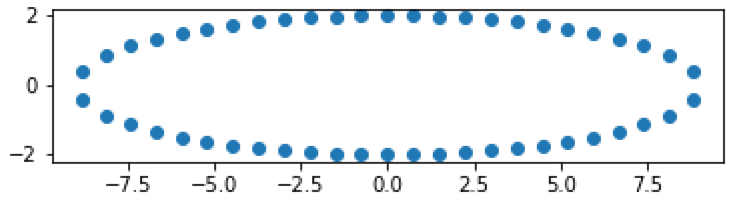
Task-C: Regression outlier effect.

Objective:Visualization best fit linear regression line for different scenarios

In [1]:
```
# you should not import any other packages
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from sklearn.linear_model import SGDRegressor
```

In [2]:
```
import numpy as np
import scipy as sp
import scipy.optimize

def angles_in_ellipse(num,a,b):
    assert(num > 0)
    assert(a < b)
    angles = 2 * np.pi * np.arange(num) / num
    if a != b:
        e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
        tot_size = sp.special.ellipeinc(2.0 * np.pi, e)
        arc_size = tot_size / num
        arcs = np.arange(num) * arc_size
        res = sp.optimize.root(
            lambda x: (sp.special.ellipeinc(x, e) - arcs), angles)
        angles = res.x
    return angles
```

In [3]:
```
a = 2
b = 9
n = 50

phi = angles_in_ellipse(n, a, b)
e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
arcs = sp.special.ellipeinc(phi, e)

fig = plt.figure()
ax = fig.gca()
ax.axes.set_aspect('equal')
ax.scatter(b * np.sin(phi), a * np.cos(phi))
plt.show()
```



In [4]:
```
X= b * np.sin(phi)
Y= a * np.cos(phi)
```

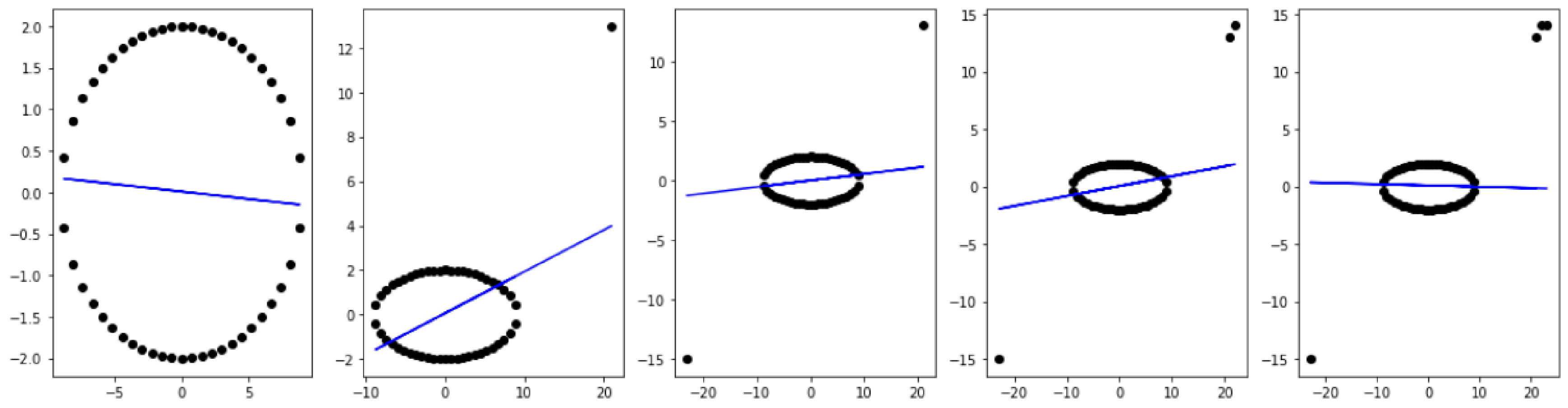
1. As a part of this assignment you will be working the regression problem and how regularization helps to get rid of outliers

2. Use the above created X, Y for this experiment.

3. to do this task you can either implement your own SGDRegression(prefered) excatly similar to "SGD assignment" with mean sequared err or or
you can use the SGDRegression of sklearn, for example "SGDRegressor(alpha=0.001, eta0=0.001, learning_rate='constant',random_state=0)"
*note that you have to use the constant learning rate and learning rate **eta0** initialized.*

4. as a part of this experiment you will train your linear regression on the data (X, Y) with different regularizations alpha=[0.0001, 1, 100] a nd
observe how prediction hyper plan moves with respect to the outliers

5. This the results of one of the experiment we did (title of the plot was not metioned intentionally)



in each iteration we were adding single outlier and observed the movement of the hyper plane.

6. please consider this list of outliers: [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)] in each of tuple the first elemet *is the input feature(X) and the second element is the output(Y)*

7. for each regularizer, you need to add these outliers one at time to data and then train your model again on the updated data.

*8. you should plot a 3*5 grid of subplots, where each row corresponds to results of model with a single regularizer.*

9. Algorithm:

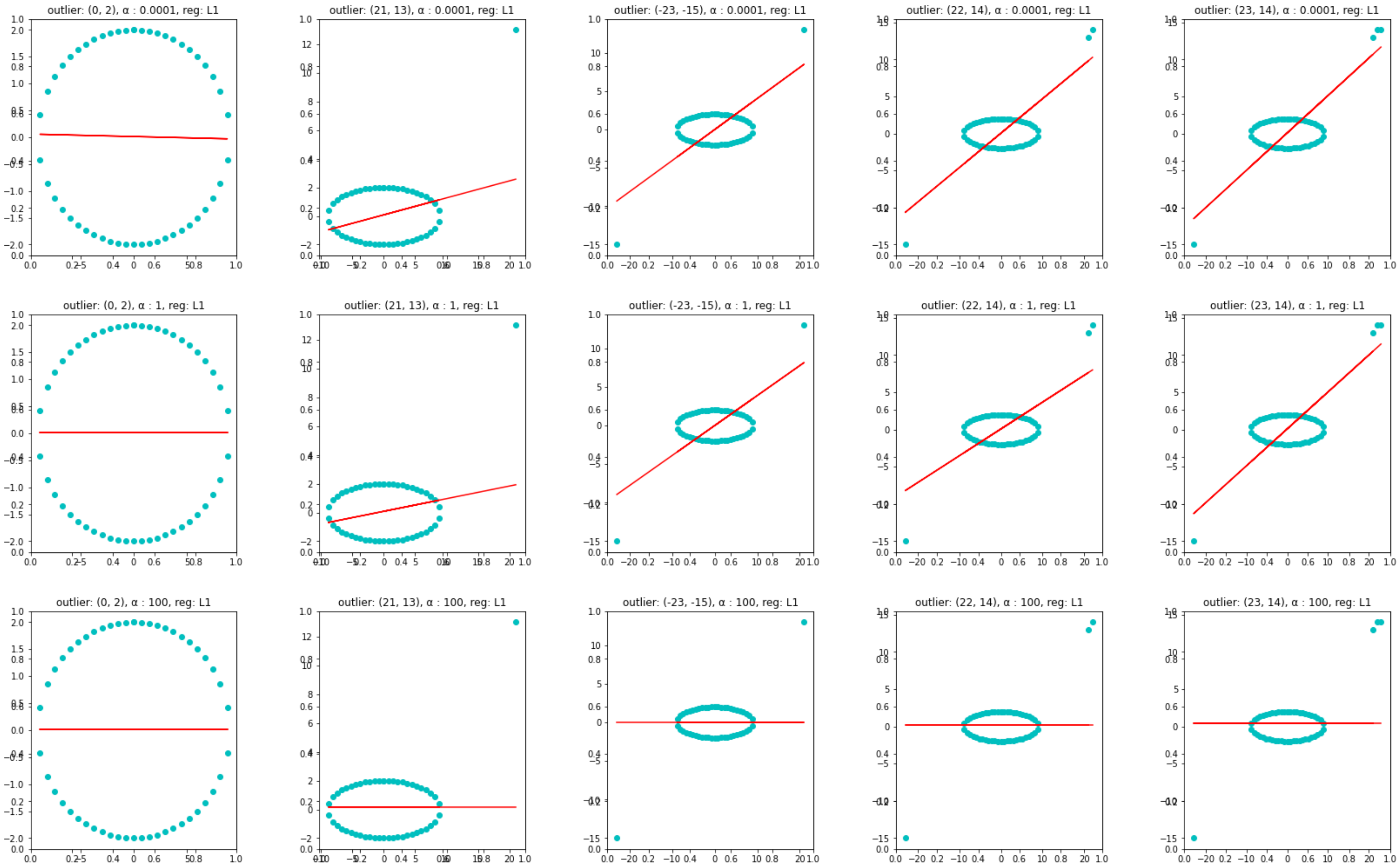
```
for each regularizer:
  for each outlier:
      #add the outlier to the data
      #fit the linear regression to the updated data
      #get the hyper plane
      #plot the hyperplane along with the data points
```

10. MAKE SURE YOU WRITE THE DETAILED OBSERVATIONS, PLEASE CHECK THE LOSS FUNCTION IN THE SKLEARN DOCUMENTA TION (please do search for it).

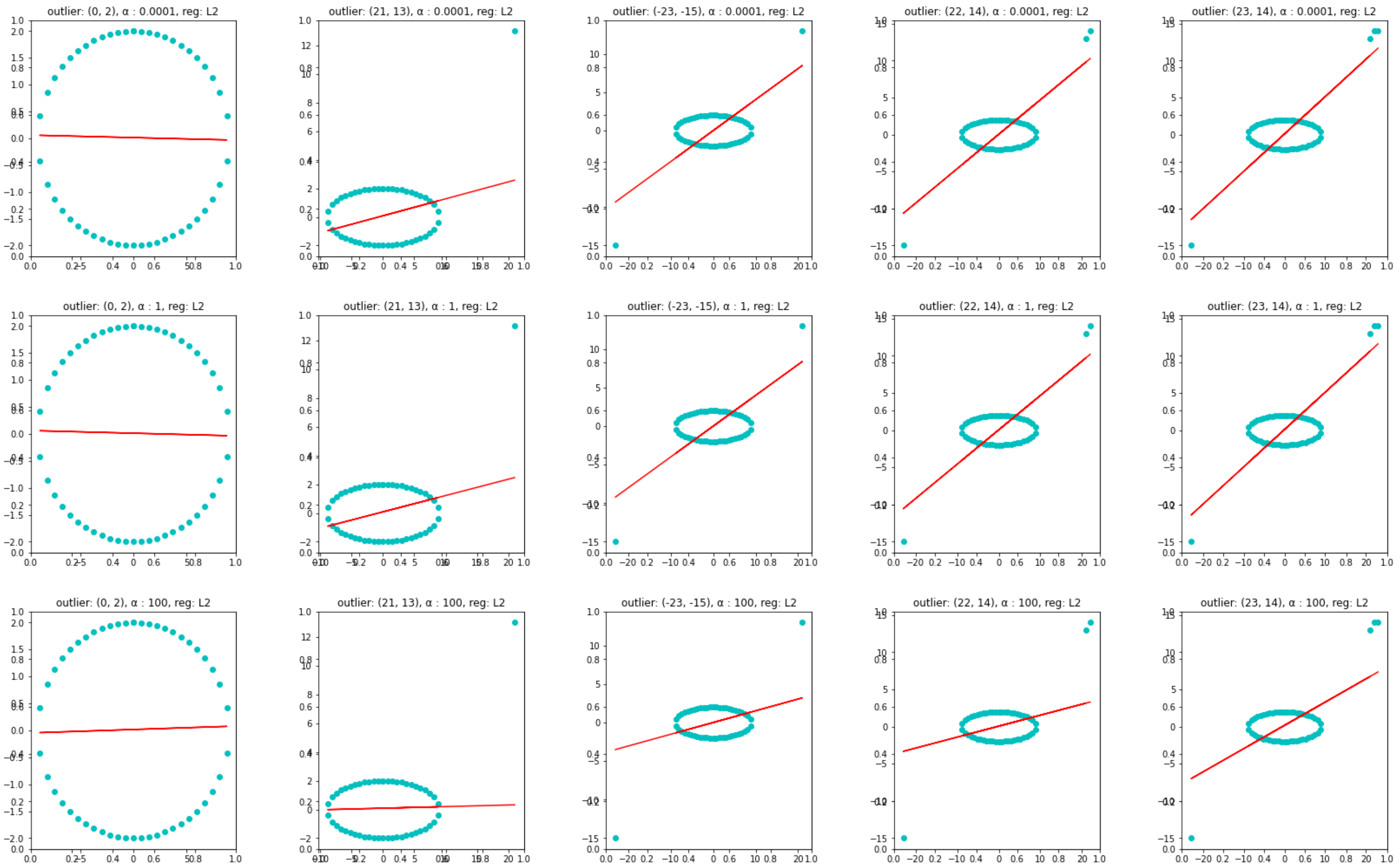
In [5]:
```
def getting_these_dudes():
    a = 2
    b = 9
    n = 50

    phi = angles_in_ellipse(n, a, b)
    e = (1.0 - a ** 2.0 / b ** 2.0) ** 0.5
    arcs = sp.special.ellipeinc(phi, e)
    X= b * np.sin(phi)
    y= a * np.cos(phi)

    return X, y
```

In [11]:
```
def tryingmybest(outliers, alpha, penalty):
    num=1
    fig, ax = plt.subplots(1, 5, figsize=(15,15))
    fig.subplots_adjust(hspace=0.4, wspace=0.4)

    X, y = getting_these_dudes()

    for outlier in outliers:

        X = np.append(X, outlier[0])
        y = np.append(y, outlier[1])
        X = X.reshape(-1, 1)


        model2 = SGDRegressor(alpha=alpha, eta0=0.001, penalty=penalty, loss='squared_loss', learning_rate='c
onstant', random_state=0)
        model2.fit(X, y)

        ax = fig.add_subplot(1, 5, num)
        fig.set_size_inches(28,5)


        plt.scatter(X, y,color='c')
        plt.plot(X, model2.predict(X),color='red')
        plt.title('outlier: {0}, α : {1}, reg: {2}'.format(str(outlier), str(alpha), str(penalty).upper()))

        num +=1
```

In [12]:
```
outliers = [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)]

tryingmybest(outliers, alpha=0.0001, penalty='l1')
tryingmybest(outliers, alpha=1, penalty='l1')
tryingmybest(outliers, alpha=100, penalty='l1')
```



In [13]:
```
tryingmybest(outliers, alpha=0.0001, penalty='l2')
tryingmybest(outliers, alpha=1, penalty='l2')
tryingmybest(outliers, alpha=100, penalty='l2')
```



Observation:

L1:

A L1 loss function considers the absolute differences between the true value and the predicted value. We can clearly see that when using L1 with outlier points, it is not affected as much as L2. The reason, as we are only taking absolute differences, the squared error is minimal

L2:

With a L2 loss function, we are consdering the squared differences between the true value and the predicted value. That maximizes the errors as sqaured values are large enough to disorient our hyperplane. Moreover, as research is proof, L1 norm performs much better with outliers