In [1]:
```
from IPython.display import Image, display

# Specify the path to your image file
image_path = r'C:\Users\jangi\Downloads\Screenshot 2024-05-15 200823.png'

# Display the image
display(Image(filename=image_path))
```

The accumulation of term deposits made in the accounts of the West African Economic and Monetary Union (WAEMU) banks decreased by 15.1% at the end of 2018 compared to 2017, according to data from the Central Bank. West African States (BCEAO).

Insight of this dataset:

1. Predicting Financial Health or Bankruptcy (e.g., using Zscore):

- Logistic Regression: Suitable for binary classification (e.g., predicting whether a bank will go bankrupt).
- Random Forest: Effective for classification problems and can handle non-linear relationships.
- Gradient Boosting (e.g., XGBoost): Provides high performance for classification tasks and handles various complexities in the data.
- Neural Networks: Useful for complex patterns but requires more data and computational power.

2. Assessing Profitability and Sustainability (PS):

- Linear Regression: Simple and interpretable, good for predicting continuous outcomes.
- Random Forest Regression: Handles non-linear relationships and provides variable importance measures.
- Gradient Boosting Regression: Offers high accuracy and can model complex patterns.
- Neural Networks: Suitable for capturing complex dependencies in large datasets.

3. Solvency and Financial Stability (SFS):

- Linear Regression or Logistic Regression: Depending on whether SFS is a continuous or binary variable.
- Support Vector Machines (SVM): Effective for both regression and classification, especially in high-dimensional spaces.
- Random Forest/Gradient Boosting: Versatile and effective for both regression and classification.

4. Liquidity Risk (RL):

- Logistic Regression: If predicting the probability of high liquidity risk.
- Random Forest/Gradient Boosting: Useful for classification tasks and handling various feature types.
- Time Series Analysis (e.g., ARIMA, LSTM): If liquidity risk is being predicted over time.

5. Capital Adequacy (CC):

- Linear Regression: For predicting continuous capital adequacy ratios.
- Random Forest/Gradient Boosting Regression: Effective for non-linear relationships.
- Support Vector Regression (SVR): Good for continuous predictions with non-linear relationships.
- Algorithm Selection Based on Data Characteristics
- Logistic Regression: Best for binary outcomes and interpretability.
- Random Forest/Gradient Boosting: Good for handling non-linear relationships and feature interactions.
- Neural Networks: Suitable for large datasets with complex patterns but require more computational resources.
- Time Series Models: Necessary if predicting variables over time.
- Example Workflow for Predicting Financial Health (using Zscore)

Step-by-Step Process

- Import Libraries

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClass
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

```

In [3]:
```
df = pd.read_csv(r"WAEMU_Banking.csv")
```

In [4]:
```
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
```

In [5]:
```
df.head()
```

Out[5]:

	Countries_Num	id	Countries	Banks	Year	RIR	SFS	INF	ERA	
0	1	1	2	13	2013	3.836593	26.861971	0.428889	3.196428	12.076
1	1	1	2	13	2014	5.599992	29.965430	-0.548758	3.045024	8.884
2	1	1	2	13	2015	4.266334	30.984761	0.218786	2.394557	8.583
3	1	1	2	13	2016	4.580100	29.832095	-0.794050	3.712403	5.720
4	1	1	2	13	2017	7.329021	28.630991	1.769412	3.833422	6.256

Data Preprocessing:

- Handle missing values.
- Normalize/standardize numerical features.
- Encode categorical variables.

Feature Selection/Engineering:

- Select relevant features that contribute to the target variable.
- Create new features if necessary (e.g., ratios, interaction terms).

Model Selection:

- Choose an appropriate algorithm (e.g., Random Forest, Gradient Boosting).

Model Training and Evaluation:

- Split data into training and testing sets.
- Train the model on the training set.
- Evaluate performance using metrics like accuracy, precision, recall, F1 score (for classification) or RMSE, MAE (for regression).

Hyperparameter Tuning:

- Use techniques like Grid Search or Random Search to find the best hyperparameters.

Model Validation:

- Validate the model using cross-validation to ensure it generalizes well to unseen data.

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 742 entries, 0 to 741
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Countries_Num  742 non-null    int64  
 1   id             742 non-null    int64  
 2   Countries      742 non-null    int64  
 3   Banks          742 non-null    int64  
 4   Year           742 non-null    int64  
 5   RIR            742 non-null    float64
 6   SFS            742 non-null    float64
 7   INF            742 non-null    float64
 8   ERA            742 non-null    float64
 9   INL            742 non-null    float64
 10  Zscore         742 non-null    float64
 11  DEBT           742 non-null    float64
 12  SIZE           742 non-null    float64
 13  CC             742 non-null    float64
 14  GE             742 non-null    float64
 15  PS             742 non-null    float64
 16  RQ             742 non-null    float64
 17  RL             742 non-null    float64
 18  VA             742 non-null    float64
dtypes: float64(14), int64(5)
memory usage: 110.3 KB
```

In [7]:
```
df.shape
```

Out[7]:
```
(742, 19)
```

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

Out[8]:
```
Countries_Num    0
id               0
Countries        0
Banks            0
Year             0
RIR              0
SFS              0
INF              0
ERA              0
INL              0
Zscore           0
DEBT             0
SIZE             0
CC               0
GE               0
PS               0
RQ               0
RL               0
VA               0
dtype: int64
```

In [9]:
```
df.describe().T.style.background_gradient(cmap = 'Reds')
```

Out[9]:

	count	mean	std	min	25%	50%	
Countries_Num	742.000000	4.575472	2.316191	1.000000	3.000000	5.000000	
id	742.000000	53.500000	30.618842	1.000000	27.000000	53.500000	8
Countries	742.000000	3.650943	2.380928	0.000000	2.000000	3.000000	
Banks	742.000000	48.716981	27.703134	0.000000	26.000000	47.500000	7
Year	742.000000	2016.000000	2.001349	2013.000000	2014.000000	2016.000000	201
RIR	742.000000	3.711618	4.114070	-23.137938	3.367054	4.266334	
SFS	742.000000	31.965249	8.239127	15.829639	26.927042	29.918364	3
INF	742.000000	0.524620	1.269270	-3.233389	-0.258090	0.685881	
ERA	742.000000	9.766718	18.965490	-179.747455	4.166364	7.460494	1
INL	742.000000	11.652364	10.885343	0.000000	4.656050	8.895584	1
Zscore	742.000000	2.967964	5.123174	-47.777093	1.482417	2.149078	
DEBT	742.000000	38.780738	11.664036	18.503746	30.666445	36.494147	4
SIZE	742.000000	12.077644	1.114687	8.677440	11.359607	12.156868	1
CC	742.000000	37.292780	12.589451	13.461540	27.403850	33.173080	4
GE	742.000000	26.747784	10.261396	8.530806	18.009480	26.442310	3
PS	742.000000	23.891205	14.741995	3.773585	12.857140	17.061610	3
RQ	742.000000	35.432019	9.474103	13.942310	28.365390	34.134620	4
RL	742.000000	32.653598	10.138651	6.250000	27.403850	30.769230	3
VA	742.000000	41.074591	11.071831	18.309860	33.004920	38.423650	5

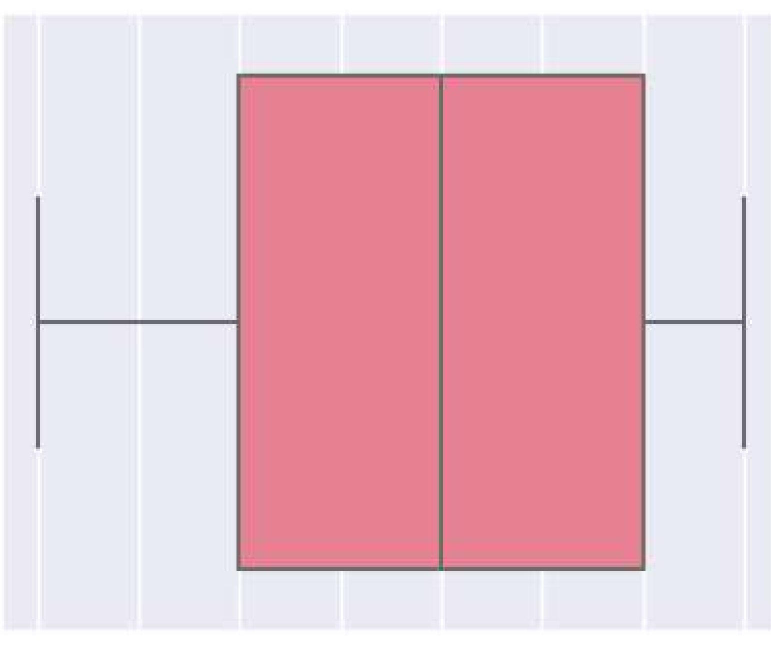
In [10]:
```
df.duplicated()
```

Out[10]:
```
0      False
1      False
2      False
3      False
4      False
       ...
737    False
738    False
739    False
740    False
741    False
Length: 742, dtype: bool
```

In [11]:
```
df.nunique().sort_values(ascending=True)
```

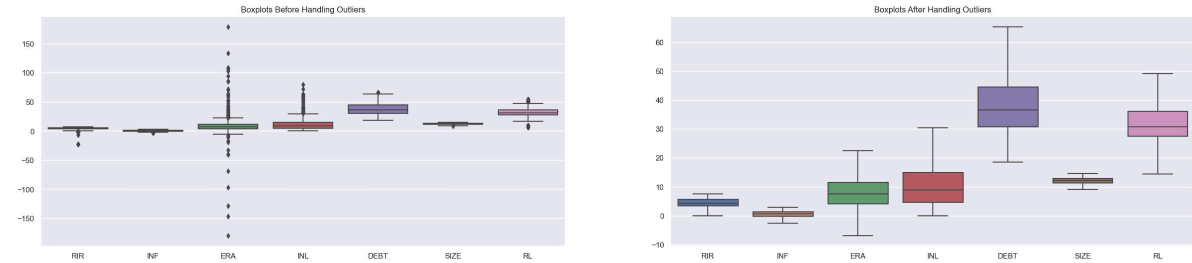
Out[11]:
```
Year              7
Countries_Num     8
Countries         8
RQ               42
CC               42
RL               42
GE               47
VA               47
PS               50
SFS              56
DEBT             56
RIR              56
INF              56
Banks            98
id              106
ERA             711
INL             712
SIZE            714
Zscore          718
dtype: int64
```

In [12]:
```
def boxplots(col):
    plt.figure(figsize=(5,4))
    sns.boxplot(df,x=col,palette='husl')
    plt.show()

# Assuming 'df' is your DataFrame
for i in list(df.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```



In [13]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load your own dataset
# Replace 'your_dataset.csv' with the actual file path or URL of your data
df = df

# Function to handle outliers using clip
def handle_outliers(dataframe, col):
    q3 = dataframe[col].quantile(0.75)
    q1 = dataframe[col].quantile(0.25)
    IQR = q3 - q1
    Lower = q1 - 1.5 * IQR
    Upper = q3 + 1.5 * IQR

    # Clip the values to be within the Lower and Upper bounds
    dataframe[col] = dataframe[col].clip(lower=Lower, upper=Upper)

# Specify the columns to handle outliers
columns_to_handle_outliers = [
                              'RIR',
                              'INF',
                              'ERA',
                              'INL' ,
                              'DEBT' ,
                              'SIZE',
                              'RL',
                                       ]

# Plot boxplots before handling outliers
plt.figure(figsize=(30, 6))
plt.subplot(1, 2, 1)
sns.boxplot(data=df[columns_to_handle_outliers].dropna())
plt.title('Boxplots Before Handling Outliers')

# Handle outliers for each specified column
for col in columns_to_handle_outliers:
    handle_outliers(df, col)

# Plot boxplots after handling outliers
plt.subplot(1, 2, 2)
sns.boxplot(data=df[columns_to_handle_outliers].dropna())
plt.title('Boxplots After Handling Outliers')

plt.show()

```



In [14]:

```
df.hist(figsize = (25,20))
plt.show()
```

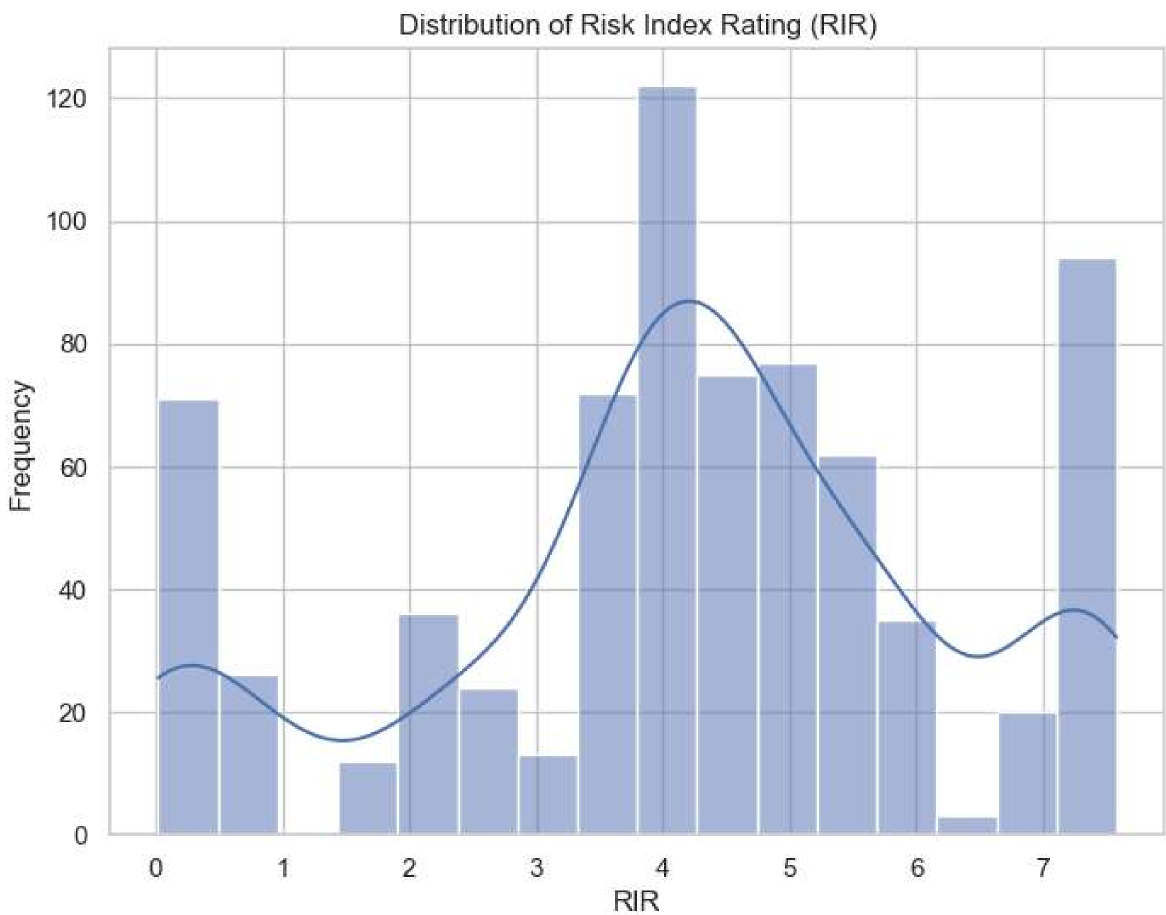
In [15]:

```python
# Exploratory Data Analysis (EDA)

# Setting aesthetics for plots
sns.set_style('whitegrid')

# A. Descriptive Statistics
print("Descriptive Statistics:")
print(df.describe())
```

```
Descriptive Statistics:
       Countries_Num          id   Countries       Banks         Year  \
count     742.000000  742.000000  742.000000  742.000000   742.000000
mean        4.575472   53.500000    3.650943   48.716981  2016.000000
std         2.316191   30.618842    2.380928   27.703134     2.001349
min         1.000000    1.000000    0.000000    0.000000  2013.000000
25%         3.000000   27.000000    2.000000   26.000000  2014.000000
50%         5.000000   53.500000    3.000000   47.500000  2016.000000
75%         7.000000   80.000000    6.000000   74.000000  2018.000000
max         8.000000  106.000000    7.000000   97.000000  2019.000000

              RIR         SFS         INF         ERA         INL      Zscore
\
count  742.000000  742.000000  742.000000  742.000000  742.000000  742.000000
mean     4.180301   31.965249    0.535323    8.683170   10.825335    2.967964
std      2.065988    8.239127    1.239726    6.636001    8.166584    5.123174
min      0.017647   15.829639   -2.622454   -6.865332    0.000000  -47.777093
25%      3.367054   26.927042   -0.258090    4.166364    4.656050    1.482417
50%      4.266334   29.918364    0.685881    7.460494    8.895584    2.149078
75%      5.599992   37.793417    1.318153   11.520829   14.951115    3.468952
max      7.578020   51.682209    2.967604   22.552525   30.393713   39.380715

             DEBT        SIZE          CC          GE          PS          RQ
\
count  742.000000  742.000000  742.000000  742.000000  742.000000  742.000000
mean    38.778321   12.078829   37.292780   26.747784   23.891205   35.432019
std     11.658474    1.111316   12.589451   10.261396   14.741995    9.474103
min     18.503746    9.061235   13.461540    8.530806    3.773585   13.942310
25%     30.666445   11.359607   27.403850   18.009480   12.857140   28.365390
50%     36.494147   12.156868   33.173080   26.442310   17.061610   34.134620
75%     44.508726   12.891855   49.038460   35.096150   38.571430   43.269230
max     65.272147   14.582210   58.653850   50.961540   58.293840   52.606640

               RL          VA
count  742.000000  742.000000
mean    32.523353   41.074591
std      9.209518   11.071831
min     14.423090   18.309860
25%     27.403850   33.004920
50%     30.769230   38.423650
75%     36.057690   53.490507
max     49.038450   60.098520
```

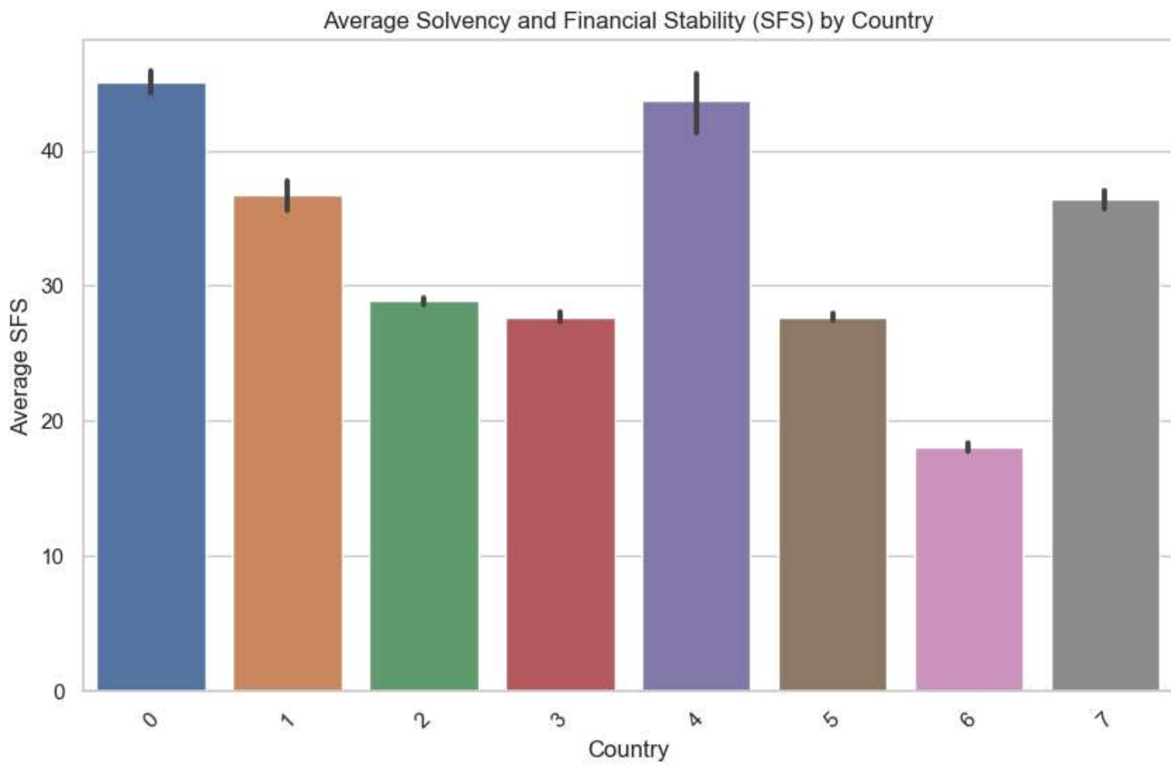
In [16]:

```
# B. Distribution of Key Variables
# Example: Distribution of Risk Index Rating (RIR)
plt.figure(figsize=(8, 6))
sns.histplot(df['RIR'], kde=True)
plt.title('Distribution of Risk Index Rating (RIR)')
plt.xlabel('RIR')
plt.ylabel('Frequency')
plt.show()
```



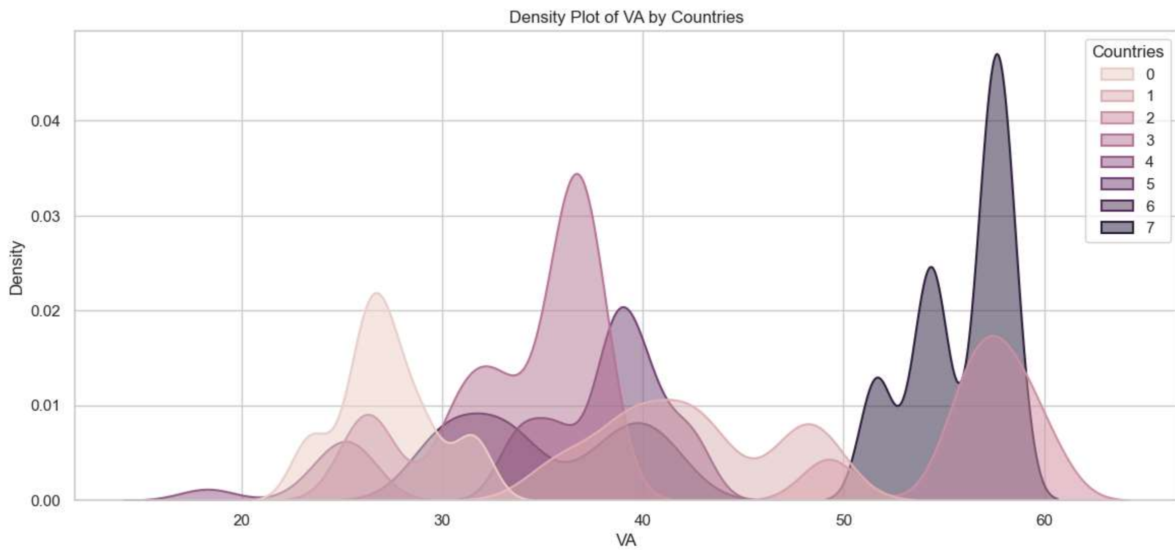
In [17]:

```
# C. Comparison of Metrics Across Different Countries
# Example: Average Solvency and Financial Stability (SFS) by Country
plt.figure(figsize=(10, 6))
sns.barplot(x='Countries', y='SFS', data=df)
plt.title('Average Solvency and Financial Stability (SFS) by Country')
plt.xlabel('Country')
plt.ylabel('Average SFS')
plt.xticks(rotation=45)
plt.show()
```

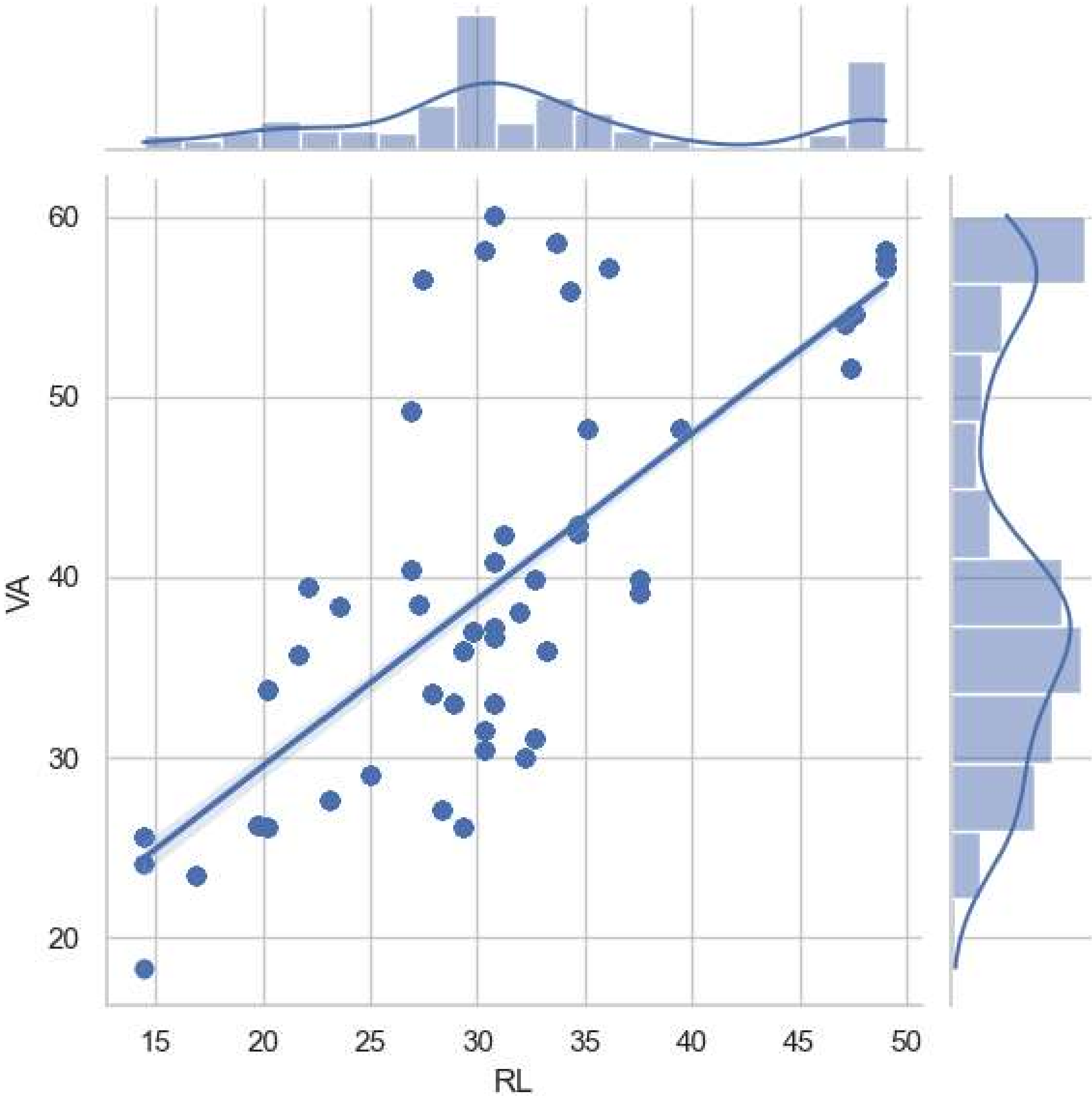


In [18]:

```
plt.figure(figsize=(14, 6))
sns.kdeplot(data=df, x='VA',
            hue='Countries',          # to differentiate between the diffe
            fill=True,                # to fill the area under the curve
            alpha=0.5, linewidth=1.5  # to make the plot more visually app
           )


# Add a title and labels to the plot using Matplotlib
plt.title("Density Plot of VA by Countries")
plt.xlabel("VA")
plt.ylabel("Density")

# display the plot
plt.show()
```



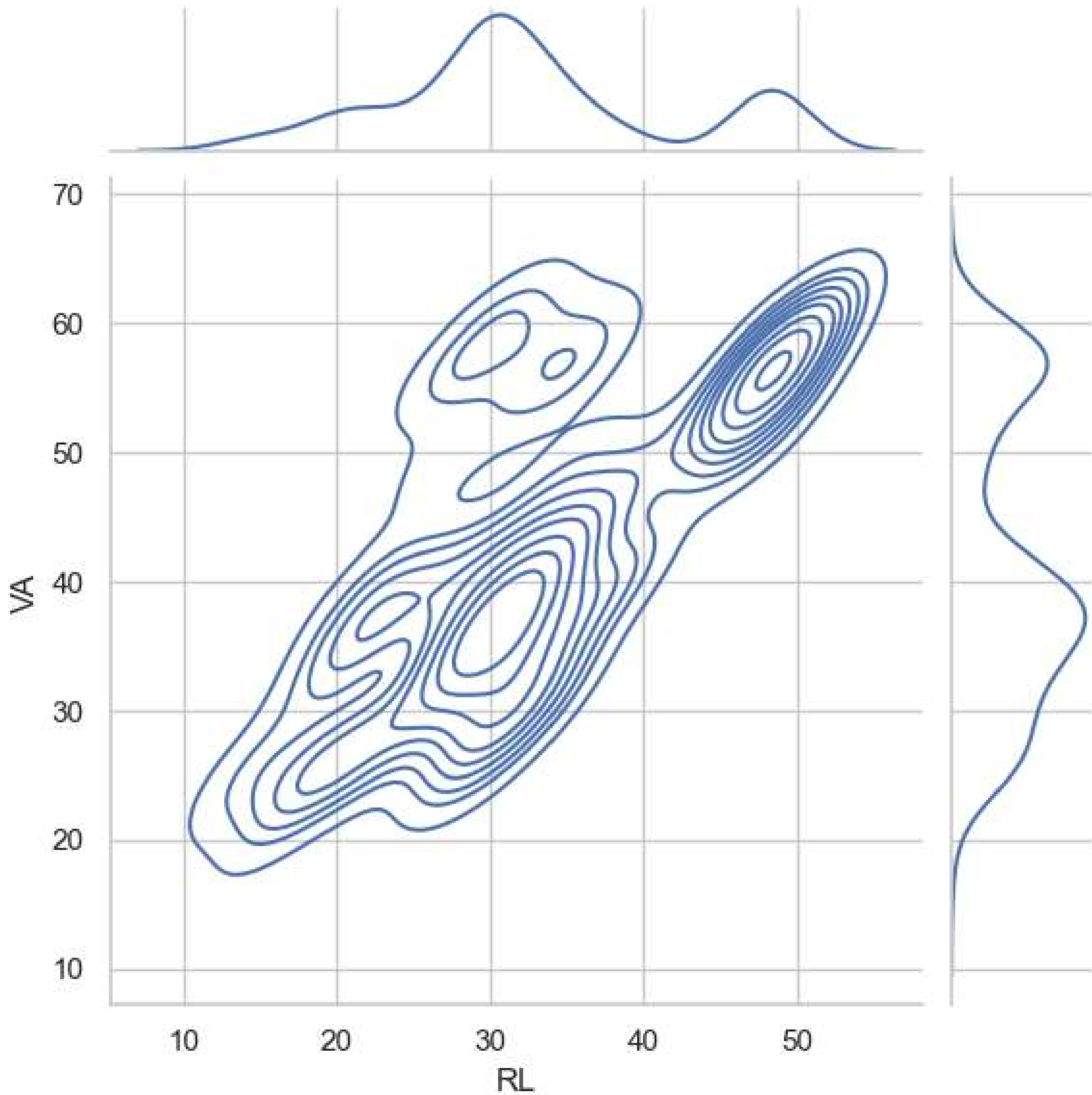
In [19]:
```
sns.jointplot(data=df,          # Plot bivariate distribution
              x='RL', y='VA',
              kind='reg')       # kind= reg, kde, hex

# display the plot
plt.show()
```

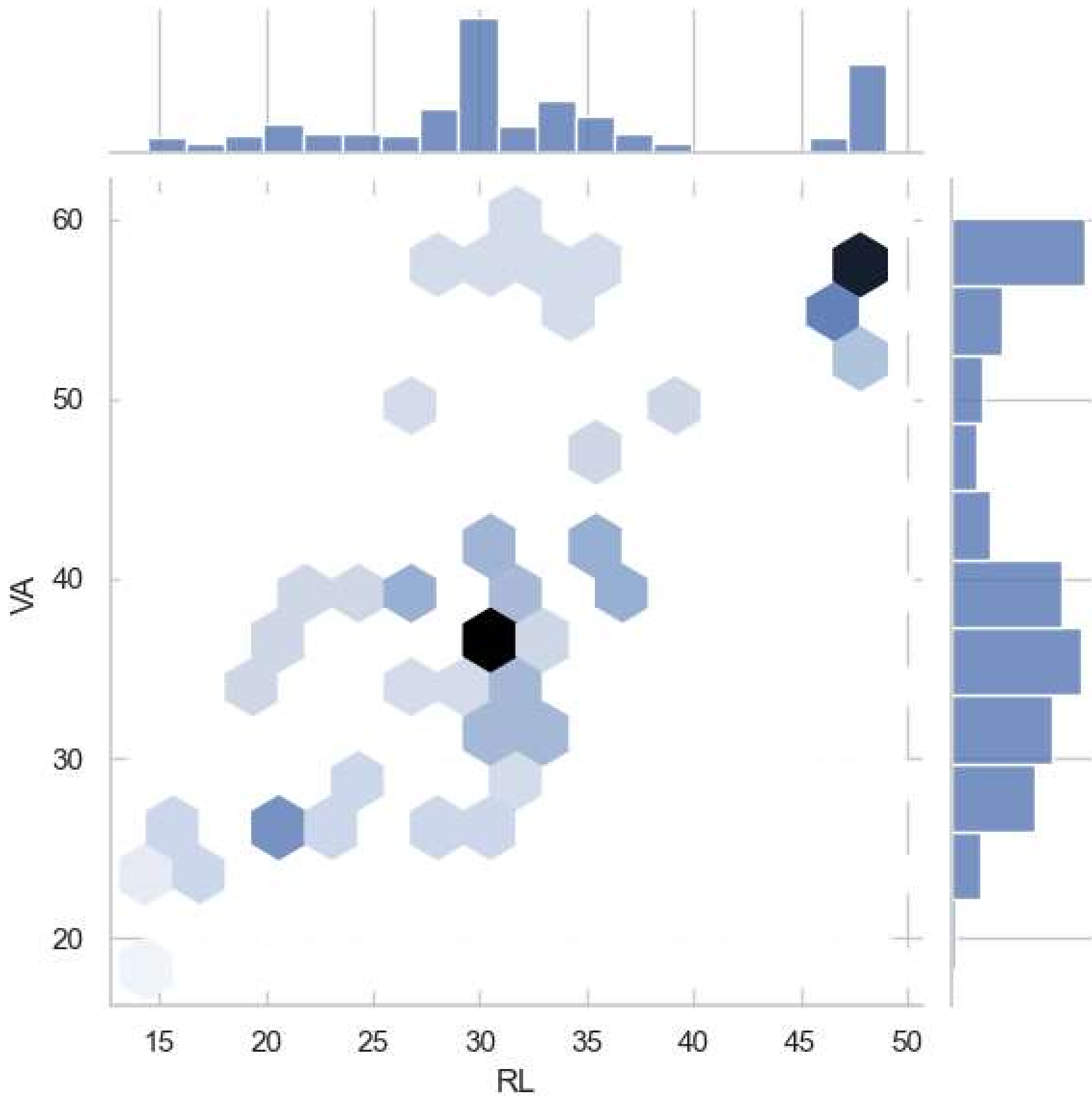


In [20]:

```
# let's see what happens if we change the kind to kde
sns.jointplot(x='RL', y='VA', data=df, kind='kde')

# let's see what happens if we change the kind to hex
sns.jointplot(x='RL', y='VA', data=df, kind='hex')
```

Out[20]: <seaborn.axisgrid.JointGrid at 0x155c762bbb0>



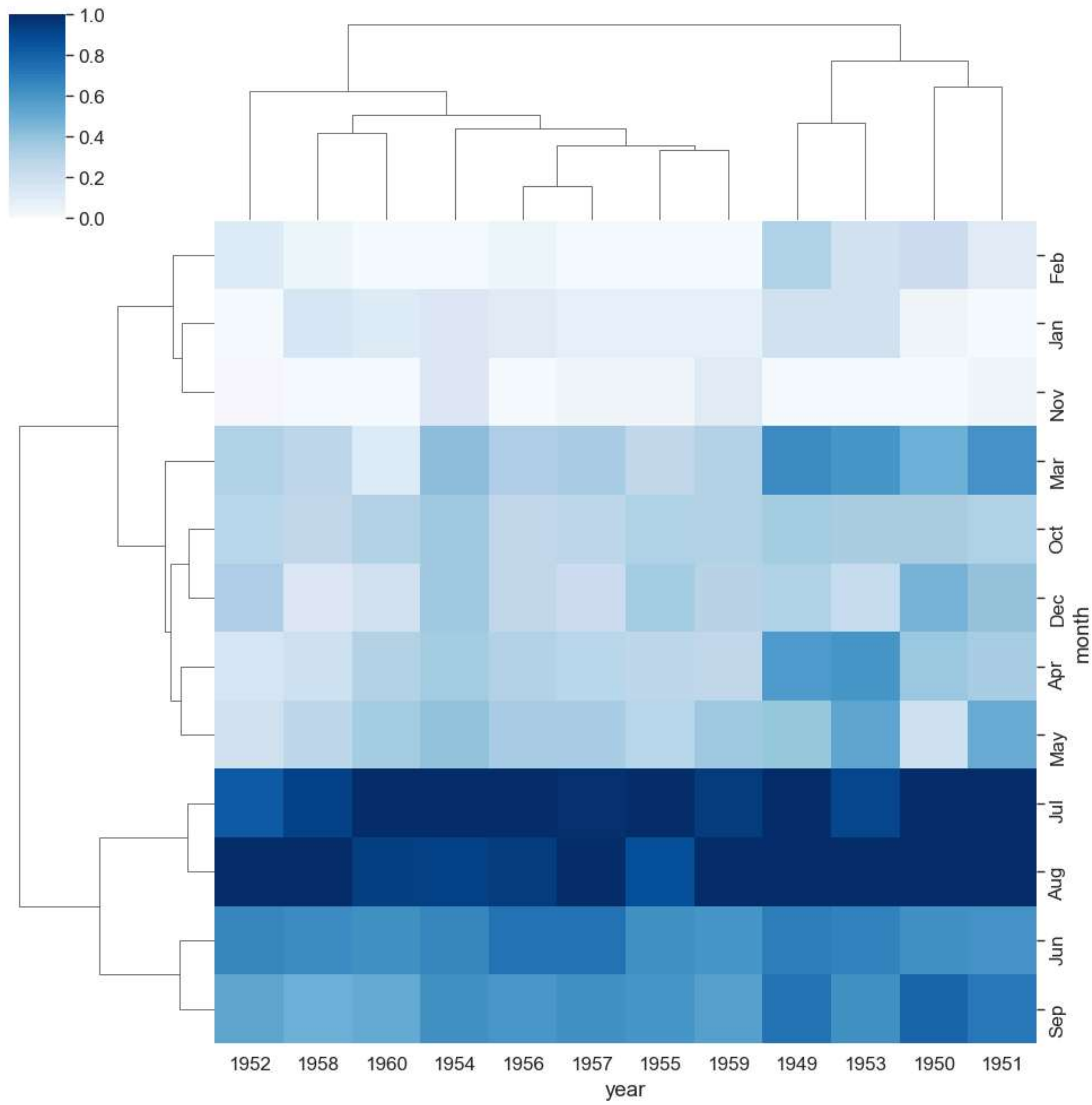

60
50
40
30
20
VA
15
20
25
30
35
40
45
50
RL

In [21]:

```python
plt.figure(figsize=(4,6))
sns.set_context('paper', font_scale=1.4)

# let's use flights dataset
flights = sns.load_dataset("flights")
flights = flights.pivot_table(index='month', columns='year', values='passe

sns.clustermap(flights,cmap="Blues", standard_scale=1)
```

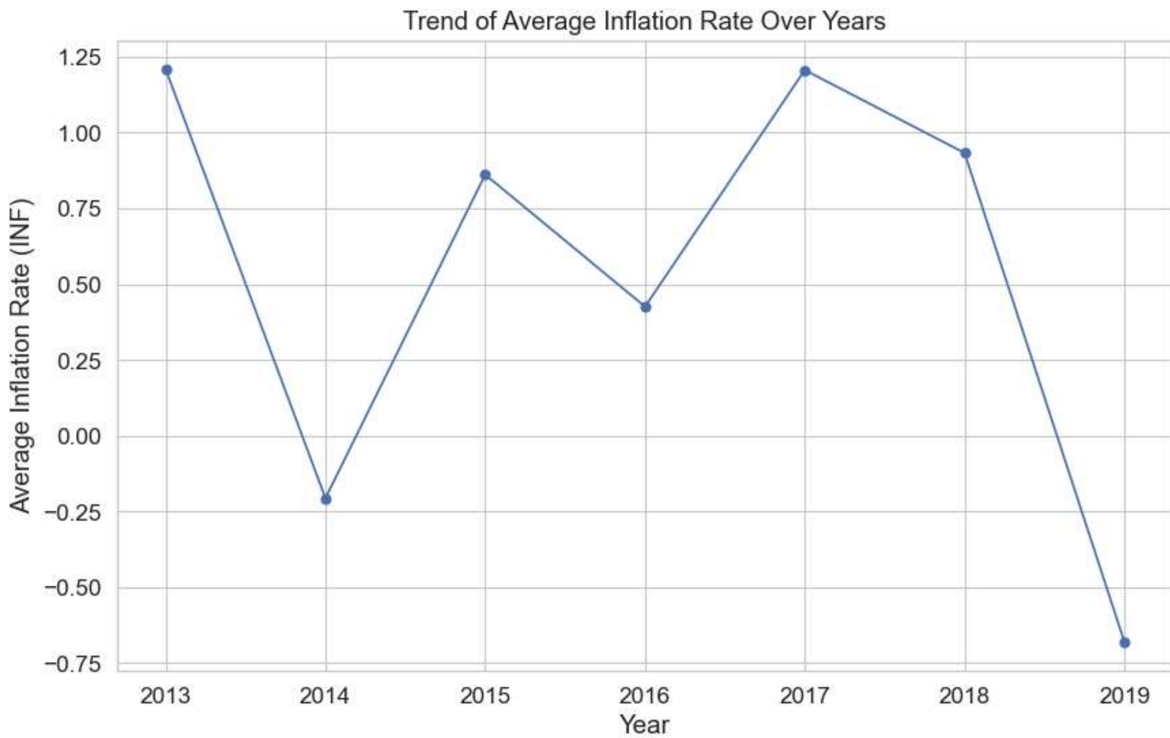
Out[21]:

```
<seaborn.matrix.ClusterGrid at 0x155c736b040>

<Figure size 400x600 with 0 Axes>
```



In [22]:

```
# D. Time-Series Analysis for Yearly Data
# Example: Trend of Average Inflation Rate (INF) Over Years
plt.figure(figsize=(10, 6))
df.groupby('Year')['INF'].mean().plot(kind='line', marker='o')
plt.title('Trend of Average Inflation Rate Over Years')
plt.xlabel('Year')
plt.ylabel('Average Inflation Rate (INF)')
plt.show()
```



In [23]:

```
df['Countries'] = df['Countries'].astype('category')
df['Countries'] = df['Countries'].cat.codes
df.head()
```

Out[23]:

	Countries_Num	id	Countries	Banks	Year	RIR	SFS	INF	ERA	
0	1	1	2	13	2013	3.836593	26.861971	0.428889	3.196428	12.076
1	1	1	2	13	2014	5.599992	29.965430	-0.548758	3.045024	8.884
2	1	1	2	13	2015	4.266334	30.984761	0.218786	2.394557	8.583
3	1	1	2	13	2016	4.580100	29.832095	-0.794050	3.712403	5.720
4	1	1	2	13	2017	7.329021	28.630991	1.769412	3.833422	6.256

In [24]:
```
df['Banks'] = df['Banks'].astype('category')
df['Banks'] = df['Banks'].cat.codes
df.head()
```

Out[24]:

	Countries_Num	id	Countries	Banks	Year	RIR	SFS	INF	ERA	
0	1	1	2	13	2013	3.836593	26.861971	0.428889	3.196428	12.076
1	1	1	2	13	2014	5.599992	29.965430	-0.548758	3.045024	8.884
2	1	1	2	13	2015	4.266334	30.984761	0.218786	2.394557	8.583
3	1	1	2	13	2016	4.580100	29.832095	-0.794050	3.712403	5.720
4	1	1	2	13	2017	7.329021	28.630991	1.769412	3.833422	6.256

In [25]:
```
# Binning Zscore into categories (example: high risk = 1, low risk = 0)
df['Risk_Category'] = pd.cut(df['Zscore'], bins=[-np.inf, 1.8, np.inf], la
```

Purpose

The purpose of this line is to transform a continuous measure (Zscore) into a binary categorical variable (Risk_Category), which can be useful for classification tasks. In this case:

- A Zscore of 1.8 or below is labeled as 1 (high risk).
- A Zscore above 1.8 is labeled as 0 (low risk).

In [26]:
```
# Features and target
features = df.drop(columns=['Zscore', 'id', 'Countries', 'Banks', 'Risk_Ca
target = df['Risk_Category']
```

In [27]:
```
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(features, target, test
```

In [28]:
```
# Creating the preprocessing
numeric_features = features.select_dtypes(include=['int64', 'float64']).co
numeric_transformer = StandardScaler()

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
    ])
```

In [29]:
```
# Creating the pipeline

from sklearn.ensemble import VotingClassifier

# Creating individual models
rf_model = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', RandomForestClassifier(n_estimat

gb_model = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', GradientBoostingClassifier(n_est

# Creating the voting classifier
voting_clf = VotingClassifier(estimators=[('rf', rf_model), ('gb', gb_mode

```

In [30]:
```
# Fitting the model
voting_clf.fit(X_train, y_train)
```

Out[30]:
```
VotingClassifier(estimators=[('rf',
                              Pipeline(steps=[('preprocessor',
                                               ColumnTransformer(transformers
=[('num',

StandardScaler(),

Index(['Countries_Num', 'Year', 'RIR', 'SFS', 'INF', 'ERA', 'INL', 'DEBT',
       'SIZE', 'CC', 'GE', 'PS', 'RQ', 'RL', 'VA'],
      dtype='object'))])),
                                              ('classifier',
                                               RandomForestClassifier(random_
state=42))])),
                             ('gb',
                              Pipeline(steps=[('preprocessor',
                                               ColumnTransformer(transformers
=[('num',

StandardScaler(),

Index(['Countries_Num', 'Year', 'RIR', 'SFS', 'INF', 'ERA', 'INL', 'DEBT',
       'SIZE', 'CC', 'GE', 'PS', 'RQ', 'RL', 'VA'],
      dtype='object'))])),
                                              ('classifier',
                                               GradientBoostingClassifier(ran
dom_state=42))]))],
                 voting='soft')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:
```
# Predictions
y_pred_train = voting_clf.predict(X_train)
y_pred_test = voting_clf.predict(X_test)
```

In [32]:
```
# Evaluation
print("Training Classification Report:")
print(classification_report(y_train, y_pred_train))

print("********"*8)
print("Test Classification Report:")
print(classification_report(y_test, y_pred_test))

```

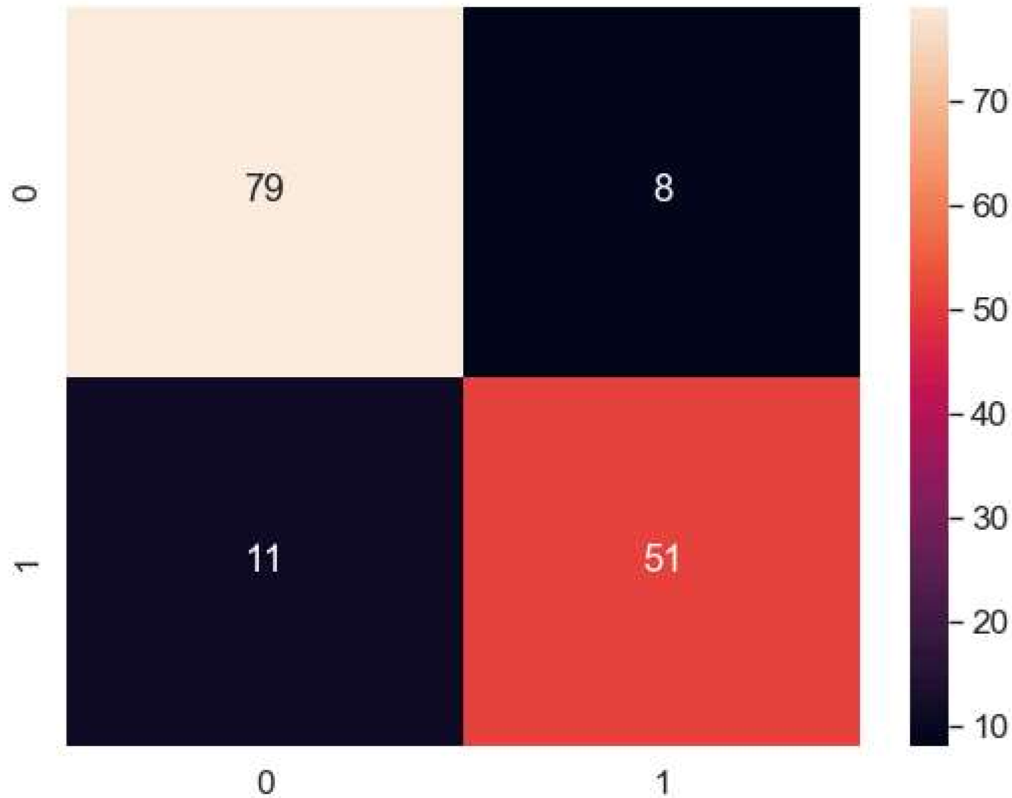
```
Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       374
           1       1.00      1.00      1.00       219

    accuracy                           1.00       593
   macro avg       1.00      1.00      1.00       593
weighted avg       1.00      1.00      1.00       593

****************************************************************
Test Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.91      0.89        87
           1       0.86      0.82      0.84        62

    accuracy                           0.87       149
   macro avg       0.87      0.87      0.87       149
weighted avg       0.87      0.87      0.87       149
```

In [33]:
```
# Confusion Matrix
print("Confusion Matrix:")
cf_matrix = confusion_matrix(y_test, y_pred_test)
print(cf_matrix)
```

```
Confusion Matrix:
[[79  8]
 [11 51]]
```

In [34]:
```
import seaborn as sns
sns.heatmap(cf_matrix, annot=True)
```

Out[34]: <Axes: >



In [ ]:
```

```