```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.decomposition import IncrementalPCA
from tqdm import tqdm



# Load training dataset
train_data = pd.read_csv("fraudTrain.csv")


# Load testing dataset
test_data = pd.read_csv("fraudTest.csv")



# Combine training and testing data for encoding consistency
combined_data = pd.concat([train_data, test_data], axis=0)


# Extract relevant features from the "trans_date_trans_time" column
def extract_datetime_features(df):
    df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'])
    df['day_of_week'] = df['trans_date_trans_time'].dt.dayofweek
    df['hour_of_day'] = df['trans_date_trans_time'].dt.hour
    df.drop('trans_date_trans_time', axis=1, inplace=True)
    return df

combined_data = extract_datetime_features(combined_data)


# Drop irrelevant columns (you can customize this based on your data)
columns_to_drop = ["first", "last", "job", "dob", "trans_num", "street"]
combined_data.drop(columns_to_drop, axis=1, inplace=True)


# Separate features and target variable
X_combined = combined_data.drop("is_fraud", axis=1)
y_combined = combined_data["is_fraud"]


# Encode the "merchant" and "category" columns using LabelEncoder
label_encoder = LabelEncoder()
X_combined["merchant"] = label_encoder.fit_transform(X_combined["merchant"])
X_combined["category"] = label_encoder.fit_transform(X_combined["category"])



# One-hot encode categorical variables with handle_unknown='ignore'
categorical_columns = ["gender", "city", "state"]
onehot_encoder = OneHotEncoder(sparse=False, drop="first", handle_unknown='ignore')
X_combined_categorical = onehot_encoder.fit_transform(X_combined[categorical_columns])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
```

```
# Standardize the numeric features
scaler = StandardScaler()
X_combined_numeric = scaler.fit_transform(X_combined.drop(categorical_columns, axis=1))



# Combine one-hot encoded categorical features with numeric features
X_combined_encoded = np.hstack((X_combined_numeric, X_combined_categorical))


# Split the combined data back into training and test datasets
X_train = X_combined_encoded[:len(train_data)]
X_test = X_combined_encoded[len(train_data):]
y_train = y_combined[:len(train_data)]
y_test = y_combined[len(train_data):]
```

```
import numpy as np

# Identify rows with NaN values in y_train
nan_indices = np.isnan(y_train)

# Remove rows with NaN values from both X_train and y_train
X_train_cleaned = X_train[~nan_indices]
y_train_cleaned = y_train[~nan_indices]

# Address class imbalance using SMOTE on the cleaned training data
sm = SMOTE(random_state=42)
X_resampled, y_resampled = sm.fit_resample(X_train_cleaned, y_train_cleaned)



# Apply Incremental PCA for dimensionality reduction
n_components = 100  # Adjust the number of components as needed
ipca = IncrementalPCA(n_components=n_components)


# Apply Incremental PCA to training data with progress bar
for batch in tqdm(np.array_split(X_resampled, 10), desc="Applying Incremental PCA"):
    ipca.partial_fit(batch)
```

```
Applying Incremental PCA: 100%|██████████| 10/10 [00:11<00:00,  1.13s/it]
```

```
from sklearn.impute import SimpleImputer

# Handle missing values in X_test
imputer = SimpleImputer(strategy='mean')
X_test_imputed = imputer.fit_transform(X_test)

# Transform the testing data after handling missing values
X_test_pca = ipca.transform(X_test_imputed)



# Define and train the Random Forest model
rf_classifier = RandomForestClassifier(random_state=42)



# Fit the model with the resampled data
rf_classifier.fit(X_resampled_pca, y_resampled)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Predict using the trained model
y_pred = rf_classifier.predict(X_test_pca)


from sklearn.impute import SimpleImputer

# Convert pandas Series to numpy array
y_test_array = y_test.to_numpy()

# Handle missing values in y_test
imputer = SimpleImputer(strategy='most_frequent')
y_test_imputed = imputer.fit_transform(y_test_array.reshape(-1, 1))

# Flatten the imputed array
y_test_imputed = y_test_imputed.flatten()

# Calculate accuracy, confusion matrix, and classification report
accuracy = accuracy_score(y_test_imputed, y_pred)
confusion = confusion_matrix(y_test_imputed, y_pred)
report = classification_report(y_test_imputed, y_pred)



# Display the necessary report on ML Model Metrics for evaluation
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{confusion}")
print(f"Classification Report:\n{report}")
```

```
Accuracy: 0.9969195225259915
Confusion Matrix:
[[15534     1]
 [   47     0]]
```

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

         0.0       1.00      1.00      1.00     15535
         1.0       0.00      0.00      0.00        47

    accuracy                           1.00     15582
   macro avg       0.50      0.50      0.50     15582
weighted avg       0.99      1.00      1.00     15582
```

Start coding or generate with AI.