# Project Title

**Predictive Modeling for Cardiovascular Disease Risk Using Machine Learning and Cloud-Based Solutions**

# Author

**Siddhant Sarnobat**

siddhantsarnobat20@gmail.com

# TABLE OF CONTENTS

# Overview

Cardiovascular diseases (CVD) continue to be a major global health concern, accounting for a significant proportion of mortality and morbidity rates worldwide. The ability to detect individuals at risk of developing CVD early can lead to timely medical intervention, improved patient outcomes, and reduced healthcare costs. Recent advancements in big data and machine learning offer transformative potential in healthcare by enabling the development of predictive models that leverage large, diverse datasets to identify at-risk individuals with high accuracy.

In this project, a publicly available dataset from Kaggle is utilized. This dataset contains a rich collection of features, including objective metrics like age, cholesterol levels, and blood pressure; examination features derived from medical tests; and subjective factors such as lifestyle habits and self-reported health conditions. By analyzing these features, the project aims to uncover patterns and insights that could assist in the prediction of CVD presence.

# Objective

The primary objective of this project is to develop an accurate and reliable machine learning model capable of predicting the likelihood of cardiovascular disease (CVD) in individuals. The goal is to create a streamlined workflow that integrates data ingestion, preprocessing, model training, and evaluation within a scalable cloud environment. By leveraging state-of-the-art tools like Azure Data Factory, Azure Data Lake, and Azure Databricks, this project seeks to enable efficient data handling, seamless model deployment, and actionable insights into cardiovascular health risks.

- Accurate Risk Classification: The project focuses on building a predictive model that can classify individuals as either at risk of CVD or not, based on a diverse set of features that include clinical metrics, examination results, and lifestyle factors.
- Seamless Data Management: Through a fully automated data pipeline, raw data from multiple sources will be transformed into a clean, analysis-ready format while maintaining security and scalability.
- Comprehensive Model Development: The project will explore a variety of machine learning algorithms, including ensemble methods, to ensure robust and high-performing predictions. By leveraging Azure's computing power, the models will be tuned for both accuracy and computational efficiency.
- Healthcare Impact: This project aims to contribute to early detection strategies, aiding healthcare providers in proactive decision-making, reducing patient risk, and ultimately improving outcomes for individuals at risk of cardiovascular disease.

By achieving these objectives, the project aims to demonstrate the power of cloud-based machine learning solutions in addressing critical healthcare challenges.

# Dataset Description

The dataset includes various features categorized as:

- **Objective Features:** Attributes such as age, height, weight, and gender.
- **Examination Features**: Clinical metrics like systolic and diastolic blood pressure, cholesterol, and glucose levels.
- **Subjective Features:** Lifestyle factors, including smoking, alcohol intake, and physical activity.

The target variable, cardio, indicates the presence or absence of cardiovascular disease, coded as binary (1 for presence, 0 for absence).

Link to dataset:
https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset

# Methodology and Tools

This project adopts a structured and systematic approach to building an end-to-end data pipeline and machine learning workflow, ensuring efficiency, scalability, and accuracy in predicting cardiovascular disease (CVD). The key components of the methodology and the tools used are detailed below:

### 1. Data Ingestion and Storage

- **Data Sources:** Data will be ingested from a variety of sources, such as on-premises databases, cloud-based systems, and APIs, to ensure diverse and comprehensive input.
- **Azure Data Factory:** This tool will be used to automate the extraction, transformation, and loading (ETL) process. It facilitates seamless data transfer from source systems into a centralized storage solution.
- **Azure Data Lake Gen 2:** The ingested data will be securely stored in Azure Data Lake, providing a scalable and high-performance storage repository. This storage solution is optimized for large-scale analytics and machine learning workflows, enabling efficient data retrieval and processing.

### 2. Data Transformation and Preprocessing

- **Cleaning and Preprocessing:**
    - Missing values will be addressed by either imputing data using statistical methods (e.g., mean or median) or removing incomplete rows if necessary.
    - Normalization will be applied to numerical features to standardize their scales, ensuring better performance during model training.
    - Categorical variables will be transformed using techniques like one-hot encoding or label encoding to make them compatible with machine learning models.
- **Azure Databricks:** The cleaning and preprocessing steps will be executed using Azure Databricks, which combines the scalability of Apache Spark with the flexibility of a notebook-based environment. This ensures efficient data transformations for large datasets.

## 3. Model Development

- **Machine Learning Framework:**
  SparkMLlib on Azure Databricks will serve as the primary framework for model training and development, offering robust and scalable machine learning libraries.

- **Predictive Models:**
  - **Logistic Regression**: A simple yet powerful linear model used for binary classification, predicting the probability of an outcome based on one or more input features. It is widely used in healthcare for risk prediction, such as for cardiovascular diseases.
  - **Support Vector Machine (SVM)**: A robust classifier that constructs decision boundaries in high-dimensional spaces, making it particularly effective for complex classification tasks where a clear margin of separation is needed.
  - **Gradient Boosting**: An ensemble learning method that builds a strong predictive model by combining multiple weak learners, usually decision trees, with a focus on correcting errors made by previous models in the sequence.
  - **Decision Tree**: A tree-like model used for both classification and regression tasks. It is popular for its interpretability, as it clearly outlines decision rules, making it suitable for healthcare predictions where understanding model decisions is critical.
  - **Naïve Bayes:** A probabilistic classifier based on Bayes' theorem, assuming independence between predictors. It is highly efficient for text classification tasks like spam detection or sentiment analysis and performs well on small datasets despite its simplicity. Its speed and effectiveness make it a go-to choice in real-time applications like email filtering and recommendation systems.

These models offer various strengths, such as interpretability, accuracy, and scalability, making them valuable for tasks like disease prediction and risk assessment.

## 4. Evaluation

- **Performance Metrics:** The models will be assessed using key metrics such as:
    - **Accuracy:** Measures overall correctness of predictions.
    - **Precision:** Evaluates the proportion of true positive predictions out of all positive predictions, indicating the model's reliability in identifying at-risk individuals.
    - **Recall:** Indicates the model's ability to correctly identify individuals who truly are at risk.
    - **F1-Score:** Balances precision and recall to give a single performance measure.
- **Comparison:** The evaluation results will guide the selection of the best-performing model, balancing accuracy and computational efficiency.

By following this comprehensive methodology, the project ensures a robust and scalable pipeline, enabling the creation of reliable machine learning models for predicting cardiovascular disease.

# Comparative Analysis of Tools

1. **Spark MLlib**

   **Description:**
   Spark MLlib is an open-source distributed machine learning library that is part of the Apache Spark ecosystem. It is designed for large-scale data processing and provides a wide range of tools for classification, regression, clustering, collaborative filtering, and more. Spark MLlib operates in a distributed environment, enabling it to handle massive datasets efficiently.

   **Advantages:**

   - Highly scalable and optimized for distributed computing, making it suitable for big data applications.
   - Seamless integration with Spark's data processing ecosystem, including Spark SQL and DataFrames.
   - Cost-effective when used with self-managed Spark clusters.
   - Supports both batch processing and streaming data through Spark Streaming.

   **Disadvantages:**

   - Requires knowledge of Spark APIs, which can have a steep learning curve for beginners.
   - Lacks some advanced features like built-in AutoML or pre-configured hyperparameter tuning.
   - Deployment requires additional setup, such as creating REST APIs or integrating with other systems.

   **Example Scenarios:**

   - Large-scale data preprocessing and machine learning pipelines in industries like finance or e-commerce.
   - Building predictive models for recommendation systems on platforms with massive user interactions.
   - Analyzing sensor data in IoT environments for predictive maintenance.

2. **H2O**

**Description:**
H2O is an open-source machine learning platform designed for distributed, large-scale data analysis. It offers a range of tools, including H2O-3 for general-purpose ML and Driverless AI for automated machine learning (AutoML). H2O integrates well with other big data frameworks like Hadoop and Spark.

**Advantages:**

- Provides user-friendly APIs in multiple languages (Python, R, Java) and supports AutoML for ease of use.
- Fast model training and built-in capabilities for hyperparameter tuning.
- Strong distributed architecture, ideal for processing large datasets.
- Open-source, making it accessible without upfront licensing costs.

**Disadvantages:**

- Although open-source, enterprise-level support and infrastructure can incur additional costs.
- Requires integration with other frameworks for a complete data pipeline.
- May not have as extensive a managed deployment ecosystem compared to cloud-native solutions like AWS SageMaker.

**Example Scenarios:**

- Developing customer churn prediction models for subscription-based businesses.
- Fraud detection systems in financial institutions, leveraging distributed capabilities.
- Using AutoML to rapidly prototype models in environments with limited data science expertise.

3. **AWS SageMaker**

**Description:**
AWS SageMaker is a fully managed machine learning service by Amazon

Web Services (AWS). It simplifies the process of building, training, and deploying machine learning models by offering pre-configured environments, AutoML capabilities, and seamless integration with other AWS services like S3, Lambda, and EC2.

**Advantages:**

- Fully managed service, eliminating the need for infrastructure setup and maintenance.
- Highly scalable with auto-scaling capabilities within the AWS ecosystem.
- Provides extensive documentation, pre-built Jupyter notebooks, and AutoML for ease of use.
- Simplifies deployment through easily accessible REST endpoints.

**Disadvantages:**

- Higher costs, especially when used at scale, with additional expenses for data storage and AWS services.
- Strong dependence on AWS infrastructure, which may lead to vendor lock-in.
- Limited offline usage and reliance on internet connectivity for operations.

**Example Scenarios:**

- Deploying machine learning models in production environments with minimal infrastructure overhead.
- Building real-time recommendation systems integrated with AWS Lambda and S3 for e-commerce platforms.
- Training deep learning models with large datasets in a scalable and optimized cloud environment.

**Summary:**

- **Spark MLlib** excels in environments with large-scale distributed data and is ideal for organizations already using Apache Spark for data processing.

- **H2O** provides a balance between scalability and ease of use, particularly beneficial for teams that value AutoML features or need distributed learning capabilities.
- **AWS SageMaker** is the best choice for those seeking a managed, scalable solution deeply integrated into the AWS ecosystem, particularly in production and real-time applications.

**Spark MLlib vs Traditional ML library (ex: scikit-learn)**

Comparing Spark MLlib and traditional machine learning libraries like scikit-learn highlights the trade-offs between big data processing and single-machine simplicity. Here's a breakdown of the pros and cons of each approach:

**Scikit-Learn (Traditional ML Library)**

Advantages:

- Ease of Use: Scikit-learn has a straightforward, intuitive API that's accessible even to beginners.
- Wide Range of Algorithms: It includes many well-optimized ML algorithms, from basic to advanced, suited for various use cases.
- Strong Community and Documentation: Extensive documentation and support from a large, active community make it easy to troubleshoot issues.
- Ideal for Prototyping: Great for quick prototyping, as it doesn't require distributed computing.

Disadvantages:

- Single-Machine Limitations: Struggles with datasets exceeding the machine's memory.
- No Native Parallel Processing: Limited multi-core support without distributed computing.
- Not Ideal for Big Data: Infeasible for terabyte-scale datasets without scaling or using Spark.

**Spark MLlib**

Advantages:

- Distributed Processing: It handles large datasets across clusters, ideal for big data.
- In-Memory Processing: Spark's in-memory processing reduces latency & boosts speed.
- Big Data Ecosystem Integration: Integrates with Spark's ecosystem for seamless data processing, storage, and ML.
- Scalability: Scales horizontally, handling datasets beyond single-machine capacity.
- Pipeline Integration: Supports end-to-end ML pipelines for data ingestion, preprocessing, and model training/testing.

Disadvantages:

- Steeper Learning Curve: Requires understanding distributed computing, making it tougher for beginners.
- Limited Algorithm Variety: Offers fewer algorithms than Scikit-learn but covers essential ones.
- Overhead for Small Data: Distributed nature adds inefficiency for smaller datasets.
- Fewer Interpretability Tools: Lacks as many model interpretation tools as Scikit-learn (e.g., feature importance).

# Flow of Work

## I. GitHub

**Purpose**: To use GitHub as a data source.

1. Create a GitHub public repository.
2. Upload the dataset file to the repository.
3. Get the "raw" URL of the dataset file.

## II. Azure Cloud

### 1. Resource Group

**Purpose**: Simplify resource management by grouping all related resources.

1. Create a resource group (e.g., big_data_project) and attach every Azure service/resource to this group.

### 2. Storage Account

**Purpose**: To store data securely in Azure cloud.

1. Create a storage account (e.g., cardiodisease), enabling the hierarchical storage option.
2. Create a container (e.g., dataset) to hold your data.
3. Create a directory inside the container (e.g., cardio_data) to store the dataset.

### 3. Data Factory (for data ingestion)

**Purpose**: Ingest and transfer data from GitHub to Azure storage.

1. Create a Data Factory resource (e.g., dataset-ingestion).
2. Go to "Author" → Create pipeline → Select "Move and Transform" → Drag "Copy data".
3. Name the pipeline (e.g., data_ingestion_pipeline).
4. Add source to the pipeline: Select new source → HTTP → Choose file format (CSV).

5. Add sink to the pipeline: Choose Azure Data Lake Storage Gen2 → Select file format (CSV).
6. Validate and debug the pipeline to ensure it’s working correctly.

## 4. App Registration (to give Databricks access)

**Purpose**: Allow Databricks to access your Azure Storage account.

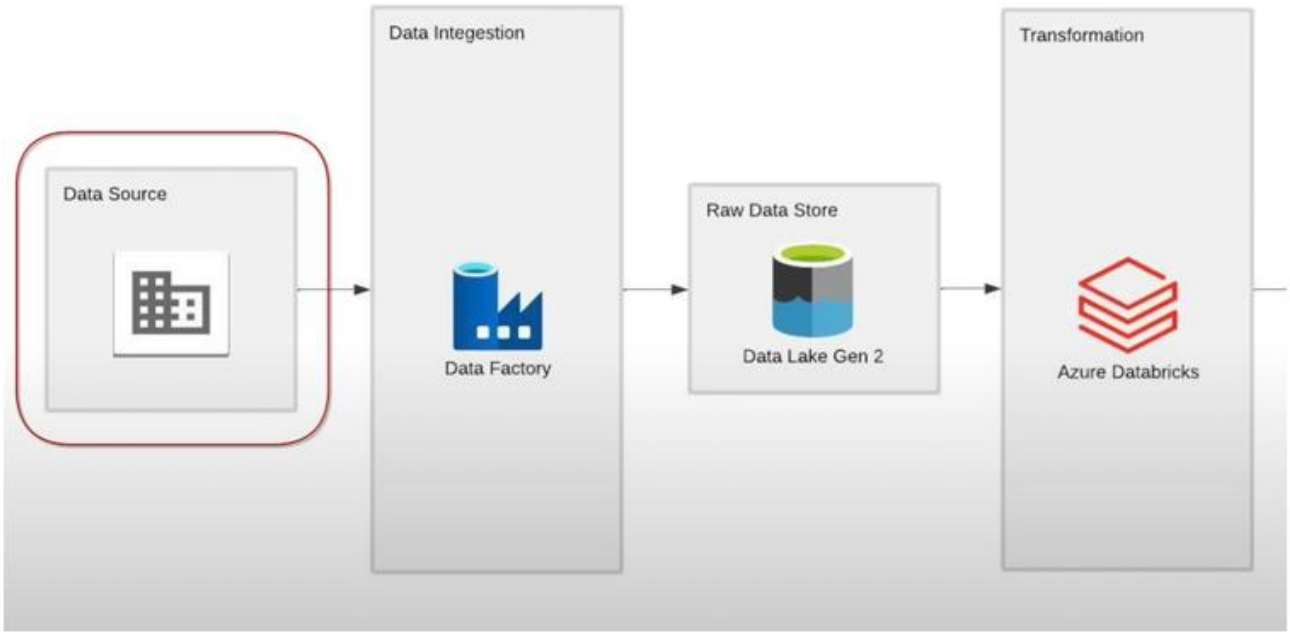
1. Search for App Registration in Azure and name it (e.g., App_cardio_disease).
2. Copy the "Application (client) ID" and "Directory (tenant) ID".
3. Go to "Certificates & Secrets" → Create new secret → Copy the value.

## 5. Databricks (for machine learning)

**Purpose**: To run machine learning models using Spark MLlib in a distributed environment.

1. Create a Databricks resource (e.g., databricks_cardiodisease).
2. Create a new cluster and give it a name.
3. Create a new notebook and write the code for machine learning models.

## Flow Diagram




# Implementation (Code)

## Exploratory Data Analysis

### Inspect the Data

```
df.printSchema()
df.show(5)
```

```
 |-- weight: double (nullable = true)
 |-- ap_hi: integer (nullable = true)
 |-- ap_lo: integer (nullable = true)
 |-- cholesterol: integer (nullable = true)
 |-- gluc: integer (nullable = true)
 |-- smoke: integer (nullable = true)
 |-- alco: integer (nullable = true)
 |-- active: integer (nullable = true)
 |-- cardio: integer (nullable = true)

+---+-----+------+------+------+-----+-----+-----------+----+-----+----+------+------+
| id|  age|gender|height|weight|ap_hi|ap_lo|cholesterol|gluc|smoke|alco|active|cardio|
+---+-----+------+------+------+-----+-----+-----------+----+-----+----+------+------+
|  0|18393|     2|   168|  62.0|  110|   80|          1|   1|    0|   0|     1|     0|
|  1|20228|     1|   156|  85.0|  140|   90|          3|   1|    0|   0|     1|     1|
|  2|18857|     1|   165|  64.0|  130|   70|          3|   1|    0|   0|     0|     1|
|  3|17623|     2|   169|  82.0|  150|  100|          1|   1|    0|   0|     1|     1|
|  4|17474|     1|   156|  56.0|  100|   60|          1|   1|    0|   0|     0|     0|
+---+-----+------+------+------+-----+-----+-----------+----+-----+----+------+------+
only showing top 5 rows
```

```
# Check the number of rows and columns
print(f"Rows: {df.count()}, Columns: {len(df.columns)}")
```

```
Rows: 70000, Columns: 13
```

## Profiling Numeric Columns

```
from pyspark.mllib.stat import Statistics

# List of numeric columns
numeric_cols = ["age", "height", "weight", "ap_hi", "ap_lo"]

# Convert numeric columns into an RDD for MLlib
numeric_rdd = df.select(numeric_cols).rdd.map(lambda row: [float(x) for x in
row])

# Compute summary statistics
summary = Statistics.colStats(numeric_rdd)
```

```
# Display statistics for each column
for idx, col_name in enumerate(numeric_cols):
    print(f"{col_name}:")
    print(f"  Min: {summary.min()[idx]:.2f}")
    print(f"  Max: {summary.max()[idx]:.2f}")
    print(f"  Mean: {summary.mean()[idx]:.2f}")
    print(f"  Std Dev: {summary.variance()[idx] ** 0.5:.2f}")
```

```
age:
  Min: 10798.00   Max: 23713.00     Mean: 19468.87  Std Dev: 2467.25
height:
  Min: 55.00            Max: 250.00       Mean: 164.36          Std Dev: 8.21
weight:
  Min: 10.00            Max: 200.00       Mean: 74.21     Std Dev: 14.40
ap_hi:
  Min: -150.00    Max: 16020.00     Mean: 128.82          Std Dev: 154.01
ap_lo:
  Min: -70.00     Max: 11000.00     Mean: 96.63     Std Dev: 188.47
```

The age column have abnormal values and doesn't make sense, so we drop it

```
# Dropping the column 'age'
df = df.drop('age')

# Show the updated DataFrame
df.show(5)
```

```
+---+------+------+------+-----+-----+-----------+----+-----+----+------+------+
| id|gender|height|weight|ap_hi|ap_lo|cholesterol|gluc|smoke|alco|active|cardio|
+---+------+------+------+-----+-----+-----------+----+-----+----+------+------+
|  0|     2|   168|  62.0|  110|   80|          1|   1|    0|   0|     1|     0|
|  1|     1|   156|  85.0|  140|   90|          3|   1|    0|   0|     1|     1|
|  2|     1|   165|  64.0|  130|   70|          3|   1|    0|   0|     0|     1|
|  3|     2|   169|  82.0|  150|  100|          1|   1|    0|   0|     1|     1|
|  4|     1|   156|  56.0|  100|   60|          1|   1|    0|   0|     0|     0|
+---+------+------+------+-----+-----+-----------+----+-----+----+------+------+
only showing top 5 rows
```

## Checking for Missing Values

```
from pyspark.sql.functions import col, when, isnan, count
```

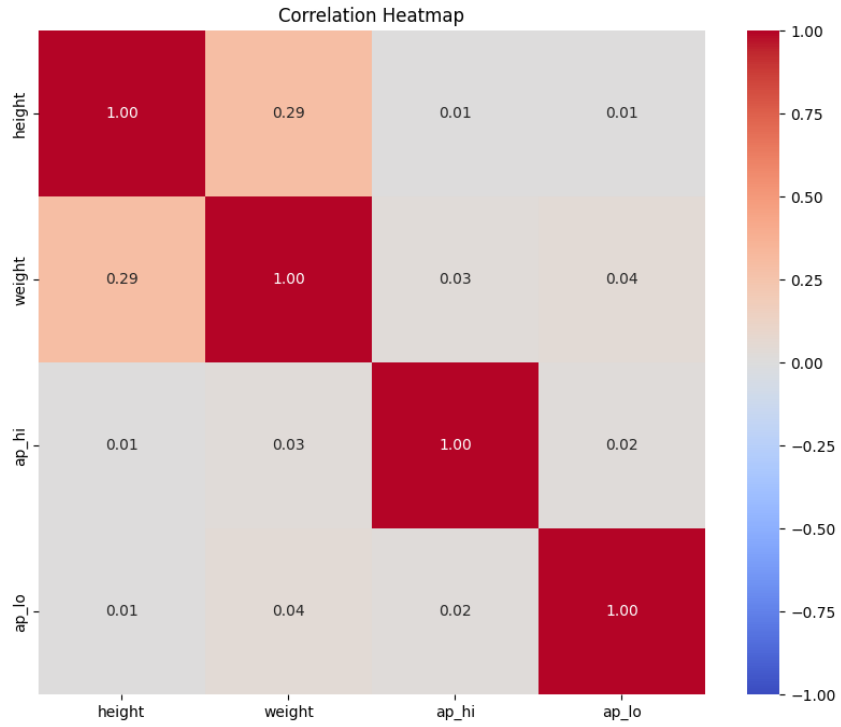
```
# Count nulls for all columns
missing_values = df.select([(count(when(col(c).isNull() | isnan(c), c))).alias(c) for c in df.columns])
missing_values.show()
```

```
+---+------+------+------+-----+-----+-----------+----+-----+----+------+------+
| id|gender|height|weight|ap_hi|ap_lo|cholesterol|gluc|smoke|alco|active|cardio|
+---+------+------+------+-----+-----+-----------+----+-----+----+------+------+
|  0|     0|     0|     0|    0|    0|          0|   0|    0|   0|     0|     0|
+---+------+------+------+-----+-----+-----------+----+-----+----+------+------+
```

## Correlation Heatmap

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.mllib.stat import Statistics

# List of numeric columns
numeric_cols = ["height", "weight", "ap_hi", "ap_lo"]

# Convert numeric columns into an RDD for MLlib
numeric_rdd = df.select(numeric_cols).rdd.map(lambda row: [float(x) for x in row])

# Compute the correlation matrix
correlation_matrix = Statistics.corr(numeric_rdd, method="pearson")

# Convert the correlation matrix to a Pandas DataFrame for easier visualization
correlation_df = pd.DataFrame(correlation_matrix, columns=numeric_cols, index=numeric_cols)

# Display the correlation matrix in the console
print("Correlation Matrix:")
for i, col1 in enumerate(numeric_cols):
    for j, col2 in enumerate(numeric_cols):
        print(f"Correlation({col1}, {col2}): {correlation_matrix[i][j]:.2f}")

# Plot the heatmap using seaborn
```

```
plt.figure(figsize=(10, 8))  # Adjust the size of the heatmap
sns.heatmap(correlation_df, annot=True, cmap="coolwarm", fmt=".2f", vmin=-1,
vmax=1)
plt.title("Correlation Heatmap")
plt.show()
```




We have analysed the columns and removed the outliers before proceeding to build the ML model

# Data Preprocessing

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.sql.functions import col

# Convert the DataFrame into LabeledPoint RDD
def transform_to_labeled_point(row):
    return LabeledPoint(row['cardio'], [
        row['gender'], row['height'], row['weight'],
        row['ap_hi'], row['ap_lo'], row['cholesterol'], row['gluc'],
        row['smoke'], row['alco'], row['active']
```

```
    ])
labeled_data = df_no_outliers.rdd.map(transform_to_labeled_point)
# Split into training and test sets
train_data, test_data = labeled_data.randomSplit([0.8, 0.2], seed=42)
```

# ML Models
# Decision Tree;

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree
from pyspark.mllib.evaluation import MulticlassMetrics

#  Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainClassifier(train_data, numClasses=2,
categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(test_data.map(lambda x: x.features))
labelsAndPredictions = test_data.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(test_data.count())
print('Test Error = ' + str(testErr))
print('Learned classification tree model:')
print(model.toDebugString())
```

## Output

```
    If (feature 4 <= 89.5)
     If (feature 8 <= 0.5)
      Predict: 1.0
     Else (feature 8 > 0.5)
      Predict: 0.0
    Else (feature 4 > 89.5)
     Predict: 1.0
   Else (feature 5 > 2.5)
    Predict: 1.0
  Else (feature 3 > 138.5)
   If (feature 3 <= 147.5)
    If (feature 6 <= 2.5)
     Predict: 1.0
    Else (feature 6 > 2.5)
     If (feature 4 <= 69.5)
      Predict: 0.0
     Else (feature 4 > 69.5)
      Predict: 1.0
   Else (feature 3 > 147.5)
    Predict: 1.0
```

# SVM

```
from pyspark.mllib.classification import SVMWithSGD
# SVM
model_svm = SVMWithSGD.train(train_data, 100)
labelsAndPreds_svm = test_data.map(lambda p: (p.label,
model_svm.predict(p.features)))
testErr_svm = labelsAndPreds_svm.filter(lambda lp: lp[0] != lp[1]).count() /
float(test_data.count())
print("SVM Test Error = " + str(testErr_svm))
```

## Output

```
SVM Test Error = 0.4867685426761088
```

# Gradient Boosting

```
from pyspark.mllib.tree import GradientBoostedTrees,
GradientBoostedTreesModel
```

```
# Gradient Boosting
model_gb = GradientBoostedTrees.trainClassifier(train_data,
                                                categoricalFeaturesInfo={},
numIterations=100)
predictions_gb = model_gb.predict(test_data.map(lambda x: x.features))
labelsAndPredictions_gb = test_data.map(lambda lp:
lp.label).zip(predictions_gb)
testErr_gb = labelsAndPredictions_gb.filter(lambda lp: lp[0] !=
lp[1]).count() / float(test_data.count())
print('Gradient Boosting Test Error = ' + str(testErr_gb))
```

### Output

```
Gradient Boosting Test Error = 0.2746179649645919
```

## Naive Bayes

```
# Train the Naive Bayes model
from pyspark.mllib.classification import NaiveBayes
model_nb = NaiveBayes.train(train_data, lambda_=1.0)  # Adjust lambda_ as
needed
# Predict on test data
predictions = model_nb.predict(test_data.map(lambda x: x.features))
# Evaluate model
labels_and_predictions = test_data.map(lambda lp: lp.label).zip(predictions)
accuracy = labels_and_predictions.filter(lambda x: x[0] == x[1]).count() /
float(test_data.count())
print(f"Accuracy: {accuracy}")
```

### Output

```
Accuracy: 0.6074996272551066
```

# Logistic Regression

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS

# Train Logistic Regression model
model_lr = LogisticRegressionWithLBFGS.train(train_data)

# Predict using the model
labelsAndPreds_lr = test_data.map(lambda p: (p.label,
model_lr.predict(p.features)))

# Calculate test error
testErr_lr = labelsAndPreds_lr.filter(lambda lp: lp[0] != lp[1]).count() /
float(test_data.count())

# Print test error
print("Logistic Regression Test Error = " + str(testErr_lr))
```

## Output

```
Logistic Regression Test Error = 0.28706671636228104
```

# Results and Analysis

**Support Vector Machine**

| Accuracy | Test Error Rate | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 51.32 % | 48.68 % | 64.86 % | 51.32 % | 37.73 % |

**Logistic Regression**

| Accuracy | Test Error Rate | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 71.29 % | 28.71 % | 71.65 % | 71.29 % | 71.41 % |

**Naive Bayes**

| Accuracy | Test Error Rate | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 71.51 % | 28.49 % | 71.69 % | 71.51 % | 71.36 % |

**Decision Trees**

| Accuracy | Test Error Rate | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 72.08 % | 27.92 % | 72.41 % | 72.08 % | 71.94 % |

**Gradient Boosting Trees**

| Accuracy | Test Error Rate | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 72.54 % | 27.46 % | 73.02 % | 72.54 % | 72.36 % |

Overall, the Gradient Boosting Trees model outperformed others with the highest accuracy (72.54%) and F1 score (72.36%), followed closely by Decision Trees, Naive Bayes, and Logistic Regression, all achieving similar metrics around 71-72%.

SVM lagged significantly, with the lowest accuracy (51.32%) and F1 score (37.73%), indicating poor performance compared to other models. Gradient Boosting Trees show strong precision and generalization, making it the most effective choice for this dataset.

# Conclusion

By automating the data pipeline and utilizing cloud-based machine learning tools on Databricks, this project develops a robust suite of predictive models to assess the risk of cardiovascular disease (CVD). The focus on scalable and interpretable algorithms ensures that the models are both efficient and comprehensible for real-world healthcare applications.

The selected algorithms—Support Vector Machine (SVM), Naive Bayes, Logistic Regression, Decision Trees, and Gradient Boosting Trees—were evaluated for accuracy, with their respective performances at 51.33%, 60.75%, 71.3%, 72.08%, and 72.54%. Gradient Boosting Trees emerged as the most accurate model, leveraging its ensemble learning approach to handle complex, non-linear relationships in the data effectively. Logistic Regression demonstrated strong baseline performance with straightforward interpretability, making it ideal for scenarios requiring clear risk stratification. Decision Trees provided interpretable results with comparable accuracy, while SVM and Naive Bayes offered insights into smaller subsets and probabilistic classification. The ultimate goal is to equip healthcare providers with a reliable, data-driven tool for the early detection of CVD. This empowers proactive and targeted interventions, potentially reducing the burden of the disease through improved patient outcomes and optimized resource allocation. The combination of scalable technologies and carefully chosen models creates a pathway for impactful, real-world deployment in modern healthcare systems.

# Future Scope

Future exploration for this project could focus on enhancing model robustness and generalizability by incorporating additional features, such as genetic data or lifestyle factors, to improve prediction accuracy. Employing advanced techniques like deep learning models, particularly for large and unstructured datasets, could also be investigated. Integrating privacy-preserving machine learning methods, such as federated learning or differential privacy, can ensure compliance with data protection regulations while enabling collaboration across healthcare institutions. Additionally, deploying the predictive models in real-time clinical settings and evaluating their impact on patient outcomes could provide valuable insights into practical applications and scalability.

# References


1. World Health Organization. (2021). Cardiovascular diseases (CVDs). [link]
2. Meng, X., Bradley, J., Yuvaz, B., Sparks, E., Venkataraman, S., Liu, D., & Zaharia, M. (2016). *MLlib: Machine Learning in Apache Spark.* Journal of Machine Learning Research, 17(34), 1-7.
3. Ghodsi, A., Zaharia, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). *Databricks: A Cloud Platform for Massive-Scale Data Engineering and Collaborative Analytics.* Proceedings of the VLDB Endowment, 11(12), 2249-2261.
4. M. Assefi, E. Behravesh, G. Liu and A. P. Tafti, "Big data machine learning using apache spark MLlib," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 3492-3498, doi: 10.1109/BigData.2017.8258338.
5. Zaharia, M., et al. "Apache Spark: a unified engine for big data processing." *Communications of the ACM*, vol. 59, no. 11, 2016, pp. 56-65. https://doi.org/10.1145/2934664
6. Jorge Gonzalez-Lopez, Sebastián Ventura, Alberto Cano, Distributed nearest neighbor classification for large-scale multi-label data on spark, Future Generation Computer Systems, Volume 87, 2018, Pages 66-82, ISSN 0167-739X, https://doi.org/10.1016/j.future.2018.04.094.
7. Microsoft Corporation. (2023). *Azure Data Lake Storage Gen2: A Hyper-Scale Big Data Storage Service.* Retrieved from https://azure.microsoft.com
8. Venkata, Ramana, Reddy, Bussu. (2024). Databricks- Data Intelligence Platform for Advanced Data Architecture. International journal of innovative science and research technology, doi: 10.38124/ijisrt/ijisrt24apr166
9. NMNH, Division, of, Mammals. (2021). 4. Databricks. doi: 10.1007/978-1-4842-8233-5_3
10. Keith, Gutfreund., Elsevier, Labs,, Cambridge,, Ma. (2017). Big Data Techniques for Predictive Business Intelligence. Journal of Advanced Management Science, 5(2):158-163. doi: 10.18178/JOAMS.5.2.158-163