In [1]:
```
# Import necessary libraries
import psycopg2 # PostgreSQL database adapter for Python
import pandas as pd # Data manipulation and analysis library
import numpy as np # Numerical computing library
from warnings import simplefilter # For warning control
simplefilter('ignore')#ignore warning to keep output clean
```

In [2]:
```
# Establish connection to postgreSQL database
connection = psycopg2.connect(
    host="branchhomeworkdb.cv8nj4hg6yra.ap-south-1.rds.amazonaws.com",
    database="branchdsprojectgps",
    user="datascientist",
    password="47eyYBLT0laW5j9U24Uuy8gLcrN")

# Define SQL queries to extract data from different table
query1 = "SELECT * FROM loan_outcomes"  # Loan application outcomes
query2 = "SELECT * FROM gps_fixes" # GPS location data
query3 = "SELECT * FROM user_attributes"  # User demographic data

# Execute queries and load results into pandas DataFrames
df_loan = pd.read_sql(query1, connection)  # Loan outcomes data
df_gps = pd.read_sql(query2, connection)  # GPS location data
df_userattributes = pd.read_sql(query3, connection) # User attributes data
```

In [3]:
```
# Display first few rows of loan outcomes data
df_loan.head()
```

Out[3]:

	user_id	application_at	loan_outcome
0	1	2017-08-14 09:08:50.000000	defaulted
1	2	2016-05-17 10:10:12.447976	repaid
2	3	2016-10-20 10:07:20.459081	defaulted
3	4	2017-01-13 13:03:34.000000	defaulted
4	5	2016-11-03 15:41:39.124610	repaid

In [4]:
```
# Print dimensions and unique value counts for Loan outcomes data
print('The dimension of the Loan_outcome is:')
print(df_loan.shape)  # Shows (rows, columns)
print('The number of unique entries is:')
print(df_loan.nunique())  # Counts unique values per column
```

```
The dimension of the Loan_outcome is:
(400, 3)
The number of unique entries is:
user_id           400
application_at    400
loan_outcome        2
dtype: int64
```

In [5]:
```
# Display first few rows of user attributes data
df_gps.head()
```

Out[5]:

	gps_fix_at	server_upload_at	longitude	latitude	accuracy	altitude	bearing	location_provider	user_id
0	2017-06-22 09:37:20	2017-06-22 09:43:42	36.840540	-1.294342	68.4	0.0	0.0	fused	1
1	2017-08-14 07:50:27	2017-08-14 09:05:27	36.895270	-1.341928	1409.0	0.0	0.0	fused	1
2	2017-06-13 10:34:29	2017-06-13 10:54:48	36.811903	-1.307220	68.4	0.0	0.0	fused	1
3	2017-06-18 12:16:20	2017-06-18 12:16:24	36.907049	-1.309984	1581.0	0.0	0.0	fused	1
4	2017-06-28 09:39:08	2017-06-28 09:58:12	36.839396	-1.280310	1396.0	0.0	0.0	fused	1

In [6]:
```
# Print dimensions and unique value counts for gps
print('The dimension of the GPS table is:')
print(df_gps.shape)  # Shows (rows, columns)
print('The number of unique entries is:')
print(df_gps.nunique()) # Counts unique values per column
```

```
The dimension of the GPS table is:
(26710, 9)
The number of unique entries is:
gps_fix_at           22057
server_upload_at     22274
longitude            26297
latitude             26069
accuracy              5065
altitude              2972
bearing                796
location_provider        4
user_id                372
dtype: int64
```

In [7]:
```
# Display first few rows of user attributes data
df_userattributes.head()
```

Out[7]:

	user_id	age	cash_incoming_30days
0	1	42	8988.12
1	2	36	9968.12
2	3	27	59.04
3	4	38	2129.03
4	5	33	2102.53

In [8]:
```
# Print dimensions and unique value counts for user attributes
print('The dimension of the User attributes is:')
print(df_userattributes.shape) # Shows (rows, columns)
print('The number of unique entries is:')
print(df_userattributes.nunique()) # Counts unique values per column
```

```
The dimension of the User attributes is:
(400, 3)
The number of unique entries is:
user_id                 400
age                      56
cash_incoming_30days    400
dtype: int64
```

In [9]:
```
# Close the database connection
connection.close()
```

In [10]:
```
# Merge loan outcomes with user attributes on user_id
merge_df1 = pd.merge(
    df_loan,
    df_userattributes,
    on="user_id",
    how="inner"  # Merge loan outcomes with user attributes on user_id
)
merge_df1.head()
```

Out[10]:

	user_id	application_at	loan_outcome	age	cash_incoming_30days
0	1	2017-08-14 09:08:50.000000	defaulted	42	8988.12
1	2	2016-05-17 10:10:12.447976	repaid	36	9968.12
2	3	2016-10-20 10:07:20.459081	defaulted	27	59.04
3	4	2017-01-13 13:03:34.000000	defaulted	38	2129.03
4	5	2016-11-03 15:41:39.124610	repaid	33	2102.53

In [11]:
```
# Aggregate GPS data by user_id (calculating mean values)
gps_fixes_agg_df = df_gps.groupby("user_id").agg({
    "latitude": "mean",
    "longitude": "mean",
    "accuracy": "mean"
}).reset_index()

# Merge the aggregated GPS data with the previously merged DataFrame
final_merged_df = pd.merge(
    merge_df1, gps_fixes_agg_df,
    on="user_id",
    how="inner")  # Inner join to keep only complete records

# Display the first few rows of the final merged DataFrame
final_merged_df.head()
```

Out[11]:

	user_id	application_at	loan_outcome	age	cash_incoming_30days	latitude	longitude	accuracy
0	1	2017-08-14 09:08:50.000000	defaulted	42	8988.12	-1.270427	36.782813	1105.084571
1	2	2016-05-17 10:10:12.447976	repaid	36	9968.12	-1.488055	37.118432	48.596000
2	3	2016-10-20 10:07:20.459081	defaulted	27	59.04	-0.889673	35.707550	6.500000
3	4	2017-01-13 13:03:34.000000	defaulted	38	2129.03	-0.306798	36.082255	2172.200000
4	5	2016-11-03 15:41:39.124610	repaid	33	2102.53	17.800041	-12.370879	43.461111

In [12]:

```
# Check for NULL values in the final DataFrame
print('The number of NULL entires is:')
final_merged_df.isnull().sum()
```

```
The number of NULL entires is:
```

Out[12]:

```
user_id                 0
application_at          0
loan_outcome            0
age                     0
cash_incoming_30days    0
latitude                0
longitude               0
accuracy                0
dtype: int64
```

In [13]:

```
# Generate descriptive statistics for numerical columns
final_merged_df.describe()
```

Out[13]:

	user_id	application_at	age	cash_incoming_30days	latitude	longitude	accuracy
count	372.000000	372	372.000000	372.000000	372.000000	372.000000	372.000000
mean	204.403226	2017-08-07 17:50:29.754975232	36.559140	7781.500108	-0.986529	36.522056	1064.288566
min	1.000000	2015-06-29 15:30:55	18.000000	11.900000	-4.096755	-12.370879	-15021.129286
25%	105.500000	2016-12-29 08:57:30.500000	27.000000	2555.560000	-1.303446	36.245872	384.073747
50%	205.500000	2017-08-18 22:42:25	34.000000	6004.240000	-1.225855	36.810147	898.105780
75%	304.250000	2018-03-30 23:11:54	44.000000	10789.942500	-0.497599	36.935694	1622.927284
max	400.000000	2018-12-19 11:42:23	105.000000	41657.810000	17.800041	42.315663	3975.348148
std	115.753357	NaN	13.410952	6877.444522	1.652212	3.175649	1221.968236

In [14]:

```
# Feature engineering: Create new features from existing data

# Convert loan_outcome to binary (1 for repaid, 0 for defaulted)
final_merged_df['loan_outcome_binary']=final_merged_df['loan_outcome'].apply(lambda x: 1 if x == 'repaid' else 0)

final_merged_df['Date'] = pd.to_datetime(final_merged_df['application_at'])

# Extract date components from application_at timestamp
final_merged_df['Day'] = final_merged_df['Date'].dt.day # Day of month
final_merged_df['Month'] = final_merged_df['Date'].dt.month # Month
final_merged_df['Year'] = final_merged_df['Date'].dt.year # Year
final_merged_df['Day_of_Week'] = final_merged_df['Date'].dt.dayofweek # Weekday (0=Monday)

# Display the first few rows with new features
final_merged_df.head()
```

Out[14]:

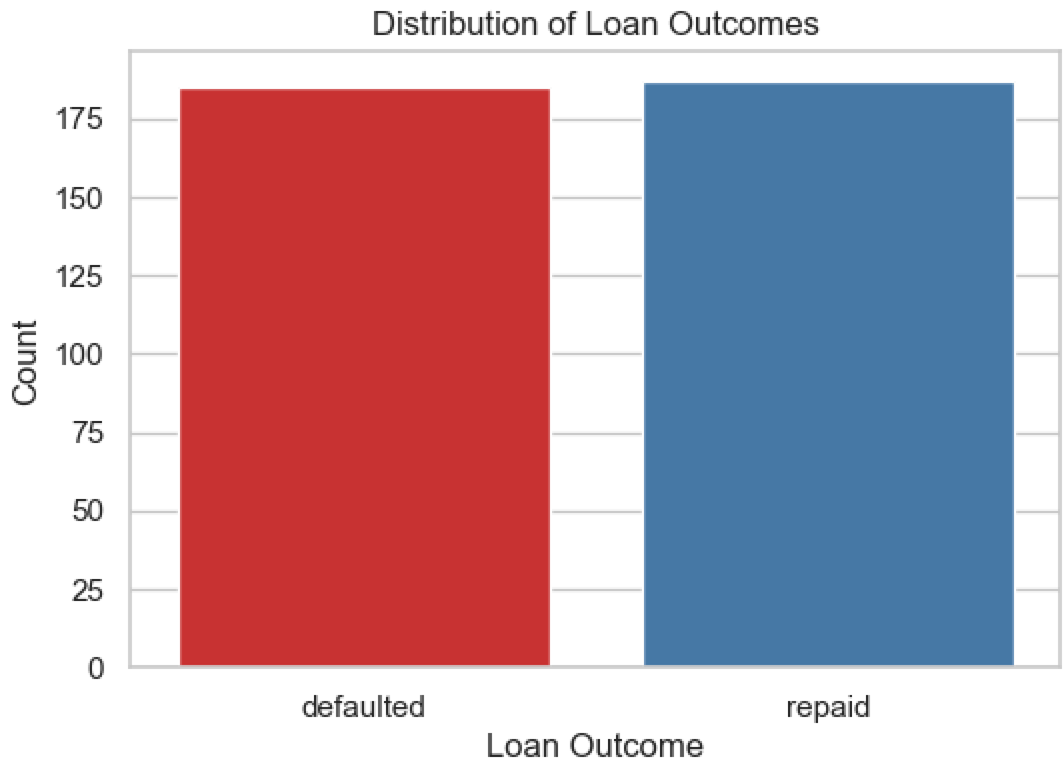
	user_id	application_at	loan_outcome	age	cash_incoming_30days	latitude	longitude	accuracy	loan_outcome_binary
0	1	2017-08-14 09:08:50.000000	defaulted	42	8988.12	-1.270427	36.782813	1105.084571	0
1	2	2016-05-17 10:10:12.447976	repaid	36	9968.12	-1.488055	37.118432	48.596000	1
2	3	2016-10-20 10:07:20.459081	defaulted	27	59.04	-0.889673	35.707550	6.500000	0
3	4	2017-01-13 13:03:34.000000	defaulted	38	2129.03	-0.306798	36.082255	2172.200000	0
4	5	2016-11-03 15:41:39.124610	repaid	33	2102.53	17.800041	-12.370879	43.461111	1

In [15]:

```
# Import visualization libraries
import matplotlib.pyplot as plt
import seaborn as sn
```

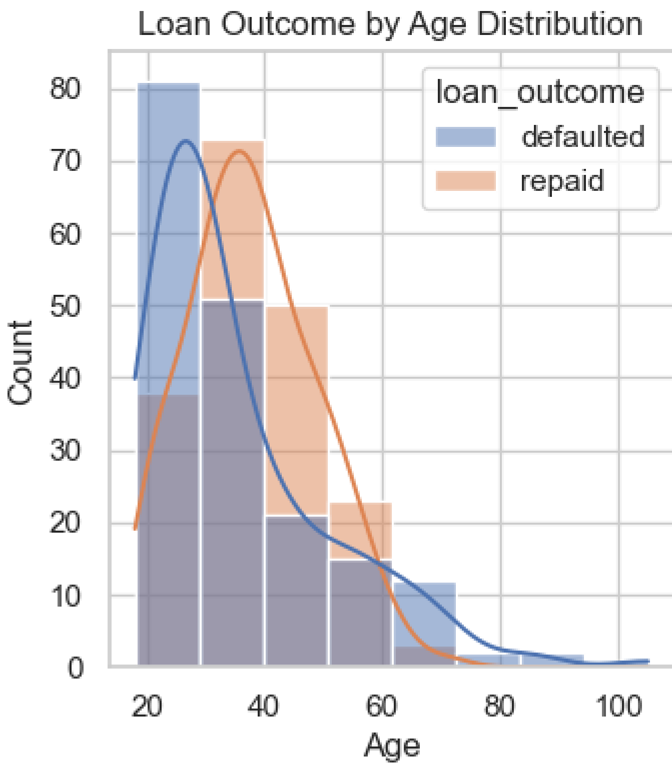
In [16]:

```
# Set visualization style
sn.set(style="whitegrid")

# Plot 1: Distribution of Loan Outcomes
plt.figure(figsize=(6, 4))
sn.countplot(data=final_merged_df, x='loan_outcome', palette='Set1')
plt.title('Distribution of Loan Outcomes')
plt.xlabel('Loan Outcome')
plt.ylabel('Count')
plt.show()
```

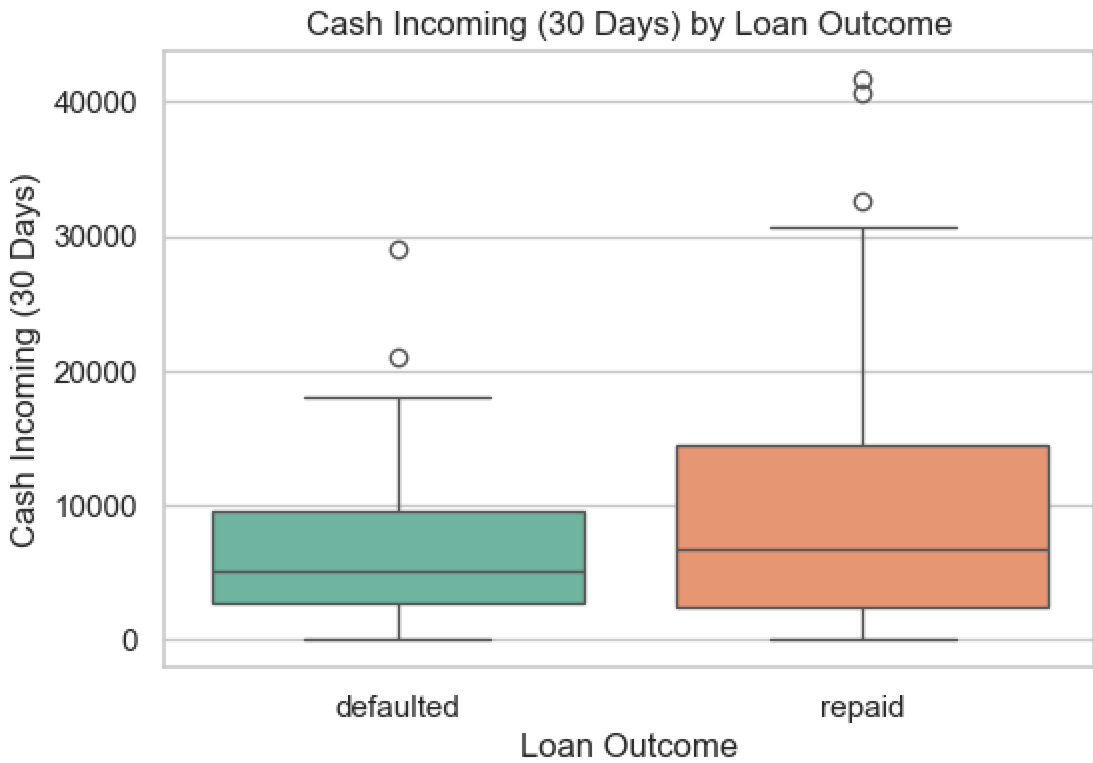


In [17]:
```
# Plot 2: Distribution of Age by Loan Outcomes
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 2)
sn.histplot(data=final_merged_df, x='age', hue='loan_outcome', kde=True, bins=8 )
plt.title('Loan Outcome by Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
```

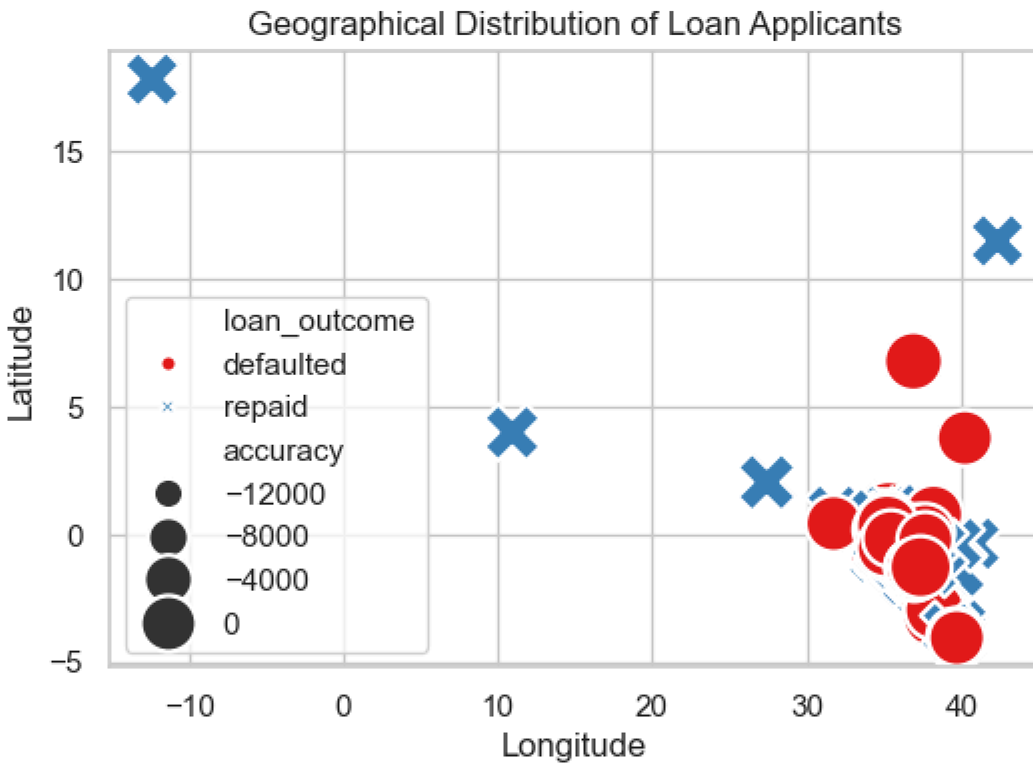
Out[17]:
```
Text(0, 0.5, 'Count')
```



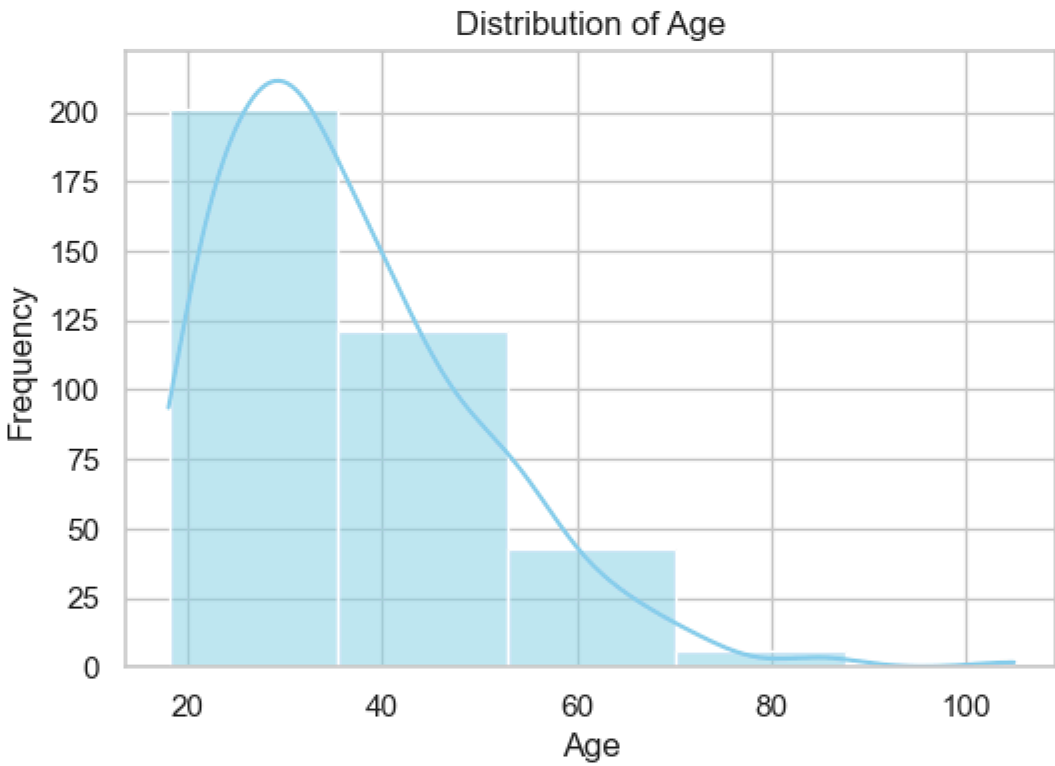
In [18]:
```
# Plot 3: Cash Incoming by Loan Outcome
plt.figure(figsize=(6, 4))
sn.boxplot(data=final_merged_df, x='loan_outcome', y='cash_incoming_30days', palette='Set2')
plt.title('Cash Incoming (30 Days) by Loan Outcome')
plt.xlabel('Loan Outcome')
plt.ylabel('Cash Incoming (30 Days)')
plt.show()
```



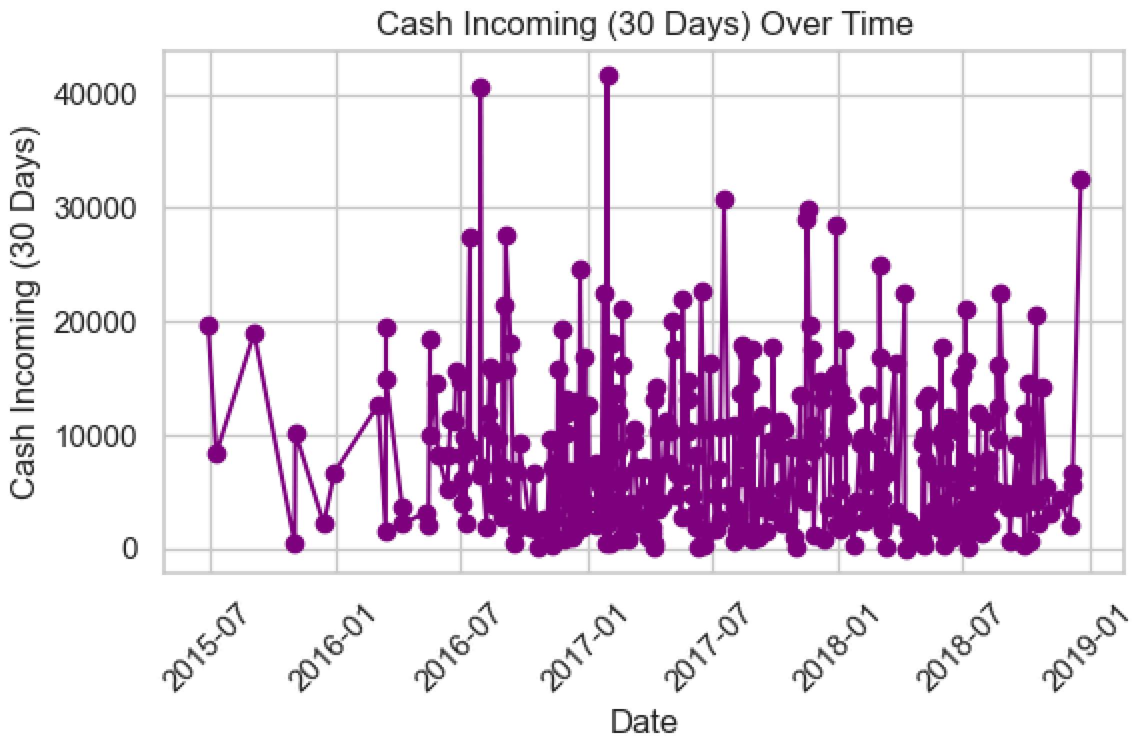
In [19]:
```
# Plot 4: Geographical Distribution of Applicants
plt.figure(figsize=(6, 4))
sn.scatterplot(data=final_merged_df,
               x='longitude',
               y='latitude',
               hue='loan_outcome',
               style='loan_outcome',
               size='accuracy',
               sizes=(50, 500), palette='Set1')
plt.title('Geographical Distribution of Loan Applicants')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



In [20]:
```
# Plot 5: Age Distribution
plt.figure(figsize=(6, 4))
sn.histplot(data=final_merged_df, x='age', bins=5, kde=True, color='skyblue')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



In [21]:
```
# Plot 6: Cash Incoming Over Time
plt.figure(figsize=(6, 4))
df_sorted = final_merged_df.sort_values('Date')  # Sort by date for a proper line plot
plt.plot(df_sorted['Date'], df_sorted['cash_incoming_30days'], marker='o', color='purple')
plt.title('Cash Incoming (30 Days) Over Time')
plt.xlabel('Date')
plt.ylabel('Cash Incoming (30 Days)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [22]:
```
# Import machine learning libraries
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
```

In [23]:
```
# Handle geographical outlier
final_merged_df['latitude'] = final_merged_df['latitude'].clip(lower=-1.5, upper=0)
final_merged_df['longitude'] = final_merged_df['longitude'].clip(lower=35, upper=38)

# # Define features (X) and target (y) for modeling
features = ['age', 'cash_incoming_30days', 'latitude', 'longitude', 'accuracy', 'Day', 'Month', 'Year', 'Day_of_Week
X = final_merged_df[features] # feature variable
y = final_merged_df['loan_outcome_binary'] #Target variable
```

```
# Print shapes and check for null values
print(X.shape)
print(y.shape)
print(X.isnull().sum())
print(y.isnull().sum())
```

```
(372, 9)
(372,)
age                    0
cash_incoming_30days   0
latitude               0
longitude              0
accuracy               0
Day                    0
Month                  0
Year                   0
Day_of_Week            0
dtype: int64
0
```

In [24]:
```
# Feature selection using SelectKBest with ANOVA F-value
selector = SelectKBest(f_classif, k=5)
X_selected = selector.fit_transform(X,y)

# Get names of selected features
selected_indices = selector.get_support(indices=True)
selected_features = X.columns[selected_indices]

print(f"Selected feature: {selected_features}")
```

```
Selected feature: Index(['age', 'cash_incoming_30days', 'accuracy', 'Month', 'Year'], dtype='object')
```

In [25]:
```
# Split data into training and test sets (75% train, 25% test)
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.25, random_state=42)

#Standardizing features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Print shapes of train/test sets
print(X_train.shape)
```

```
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(279, 5)
(93, 5)
(279,)
(93,)
```

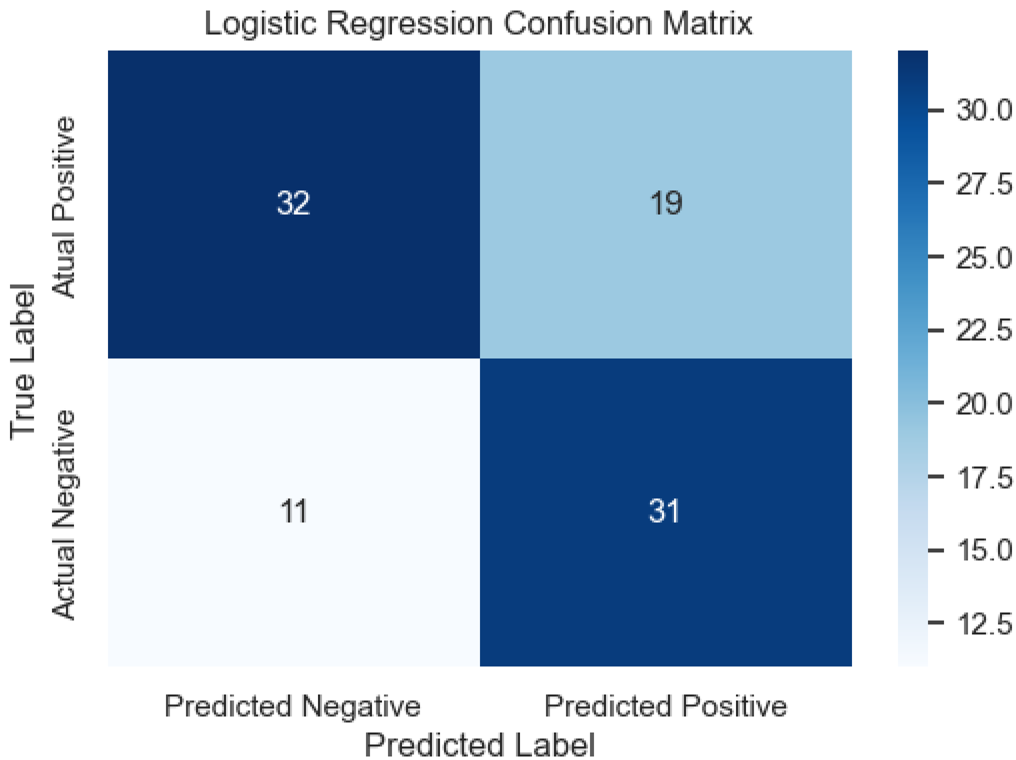
In [26]:
```
# Model 1: Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train,y_train) # Train model
lr_pred = lr_model.predict(X_test) # Train model

# Evaluate Logistic Regression model
lr_accuracy = accuracy_score(y_test,lr_pred)
lr_class_report = classification_report(y_test,lr_pred)
```

In [27]:
```
print(f"\nLogistic Regression Accuracy: {lr_accuracy:.2f}")
print("Classification Report:\n",lr_class_report)
```

```
Logistic Regression Accuracy: 0.68
Classification Report:
               precision    recall  f1-score   support

           0       0.74      0.63      0.68        51
           1       0.62      0.74      0.67        42

    accuracy                           0.68        93
   macro avg       0.68      0.68      0.68        93
weighted avg       0.69      0.68      0.68        93
```

In [28]:
```
# Plot confusion matrix for Logistic Regression
lr_model_cm = confusion_matrix(y_test,lr_pred)

# Visualize the confusion matrix
plt.figure(figsize=(6,4))
sn.heatmap(lr_model_cm, annot = True,
            fmt='d',cmap='Blues',
            xticklabels=(['Predicted Negative','Predicted Positive']),
            yticklabels=(['Atual Positive','Actual Negative']))
```

```
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [29]:

```
# Model 2: Random Forest Classification
rfc_model = RandomForestClassifier(n_estimators=100,random_state=42)
rfc_model.fit(X_train,y_train) # Train model
rfc_pred = rfc_model.predict(X_test) # Make predictions

# Evaluate Random Forest model
rfc_accuracy = accuracy_score(y_test,rfc_pred)
rfc_class_report = classification_report(y_test,rfc_pred)
```

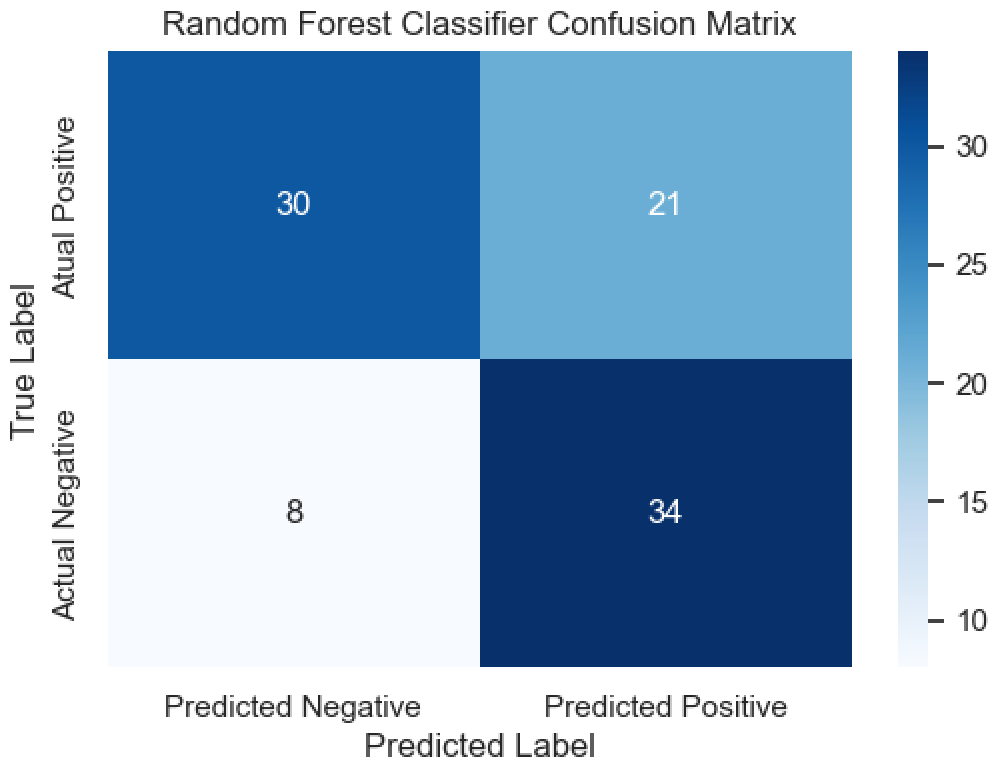
In [30]:

```
print(f"\nRandom Forest Classifier Accuracy: {rfc_accuracy:.2f}")
print("Classification Report:\n",rfc_class_report)
```

```
Random Forest Classifier Accuracy: 0.69
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.59      0.67        51
           1       0.62      0.81      0.70        42

    accuracy                           0.69        93
   macro avg       0.70      0.70      0.69        93
weighted avg       0.71      0.69      0.69        93
```

In [31]:

```
# Plot confusion matrix for Random Forest
rfc_model_cm = confusion_matrix(y_test,rfc_pred)

# Visualize the confusion matrix
plt.figure(figsize=(6,4))
sn.heatmap(rfc_model_cm, annot = True,
            fmt='d',cmap='Blues',
            xticklabels=(['Predicted Negative','Predicted Positive']),
            yticklabels=(['Atual Positive','Actual Negative']))
plt.title('Random Forest Classifier Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

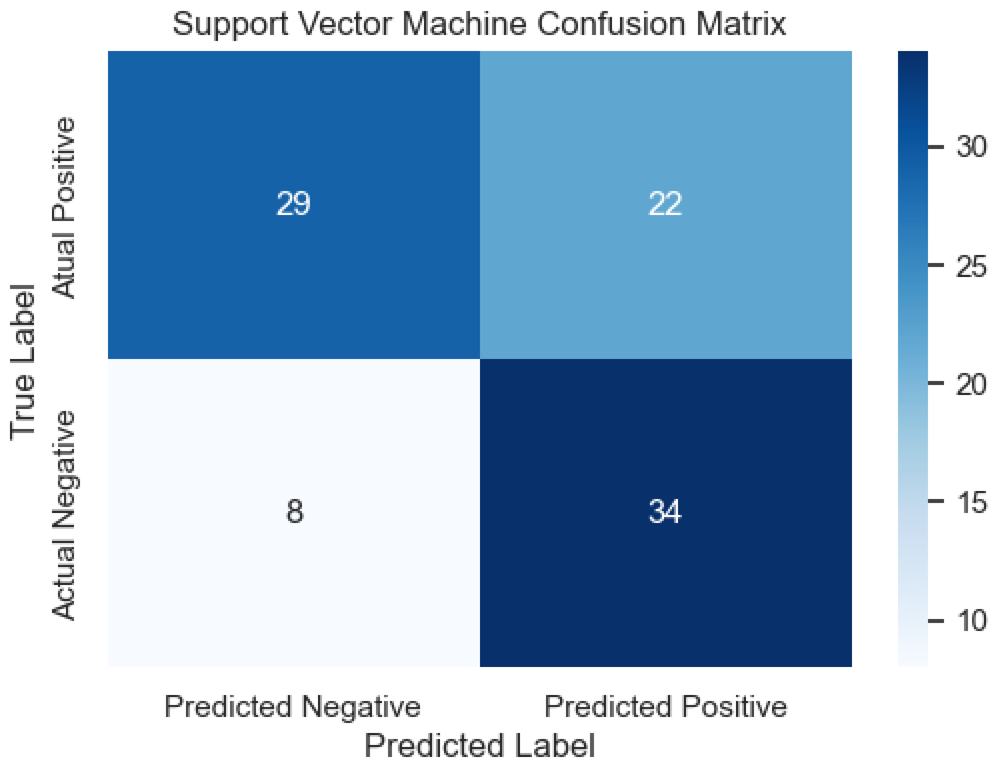


In [32]:
```
# Model 3: Support Vector Mechine
svc_model = SVC(kernel='rbf',random_state=42)
svc_model.fit(X_train,y_train) # Train Model
svc_pred = svc_model.predict(X_test) # Make predictions

# Evaluate SVC model
svc_accuracy = accuracy_score(y_test,svc_pred)
svc_class_report = classification_report(y_test,svc_pred)
```

In [33]:
```
print(f"\nSupport Vector Mechine: {svc_accuracy:.2f}")
print("Classification Report:\n",svc_class_report)
```

```
Support Vector Mechine: 0.68
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.57      0.66        51
           1       0.61      0.81      0.69        42

    accuracy                           0.68        93
   macro avg       0.70      0.69      0.68        93
weighted avg       0.70      0.68      0.67        93
```

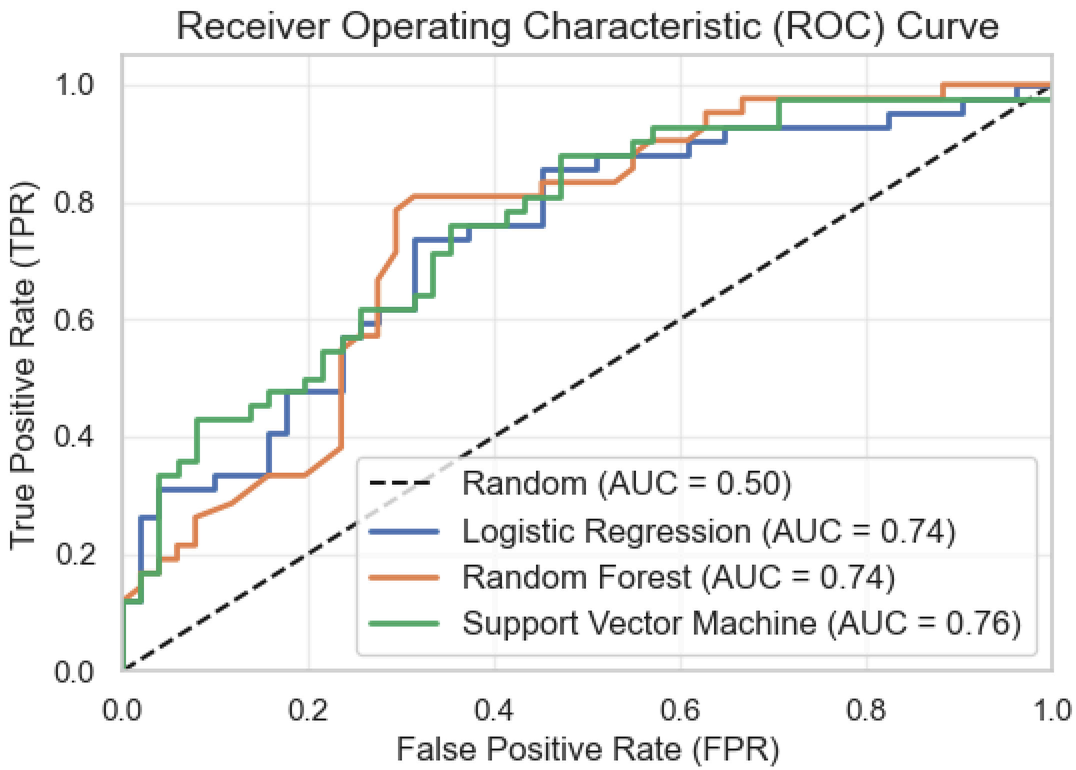
In [34]:
```
# Plot confusion matrix for SVC
svc_model_cm = confusion_matrix(y_test,svc_pred)

# Visualize the confusion matrix
plt.figure(figsize=(6,4))
sn.heatmap(svc_model_cm, annot = True,
            fmt='d',cmap='Blues',
            xticklabels=(['Predicted Negative','Predicted Positive']),
            yticklabels=(['Atual Positive','Actual Negative']))
plt.title('Support Vector Machine Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [35]:
```
# Function to plot ROC curves for all models
def plot_roc_curves(models, model_names, X_test, y_test):
    plt.figure(figsize=(6, 4))

    # Plot diagonal line (random classifier)
    plt.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.50)')

    # Initialize dictionary to store AUC scores
    auc_scores = {}

    for model, name in zip(models, model_names):
        # Get predicted probabilities for the positive class
        if hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X_test)[:, 1]
        else:  # For SVM which might not have predict_proba by default
            y_probs = model.decision_function(X_test)
            y_probs = (y_probs - y_probs.min()) / (y_probs.max() - y_probs.min())
```

```
        # Calculate ROC curve and AUC
        fpr, tpr, thresholds = roc_curve(y_test, y_probs)
        roc_auc = auc(fpr, tpr)
        auc_scores[name] = roc_auc

        # Plot ROC curve
        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})', linewidth=2)

    # Customize the plot
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (FPR)', fontsize=12)
    plt.ylabel('True Positive Rate (TPR)', fontsize=12)
    plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=14)
    plt.legend(loc="lower right", fontsize=12)
    plt.grid(True, alpha=0.3)

    # Display AUC scores
    print("\nAUC Scores:")
    for name, score in auc_scores.items():
        print(f"{name}: {score:.3f}")

    plt.show()

# Create list of your trained models and their names
models = [lr_model, rfc_model, svc_model]
model_names = ['Logistic Regression', 'Random Forest', 'Support Vector Machine']

# Plot ROC curves for all models
plot_roc_curves(models, model_names, X_test, y_test)
```

```
AUC Scores:
Logistic Regression: 0.739
Random Forest: 0.745
Support Vector Machine: 0.760
```

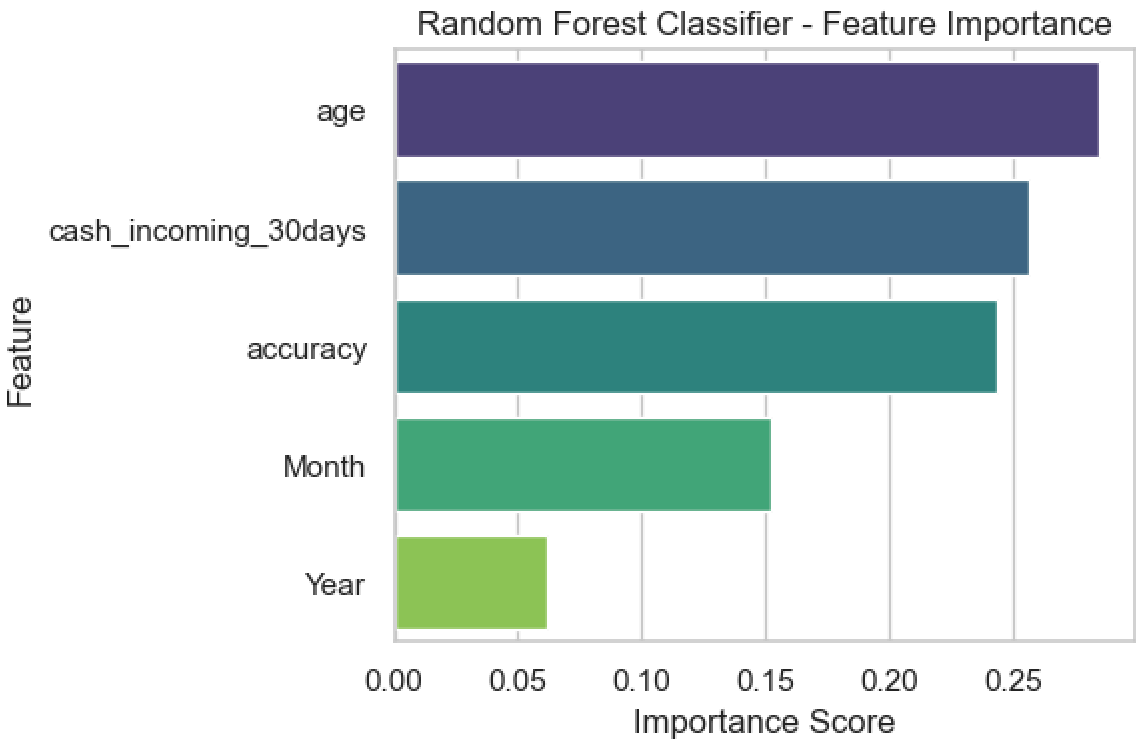


In [36]:
```
# Feature Importance Analysis

# Get feature importance scores from the trained Random Forest model
feature_importance = rfc_model.feature_importances_

# Create a DataFrame to store feature names and their importance scores
importance_df = pd.DataFrame({
    'Feature': selected_features,  # Using the selected features from SelectKBest
    'Importance': feature_importance
})

# Sort features by importance (descending order)
importance_df = importance_df.sort_values('Importance', ascending=False)

# Print the feature importance table
print("\nRandom Forest Feature Importance:")
print(importance_df.to_string(index=False))
```

```
Random Forest Feature Importance:
               Feature  Importance
                   age    0.284997
   cash_incoming_30days    0.256758
              accuracy    0.243674
                 Month    0.152417
                  Year    0.062153
```

In [37]:

```
# Visualize feature importance
plt.figure(figsize=(6, 4))
sn.barplot(
    data=importance_df,
    x='Importance',
    y='Feature',
    palette='viridis'
)
plt.title('Random Forest Classifier - Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```



In [38]:
```
# Import joblib and save model
import joblib

# Save the model and scaler
joblib.dump(rfc_model, 'rfc_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(selector, 'feature_selector.pkl')
```

Out[38]: ['feature_selector.pkl']

PREPARED AND TRAINED BY VICTOR ITINAH INIOBONG