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

```
C:\Users\hp\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.
0 is required for this version of SciPy (detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
df = pd.read_excel("C:/Users/hp/Downloads/WHO-UCN-GTB-PCI-2021.7-eng.xlsx", sheet_name ='Mutation_catalogue')
```

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

Out[3]:

	Drug	Tier	Common Variant	Genome position	algorithm pass	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	...	RIF CC guide 2021	Ha GenoTy MTBDRpl V:
0	AMI	1.0	rrs_a1401g	1473246.0	1.0	918.0	966.0	50.0	15640.0	939.0	...	NaN	N
1	AMI	1.0	eis_c-14t	2715346.0	1.0	28.0	77.0	51.0	7325.0	32.0	...	NaN	N
2	AMI	1.0	rrs_g1484t	1473329.0	1.0	5.0	7.0	2.0	15688.0	6.0	...	NaN	N
3	AMI	1.0	rrs_c1402t	1473247.0	1.0	4.0	13.0	10.0	15680.0	5.0	...	NaN	N
4	AMI	2.0	whiB6_A77V	NaN	1.0	3.0	100.0	97.0	15593.0	3.0	...	NaN	N

5 rows × 52 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17396 entries, 0 to 17395
Data columns (total 52 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Drug                              17396 non-null  object 
 1   Tier                              17348 non-null  float64
 2   Common Variant                    17396 non-null  object 
 3   Genome position                   6486 non-null   float64
 4   algorithm pass                    17388 non-null  float64
 5   Present_SOLO_R                    17388 non-null  float64
 6   Present_SOLO_SR                   17388 non-null  float64
 7   Present_S                         17388 non-null  float64
 8   Absent_S                          17388 non-null  float64
 9   Present_R                         17388 non-null  float64
 10  Absent_R                          17388 non-null  float64
 11  PPV                               17388 non-null  float64
 12  PPV_lb                            17388 non-null  float64
 13  PPV_ub                            17388 non-null  float64
 14  PPV | SOLO                        16074 non-null  float64
 15  PPV | SOLO_lb                     16074 non-null  float64
 16  PPV | SOLO_ub                     16074 non-null  float64
 17  Sensitivity                       17388 non-null  float64
 18  Sensitivity_lb                    17388 non-null  float64
 19  Sensitivity_ub                    17388 non-null  float64
 20  Specificity                       17388 non-null  float64
 21  Specificity_lb                    17388 non-null  float64
 22  Specificity_ub                    17388 non-null  float64
 23  LR+                               17388 non-null  float64
 24  LR+_lb                            17388 non-null  float64
 25  LR+_ub                            17388 non-null  float64
 26  LR-                               17385 non-null  float64
 27  LR-_lb                            17388 non-null  float64
 28  LR-_ub                            17388 non-null  float64
 29  OR                                17385 non-null  float64
 30  OR_lb                             17388 non-null  float64
 31  OR_ub                             17388 non-null  float64
 32  OR SOLO                           7696 non-null   float64
 33  OR SOLO_lb                        7696 non-null   float64
 34  OR SOLO_ub                        7696 non-null   float64
 35  OR SOLO_FE-sig                    17388 non-null  float64
 36  Neutral masked                    17388 non-null  float64
 37  INITIAL CONFIDENCE GRADING        17388 non-null  object 
 38  DATASET(S)                        17388 non-null  object 
 39  Miotto et al. (PMID 29284687)     311 non-null    object 
 40  NGS Guide 2018                    256 non-null    object 
 41  Level of resistance to INH or MXF 131 non-null    object 
 42  RIF CC guide 2021                 135 non-null    object 
 43  Hain GenoType MTBDRplus V2.0      151 non-null    object 
 44  Nipro Genoscholar NTM+MDRTB II    176 non-null    object 
 45  Cepheid Xpert MTB/RIF             135 non-null    object 
 46  Cepheid Xpert MTB/RIF Ultra       139 non-null    object 
 47  Hain GenoType MTBDRsl V2.0        76 non-null     object 
 48  Cepheid Xpert MTB/XDR             199 non-null    object 
 49  Nipro Genoscholar PZA-TB II       503 non-null    object 
 50  Additional grading criteria       936 non-null    object 
 51  FINAL CONFIDENCE GRADING          17396 non-null  object 
dtypes: float64(35), object(17)
memory usage: 6.9+ MB
```

In [5]:
```
df.isnull().sum()#Checking for missing values
```

Out[5]:
```
Drug                                  0
Tier                                 48
Common Variant                        0
Genome position                   10910
algorithm pass                        8
Present_SOLO_R                        8
Present_SOLO_SR                       8
Present_S                             8
Absent_S                              8
Present_R                             8
Absent_R                              8
PPV                                   8
PPV_lb                                8
PPV_ub                                8
PPV | SOLO                         1322
PPV | SOLO_lb                      1322
PPV | SOLO_ub                      1322
Sensitivity                           8
Sensitivity_lb                        8
Sensitivity_ub                        8
Specificity                           8
Specificity_lb                        8
Specificity_ub                        8
LR+                                   8
LR+_lb                                8
LR+_ub                                8
LR-                                  11
LR-_lb                                8
LR-_ub                                8
OR                                   11
OR_lb                                 8
OR_ub                                 8
OR SOLO                            9700
OR SOLO_lb                         9700
OR SOLO_ub                         9700
OR SOLO_FE-sig                        8
Neutral masked                        8
INITIAL CONFIDENCE GRADING            8
DATASET(S)                            8
Miotto et al. (PMID 29284687)     17085
NGS Guide 2018                    17140
Level of resistance to INH or MXF 17265
RIF CC guide 2021                 17261
Hain GenoType MTBDRplus V2.0      17245
Nipro Genoscholar NTM+MDRTB II    17220
Cepheid Xpert MTB/RIF             17261
Cepheid Xpert MTB/RIF Ultra       17257
Hain GenoType MTBDRsl V2.0        17320
Cepheid Xpert MTB/XDR             17197
Nipro Genoscholar PZA-TB II       16893
Additional grading criteria       16460
FINAL CONFIDENCE GRADING              0
dtype: int64
```

In [6]:
```
df.columns #Checking for the number of features
```

Out[6]:
```
Index(['Drug', 'Tier', 'Common Variant', 'Genome position', 'algorithm pass',
       'Present_SOLO_R', 'Present_SOLO_SR', 'Present_S', 'Absent_S',
       'Present_R', 'Absent_R', 'PPV', 'PPV_lb', 'PPV_ub', 'PPV | SOLO',
       'PPV | SOLO_lb', 'PPV | SOLO_ub', 'Sensitivity', 'Sensitivity_lb',
       'Sensitivity_ub', 'Specificity', 'Specificity_lb', 'Specificity_ub',
       'LR+', 'LR+_lb', 'LR+_ub', 'LR-', 'LR-_lb', 'LR-_ub', 'OR', 'OR_lb',
       'OR_ub', 'OR SOLO', 'OR SOLO_lb', 'OR SOLO_ub', 'OR SOLO_FE-sig',
       'Neutral masked', 'INITIAL CONFIDENCE GRADING', 'DATASET(S)',
       'Miotto et al. (PMID 29284687)', 'NGS Guide 2018',
       'Level of resistance to INH or MXF', 'RIF CC guide 2021',
       'Hain GenoType MTBDRplus V2.0', 'Nipro Genoscholar NTM+MDRTB II',
       'Cepheid Xpert MTB/RIF', 'Cepheid Xpert MTB/RIF Ultra',
       'Hain GenoType MTBDRsl V2.0', 'Cepheid Xpert MTB/XDR',
       'Nipro Genoscholar PZA-TB II', 'Additional grading criteria',
       'FINAL CONFIDENCE GRADING'],
      dtype='object')
```

In [7]:
```
drop_columns = ['Genome position', 'algorithm pass', 'PPV', 'PPV_lb', 'PPV_ub',  'PPV | SOLO',
                'PPV | SOLO_lb', 'PPV | SOLO_ub','Sensitivity', 'Sensitivity_lb', 'Sensitivity_ub',
                'Specificity', 'Specificity_lb', 'Specificity_ub', 'LR+', 'LR+_lb', 'LR+_ub', 'LR-', 'LR-_lb',
                'LR-_ub', 'OR', 'OR_lb', 'OR_ub', 'OR SOLO', 'OR SOLO_lb', 'OR SOLO_ub', 'OR SOLO_FE-sig',
                'DATASET(S)', 'Miotto et al. (PMID 29284687)', 'NGS Guide 2018', 'Level of resistance to INH or
                'RIF CC guide 2021', 'Hain GenoType MTBDRplus V2.0', 'Nipro Genoscholar NTM+MDRTB II',
                'Cepheid Xpert MTB/RIF', 'Cepheid Xpert MTB/RIF Ultra', 'Hain GenoType MTBDRsl V2.0',
                'Cepheid Xpert MTB/XDR', 'Nipro Genoscholar PZA-TB II', 'Additional grading criteria']

# Dropping the specified columns from the DataFrame
df.drop(columns=drop_columns, inplace=True)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17396 entries, 0 to 17395
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Drug                      17396 non-null  object 
 1   Tier                      17348 non-null  float64
 2   Common Variant            17396 non-null  object 
 3   Present_SOLO_R            17388 non-null  float64
 4   Present_SOLO_SR           17388 non-null  float64
 5   Present_S                 17388 non-null  float64
 6   Absent_S                  17388 non-null  float64
 7   Present_R                 17388 non-null  float64
 8   Absent_R                  17388 non-null  float64
 9   Neutral masked            17388 non-null  float64
 10  INITIAL CONFIDENCE GRADING  17388 non-null  object 
 11  FINAL CONFIDENCE GRADING  17396 non-null  object 
dtypes: float64(8), object(4)
memory usage: 1.6+ MB
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Drug                           0
Tier                          48
Common Variant                 0
Present_SOLO_R                 8
Present_SOLO_SR                8
Present_S                      8
Absent_S                       8
Present_R                      8
Absent_R                       8
Neutral masked                 8
INITIAL CONFIDENCE GRADING     8
FINAL CONFIDENCE GRADING       0
dtype: int64
```

In [10]:
```
df.dtypes
```

Out[10]:
```
Drug                           object
Tier                          float64
Common Variant                 object
Present_SOLO_R                float64
Present_SOLO_SR               float64
Present_S                     float64
Absent_S                      float64
Present_R                     float64
Absent_R                      float64
Neutral masked                float64
INITIAL CONFIDENCE GRADING     object
FINAL CONFIDENCE GRADING       object
dtype: object
```

In [11]:
```
#Handle missing values in categorical columns by imputing with mode
categorical_cols = ['Drug', 'Common Variant', 'Neutral masked', 'INITIAL CONFIDENCE GRADING', 'FINAL CONFIDENCE
df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])

df.head()
```

Out[11]:

	Drug	Tier	Common Variant	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R	Neutral masked	INITIAL CONFIDENCE GRADING	CONFI GR
0	AMI	1.0	rrs_a1401g	918.0	966.0	50.0	15640.0	939.0	349.0	0.0	Assoc w R	1) Ass
1	AMI	1.0	eis_c-14t	28.0	77.0	51.0	7325.0	32.0	632.0	0.0	Assoc w R	1) Ass
2	AMI	1.0	rrs_g1484t	5.0	7.0	2.0	15688.0	6.0	1282.0	0.0	Assoc w R	2) Asso
3	AMI	1.0	rrs_c1402t	4.0	13.0	10.0	15680.0	5.0	1283.0	0.0	Uncertain significance	2) Asso
4	AMI	2.0	whiB6_A77V	3.0	100.0	97.0	15593.0	3.0	141.0	0.0	Uncertain significance	3) Ur sign

In [12]:
```
df.info()
```

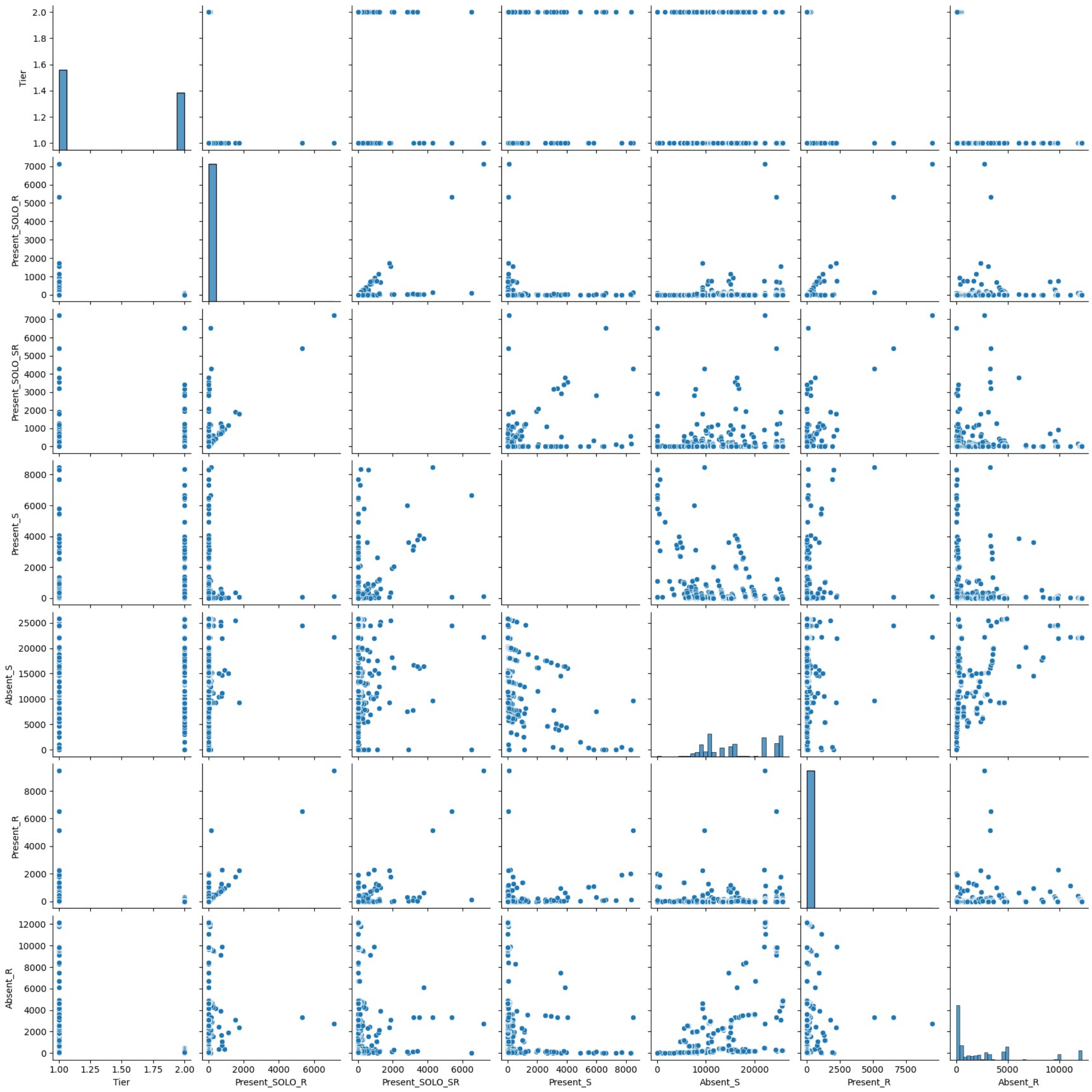
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17396 entries, 0 to 17395
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Drug                       17396 non-null  object 
 1   Tier                       17348 non-null  float64
 2   Common Variant             17396 non-null  object 
 3   Present_SOLO_R             17388 non-null  float64
 4   Present_SOLO_SR            17388 non-null  float64
 5   Present_S                  17388 non-null  float64
 6   Absent_S                   17388 non-null  float64
 7   Present_R                  17388 non-null  float64
 8   Absent_R                   17388 non-null  float64
 9   Neutral masked             17396 non-null  float64
 10  INITIAL CONFIDENCE GRADING  17396 non-null  object 
 11  FINAL CONFIDENCE GRADING    17396 non-null  object 
dtypes: float64(8), object(4)
memory usage: 1.6+ MB
```

In [13]:
```
unique_values = ['Drug', 'Common Variant', 'INITIAL CONFIDENCE GRADING', 'FINAL CONFIDENCE GRADING']

# Calculate the number of unique values for the specified columns
unique_count = df[unique_values].nunique()

unique_count
```

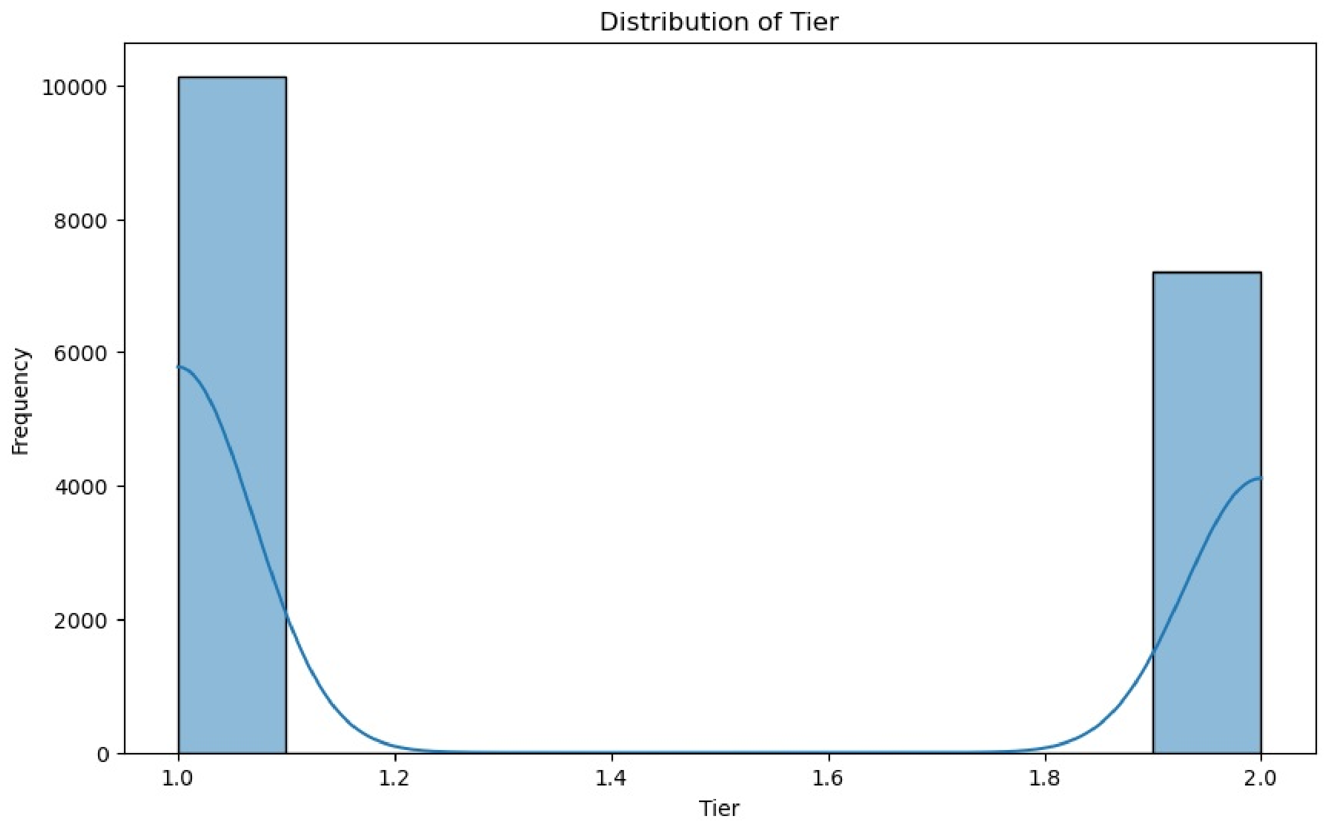
Out[13]:
```
Drug                              15
Common Variant                 13449
INITIAL CONFIDENCE GRADING         6
FINAL CONFIDENCE GRADING           6
dtype: int64
```

In [14]:
```
numeric_columns = ['Tier', 'Present_SOLO_R', 'Present_SOLO_SR', 'Present_S', 'Absent_S',
                   'Present_R', 'Absent_R']
```

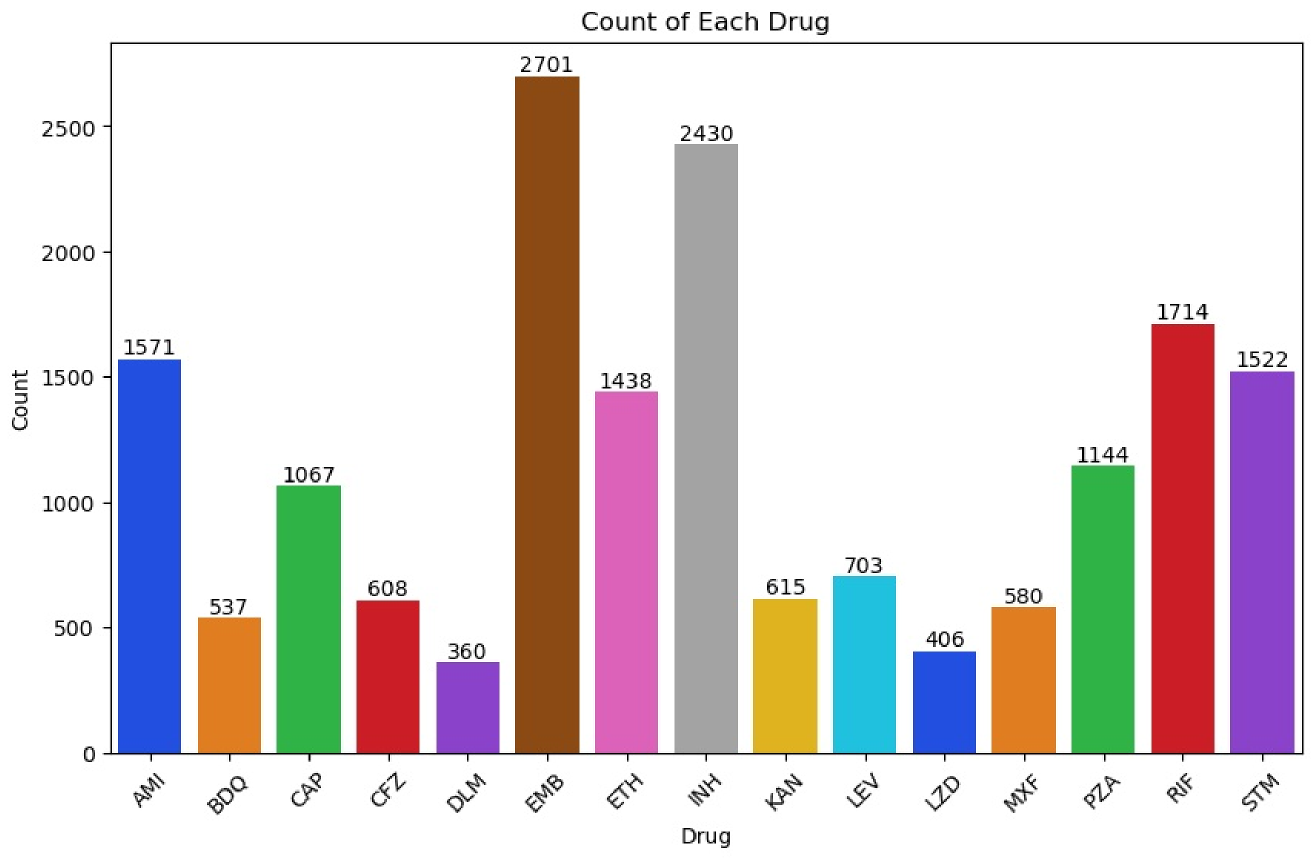
In [15]:
```
sns.pairplot(df[numeric_columns])
plt.show()
```



In [16]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Tier', bins=10, kde=True)
plt.xlabel('Tier')
plt.ylabel('Frequency')
plt.title('Distribution of Tier')
plt.show()
```

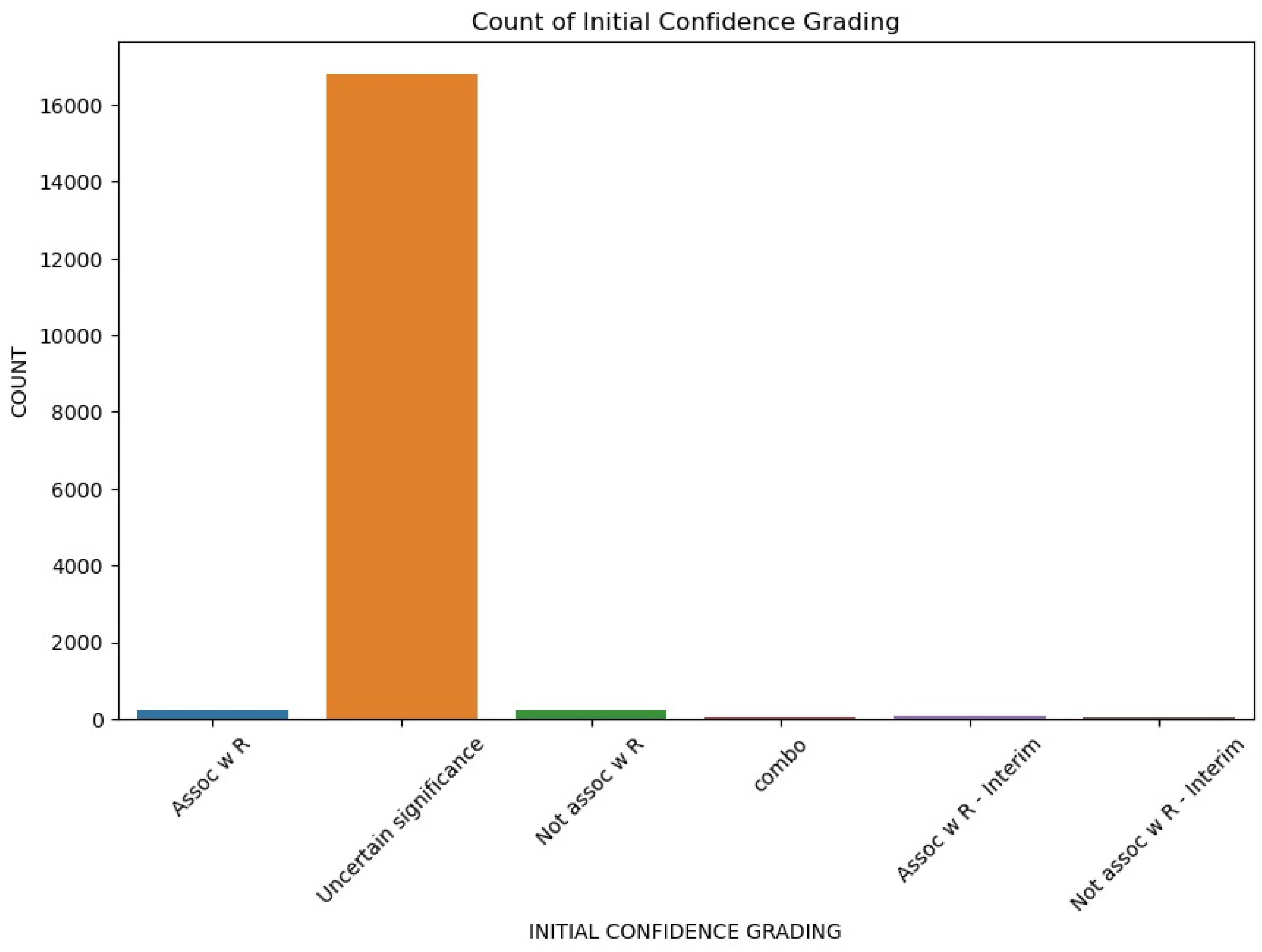


In [17]:
```
plt.figure(figsize=(10, 6))
ax = sns.countplot(data=df, x='Drug', palette='bright')

# Add text annotations to the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.xticks(rotation=45)
plt.xlabel('Drug')
plt.ylabel('Count')
plt.title('Count of Each Drug')
plt.show()
```

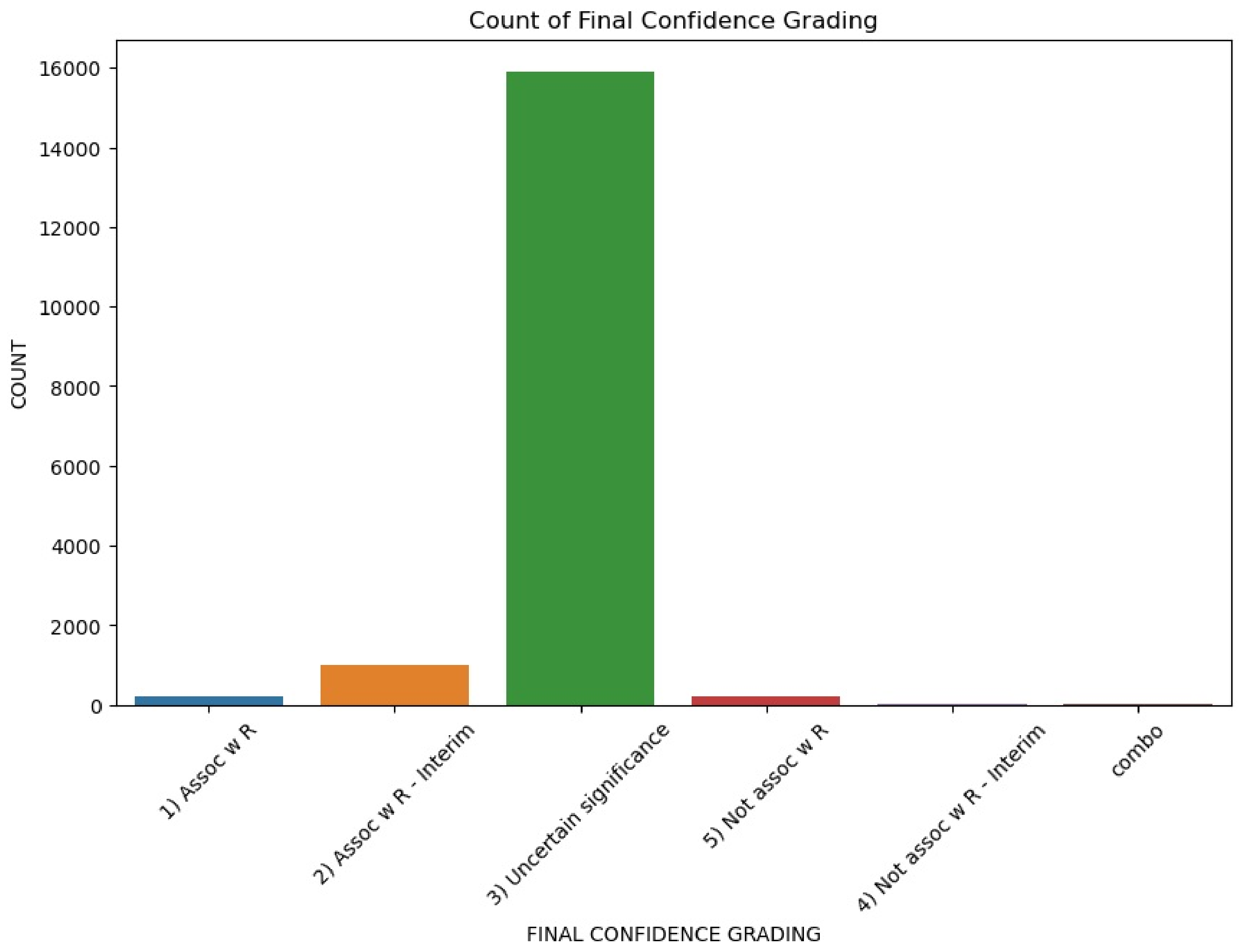


In [18]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='INITIAL CONFIDENCE GRADING')
```

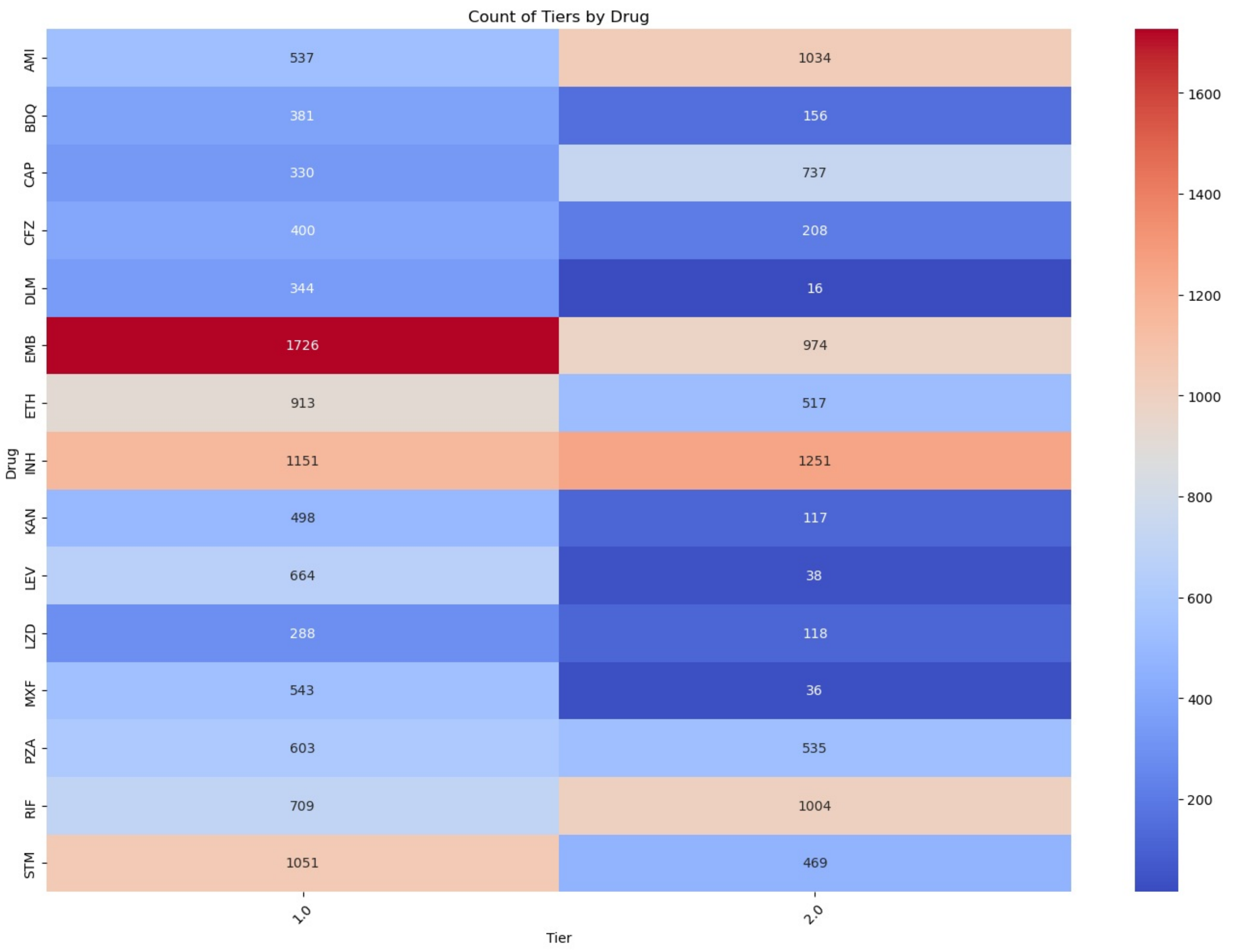
```
plt.xticks(rotation=45)
plt.xlabel('INITIAL CONFIDENCE GRADING')
plt.ylabel('COUNT')
plt.title('Count of Initial Confidence Grading')
plt.show()
```



In [19]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='FINAL CONFIDENCE GRADING')
plt.xticks(rotation=45)
plt.xlabel('FINAL CONFIDENCE GRADING')
plt.ylabel('COUNT')
plt.title('Count of Final Confidence Grading')
plt.show()
```

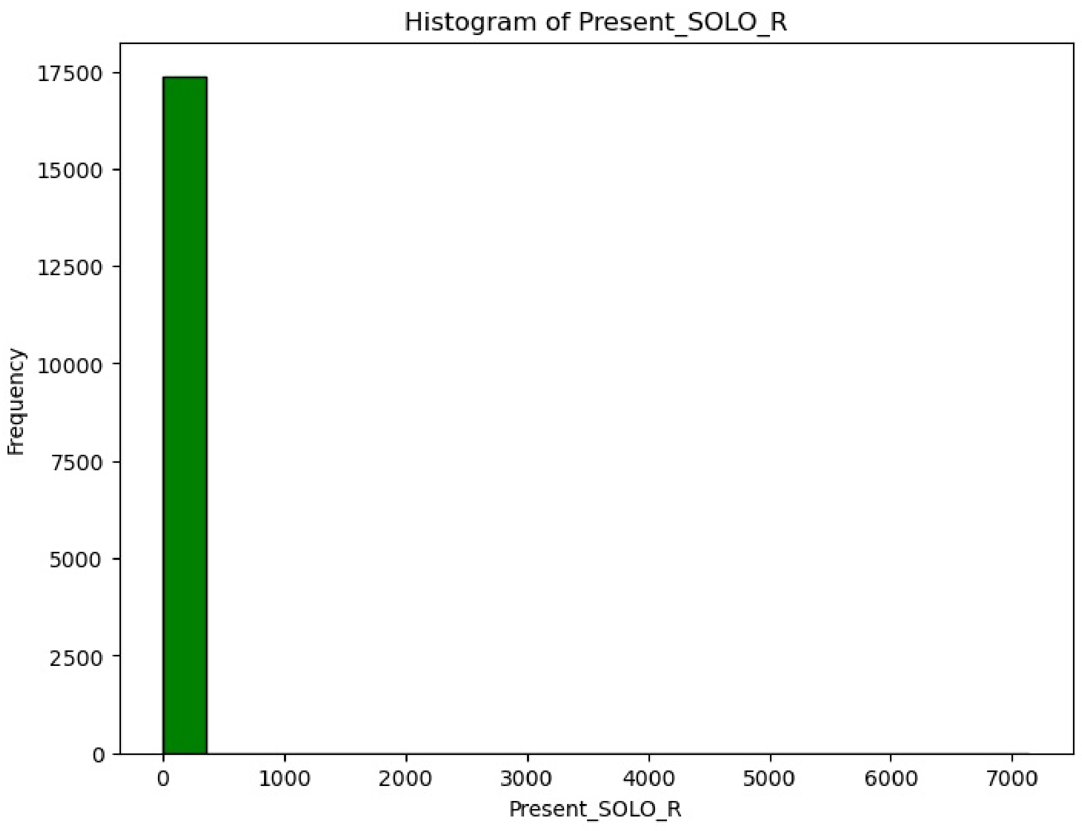


```
In [20]: pivot_table = df.pivot_table(index='Drug', columns='Tier', aggfunc='size', fill_value=0)

         # Plot heatmap
         plt.figure(figsize=(14, 10))
         sns.heatmap(pivot_table, cmap='coolwarm', annot=True, fmt='d')
         plt.title('Count of Tiers by Drug')
         plt.xlabel('Tier')
         plt.ylabel('Drug')
         plt.xticks(rotation=45)
         plt.tight_layout()
         plt.show()
```

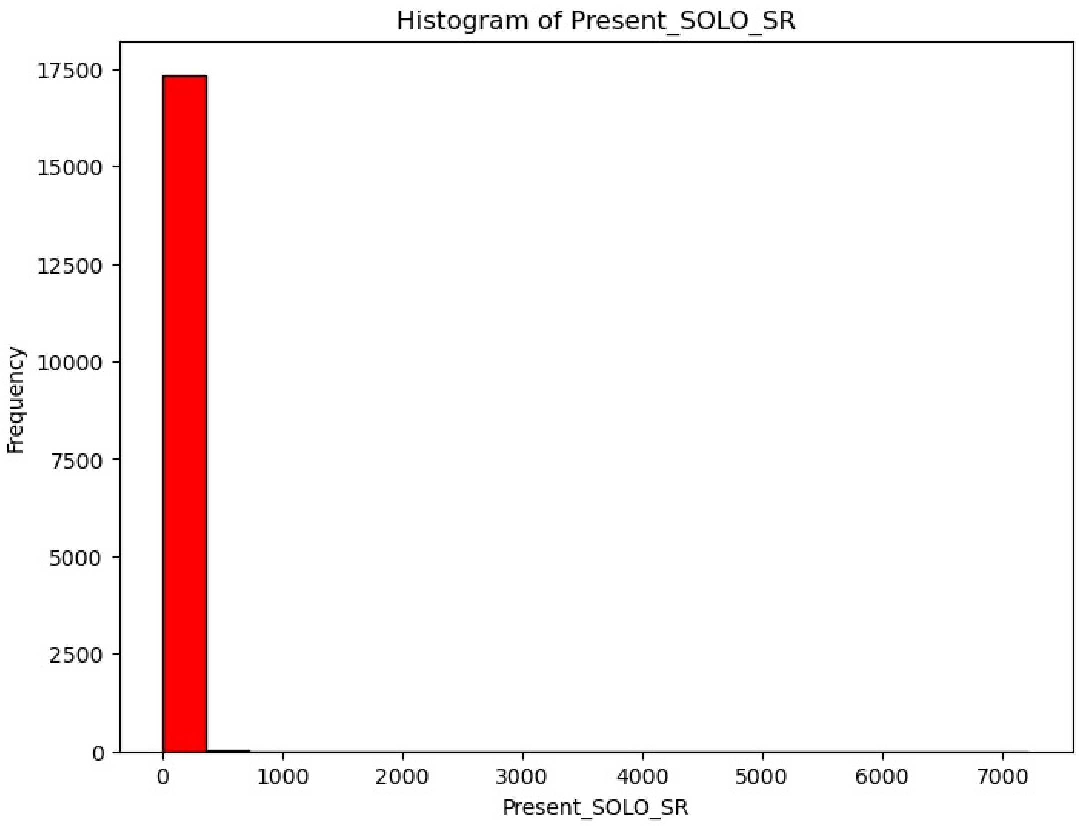


In [21]:
```
from sklearn.preprocessing import LabelEncoder
#categorical_cols = df.select_dtypes(include='object').columns
```

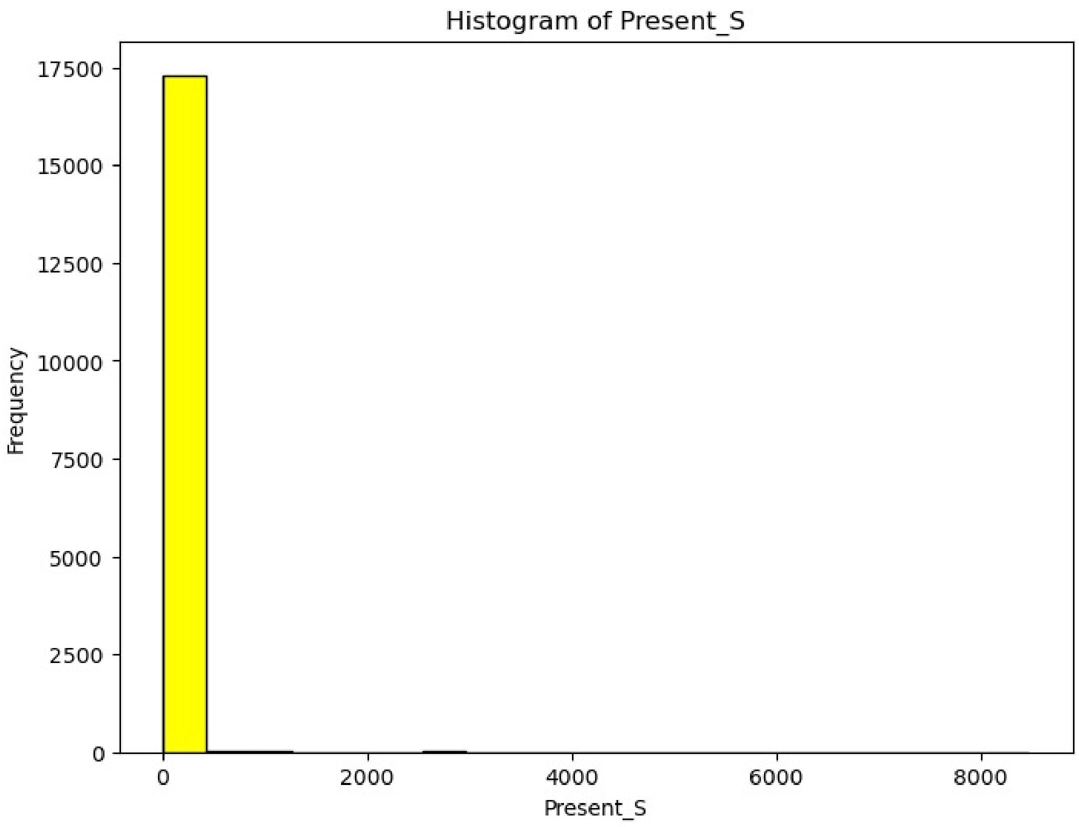
In [22]:
```
plt.figure(figsize=(8, 6))
plt.hist(df['Present_SOLO_R'], bins=20, color='green', edgecolor='black')
plt.title('Histogram of Present_SOLO_R')
plt.xlabel('Present_SOLO_R')
plt.ylabel('Frequency')
plt.grid(False)
plt.show()
```



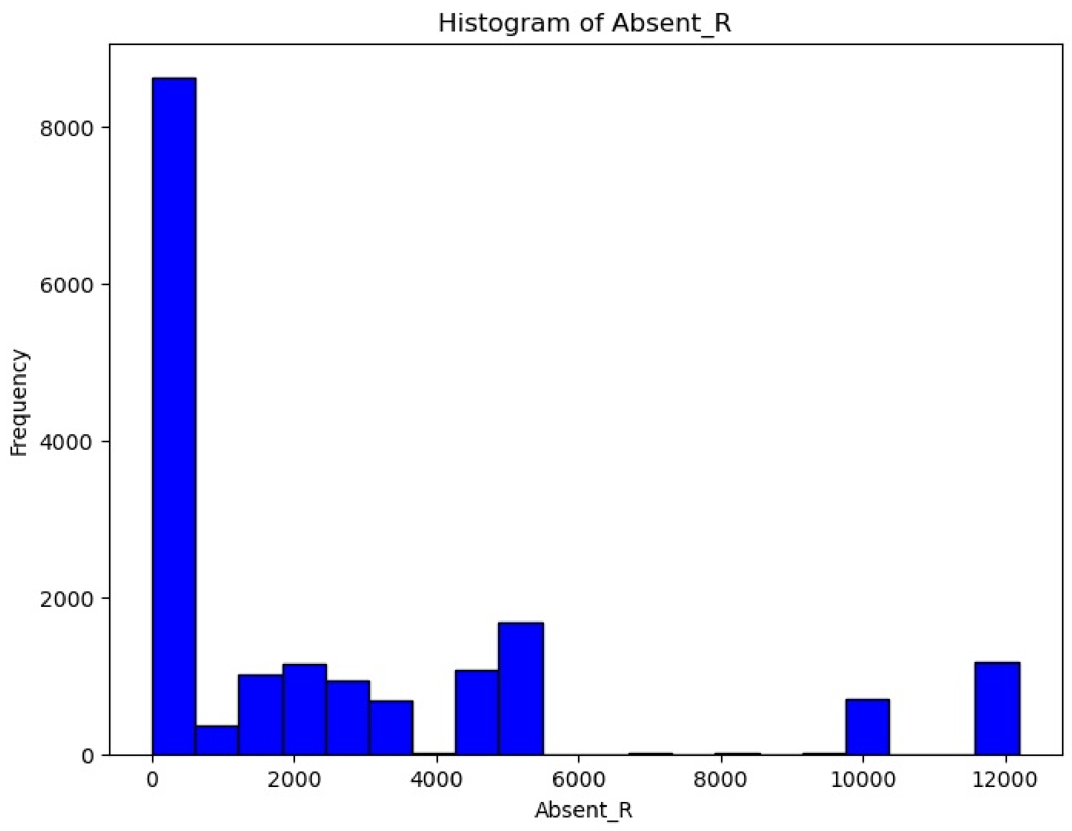
In [23]:
```
plt.figure(figsize=(8, 6))
plt.hist(df['Present_SOLO_SR'], bins=20, color='red', edgecolor='black')
plt.title('Histogram of Present_SOLO_SR')
plt.xlabel('Present_SOLO_SR')
plt.ylabel('Frequency')
plt.grid(False)
plt.show()
```



In [24]:
```
plt.figure(figsize=(8, 6))
plt.hist(df['Present_S'], bins=20, color='yellow', edgecolor='black')
plt.title('Histogram of Present_S')
plt.xlabel('Present_S')
plt.ylabel('Frequency')
plt.grid(False)
plt.show()
```



In [25]:
```
plt.figure(figsize=(8, 6))
plt.hist(df['Absent_R'], bins=20, color='blue', edgecolor='black')
plt.title('Histogram of Absent_R')
plt.xlabel('Absent_R')
plt.ylabel('Frequency')
plt.grid(False)
plt.show()
```



In [26]:
```
#  unique values and their counts in the 'Absent_R' feature
unique_values_counts = df['Absent_R'].value_counts()

unique_values_counts
```

Out[26]:
```
4900.0     1405
480.0      1173
144.0       991
321.0       964
229.0       932
           ...
12147.0       1
12145.0       1
12142.0       1
12139.0       1
2543.0        1
Name: Absent_R, Length: 535, dtype: int64
```

In [ ]:
```
import numpy as np
 Replace infinity values with a very large number
df.replace([np.inf, -np.inf], np.finfo(np.float64).max, inplace=True)
```

In [ ]:
```
#from sklearn.preprocessing import OneHotEncoder

# Select categorical columns for one-hot encoding
#categorical_cols = ['Drug', 'Common Variant', 'INITIAL CONFIDENCE GRADING', 'FINAL CONFIDENCE GRADING']

# Perform one-hot encoding
#encoded_df = pd.get_dummies(df, columns=categorical_cols)

# Display the encoded DataFrame
#print(encoded_df)
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

# LabelEncoder
encoder = LabelEncoder()

#Encode categorical columns
categorical_cols = ['Drug', 'Common Variant', 'INITIAL CONFIDENCE GRADING', 'FINAL CONFIDENCE GRADING']
df[categorical_cols] = df[categorical_cols].apply(encoder.fit_transform)
```

In [ ]:
```
#import pandas as pd
#from sklearn.preprocessing import OneHotEncoder

# Assuming X_encoded contains the one-hot encoded features and y contains the target variable
# Concatenate X_encoded with the target variable y
#encoded_df = pd.concat([X_encoded, y], axis=1)

# Now encoded_df contains the one-hot encoded features along with the target variable
# You can use encoded_df for further analysis, modeling, etc.
```

In [57]:
```
df.head()
```

Out[57]:

	Drug	Tier	Common Variant	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R	Neutral masked	INITIAL CONFIDENCE GRADING	FI CONFIDE GRAD
0	0	0.0	11986	0.128571	0.133776	0.005898	0.606014	0.099019	0.028618	0.0	0	
1	0	0.0	2333	0.003922	0.010663	0.006016	0.283827	0.003374	0.051825	0.0	0	
2	0	0.0	12271	0.000700	0.000969	0.000236	0.607874	0.000633	0.105125	0.0	0	
3	0	0.0	12111	0.000560	0.001800	0.001180	0.607564	0.000527	0.105207	0.0	4	
4	0	1.0	12971	0.000420	0.013848	0.011443	0.604192	0.000316	0.011562	0.0	4	

In [58]:
```
df.tail()
```

Out[58]:

	Drug	Tier	Common Variant	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R	Neutral masked	INITIAL CONFIDENCE GRADING	CON C
17391	14	0.0	13290	0.00028	0.036837	0.033620	0.240081	0.002109	0.208446	1.0	2	
17392	14	0.0	12149	0.00000	0.000000	0.042350	0.237213	0.002004	0.208528	1.0	2	
17393	14	1.0	11603	0.00000	0.000000	0.005898	0.249186	0.000000	0.006970	1.0	2	
17394	14	1.0	12784	0.00000	0.000000	0.007432	0.248683	0.000000	0.006970	1.0	2	
17395	14	NaN	11566	0.00000	0.000000	0.000000	0.362252	0.000844	0.379418	0.0	5	

In [59]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17396 entries, 0 to 17395
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Drug                        17396 non-null  int32  
 1   Tier                        17348 non-null  float64
 2   Common Variant              17396 non-null  int32  
 3   Present_SOLO_R              17388 non-null  float64
 4   Present_SOLO_SR             17388 non-null  float64
 5   Present_S                   17388 non-null  float64
 6   Absent_S                    17388 non-null  float64
 7   Present_R                   17388 non-null  float64
 8   Absent_R                    17388 non-null  float64
 9   Neutral masked              17396 non-null  float64
 10  INITIAL CONFIDENCE GRADING  17396 non-null  int32  
 11  FINAL CONFIDENCE GRADING    17396 non-null  int32  
dtypes: float64(8), int32(4)
memory usage: 1.3 MB
```

In [63]:
```
df.isnull().sum()
```

Out[63]:
```
Drug                           0
Tier                          48
Common Variant                 0
Present_SOLO_R                 8
Present_SOLO_SR                8
Present_S                      8
Absent_S                       8
Present_R                      8
Absent_R                       8
Neutral masked                 0
INITIAL CONFIDENCE GRADING     0
FINAL CONFIDENCE GRADING       0
dtype: int64
```

In [60]:
```
df.dtypes
```

Out[60]:
```
Drug                            int32
Tier                          float64
Common Variant                  int32
Present_SOLO_R                float64
Present_SOLO_SR               float64
Present_S                     float64
Absent_S                      float64
Present_R                     float64
Absent_R                      float64
Neutral masked                float64
INITIAL CONFIDENCE GRADING      int32
FINAL CONFIDENCE GRADING        int32
dtype: object
```

In [61]:
```
df.describe()
```

Out[61]:

	Drug	Tier	Common Variant	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_
count	17396.000000	17348.000000	17396.00000	17388.000000	17388.000000	17388.000000	17388.000000	17388.000000	17388.0000
mean	7.132847	0.415610	6947.03863	0.000279	0.000954	0.001940	0.644772	0.000476	0.2146
std	4.349835	0.492841	4133.50626	0.010180	0.016847	0.028009	0.248664	0.011424	0.2840
min	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	5.000000	0.000000	3183.75000	0.000000	0.000000	0.000118	0.422117	0.000000	0.0168
50%	7.000000	0.000000	7145.00000	0.000000	0.000000	0.000118	0.587725	0.000000	0.0794
75%	12.000000	1.000000	10673.25000	0.000000	0.000138	0.000236	0.861942	0.000000	0.3799
max	14.000000	1.000000	13448.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000

In [64]:
```
for feature in df.columns:
    median_value = df[feature].median()
    df[feature].fillna(median_value, inplace=True)
```

In [65]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17396 entries, 0 to 17395
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Drug                        17396 non-null  int32  
 1   Tier                        17396 non-null  float64
 2   Common Variant              17396 non-null  int32  
 3   Present_SOLO_R              17396 non-null  float64
 4   Present_SOLO_SR             17396 non-null  float64
 5   Present_S                   17396 non-null  float64
 6   Absent_S                    17396 non-null  float64
 7   Present_R                   17396 non-null  float64
 8   Absent_R                    17396 non-null  float64
 9   Neutral masked              17396 non-null  float64
 10  INITIAL CONFIDENCE GRADING  17396 non-null  int32  
 11  FINAL CONFIDENCE GRADING    17396 non-null  int32  
dtypes: float64(8), int32(4)
memory usage: 1.3 MB
```

In [67]:
```
df.corr()
```

Out[67]:

	Drug	Tier	Common Variant	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R	Neutr mask
Drug	1.000000	-0.093594	0.107236	0.012414	0.012896	-0.000168	0.036932	0.017281	0.231247	0.0000
Tier	-0.093594	1.000000	-0.076048	-0.021700	-0.007533	0.018833	0.100121	-0.031870	-0.569943	0.0280
Common Variant	0.107236	-0.076048	1.000000	0.010508	0.007713	0.007509	-0.179507	0.007627	0.030117	0.0074
Present_SOLO_R	0.012414	-0.021700	0.010508	1.000000	0.631376	0.020307	0.008458	0.871117	0.006031	0.0017
Present_SOLO_SR	0.012896	-0.007533	0.007713	0.631376	1.000000	0.413592	-0.022704	0.668213	-0.006268	0.2072
Present_S	-0.000168	0.018833	0.007509	0.020307	0.413592	1.000000	-0.085283	0.299541	-0.031625	0.4806
Absent_S	0.036932	0.100121	-0.179507	0.008458	-0.022704	-0.085283	1.000000	-0.008724	0.373310	-0.0847
Present_R	0.017281	-0.031870	0.007627	0.871117	0.668213	0.299541	-0.008724	1.000000	0.013122	0.1075
Absent_R	0.231247	-0.569943	0.030117	0.006031	-0.006268	-0.031625	0.373310	0.013122	1.000000	-0.0436
Neutral masked	0.000060	0.028058	0.007443	0.001739	0.207243	0.480660	-0.084763	0.107510	-0.043654	1.0000
INITIAL CONFIDENCE GRADING	-0.100572	0.089321	-0.064167	-0.175689	-0.200729	-0.197813	0.075625	-0.191941	0.005752	-0.3966
FINAL CONFIDENCE GRADING	-0.117773	0.162170	-0.048383	-0.112212	0.040371	0.265929	0.023106	-0.030491	-0.096255	0.5498

In [ ]:

In [68]:

```python
from sklearn.preprocessing import MinMaxScaler

# numerical columns for  normalizing
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Min-Max scaling (scaling features between 0 and 1)
min_max_scaler = MinMaxScaler()
df[numerical_columns] = min_max_scaler.fit_transform(df[numerical_columns])
```

In [69]:

```python
df[numerical_columns]
```

Out[69]:

	Tier	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R	Neutral masked
0	0.0	0.128571	0.133776	0.005898	0.606014	0.099019	0.028618	0.0
1	0.0	0.003922	0.010663	0.006016	0.283827	0.003374	0.051825	0.0
2	0.0	0.000700	0.000969	0.000236	0.607874	0.000633	0.105125	0.0
3	0.0	0.000560	0.001800	0.001180	0.607564	0.000527	0.105207	0.0
4	1.0	0.000420	0.013848	0.011443	0.604192	0.000316	0.011562	0.0
...	...	...	...	...	...	...	...	...
17391	0.0	0.000280	0.036837	0.033620	0.240081	0.002109	0.208446	1.0
17392	0.0	0.000000	0.000000	0.042350	0.237213	0.002004	0.208528	1.0
17393	1.0	0.000000	0.000000	0.005898	0.249186	0.000000	0.006970	1.0
17394	1.0	0.000000	0.000000	0.007432	0.248683	0.000000	0.006970	1.0
17395	0.0	0.000000	0.000000	0.000000	0.362252	0.000844	0.379418	0.0

17396 rows × 8 columns

In [70]:

```python
# Selecting categorical columns
categorical_cols = df[['Drug', 'Common Variant', 'INITIAL CONFIDENCE GRADING', 'FINAL CONFIDENCE GRADING']]

# Selecting numerical columns
numerical_columns = df[['Tier', 'Present_SOLO_R', 'Present_SOLO_SR',
       'Present_S', 'Absent_S', 'Present_R', 'Absent_R', 'Neutral masked']]

# Concatenate along columns(axis=1) to merge them
df_merged = pd.concat([categorical_cols, numerical_columns], axis=1)

# Print the merged DataFrame
df_merged
```

Out[70]:

	Drug	Common Variant	INITIAL CONFIDENCE GRADING	FINAL CONFIDENCE GRADING	Tier	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R
0	0	11986	0	0	0.0	0.128571	0.133776	0.005898	0.606014	0.099019	0.028618
1	0	2333	0	0	0.0	0.003922	0.010663	0.006016	0.283827	0.003374	0.051825
2	0	12271	0	1	0.0	0.000700	0.000969	0.000236	0.607874	0.000633	0.105125
3	0	12111	4	1	0.0	0.000560	0.001800	0.001180	0.607564	0.000527	0.105207
4	0	12971	4	2	1.0	0.000420	0.013848	0.011443	0.604192	0.000316	0.011562
...	...	...	...	...	...	...	...	...	...	...	...
17391	14	13290	2	4	0.0	0.000280	0.036837	0.033620	0.240081	0.002109	0.208446
17392	14	12149	2	4	0.0	0.000000	0.000000	0.042350	0.237213	0.002004	0.208528
17393	14	11603	2	4	1.0	0.000000	0.000000	0.005898	0.249186	0.000000	0.006970
17394	14	12784	2	4	1.0	0.000000	0.000000	0.007432	0.248683	0.000000	0.006970
17395	14	11566	5	5	0.0	0.000000	0.000000	0.000000	0.362252	0.000844	0.379418

17396 rows × 12 columns

In [71]:

```
#from sklearn.preprocessing import OneHotEncoder

# Initialize the OneHotEncoder
#encoder = OneHotEncoder()

# Fit and transform the categorical columns
#categorical_cols = ['Drug', 'Common Variant', 'INITIAL CONFIDENCE GRADING', 'FINAL CONFIDENCE GRADING']
#encoded_data = encoder.fit_transform(df[categorical_cols])

# Convert the encoded data to a DataFrame
#encoded_df = pd.DataFrame(encoded_data.toarray(), columns=encoder.get_feature_names_out(categorical_cols))

# Drop the original categorical columns from the DataFrame
#df.drop(columns=categorical_cols, inplace=True)

# Concatenate the encoded DataFrame with the original DataFrame
#df_encoded = pd.concat([df, encoded_df], axis=1)
```

In [73]:

```
# Swap the values of "final confidence" and "initial confidence" in the last row
last_index = df.index[-1]  # Get the index of the last row
final_confidence = df.at[last_index, 'FINAL CONFIDENCE GRADING']
initial_confidence = df.at[last_index, 'INITIAL CONFIDENCE GRADING']

# Swap the values
df.at[last_index, 'INITIAL CONFIDENCE GRADING'] = final_confidence
df.at[last_index, 'FINAL CONFIDENCE GRADING'] = initial_confidence

# Print the DataFrame to verify the changes
print(df)
```

```
       Drug  Tier  Common Variant  Present_SOLO_R  Present_SOLO_SR  Present_S  \
0         0   0.0           11986        0.128571         0.133776   0.005898   
1         0   0.0            2333        0.003922         0.010663   0.006016   
2         0   0.0           12271        0.000700         0.000969   0.000236   
3         0   0.0           12111        0.000560         0.001800   0.001180   
4         0   1.0           12971        0.000420         0.013848   0.011443   
...     ...   ...             ...             ...              ...        ...   
17391    14   0.0           13290        0.000280         0.036837   0.033620   
17392    14   0.0           12149        0.000000         0.000000   0.042350   
17393    14   1.0           11603        0.000000         0.000000   0.005898   
17394    14   1.0           12784        0.000000         0.000000   0.007432   
17395    14   0.0           11566        0.000000         0.000000   0.000000   

       Absent_S  Present_R  Absent_R  Neutral masked  \
0      0.606014   0.099019  0.028618             0.0   
1      0.283827   0.003374  0.051825             0.0   
2      0.607874   0.000633  0.105125             0.0   
3      0.607564   0.000527  0.105207             0.0   
4      0.604192   0.000316  0.011562             0.0   
...         ...        ...       ...             ...   
17391  0.240081   0.002109  0.208446             1.0   
17392  0.237213   0.002004  0.208528             1.0   
17393  0.249186   0.000000  0.006970             1.0   
17394  0.248683   0.000000  0.006970             1.0   
17395  0.362252   0.000844  0.379418             0.0   

       INITIAL CONFIDENCE GRADING  FINAL CONFIDENCE GRADING  
0                               0                         0  
1                               0                         0  
2                               0                         1  
3                               4                         1  
4                               4                         2  
...                           ...                       ...  
17391                           2                         4  
17392                           2                         4  
17393                           2                         4  
17394                           2                         4  
17395                           5                         5  

[17396 rows x 12 columns]
```

In [41]:
```python
df.head()
```

Out[41]:

	Drug	Tier	Common Variant	Present_SOLO_R	Present_SOLO_SR	Present_S	Absent_S	Present_R	Absent_R	Neutral masked	INITIAL CONFIDENCE GRADING	FI CONFIDE GRAD
0	0	0.0	11986	0.128571	0.133776	0.005898	0.606014	0.099019	0.028618	0.0	0	
1	0	0.0	2333	0.003922	0.010663	0.006016	0.283827	0.003374	0.051825	0.0	0	
2	0	0.0	12271	0.000700	0.000969	0.000236	0.607874	0.000633	0.105125	0.0	0	
3	0	0.0	12111	0.000560	0.001800	0.001180	0.607564	0.000527	0.105207	0.0	4	
4	0	1.0	12971	0.000420	0.013848	0.011443	0.604192	0.000316	0.011562	0.0	4	

In [74]:
```python
# Check for specific values in the 'Drug' column
drug_counts = df['Drug'].value_counts()
print("Counts for each drug:")
print(drug_counts)
```

```
Counts for each drug:
5     2701
7     2430
13    1714
0     1571
14    1522
6     1438
12    1144
2     1067
9      703
8      615
3      608
11     580
1      537
10     406
4      360
Name: Drug, dtype: int64
```

In [75]:
```python
# Check for specific values in the 'Tier' column
tier_counts = df['Tier'].value_counts()
print("\nCounts for each tier:")
print(tier_counts)
```

```
Counts for each tier:
0.0    10186
1.0     7210
Name: Tier, dtype: int64
```

```
In [76]: INITIAL_CONFIDENCE_GRADING = df['INITIAL CONFIDENCE GRADING'].value_counts()

         # Print the counts for INITIAL CONFIDENCE GRADING
         print("Counts for INITIAL CONFIDENCE GRADING:")
         print(INITIAL_CONFIDENCE_GRADING)
```

```
Counts for INITIAL CONFIDENCE GRADING:
4    16805
0      226
2      222
1       88
5       40
3       15
Name: INITIAL CONFIDENCE GRADING, dtype: int64
```

```
In [77]: FINAL_CONFIDENCE_GRADING = df['FINAL CONFIDENCE GRADING'].value_counts()

         # Print the counts for INITIAL CONFIDENCE GRADING
         print("Counts for FINAL CONFIDENCE GRADING:")
         print(FINAL_CONFIDENCE_GRADING)
```

```
Counts for FINAL CONFIDENCE GRADING:
2    15910
1     1004
4      213
0      196
5       40
3       33
Name: FINAL CONFIDENCE GRADING, dtype: int64
```

```
In [78]: from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score

         # Step 1: Data Preprocessing
         X = df.drop(columns=['FINAL CONFIDENCE GRADING'])
         y = df['FINAL CONFIDENCE GRADING']
```

```
In [81]: # Splitting the dataset into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

         # Step 2: Model Selection and Training
         model = RandomForestClassifier()  # You can use any other classifier here
         model.fit(X_train, y_train)
         # Step 3: Model Evaluation
         y_pred = model.predict(X_test)
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy:", accuracy)
```

```
Accuracy: 0.9804597701149426
```

```
In [82]: from sklearn.metrics import classification_report


         # Here, y_test and y_pred are the true labels and predicted labels respectively

         # Print classification report
         print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        46
           1       0.84      0.84      0.84       203
           2       0.99      0.99      0.99      3180
           3       1.00      0.75      0.86         4
           4       1.00      1.00      1.00        40
           5       1.00      0.71      0.83         7

    accuracy                           0.98      3480
   macro avg       0.96      0.88      0.91      3480
weighted avg       0.98      0.98      0.98      3480
```

```
In [83]: from sklearn.metrics import accuracy_score

         # Assuming you have already trained your model and obtained predictions (y_pred) on the testing set (X_test)
         # Here, model is your trained classifier

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

         # Calculate accuracy on the testing set
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy on testing set:", accuracy)
```

```
Accuracy on testing set: 0.9804597701149426
```

```
In [88]: from sklearn.ensemble import RandomForestClassifier

         # X_train and y_train are the training features and labels
```

```python
rf_classifier = RandomForestClassifier()

# Fit the model
rf_classifier.fit(X_train, y_train)

# Get feature importances
feature_importances = rf_classifier.feature_importances_
```

In [85]:
```python
# Match feature importances with feature names
feature_importance_dict = dict(zip(X_train.columns, feature_importances))

# Sort feature importances in descending order
sorted_feature_importances = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

# Print feature importances
for feature, importance in sorted_feature_importances:
    print(f"Feature: {feature}, Importance: {importance}")
```

```
Feature: Common Variant, Importance: 0.3616054767513026
Feature: INITIAL CONFIDENCE GRADING, Importance: 0.18772829410611908
Feature: Present_SOLO_R, Importance: 0.09160159071097614
Feature: Absent_R, Importance: 0.07203442353458094
Feature: Present_R, Importance: 0.06035632406647119
Feature: Present_S, Importance: 0.05831126068993332
Feature: Absent_S, Importance: 0.05264770211189792
Feature: Neutral masked, Importance: 0.04520675704198554
Feature: Present_SOLO_SR, Importance: 0.032340385919536425
Feature: Drug, Importance: 0.031238420898142365
Feature: Tier, Importance: 0.006929364169054611
```

In [91]:
```python
# Select the most important features identified
important_features = ['Common Variant', 'INITIAL CONFIDENCE GRADING' ,
                      'Present_SOLO_R', 'Neutral masked','Present_R','Present_S', 'Absent_R', 'Absent_S']

# Analyze the relationship between each important feature and final confidence gradings
for feature in important_features:
    # Group the data by the feature and calculate the mean final confidence grading for each group
    feature_confidence_mean = df.groupby(feature)['FINAL CONFIDENCE GRADING'].mean()

    # Print the relationship between the feature and final confidence gradings
    print(f"\nRelationship between '{feature}' and Final Confidence Gradings:\n")
    print(feature_confidence_mean)
```

```
Relationship between 'Common Variant' and Final Confidence Gradings:

Common Variant
0        2.0
1        2.0
2        2.0
3        2.0
4        2.0
        ...
13444    2.0
13445    2.0
13446    2.0
13447    2.0
13448    2.0
Name: FINAL CONFIDENCE GRADING, Length: 13449, dtype: float64

Relationship between 'INITIAL CONFIDENCE GRADING' and Final Confidence Gradings:

INITIAL CONFIDENCE GRADING
0    0.141593
1    1.000000
2    3.959459
3    3.000000
4    1.947694
5    5.000000
Name: FINAL CONFIDENCE GRADING, dtype: float64

Relationship between 'Present_SOLO_R' and Final Confidence Gradings:

Present_SOLO_R
0.000000    2.003617
0.000140    1.714411
0.000280    1.629771
0.000420    1.657895
0.000560    1.633803
              ...
0.158683    0.000000
0.216246    0.000000
0.243417    0.000000
0.746779    0.000000
1.000000    0.000000
Name: FINAL CONFIDENCE GRADING, Length: 92, dtype: float64

Relationship between 'Neutral masked' and Final Confidence Gradings:
```

```
Neutral masked
0.0    1.927099
1.0    3.959459
Name: FINAL CONFIDENCE GRADING, dtype: float64

Relationship between 'Present_R' and Final Confidence Gradings:

Present_R
0.000000    1.995467
0.000105    1.805730
0.000211    1.788214
0.000316    1.864035
0.000422    1.803922
              ...
0.236739    0.000000
0.241485    0.000000
0.540124    4.000000
0.689233    0.000000
1.000000    0.000000
Name: FINAL CONFIDENCE GRADING, Length: 152, dtype: float64

Relationship between 'Present_S' and Final Confidence Gradings:

Present_S
0.000000    1.712017
0.000118    1.967196
0.000236    1.970423
0.000354    1.954918
0.000472    1.952278
              ...
0.908104    4.000000
0.908458    4.000000
0.976997    4.000000
0.984664    4.000000
1.000000    4.000000
Name: FINAL CONFIDENCE GRADING, Length: 253, dtype: float64

Relationship between 'Absent_R' and Final Confidence Gradings:

Absent_R
0.000000    4.0
0.000082    4.0
0.000246    4.0
0.000574    4.0
0.000738    4.0
           ...
0.999508    5.0
0.999672    5.0
0.999754    5.0
0.999918    5.0
1.000000    5.0
Name: FINAL CONFIDENCE GRADING, Length: 535, dtype: float64

Relationship between 'Absent_S' and Final Confidence Gradings:

Absent_S
0.000000    4.000000
0.000116    4.000000
0.000155    4.000000
0.000271    4.000000
0.000426    4.000000
              ...
0.999845    2.000000
0.999884    1.987421
0.999923    2.000000
0.999961    2.000000
1.000000    2.021898
Name: FINAL CONFIDENCE GRADING, Length: 748, dtype: float64
```

In [92]:

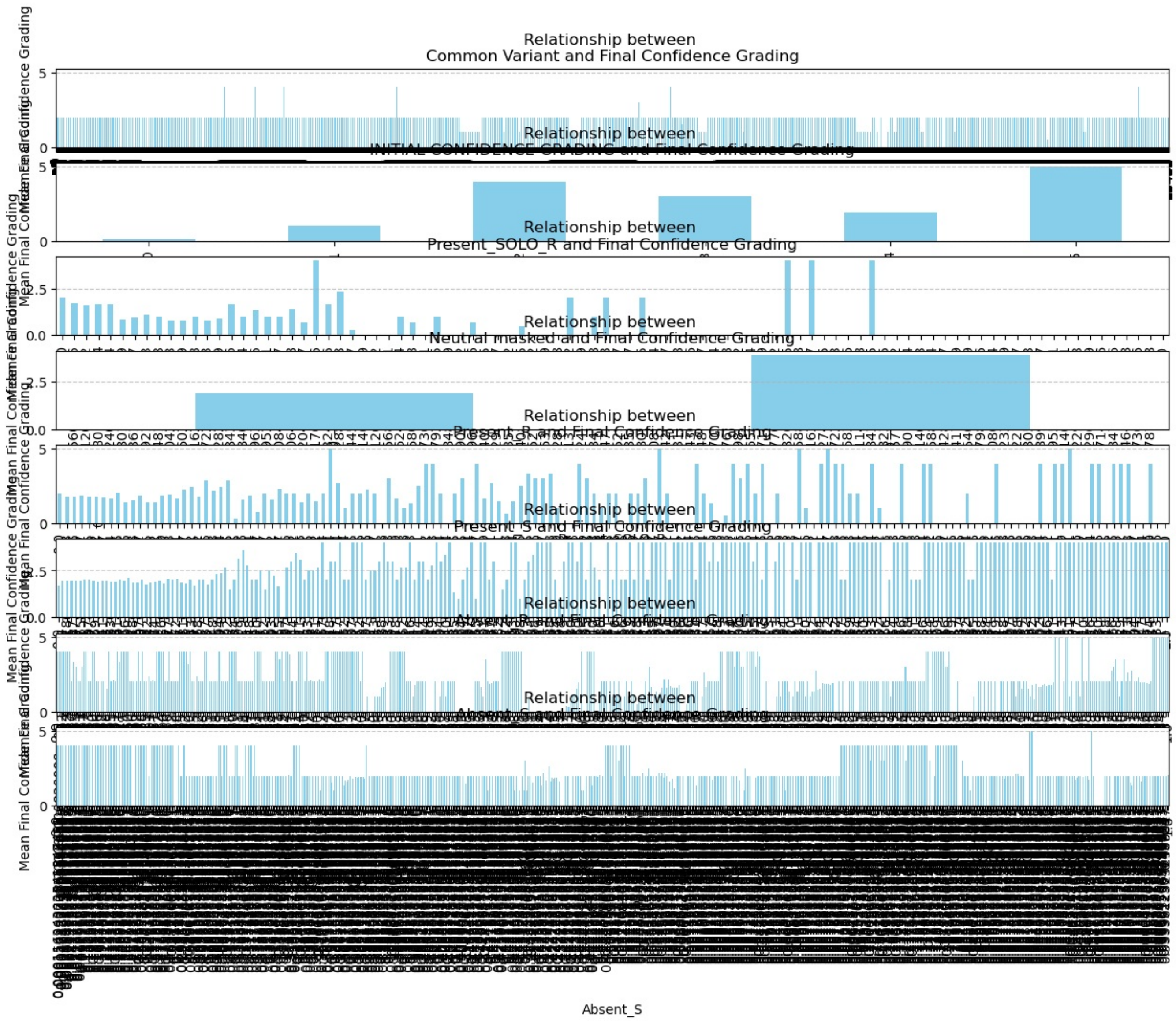
```python
from scipy import stats

# Select the most important features identified
important_features = ['Common Variant', 'INITIAL CONFIDENCE GRADING' ,
                      'Present_SOLO_R', 'Neutral masked','Present_R','Present_S', 'Absent_R', 'Absent_S']

# Analyze the relationship between each important feature and final confidence gradings
for feature in important_features:
    # Find pointbserialr correlation coefficient
    correlation_coefficient, p_value = stats.pointbiserialr(df[feature], df['FINAL CONFIDENCE GRADING'])

    # Print the results
    print(f"\nPoint-biserial correlation coefficient between '{feature}' and 'FINAL CONFIDENCE GRADING': {corre
    print(f"P-value: {p_value}")
```

```
Point-biserial correlation coefficient between 'Common Variant' and 'FINAL CONFIDENCE GRADING': -0.048382506278
49848
P-value: 1.7180372270874056e-10

Point-biserial correlation coefficient between 'INITIAL CONFIDENCE GRADING' and 'FINAL CONFIDENCE GRADING': 0.2
797176016094426
P-value: 4.0578401675859e-310

Point-biserial correlation coefficient between 'Present_SOLO_R' and 'FINAL CONFIDENCE GRADING': -0.112212379221
96936
P-value: 7.387710645132828e-50

Point-biserial correlation coefficient between 'Neutral masked' and 'FINAL CONFIDENCE GRADING': 0.5498151025126
343
P-value: 0.0

Point-biserial correlation coefficient between 'Present_R' and 'FINAL CONFIDENCE GRADING': -0.03049148049458477
P-value: 5.763592431595842e-05

Point-biserial correlation coefficient between 'Present_S' and 'FINAL CONFIDENCE GRADING': 0.26592890781587236
P-value: 2.162312346552411e-279

Point-biserial correlation coefficient between 'Absent_R' and 'FINAL CONFIDENCE GRADING': -0.0962551796587794
P-value: 4.343903697446709e-37

Point-biserial correlation coefficient between 'Absent_S' and 'FINAL CONFIDENCE GRADING': 0.02310558389647338
P-value: 0.0023062047405744313
```

In [93]:

```python
import matplotlib.pyplot as plt

# Select the most important features identified
important_features = ['Common Variant', 'INITIAL CONFIDENCE GRADING' ,
                      'Present_SOLO_R', 'Neutral masked','Present_R','Present_S', 'Absent_R', 'Absent_S']

# Set up subplots for each feature
fig, axes = plt.subplots(nrows=len(important_features), figsize=(15, 10))

# Analyze the relationship between each important feature and final confidence gradings
for i, feature in enumerate(important_features):
    # Group the data by the feature and calculate the mean final confidence grading for each group
    feature_confidence_mean = df.groupby(feature)['FINAL CONFIDENCE GRADING'].mean()

    # Plot the bar plot
    feature_confidence_mean.plot(kind='bar', ax=axes[i], color='skyblue')
    axes[i].set_title(f'Relationship between \n{feature} and Final Confidence Grading')
    axes[i].set_ylabel('Mean Final Confidence Grading')
    axes[i].set_xlabel(feature)
    axes[i].grid(axis='y', linestyle='--', alpha=0.95)

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

```
C:\Users\hp\AppData\Local\Temp\ipykernel_12104\3065845851.py:22: UserWarning: Tight layout not applied. tight_l
ayout cannot make axes height small enough to accommodate all axes decorations.
  plt.tight_layout()
```



```
In [ ]:
```

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js