Deriving Linear Regression using Geometry

Follow Mukesh Manral, Offering free aggregated Data Science content for Working and Aspiring data professionals

Also known as:

`1. Ordinary Least Square:(OLS)`

`2. Linear Least Square:(LLS)`

This is real Regression Algorithm, not like Logistic Regression(classification algo)

`Here:`

$$D = (x_i, y_i)_{i=1}^{n}$$

`and`

$x_i \in \mathbb{R}$

`Regression setting, to focus on:`

$y_i \in \mathbb{R}$

`Classification setting, to not focus on:`

$y_i \in (-1, +1$

- $\mathbb{R}$
  is Real number say weight it can be 80.9 kg

Objective of Logistic Regression

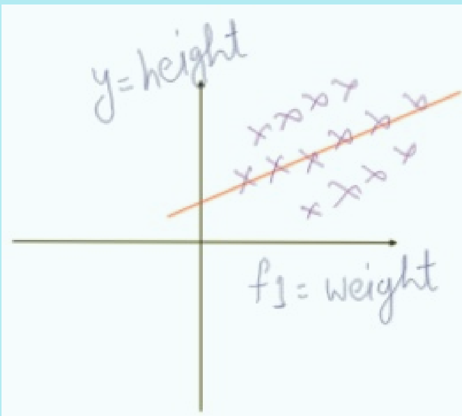
`Find a Line/Plane/Hyperplane that best fit's given datapoints`

`Equation of Line:`

$$y = mx + c$$

Say we have first feature `f1 as weight` and target `y as height`, generall data tells that as height increases weight increases, it can be represented as:

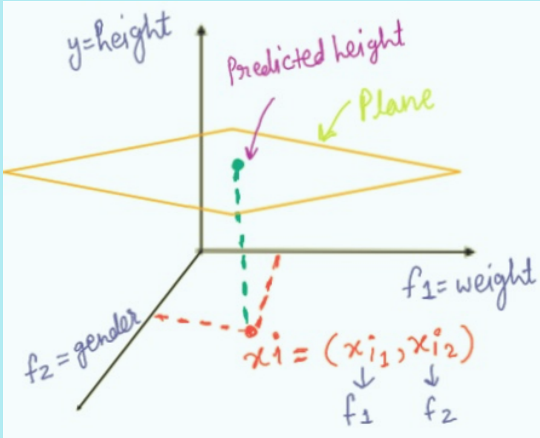
`See in 2-D Space:`



`For 2D Space:`

$$height = (w_1 * weight$$

`See in 3-D Space:`



`For 3D Space:`

$$height = (w_1 * f_1$$

`Can be seen as:`

$$y_i = (w_1 * x_{i1} + w_2 * x_{i2} + w_0)$$

`Similar to equation of line/plane:`

$$y_i = (W^T * x_i + w_0)$$

`Objective:` `Find a Line/Plane/Hyperplane that best fit's given datapoints`

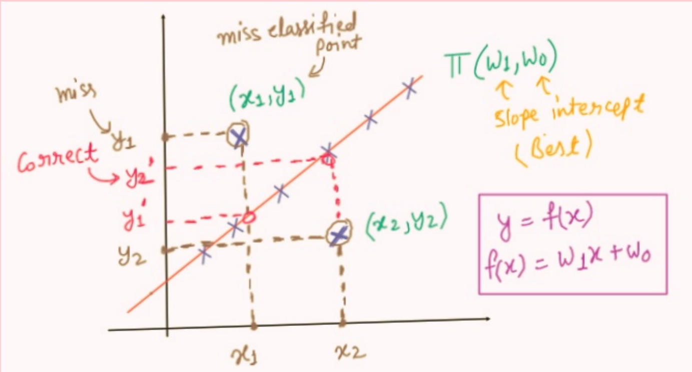
`What Best fit means??`

`Let's say we have 1 feature (x) and we want to predict (y) given (x)` , `We are trying to predict for:`

$$y = f(x)$$

`Here, f(x) form will be of a line:`

$$f(x) = (w_1 x + w_0)$$



`For points (x1), y_hat is not exactly equall to y_1, see in above image, given as:`

$$f(x_1) = (\hat{y}_1) \neq y_1$$

`same for (x2):`

$$f(x_2) = (\hat{y}_2) \neq y_2$$

we can say for the perfet plane(pi) with some slope(w1) and intersept of (w0), points x1 and x2 have some errors, Error representation for point (x_1):

$$err = (y_1 - \hat{y}_1$$

Error representation for point (x_2):

$$err = (y_2 - \hat{y}_2$$

- If we will find error for any other point which is on the line than error will be 0

Keeping these points in head we can write some Optimization Problem

Given some error we have to best fit the plane, which means Error must be least for the best plane

Minimize sum of Errors for all the points i.e. across training data

$$min \sum_i err_i$$

Error = what model is saying - what true value is

Best fit Line is one which minimizes sum of errors

Error can be of two types, Positive Error and Negative Error, to deal with it we have to take square of Error

Equation of Plane not Passing through Origin:

$$\pi : W^T x + W_0 = 0$$

here:

W : vector, W_0 : scalar

AIM is to find optimal (W,W0) such that it minimizes the entire error of dataset:

$$(W^*, {W^*}_0$$

here:

$$\hat{y}_i = f(x_i) = (W^T x_i + x_0)$$

Final Optimization Problem of Logistic Regression

$$(W^*, {W^*}_0$$

Linear Regression is also known as Ordinary Least Square(OLS) or Linear Least Square(LLS)

Linear Term:

$$(W^T x_i + x_0)$$

`Sqared Loss Term:`

$$(y_i - (W^T x_i + x_0))^2$$

Regularization Linear Regression

`Regularization is same as Logistic Regression`

$$(W^*, {W^*}_0$$

`L1 Reg here to creates sparcity and kicks unwanted features to 0`

Loss Minimization Perspective of Linear Regression

`Main Equation:`

$$(W^*, {W^*}_0$$

`here:`

$$f(x_i) = (W^T x_i + x_0)$$

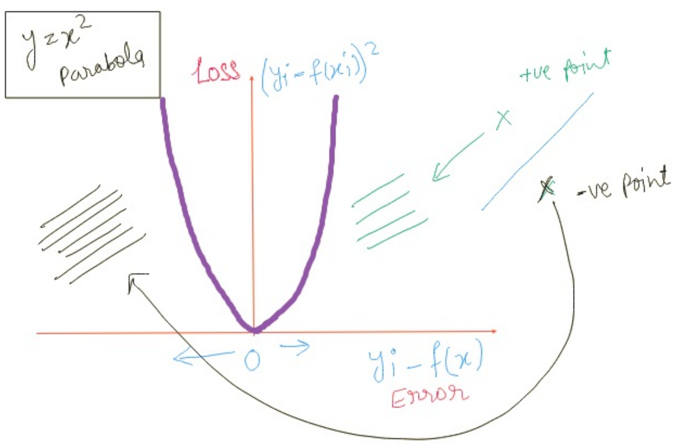
`Equation can be written as:`

$$(W^*, {W^*}_0$$

`Remember that equation of Hyperbola is:`

$$y = x^2$$

`Loss function:`



`Remember in Logistic Regression if Z_i value be -ve than point was predicted miss classified and if +ve perfectly classified:`

$$Z_i = y_i(f(x_i))$$

`here:`

$$f(x_i) = W^T x_i$$

`Here Z_i tells we are classifing correctly or incorrectly`

- In classification we take +ve or -ve side

`Remember in Liner Regression if error is less best fit line:`

$$err = y_i - f(x_i)$$

`or`

$$err = (y_i - \hat{y}_i)$$

`here:`

$$f(x_i) = W^T x_i + W_0$$

`x-axis shows error, see above image for reference`

`Loss Term in Regression is` `Squared Loss` i.e. makes a parabola, see above image

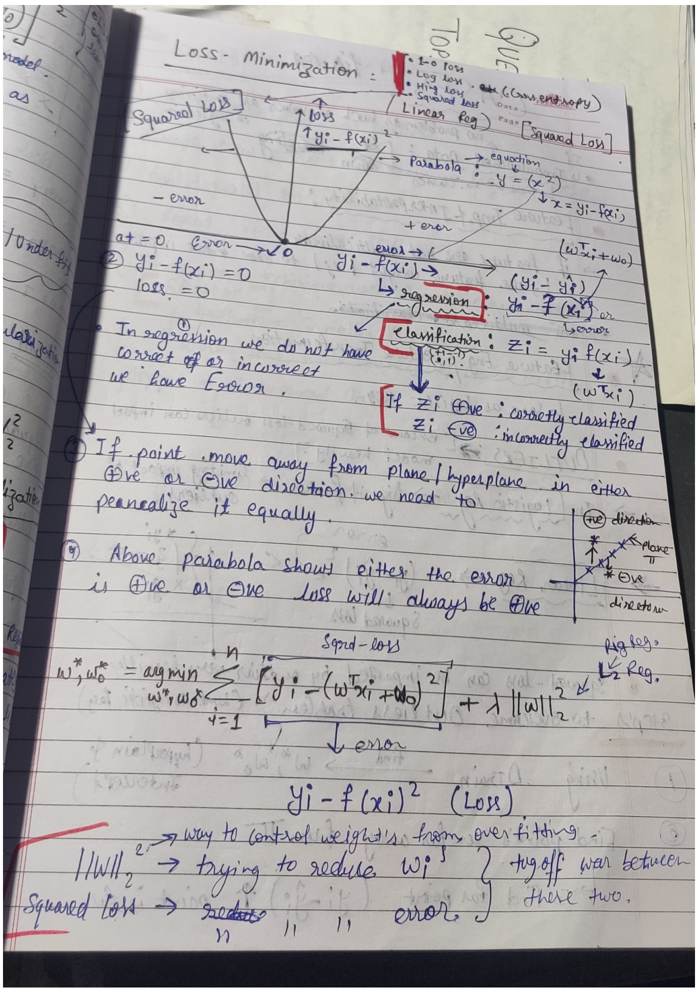
`At origin Error = 0 i.e.`

$$y_i - f(x_i) = 0$$

`now error in +ve direction or error in -ve direction  from hyperplane will be penalized equally`

`NOTE`

`In Regression we do not have Correct or Incorrect classified points we have only Error to consider`



Points (LinearRegression)

`Decision Surface:`

- Linear / HyperPlane

`Assumptions:`

- Data is Linearly/Almost Linearly Septrable

`Feature Importance & Model Interpretability:`

- if Features are not collinear or Multi-collinear, than only we can use absolute value of Weights i.e. |W| otherwise use techniques like:
  - Forward feature Selection
  - VIF and other....

`If Data is Imbalanced:`

- No logic of Imbalance data target is continious
  - Linear Regression solves Regression problem means it works with continious values not classes

`If Outliers in Data:`

- Linear Regression is impacted by outliers (10-9=1, 100-99=1,1000-999=1), as Squared loss is impacted by outliers

- Logistic Regression was less impacted by outliers `because of Sigmoid function` as it squaces values between range of (0and1)

  - Sigmoid can deals fine with Outlies, but not completly
  - Sigmoid can limit impact of outlier to some degree

`For Outliers in Data:`

1. Using all Training dat find best (W,W_0)
2. Find all points which are very far away from the Hyperplane
   - It is very easy to find, for points if (y - y_hat) is a big number,than point is very far away from plane
3. Remove points which are outliers i.e. very far away from the plane
4. Create new dataset i.e. dataset which dont have outliers
5. Fit Liner Regression model on new Training data which dont have outliers
6. We can repeat step:1 to 5, if ther exis any outliers

`NOTE: This repetative technique in Statistics is called Random Sampling Concences:(RANSAC)`

- `Technique like RANSAC works for most of the models in ML`

`If Missing values in Data:`

- Mean Imputation
- Median Imputation
- Mode Imputation
- kNN...etc

`If Lo-latency is required dont forget to use L1-Regulizer(i.e. increasing hyperparameter lambda), it generates sparcity which kicks unwanted features to 0`

Understanding Linear Regression

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [2]:
```
boston = load_boston()
```

```
print(boston.data.shape,'\n')
print(boston.feature_names,'\n')
print(boston.target)
```

```
(506, 13)

['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']

[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_b
oston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

In [3]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually t
he target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Colli
nearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth
International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Ka
ufmann.
```

In [4]:
```
bos = pd.DataFrame(boston.data)
bos['price'] = boston.target

bos.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

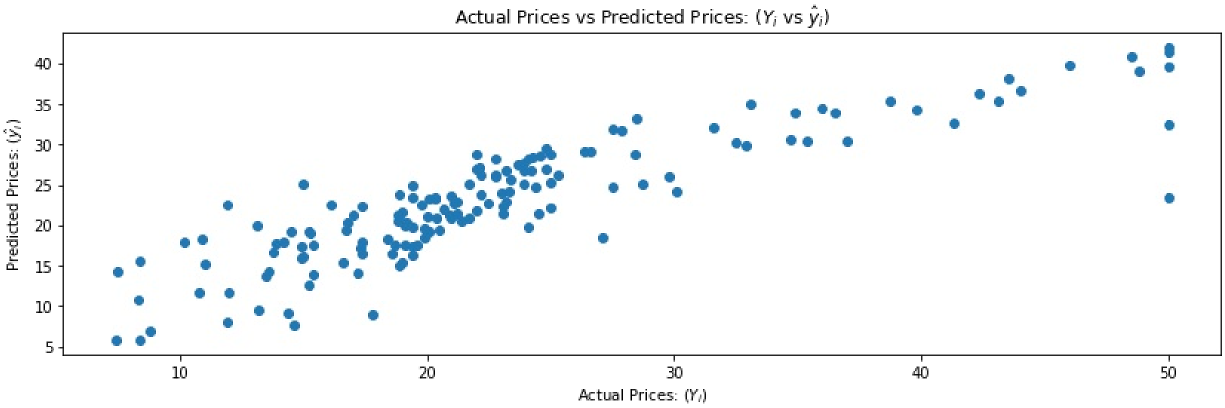
In [5]:
```
X = bos.drop('price',axis=1)
y = bos['price']
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=108)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

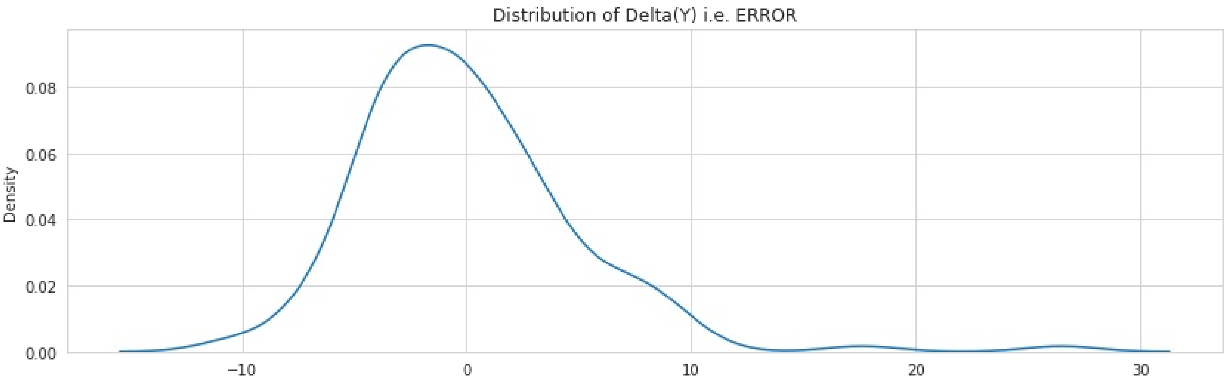
Out[6]:
```
((354, 13), (152, 13), (354,), (152,))
```

In [7]:
```
lr = LinearRegression()
lr.fit(X_train,y_train)

y_pred = lr.predict(X_test)

#finding Errors
delta_y = y_test - y_pred
```

In [8]:
```
plt.figure(figsize=(14,4))
plt.scatter(y_test,y_pred)
plt.xlabel("Actual Prices: ($Y_i$)")
plt.ylabel("Predicted Prices: ($\hat{y}_i$)")
plt.title("Actual Prices vs Predicted Prices: ($Y_i$ vs $\hat{y}_i$)")
plt.show()
```

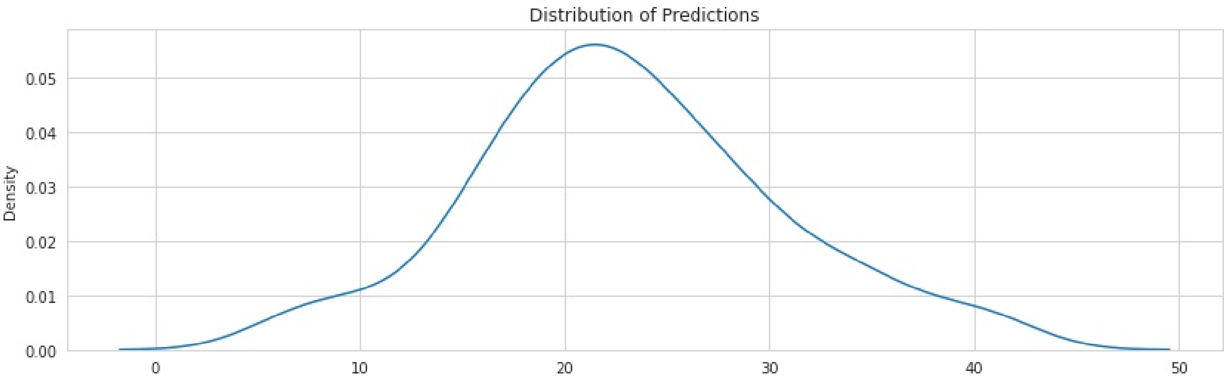


In [9]:
```
plt.figure(figsize=(14,4))
sns.set_style('whitegrid')
sns.kdeplot(np.array(delta_y),bw_adjust=0.9)
plt.title("Distribution of Delta(Y) i.e. ERROR")
plt.show()
```



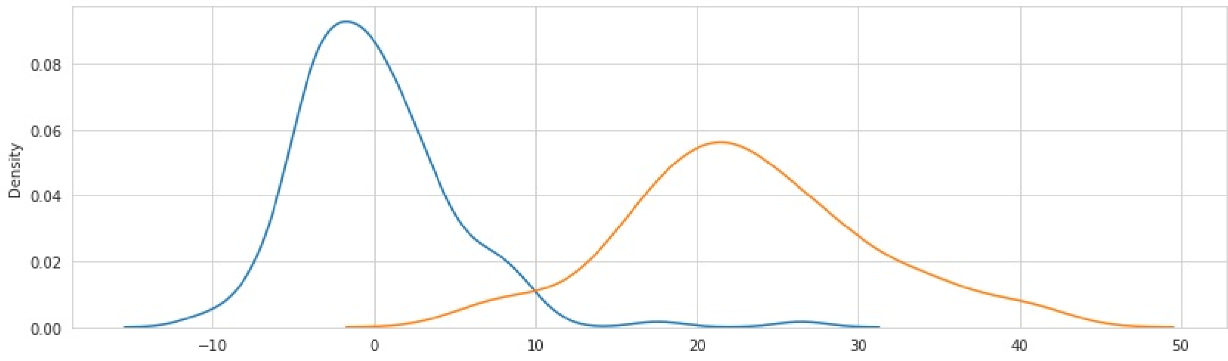
- Errors are almost 0-centric, i.e. having mean of 0
- Most of the error are in negative side

In [10]:
```
plt.figure(figsize=(14,4))
sns.kdeplot(np.array(y_pred),bw_adjust=0.9)
plt.title("Distribution of Predictions")
plt.show()
```



- Mean is of almost 22

In [11]:
```
plt.figure(figsize=(14,4))

sns.kdeplot(np.array(delta_y),bw_adjust=0.9)
sns.kdeplot(np.array(y_pred),bw_adjust=0.9)
plt.show()
```



- Errors are lower than predicted values, this solution can be a descent solution so far
- To improve this solution we have reduce region between (-10,0)
  - Idea distribution of error must be very thin with 0-mean
- More Feature engineering/transformation can be used

Follow Mukesh Manral, Offering free aggregated Data Science content for Working and Aspiring data professionals

Newsletter Click Me

YT (AWS,Sagemaker)Click Me