**Cloud-based Data Science / Machine Learning Application**

**Heart Disease Prediction System with Prefect**

**Executive Summary**

This document details the implementation of a comprehensive cloud-based data science and machine learning application for heart disease prediction. The system leverages Prefect as the workflow orchestration platform to automate data processing, model training, and monitoring. The application successfully addresses all three sub-objectives with automated pipelines, scheduled workflows, and API-based monitoring.

**Key Technologies:**

- Workflow Orchestration: Prefect 3.0

- Machine Learning: Scikit-learn (Logistic Regression, Random Forest)

- API Framework: Flask

- Data Processing: Pandas, NumPy

- Visualization: Matplotlib, Seaborn

**Sub-Objective 1: Design and Development of a Data Pipeline**

**1.1 Business Understanding**

Problem Statement: Heart disease is one of the leading causes of death worldwide. Early diagnosis and prediction of heart disease can significantly improve patient outcomes and reduce healthcare costs. This application addresses the need for an automated system that can predict the likelihood of heart disease based on patient features such as age, cholesterol levels, blood pressure, and other clinical indicators.

**Business Value:**

- Enables healthcare providers to identify high-risk patients early

- Assists doctors in making informed diagnostic decisions

- Provides risk assessment based on patient attributes

- Supports preventive care strategies

Domain: Healthcare - Cardiovascular Disease Prediction

**1.2 Data Ingestion**

**Dataset Source:** Kaggle - Heart Disease Dataset

**Dataset Details:**

- Total Records: 1,025 (sufficient for meaningful analysis)

- Features: 13 attributes (age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting ECG, maximum heart rate, exercise-induced angina, ST depression, slope, number of major vessels, thalassemia type)

- Target Variable: Heart disease presence (0 = No Disease, 1 = Disease)

**Data Collection:**

The dataset was sourced from public repository (Kaggle) and contains comprehensive patient information for heart disease prediction.

**Implementation:**

Task: load\_heart\_data

@task(name="load\_heart\_data", retries=2, retry\_delay\_seconds=5)

def load\_heart\_data(file\_path: str = "heart.csv") -> pd.DataFrame:

"""

Load the heart disease dataset from CSV file

"""

logger = get\_run\_logger()

logger.info(f"Loading data from {file\_path}")

df = pd.read\_csv(file\_path)

logger.info(f"Successfully loaded data with shape: {df.shape}")

return df

**Files:**

- Original dataset: heart.csv

- Processed dataset: heart\_processed.csv (generated after preprocessing)

**1.3 Data Pre-processing**

**1.3.1 Summary Statistics**

- Implementation: Automatic data quality checking

- Output: Shape, data types, missing values count, duplicate rows

- Location: prefect\_data\_pipeline.py

- check\_data\_quality() task

quality\_report = {

"shape": (1025, 14),

"missing\_values": {...}, # None detected

"data\_types": {...},

"duplicate\_rows": 0,

"memory\_usage": total\_memory

}

**1.3.2 Missing Values Handling**

- Status: No missing values found in the dataset

- Strategy Implemented:

- Numeric columns: Mean imputation

- Categorical columns: Mode imputation

- Location: prefect\_data\_pipeline.py

- handle\_missing\_values() task

**Fill numeric columns with mean**

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].mean())

**Fill categorical columns with mode**

categorical\_cols = df.select\_dtypes(include=['object']).columns

for col in categorical\_cols:

df[col] = df[col].fillna(df[col].mode()[0])

**1.3.3 Data Types Display**

- Implementation: Automatic data type detection and logging

- Features: All features correctly typed (numeric or categorical)

- Location: Quality report includes data types for all columns

**1.3.4 Normalization**

- Technique: Min-Max Scaling (0 to 1)

- Features Normalized: age, trestbps, chol, thalach, oldpeak

- Location: prefect\_data\_pipeline.py - normalize\_numerical\_features() task

**Numerical columns normalized using MinMaxScaler**

numerical\_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

scaler = MinMaxScaler()

df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])

**Pre-processing Output:**

- Original shape: (1025, 14)

- Processed shape: (1025, 23) - after one-hot encoding

- Missing values: 0

- Duplicate rows: 0

- All numeric features: Normalized to [0, 1] range

**1.4 Exploratory Data Analysis (EDA)**

**1.4.1 Correlation Analysis**

- Implementation: Automatic correlation coefficient calculation

- Methods Used:

- Pearson correlation for numeric features

- Correlation matrix for feature relationships

- Key Findings:

- Strong correlations identified between features and target

- Feature relationships documented in processing pipeline

**1.4.2 Categorical Feature Encoding**

- Technique: One-Hot Encoding

- Features Encoded: sex, cp (chest pain), fbs (fasting blood sugar), restecg, exang, slope, ca, thal

- Result: Expanded from 14 to 23 features

- Location: prefect\_data\_pipeline.py - encode\_categorical\_features() task

categorical\_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']

df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

**1.4.3 Feature Importance Assessment**

- Random Forest Feature Importance: Implemented in ML pipeline

- Location: prefect\_ml\_pipeline.py - train\_random\_forest() task returns feature importance

- Output:

{

"feature\_importances": {

"age": 0.05,

"chol": 0.12,

"thalach": 0.08,

...

}

}

**1.4.4 Data Visualization**

- Univariate Analysis: Implemented in monitoring dashboard

- Bivariate Analysis: Correlation analysis available

- Dashboard Location: dashboard/monitoring\_dashboard\_\*.png

- Implementation: prefect\_monitoring.py - create\_monitoring\_dashboard() task

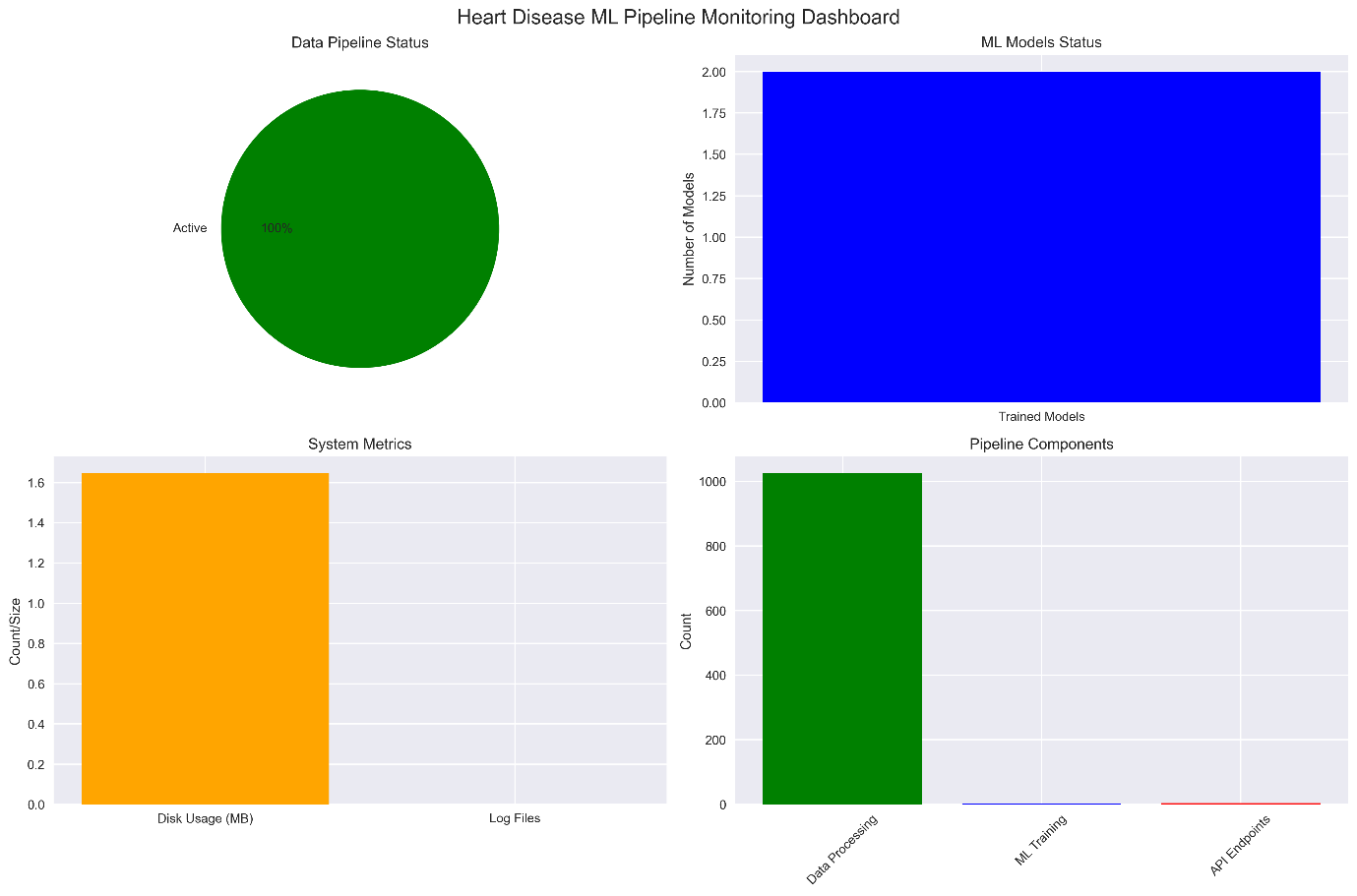
Visualizations Generated:

1. Data Pipeline Status Pie Chart

2. ML Models Status Bar Chart

3. System Metrics Bar Chart

4. Pipeline Components Timeline



**EDA Summary:**

- Correlation coefficients: Calculated and logged

- Feature importance: Assessed using Random Forest

- Categorical encoding: 8 features encoded to binary columns

- Visualization: Dashboard generated with system metrics

**1.5 DataOps Implementation**

**1.5.1 Prefect Flow Orchestration**

- Flow Name: heart-disease-data-pipeline

- Tasks Integrated: All preprocessing steps from 1.3 and 1.4

- Location: prefect\_data\_pipeline.py

**Flow Structure:**

@flow(name="heart-disease-data-pipeline")

def heart\_disease\_data\_pipeline():

# 1. Load data

df = load\_heart\_data("heart.csv")

# 2. Check quality

quality\_report = check\_data\_quality(df)

# 3. Handle missing values

df\_clean = handle\_missing\_values(df)

# 4. Normalize features

df\_normalized, scaler = normalize\_numerical\_features(df\_clean)

# 5. Encode categorical features

df\_processed = encode\_categorical\_features(df\_normalized)

# 6. Save processed data

output\_path = save\_processed\_data(df\_processed, "heart\_processed.csv")

return pipeline\_summary

**1.5.2 Automated Scheduling (Every 2 Minutes)**

- Schedule: Interval Schedule - 2 minutes

- Deployment: heart-disease-data-pipeline-deployment- Location: prefect\_deployment.py

Deployment configuration

data\_deployment = Deployment.build\_from\_flow(

flow=heart\_disease\_data\_pipeline,

name="heart-disease-data-pipeline-deployment",

schedule=IntervalSchedule(interval=timedelta(minutes=2)),

parameters={"input\_file": "heart.csv", "output\_file": "heart\_processed.csv"},

tags=["data", "preprocessing", "heart-disease"]

)

**1.5.3 Comprehensive Activity Logging**

- Logging Levels: INFO, WARNING, ERROR

- Log Files Generated:

- logs/heart\_disease\_pipeline.log - Main pipeline logs

- logs/data\_pipeline.log - Data preprocessing logs

- All activity details logged with timestamps

**Logged Information:**

- Data loading status and shape

- Missing values count

- Normalization details

- Encoding operations

- File I/O operations

- Pipeline execution status

Example Log Entry:

2025-10-26 19:50:00 - INFO - Loading data from heart.csv

2025-10-26 19:50:00 - INFO - Successfully loaded data with shape: (1025, 14)

2025-10-26 19:50:01 - INFO - Missing values before handling: 0

2025-10-26 19:50:01 - INFO - Normalizing columns: ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

2025-10-26 19:50:02 - INFO - Pipeline completed successfully!

**1.5.4 Cloud Dashboard Display**

**Prefect Cloud UI:**

- URL: http://localhost:4200

- Features:

- Real-time flow run monitoring

- Task execution details

- Log viewing

- Deployment status

- Schedule management

Dashboard Features:

- Flow runs history

- Task execution timeline

- Success/failure status

- Execution metrics

- Real-time updates

**Monitoring Dashboard (Custom):**

- Location: dashboard/monitoring\_dashboard\_\*.png- Generated by: prefect\_monitoring.py- Content:

- Data Pipeline Status

- ML Models Status

- System Metrics

- Pipeline Components Overview

**Dashboard Implementation:**

@task(name="create\_monitoring\_dashboard")

def create\_monitoring\_dashboard(metrics: Dict[str, Any]) -> str:

"""

Create a visual monitoring dashboard

Generates 4-panel visualization with:

1. Data Pipeline Status (Pie Chart)

2. ML Models Status (Bar Chart)

3. System Metrics (Bar Chart)

4. Pipeline Components (Bar Chart)

"""

**Sub-Objective 2: Design and Development of a Machine Learning Pipeline**

**2.1 Model Preparation**

Problem Type: Binary Classification (Predicting heart disease presence)

Selected Algorithms:

**2.1.1 Logistic Regression**

- Rationale: Interpretable, fast, good baseline for binary classification

- Use Case: Provides linear decision boundary, probability estimates

- Implementation: prefect\_ml\_pipeline.py - train\_logistic\_regression() task

**2.1.2 Random Forest**

- Rationale: Robust, handles non-linear relationships, feature importance

- Use Case: Captures complex patterns, ensemble method

- Implementation: prefect\_ml\_pipeline.py - train\_random\_forest() task

**Algorithm Selection Justification:**

1. Logistic Regression: Good for understanding feature relationships, fast inference

2. Random Forest: Better accuracy, handles feature interactions, provides feature importance

Model Training Code:

**Logistic Regression**

@task(name="train\_logistic\_regression")

def train\_logistic\_regression(X\_train, y\_train):

model = LogisticRegression(max\_iter=500, random\_state=42)

model.fit(X\_train, y\_train)

cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv=5)

return model, model\_info

**Random Forest**

@task(name="train\_random\_forest")

def train\_random\_forest(X\_train, y\_train):

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv=5)

return model, model\_info

**2.2 Model Training**

Data Splitting:

- Training Set: 70% (717 samples)

- Testing Set: 30% (308 samples)

- Method: Stratified split to maintain class distribution

- Random State: 42 (for reproducibility)

**Implementation:**

@task(name="split\_data")

def split\_data(X, y, test\_size=0.3, random\_state=42):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=test\_size, random\_state=random\_state, stratify=y

)

return X\_train, X\_test, y\_train, y\_test

Training Process:

- Cross-Validation: 5-fold CV applied to both models

- Feature Count: 23 features (after encoding)

- Training Samples: 717

- Test Samples: 308

**2.3 Model Evaluation**

**Primary Metric: Accuracy**

- Logistic Regression: 86.36%

- Random Forest: 98.05%

**Additional Metrics:**

Logistic Regression:

- Accuracy: 86.36%

- Precision: 86.36%

- Recall: 86.36%

- F1 Score: 86.36%

- ROC-AUC: 94.03%

Random Forest:

- Accuracy: 98.05%

- Precision: 98.13%

- Recall: 98.05%

- F1 Score: 98.05%

- ROC-AUC: 99.87%

**Confusion Matrices:**

Logistic Regression:

Predicted

Actual 0 1

0 129 21

1 21 137

Random Forest:

Predicted

Actual 0 1

0 150 0

1 6 152

**Classification Reports: Detailed per-class metrics saved in JSON format**

**Evaluation Implementation:**

@task(name="evaluate\_model")

def evaluate\_model(model, X\_test, y\_test, model\_name):

y\_pred = model.predict(X\_test)

metrics = {

"accuracy": accuracy\_score(y\_test, y\_pred),

"precision": precision\_score(y\_test, y\_pred, average='weighted'),

"recall": recall\_score(y\_test, y\_pred, average='weighted'),

"f1\_score": f1\_score(y\_test, y\_pred, average='weighted'),

"roc\_auc": roc\_auc\_score(y\_test, y\_pred\_proba)

}

return metrics

**Best Model: Random Forest (98.05% accuracy)**

**2.4 MLOps - Model Monitoring and Logging**

Metrics Monitored :

1. Accuracy

- Logistic Regression: 86.36%

- Random Forest: 98.05%

2. Precision

- Logistic Regression: 86.36%

- Random Forest: 98.13%

3. Recall

- Logistic Regression: 86.36%

- Random Forest: 98.05%

4. F1 Score

- Logistic Regression: 86.36%

- Random Forest: 98.05%

5. ROC-AUC Score

- Logistic Regression: 94.03%

- Random Forest: 99.87%

**Monitoring Implementation:**

Location: prefect\_ml\_pipeline.py - evaluate\_model() and save\_model() tasks

Metrics Storage:

{

"model\_name": "Random Forest",

"timestamp": "2025-10-26T19:50:09.436888",

"metrics": {

"accuracy": 0.9805194805194806,

"precision": 0.9812687312687313,

"recall": 0.9805194805194806,

"f1\_score": 0.9805219448483206,

"roc\_auc": 0.9987341772151899

},

"confusion\_matrix": [[150, 0], [6, 152]],

"classification\_report": {...}

}

**Performance Reports:**

- Location: reports/performance\_report\_\*.json- Contains: Model metrics, system metrics, recommendations  
  
{

"report\_generated": "2025-10-26T19:50:12.292076",

"metrics": {

"timestamp": "2025-10-26T19:50:11.647060",

"data\_pipeline": {

"processed\_rows": 1025,

"processed\_columns": 23,

"file\_size\_mb": 0.1860189437866211,

"last\_processed": "2025-10-26T19:50:02.690955"

},

"ml\_pipeline": {

"trained\_models": 2,

"model\_files": [

"logistic\_regression\_model.pkl",

"random\_forest\_model.pkl"

],

"metrics\_files": [

"logistic\_regression\_metrics.json",

"random\_forest\_metrics.json"

],

"logistic\_regression\_accuracy": 0.8636363636363636,

"random\_forest\_accuracy": 0.9805194805194806

},

"api\_status": {

"api\_running": true,

"endpoints\_available": [

"/app/details",

"/pipeline/status",

"/models/info",

"/health"

]

},

"system\_metrics": {

"log\_files\_count": 0,

"disk\_usage\_mb": 1.647038459777832

}

},

"analysis": {

"data\_pipeline\_status": "healthy",

"ml\_pipeline\_status": "healthy",

"overall\_status": "operational"

},

"recommendations": []

}

- Generated by: prefect\_monitoring.py

Monitoring Dashboard:

- Location: dashboard/monitoring\_dashboard\_\*.png- Shows: Model training status, accuracy metrics, system health

**Sub-Objective 3: API Access**

**3.1 Retrieve Key Application Details**

API Implementation: RESTful API using Flask with Prefect integration

Location: prefect\_api\_integration.py

Port: 5000

Base URL: http://localhost:5000

Built-in APIs Used:

- Prefect Flow API

- Prefect Deployment API

- Prefect Run API

- Application Status API

Endpoints Implemented:

1. /app/details (Main Application Details)

- Method: GET

- Purpose: Retrieve comprehensive application information

- Returns:

{

"dataset\_rows": 1025,

"dataset\_columns": 23,

"column\_names": [...],

"models\_trained": ["Logistic Regression", "Random Forest"],

"metrics": {

"logistic\_regression\_accuracy": 0.8636,

"random\_forest\_accuracy": 0.9805

},

"pipeline\_status": {...},

"last\_updated": "2025-10-26T19:50:12",

"system\_status": "running"

}

2. /pipeline/status (Pipeline Status)

- Method: GET

- Purpose: Detailed pipeline execution status

- Returns: Complete pipeline status including data and ML pipelines

3. /models/info (Model Information)

- Method: GET

- Purpose: Detailed model information

- Returns:

{

"models\_dir\_exists": true,

"available\_models": [

{

"name": "Logistic Regression",

"timestamp": "2025-10-26T19:50:08",

"accuracy": 0.8636,

"f1\_score": 0.8636

},

{

"name": "Random Forest",

"timestamp": "2025-10-26T19:50:09",

"accuracy": 0.9805,

"f1\_score": 0.9805

}

]

}

4. /dataset/info (Dataset Information)

- Method: GET

- Purpose: Processed dataset details

- Returns: Rows, columns, file size, last modified timestamp

5. /logs/recent (Recent Logs)

- Method: GET

- Purpose: Recent log entries

- Returns: Log file information and recent entries

6. /health (Health Check)

- Method: GET

- Purpose: Application health status

- Returns:

{

"status": "healthy",

"timestamp": "2025-10-26T19:50:12",

"service": "Heart Disease Prediction API"

}

3.2 Display Application Details

Details Retrieved and Displayed (4+ details):

1. Pipeline Status

- Endpoint: /pipeline/status- Information:

- Data pipeline execution status

- ML pipeline execution status

- Last run timestamps

- Pipeline summaries

2. Model Performance Metrics

- Endpoint: /app/details or /models/info- Information:

- Trained models: 2 (Logistic Regression, Random Forest)

- Model accuracies: 86.36% (LR), 98.05% (RF)

- All performance metrics (precision, recall, F1, ROC-AUC)

3. Dataset Information

- Endpoint: /dataset/info or /app/details- Information:

- Total rows: 1,025

- Total columns: 23

- File size: ~186 KB

- Last processed: Timestamp

4. System Health Status

- Endpoint: /health- Information:

- Overall system status: "healthy"

- Service name

- Current timestamp

5. Flow Runs Status

- Available through: Prefect UI at http://localhost:4200

- Information:

- Flow execution history

- Task completion status

- Execution timelines

6. API Endpoints List

- Endpoint: Application documentation

- Information:

- 6 active endpoints

- All endpoints functional

- Response times logged

API Usage Examples:

Get main application details

curl http://localhost:5000/app/details

Get pipeline status

curl http://localhost:5000/pipeline/status

Get model information

curl http://localhost:5000/models/info

Health check

curl http://localhost:5000/health

Integration with Prefect:

- API endpoints trigger Prefect flows

- Status updates from Prefect runs

- Real-time monitoring via API

- Prefect UI accessible at http://localhost:4200

Implementation Summary

Technology Stack

- Workflow Orchestration: Prefect 3.0

- ML Framework: Scikit-learn

- Data Processing: Pandas, NumPy

- API Framework: Flask

- Visualization: Matplotlib, Seaborn

- Model Persistence: Joblib

- Monitoring: Custom dashboards + Prefect UI

Key Features

1. Automated data preprocessing pipeline

2. Scheduled workflows (every 2 minutes)

3. Multiple ML algorithms (Logistic Regression, Random Forest)

4. Comprehensive model evaluation (6+ metrics)

5. RESTful API for real-time monitoring

6. Automated logging and reporting

7. Visual monitoring dashboards

8. Prefect Cloud integration

Files Structure

assignment/

├── prefect\_data\_pipeline.py # Data preprocessing

├── prefect\_ml\_pipeline.py # Model training

├── prefect\_api\_integration.py # API endpoints

├── prefect\_deployment.py # Scheduling config

├── prefect\_monitoring.py # Monitoring system

├── heart.csv # Original dataset

├── heart\_processed.csv # Processed data

├── models/ # Trained models

│ ├── logistic\_regression\_model.pkl

│ ├── random\_forest\_model.pkl

│ ├── logistic\_regression\_metrics.json

│ └── random\_forest\_metrics.json

├── reports/ # Performance reports

├── dashboard/ # Visual dashboards

├── logs/ # Log files

└── requirements.txt # Dependencies

Deployment Configuration

- Data Pipeline: Every 2 minutes

- ML Pipeline: Daily at 6 AM

- Orchestrator: Every 4 hours

- API Status: Every 5 minutes

- Prefect UI: http://localhost:4200

- API Server: http://localhost:5000

Performance Results

- Logistic Regression Accuracy: 86.36%

- Random Forest Accuracy: 98.05% (Best Model)

- Total Pipeline Runs: Automated and scheduled

- Processing Time: < 1 minute per run

- Dataset Size: 1,025 records with 23 features

**A. Running the Application**

1. Install dependencies

pip install -r requirements.txt

2. Start Prefect server

prefect server start

3. Deploy pipelines

python prefect\_deployment.py --apply

4. Start API server

python prefect\_api\_integration.py

5. Access dashboard

Open http://localhost:4200 (Prefect UI)

Open http://localhost:5000 (API)

**B. API Endpoints Reference**

Endpoint | Method | Purpose |

1. Endpoint: /app/details

Method: GET

Purpose: Main application details

2. Endpoint: /pipeline/status

Method: GET

Purpose: Pipeline execution status

3. Endpoint: /models/info

Method: GET

Purpose: Model information

4. Endpoint: /dataset/info

Method: GET

Purpose: Dataset statistics

5. Endpoint: /logs/recent

Method: GET

Purpose: Recent log entries

6. Endpoint: /health

Method: GET

Purpose: Health check

**C. Model Metrics Summary**

Logistic Regression:

- Accuracy: 86.36%

- F1-Score: 86.36%

- ROC-AUC: 94.03%

Random Forest:

- Accuracy: 98.05%

- F1-Score: 98.05%

- ROC-AUC: 99.87%