Blood Donor Behavior Analysis & Prediction using UCI Blood Transfusion Dataset

A **Data Analyst** project by

AMIT BHADE

Tools Used – ***Google Colab, Excel***

Techniques – ***Python, SQL, Machine Learning***

OBJECTIVES

To analyze donor behavior patterns using historical donation records (recency, frequency, volume, time since first donation). And to predict the likelihood of donating again.

DATASET

Click here to download the dataset

ABOUT DATASET

The Dataset contains Blood donation history which can be used to predict the donor behaviour.

- **Recency**: Months since last donation
- **Frequency**: Number of donations
- **Monetary**: Total volume donated (ml)
- **Time**: Months since first donation
- **Class**: Target (1 = donated, 0 = did not)

PROJECT SUMMERY

Blood donation is a critical component of healthcare systems worldwide. Predicting whether past donors are likely to donate again it helps blood banks improve outreach, ensure supply, and reduce costs of donor acquisition.

This project analyses the UCI Blood Transfusion dataset to understand donor behavior and build predictive models for donor re-engagement. Unlike full clinical blood datasets, this dataset focuses solely on donation history, not medical health indicators.

PROJECT STEPS:

1. **Problem Definition** – Understand the goal: analyze donor behavior and predict likelihood of future donation.
2. **Data Collection** – Use the UCI Blood Transfusion dataset (blood.csv).
3. **Data Preprocessing** – Check for missing values, duplicates, and clean the dataset.
4. **Exploratory Data Analysis (EDA)** – Visualize distributions, correlations, and trends (Recency, Frequency, Monetary, Time vs Class).

5. **Insights Generation** – Identify key patterns (recent & frequent donors more likely to return).
6. **Modeling (optional DS extension)** – Train ML models (Logistic Regression, Random Forest, etc.) to predict donor behavior.
7. **Evaluation** – Compare models using accuracy, recall, F1, ROC-AUC; tune thresholds.
8. **Recommendations** – Segment donors (Active, Warm, Lapsed, Dormant) and suggest targeted engagement strategies.
9. **Conclusion** – Summarize findings, highlight limitations, and suggest future improvements.

STEP BY STEP PROJECT IMPLEMENTATION

Step 1: Import Libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    roc_curve, precision_recall_curve, confusion_matrix, classification_report
)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC

from IPython.display import display, Markdown
import warnings
warnings.filterwarnings("ignore")

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
```

Step 2: Load Dataset

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

# Load the dataset
df = pd.read_csv('blood.csv')
display(df.head())
print(df.shape)
df.info()
```

Step 3: Data Preprocessing

```
display(Markdown("### Basic Quality Checks"))

# Nulls and duplicates
nulls = df.isna().sum()
dups = df.duplicated().sum()

display(Markdown(f"- **Missing values per column:**\n\n{nulls.to_string()}"))
display(Markdown(f"- **Duplicate rows:** {dups}"))

# Target distribution
class_counts = df['Class'].value_counts().sort_index()
class_ratio = (class_counts / class_counts.sum()).round(3)
display(Markdown("**Target (Class) distribution:**"))
display(pd.DataFrame({'count': class_counts, 'ratio': class_ratio}))
```

Step 4: Exploratory Data Analysis

```
display(Markdown("## Exploratory Data Analysis"))

num_cols = ['Recency', 'Frequency', 'Monetary', 'Time']
target_col = 'Class'

# Histograms
for col in num_cols:
    plt.figure(figsize=(6,4))
    plt.hist(df[col], bins=20)
    plt.title(f"Distribution: {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.show()
```

```
# Boxplots
for col in num_cols:
    plt.figure(figsize=(4,5))
    plt.boxplot(df[col], vert=True)
    plt.title(f"Boxplot: {col}")
    plt.ylabel(col)
    plt.show()

# Correlation matrix (numerical only)
corr = df[num_cols + [target_col]].corr()
plt.figure(figsize=(6,5))
plt.imshow(corr, interpolation='nearest')
plt.xticks(range(corr.shape[1]), corr.columns, rotation=45, ha='right')
plt.yticks(range(corr.shape[0]), corr.index)
plt.title("Correlation Matrix")
plt.colorbar()
plt.tight_layout()
plt.show()

# Simple scatter relationships colored by Class (0/1)
colors = df[target_col].map({0: 'tab:blue', 1: 'tab:orange'})
pairs = [('Recency','Frequency'), ('Recency','Monetary'), ('Frequency','Time')]
for x, y in pairs:
    plt.figure(figsize=(6,4))
    plt.scatter(df[x], df[y], c=colors)
    plt.xlabel(x); plt.ylabel(y)
    plt.title(f"{x} vs {y} (colored by Class)")
    plt.show()

# Brief numeric summary by target
summary = df.groupby(target_col)[num_cols].agg(['mean', 'median', 'std'])
display(Markdown("**Feature summary grouped by Class:**"))
display(summary)
```

Step 5: Train / Test Split

```
X = df.drop(columns=[target_col])
y = df[target_col]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=RANDOM_STATE
)

X_train.shape, X_test.shape, y_train.mean(), y_test.mean()
```

Step 6: Baseline Models & Cross-Validation

```
display(Markdown("## Baseline Model Comparison (5-fold CV ROC-AUC on training
set)"))

models = {
    "LogisticRegression": Pipeline([
        ("scaler", StandardScaler()),
        ("clf", LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=RANDOM_STATE))
    ]),
    "RandomForest": Pipeline([
        ("clf", RandomForestClassifier(
            n_estimators=300, max_depth=None, min_samples_leaf=1,
            class_weight='balanced', random_state=RANDOM_STATE
        ))
    ]),
    "GradientBoosting": Pipeline([
        ("clf", GradientBoostingClassifier(random_state=RANDOM_STATE))
    ]),
    "SVC": Pipeline([
        ("scaler", StandardScaler()),
        ("clf", SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=RANDOM_STATE))
    ]),
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
cv_results = []

for name, pipe in models.items():
    auc_scores = cross_val_score(pipe, X_train, y_train, cv=cv, scoring='roc_auc')
    cv_results.append({
        "model": name,
        "cv_auc_mean": np.mean(auc_scores),
        "cv_auc_std": np.std(auc_scores),
    })

cv_df = pd.DataFrame(cv_results).sort_values(by="cv_auc_mean",
ascending=False).reset_index(drop=True)
display(cv_df)
```

Step 7: Fit All, Evaluate on Test, Pick Best

```
display(Markdown("## Test Set Performance (default threshold = 0.5)"))

def evaluate_at_threshold(y_true, y_proba, thr=0.5, positive_label=1):
    y_pred = (y_proba >= thr).astype(int)
    return {
        "accuracy": accuracy_score(y_true, y_pred),
        "precision": precision_score(y_true, y_pred, pos_label=positive_label,
zero_division=0),
        "recall": recall_score(y_true, y_pred, pos_label=positive_label, zero_division=0),
        "f1": f1_score(y_true, y_pred, pos_label=positive_label, zero_division=0),
        "roc_auc": roc_auc_score(y_true, y_proba),
        "threshold": thr
    }

test_results = []
fitted_models = {}

for name, pipe in models.items():
    pipe.fit(X_train, y_train)
    fitted_models[name] = pipe
    y_proba = pipe.predict_proba(X_test)[:, 1]
    metrics = evaluate_at_threshold(y_test, y_proba, thr=0.5)
    metrics["model"] = name
    test_results.append(metrics)

test_df = pd.DataFrame(test_results).sort_values(by="roc_auc",
ascending=False).reset_index(drop=True)
display(test_df)

best_model_name = test_df.iloc[0]["model"]
best_model = fitted_models[best_model_name]
display(Markdown(f"**Selected best model:** `{best_model_name}` (by ROC-AUC on
test)"))
```

Step 8: Threshold Tuning (Youden's J)

```
display(Markdown("## Threshold Tuning (maximize Youden's J = TPR - FPR)"))

y_proba_best = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba_best)

youden_j = tpr - fpr
best_idx = np.argmax(youden_j)
best_thr = thresholds[best_idx]

tuned_metrics = evaluate_at_threshold(y_test, y_proba_best, thr=best_thr)
```

```
display(Markdown(f"**Optimal threshold:** {best_thr:.3f}"))
display(pd.DataFrame([tuned_metrics]))

# ROC Curve
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label="ROC")
plt.plot([0,1], [0,1], linestyle="--")
plt.title("ROC Curve (Best Model)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

# Precision-Recall Curve
prec, rec, pr_thr = precision_recall_curve(y_test, y_proba_best)
plt.figure(figsize=(6,5))
plt.plot(rec, prec)
plt.title("Precision-Recall Curve (Best Model)")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.show()
```

Step 9: Confusion Matrices (Default vs Tuned)

```
def plot_conf_mat(y_true, y_proba, thr, title):
    y_pred = (y_proba >= thr).astype(int)
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4,4))
    plt.imshow(cm, interpolation='nearest')
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.xticks([0,1],[0,1])
    plt.yticks([0,1],[0,1])
    for (i, j), v in np.ndenumerate(cm):
        plt.text(j, i, str(v), ha='center', va='center')
    plt.colorbar()
    plt.tight_layout()
    plt.show()

plot_conf_mat(y_test, y_proba_best, thr=0.5, title="Confusion Matrix (Default thr = 0.5)")
plot_conf_mat(y_test, y_proba_best, thr=best_thr, title=f"Confusion Matrix (Tuned thr =
{best_thr:.3f})")

print("Classification report (tuned threshold):")
print(classification_report(y_test, (y_proba_best >= best_thr).astype(int),
zero_division=0))
```

Step 10: Model Explainability: Feature Importance

```
display(Markdown("## Feature Importance / Coefficients"))

feature_names = X.columns.tolist()

if best_model_name == "RandomForest":
    rf = best_model.named_steps['clf'] if 'clf' in best_model.named_steps else best_model
    importances = rf.feature_importances_
    fi = pd.DataFrame({"feature": feature_names, "importance":
importances}).sort_values("importance", ascending=False)
    display(fi)
    plt.figure(figsize=(6,4))
    plt.bar(fi["feature"], fi["importance"])
    plt.title("RandomForest Feature Importances")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

elif best_model_name == "GradientBoosting":
    gb = best_model.named_steps['clf'] if 'clf' in best_model.named_steps else best_model
    importances = gb.feature_importances_
    fi = pd.DataFrame({"feature": feature_names, "importance":
importances}).sort_values("importance", ascending=False)
    display(fi)
    plt.figure(figsize=(6,4))
    plt.bar(fi["feature"], fi["importance"])
    plt.title("GradientBoosting Feature Importances")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

elif best_model_name == "LogisticRegression":
    # Extract coefficients after scaling
    lr = best_model.named_steps['clf']
    scaler = best_model.named_steps['scaler']
    coefs = lr.coef_.flatten()
    coef_df = pd.DataFrame({"feature": feature_names, "coef": coefs}).sort_values("coef",
ascending=False)
    display(coef_df)
    plt.figure(figsize=(6,4))
    plt.bar(coef_df["feature"], coef_df["coef"])
    plt.title("Logistic Regression Coefficients")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

elif best_model_name == "SVC":
    display(Markdown("_SVC with RBF kernel is not inherently interpretable via feature
importances._"))
```

```
else:
    display(Markdown("_No supported importance for this model._"))
```

Step 11: Donor Segmentation & Recommendations

```
display(Markdown("## Donor Segmentation & Recommendations"))

def donor_segment(row):
    # Simple rules: tweak as needed
    if row['Recency'] <= 2 and row['Frequency'] >= 10:
        return "Active-Regular"
    if row['Recency'] <= 3 and row['Frequency'] >= 5:
        return "Active"
    if row['Recency'] <= 6:
        return "Warm"
    if row['Recency'] <= 12:
        return "Lapsed"
    return "Dormant"

def generate_recommendation(prob, segment):
    # prob = predicted probability of future donation
    if prob >= 0.70:
        return "High likelihood — send thank-you + schedule next appointment"
    if prob >= 0.50:
        return "Moderate likelihood — reminder with easy booking link"
    if prob >= 0.30:
        return "Low-moderate — re-engagement campaign + emphasize impact"
    return "Low — low-priority nurture sequence"

# Apply to test set
test_with_scores = X_test.copy()
test_with_scores['proba'] = y_proba_best
test_with_scores['segment'] = test_with_scores.apply(donor_segment, axis=1)
test_with_scores['recommendation'] = [
    generate_recommendation(p, s) for p, s in zip(test_with_scores['proba'],
test_with_scores['segment'])
]

display(test_with_scores.head(10))
```

Step12: Save Best Model

```
import joblib

model_artifact = f"best_model_{best_model_name}.joblib"
joblib.dump(best_model, model_artifact)
print(f"Saved: {model_artifact}")

# Example: predict for a new donor
example = pd.DataFrame([{
    "Recency": 2,
    "Frequency": 12,
    "Monetary": 3000,
    "Time": 48
}])
pred_prob = best_model.predict_proba(example)[:,1][0]
print(f"Predicted probability of future donation: {pred_prob:.3f}")
```

KEY INSIGHTS

- Most donors did not donate again (class imbalance: ~76% non-donors vs 24% donors).
- Recency is the strongest factor — recent donors are much more likely to donate again.
- Frequency (past donations) is highly correlated with total blood donated (Monetary) and predicts future donations.
- Time since first donation has less impact; recent activity matters more than donor history length.
- Donors can be segmented as Active, Warm, Lapsed, Dormant for targeted engagement.

CONCLUSION

What we did

- Explored the UCI Blood Transfusion dataset (donation behavior features only).
- Built and compared several models (Logistic Regression, Random Forest, Gradient Boosting, SVC) using stratified CV.
- Selected the best model by ROC-AUC on a held-out test set and tuned the classification threshold using Youden's J.
- Produced confusion matrices, ROC/PR curves, and feature importance (when available).
- Created a simple donor segmentation + recommendation layer to translate predictions into actions.

Key insights (typical patterns which are likely to observe)

- Frequency and Recency are strong indicators of future donation.
- A tuned threshold often improves recall of likely donors (useful if the goal is to prioritize outreach).
- Class imbalance exists but is modest; using class weights helps.

Best model & performance

- See the table in “Test Set Performance” for exact numbers (varies slightly run-to-run).
- We chose the model with the highest ROC-AUC, then tuned the threshold for a better balance between TPR and FPR.

Limitations

- Dataset lacks clinical/lab features (no hemoglobin, cholesterol, etc.); we cannot make medical recommendations.
- Feature space is small; limited room for feature engineering.

Future Work

- Add donor demographics (age, location), campaign history (channels, responses), and appointment availability.
- Use cost-sensitive evaluation (e.g., cost of contacting a donor vs. value of a successful donation) to pick an operating threshold.
- Explore calibration (Platt/Isotonic) for better probability quality.

Business Impact

- The model can help a blood bank rank donors by probability of donating again.
- Pair the probability with segment-driven messaging to improve conversions:
- Active-Regular → express gratitude & auto-schedule
- Warm → quick reminder + easy booking link
- Lapsed → personalized re-engagement (impact stories, incentives)
- Dormant → low-priority nurture

REFERENCE