```
# Produced by: Kirubel Temesgen
# College ID: C00260396
# Description: To use linear aggression to predict employee salaries
#              given the business sector and size.

from flask import Flask, request, render_template, jsonify
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib
matplotlib.use('Agg')  # Use of the non-interactive backend for generating images
import matplotlib.pyplot as plt
import io
import base64

app = Flask(__name__)

# Route to get column names from uploaded CSV
@app.route('/columns', methods=['POST'])
def columns():
    file = request.files['file']
    # Load the dataset
    df = pd.read_csv(file)

    # Prepare the coloumns
    df.columns = df.columns.str.strip()
    columns = df.columns.tolist()

    # Return the coloumns in a json file
    return jsonify(columns)
```

```
# Route to get unique employee sizes from the selected column
@app.route('/employee_sizes', methods=['POST'])
def employee_sizes():
    # Return the selected value
    column = request.args.get('column')
    file = request.files['file']
    df = pd.read_csv(file)
    employee_sizes = df[column].unique().tolist()
    return jsonify(employee_sizes)


# Main page route
@app.route('/')
def index():
    return render_template('indexs.html')
@app.route('/predict', methods=['POST'])
def predict():
    file = request.files['file']
    df = pd.read_csv(file)

    # Get user selections
    salary_column = request.form.get('salary_column')  # Typically 'VALUE'
    sector_column = request.form.get('sector_column')  # Typically 'Economic Sector'
    employee_size_column = request.form.get('employee_size_column')  # Typically 'Size of
Employees per Enterprise'
    employee_size = request.form.get('employee_size')

    # Filter dataset by employee size
    df_filtered = df[df[employee_size_column] == employee_size]

    # Handle missing or invalid salary values
```

```
df_filtered = df_filtered[df_filtered[salary_column] > 0].dropna()

# One-hot encode sector
df_encoded = pd.get_dummies(df_filtered[[sector_column]], drop_first=True)

# Define (X) as the one-hot encoded sector colum and (y) as salary
X = df_encoded
y = df_filtered[salary_column]  # Define 'y' as the salary column which will be predicted

# Split data into training and testing sets, and include the sector column
X_train, X_test, y_train, y_test, sectors_train, sectors_test = train_test_split(
   X, y, df_filtered[sector_column], test_size=0.2, random_state=42)

# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Color map for each sector
unique_sectors = sectors_test.unique()
colors = plt.cm.get_cmap('tab10', len(unique_sectors))

# Create the scatter plot and color-code by sector
plt.figure(figsize=(17, 8))
```

```
    # Assign different colors for each sector and ensuring each sector is labeled once in the
legend
    for i, sector in enumerate(unique_sectors):
        sector_filter = sectors_test == sector
        plt.scatter(y_test[sector_filter], y_pred[sector_filter], color=colors(i), label=sector)

    # Add labels and legend
    plt.xlabel('Actual Salary')
    plt.ylabel('Predicted Salary')
    plt.title('Actual vs Predicted Salary (Linear Regression) with Sectors')

    #**                    **#
    # Move the legend outside of the plot and adjust layout to fit everything
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), title="Sectors")


    # Use tight_layout to adjust the figure and make sure everything fits
    plt.tight_layout(pad=2.0, rect=[0, 0, 0.85, 1])

    # Convert plot to PNG image
    img = io.BytesIO()
    plt.savefig(img, format='png')
    img.seek(0)
    plot_url = base64.b64encode(img.getvalue()).decode()

    #**                    **#

    return render_template('result.html', mse=mse, r2=r2, plot_url=plot_url)


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