LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.04

A.1 Aim: Implement Simple and Multiple Linear Regression on real world dataset and estimate the parameters of regression. Analyze the effect of varying learning rate and number of iterations

i. Implement Gradient Descent algorithm using Numpy on toy dataset and observe the effect of varying learning rate and number of iterations.
ii. Implement Simple and Multiple Linear Regression using scikit learn
iii. Examine the effect of penalizing the parameters

A.2 Prerequisite:

Python Programming, concept of linear regression, ridge regression and Lasso regression
Students should go through the prerequisite documents if any, provided by faculty before attending lab.

A.3 Outcome:

After successful completion of this experiment students will be able to:

1. Identify difference between simple and multiple regression and which one to apply when.
2. Apply simple and multiple regression on real world dataset
3. Examine the effect of regularization on model performance

A.4 Theory

Linear Regression is a **machine learning** algorithm based on supervised **learning**. ... **Linear regression** performs the task to predict a dependent variable value (y) based on a given independent variable (x). More specifically, that y can be calculated from a linear combination of the input variables (x). When there is a single input variable (x), the method is referred to as simple linear regression.

Different techniques can be used to prepare or train the linear regression equation from data, the most common of which is called Ordinary Least Squares. Both the input values (x) and the output value are numeric. The linear equation assigns one scale factor to each input value or column, called a coefficient. One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model would be: $y(pred) = \theta_0 + \theta_1 * x$. In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients. When a coefficient becomes zero, it effectively

removes the influence of the input variable on the model and therefore from the prediction made from the model (0 * x = 0).

Learning a linear regression model means estimating the values of the coefficients used in the representation with the data that we have available. The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between each point to the regression line. It is also called as cost function and is indicated by J(θ) or simply J. In the equation of cost function below, Y is actual value of output and $\hat{y}$ is output value predicted by model.

$$J = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2$$

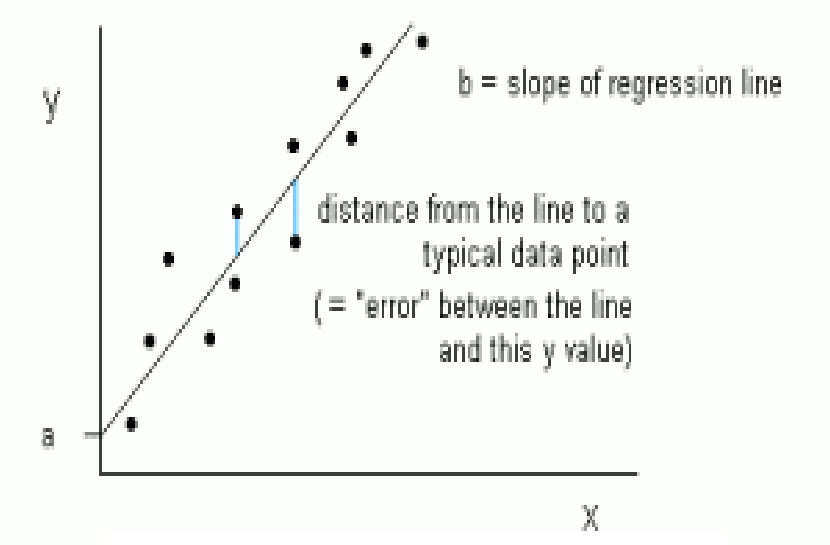


Figure1. Regression model minimizing the

For simple linear regression, y(pred) = b0 + b1*x. The values b0 and b1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.

If the model does not include x i.e. x=0, then the prediction will become meaningless with only b0. For example, we have a dataset that relates height(x) and weight(y). Taking x=0(that is height as 0), will make equation have only b0 value which is completely meaningless as in real-time height and weight can never be zero. This resulted due to considering the model values beyond its scope.

If the model includes value 0, then ‘b0’ will be the average of all predicted values when x=0. But, setting zero for all the predictor variables is often impossible. The value of b0 guarantee that residual have mean zero. If there is no ‘b0’ term, then regression will be forced to pass over the origin. Both the regression co-efficient and prediction will be biased.

When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients.

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all of the squared errors together. This is the quantity that ordinary least squares seeks to minimize.

Gradient Descent

Gradient Descent is the process of minimizing a function by following the gradients of the cost function.

This involves knowing the form of the cost as well as the derivative so that from a given point you know the gradient and can move in that direction, e.g. downhill towards the minimum value.

Cost Function

It turns out that to make the best line to model the data, we want to pick parameters $\beta\beta$ that allows our predicted value to be as close to the actual value as possible. In other words, we want the distance or residual between our hypothesis $h(x)$ and y to be minimized.

So we formally define a cost function using ordinary least squares that is simply the sum of the squared distances. To find the liner regression line, we minimize:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Hypothesis that we're trying to find is given by the linear model:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

The parameters of the model are the theta values. We adjust θj to minimize the cost function (θ)

And we can use batch gradient descent where each iteration performs the update

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

Gradient descent simply is an algorithm that makes small steps along a function to find a local minimum.

The mathematical expression of linear regression is as follows.

$$Y = \theta_0 + \theta_1 X_1 + \theta_2 X_1 + \cdots + \theta_p X_{p-1} + \epsilon$$

This can then be expressed in matrix version.

$$Y = X\,\theta + e \quad where\ \theta\ is\ \theta_0, \theta_1, \ldots \theta_p$$

Math for Ridge Regression:

OLS method basically finds the $\beta\beta$'s to minimize Residual Sum of Squares (RSS).

$$RSS=\sum_{i=1}^{n}(y_i-\hat{y})^2=\sum_{j=1}^{n}(y_i-\sum^{p} x_{ij}\theta_j)^2$$

What Ridge Regression does is penalize RSS by adding another term and for searching the minimization.

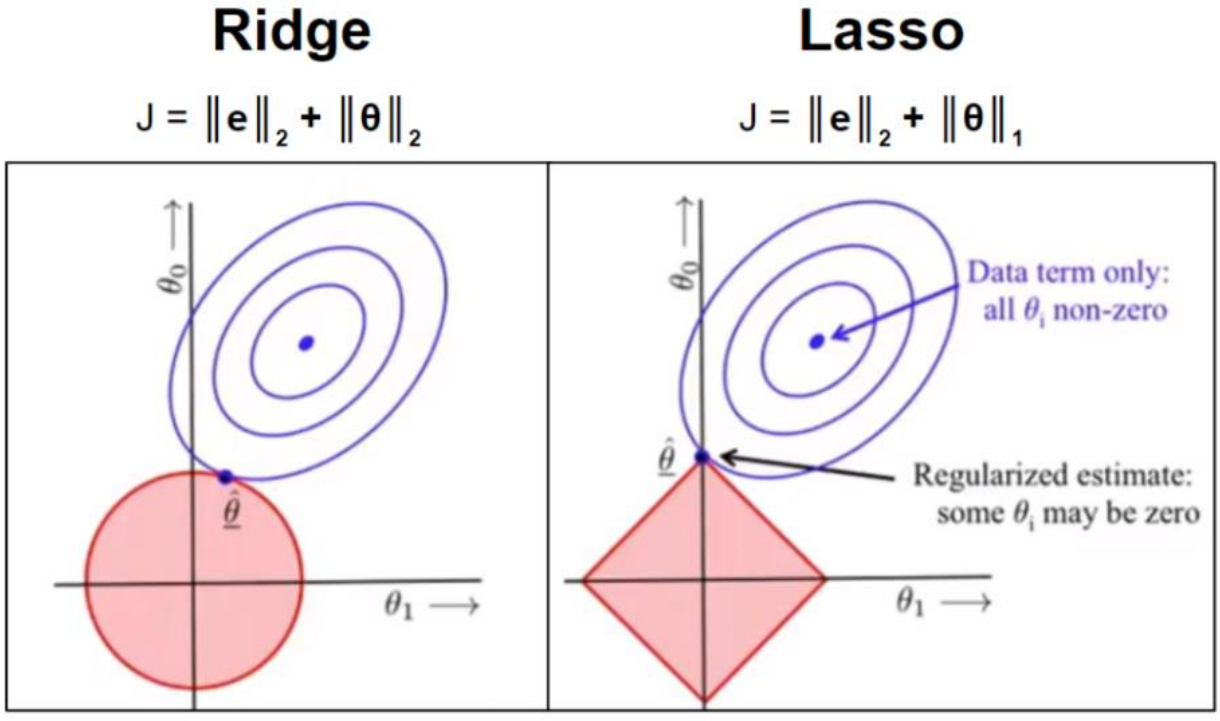
$$RSS\ with\ Penalty=\sum_{i=1}^{n}(y_i-\sum_{j=1}^{p} x_{ij}\theta_j)^2+\lambda\sum_{j=1}^{p}\theta j^2$$

where λ is a constant

As with Ridge Regression, OLS method is modified for Lasso Regression. In fact, only difference is the penalty term.

$$RSS\ with\ Penalty=\sum_{i=1}^{n}(y_i-\sum_{j=1}^{p} x_{ij}\theta_j)^2+\lambda\sum_{j=1}^{p}|\theta j|$$

Any coefficients can never be zero in Ridge Regression, but can be in Lasso Regression. This would be understood in an easier way with a geometric figure. Let's say there is a model with two variables and we want to know their coefficients; θ1 and θ2. They can be mapped as follows.



Ridge vs. Lasso the contours in blue represent sets of RSS and the center is the coefficients found by OLS. The square and the circle are L1 penalty and L2 penalty, respectively. Recall that the constraints for each regression to understand the shapes of penalty.

Lasso constraint: $|\beta 1|+|\beta 2|=1$

Ridge constraint: $\beta 1^2+\beta 2^2=1$

The points the shapes meet the edge contours are where shrinkage methods estimate the coefficients. As seen on the maps, the estimates by shrinkage methods are found less than those of OLS because the shapes play a role of penalizing. Depending on sizes of the shapes and contours, these points would locate differently. This is a trade-off between the penalty term and RSS.
Again, the left represents Ridge Regression and the right represents LASSO Regression. Moving the penalty terms up and down, the contours would be pushed up and down, changing the estimates where they contact. The difference between the two regression are here that the estimates can be zero for some variable in Lasso, but can't be in Ridge Regression. The Lasso on the map shows this property, having $\theta=0$.

Usage of Ridge/Lasso Regression:
Recall that the two of them were devised to upgrade the traditional linear regression. Therefore, they are capable of what linear regression is capable of. Just as linear regression can be used for regression and also clasification, they can do the same job mostly more effectively. However, Ridge Regression is not capable of variable selection as explained with the equation. On the contrary, Lasso Regression can select variables by manipulating the lambda which can make some θ(s) equal to zero.

They can be used both for regression and classification problems. Ridge Regression is good at handling overfitting. Lasso Regression can be used for feature selection

A.5 Task:

1. Select dataset with (.csv file) of sufficiently large size (>300 rows). It should contain columns with different data types.

2. Identify input features and output in the dataset.

3. Write a code for gradient descent algorithm to update parameters of regression.

4. Apply regression on dataset using Scikit learn library and compute parameters and R2 value.

5. Apply ridge and lasso regression and compare performance with that of regression obtained in step 3

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Black board access available)

Roll No. B027	Name: Aman Kothari
Class : B tech	Batch : B1
Date of Experiment:	Date of Submission
Grade :	

B.1 Software Code written by student:

(Paste your code completed during the 2 hours of practical in the lab here)

```
df1 = pd.read_csv('Life Expectancy Data.csv')

df1.head()


df1.set_index("Country",inplace= True)


df1.columns


useless_col = ['percentage expenditure','Hepatitis B','Status',
    'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
    'Diphtheria ', ' HIV/AIDS', 'Population',
    ' thinness  1-19 years', ' thinness 5-9 years',
    'Income composition of resources']
df1.drop(useless_col,axis=1,inplace=True)


df1.head()
```

```
df1.info()
df1.isnull().sum()

df1.Alcohol.fillna(0.01,inplace=True)
b=df1.GDP.median()
df1.GDP.fillna(b,inplace=True)
df1.dropna(inplace=True)

df1.isnull().sum()

df1.corr()

sns.heatmap(df1.corr())

X = df1.values[:, 2:7]  #  input from last 5 columns
y = df1.values[:, 1]  # Life expectency as result
m = len(y) # Number of training examples

print('Total no of training examples (m) = %s \n' %(m))

for i in range(5):
    print('x =', X[i, ], ', y =', y[i])
```

```
#Normalising values as they are not in same range
def feature_normalize(X):
  mu1 = np.mean(X, axis = 0)
  sigma1 = np.std(X, axis= 0, ddof = 1)
  X_norm = (X - mu1)/sigma1
  return X_norm, mu1, sigma1

X, mu1, sigma1 = feature_normalize(X)

print('mu= ', mu1)
print('sigma= ', sigma1)
print('X_norm= ', X[:5])

mu_testing = np.mean(X, axis = 0) # mean
mu_testing

sigma_testing = np.std(X, axis = 0, ddof = 1) # mean
sigma_testing

X = np.hstack((np.ones((m,1)), X))
X[:5]

from sklearn.model_selection import train_test_split
X, x_test, y, y_test = train_test_split(X,y,test_size =0.2)
```

```
def compute_cost(X, y, theta):
  predictions = X.dot(theta)
  errors = np.subtract(predictions, y)
  sqrErrors = np.square(errors)
  J = 1/(2 * m) * errors.T.dot(errors)

  return J

def gradient_descent(X, y, theta, alpha, iterations):
  cost_history = np.zeros(iterations)

  for i in range(iterations):
    predictions = X.dot(theta)
    errors = np.subtract(predictions, y)
    sum_delta = (alpha / m) * X.transpose().dot(errors);
    theta = theta - sum_delta;

    cost_history[i] = compute_cost(X, y, theta)

  return theta, cost_history

theta = np.zeros(6)
iterations = 400;
```

```
alpha = 0.15;

theta, cost_history = gradient_descent(X, y, theta, alpha, iterations)
print('Final value of theta =', theta)
print('First 5 values from cost_history =', cost_history[:5])
print('Last 5 values from cost_history =', cost_history[-5 :])

import matplotlib.pyplot as plt
plt.plot(range(1, iterations +1), cost_history, color ='blue')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel("Number of iterations")
plt.ylabel("cost (J)")
plt.title("Convergence of gradient descent")

#Effext of changing learning rate on convergence
iterations = 400;
theta = np.zeros(6)

alpha = 0.005;
theta_1, cost_history_1 = gradient_descent(X, y, theta, alpha, iterations)

alpha = 0.01;
theta_2, cost_history_2 = gradient_descent(X, y, theta, alpha, iterations)
```

```
alpha = 0.02;
theta_3, cost_history_3 = gradient_descent(X, y, theta, alpha, iterations)

plt.plot(range(1, iterations +1), cost_history_1, color ='black', label = 'alpha = 0.005')
plt.plot(range(1, iterations +1), cost_history_2, color ='blue', label = 'alpha = 0.01')
plt.plot(range(1, iterations +1), cost_history_3, color ='green', label = 'alpha = 0.02')

plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel("Number of iterations")
plt.ylabel("cost (J)")
plt.title("Effect of Learning Rate On Convergence of Gradient Descent")
plt.legend()

iterations = 100;
theta = np.zeros(6)

alpha = 1.32;
theta_6, cost_history_6 = gradient_descent(X, y, theta, alpha, iterations)

plt.plot(range(1, iterations +1), cost_history_6, color ='brown')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
```

```
plt.xlabel("Number of iterations")
plt.ylabel("cost (J)")
plt.title("Effect of Large Learning Rate On Convergence of Gradient Descent")

for i in range(50):
    hu = (np.array(x_test[i]))
    ki=hu[1:6]
    normalize_test_data = ((ki - mu1) / sigma1)
    normalize_test_data = np.hstack((np.ones(1), normalize_test_data))
    y_pred = normalize_test_data.dot(theta_3)
    print('Predicted age : {},real y value: {}'.format(y_pred, y_test[i]) )

#using sklearn's linear regression

from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
# Instantiate model
lm2 = LinearRegression()
# Fit Model
lm2.fit(X_train, y_train)
# Predict
```

```
y_pred2 = lm2.predict(X_test)

# RMSE

print('RMSE: ',np.sqrt(metrics.mean_squared_error(y_test, y_pred2)))

# calculate our own accuracy where prediction within 10% is ok

diff2 = (y_pred2 / y_test * 100)

print('Mean of results: ',diff2.mean())

print('Deviation of results: ',diff2.std())

print('Results within 10% support/resistance: ', len(np.where(np.logical_and(diff2>=90, diff2<=110))[0]) / len(y_pred2) * 100)
```

B.2 Input and Output:

(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)

```
[20] df1 = pd.read_csv('Life Expectancy Data.csv')
     df1.head()
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	I
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	

5 rows × 22 columns

```
[28] df1.Alcohol.fillna(0.01,inplace=True)
     b=df1.GDP.median()
     df1.GDP.fillna(b,inplace=True)
     df1.dropna(inplace=True)
```

```
[64] df1.isnull().sum()
```

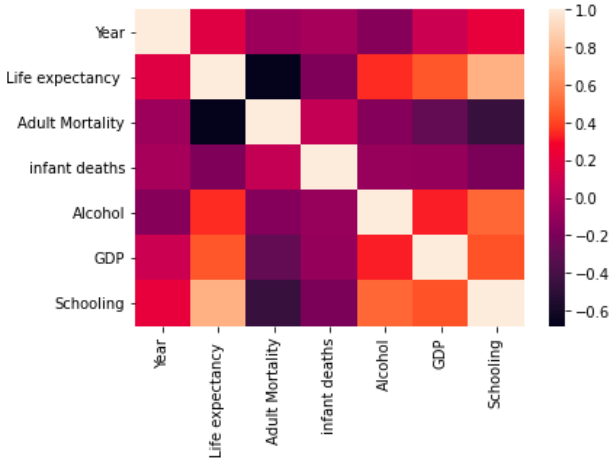
```
Year                 0
Life expectancy      0
Adult Mortality      0
infant deaths        0
Alcohol              0
GDP                  0
Schooling            0
dtype: int64
```

```
[30] df1.corr()
```

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	GDP	Schooling
Year	1.000000	0.171042	-0.075688	-0.037298	-0.152626	0.096364	0.213265
Life expectancy	0.171042	1.000000	-0.684585	-0.179548	0.348985	0.447565	0.751975
Adult Mortality	-0.075688	-0.684585	1.000000	0.063906	-0.163748	-0.288519	-0.454612
infant deaths	-0.037298	-0.179548	0.063906	1.000000	-0.094712	-0.101356	-0.195202
Alcohol	-0.152626	0.348985	-0.163748	-0.094712	1.000000	0.326498	0.499465
GDP	0.096364	0.447565	-0.288519	-0.101356	0.326498	1.000000	0.437770
Schooling	0.213265	0.751975	-0.454612	-0.195202	0.499465	0.437770	1.000000

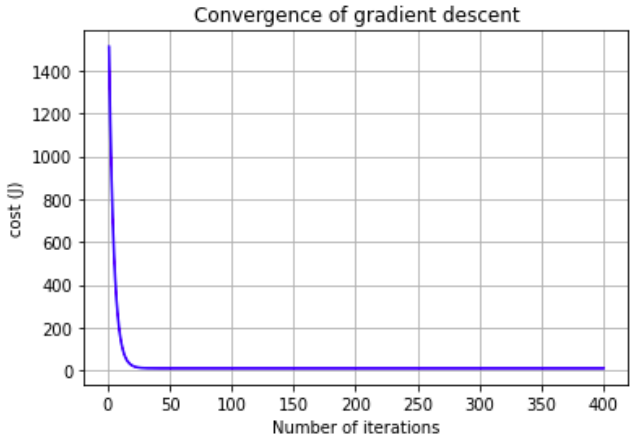
```
sns.heatmap(df1.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c8ade9490>
```

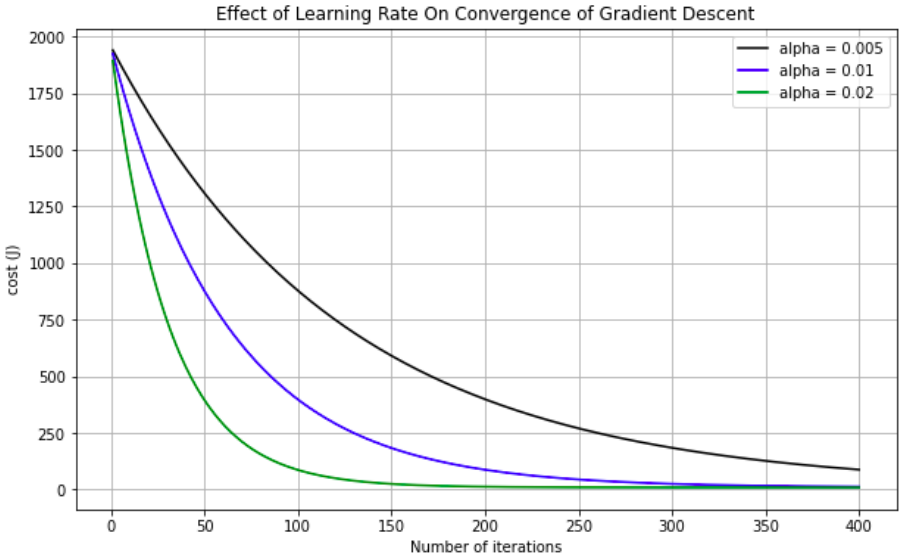



```
52] import matplotlib.pyplot as plt
    plt.plot(range(1, iterations +1), cost_history, color ='blue')
    plt.rcParams["figure.figsize"] = (10,6)
    plt.grid()
    plt.xlabel("Number of iterations")
    plt.ylabel("cost (J)")
    plt.title("Convergence of gradient descent")
```

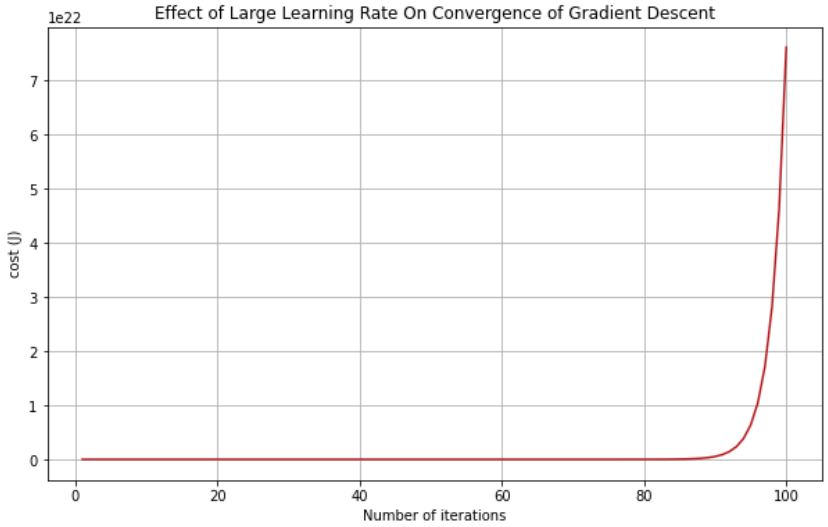
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



```
<matplotlib.legend.Legend at 0x7f7c8eb34890>
```




```
Text(0.5, 1.0, 'Effect of Large Learning Rate On Convergence of Gradient Descent')
```



```
#using sklearn's linear regression

from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
# Instantiate model
lm2 = LinearRegression()
# Fit Model
lm2.fit(X_train, y_train)
# Predict
y_pred2 = lm2.predict(X_test)
# RMSE
print('RMSE: ',np.sqrt(metrics.mean_squared_error(y_test, y_pred2)))
# calculate our own accuracy where prediction within 10% is ok
diff2 = (y_pred2 / y_test * 100)
print('Mean of results: ',diff2.mean())
print('Deviation of results: ',diff2.std())
print('Results within 10% support/resistance: ', len(np.where(np.logical_and(diff2>=90, diff2<=110))[0]) / len(y_pred2) *
```

```
RMSE:  4.695956885630938
Mean of results:  101.23542496545107
Deviation of results:  8.127135445581205
Results within 10% support/resistance:  89.71119133574007
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)

Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function. The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

In conclusion, after completing the experiment I was able to Identify difference between simple and multiple regression and which one to apply when, apply simple and multiple regression on real world dataset and examine the effect of regularization on model performance.

B.5 Question of Curiosity

1. What is the observation about the parameters before and after applying regularization? Plot the graph/s for the same.

Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or under fitting.

Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.