Import the necessary libraries to compare performance of machine learning models with a focus on the ROC Curve as a key metric.

- Logistic Regression
- Random Forest
- Neural Networks
- XGBoost

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.utils import resample
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold, cross_val_predict
import time
from joblib import Parallel, delayed
print("Libraries successfully imported")

# Use R for implementing Delong's test
# Note: this may require intalling Seurat 3.0.2 if kernel crashes
import os
os.environ['R_HOME'] = "C:/PROGRA~1/R/R-43~1.1"

try:
    import rpy2.robjects as ro
    from rpy2.robjects import pandas2ri
    from rpy2.robjects.packages import importr
    pandas2ri.activate()
    pROC = importr('pROC')
    print("rpy2 and pROC successfully imported")
except Exception as e:
    print(f"Error: Try installing Seurat 3.0.2 {e}")
```

```
Libraries successfully imported
rpy2 and pROC successfully imported
```

Load the dataset

In [2]:
```
data = pd.read_csv(r'C:\Users\tanch\bank-account-fraud-dataset-neurips-2022\Base.csv')
```

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

Out[3]:

	fraud_bool	income	name_email_similarity	prev_address_months_count	current_address_months_cou
0	0	0.3	0.986506	-1	
1	0	0.8	0.617426	-1	
2	0	0.8	0.996707	9	
3	0	0.6	0.475100	11	
4	0	0.9	0.842307	-1	

5 rows × 32 columns

In [4]:
```
# Display basic information about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 32 columns):
 #   Column                             Non-Null Count    Dtype
---  ------                             --------------    -----
 0   fraud_bool                         1000000 non-null  int64
 1   income                             1000000 non-null  float64
 2   name_email_similarity              1000000 non-null  float64
 3   prev_address_months_count          1000000 non-null  int64
 4   current_address_months_count       1000000 non-null  int64
 5   customer_age                       1000000 non-null  int64
 6   days_since_request                 1000000 non-null  float64
 7   intended_balcon_amount             1000000 non-null  float64
 8   payment_type                       1000000 non-null  object
 9   zip_count_4w                       1000000 non-null  int64
 10  velocity_6h                        1000000 non-null  float64
 11  velocity_24h                       1000000 non-null  float64
 12  velocity_4w                        1000000 non-null  float64
 13  bank_branch_count_8w               1000000 non-null  int64
 14  date_of_birth_distinct_emails_4w   1000000 non-null  int64
 15  employment_status                  1000000 non-null  object
 16  credit_risk_score                  1000000 non-null  int64
 17  email_is_free                      1000000 non-null  int64
 18  housing_status                     1000000 non-null  object
 19  phone_home_valid                   1000000 non-null  int64
 20  phone_mobile_valid                 1000000 non-null  int64
 21  bank_months_count                  1000000 non-null  int64
 22  has_other_cards                    1000000 non-null  int64
 23  proposed_credit_limit              1000000 non-null  float64
 24  foreign_request                    1000000 non-null  int64
 25  source                             1000000 non-null  object
 26  session_length_in_minutes          1000000 non-null  float64
 27  device_os                          1000000 non-null  object
 28  keep_alive_session                 1000000 non-null  int64
 29  device_distinct_emails_8w          1000000 non-null  int64
 30  device_fraud_count                 1000000 non-null  int64
 31  month                              1000000 non-null  int64
dtypes: float64(9), int64(18), object(5)
memory usage: 244.1+ MB
```

In [5]:
```
# Statistical summary of the dataset
display(data.describe())
```

	fraud_bool	income	name_email_similarity	prev_address_months_count	current_adc
count	1000000.000000	1000000.000000	1000000.000000	1000000.000000	
mean	0.011029	0.562696	0.493694	16.718568	
std	0.104438	0.290343	0.289125	44.046230	
min	0.000000	0.100000	0.000001	-1.000000	
25%	0.000000	0.300000	0.225216	-1.000000	
50%	0.000000	0.600000	0.492153	-1.000000	
75%	0.000000	0.800000	0.755567	12.000000	
max	1.000000	0.900000	0.999999	383.000000	

8 rows × 27 columns

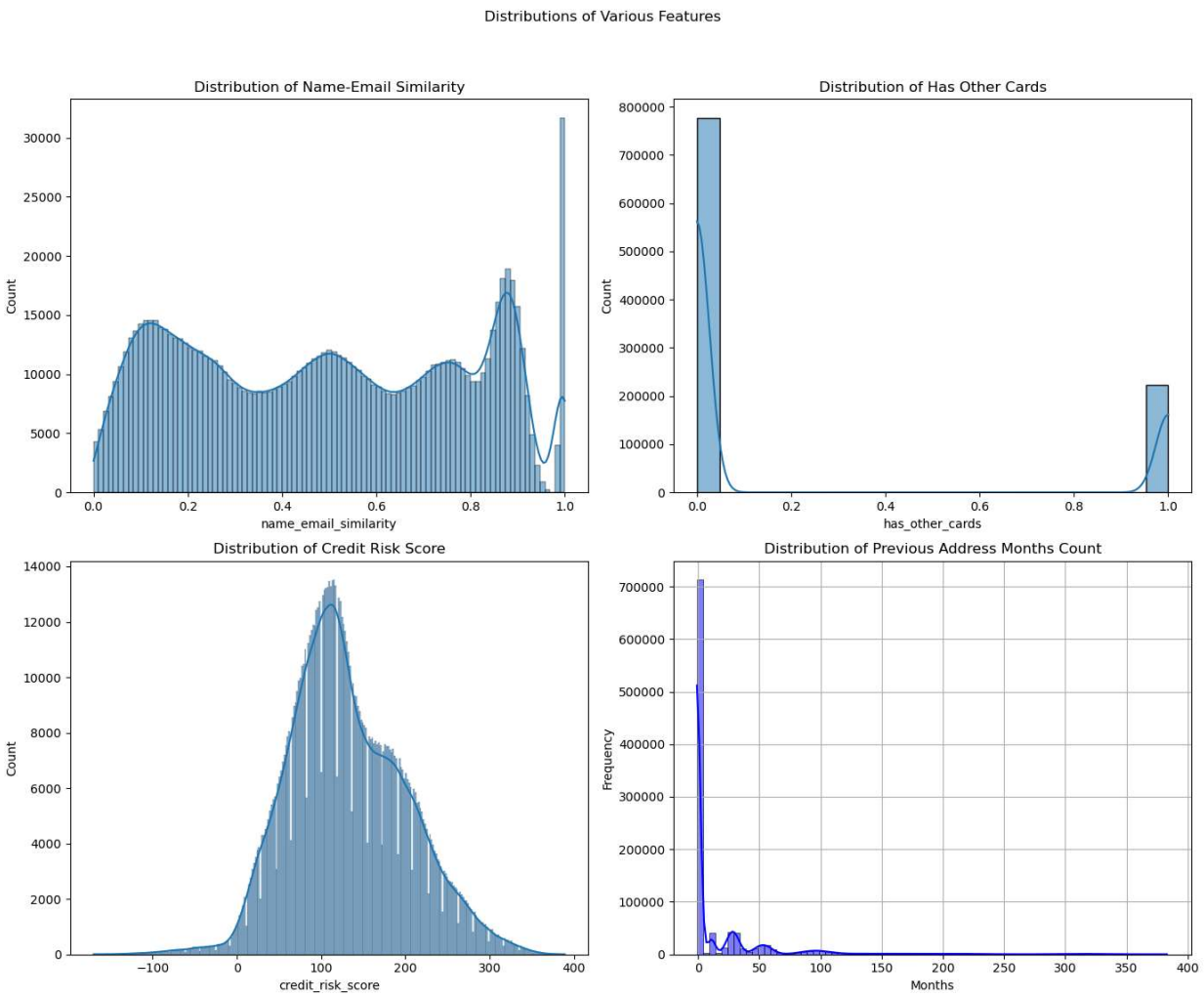
In [6]:
```
# Get the summary statistics
summary = data.describe()

# Identify columns where the minimum value is negative
negative_columns = summary.loc['min'][summary.loc['min'] < 0].index.tolist()

# Display the columns with negative minimum values
print("Columns with negative minimum values:", negative_columns)
```

```
Columns with negative minimum values: ['prev_address_months_count', 'current_address_
months_count', 'intended_balcon_amount', 'velocity_6h', 'credit_risk_score', 'bank_mo
nths_count', 'session_length_in_minutes', 'device_distinct_emails_8w']
```

Plot the distribution for select features

In [7]:
```
fig, axes = plt.subplots(2, 2, figsize=(14, 12))
fig.suptitle('Distributions of Various Features')

# Plot distribution for name_email_similarity
sns.histplot(data['name_email_similarity'], kde=True, ax=axes[0, 0])
axes[0, 0].set_title('Distribution of Name-Email Similarity')

# Plot distribution for has_other_cards
sns.histplot(data['has_other_cards'], kde=True, ax=axes[0, 1])
axes[0, 1].set_title('Distribution of Has Other Cards')

# Plot distribution for credit_risk_score
sns.histplot(data['credit_risk_score'], kde=True, ax=axes[1, 0])
axes[1, 0].set_title('Distribution of Credit Risk Score')

# Plot distribution for prev_address_months_count
sns.histplot(data['prev_address_months_count'], kde=True, color='blue', binwidth=5, ax
axes[1, 1].set_title('Distribution of Previous Address Months Count')
axes[1, 1].set_xlabel('Months')
axes[1, 1].set_ylabel('Frequency')
```

```
axes[1, 1].grid(True)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust the layout to make room for the ma
plt.show()
```



In [8]:
```
if 'month' in data.columns:
    unique_months = data['month'].unique()
    print("Unique values in the 'month' column:", unique_months)
else:
    print("The 'month' column does not exist in the data.")
```

```
Unique values in the 'month' column: [0 1 2 3 4 5 6 7]
```

Data preprocessing: Replace negative values with the median, encode categorical values, split the data set into training and test sets based on temporal information, and scale the features

In [9]:
```
# Replace negative values with the median of their respective columns
for col in negative_columns:
    median = data[col][data[col] >= 0].median()  # Calculate median ignoring negative
    count_negatives = (data[col] < 0).sum()  # Count negative values
    data[col] = data[col].apply(lambda x: median if x < 0 else x)
    print(f'Column "{col}": {count_negatives} negative values updated to median.')
print("Negative values replaced with medians.")
```

```
# Encode categorical features if any
data = pd.get_dummies(data)
print("Categorical features encoded.")

# Split the dataset based on temporal information (months 0-5 for training, months 6-7
train_data = data[data['month'] <= 5] # First 6 months (month 0 thru month 5)
test_data = data[data['month'] >= 6] # Last 2 months (month 6 & 7)
print("Dataset split based on temporal information.")

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('fraud_bool', axis=1), d
print("Dataset split into training and test sets.")

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# TODO: Display successful execution of feature scaling
print("Feature scaling completed.")
```

```
Column "prev_address_months_count": 712920 negative values updated to median.
Column "current_address_months_count": 4254 negative values updated to median.
Column "intended_balcon_amount": 742523 negative values updated to median.
Column "velocity_6h": 44 negative values updated to median.
Column "credit_risk_score": 14445 negative values updated to median.
Column "bank_months_count": 253635 negative values updated to median.
Column "session_length_in_minutes": 2015 negative values updated to median.
Column "device_distinct_emails_8w": 359 negative values updated to median.
Negative values replaced with medians.
Categorical features encoded.
Dataset split based on temporal information.
Dataset split into training and test sets.
Feature scaling completed.
```

Initialize and train the models using hypertuning parameters from Kaggle notebooks

In [10]:

```
# Logistic Regression with the best parameters and parallel processing
logreg = LogisticRegression(
    penalty='l1',
    solver='saga',
    C=0.0425311356094682,
    tol=1.0741774867265052e-05,
    max_iter=1000,
    n_jobs=-1  # Use all available CPU cores
)

# Random Forest with the best parameters and parallel processing
forest = RandomForestClassifier(
    bootstrap=False,
    n_estimators=900,
    max_features='log2',
    max_depth=25,
    min_samples_split=14,
    min_samples_leaf=9,
    criterion='entropy',
```

```
    n_jobs=-1  # Use all available CPU cores
)

# Neural Network (MLPClassifier) with the best parameters
neural_net = MLPClassifier(max_iter=2000)

# XGBoost with the hyperparameters removed due to slightly lower performance
xgboost = XGBClassifier(
    n_jobs=-1  # Use all available CPU cores
)

# Train the models with progress printing
print("Training Logistic Regression...")
start_time = time.time()
logreg.fit(X_train, y_train)
print(f"Logistic Regression trained in {time.time() - start_time:.2f} seconds.")

print("Training Random Forest...")
start_time = time.time()
forest.fit(X_train, y_train)
print(f"Random Forest trained in {time.time() - start_time:.2f} seconds.")

print("Training Neural Network...")
start_time = time.time()
neural_net.fit(X_train, y_train)
print(f"Neural Network trained in {time.time() - start_time:.2f} seconds.")

print("Training XGBoost...")
start_time = time.time()
xgboost.fit(X_train, y_train)
print(f"XGBoost trained in {time.time() - start_time:.2f} seconds.")
```

```
Training Logistic Regression...
Logistic Regression trained in 149.13 seconds.
Training Random Forest...
Random Forest trained in 1653.02 seconds.
Training Neural Network...
Neural Network trained in 146.91 seconds.
Training XGBoost...
XGBoost trained in 12.61 seconds.
```

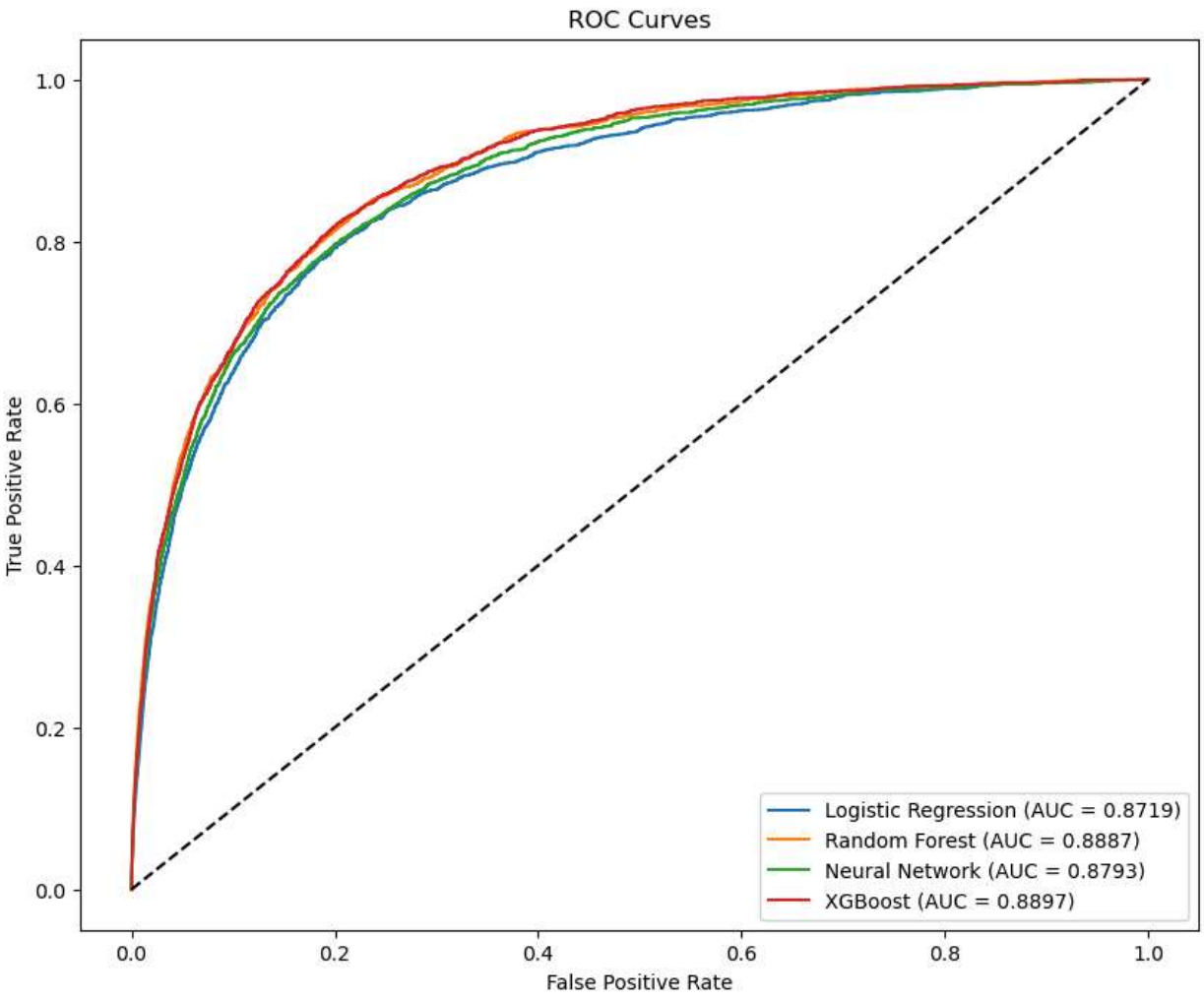
In [11]:

```
# Evaluate the models' AUC of the ROC
logreg_probs = logreg.predict_proba(X_test)[:, 1]
forest_probs = forest.predict_proba(X_test)[:, 1]
neural_net_probs = neural_net.predict_proba(X_test)[:, 1]
xgboost_probs = xgboost.predict_proba(X_test)[:, 1]

logreg_auc = roc_auc_score(y_test, logreg_probs)
forest_auc = roc_auc_score(y_test, forest_probs)
nn_auc = roc_auc_score(y_test, neural_net_probs)
xgb_auc = roc_auc_score(y_test, xgboost_probs)

print(f'Logistic Regression AUC: {logreg_auc:.4f}')
print(f'Random Forest AUC: {forest_auc:.4f}')
print(f'Neural Network AUC: {nn_auc:.4f}')
print(f'XGBoost AUC: {xgb_auc:.4f}')
```

```
Logistic Regression AUC: 0.8719
Random Forest AUC: 0.8887
Neural Network AUC: 0.8793
XGBoost AUC: 0.8897
```

Plot ROC curves

In [12]:
```python
def plot_roc_curve(model, model_name):
    probs = model.predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.4f})')

plt.figure(figsize=(10, 8))
plot_roc_curve(logreg, 'Logistic Regression')
plot_roc_curve(forest, 'Random Forest')
plot_roc_curve(neural_net, 'Neural Network')
plot_roc_curve(xgboost, 'XGBoost')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend()
plt.show()
```



Cross Validation for robustness

In [13]:
```
# Perform temporal cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Placeholder for the average AUCs
logreg_scores = []
forest_scores = []
neural_net_scores = []
xgboost_scores = []

def train_and_evaluate(train_index, test_index):
    X_train_cv, X_test_cv = X_train[train_index], X_train[test_index]
    y_train_cv, y_test_cv = y_train.iloc[train_index], y_train.iloc[test_index]

    # Train the models
    logreg.fit(X_train_cv, y_train_cv)
    forest.fit(X_train_cv, y_train_cv)
    neural_net.fit(X_train_cv, y_train_cv)
    xgboost.fit(X_train_cv, y_train_cv)

    # Predict probabilities
    logreg_proba = logreg.predict_proba(X_test_cv)[:, 1]
    forest_proba = forest.predict_proba(X_test_cv)[:, 1]
    neural_net_proba = neural_net.predict_proba(X_test_cv)[:, 1]
    xgboost_proba = xgboost.predict_proba(X_test_cv)[:, 1]

    # Evaluate the models using AUC
    logreg_score = roc_auc_score(y_test_cv, logreg_proba)
    forest_score = roc_auc_score(y_test_cv, forest_proba)
    neural_net_score = roc_auc_score(y_test_cv, neural_net_proba)
    xgboost_score = roc_auc_score(y_test_cv, xgboost_proba)

    return logreg_score, forest_score, neural_net_score, xgboost_score

# Parallel processing of cross-validation folds
results = Parallel(n_jobs=-1)(
    delayed(train_and_evaluate)(train_index, test_index)
    for train_index, test_index in tscv.split(X_train)
)

# Unpack the results
logreg_scores, forest_scores, neural_net_scores, xgboost_scores = zip(*results)

# Print the average AUC for each model
print(f'Temporal Cross-Validation - Logistic Regression Average AUC: {np.mean(logreg_s
print(f'Temporal Cross-Validation - Random Forest Average AUC: {np.mean(forest_scores)
print(f'Temporal Cross-Validation - Neural Network Average AUC: {np.mean(neural_net_sc
print(f'Temporal Cross-Validation - XGBoost Average AUC: {np.mean(xgboost_scores):.4f}
```

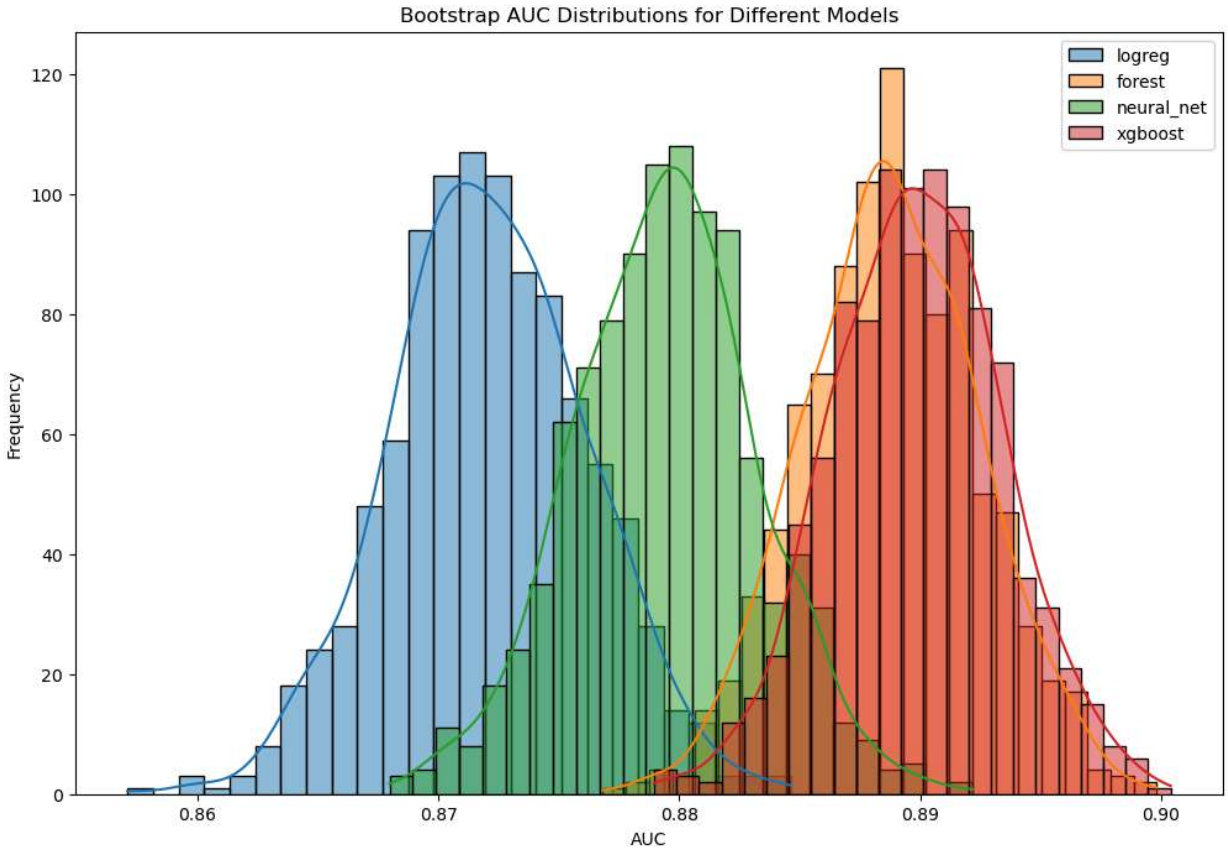
```
Temporal Cross-Validation - Logistic Regression Average AUC: 0.8714
Temporal Cross-Validation - Random Forest Average AUC: 0.8814
Temporal Cross-Validation - Neural Network Average AUC: 0.8393
Temporal Cross-Validation - XGBoost Average AUC: 0.8762
```

Bootstrapping to examine confidence intervals

In [14]:

```
# Number of bootstrap samples
n_bootstraps = 1000

# Placeholder for bootstrapped AUCs
bootstrapped_aucs = {
    'logreg': [],
    'forest': [],
    'neural_net': [],
    'xgboost': []
}

# Define models and their names
models = [logreg, forest, neural_net, xgboost]
model_names = ['logreg', 'forest', 'neural_net', 'xgboost']

# Generate bootstrap samples and calculate AUC for each model
for i in range(n_bootstraps):
    # Generate a bootstrap sample
    X_test_resampled, y_test_resampled = resample(X_test, y_test)

    for model, name in zip(models, model_names):
        probs = model.predict_proba(X_test_resampled)[:, 1]
        auc_score = roc_auc_score(y_test_resampled, probs)
        bootstrapped_aucs[name].append(auc_score)

# Convert lists to numpy arrays for easier statistical analysis
for key in bootstrapped_aucs:
    bootstrapped_aucs[key] = np.array(bootstrapped_aucs[key])

# Calculate means and 95% confidence intervals
for model in model_names:
    mean_auc = np.mean(bootstrapped_aucs[model])
    ci_lower = np.percentile(bootstrapped_aucs[model], 2.5)
    ci_upper = np.percentile(bootstrapped_aucs[model], 97.5)
    print(f'{model} AUC: {mean_auc:.4f} (95% CI: {ci_lower:.4f} - {ci_upper:.4f})')

# Compare the AUC distributions between models
plt.figure(figsize=(12, 8))
for model in model_names:
    sns.histplot(bootstrapped_aucs[model], kde=True, label=model)

plt.title('Bootstrap AUC Distributions for Different Models')
plt.xlabel('AUC')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

```
logreg AUC: 0.8721 (95% CI: 0.8642 - 0.8800)
forest AUC: 0.8888 (95% CI: 0.8817 - 0.8960)
neural_net AUC: 0.8794 (95% CI: 0.8718 - 0.8869)
xgboost AUC: 0.8899 (95% CI: 0.8829 - 0.8969)
```



In [15]:

```
# Delongs test: p-values
# Get predicted probabilities for the positive class
probs_logreg = logreg.predict_proba(X_test)[:, 1]
probs_forest = forest.predict_proba(X_test)[:, 1]
probs_neural_net = neural_net.predict_proba(X_test)[:, 1]
probs_xgboost = xgboost.predict_proba(X_test)[:, 1]

# Convert data to R objects
r_y_test = ro.FloatVector(y_test)
r_probs_logreg = ro.FloatVector(probs_logreg)
r_probs_forest = ro.FloatVector(probs_forest)
r_probs_neural_net = ro.FloatVector(probs_neural_net)
r_probs_xgboost = ro.FloatVector(probs_xgboost)

# Perform DeLong's test using pROC
roc_logreg = pROC.roc(r_y_test, r_probs_logreg)
roc_forest = pROC.roc(r_y_test, r_probs_forest)
roc_neural_net = pROC.roc(r_y_test, r_probs_neural_net)
roc_xgboost = pROC.roc(r_y_test, r_probs_xgboost)

# Compare AUCs using DeLong's test
test_logreg_vs_forest = pROC.roc_test(roc_logreg, roc_forest, method="delong")
test_logreg_vs_neural_net = pROC.roc_test(roc_logreg, roc_neural_net, method="delong")
test_logreg_vs_xgboost = pROC.roc_test(roc_logreg, roc_xgboost, method="delong")
test_forest_vs_neural_net = pROC.roc_test(roc_forest, roc_neural_net, method="delong")
test_forest_vs_xgboost = pROC.roc_test(roc_forest, roc_xgboost, method="delong")

# Print the p-values of the tests
print("Logistic Regression vs Random Forest:", test_logreg_vs_forest[2][0])
print("Logistic Regression vs Neural Network:", test_logreg_vs_neural_net[2][0])
print("Logistic Regression vs XGBoost:", test_logreg_vs_xgboost[2][0])
```

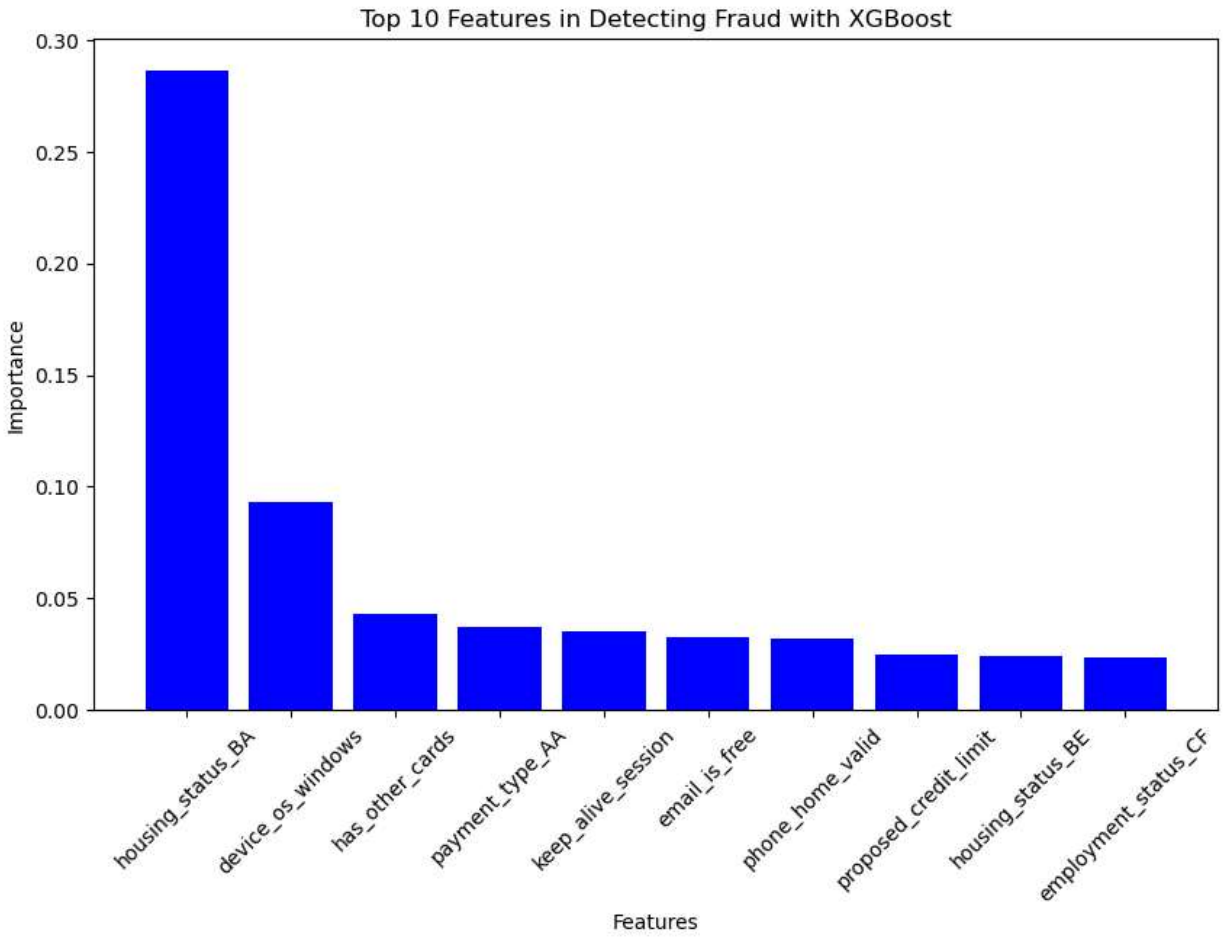
```
print("Random Forest vs Neural Network:", test_forest_vs_neural_net[2][0])
print("Random Forest vs XGBoost:", test_forest_vs_xgboost[2][0])
```

```
R[write to console]: Setting levels: control = 0, case = 1

R[write to console]: Setting direction: controls < cases

R[write to console]: Setting levels: control = 0, case = 1

R[write to console]: Setting direction: controls < cases

R[write to console]: Setting levels: control = 0, case = 1

R[write to console]: Setting direction: controls < cases

R[write to console]: Setting levels: control = 0, case = 1

R[write to console]: Setting direction: controls < cases
```

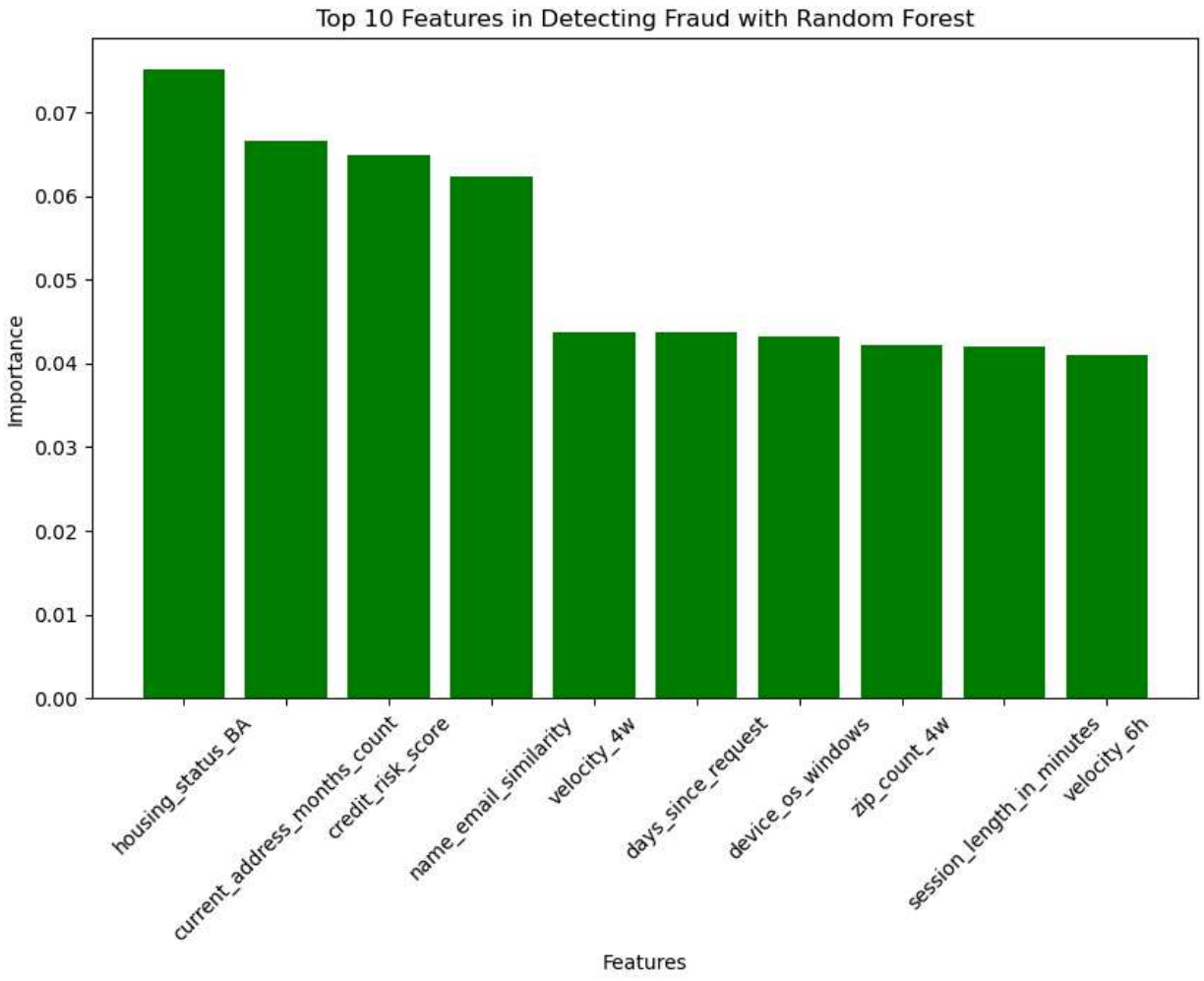
```
Logistic Regression vs Random Forest: 0.8719016950768135
Logistic Regression vs Neural Network: 0.8719016950768135
Logistic Regression vs XGBoost: 0.8719016950768135
Random Forest vs Neural Network: 0.8886516591521129
Random Forest vs XGBoost: 0.8886516591521129
```

In [16]:
```
# Get column names
X_train_df = pd.DataFrame(X_train, columns=data.drop('fraud_bool', axis=1).columns)

# Train XGBoost model
#model = xgb.XGBClassifier(random_state=42)
xgboost.fit(X_train_df, y_train)

# Get feature importances and match them to the column names
xgb_importances = pd.DataFrame(xgboost.feature_importances_,
                                    index = X_train_df.columns,
                                    columns=['importance']).sort_values('importance', a
```

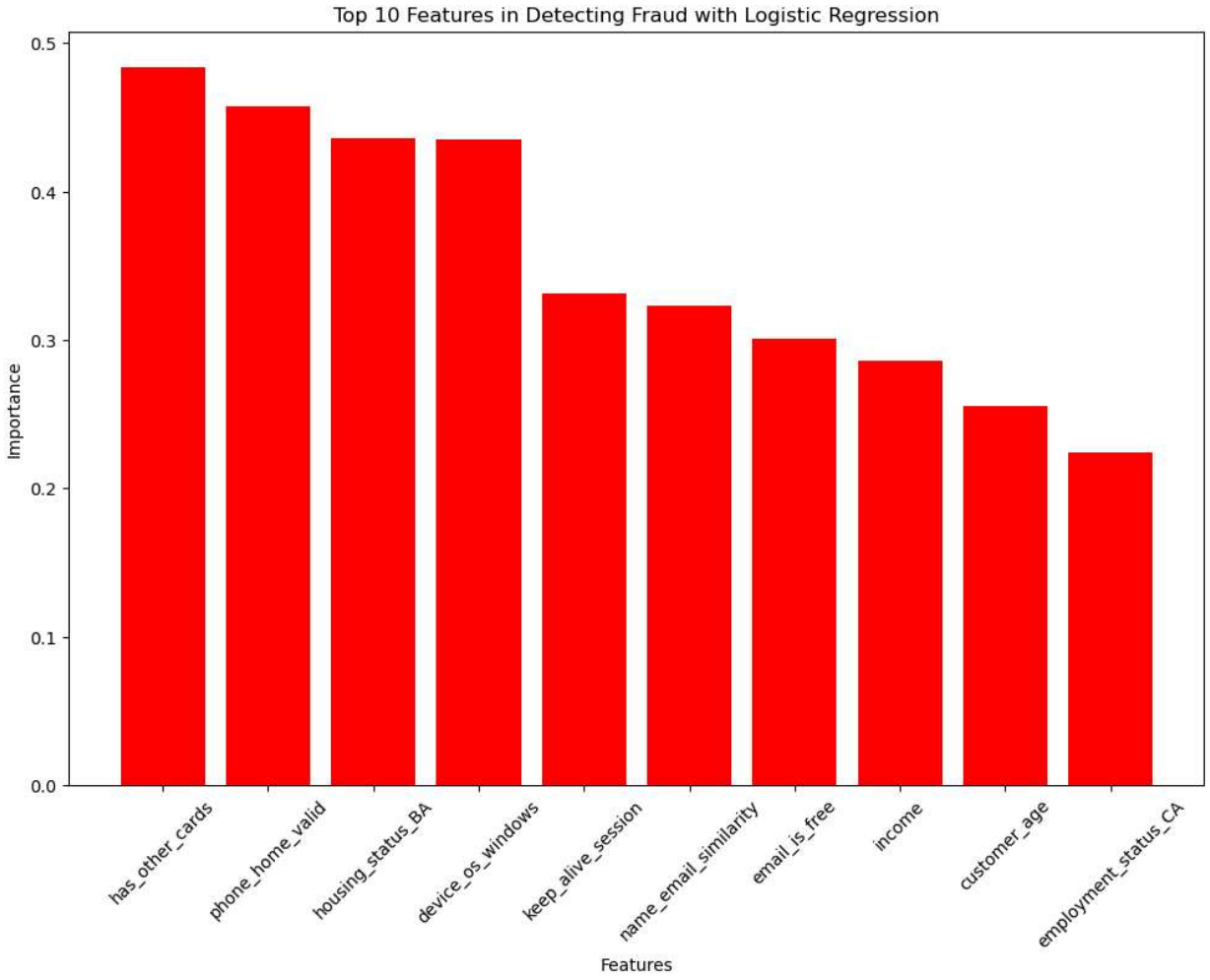
In [17]:
```
# Plot the top 10 features
top_features = xgb_importances.head(10)
plt.figure(figsize=(10, 6))
plt.bar(top_features.index, top_features['importance'], color='blue')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Top 10 Features in Detecting Fraud with XGBoost')
plt.xticks(rotation=45)
plt.show()
```



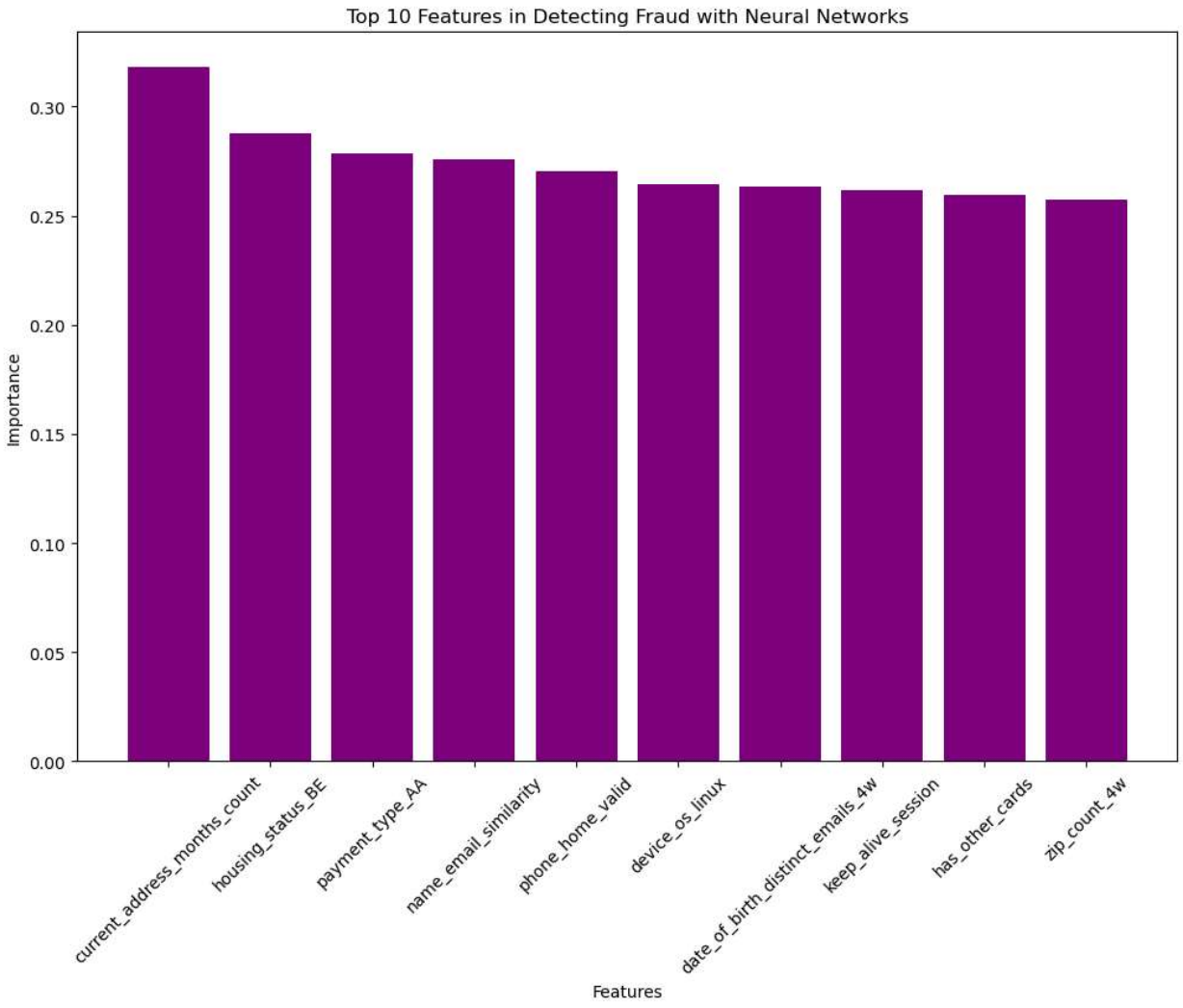
In [18]:
```
# Get feature importances and match them to the column names
rf_importances = pd.DataFrame(forest.feature_importances_,
                              index = X_train_df.columns,
                              columns=['importance']).sort_values('importance', ascend

# Plot the top 10 features
top_rf_features = rf_importances.head(10)
plt.figure(figsize=(10, 6))
plt.bar(top_rf_features.index, top_rf_features['importance'], color='green')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Top 10 Features in Detecting Fraud with Random Forest')
plt.xticks(rotation=45)
plt.show()
```



In [19]:
```
# Get coefficients and match them to the column names
logreg_importances = pd.DataFrame(abs(logreg.coef_[0]),
                                  index=X_train_df.columns,
                                  columns=['importance']).sort_values('importance', as

# Plot the top 10 features
top_logreg_features = logreg_importances.head(10)
plt.figure(figsize=(12, 8))
plt.bar(top_logreg_features.index, top_logreg_features['importance'], color='red')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Top 10 Features in Detecting Fraud with Logistic Regression')
plt.xticks(rotation=45)
plt.show()
```



In [20]:
```
# Approximate feature importance by averaging the absolute weights of the first layer
neural_net_importances = pd.DataFrame(data=np.mean(np.abs(neural_net.coefs_[0]), axis=
                                index=X_train_df.columns,
                                columns=['importance']).sort_values(by='importance', as

# Plot the top 10 features
top_neural_net_features = neural_net_importances.head(10)
plt.figure(figsize=(12, 8))
plt.bar(top_neural_net_features.index, top_neural_net_features['importance'], color='p
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Top 10 Features in Detecting Fraud with Neural Networks')
plt.xticks(rotation=45)
plt.show()
```



In [22]:

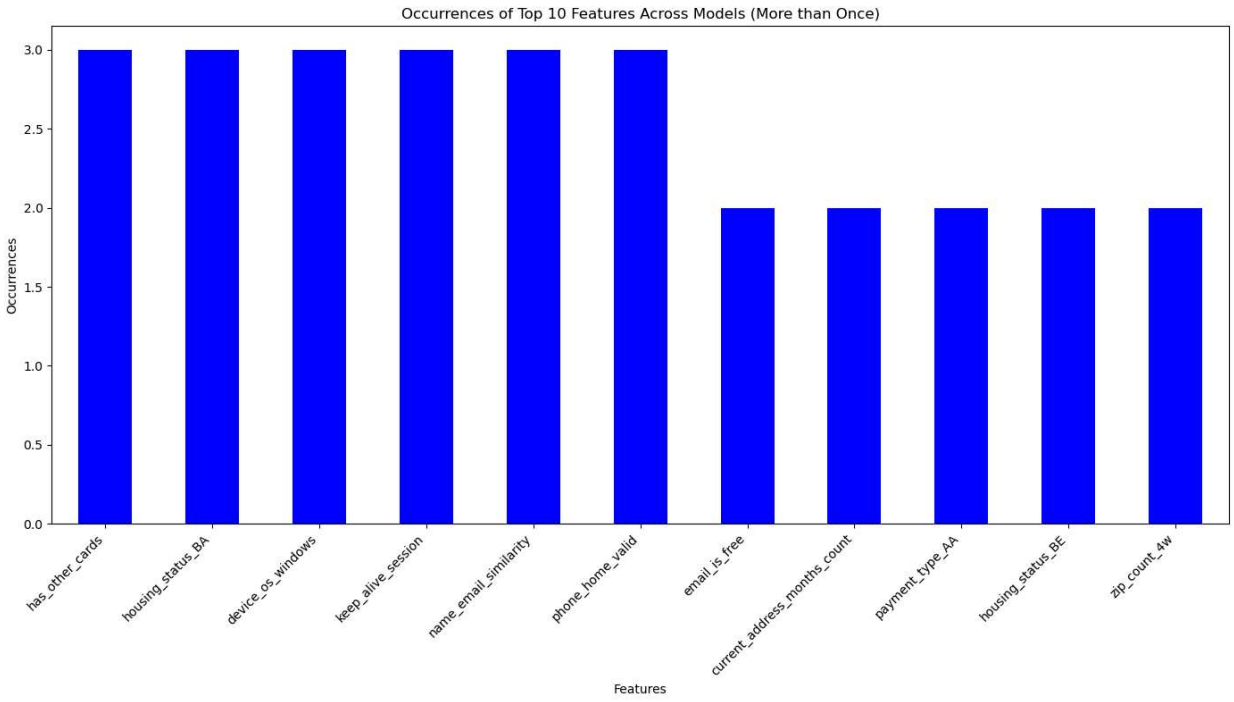
```
# Extract the top 10 features for each model
top_10_rf = rf_importances.head(10)
top_10_logreg = logreg_importances.head(10)
top_10_neural_net = neural_net_importances.head(10)
top_10_xgboost = xgb_importances.head(10)

# Create a DataFrame for better visualization including XGBoost
feature_summary = pd.DataFrame({
    'Random Forest': top_10_rf.index.tolist(),
    'RF Importance': top_10_rf['importance'].tolist(),
    'Logistic Regression': top_10_logreg.index.tolist(),
    'LR Importance': top_10_logreg['importance'].tolist(),
    'Neural Networks': top_10_neural_net.index.tolist(),
    'NN Importance': top_10_neural_net['importance'].tolist(),
    'XGBoost': top_10_xgboost.index.tolist(),
    'XGB Importance': top_10_xgboost['importance'].tolist()
})

# Display the DataFrame
print(feature_summary)
```

```
                Random Forest  RF Importance    Logistic Regression  \
0            housing_status_BA       0.075090        has_other_cards
1  current_address_months_count       0.066532       phone_home_valid
2            credit_risk_score       0.064917      housing_status_BA
3        name_email_similarity       0.062368      device_os_windows
4                  velocity_4w       0.043707     keep_alive_session
5           days_since_request       0.043687  name_email_similarity
6            device_os_windows       0.043266          email_is_free
7                 zip_count_4w       0.042282                 income
8    session_length_in_minutes       0.042068           customer_age
9                  velocity_6h       0.041019   employment_status_CA

   LR Importance                   Neural Networks  NN Importance  \
0       0.483560     current_address_months_count       0.318279
1       0.457731                housing_status_BE       0.287526
2       0.436147                  payment_type_AA       0.278267
3       0.434927            name_email_similarity       0.275730
4       0.331013                 phone_home_valid       0.270493
5       0.323219                  device_os_linux       0.264219
6       0.300714  date_of_birth_distinct_emails_4w       0.263505
7       0.285863               keep_alive_session       0.261471
8       0.255878                  has_other_cards       0.259780
9       0.224607                     zip_count_4w       0.257411

                XGBoost  XGB Importance
0      housing_status_BA        0.286266
1      device_os_windows        0.093340
2        has_other_cards        0.043025
3        payment_type_AA        0.037112
4     keep_alive_session        0.034906
5           email_is_free        0.032882
6       phone_home_valid        0.032198
7  proposed_credit_limit        0.024647
8      housing_status_BE        0.024337
9   employment_status_CF        0.023775
```

In [23]:

```python
# List of top 10 features for each model
top_features_logreg = pd.Series(logreg_importances.head(10).index)
top_features_forest = pd.Series(rf_importances.head(10).index)
top_features_neural_net = pd.Series(neural_net_importances.head(10).index)
top_features_xgboost = pd.Series(xgb_importances.head(10).index)

# Combine all top features
all_top_features = pd.concat([top_features_logreg, top_features_forest, top_features_r

# Count occurrences of each feature
feature_counts = all_top_features.value_counts()

# Filter features that occurred more than once
filtered_feature_counts = feature_counts[feature_counts > 1]

# Plot the feature occurrences
plt.figure(figsize=(14, 8))
ax = filtered_feature_counts.plot(kind='bar', color='blue')
plt.xlabel('Features')
plt.ylabel('Occurrences')
plt.title('Occurrences of Top 10 Features Across Models (More than Once)')
plt.xticks(rotation=45, ha='right')
```

```
plt.tight_layout()  # Adjust layout to make room for labels
plt.show()
```



In [24]:
```
filtered_feature_counts
```

Out[24]:
```
has_other_cards                3
housing_status_BA              3
device_os_windows              3
keep_alive_session             3
name_email_similarity          3
phone_home_valid               3
email_is_free                  2
current_address_months_count   2
payment_type_AA                2
housing_status_BE              2
zip_count_4w                   2
dtype: int64
```

Ensemble methods for all 4 models

In [29]:
```
print(f'Neural Network AUC: {nn_auc:.4f}')
print(f'XGBoost AUC: {xgb_auc:.4f}')
print(f'Logistic Regression AUC: {logreg_auc:.4f}')
print(f'Random Forest AUC: {forest_auc:.4f}')
forest_probs = forest.predict_proba(X_test)[:, 1]

# Ensemble method #1: Simple Averaging (Soft Voting)
avg_probs = (logreg_probs + forest_probs + neural_net_probs + xgboost_probs) / 4
ensemble_auc = roc_auc_score(y_test, avg_probs)
print(f'Ensemble AUC (Simple Averaging): {ensemble_auc:.4f}')

# Ensemble method #2: Weighted Voting
weights = {
    'logreg': logreg_auc,
    'forest': forest_auc,
    'neural_net': nn_auc,
```

```
    'xgboost': xgb_auc
}
weighted_probs = (logreg_probs * weights['logreg'] +
                  forest_probs * weights['forest'] +
                  neural_net_probs * weights['neural_net'] +
                  xgboost_probs * weights['xgboost']) / sum(weights.values())
weighted_ensemble_auc = roc_auc_score(y_test, weighted_probs)
print(f'Weighted Ensemble AUC: {weighted_ensemble_auc:.4f}')

# Ensemble method #3: Stacking

stacked_features = np.column_stack((logreg_probs, forest_probs, neural_net_probs, xgbc
meta_classifier = LogisticRegression()

# Generate out-of-fold predictions for stacking
stacked_train_features = np.column_stack((
    cross_val_predict(logreg, X_train, y_train, cv=5, method='predict_proba')[:, 1],
    cross_val_predict(forest, X_train, y_train, cv=5, method='predict_proba')[:, 1],
    cross_val_predict(neural_net, X_train, y_train, cv=5, method='predict_proba')[:, 1
    cross_val_predict(xgboost, X_train, y_train, cv=5, method='predict_proba')[:, 1]
))
meta_classifier.fit(stacked_train_features, y_train)
final_stacked_predictions = meta_classifier.predict_proba(stacked_features)[:, 1]
stacked_auc = roc_auc_score(y_test, final_stacked_predictions)
print(f'Stacking Ensemble AUC: {stacked_auc:.4f}')
```

```
Neural Network AUC: 0.8793
XGBoost AUC: 0.8897
Logistic Regression AUC: 0.8719
Random Forest AUC: 0.8887
Ensemble AUC (Simple Averaging): 0.8933
Weighted Ensemble AUC: 0.8934
Stacking Ensemble AUC: 0.8912
```

Ensemble methods for top 3 models

In [33]:

```
# Ensemble method #1: Simple Averaging (Soft Voting)
avg_probs = (forest_probs + neural_net_probs + xgboost_probs) / 3
ensemble_auc = roc_auc_score(y_test, avg_probs)
print(f'Ensemble AUC (Simple Averaging): {ensemble_auc:.4f}')

# Ensemble method #2: Weighted Voting
weights = {
    'forest': forest_auc,
    'neural_net': nn_auc,
    'xgboost': xgb_auc
}
weighted_probs = (forest_probs * weights['forest'] +
                  neural_net_probs * weights['neural_net'] +
                  xgboost_probs * weights['xgboost']) / sum(weights.values())
weighted_ensemble_auc = roc_auc_score(y_test, weighted_probs)
print(f'Weighted Ensemble AUC: {weighted_ensemble_auc:.4f}')

# Ensemble method #3: Stacking

stacked_features = np.column_stack((forest_probs, neural_net_probs, xgboost_probs))
meta_classifier = LogisticRegression()

# Generate out-of-fold predictions for stacking
```

```
stacked_train_features = np.column_stack((
    cross_val_predict(forest, X_train, y_train, cv=5, method='predict_proba')[:, 1],
    cross_val_predict(neural_net, X_train, y_train, cv=5, method='predict_proba')[:, 1
    cross_val_predict(xgboost, X_train, y_train, cv=5, method='predict_proba')[:, 1]
))
meta_classifier.fit(stacked_train_features, y_train)
final_stacked_predictions = meta_classifier.predict_proba(stacked_features)[:, 1]
stacked_auc = roc_auc_score(y_test, final_stacked_predictions)
print(f'Stacking Ensemble AUC: {stacked_auc:.4f}')
```

```
Ensemble AUC (Simple Averaging): 0.8943
Weighted Ensemble AUC: 0.8943
Stacking Ensemble AUC: 0.8909
```

Optimized Ensemble method for top 2 models

In [34]:

```
# Ensemble method #1: Simple Averaging (Soft Voting)
avg_probs = (forest_probs + xgboost_probs) / 2
ensemble_auc = roc_auc_score(y_test, avg_probs)
print(f'Ensemble AUC (Simple Averaging): {ensemble_auc:.4f}')

# Ensemble method #2: Weighted Voting
weights = {
    'forest': forest_auc,
    'xgboost': xgb_auc
}
weighted_probs = (forest_probs * weights['forest'] +
                  xgboost_probs * weights['xgboost']) / sum(weights.values())
weighted_ensemble_auc = roc_auc_score(y_test, weighted_probs)
print(f'Weighted Ensemble AUC: {weighted_ensemble_auc:.4f}')

# Ensemble method #3: Stacking

stacked_features = np.column_stack((forest_probs, xgboost_probs))
meta_classifier = LogisticRegression()

# Generate out-of-fold predictions for stacking
stacked_train_features = np.column_stack((
    cross_val_predict(forest, X_train, y_train, cv=5, method='predict_proba')[:, 1],
    cross_val_predict(xgboost, X_train, y_train, cv=5, method='predict_proba')[:, 1]
))
meta_classifier.fit(stacked_train_features, y_train)
final_stacked_predictions = meta_classifier.predict_proba(stacked_features)[:, 1]
stacked_auc = roc_auc_score(y_test, final_stacked_predictions)
print(f'Stacking Ensemble AUC: {stacked_auc:.4f}')
```

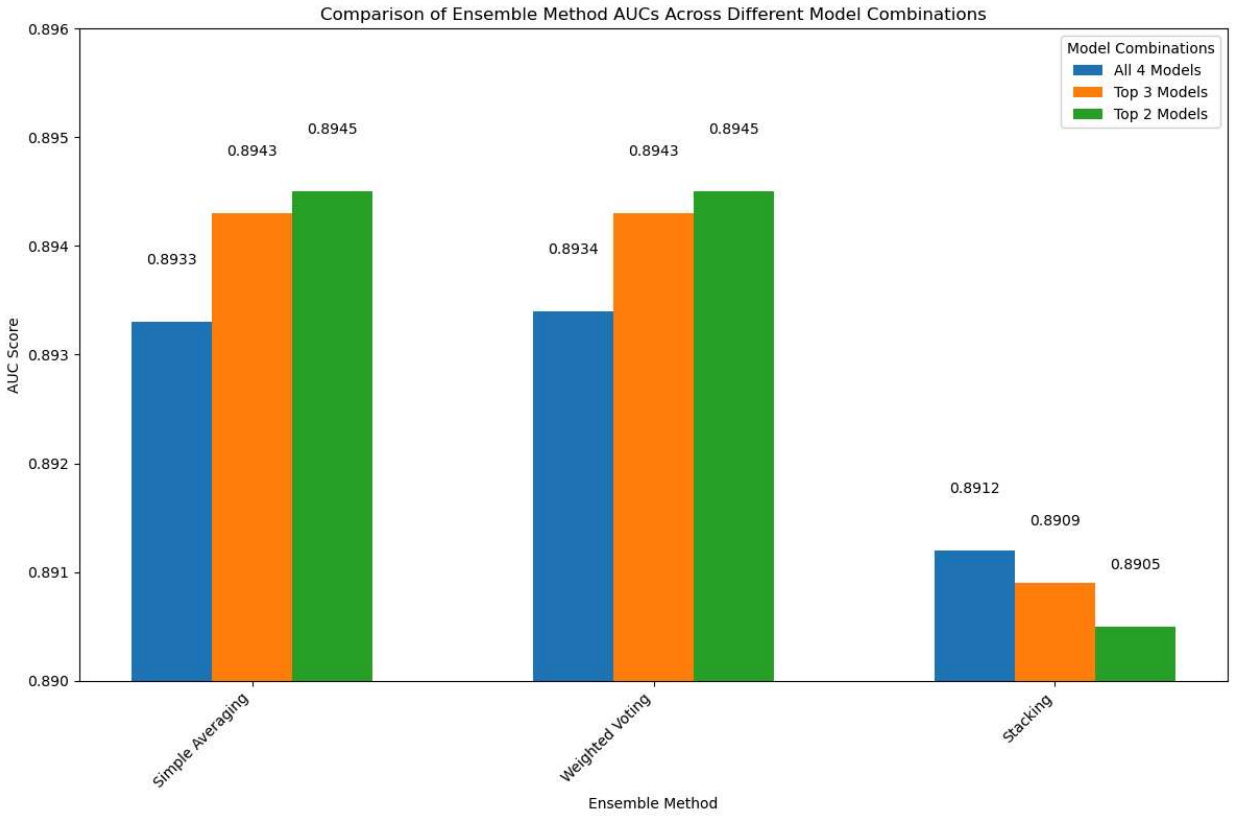
```
Ensemble AUC (Simple Averaging): 0.8945
Weighted Ensemble AUC: 0.8945
Stacking Ensemble AUC: 0.8905
```

In [38]:

```
# Results from ensemble methods visualized

# AUC results for the ensemble methods
ensemble_aucs = {
    'All 4 Models': {
        'Simple Averaging': 0.8933,
        'Weighted Voting': 0.8934,
        'Stacking': 0.8912
    },
```

```
    'Top 3 Models': {
        'Simple Averaging': 0.8943,
        'Weighted Voting': 0.8943,
        'Stacking': 0.8909
    },
    'Top 2 Models': {
        'Simple Averaging': 0.8945,
        'Weighted Voting': 0.8945,
        'Stacking': 0.8905
    }
}

# Define bar width and positions
bar_width = 0.2
positions_all_4 = np.arange(len(ensemble_aucs['All 4 Models']))
positions_top_3 = positions_all_4 + bar_width
positions_top_2 = positions_top_3 + bar_width

# Plotting the results
plt.figure(figsize=(12, 8))

# Plot each group of bars
plt.bar(
    positions_all_4,
    [auc for auc in ensemble_aucs['All 4 Models'].values()],
    width=bar_width,
    label='All 4 Models'
)
plt.bar(
    positions_top_3,
    [auc for auc in ensemble_aucs['Top 3 Models'].values()],
    width=bar_width,
    label='Top 3 Models'
)
plt.bar(
    positions_top_2,
    [auc for auc in ensemble_aucs['Top 2 Models'].values()],
    width=bar_width,
    label='Top 2 Models'
)

# Adding the numerical scores on the bars for clarity
for i, auc in enumerate(ensemble_aucs['All 4 Models'].values()):
    plt.text(positions_all_4[i], auc + 0.0005, f'{auc:.4f}', ha='center', va='bottom')
for i, auc in enumerate(ensemble_aucs['Top 3 Models'].values()):
    plt.text(positions_top_3[i], auc + 0.0005, f'{auc:.4f}', ha='center', va='bottom')
for i, auc in enumerate(ensemble_aucs['Top 2 Models'].values()):
    plt.text(positions_top_2[i], auc + 0.0005, f'{auc:.4f}', ha='center', va='bottom')

# Adding title and labels
plt.title('Comparison of Ensemble Method AUCs Across Different Model Combinations')
plt.xlabel('Ensemble Method')
plt.ylabel('AUC Score')
plt.xticks(
    positions_all_4 + bar_width,  # Centers the ticks between the bars
    ['Simple Averaging', 'Weighted Voting', 'Stacking']
)
plt.xticks(rotation=45, ha='right')

# Show the plot
```

```
plt.ylim(0.89, 0.896)  # Adjusting the y-axis limits for better visual representation
plt.legend(title="Model Combinations")
plt.tight_layout()
plt.show()
```



In [32]:
```
data.to_csv('cleaned_data.csv', index=False)
print("Dataset saved as 'cleaned_data.csv'")
```

```
Dataset saved as 'cleaned_data.csv'
```

In [ ]: