```
import pandas as pd

# Load datasets
train_df = pd.read_csv("/content/fraudTrain.csv")  # Replace with actual file path
test_df = pd.read_csv("/content/fraudTest.csv")

# Display basic info
print(train_df.info())
print(train_df.head())
```

```
 11  state                  7814 non-null   object
 12  zip                    7814 non-null   float64
 13  lat                    7814 non-null   float64
 14  long                   7814 non-null   float64
 15  city_pop               7814 non-null   float64
 16  job                    7814 non-null   object
 17  dob                    7814 non-null   object
 18  trans_num              7814 non-null   object
 19  unix_time              7814 non-null   float64
 20  merch_lat              7814 non-null   float64
 21  merch_long             7814 non-null   float64
 22  is_fraud               7814 non-null   float64
dtypes: float64(9), int64(2), object(12)
memory usage: 1.4+ MB
None
   Unnamed: 0 trans_date_trans_time            cc_num  \
0           0   2019-01-01 00:00:18  2703186189652095
1           1   2019-01-01 00:00:44      630423337322
2           2   2019-01-01 00:00:51    38859492057661
3           3   2019-01-01 00:01:16  3534093764340240
4           4   2019-01-01 00:03:06   375534208663984

                             merchant       category     amt      first  \
0          fraud_Rippin, Kub and Mann       misc_net    4.97   Jennifer
1     fraud_Heller, Gutmann and Zieme    grocery_pos  107.23  Stephanie
2                fraud_Lind-Buckridge  entertainment  220.11     Edward
3  fraud_Kutch, Hermiston and Farrell  gas_transport   45.00     Jeremy
4                 fraud_Keeling-Crist       misc_pos   41.96      Tyler

      last gender                        street  ...      lat      long  \
0    Banks      F                561 Perry Cove  ...  36.0788  -81.1781
1     Gill      F  43039 Riley Greens Suite 393  ...  48.8878 -118.2105
2  Sanchez      M      594 White Dale Suite 530  ...  42.1808 -112.2620
3    White      M   9443 Cynthia Court Apt. 038  ...  46.2306 -112.1138
4   Garcia      M              408 Bradley Rest  ...  38.4207  -79.4629

   city_pop                              job         dob  \
0    3495.0        Psychologist, counselling  1988-03-09
1     149.0  Special educational needs teacher  1978-06-21
2    4154.0      Nature conservation officer  1962-01-19
3    1939.0                  Patent attorney  1967-01-12
4      99.0   Dance movement psychotherapist  1986-03-28

                          trans_num     unix_time  merch_lat  merch_long  \
0  0b242abb623afc578575680df30655b9  1.325376e+09  36.011293  -82.048315
1  1f76529f8574734946361c461b024d99  1.325376e+09  49.159047 -118.186462
2  a1a22d70485983eac12b5b88dad1cf95  1.325376e+09  43.150704 -112.154481
3  6b849c168bdad6f867558c3793159a81  1.325376e+09  47.034331 -112.561071
4  a41d7549acf90789359a9aa5346dcb46  1.325376e+09  38.674999  -78.632459

   is_fraud
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0

[5 rows x 23 columns]
```

```
print(train_df.isnull().sum())  # Count missing values
print(train_df.dtypes)  # Check data types
```

```
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
```

```
last                     0
gender                   1
street                   1
city                     1
state                    1
zip                      1
lat                      1
long                     1
city_pop                 1
job                      1
dob                      1
trans_num                1
unix_time                1
merch_lat                1
merch_long               1
is_fraud                 1
dtype: int64
Unnamed: 0                 int64
trans_date_trans_time     object
cc_num                     int64
merchant                  object
category                  object
amt                      float64
first                     object
last                      object
gender                    object
street                    object
city                      object
state                     object
zip                      float64
lat                      float64
long                     float64
city_pop                 float64
job                       object
dob                       object
trans_num                 object
unix_time                float64
merch_lat                float64
merch_long               float64
is_fraud                 float64
dtype: object
```

```
print(train_df['is_fraud'].value_counts())  # Check fraud vs. non-fraud cases
```

```
is_fraud
0.0    7769
1.0      45
Name: count, dtype: int64
```

```
# Fill missing values in 'is_fraud' column
train_df['is_fraud'].fillna(0, inplace=True)  # Assuming missing values mean non-fraud (0)

# Verify there are no more missing values
print(train_df['is_fraud'].isnull().sum())  # Should print 0
```

```
0
```

```
from sklearn.impute import SimpleImputer

# Impute missing values with the mean for numerical columns
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)  # Apply imputation

# Convert back to DataFrame
X_train = pd.DataFrame(X_train_imputed, columns=X_train.columns)

# Verify there are no NaNs
print("Missing values in X_train after imputation:", X_train.isnull().sum().sum())  # Should be 0
```

```
Missing values in X_train after imputation: 0
```

```
# Apply SMOTE
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Print new class distribution
```

```
print("Class distribution after SMOTE:")
print(y_train_resampled.value_counts())
```

```
Class distribution after SMOTE:
is_fraud
0.0    6216
1.0    3108
Name: count, dtype: int64
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the validation set
y_pred = rf_model.predict(X_val)

# Evaluate the model
print("Model Performance:")
print(classification_report(y_val, y_pred))
```

```
Model Performance:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1554
         1.0       0.64      0.78      0.70         9

    accuracy                           1.00      1563
   macro avg       0.82      0.89      0.85      1563
weighted avg       1.00      1.00      1.00      1563
```

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameters to tune
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees
    'max_depth': [10, 20, None],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples per split
    'min_samples_leaf': [1, 2, 4]  # Minimum samples per leaf
}

# Initialize Grid Search
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
                           param_grid, cv=3, scoring='recall', n_jobs=-1)

# Run Grid Search on the resampled dataset
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Train Random Forest with the best parameters
rf_optimized = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)

rf_optimized.fit(X_train_resampled, y_train_resampled)
```

```
# Make predictions on the validation set
y_pred_optimized = rf_optimized.predict(X_val)

# Evaluate the optimized model
print("Optimized Model Performance:")
print(classification_report(y_val, y_pred_optimized))
```

```
Optimized Model Performance:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1554
         1.0       0.64      0.78      0.70         9

    accuracy                           1.00      1563
   macro avg       0.82      0.89      0.85      1563
weighted avg       1.00      1.00      1.00      1563
```

Generate 10 random numbers using numpy Close

```
# Train Random Forest with class weights
rf_weighted = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1,
    class_weight='balanced',  # Adjust class weights
    random_state=42
)

rf_weighted.fit(X_train_resampled, y_train_resampled)

# Predict on validation set
y_pred_weighted = rf_weighted.predict(X_val)

# Evaluate model
print("Weighted Random Forest Performance:")
print(classification_report(y_val, y_pred_weighted))
```

```
Weighted Random Forest Performance:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1554
         1.0       0.54      0.78      0.64         9

    accuracy                           0.99      1563
   macro avg       0.77      0.89      0.82      1563
weighted avg       1.00      0.99      1.00      1563
```

```
import joblib

# Save the trained model
joblib.dump(rf_weighted, "fraud_detection_model.pkl")
print("Model saved successfully!")
```

```
Model saved successfully!
```

```
# Load the model
model = joblib.load("fraud_detection_model.pkl")

# Make a test prediction
sample_transaction = X_val.iloc[0].values.reshape(1, -1)  # Take one transaction from validation set
prediction = model.predict(sample_transaction)

print("Fraud Prediction:", prediction[0])  # Output: 0 (Legit) or 1 (Fraud)
```

```
Fraud Prediction: 0.0
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but RandomFc
  warnings.warn(
import pandas as pd

# Convert the sample transaction into a DataFrame with column names
sample_transaction_df = pd.DataFrame([X_val.iloc[0]], columns=X_val.columns)

# Make prediction again
prediction = model.predict(sample_transaction_df)
print("Fraud Prediction:", prediction[0])
```

```
Fraud Prediction: 0.0
```

```
%%writefile app.py
from flask import Flask, request, jsonify
import joblib
import pandas as pd

# Load the trained model
model = joblib.load("fraud_detection_model.pkl")

# Define Flask app
app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get JSON data from request
        data = request.json
        features = data['features']  # Extract feature values

        # Convert input data into DataFrame
        input_df = pd.DataFrame([features], columns=['amt', 'city_pop', 'merch_lat', 'merch_long'])  # Update based on your feature names

        # Make prediction
        prediction = model.predict(input_df)[0]

        # Return response
        return jsonify({'Fraud Prediction': int(prediction)})  # Convert NumPy int to Python int

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    app.run(debug=True)
```

```
Writing app.py
```

```
%%writefile requirements.txt
flask
joblib
pandas
scikit-learn
requests
gunicorn
```

```
Writing requirements.txt
```

```
from google.colab import files

# Download both files
files.download("app.py")
files.download("requirements.txt")
```

```
import joblib
import pandas as pd

# Load trained model
model = joblib.load("fraud_detection_model.pkl")

# Print expected feature names
```

```
print("Model was trained with these features:")
print(model.feature_names_in_)  # This prints the expected feature names
```

```
Model was trained with these features:
['amt' 'city_pop' 'merch_lat' 'merch_long' 'category_food_dining'
 'category_gas_transport' 'category_grocery_net' 'category_grocery_pos'
 'category_health_fitness' 'category_home' 'category_kids_pets'
 'category_misc_net' 'category_misc_pos' 'category_personal_care'
 'category_shopping_net' 'category_shopping_pos' 'category_travel']
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
import joblib

# Load training and validation data (Ensure X_train_resampled and y_train_resampled are available)
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=500, class_weight="balanced", random_state=42),
    "XGBoost": XGBClassifier(n_estimators=100, max_depth=6, learning_rate=0.1, random_state=42)
}

# Train and evaluate each model
best_model = None
best_f1_score = 0
model_results = {}

for name, model in models.items():
    print(f"\nTraining {name}...")
    model.fit(X_train_resampled, y_train_resampled)

    # Predictions
    y_pred = model.predict(X_val)

    # Evaluate model
    report = classification_report(y_val, y_pred, output_dict=True)
    f1_score = report["1"]["f1-score"]  # Focus on fraud detection (Class 1)

    print(f"\n{name} Performance:\n", classification_report(y_val, y_pred))

    # Store results
    model_results[name] = f1_score

    # Select the best model based on F1-score for fraud (Class 1)
    if f1_score > best_f1_score:
        best_f1_score = f1_score
        best_model = model

# Print Best Model
print("\nBest Model:", best_model)

# Save the best model
joblib.dump(best_model, "best_fraud_detection_model.pkl")
print("\nBest model saved as 'best_fraud_detection_model.pkl'")
```

```
Training Random Forest...
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-21-6ed188367925> in <cell line: 0>()
     26     # Evaluate model
     27     report = classification_report(y_val, y_pred, output_dict=True)
---> 28     f1_score = report["1"]["f1-score"]  # Focus on fraud detection (Class 1)
     29 
     30     print(f"\n{name} Performance:\n", classification_report(y_val, y_pred))

KeyError: '1'
```

Next steps: Explain error

```
print("Fraud cases in validation set:", y_val.value_counts())
```

```
Fraud cases in validation set: is_fraud
0.0    1554
1.0       9
Name: count, dtype: int64
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
import joblib

# Define models to compare
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=500, class_weight="balanced", random_state=42),
    "XGBoost": XGBClassifier(n_estimators=100, max_depth=6, learning_rate=0.1, random_state=42)
}

# Track best model
best_model = None
best_f1_score = 0
model_results = {}

for name, model in models.items():
    print(f"\nTraining {name}...")
    model.fit(X_train_resampled, y_train_resampled)

    # Predictions
    y_pred = model.predict(X_val)

    # Evaluate model
    report = classification_report(y_val, y_pred, output_dict=True)

    # Get F1-score for fraud cases (Class 1.0)
    f1_score = report["1.0"]["f1-score"]

    print(f"\n{name} Performance:\n", classification_report(y_val, y_pred))

    # Store results
    model_results[name] = f1_score

    # Select the best model based on F1-score for fraud detection
    if f1_score > best_f1_score:
        best_f1_score = f1_score
        best_model = model

# Print Best Model
print("\nBest Model:", best_model)

# Save the best model
joblib.dump(best_model, "best_fraud_detection_model.pkl")
print("\nBest model saved as 'best_fraud_detection_model.pkl'")
```

```

Training Random Forest...

Random Forest Performance:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1554
         1.0       0.64      0.78      0.70         9

    accuracy                           1.00      1563
   macro avg       0.82      0.89      0.85      1563
weighted avg       1.00      1.00      1.00      1563


Training Logistic Regression...
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Performance:
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98      1554
         1.0       0.11      0.78      0.19         9

    accuracy                           0.96      1563
   macro avg       0.55      0.87      0.59      1563
weighted avg       0.99      0.96      0.98      1563


Training XGBoost...

XGBoost Performance:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1554
         1.0       0.58      0.78      0.67         9

    accuracy                           1.00      1563
   macro avg       0.79      0.89      0.83      1563
weighted avg       1.00      1.00      1.00      1563


Best Model: RandomForestClassifier(max_depth=10, random_state=42)

Best model saved as 'best_fraud_detection_model.pkl'
```