```
import pandas as pd

df = pd.read_csv('/content/client_churn_synthetic.csv')
display(df.head())
```

	customer_id	gender	senior_citizen	partner	dependents	tenure	contract	payment_method
0	CUST100000	Male	0	No	No	22	Month-to-month	Credit card (automatic
1	CUST100001	Female	0	Yes	No	25	Month-to-month	Mailed check
2	CUST100002	Female	0	Yes	No	16	Month-to-month	Electronic check
3	CUST100003	Female	0	No	Yes	14	Month-to-month	Bank transfer (automatic
4	CUST100004	Male	0	No	No	13	Month-to-month	Mailed check

```
# Initialize and train the RandomForestClassifier on the resampled data
model_resampled = RandomForestClassifier(n_estimators=100, random_state=42)
model_resampled.fit(X_train_resampled, y_train_resampled)

# Make predictions on the resampled test set
y_pred_resampled = model_resampled.predict(X_test_resampled)

# Evaluate the model on the resampled test set
print("Model Evaluation on Resampled Data:")
print("Accuracy:", accuracy_score(y_test_resampled, y_pred_resampled))
print("\nConfusion Matrix:\n", confusion_matrix(y_test_resampled, y_pred_re
print("\nClassification Report:\n", classification_report(y_test_resampled,
```

```
Model Evaluation on Resampled Data:
Accuracy: 0.8291393529287375

Confusion Matrix:
 [[2079  286]
 [ 522 1842]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.84      2365
           1       0.87      0.78      0.82      2364

    accuracy                           0.83      4729
   macro avg       0.83      0.83      0.83      4729
weighted avg       0.83      0.83      0.83      4729
```

```
from imblearn.over_sampling import SMOTE
from collections import Counter
```

```
# Separate features and target again after previous steps
X = df.drop('churn', axis=1)
y = df['churn']

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

print("Original dataset shape:", Counter(y))
print("Resampled dataset shape:", Counter(y_resampled))

# Now split the resampled data into training and testing sets
# It's important to split AFTER resampling to avoid data leakage
X_train_resampled, X_test_resampled, y_train_resampled, y_test_resampled = train_test_split

# Display the shapes of the new training and testing sets
print("\nShape of X_train after resampling and splitting:", X_train_resampled.shape)
print("Shape of y_train after resampling and splitting:", y_train_resampled.shape)
print("Shape of X_test after resampling and splitting:", X_test_resampled.shape)
print("Shape of y_test after resampling and splitting:", y_test_resampled.shape)
```

```
Original dataset shape: Counter({0: 7881, 1: 2119})
Resampled dataset shape: Counter({1: 7881, 0: 7881})

Shape of X_train after resampling and splitting: (11033, 30)
Shape of y_train after resampling and splitting: (11033,)
Shape of X_test after resampling and splitting: (4729, 30)
Shape of y_test after resampling and splitting: (4729,)
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Define features (X) and target (y)
# Assuming 'churn' is the target variable
X = df.drop('churn', axis=1)
y = df['churn']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, s

# Initialize and train the RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model Evaluation:
Accuracy: 0.7833333333333333

Confusion Matrix:
 [[2345   19]
 [ 631    5]]

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.99      0.88      2364
           1       0.21      0.01      0.02       636

    accuracy                           0.78      3000
   macro avg       0.50      0.50      0.45      3000
weighted avg       0.67      0.78      0.70      3000
```

```
# Example Feature Engineering: Create an interaction term between tenure and monthly charge
df['tenure_monthly_charges'] = df['tenure'] * df['monthly_charges']

# Example Feature Engineering: Create a new feature for the ratio of total charges to tenur
# Avoid division by zero if tenure is 0
df['total_charges_tenure_ratio'] = df.apply(lambda row: row['total_charges'] / row['tenure'

display(df.head())
```

	senior_citizen	tenure	monthly_charges	total_charges	churn	gender_Male	partner_Yes	c
0	0	22	50.44	1127.91	1	True	False	
1	0	25	37.63	924.65	1	False	True	
2	0	16	52.87	841.74	0	False	True	
3	0	14	23.34	322.18	1	False	False	
4	0	13	22.57	282.98	0	True	False	

5 rows × 31 columns

```
# Remove customer_id columns as they are not useful for modeling
df = df.loc[:, ~df.columns.str.startswith('customer_id_')]

display(df.head())
```

	senior_citizen	tenure	monthly_charges	total_charges	churn	gender_Male	partner_Yes	c
0	0	22	50.44	1127.91	1	True	False	
1	0	25	37.63	924.65	1	False	True	
2	0	16	52.87	841.74	0	False	True	
3	0	14	23.34	322.18	1	False	False	
4	0	13	22.57	282.98	0	True	False	

5 rows × 29 columns

```
# Check for missing values
print("Missing values before preprocessing:")
print(df.isnull().sum())

# Handle missing values (example: fill with median for numerical, mode for categorical)
for col in df.columns:
    if df[col].dtype == 'object':
        # Use .loc to avoid the SettingWithCopyWarning and FutureWarning
        df.loc[:, col] = df[col].fillna(df[col].mode()[0])
    else:
        # Use .loc to avoid the SettingWithCopyWarning and FutureWarning
        df.loc[:, col] = df[col].fillna(df[col].median())

print("\nMissing values after preprocessing:")
print(df.isnull().sum())

# Identify categorical and numerical features
categorical_features = df.select_dtypes(include=['object']).columns
numerical_features = df.select_dtypes(exclude=['object']).columns

# Handle potential non-numeric values in numerical columns that were not detected as object
for col in numerical_features:
    if df[col].dtype == 'object':
        # Attempt to convert to numeric, coercing errors
        df.loc[:, col] = pd.to_numeric(df[col], errors='coerce')
        # Fill any new NaNs created by coercion
        df.loc[:, col] = df[col].fillna(df[col].median())


# One-Hot Encode categorical features
df = pd.get_dummies(df, columns=categorical_features, drop_first=True)

# Display the first few rows of the preprocessed data
display(df.head())
```

```
Missing values before preprocessing:
senior_citizen                          0
tenure                                  0
monthly_charges                         0
total_charges                           0
churn                                   0
                                       ..
tech_support_Yes                        0
streaming_tv_No internet service        0
streaming_tv_Yes                        0
streaming_movies_No internet service    0
streaming_movies_Yes                    0
Length: 10028, dtype: int64

Missing values after preprocessing:
senior_citizen                          0
tenure                                  0
monthly_charges                         0
total_charges                           0
churn                                   0
                                       ..
tech_support_Yes                        0
streaming_tv_No internet service        0
streaming_tv_Yes                        0
streaming_movies_No internet service    0
streaming_movies_Yes                    0
Length: 10028, dtype: int64
```

	senior_citizen	tenure	monthly_charges	total_charges	churn	customer_id_CUST100001	cus
0	0	22	50.44	1127.91	1	False	
1	0	25	37.63	924.65	1	True	
2	0	16	52.87	841.74	0	False	
3	0	14	23.34	322.18	1	False	
4	0	13	22.57	282.98	0	False	

5 rows × 10028 columns