NAME: ASHWINI B N
EMAIL: ASHINIBN3@ICLOUD.COM
BATCH CODE: LISUM46
SUBMISSION DATE: 7/07/2025
ASSIGNMENT TITLE: ILLNESS PREDICTION WEB APP DEVELOPMENT USING HEROKU

1. Dataset Description:

The dataset used for this illness prediction model is a small toy dataset containing synthetic information about individuals from the city of Dallas. The dataset includes the following features:

- City: Location of the individual (all entries are 'Dallas' in this sample).
- Gender: Gender of the individual (Male/Female).
- Age: Age of the individual in years.
- Income: Annual income in USD.
- Illness: Target variable indicating whether the individual has an illness (Yes/No).

To prepare the data for modeling:

- The 'Number' column was dropped as it was only a serial identifier and did not contribute to prediction.
- Categorical variables (City, Gender, Illness) were encoded into numerical values using Label Encoding to ensure compatibility with the machine learning algorithm.

```
## Importing necessary libraries and loading the dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import joblib

df = pd.read_csv("/Users/darshanbn/Downloads/toy_dataset.csv")
print(df.head())
```

```
   Number    City Gender  Age   Income Illness
0       1  Dallas   Male   41  40367.0      No
1       2  Dallas   Male   54  45084.0      No
2       3  Dallas   Male   42  52483.0      No
3       4  Dallas   Male   40  40941.0      No
4       5  Dallas   Male   46  50289.0      No
```

```
# Dropping unnecessary column
df.drop('Number', axis=1, inplace=True)
```

The Number column is just serial index and does not help with prediction. So removing it.

```
# Encode Categorical Variables
#Converting the categorical columns ('City', 'Gender', 'Illness') into numerical format using Label Encoding.
label_encoders = {}
categorical_columns = ['City', 'Gender', 'Illness']

for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

Figure 1: Snapshot of the toy dataset and initial preprocessing steps including column drop and label encoding.

2. Model Training

After preprocessing the dataset (e.g., label encoding of categorical variables and dropping unnecessary columns), we trained **a** Random Forest Classifier to predict the likelihood of illness. The dataset was split into training and testing sets using an 80/20 split via train_test_split. Label encoding was applied to convert categorical variables such as City, Gender, and Illness into numeric form.

```
# Splitting dataset into training and testing sets
# Used 80% of the data for training and 20% for testing.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
Training set size: (120000, 4)
Test set size: (30000, 4)
```

```
# Training model using Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Figure 2: Splitting the dataset into training and test sets and training the Random Forest Classifier.

3. Model Evaluation

After training the classification model on the training dataset, we evaluated its performance using the test dataset (which consisted of 30,000 samples). The evaluation was done using accuracy score and the classification report, which includes precision, recall, and F1-score for each class.

```
# Evaluate model Performance
#Checking accuracy and get a classification report on test data
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8696666666666667

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93     27623
           1       0.08      0.06      0.07      2377

    accuracy                           0.87     30000
   macro avg       0.50      0.50      0.50     30000
weighted avg       0.85      0.87      0.86     30000
```

Figure 3: Evaluating model performance using accuracy and classification report on test data.

Once trained, the model was evaluated using accuracy and classification metrics, and finally saved as a .pkl file using joblib:

```python
# Saving the trained model using joblib

joblib.dump(model, 'illness_model.pkl')
```

```
['illness_model.pkl']
```

Figure 4: Saving the model using joblib

4. Flask Backend

The Flask backend (app.py) connects the machine learning model to a simple web form. When the app starts, it loads the saved model using pickle.

Two main routes are defined:

- '/' shows the input form.
- '/predict' collects form data, sends it to the model, and shows the result.

User inputs are taken from the form, converted to the right format, and passed into the model. The prediction ("Yes" or "No") is then displayed back on the page.

This setup lets users interact with the model easily through their browser.

Folder Structure: The project includes:

- app.py – Flask backend
- model.pkl – saved model
- templates/index.html – frontend form
- requirements.txt – package dependencies
- Procfile – for Heroku deployment

```python
from flask import Flask, render_template, request, jsonify
import joblib
import numpy as np

# Load the trained model
model = joblib.load('illness_model.pkl')

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        city = int(request.form['city'])
        gender = int(request.form['gender'])
        age = int(request.form['age'])
        income = float(request.form['income'])

        # ✅ Print inputs to terminal (for your screenshot and testing)
        print("Received Input:")
        print("City:", city)
        print("Gender:", gender)
        print("Age:", age)
        print("Income:", income)

        input_data = np.array([[city, gender, age, income]])
        prediction = model.predict(input_data)[0]
        result = "Yes" if prediction == 1 else "No"

        return render_template('index.html', prediction_text=f"Illness Prediction: {result}")
    except:
        return render_template('index.html', prediction_text="Error: Invalid input.")

@app.route('/api/predict', methods=['POST'])
def api_predict():
    data = request.get_json(force=True)
    city = int(data['city'])
    gender = int(data['gender'])
    age = int(data['age'])
    income = float(data['income'])

    input_data = np.array([[city, gender, age, income]])
    prediction = model.predict(input_data)[0]
    return jsonify({'prediction': int(prediction)})

if __name__ == "__main__":
    app.run(debug=True)
```

Figure 5: Flask backend code (app.py) handling form input, loading the model, and returning prediction results.

HTML form served locally for illness prediction. User inputs are submitted and prediction result is shown on the same page.

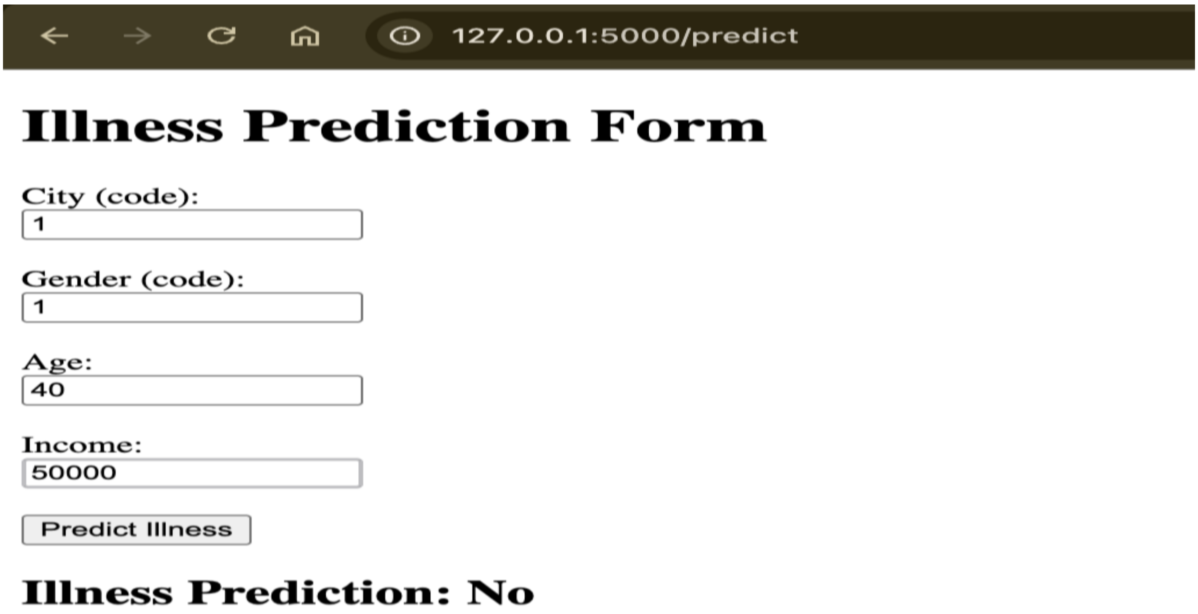


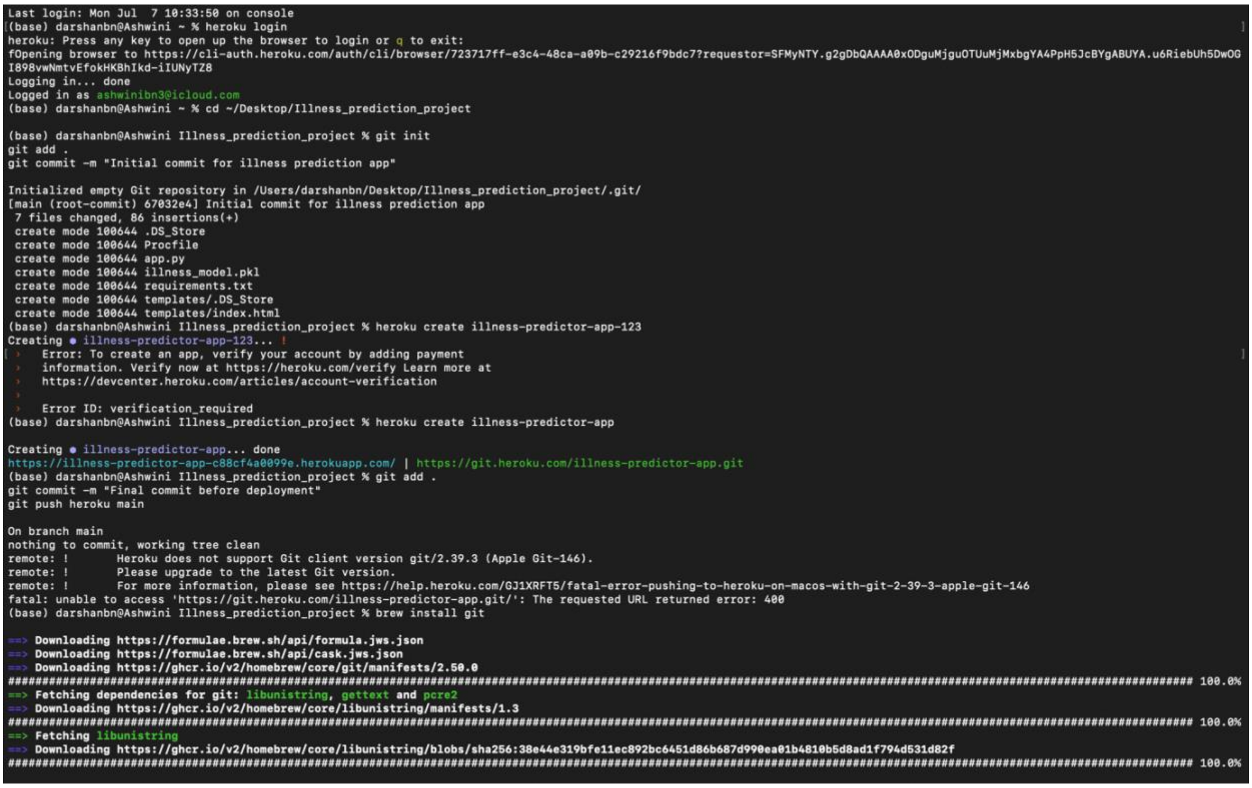
Figure 6: Local Flask web application interface displaying illness prediction based on user inputs.

5. Heroku Deployment

This step involved deploying the Flask application to Heroku to make it accessible via a public URL.

Key Steps:

1. Created Heroku Project & Linked Git
    - Initialized a Git repository:
    - Create a Heroku app and set remote:
2. Configured Python Runtime
    - Created runtime.txt with the version:
3. Added Required Files
    - requirements.txt for dependencies
    - Procfile to specify the web process:
4. Deployed to Heroku
    - Added and committed all files:
    - Pushed to Heroku:



```
Last login: Mon Jul  7 10:33:50 on console
(base) darshanbn@Ashwini ~ % heroku login
heroku: Press any key to open up the browser to login or q to exit:
fOpening browser to https://cli-auth.heroku.com/auth/cli/browser/723717ff-e3c4-48ca-a09b-c29216f9bdc7?requestor=SFMyNTY.g2gDbQAAAA0xODguMjguOTUuMjMxbgYA4PpH5JcBYgABUYA.u6RiebUh5DwOG
I898vwNmtvEfokHKBhIkd-iIUNyTZ8
Logging in... done
Logged in as ashwinibn3@icloud.com
(base) darshanbn@Ashwini ~ % cd ~/Desktop/Illness_prediction_project

(base) darshanbn@Ashwini Illness_prediction_project % git init
git add .
git commit -m "Initial commit for illness prediction app"

Initialized empty Git repository in /Users/darshanbn/Desktop/Illness_prediction_project/.git/
[main (root-commit) 67032e4] Initial commit for illness prediction app
 7 files changed, 86 insertions(+)
 create mode 100644 .DS_Store
 create mode 100644 Procfile
 create mode 100644 app.py
 create mode 100644 illness_model.pkl
 create mode 100644 requirements.txt
 create mode 100644 templates/.DS_Store
 create mode 100644 templates/index.html
(base) darshanbn@Ashwini Illness_prediction_project % heroku create illness-predictor-app-123
Creating ● illness-predictor-app-123... !
 ›   Error: To create an app, verify your account by adding payment
 ›   information. Verify now at https://heroku.com/verify Learn more at
 ›   https://devcenter.heroku.com/articles/account-verification
 ›
 ›   Error ID: verification_required
(base) darshanbn@Ashwini Illness_prediction_project % heroku create illness-predictor-app

Creating ● illness-predictor-app... done
https://illness-predictor-app-c88cf4a0099e.herokuapp.com/ | https://git.heroku.com/illness-predictor-app.git
(base) darshanbn@Ashwini Illness_prediction_project % git add .
git commit -m "Final commit before deployment"
git push heroku main

On branch main
nothing to commit, working tree clean
remote: !       Heroku does not support Git client version git/2.39.3 (Apple Git-146).
remote: !       Please upgrade to the latest Git version.
remote: !       For more information, please see https://help.heroku.com/GJ1XRFT5/fatal-error-pushing-to-heroku-on-macos-with-git-2-39-3-apple-git-146
fatal: unable to access 'https://git.heroku.com/illness-predictor-app.git/': The requested URL returned error: 400
(base) darshanbn@Ashwini Illness_prediction_project % brew install git

==> Downloading https://formulae.brew.sh/api/formula.jws.json
==> Downloading https://formulae.brew.sh/api/cask.jws.json
==> Downloading https://ghcr.io/v2/homebrew/core/git/manifests/2.50.0
######################################################################################################################## 100.0%
==> Fetching dependencies for git: libunistring, gettext and pcre2
==> Downloading https://ghcr.io/v2/homebrew/core/libunistring/manifests/1.3
######################################################################################################################## 100.0%
==> Fetching libunistring
==> Downloading https://ghcr.io/v2/homebrew/core/libunistring/blobs/sha256:38e44e319bfe11ec892bc6451d86b687d990ea01b4810b5d8ad1f794d531d82f
######################################################################################################################## 100.0%
```

Figure 7: Initializing Git, creating the Heroku app, and preparing the project for deployment via the Heroku CLI.

Once the Heroku app was created and the project was initialized with Git, the next step was to push the complete codebase to Heroku. This included all required files like app.py, model.pkl, requirements.txt, Procfile, and the HTML template.

Using the command git push heroku main, the app was deployed to the Heroku cloud platform. During this process, Heroku installed all specified dependencies, launched the web app, and generated the live application URL

```
remote:        Collecting markupsafe>=2.1.1 (from Flask->-r requirements.txt (line 1))
remote:          Downloading MarkupSafe-3.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.0 kB)
remote:        Collecting werkzeug>=3.1.0 (from Flask->-r requirements.txt (line 1))
remote:          Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
remote:        Collecting packaging (from gunicorn->-r requirements.txt (line 6))
remote:          Downloading packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
remote:        Collecting six>=1.5 (from python-dateutil>=2.8.1->pandas==1.5.3->-r requirements.txt (line 3))
remote:          Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
remote:        Downloading numpy-1.23.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
remote:        Downloading pandas-1.5.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.1 MB)
remote:        Downloading scikit_learn-1.2.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.6 MB)
remote:        Downloading flask-3.1.1-py3-none-any.whl (103 kB)
remote:        Downloading joblib-1.5.1-py3-none-any.whl (307 kB)
remote:        Downloading gunicorn-23.0.0-py3-none-any.whl (85 kB)
remote:        Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
remote:        Downloading click-8.2.1-py3-none-any.whl (102 kB)
remote:        Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
remote:        Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
remote:        Downloading MarkupSafe-3.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
remote:        Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
remote:        Downloading pytz-2025.2-py2.py3-none-any.whl (509 kB)
remote:        Downloading scipy-1.15.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (37.7 MB)
remote:        Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
remote:        Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
remote:        Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
remote:        Downloading packaging-25.0-py3-none-any.whl (66 kB)
remote:        Installing collected packages: pytz, threadpoolctl, six, packaging, numpy, markupsafe, joblib, itsdangerous, click, blinker, werkzeug, scipy, python-dateutil, jinja2,
 gunicorn, scikit-learn, pandas, Flask
remote:        Successfully installed Flask-3.1.1 blinker-1.9.0 click-8.2.1 gunicorn-23.0.0 itsdangerous-2.2.0 jinja2-3.1.6 joblib-1.5.1 markupsafe-3.0.2 numpy-1.23.5 packaging-25.0
 pandas-1.5.3 python-dateutil-2.9.0.post0 pytz-2025.2 scikit-learn-1.2.2 scipy-1.15.3 six-1.17.0 threadpoolctl-3.6.0 werkzeug-3.1.3
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 137.4M
remote: -----> Launching...
remote:        Released v4
remote:        https://illness-predictor-app-c88cf4a0099e.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/illness-predictor-app.git
   67032e4..5d6ae98  main -> main
(base) darshanbn@Ashwini Illness_prediction_project % heroku apps:info -a illness-predictor-app

=== illness-predictor-app

Auto Cert Mgmt: false
Dynos:          web: 1
Git URL:        https://git.heroku.com/illness-predictor-app.git
Owner:          ashwinibn3@icloud.com
Region:         us
Repo Size:      35 MB
Slug Size:      137 MB
Stack:          heroku-24
Web URL:        https://illness-predictor-app-c88cf4a0099e.herokuapp.com/
(base) darshanbn@Ashwini Illness_prediction_project %
```

Figure 8: Successful deployment of the app to Heroku and confirmation of the live web app URL.

6. Web Output

After deploying the app on Heroku, I tested it by opening the live URL in a browser. The webpage shows a simple form where users can enter values like city code, gender, age, and income. Once the form is filled out and the "Predict Illness" button is clicked, the app sends the input to the model and displays the result instantly.

In the screenshot, the model responded with **"Illness Prediction: No"**, which means the person is unlikely to be ill based on the information entered.

This confirms that the full pipeline — from user input to model prediction — is working smoothly through the deployed web interface.

Figure 9: Web application hosted on Heroku generating an illness prediction based on user input.

7. Conclusion

To sum it up, I successfully trained a machine learning model to predict illness based on user inputs like city, gender, age, and income. I then built a simple Flask web app to connect this model to a user-friendly HTML form. After testing it locally, I deployed the app to Heroku, making it accessible online. Finally, I verified that the prediction works correctly by submitting inputs and getting a valid result from the model.

This project demonstrates a complete end-to-end ML deployment — from model training to live web prediction.