Customer Classification Model

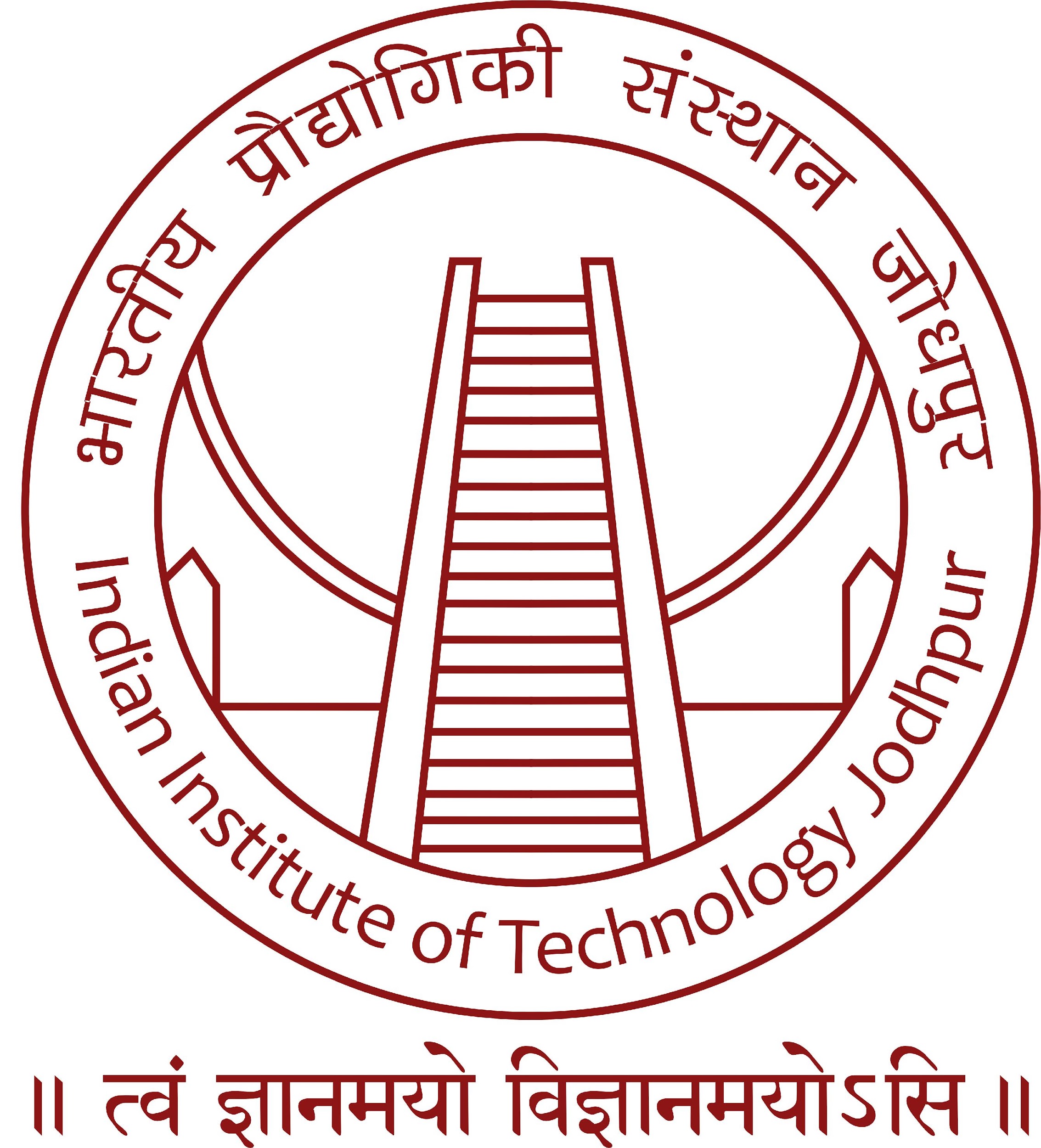
A Pg. Diploma Project Report

Submitted by

**Prince Dubey (G23AI2014)**

Under the Supervision

Of

Prof. Sumit Kalra

Department of Artificial Intelligence

Indian Institute of Technology Jodhpur

Project Designed Document

## **1. Introduction**

### 1.1. Project Overview

* **Objective**: The project aims to classify customers into different groups based on their purchasing behavior using the **K-means clustering** algorithm. This model will be deployed on **Google App Engine** to make it accessible as a web service that can be used by marketing teams for targeted campaigns.
* **Scope**: The scope of the project includes building the K-means model, deploying it on Google Cloud, and creating a simple user interface for predictions.
* **Stakeholders**: Marketing teams, data scientists, product managers, and developers.

### 1.2. Background

* With an increasing customer base, personalized marketing becomes critical. Grouping customers into clusters can help the marketing team target specific groups with tailored campaigns, improving conversion rates and customer engagement. The K-means algorithm is chosen due to its simplicity and efficiency in unsupervised classification tasks.

## **2. Requirements**

### 2.1. Functional Requirements

* **Data Ingestion**: The system should accept customer data in CSV format, including features like Annual income, spending score etc.
* **Clustering**: The model should apply K-means clustering to categorize customers into distinct groups.
* **API for Predictions**: Deploy an API that allows external applications to submit customer data and receive cluster predictions.
* **Model Training**: The ability to retrain the model with new data periodically to accommodate shifts in customer behavior.

### 2.2. Non-Functional Requirements

* **Performance**: The API should be able to handle 1000 requests per minute with a response time of less than 200ms.
* **Scalability**: The system should scale horizontally to handle increased traffic..
* **Availability**: 99.9% uptime, leveraging GCP's managed services for high availability.

## **3. Architecture and Design**

### 3.1. System Architecture

* **Google Cloud Platform (GCP)**: The application will use GCP for deployment, leveraging Google App Engine (GAE) for hosting.

**Components**:

* + **Frontend**: A simple web interface.
  + **Backend**: Python/Flask-based API providing access to the K-means clustering model.
  + **Model**: K-means clustering model implemented in Python using scikit-learn.
  + **Monitoring**: Stackdriver for logging and monitoring.

**Architecture Diagram**:

* + Client → App Engine (API) → K-means Model (in App Engine) → CSV fiel (for data).

### 3.2. Database Design

* **Google Cloud Storage**: For storing customer data as CSV files, which the model will periodically ingest for retraining.

## **4. Workflow and Process**

### 4.1. Use Case Diagrams

* **Use Case 1**: Marketing team provides new customer data → API processes the data → Returns the customer’s cluster.

### 4.2. Process Flow

1. User uploads data via API or CLI.
2. Data is stored in Google Cloud Storage.
3. K-means model is trained using customer features.
4. API is used to classify customers into clusters in real-time or for periodic batch predictions.

## **5. Technology Stack**

* **Modeling**: Python, Scikit-learn for building the K-means clustering model.
* **API Development**: Python with Flask for API services.
* **Deployment**: Google App Engine for scalable deployment.
* **Monitoring**: Stackdriver for logging and performance monitoring.

## **6. Security Considerations**

* **Data Encryption**: All data (in transit and at rest) will be encrypted using GCP’s default encryption standards (AES-256).
* **Authentication**: APIs will be protected using OAuth 2.0, with roles and permissions managed via Google Identity and Access Management (IAM).
* **Access Control**: Restrict access to the GCP environment and APIs through IAM roles.

## **7. Performance Considerations**

* **Model Latency**: The K-means clustering model will be optimized to respond to classification requests within 100ms for up to 1000 concurrent requests.
* **Load Balancing**: Google App Engine’s auto scaling will handle traffic spikes, ensuring that the API remains performant.
* **Caching**: Use of in-memory caching (e.g., Radis) can be considered to store recent customer classifications.

## **8. Testing and Validation**

### 8.1. Unit Testing

* Implement unit tests for each part of the API and model functions using **pytest**.

### 8.2. Integration Testing

* Test end-to-end integration by simulating requests to the API and checking the model’s responses.

### 8.3. User Acceptance Testing (UAT)

* Allow internal users (marketing team) to test the system and provide feedback on classification accuracy and API usability.

## **9. Deployment Strategy**

* **Development Environment**: Build the model and API locally and test using GCP Cloud Shell or local Docker environments.
* **Staging**: Deploy to a GCP staging environment for UAT.
* **Production**: Deploy the API and model on Google App Engine, enabling auto-scaling and ensuring proper logging and monitoring.

## **10. Risks and Mitigation**

* **Risk**: Model could become outdated as customer behaviors change over time.
  + **Mitigation**: Implement periodic retraining of the K-means model using new data from Cloud Storage.
* **Risk**: Scalability issues during traffic spikes.
  + **Mitigation**: Use App Engine’s autoscaling feature and monitor performance via Stackdriver.

## **11. Timeline**

### Phase 1: Initial Development

* **Week 1-2**: Data preparation, feature engineering, model training, and initial testing.

### Phase 2: API and GCP Integration

* **Week 3-4**: API development, testing, and deployment on Google App Engine.

### Phase 3: Final Testing and Go-Live

* **Week 5**: User Acceptance Testing (UAT), performance optimization, and go-live deployment.

## **12. Appendix**

* **Links**:
  + K-means Clustering
  + Google App Engine Documentation