# Program 1: Implementation of Linear Regression on California Housing Dataset

## Aim

To implement simple linear regression on the California Housing dataset using Gradient Descent and the Normal Equation, and to compare the two using MSE and $R^2$ score.

## Algorithm

1. Start.
2. Import the required libraries (NumPy, Matplotlib, Scikit-learn).
3. Load the California Housing dataset.
4. Select one feature (AveRooms) as the input (X) and the house price as the output (y).
5. Split the data into training and testing sets (80:20) and standardize the feature using the training mean and standard deviation.
6. Initialize the parameters (theta), learning rate and number of iterations.
7. Gradient Descent: compute the gradients of the cost, update theta, and repeat until all iterations are completed.
8. Display the intercept and slope obtained from Gradient Descent.
9. Normal Equation: add a column of ones to the input data and calculate theta using the Normal Equation.
10. Predict the output on the test data using the obtained parameters.
11. Calculate the Mean Squared Error (MSE) and $R^2$ score for both methods.
12. Plot the data points and the fitted regression line.
13. Stop.

## Program

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

housing = fetch_california_housing()

X = housing.data[:, housing.feature_names.index("AveRooms")]
y = housing.target

X = X.reshape(-1, 1)

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

mean = X_train.mean()
std = X_train.std()
X_train_scaled = (X_train - mean) / std
X_test_scaled = (X_test - mean) / std

m = len(X_train_scaled)
X_b = np.c_[np.ones((m, 1)), X_train_scaled]
theta = np.zeros(2)
learning_rate = 0.01
iterations = 1000

for i in range(iterations):
    gradients = (2/m) * X_b.T.dot(X_b.dot(theta) - y_train)
    theta -= learning_rate * gradients

print("Gradient Descent Parameters:")
print("Intercept:", theta[0])
print("Slope:", theta[1])

X_test_b = np.c_[np.ones((len(X_test_scaled), 1)), X_test_scaled]
y_pred_gd = X_test_b.dot(theta)
```

```
X_train_normal = np.c_[np.ones((len(X_train), 1)), X_train]
theta_normal = np.linalg.inv(
    X_train_normal.T.dot(X_train_normal)
).dot(X_train_normal.T).dot(y_train)

print("\nNormal Equation Parameters:")
print("Intercept:", theta_normal[0])
print("Slope:", theta_normal[1])

X_test_normal = np.c_[np.ones((len(X_test), 1)), X_test]
y_pred_normal = X_test_normal.dot(theta_normal)

print("\n--- Gradient Descent ---")
print("MSE:", mean_squared_error(y_test, y_pred_gd))
print("R2 Score:", r2_score(y_test, y_pred_gd))

print("\n--- Normal Equation ---")
print("MSE:", mean_squared_error(y_test, y_pred_normal))
print("R2 Score:", r2_score(y_test, y_pred_normal))

plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, alpha=0.5, label="Actual Data")

sorted_idx = np.argsort(X_test[:, 0])
plt.plot(
    X_test[sorted_idx],
    y_pred_normal[sorted_idx],
    linewidth=2,
    label="Regression Line"
)

plt.xlabel("Average Number of Rooms (AveRooms)")
plt.ylabel("Median House Value")
plt.title("Simple Linear Regression on California Housing Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

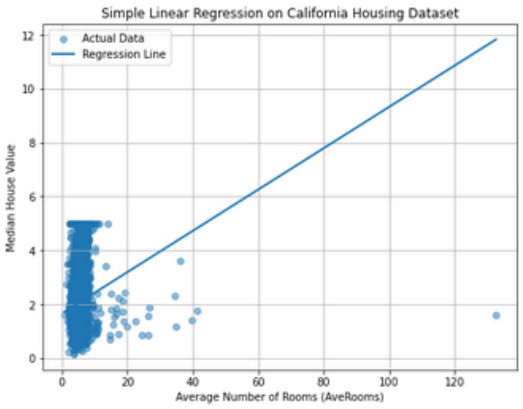
### Output

```
Gradient Descent Parameters:
Intercept: 2.071946933891858
Slope: 0.18323881682653

Normal Equations Parameters:
Intercept: 1.6547622685968426
Slope: 0.07675558963126718

--- Gradient Descent ---
MSE: 1.2923314439327267
R2 Score: 0.013795337645229111

--- Normal Equations ---
MSE: 1.2923314440807299
R2 Score: 0.013795337532284901
```



## Program 2: Implementation of Ridge and Lasso Regression on Diabetes Dataset

### Aim

To implement Ridge and Lasso regression on the Diabetes dataset and compare the performance of these regularized models with standard linear regression using MSE and $R^2$ score.

### Algorithm

1. Start the program.
2. Import the required libraries.
3. Load the Diabetes dataset.
4. Split the dataset into training and testing sets.
5. Create and train the Linear Regression model using the training data.
6. Predict the output using the Linear Regression model.
7. Calculate the Mean Squared Error (MSE) and $R^2$ score for the Linear Regression model.
8. Create and train the Ridge Regression model.
9. Predict the output using the Ridge Regression model.
10. Calculate the MSE and $R^2$ score for the Ridge Regression model.
11. Create and train the Lasso Regression model.
12. Predict the output using the Lasso Regression model.
13. Calculate the MSE and $R^2$ score for the Lasso Regression model.
14. Compare the performance of the three models using the MSE and $R^2$ score.
15. Plot a bar graph showing the $R^2$ score of Linear, Ridge and Lasso Regression models.
16. Stop.

### Program

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score

data = load_diabetes()
X = data.data
y = data.target

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

lr = LinearRegression()
lr.fit(X_train, y_train)
y1 = lr.predict(X_test)
print("Linear Regression")
print("MSE:", mean_squared_error(y_test, y1))
print("R2:", r2_score(y_test, y1))

ridge = Ridge()
ridge.fit(X_train, y_train)
y2 = ridge.predict(X_test)
print("\nRidge Regression")
print("MSE:", mean_squared_error(y_test, y2))
print("R2:", r2_score(y_test, y2))

lasso = Lasso()
lasso.fit(X_train, y_train)
y3 = lasso.predict(X_test)
print("\nLasso Regression")
print("MSE:", mean_squared_error(y_test, y3))
print("R2:", r2_score(y_test, y3))
```

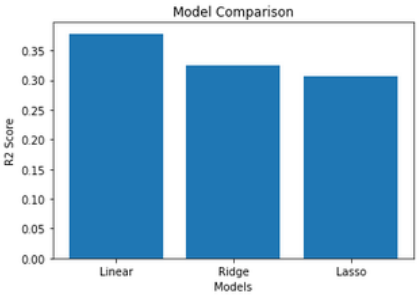
```
models = ["Linear", "Ridge", "Lasso"]
r2 = [
    r2_score(y_test, y1),
    r2_score(y_test, y2),
    r2_score(y_test, y3)
]
plt.bar(models, r2)
plt.xlabel("Models")
plt.ylabel("R2 Score")
plt.title("Model Comparison")
plt.show()
```

### Output

```
Linear Regression
MSE: 3389.721585186079
R2: 0.37830987257315696

Ridge Regression
MSE: 3679.4545360154902
R2: 0.32517155115228313

Lasso Regression
MSE: 3778.253718833672
R2: 0.30705133832290665
```



# Program 3: MLE and MAP Estimation of Multinomial Distribution using 20 Newsgroups

## Aim

To use MLE and MAP to estimate the parameters of a multinomial distribution on the 20 Newsgroups dataset and explore the impact of the prior on the estimation.

## Algorithm

1. Start the program.
2. Import the required libraries.
3. Load the 20 Newsgroups dataset.
4. Convert the text documents into numerical features using CountVectorizer.
5. Split the data into training and testing sets.
6. Create the Multinomial Naive Bayes (MLE) model.
7. Train the MLE model using the training data.
8. Predict the class labels for the test data.
9. Calculate and display the accuracy of the MLE model.
10. Create the Multinomial Naive Bayes (MAP) model using Laplace smoothing.
11. Train the MAP model using the training data.
12. Predict the class labels for the test data.
13. Calculate and display the accuracy of the MAP model.
14. Compare the accuracy of the MLE and MAP models.
15. End the program.

## Program

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

categories = ['alt.atheism', 'comp.graphics', 'sci.med']
train = fetch_20newsgroups(subset='train', categories=categories,
                           shuffle=True, random_state=42)
test = fetch_20newsgroups(subset='test', categories=categories,
                          shuffle=True, random_state=42)

vectorizer = CountVectorizer(stop_words='english')
X_train = vectorizer.fit_transform(train.data)
X_test = vectorizer.transform(test.data)
y_train = train.target
y_test = test.target

mle = MultinomialNB(alpha=1e-10)
mle.fit(X_train, y_train)
pred1 = mle.predict(X_test)
print("MLE (Approximation)")
print("Accuracy:", accuracy_score(y_test, pred1))

map_model = MultinomialNB(alpha=1.0)
map_model.fit(X_train, y_train)
pred2 = map_model.predict(X_test)
print("\nMAP (Laplace Smoothing)")
print("Accuracy:", accuracy_score(y_test, pred2))
```

## Output

```
MLE (Approximation)
Accuracy: 0.9411231884057971

MAP (Laplace Smoothing)
Accuracy: 0.9565217391304348
```

# Program 4: Implementation of Logistic Regression on Pima Indians Diabetes Dataset

## Aim

To implement a logistic regression model to predict the likelihood of diabetes using the Pima Indians Diabetes dataset and evaluate it using accuracy, precision, recall and F1-score.

## Algorithm

1. Start the program.
2. Import the required libraries.
3. Load the Pima Indians Diabetes dataset.
4. Separate the input features and target variable.
5. Split the dataset into training and testing sets.
6. Create the Logistic Regression model.
7. Train the model using the training data.
8. Predict the class labels for the test data.
9. Calculate the Accuracy, Precision, Recall and F1 Score.
10. Display the performance metrics.
11. End the program.

## Program

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

url = ("https://raw.githubusercontent.com/jbrownlee/Datasets/"
       "master/pima-indians-diabetes.data.csv")
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
           'Insulin', 'BMI', 'DPF', 'Age', 'Outcome']
data = pd.read_csv(url, names=columns)

X = data.drop('Outcome', axis=1)
y = data['Outcome']

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

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy :", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall   :", recall_score(y_test, y_pred))
print("F1 Score :", f1_score(y_test, y_pred))
```

## Output

```
Accuracy : 0.7532467532467533
Precision: 0.6444444444444445
Recall   : 0.5686274509803921
F1 Score : 0.6041666666666666
```

# Program 5: Implementation of K-Nearest Neighbors for Image Classification using Fashion MNIST Dataset

## Aim

To implement the K-Nearest Neighbors (KNN) algorithm for image classification on the Fashion MNIST dataset and analyze the impact of different values of K on model performance.

## Algorithm

1. Start the program.
2. Import the required libraries.
3. Load the Fashion MNIST dataset.
4. Select 5,000 training images and 1,000 testing images.
5. Convert the images into one-dimensional arrays.
6. Normalize the pixel values between 0 and 1.
7. Select different values of K: 3, 7 and 15.
8. Create a KNN model for each value of K.
9. Train the KNN model using the training data.
10. Predict the classes of the test images.
11. Calculate the accuracy of the prediction.
12. Display the accuracy for each value of K.
13. Compare the results for K = 3, 7 and 15.
14. Stop the program.

## Program

```
from sklearn.datasets import fetch_openml
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

print("Loading dataset...")
X, y = fetch_openml('Fashion-MNIST', version=1, return_X_y=True,
                    as_frame=False)

X_train, X_test = X[:5000] / 255.0, X[60000:61000] / 255.0
y_train, y_test = y[:5000], y[60000:61000]

for k in [3, 7, 15]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred = knn.predict(X_test)
    print(f"K={k} Accuracy: {accuracy_score(y_test, pred):.2f}")
```

## Output

```
Loading dataset...
K=3 Accuracy: 0.81
K=7 Accuracy: 0.82
K=15 Accuracy: 0.82
```