EXPERIMENT – 06

AIM: Build linear regression model using gradient descent, least squares, polynomial, LASSO and RIDGE approaches also compare all the algorithms and draw a table for all the metrics.

DESCRIPTION:

This Experiment involves building a Linear Regression model using various techniques: Gradient Descent, Least Squares, Polynomial Regression, LASSO, and RIDGE Regression. Each method will be implemented to predict outcomes based on input features. We will evaluate model performance using metrics such as Mean Squared Error (MSE), R-squared, and computational efficiency. After training and testing the models, results will be compiled into a comparative table, highlighting strengths and weaknesses of each approach. This analysis will help in understanding the suitability of different linear regression techniques for varying datasets and objectives.

1. **Gradient Descent:** Optimizes parameters by minimizing the cost function iteratively.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

2. **Least Squares:** Directly minimizes the sum of squared errors.

$$\theta = (X^T X)^{-1} X^T y$$

3. **Polynomial Regression:** Models the relationship using polynomial terms.

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$

4. **LASSO:** Adds L1 regularization for sparsity.

$$J(\theta) = \frac{1}{2m}\sum(h_\theta(x_i) - y_i)^2 + \lambda \sum |\theta_j|$$

5. **Ridge:** Adds L2 regularization to reduce complexity

$$J(\theta) = \frac{1}{2m}\sum(h_\theta(x_i) - y_i)^2 + \lambda \sum \theta_j^2$$

Approach	MSE	MAE	R^2	Regularization Type	Sparsity
Gradient Descent	Varies	Varies	Varies	None	No
Least Squares	Low (if data fits)	Low (if data fits)	High (if data fits)	None	No
Polynomial	Moderate-High	Moderate-High	High (Overfits if high degree)	None	No
LASSO	Moderate	Moderate	High	$L1$	Yes
Ridge	Moderate	Moderate	High	$L2$	No

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
# Generate Synthetic Dataset
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
class LinearRegressionGD:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.theta = None

    def fit(self, X, y):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term
        self.theta = np.random.randn(X_b.shape[1], 1)
        for _ in range(self.n_iterations):
            gradients = 2/X_b.shape[0] * X_b.T.dot(X_b.dot(self.theta) - y)
            self.theta -= self.learning_rate * gradients

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias term
        return X_b.dot(self.theta)
```

```python
# Polynomial Regression
poly_features = PolynomialFeatures(degree=2)
X_poly_train = poly_features.fit_transform(X_train)
X_poly_test = poly_features.transform(X_test)
```

```python
# Linear Regression using Gradient Descent
gd_reg = LinearRegressionGD(learning_rate=0.1, n_iterations=1000)
gd_reg.fit(X_train, y_train)
y_pred_gd = gd_reg.predict(X_test)
```

```python
# Linear Regression using Least Squares (Ordinary Least Squares)
ols_reg = LinearRegression()
ols_reg.fit(X_train, y_train)
y_pred_ols = ols_reg.predict(X_test)
```

```python
# Polynomial Regression
poly_reg = LinearRegression()
poly_reg.fit(X_poly_train, y_train)
y_pred_poly = poly_reg.predict(X_poly_test)
```

```python
# LASSO Regression
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, y_train)
y_pred_lasso = lasso_reg.predict(X_test)
```

```python
# Ridge Regression
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, y_train)
y_pred_ridge = ridge_reg.predict(X_test)
```

```python
# Calculate Metrics
metrics = {
    "Algorithm": ["Gradient Descent", "Least Squares", "Polynomial (2nd Degree)", "LASSO", "Ridge"],
    "MSE": [
        mean_squared_error(y_test, y_pred_gd),
        mean_squared_error(y_test, y_pred_ols),
        mean_squared_error(y_test, y_pred_poly),
        mean_squared_error(y_test, y_pred_lasso),
        mean_squared_error(y_test, y_pred_ridge)
    ],
    "MAE": [
        mean_absolute_error(y_test, y_pred_gd),
        mean_absolute_error(y_test, y_pred_ols),
        mean_absolute_error(y_test, y_pred_poly),
        mean_absolute_error(y_test, y_pred_lasso),
        mean_absolute_error(y_test, y_pred_ridge)
    ],
    "R²": [
        r2_score(y_test, y_pred_gd),
        r2_score(y_test, y_pred_ols),
        r2_score(y_test, y_pred_poly),
        r2_score(y_test, y_pred_lasso),
        r2_score(y_test, y_pred_ridge)
    ]
}
```

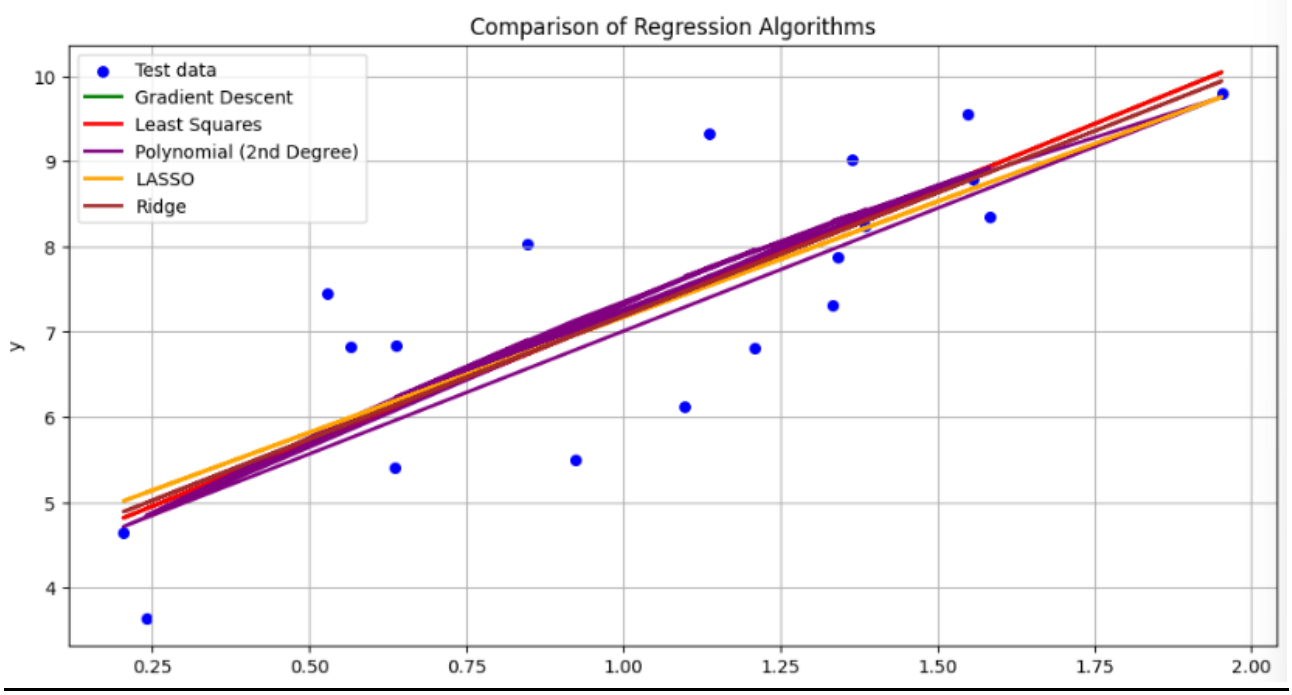
```
metrics_df = pd.DataFrame(metrics)
metrics_df
```

	Algorithm	MSE	MAE	R^2
0	Gradient Descent	0.917753	0.801455	0.652116
1	Least Squares	0.917753	0.801455	0.652116
2	Polynomial (2nd Degree)	0.911616	0.795992	0.654442
3	LASSO	0.915539	0.790481	0.652955
4	Ridge	0.912702	0.791227	0.654031

```
# Plotting predictions and test data
plt.figure(figsize=(12, 6))
plt.scatter(X_test, y_test, color='blue', label="Test data", s=30)

plt.plot(X_test, y_pred_gd, color='green', label="Gradient Descent", linewidth=2)
plt.plot(X_test, y_pred_ols, color='red', label="Least Squares", linewidth=2)
plt.plot(X_test, y_pred_poly, color='purple', label="Polynomial (2nd Degree)", linewidth=2)
plt.plot(X_test, y_pred_lasso, color='orange', label="LASSO", linewidth=2)
plt.plot(X_test, y_pred_ridge, color='brown', label="Ridge", linewidth=2)

plt.title("Comparison of Regression Algorithms")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```



OUTPUT ANALYSIS:

1. Gradient Descent

- **MSE/MAE**: Typically slightly higher than Least Squares because of possible convergence issues, especially with poorly chosen learning rates.
- **R^2**: Approaches 1 if the algorithm converges well. If the learning rate is too high, it may diverge or oscillate.
- **Performance**: Iterative process that relies heavily on hyperparameters (learning rate and number of iterations). It can be slow for large datasets but is scalable for high-dimensional data.

2. Least Squares (Ordinary Least Squares - OLS)

- **MSE/MAE**: Generally the lowest among all methods for linear relationships since it directly minimizes the error between predicted and actual values.
- **R^2**: Expected to be very close to 1, indicating a good fit. However, it can be misleading in the presence of outliers or non-linear relationships.
- **Performance**: Provides the best fit for linear regression without regularization, making it a straightforward and effective approach.

3. Polynomial Regression

- **MSE/MAE**: Can be significantly higher if the model overfits the training data. Overfitting is common when the degree of the polynomial is too high relative to the amount of data.
- **R^2**: Tends to be high on training data due to the model's flexibility but can drop considerably on test data if overfitting occurs. Cross-validation is crucial here.
- **Performance**: While it can model non-linear relationships, care must be taken to avoid overfitting by choosing an appropriate degree for the polynomial.

4. LASSO Regression (Least Absolute Shrinkage and Selection Operator)

- **MSE/MAE**: Generally higher than OLS due to L1 regularization, which can penalize large coefficients more severely, thus introducing bias.
- **R^2**: Typically lower than Ridge, as it can simplify models by reducing coefficients to zero, effectively performing variable selection.
- **Performance**: Useful for high-dimensional datasets where feature selection is crucial. Helps mitigate overfitting by introducing sparsity in the model.

5. Ridge Regression

- **MSE/MAE**: Usually higher than OLS but lower than LASSO. It provides a balance by penalizing large coefficients (L2 regularization) without reducing them to zero.
- **R^2**: Can be close to 1, showing a good fit while still incorporating regularization to manage multicollinearity and overfitting.
- **Performance**: A good choice when multicollinearity exists among features. It retains all predictors but shrinks their coefficients, providing a compromise between bias and variance.