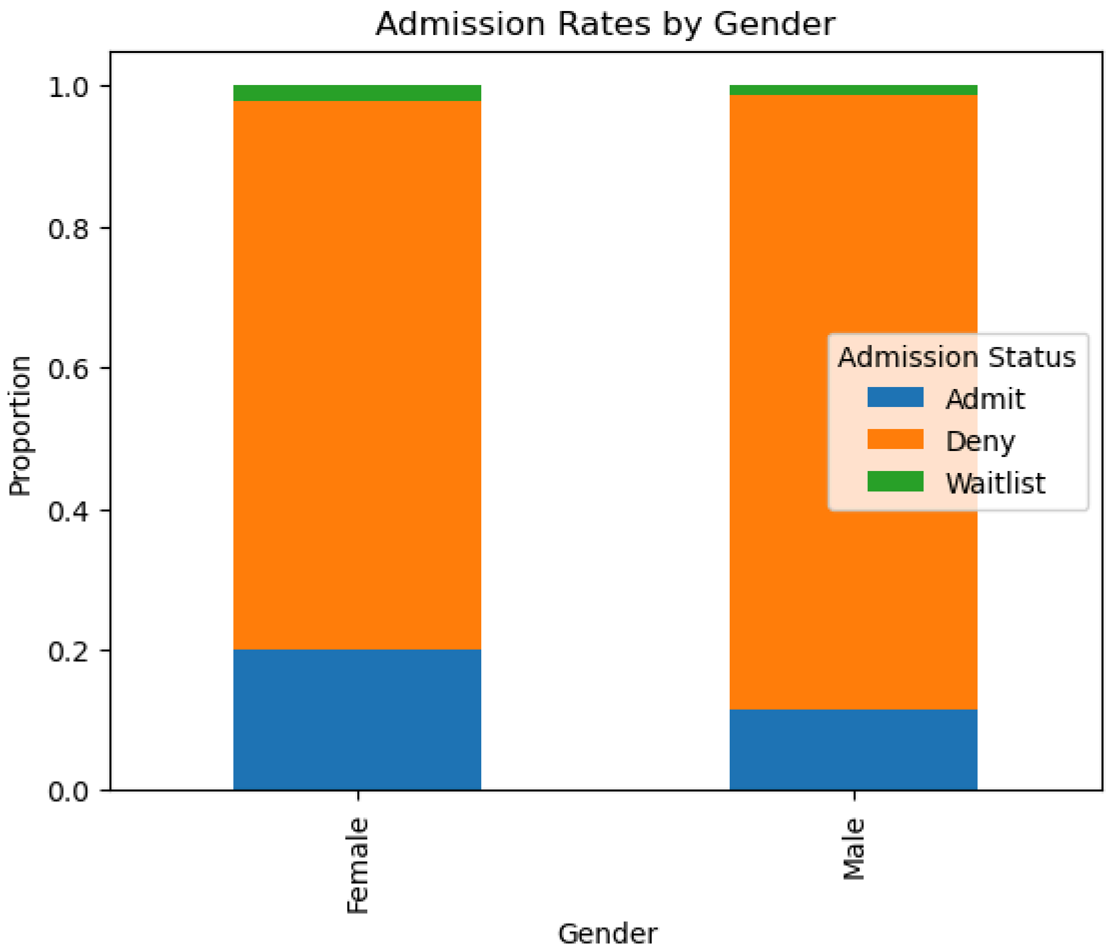
In [24]:
```
# Save the Clean DataFrame to a CSV file
df.to_csv('clean_data.csv', index=False)
print("DataFrame saved as 'processed_data.csv'")
```

```
DataFrame saved as 'processed_data.csv'
```

EDA

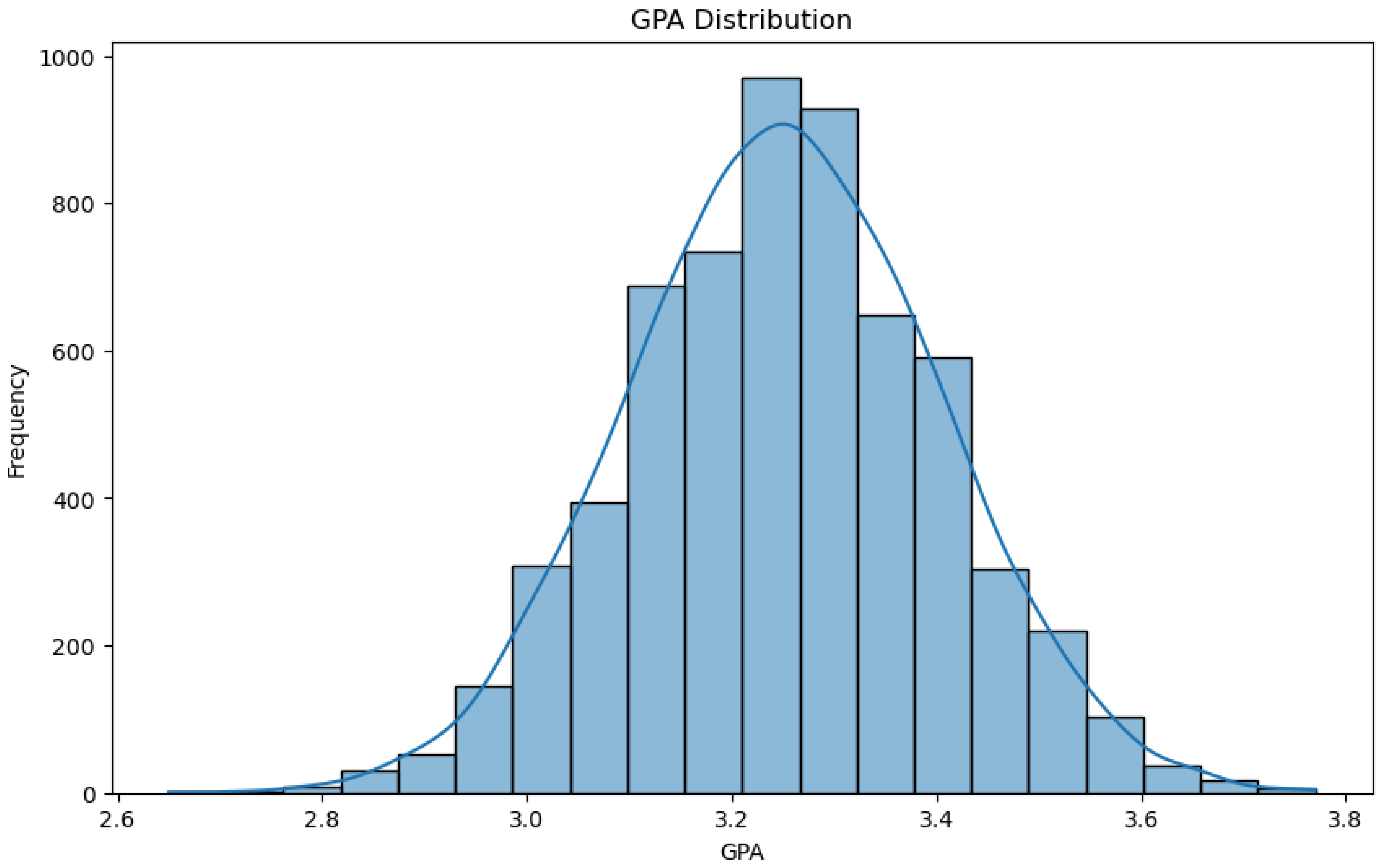
In [25]:
```
# 1. Admission Rates by Gender
admission_by_gender = df.groupby('gender')['admission'].value_counts(normalize=True).unstack()
admission_by_gender.plot(kind='bar', stacked=True)
```

```
plt.title('Admission Rates by Gender')
plt.xlabel('Gender')
plt.ylabel('Proportion')
plt.legend(title='Admission Status')
plt.show()
```



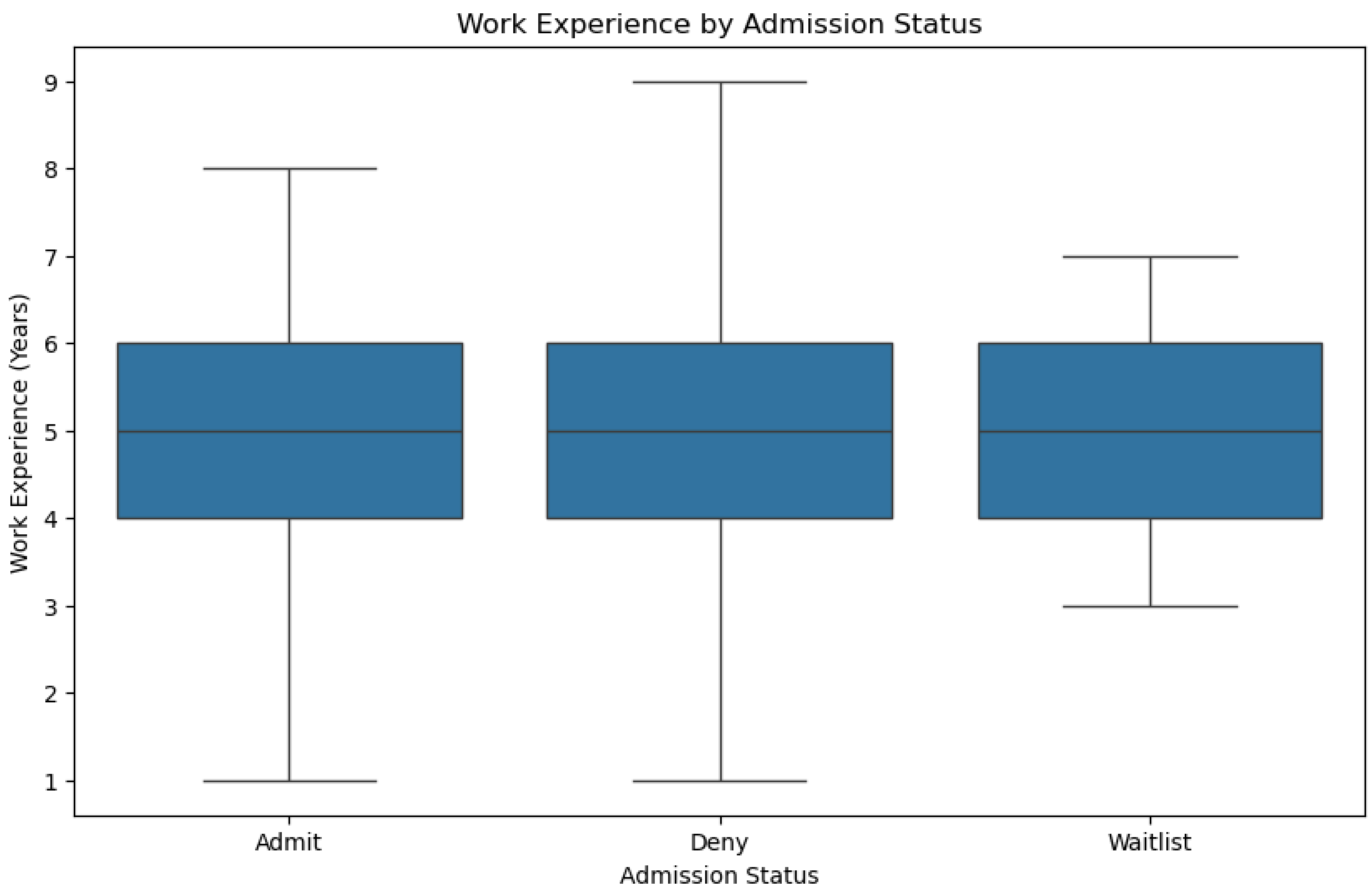
In [26]:
```
# 2. GPA Distribution
plt.figure(figsize=(10, 6))
sns.histplot(df['gpa'], bins=20, kde=True)
plt.title('GPA Distribution')
plt.xlabel('GPA')
```

```
plt.ylabel('Frequency')
plt.show()
```



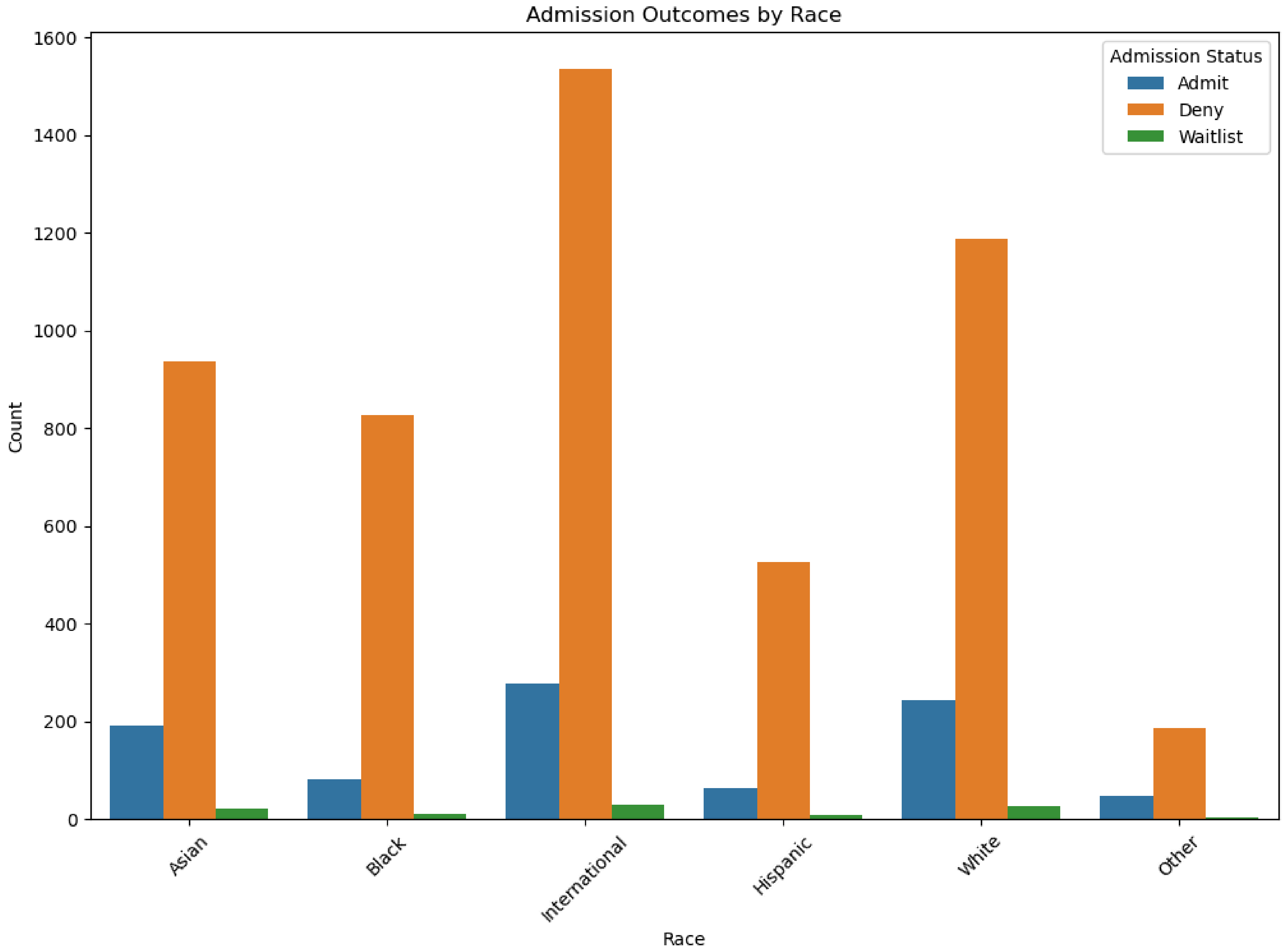
In [27]:
```
# 3. Work Experience vs Admission Status
plt.figure(figsize=(10, 6))
sns.boxplot(x='admission', y='work_exp', data=df)
plt.title('Work Experience by Admission Status')
plt.xlabel('Admission Status')
```

```
plt.ylabel('Work Experience (Years)')
plt.show()
```

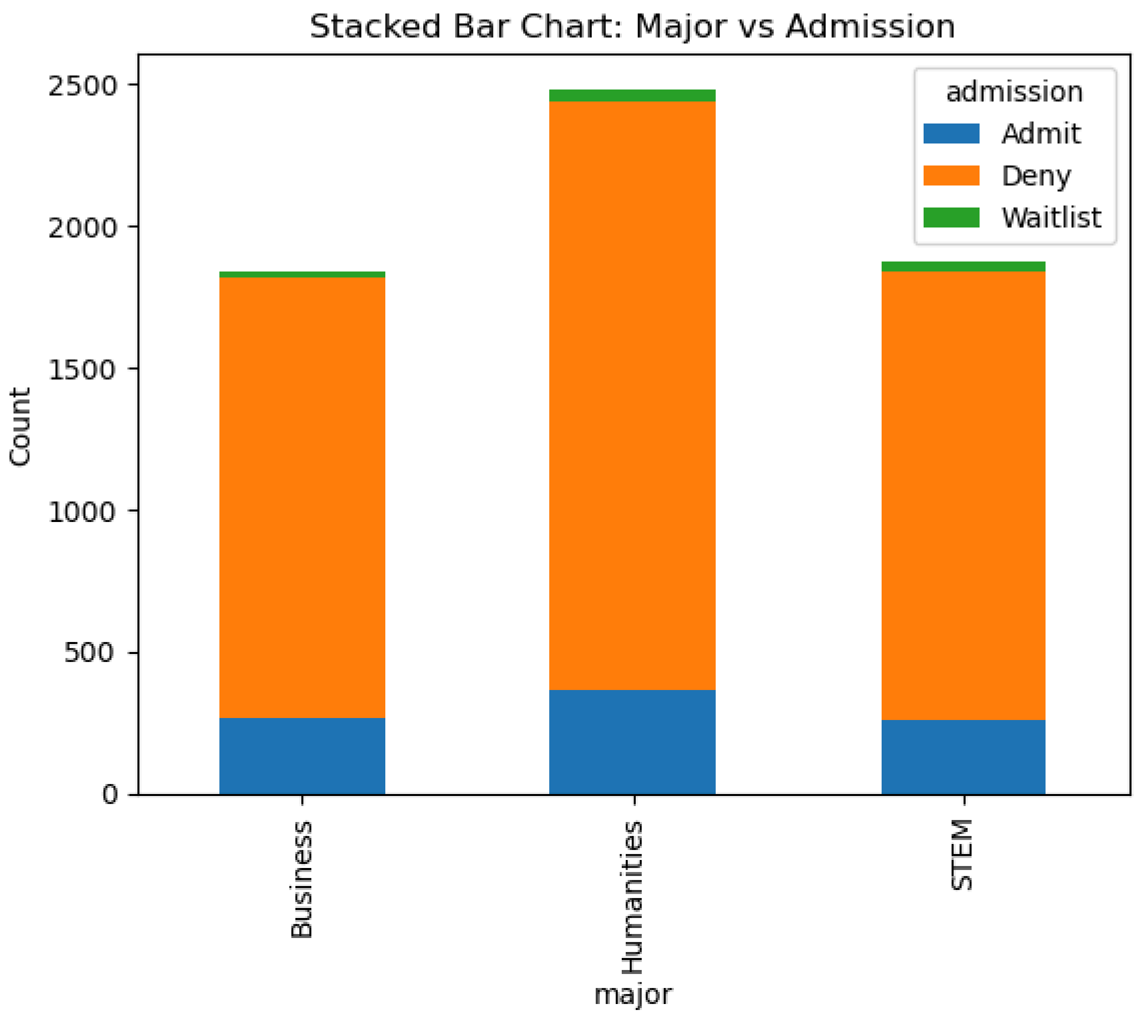


In [28]:
```
# 4. Race and Admission Outcomes
plt.figure(figsize=(12, 8))
sns.countplot(data=df, x='race', hue='admission')
plt.title('Admission Outcomes by Race')
plt.xlabel('Race')
```

```
plt.ylabel('Count')
plt.legend(title='Admission Status')
plt.xticks(rotation=45)
plt.show()
```

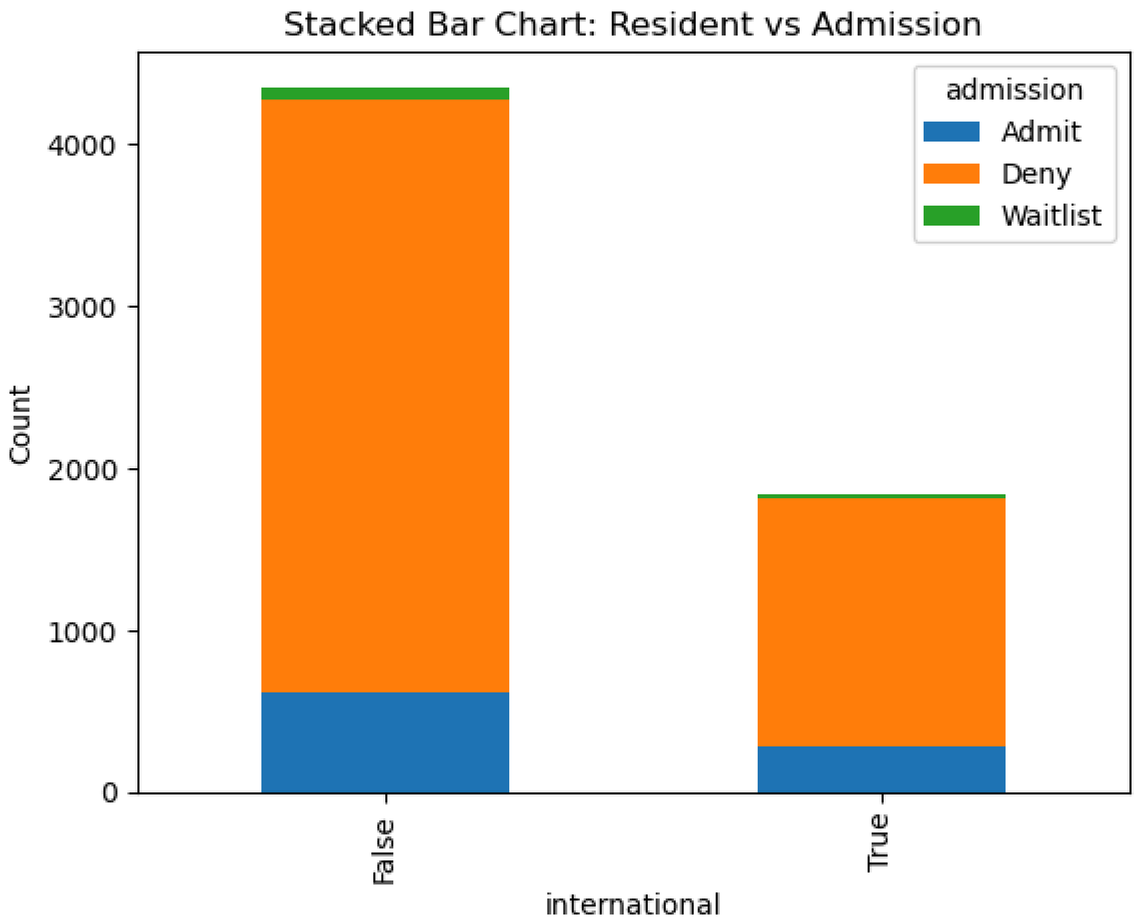

Admission Outcomes by Race
Admission Status
Admit
Deny
Waitlist
Count
0
200
400
600
800
1000
1200
1400
1600
Asian
Black
International
Hispanic
White
Other
Race

In [29]:
```
pd.crosstab(df['major'], df['admission']).plot(kind='bar', stacked=True)
plt.ylabel('Count')
plt.title('Stacked Bar Chart: Major vs Admission')
plt.show()
```

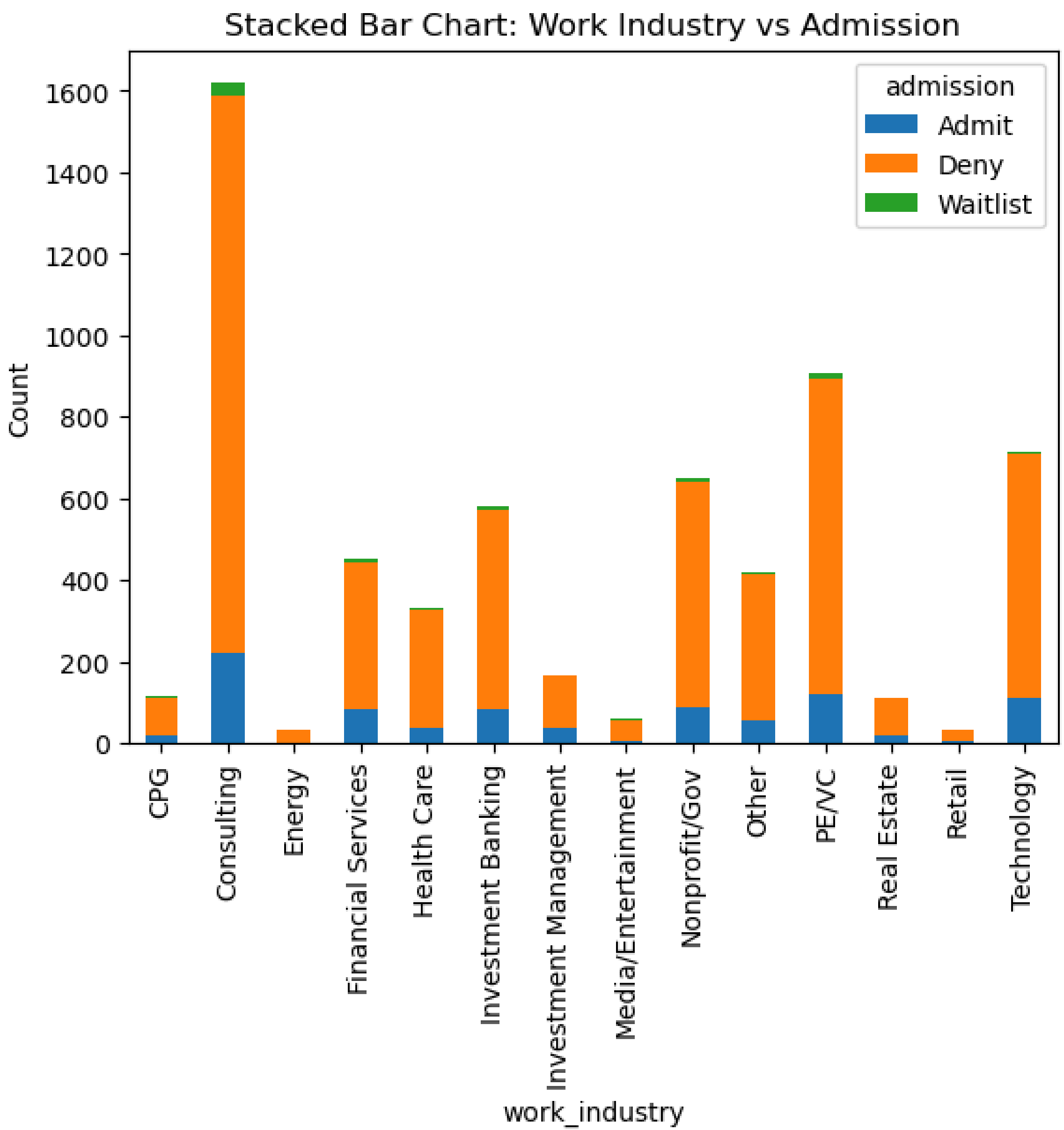


In [30]:
```
pd.crosstab(df['international'], df['admission']).plot(kind='bar', stacked=True)
plt.ylabel('Count')
```

```
plt.title('Stacked Bar Chart: Resident vs Admission')
plt.show()
```



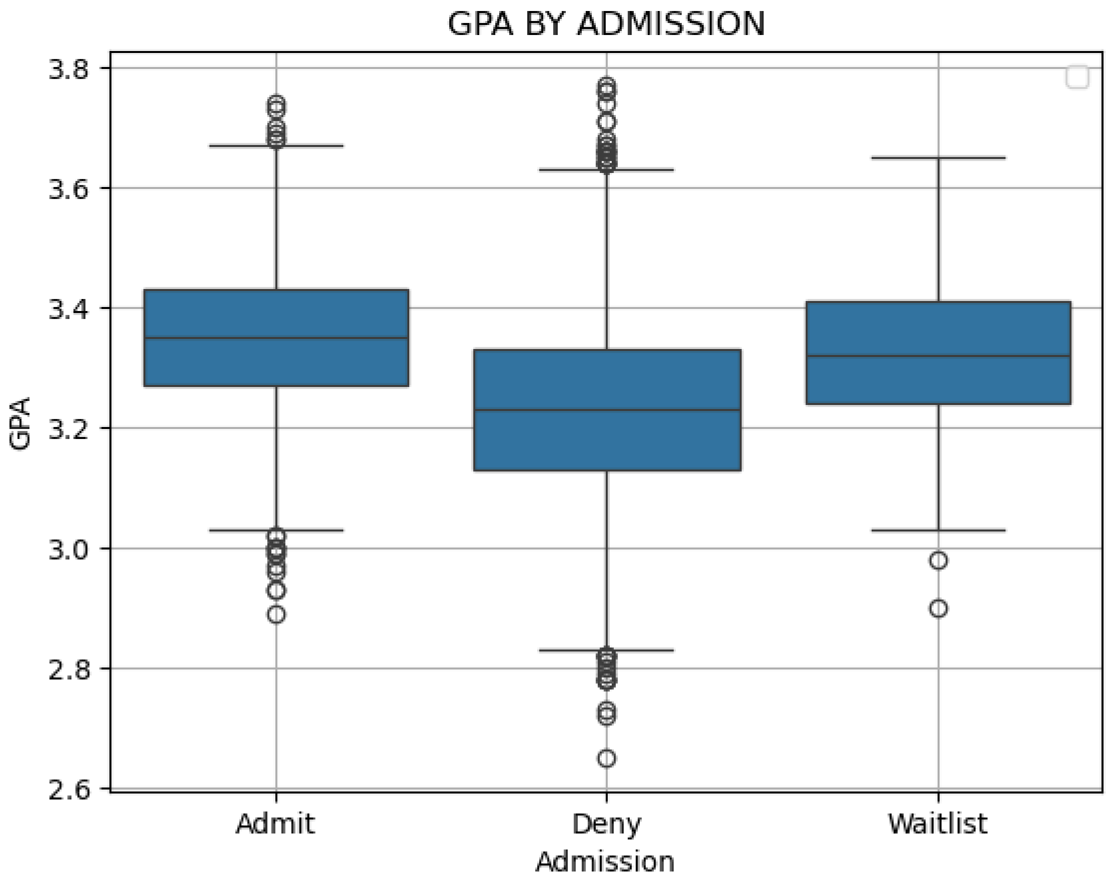
In [31]:
```
pd.crosstab(df['work_industry'], df['admission']).plot(kind='bar', stacked=True)
plt.ylabel('Count')
plt.title('Stacked Bar Chart: Work Industry vs Admission')
plt.show()
```



In [32]:
```
plt.Figure(figsize=(8,6))
sns.boxplot(x = 'admission', y = 'gpa', data = df)
```

```
plt.title('GPA BY ADMISSION')
plt.xlabel('Admission')
plt.ylabel('GPA')
plt.legend()
plt.grid()
```

```
C:\Users\Ritik\AppData\Local\Temp\ipykernel_1892\310484407.py:7: UserWarning: No artists with labels found to put in legend.  N
ote that artists whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```

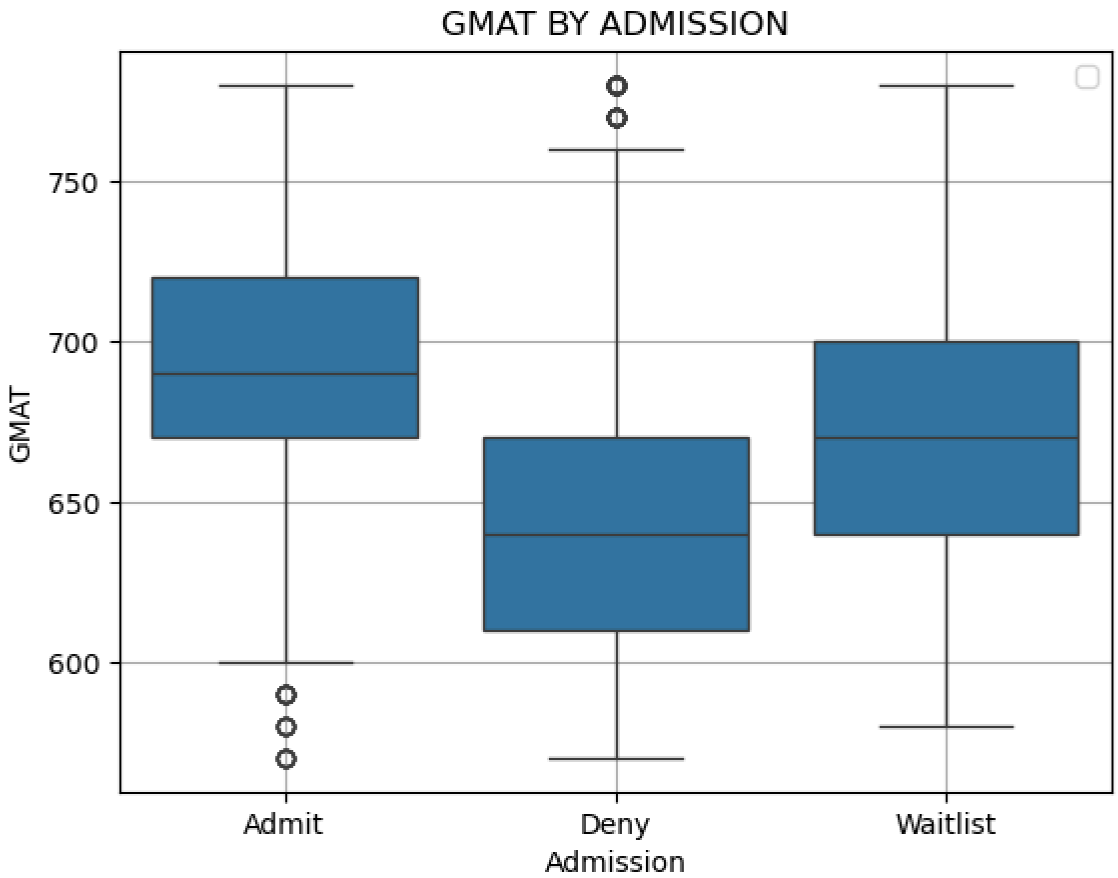


In [33]:
```
plt.Figure(figsize=(8,6))
sns.boxplot(x = 'admission', y = 'gmat', data = df)

plt.title('GMAT BY ADMISSION')
```

```
plt.xlabel('Admission')
plt.ylabel('GMAT')
plt.legend()
plt.grid()
```

```
C:\Users\Ritik\AppData\Local\Temp\ipykernel_1892\3329509654.py:7: UserWarning: No artists with labels found to put in legend.
Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```

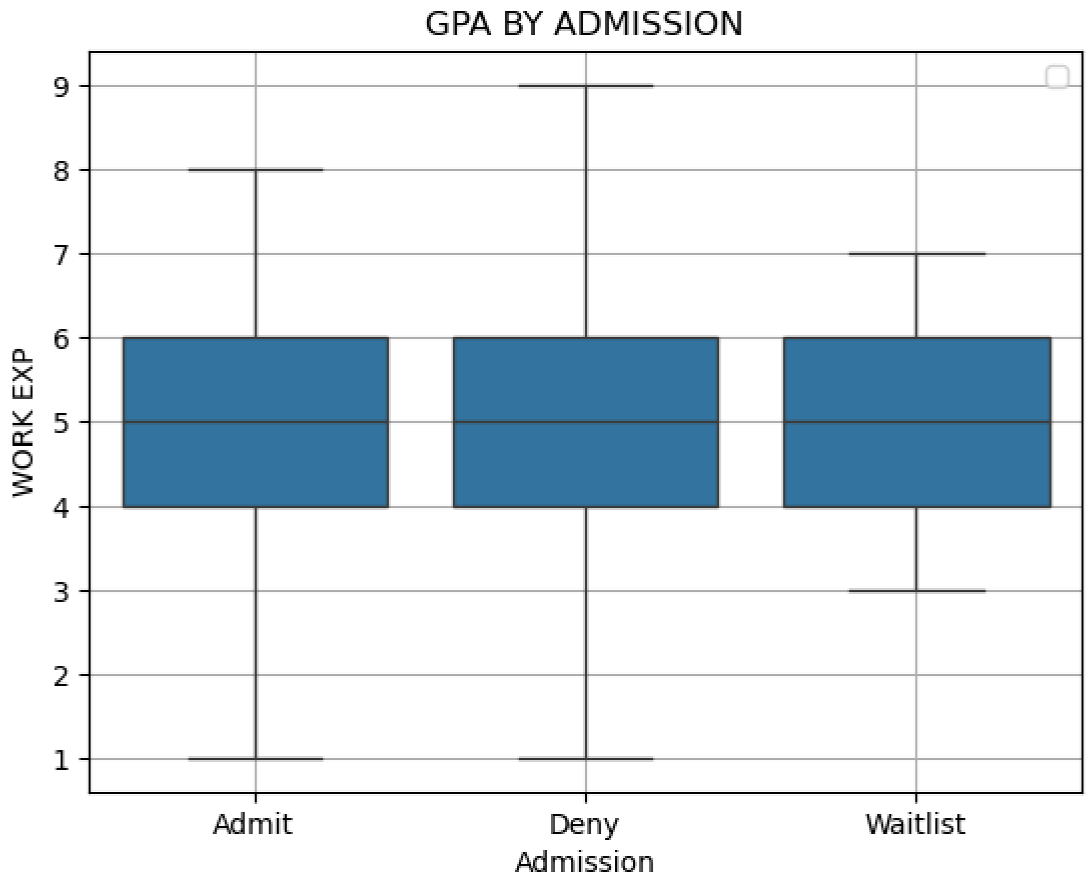


In [34]:
```
plt.Figure(figsize=(8,6))
sns.boxplot(x = 'admission', y = 'work_exp', data = df)

plt.title('GPA BY ADMISSION')
plt.xlabel('Admission')
```

```
plt.ylabel('WORK EXP')
plt.legend()
plt.grid()
```

```
C:\Users\Ritik\AppData\Local\Temp\ipykernel_1892\865309916.py:7: UserWarning: No artists with labels found to put in legend.  N
ote that artists whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend()
```



In [35]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

In [36]:
```
# Preprocessing
# Convert categorical variables to numerical values
label_encoders = {}
categorical_cols = ['gender', 'major', 'race', 'work_industry', 'admission']

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

In [37]:
```
# Define features (X) and target (y)
X = df.drop('admission', axis=1)  # Features
y = df['admission']  # Target variable
```

In [38]:
```
X
```

Out[38]:

	application_id	gender	international	gpa	major	race	gmat	work_exp	work_industry
0	1	0	False	3.30	0	0	620	3	3
1	2	1	False	3.28	1	1	680	5	6
2	3	0	True	3.30	0	3	710	5	13
3	4	1	False	3.47	2	1	690	6	13
4	5	1	False	3.35	2	2	590	5	1
...	...	...	...	...	...	...	...	...	...
6189	6190	1	False	3.49	0	5	640	5	9
6190	6191	1	False	3.18	2	1	670	4	1
6191	6192	0	True	3.22	0	3	680	5	4
6192	6193	1	True	3.36	0	3	590	5	9
6193	6194	1	False	3.23	2	2	650	4	1

6194 rows × 9 columns

In [39]:

```
y
```

Out[39]:

```
0       0
1       1
2       0
3       1
4       1
       ..
6189    1
6190    1
6191    0
6192    1
6193    1
Name: admission, Length: 6194, dtype: int32
```

In [40]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [41]:
```
# Create and train the decision tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

Out[41]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [42]:
```
# Make predictions
y_pred = model.predict(X_test)
```

In [43]:
```
# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[ 138  154    7]
 [ 117 1396   12]
 [   9   20    6]]
```

The problem is actually a multi-class classification problem, where there are more than two possible target classes.

The additional row in the confusion matrix represents:

Row 1: Predictions for the first class Row 2: Predictions for the second class Row 3: Predictions for the third class Specifically, the last row shows the following:

9: Number of samples that were correctly predicted as the third class 20: Number of samples that were incorrectly predicted as the third class 6: Number of samples that were actually the third class, but were not correctly predicted

In [44]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       0.52      0.46      0.49       299
           1       0.89      0.92      0.90      1525
           2       0.24      0.17      0.20        35

    accuracy                           0.83      1859
   macro avg       0.55      0.52      0.53      1859
weighted avg       0.82      0.83      0.82      1859
```

Imbalanced Data:

The support (number of samples) for the different classes is highly imbalanced. Class 1 has 1525 samples, while Class 2 has only 35 samples. This imbalance in the dataset can lead to the model performing poorly on the minority class (Class 2). Low Precision and Recall for Class 2:

The precision for Class 2 is only 0.24, meaning that only 24% of the samples predicted as Class 2 are actually from that class. The recall for Class 2 is 0.17, meaning that the model is only able to correctly identify 17% of the actual Class 2 samples. This indicates that the model is struggling to accurately classify the samples belonging to the minority Class 2. Overall Accuracy is Not Satisfactory:

The overall accuracy of the model is 0.83, which may not be high enough, especially given the imbalanced nature of the dataset. The weighted average F1-score is 0.82, which suggests that the model's overall performance could be improved. To address these issues, we consider the following steps:

Handle Imbalanced Data: Employ up techniques like oversampling the minority class, undersampling the majority class, or using class weights to balance the dataset. This can help the model learn the patterns in the minority class more effectively.

Improve Model Performance: Try different classification algorithms or hyperparameter tuning to see if you can improve the model's performance on the minority class. Consider using ensemble methods, such as bagging or boosting, which can sometimes handle imbalanced data better. Evaluate Model Holistically:

In addition to accuracy, also consider other metrics like precision, recall, and F1-score for each class to get a more comprehensive understanding of the model's performance. These metrics can provide insights into the model's strengths and weaknesses, especially when dealing with imbalanced datasets.

Oversample the Minority Class:

In [45]:
```
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE

# Assuming X_train and y_train are your training data and labels
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

In [46]:
```
print(f"Original class distribution:\n{y_train.value_counts()}")
print(f"Resampled class distribution:\n{pd.Series(y_train_resampled).value_counts()}")
```

```
Original class distribution:
admission
1    3669
0     601
2      65
Name: count, dtype: int64
Resampled class distribution:
admission
0    3669
1    3669
2    3669
Name: count, dtype: int64
```

In [ ]:

Undersample the Majority Class

In [47]:
```
from imblearn.under_sampling import RandomUnderSampler

# Assuming X_train and y_train are your training data and labels
rus = RandomUnderSampler()
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)
```

In [48]:
```
print(f"Original class distribution:\n{y_train.value_counts()}")
print(f"Resampled class distribution:\n{pd.Series(y_train_resampled).value_counts()}")
```

```
Original class distribution:
admission
1    3669
0     601
2      65
Name: count, dtype: int64
Resampled class distribution:
admission
0    65
1    65
2    65
Name: count, dtype: int64
```

Combination of Oversampling and Undersampling

In [49]:
```
#Combined oversampling for minority classes and
#undersampling for majority classes using SMOTEENN

from imblearn.combine import SMOTEENN

smote_enn = SMOTEENN(random_state=42)
X_train_resampled, y_train_resampled = smote_enn.fit_resample(X_train, y_train)
print(f"Resampled class distribution:\n{pd.Series(y_train_resampled).value_counts()}")
```

```
Resampled class distribution:
admission
2    2899
0    2493
1    2155
Name: count, dtype: int64
```

In [ ]:

Use Class Weights:

In [50]:
```
from sklearn.tree import DecisionTreeClassifier

# Assuming X_train and y_train are your training data and labels
class_weights = {0: 1, 1: 5,2:10}  # Assign a higher weight to the minority class
```

```
model = DecisionTreeClassifier(class_weight=class_weights)
model.fit(X_train, y_train)
```

Out[50]:

```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight={0: 1, 1: 5, 2: 10})
```

In [51]:

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.47      0.44      0.45       299
           1       0.89      0.90      0.89      1525
           2       0.31      0.26      0.28        35

    accuracy                           0.81      1859
   macro avg       0.55      0.53      0.54      1859
weighted avg       0.81      0.81      0.81      1859
```

Try Different Algorithms and Hyperparameter Tuning:

In [52]:

```
#Used Random Forest here to check
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Assuming X_train, y_train, X_test, y_test are your data
rf = RandomForestClassifier()
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(rf, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

Evaluate Model Holistically:

In [53]:
```
from sklearn.metrics import classification_report

y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.18      0.30       299
           1       0.85      0.99      0.92      1525
           2       1.00      0.09      0.16        35

    accuracy                           0.85      1859
   macro avg       0.89      0.42      0.46      1859
weighted avg       0.85      0.85      0.80      1859
```

Accuracy achieved of 85 with Random Forest

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

#  X_train, y_train, X_test, y_test are data
dt = DecisionTreeClassifier()
param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

grid_search = GridSearchCV(dt, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

In [55]:
```
from sklearn.metrics import classification_report

y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.19      0.30       299
           1       0.85      0.98      0.91      1525
           2       1.00      0.03      0.06        35

    accuracy                           0.84      1859
   macro avg       0.83      0.40      0.42      1859
weighted avg       0.82      0.84      0.80      1859
```

Accuracy of 84 is achieved by the use of a Decision Tree Classifier and hyperparameter tuning, but the performance metrics shown in the classification report indicate that the model is still struggling with the imbalanced dataset.

Some key observations:

Accuracy vs. Holistic Metrics: The overall accuracy of the model is 0.84, which seems reasonably good. However, the macro- average and weighted-average metrics (precision, recall, F1-score) are much lower, suggesting that the model is not performing well across all classes.

To address the imbalanced data issue more effectively, we may need to consider additional techniques, such as:

Oversampling the Minority Class: Using techniques like SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples of the minority class, which can help the model learn better from the underrepresented classes.

Undersampling the Majority Class: Removing some samples from the majority class to balance the dataset.

Class Weighting: Assigning higher weights to the minority classes during the training process to make the model more sensitive to the underrepresented classes.

Ensemble Methods: Trying other ensemble algorithms, such as Random Forest or Gradient Boosting, which can sometimes perform better on imbalanced datasets compared to a single Decision Tree Classifier.

In [56]:

```
#Decison tree : higher weight
```

In [57]:

```
from imblearn.over_sampling import SMOTE
```

```
# Assuming X_train, y_train are your original training data
sm = SMOTE()
X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)

# Train the Decision Tree Classifier on the resampled data
dt = DecisionTreeClassifier()
dt.fit(X_train_resampled, y_train_resampled)
```

Out[57]:

```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [58]:

```
#Hyperparameter tuning
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

dt = DecisionTreeClassifier()
grid_search = GridSearchCV(dt, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

In [59]:

```
from sklearn.metrics import classification_report

y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.64      0.19      0.30       299
           1       0.85      0.98      0.91      1525
           2       1.00      0.03      0.06        35

    accuracy                           0.84      1859
   macro avg       0.83      0.40      0.42      1859
weighted avg       0.82      0.84      0.80      1859
```

In [60]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\programdata\anaconda3\lib\site-packages (2.1.3)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.13.1)
Note: you may need to restart the kernel to use updated packages.
```

Use advanced techniques

A)Using Ensemble Models

In [61]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier(scale_pos_weight=10)  # Adjust `scale_pos_weight` for class imbalance
xgb.fit(X_train, y_train)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [19:43:20] WARNING: C:\buildkite-agent\builds\buil
dkite-windows-cpu-autoscaling-group-i-0c55ff5f71b100e98-1\xgboost\xgboost-ci-windows\src\learner.cc:740: 
Parameters: { "scale_pos_weight" } are not used.

  warnings.warn(smsg, UserWarning)
```

Out[61]:

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=None, 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, monotone_constraints=None,
```

In [62]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

In [63]:

```
# Split your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [64]:

```
# Initialize the XGBoost classifier
model = XGBClassifier(objective='multi:softmax', eval_metric='mlogloss')

# Train the model
model.fit(X_train, y_train)
```

Out[64]:

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='mlogloss',
              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 [65]:

```
# Make predictions
y_pred = model.predict(X_test)

# Evaluate
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.54      0.34      0.42       299
           1       0.87      0.95      0.91      1525
           2       0.56      0.14      0.23        35

    accuracy                           0.83      1859
   macro avg       0.66      0.48      0.52      1859
weighted avg       0.81      0.83      0.82      1859
```

In [66]:

```
# Use Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Assuming X_train, y_train, X_test, y_test are your data
gbc = GradientBoostingClassifier()
param_grid = {
    'n_estimators': [100, 200, 300],
```

```
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001]
}

grid_search = GridSearchCV(gbc, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.15      0.26       299
           1       0.85      1.00      0.91      1525
           2       0.42      0.14      0.21        35

    accuracy                           0.84      1859
   macro avg       0.71      0.43      0.46      1859
weighted avg       0.84      0.84      0.80      1859
```

In [ ]: