Outline the project's objective, design thinking process, and development phases.

Describe the predictive use case, dataset selection, model training, deployment process, and integration steps.

Explain how the deployed model can be accessed and utilized for real-time predictions.

Objective:

Predict whether a customer will make a purchase based on their browsing behavior on an e-commerce website.

Design Thinking Process:

1. Empathize:

Conduct interviews with potential users to understand their needs and pain points.

2. Define:

Problem Statement: Increase conversion rates on the e-commerce website by targeting potential buyers more effectively.

3. Ideate:

Generate ideas for features that could be important in predicting purchases (e.g., time spent on product pages, number of items in the cart).

4. Prototype:

Create a preliminary dataset with synthesized data for testing initial models.

5. Test:

Train basic models and gather feedback from stakeholders on model performance.

Development Phases:

1. Data Collection and Preparation:

Synthetic Dataset: Generate synthetic data with features like time_on_page, items_in_cart, pages_visited, and purchase labels (1 for purchase, 0 for no purchase).

2. Model Selection and Training:

Use a simple logistic regression model as a starting point.

Program:

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

# Load the data

# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train the model

model = LogisticRegression()

model.fit(X_train, y_train)

# Evaluate the model

y_pred = model.predict(X_test)
```

accuracy = accuracy_score(y_test, y_pred)

Output:

data = pd.read_csv('data.csv')

X = data.drop('target_column', axis=1)

y = data['target_column']

3. Hyperparameter Tuning and Optimization:

Fine-tune hyperparameters using cross-validation.

4. Model Evaluation and Validation:

Calculate additional metrics (e.g., precision, recall, ROC-AUC).

Predictive Use Case:

Describe the specific predictive use case:

Predict whether a customer is likely to make a purchase based on their browsing behavior.

Dataset Selection:

Describe the dataset:

Synthetic dataset with features like time_on_page, items_in_cart, pages_visited, and purchase labels.

Model Training:

Specify the model used:

Logistic Regression.

Training process:

Load data, split, train, and evaluate as shown in the example above.

Deployment Process:

Hosting Environment:

Deploy the model on a cloud service like AWS, using a service like AWS Lambda or AWS SageMaker.

Model Serialization:

Serialize the trained model using a library like joblib or pickle.

API Creation:

Create an API endpoint using a framework like Flask.

Program

```
import joblib

# Serialize the model

joblib.dump(model, 'purchase_prediction_model.joblib')

from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/predict', methods=['POST'])

def predict():

  data = request.json

  features = [data['time_on_page'], data['items_in_cart'],
data['pages_visited']]

  prediction = model.predict([features])

return jsonify({'prediction': int(prediction[0])})
```

```python
if __name__ == '__main__':
app.run()
```

Integration Steps:

Deploy the Flask application on a server and configure it to accept POST requests with JSON data.

Accessing and Utilizing the Deployed Model:

API Endpoint:

The API is hosted at http://example.com/predict.

Input Format:

Send a POST request with JSON data containing features (time_on_page, items_in_cart, pages_visited).

Output Format:

Receive a JSON response with the predicted label (1 for purchase, 0 for no purchase).

Real-time Predictions:

Any system or application can send a POST request to the API to get real-time predictions for potential customers.