Work Steps

- We uploaded an Excel file into the Jupyter Lab environment.
- We checked for errors or duplicate data
- We converted the **Onboard_date** column from object type to datetime.
- We analyzed the data and created visualizations to study the data more deeply.
- We saved the visualizations into a single Excel sheet to make it easier for anyone who just wants to quickly review them.
- For machine learning, we created a new dataframe containing only numerical data, (excluding textual data since it does not help in predicting the outcome).
- We used SMOTE to balance the dataset.
- We applied GridSearchCV to find the best model hyperparameters.
- Then, we identified the best model.
- We adjusted the decision threshold to improve the results.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import math
from openpyxl import Workbook
from openpyxl.drawing.image import Image
```

In [2]:
```
# import data from sheet exel
df=pd.read_csv(r'C:\Users\youne\OneDrive\Bureau\data analys\training\churn1\custome
```

data explor

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

Out[3]:

	Names	Age	Total_Purchase	Account_Manager	Years	Num_Sites	Onboard_date	
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-08-30 07:00:40	Barl
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-08-13 00:38:46	6 Car
2	Eric Lozano	38.0	12884.75	0	6.67	12.0	2016-06-29 06:20:07	1 Alys
3	Phillip White	42.0	8010.76	0	6.71	10.0	2014-04-22 12:43:12	An W
4	Cynthia Norton	37.0	9191.58	0	5.56	9.0	2016-01-19 15:31:15	Ka M

In [4]:
```
# find nan value
df.isnull().sum()
```

Out[4]:
```
Names             0
Age               0
Total_Purchase    0
Account_Manager   0
Years             0
Num_Sites         0
Onboard_date      0
Location          0
Company           0
Churn             0
dtype: int64
```

In [5]:
```
# Find duplicate values
df.duplicated().sum()
```

Out[5]:
```
np.int64(0)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Names            900 non-null    object 
 1   Age              900 non-null    float64
 2   Total_Purchase   900 non-null    float64
 3   Account_Manager  900 non-null    int64  
 4   Years            900 non-null    float64
 5   Num_Sites        900 non-null    float64
 6   Onboard_date     900 non-null    object 
 7   Location         900 non-null    object 
 8   Company          900 non-null    object 
 9   Churn            900 non-null    int64  
dtypes: float64(4), int64(2), object(4)
memory usage: 70.4+ KB
```

In [7]:
```
# transform columns to small later
df.columns = df.columns.str.lower()
```

In [8]:
```
# transform dtype of onboard_date from object to detetime64[ns]
df['onboard_date'] = pd.to_datetime(df['onboard_date'])
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   names            900 non-null    object        
 1   age              900 non-null    float64       
 2   total_purchase   900 non-null    float64       
 3   account_manager  900 non-null    int64         
 4   years            900 non-null    float64       
 5   num_sites        900 non-null    float64       
 6   onboard_date     900 non-null    datetime64[ns]
 7   location         900 non-null    object        
 8   company          900 non-null    object        
 9   churn            900 non-null    int64         
dtypes: datetime64[ns](1), float64(4), int64(2), object(3)
memory usage: 70.4+ KB
```

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

Out[10]:

	age	total_purchase	account_manager	years	num_sites	onboard_
count	900.000000	900.000000	900.000000	900.000000	900.000000	
mean	41.816667	10062.824033	0.481111	5.273156	8.587778	2011-0 15:50:48.35888
min	22.000000	100.000000	0.000000	1.000000	3.000000	2006-0 04:
25%	38.000000	8497.122500	0.000000	4.450000	7.000000	2008-0 16:43:21.24999
50%	42.000000	10045.870000	0.000000	5.215000	8.000000	2011-0 04:36:26.50
75%	46.000000	11760.105000	1.000000	6.110000	10.000000	2014-0 00:01:16.24999
max	65.000000	18026.010000	1.000000	9.150000	14.000000	2016-1 04:(
std	6.127560	2408.644532	0.499921	1.274449	1.764836	

data analys

In [11]:
```
df.head()
```

Out[11]:

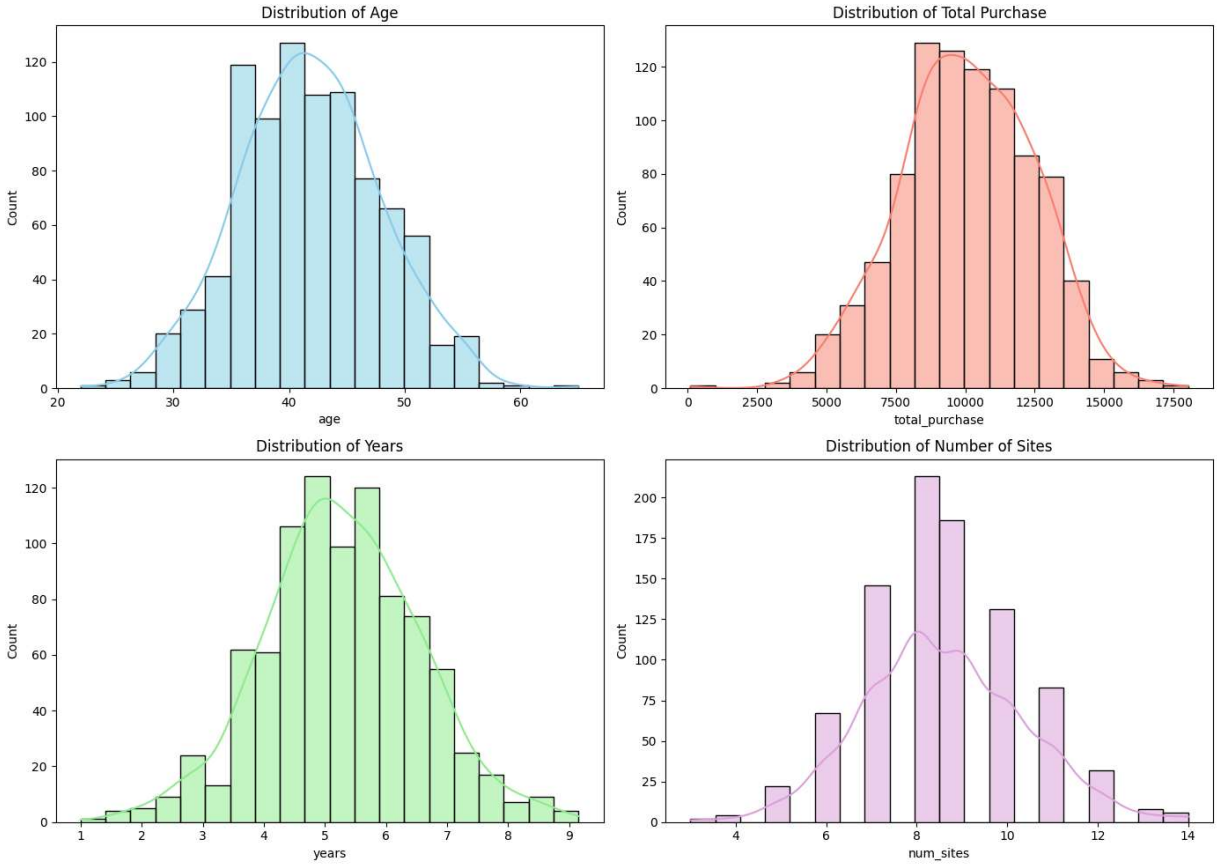
	names	age	total_purchase	account_manager	years	num_sites	onboard_date	lo
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-08-30 07:00:40	El Barke AK
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-08-13 00:38:46	615 C Su Carlo
2	Eric Lozano	38.0	12884.75	0	6.67	12.0	2016-06-29 06:20:07	133 Alyss DI
3	Phillip White	42.0	8010.76	0	6.71	10.0	2014-04-22 12:43:12	Ange WY
4	Cynthia Norton	37.0	9191.58	0	5.56	9.0	2016-01-19 15:31:15	76 Kar MH

In [57]:
```
df['total_purchase'].sum()
```

Out[57]:
```
np.float64(9056541.629999999)
```

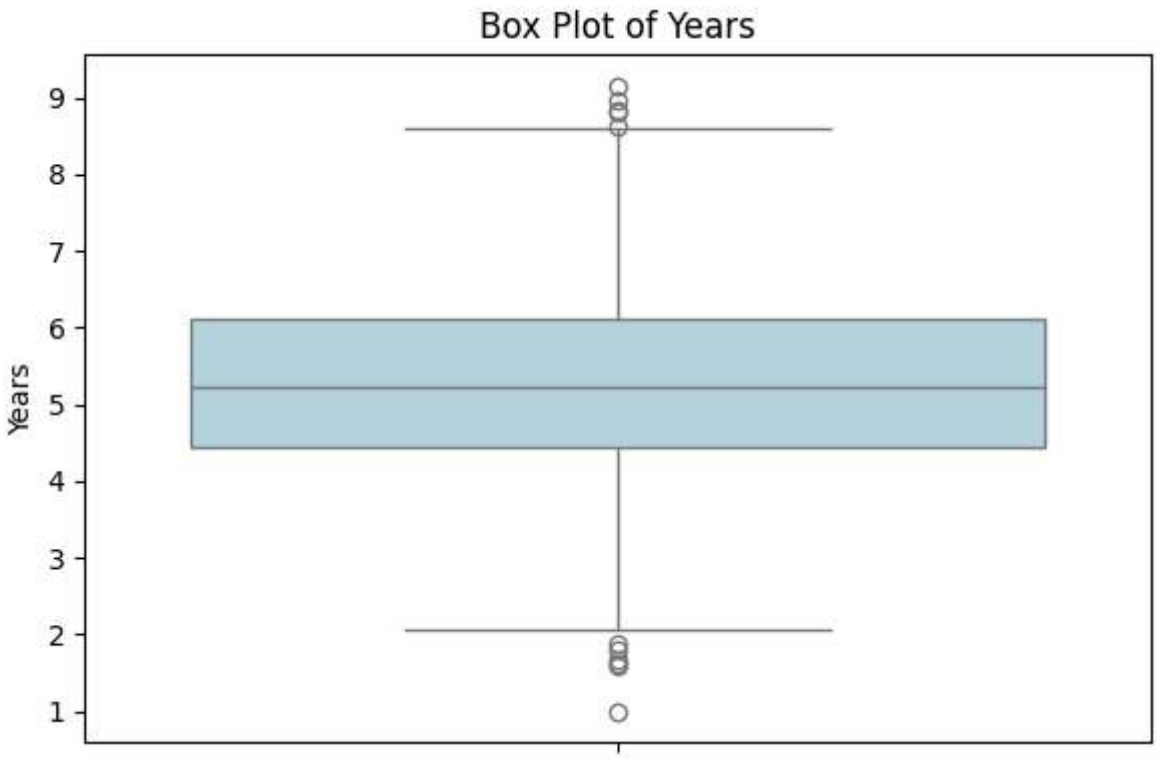
In [12]:
```
plt.figure(figsize=(14, 10))

plt.subplot(2, 2, 1)
sns.histplot(df['age'], bins=20, kde=True, color='skyblue')
plt.title('Distribution of Age')

plt.subplot(2, 2, 2)
sns.histplot(df['total_purchase'], bins=20, kde=True, color='salmon')
plt.title('Distribution of Total Purchase')

plt.subplot(2, 2, 3)
sns.histplot(df['years'], bins=20, kde=True, color='lightgreen')
plt.title('Distribution of Years')

plt.subplot(2, 2, 4)
sns.histplot(df['num_sites'], bins=20, kde=True, color='plum')
plt.title('Distribution of Number of Sites')

plt.tight_layout()
```

```
# To save the plots as PNG images
#plt.savefig('distributions.png')

plt.show()

# To save the plots in exel sheet
#wb = Workbook()
#ws = wb.active
#ws.title = "Distributions"

#img = Image("distributions.png")
#ws.add_image(img, "B2")

#wb.save("distributions_report.xlsx")
```

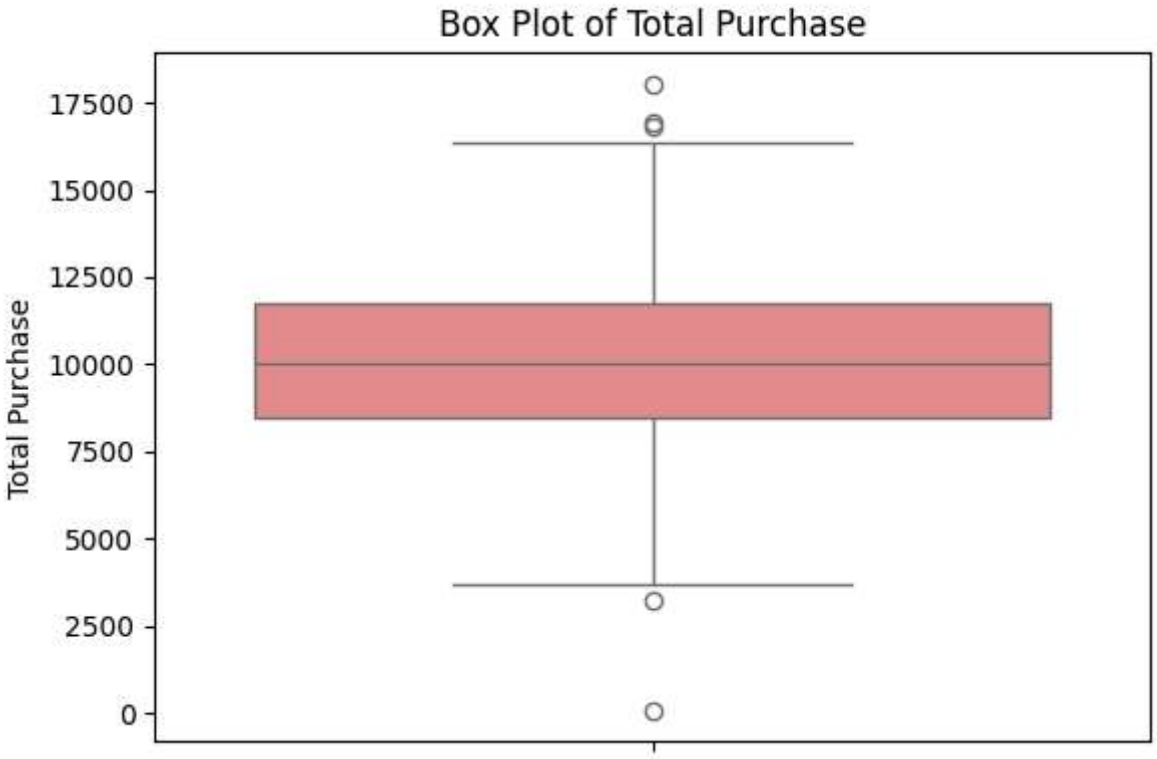


In [13]:
```
plt.figure(figsize=(6, 4))
sns.boxplot(y=df['years'], color='lightblue')
plt.title('Box Plot of Years')
plt.ylabel('Years')
plt.tight_layout()
#plt.savefig("boxplot_years.png")
plt.show()


q1 = df['years'].quantile(0.25)
q2 = df['years'].median()
q3 = df['years'].quantile(0.75)
mean = df['years'].mean()
std = df['years'].std()
```

```
# Create an Excel sheet and save the plot in it
#wb = Workbook()
#ws = wb.active
#ws.title = "BoxPlot_Years"


#img = Image("boxplot_years.png")
#ws.add_image(img, "B2")

#ws["D2"] = "Statistic"
#ws["E2"] = "Value"

#ws["D3"] = "Q1 (25th percentile)"
#ws["E3"] = round(q1, 2)

#ws["D4"] = "Q2 (Median)"
#ws["E4"] = round(q2, 2)

#ws["D5"] = "Q3 (75th percentile)"
#ws["E5"] = round(q3, 2)

#ws["D6"] = "Mean"
#ws["E6"] = round(mean, 2)

#ws["D7"] = "Std Deviation"
#ws["E7"] = round(std, 2)


#wb.save("boxplot_years_report.xlsx")
```

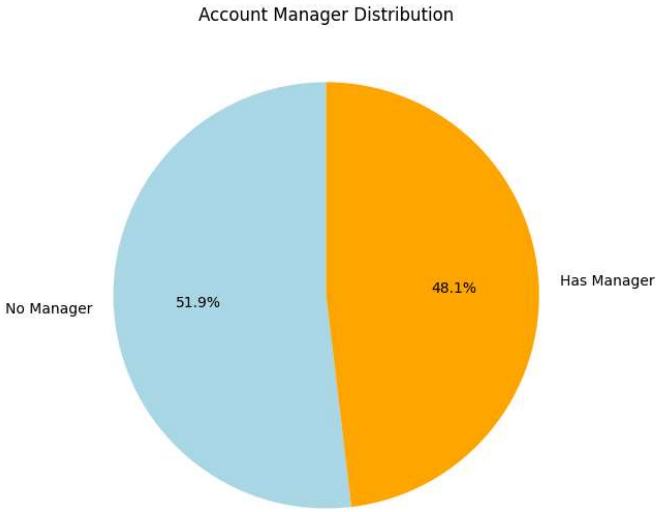


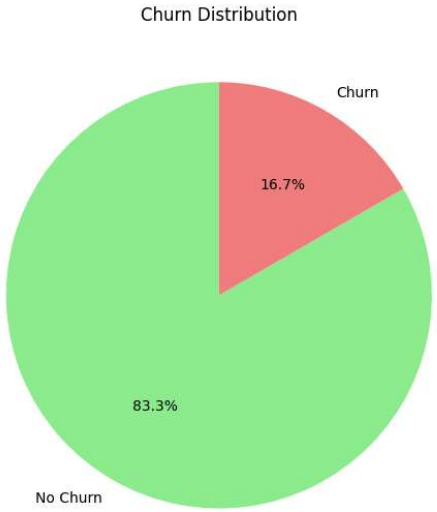
In [14]:
```
# 1. Create the box plot and save it
plt.figure(figsize=(6, 4))
sns.boxplot(y=df['total_purchase'], color='lightcoral')
plt.title('Box Plot of Total Purchase')
plt.ylabel('Total Purchase')
plt.tight_layout()
#plt.savefig("boxplot_total_purchase.png")
plt.show()

# 2. Calculate key statistics
q1 = df['total_purchase'].quantile(0.25)
q2 = df['total_purchase'].median()
q3 = df['total_purchase'].quantile(0.75)
mean = df['total_purchase'].mean()
std = df['total_purchase'].std()

# 3. Create Excel file and insert image + statistics
#wb = Workbook()
#ws = wb.active
#ws.title = "TotalPurchase_BoxPlot"

# Insert the box plot image into the Excel sheet
#img = Image("boxplot_total_purchase.png")
#ws.add_image(img, "B2")  # You can change cell location as needed

# Add the statistics next to the image
#ws["D2"] = "Statistic"
#ws["E2"] = "Value"

#ws["D3"] = "Q1 (25th percentile)"
#ws["E3"] = round(q1, 2)

#ws["D4"] = "Q2 (Median)"
#ws["E4"] = round(q2, 2)

#ws["D5"] = "Q3 (75th percentile)"
#ws["E5"] = round(q3, 2)

#ws["D6"] = "Mean"
#ws["E6"] = round(mean, 2)

#ws["D7"] = "Std Deviation"
#ws["E7"] = round(std, 2)

# Save the Excel file
#wb.save("boxplot_total_purchase_report.xlsx")
```



In [15]:
```
# Count the number of occurrences for each value in the two columns
account_manager_counts = df['account_manager'].value_counts()
churn_counts = df['churn'].value_counts()

# Create a figure with 2 pie charts side by side
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Pie chart for 'account_manager' column
axes[0].pie(account_manager_counts,
            labels=account_manager_counts.index.map({0: 'No Manager', 1: 'Has Manag
            autopct='%1.1f%%',
            colors=['lightblue', 'orange'],
            startangle=90)
axes[0].set_title('Account Manager Distribution')

# Pie chart for 'churn' column
axes[1].pie(churn_counts,
            labels=churn_counts.index.map({0: 'No Churn', 1: 'Churn'}),
            autopct='%1.1f%%',
            colors=['lightgreen', 'lightcoral'],
            startangle=90)
axes[1].set_title('Churn Distribution')

plt.tight_layout()

# Save the plot as a PNG image before displaying it
#plt.savefig('pie_charts.png')
plt.show()

# Create a new Excel workbook and add the image
#wb = Workbook()
#ws = wb.active
```

```
#ws.title = "Pie Charts"

#img = Image("pie_charts.png")
#ws.add_image(img, "B2")

# Add a short description below the image
#ws["B20"] = "Pie charts showing the distribution of Account Manager status and Chu

# Save the Excel file
#wb.save("pie_charts_report.xlsx")
```

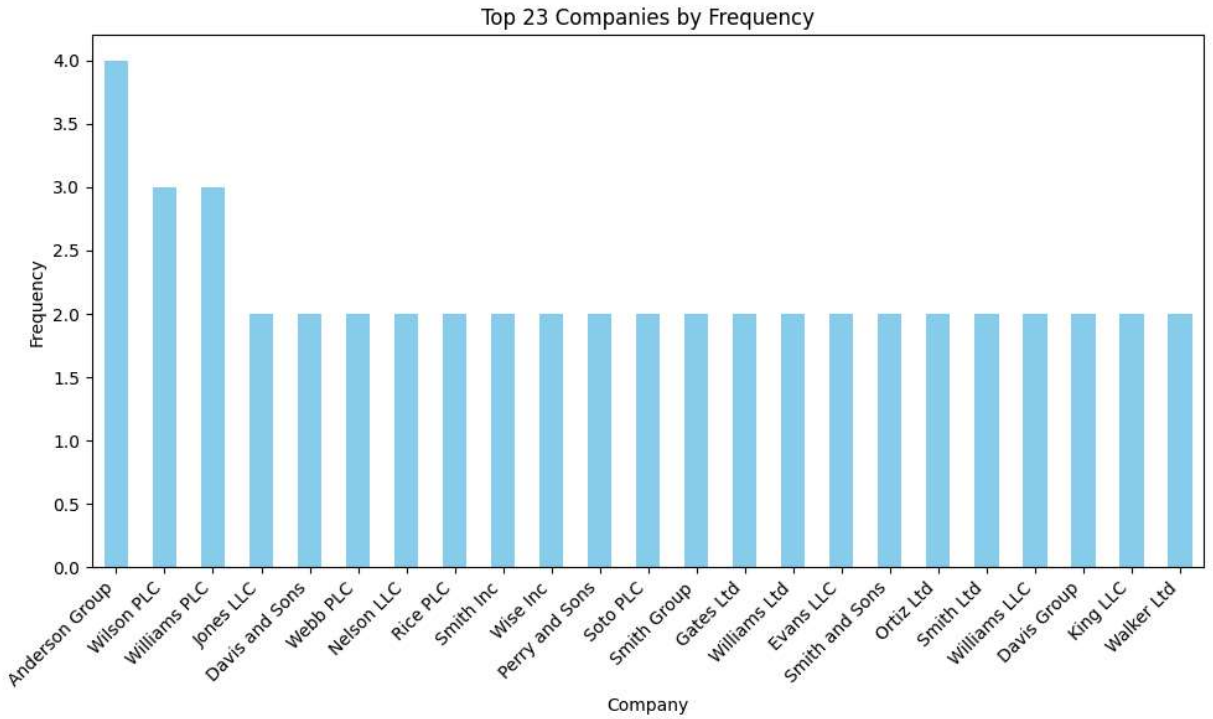




The number of churn cases is much higher than no churn, meaning the data is imbalanced.

In [16]:
```
top_n = 23
counts_top = df['company'].value_counts().nlargest(top_n)
plt.figure(figsize=(10, 6))
counts_top.plot(kind='bar', color='skyblue')
plt.xlabel('Company')
plt.ylabel('Frequency')
plt.title(f'Top {top_n} Companies by Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Save the plot as PNG
#plt.savefig("top_companies_bar_chart.png")
plt.show()

# Create a new Excel workbook and insert the plot
#wb = Workbook()
#ws = wb.active
#ws.title = "Top Companies Chart"

#img = Image("top_companies_bar_chart.png")
#ws.add_image(img, "B2")

# Add a description below the image
#ws["B30"] = f"Bar chart showing the top {top_n} companies based on frequency in th
```

```
# Save the Excel file
#wb.save("top_companies_report.xlsx")
```



In [17]:
```
# I wanted to check if there is a person who has interacted with us on two differen
#  Compute the count of each name
name_counts = df['names'].value_counts()
#  Select names that appear more than once
duplicate_names = name_counts[name_counts > 1].index.tolist()
duplicate_names
```

Out[17]:
```
['Jennifer Wood']
```

In [18]:
```
df[df['names'] == 'Jennifer Wood']
```

Out[18]:

	names	age	total_purchase	account_manager	years	num_sites	onboard_date	loca
22	Jennifer Wood	35.0	9381.12	1	6.78	11.0	2006-03-27 20:52:42	Ph H Wil 00
439	Jennifer Wood	48.0	11585.16	0	4.61	9.0	2012-03-13 03:24:04	DP 15

If we look closely at the age, it doesn't seem to be the same person(ven if it's the same person, it won't affect the machine learning training)

In [19]:
```
# Information about the person who spent the most.
df[df['total_purchase'] == df['total_purchase'].max()]
```

Out[19]:

	names	age	total_purchase	account_manager	years	num_sites	onboard_date	lo
842	Ethan Cordova	39.0	18026.01	1	3.82	9.0	2016-02-28 07:42:24	482 Mo Jas

In [20]:
```
# Information about the person who spent the least.
df[df['total_purchase'] == df['total_purchase'].min()]
```

Out[20]:

	names	age	total_purchase	account_manager	years	num_sites	onboard_date	lo
659	Kayla Reeves	38.0	100.0	0	5.27	5.0	2006-10-03 09:01:48	1511 Amb NC

In [21]:
```
# Information about the people with the highest number of sites.
df[df['num_sites'] == df['num_sites'].max()]
```

Out[21]:

	names	age	total_purchase	account_manager	years	num_sites	onboard_date	
59	Daniel Bartlett	45.0	6749.49	0	5.88	14.0	2006-08-19 17:13:19	35 M
91	Brian Young	49.0	10522.21	0	7.25	14.0	2014-11-08 00:40:36	
105	Eric Kelley	43.0	10271.19	1	5.31	14.0	2011-07-19 10:28:53	
112	Katie Sutton	40.0	11611.51	1	5.64	14.0	2010-06-19 16:02:16	56 W N
130	Douglas Joseph	44.0	8474.11	0	6.61	14.0	2012-10-01 08:10:23	
186	James Contreras	46.0	11119.11	1	3.51	14.0	2008-12-08 09:23:02	10 Je

In [22]:
```
# Information about the people with the lowest number of sites.
df[df['num_sites'] == df['num_sites'].min()]
```

Out[22]:

	names	age	total_purchase	account_manager	years	num_sites	onboard_date	lo
283	Regina Kirby	41.0	10921.54	0	5.07	3.0	2015-09-04 11:32:49	31 Und Sui Kev
829	James Stevens	44.0	12400.33	1	5.18	3.0	2007-07-18 07:06:46	Jo Fl 5

In [23]:
```
df['company'].nunique()
```

Out[23]: 873

In [24]:
```
df['location'].nunique()
```

Out[24]: 900

Analyze data of the person who churned

In [25]:
```
churn1 = df[df['churn'] == 1]
```

In [ ]:

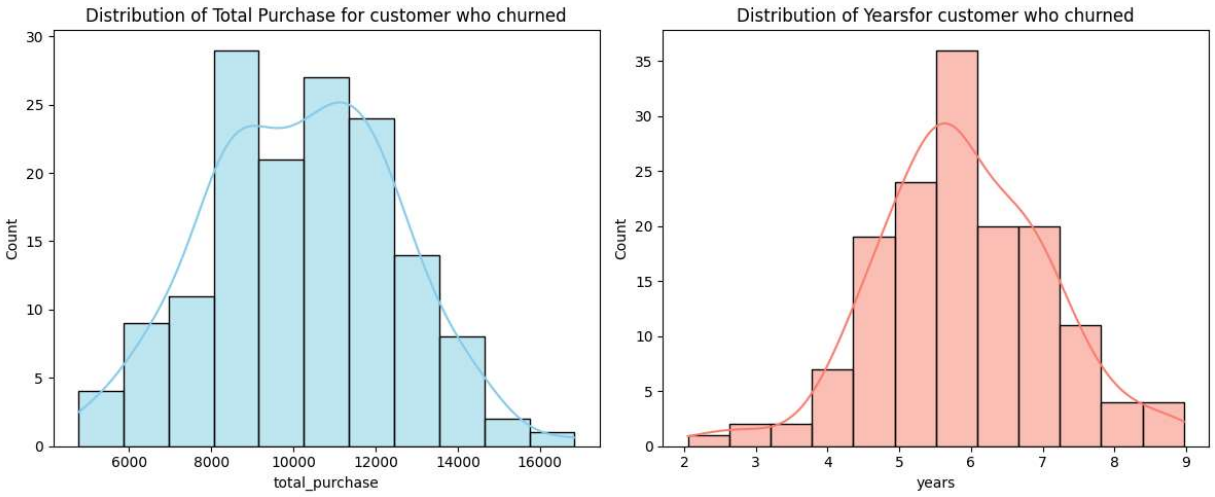
In [26]:
```
df.head()
```

Out[26]:

	names	age	total_purchase	account_manager	years	num_sites	onboard_date	lo
0	Cameron Williams	42.0	11066.80	0	7.22	8.0	2013-08-30 07:00:40	El Barke AK
1	Kevin Mueller	41.0	11916.22	0	6.50	11.0	2013-08-13 00:38:46	615 C Su Carlo
2	Eric Lozano	38.0	12884.75	0	6.67	12.0	2016-06-29 06:20:07	133 Alyss DI
3	Phillip White	42.0	8010.76	0	6.71	10.0	2014-04-22 12:43:12	Ange WY
4	Cynthia Norton	37.0	9191.58	0	5.56	9.0	2016-01-19 15:31:15	76 Kar MH

In [27]:
```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.histplot(churn1['total_purchase'], kde=True, color='skyblue')
plt.title('Distribution of Total Purchase for customer who churned')
```

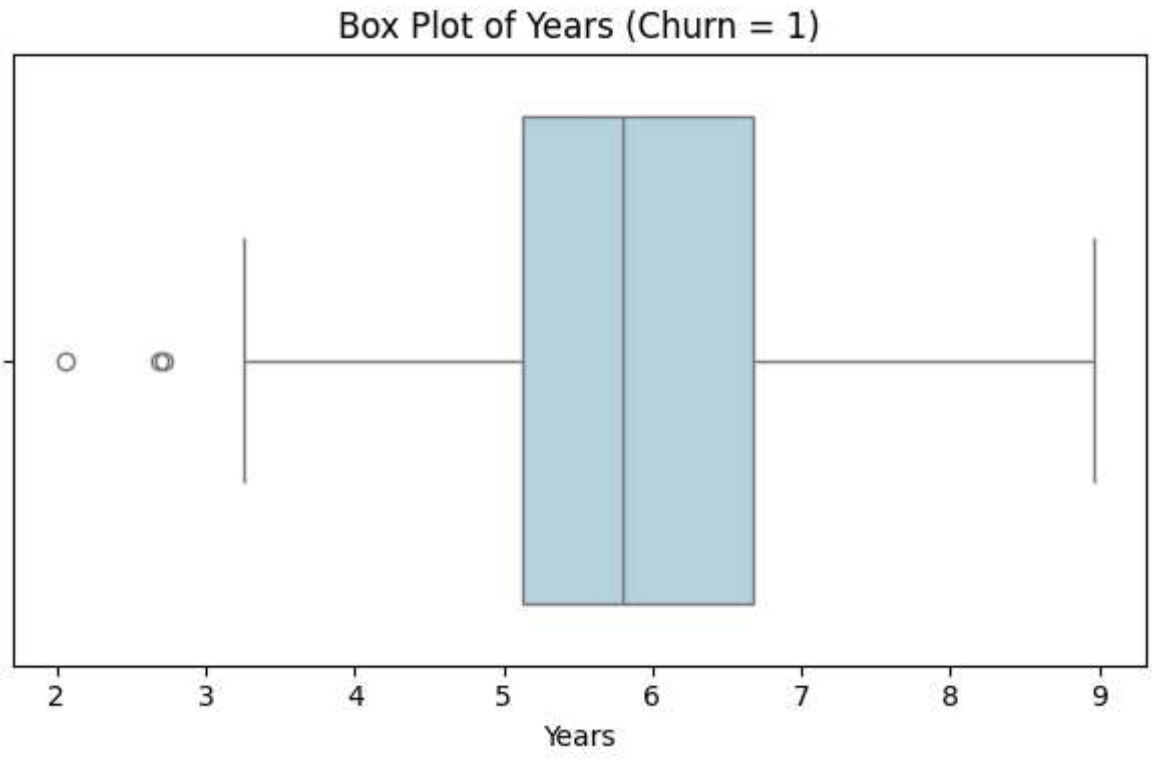
```
plt.subplot(1, 2, 2)
sns.histplot(churn1['years'], kde=True, color='salmon')
plt.title('Distribution of Yearsfor customer who churned')

plt.tight_layout()

# Save the plot as PNG
#plt.savefig("churn1_distributions.png")
plt.show()

# Create Excel workbook and insert the plot
#wb = Workbook()
#ws = wb.active
#ws.title = "Churn1 Distributions"

#img = Image("churn1_distributions.png")
#ws.add_image(img, "B2")

# Add description
#ws["B20"] = "Histograms of Total Purchase and Years for customers with churn=1."

# Save the Excel file
#wb.save("churn1_distributions_report.xlsx")
```



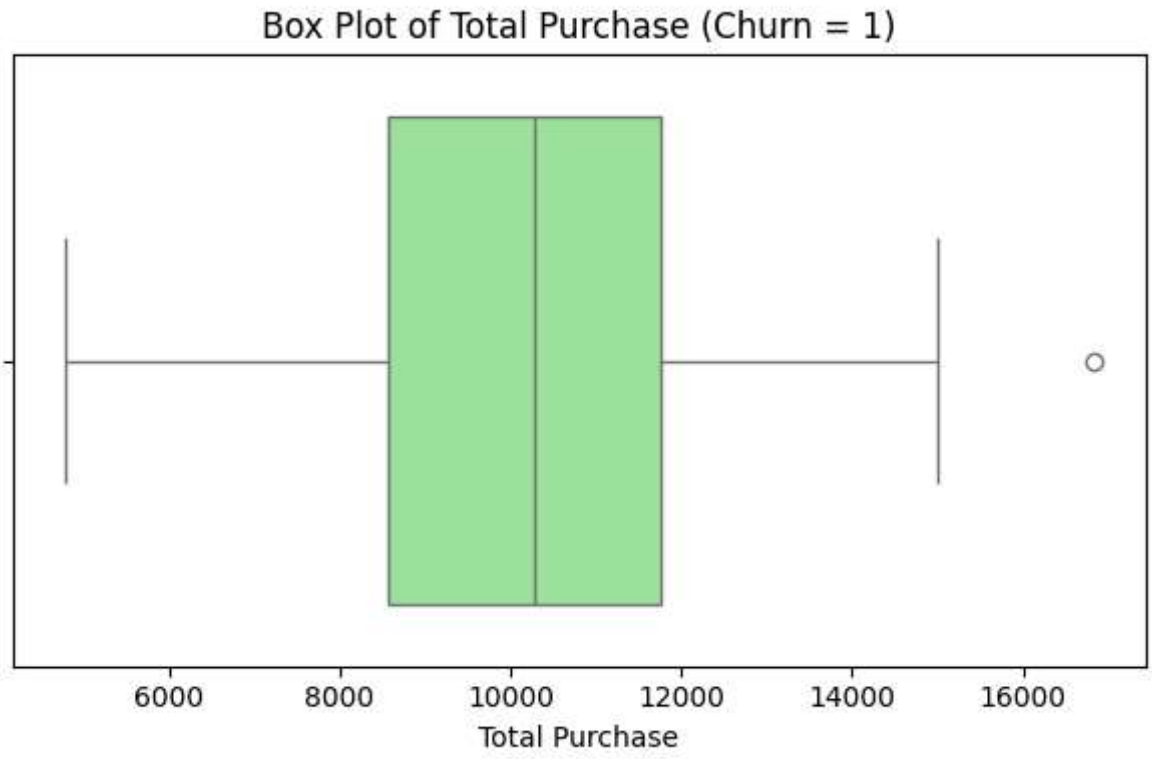
In [28]:

```
plt.figure(figsize=(6, 4))
sns.boxplot(x=churn1['years'], color='lightblue')
plt.title('Box Plot of Years (Churn = 1)')
plt.xlabel('Years')
plt.tight_layout()
#plt.savefig("boxplot_years_churn1.png")
plt.show()

# Calculate statistics
q1 = churn1['years'].quantile(0.25)
q2 = churn1['years'].quantile(0.50)  # or use median()
q3 = churn1['years'].quantile(0.75)
mean = churn1['years'].mean()
median = churn1['years'].median()
std = churn1['years'].std()
```

```
# Create Excel workbook
#wb = Workbook()
#ws = wb.active
#ws.title = "BoxPlot_Years_Churn1"

# Insert the image
#img = Image("boxplot_years_churn1.png")
#ws.add_image(img, "B2")

# Add statistics next to the plot
#ws["D2"] = "Statistic"
#ws["E2"] = "Value"

#ws["D3"] = "Q1 (25%)"
#ws["E3"] = round(q1, 2)

#ws["D4"] = "Q2 (Median - 50%)"
#ws["E4"] = round(q2, 2)

#ws["D5"] = "Q3 (75%)"
#ws["E5"] = round(q3, 2)

#ws["D6"] = "Mean"
#ws["E6"] = round(mean, 2)

#ws["D7"] = "Standard Deviation"
#ws["E7"] = round(std, 2)

# Save Excel file
#wb.save("boxplot_years_churn1_report.xlsx")
```

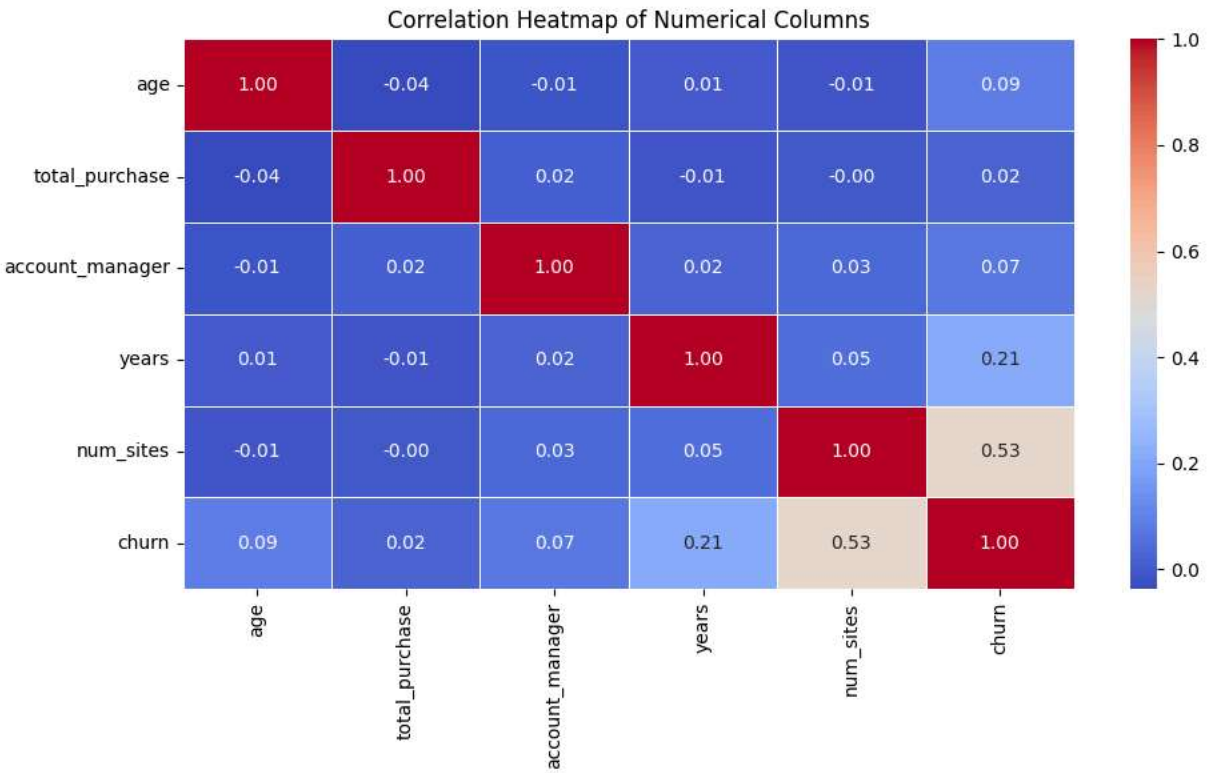


In [29]:
```
plt.figure(figsize=(6, 4))
sns.boxplot(x=churn1['total_purchase'], color='lightgreen')
plt.title('Box Plot of Total Purchase (Churn = 1)')
plt.xlabel('Total Purchase')
plt.tight_layout()
#plt.savefig("boxplot_total_purchase_churn1.png")
plt.show()

# Calculate statistics
q1 = churn1['total_purchase'].quantile(0.25)
q2 = churn1['total_purchase'].quantile(0.50)  # Median
q3 = churn1['total_purchase'].quantile(0.75)
mean = churn1['total_purchase'].mean()
median = churn1['total_purchase'].median()
std = churn1['total_purchase'].std()

# Create an Excel workbook and worksheet
#wb = Workbook()
#ws = wb.active
#ws.title = "BoxPlot_Purchase_Churn1"

# Insert the image
#img = Image("boxplot_total_purchase_churn1.png")
#ws.add_image(img, "B2")

# Write summary statistics
#ws["D2"] = "Statistic"
#ws["E2"] = "Value"

#ws["D3"] = "Q1 (25%)"
#ws["E3"] = round(q1, 2)

#ws["D4"] = "Q2 (Median - 50%)"
#ws["E4"] = round(q2, 2)

#ws["D5"] = "Q3 (75%)"
#ws["E5"] = round(q3, 2)

#ws["D6"] = "Mean"
#ws["E6"] = round(mean, 2)

#ws["D7"] = "Standard Deviation"
#ws["E7"] = round(std, 2)

# Save the Excel file
#wb.save("boxplot_total_purchase_churn1_report.xlsx")
```



In [ ]:

In [30]:
```
# Compute the correlation matrix
correlation_matrix = df.corr(numeric_only=True)

# Plot the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=
plt.title('Correlation Heatmap of Numerical Columns')
plt.tight_layout()
#plt.savefig("correlation_heatmap.png")
plt.show()

# Create an Excel workbook and insert the heatmap
#wb = Workbook()
#ws = wb.active
#ws.title = "Correlation Heatmap"

#img = Image("correlation_heatmap.png")
#ws.add_image(img, "B2")

# Optional: Add description
#ws["B20"] = "This heatmap shows the Pearson correlation coefficients between numer

# Save the Excel file
#wb.save("correlation_heatmap_report.xlsx")
```



learn machin

In [31]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score , recall_score , f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

In [32]:
```
# Feature Scaling
scaler = MinMaxScaler()
data = df.copy()
data[['total_purchase']] = scaler.fit_transform(data[['total_purchase']])
#data[['age']] = scaler.fit_transform(data[['age']])
```

In [33]:
```
data = data.drop(['names', 'location', 'company','onboard_date'], axis=1)
```

In [34]:
```
data.head()
```

Out[34]:

	age	total_purchase	account_manager	years	num_sites	churn
0	42.0	0.611781	0	7.22	8.0	1
1	41.0	0.659166	0	6.50	11.0	1
2	38.0	0.713196	0	6.67	12.0	1
3	42.0	0.441301	0	6.71	10.0	1
4	37.0	0.507173	0	5.56	9.0	1

In [35]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype         
---  ------           --------------  -----         
 0   names            900 non-null    object        
 1   age              900 non-null    float64       
 2   total_purchase   900 non-null    float64       
 3   account_manager  900 non-null    int64         
 4   years            900 non-null    float64       
 5   num_sites        900 non-null    float64       
 6   onboard_date     900 non-null    datetime64[ns]
 7   location         900 non-null    object        
 8   company          900 non-null    object        
 9   churn            900 non-null    int64         
dtypes: datetime64[ns](1), float64(4), int64(2), object(3)
memory usage: 70.4+ KB
```

model creating

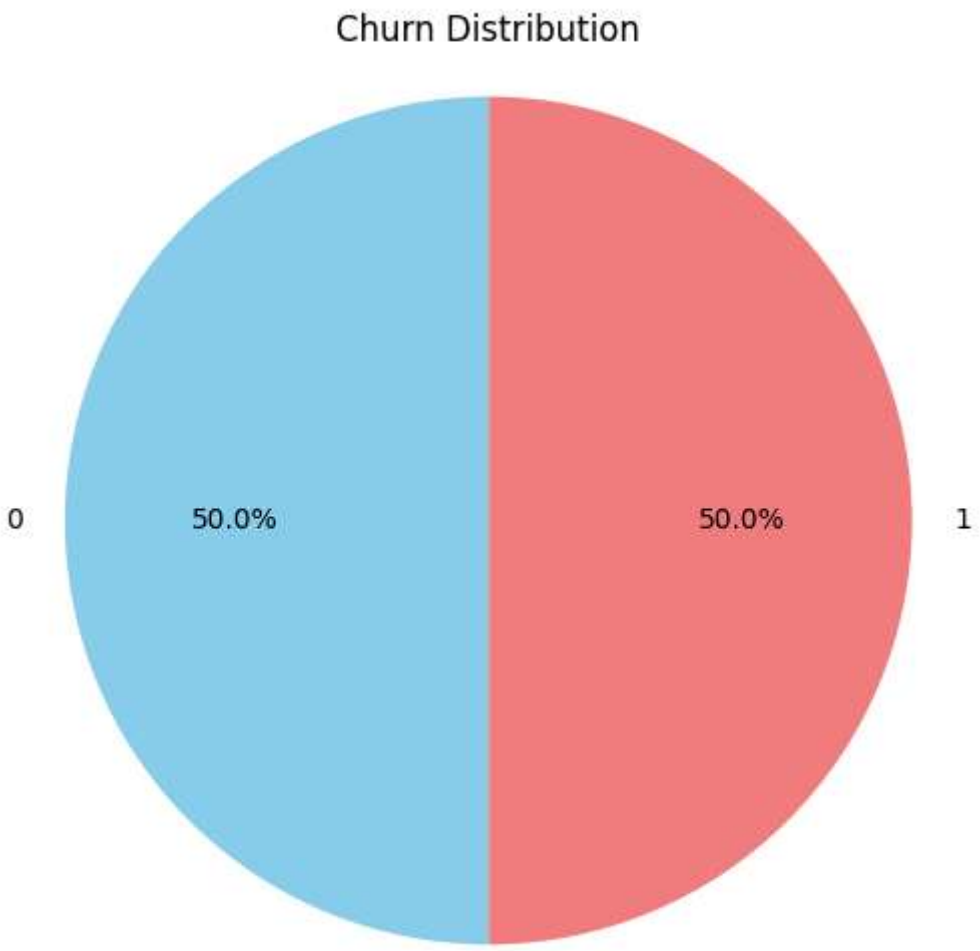
In [36]:
```
# split data
x = data.drop('churn', axis=1)
y = data['churn']
```

In [37]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=21)
```

In [38]:
```
# Data balance to become number of nochurned=churned
smote = SMOTE(random_state=42)
x_train, y_train = smote.fit_resample(x_train, y_train)
```

In [39]:
```
churn_counts = y_train.value_counts()


plt.figure(figsize=(6, 6))
plt.pie(churn_counts, labels=churn_counts.index, autopct='%1.1f%%', startangle=90,
plt.title('Churn Distribution')
```

```
plt.axis('equal')  # يجعل الدائرة فعلاً دائرية
plt.show()
```



In [40]:
```
# finding the best parameters in randomforestclassifier model
rf = RandomForestClassifier(random_state=42)


param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4] }

grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf,
                              cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

grid_search_rf.fit(x_train, y_train)

print("Best parameters for RandomForestClassifier:")
print(grid_search_rf.best_params_)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters for RandomForestClassifier:
{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
```

In [41]:
```
# finding the best parameters in GradientBoostingClassifier model
gbc = GradientBoostingClassifier(random_state=42)

param_grid_gbc = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0]
}

search = RandomizedSearchCV(gbc, param_distributions=param_grid_gbc,
                            n_iter=10, cv=3, scoring='accuracy', n_jobs=-1, verbose

search.fit(x_train, y_train)
print("Best parameters:", search.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best parameters: {'subsample': 0.8, 'n_estimators': 200, 'max_depth': 5, 'learning_r
ate': 0.1}
```

In [42]:
```
#finding the best parameters in SVC model
svc = SVC(random_state=42)

param_grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto'],

}

search = RandomizedSearchCV(svc, param_distributions=param_grid_svc,
                            n_iter=6, cv=3, scoring='accuracy', n_jobs=-1, verbose=

search.fit(x_train, y_train)

print("Best parameters for SVC:")
print(search.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best parameters for SVC:
{'kernel': 'rbf', 'gamma': 'scale', 'C': 10}
```

In [43]:
```
# finding the best parameters in LogisticRegression model
log_reg = LogisticRegression(solver='liblinear', random_state=42)

param_grid_log = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2']
}

grid_search_log = GridSearchCV(estimator=log_reg, param_grid=param_grid_log,
                               cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

grid_search_log.fit(x_train, y_train)
```

```
print("Best parameters for Logistic Regression:")
print(grid_search_log.best_params_)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for Logistic Regression:
{'C': 10, 'penalty': 'l1'}
```

In [44]:
```
# finding the best parameters in KNeighborsClassifier model
knn = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid_search_knn = GridSearchCV(estimator=knn, param_grid=param_grid_knn,
                               cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

grid_search_knn.fit(x_train, y_train)

print("Best parameters for KNeighborsClassifier:")
print(grid_search_knn.best_params_)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Best parameters for KNeighborsClassifier:
{'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
```

In [45]:
```
# call models with best parameters
rf = RandomForestClassifier(
    max_depth=10,
    min_samples_leaf=1,
    min_samples_split=2,
    n_estimators=50,
    random_state=42)



gbc = GradientBoostingClassifier(
    subsample=0.8,
    n_estimators=200,
    max_depth=5,
    learning_rate=0.1,
    random_state=42)


knn = KNeighborsClassifier(
    metric='manhattan',
    n_neighbors=3,
    weights='distance')


logreg = LogisticRegression(
    C=10,
    penalty='l1',
    solver='liblinear',
    random_state=42)
```

```
svc = SVC(
    kernel='rbf',
    gamma='scale',
    C=10,
    random_state=42)
```

In [46]:
```
def pred(model):
    model.fit(x_train,y_train)
    pre=model.predict(x_test)
    print(f"classification_report: {classification_report(y_test, pre)}")
    print(f"Mean Squared Error: {mean_squared_error(y_test, pre)}")
    print(f"Confusion Matrix:\n{confusion_matrix(y_test, pre)}")
    print(f"Accuracy Score: {accuracy_score(y_test, pre)}")
    print(f"Recall Score: {recall_score(y_test, pre)}")
    print(f"F1 Score: {f1_score(y_test, pre)}")
```

In [47]:
```
pred(rf)
```

```
classification_report:               precision    recall  f1-score   support

           0       0.93      0.90      0.92       150
           1       0.57      0.67      0.62        30

    accuracy                           0.86       180
   macro avg       0.75      0.78      0.77       180
weighted avg       0.87      0.86      0.87       180

Mean Squared Error: 0.1388888888888889
Confusion Matrix:
[[135  15]
 [ 10  20]]
Accuracy Score: 0.8611111111111112
Recall Score: 0.6666666666666666
F1 Score: 0.6153846153846154
```

In [48]:
```
pred(gbc)
```

```
classification_report:               precision    recall  f1-score   support

           0       0.93      0.91      0.92       150
           1       0.61      0.67      0.63        30

    accuracy                           0.87       180
   macro avg       0.77      0.79      0.78       180
weighted avg       0.88      0.87      0.87       180

Mean Squared Error: 0.12777777777777777
Confusion Matrix:
[[137  13]
 [ 10  20]]
Accuracy Score: 0.8722222222222222
Recall Score: 0.6666666666666666
F1 Score: 0.6349206349206349
```

In [49]:
```
pred(knn)
```

```
classification_report:               precision    recall  f1-score   support

           0       0.92      0.86      0.89       150
           1       0.47      0.63      0.54        30

    accuracy                           0.82       180
   macro avg       0.70      0.75      0.72       180
weighted avg       0.85      0.82      0.83       180

Mean Squared Error: 0.17777777777777778
Confusion Matrix:
[[129  21]
 [ 11  19]]
Accuracy Score: 0.8222222222222222
Recall Score: 0.6333333333333333
F1 Score: 0.5428571428571428
```

In [50]:
```
pred(logreg)
```

```
classification_report:               precision    recall  f1-score   support

           0       0.96      0.83      0.89       150
           1       0.49      0.83      0.62        30

    accuracy                           0.83       180
   macro avg       0.73      0.83      0.75       180
weighted avg       0.88      0.83      0.84       180

Mean Squared Error: 0.17222222222222222
Confusion Matrix:
[[124  26]
 [  5  25]]
Accuracy Score: 0.8277777777777777
Recall Score: 0.8333333333333334
F1 Score: 0.6172839506172839
```

In [51]:
```
pred(svc)
```

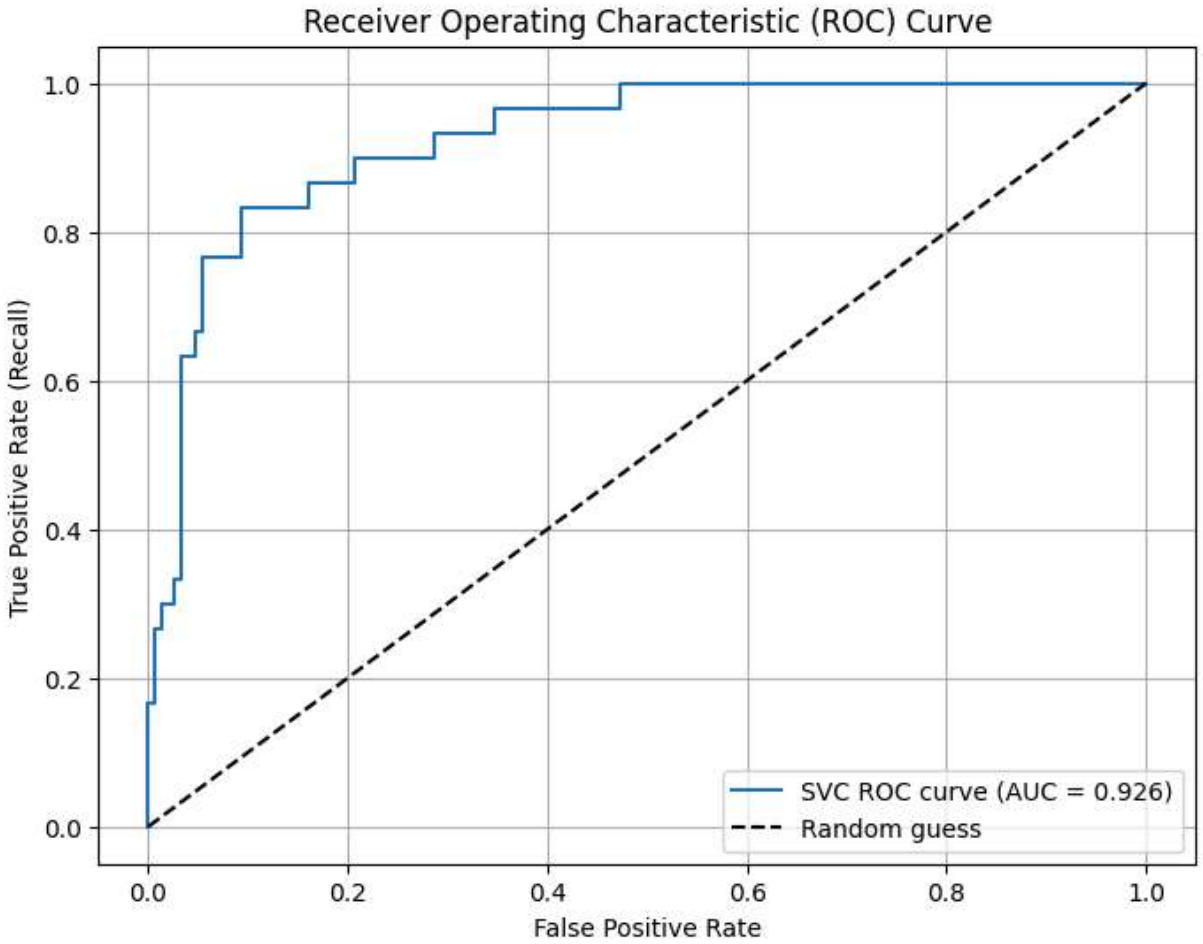
```
classification_report:               precision    recall  f1-score   support

           0       0.97      0.84      0.90       150
           1       0.52      0.87      0.65        30

    accuracy                           0.84       180
   macro avg       0.74      0.85      0.78       180
weighted avg       0.89      0.84      0.86       180

Mean Squared Error: 0.15555555555555556
Confusion Matrix:
[[126  24]
 [  4  26]]
Accuracy Score: 0.8444444444444444
Recall Score: 0.8666666666666667
F1 Score: 0.65
```

the best model is svc

Trying to change threshold To balance the results between true Positives true negatives

In [52]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [53]:
```
y_scores = svc.decision_function(x_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
auc_score = roc_auc_score(y_test, y_scores)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'SVC ROC curve (AUC = {auc_score:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



In [54]:
```
# searching for the best threshold
decision_scores = svc.decision_function(x_test)
for thresh in [0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75]:
    y_pred_thresh = (decision_scores >= thresh).astype(int)
    cm = confusion_matrix(y_test, y_pred_thresh)
    tn, fp, fn, tp = cm.ravel()
```

```
    print(f"\nThreshold: {thresh}")
    print(f"TN: {tn}, FP: {fp}, FN: {fn}, TP: {tp}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred_thresh):.2f}")
    print(f"Recall: {recall_score(y_test, y_pred_thresh):.2f}")
    print(f"F1 Score: {f1_score(y_test, y_pred_thresh):.2f}")
```

```
Threshold: 0.35
TN: 136, FP: 14, FN: 5, TP: 25
Accuracy: 0.89
Recall: 0.83
F1 Score: 0.72

Threshold: 0.4
TN: 136, FP: 14, FN: 7, TP: 23
Accuracy: 0.88
Recall: 0.77
F1 Score: 0.69

Threshold: 0.45
TN: 139, FP: 11, FN: 7, TP: 23
Accuracy: 0.90
Recall: 0.77
F1 Score: 0.72

Threshold: 0.5
TN: 139, FP: 11, FN: 7, TP: 23
Accuracy: 0.90
Recall: 0.77
F1 Score: 0.72

Threshold: 0.55
TN: 140, FP: 10, FN: 7, TP: 23
Accuracy: 0.91
Recall: 0.77
F1 Score: 0.73

Threshold: 0.6
TN: 142, FP: 8, FN: 7, TP: 23
Accuracy: 0.92
Recall: 0.77
F1 Score: 0.75

Threshold: 0.65
TN: 142, FP: 8, FN: 9, TP: 21
Accuracy: 0.91
Recall: 0.70
F1 Score: 0.71

Threshold: 0.7
TN: 142, FP: 8, FN: 10, TP: 20
Accuracy: 0.90
Recall: 0.67
F1 Score: 0.69

Threshold: 0.75
TN: 143, FP: 7, FN: 10, TP: 20
Accuracy: 0.91
Recall: 0.67
F1 Score: 0.70
```

0.6 is the best threshold

In [55]:
```
# train the model with 0.6 threshold
def call(model):
    threshold = 0.6
    model.fit(x_train, y_train)
    decision_scores = model.decision_function(x_test)
    y_pred_custom = (decision_scores >= threshold).astype(int)

    print(f"Threshold: {threshold}")
    print(f"classification_report:\n{classification_report(y_test, y_pred_custom)}"
    print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred_custom)}")
    print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred_custom)}")
    print(f"Accuracy Score: {accuracy_score(y_test, y_pred_custom)}")
    print(f"Recall Score: {recall_score(y_test, y_pred_custom)}")
    print(f"F1 Score: {f1_score(y_test, y_pred_custom)}")
```

In [56]:
```
call(svc)
```

```
Threshold: 0.6
classification_report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       150
           1       0.74      0.77      0.75        30

    accuracy                           0.92       180
   macro avg       0.85      0.86      0.85       180
weighted avg       0.92      0.92      0.92       180

Mean Squared Error: 0.08333333333333333
Confusion Matrix:
[[142   8]
 [  7  23]]
Accuracy Score: 0.9166666666666666
Recall Score: 0.7666666666666667
F1 Score: 0.7540983606557377
```

We observe an improvement in the results.

In [ ]:
```
```

In [ ]:
```
#new_data_processed = scaler.transform(#put your new data hare)


#predictions = model.predict(new_data_processed)


#new_df_with_predictions = new_df.copy()
#new_df_with_predictions['Churn_Predicted'] = predictions


#print(new_df_with_predictions.head())
```

Final Report

After analyzing the data, we concluded the following:

- The age of our users ranges from 20 to over 60 years, but the majority are between approximately 25 and 55 years old.
- The total spending reached **nine million and fifty-six thousand**.
- 90% of users spent between **8,497 and 11,760**.
- The highest recorded spending reached **eighteen thousand**.
- 90% of our users have been with us for **4 to 6 years**.
- The churn rate is **16.7%**, while **83.3%** of users stayed — which is a good retention rate.
- 90% of the users who churned had been with us for **five to six years**, which is a positive indicator. Their spending ranged between **eight thousand and eleven thousand**.

Machine Learning Performance

The **SVC (Support Vector Classifier)** model delivered strong performance.

After adjusting the threshold, the final results were as follows:

- **77%** of users who actually churned were correctly identified by the model (**recall**).
- **74%** of the users predicted to churn had in fact churned (**precision**).
- The prediction accuracy for users who did not churn reached **95%.**
- The overall accuracy of the model was **91%.**
- The error rate was **0.08** (Mean Squared Error).

Confusion Matrix Breakdown:

- **True Negatives (TN) = 142**: Correctly identified as users who did not churn.
- **False Positives (FP) = 8**: Incorrectly predicted to churn, but they actually stayed.
- **False Negatives (FN) = 7**: The model failed to identify 7 users who actually churned.
- **True Positives (TP) = 23**: Correctly identified as users who churned.

- These results indicate that the **SVC model**, with the threshold adjusted to **0.6**, is capable of achieving an excellent balance between minimizing customer loss (through high **recall**) and reducing false alarms (through good **precision**). This makes it a strong candidate for use in **customer retention strategies.**

In [ ]:

In [ ]: