# *CRC_PROJECT*

**Vibhanshu Singh**

**Methodologies –**

1) Data Preprocessing
    - Transformation
    - Cut off
    - Cho1 Formula
    - Prevalence calculation
2) Feature selection
    - Variance Threshold (Filter Method)
    - Univariate Feature Selection using ANOVA F-test (Filter Method)
    - Mutual Information (Filter Method)
    - Recursive Feature Elimination (Wrapper Method) with Logistic Regression (Optimized)
    - Feature Importance from Random Forest (Embedded Method)
    - L1 Regularization (Lasso) for Feature Selection (Embedded Method)
3) Hyperparameter Tuning
    - RandomForest
    - SVM
    - LogisticRegression
4) Model building
    - RandomForest
    - SVM
    - LogisticRegression
5) Model evaluation
    - Learning curve
    - Accuracy
    - Precision
    - Recall
    - F1

# CHAPTER 1 - Data Preprocessing

## 1.1- CLR Transformation

```
import pandas as pd
import numpy as np

# Assuming your dataframe is df_filtered (species x samples)

# Step 1: Add pseudocount to avoid log(0)
df_pseudo = species_selected + 1

# Step 2: Compute geometric mean for each species (row)
# CLR is log(value / geometric mean of row)
geometric_mean = np.exp(np.log(df_pseudo).mean(axis=1))  # row-wise geometric mean

# Step 3: Apply CLR transformation
df_clr = np.log(df_pseudo.div(geometric_mean, axis=0))

print("CLR-transformed table:")
print(df_clr)
```

df_pseudo = species_selected + 1

In abundance tables have **zero counts**.

But **log(0) is undefined**, so before log, add a **pseudocount (1)** to avoid math errors.

compute the **row-wise geometric mean** for each species, and then compute the **Centered Log-Ratio** transform:

$$\mathrm{CLR}(x_{ij}) = \log\left(\frac{x_{ij}}{g_i}\right)$$

where $x_{ij}$is species $i$in sample $j$, and $g_i$is the geometric mean of row $i$. CLR maps compositional data into a real space where standard statistics make sense and each row has mean zero.

**geometric_mean = np.exp(np.log(df_pseudo).mean(axis=1))**

- np.log(df_pseudo) applies natural log elementwise.
- .mean(axis=1) computes the **mean of the logs across columns** (samples), separately for each row (species).
- np.exp(...) exponentiates back to produce the geometric mean.

```
df_clr = np.log(df_pseudo.div(geometric_mean, axis=0))
```

- df_pseudo.div(geometric_mean, axis=0) divides each row by that row's geometric mean.
- Then np.log(...) computes the natural log of the ratio.

**Significance-** with the help of CLR transformation we can normalise and transform the data together.

## 1.2 – Cut off

```
# Find samples with total reads < 100k(1 million)
low_read_samples = sample_sums[sample_sums < 100000].index.tolist()
print(f"Number of low-read samples (<100k): {len(low_read_samples)}")

df_filtered = df_out.drop(columns=low_read_samples)
df_filtered
```

**Significance**- we can remove less important features

## 1.3 – Prevalence Calculation

```
# Select species that are present in more than 90% of samples
import pandas as pd
# Step 1: Convert counts to presence/absence (1 if count > 0, else 0)
presence_absence = (brack_df > 0).astype(int)#Creates a Boolean DataFrame (True if value > 0, False otherwise). Converts True → 1 and False → 0

# Step 2: Calculate prevalence (proportion of samples where species is present)
prevalence = presence_absence.mean(axis=1)

# Step 3: Filter species with prevalence
threshold = 0.9
selected_species = prevalence[prevalence > threshold].index

# Step 4: Filter the original DataFrame to include only selected species
filtered_df = brack_df.loc[selected_species]

# Step 5: Output the filtered table
print(f"Filtered table for species with prevalence > {threshold}:")
print(filtered_df)
```

## 1.4 – Cho1 Formula

**Alpha diversity = diversity within a single sample (richness + evenness).**

**Beta diversity = diversity between samples (community composition differences).**

### Chao1 formula

computing the Chao1 index, which is a widely used alpha diversity metric in ecology and metagenomics. It estimates the true species richness of a community by correcting for unobserved (rare) species using the number of singletons and doubletons.

F1 → number of species that appear only once (singletons).

F2 → number of species that appear exactly twice (doubletons).

The **Chao1 formula** is:

$$\text{Chao1} = S_{\text{obs}} + \frac{F1^2}{2F2}$$

- If there are no doubletons (`F2=0`), it uses a modified correction formula:

$$S_{\text{obs}} + \frac{F1(F1-1)}{2}$$

```
import pandas as pd
import numpy as np

def chao1(counts):
    """Compute Chao1 index for one sample (counts = array of species counts)."""
    counts = np.array(counts)
    S_obs = np.sum(counts > 0)       # observed species richness
    F1 = np.sum(counts == 1)         # singletons
    F2 = np.sum(counts == 2)         # doubletons

    if F2 == 0:  # avoid division by zero
        return S_obs + (F1 * (F1 - 1)) / 2 if F1 > 1 else S_obs
    else:
        return S_obs + (F1**2) / (2 * F2)

# Apply Chao1 to each sample (column)
chao1_values = brack_df.apply(chao1, axis=0)

# Put into a DataFrame
chao1_df = pd.DataFrame({"Sample": chao1_values.index, "Chao1": chao1_values.values})

# Show first results

chao1_df
```

# CHAPTER 2 – Feature Selection

## 2.1 – various feature selection techniques

### 2.1.1- Variance Threshold ( Filter Method)

```
# 1. Variance Threshold (Filter Method)
selector_var = VarianceThreshold(threshold=0.01)
X_var = selector_var.fit_transform(X)
# Fit: Computes the variance of each feature in X. & Transform: Removes the low-variance features and returns a reduced dataset X_var.
#Result: A NumPy array with only the features that passed the variance threshold.
selected_features_var = X.columns[selector_var.get_support()] # returns a Boolean mask array indicating which features were selected ( those with variance > 0.01).
# X.columns[...] applies that mask to the column names of the original DataFrame X.
X_var_df = pd.DataFrame(X_var, columns=selected_features_var)
print(f"Selected features with variance > 0.01: {len(selected_features_var)}")
```

## 📊 What is Variance?

Variance measures how much a feature's values **vary** (spread out) across the dataset.

- Formula for variance of a feature:

$$\text{Variance} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

where:

  - $x_i$ = individual sample value of the feature
  - $\bar{x}$ = mean of the feature across all samples
  - $n$ = number of samples

---

## ⚡ Interpretation of the Threshold

- **If Variance ≤ 0.01:**
  - The feature's values hardly change across samples.
  - Example: A feature column where almost every value is constant or varies very slightly (e.g., [0, 0, 0, 0, 0.01]).
  - Such a feature provides **very little information** to help a machine learning model distinguish between samples.

## 2.1.2- Univariate Feature Selection using ANOVA F-test (Filter Method)

```
# 2. Univariate Feature Selection using ANOVA F-test (Filter Method)
selector_anova = SelectKBest(f_classif, k=100)
X_anova = selector_anova.fit_transform(X, y)

# `f_classif` is the scoring function based on ANOVA F-test for classification.
# It measures the linear dependency between each feature and the target variable y.
# For each feature, it computes an F-statistic and a p-value.
# Higher F-statistic → more relevant feature.
# k=100 means: Keep the top 100 features that have the highest F-scores.

selected_features_anova = X.columns[selector_anova.get_support()]
X_anova_df = pd.DataFrame(X_anova, columns=selected_features_anova)
print(f"Selected top 100 features using ANOVA: {selected_features_anova}")
```

### ✅ What is ANOVA?

👉 **ANOVA** stands for **Analysis of Variance**.

It is a statistical method used to determine if there are **significant differences between the means of two or more groups**.

---

### 🧱 How Does ANOVA Work?

- Imagine we have a dataset where:
  - **X** = multiple features (numerical values).
  - **y** = a categorical target (class labels).

**Example:**

| Sample | Feature X | Class (y) |
|---|---|---|
| 1 | 5.2 | A |
| 2 | 4.8 | A |
| 3 | 6.1 | B |
| 4 | 5.9 | B |
| 5 | 7.0 | C |
| 6 | 6.8 | C |

👉 ANOVA checks:

- Is the **mean of Feature X significantly different across the classes (A, B, C)?**

---

### ⚡ The Key Idea

- It compares **within-group variance** vs. **between-group variance**.

$$F = \frac{\text{Variance between groups}}{\text{Variance within groups}}$$

- **High F-value** → Strong evidence that at least one group's mean is different.

## 2.1.3- *Mutual Information (Filter Method)*

```
# 3. Mutual Information (Filter Method)
selector_mi = SelectKBest(mutual_info_classif, k=100)
#  "SelectKBest" is a feature selection tool from sklearn that keeps only the top k features.
# "mutual_info_classif" is the Mutual Information scoring function used for classification problems.

X_mi = selector_mi.fit_transform(X, y)
selected_features_mi = X.columns[selector_mi.get_support()] #Retrieves the names of the selected features by applying the boolean mask from .get_support().
X_mi_df = pd.DataFrame(X_mi, columns=selected_features_mi)
print(f"Selected top 100 features using Mutual Information: {selected_features_mi}")
```

1. Mutual Information measures the dependency between two variables.
2. In this method feature selection, it quantifies how much information a feature provides about the target variable.
3. Unlike correlation, it can capture non-linear relationships.

**Mutual Information (MI)** measures **how much information a feature gives about the target (class label).**
In simple words:

> **MI tells us how much knowing `X` helps to predict `Y`.**

- If a feature and target are **independent**, MI = **0** (feature is useless).
- If a feature is **strongly related** to the target, MI is **high** (feature is useful).

---

### ✅ Mathematical Intuition (Simple Form)

Mutual Information is defined as:

$$MI(X,Y) = H(Y) - H(Y|X)$$

Where:

- $H(Y)$ = Uncertainty in target **before** knowing `X`
- $H(Y|X)$ = Uncertainty in target **after** knowing `X`

So,

→ If knowing `X` **reduces uncertainty** in `Y`, then $H(Y|X)$ becomes small → **high MI**

→ If knowing `X` **does not help**, then $H(Y|X) \approx H(Y)$ → **MI ≈ 0**

### 2.1.4- *Recursive Feature Elimination (Wrapper Method) with Logistic Regression*

```
# Recursive Feature Elimination (RFE) using Logistic Regression as the base model, selecting the top 50 most important features from dataset X in relation to the target variable y.
model_lr = LogisticRegression(solver='liblinear', max_iter=500, random_state=42)  # Faster solver and reduced max_iter
# max_iter=500: Sets the maximum number of iterations for convergence.
selector_rfe = RFE(model_lr, n_features_to_select=50, step=0.1)  # Step=0.1 removes 10% of features per iteration
X_rfe = selector_rfe.fit_transform(X, y)
selected_features_rfe = X.columns[selector_rfe.get_support()]
X_rfe_df = pd.DataFrame(X_rfe, columns=selected_features_rfe)
print(f"Selected top 50 features using RFE: {selected_features_rfe}")

# Fit Logistic Regression on the selected features to get coefficients
model_lr_fitted = LogisticRegression(solver='liblinear', max_iter=500, random_state=42)
model_lr_fitted.fit(X_rfe, y)
coefficients = np.abs(model_lr_fitted.coef_[0])  # Absolute values for importance

# Extracts the learned coefficients from the model.
# model_lr_fitted.coef_  returns a 2D array (since scikit-learn expects multi-class by default).
# [0] selects the coefficients for the binary classification case.
# Takes the absolute value of the coefficients:
# Why? Because in feature importance, we often care about the magnitude of the effect, not the direction (positive or negative).
```

### What RFE is doing (in one sentence)

RFE repeatedly trains a model, ranks the features by importance, removes the weakest ones, and repeats this until only the desired number of features remains.

---

### How RFE Works — Step-by-Step Logic

Assume you start with 100 features and want 50.

| Step | What happens? | Result |
|---|---|---|
| 1. Train | Train Logistic Regression on all features | Model learns coefficients (importance scores) |
| 2. Rank features | Sort features by importance (absolute coefficient value) | Weakest features identified |
| 3. Remove features | Remove bottom 10% (because step = 0.1), which removes the 10 worst features | 90 features remain |
| 4. Repeat | Train again, rank again, remove 10% | Feature count keeps shrinking |
| 5. Stop | Stop when only 50 features are left | These are the "best" features |

Key idea: After every removal, the model is retrained because feature importance changes when some features are removed.

---

### Why is this "Recursive"?

Because it repeats the same cycle (Train → Rank → Remove → Train → Rank → Remove ...) until the target number of features is reached.

---

### Why Logistic Regression works for RFE

- Logistic Regression gives a coefficient (weight) for every feature.
- Large magnitude means more influence (important feature).
- Small magnitude means weak influence (bad feature).
- Therefore, RFE removes the smallest-weight features first.

---

### Short Summary

Train → Rank → Remove → Repeat → Stop (until the desired number of features remains).

---

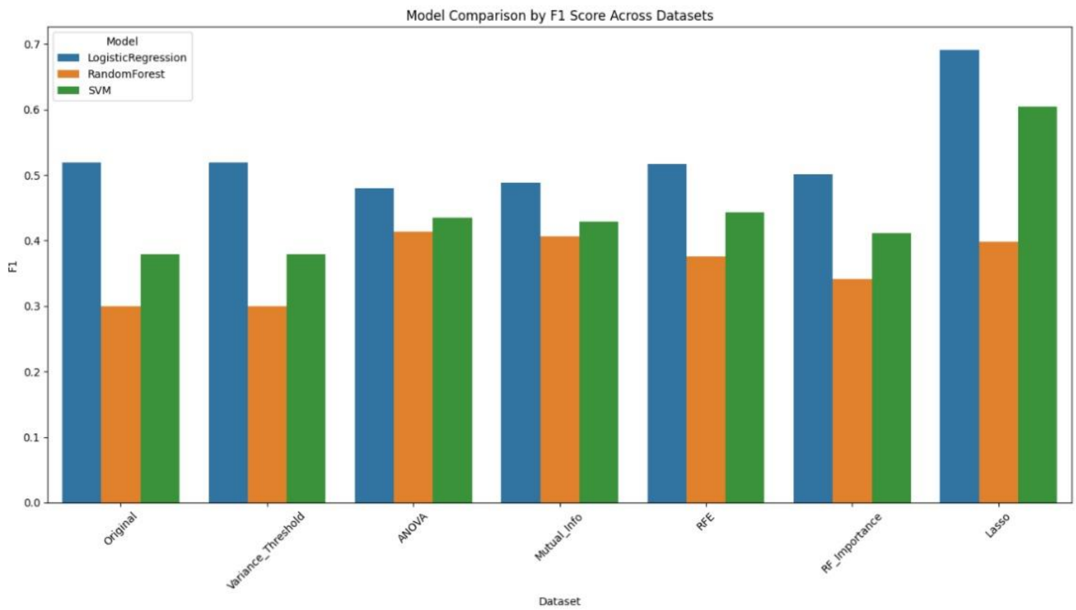
## 2.1.5- *Feature Importance from Random Forest (Embedded Method)*

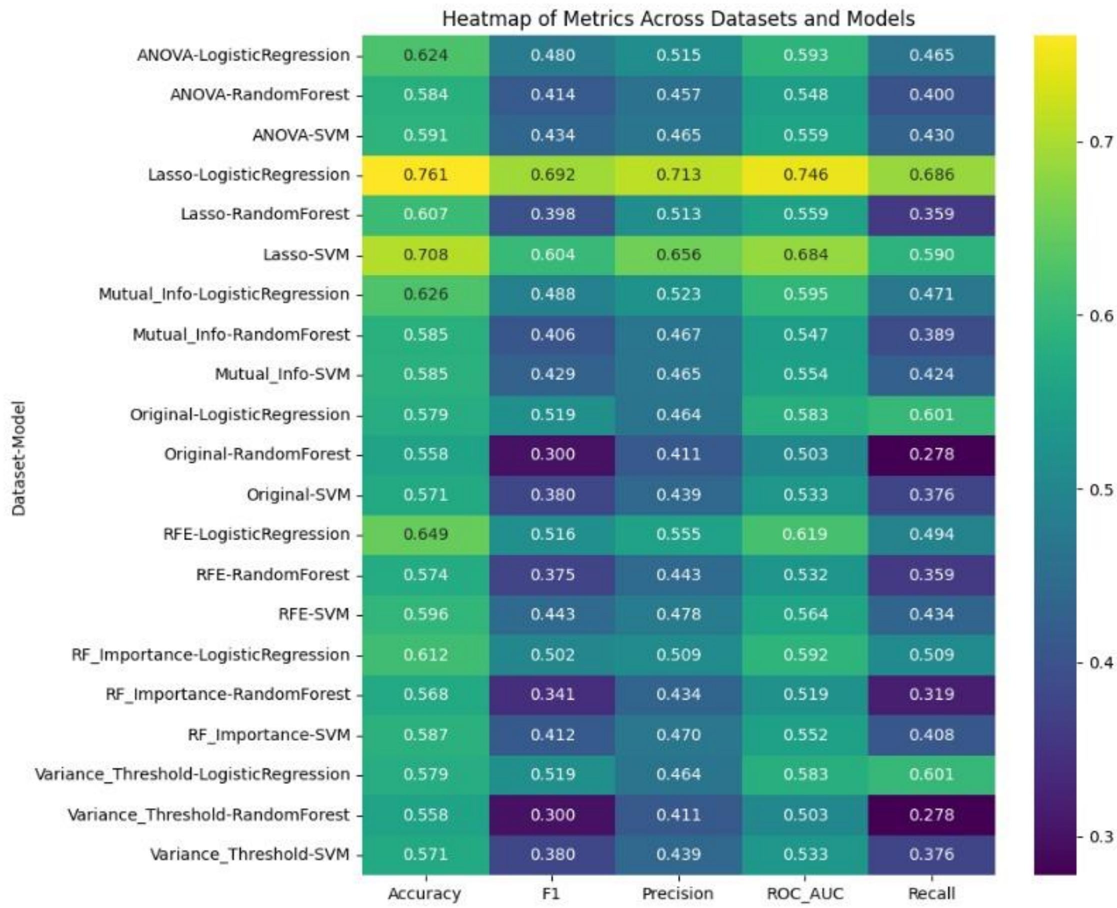
```
# 5. Feature Importance from Random Forest (Embedded Method)
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X, y)
selector_rf = SelectFromModel(model_rf, prefit=True, threshold="mean")

# SelectFromModel: A meta-transformer that selects features based on feature importance provided by an estimator
(here, the random forest).
# model_rf: The pre-trained Random Forest model is already fitted, so prefit=True.
# threshold="mean": Select features whose importance is above the mean importance.

X_rf = selector_rf.transform(X) # Apply the selector to X to retain only the most important features.
selected_features_rf = X.columns[selector_rf.get_support()]
X_rf_df = pd.DataFrame(X_rf, columns=selected_features_rf)
print(f"Selected features using RF importance (threshold=mean): {len(selected_features_rf)}")
```

### 2.1.6- *L1 Regularization (Lasso) for Feature Selection (Embedded Method)*

```
lasso = LassoCV(cv=5, random_state=42)
lasso.fit(X, y)
selected_features_lasso = X.columns[lasso.coef_ != 0]
# lasso.coef_: Array of feature coefficients after training.
# We filter the columns where the coefficient is non-zero (important features).
# The result is a list of selected important feature names that have predictive power
X_lasso = X[selected_features_lasso]
```

# RESULTS





Heatmap of Metrics Across Datasets and Models
ANOVA-LogisticRegression
ANOVA-RandomForest
ANOVA-SVM
Lasso-LogisticRegression
Lasso-RandomForest
Lasso-SVM
Mutual_Info-LogisticRegression
Mutual_Info-RandomForest
Mutual_Info-SVM
Original-LogisticRegression
Original-RandomForest
Original-SVM
RFE-LogisticRegression
RFE-RandomForest
RFE-SVM
RF_Importance-LogisticRegression
RF_Importance-RandomForest
RF_Importance-SVM
Variance_Threshold-LogisticRegression
Variance_Threshold-RandomForest
Variance_Threshold-SVM
Dataset-Model
0.624 0.480 0.515 0.593 0.465
0.584 0.414 0.457 0.548 0.400
0.591 0.434 0.465 0.559 0.430
0.761 0.692 0.713 0.746 0.686
0.607 0.398 0.513 0.559 0.359
0.708 0.604 0.656 0.684 0.590
0.626 0.488 0.523 0.595 0.471
0.585 0.406 0.467 0.547 0.389
0.585 0.429 0.465 0.554 0.424
0.579 0.519 0.464 0.583 0.601
0.558 0.300 0.411 0.503 0.278
0.571 0.380 0.439 0.533 0.376
0.649 0.516 0.555 0.619 0.494
0.574 0.375 0.443 0.532 0.359
0.596 0.443 0.478 0.564 0.434
0.612 0.502 0.509 0.592 0.509
0.568 0.341 0.434 0.519 0.319
0.587 0.412 0.470 0.552 0.408
0.579 0.519 0.464 0.583 0.601
0.558 0.300 0.411 0.503 0.278
0.571 0.380 0.439 0.533 0.376
Accuracy
F1
Precision
ROC_AUC
Recall
0.7
0.6
0.5
0.4
0.3

# CHAPTER 3 - Hyperparameter Tuning

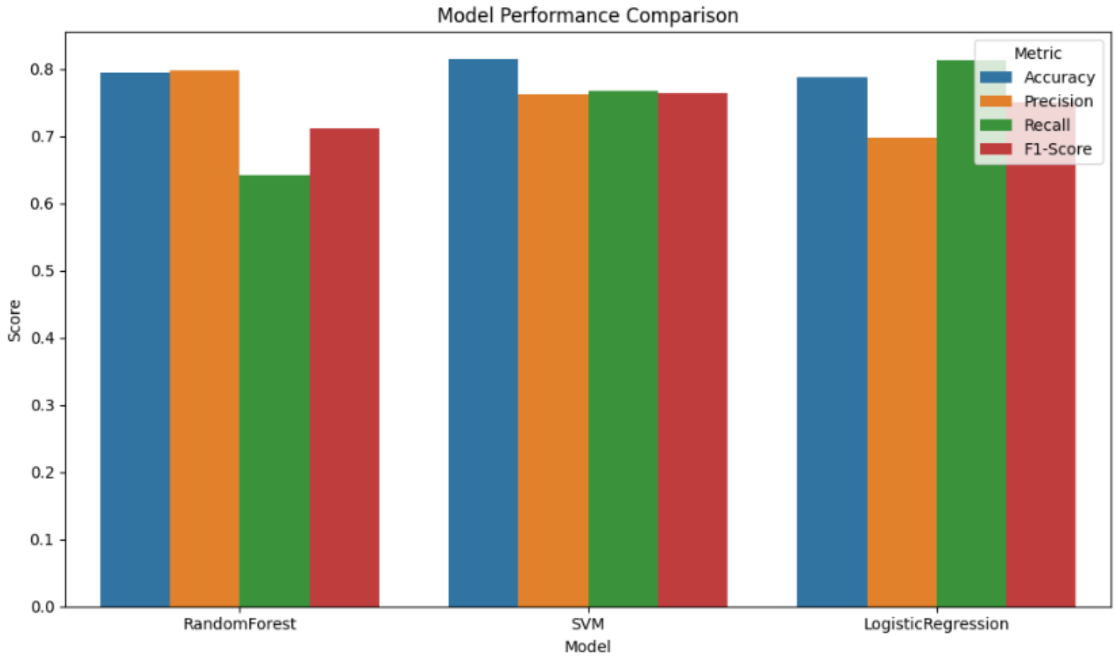
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.feature_selection import SelectKBest, f_classif, RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
# Step 2: 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
) #stratify=y - Ensures the train/test split maintains the same class distribution as
# Step 3: Model Building and Hyperparameter Tuning
models = {
  'RandomForest': {
    'model': RandomForestClassifier(random_state=42),
    'params': {
      'n_estimators': [50, 100, 200],
      'max_depth': [None, 10, 20],
      'min_samples_split': [2, 5, 10]
    }
  },
  'SVM': {
    'model': SVC(probability=False, random_state=42),
    'params': {
      'C': [0.1, 1, 10],
      'kernel': ['linear', 'rbf'],
      'gamma': ['scale', 'auto']
    }
  },
  'LogisticRegression': {
    'model': LogisticRegression(max_iter=1000, random_state=42),
    'params': {
      'C': [0.1, 1, 10],
      'solver': ['liblinear', 'lbfgs']
    }
  }

  }
# Dictionary to store results
results = {}
# Stratified 5-fold CV
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)# save this result
```

```
# Train and tune each model
for name, config in models.items():
    print(f"\n Tuning {name}...")
    grid_search = GridSearchCV(
        estimator=config['model'],
        param_grid=config['params'],
        cv=cv,
        scoring='f1',
        n_jobs=-1,
        verbose=1
    )
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    print(f" Best parameters for {name}: {grid_search.best_params_}")

    # Evaluate on test set
    y_pred = best_model.predict(X_test)
    y_prob = best_model.predict_proba(X_test)[:, 1] if hasattr(best_model, 'predict_proba') else None

    results[name] = {
        'best_params': grid_search.best_params_,
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred),
        'f1': f1_score(y_test, y_pred),
        'roc_auc': roc_auc_score(y_test, y_prob) if y_prob is not None else None,
        'classification_report': classification_report(y_test, y_pred)
    }
# Step 4: Evaluation and Comparison
# Step 5: Model Comparison DataFrame
# visualize performance
```

# RESULTS

Model Comparison:

| | Model | Accuracy | Precision | Recall | F1-Score | ROC-AUC |
|---|---|---|---|---|---|---|
| 0 | RandomForest | 0.794934 | 0.797710 | 0.641104 | 0.710884 | 0.894654 |
| 1 | SVM | 0.814234 | 0.762195 | 0.766871 | 0.764526 | N/A |
| 2 | LogisticRegression | 0.787696 | 0.697368 | 0.812883 | 0.750708 | 0.868458 |



# CHAPTER 4 – Model Building

(after doing all experiment once again with selected parameters and methods )

ORIGINAL DATASET-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')

# --- Step 1: Prepare X and y ---
# Get feature names before converting to NumPy array
feature_names = ml_df.drop(columns=['class_label']).columns
X = ml_df.drop(columns=['class_label']).values  # features
y = ml_df['class_label'].values              # labels

# --- Step 2: Split into training and test sets (80/20) ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

X_train_scaled = X_train
X_test_scaled = X_test
```

```
# --- Step 4: Define multiple models ---
models = {
    'Logistic Regression': LogisticRegression(
        random_state=42,
        max_iter=1000,
        penalty='l2',
        solver='lbfgs',
        C=1.0
    ),

    'Random Forest': RandomForestClassifier(
        random_state=42,
        n_estimators=300,
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=1,
        max_features='sqrt',
        bootstrap=True,
        n_jobs=-1
    ),

    'Support Vector Machine': SVC(
        kernel='rbf',
        random_state=42,
        C=1.0,
        gamma='scale',
        probability=True
    )
}
# --- Step 5: Train, predict, and evaluate each model ---
results = {}
conf_mats = {}
accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
    conf_mats[name] = confusion_matrix(y_test, y_pred)

    print(f"\n--- {name} ---")
    print(f"Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

# --- Step 6: Compare models by accuracy ---
print("\n--- Model Comparison ---")
comparison_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])
comparison_df = comparison_df.sort_values(by='Accuracy', ascending=False)
print(comparison_df)
```

# RESULTS

```
--- Logistic Regression ---
Accuracy: 0.8022

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.81      0.83       503
           1       0.73      0.79      0.76       326

    accuracy                           0.80       829
   macro avg       0.79      0.80      0.80       829
weighted avg       0.81      0.80      0.80       829


Confusion Matrix:
[[407  96]
 [ 68 258]]

--- Random Forest ---
Accuracy: 0.7973

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       503
           1       0.81      0.64      0.71       326

    accuracy                           0.80       829
   macro avg       0.80      0.77      0.78       829
weighted avg       0.80      0.80      0.79       829


Confusion Matrix:
[[453  50]
 [118 208]]

--- Support Vector Machine ---
Accuracy: 0.8034

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       503
           1       0.77      0.71      0.74       326

    accuracy                           0.80       829
   macro avg       0.80      0.79      0.79       829
weighted avg       0.80      0.80      0.80       829


Confusion Matrix:
[[434  69]
 [ 94 232]]

--- Model Comparison ---
                    Model  Accuracy
2  Support Vector Machine  0.803378
0     Logistic Regression  0.802171
1           Random Forest  0.797346
```

# Feature Selection via L1 Regularization (Lasso)

```
# --- Step 3: Feature Selection via L1 Regularization (Lasso) Methods ---

from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler # Import StandardScaler
import numpy as np
import pandas as pd

# --- Assume X_train, X_test, y_train, y_test are already defined ---

# If X_train is a pandas DataFrame, extract feature names
if isinstance(X_train, pd.DataFrame):
    feature_names = X_train.columns
else:
    feature_names = [f"feature_{i}" for i in range(X_train.shape[1])]
#  Logistic Regression with L1 penalty
lasso_estimator = LogisticRegression(
    penalty='l1',
    solver='liblinear',
    random_state=42,
    C=0.1 # Reduced C for more regularization
)
#  Use SelectFromModel for feature selection (faster than RFE)
selector = SelectFromModel(estimator=lasso_estimator)
# Fit the selector on the scaled training data
selector.fit(X_train_scaled, y_train)
#  Transform training and test data to selected features
X_train_lasso = selector.transform(X_train_scaled) # Use scaled data for transformation
X_test_lasso = selector.transform(X_test_scaled) # Use scaled data for transformation
#  Extract selected feature names
selected_features_lasso = np.array(feature_names)[selector.get_support()].tolist()
#  Print output
print("Selected features (L1/Lasso):")
for f in selected_features_lasso:
    print("-", f)
print(f"\nTotal selected features: {len(selected_features_lasso)} / {len(feature_names)}")
```

```
X_train_selected = X_train_lasso
X_test_selected = X_test_lasso
selected_features = selected_features_lasso

# --- Step 4: Model Building on Selected Features ---

models = {
    'Logistic Regression': LogisticRegression(random_state=42),
    'SVM': SVC(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42)
}

trained_models = {}
predictions = {}

for name, model in models.items():
    # Train the model
    trained_models[name] = model.fit(X_train_selected, y_train)

    # Predict on test set
    y_pred = model.predict(X_test_selected)
    predictions[name] = y_pred
# --- Step 5: Model Evaluation ---
print("\n--- Model Evaluation ---")
for name in models.keys():
    y_pred = predictions[name]
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\n{name}:")
    print(f"Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

# --- Compare Model Accuracy ---
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Calculate accuracy for each model
accuracies = {name: accuracy_score(y_test, predictions[name]) for name in models.keys()}

# Print accuracies
print("\n--- Model Accuracy Comparison ---")
for name, acc in accuracies.items():
    print(f"{name}: {acc:.4f}")
```

# RESULTS

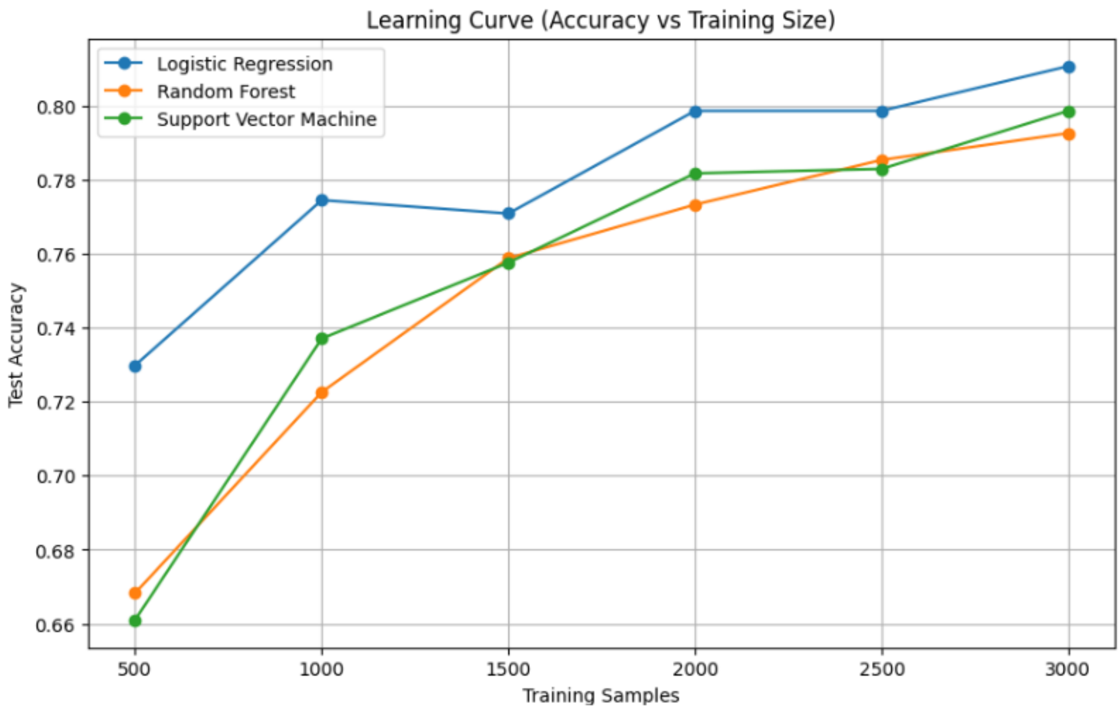
```
--- Model Evaluation ---

Logistic Regression:
Accuracy: 0.8070

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       503
           1       0.75      0.77      0.76       326

    accuracy                           0.81       829
   macro avg       0.80      0.80      0.80       829
weighted avg       0.81      0.81      0.81       829


Confusion Matrix:
[[418  85]
 [ 75 251]]

SVM:
Accuracy: 0.8118

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.85      0.85       503
           1       0.77      0.75      0.76       326

    accuracy                           0.81       829
   macro avg       0.80      0.80      0.80       829
weighted avg       0.81      0.81      0.81       829


Confusion Matrix:
[[429  74]
 [ 82 244]]

Random Forest:
Accuracy: 0.7998

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.90      0.84       503
           1       0.80      0.65      0.72       326

    accuracy                           0.80       829
   macro avg       0.80      0.77      0.78       829
weighted avg       0.80      0.80      0.80       829


Confusion Matrix:
[[451  52]
 [114 212]]

--- Model Accuracy Comparison ---
Logistic Regression: 0.8070
SVM: 0.8118
Random Forest: 0.7998
```

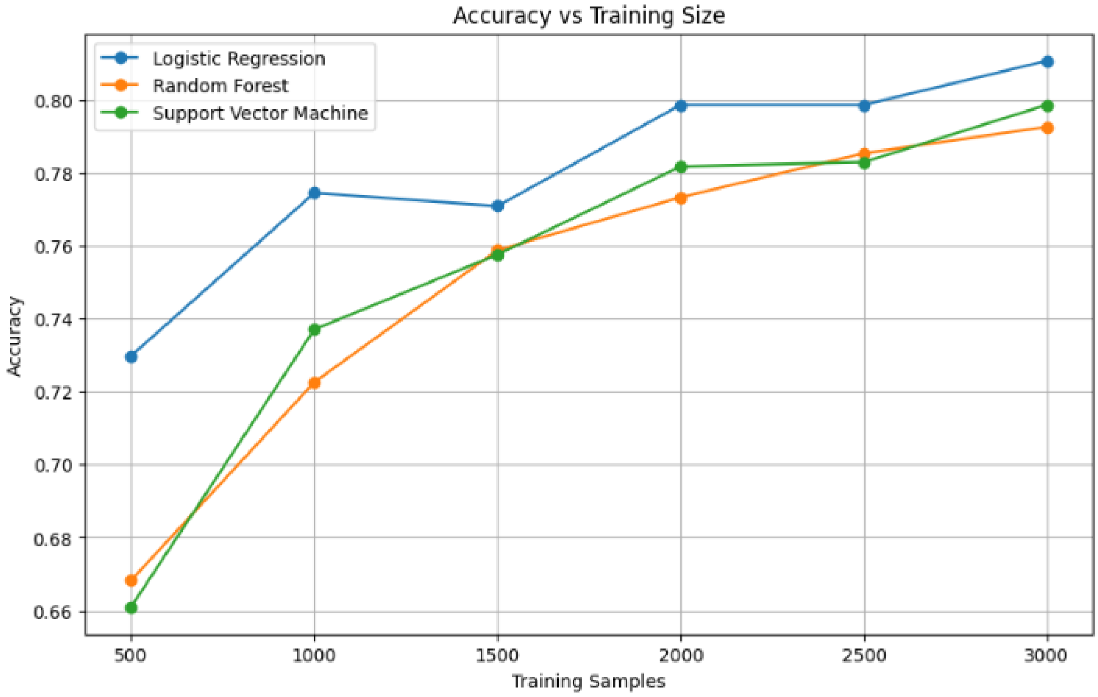
## ITERATION Method (selection of Train/Test based on 500 sample per iteration)

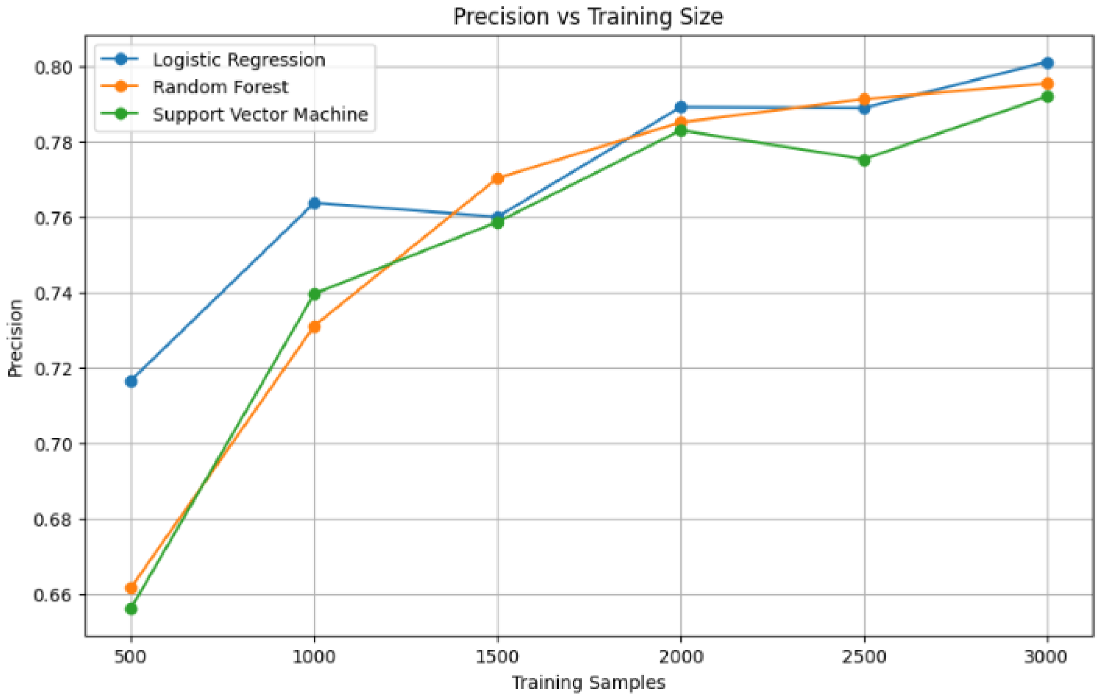
```
# --- Step 1: Prepare X and y ---
feature_names = ml_df.drop(columns=['class_label']).columns
X = ml_df.drop(columns=['class_label']).values
y = ml_df['class_label'].values
# --- Step 2: Train-Test Split (80/20) ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
# --- Step 3: Define Models ---
models = {
    'Logistic Regression': LogisticRegression(
        random_state=42,
        max_iter=1000,
        penalty='l2',
        solver='lbfgs',
        C=1.0
    ),

    'Random Forest': RandomForestClassifier(
        random_state=42,
        n_estimators=300,
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=1,
        max_features='sqrt',
        bootstrap=True,
        n_jobs=-1
    ),

    'Support Vector Machine': SVC(
        kernel='rbf',
        random_state=42,
        C=1.0,
        gamma='scale',
        probability=True
    )
}
# --- Step 4: Iterative Training Setup ---
train_sizes = list(range(500, len(X_train), 500))
results = {model: [] for model in models.keys()}
# --- Step 5: Iterative Training ---
for size in train_sizes:
    X_sub = X_train[:size]
    y_sub = y_train[:size]

    for name, model in models.items():
        model.fit(X_sub, y_sub)
        y_pred = model.predict(X_test)

        acc = accuracy_score(y_test, y_pred)
        results[name].append(acc)
```

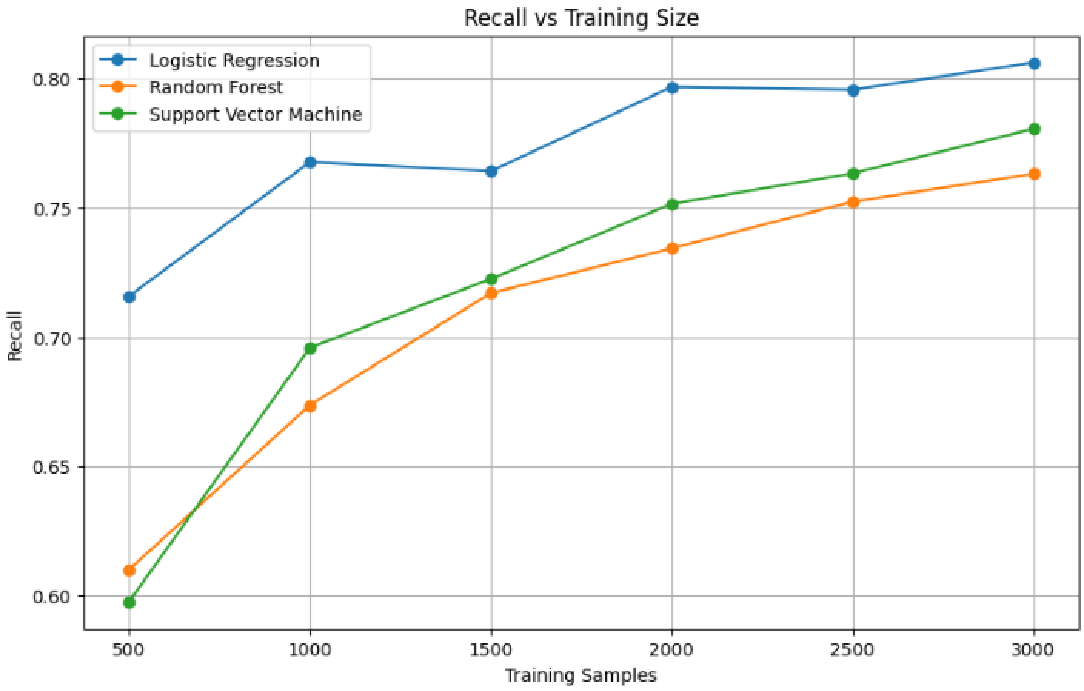
# RESULTS-

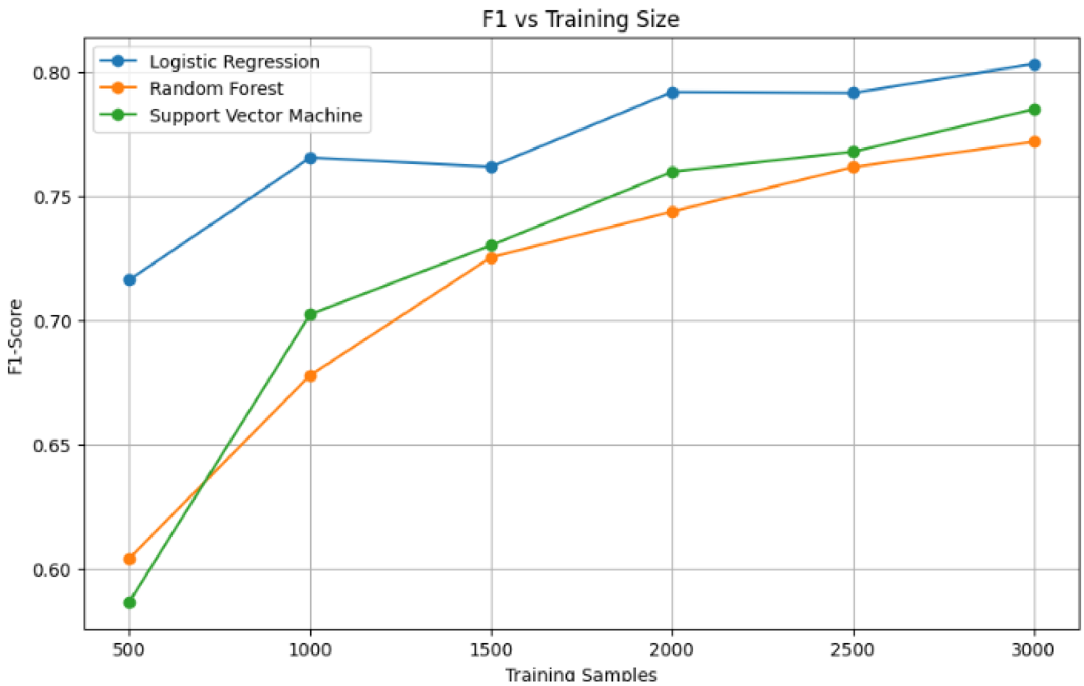
```
Accuracy over different training sizes:

Logistic Regression | Train Size: 500 | Accuracy: 0.7298
Logistic Regression | Train Size: 1000 | Accuracy: 0.7744
Logistic Regression | Train Size: 1500 | Accuracy: 0.7708
Logistic Regression | Train Size: 2000 | Accuracy: 0.7986
Logistic Regression | Train Size: 2500 | Accuracy: 0.7986
Logistic Regression | Train Size: 3000 | Accuracy: 0.8106

Random Forest | Train Size: 500 | Accuracy: 0.6683
Random Forest | Train Size: 1000 | Accuracy: 0.7226
Random Forest | Train Size: 1500 | Accuracy: 0.7587
Random Forest | Train Size: 2000 | Accuracy: 0.7732
Random Forest | Train Size: 2500 | Accuracy: 0.7853
Random Forest | Train Size: 3000 | Accuracy: 0.7925

Support Vector Machine | Train Size: 500 | Accuracy: 0.6610
Support Vector Machine | Train Size: 1000 | Accuracy: 0.7370
Support Vector Machine | Train Size: 1500 | Accuracy: 0.7575
Support Vector Machine | Train Size: 2000 | Accuracy: 0.7817
Support Vector Machine | Train Size: 2500 | Accuracy: 0.7829
Support Vector Machine | Train Size: 3000 | Accuracy: 0.7986
```




Accuracy vs Training Size
Logistic Regression
Random Forest
Support Vector Machine
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.66
Accuracy
500
1000
1500
2000
2500
3000
Training Samples


Precision vs Training Size
Logistic Regression
Random Forest
Support Vector Machine
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.66
Precision
500
1000
1500
2000
2500
3000
Training Samples


Recall vs Training Size
Logistic Regression
Random Forest
Support Vector Machine
Recall
0.60
0.65
0.70
0.75
0.80
500
1000
1500
2000
2500
3000
Training Samples


F1 vs Training Size
Logistic Regression
Random Forest
Support Vector Machine
F1-Score
0.60
0.65
0.70
0.75
0.80
500
1000
1500
2000
2500
3000
Training Samples

# Project Wise Iteration (selection of Train/Test based on PROJECT)

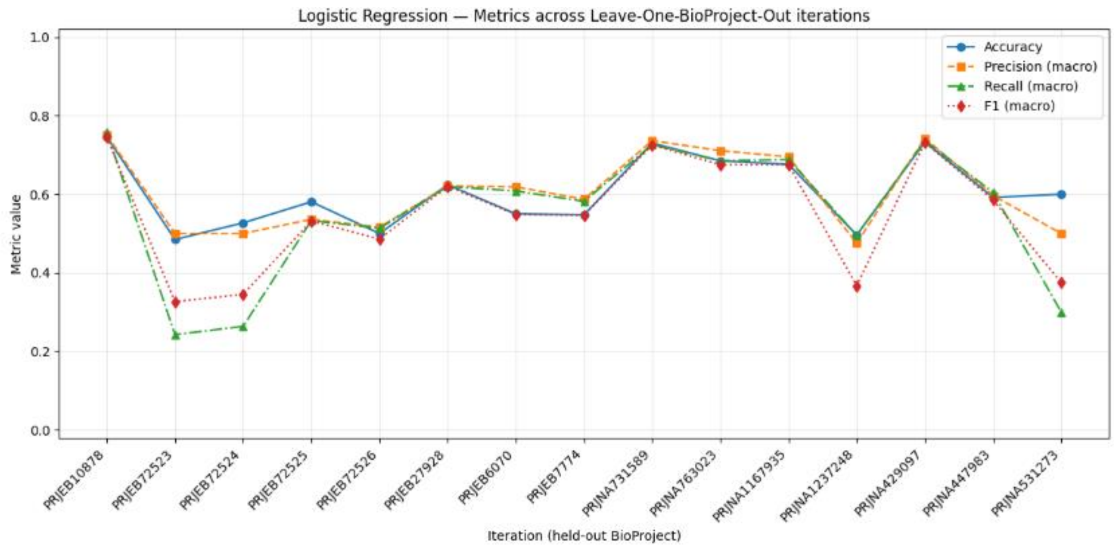
```
# --- Step 1: Prepare X and y ---
X = ml_df.drop(columns=['class_label', 'BioProject'])
y = ml_df['class_label']
groups = ml_df['BioProject']

# --- Verify columns exist ---
required_cols = {'class_label', 'BioProject'}
if not required_cols.issubset(set(ml_df.columns)):
    raise ValueError(f"ml_df must contain columns: {required_cols}. Found: {ml_df.columns.tolist()}")

# Prepare features and labels
X_df = ml_df.drop(columns=['class_label', 'BioProject'])
y = ml_df['class_label'].values
groups = ml_df['BioProject'].values
feature_names = X_df.columns.tolist()
X = X_df.values

# Define models (using the parameters you provided)
models = {
    'Logistic Regression': LogisticRegression(
        random_state=42,
        max_iter=1000,
        penalty='l2',
        solver='lbfgs',
        C=1.0
    ),
    'Random Forest': RandomForestClassifier(
        random_state=42,
        n_estimators=300,
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=1,
        max_features='sqrt',
        bootstrap=True,
        n_jobs=-1
    ),
    'Support Vector Machine': SVC(
        kernel='rbf',
        random_state=42,
        C=1.0,
        gamma='scale',
        probability=True
    )
}

# Order of BioProjects to iterate (preserve appearance order)
unique_projects = list(pd.Series(groups).unique())
n_projects = len(unique_projects)
print(f"Found {n_projects} unique BioProjects. Iterating leave-one-project-out...")
```

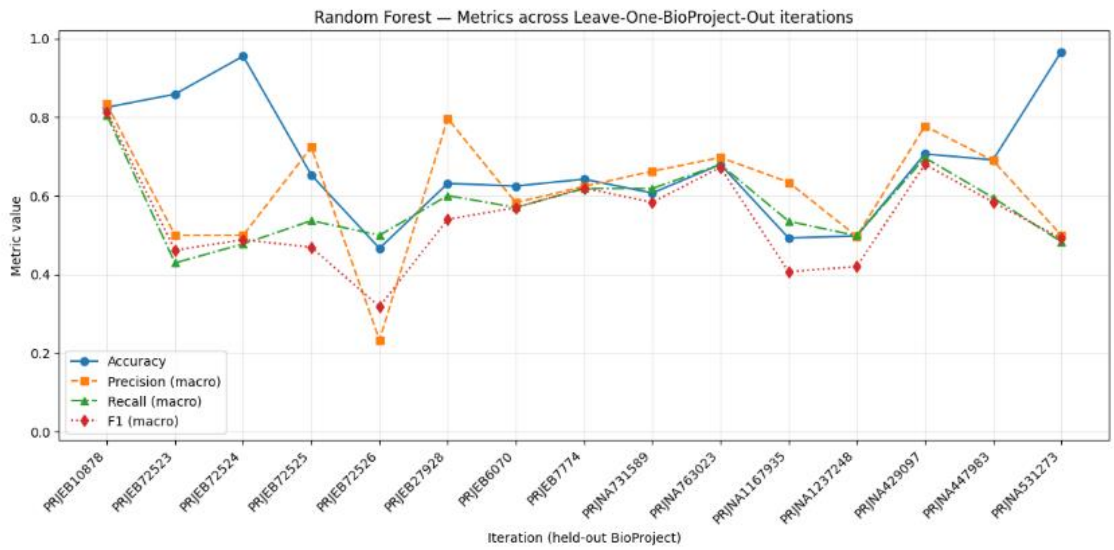
```
# Storage for metrics across iterations
# structure: metrics[model_name] = {'test_projects': [], 'accuracy': [], 'precision': [], 'recall': [], 'f1': [], 'train_sizes': []}
metrics = {}
for name in models.keys():
    metrics[name] = {
        'test_projects': [],
        'accuracy': [],
        'precision': [],
        'recall': [],
        'f1': [],
        'train_sizes': []
    }

# Loop over projects
for idx, test_proj in enumerate(unique_projects):
    # boolean masks
    test_mask = (groups == test_proj)
    train_mask = ~test_mask

    X_train = X[train_mask]
    y_train = y[train_mask]
    X_test = X[test_mask]
    y_test = y[test_mask]

    train_size = X_train.shape[0]
    test_size = X_test.shape[0]
    print(f"[{idx+1}/{n_projects}] Test project: {test_proj} (train_size={train_size}, test_size={test_size})")

    # If a test project contains no samples or train has no samples, skip (safety)
    if test_size == 0 or train_size == 0:
        print(f"  Skipping {test_proj} because train or test size is zero.")
        continue

    for name, model in models.items():
        # Clone model to avoid warm-starting from previous fit
        # simple way: re-create new instance with same params
        # we'll use the same class and its get_params to re-instantiate
        cls = model.__class__
        params = model.get_params()
        clf = cls(**params)

        # Train and predict
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        # Compute metrics (macro-average); guard against undefined metrics
        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred, average='macro', zero_division=0)
        rec = recall_score(y_test, y_pred, average='macro', zero_division=0)
        f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)
```

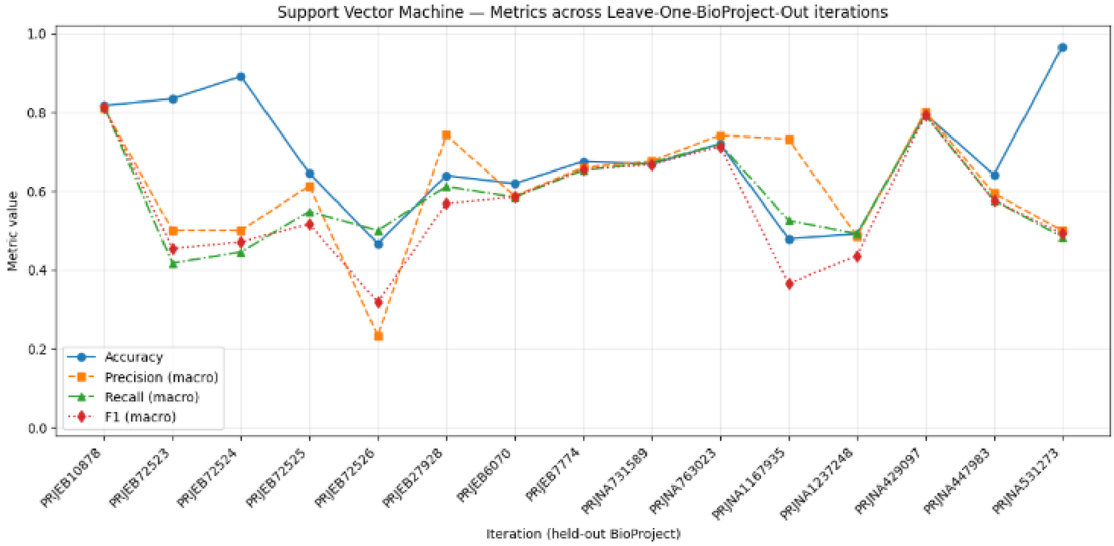
```
        # Store
        metrics[name]['test_projects'].append(test_proj)
        metrics[name]['accuracy'].append(acc)
        metrics[name]['precision'].append(prec)
        metrics[name]['recall'].append(rec)
        metrics[name]['f1'].append(f1)
        metrics[name]['train_sizes'].append(train_size)

# --- Create a long-format summary DataFrame for easier viewing ---
rows = []
for name in models.keys():
    for i, test_proj in enumerate(metrics[name]['test_projects']):
        rows.append({
            'Model': name,
            'Test_BioProject': test_proj,
            'Train_Size': metrics[name]['train_sizes'][i],
            'Accuracy': metrics[name]['accuracy'][i],
            'Precision_macro': metrics[name]['precision'][i],
            'Recall_macro': metrics[name]['recall'][i],
            'F1_macro': metrics[name]['f1'][i]
        })

summary_df = pd.DataFrame(rows)

# Order rows nicely
summary_df = summary_df[['Model', 'Test_BioProject', 'Train_Size', 'Accuracy', 'Precision_macro',
'Recall_macro', 'F1_macro']]

# Display summary table
pd.set_option('display.float_format', lambda x: f"{x:.4f}")
print("\nPer-iteration results (first 30 rows):")
print(summary_df.head(30))

# --- Plotting: One combined plot per model (Accuracy, Precision, Recall, F1 vs iteration)
metric_names = ['Accuracy', 'Precision_macro', 'Recall_macro', 'F1_macro']
ylabel_map = {
    'Accuracy': 'Accuracy',
    'Precision_macro': 'Precision (macro)',
    'Recall_macro': 'Recall (macro)',
    'F1_macro': 'F1 (macro)'
}

for name in models.keys():
    df_model = summary_df[summary_df['Model'] == name].reset_index(drop=True)
    if df_model.shape[0] == 0:
        continue
```

# RESULTS

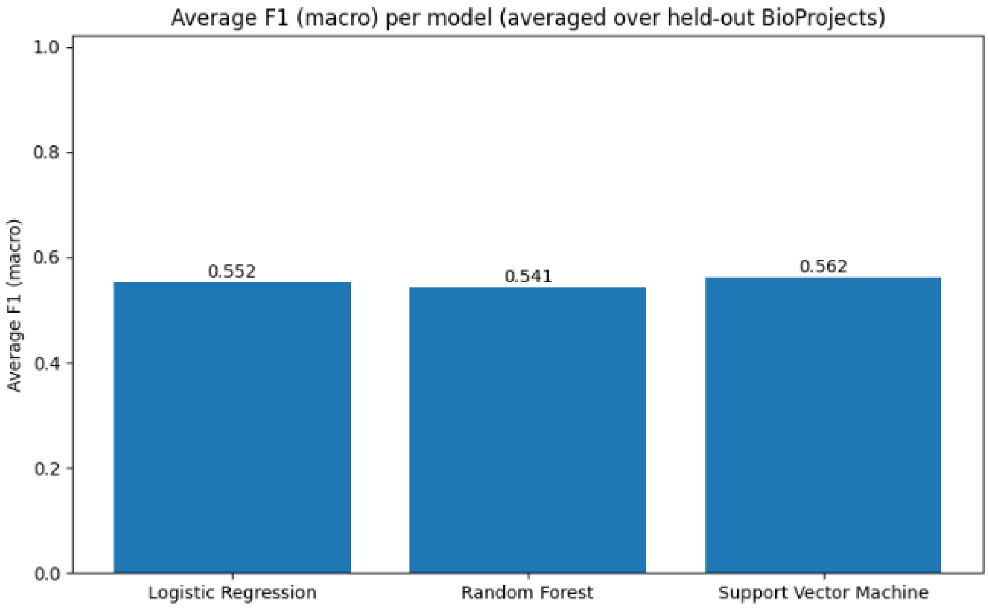
```
Found 15 unique BioProjects. Iterating leave-one-project-out...
[1/15] Test project: PRJEB10878 (train_size=4017, test_size=126)
[2/15] Test project: PRJEB72523 (train_size=3980, test_size=163)
[3/15] Test project: PRJEB72524 (train_size=3940, test_size=203)
[4/15] Test project: PRJEB72525 (train_size=4019, test_size=124)
[5/15] Test project: PRJEB72526 (train_size=4083, test_size=60)
[6/15] Test project: PRJEB27928 (train_size=3885, test_size=258)
[7/15] Test project: PRJEB6070 (train_size=2865, test_size=1278)
[8/15] Test project: PRJEB7774 (train_size=3838, test_size=305)
[9/15] Test project: PRJNA731589 (train_size=3980, test_size=163)
[10/15] Test project: PRJNA763023 (train_size=3943, test_size=200)
[11/15] Test project: PRJNA1167935 (train_size=4072, test_size=71)
[12/15] Test project: PRJNA1237248 (train_size=3292, test_size=851)
[13/15] Test project: PRJNA429097 (train_size=3952, test_size=191)
[14/15] Test project: PRJNA447983 (train_size=4023, test_size=120)
[15/15] Test project: PRJNA531273 (train_size=4113, test_size=30)

Per-iteration results (first 30 rows):
                  Model Test_BioProject  Train_Size  Accuracy  \
0   Logistic Regression      PRJEB10878        4017    0.7460
1   Logistic Regression      PRJEB72523        3980    0.4847
2   Logistic Regression      PRJEB72524        3940    0.5271
3   Logistic Regression      PRJEB72525        4019    0.5806
4   Logistic Regression      PRJEB72526        4083    0.5000
5   Logistic Regression      PRJEB27928        3885    0.6240
6   Logistic Regression       PRJEB6070        2865    0.5501
7   Logistic Regression       PRJEB7774        3838    0.5475
8   Logistic Regression     PRJNA731589        3980    0.7301
9   Logistic Regression     PRJNA763023        3943    0.6850
10  Logistic Regression    PRJNA1167935        4072    0.6761
11  Logistic Regression    PRJNA1237248        3292    0.4959
12  Logistic Regression     PRJNA429097        3952    0.7330
13  Logistic Regression     PRJNA447983        4023    0.5917
14  Logistic Regression     PRJNA531273        4113    0.6000
15        Random Forest      PRJEB10878        4017    0.8254
16        Random Forest      PRJEB72523        3980    0.8589
17        Random Forest      PRJEB72524        3940    0.9557
18        Random Forest      PRJEB72525        4019    0.6532
19        Random Forest      PRJEB72526        4083    0.4667
20        Random Forest      PRJEB27928        3885    0.6318
21        Random Forest       PRJEB6070        2865    0.6252
22        Random Forest       PRJEB7774        3838    0.6426
23        Random Forest     PRJNA731589        3980    0.6074
24        Random Forest     PRJNA763023        3943    0.6800
25        Random Forest    PRJNA1167935        4072    0.4930
26        Random Forest    PRJNA1237248        3292    0.4982
27        Random Forest     PRJNA429097        3952    0.7068
28        Random Forest     PRJNA447983        4023    0.6917
29        Random Forest     PRJNA531273        4113    0.9667

    Precision_macro  Recall_macro  F1_macro
0            0.7508        0.7581    0.7450
1            0.5000        0.2423    0.3264
2            0.5000        0.2635    0.3452
3            0.5361        0.5329    0.5319
4            0.5159        0.5134    0.4857
5            0.6210        0.6196    0.6199
6            0.6187        0.6082    0.5483
7            0.5871        0.5820    0.5461
8            0.7363        0.7259    0.7255
9            0.7108        0.6850    0.6750
10           0.6952        0.6883    0.6750
11           0.4753        0.4954    0.3670
12           0.7423        0.7363    0.7319
13           0.5965        0.6048    0.5852
14           0.5000        0.3000    0.3750
15           0.8333        0.8056    0.8134
16           0.5000        0.4294    0.4620
17           0.5000        0.4778    0.4887
18           0.7235        0.5371    0.4693
19           0.2333        0.5000    0.3182
20           0.7970        0.6008    0.5405
21           0.5830        0.5707    0.5702
22           0.6242        0.6189    0.6203
23           0.6628        0.6188    0.5835
24           0.6978        0.6800    0.6726
25           0.6348        0.5357    0.4072
26           0.4971        0.4987    0.4203
27           0.7777        0.6972    0.6803
28           0.6899        0.5954    0.5842
29           0.5000        0.4833    0.4915
```


Logistic Regression — Metrics across Leave-One-BioProject-Out iterations
Accuracy
Precision (macro)
Recall (macro)
F1 (macro)
Metric value
Iteration (held-out BioProject)
PRJEB10878
PRJEB72523
PRJEB72524
PRJEB72525
PRJEB72526
PRJEB27928
PRJEB6070
PRJEB7774
PRJNA731589
PRJNA763023
PRJNA1167935
PRJNA1237248
PRJNA429097
PRJNA447983
PRJNA531273


Random Forest — Metrics across Leave-One-BioProject-Out iterations
Accuracy
Precision (macro)
Recall (macro)
F1 (macro)
Metric value
Iteration (held-out BioProject)
PRJEB10878
PRJEB72523
PRJEB72524
PRJEB72525
PRJEB72526
PRJEB27928
PRJEB6070
PRJEB7774
PRJNA731589
PRJNA763023
PRJNA1167935
PRJNA1237248
PRJNA429097
PRJNA447983
PRJNA531273

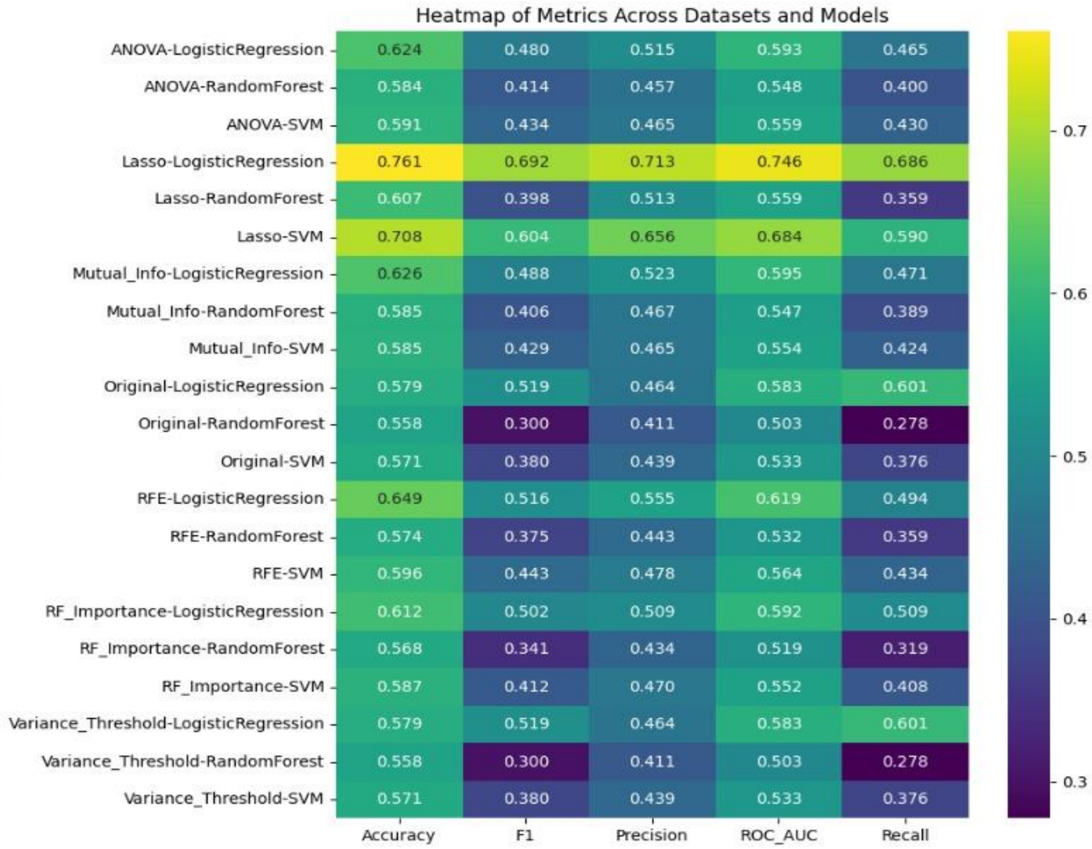


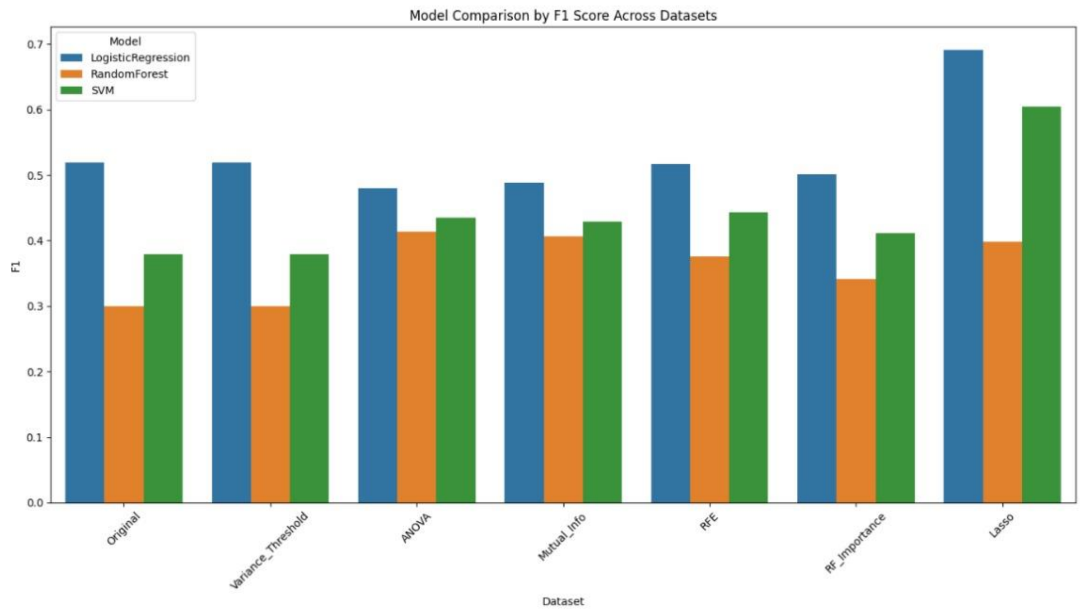

```
Average metrics across all iterations:
                        Accuracy  Precision_macro  Recall_macro  F1_macro
Model
Logistic Regression       0.6048           0.6057        0.5570    0.5522
Random Forest             0.6869           0.6170        0.5766    0.5415
Support Vector Machine    0.6901           0.6119        0.5893    0.5618
```



# FINAL SUMMARY REPORT

1) In starting, I had 4,143 samples after merging. Then, I performed data preprocessing (CLR transformation and threshold cut-off) then feature selection using various methods such as Variance Threshold, Univariate Analysis, RFE, LASSO, Mutual Information, and Feature Importance. After that, I trained three machine learning models SVM, Random Forest, and Logistic Regression then I got this result.

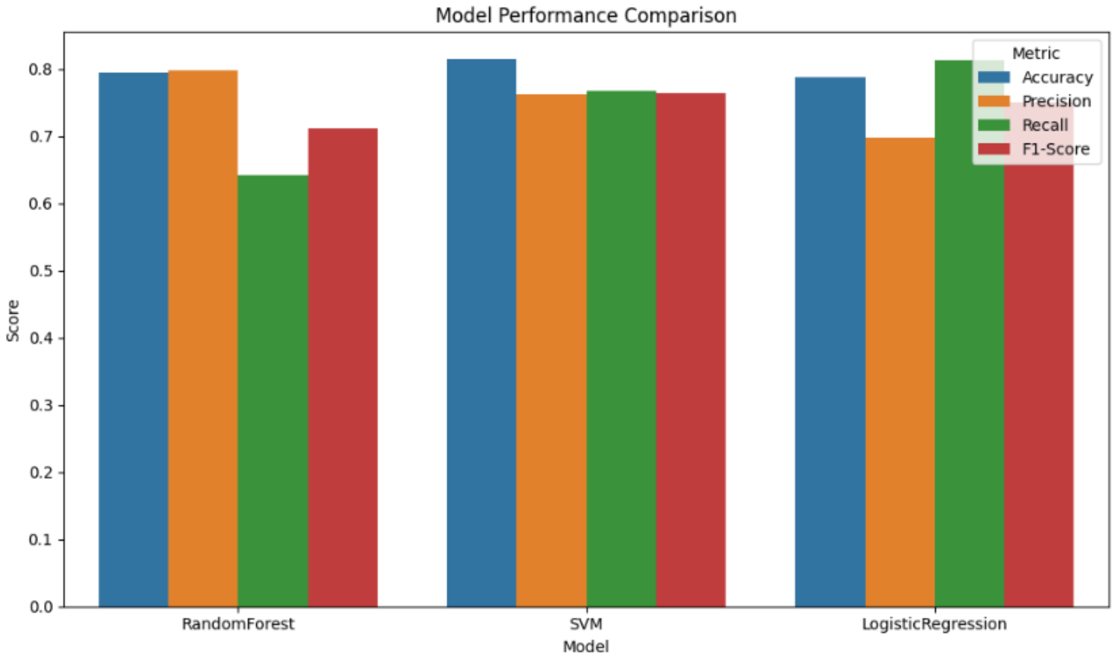


2) Then I did hyperparameter tuning using 5-fold cross-validation and trained the three models (SVM, Random Forest, and Logistic Regression ) .Then I got this result.

Model Comparison:

| | Model | Accuracy | Precision | Recall | F1-Score | ROC-AUC |
|---|---|---|---|---|---|---|
| 0 | RandomForest | 0.794934 | 0.797710 | 0.641104 | 0.710884 | 0.894654 |
| 1 | SVM | 0.814234 | 0.762195 | 0.766871 | 0.764526 | N/A |
| 2 | LogisticRegression | 0.787696 | 0.697368 | 0.812883 | 0.750708 | 0.868458 |




3) Then I trained the model on original dataset without feature selection then I got this.

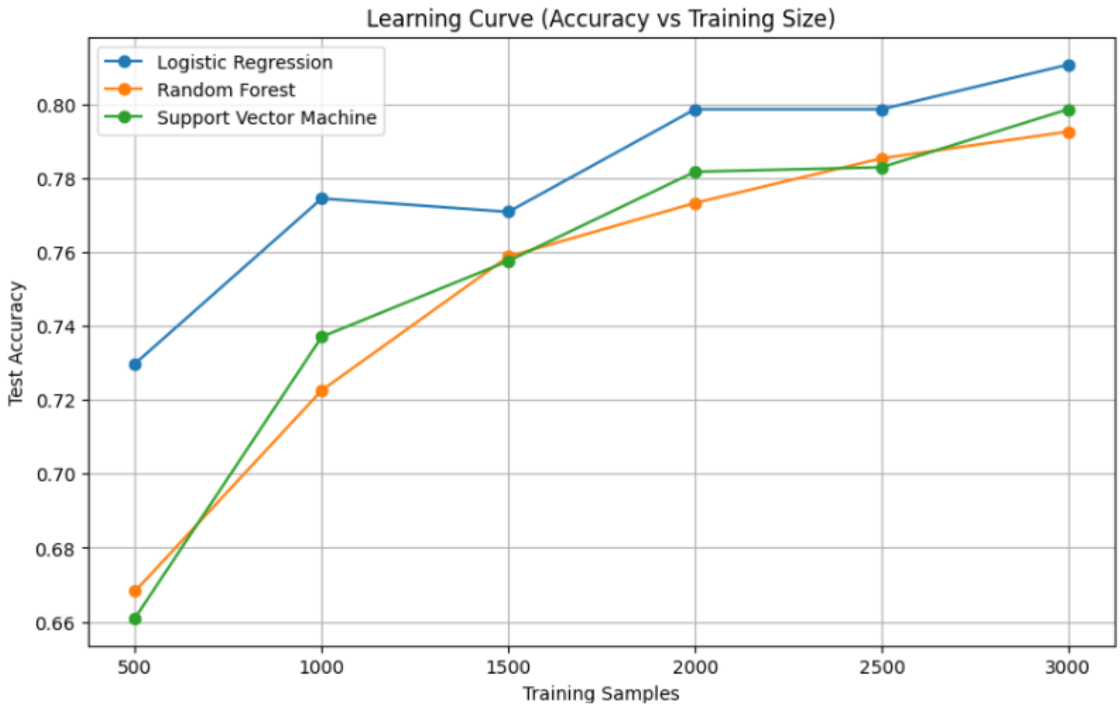
```
--- Logistic Regression ---
Accuracy: 0.8022

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.81      0.83       503
           1       0.73      0.79      0.76       326

    accuracy                           0.80       829
   macro avg       0.79      0.80      0.80       829
weighted avg       0.81      0.80      0.80       829


Confusion Matrix:
[[407  96]
 [ 68 258]]

--- Random Forest ---
Accuracy: 0.7973

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       503
           1       0.81      0.64      0.71       326

    accuracy                           0.80       829
   macro avg       0.80      0.77      0.78       829
weighted avg       0.80      0.80      0.79       829


Confusion Matrix:
[[453  50]
 [118 208]]

--- Support Vector Machine ---
Accuracy: 0.8034

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       503
           1       0.77      0.71      0.74       326

    accuracy                           0.80       829
   macro avg       0.80      0.79      0.79       829
weighted avg       0.80      0.80      0.80       829


Confusion Matrix:
[[434  69]
 [ 94 232]]

--- Model Comparison ---
                    Model  Accuracy
2  Support Vector Machine  0.803378
0     Logistic Regression  0.802171
1           Random Forest  0.797346
```

4) Then I did feature selection with L1 lasso and then trained the model.

```
--- Model Evaluation ---

Logistic Regression:
Accuracy: 0.8070

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       503
           1       0.75      0.77      0.76       326

    accuracy                           0.81       829
   macro avg       0.80      0.80      0.80       829
weighted avg       0.81      0.81      0.81       829


Confusion Matrix:
[[418  85]
 [ 75 251]]

SVM:
Accuracy: 0.8118

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.85      0.85       503
           1       0.77      0.75      0.76       326

    accuracy                           0.81       829
   macro avg       0.80      0.80      0.80       829
weighted avg       0.81      0.81      0.81       829


Confusion Matrix:
[[429  74]
 [ 82 244]]

Random Forest:
Accuracy: 0.7998

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.90      0.84       503
           1       0.80      0.65      0.72       326

    accuracy                           0.80       829
   macro avg       0.80      0.77      0.78       829
weighted avg       0.80      0.80      0.80       829


Confusion Matrix:
[[451  52]
 [114 212]]

--- Model Accuracy Comparison ---
Logistic Regression: 0.8070
SVM: 0.8118
Random Forest: 0.7998
```

5) Then I used iteration method (500 samples per iteration).

```
Accuracy over different training sizes:

Logistic Regression | Train Size: 500 | Accuracy: 0.7298
Logistic Regression | Train Size: 1000 | Accuracy: 0.7744
Logistic Regression | Train Size: 1500 | Accuracy: 0.7708
Logistic Regression | Train Size: 2000 | Accuracy: 0.7986
Logistic Regression | Train Size: 2500 | Accuracy: 0.7986
Logistic Regression | Train Size: 3000 | Accuracy: 0.8106

Random Forest | Train Size: 500 | Accuracy: 0.6683
Random Forest | Train Size: 1000 | Accuracy: 0.7226
Random Forest | Train Size: 1500 | Accuracy: 0.7587
Random Forest | Train Size: 2000 | Accuracy: 0.7732
Random Forest | Train Size: 2500 | Accuracy: 0.7853
Random Forest | Train Size: 3000 | Accuracy: 0.7925

Support Vector Machine | Train Size: 500 | Accuracy: 0.6610
Support Vector Machine | Train Size: 1000 | Accuracy: 0.7370
Support Vector Machine | Train Size: 1500 | Accuracy: 0.7575
Support Vector Machine | Train Size: 2000 | Accuracy: 0.7817
Support Vector Machine | Train Size: 2500 | Accuracy: 0.7829
Support Vector Machine | Train Size: 3000 | Accuracy: 0.7986
```




6) Then I used project wise method and I got this.

Average metrics across all iterations:

| Model | Accuracy | Precision_macro | Recall_macro | F1_macro |
|---|---|---|---|---|
| Logistic Regression | 0.6048 | 0.6057 | 0.5570 | 0.5522 |
| Random Forest | 0.6869 | 0.6170 | 0.5766 | 0.5415 |
| Support Vector Machine | 0.6901 | 0.6119 | 0.5893 | 0.5618 |