In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [2]:
```
class LogisticRegression:
    def __init__(self):
        self.weights = None
        self.bias = 0
        self.m = 0
        self.n = 0
        self.cost_values = np.array([])

    def Fit(self, df_features, df_prob, learning_rate):
        self.m, self.n = df_features.shape
        self.weights = np.zeros((self.n,1))
        df_prob = df_prob.to_numpy()
        df_prob_pred = self.sigmoid(df_features)

        self.GD(df_prob, df_features, learning_rate)


    def GD(self, y, x, rate):
        iterations = 1
        while iterations < 10000:
            y_pred = self.sigmoid(x)
            self.cost_values = np.append(self.cost_function(y_pred,y),self.cost_val
            dw = (1/self.m) * np.dot(x.T,(y_pred - y))
            db = (1/self.m) * np.sum(y_pred - y)
            self.weights -= rate*dw
            self.bias -= rate*db
            iterations += 1

    def cost_function(self,y_hat,y):
        # np.append(self.cost_values,[(1/self.m) * (np.dot(y.T,np.log(y_hat)) + np.
        return [((1/self.m) * (np.dot(y.T,np.log(y_hat)) + np.dot((1-y).T,np.log(1-

    def sigmoid(self, features):
        z = np.dot(features, self.weights) + self.bias
        return 1/(1+np.exp(-1*z))

    def predict(self, features):
        z = np.dot(features, self.weights) + self.bias
        return np.where(z > 0, 1, 0)

    def Split(self, df, train_fraction): # I used train fraction as a parameter to
        df_train = df.sample(frac = train_fraction)
        df_test = df.iloc[[indices for indices in df.index if indices not in df_tra
        return df_train,df_test
```

In [3]:
```
df = pd.read_csv("diabetes (1).csv")
```

Now i will implement Z-Normalization scalling

In [4]:
```
df_orignial = df
df
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

In [5]:
```
for feature in df.keys():
    if feature != 'Outcome':
        df[feature] = (df[feature] - df[feature].mean())/df[feature].std()
```

In [6]:
```
df # after scalling
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
0	0.639530	0.847771	0.149543	0.906679	-0.692439	0.203880	
1	-0.844335	-1.122665	-0.160441	0.530556	-0.692439	-0.683976	
2	1.233077	1.942458	-0.263769	-1.287373	-0.692439	-1.102537	
3	-0.844335	-0.997558	-0.160441	0.154433	0.123221	-0.493721	
4	-1.141108	0.503727	-1.503707	0.906679	0.765337	1.408828	
...	...	...	...	...	...	...	
763	1.826623	-0.622237	0.356200	1.721613	0.869464	0.115094	
764	-0.547562	0.034575	0.046215	0.405181	-0.692439	0.609757	
765	0.342757	0.003299	0.149543	0.154433	0.279412	-0.734711	
766	-0.844335	0.159683	-0.470426	-1.287373	-0.692439	-0.240048	
767	-0.844335	-0.872451	0.046215	0.655930	-0.692439	-0.201997	

768 rows × 9 columns

In [7]:
```
lr = LogisticRegression()
```

In [8]:
```
df_train,df_test = lr.Split(df, 0.8)
```

In [9]:
```
df_features = df.columns.values.tolist() # Since we know df can be treated as a dic
df_features.pop(8)
x_train = df_train[df_features]
y_train = df_train[['Outcome']]
x_test = df_test[df_features]
y_test = df_test[['Outcome']]
```

In [10]:
```
lr.Fit(x_train, y_train, learning_rate = 0.001)
```

In [11]:
```
lr.weights
```

Out[11]:
```
array([[ 2.42903144e-01],
       [ 7.91197273e-01],
       [-8.59711790e-02],
       [-1.21688914e-02],
       [-1.54426227e-04],
       [ 4.91697280e-01],
       [ 2.31315308e-01],
       [ 2.02858055e-01]])
```

In [12]:
```
lr.bias
```

Out[12]:
```
-0.522703941459954
```

In [13]:
```
y_pred_test = lr.predict(x_test)
y_pred_test.flatten()
```

Out[13]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0])
```

Note : I have not used any seed for randomizing the data while spliting

Now i will calculate the percentage accuracy:

In [14]:
```
percentage = (((lr.predict(x_train) == y_train).sum().values[0])/np.shape((y_train)
percentage = "{:.2f}".format(percentage)
print(f"The percetage accuracy of the model on train data is {percentage} %")
```

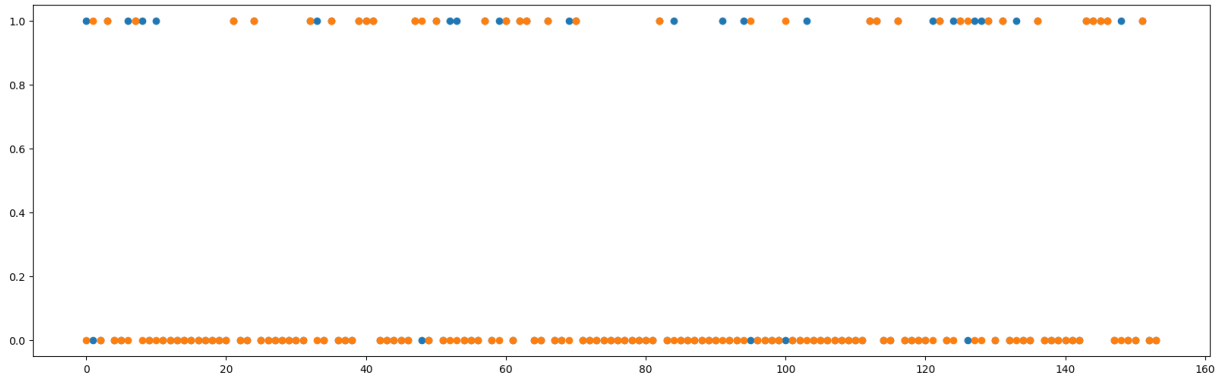
```
The percetage accuracy of the model on train data is 75.08 %
```

In [15]:
```
percentage = (((y_pred_test == y_test).sum().values[0])/np.shape((y_pred_test))[0])
percentage = "{:.2f}".format(percentage)
print(f"The percetage accuracy of the model on test data is {percentage} %")
```

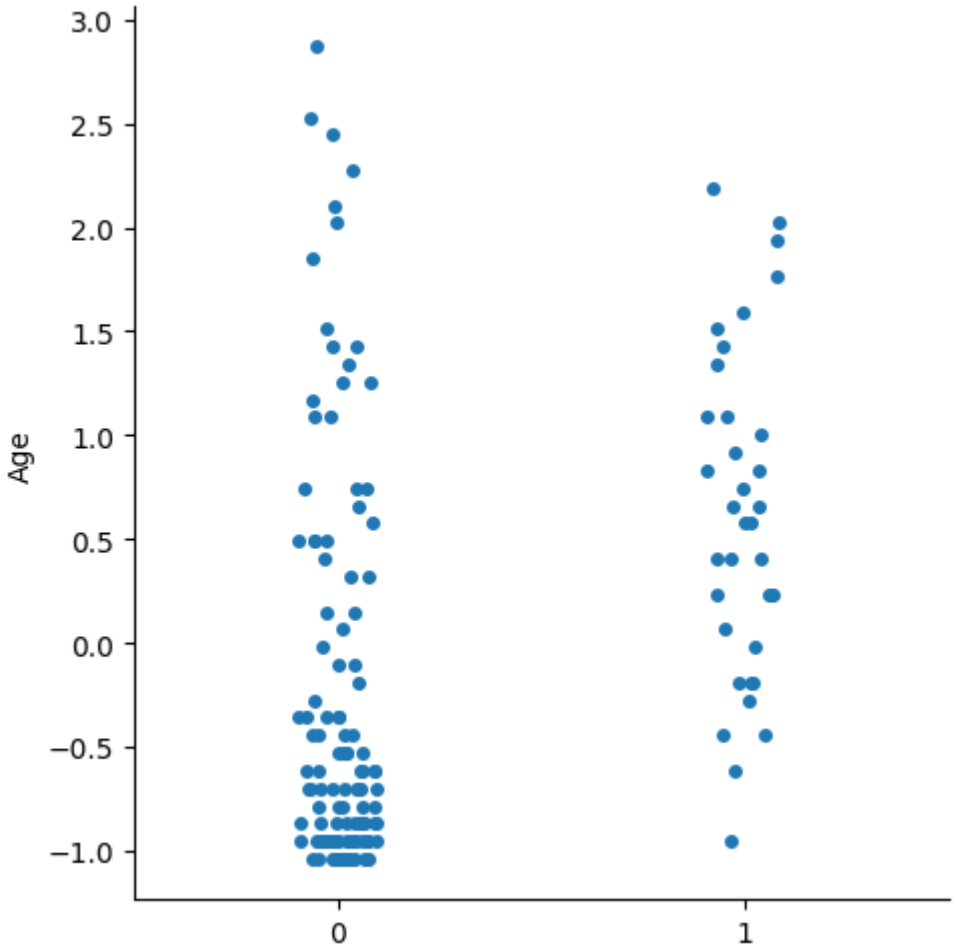
```
The percetage accuracy of the model on test data is 84.42 %
```

Graphs

In [16]:
```
plt.figure(1, figsize=(20, 6))
plt.scatter([i for i in range(np.shape((y_pred_test))[0])], y_pred_test)
plt.scatter([i for i in range(np.shape((y_test))[0])], y_test);
```



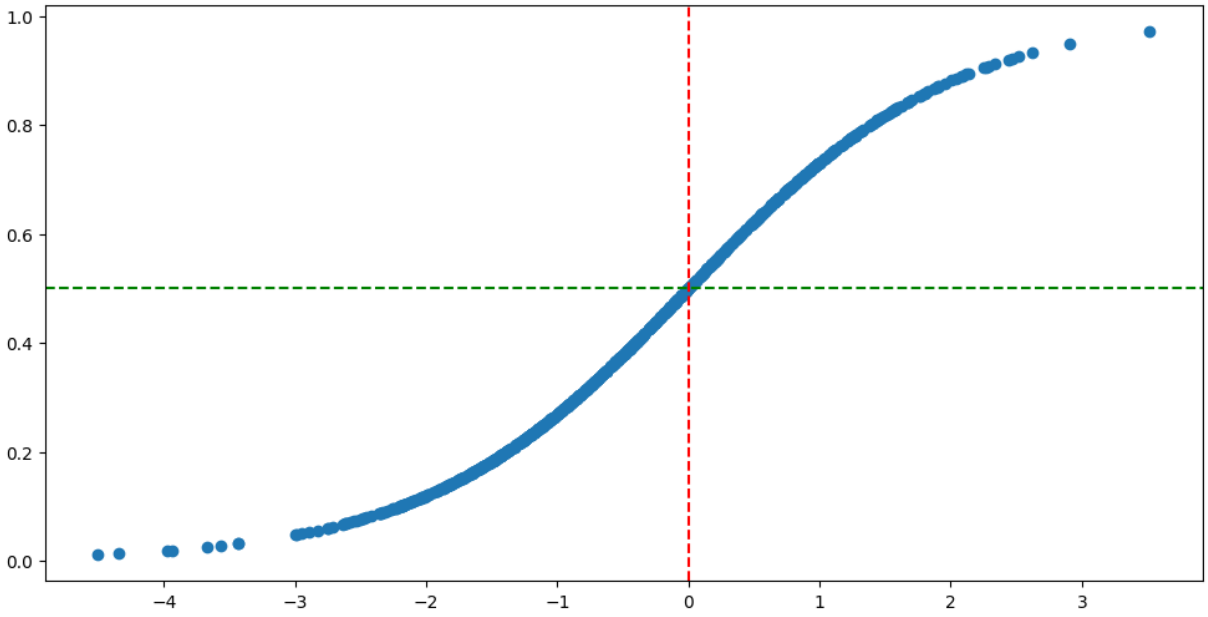
In [17]:
```
sns.catplot(x = y_test.values.ravel(), y = 'Age' , data = df_test);
```



Sigmoid curve

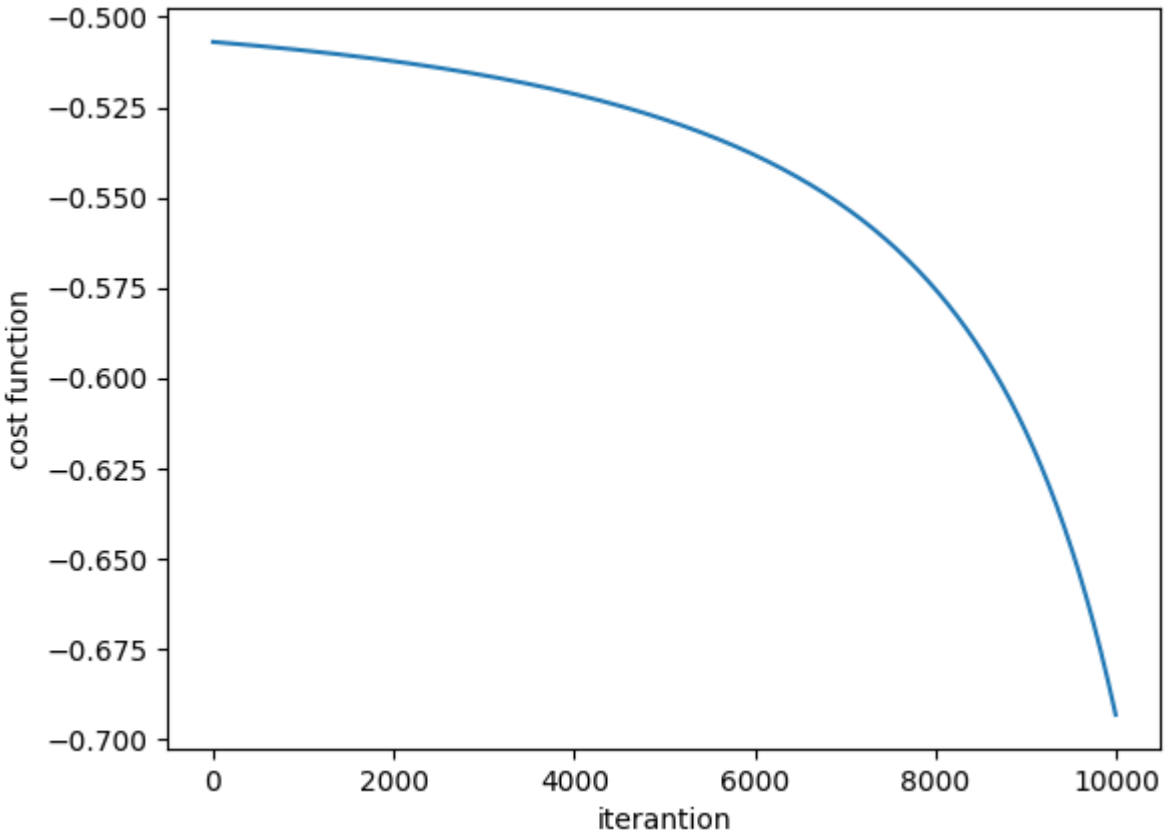
In [18]:
```
z = np.dot(df[df_features], lr.weights) + lr.bias
y = 1/(1+np.exp(-1*z))
```

In [19]:
```
plt.figure(figsize = (12,6))
plt.scatter(z,y)
plt.axvline(x=0, color='red', linestyle='--', label='x=0')
plt.axhline(y=0.5, color='green', linestyle='--', label='y=0.5');
```



Learning Rate (Cost function vs iterations)

In [21]:
```
plt.plot([i for i in range(9999)], lr.cost_values)
plt.xlabel("iterantion")
plt.ylabel("cost function");
# Cost function decreasing iteration by iteration
```



In [ ]: