Foundations of Data Science Fall 2022 - Homework 1 (32 points)

Student Name: Yamini Lakshmi Narasimhan

Student Net Id: yl9822

Part 0: Data Exploration and Linear Regression (5 Points)

In [1]:
```
# Importing all important packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Downloading data

fn_src ='https://drive.google.com/uc?export=download&id=1RzQpKONOcXSMxH2ZzOI4i\
fn_dst ='eeg_dat.p'

import os
from six.moves import urllib

if os.path.isfile(fn_dst):
    print('File %s is already downloaded' % fn_dst)
else:
    print('Fetching file %s [53MB].  This may take a minute..' % fn_dst)
    urllib.request.urlretrieve(fn_src, fn_dst)
    print('File %s downloaded' % fn_dst)
```
```
File eeg_dat.p is already downloaded
```

In [3]:
```
# Loading data

import pickle
fn = 'eeg_dat.p'
with open(fn, 'rb') as fp:
    [X,Y] = pickle.load(fp)
```

> **1. (1 Point)** *Display the first 5 rows of X and Y and print the dimensions for both datasets*

In [4]:
```
# Write your code here
```

```
print("Shape of X",X.shape)
print("Shape of X[0]",X[0].shape)
print("Shape of Y",Y.shape)
print("Shape of Y[0]",Y[0].shape)
print("First 5 rows of X\n",X[0:5])
print("First 5 rows of Y\n",Y[0:5])
```

```
Shape of X (305, 22494)
Shape of X[0] (22494,)
Shape of Y (305, 85)
Shape of Y[0] (85,)
First 5 rows of X
 [[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [-0.00529827  0.00491268  0.00598048 ...  0.00162549 -0.0023215
   0.00115952]
 [-0.01522969  0.00059051 -0.00191475 ...  0.00331738 -0.00214887
   0.00160143]]
First 5 rows of Y
 [[ 0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.        ]
 [-0.09276205 -0.08985079 -0.07608291 -0.05530946 -0.03231258 -0.01211071
   0.00080852  0.00313692 -0.00678247 -0.02869592 -0.06052095 -0.09864336
  -0.1384954  -0.17518806 -0.20431671 -0.22255981 -0.22819203 -0.22131916
  -0.20387241 -0.17923614 -0.15177045 -0.12609152 -0.1063793  -0.09570701
  -0.09557988 -0.10568318 -0.12395217 -0.14689077 -0.17011059 -0.18902799
  -0.19959717 -0.19891403 -0.1857485  -0.16069391 -0.12614103 -0.08592358
  -0.04474724 -0.00750158  0.02150801  0.0391705   0.04407476  0.03669187
   0.01933823 -0.00419942 -0.02922969 -0.05085636 -0.06472243 -0.06767898
  -0.05825114 -0.0368678  -0.00578963  0.03120213  0.06941139  0.10389435
   0.13022606  0.14516165  0.14714178  0.13655022  0.11566809  0.0883866
   0.05964747  0.03476927  0.01866437  0.01515595  0.02640921  0.05260544
   0.09186162  0.14047642  0.19336142  0.24469149  0.28863894  0.32010343
   0.33535487  0.33248343  0.31159913  0.27481392  0.225944    0.17002921
   0.11269917  0.05947506  0.01514124 -0.01681553 -0.03456445 -0.0380913
  -0.0291009 ]
 [ 0.01241803 -0.00569839 -0.02801367 -0.05359459 -0.08104797 -0.10851966
  -0.13379319 -0.15448279 -0.16826742 -0.17320027 -0.16799938 -0.15227921
  -0.12673569 -0.09313222 -0.05422283 -0.01347576  0.02528186  0.05832127
   0.08244709  0.09545931  0.09641075  0.08583552  0.0656907   0.03920295
   0.01049631 -0.01589401 -0.03554677 -0.04473862 -0.04091731 -0.0230484
   0.00821874  0.05053668  0.10010532  0.15206901  0.2010497   0.24171443
   0.26942073  0.28069672  0.2736623   0.24824396  0.20617207  0.15082053
   0.08683377  0.01960157 -0.04528859 -0.10268976 -0.14839354 -0.17960091
  -0.19515903 -0.1956481  -0.18327567 -0.16156722 -0.13492094 -0.10808104
  -0.08557384 -0.07114955 -0.06736727 -0.07527035 -0.0942851  -0.12226838
  -0.15576918 -0.19040635 -0.22139751 -0.24410465 -0.25457232 -0.25000458
  -0.22908469 -0.19217531 -0.14128675 -0.07991302 -0.01264023  0.05527446
   0.11845781  0.1719523   0.21171481  0.23496873  0.24047115  0.22856528
   0.20111359  0.16122806  0.11293382  0.06073957  0.00915674 -0.03771494
  -0.0765598 ]]
```

Background: This is an ECG dataset that contains voltage measurements from a number of points on the scalp. A common task is to estimate which parts of the brain caused the measured response, which can help identify which parts of the brain are involved in specific tasks. However, the number of possible locations in the brain is much larger than the number of measurements, which makes this an appropriate task to assess the use of regularization to determine the brain region that is active under stimulus.

In particular, there are three key variables:

- `nt` = number of time steps that we measure data
- `nchan` = number of channels (i.e. electrodes) measured in each time step
- `ncur` = number of currents in the brain that we want to estimate.

Each current comes from one brain region (called a *voxel*) in either the `x`, `y` or `z` direction. So,

```
nvoxels = ncur / 3
```

The components of the `X` and `Y` matrices are:

- `Y[i,k]` = electric field measurement on channel `i` at time `k`
- `X[i,j]` = sensitivity of channel `i` to current `j`.

In [5]:

```
print("Just trying to understand the data")
print(len(X[0]))
print(len(Y))

df_dataset = pd.DataFrame()
df_dataset["X"] = list(X)
df_dataset["Y"] = list(Y)
max_list =[]
min_list = []

for i in X:
    max_list.append(max(i))
    min_list.append(min(i))

df_dataset["X_min"] = min_list
df_dataset["X_max"] = max_list

max_list =[]
min_list = []

for i in Y:
    max_list.append(max(i))
    min_list.append(min(i))

df_dataset["Y_min"] = min_list
df_dataset["Y_max"] = max_list

print("Unique_counts for X_max", len(df_dataset["X_max"].unique()))
print("Unique_counts for X_min", len(df_dataset["X_min"].unique()))

print("Unique_counts for Y_max", len(df_dataset["Y_max"].unique()))
print("Unique_counts for Y_min", len(df_dataset["Y_min"].unique()))
```

```
Just trying to understand the data
22494
305
Unique_counts for X_max 303
Unique_counts for X_min 303
Unique_counts for Y_max 303
Unique_counts for Y_min 303
```

In [6]:

```
df_dataset.head()
```

Out[6]:

	X	Y	X_min	X_max	Y_min	Y_r
0	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.000000	0.000000	0.000000	0.000
1	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.000000	0.000000	0.000000	0.000
2	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.000000	0.000000	0.000000	0.000
3	[-0.005298272870862167, 0.004912682004587315, ...	[-0.09276204596036008, -0.08985079131166167, -...	-0.026475	0.020312	-0.228192	0.335
4	[-0.015229694817309507, 0.0005905057848969534,...	[0.012418027849942948, -0.005698385917103518, ...	-0.032046	0.028179	-0.254572	0.280

> **2. (1 Point)** Split the data into training and testing. What split do you use, and why?

[Place your answer here regarding the split]

ANSWER

Available Options

1. Simple Random Technique because each value is unique like for X it measures sensitivity for each i and j similarly for

Y that measures electric field measurement for each i at a time k and since there is no grouping exactly I tried grouping based on maximum and minimum values obtained at a time for X and Y but since most of them are unique that is 303/305 no point

Direction of current information is also not given so can't group based on that as well. So Systemic, Stratified and Cluster sampling is ruled out

Taking a 20% split for testset and 80% is trainset

In [7]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, rando
print(X_train.shape)
print(Y_train.shape)
```
```
(244, 22494)
(244, 85)
```

> **3. (3 Points)** Perform linear regression. As a reminder, the optimization for linear regression using ordinary least squares is written below, the data consists of n observations {x_i,y_i}. Each observation i includes a scalar response y_i and a column vector x_i of p variables (regressors).

> > The evaluation metric used for linear regression is the $R^2$ score. Explain what $R^2$ tells us, and interpret your results based on this.*

\begin{align} \text{Ordinary Least Squares} \end{align}

\begin{align} \underset{\beta}{\operatorname{arg min}} \left[\sum_{i=1}^n \left(y_i - β_o - \sum_{j=1}^p β_jx_{ij} \right)^2 \right] \end{align}

In [8]:
```
# Write your code here
from sklearn.linear_model import LinearRegression,Ridge, RidgeCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt


linear_regression_model = LinearRegression()
reg = linear_regression_model.fit(X_train, Y_train)

Y_predicted = linear_regression_model.predict(X_test)
r2 = r2_score(Y_test, Y_predicted)
print("Linear Regressor R2 score",r2)
```

```
Linear Regressor R2 score -5.7463712081808165
```

$$ R^2 =1- SS_{regression}/ SS_{Total} $$

> > Place your written answer here.

ANSWER

The objective of ordinary least squared regression is to get a line which minimized the sum squared error. Ordinary least Squared Regression and R-squared are two different things. You do a regression to correlate one dependent variable to many independent variables and use the R-squared to see if this regression makes sense.

1. $R^2$ is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model.
2. Whereas correlation explains the strength of the relationship between an independent and dependent variable, $R^2$ explains to what extent the variance of one variable explains the variance of the second variable. So, if the $R^2$ of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs.
3. $R^2$(coefficient of determination) regression score function.
4. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

$R^2$ is always between 0 and 1:

0 represents a model that does not explain any of the variation in the response variable around its mean. The mean of the dependent variable predicts the dependent variable as

well as the regression model. 1 represents a model that explains all the variation in the response variable around its mean. Usually, the larger the R2, the better the regression model fits your observations

Since R^2 is negative in this case it means its worse than the ground truth model that is the model that predicts the same value for all input for example when the ground truth y is non-constant, a constant model that always predicts the average y disregarding the input features would get a R^2 score of 0.0. -In other words, the mean of the data is a better model than the regression.

-chosen model does not follow the trend of the data, so fits worse than a horizontal line

Part 1: Ridge Regularization (10 Points)

> **1. (1 Point)** *Now we will use ridge regression, a method that is used to analyse data that suffers from multicollinearity. Notice the regularization term added to the optimization below, which uses the squared value of $β$. It is also known as L2 regularization. First, what is the degree of collinearity in the data? Show your work and give rationale for the degree of collinearity.*

\begin{align} \text{Ridge Regression} \end{align}
\begin{align} \underset{\beta}{\operatorname{arg min}} \left[\sum_{i=1}^n \left(y_i - βo - \sum_{j=1}^p β_jx_{ij} \right)^2 + λ\sum_{j=1}^p|β_j|^2 \right] \end{align}

For finding collinearity I am using Pearson's Correlation because

1. It gives information about the magnitude of the association, or correlation
2. As well as direction if it affects negatively or positively
3. It provides a standardized, absolute measure of the strength of the relationship, bounded by -1.0 and 1.0

VIF (Variable Inflation Factors). VIF determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable. VIF score of an independent variable represents how well the variable is explained by other independent variables. R^{2} value is determined to find out how well an independent variable is described by the other independent variables.

- Training a regressor everytime and then finding R^{2} VIF = 1-1/R^{2} is time consuming
- VIF does'nt have upper bound so to compare it across features makes it difficult

That's why I used Pearson's coefficient

Using Correlation_matrix

In [9]:
```
df_corr = pd.DataFrame(X)
au_corr = df_corr.corr(method = 'pearson')
display(au_corr)
```

	0	1	2	3	4	5	6	
0	1.000000	-0.120606	0.024609	0.612473	-0.256557	-0.082040	0.879364	-0.0865
1	-0.120606	1.000000	0.987928	-0.224377	0.848086	0.793126	-0.032677	0.866(
2	0.024609	0.987928	1.000000	-0.135492	0.823442	0.796310	0.098403	0.856!
3	0.612473	-0.224377	-0.135492	1.000000	-0.255341	0.021565	0.244308	-0.0465
4	-0.256557	0.848086	0.823442	-0.255341	1.000000	0.949631	-0.106986	0.7535
...	...	...	...	...	...	...	...	
22489	0.015851	-0.039152	-0.039815	0.034141	-0.055252	-0.047494	-0.003583	-0.0444
22490	0.014878	-0.099530	-0.084680	0.010353	-0.125295	-0.110548	0.004193	-0.0793
22491	-0.012453	-0.000755	-0.003816	-0.005486	0.024929	0.022430	-0.007126	0.0104
22492	0.009996	0.011270	0.011402	0.004891	-0.012469	-0.013313	0.005108	-0.0056
22493	-0.000688	-0.016880	-0.014927	0.001102	-0.019550	-0.016018	-0.002556	-0.0074

22494 rows × 22494 columns

Answer

Plotted Correlation matrix for all the features in X, we can see that they're highly correlated for example feature 4 and feature 5 are correlated with value of 0.949631

There is a high degree of collinearity in the data as many other columns exhibit similar characteristic another example of highly correlated features are 3 and 9

Using Pearson's coefficient

In [10]:
```
from scipy.stats import pearsonr

df_corr_pc_dict ={"column":[],"max_corr_column":[], "max_corr":[], "min_corr_co
for i in range(20):
    min_corr = 1
    max_corr = -1
    max_corr_index = i
    min_corr_index = i
    df_corr_pc_dict["column"].append(i)
    for j in range(X_train.shape[1]):
        if j!=i:
            corr, _ = pearsonr(X_train[:,i], X_train[:,j])
            max_corr_index = j if max_corr<corr else max_corr_index
            max_corr = max(max_corr, corr)
            min_corr_index = j if min_corr>corr else min_corr_index
            min_corr = min(min_corr,corr)
    df_corr_pc_dict["max_corr_column"].append(max_corr_index)
    df_corr_pc_dict["max_corr"].append(max_corr)
    df_corr_pc_dict["min_corr_column"].append(min_corr_index)
    df_corr_pc_dict["min_corr"].append(min_corr)

df_corr_pc = pd.DataFrame.from_dict(df_corr_pc_dict)

print("This df essentially shows for features till 20 which features is maximum
df_corr_pc
```

```
This df essentially shows for features till 20 which features is maximum colli
near in the negative(inversely proportionate) and positive(directly proportion
ate) direction
```

Out[10]:

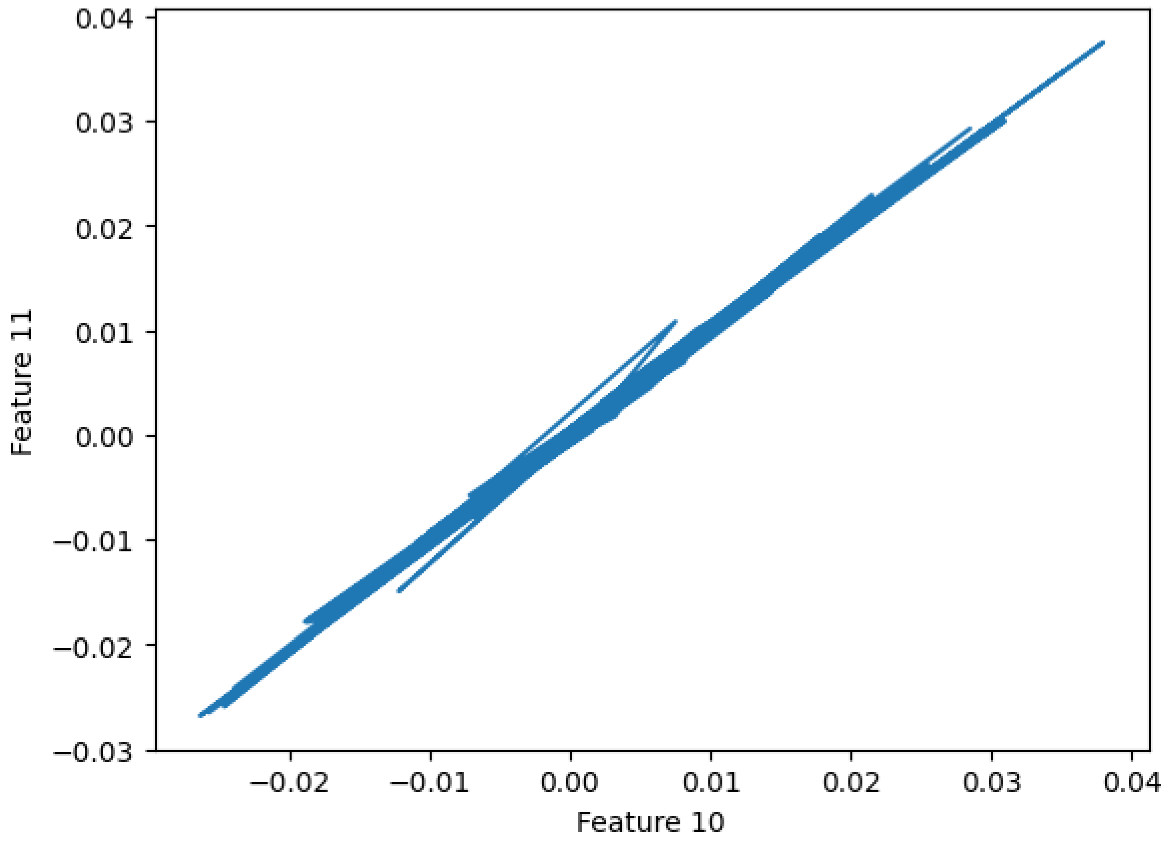
	column	max_corr_column	max_corr	min_corr_column	min_corr
0	0	24	0.990883	11420	-0.526933
1	1	22	0.987868	557	-0.562868
2	2	1	0.987558	473	-0.562414
3	3	69	0.988843	235	-0.549358
4	4	91	0.970264	11246	-0.480402
5	5	71	0.966713	11257	-0.499186
6	6	54	0.984978	11594	-0.473715
7	7	31	0.981602	674	-0.520705
8	8	104	0.979151	755	-0.533026
9	9	72	0.992724	11330	-0.538643
10	10	11	0.998497	548	-0.525648
11	11	10	0.998497	548	-0.517458
12	12	114	0.991158	538	-0.558260
13	13	115	0.991204	11243	-0.473634
14	14	116	0.990514	11209	-0.487238
15	15	3	0.986104	235	-0.547251
16	16	17	0.988469	614	-0.501555
17	17	16	0.988469	614	-0.498551
18	18	69	0.985766	400	-0.563684
19	19	70	0.969250	11251	-0.487977

In [11]:

```
plt.plot(X_train[:,10],X_train[:,11])
plt.xlabel('Feature 10')
plt.ylabel('Feature 11')
```

Out[11]:

```
Text(0, 0.5, 'Feature 11')
```



Answer

For the first 20 features in X, there is always a column that is extremely correlated with the one of them as you can see feature 10 correlates with feature 11 with pearson coefficient of 0.998 which makes it highly correlated. The plot shows the same as well.

There is a high degree of collinearity between Feature 10 and 11 as shown in the plot and Pearson Correlation value of 0.998 So you can see that each feature has maximum collinearity of atleast 0.95 with some feature in the dataset

> **1b**. **(2 Points)** *Now implement ridge regression using sklearn. Start with the default alpha value, and print the $R^2$ score*

In [12]:
```
# Write your code here
ridge_regression_model = Ridge(alpha=1.0)
ridge_regression_model.fit(X_train, Y_train)

Y_predicted = ridge_regression_model.predict(X_test)
r2 = r2_score(Y_test, Y_predicted)
print("Ridge Regressor R2 score", r2)
```

```
Ridge Regressor R2 score 0.14919226523568813
```

> **2. (5 Points)** *Now we will look at what happens when varying the parameter. Try 10 different values for alpha and print the*

> > *alpha value with corresponding R^2 score using Cross-Validation for training data. Describe how you pick the range of alpha values to examine.*

> > [Place your reasoning for the alpha values here]

Answer

Regularization strength - alpha; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization.

Constant that multiplies the L2 term, controlling regularization strength. alpha must be a non-negative float i.e. in [0, inf). When alpha = 0, the objective is equivalent to ordinary least squares, solved by the LinearRegression object. For numerical reasons, using alpha = 0 with the Ridge object is not advised. Instead, you should use the LinearRegression object. If an array is passed, penalties are assumed to be specific to the targets.

Since the full range is [0,inf) - I have to find the point where it reaches peak and drops so I start with 0 to 100 then find peak and then narrow down the range.

Step 1: Take a larger range like 1- 100

Step 2: Then keep choosing smaller range based on the graph

In [13]:
```python
X_train_train, X_val, Y_train_train, Y_val = train_test_split(X_train, Y_train,
print(X_train_train.shape)
print(Y_train_train.shape)

print(X_val.shape)
print(Y_val.shape)
```

```
(195, 22494)
(195, 85)
(49, 22494)
(49, 85)
```

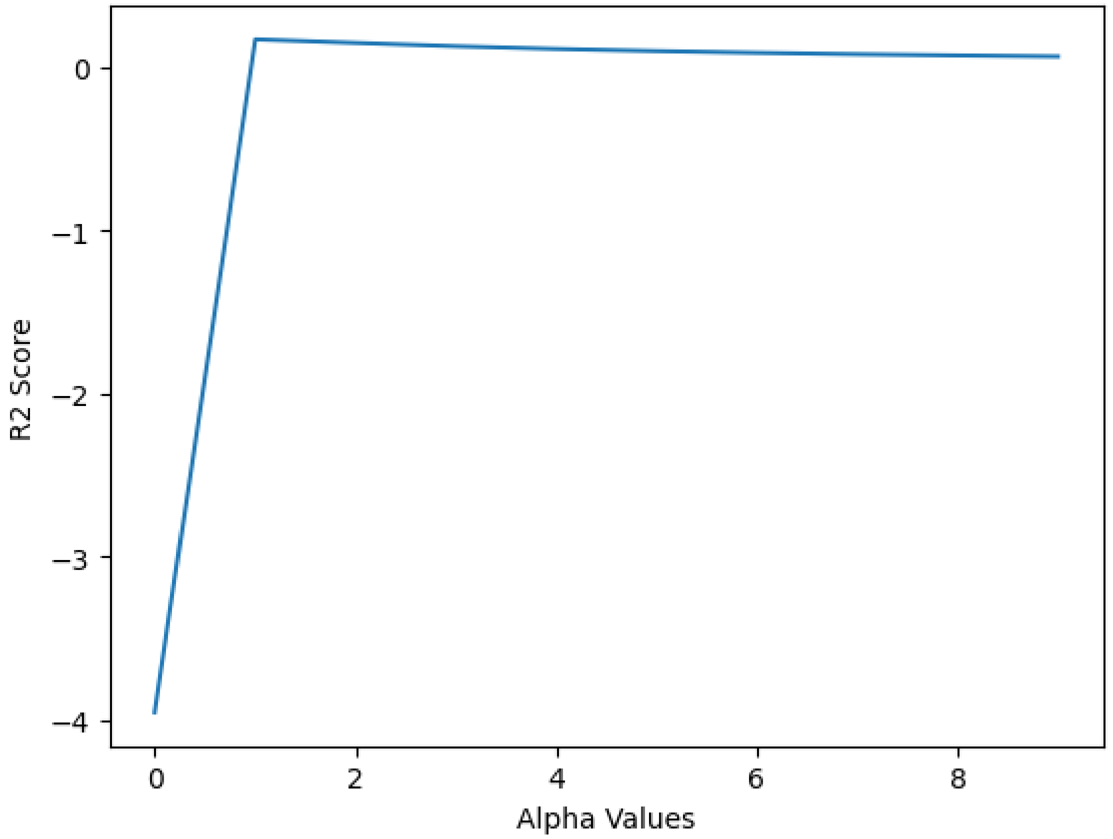
Using Ridge() - for multiple values of ${R_2}$ score ran it in loop

In [14]:
```python
def ridge_only(min_alpha, max_alpha, addition, divide=10):

    r2_list = []
    alpha_list = []
    max_r2 = float('-inf')
    alpha_value = 0
    for alpha in range(min_alpha, max_alpha, addition):
        ridge_regression_model = Ridge(alpha=alpha/divide)
        ridge_regression_model.fit(X_train_train, Y_train_train)
        Y_predicted = ridge_regression_model.predict(X_val)
        r2 = r2_score(Y_val, Y_predicted)
        r2_list.append(r2)
```

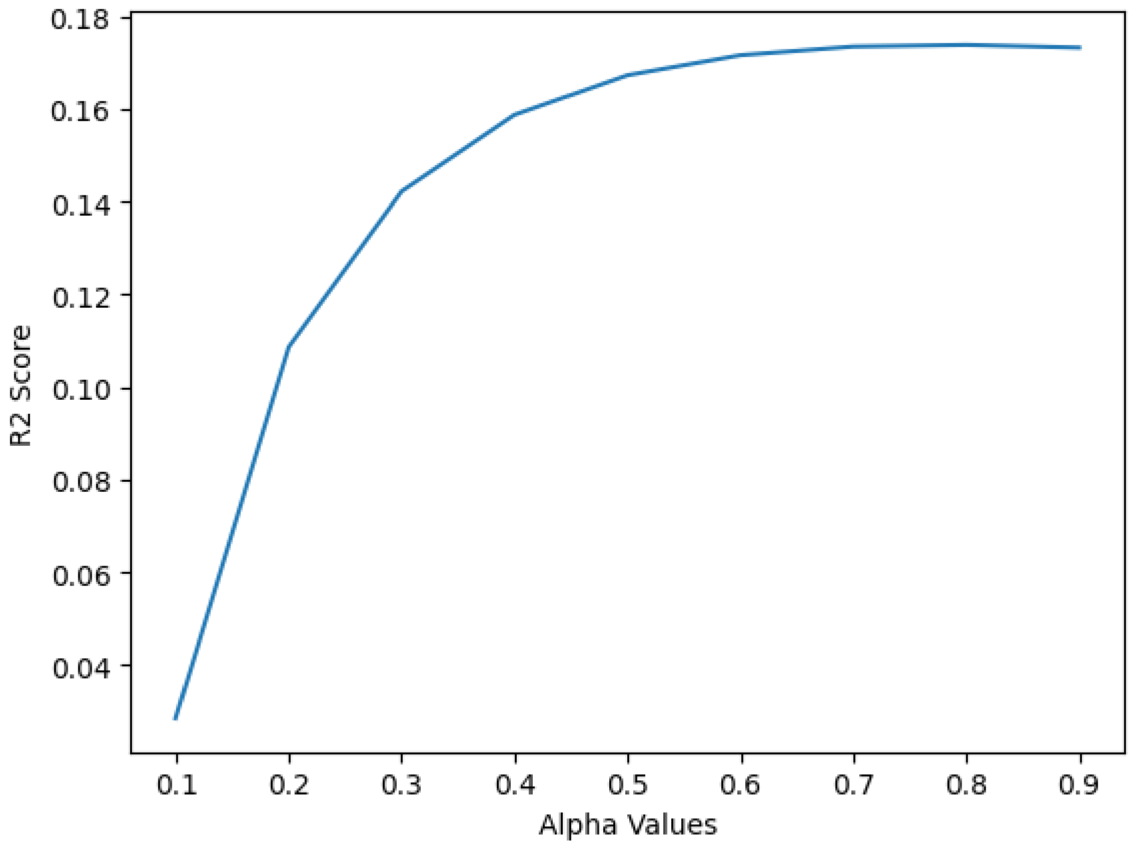
```
            alpha_list.append(alpha/divide)
            alpha_value = float(alpha/divide) if max_r2 < r2 else alpha_value
            max_r2 = r2 if max_r2 < r2 else max_r2


        plt.plot(alpha_list,r2_list)
        plt.xlabel('Alpha Values')
        plt.ylabel('R2 Score')
        plt.show()

        print("Maximum Value of R2 is ",max_r2," For alpha = ",float(alpha_value))
        print("ALPHA Range = (%s,%s)" %(str(min_alpha/divide),str(max_alpha/divide)
        return max_r2, alpha_value, alpha_list, r2_list


print("STEP 1\n")
max_r2, alpha_value, alpha_list, r2_list = ridge_only(0,100,10,10)
print("\n\nR2 Values", r2_list)
print("for corresponding alpha List Values", alpha_list)

print("\n\nSince the graph has dipped after peaking at my alpha = %s \nI am goi

print("STEP 2\n")
max_r2, alpha_value,_,_ = ridge_only(1,10,1,10)
```

```
STEP 1
```

```
/Users/yamini/Library/Python/3.8/lib/python/site-packages/sklearn/linear_mode
l/_ridge.py:251: UserWarning: Singular matrix in solving dual problem. Using l
east-squares solution instead.
  warnings.warn(
```



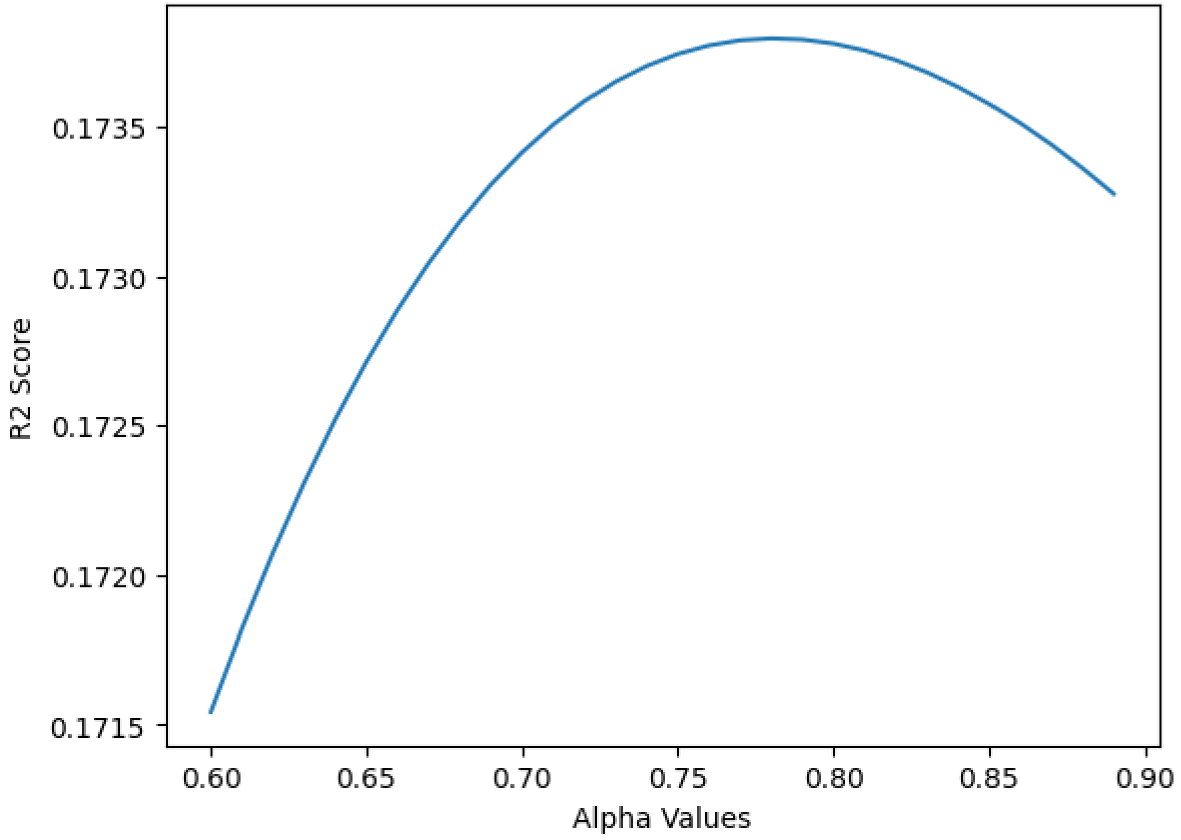
```
Maximum Value of R2 is  0.17196033893255458  For alpha =  1.0
ALPHA Range = (0.0,10.0)


R2 Values [-3.950484015167676, 0.17196033893255458, 0.15023759110945728, 0.129
59316960697576, 0.11314901606352498, 0.10004081908971094, 0.08940208077084012,
0.08060403986484985, 0.07320525763950972, 0.0668924998459723]
for corresponding alpha List Values [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
8.0, 9.0]


Since the graph has dipped after peaking at my alpha = 1.0
I am going to run it in the smaller set to find the exact value of alpha that
is from 0.1 to 1


STEP 2
```



```
Maximum Value of R2 is  0.1737773479316073  For alpha =  0.8
ALPHA Range = (0.1,1.0)
```

In [15]:
```python
print("STEP 3")
max_r2_ridge, alpha_value_ridge,_,_ = ridge_only(60,90,1,100)
```

```
STEP 3
```



```
Maximum Value of R2 is  0.17379474436870712  For alpha =  0.78
ALPHA Range = (0.6,0.9)
```

In [16]:
```
print("Ridge : On the Test Data")
alpha_test_ridge = alpha_value_ridge
ridge_regression_model = Ridge(alpha = alpha_value_ridge)
ridge_regression_model.fit(X_train_train, Y_train_train)
Y_predicted = ridge_regression_model.predict(X_test)
r2_test_ridge = r2_score(Y_test, Y_predicted)
print("R2_score on the test Data",r2_test_ridge)
print("For alpha value = ",alpha_test_ridge)
```

```
Ridge : On the Test Data
R2_score on the test Data 0.14438812846069535
For alpha value =  0.78
```

> **3. (2 Points)** What is the highest R^2 you can achieve?

In [31]:
```
print(" Highest R\u00b2 for test that can be achieved for this particular datas
```

```
 Highest R² for test that can be achieved for this particular dataset is 0.144
388 and that is at alpha = 0.780000
```

Part 2: Lasso Regularization (12 Points)

> **1. (3 Points)** *Another method often used to improve performance is Lasso regularization. Lasso regression improves performance by using shrinkage, where data values are shrunk*

> *towards a central point, like the mean. Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This can result in sparse models with few coefficients. Some of the coefficients can become zero and eliminated from the model. Implement this using sklearn with default parameter values and print the R^2 score*

In [18]:
```
lasso_regression_model = Lasso(alpha=1.0)
lasso_regression_model.fit(X_train, Y_train)
Y_predicted = lasso_regression_model.predict(X_test)
r2 = r2_score(Y_test, Y_predicted)
print("Alpha Value =",1.0)
print("R\u00b2 =",r2)
```

```
Alpha Value = 1.0
R² = -0.03926823982754499
```

\begin{align} \text{LASSO Regression} \end{align}

\begin{align} \underset{\beta}{\operatorname{arg min}} \left[\sum_{i=1}^n \left(y_i - βo - \sum_{j=1}^p β_jx_ij \right)^2 + λ\sum_{j=1}^p|β_j| \right] \end{align}

> **2. (5 Points)** *Try 10 different values for the regularization parameter Alpha and print the alpha value with corresponding R^2 score using cross-validation. Again, give your reasoning for the choices of alpha. If you choose to tune any other hyperparameters of the model then justify your reason for it*

Answer :

Constant that multiplies the L1 term, controlling regularization strength. alpha must be a non-negative float i.e. in [0, inf). When alpha = 0, the objective is equivalent to ordinary least squares, solved by the LinearRegression object. For numerical reasons, using alpha = 0 with the Lasso object is not advised.

Step 1: Take a larger range like 1- 100 and then focus on smaller range based on the peak
Step 2: Then keep choosing smaller range based on the graph

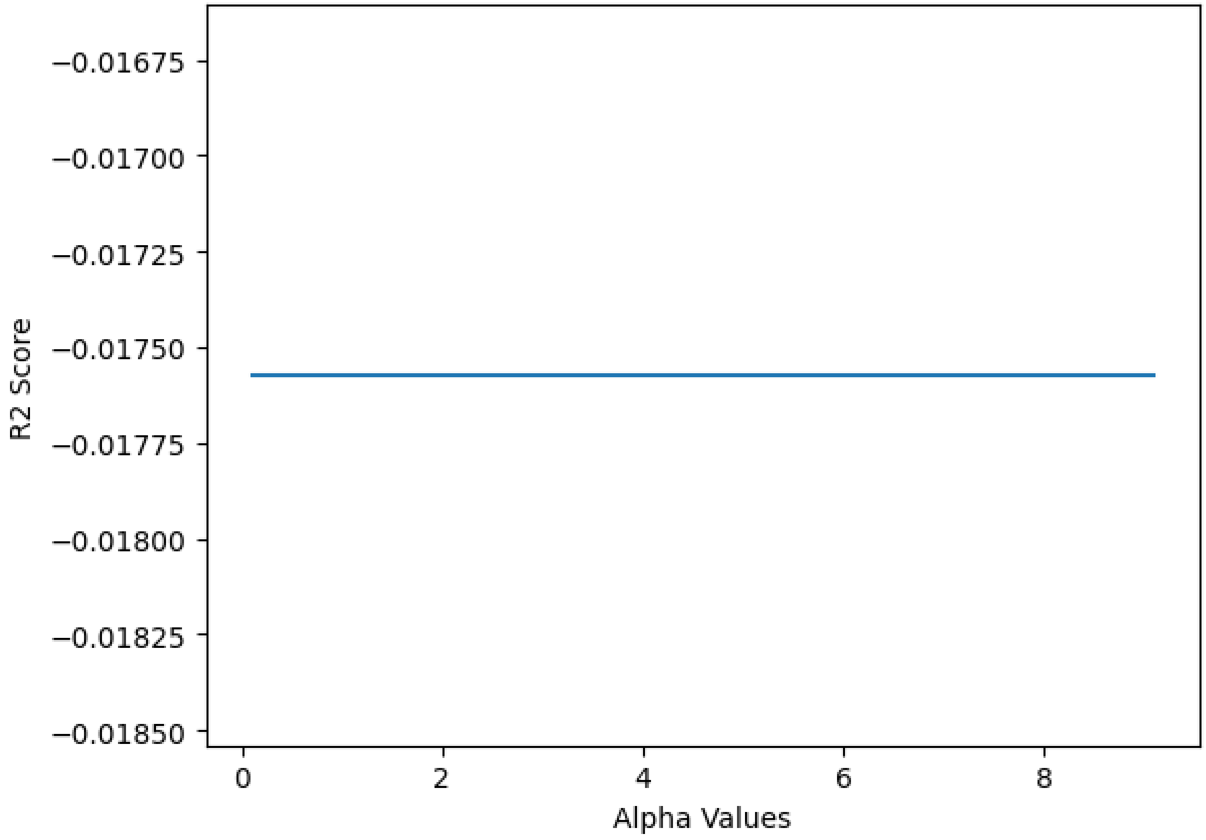
> **3. (2 Point)** *Plot a graph for different Alpha values and corresponding R^2 score. Explain the shape of the curve.*

Using Lasso() - This does not have Cross Validation but gives multiple values for $R2$ score so ran it in loop

In [32]:
```
from sklearn import linear_model
```

```
def lasso_only(min_alpha, max_alpha, addition, divide=1000):
    r2_list = []
    alpha_list = []
    max_r2 = float('-inf')
    alpha_value = 0
    for alpha in range(min_alpha, max_alpha, addition):
        lasso_regression_model = Lasso(alpha=alpha/divide)
        lasso_regression_model.fit(X_train_train, Y_train_train)
        Y_predicted = lasso_regression_model.predict(X_val)
        r2 = r2_score(Y_val, Y_predicted)
        r2_list.append(r2)
        alpha_list.append(alpha/divide)
        alpha_value = float(alpha/divide) if max_r2 < r2 else alpha_value
        max_r2 = r2 if max_r2 < r2 else max_r2


    plt.plot(alpha_list,r2_list)
    plt.xlabel('Alpha Values')
    plt.ylabel('R2 Score')
    plt.show()

    print("Maximum Value of R\u00b2 is ",max_r2," For alpha = ",str(alpha_value
    return max_r2, alpha_value

print("STEP 1")
max_r2, alpha_value = lasso_only(1,100,10,10)
print("\n\nThe graph is is straight line so looking at a range smaller than tha


print("STEP 2")
max_r2, alpha_value = lasso_only(1,10,1,1000)
print("Peaking at ",alpha_value, "so drilling further down")
```

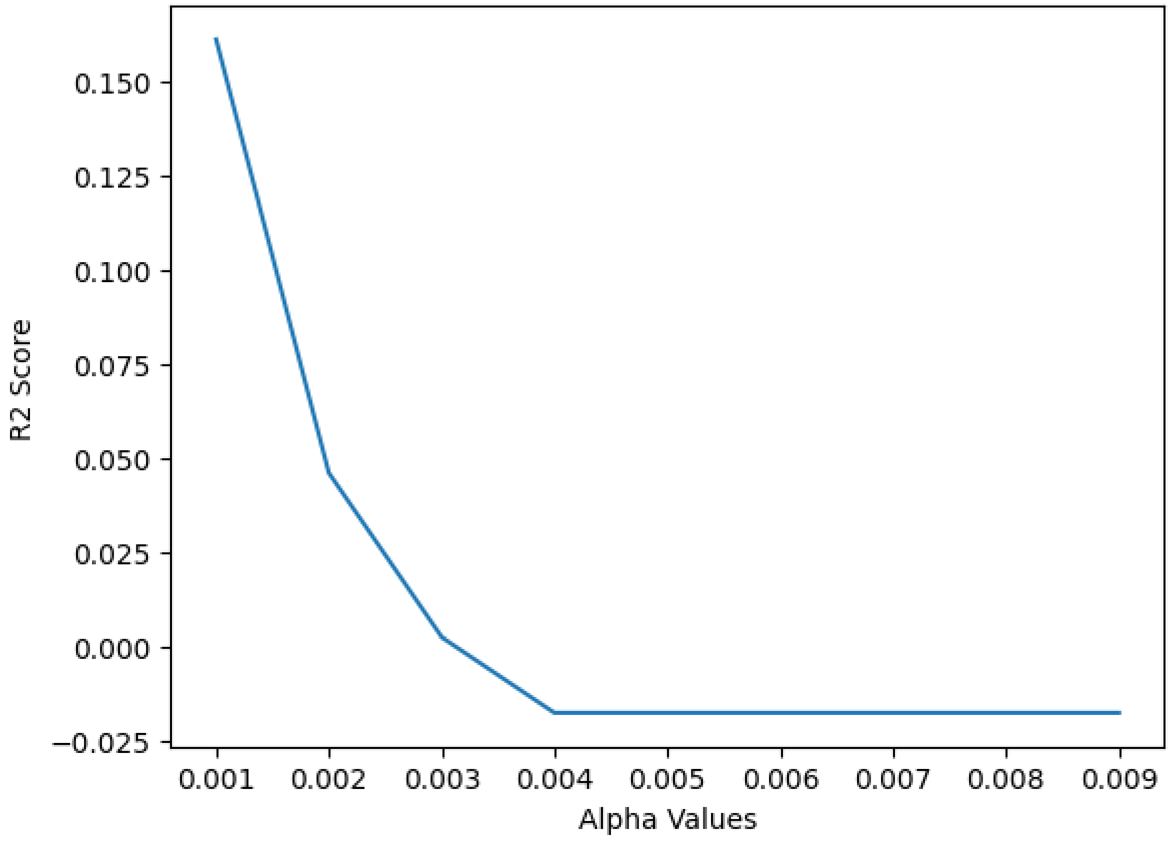
```
STEP 1
```



```
Maximum Value of R² is  -0.017572677740573677  For alpha =  0.1  for alpha in
range (0.1,10.0)
```

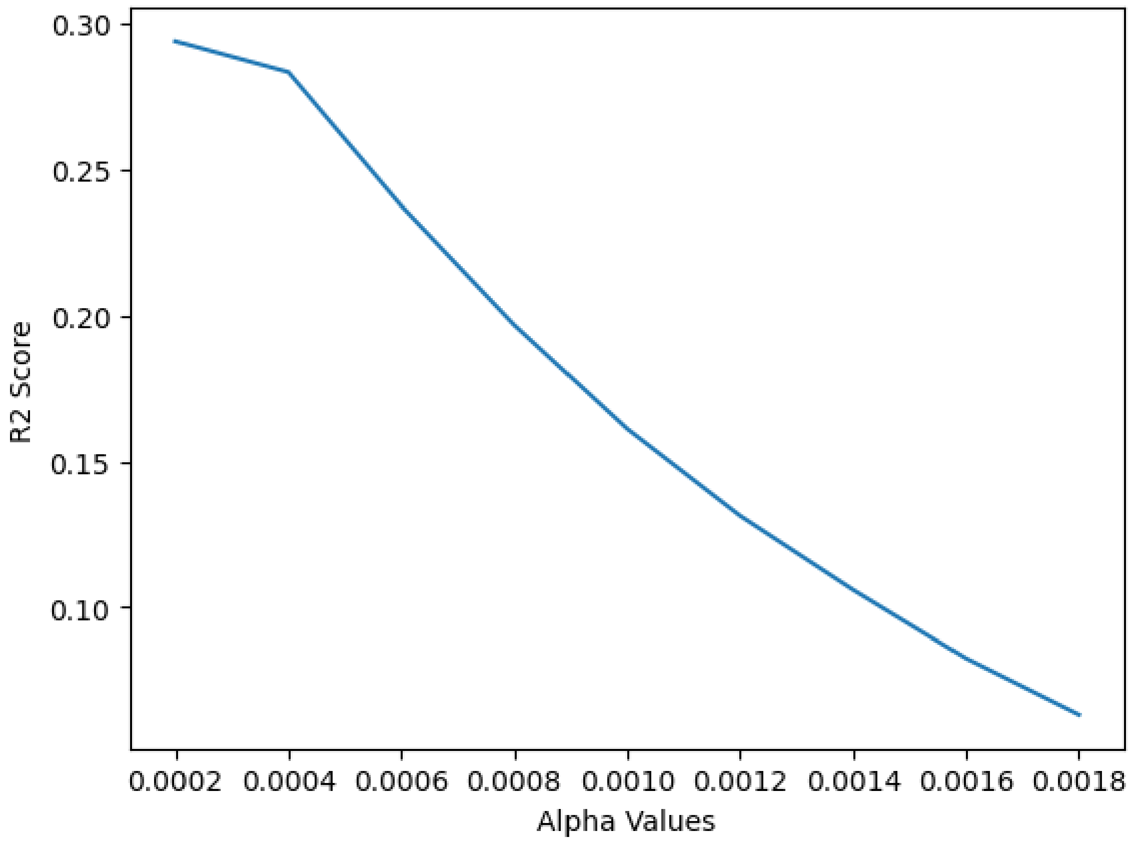
The graph is is straight line so looking at a range smaller than that like 0-1 to see if there is any change in the graph

STEP 2



```
Maximum Value of R² is  0.16111213151372356  For alpha =  0.001  for alpha in
range (0.001,0.01)
Peaking at  0.001 so drilling further down
```

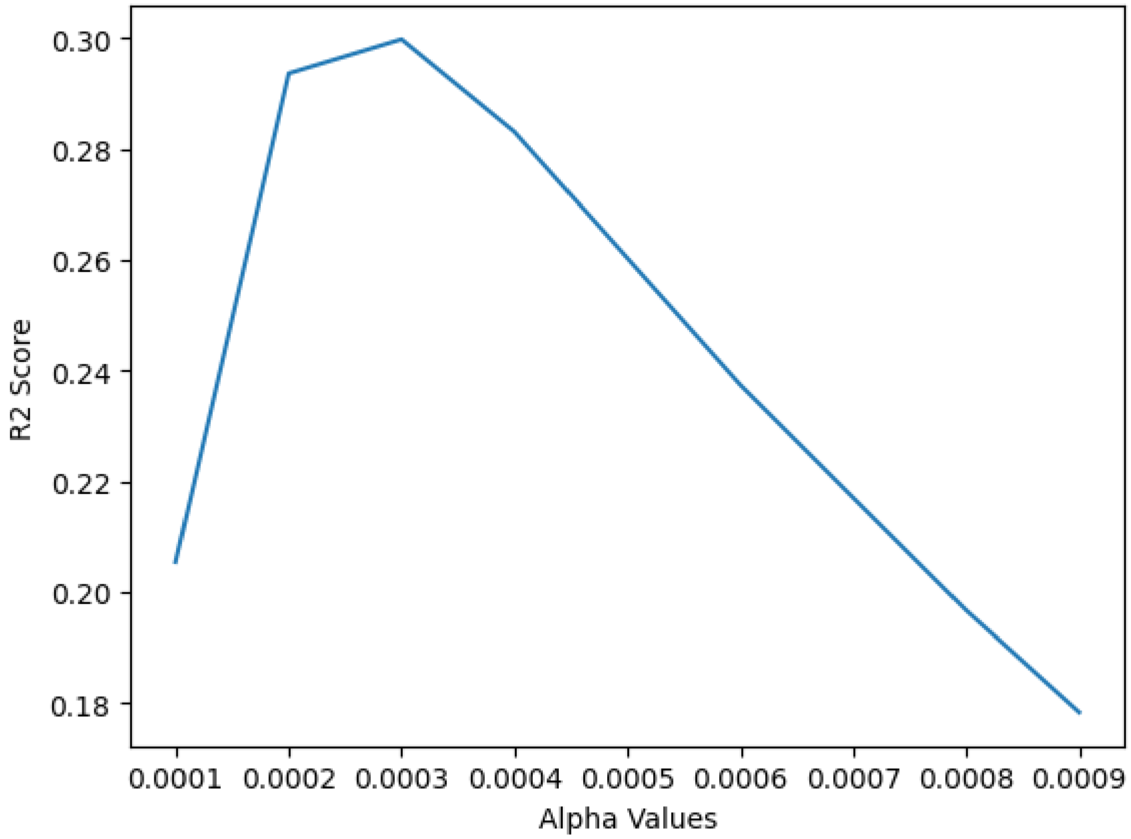
In [20]:
```
max_r2, alpha_value = lasso_only(1,10,1,5000)
print("Peaking at 0.002 so drilling further down")
```



```
Maximum Value of R² is  0.2936526820041159  For alpha =  0.0002  for alpha in
range (0.0002,0.002)
Peaking at 0.002 so drilling further down
```

In [21]:
```
max_r2_lasso, alpha_value_lasso = lasso_only(1,10,1,10000)
```

```
/Users/yamini/Library/Python/3.8/lib/python/site-packages/sklearn/linear_mode
l/_coordinate_descent.py:648: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations, check the scale of the fe
atures or consider increasing regularisation. Duality gap: 2.031e-03, toleranc
e: 1.550e-03
  model = cd_fast.enet_coordinate_descent(
```



```
Maximum Value of R² is  0.2998068457932265  For alpha =  0.0003  for alpha in
range (0.0001,0.001)
```

In [22]:
```python
print("On the Test Data")
alpha_test_lasso = alpha_value_lasso
lasso_regression_model = Lasso(alpha = alpha_test_lasso)
lasso_regression_model.fit(X_train_train, Y_train_train)
Y_predicted = lasso_regression_model.predict(X_test)
r2_test_lasso = r2_score(Y_test, Y_predicted)
print("R2_score on the test Data",r2_test_lasso)
print("For alpha value = ",alpha_test_lasso)
```

```
On the Test Data
R2_score on the test Data 0.206751864458943336
For alpha value =  0.0003
```

> [Provide your explanation here]

ANSWER

For alpha values greater than 0.3 turns all coefficent of features 0 that's why there is no change in ${R^2}$ value after that and it stays constant as alpha values increases, coefficient for features becomes 0 thereby having constant ${R^2}$

> **4. (2 Point)** *Which of the following two methods of Regularization worked better and why?*

ANSWER

For a Test Data,

L1 gave 0.20675186445894336 ${R^2}$ for alpha = 0.0003 Lasso

L2 gave 0.14438812846069535 ${R^2}$ for alpha = 0.78 Ridge

Since there is high collinearity among the features in X as shown in the Pearson coefficient between feature 10 and 11, L1 works better as it moves the coefficient to 0 and removes the feature from being used unlike L2 that shrinks coefficients evenly.

The lasso regression algorithm suggests a simple, sparse models (i.e. models with fewer parameters), which is well-suited for models or data showing high levels of multicollinearity or when we would like to automate certain parts of model selection, like variable selection or parameter elimination using feature engineering.

Lasso Regression algorithm utilises L1 regularization technique It is taken into consideration when there are more number of features because it automatically performs feature selection. In our case no of features are high so feature selection might be a useful technique

On the other hand, Ridge might not be a good regression model in this case if the collinearity in the dataset is very high as then there can be some bias value. Also since ridge can only shrink and cannot equate the coefficent to 0. Therefore Lasso regression ${R^2}$ score is better than Ridge Regression

Part 3: Logistic Regression (5 Points)

1. (1 point) Now let's consider logistic regression. What is the difference betweeen a linear and logistic regression problem? Explain. (You may take the example of the dataset and task used in this assignment to explain how it would look different for a logistic regression problem)

ANSWER

Linear regression is used to predict the continuous dependent variable using a given set of independent variables. Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables like a classification problem.

The output of Logistic Regression problem can be only between the 0 and 1. Linear regression predicts the output for the continuous dependent variable.

In Linear regression, it is required that relationship between dependent variable and independent variable must be linear. In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.

In linear regression, there may be collinearity between the independent variables. In logistic regression, there should not be collinearity between the independent variable.

If we convert the dataset to categorical variable and make X a set of independent variable then we can use logistic regression. Example we can use Feature 10 in X and converting Y to 2 class target that is True and False

Feature Selection in X such that features in X are independent

X : I am using Feature 10 as it is highly correlated to Feature 11 and other columns(Given below) so for maintaining independant variables

input is Feature 10 and output Y is 2 class variable

In [23]:
```
print("X Input TRANSFORMATION")
from scipy.stats import pearsonr

df_dict ={"Feature I":[],"Feature II":[], "corr":[]}

i=10
for j in range(0,12,1):
    if j!=i:
        corr, _ = pearsonr(X_train[:,i], X_train[:,j])
        df_dict["corr"].append(corr)
        df_dict["Feature II"].append(j)
        df_dict["Feature I"].append(i)

df = pd.DataFrame.from_dict(df_dict)
df
```

```
X Input TRANSFORMATION
```

Out[23]:

	Feature I	Feature II	corr
0	10	0	0.019933
1	10	1	0.887930
2	10	2	0.892406
3	10	3	-0.130038
4	10	4	0.781151
5	10	5	0.742974
6	10	6	0.141554
7	10	7	0.596102
8	10	8	0.645276
9	10	9	-0.026702
10	10	11	0.998497

ANSWER

Y : Y is a 85 output variable and as you can see down the maximum and minimum 85 variables vary from -ve to +ve so values>0 is True and values<0 False, Out of 85 values whichever is majority is the class value, Example if out of 85 features no of positive values> no of negative values then it belongs to Class True and if no of positive values< no of negative values then it belongs to Class False

In [24]:
```
print("Y Transformation")
df_dataset.head()
```

```
Y Transformation
```

Out[24]:

	X	Y	X_min	X_max	Y_min	Y_r
0	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.000000	0.000000	0.000000	0.000
1	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.000000	0.000000	0.000000	0.000
2	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	0.000000	0.000000	0.000000	0.000
3	[-0.005298272870862167, 0.004912682004587315, ...	[-0.09276204596036008, -0.08985079131166167, -...	-0.026475	0.020312	-0.228192	0.335
4	[-0.015229694817309507, 0.0005905057848969534,...	[0.012418027849942948, -0.005698385917103518, ...	-0.032046	0.028179	-0.254572	0.280

2. (1 point) Following your answer in the previous question, convert the dataset such that for the same outcome prediction task, a logistic regression is suitable.

In [25]:
```
Y_logistic = Y>0
Y_logistic_single = []

for Y_temp in Y_logistic:
    y_temp = False
    count = 0
    for y in Y_temp:
        if y == True:
            count+=1

    if count>42:
        y_temp = True

    Y_logistic_single.append(y_temp)

Y_logistic_single = np.asarray(Y_logistic_single)
unique, counts = np.unique(Y_logistic_single, return_counts=True)
print("This has comparitively equal no of both classes")
print("Y values", unique, counts)
```

```
This has comparitively equal no of both classes
Y values [False  True] [145 160]
```

In [26]:
```
X_logistic =[]
for i in X[:,10]:
    X_logistic.append([i])
```

3. (1 point) Split this new dataset into training and testing with a 2:1 split. Implement logistic regression.

In [27]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
print("Can use Stratified Split in order to make sure train and test sample has

X_train_logistic, X_test_logistic, Y_train_logistic, Y_test_logistic = train_te
unique, counts = np.unique(Y_train_logistic, return_counts=True)
print("Train Data")
print("Y",unique, counts)

unique, counts = np.unique(Y_test_logistic, return_counts=True)
print("Test Data")
print("X",unique, counts)

logistic_model = LogisticRegression(random_state=0).fit(X_train_logistic, Y_tra
Y_logistic_predicted = logistic_model.predict(X_test_logistic)
```

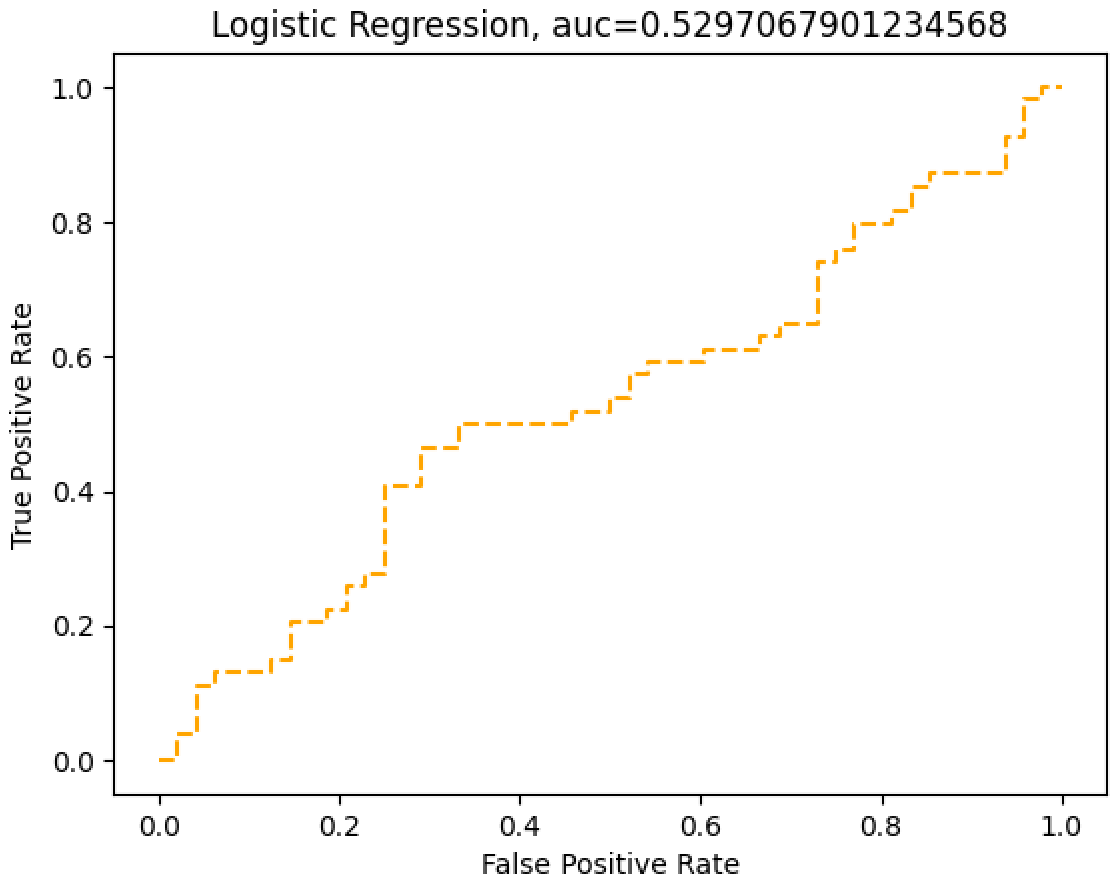
```
Can use Stratified Split in order to make sure train and test sample has both
classes values but random is itself giving fairly distributed train and test s
plit
Train Data
Y [False  True] [ 97 106]
Test Data
X [False  True] [48 54]
```

4. (2 point) Report the performance of the model on the test data using accuracy. Also plot the confusion matrix and ROC-AUC curve. What can you infer from this confusion matrix? Also can you compare the logistic and linear regressions?

In [28]:
```
print("Accuracy of the model is",float(logistic_model.score(X_test_logistic, Y_
```

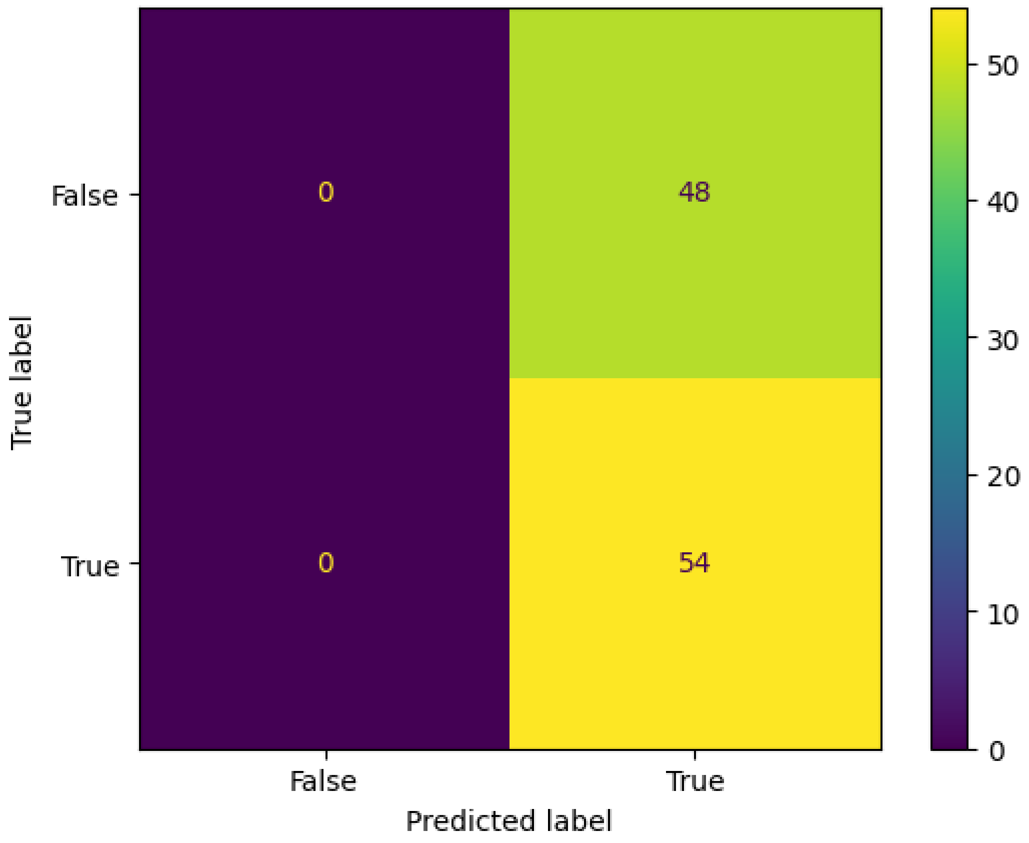
```
Accuracy of the model is 0.5294117647058824
```

In [29]:
```
from sklearn.metrics import plot_confusion_matrix

y_pred_proba = logistic_model.predict_proba(X_test_logistic)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test_logistic,  y_pred_proba)

auc = metrics.roc_auc_score(Y_test_logistic, y_pred_proba)
plt.plot(fpr,tpr,linestyle='--',color='orange')
plt.title(label="Logistic Regression, auc="+str(auc))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()



plot_confusion_matrix(logistic_model, X_test_logistic, Y_test_logistic)
```



```
/Users/yamini/Library/Python/3.8/lib/python/site-packages/sklearn/utils/deprec
ation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Func
tion `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2.
Use one of the class methods: ConfusionMatrixDisplay.from_predictions or Confu
sionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
Out[29]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17ffb60d0>
```



ANSWER

Confusion Matrix:

It is clearly predicting all True and no False values based on Input Test Data, That is why the Accuracy is around 50% - This is like base model that predicts True everytime.

Can calculate accuracy based on confusion matrix and how many Total positive,

```
In [30]: unique, counts = np.unique(Y_logistic_predicted, return_counts=True)
         print("Y predicted values", unique, counts)
         print("Which means the model really dint learn anything its still the base mode
```

```
Y predicted values [ True] [102]
Which means the model really dint learn anything its still the base model as i
t predicts the larger count class
```

ANSWER

Comparing Linear and Logistic Regression Can't really compare as one predicts continuous variable and the other predicts probabilities

Since R_{2} =0 is the base model in Linear regression and Base Accuracy = max count of class that is predicted/ total dataset count in Logistic Regression.

Logistic Regression

1. It looks like logistic regression model did not really learn anything properly as Accuracy is close to 50% which is the percentage of True in the dataset and it does not learn how to predict False based on the input - So highly biased towards one class model - did not learn anything - is as good as constant(True) predicting model.
2. Also if you see the probability scores its all in the range (0.53-0.54) which shows the confusion.

Linear Regression

1. Since R_{2} is also extremely close to 0 it means that its not able to find a proper correlation between the input X and Y which is what logistic regression as well suggests
2. There is multicollinearity between features but not between features and target.

```
In [ ]:
```