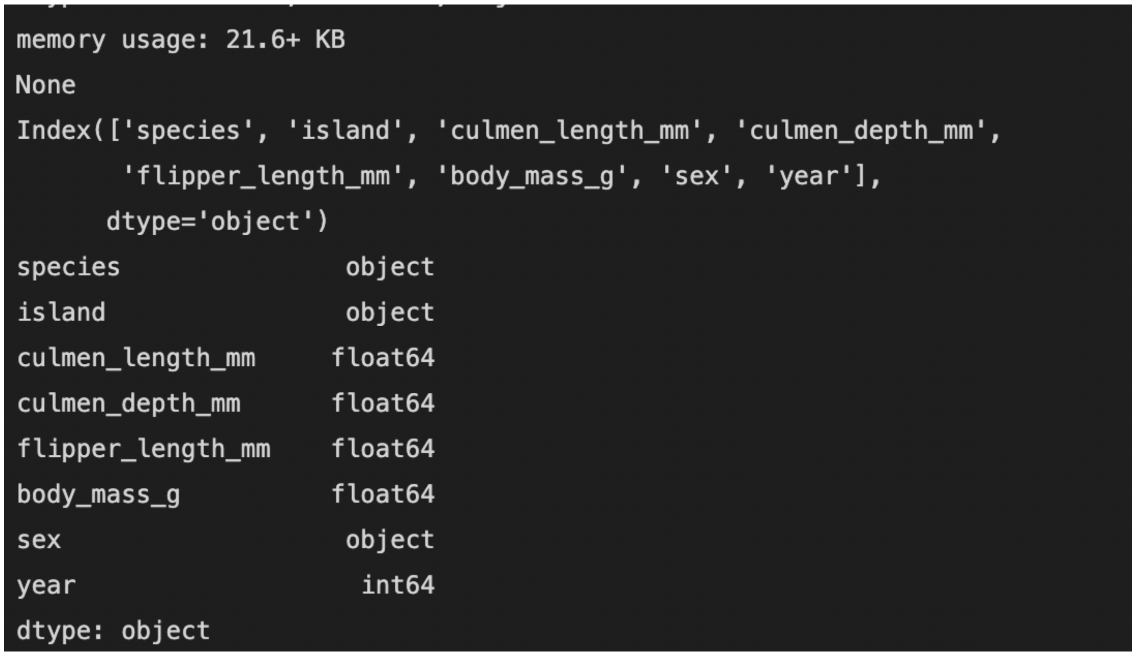
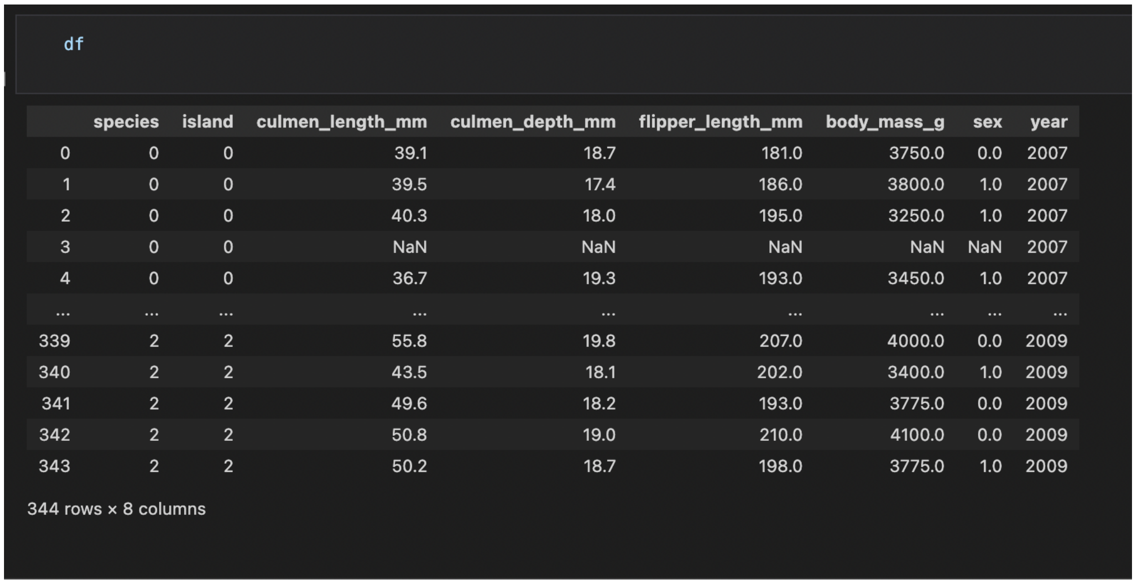
I. LOGISTIC REGRESSION:

The dataset we have used is ‘penguins.csv’. This dataset contains 344 entries about penguins. The dataset has the following features:



```
memory usage: 21.6+ KB
None
Index(['species', 'island', 'culmen_length_mm', 'culmen_depth_mm',
       'flipper_length_mm', 'body_mass_g', 'sex', 'year'],
      dtype='object')
species               object
island                object
culmen_length_mm     float64
culmen_depth_mm      float64
flipper_length_mm    float64
body_mass_g          float64
sex                   object
year                   int64
dtype: object
```

Statistics:



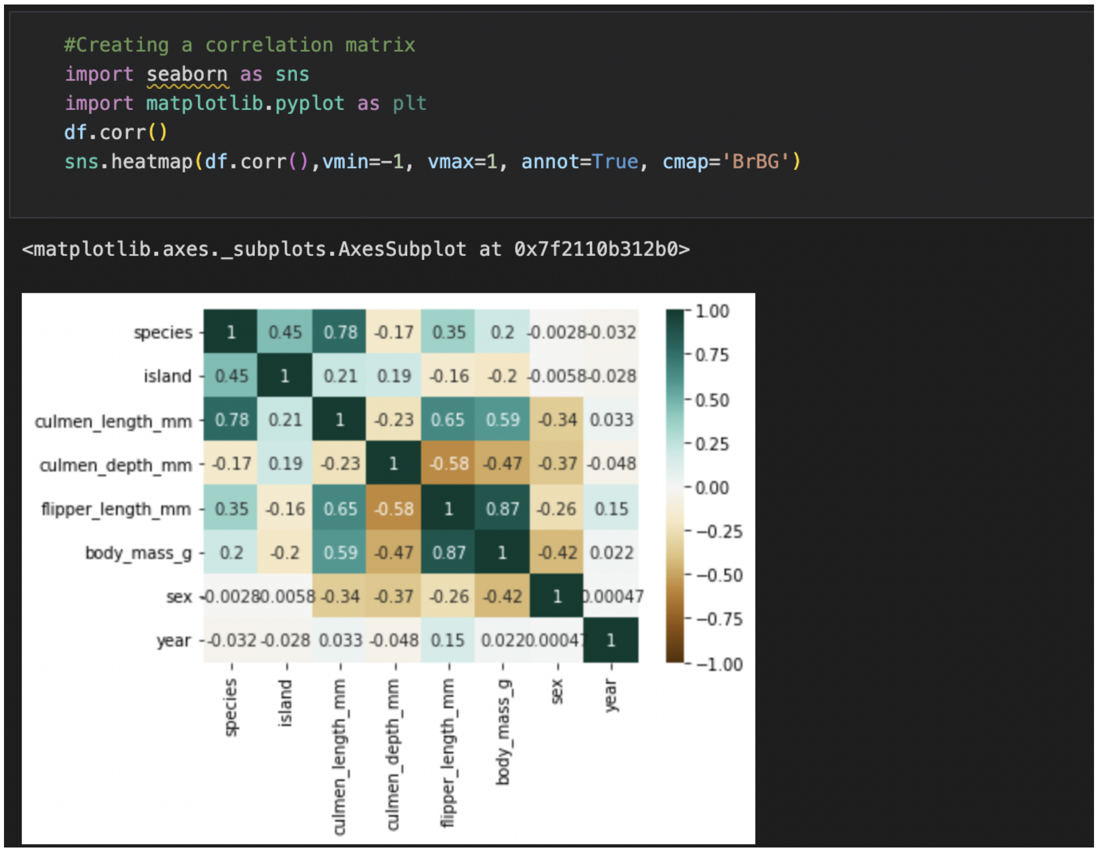
```
df
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	0	0	39.1	18.7	181.0	3750.0	0.0	2007
1	0	0	39.5	17.4	186.0	3800.0	1.0	2007
2	0	0	40.3	18.0	195.0	3250.0	1.0	2007
3	0	0	NaN	NaN	NaN	NaN	NaN	2007
4	0	0	36.7	19.3	193.0	3450.0	1.0	2007
...	...	...	...	...	...	...	...	...
339	2	2	55.8	19.8	207.0	4000.0	0.0	2009
340	2	2	43.5	18.1	202.0	3400.0	1.0	2009
341	2	2	49.6	18.2	193.0	3775.0	0.0	2009
342	2	2	50.8	19.0	210.0	4100.0	0.0	2009
343	2	2	50.2	18.7	198.0	3775.0	1.0	2009

344 rows × 8 columns

1)

2) Correlations between the features is shown in the below figure:



```
#Creating a correlation matrix
import seaborn as sns
import matplotlib.pyplot as plt
df.corr()
sns.heatmap(df.corr(),vmin=-1, vmax=1, annot=True, cmap='BrBG')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2110b312b0>
```

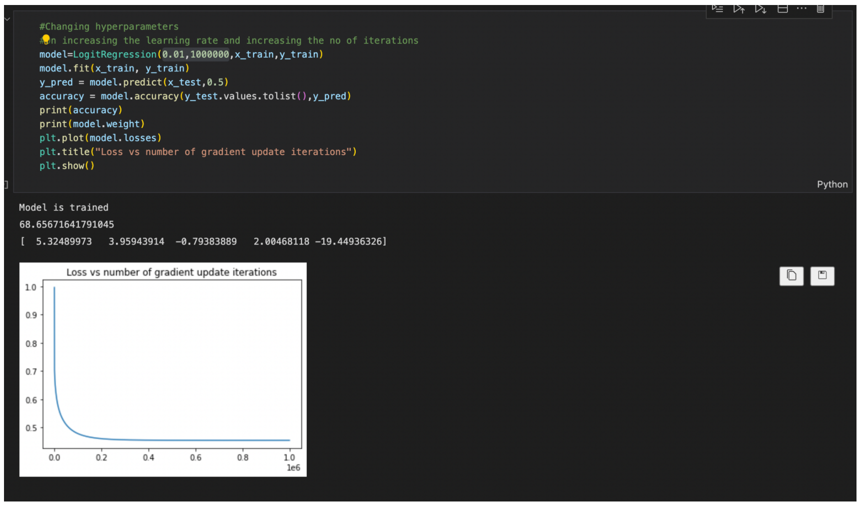
species 1 0.45 0.78 -0.17 0.35 0.2 -0.0028 -0.032
island 0.45 1 0.21 0.19 -0.16 -0.2 -0.0058 -0.028
culmen_length_mm 0.78 0.21 1 -0.23 0.65 0.59 -0.34 0.033
culmen_depth_mm -0.17 0.19 -0.23 1 -0.58 -0.47 -0.37 -0.048
flipper_length_mm 0.35 -0.16 0.65 -0.58 1 0.87 -0.26 0.15
body_mass_g 0.2 -0.2 0.59 -0.47 0.87 1 -0.42 0.022
sex -0.0028 -0.0058 -0.34 -0.37 -0.26 -0.42 1 0.00047
year -0.032 -0.028 0.033 -0.048 0.15 0.022 0.00047 1
species island culmen_length_mm culmen_depth_mm flipper_length_mm body_mass_g sex year
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00

1) Provide your best accuracy - 68.6567 is the best accuracy we could get.

The hyperparameter values are - learning rate: 0.01

Number of iterations - 1000000

The model weights are - [5.32489973 3.95943914 -0.79383889 2.00468118 -19.44936326]

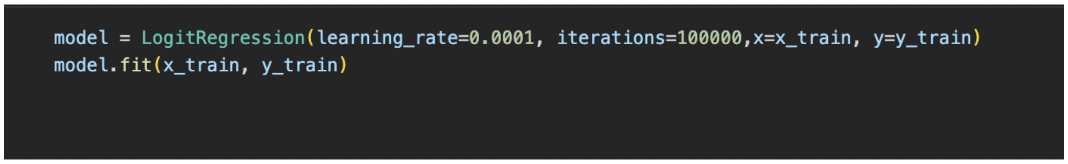


```
#Changing hyperparameters
#On increasing the learning rate and increasing the no of iterations
model=LogitRegression(0.01,1000000,x_train,y_train)
model.fit(x_train, y_train)
y_pred = model.predict(x_test,0.5)
accuracy = model.accuracy(y_test.values.tolist(),y_pred)
print(accuracy)
print(model.weight)
plt.plot(model.losses)
plt.title("Loss vs number of gradient update iterations")
plt.show()
```

```
Model is trained
68.65671641791045
[  5.32489973   3.95943914  -0.79383889   2.00468118 -19.44936326]
```

2) Include loss graph and provide a short analysis of the results.

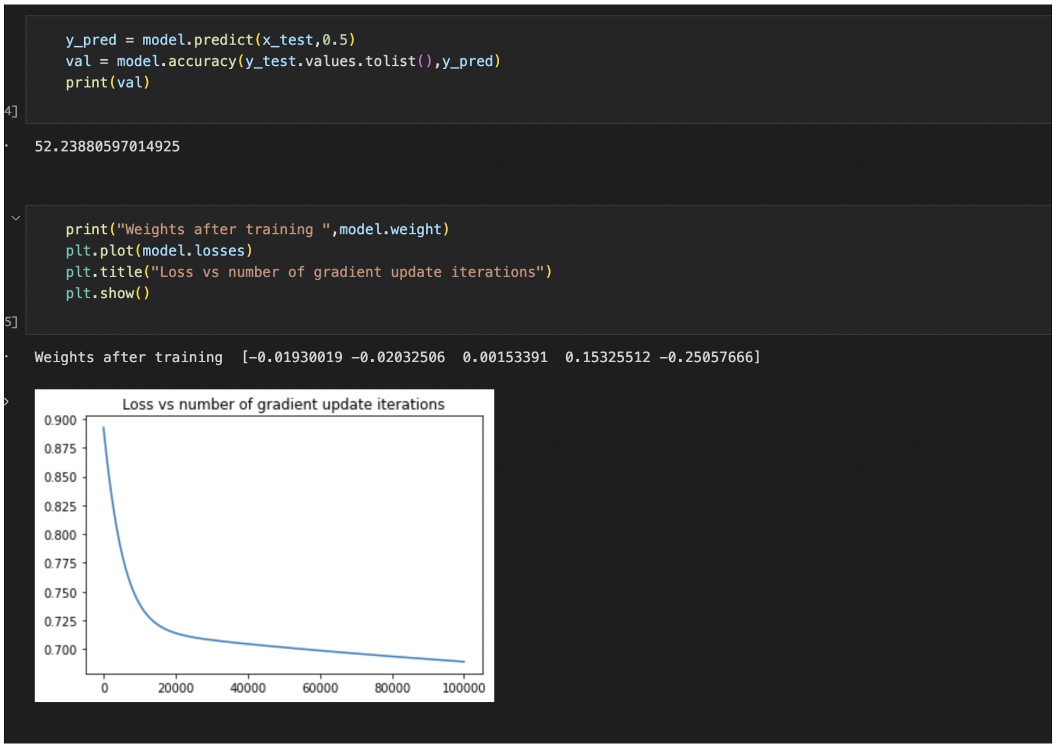
We used the following the hyperparameters for training



```
model = LogitRegression(learning_rate=0.0001, iterations=100000,x=x_train, y=y_train)
model.fit(x_train, y_train)
```

On plotting the Loss vs Gradient Descent update graph:

We can see that for the first 20000 iterations, the loss kept decreasing in huge numbers. From the 40000 iteration, the loss has more or less remained the same which is close to 0.700. The loss graph is a logarithmic function graph. From the below, you can deduce that with increasing iterations, the loss reduces.



```
y_pred = model.predict(x_test,0.5)
val = model.accuracy(y_test.values.tolist(),y_pred)
print(val)
```

```
52.23880597014925
```

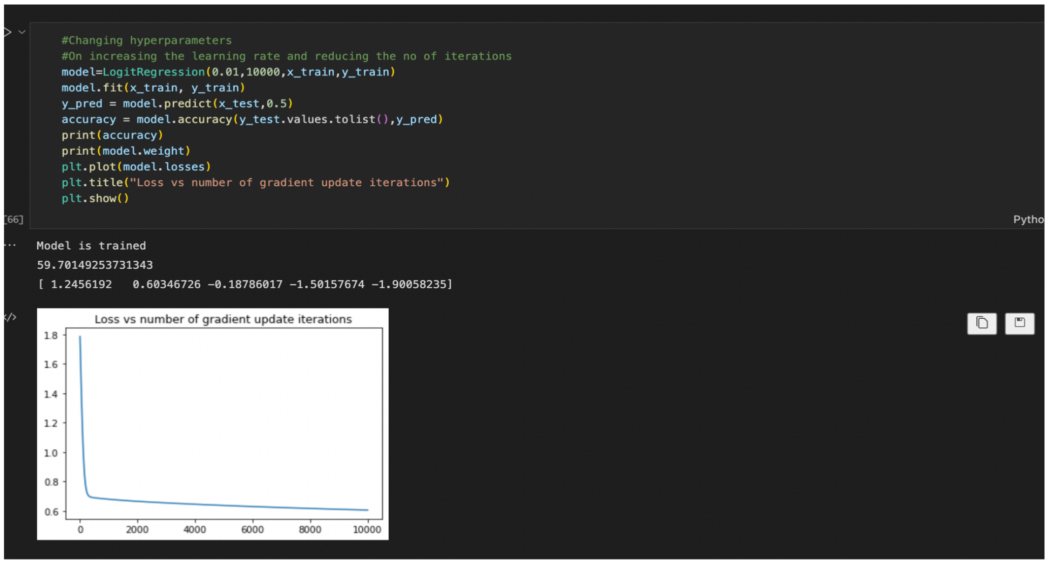
```
print("Weights after training ",model.weight)
plt.plot(model.losses)
plt.title("Loss vs number of gradient update iterations")
plt.show()
```

```
Weights after training  [-0.01930019 -0.02032506  0.00153391  0.15325512 -0.25057666]
```

3. Explain how hyperparameters influence the accuracy of the model. Provide at least 3 different setups with learning rate and #iterations and discuss the results along with plotting of graphs.

The below are the screenshots of Loss v/s Gradient Descent Iterations graph with various combinations of the hyperparameters i.e., learning rate and the no of iterations

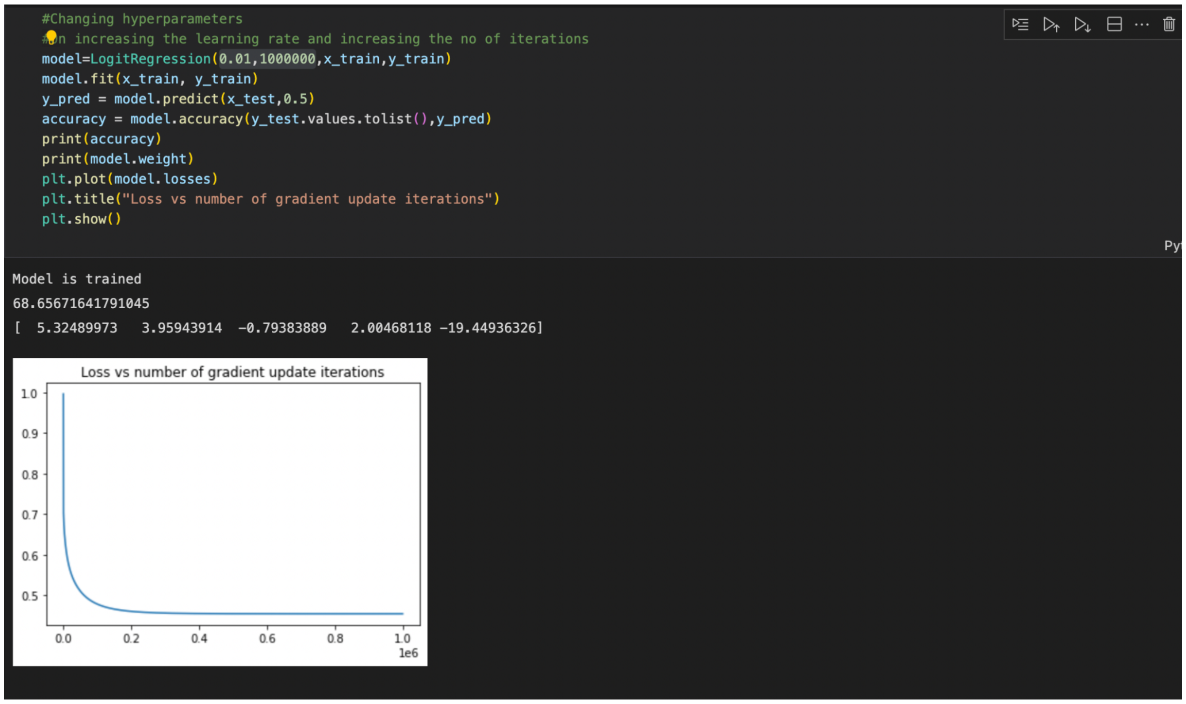
1) As you can see below with the increasing gradient descent updates, the loss has significantly reduced. The minimum we could achieve for 10000 iterations is approximately 0.8



```
#Changing hyperparameters
#On increasing the learning rate and reducing the no of iterations
model=LogitRegression(0.01,10000,x_train,y_train)
model.fit(x_train, y_train)
y_pred = model.predict(x_test,0.5)
accuracy = model.accuracy(y_test.values.tolist(),y_pred)
print(accuracy)
print(model.weight)
plt.plot(model.losses)
plt.title("Loss vs number of gradient update iterations")
plt.show()
```

```
Model is trained
59.70149253731343
[ 1.2456192   0.60346726 -0.18786017 -1.50157674 -1.90058235]
```

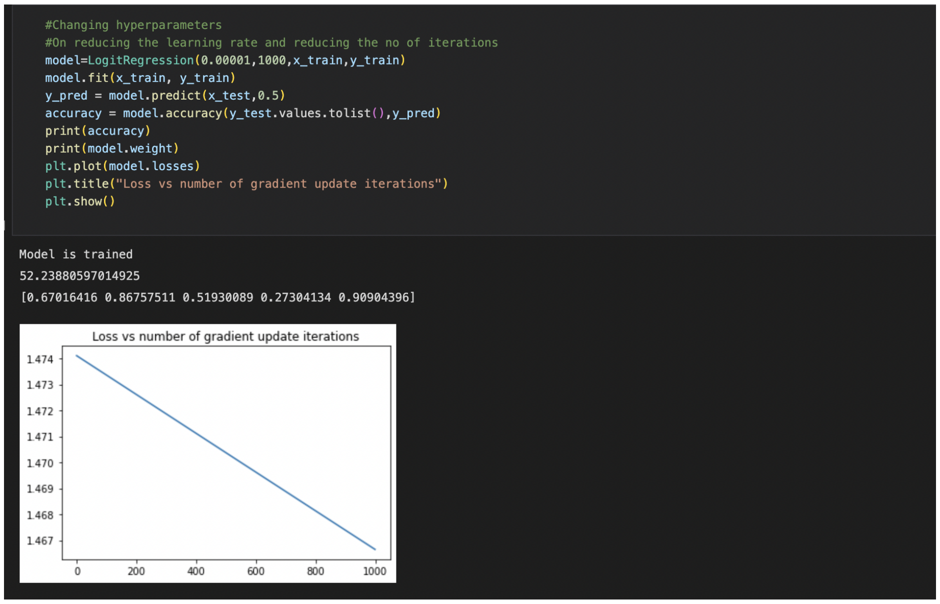
3) On increasing the learning rate and the no of iterations, the loss graph has become smoother and the minimum loss now has become nearly 0



```
#Changing hyperparameters
#On increasing the learning rate and increasing the no of iterations
model=LogitRegression(0.01,1000000,x_train,y_train)
model.fit(x_train, y_train)
y_pred = model.predict(x_test,0.5)
accuracy = model.accuracy(y_test.values.tolist(),y_pred)
print(accuracy)
print(model.weight)
plt.plot(model.losses)
plt.title("Loss vs number of gradient update iterations")
plt.show()
```

```
Model is trained
68.65671641791045
[  5.32489973   3.95943914  -0.79383889   2.00468118 -19.44936326]
```

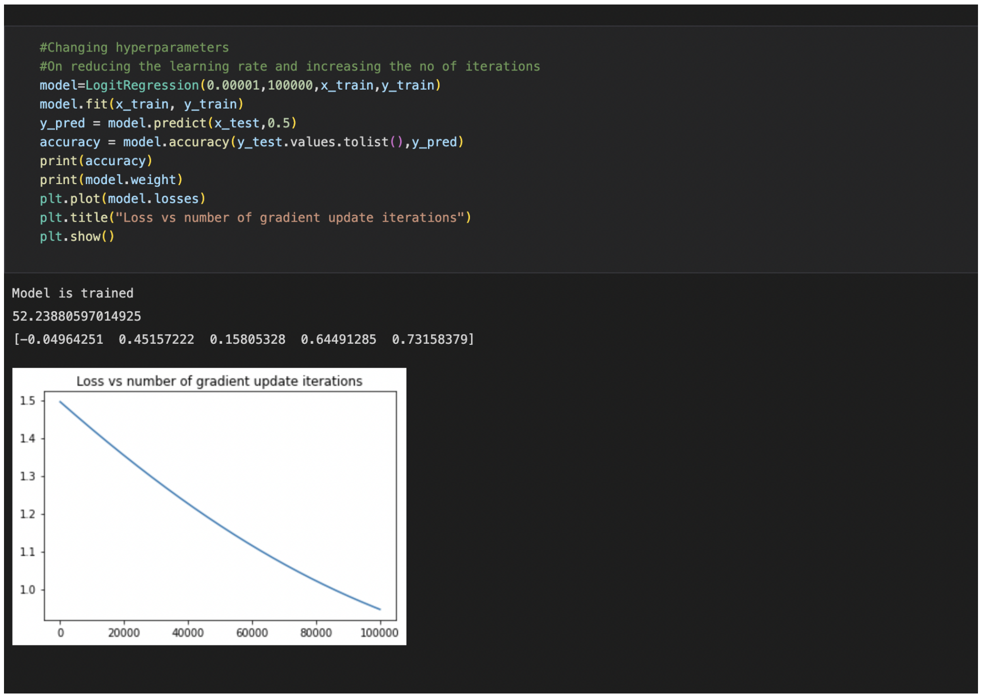
4) On reducing the learning rate and no of iterations, you can see that the minimum loss is large. You can see that there is a continuous reduction with the increasing gradient descent updates. From the above graphs, you learn that the loss becomes nearly constant after reaching a threshold.



```
#Changing hyperparameters
#On reducing the learning rate and reducing the no of iterations
model=LogitRegression(0.00001,1000,x_train,y_train)
model.fit(x_train, y_train)
y_pred = model.predict(x_test,0.5)
accuracy = model.accuracy(y_test.values.tolist(),y_pred)
print(accuracy)
print(model.weight)
plt.plot(model.losses)
plt.title("Loss vs number of gradient update iterations")
plt.show()
```

```
Model is trained
52.23880597014925
[0.67016416 0.86757511 0.51930089 0.27304134 0.90904396]
```

5) Accuracy has reduced when we reduced the learning rate while still having the number of iterations as 10000. This shows us that we need to have a bigger learning rate i.e., >=0.01



```
#Changing hyperparameters
#On reducing the learning rate and increasing the no of iterations
model=LogitRegression(0.00001,100000,x_train,y_train)
model.fit(x_train, y_train)
y_pred = model.predict(x_test,0.5)
accuracy = model.accuracy(y_test.values.tolist(),y_pred)
print(accuracy)
print(model.weight)
plt.plot(model.losses)
plt.title("Loss vs number of gradient update iterations")
plt.show()
```

```
Model is trained
52.23880597014925
[-0.04964251  0.45157222  0.15805328  0.64491285  0.73158379]
```

4) Discuss the benefits/drawbacks of using a Logistic Regression model.

Benefits:

1) Logistic Regression is easier to implement
2) Logistic Regression can be extended to multinomial regression conveniently.
3) It performs well when the data is linearly separable
4) Logistic Regression doesn’t easily overfit.

Drawbacks:

1) Non linear problems cannot be solved by Logistic Regression since it works on linear surfaces.
2) Logistic Regression requires that there is not much similarity/collinearity between the variables. It requires that if two variables are highly collinear, then we have to use either of them to reduce the loss incurred.

II. LINEAR REGRESSION:

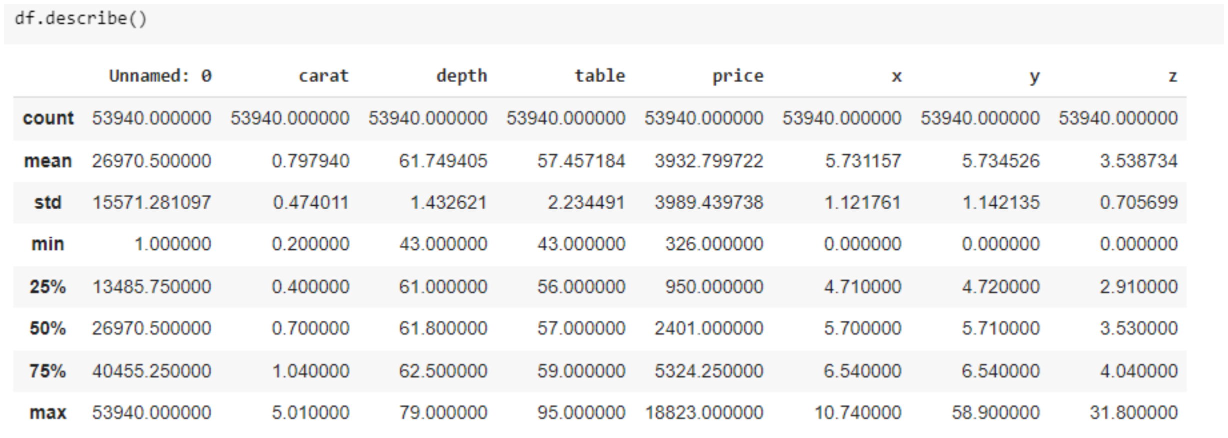
DATASET:

The dataset we have used is ‘diamond.csv’. This dataset contains approximately 54,000 datapoints about diamonds.

The dataset has 10 features, which are:

Feature name	**Explanation**	**Datatype**
carat	weight of the diamond	float
cut	quality of the cut	object
color	diamond color	object
clarity	how clear the diamond	object
depth	total depth percentage	float
table	width of top of diamond relative to widest point	float
price	in US dollars	float
x	length in mm	int
y	width in mm	float
z	depth in mm	float

STATISTICS:

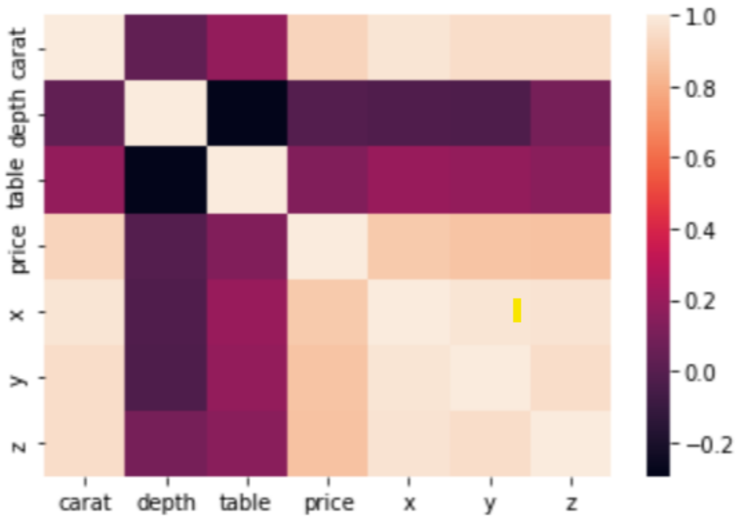


```
df.describe()
```

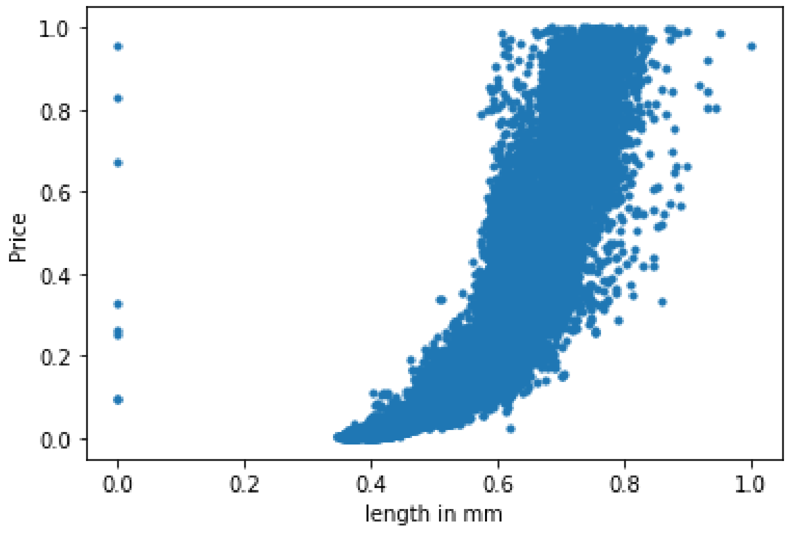
	Unnamed: 0	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	26970.500000	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	15571.281097	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	13485.750000	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	26970.500000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	40455.250000	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	53940.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

- The feature values of this dataset don’t not have any missing values.

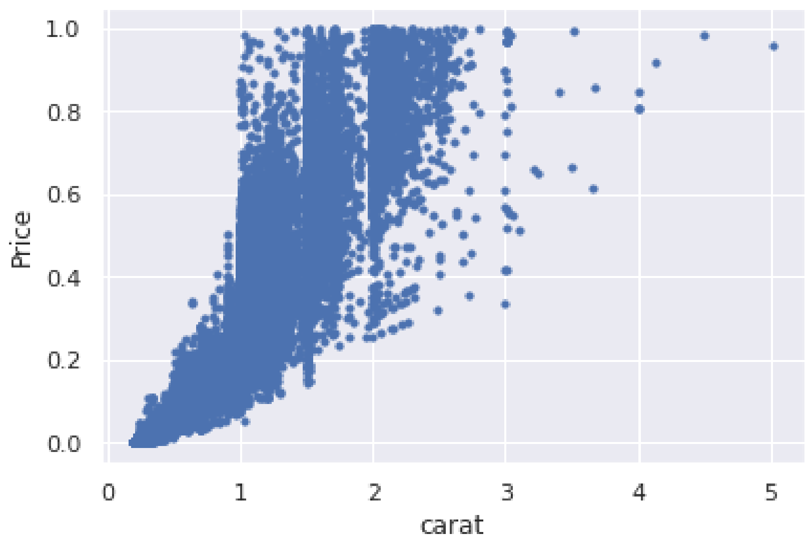
VISUALIZATION:



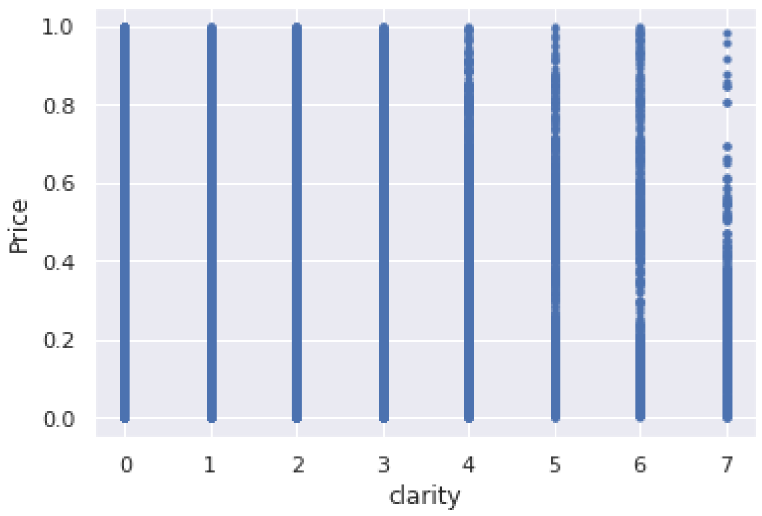
- This heatmap depicts the correlation between all the feature values in the dataset.



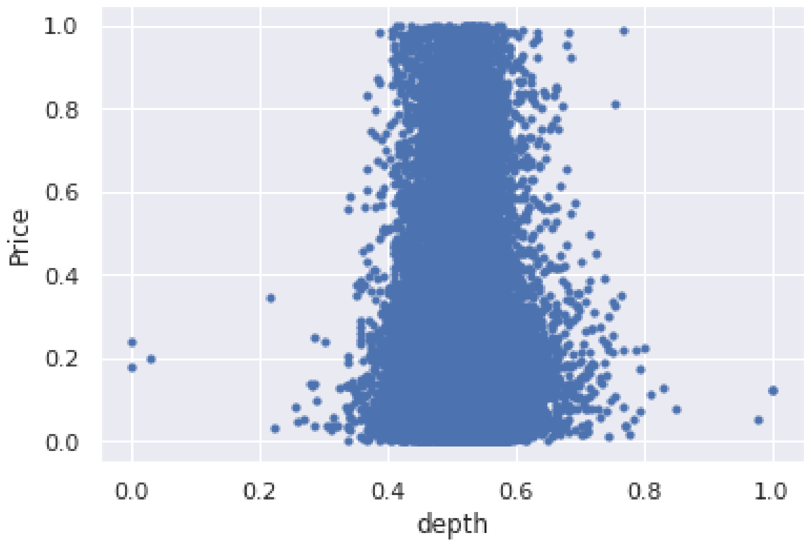
- This scatter plot depicts the relationship between the price of the diamond (Y axis) and the length of the diamond in mm (X axis).
- From this graph we can interpret that the relationship is linear and can be used as one of the features to train our model.



- This scatter plot depicts the relationship between the price of the diamond (Y axis) and the weight of the diamond – carat (X axis).
- From this graph we can interpret that the relationship is linear and can be used as one of the features to train our model.



- This scatter plot depicts the relationship between the price of the diamond (Y axis) and the clarity of the diamond (X axis).
- From this graph we can interpret that the relationship is not linear and is not a very feature to train our model.



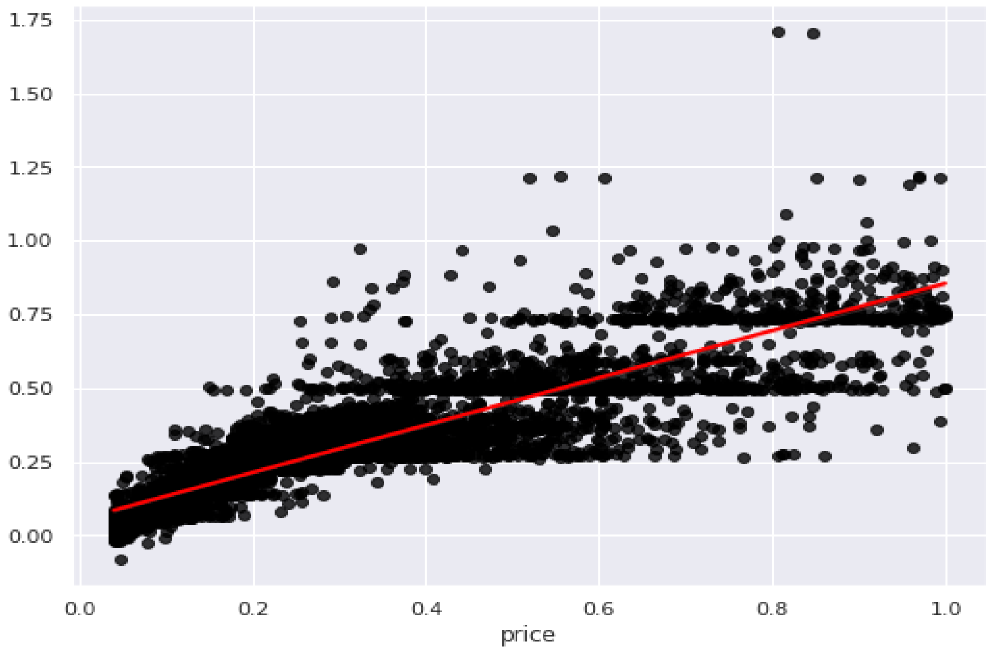
- This scatter plot depicts the relationship between the price of the diamond (Y axis) and the depth percentage of the diamond (X axis).
- From this graph we can interpret that the relationship is not exactly linear and using this to train our model might not give the most optimum loss value.

LOSS VALUE:

Weights = **[0.52641282, -0.61668997, 1.84774709, -0.74353044]**

After calculating the weights using Ordinary least square method, the Mean squared error: **0.009239127204716742**

REGRESSION PLOT:



In the above graph, the X axis denotes the actual price and Y axis denotes the predicted price of the diamonds in the dataset.

BENEFITS OF USING OLS:

- OLS is used widely in numerical data analysis to interpret data statistics. It is easy to interpret and implement.
- OLS is used for interpreting complex relationships between variables as it can easily work with multiple predictor variables.

DRAWBACKS OF USING OLS:

- OLS is often Prone to overfitting and underfitting.
- OLS assumes a linear relationship between the target values and test values. If the relationship is non-linear, OLS may not be a good fit.

BENEFITS OF USING A LINEAR REGRESSION:

- Linear regression provides a straightforward way to interpret the relationship between the predicted values and the actual values.
- Linear regression can be run on a low resource system to get real time results on a stream of data.

DRAWBACKS OF USING A LINEAR REGRESSION MODEL:

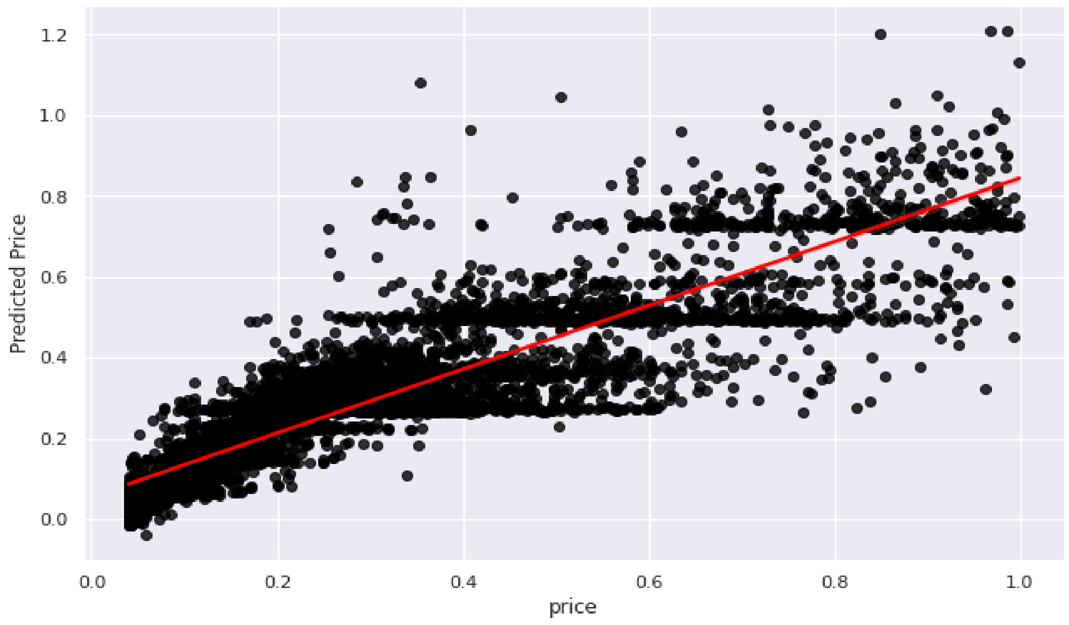
- Linear regression can be sensitive to outliers.
- When the predicted values are highly correlated with each, linear regression may not be a good fit as it can be susceptible to multicollinearity.

II. RIDGE REGRESSION:

LOSS VALUE:

Weights = **[0.5210347, -0.38896239, -0.0265713, -0.14960622]**
After calculating the weights using Ordinary least square method, the Mean squared error: **36.535328244081214**

REGRESSION PLOT:



In the above graph, the X axis denotes the actual price and Y axis denotes the predicted price of the diamonds in the dataset.

- Linear regression/ l1 regularization is the sum of absolute value of weights, whereas Ridge regression/ l2 regularization is the sum of square of weights.
- Linear regression/ l1 regularization is robust to outliers, while Ridge regression/ l2 regularization is not.
- The main motivation for using L2 regularization is to prevent overfitting.
- It improves the performance of the model, in situations where the data is high-dimensional or contains multicollinearity.

ADVANTAGES OF RIDGE REGRESSION:

- The Ridge Regression model does not require unbiased estimators.
- When there is multicollinearity, the ridge estimator is good at improving least-squares estimate.

DISADVANTAGES OF RIDGE REGRESSION:

- Ridge Regression models are unable to perform feature selection.
- In most cases Ridge Regression model shrinks the coefficients towards zero and they trade the variance for bias.

References used:

1) https://towardsdatascience.com/https-medium-com-chayankathuria-optimization-ordinary-least-squares-gradient-descent-from-scratch-8b48151ba756
2) https://datascience.stackexchange.com/questions/110010/difference-between-ols-and-gradient-descent-in-linear-regression
3) https://medium.com/@vijay.swamy1/lasso-versus-ridge-versus-elastic-net-1d57cfc64b58

Team Member	Assignment Part	Contribution (%)
Neeraja Sanjay	Part i Part ii Part iii and bonus	50%
Anurima Vaishnavi Kumar	Part i Part ii Part iii and bonus	50%