Kartikeya Agarwal
2019UCO1692

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

Machine Learning
(COCSC17)

Name: Kartikeya Agarwal
Roll Number: 2019UCO1692
Branch: COE
Section: 3
Semester: 3

Linear Regression

Importing Important Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

Reading Data Files

In [ ]:
```
df = pd.read_csv('imports-85.data')
df.columns = ['symboling','normalized-losses','make','fuel-type','aspiration','num-o
```

In [ ]:
```
df.head()
```

Out[ ]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8

5 rows × 26 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          204 non-null    int64
 1   normalized-losses  204 non-null    object
 2   make               204 non-null    object
 3   fuel-type          204 non-null    object
 4   aspiration         204 non-null    object
 5   num-of-doors       204 non-null    object
 6   body-style         204 non-null    object
 7   drive-wheels       204 non-null    object
 8   engine-location    204 non-null    object
 9   wheel-base         204 non-null    float64
 10  length             204 non-null    float64
 11  width              204 non-null    float64
 12  height             204 non-null    float64
 13  curb-weight        204 non-null    int64
 14  engine-type        204 non-null    object
```

```
 15  num-of-cylinders   204 non-null    object
 16  engine-size        204 non-null    int64
 17  fuel-system        204 non-null    object
 18  bore               204 non-null    object
 19  stroke             204 non-null    object
 20  compression-ratio  204 non-null    float64
 21  horsepower         204 non-null    object
 22  peak-rpm           204 non-null    object
 23  city-mpg           204 non-null    int64
 24  highway-mpg        204 non-null    int64
 25  price              204 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.6+ KB
```

In [ ]:
```
columns_to_drop = ['bore','compression-ratio','peak-rpm','wheel-base','curb-weight',
df = df.drop(columns_to_drop,axis=1)
```

In [ ]:
```
df.head()
```

Out[ ]:

	symboling	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	length	width	height
0	3	alfa-romero	gas	std	two	convertible	rwd	front	168.8	64.1	48.8
1	1	alfa-romero	gas	std	two	hatchback	rwd	front	171.2	65.5	52.4
2	2	audi	gas	std	four	sedan	fwd	front	176.6	66.2	54.3
3	2	audi	gas	std	four	sedan	4wd	front	176.6	66.4	54.3
4	2	audi	gas	std	two	sedan	fwd	front	177.3	66.3	53.1

Cleaning the DataFrame

In [ ]:
```
df_clean = df.replace(to_replace='?',value = np.nan)
```

In [ ]:
```
df_clean = df_clean.fillna(value = df_clean.median())
```

In [ ]:
```
df_clean = df_clean.dropna()
```

In [ ]:
```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202 entries, 0 to 203
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   symboling         202 non-null    int64
 1   make              202 non-null    object
 2   fuel-type         202 non-null    object
 3   aspiration        202 non-null    object
 4   num-of-doors      202 non-null    object
```

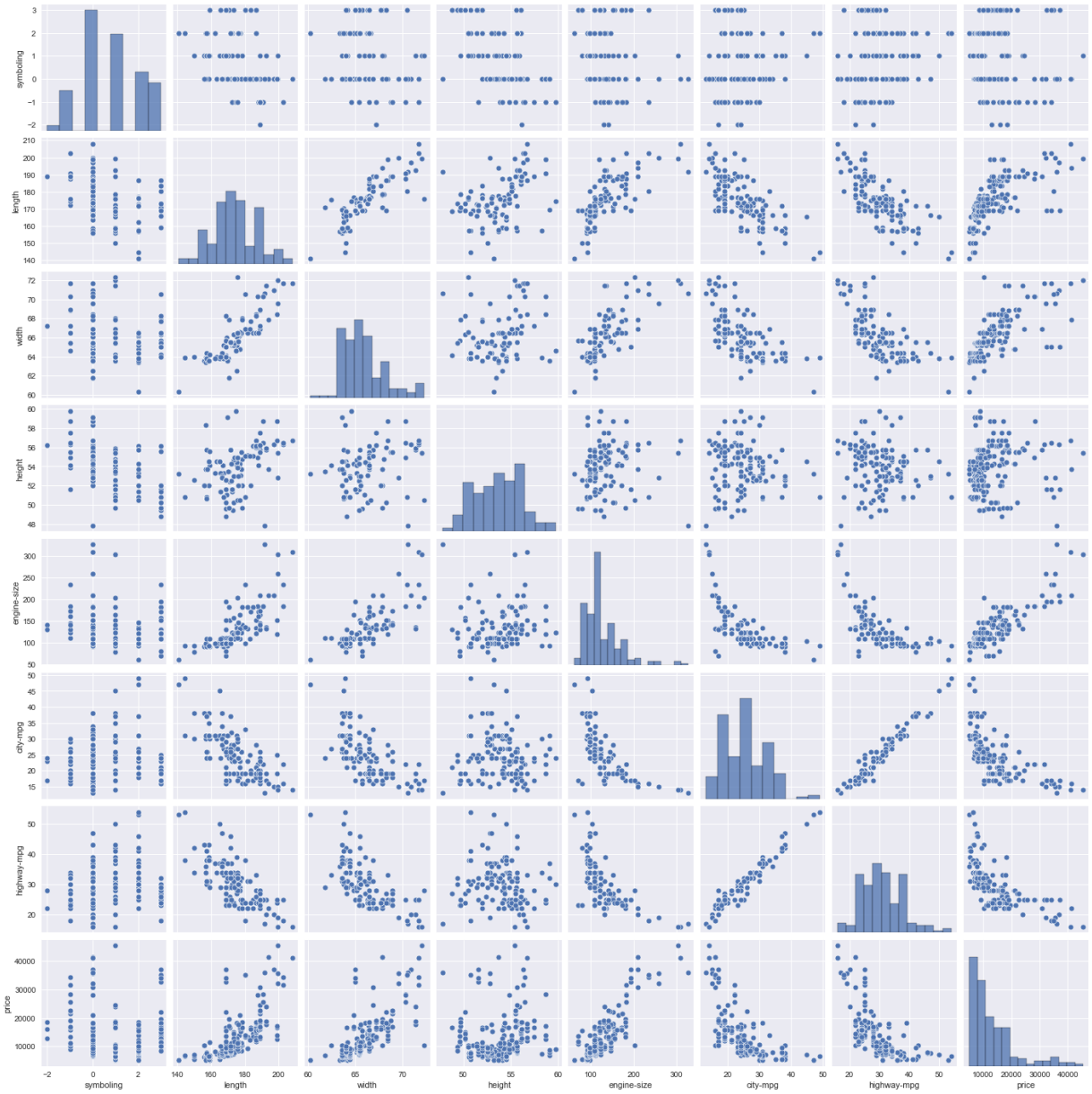
```
 5   body-style         202 non-null    object
 6   drive-wheels       202 non-null    object
 7   engine-location    202 non-null    object
 8   length             202 non-null    float64
 9   width              202 non-null    float64
 10  height             202 non-null    float64
 11  engine-type        202 non-null    object
 12  num-of-cylinders   202 non-null    object
 13  engine-size        202 non-null    int64
 14  fuel-system        202 non-null    object
 15  stroke             202 non-null    object
 16  horsepower         202 non-null    object
 17  city-mpg           202 non-null    int64
 18  highway-mpg        202 non-null    int64
 19  price              202 non-null    object
dtypes: float64(3), int64(4), object(13)
memory usage: 33.1+ KB
```

In [ ]:
```
df_clean['price'] = df_clean['price'].astype('float64')
```

Visualizing Data to make a Co-relation Matrix

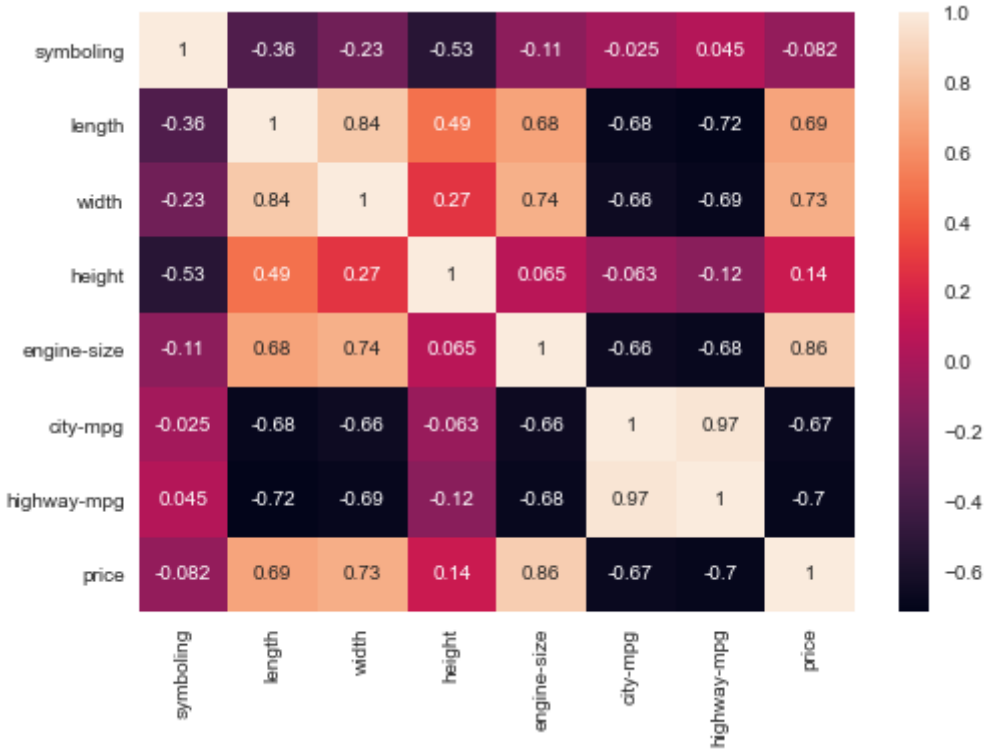
In [ ]:
```
import seaborn as sns
```

In [ ]:
```
sns.pairplot(df_clean)
plt.show()
```



In [ ]:

```
sns.heatmap(df_clean.corr(),annot=True)
plt.show()
```



Encoding Class Data to create Continuous Regression Data

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
columns_to_encode = ['make','fuel-type','aspiration','num-of-doors','body-style','dr
for i in columns_to_encode:
    df_clean[i] = le.fit_transform(df_clean[i].astype(str))
```

In [ ]:

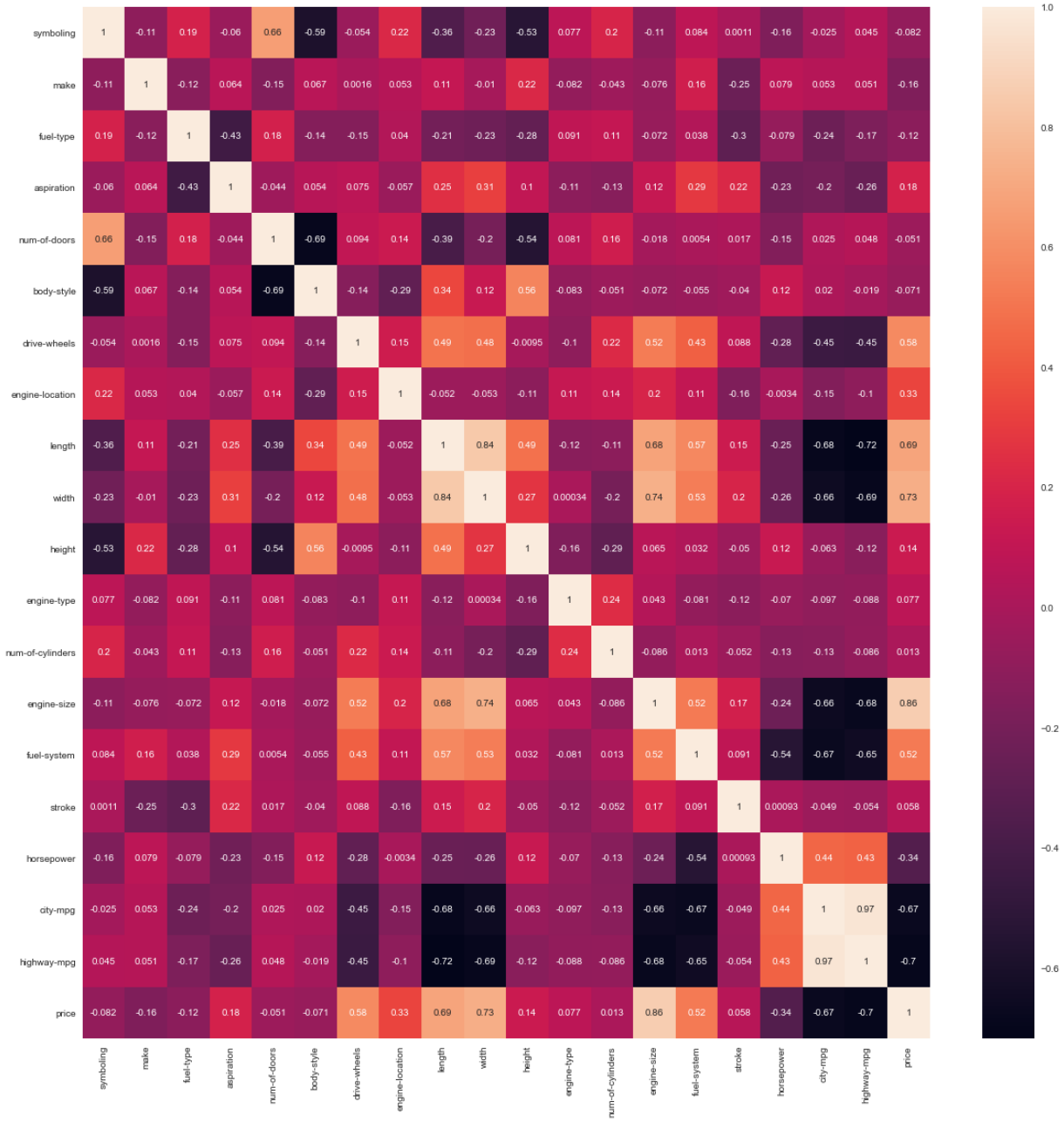
```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202 entries, 0 to 203
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   symboling         202 non-null    int64  
 1   make              202 non-null    int32  
 2   fuel-type         202 non-null    int32  
 3   aspiration        202 non-null    int32  
 4   num-of-doors      202 non-null    int32  
 5   body-style        202 non-null    int32  
 6   drive-wheels      202 non-null    int32  
 7   engine-location   202 non-null    int32  
 8   length            202 non-null    float64
 9   width             202 non-null    float64
 10  height            202 non-null    float64
 11  engine-type       202 non-null    int32  
 12  num-of-cylinders  202 non-null    int32  
 13  engine-size       202 non-null    int64  
 14  fuel-system       202 non-null    int32  
 15  stroke            202 non-null    int32  
 16  horsepower        202 non-null    int32  
 17  city-mpg          202 non-null    int64  
 18  highway-mpg       202 non-null    int64
```

```
 19  price             202 non-null    float64
dtypes: float64(4), int32(12), int64(4)
memory usage: 23.7 KB
```

In [ ]:

```
plt.figure(figsize=(20,20))
sns.heatmap(df_clean.corr(),annot=True)
plt.show()
```



Purging Features with very low correlation

In [ ]:

```
df_clean = df_clean.drop(['symboling', 'body-style', 'num-of-doors', 'stroke','num-o
```

In [ ]:

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202 entries, 0 to 203
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
```

```
 0   make              202 non-null    int32
 1   fuel-type         202 non-null    int32
 2   aspiration        202 non-null    int32
 3   drive-wheels      202 non-null    int32
 4   engine-location   202 non-null    int32
 5   length            202 non-null    float64
 6   width             202 non-null    float64
 7   height            202 non-null    float64
 8   engine-size       202 non-null    int64
 9   fuel-system       202 non-null    int32
 10  horsepower        202 non-null    int32
 11  city-mpg          202 non-null    int64
 12  highway-mpg       202 non-null    int64
 13  price             202 non-null    float64
dtypes: float64(4), int32(7), int64(3)
memory usage: 18.1 KB
```

In [ ]:
```
X,Y = df_clean.values[:,:-1],df_clean.values[:,-1]
```

In [ ]:
```
X.shape,Y.shape
```

Out[ ]:
```
((202, 13), (202,))
```

Data Normalization

In [ ]:
```
u = np.mean(X, axis=0)
# print(u.shape)
std = np.std(X, axis=0)
X = (X-u)/std
```

In [ ]:
```
ones = np.ones((X.shape[0],1))
X = np.hstack((ones,X))
print(X)
```

```
[[ 1.         -1.98265203  0.32221908 ... -1.46925631 -0.64279561
  -0.54119201]
 [ 1.         -1.98265203  0.32221908 ... -0.63760179 -0.9493947
  -0.68675896]
 [ 1.         -1.82168108  0.32221908 ... -1.63558721 -0.18289698
  -0.10449113]
 ...
 [ 1.          1.3977378   0.32221908 ... -1.02570723 -1.10269424
  -1.12345984]
 [ 1.          1.3977378  -3.10347852 ... -1.58014358  0.12370211
  -0.54119201]
 [ 1.          1.3977378   0.32221908 ... -1.35836904 -0.9493947
  -0.83232592]]
```

Splitting the Data into Test Train Split

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_stat
```

Multiple Linear Regression

In [ ]:

```
def r2_score(y,y_):
    num = np.sum((y-y_)**2)
    denom = np.sum((y-y.mean())**2)
    score = 1-(num/denom)
    return score*100

def train_val_data(X,Y):
    train_X,val_X,train_Y,val_Y = train_test_split(X, Y, test_size=0.05,shuffle=True
    return train_X,val_X,train_Y,val_Y
```

In [ ]:

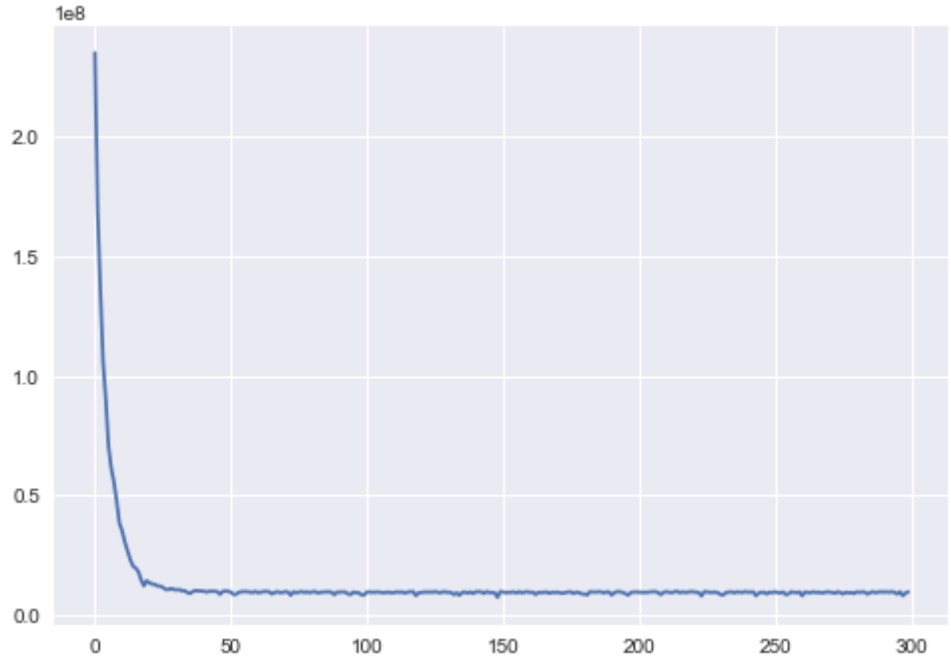
```
def hypothesis(X,theta):
    return np.dot(X,theta)

def error(X,y,theta):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis(X,theta)
    e = (np.sum((y-y_)**2))
    return e/m
def gradient(X,y,theta):
    y_ = hypothesis(X,theta)
    grad = np.dot(X.T,(y_-y))
    m = X.shape[0]
    return grad/m
def gradient_Descent(X,y,learning_rate = 0.1,max_iters = 300):
    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
    val_score = []

    for i in range(max_iters):
        train_X,val_X,train_Y,val_Y = train_val_data(X,y)

        e = error(train_X,train_Y,theta)
        error_list.append(e)

        grad = gradient(train_X,train_Y,theta)
        theta = theta - learning_rate*grad

        y_ = hypothesis(val_X,theta)
        val_score.append(r2_score(val_Y,y_))
    return theta,error_list,val_score
```

In [ ]:

```
theta,error_list,val_score = gradient_Descent(X_train,y_train)
```

In [ ]:

```
plt.plot(error_list)
plt.show()
```



In [ ]:
```
val_score = pd.DataFrame(np.array(val_score),columns=["Val Score"])
print(val_score.describe())
```

```
         Val Score
count   300.000000
mean     53.485459
std     111.013255
min   -1539.427710
25%      55.857035
50%      78.884615
75%      89.711316
max      98.182104
```

In [ ]:
```
y_ = hypothesis(X_test,theta)
print("Overall R2-Score: ",r2_score(y_test,y_))
```

```
Overall R2-Score:  93.73951859962185
```

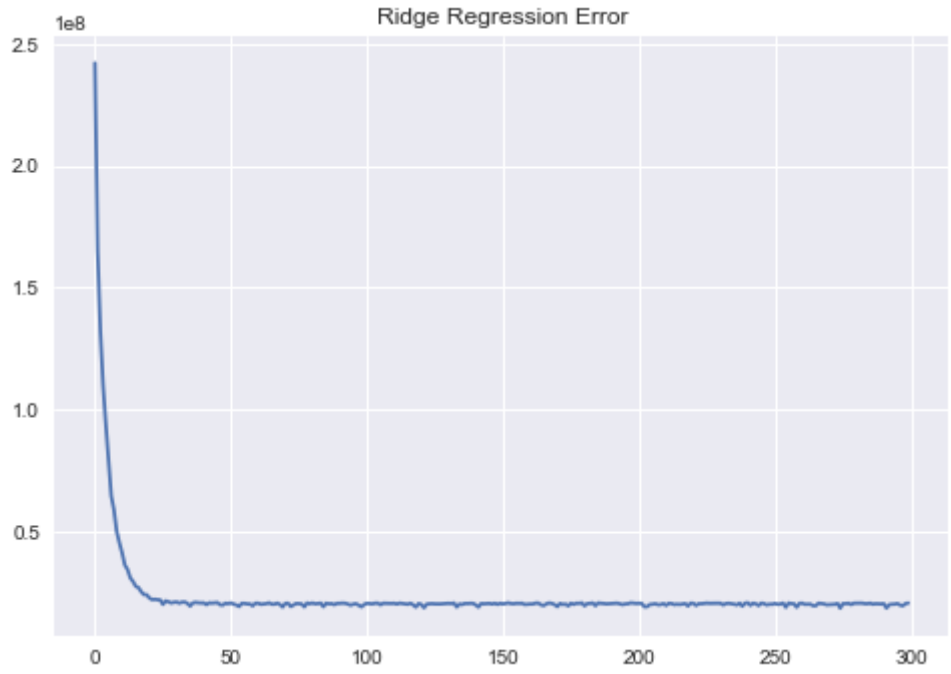
Ridge Regression

In [ ]:
```
def hypothesis_r(X,theta):
    return np.dot(X,theta)

def error_r(X,y,theta, lam):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis_r(X,theta)
    e = np.sum((y-y_)**2) + lam*np.sum(np.square(theta))
    return e/m
def gradient_r(X,y,theta,lam):
    y_ = hypothesis_r(X,theta)
    grad = np.dot(X.T,(y_-y)) + lam*theta
    m = X.shape[0]
    return grad/m
def gradient_Descent_r(X,y,lam,learning_rate = 0.1,max_iters = 300):
    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
    val_score = []
```

```
    for i in range(max_iters):
        train_X,val_X,train_Y,val_Y = train_val_data(X,y)

        e = error_r(train_X,train_Y,theta,lam)
        error_list.append(e)

        grad = gradient_r(train_X,train_Y,theta,lam)
        theta = theta - learning_rate*grad

        y_ = hypothesis_r(val_X,theta)
        val_score.append(r2_score(val_Y,y_))
    return theta,error_list,val_score
```

In [ ]:
```
theta,error_list,val_score = gradient_Descent_r(X_train,y_train,lam=10)
```

In [ ]:
```
plt.plot(error_list)
plt.title("Ridge Regression Error")
plt.show()
```



In [ ]:
```
val_score = pd.DataFrame(np.array(val_score),columns=["Val Score Ridge Regression"])
print(val_score.describe())
```

```
       Val Score Ridge Regression
count                  300.000000
mean                    63.269598
std                     65.456889
min                   -643.204399
25%                     65.353973
50%                     82.525613
75%                     89.126761
max                     97.744985
```

In [ ]:
```
y_ = hypothesis_r(X_test,theta)
print("Overall R2-Score: ",r2_score(y_test,y_))
```

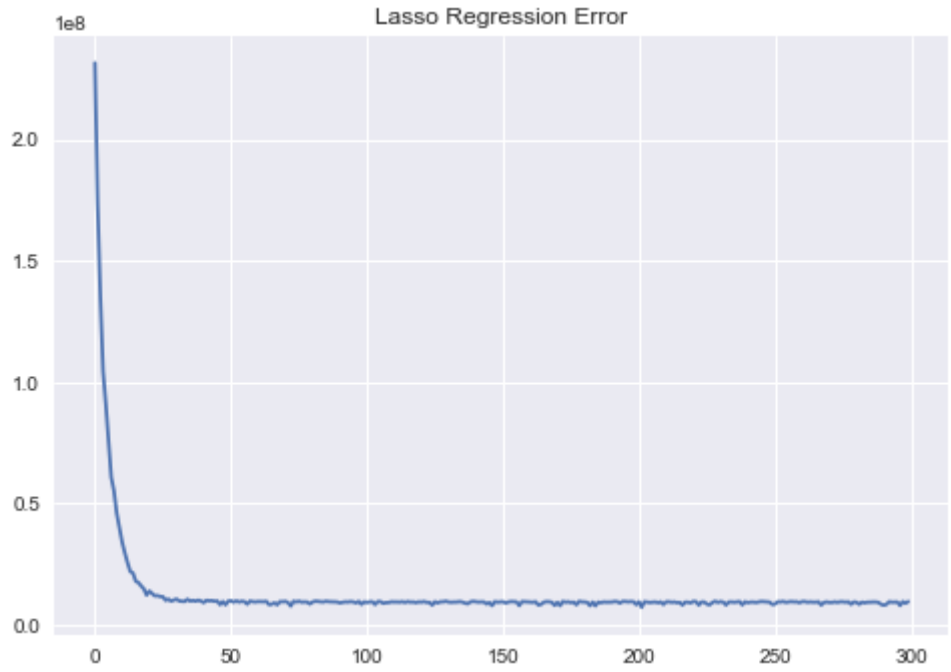
```
Overall R2-Score:  94.26890687581202
```

Lasso Regression

In [ ]:
```
def hypothesis_l(X,theta):
    return np.dot(X,theta)

def error_l(X,y,theta, lam):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis_l(X,theta)
    e = np.sum((y-y_)**2) + lam*np.sum(np.abs(theta))
    return e/m
def gradient_l(X,y,theta,lam):
    y_ = hypothesis_l(X,theta)
    grad = np.dot(X.T,(y_-y)) + lam*(2*np.sinc(theta)-1)
    m = X.shape[0]
    return grad/m
def gradient_Descent_l(X,y,lam,learning_rate = 0.1,max_iters = 300):
    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
    val_score = []

    for i in range(max_iters):
        train_X,val_X,train_Y,val_Y = train_val_data(X,y)

        e = error_l(train_X,train_Y,theta,lam)
        error_list.append(e)

        grad = gradient_l(train_X,train_Y,theta,lam)
        theta = theta - learning_rate*grad

        y_ = hypothesis_l(val_X,theta)
        val_score.append(r2_score(val_Y,y_))
    return theta,error_list,val_score
```

In [ ]:
```
theta,error_list,val_score = gradient_Descent_l(X_train,y_train,lam=10)
```

In [ ]:
```
plt.plot(error_list)
plt.title("Lasso Regression Error")
plt.show()
```



In [ ]:
```
val_score = pd.DataFrame(np.array(val_score),columns=["Val Score Lasso Regression"])
print(val_score.describe())
```

```
       Val Score Lasso Regression
count                  300.000000
mean                    65.963418
std                     47.818503
min                   -281.041242
25%                     63.649129
50%                     81.367128
75%                     90.137017
max                     99.064463
```

In [ ]:
```
y_ = hypothesis_r(X_test,theta)
print("Overall R2-Score: ",r2_score(y_test,y_))
```

```
Overall R2-Score:  93.77253605140649
```

In [ ]:

Logistic Regression

Importing Libraries

In [ ]:
```
import pandas as pd
import numpy as np
```

Reading Dataset

In [ ]:
```
df = pd.read_csv("Train.csv")
```

In [ ]:
```
df.tail()
```

Out[ ]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
1004	1.0	1.0	Blank, Mr. Henry	male	40.0	0.0	0.0	112277	31.0000	A31	C
1005	3.0	0.0	Laitinen, Miss. Kristina Sofia	female	37.0	0.0	0.0	4135	9.5875	NaN	S
1006	1.0	1.0	Newell, Miss. Marjorie	female	23.0	1.0	0.0	35273	113.2750	D36	C
1007	3.0	1.0	Nicola-Yarred, Master. Elias	male	12.0	1.0	0.0	2651	11.2417	NaN	C
1008	3.0	0.0	Corn, Mr. Harry	male	30.0	0.0	0.0	SOTON/OQ 392090	8.0500	NaN	S

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   pclass     1009 non-null   float64
 1   survived   1009 non-null   float64
 2   name       1009 non-null   object 
 3   sex        1009 non-null   object 
 4   age        812 non-null    float64
 5   sibsp      1009 non-null   float64
 6   parch      1009 non-null   float64
 7   ticket     1009 non-null   object 
 8   fare       1008 non-null   float64
 9   cabin      229 non-null    object 
 10  embarked   1008 non-null   object 
 11  boat       374 non-null    object 
 12  body       98 non-null     float64
```

```
 13  home.dest  582 non-null    object
dtypes: float64(7), object(7)
memory usage: 110.5+ KB
```

Dropping Redundant Columns

In [ ]:
```
columns_to_drop = ["cabin","embarked","home.dest","name","body","boat", "ticket"]
```

In [ ]:
```
data_clean = df.drop(columns_to_drop,axis=1)
```

In [ ]:
```
data_clean.head()
```

Out[ ]:

	pclass	survived	sex	age	sibsp	parch	fare
0	3.0	0.0	female	NaN	0.0	0.0	7.750
1	2.0	0.0	male	39.0	0.0	0.0	26.000
2	2.0	1.0	female	40.0	0.0	0.0	13.000
3	3.0	1.0	female	31.0	1.0	1.0	20.525
4	3.0	1.0	female	NaN	2.0	0.0	23.250

Encoding Class Labels to Numeric Labels

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_clean['sex'] = le.fit_transform(data_clean['sex'])
```

In [ ]:
```
data_clean.head()
```

Out[ ]:

	pclass	survived	sex	age	sibsp	parch	fare
0	3.0	0.0	0	NaN	0.0	0.0	7.750
1	2.0	0.0	1	39.0	0.0	0.0	26.000
2	2.0	1.0	0	40.0	0.0	0.0	13.000
3	3.0	1.0	0	31.0	1.0	1.0	20.525
4	3.0	1.0	0	NaN	2.0	0.0	23.250

In [ ]:
```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pclass    1009 non-null   float64
 1   survived  1009 non-null   float64
 2   sex       1009 non-null   int32
 3   age       812 non-null    float64
```

```
 4   sibsp     1009 non-null   float64
 5   parch     1009 non-null   float64
 6   fare      1008 non-null   float64
dtypes: float64(6), int32(1)
memory usage: 51.4 KB
```

In [ ]:
```
data_clean = data_clean.fillna(data_clean['age'].mean())
data_clean = data_clean.fillna(data_clean['fare'].mode())
```

In [ ]:
```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pclass    1009 non-null   float64
 1   survived  1009 non-null   float64
 2   sex       1009 non-null   int32
 3   age       1009 non-null   float64
 4   sibsp     1009 non-null   float64
 5   parch     1009 non-null   float64
 6   fare      1009 non-null   float64
dtypes: float64(6), int32(1)
memory usage: 51.4 KB
```

Dividing Data into X and Y

In [ ]:
```
input_cols = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare']
output_cols = ['survived']
```

In [ ]:
```
X = data_clean[input_cols]
Y = data_clean[output_cols]
```

In [ ]:
```
X.shape,Y.shape
```

Out[ ]:
```
((1009, 6), (1009, 1))
```

Defining Entropy and Information Gain

In [ ]:
```
def entropy(col):
    count = np.unique(col,return_counts=True)
    N = float(col.shape[0])
    ent = 0.0
    for ix in count[1]:
        p = ix/N
        ent+=(-1.0 * p*np.log2(p))
    return ent

def divide_data(x_data,fkey,fval):
    x_right = pd.DataFrame([],columns=x_data.columns)
    x_left = pd.DataFrame([],columns=x_data.columns)
    for ix in range(x_data.shape[0]):
        val = x_data[fkey].loc[ix]
```

```
        if val>fval:
            x_right = x_right.append(x_data.loc[ix])
        else:
            x_left = x_left.append(x_data.loc[ix])
    return x_left,x_right

def info_gain(x_data,fkey,fval):
    left,right = divide_data(x_data,fkey,fval)

    l = float(left.shape[0])/x_data.shape[0]
    r = float(right.shape[0])/x_data.shape[0]

    if(left.shape[0] == 0 or right.shape[0] == 0):
        return -100000
    i_gain = entropy(x_data.survived) - (l*entropy(left.survived) + r*entropy(right.
    return i_gain
```

In [ ]:

```
for fx in X.columns:
    print(fx)
    print(info_gain(data_clean,fx,data_clean[fx].mean()))
```

```
pclass
0.055456910002982474
sex
0.19274737190850932
age
0.001955929827451075
sibsp
0.006492394392888956
parch
0.01975608012294816
fare
0.04242793401428169
```

Implementing Decision Tree Class

In [ ]:

```
class DecisionTree:
    def __init__(self,depth = 0,max_depth = 5):
        self.left = None
        self.right = None
        self.fkey = None
        self.fval = None
        self.max_depth = max_depth
        self.depth = depth
        self.target = None

    def train(self,X_train):
        features = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare']
        info_gains = []
        for ix in features:
            i_gain = info_gain(X_train,ix,X_train[ix].mean())
            info_gains.append(i_gain)
        self.fkey = features[np.argmax(info_gains)]
        self.fval = X_train[self.fkey].mean()

        print("Making Decision Tree, Current Node is: ",self.fkey)

        data_left,data_right = divide_data(X_train,self.fkey,self.fval)
        data_left = data_left.reset_index(drop=True)
        data_right = data_right.reset_index(drop=True)
```

```
        if data_left.shape[0] == 0 or data_right.shape[0] == 0:
            if(X_train.survived.mean()>0.5):
                self.target = "Survived"
            else:
                self.target = "Dead"
            return
        if self.depth >= self.max_depth:
            if(X_train.survived.mean()>0.5):
                self.target = "Survived"
            else:
                self.target = "Dead"
            return

        self.left = DecisionTree(depth=self.depth+1,max_depth=self.max_depth)
        self.left.train(data_left)

        self.right = DecisionTree(depth=self.depth+1,max_depth=self.max_depth)
        self.right.train(data_right)

        if(X_train.survived.mean()>0.5):
            self.target = "Survived"
        else:
            self.target = "Dead"
        return

    def predict(self,test):
        if test[self.fkey] > self.fval:
            if self.right is None:
                return self.target
            return self.right.predict(test)
        else:
            if self.left is None:
                return self.target
            return self.left.predict(test)
```

Creating test and train split

In [ ]:

```
split = int(0.7*data_clean.shape[0])
train_data = data_clean[:split]
test_data = data_clean[split:]
test_data = test_data.reset_index(drop=True)
```

In [ ]:

```
print(train_data.shape)
```

```
(706, 7)
```

Training Decsion Tree

In [ ]:

```
dt = DecisionTree(max_depth=5)
dt.train(train_data)
```

```
Making Decision Tree, Current Node is:  sex
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
```

```
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
```

Predicting test data

In [ ]:

```
y_pred = []
for ix in range(test_data.shape[0]):
    y_pred.append(dt.predict(test_data.loc[ix]))
```

In [ ]:
```
y_actual = test_data[output_cols]
```

In [ ]:
```
le = LabelEncoder()
y_pred = le.fit_transform(y_pred)
```

In [ ]:
```
y_pred = np.array(y_pred).reshape((-1,1))
print(y_pred.shape)
acc = np.sum((y_pred==y_actual))/y_pred.shape[0]
```

```
(303, 1)
```

In [ ]:
```
print(acc)
```

```
survived    0.752475
dtype: float64
```

Creating Random forest Classifier Object using Sk-Learn

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
X_train = train_data[input_cols]
Y_train = np.array(train_data[output_cols]).reshape((-1,))
X_test = test_data[input_cols]
Y_test = np.array(test_data[output_cols]).reshape((-1,))
```

In [ ]:
```
rf = RandomForestClassifier(n_estimators=12,criterion='entropy',max_depth=5)
rf.fit(X_train,Y_train)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy', max_depth=5, n_estimators=12)
```

In [ ]:
```
rf.score(X_train,Y_train)
```

Out[ ]:
```
0.8569405099150141
```

Decision Tree

Importing Necessary Libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
plt.style.use('seaborn')
```

Importing the Data

In [ ]:
```
df = pd.read_csv("diabetes2.csv")
df.head()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [ ]:
```
df.describe()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [ ]:
```
data = df.values
```

Splitting the Data

In [ ]:
```
x = data[:,:-1]
y = data[:,-1]
```

Normalization

In [ ]:
```
#Normalisation
x_mean = x.mean(axis=0)
x_std = x.std(axis=0)

X = (x-x_mean)/x_std
```

In [ ]:
```
X.shape,y.shape
```

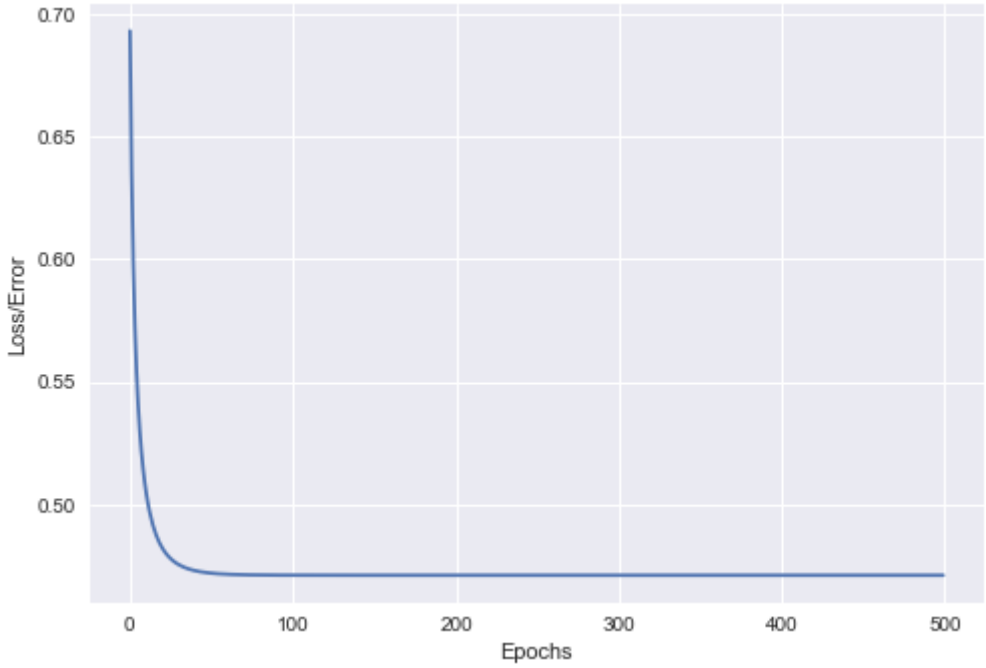
Out[ ]:
```
((768, 8), (768,))
```

Logistic Regression

In [ ]:
```
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))
def hypothesis(X,theta):
    return sigmoid(np.dot(X,theta))
def error(X,y,theta):
    hi = hypothesis(X,theta)
    e = -1*np.mean((y*np.log(hi) + ((1-y)*np.log(1-hi))))
    return e
def gradient(X,y,theta):
    hi = hypothesis(X,theta)
    grad = -(np.dot(X.T,(y-hi)))
    m = X.shape[0]
    return grad/m
def gradientDescent(X,y,lr = 0.1, epochs = 300):
    ones = np.ones((X.shape[0],1))
    X_ = np.hstack((ones,X))
    y = y.reshape((-1,1))

    n = X_.shape[1]
    theta = np.zeros((n,1))
    error_list = []
    for i in range(epochs):
        error_list.append(error(X_,y,theta))
        grad = gradient(X_,y,theta)
        theta = theta - lr*grad
    return theta,error_list
```

In [ ]:
```
theta,err = gradientDescent(X,y, lr=0.5,epochs=500)
```

In [ ]:
```
plt.plot(err)
plt.xlabel("Epochs")
plt.ylabel("Loss/Error")
plt.show()
```



Weights of the Logistic Regression Agorithm

In [ ]:
```
theta
```

Out[ ]:
```
array([[-0.87110174],
       [ 0.41480203],
       [ 1.12354378],
       [-0.25717844],
       [ 0.00986739],
       [-0.13724668],
       [ 0.70675627],
       [ 0.31296112],
       [ 0.17474908]])
```

Making Predictions

In [ ]:
```
def predict(X,theta):
    ones = np.ones((X.shape[0],1))
    X_ = np.hstack((ones,X))
    h = hypothesis(X_,theta)
    output = np.zeros(h.shape)
    output[h>=0.5] = 1
    output = output.astype('int')
    return output
XT_pred = predict(X,theta)
```

Making a Confusion Matrix

In [ ]:
```
CM = confusion_matrix(XT_pred, y, labels=[1,0])
TP=CM[0][0]
FP=CM[0][1]
FN=CM[1][0]
TN=CM[1][1]

ACC = (TP+TN)/(TP+TN+FP+FN)
```

```
print('Accuracy is : \n', ACC)
print('---------------------------------')
Rec = TP/(TP+FN)
print('Recall is : \n', Rec)
print('---------------------------------')
Prec = TP/(TP+FP)
print('Precsion is : \n', Prec)
print('---------------------------------')
F1 = 2 * ((Prec * Rec)/(Prec + Rec))
print('F1 score is : \n', F1)
print('---------------------------------')
```

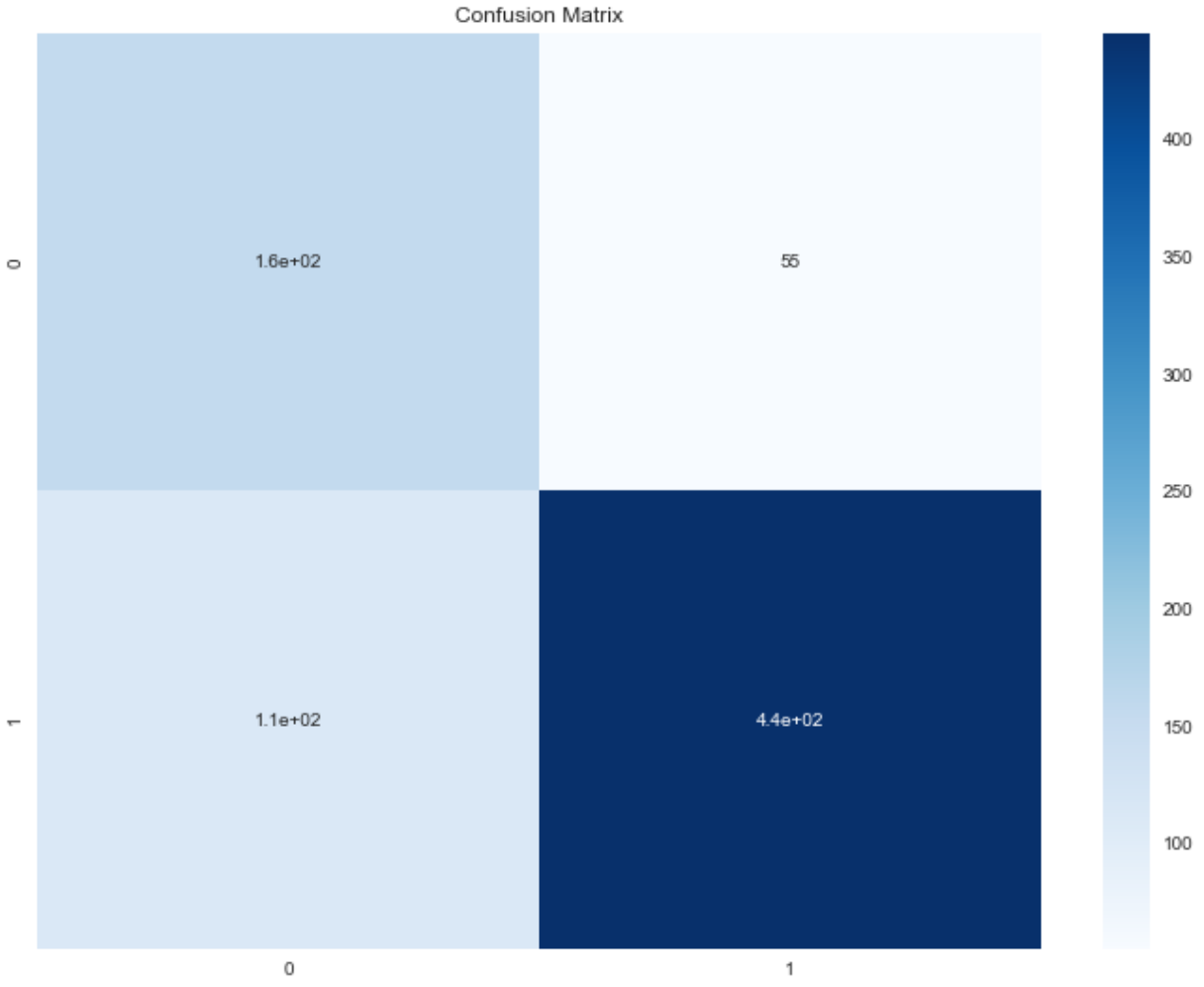
```
Accuracy is :
 0.7825520833333334
---------------------------------
Recall is :
 0.582089552238806
---------------------------------
Precsion is :
 0.7393364928909952
---------------------------------
F1 score is :
 0.651356993736952
---------------------------------
```

In [ ]:

```
import seaborn as sns

plt.figure(figsize=(12,9))
plt.title("Confusion Matrix")
sns.heatmap(CM,annot=True,cmap=plt.cm.Blues)
plt.plot()
```

Out[ ]:

```
[]
```


Confusion Matrix
1.6e+02
55
1.1e+02
4.4e+02
0
1
0
1
400
350
300
250
200
150
100

In [ ]: