Linear Regression From Scratch

This is a documentation of me implementing **Linear Regression** from scratch using only **numpy**

```
import numpy as np
import matplotlib.pyplot as plt
```

- Importing the necessary library
  - **NumPy** for array and matrices manipulation
  - **Matplotlib** for plotting graphs and visualization

Generating dummy data

```
def generate_dummy_data(n=100, p=3, noise_std=1.0, seed=420):
    np.random.seed(seed)

    X_raw = np.random.randn(
        n, p - 1
    )
    intercept = np.ones((n, 1))

    X = np.hstack([intercept, X_raw])

    beta_true = np.array(
        [3] + [2] * (p - 1)
    )
    noise = np.random.randn(n) * noise_std

    y = X @ beta_true + noise

    return X, y, beta_true, noise
```

- Generating dummy data to test our **Linear Regression** functions which takes this form :

$$Y = X\beta + \varepsilon$$

- `X_raw` a matrix of **dimensions** $(n \times p - 1)$, represent our **observations**
- `intercept` a vector assumed to contain only 1 for testing purposes (will change on real data)
- `X` represent X the **matrix** that contain both the `X_raw` and the `intercept`
- `beta_true` array is the true values of the vector $\beta_0 = 3$ $\beta_1 \dots \beta_p = 2$
- `noise` randomly generated **noise** to simulate real life noise and **bias**
- `y` is the predicted response given by our model

Phase 1 : Core Implementation

The First thing to do in almost every machine learning algorithm is to estimate the **coefficients** β to minimize the **Loss Function** which is in the Linear Regression case MSE

$$MSE = \frac{1}{n}\sum^{n}(y_i - \hat{y_i})^2$$

There is mainly two methods to estimate the coefficients β :

- **Ordinary Least Squares**
- **Gradient Descent**

Ordinary Least Squares (OLS) :

```
def ols_estimate(X, Y):

    X_transpose = X.T
```

```
    gram_matrix = X_transpose @ X

    if not is_invertible(gram_matrix):
            raise ValueError(f"The gram matrix with shape{gram_matrix.shape} is
sningular")

    beta_hat = (
        np.linalg.inv(gram_matrix) @ X_transpose @ Y
    )

    return beta_hat, gram_matrix
```

The Closed-Form formula for $\hat{\beta}$:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

- Its derived from minimizing the Residual sum squared $\sum e^T e$ Ordinary Least Squares
- The **OLS** works great on small to medium sized datasets
- `gram_matrix` is the **inner products** of the matrix X which represents the relationship between vectors in a set

$$\text{Gram matrix} = X^T X$$

Gradient Descent

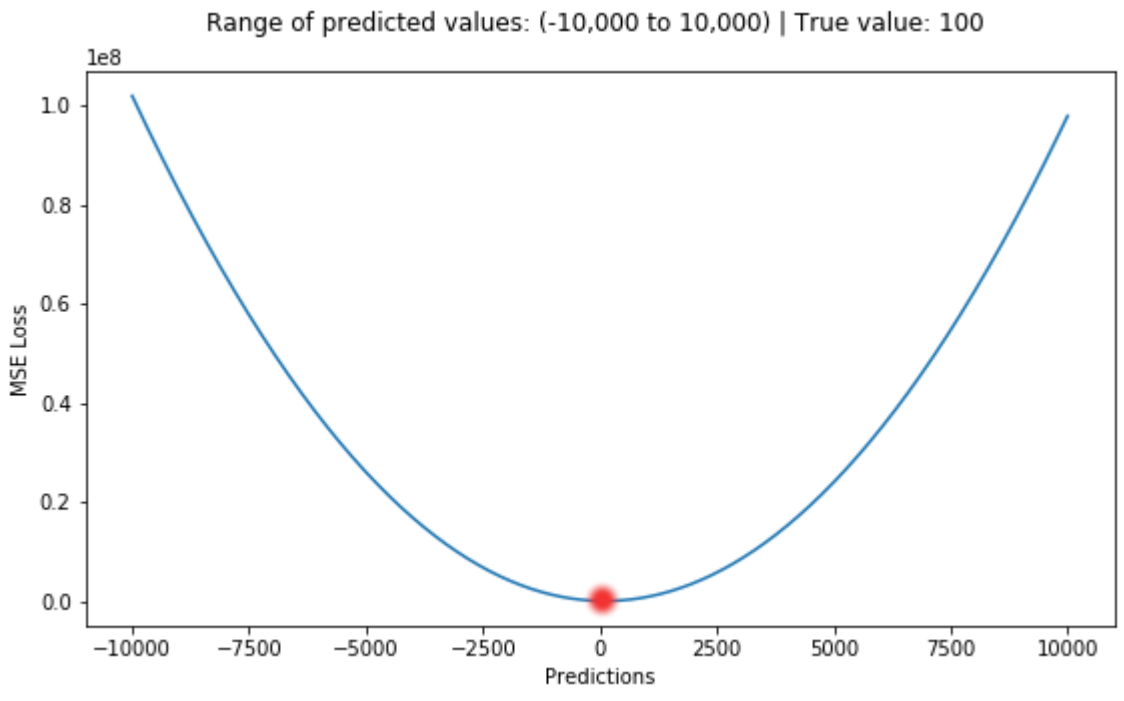
The Gradient Descent Algorithm came to fix a problem in the **OLS** where as we saw in the closed from we calculate the **inverse** matrix for the `gram_matrix`

$$(X^T X)^{-1}$$

- When dealing with large datasets its becomes very expansive to compute, Here where the Gradient Descent comes by using the two very important concepts :
  - Gradients Vectors
  - Loss Function

Loss Function MSE

Its a Known that the **Mean Squared Error** Graph In Linear Regression takes on a **U-Shape**



IF YOU READING THIS CURRENTLY WRITING THE REST OF THE DOCUMENTATION CHECK IN AFTER SOME HOURS