*Southeast University (SEU)*

Department of Computer Science &amp; Engineering (CSE)

# Glucose Level-Prediction application

DATA MINING(CSE 353.2)

**Submitted To:**

**Md. Mijanur Rahman**
Assistant Professor
CSE Dept. of Southeast University

**Submitted By:**

1. Khandoker humayoun kobir **ID:** 2020000000**137**

# _Linear Regression Model:

**Linear regression** analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the **independent** variable.

This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values.

we **can perform the linear regression method** in a variety of programs and environments, including:

- R linear regression
- MATLAB linear regression
- Sklearn linear regression
- Linear regression Python
- Excel linear regression

In this project I've used SKLearn linear regression.

## SKlearn Linear regression:

---

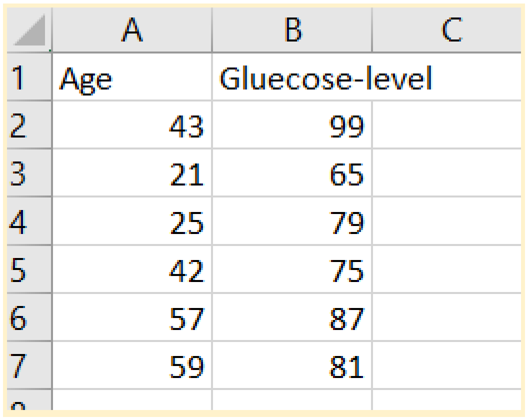
**Scikit-learn** is a Python package that makes it easier to apply a variety of Machine Learning (ML) algorithms for predictive data analysis, such as **linear regression.**

The line can then be projected to forecast fresh data points. Because of its simplicity and essential features, linear regression is a fundamental Machine Learning method.

## Glucose Level-Prediction application :

A web application which uses Machine Learning algorithm to predict the glucose level of a person by providing input of a person's age built using Flask.

## Train model with CSV file:



| | A | B | C |
|---|---|---|---|
| 1 | Age | Gluecose-level | |
| 2 | 43 | 99 | |
| 3 | 21 | 65 | |
| 4 | 25 | 79 | |
| 5 | 42 | 75 | |
| 6 | 57 | 87 | |
| 7 | 59 | 81 | |

**I've completed my project in three steps:**

1.linear regression model train in **jupyter notebook**

2.Created a python project named Glucose_level_age for development in **pycharm.**

3In my project I used another folder **'template'** under which I developed python code in '**index.html**' file.

## Installation:

The Code is written in jupyter notebook(Anaconda 3). If you don't have this version we can find it [here](https://www.anaconda.com/products/distribution). To install the required packages and libraries, run this command in the project directory after cloning the repository:

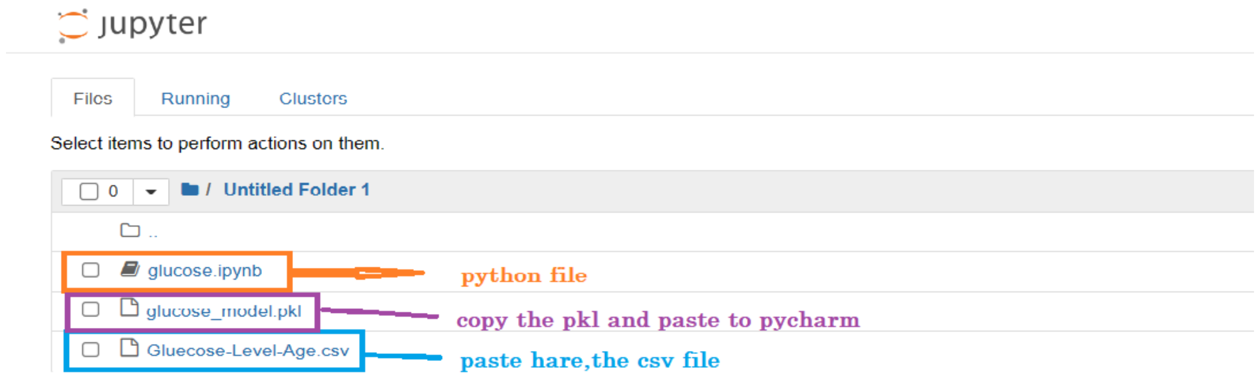
**For jupyter  notebook:**

| name | Version number |
|---|---|
| Pandas | 1.2.4 |
| Numpy | 1.20.2 |
| Pikkle | 1.2.0 |
| Sci-kit-learn | 1.0.2 |

**For pie charm:**

| name | Version number |
|---|---|
| Pandas | 1.2.4 |
| Flask | 1.20.2 |
| Pikkle | 1.2.0 |
| Sci-kit-learn | 1.0.2 |

## Linear regression model training in jupyter notebook:



Select items to perform actions on them.

0 / Untitled Folder 1

..

glucose.ipynb — python file

glucose_model.pkl — copy the pkl and paste to pycharm

Gluecose-Level-Age.csv — paste hare,the csv file

```
import numpy as np   # import the numpy library for numerical computations

import pandas as pd   # import the pandas library for data manipulation



data = pd.read_csv("Gluecose-Level-Age.csv")   # load the dataset from a CSV file

data  # display the loaded dataset

data.shape   # display the shape of the dataset (number of rows and columns)



x=data['Age']  # extract the age column from the dataset

y=data['Gluecose-level']  # extract the glucose level column from the dataset



from sklearn.model_selection import train_test_split   # import the train_test_split function for splitting the dataset

xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.30,random_state=1)  # split the dataset into training and testing sets
```

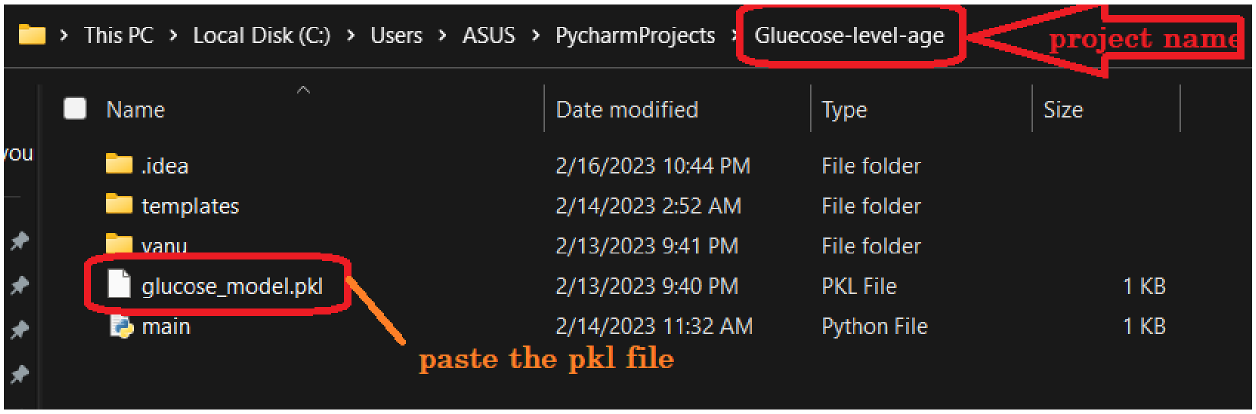
```
from sklearn.linear_model import LinearRegression  # import the LinearRegression class for creating a linear regression model

le = LinearRegression()   # create an instance of the LinearRegression class

le.fit(xtrain.values.reshape(-1,1),ytrain)   # fit the training data to the linear regression model

le.predict(xtest.values.reshape(-1, 1))   # make predictions on the testing data using the trained model

le.predict([[55]])   # make a single prediction for a new age value of 55



import pickle  # import the pickle library for saving and loading Python objects

pickle.dump(le, open('glucose_model.pkl', 'wb')

pipe=pickle.load(open("glucose_model.pkl",'rb'))
pipe.predict([[55]]
```

**This is a Python script for training a simple linear regression model on a dataset of glucose levels and ages, and then saving the model using pickle. Here's a breakdown of the code:**

1. `import numpy as np and import pandas as pd` import the numpy and pandas libraries for numerical computations and data manipulation, respectively.
2. `data = pd.read_csv("Gluecose-Level-Age.csv")` loads the dataset from a CSV file using pandas.
3. `data` displays the loaded dataset.
4. `data.shape` displays the shape of the dataset (number of rows and columns).
5. **x=data['Age'] and y=data['Gluecose-level']** extract the age and glucose level columns from the dataset, respectively.
6. `from sklearn.model_selection import train_test_split` imports the train_test_split function from scikit-learn's model_selection module for splitting the dataset into training and testing sets.
7. `xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.30,random_state=1)` splits the dataset into training and testing sets with a test size of 30% and a random state of 1.

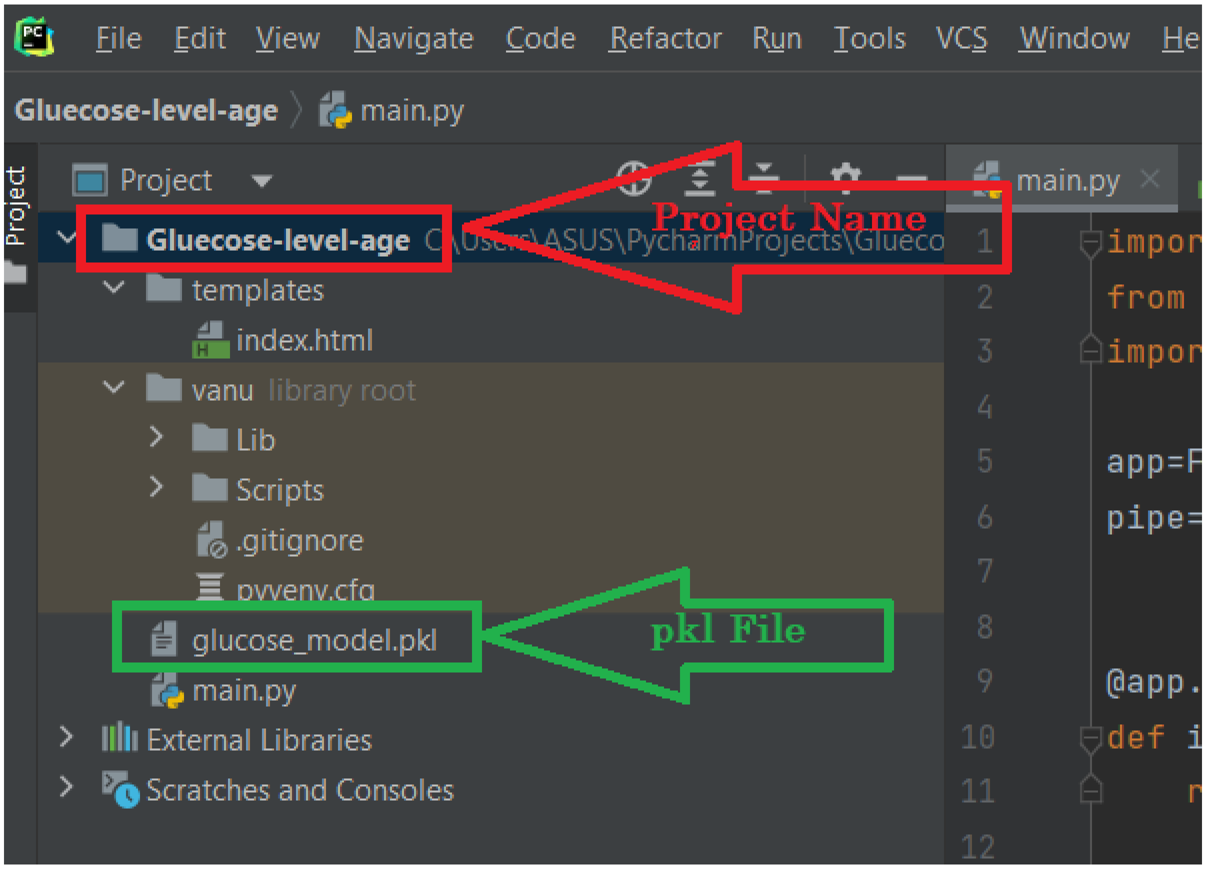
8. `from sklearn.linear_model import LinearRegression` imports the LinearRegression class from scikit-learn's linear_model module for creating a linear regression model.
9. `le = LinearRegression()` creates an instance of the LinearRegression class.
10. `le.fit(xtrain.values.reshape(-1,1),ytrain)` fits the training data to the linear regression model.
11. `le.predict(xtest.values.reshape(-1, 1))` makes predictions on the testing data using the trained linear regression model.
12. `le.predict([[55]])` makes a single prediction for a new age value of 55.
13. `import pickle` imports the pickle library for serializing and deserializing Python objects.
14. `pickle.dump(le, open('glucose_model.pkl', 'wb'))` saves the trained model as a pickled file.
15. `pipe=pickle.load(open("glucose_model.pkl",'rb'))` loads the saved model from the pickled file.
16. `pipe.predict([[55]])` makes a prediction using the loaded model for a new age value of 55.

## Python code for development in pycharm.



This PC › Local Disk (C:) › Users › ASUS › PycharmProjects › Gluecose-level-age

| Name | Date modified | Type | Size |
|---|---|---|---|
| .idea | 2/16/2023 10:44 PM | File folder | |
| templates | 2/14/2023 2:52 AM | File folder | |
| vanu | 2/13/2023 9:41 PM | File folder | |
| glucose_model.pkl | 2/13/2023 9:40 PM | PKL File | 1 KB |
| main | 2/14/2023 11:32 AM | Python File | 1 KB |



```
import pandas as pd  # import the pandas library

from  flask  import  Flask,render_template,request    #  import  the
Flask library and two submodules

import pickle  # import the pickle library for loading the model

app=Flask(__name__)  # create a Flask application instance

pipe=pickle.load(open("glucose_model.pkl",'rb'))      #   load   the
machine learning model

@app.route('/')  # map the URL "/" to the index function

def index():

return render_template('index.html')  # render the HTML template
for the index page

@app.route('/predict',  methods=['POST'])       #   map   the   URL
"/predict" to the predict function for handling POST requests
```

```
def predict():

Age=request.form.get('Age')    # extract the age input from the
form data

input=pd.DataFrame([[Age]],columns=[Age])   # convert the input to
a Pandas dataframe

prediction=pipe.predict(input)[0]   # make a prediction using the
machine learning model

return str(prediction)   # return the predicted glucose level as a
string

if __name__=="__main__":
app.run(debug=True,  port=5000)    # start the Flask application
with debug mode enabled and listen on port 5000
```

This is a **Python file** that implements a web application using **Flask** for predicting glucose levels based on age, using a pre-trained machine learning model. Here's a line-by-line explanation:

1. `import pandas as pd` imports the pandas library for working with data.

2. `from flask import Flask,render_template,request` imports the Flask class for building web applications, as well as two functions for rendering templates and handling HTTP requests.

3. `import pickle` imports the pickle module for loading the pre-trained machine learning model.

4. `app=Flask(__name__)` creates a new Flask application instance.

5. `pipe=pickle.load(open("glucose_model.pkl",'rb'))` loads the pre-trained machine learning model from a pickle file named "glucose_model.pkl" using the pickle.load() method and assigns it to the variable "pipe".

6. `@app.route('/')` is a decorator that specifies the URL route for the "index" function.

7. `def index():` defines the "index" function, which returns the contents of an HTML file called "index.html" using the Flask function "render_template".

8. `@app.route('/predict', methods=['POST'])` is a decorator that specifies the URL route for the "predict" function, which accepts HTTP POST requests containing form data.
9.
10. `def predict():` defines the "predict" function, which extracts the age input from the submitted form data, creates a pandas DataFrame containing the input data, uses the pre-trained machine learning model to make a prediction, and returns the predicted glucose level as a string.

11. `if __name__=="__main__":` is a conditional statement that checks if this script is being run directly as the main program.

12. `app.run(debug=True, port=5000)` starts the Flask development server on port 5000, enabling debugging mode with "debug=True"

## **Development with python code:**

```
i<!DOCTYPE html>

<html>

<head>

     <title>Glucose Level Predictor</title>

</head>

<body>

     <h1>Glucose Level Predictor</h1>

     <form action="/predict" method="post">

          <label for="Age">Age:</label>

          <input type="text" id="Age" name="Age">

          <br><br>

          <input type="submit" value="Predict">

     </form>

     <div id="result"></div>
```

```
      <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.mi
n.js"></script>

      <script>

            $(function() {

                  $('form').on('submit', function(event) {

                        event.preventDefault();

                        $.ajax({

                              url: '/predict',

                              data: $('form').serialize(),

                              type: 'POST',

                              success: function(response) {

                                    $('#result').text('Predicted
glucose level: ' + response);

                              },

                              error: function(error) {

                                    console.log(error);

                              }

                        });

                  });

            });

      </script>

</body>

</html>
```

This is an **HTML file** that contains a form for predicting **glucose levels** using a machine learning model. Let's break down the code:

1. <!DOCTYPE html> declares that this is an HTML document.
2. <html> and </html> enclose the entire HTML document.
3. <head> and </head> enclose information about the document that is not displayed, such as the document's title.
4. <title> and </title> enclose the text that is displayed in the browser's title bar.
5. <body> and </body> enclose the visible content of the HTML document.
6. <h1> and </h1> enclose the main heading of the form.
7. <form> and </form> enclose the form for inputting age and submitting the prediction request.
8. action="/predict" specifies the URL to which the form data is sent when the form is submitted.
9. method="post" specifies the HTTP method used to submit the form data.
10. <label for="Age"> and </label> enclose the label for the age input field.
11. id="Age" identifies the input field for age.
12. name="Age" specifies the name of the input field for age.
13. <input type="text"> specifies that the input field is a text field.
14. <br><br> adds a line break.
15. <input type="submit"> creates a button for submitting the form.
16. value="Predict" specifies the text displayed on the button.
17. <div id="result"></div> creates a div element with an id of "result" that will be used to display the predicted glucose level.
18. <script> and </script> enclose JavaScript code.
19. **`src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"`** specifies the location of the jQuery library.
20. $(function() {...}) is a jQuery shortcut for the document ready event.
21. $('form').on('submit', function(event) {...}) sets up a listener for the form's submit event.
22. event.preventDefault() prevents the form from being submitted in the default way.
23. $.ajax({...}) makes an AJAX request to the server.
24. url: '/predict' specifies the URL to which the AJAX request is sent.
25. data: $('form').serialize() serializes the form data for submission.
26. type: 'POST' specifies the HTTP method used for the AJAX request.
27. success: function(response) {...} defines a callback function that is executed when the AJAX request is successful.
28. $('#result').text('Predicted glucose level: ' + response) sets the text content of the "result" div to the predicted glucose level returned by the server.
29. error: function(error) {...} defines a callback function that is executed when the AJAX request fails.
30. console.log(error) logs the error to the console.

**AJAX Request:**

An AJAX (Asynchronous JavaScript and XML) request is a technique for sending and receiving data from a web server asynchronously without the need to reload the entire web page.

This means that a web page can make requests to the server, retrieve data, and update specific parts of the page without requiring the user to refresh the entire page. AJAX requests can be used to perform a variety of tasks, such as submitting a form, retrieving data, and updating the page dynamically based on user interaction.

AJAX requests typically use JavaScript to make requests to the server and receive responses in the form of XML, JSON, HTML, or plain text. The data returned by the server can then be used to update specific parts of the page or execute other client-side code.

-----------------------------------------------