ML Lab Assignemnt 1

- Sai Karthik AP21110010310 CSE-E

1. Implement Linear Regression and calculate sum of residual error on the following Datasets. x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] y = [1, 3, 2, 5, 7, 8, 8, 9, 10, 12]

i) Compute the regression coefficients using analytic formulation and calculate Sum Squared Error (SSE) and R2 value.

ii) Implement gradient descent (both Full-batch and Stochastic with stopping criteria) on Least Mean Square loss formulation to compute the coefficients of regression matrix and compare the results using performance measures such as R2 SSE etc.

In [24]:

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
```

In [3]:

```
def linear_regression(x, y):
    n = len(x)
    x_mean = np.mean(x)
    y_mean = np.mean(y)

    slope = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x - x_mean) **
    intercept = y_mean - slope * x_mean

    y_pred = slope * x + intercept

    sse = np.sum((y - y_pred) ** 2)

    ss_total = np.sum((y - y_mean) ** 2)
    r2 = 1 - (sse / ss_total)

    return slope, intercept, sse, r2

slope, intercept, sse, r2 = linear_regression(x, y)
```

In [4]:
```
print("Slope:", slope)
print("Intercept:", intercept)
print("Sum of Squared Error (SSE):", sse)
print("R^2 value:", r2)
```

```
Slope: 1.1696969696969697
Intercept: 1.2363636363636363
Sum of Squared Error (SSE): 5.624242424242423
R^2 value: 0.952538038613988
```

In [5]:
```
def linear_regression_gradient_descent(x, y, learning_rate=0.01, epochs=
    n = len(x)
    slope = 0
    intercept = 0

    if batch_size is None:
        batch_size = n

    for epoch in range(epochs):
        if batch_size == n:
            x_batch = x
            y_batch = y
        else:
            indices = np.random.choice(n, batch_size)
            x_batch = x[indices]
            y_batch = y[indices]

        y_pred = slope * x_batch + intercept

        slope_gradient = -(2 / batch_size) * np.sum(x_batch * (y_batch -
        intercept_gradient = -(2 / batch_size) * np.sum(y_batch - y_pred

        slope -= learning_rate * slope_gradient
        intercept -= learning_rate * intercept_gradient

    y_pred = slope * x + intercept

    sse = np.sum((y - y_pred) ** 2)

    y_mean = np.mean(y)
    ss_total = np.sum((y - y_mean) ** 2)
    r2 = 1 - (sse / ss_total)

    return slope, intercept, sse, r2
```

In [10]:
```
slope_batch, intercept_batch, sse_batch, r2_batch = linear_regression_gra
print("\nFull-batch gradient descent:")
print("Slope:", slope_batch)
print("Intercept:", intercept_batch)
print("Sum of Squared Error (SSE):", sse_batch)
print("R2 value:", r2_batch)

slope_stochastic, intercept_stochastic, sse_stochastic, r2_stochastic = 
print("\nStochastic gradient descent:")
print("Slope:", slope_stochastic)
```

```
print("Intercept:", intercept_stochastic)
print("Sum of Squared Error (SSE):", sse_stochastic)
print("R2 value:", r2_stochastic)
```

```
Full-batch gradient descent:
Slope: 1.170263693076768
Intercept: 1.2328099487610318
Sum of Squared Error (SSE): 5.624278989977716
R2 value: 0.9525377300423822

Stochastic gradient descent:
Slope: 1.1684728386473255
Intercept: 1.275337914338942
Sum of Squared Error (SSE): 5.63556557313237
R2 value: 0.9524424846149168
```

2. Download Boston Housing Rate Dataset. Analyse the input attributes and find out the attribute that best follow the linear relationship with the output price. Implement both the analytic formulation and gradient descent (Full-batch, stochastic) on LMS loss formulation to compute the coefficients of regression matrix and compare the results.S

In [44]:

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
from sklearn.preprocessing import StandardScaler

boston = fetch_openml(name='boston', version=1)

X = boston.data
y = boston.target

X = np.array(X)


scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

correlations = np.abs(np.corrcoef(X.T, y)[:,-1])[:-1]

df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target

correlation_matrix = df.corr()
correlation_with_price = correlation_matrix['PRICE'].abs().sort_values(a
print("Correlation of input attributes with the output price:")
print(correlation_with_price)
X_train_with_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]
theta_analytic = np.linalg.inv(X_train_with_bias.T.dot(X_train_with_bias
print()
```

```
def gradient_descent(X, y, theta, alpha, num_iters):
    m = len(y)
    for i in range(num_iters):
        gradient = (1/m) * X.T.dot(X.dot(theta) - y)
        theta -= alpha * gradient
    return theta

alpha = 0.01
num_iters = 1000

theta_full_batch = np.random.randn(X_train_with_bias.shape[1])
theta_full_batch = gradient_descent(X_train_with_bias, y_train, theta_ful

def stochastic_gradient_descent(X, y, theta, alpha, num_iters):
    m = len(y)
    for i in range(num_iters):
        for j in range(m):
            random_index = np.random.randint(m)
            Xj = X[random_index:random_index+1]
            yj = y[random_index:random_index+1]
            gradient = Xj.T.dot(Xj.dot(theta) - yj)
            theta -= alpha * gradient
    return theta

theta_stochastic = np.random.randn(X_train_with_bias.shape[1])
theta_stochastic = stochastic_gradient_descent(X_train_with_bias, y_train

def calculate_r2(y_true, y_pred):
    mean_y = np.mean(y_true)
    total_sum_squares = np.sum((y_true - mean_y) ** 2)
    residual_sum_squares = np.sum((y_true - y_pred) ** 2)
    r2 = 1 - (residual_sum_squares / total_sum_squares)
    return r2

def calculate_mse(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    return mse

X_test_with_bias = np.c_[np.ones((X_test.shape[0], 1)), X_test]
y_pred_analytic = X_test_with_bias.dot(theta_analytic)
y_pred_full_batch = X_test_with_bias.dot(theta_full_batch)
y_pred_stochastic = X_test_with_bias.dot(theta_stochastic)

r2_analytic = calculate_r2(y_test, y_pred_analytic)
r2_full_batch = calculate_r2(y_test, y_pred_full_batch)
r2_stochastic = calculate_r2(y_test, y_pred_stochastic)

mse_analytic = calculate_mse(y_test, y_pred_analytic)
mse_full_batch = calculate_mse(y_test, y_pred_full_batch)
mse_stochastic = calculate_mse(y_test, y_pred_stochastic)
print()
print("R^2 scores:")
print("Analytic solution:", r2_analytic)
print("Full-batch gradient descent:", r2_full_batch)
print("Stochastic gradient descent:", r2_stochastic)
print()
print("\nMSE scores:")
```

```
print("Analytic solution:", mse_analytic)
print("Full-batch gradient descent:", mse_full_batch)
print("Stochastic gradient descent:", mse_stochastic)
```

```
Correlation of input attributes with the output price:
PRICE      1.000000
LSTAT      0.737663
RM         0.695360
PTRATIO    0.507787
INDUS      0.483725
TAX        0.468536
NOX        0.427321
CRIM       0.388305
RAD        0.381626
AGE        0.376955
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
Name: PRICE, dtype: float64
R^2 scores:
Analytic solution: 0.6687594935356311
Full-batch gradient descent: 0.6514150559454255
Stochastic gradient descent: 0.5589752905161083

MSE scores:
Analytic solution: 24.291119474973584
Full-batch gradient descent: 25.563052700251475
Stochastic gradient descent: 32.34201041363614
```