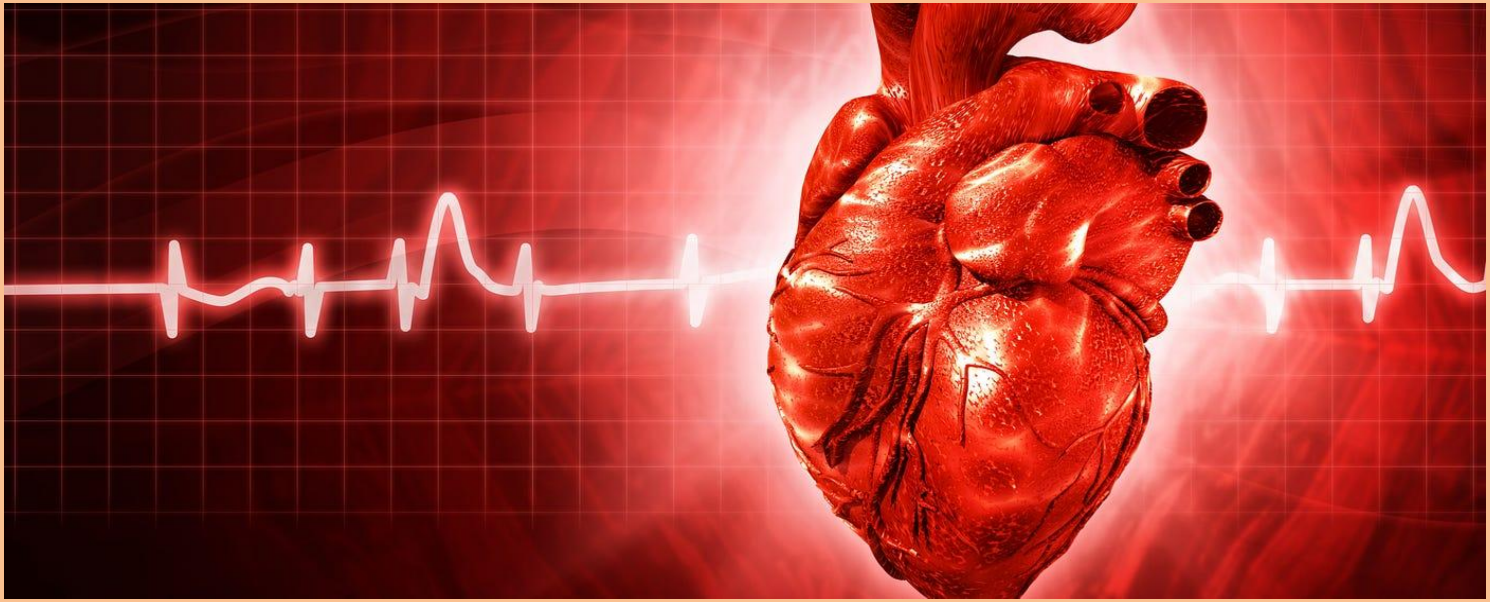
HEART DISEASE PREDICTION USING PYSPARK



TECHZEE TEAM
BIG DATA PROJECT

Problem Statement

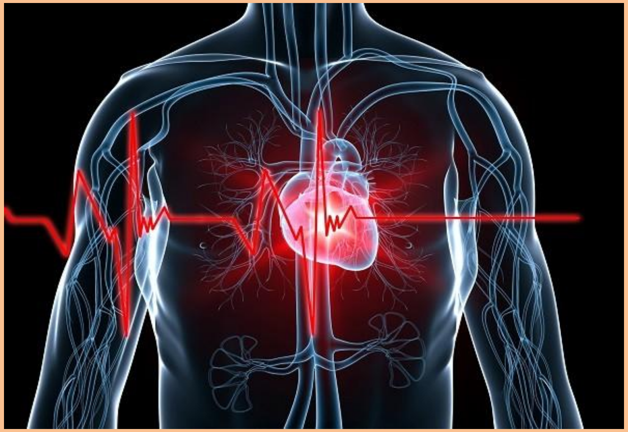
- **Objective:** Develop machine learning models to classify the presence or absence of heart disease from clinical features such as age, cholesterol levels, blood pressure, and maximum heart rate.

- **Models:** Train and compare three classification models: Logistic Regression, Random Forest, Support Vector Machine (SVM)

- **Data:** Use a dataset from the UCI Machine Learning Repository with 14 clinical attributes, along with a target attribute signifying whether a patient is suffering from heart disease.

- **Evaluation:** Assess model performance based on measures such as accuracy, precision, recall, F1-score, and Auc. Choose the top-performing model for making predictions.

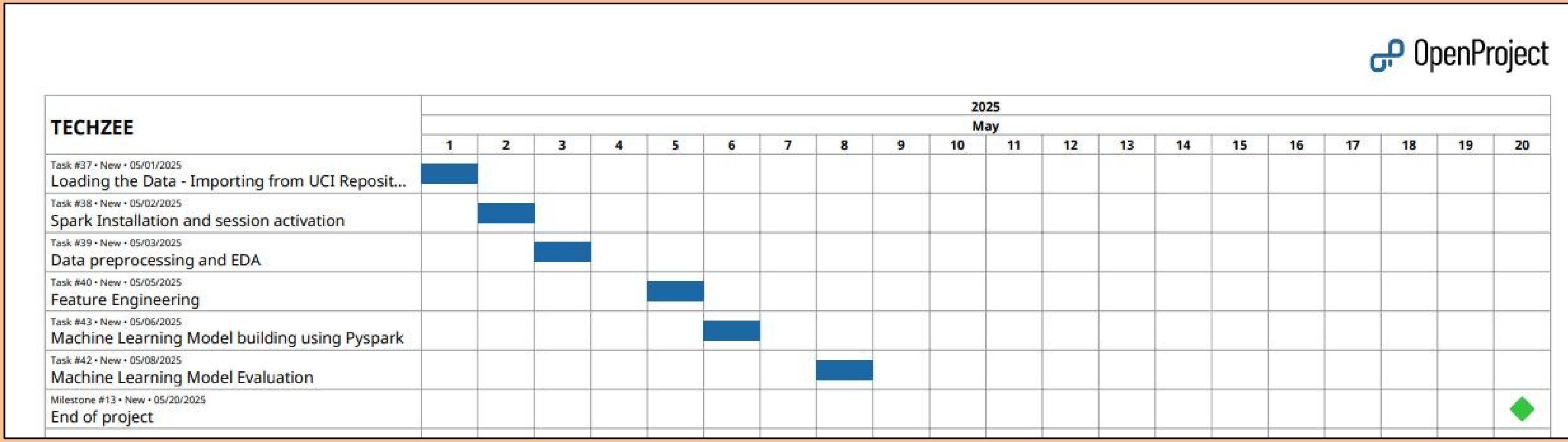
- **Prediction:** Allow predictions on new patient data where the user selects a trained model and enters the clinical features and gets a prediction of the presence of heart disease.



Literature Review

Study Title	*Authors*	*Techniques Used*	*Best Accuracy (%)*	*Dataset*	*Inspiration for Our Work*
Heart Disease Prediction using Machine Learning Algorithms	M. Anbarasi et al.	Decision Tree, Naïve Bayes, SVM	83.2	UCI Heart Disease	Model selection approach and performance benchmarking
Comparative Study of Classification Algorithms for Heart Disease	J. D. Kaur & M. Kaur	KNN, Logistic Regression, Random Forest	81.5	UCI Repository	Inclusion of multiple ML algorithms for comparison
Heart Disease Prediction using Hybrid Ensemble Learning	S. Sharma et al.	Random Forest, XGBoost (Ensemble)	85.0	Cleveland Heart Dataset	Use of ensemble techniques for better accuracy
Prediction of Heart Disease using Deep Learning	R. Gupta et al.	ANN, CNN	78.6	UCI Heart Dataset	Inclusion of ANN in model experimentation
Predictive Analysis on Heart Disease using Data Mining Techniques	K. Srinivas et al.	Naïve Bayes, Decision Tree	75.2	Public Health Records	Focus on simplicity and interpretability of models

Project Plan



Phase 1: Data Collection & Cleaning

Phase 2: Exploratory Data Analysis (EDA)

Phase 3: Feature Engineering

Phase 4: Model Training & Evaluation

Phase 5: Results Interpretation

Dataset Description - 14 clinical attributes

1. **age:** age in years
2. **sex:** sex (1 = male; 0 = female)
3. **cp:** chest pain type -Value 1: typical angina -Value 2: atypical angina -Value 3: non-anginal pain -Value 4: asymptomatic
4. **trestbps:** resting blood pressure (in mm Hg on admission to the hospital)
5. **chol:** serum cholestoral in mg/dl
6. **fbs:** (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. **restecg:** resting electrocardiographic results - Value 0: normal

   - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)

   - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. **thalach:** maximum heart rate achieved
9. **exang:** exercise induced angina (1 = yes; 0 = no)
10. **oldpeak** : ST depression induced by exercise relative to rest
11. **slope:** the slope of the peak exercise ST segment -Value 1: upsloping,Value 2: flat, Value 3: downsloping
12. **ca:** number of major vessels (0-3) colored by flourosopy
13. **thal:** 3 = normal; 6 = fixed defect; 7 = reversable defect
14. **num:** diagnosis of heart disease (angiographic disease status)

    - Value 0: < 50% diameter narrowing

    - Value 1: > 50% diameter narrowing

Coding and Implementation

Phase 1: Data Collection & Cleaning

Data Import:

Loaded UCI Heart Disease dataset using ucimlrepo.

Data Cleaning:

- Removed duplicate records.
- Checked for nulls and schema consistency.

Data Splitting:

Performed train-test split (80% training, 20% testing).

Coding and Implementation

Phase 1: Data Collection & Cleaning

```
!pip install ucimlrepo
from pyspark.sql import SparkSession
from ucimlrepo import fetch_ucirepo
import pandas as pd
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.11/dist-packages (0.0.7)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2025.4.26)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
```

```
# Initialize PySpark
spark = SparkSession.builder.appName("HeartDiseasePrediction").getOrCreate()

# Fetch dataset from UCI repository
heart_disease = fetch_ucirepo(id=45)

# Convert to Pandas DataFrame and then to Spark DataFrame
X = heart_disease.data.features
y = heart_disease.data.targets
df = spark.createDataFrame(pd.concat([X, y], axis=1))

from pyspark.sql.functions import col, count, when, isnull
from pyspark.sql.types import DoubleType

# Show initial schema and data
print("Initial Schema:")
df.printSchema()
print("\nFirst 5 rows:")
df.show(5)

# Check for missing values
print("\nMissing values count:")
df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()
```

Coding and Implementation

Phase 1: Data Collection & Cleaning

```
# Check for duplicates
print("\nDuplicate rows count:", df.count() - df.dropDuplicates().count())

# Convert target to binary (0 = no disease, 1 = disease)
df = df.withColumn("target", when(col("num") > 0, 1).otherwise(0)).drop("num")

# Data Splitting (80% train, 20% test)
train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)
print(f"\nTraining set count: {train_df.count()}")
print(f"Test set count: {test_df.count()}")
```

```
Initial Schema:
root
 |-- age: long (nullable = true)
 |-- sex: long (nullable = true)
 |-- cp: long (nullable = true)
 |-- trestbps: long (nullable = true)
 |-- chol: long (nullable = true)
 |-- fbs: long (nullable = true)
 |-- restecg: long (nullable = true)
 |-- thalach: long (nullable = true)
 |-- exang: long (nullable = true)
 |-- oldpeak: double (nullable = true)
 |-- slope: long (nullable = true)
 |-- ca: double (nullable = true)
 |-- thal: double (nullable = true)
 |-- num: long (nullable = true)
```

```
First 5 rows:
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+---+
|age|sex| cp|trestbps|chol|fbs|restecg|thalach|exang|oldpeak|slope| ca|thal|num|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+---+
| 63|  1|  1|     145| 233|  1|      2|    150|    0|    2.3|    3|0.0| 6.0|  0|
| 67|  1|  4|     160| 286|  0|      2|    108|    1|    1.5|    2|3.0| 3.0|  2|
| 67|  1|  4|     120| 229|  0|      2|    129|    1|    2.6|    2|2.0| 7.0|  1|
| 37|  1|  3|     130| 250|  0|      0|    187|    0|    3.5|    3|0.0| 3.0|  0|
| 41|  0|  2|     130| 204|  0|      2|    172|    0|    1.4|    1|0.0| 3.0|  0|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+---+
only showing top 5 rows


Missing values count:
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+---+
|age|sex| cp|trestbps|chol|fbs|restecg|thalach|exang|oldpeak|slope| ca|thal|num|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+---+
|  0|  0|  0|       0|   0|  0|      0|      0|    0|      0|    0|  0|   0|  0|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+---+
```

```
Duplicate rows count: 0

Training set count: 246
Test set count: 51
```

Coding and Implementation

Phase 2: Exploratory Data Analysis (EDA)

Descriptive Statistics:

Computed summary statistics: mean, std, min, max, etc.

Target Variable Analysis:

Analyzed distribution of heart disease (target = 0 or 1).

correlation and statistical overview.

Coding and Implementation

Phase 2: Exploratory Data Analysis (EDA)

Phase 2: Exploratory Data Analysis (EDA)

```
[ ] # Descriptive Statistics
    print("\nDescriptive Statistics for Numerical Features:")
    train_df.describe().show()

    # Target Variable Analysis
    print("\nTarget Class Distribution:")
    train_df.groupBy("target").count().show()

    # Correlation Analysis (using Pandas for visualization)
    numeric_cols = [f.name for f in train_df.schema.fields if isinstance(f.dataType, DoubleType)]
    corr_matrix = train_df.select(numeric_cols).toPandas().corr()

    import matplotlib.pyplot as plt
    import seaborn as sns
    plt.figure(figsize=(12, 10))
    sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", center=0)
    plt.title("Feature Correlation Matrix")
    plt.show()
```

Coding and Implementation

<u>Phase 2: Exploratory Data Analysis (EDA)</u>

Descriptive Statistics for Numerical Features:

summary	age	sex	cp	trestbps	chol	fbs	restecg
count	246	246	246	246	246	246	246
mean	54.89837398373984	0.6869918699186992	3.1463414634146343	132.4349593495935	249.47154471544715	0.15853658536585366	1.040650406504065 2
stddev	8.94346400259361 7	0.4646630290284314	0.982981112026217 3	17.525545128159546	53.34857650924998	0.3659880290823157	0.993023615920258 2
min	29	0	1	100	126	0	0
max	77	1	4	200	564	1	2

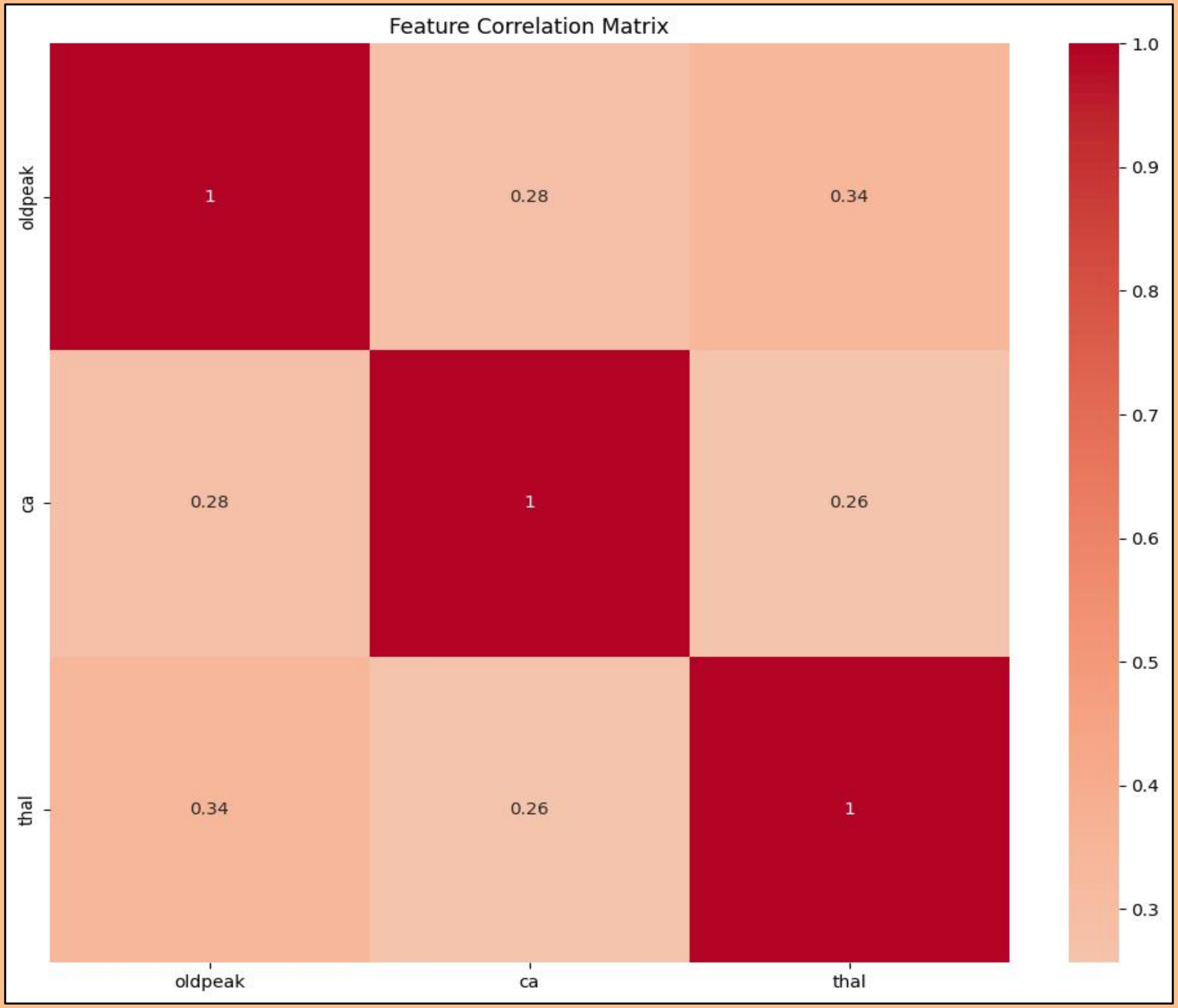
thalach	exang	oldpeak	slope	ca	thal	target
246	246	246	246	246	246	246
150.0040650406504	0.3252032520325203	1.0760162601626018	1.6422764227642277	0.6991869918699187	4.772357723577236	0.483739837398374
23.30971558279626	0.4694057895410828	1.1820917052980822	0.6341323019211401	0.9513467090444417	1.9412304456040512	0.500754367051142
71	0	0.0	1	0.0	3.0	0
202	1	6.2	3	3.0	7.0	1

Target Class Distribution:

target	count
1	119
0	127

Coding and Implementation

Phase 2: Exploratory Data Analysis (EDA)



Coding and Implementation

Phase 3: Feature Engineering

Feature Selection:

- Selected relevant features for model input (based on correlation and domain knowledge).

Feature Transformation:

- Used VectorAssembler to combine features.
- Applied StandardScaler to normalize numerical features.

Coding and Implementation

Phase 3: Feature Engineering

Phase 3: Feature Engineering

```
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml import Pipeline

# Select important features based on EDA
selected_features = ['age', 'sex', 'cp', 'trestbps', 'chol', 'thalach',
                     'exang', 'oldpeak', 'slope', 'ca', 'thal','fbs','restecg']

# Create feature pipeline
assembler = VectorAssembler(inputCols=selected_features, outputCol="rawFeatures")
scaler = StandardScaler(inputCol="rawFeatures", outputCol="features")

feature_pipeline = Pipeline(stages=[assembler, scaler])
feature_model = feature_pipeline.fit(train_df)

# Transform datasets
train_df = feature_model.transform(train_df)
test_df = feature_model.transform(test_df)

# Cache DataFrames for better performance
train_df.cache()
test_df.cache()
```

```
DataFrame[age: bigint, sex: bigint, cp: bigint, trestbps: bigint, chol: bigint, fbs: bigint, restecg: bigint, thalach: bigint, exang: bigint, oldpeak: double, slope: bigint, ca:
double, thal: double, target: int, rawFeatures: vector, features: vector]
```

Coding and Implementation

Phase 4: Model Training and Evaluation

Model Selection:

Trained the following classification models:

- Logistic Regression
- Random Forest
- Support Vector Machine (SVM)

Model Training:

Built pipelines for each model using PySpark's MLlib.

Model Evaluation:

Evaluated models using:

- Accuracy
- Precision
- Recall
- F1 Score
- Area Under ROC Curve (AUC)

Model Comparison:

- Compared model performance based on evaluation metrics.
- Identified the best model.

Coding and Implementation

Phase 4: Model Training and Evaluation

Phase 4: Model Training & Evaluation

```
[ ] from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, LinearSVC
    from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
    from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

    # Initialize models
    lr = LogisticRegression(featuresCol="features", labelCol="target")
    rf = RandomForestClassifier(featuresCol="features", labelCol="target")
    svm = LinearSVC(featuresCol="features", labelCol="target")

    # Set up evaluators
    binary_evaluator = BinaryClassificationEvaluator(labelCol="target")
    multi_evaluator = MulticlassClassificationEvaluator(labelCol="target")
```

```
# Create parameter grids for tuning
lr_paramGrid = (ParamGridBuilder()
               .addGrid(lr.regParam, [0.01, 0.1, 1.0])
               .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])
               .build())

rf_paramGrid = (ParamGridBuilder()
               .addGrid(rf.numTrees, [10, 50, 100])
               .addGrid(rf.maxDepth, [5, 10, 20])
               .build())

svm_paramGrid = (ParamGridBuilder()
                .addGrid(svm.regParam, [0.01, 0.1, 1.0])
                .build())

# Set up cross-validation
cv = CrossValidator(estimator=lr,
                    estimatorParamMaps=lr_paramGrid,
                    evaluator=multi_evaluator,
                    numFolds=5,
                    seed=42)
```

Coding and Implementation

Phase 4: Model Training and Evaluation

```
# Train Random Forest
print("\nTraining Random Forest...")
cv.setEstimator(rf)
cv.setEstimatorParamMaps(rf_paramGrid)
rf_model = cv.fit(train_df)

# Train SVM
print("\nTraining SVM...")
cv.setEstimator(svm)
cv.setEstimatorParamMaps(svm_paramGrid)
svm_model = cv.fit(train_df)

# Function to evaluate models
def evaluate_model(model, df):
    predictions = model.transform(df)

    # Calculate metrics
    results = {
        "Accuracy": multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: "accuracy"}),
        "Precision": multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: "weightedPrecision"}),
        "Recall": multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: "weightedRecall"}),
        "F1": multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: "f1"}),
        "AUC": binary_evaluator.evaluate(predictions)
    }

    return results
```

```
# Evaluate models
print("\nLogistic Regression Performance:")
lr_metrics = evaluate_model(lr_model.bestModel, test_df)
print(lr_metrics)

print("\nRandom Forest Performance:")
rf_metrics = evaluate_model(rf_model.bestModel, test_df)
print(rf_metrics)

print("\nSVM Performance:")
svm_metrics = evaluate_model(svm_model.bestModel, test_df)
print(svm_metrics)

# Compare models
import pandas as pd
metrics_comparison = pd.DataFrame({
    "Logistic Regression": lr_metrics,
    "Random Forest": rf_metrics,
    "SVM": svm_metrics
}).T

print("\nModel Comparison:")
print(metrics_comparison)
```

```
# Plot feature importance for Random Forest
rf_feature_importance = rf_model.bestModel.featureImportances.toArray()
importance_df = pd.DataFrame({
    "Feature": selected_features,
    "Importance": rf_feature_importance
}).sort_values("Importance", ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x="Importance", y="Feature", data=importance_df)
plt.title("Random Forest Feature Importance")
plt.show()

# Stop Spark session
spark.stop()
```

Coding and Implementation

Phase 5: Results Interpretation

RESULTS

```
Training Logistic Regression...

Training Random Forest...

Training SVM...

Logistic Regression Performance:
{'Accuracy': 0.8431372549019608, 'Precision': 0.8431372549019608, 'Recall': 0.8431372549019608, 'F1': 0.8431372549019608, 'AUC': 0.9124579124579123}

Random Forest Performance:
{'Accuracy': 0.8627450980392157, 'Precision': 0.86159169550173, 'Recall': 0.8627450980392157, 'F1': 0.8618086040386304, 'AUC': 0.8855218855218854}

SVM Performance:
{'Accuracy': 0.8431372549019608, 'Precision': 0.8431372549019608, 'Recall': 0.8431372549019608, 'F1': 0.8431372549019608, 'AUC': 0.9040404040404039}

Model Comparison:
                     Accuracy  Precision    Recall        F1       AUC
Logistic Regression  0.843137   0.843137  0.843137  0.843137  0.912458
Random Forest        0.862745   0.861592  0.862745  0.861809  0.885522
SVM                  0.843137   0.843137  0.843137  0.843137  0.904040
```

Model Comparison Table

Model	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*AUC*
Logistic Regression	84.31%	84.31%	84.31%	84.31%	0.912
Random Forest	86.27%	86.16%	86.27%	86.18%	0.885
SVM	84.31%	84.31%	84.31%	84.31%	0.904

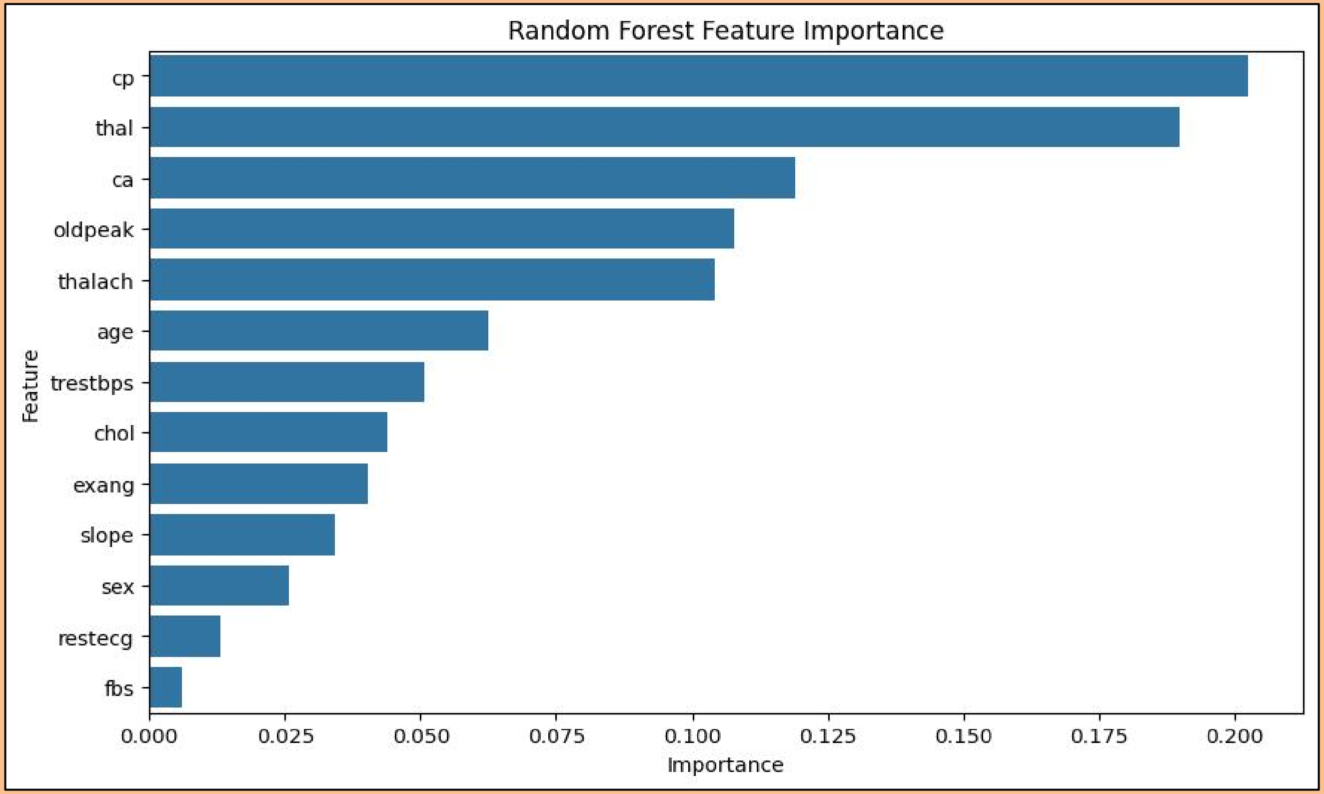
Coding and Implementation

Phase 5: Results Interpretation

RESULTS

Best Model Identified: Random Forest Classifier

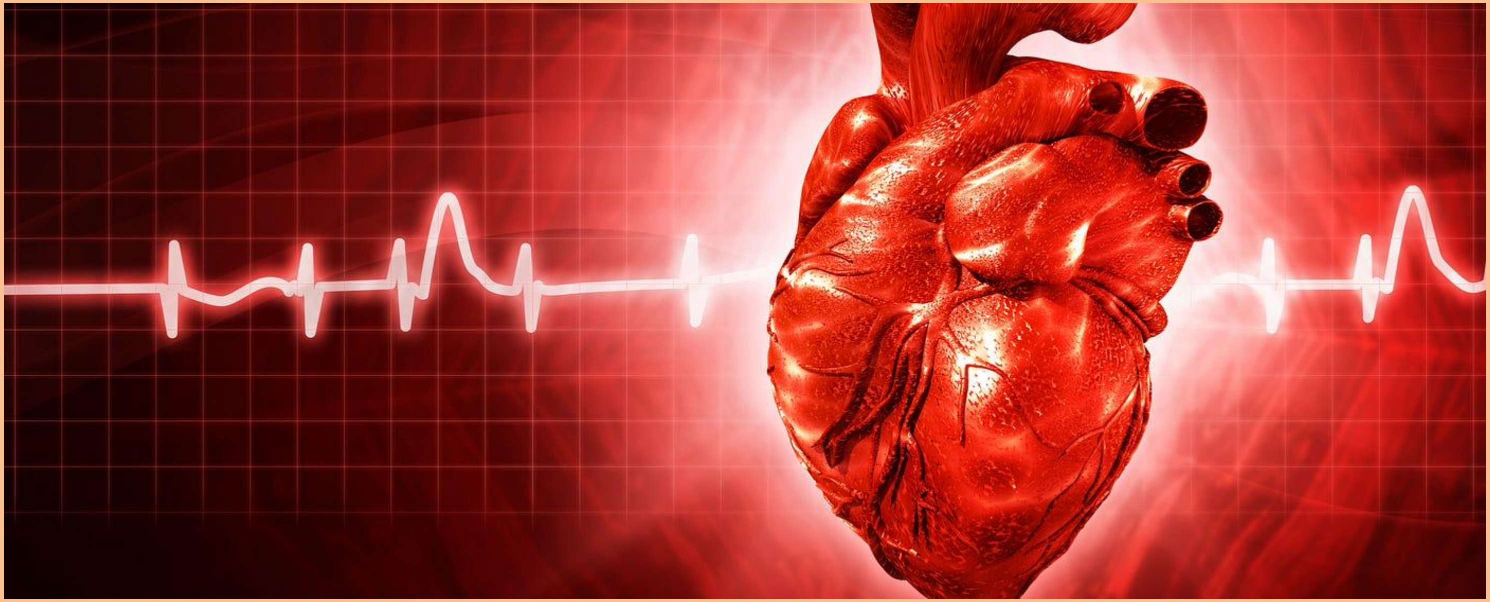
Model	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*AUC*
Random Forest	**86.27%**	**86.16%**	**86.27%**	**86.18%**	**0.885**



Contribution of each Member

Team Member	Contribution
Member 1	Data Collection & Cleaning (UCI Heart Disease Dataset)
	Data Preprocessing (Handling missing values, encoding categorical features, scaling)
	Exploratory Data Analysis (EDA) including descriptive statistics,
Member 2	Model Training
	Model Evaluation
	Result Interpretation (Comparing models, deciding on the best-performing model)
	Final Presentation Preparation

THANK YOU!



A PRESENTATION BY
DHARSHINI M 24MSP3070