Fraud Detection Project ¶

In [364]:
```
#Let's import some important libraries and then understand our dataset
```

In [365]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [366]:
```
data = pd.read_csv(r"C:\Users\USER\Downloads\archive (32)\creditcard.csv")
```

Data Pre-processing phase

The dataset contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues,we cannot provide the original features and more background information about the data.Features V1, V2, … V28 are the principal components obtained with PCA,the only features which have not been transformed with PCA are 'Time' and 'Amount', (Reference:Kaggle)

In [367]:
```
data.shape
```

Out[367]:
```
(284807, 31)
```

In [368]:
```
# So we have 284807 rows and 31 columns. Which columns are in our dataset?
```

In [369]:
```
data.columns
```

Out[369]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [370]:
```
data.head(5)
```

Out[370]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141

5 rows × 31 columns

In [371]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [372]:
```
data.duplicated().sum()
```

Out[372]:
```
1081
```

In [373]:
```
data.drop_duplicates(inplace=True)
```

In [374]:
```
data.shape
```

Out[374]:
```
(283726, 31)
```

In [375]:
```
data.describe()
```

Out[375]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000
mean	94811.077600	0.005917	-0.004135	0.001613	-0.002966	0.001828	-0.001139	0.001801	-0.000854
std	47481.047891	1.948026	1.646703	1.508682	1.414184	1.377008	1.331931	1.227664	1.179054
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	-0.689830	-0.769031	-0.552509	-0.208828
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	-0.053468	-0.275168	0.040859	0.021898
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	0.612218	0.396792	0.570474	0.325704
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208

8 rows × 31 columns

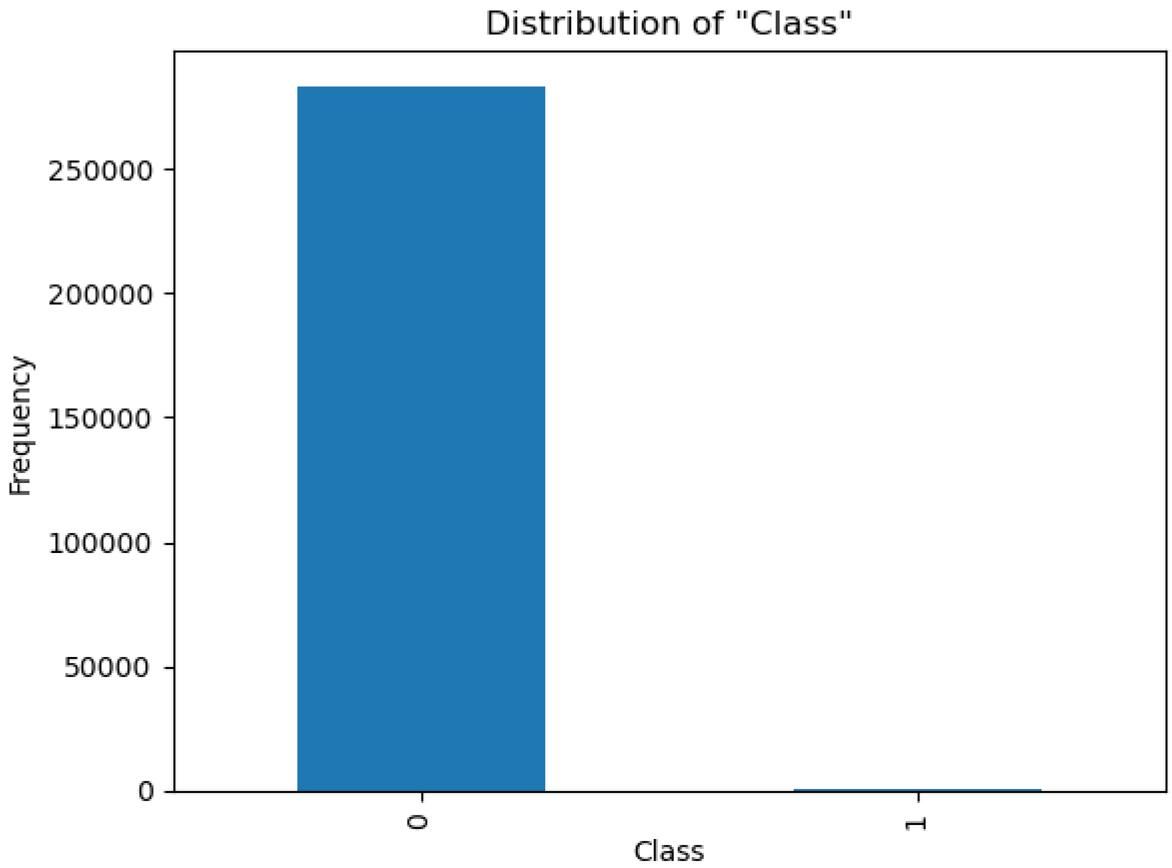
In [376]:
```
# Let's examine how "Class" column is distributed
print(data['Class'].value_counts())
print()
data['Class'].value_counts(normalize=True) * 100
```

```
Class
0    283253
1       473
Name: count, dtype: int64
```

Out[376]:
```
Class
0    99.83329
1     0.16671
Name: proportion, dtype: float64
```

We understand that the dataset is highly unbalanced and the positive class (frauds) account for 0.166% of all transactions.

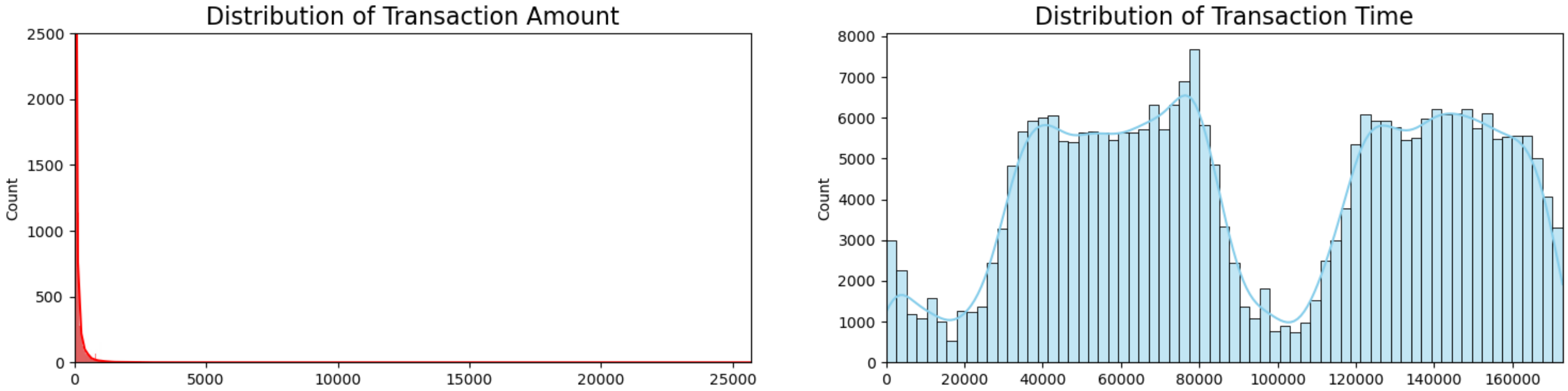
In [377]:
```
data['Class'].value_counts().plot(kind='bar')
plt.title('Distribution of "Class"')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```



In [378]:
```
#missing values
missing_values = data.isna().sum()
missing_values
```

Out[378]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [379]:
```
# Subplots of Distributions of Transactions of Time and Amount
fig, ax = plt.subplots(1, 2, figsize=(18, 4))

amount_values = data['Amount'].values
time_values = data['Time'].values

sns.histplot(amount_values, ax=ax[0], color='r', kde=True)  # Use histplot
ax[0].set_title('Distribution of Transaction Amount', fontsize=16)
ax[0].set_xlim([min(amount_values), max(amount_values)])
ax[0].set_ylim(0, 2500)

sns.histplot(time_values, ax=ax[1], color='skyblue', kde=True)  # Use histplot
ax[1].set_title('Distribution of Transaction Time', fontsize=16)
ax[1].set_xlim([min(time_values), max(time_values)])

plt.show()
```



In [380]:

```
# We want to standardize the data of columns: 'Time' and 'Amount'
from sklearn.preprocessing import StandardScaler, RobustScaler

# RobustScaler is less prone to outliers.

std_scaler = StandardScaler()
rob_scaler = RobustScaler()

data['scaled_amount'] = rob_scaler.fit_transform(data['Amount'].values.reshape(-1,1))
data['scaled_time'] = rob_scaler.fit_transform(data['Time'].values.reshape(-1,1))

data.drop(['Time','Amount'], axis=1, inplace=True)
```

In [381]:

```
# Creation of new columns of Time and Amount
scaled_amount = data['scaled_amount']
scaled_time = data['scaled_time']

data.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
data.insert(0, 'scaled_amount', scaled_amount)
data.insert(1, 'scaled_time', scaled_time)

data.head()
```

Out[381]:

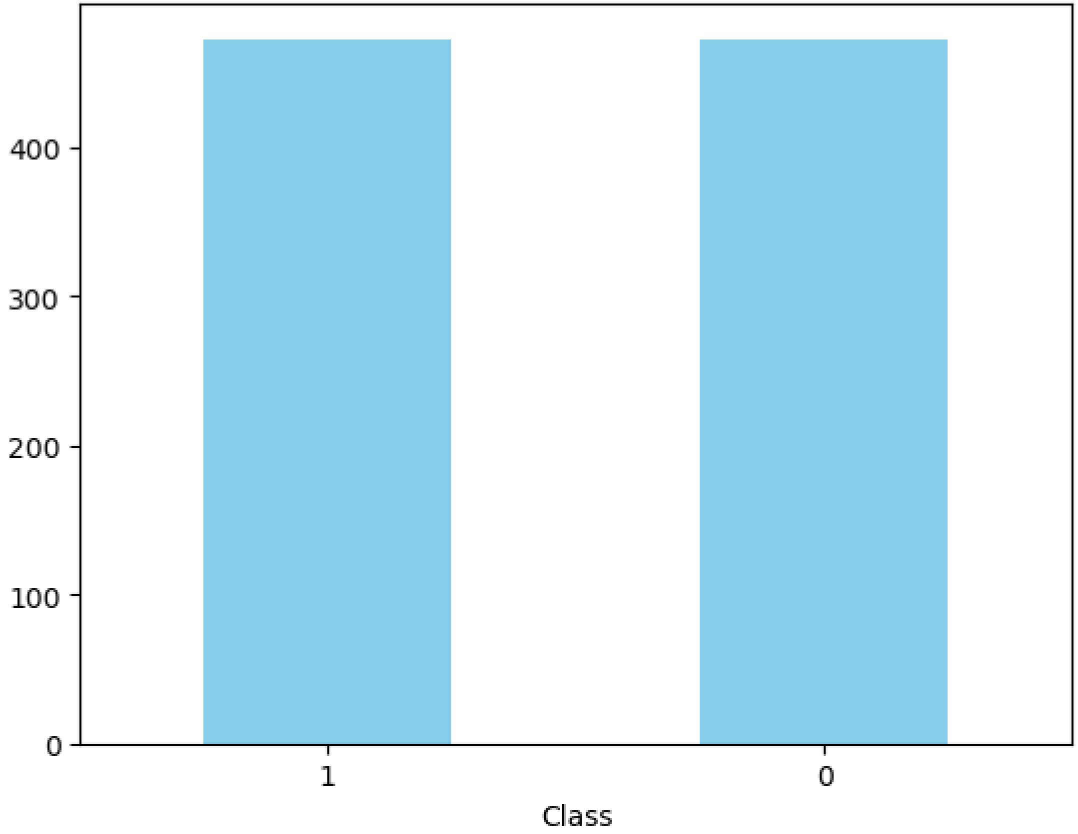
	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	
0	1.774718	-0.995290	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...	0.251412	-0.018307	0.2
1	-0.268530	-0.995290	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...	-0.069083	-0.225775	-0.6
2	4.959811	-0.995279	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	...	0.524980	0.247998	0.7
3	1.411487	-0.995279	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	...	-0.208038	-0.108300	0.0
4	0.667362	-0.995267	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	...	0.408542	-0.009431	0.7

5 rows × 31 columns

In [382]:
```
# Now we will do undersampling to create a sample that non frauds equals frauds

fraud_data = data[data['Class'] == 1]
non_fraud_data = data[data['Class'] == 0]

non_fraud_sample = non_fraud_data.sample(n=len(fraud_data), random_state=42)

balanced_data = pd.concat([fraud_data, non_fraud_sample])

print(balanced_data['Class'].value_counts())
```

```
Class
1    473
0    473
Name: count, dtype: int64
```

In [383]:
```
# Bar plot for distribution of classes
class_counts = balanced_data['Class'].value_counts()
class_counts.plot(kind='bar', color='skyblue')
plt.xticks(rotation=0)
plt.show()
```



In [384]:
```
data.head()
```

Out[384]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	
0	1.774718	-0.995290	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...	0.251412	-0.018307	0.27
1	-0.268530	-0.995290	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...	-0.069083	-0.225775	-0.63
2	4.959811	-0.995279	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	...	0.524980	0.247998	0.77
3	1.411487	-0.995279	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	...	-0.208038	-0.108300	0.00
4	0.667362	-0.995267	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	...	0.408542	-0.009431	0.79

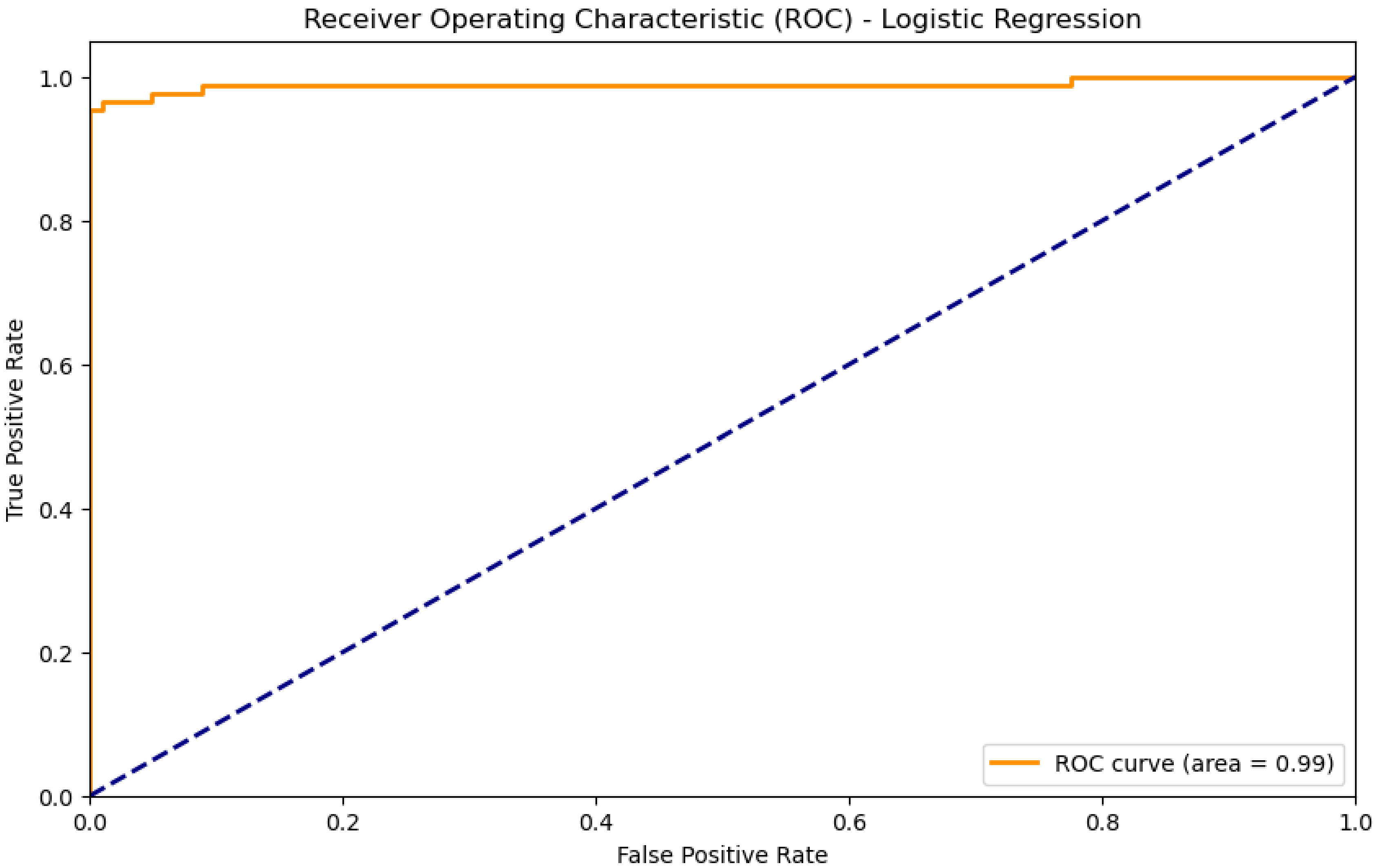
5 rows × 31 columns

Logistic Regression

In [385]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

X = balanced_data.drop('Class', axis=1)
y = balanced_data['Class']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

# Calculate and print metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Roc Curve for Logistic Regression
from sklearn.metrics import roc_curve

fpr_logistic, tpr_logistic, _ = roc_curve(y_test, model.predict_proba(X_test)[:,1])
roc_auc_logistic = roc_auc_score(y_test, model.predict_proba(X_test)[:,1])

plt.figure(figsize=(10, 6))
plt.plot(fpr_logistic, tpr_logistic, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_logistic)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Logistic Regression')
plt.legend(loc="lower right")
```

```
plt.show()
```

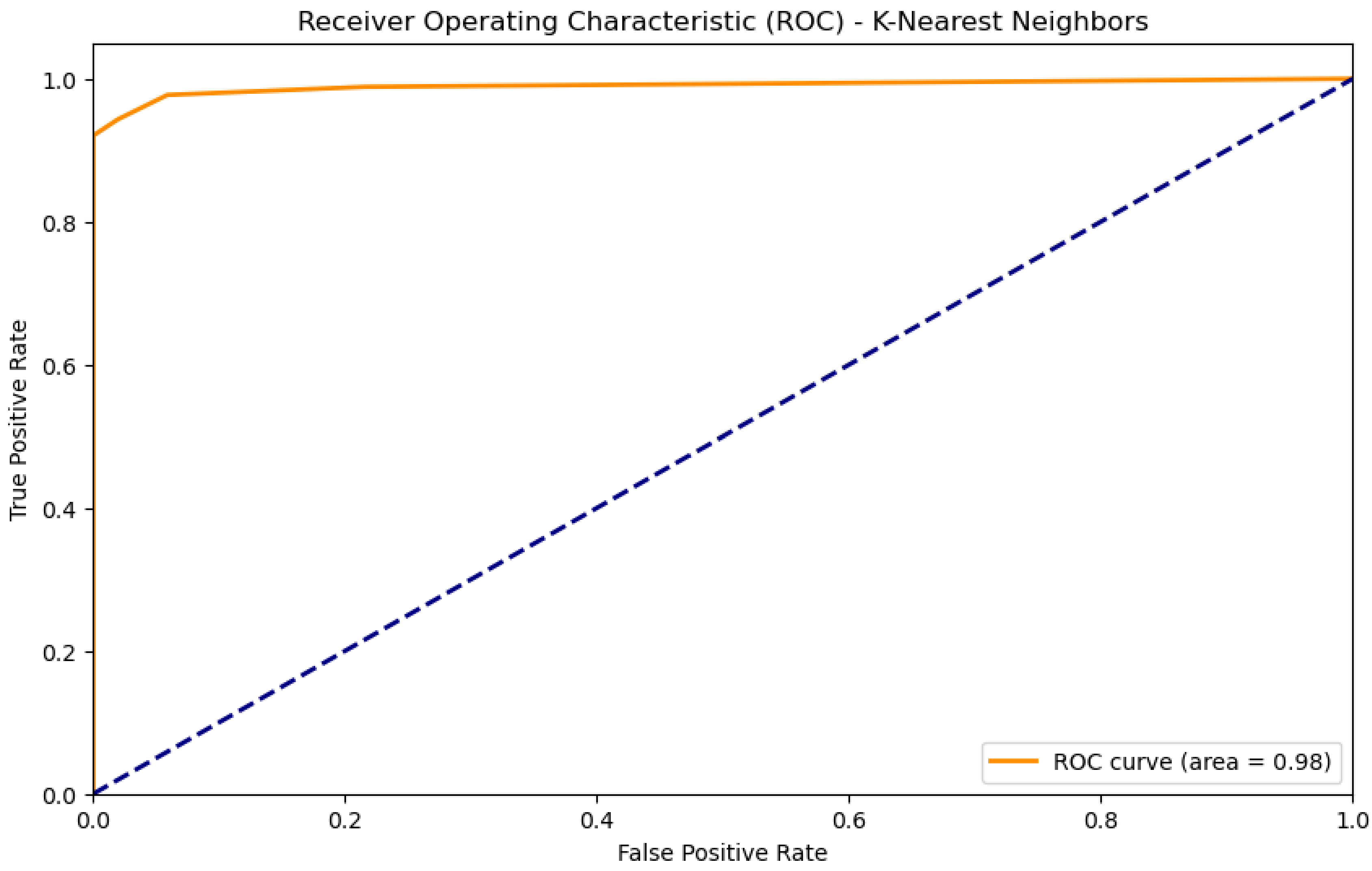
```
Accuracy: 0.9789473684210527
Precision: 0.9883720930232558
Recall: 0.9659090909090909
F1 Score: 0.9770114942528736
```



K-Nearest Neighbors (KNN):

In [386]:
```
from sklearn.neighbors import KNeighborsClassifier

# Create,fit and predict KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)

# Calculate and print metrics
knn_accuracy = accuracy_score(y_test, y_pred_knn)
knn_precision = precision_score(y_test, y_pred_knn)
knn_recall = recall_score(y_test, y_pred_knn)
knn_f1 = f1_score(y_test, y_pred_knn)

# Print KNN results
print("K-Nearest Neighbors Results:")
print("Accuracy:", knn_accuracy)
print("Precision:", knn_precision)
print("Recall:", knn_recall)
print("F1-Score:", knn_f1)

# Create a ROC curve
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_model.predict_proba(X_test)[:, 1])
# Plot ROC curve for KNN
plt.figure(figsize=(10, 6))
plt.plot(fpr_knn, tpr_knn, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - K-Nearest Neighbors')
plt.legend(loc="lower right")
plt.show()
```

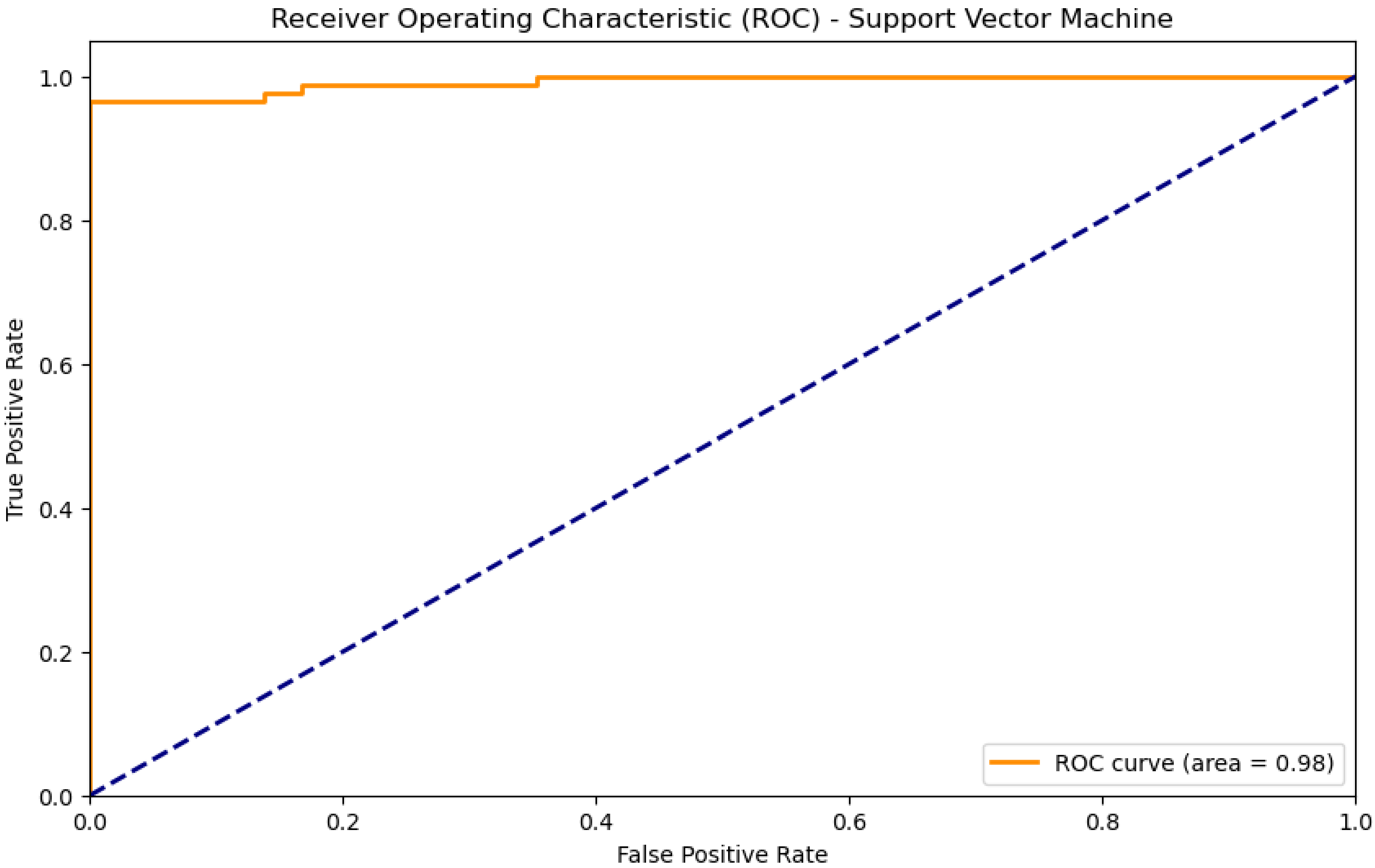
```
K-Nearest Neighbors Results:
Accuracy: 0.9631578947368421
Precision: 0.9764705882352941
Recall: 0.9431818181818182
F1-Score: 0.9595375722543352
```



Support Vector Machine(SVM)

In [387]:
```
from sklearn.svm import SVC

# Create,fit and predict SVM model
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)

# Calculate and print metrics
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm)
svm_recall = recall_score(y_test, y_pred_svm)
svm_f1 = f1_score(y_test, y_pred_svm)

# Print SVM results
print("Support Vector Machine Results:")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
print("Recall:", svm_recall)
print("F1-Score:", svm_f1)


# Create a ROC curve
fpr_svm, tpr_svm, _ = roc_curve(y_test, svm_model.predict_proba(X_test)[:, 1])
# Plot ROC curve for SVM
plt.figure(figsize=(10, 6))
plt.plot(fpr_svm, tpr_svm, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Support Vector Machine')
plt.legend(loc="lower right")
plt.show()
```

```
Support Vector Machine Results:
Accuracy: 0.968421052631579
Precision: 1.0
Recall: 0.9318181818181818
F1-Score: 0.9647058823529412
```

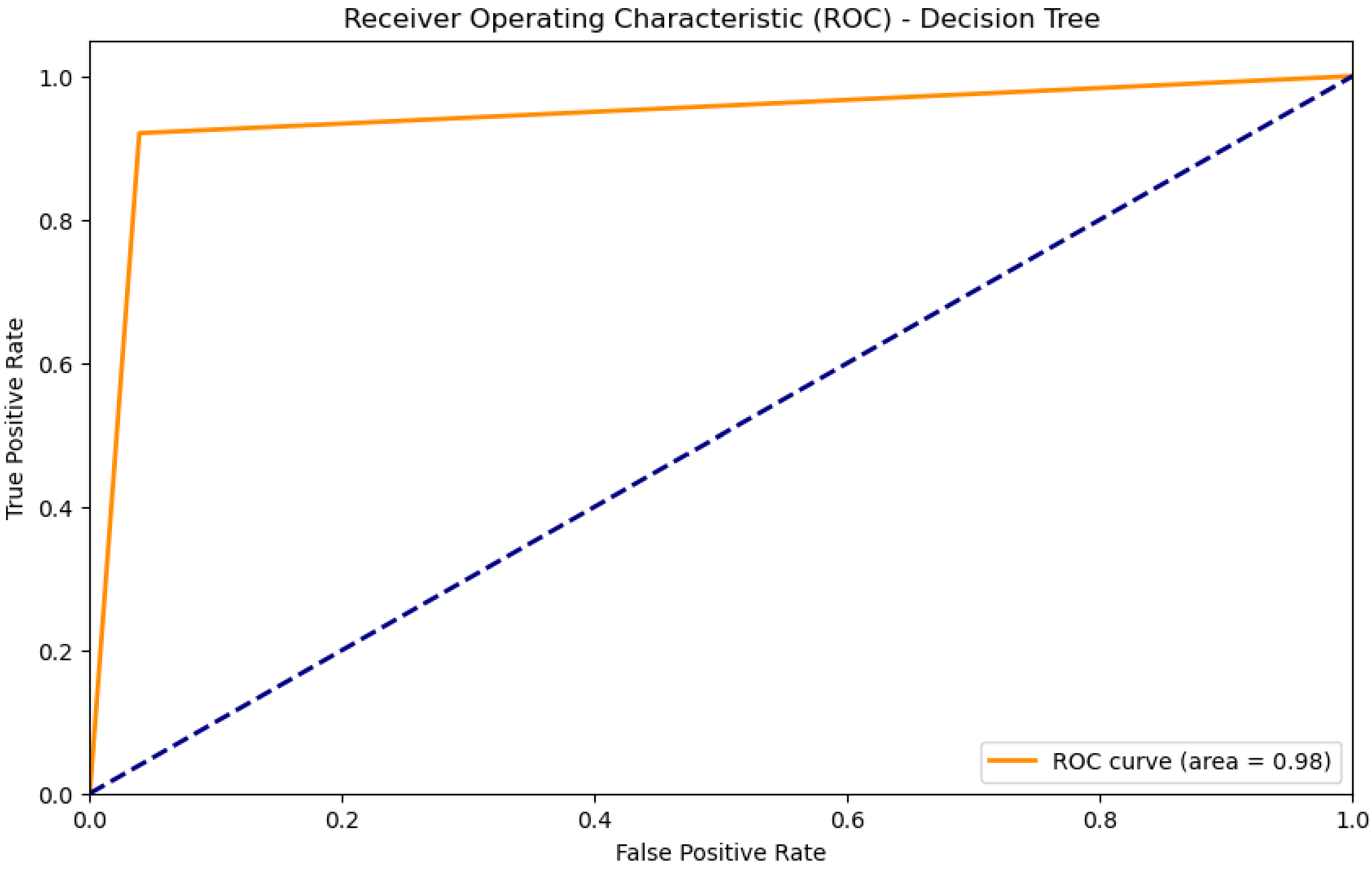


Decision Tree

In [388]:

```
from sklearn.tree import DecisionTreeClassifier

# Create, fit and predict Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

# Calculate and print metrics
dt_accuracy = accuracy_score(y_test, y_pred_dt)
dt_precision = precision_score(y_test, y_pred_dt)
dt_recall = recall_score(y_test, y_pred_dt)
dt_f1 = f1_score(y_test, y_pred_dt)

# Print Decision Tree results
print("Decision Tree Results:")
print("Accuracy:", dt_accuracy)
print("Precision:", dt_precision)
print("Recall:", dt_recall)
print("F1-Score:", dt_f1)

# Create a ROC curve
fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_model.predict_proba(X_test)[:, 1])

# Plot ROC curve for Decision Tree
plt.figure(figsize=(10, 6))
plt.plot(fpr_dt, tpr_dt, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Decision Tree')
plt.legend(loc="lower right")
plt.show()
```

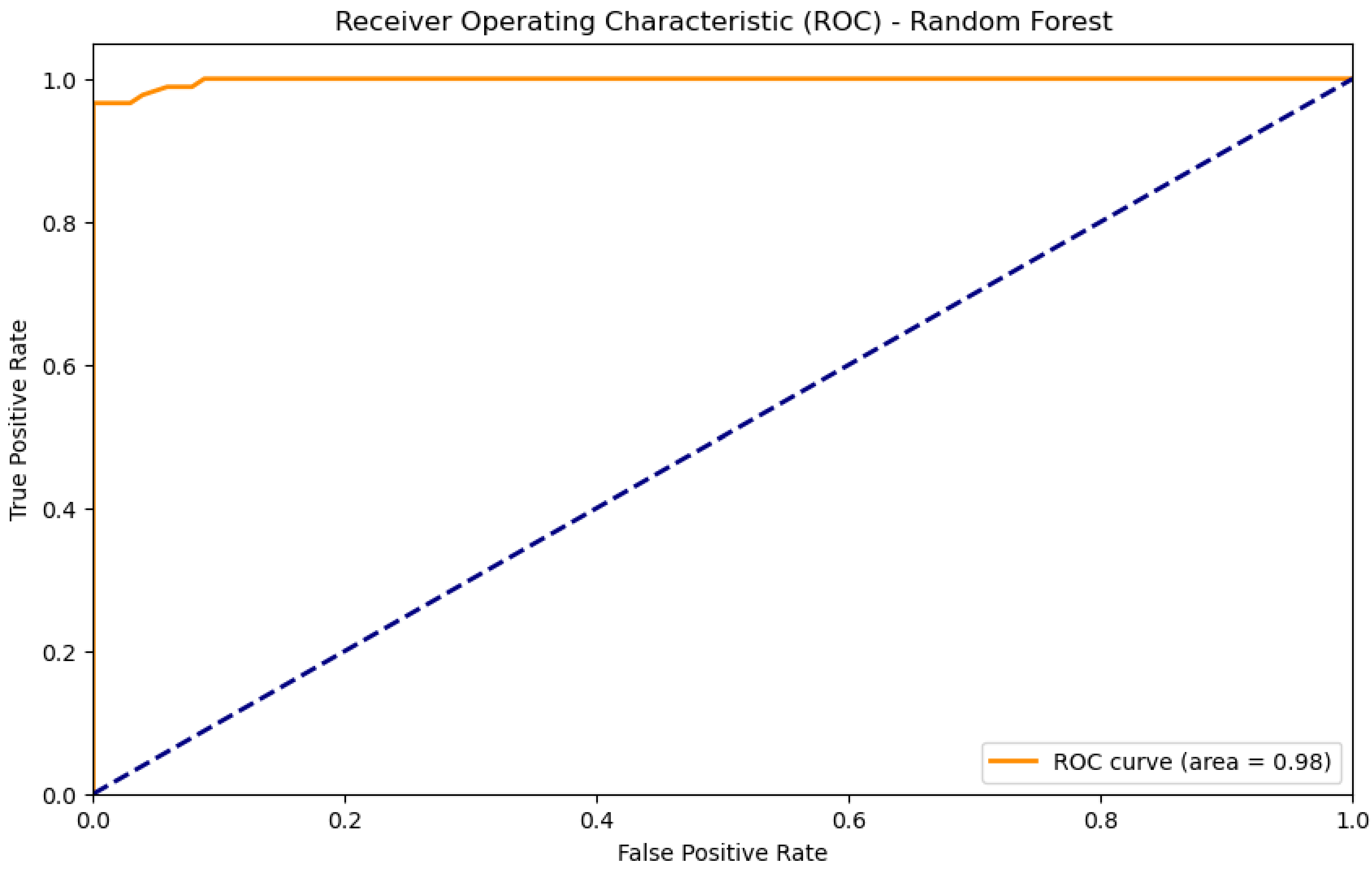
```
Decision Tree Results:
Accuracy: 0.9421052631578948
Precision: 0.9529411764705882
Recall: 0.9204545454545454
F1-Score: 0.9364161849710982
```



Random Forest

In [389]:
```
# Create,train and predict the Random Forest classifier
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Calculate metrics
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf)
rf_recall = recall_score(y_test, y_pred_rf)
rf_f1 = f1_score(y_test, y_pred_rf)

# Print metrics
print("Random Forest Results: ")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1 Score:", rf_f1)

# Create a ROC curve
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_model.predict_proba(X_test)[:, 1])

# Plot ROC curve for Decision Tree
plt.figure(figsize=(10, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Random Forest')
plt.legend(loc="lower right")
plt.show()
```

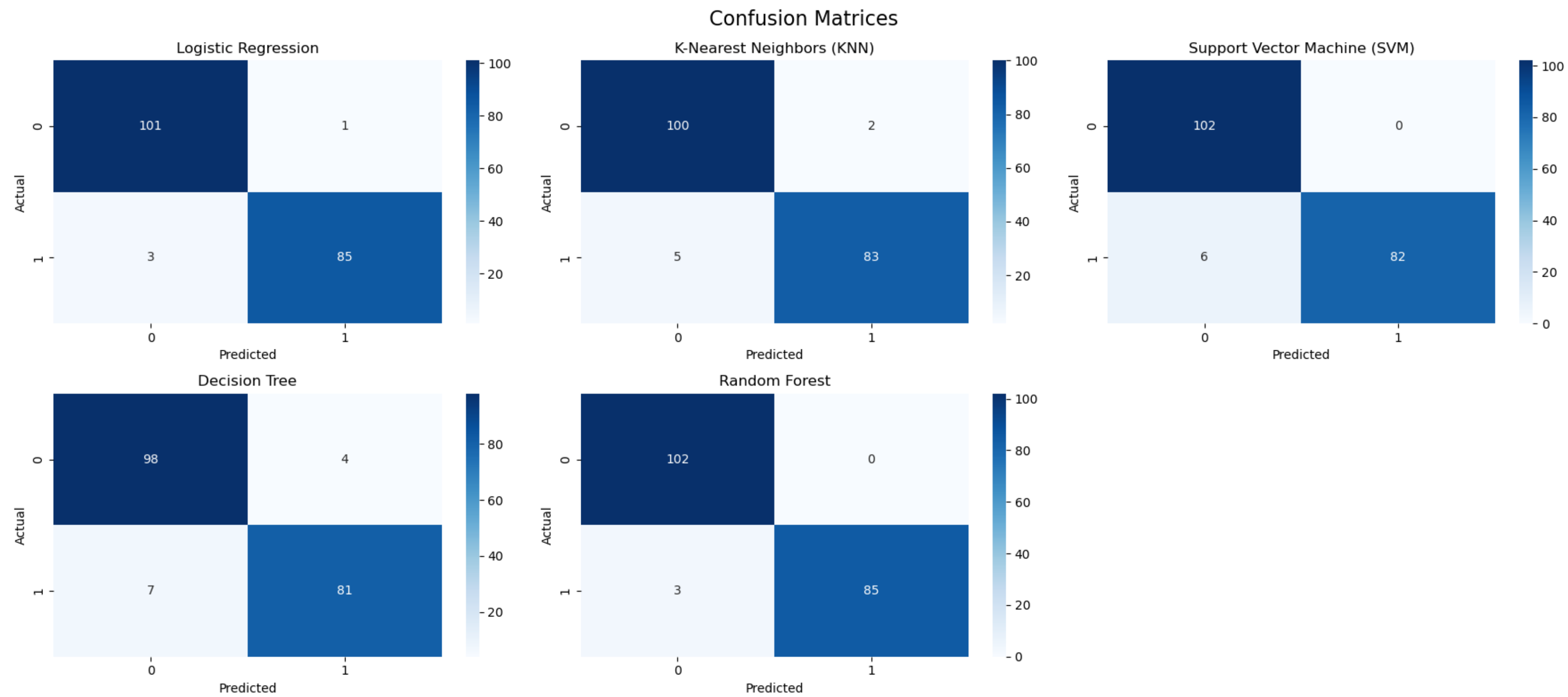
```
Random Forest Results:
Accuracy: 0.9842105263157894
Precision: 1.0
Recall: 0.9659090909090909
F1 Score: 0.9826589595375723
```



Confusion Matrices

In [390]:
```
# Calculate confusion matrices for all methods
logistic_cm = confusion_matrix(y_test, y_pred)
knn_cm = confusion_matrix(y_test, y_pred_knn)
svm_cm = confusion_matrix(y_test, y_pred_svm)
dt_cm = confusion_matrix(y_test, y_pred_dt)
rf_cm = confusion_matrix(y_test, y_pred_rf)

# Create subplots for all confusion matrices
fig, axes = plt.subplots(2, 3, figsize=(18, 8))
fig.suptitle("Confusion Matrices", fontsize=16)

# Confusion Matrix for Logistic Regression
sns.heatmap(logistic_cm, annot=True, fmt='d', cmap='Blues', ax=axes[0, 0])
axes[0, 0].set_title('Logistic Regression')
axes[0, 0].set_xlabel('Predicted')
axes[0, 0].set_ylabel('Actual')

# Confusion Matrix for K-Nearest Neighbors (KNN)
sns.heatmap(knn_cm, annot=True, fmt='d', cmap='Blues', ax=axes[0, 1])
axes[0, 1].set_title('K-Nearest Neighbors (KNN)')
axes[0, 1].set_xlabel('Predicted')
axes[0, 1].set_ylabel('Actual')

# Confusion Matrix for Support Vector Machine (SVM)
sns.heatmap(svm_cm, annot=True, fmt='d', cmap='Blues', ax=axes[0, 2])
axes[0, 2].set_title('Support Vector Machine (SVM)')
axes[0, 2].set_xlabel('Predicted')
axes[0, 2].set_ylabel('Actual')

# Confusion Matrix for Decision Tree
sns.heatmap(dt_cm, annot=True, fmt='d', cmap='Blues', ax=axes[1, 0])
axes[1, 0].set_title('Decision Tree')
axes[1, 0].set_xlabel('Predicted')
axes[1, 0].set_ylabel('Actual')

# Confusion Matrix for Random Forest
sns.heatmap(rf_cm, annot=True, fmt='d', cmap='Blues', ax=axes[1, 1])
axes[1, 1].set_title('Random Forest')
axes[1, 1].set_xlabel('Predicted')
axes[1, 1].set_ylabel('Actual')
```

```
# Hide the empty subplot
axes[1, 2].axis('off')

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



Conclusions

In this analysis, we assessed the performance of five different machine learning models for credit card fraud detection. Here are the key findings:

1. Logistic Regression achieved a high accuracy of 97.89%. It exhibited strong precision, capturing 98.84% of actual fraud cases, and recall, correctly identifying 96.59% of fraud cases. The F1 Score, a balanced measure of precision and recall, reached 97.70%, indicating a robust overall performance.

2. K-Nearest Neighbors (KNN) was consistent across two evaluations, both yielding an accuracy of 96.32%. KNN demonstrated commendable precision, successfully identifying 97.65% of predicted fraud cases. Its recall rate was 94.32%, and the F1-Score stood at 95.95%, suggesting an effective balance between precision and recall.
3. Support Vector Machine (SVM) delivered an accuracy of 96.84%, with impeccable precision, capturing all predicted fraud cases (100%). Its recall rate reached 93.18%, and the F1-Score was 96.47%, highlighting its competence in identifying fraud cases.
4. Decision Tree displayed a slightly lower accuracy of 94.21%. Despite this, it offered a respectable precision of 95.29%, accurately identifying fraud cases. The recall rate was 92.05%, and the F1-Score was 93.64%, demonstrating strong overall performance.
5. Random Forest outperformed other models with an accuracy of 98.42%. It excelled in precision, correctly identifying all predicted fraud cases (100%). Its recall rate was 96.59%, and the F1 Score reached 98.27%, indicating superior performance in detecting fraud.

In summary, the Random Forest model demonstrated the highest accuracy and precision in detecting fraudulent transactions, making it the top choice for credit card fraud detection. Logistic Regression, K-Nearest Neighbors, Support Vector Machine, and Decision Tree also exhibited