Churn Modelling Data

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

In [109…
```
data = pd.read_excel("P3- Churn_Modelling Data.xlsx")
data.head(2)
```

Out[109…

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	

In [3]:
```
data.isnull().sum()
```

Out[3]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
churned            0
dtype: int64
```

In [4]:
```
print(data.dtypes)
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
churned              int64
dtype: object
```

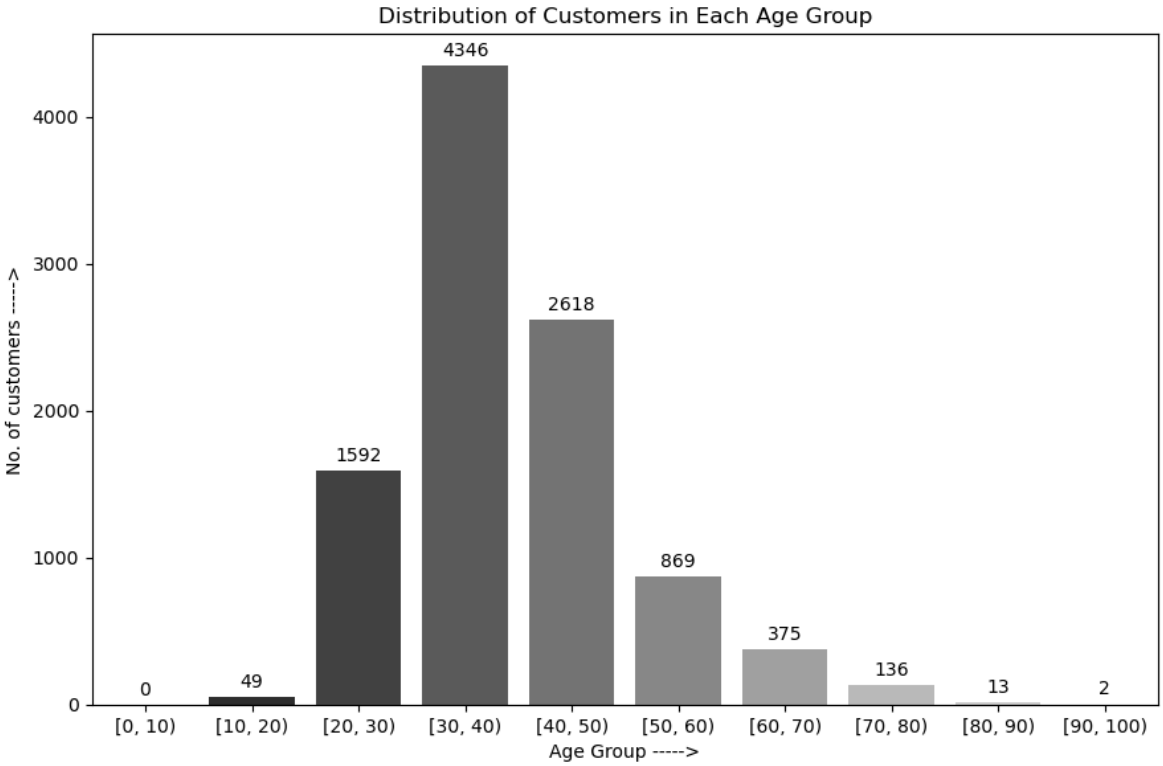
In [5]:
```
df = pd.DataFrame(data)
```

1. Customer Demographics:

- What is the distribution of customers across different age groups? - Analyze the gender distribution of customers.

What is the distribution of customers across different age groups?

In [6]:
```
df['Age Group'] = pd.cut(df['Age'], bins=range(0, 101, 10), right=False)


plt.figure(figsize=(9, 6))
barplot = sns.countplot(x='Age Group', data=df, palette='gray',)


for bar in barplot.patches:       # Adding count annotations on top of ea
    bar_height = bar.get_height()
    barplot.annotate(format(bar_height, '.0f'),
                     (bar.get_x() + bar.get_width() / 2, bar_height),
                     ha='center', va='center',
                     size=10, xytext=(0, 8),
                     textcoords='offset points')


plt.xlabel('Age Group ----->')
plt.ylabel('No. of customers ----->')
plt.title('Distribution of Customers in Each Age Group')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [7]:
```
freq_customer = df["Age"].value_counts()
freq_customer
```

Out[7]:
```
37    478
38    477
35    474
36    456
34    447
     ...
92      2
82      1
88      1
85      1
83      1
Name: Age, Length: 70, dtype: int64
```

In [8]:
```
# Convert freq_customer Series to a DataFrame with appropriate column nam
freq_customer_df = freq_customer.reset_index()
freq_customer_df.columns = ['Age', 'Count']

# Export to Excel
excel_file_path = 'frequency_counts.xlsx'  # Replace with your desired fi
freq_customer_df.to_excel(excel_file_path, index=False)

print(f"Frequency counts exported to {excel_file_path}")
```

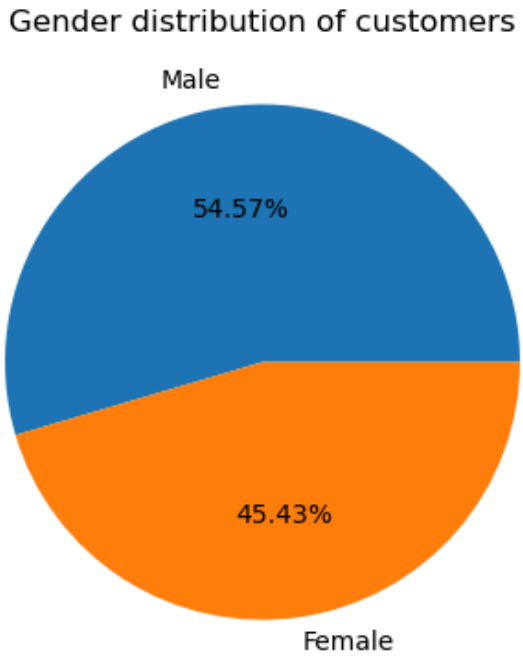
```
Frequency counts exported to frequency_counts.xlsx
```

Analyze the gender distribution of customers.

In [9]:
```
# df["Gender"].unique()
Gender_distribution = df["Gender"].value_counts()
Gender_distribution
```

Out[9]:
```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

In [10]:
```
plt.figure(figsize=(8, 4))
plt.pie(Gender_distribution, labels= ['Male','Female'], autopct='%1.2f%%'
plt.title('Gender distribution of customers\n')
plt.axis('equal')
plt.show()
```



2. Churn Analysis:

- What percentage of customers have churned? - What are the main reasons for customer churn? - Identify any patterns or trends among customers who have churned.

- What percentage of customers have churned?

In [11]:
```
data.head(2)
```

Out[11]:

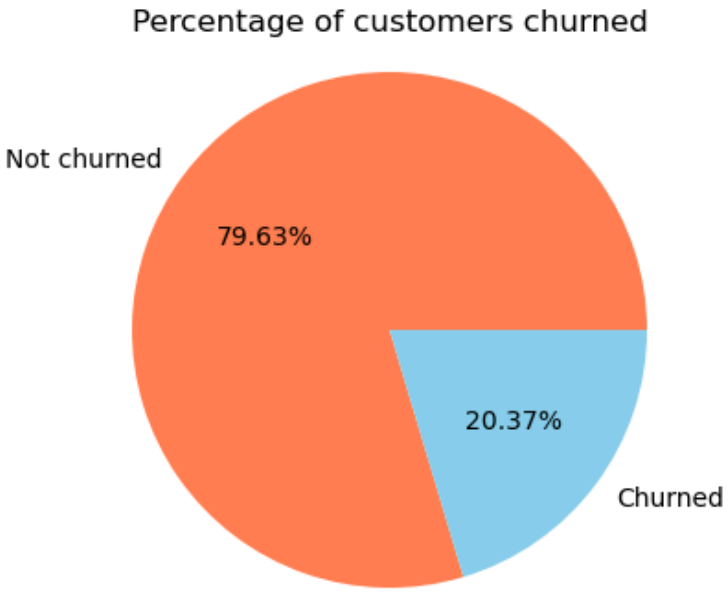
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	

In [12]:
```
churned = df['churned'].value_counts()
churned
```

Out[12]:
```
0    7963
1    2037
Name: churned, dtype: int64
```

In [13]:
```
plt.figure(figsize=(8, 4))
plt.pie(churned, labels=['Not churned','Churned'], autopct='%1.2f%%', sta
plt.title('Percentage of customers churned')
```

```
plt.axis('equal')
plt.show()
```

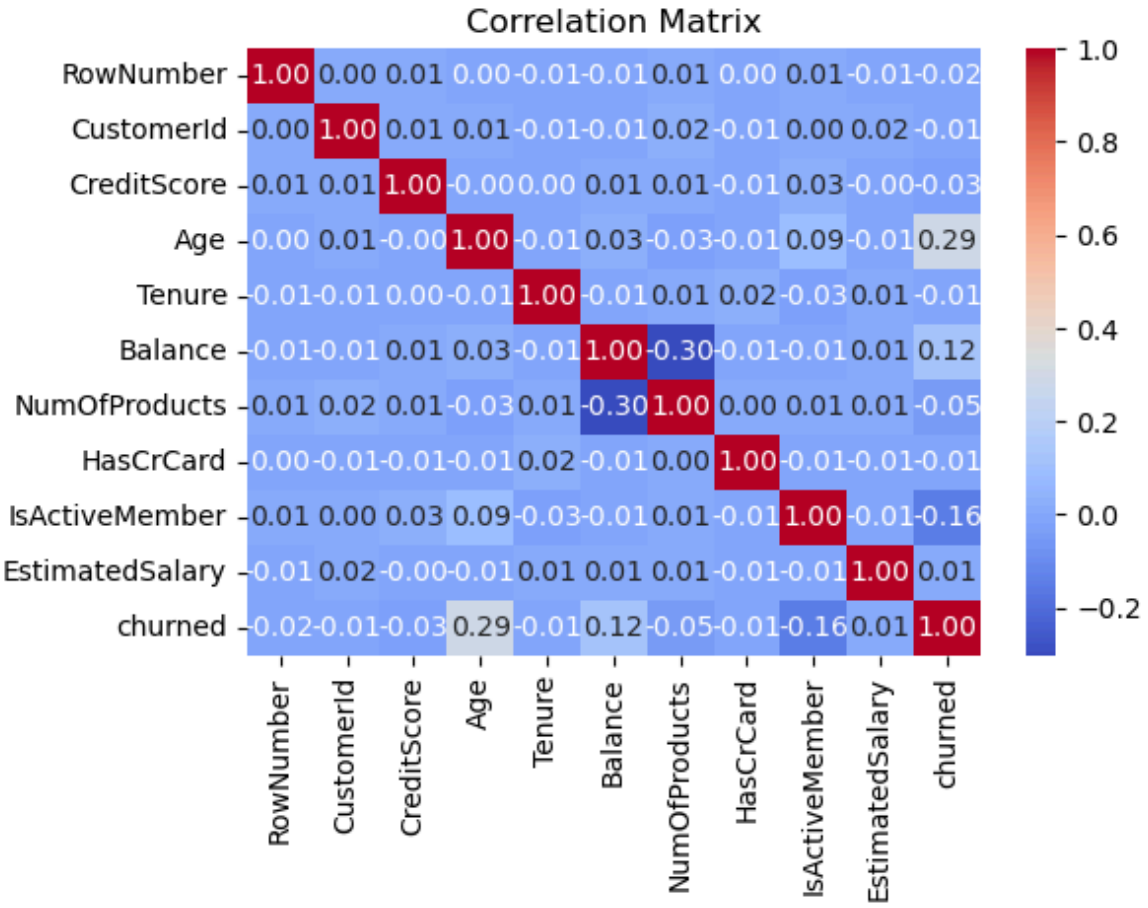


- What are the main reasons for customer churn?

In [14]:
```
correlation = data.corr()
```

```
/var/folders/0c/mt12df7d7rj2s2x_hbbf4mvw0000gn/T/ipykernel_63391/30774949
3.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr
is deprecated. In a future version, it will default to False. Select only
valid columns or specify the value of numeric_only to silence this warnin
g.
  correlation = data.corr()
```

In [15]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(correlation, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
```

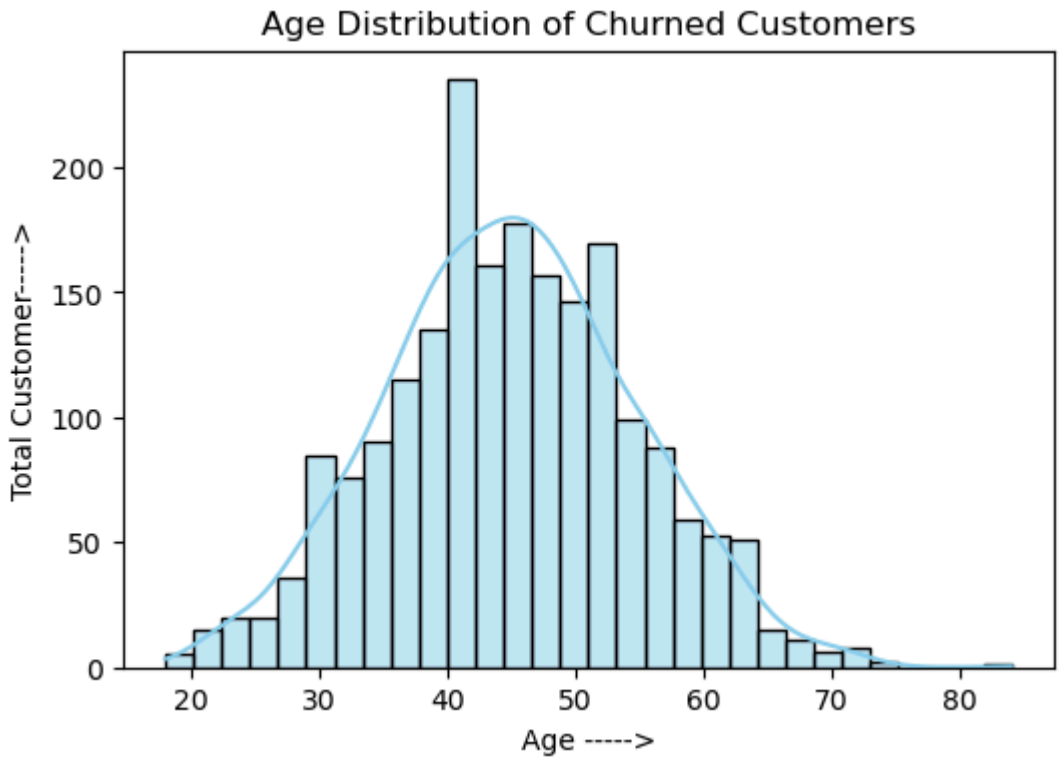


In [16]:
```
reasons_for_churn = "'Age' and 'Balance' are the main reasons for custome
print(reasons_for_churn)
```

```
'Age' and 'Balance' are the main reasons for customer churn.
```

- Identify any patterns or trends among customers who have churned.

In [17]:
```
# Analyze age distribution among churned customers
plt.figure(figsize=(6, 4))
sns.histplot(data=df[df['churned'] == 1], x='Age', bins=30, kde=True, col
plt.title('Age Distribution of Churned Customers')
plt.xlabel('Age ----->')
plt.ylabel('Total Customer----->')
plt.show()

# Analyze balance distribution among churned customers
plt.figure(figsize=(6,4))
sns.histplot(data=df[df['churned'] == 1], x='Balance', bins=30, kde=True,
plt.title('Balance Distribution of Churned Customers')
plt.xlabel('Balance ----->')
plt.ylabel('Total Customer----->')
plt.show()
```


Age Distribution of Churned Customers
Total Customer----->
0
50
100
150
200
20
30
40
50
60
70
80
Age ----->
Balance Distribution of Churned Customers
Total Customer----->
0
100
200
300
400
500
0
50000
100000
150000
200000
250000
Balance ----->
3. Product Usage:
- What are the most commonly used products or services? - Analyze the usage patterns of different customer segments.#
In [18]:
data.head(2)

Out[18]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	

In [19]:
```
print(df['Age Group'].unique())
```

```
[[40, 50), [30, 40), [50, 60), [20, 30), [60, 70), [10, 20), [70, 80), [8
0, 90), [90, 100)]
Categories (10, interval[int64, left]): [[0, 10) < [10, 20) < [20, 30) <
[30, 40) ... [60, 70) < [70, 80) < [80, 90) < [90, 100)]
```

In [20]:
```
print(df['Age Group'].value_counts())
```

```
[30, 40)     4346
[40, 50)     2618
[20, 30)     1592
[50, 60)      869
[60, 70)      375
[70, 80)      136
[10, 20)       49
[80, 90)       13
[90, 100)       2
[0, 10)         0
Name: Age Group, dtype: int64
```

In [ ]:
```

```

- What are the most commonly used products or services?

In [21]:
```
print(df["NumOfProducts"].value_counts()),
print(df["HasCrCard"].value_counts()),
print(df["IsActiveMember"].value_counts())
```

```
1    5084
2    4590
3     266
4      60
Name: NumOfProducts, dtype: int64
1    7055
0    2945
Name: HasCrCard, dtype: int64
1    5151
0    4849
Name: IsActiveMember, dtype: int64
```

- Analyze the usage patterns of different customer segments.

In [25]:
```
df.head(3)
```

Out[25]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	
2	3	15619304	Onio	502	France	Female	42	

In [65]:
```
# Segmentation
df['Age Group'] = pd.cut(df['Age'], bins=range(0, 101, 10), right=False)
df['CreditScoreGroup'] = pd.cut(data['CreditScore'], bins=range(0, 1001,

# Analysis
age_group_analysis = data.groupby('Age Group').mean()
gender_analysis = data.groupby('Gender').mean()
geography_analysis = data.groupby('Geography').mean()

# Print insights
# print("Age Group Analysis:\n", age_group_analysis)
# print("Gender Analysis:\n", gender_analysis)
# print("Geography Analysis:\n", geography_analysis)
```

```
/var/folders/0c/mt12df7d7rj2s2x_hbbf4mvw0000gn/T/ipykernel_63391/266624225
4.py:6: FutureWarning: The default value of numeric_only in DataFrameGroup
By.mean is deprecated. In a future version, numeric_only will default to F
alse. Either specify numeric_only or select only columns which should be v
alid for the function.
  age_group_analysis = data.groupby('Age Group').mean()
/var/folders/0c/mt12df7d7rj2s2x_hbbf4mvw0000gn/T/ipykernel_63391/266624225
4.py:7: FutureWarning: The default value of numeric_only in DataFrameGroup
By.mean is deprecated. In a future version, numeric_only will default to F
alse. Either specify numeric_only or select only columns which should be v
alid for the function.
  gender_analysis = data.groupby('Gender').mean()
/var/folders/0c/mt12df7d7rj2s2x_hbbf4mvw0000gn/T/ipykernel_63391/266624225
4.py:8: FutureWarning: The default value of numeric_only in DataFrameGroup
By.mean is deprecated. In a future version, numeric_only will default to F
alse. Either specify numeric_only or select only columns which should be v
alid for the function.
  geography_analysis = data.groupby('Geography').mean()
```

In [32]:
```
credit_card_counts = df[df['HasCrCard'] == 1].groupby('Age Group').size()

print(credit_card_counts)
```

```
Age Group
[0, 10)         0
[10, 20)       36
[20, 30)     1136
[30, 40)     3092
[40, 50)     1830
[50, 60)      588
[60, 70)      263
[70, 80)      100
[80, 90)        9
[90, 100)       1
dtype: int64
```

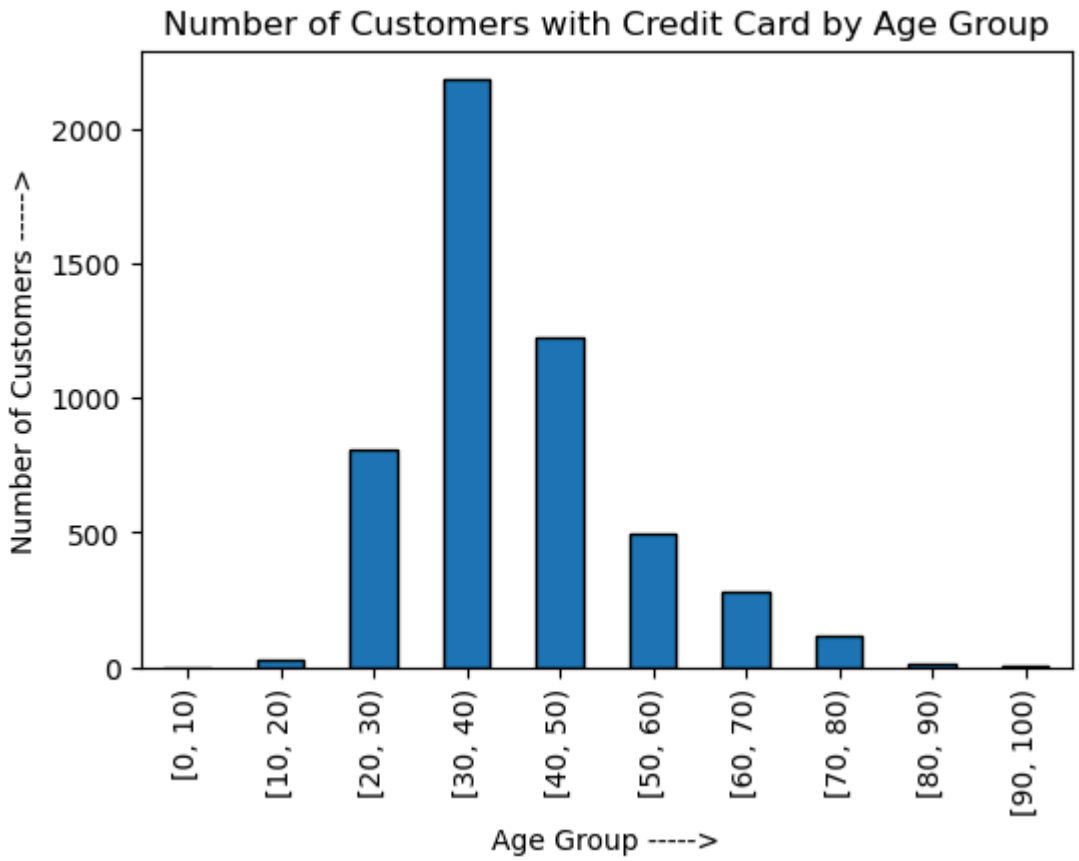
In [76]:
```
Active_member_counts = df[df['IsActiveMember'] == 1].groupby('Age Group')

print(Active_member_counts)
```

```
Age Group
[0, 10)         0
[10, 20)       28
[20, 30)      809
[30, 40)     2181
[40, 50)     1222
[50, 60)      496
[60, 70)      281
[70, 80)      120
[80, 90)       12
[90, 100)       2
dtype: int64
```

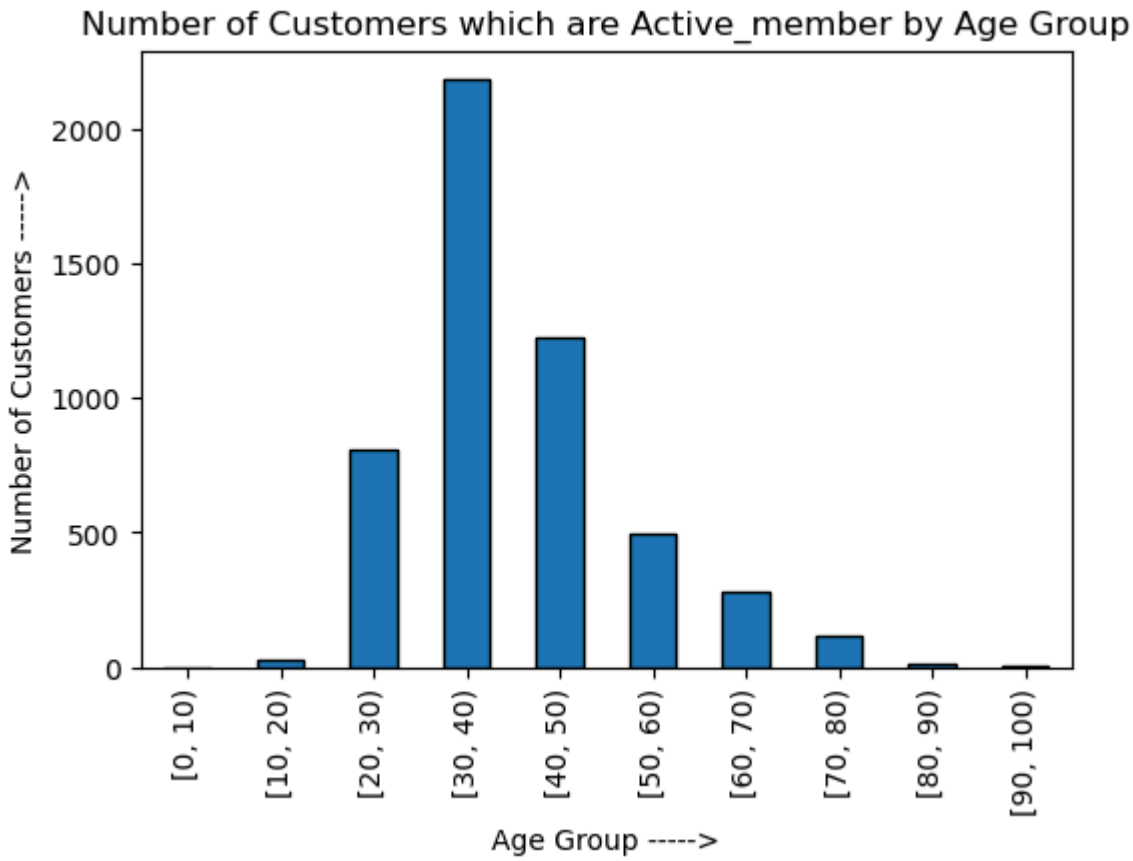
In [77]:
```
# Active_member_counts = df[df['Geography'] == 1].groupby('Age Group').si

# print(Active_member_counts)
Geography_region = df['Geography'].unique()
Geography_region
```

Out[77]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [78]:
```
plt.figure(figsize = (6,4))
bar = credit_card_counts.plot(kind='bar', edgecolor='black')
for bar in barplot.patches:       # Adding count annotations on top of ea
    bar_height = bar.get_height()
    barplot.annotate(format(bar_height, '.0f'), (bar.get_x() + bar.get_wi
plt.title('Number of Customers with Credit Card by Age Group')
plt.xlabel('Age Group ----->')
plt.ylabel('Number of Customers ----->')
plt.show()
```



In [80]:
```
plt.figure(figsize = (6,4))
bar = Active_member_counts.plot(kind='bar', edgecolor='black')
for bar in barplot.patches:       # Adding count annotations on top of ea
    bar_height = bar.get_height()
    barplot.annotate(format(bar_height, '.0f'), (bar.get_x() + bar.get_wi
plt.title('Number of Customers which are Active_member by Age Group')
plt.xlabel('Age Group ----->')
plt.ylabel('Number of Customers ----->')
plt.show()
```


Number of Customers which are Active_member by Age Group
Number of Customers ----->
2000
1500
1000
500
0
[0, 10)
[10, 20)
[20, 30)
[30, 40)
[40, 50)
[50, 60)
[60, 70)
[70, 80)
[80, 90)
[90, 100)
Age Group ----->
4. Financial Analysis:
- What is the average account balance of customers? - Compare the financial characteristics of churned vs. non-churned customers.
In [ ]:
data.head(2)
In [ ]:
data.describe()
- What is the average account balance of customers?
In [ ]:
df['Balance'].mean()
- Compare the financial characteristics of churned vs. non-churned customers.
In [ ]:
total_churned_customer = df.groupby('churned')['Balance'].sum().reset_ind
total_churned_customer
In [ ]:
total_churned_customer1 = df.groupby('churned')['EstimatedSalary'].sum().
total_churned_customer1

In [ ]:
```
plt.figure(figsize=(8, 4))

# Subplot 1: Total Balance Distribution
plt.subplot(1, 2, 1)
plt.pie(total_churned_customer['Balance'], labels=['Not Churned', 'Churne
plt.title('Total Balance Distribution')
plt.axis('equal')

# Subplot 2: Total Estimated Salary Distribution
plt.subplot(1, 2, 2)
plt.pie(total_churned_customer1['EstimatedSalary'], labels=['Not Churned'
plt.title('Total Estimated Salary Distribution')
plt.axis('equal')

# Adjust layout to prevent overlapping
plt.tight_layout()


plt.show()
```

5. Predictive Modeling:

- Which factors are the most significant predictors of customer churn? - Develop a predictive model to identify at-risk customers.

In [ ]:
```
data.head(2)
```

In [ ]:
```
# df
```

- Which factors are the most significant predictors of customer churn?

In [ ]:
```
# Age and Balance is highly correlated so these are most significant pred
```

- Develop a predictive model to identify at-risk customers

In [86]:
```
data.head(2)
```

Out[86]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Ten
0	1	15634602	Hargrave	619	France	Female	42	
1	2	15647311	Hill	608	Spain	Female	41	

In [113…
```
# sns.heatmap(data , annot = True, cmap = 'coolwarm')
```

In [ ]:

In [ ]:

In [87]:
```
#MODEL
```

In [88]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classificat
```

In [89]:
```
X = data[['Age', 'Balance']]
y = data['churned']
```

In [90]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [106…
```
X_test
```

Out[106…

	Age	Balance
6252	32	96709.07
4684	43	0.00
1731	44	0.00
4742	59	119152.10
4521	27	124995.98
…	…	…
6412	53	98268.84
8285	25	0.00
7853	47	0.00
1095	29	0.00
6929	39	115341.19

2000 rows × 2 columns

In [91]:
```
model = LogisticRegression(max_iter = 1000)
model.fit(X_train, y_train)
```

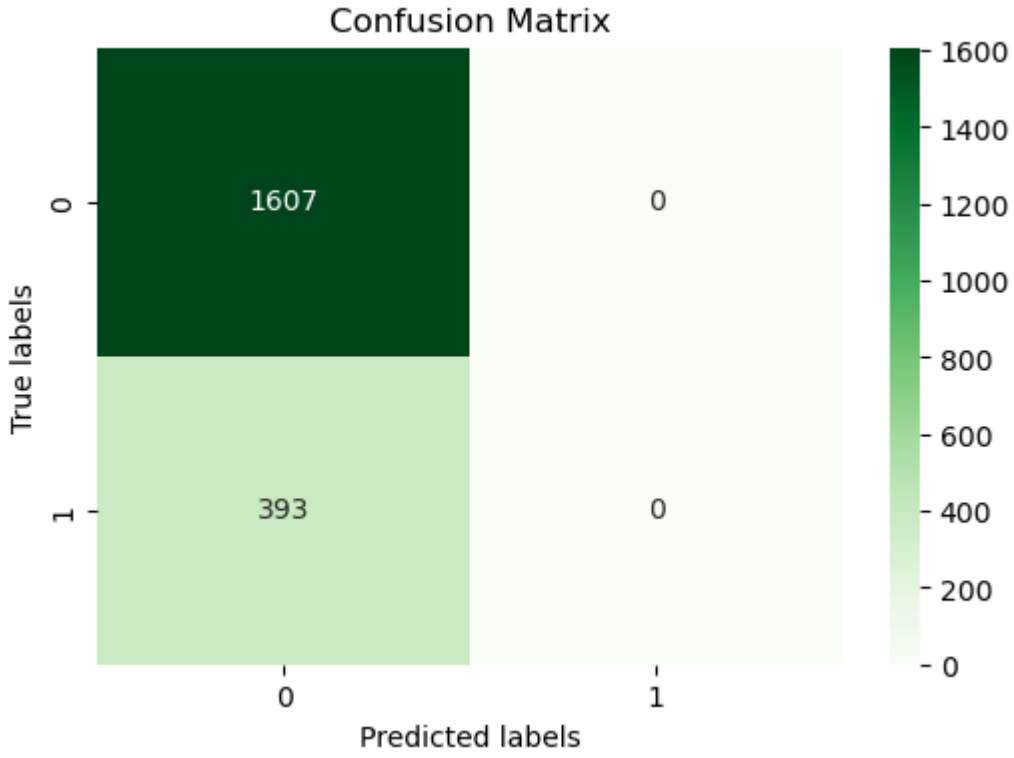
Out[91]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [92]:
```
y_pred = model.predict(X_test)
y_pred
```

Out[92]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [93]:
```
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print('Confusion Matrix:')
confusion_matrix = (confusion_matrix(y_test, y_pred))



plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Greens")
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.8035
Confusion Matrix:
```



In [94]:
```
accuracy = accuracy_score(y_pred, y_test)
accuracy
```

Out[94]: 0.8035

In [95]:
```
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

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

           0       0.80      1.00      0.89      1607
           1       0.00      0.00      0.00       393

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.45      2000
weighted avg       0.65      0.80      0.72      2000
```

```
/Users/rakeshmanawat/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1469: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/rakeshmanawat/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1469: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/rakeshmanawat/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1469: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [96]:

```
#random forest
```

In [97]:

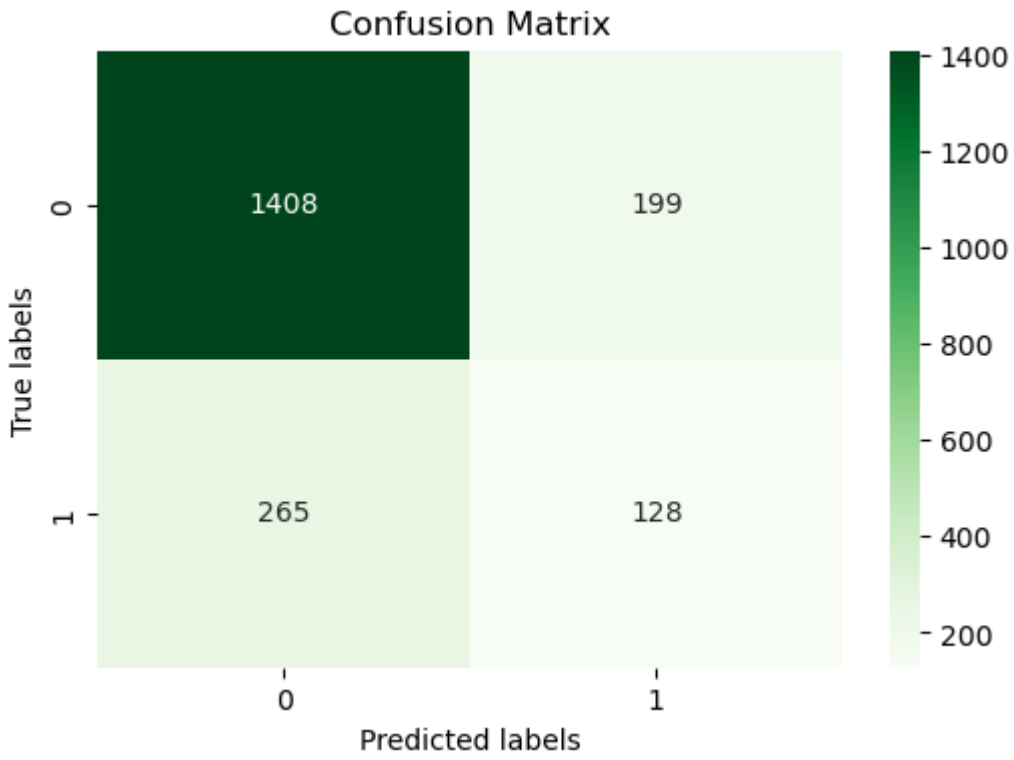
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classificat
from sklearn.metrics import r2_score

# Initialize the model
model = RandomForestClassifier(random_state=42)

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

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

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.768
```

In [98]:

```
print('Confusion Matrix:')
cm = confusion_matrix(y_test, y_pred)



plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens")
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

```
Confusion Matrix:
```



In [99]:
```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1607
           1       0.39      0.33      0.36       393

    accuracy                           0.77      2000
   macro avg       0.62      0.60      0.61      2000
weighted avg       0.75      0.77      0.76      2000
```

In [ ]:

linear regression

In [100…
```
from sklearn.linear_model import LogisticRegression
```

In [101…
```
model1 = LogisticRegression()
model1.fit(X, y)
```

Out[101…
```
▾ LogisticRegression

LogisticRegression()
```

In [102…
```
y_pred = model.predict(X)
y_pred
```

Out[102… `array([0, 0, 1, ..., 0, 1, 0])`

In [103…
```
r2 = r2_score(y, y_pred)
print(f'R-squared: {r2}')   #coefficient of determination
```

```
R-squared: 0.4710440056246886
```

In [ ]:

In [ ]:

In [ ]:

In [104…
```
# SAVE THE MODEL
```

In [107…
```
import numpy as np
import pickle
```

In [108…
```
from sklearn.linear_model import LogisticRegression

# Example: Training a model
model1 = LogisticRegression()
model1.fit(X_train, y_train)

# Saving the model to a file
with open('model1.pkl', 'wb') as file:
    pickle.dump(model, file)

# Loading the model from the file
with open('model1.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

# Using the loaded model to make predictions
predictions = loaded_model.predict(X_test)
predictions
```

Out[108… array([0, 0, 0, ..., 0, 0, 1])

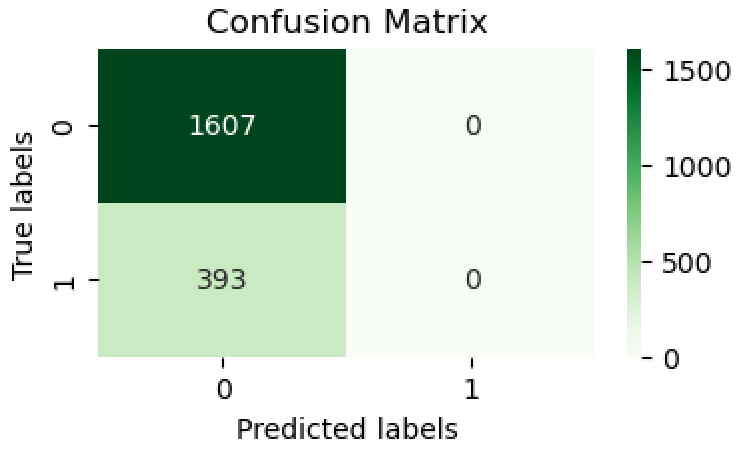
In [ ]:
```
# predictions = loaded_model.predict([[40,5000]])   #correct shape is 2D
# print(predictions)
```

In [114…
```
# Predict on the test set
y_pred = model1.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.8035
```

In [117…
```
print('Confusion Matrix:')
cm = confusion_matrix(y_test, y_pred)



plt.figure(figsize=(4, 2))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens")
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()
```

```
Confusion Matrix:
```



In [116…
```
report1 = classification_report(y_test, y_pred)
print(report1)
```

```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1607
           1       0.00      0.00      0.00       393

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.45      2000
weighted avg       0.65      0.80      0.72      2000
```

```
/Users/rakeshmanawat/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1469: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/rakeshmanawat/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1469: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/rakeshmanawat/anaconda3/lib/python3.11/site-packages/sklearn/metric
s/_classification.py:1469: UndefinedMetricWarning: Precision and F-score a
re ill-defined and being set to 0.0 in labels with no predicted samples. U
se `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]: