```
# Core ML & Data
!pip install -q pandas numpy scikit-learn xgboost shap

# Deep Learning
!pip install -q tensorflow keras

# Visualizations
!pip install -q matplotlib seaborn plotly

# Graph-based Modeling
!pip install -q networkx

!pip install -q streamlit
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 2.0 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 32.6 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 46.6 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 3.8 MB/s eta 0:00:00
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, confusion_matrix

import shap
import xgboost as xgb
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

import networkx as nx
import warnings
warnings.filterwarnings('ignore')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files Churn_Modelling.csv

- **Churn_Modelling.csv**(text/csv) - 684858 bytes, last modified: 7/12/2025 - 100% done

Start coding or generate with AI.

```
import pandas as pd

# Replace with the exact filename if different
df = pd.read_csv("Churn_Modelling.csv")
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProduct
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
!pip install xgboost shap scikit-learn matplotlib seaborn --quiet
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

	0
RowNumber	0
CustomerId	0
Surname	0
CreditScore	0
Geography	0
Gender	0
Age	0
Tenure	0
Balance	0
NumOfProducts	0
HasCrCard	0
IsActiveMember	0
EstimatedSalary	0
Exited	0

```
# Drop columns that are not useful for modeling
df = df.drop(columns=["RowNumber", "CustomerId", "Surname"])


# Label Encoding for Gender
df['Gender'] = df['Gender'].map({'Female': 0, 'Male': 1})

# One-Hot Encoding for Geography
df = pd.get_dummies(df, columns=['Geography'], drop_first=True)



print(df.isnull().sum())  # all zeros
print(df.dtypes)
```

```
CreditScore          0
Gender               0
Age                  0
```

```
Tenure              0
Balance             0
NumOfProducts       0
HasCrCard           0
IsActiveMember      0
EstimatedSalary     0
Exited              0
Geography_Germany   0
Geography_Spain     0
dtype: int64
CreditScore           int64
Gender                int64
Age                   int64
Tenure                int64
Balance             float64
NumOfProducts         int64
HasCrCard             int64
IsActiveMember        int64
EstimatedSalary     float64
Exited                int64
Geography_Germany      bool
Geography_Spain        bool
dtype: object
```

```
scaler = StandardScaler()

# Save column names for clarity
numerical_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']

df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

```
# Features (X) and Target (y)
X = df.drop(columns=["Exited"])
y = df["Exited"]
```

```
from sklearn.model_selection import train_test_split

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

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score

# Initialize and train the model
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

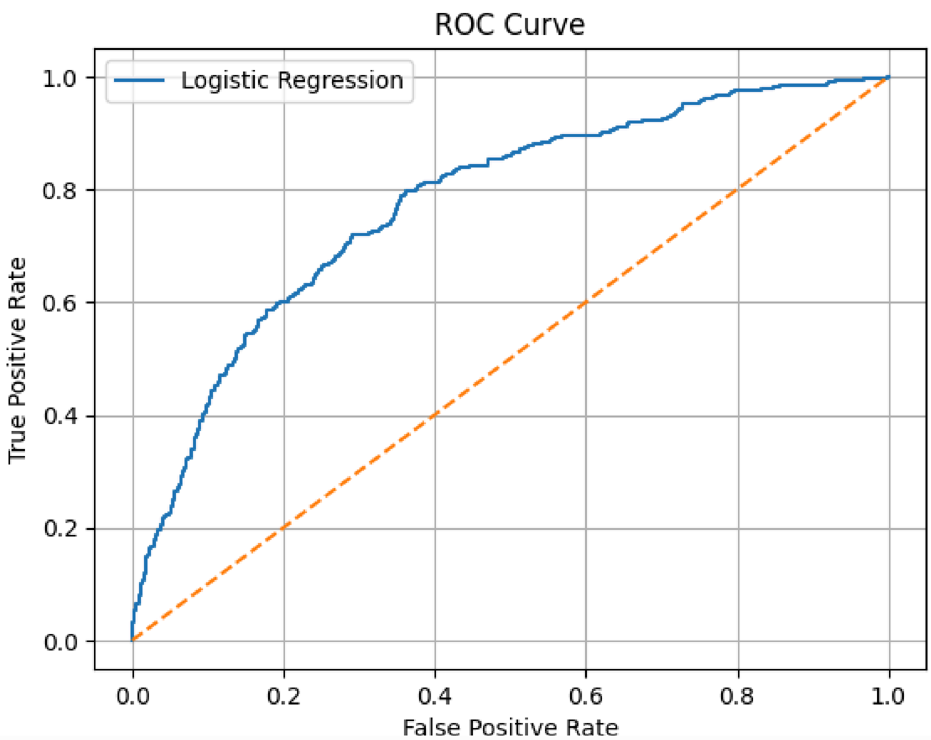
```
# Predict on test set
y_pred = logreg.predict(X_test)
y_proba = logreg.predict_proba(X_test)[:, 1]  # probability of class 1 (churn)
```

```
# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Accuracy
```

```
print("Accuracy Score:", accuracy_score(y_test, y_pred))

# ROC-AUC Score
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
```

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

           0       0.82      0.97      0.89      1593
           1       0.59      0.19      0.28       407

    accuracy                           0.81      2000
   macro avg       0.71      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000

Confusion Matrix:
 [[1540   53]
 [ 331   76]]
Accuracy Score: 0.808
ROC-AUC Score: 0.7748364697517239
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr, label="Logistic Regression")
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid()
plt.show()
```



Here we can see the results are not good at all so we rework and will make adjustments.

Problem Identified: The model is biased as it is being trained on a unbalanced dataset hence, it is returning biased results and the recall, ROC socres are very low. Why the model does this is because a normal model tries to get higher accuracy score. But in our case as their are very less churners the model tries do do well overall and hence predicts "not churn" most of the time, this gives higher accuracy but misses a lot churners.

Solution:

1. Adjust the unbalanced dataset
2. Try powerful models

```
logreg = LogisticRegression(max_iter=1000, class_weight='balanced')
logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=1000)
```

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(class_weight='balanced', random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score

# Train the improved model
logreg_balanced = LogisticRegression(max_iter=1000, class_weight='balanced')
logreg_balanced.fit(X_train, y_train)

# Predict
y_pred_bal = logreg_balanced.predict(X_test)
y_proba_bal = logreg_balanced.predict_proba(X_test)[:, 1]

# Evaluate
print("Classification Report:\n", classification_report(y_test, y_pred_bal))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_bal))
print("Accuracy Score:", accuracy_score(y_test, y_pred_bal))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba_bal))
```

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

           0       0.90      0.72      0.80      1593
           1       0.39      0.70      0.50       407

    accuracy                           0.71      2000
   macro avg       0.65      0.71      0.65      2000
weighted avg       0.80      0.71      0.74      2000

Confusion Matrix:
 [[1142  451]
 [ 122  285]]
Accuracy Score: 0.7135
ROC-AUC Score: 0.7771808788757941
```

Here we notice a lot of improvement overall:

1. The f1-score went up from 0.3 to 0.5
2. It correctly predicts 70% of churners (huge improvement).

3. Accuracy does drop but that was expected as we rebalanced the values.
4. Precision for 1 (ie: people predicted to churn but do not actually do) drops but that realistically is not that bad (better safe than sorry), we will still try to improve this.

Further steps: Use Random forest

🌳 What Is Random Forest?

A collection of decision trees that vote together. Each tree sees a different slice of the data.

The final prediction is made by majority vote (classification).

It’s great at: Handling imbalanced data

Capturing complex patterns

Not needing feature scaling

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

# Train the model
rf = RandomForestClassifier(
    n_estimators=100,           # number of trees in the forest
    max_depth=None,             # let trees grow fully
    class_weight='balanced',    # fix class imbalance
    random_state=42             # for reproducibility
)

rf.fit(X_train, y_train)

# Predict
y_pred_rf = rf.predict(X_test)
y_proba_rf = rf.predict_proba(X_test)[:, 1]

# Evaluate
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Accuracy Score:", accuracy_score(y_test, y_pred_rf))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba_rf))
```

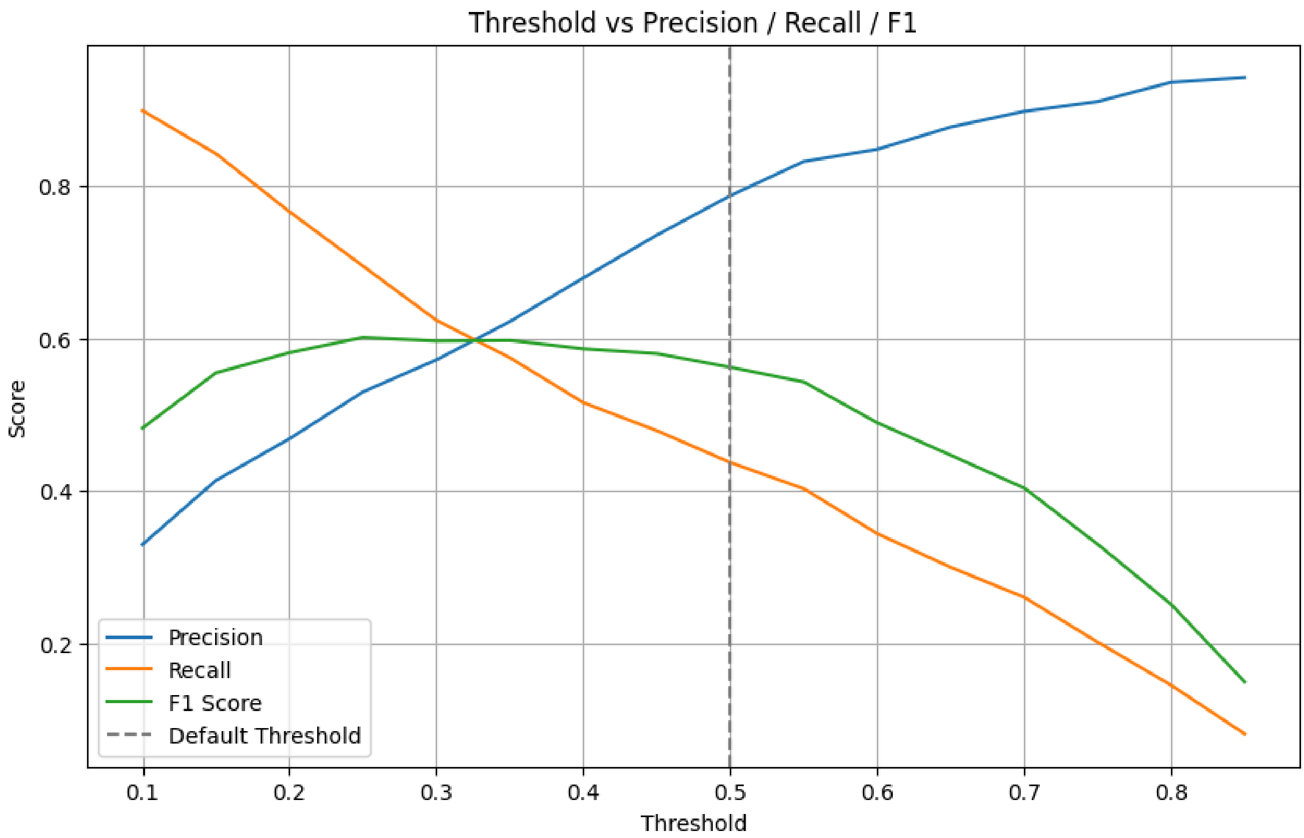
```
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.97      0.92      1593
           1       0.79      0.44      0.56       407

    accuracy                           0.86      2000
   macro avg       0.83      0.70      0.74      2000
weighted avg       0.85      0.86      0.85      2000

Confusion Matrix:
 [[1545   48]
 [ 229  178]]
Accuracy Score: 0.8615
ROC-AUC Score: 0.8536895909777266
```

```
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score

thresholds = np.arange(0.1, 0.9, 0.05)
results = []

for t in thresholds:
    y_pred_thresh = (y_proba_rf >= t).astype(int)
```

```
    precision = precision_score(y_test, y_pred_thresh)
    recall = recall_score(y_test, y_pred_thresh)
    f1 = f1_score(y_test, y_pred_thresh)
    results.append((t, precision, recall, f1))
    print(f"Threshold: {t:.2f} → Precision: {precision:.2f}, Recall: {recall:.2f}, F1: {f1:.2f}")
```

```
Threshold: 0.10 → Precision: 0.33, Recall: 0.90, F1: 0.48
Threshold: 0.15 → Precision: 0.41, Recall: 0.84, F1: 0.56
Threshold: 0.20 → Precision: 0.47, Recall: 0.77, F1: 0.58
Threshold: 0.25 → Precision: 0.53, Recall: 0.70, F1: 0.60
Threshold: 0.30 → Precision: 0.57, Recall: 0.62, F1: 0.60
Threshold: 0.35 → Precision: 0.62, Recall: 0.57, F1: 0.60
Threshold: 0.40 → Precision: 0.68, Recall: 0.52, F1: 0.59
Threshold: 0.45 → Precision: 0.74, Recall: 0.48, F1: 0.58
Threshold: 0.50 → Precision: 0.79, Recall: 0.44, F1: 0.56
Threshold: 0.55 → Precision: 0.83, Recall: 0.40, F1: 0.54
Threshold: 0.60 → Precision: 0.85, Recall: 0.34, F1: 0.49
Threshold: 0.65 → Precision: 0.88, Recall: 0.30, F1: 0.45
Threshold: 0.70 → Precision: 0.90, Recall: 0.26, F1: 0.40
Threshold: 0.75 → Precision: 0.91, Recall: 0.20, F1: 0.33
Threshold: 0.80 → Precision: 0.94, Recall: 0.14, F1: 0.25
Threshold: 0.85 → Precision: 0.94, Recall: 0.08, F1: 0.15
```

```
import matplotlib.pyplot as plt

thresholds, precisions, recalls, f1s = zip(*results)

plt.figure(figsize=(10,6))
plt.plot(thresholds, precisions, label='Precision')
plt.plot(thresholds, recalls, label='Recall')
plt.plot(thresholds, f1s, label='F1 Score')
plt.axvline(0.5, color='gray', linestyle='--', label='Default Threshold')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Threshold vs Precision / Recall / F1')
plt.legend()
plt.grid(True)
plt.show()
```



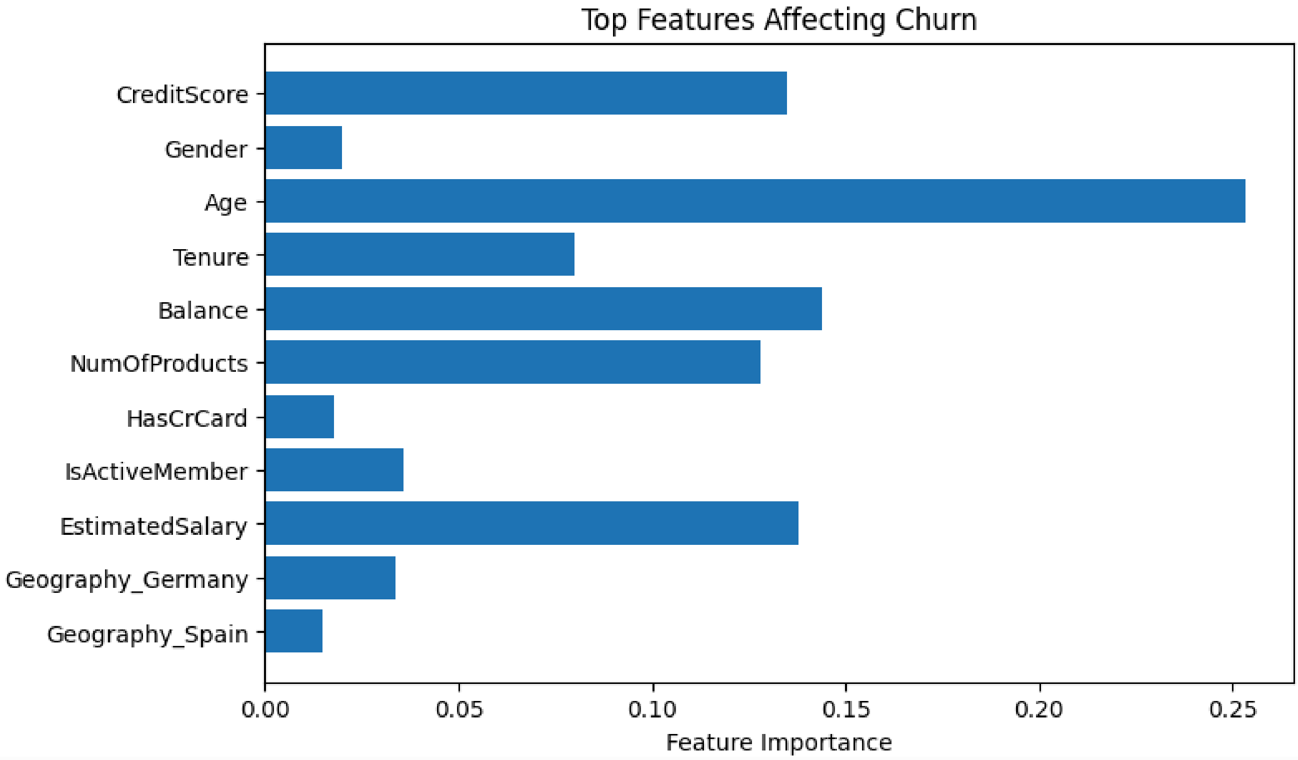

```
y_pred_final = (y_proba_rf >= 0.25).astype(int)
```

```
from sklearn.metrics import classification_report, confusion_matrix

print("Adjusted Classification Report:")
print(classification_report(y_test, y_pred_final))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_final))
print("Accuracy Score:", accuracy_score(y_test, y_pred_rf))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba_rf))
```

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

           0       0.92      0.83      0.87      1593
           1       0.52      0.71      0.60       407

    accuracy                           0.81      2000
   macro avg       0.72      0.77      0.74      2000
weighted avg       0.84      0.81      0.82      2000

Confusion Matrix:
[[1326  267]
 [ 119  288]]
Accuracy Score: 0.8615
ROC-AUC Score: 0.8536895909777266
```

```
importances = rf.feature_importances_
features = X_train.columns

# Plot
plt.figure(figsize=(8, 5))
plt.barh(features, importances)
```

```
plt.xlabel("Feature Importance")
plt.title("Top Features Affecting Churn")
plt.gca().invert_yaxis()
plt.show()
```



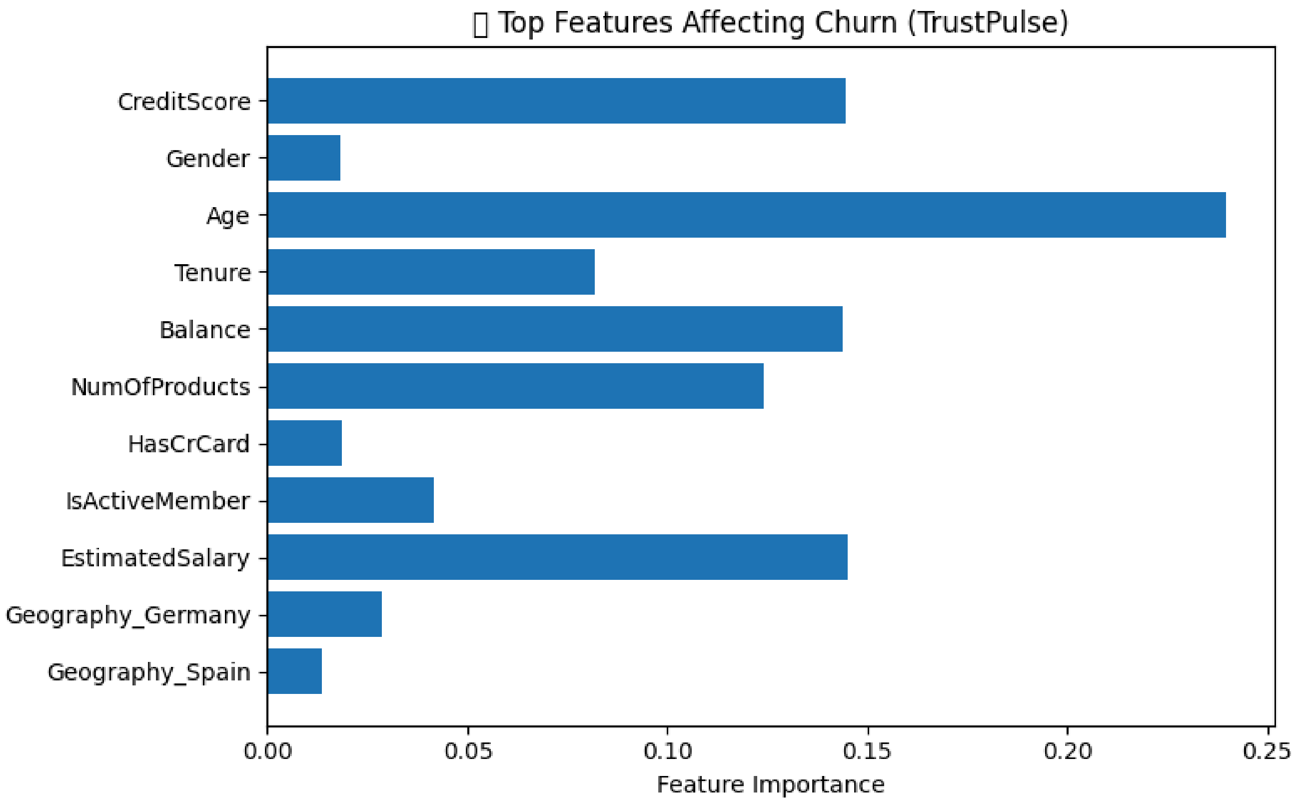

```
{
  "CustomerID": 158923,
  "Churn Probability": 0.78,
  "Trust Score": "Low",
  "Top Factors": ["CreditScore", "IsActiveMember", "Balance"]
}
```

```
{'CustomerID': 158923,
 'Churn Probability': 0.78,
 'Trust Score': 'Low',
 'Top Factors': ['CreditScore', 'IsActiveMember', 'Balance']}
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score
import matplotlib.pyplot as plt
import pandas as pd

def run_trustpulse_module(X_train, X_test, y_train, y_test, threshold=0.25):
    """
    Train RandomForest to predict churn and assign trust scores.

    Args:
        X_train, X_test: Feature data (pandas DataFrames)
        y_train, y_test: Labels (Series)
        threshold: Probability cutoff for assigning churn

    Returns:
        results_df: DataFrame with churn prob and trust score
        model: Trained RandomForestClassifier
    """
    # 1. Train the model
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
```

```
    # 2. Predict probabilities
    y_proba = model.predict_proba(X_test)[:, 1]
    y_pred = (y_proba >= threshold).astype(int)

    # 3. Print classification report
    print("📊 Classification Report:\n", classification_report(y_test, y_pred))
    print("✅ Accuracy Score:", round(accuracy_score(y_test, y_pred), 4))
    print("🏁 ROC-AUC Score:", round(roc_auc_score(y_test, y_proba), 4))
    print("🧮 Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

    # 4. Create output DataFrame with trust labels
    results_df = X_test.copy()
    results_df["Churn_Prob"] = y_proba
    results_df["Trust_Score"] = results_df["Churn_Prob"].apply(
        lambda x: "Low" if x > 0.6 else ("Medium" if x > 0.3 else "High")
    )

    # Optional: include predictions
    results_df["Predicted_Churn"] = y_pred

    # 5. Visualize feature importances
    importances = model.feature_importances_
    features = X_train.columns

    plt.figure(figsize=(8, 5))
    plt.barh(features, importances)
    plt.xlabel("Feature Importance")
    plt.title("🔍 Top Features Affecting Churn (TrustPulse)")
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()

    return results_df, model


results_df, rf_model = run_trustpulse_module(X_train, X_test, y_train, y_test)
results_df.head()
```

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

           0       0.92      0.81      0.86      1593
           1       0.50      0.73      0.59       407

    accuracy                           0.79      2000
   macro avg       0.71      0.77      0.73      2000
weighted avg       0.84      0.79      0.81      2000

✅ Accuracy Score: 0.794
🏁 ROC-AUC Score: 0.8531
🧮 Confusion Matrix:
 [[1289  304]
 [ 108  299]]
```



	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Esti
5702	-0.678012	1	-0.278604	0.687130	-1.225848	0.807737	1	0	
3667	-1.298818	1	-0.564665	-0.350204	0.874084	0.807737	0	0	
1617	-0.967722	0	0.102810	-0.350204	-1.225848	0.807737	0	1	
5673	-0.119286	1	-0.469311	-0.004426	1.008222	0.807737	0	0	
4272	-0.108939	0	-0.469311	-0.695982	0.021491	-0.911583	1	1	

Next steps: Generate code with results_df | View recommended plots | New interactive sheet

```
def summarize_trustpulse_results(results_df, model):
    """
    Summarizes trust score results with useful stats and top risks.
    """

    print("🧠 Trust Score Distribution:")
    print(results_df["Trust_Score"].value_counts())

    print("\n📊 Average Churn Probability:")
    print(round(results_df["Churn_Prob"].mean(), 4))
```

```
    print("\n⚠️ Top 5 High-Risk Customers (Highest Churn Probability):")
    display(results_df.sort_values("Churn_Prob", ascending=False).head(5)[
        ["Churn_Prob", "Trust_Score"]
    ])

    # Optional: Plot histogram of probabilities
    plt.figure(figsize=(7, 4))
    plt.hist(results_df["Churn_Prob"], bins=20, color='orange', edgecolor='black')
    plt.axvline(results_df["Churn_Prob"].mean(), color='red', linestyle='dashed', label='Mean'
    plt.title("Churn Probability Distribution")
    plt.xlabel("Probability")
    plt.ylabel("Customer Count")
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Plot feature importances again if needed
    importances = model.feature_importances_
    features = model.feature_names_in_

    sorted_indices = importances.argsort()[::-1]
    plt.figure(figsize=(7, 5))
    plt.barh(features[sorted_indices], importances[sorted_indices])
    plt.xlabel("Importance")
    plt.title("Top Influential Features (TrustPulse)")
    plt.gca().invert_yaxis()
    plt.tight_layout()
```

```
results_df, rf_model = run_trustpulse_module(X_train, X_test, y_train, y_test)
summarize_trustpulse_results(results_df, rf_model)
```

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

           0       0.92      0.81      0.86      1593
           1       0.50      0.73      0.59       407

    accuracy                           0.79      2000
   macro avg       0.71      0.77      0.73      2000
weighted avg       0.84      0.79      0.81      2000

✅ Accuracy Score: 0.794
🏁 ROC-AUC Score: 0.8531
🧮 Confusion Matrix:
 [[1289  304]
 [ 108  299]]
```