EXPERIMENT - 5

Aim - Implement the House Price Prediction model using Linear Regression in the backend

Data Parameters -

longitude

latitude

housing_median_age

total_rooms

total_bedrooms

population

households

median_income

median_house_value

ocean_proximity

Code -

model.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
import pickle

df = pd.read_csv('housing.csv')

df.dropna(inplace=True)
```

```
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return df[(df[column] >= lower) & (df[column] <= upper)]

for col in ['median_income', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households']:
    df = remove_outliers(df, col)

features = ['median_income', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'latitude', 'longitude']
X = df[features]
y = df['median_house_value']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LinearRegression()
model.fit(X_train_scaled, y_train)

with open('california_house_price_model.pkl', 'wb') as f:
    pickle.dump(model, f)

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

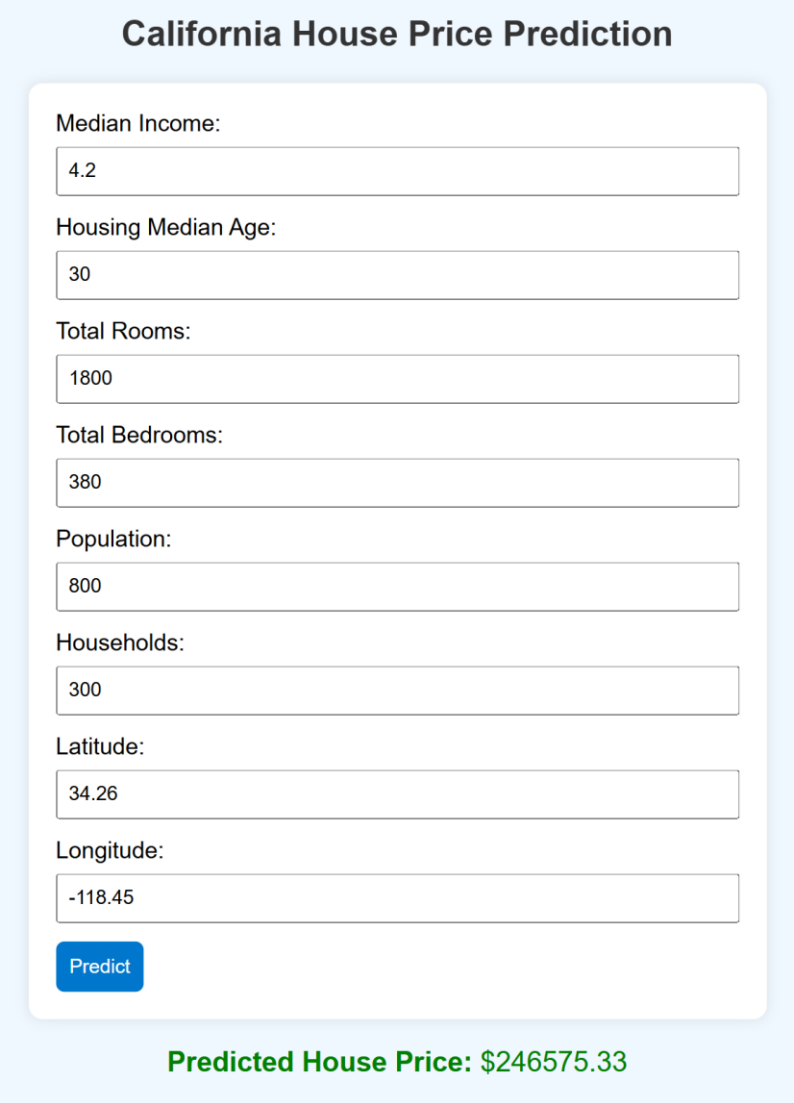
Rohan Saini
04419051622

App.py

```
from flask import Flask, render_template, request
import pickle
import numpy as np

app = Flask(__name__)

with open('california_house_price_model.pkl', 'rb') as f:
    model = pickle.load(f)

with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

@app.route('/')
def home():
    return render_template('index.html', prediction=None)

@app.route('/predict', methods=['POST'])
def predict():
    try:
        form_data = request.form
        features = [
            float(form_data['median_income']),
            float(form_data['housing_median_age']),
            float(form_data['total_rooms']),
            float(form_data['total_bedrooms']),
            float(form_data['population']),
            float(form_data['households']),
            float(form_data['latitude']),
            float(form_data['longitude']),
        ]
```

Rohan Saini
04419051622

```
        scaled_features = scaler.transform([features])
        prediction = model.predict(scaled_features)[0]
        prediction = round(prediction, 2)
        return render_template('index.html', prediction=prediction, form_data=form_data)
    except Exception as e:
        return f"Error: {str(e)}"
if __name__ == '__main__':
    app.run(debug=True)
```



Rohan Saini
04419051622

EXPERIMENT - 6

Aim - Train a model for diabetes prediction and use it in backend for predicting the disease through a UI integrated using flask framework

Data Parameters -

Pregnancies: To express the Number of pregnancies

Glucose: To express the Glucose level in blood

BloodPressure: To express the Blood pressure measurement

SkinThickness: To express the thickness of the skin

Insulin: To express the Insulin level in blood

BMI: To express the Body mass index

DiabetesPedigreeFunction: To express the Diabetes percentage

Age: To express the age

Outcome: To express the final result 1 is Yes and 0 is No

Code -

model.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pickle
df = pd.read_csv('diabetes.csv')
X = df.drop('Outcome', axis=1)
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

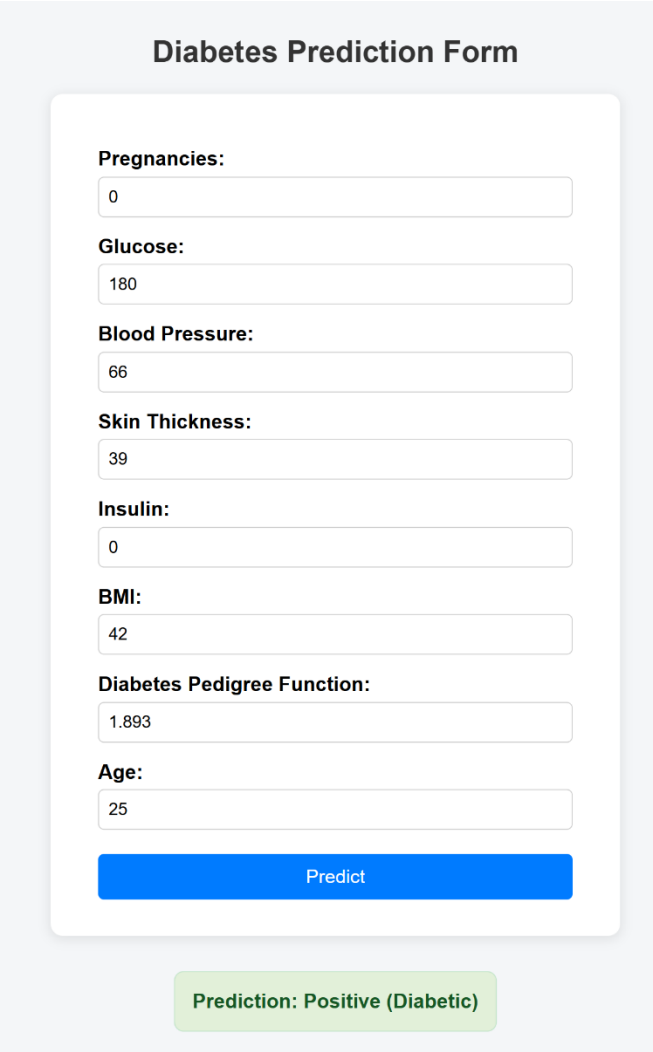
Rohan Saini
04419051622

```
with open('diabetes_model.pkl', 'wb') as f:
    pickle.dump(clf, f)
```

App.py

```
from flask import Flask, request, render_template
import numpy as np
import pickle

app = Flask(__name__)

with open('diabetes_model.pkl', 'rb') as f:
    model = pickle.load(f)

@app.route('/')
def home():
    return render_template('index.html', form_data={})

@app.route('/predict', methods=['POST'])
def predict():
    try:
        form_data = {
            'Pregnancies': request.form['Pregnancies'],
            'Glucose': request.form['Glucose'],
            'BloodPressure': request.form['BloodPressure'],
            'SkinThickness': request.form['SkinThickness'],
            'Insulin': request.form['Insulin'],
            'BMI': request.form['BMI'],
            'DiabetesPedigreeFunction': request.form['DiabetesPedigreeFunction'],
            'Age': request.form['Age']
        }
        data = [float(form_data[key]) for key in form_data]
        input_data = np.array([data])
```

```
        prediction = model.predict(input_data)[0]
        result = "Positive (Diabetic)" if prediction == 1 else "Negative (Not Diabetic)"
        return render_template('index.html', result=result, form_data=form_data)
    except Exception as e:
        return render_template('index.html', result=f"Error: {e}", form_data=request.form)
if __name__ == '__main__':
    app.run(debug=True)
```



Rohan Saini
04419051622

EXPERIMENT – 7

Aim - Design an Recommendation system model in backend for IMDb movies dataset and do prediction using UI's parameters like Age criteria ,Likeness , Preferences, Language selection etc.

Data Parameters –

- Poster_Link - Link of the poster that imdb using
- Series_Title = Name of the movie
- Released_Year - Year at which that movie released
- Certificate - Certificate earned by that movie
- Runtime - Total runtime of the movie
- Genre - Genre of the movie
- IMDB_Rating - Rating of the movie at IMDB site
- Overview - mini story/ summary
- Meta_score - Score earned by the movie
- Director - Name of the Director
- Star1,Star2,Star3,Star4 - Name of the Stars
- No_of_votes - Total number of votes
- Gross - Money earned by that movie

Code –

model.py

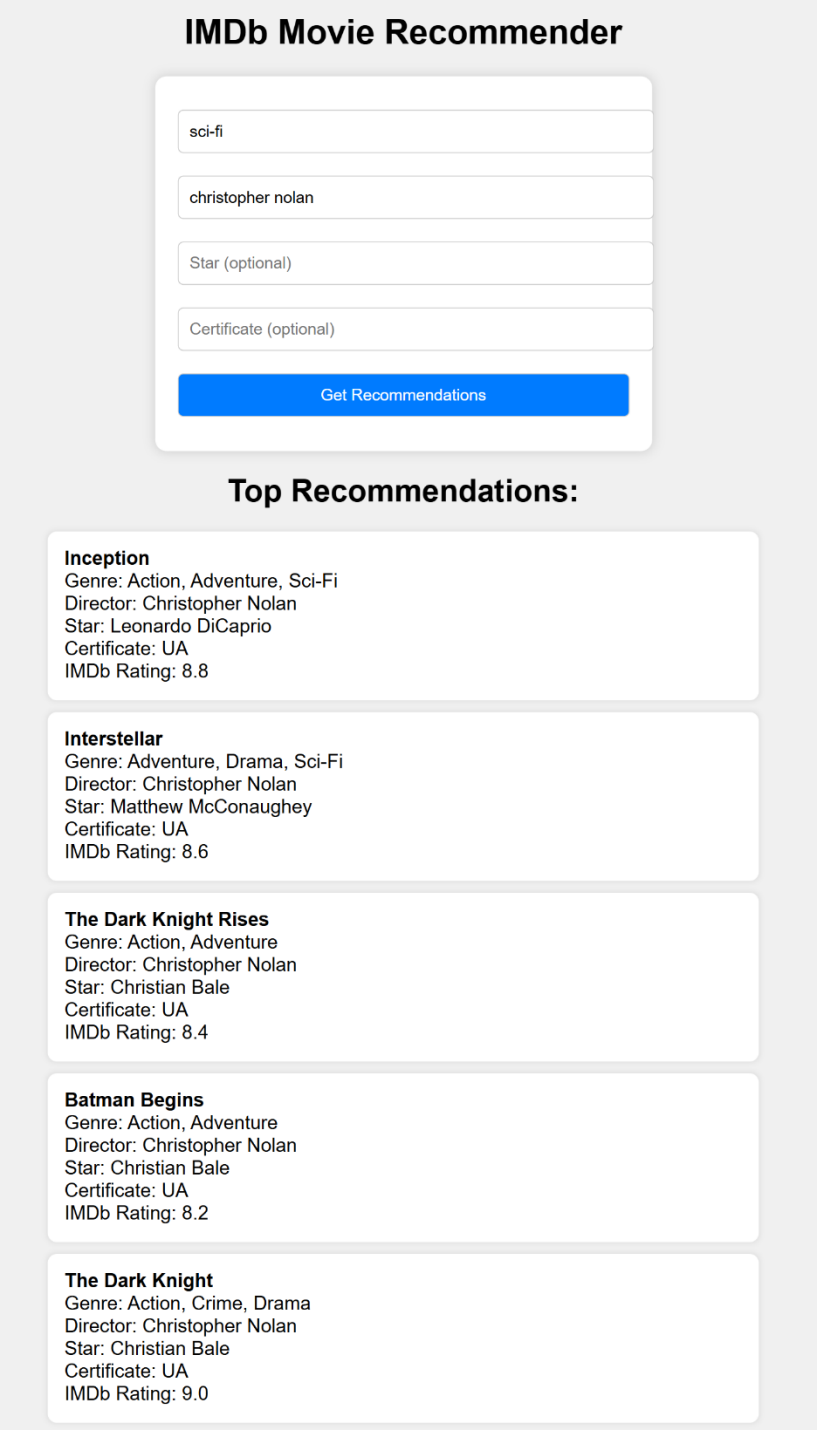
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pickle


df = pd.read_csv('imdb_top_1000.csv')
```

```
df['Certificate'] = df['Certificate'].fillna('Unknown')
df['Director'] = df['Director'].fillna('')
df['Star1'] = df['Star1'].fillna('')
df['Genre'] = df['Genre'].fillna('')
df['combined'] = df['Genre'] + ' ' + df['Director'] + ' ' + df['Star1'] + ' ' + df['Certificate']

vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df['combined'])

cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

with open('recommender_model.pkl', 'wb') as f:
    pickle.dump({
        'df': df,
        'cosine_sim': cosine_sim,
        'vectorizer': vectorizer,
        'tfidf_matrix': tfidf_matrix
    }, f)
```

App.py –

```
from flask import Flask, render_template, request
import pickle
from sklearn.metrics.pairwise import cosine_similarity

app = Flask(__name__)

with open('recommender_model.pkl', 'rb') as f:
    data = pickle.load(f)

df = data['df']
cosine_sim = data['cosine_sim']
vectorizer = data['vectorizer']
```

```
tfidf_matrix = data['tfidf_matrix']


def get_recommendations(genre, director, star, certificate):
    input_str = f"{genre} {director} {star} {certificate}"
    input_vec = vectorizer.transform([input_str])
    sim_scores = cosine_similarity(input_vec, tfidf_matrix).flatten()
    top_indices = sim_scores.argsort()[-6:][::-1]
    return df.iloc[top_indices][['Series_Title', 'Genre', 'Director', 'Star1', 'Certificate',
'IMDB_Rating']].to_dict(orient='records')[1:]


@app.route('/')
def home():
    return render_template('index.html', recommendations=None)


@app.route('/recommend', methods=['POST'])
def recommend():
    form = request.form
    genre = form['genre']
    director = form.get('director', '')
    star = form.get('star', '')
    certificate = form.get('certificate', '')


    recommendations = get_recommendations(genre, director, star, certificate)
    return render_template('index.html', recommendations=recommendations, form_data=form)


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



Rohan Saini
04419051622

EXPERIMENT – 8

Aim - Analyse Prediction of Heart Disease Based on Machine Learning and design a recommender system through UI, which will accept params like, Age, Cholesterol , Chest pain level etc. and suggest a risk level according to it

Data Parameters –

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

Code –

model.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pickle
```

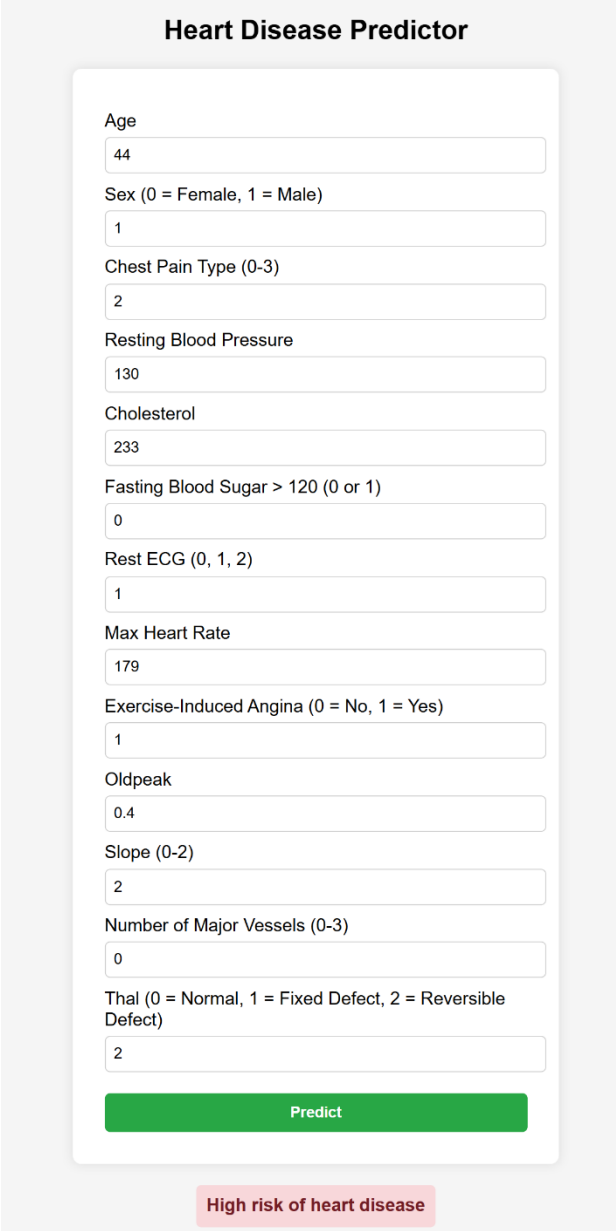
Rohan Saini
04419051622

```
df = pd.read_csv('heart.csv')
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
with open('heart_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

App.py –

```
from flask import Flask, render_template, request
import numpy as np
import pickle
app = Flask(__name__)
with open('heart_model.pkl', 'rb') as f:
    model = pickle.load(f)
@app.route('/')
def home():
    return render_template('index.html', form_data={})
@app.route('/predict', methods=['POST'])
def predict():
    try:
        form_data = {key: request.form[key] for key in request.form}
        data = [float(form_data[key]) for key in [
            'age', 'sex', 'cp', 'trestbps', 'chol',
            'fbs', 'restecg', 'thalach', 'exang',
            'oldpeak', 'slope', 'ca', 'thal'
        ]]
        prediction = model.predict([np.array(data)])[0]
        result = "High risk of heart disease" if prediction == 1 else "Low risk (no disease)"
```

```
        return render_template('index.html', result=result, form_data=form_data)
    except Exception as e:
        return render_template('index.html', result=f"Error: {e}", form_data=request.form)
if __name__ == '__main__':
    app.run(debug=True)
```



Rohan Saini
04419051622