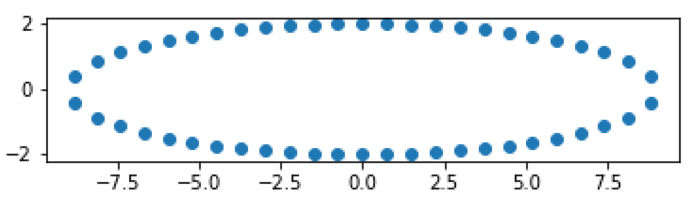
Task-C: Regression outlier effect.

Objective:Visualization best fit linear regression line for different scenarios

In [1]:
```python
# 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]:
```python
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]:
```python
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()  # get current axes
ax.axes.set_aspect('equal')
ax.scatter(b * np.sin(phi), a * np.cos(phi))
plt.show()
```



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

In [5]:
```python
print(X)
print(Y)
```

```
[ 0.00000000e+00  7.44742410e-01  1.48935470e+00  2.23369584e+00
  2.97760060e+00  3.72086049e+00  4.46319267e+00  5.20418351e+00
  5.94317435e+00  6.67899598e+00  7.40922283e+00  8.12729576e+00
  8.80003723e+00  8.80003723e+00  8.12729576e+00  7.40922283e+00
  6.67899598e+00  5.94317435e+00  5.20418351e+00  4.46319267e+00
  3.72086049e+00  2.97760060e+00  2.23369584e+00  1.48935470e+00
  7.44742410e-01  1.10218212e-15 -7.44742410e-01 -1.48935470e+00
 -2.23369584e+00 -2.97760060e+00 -3.72086049e+00 -4.46319267e+00
 -5.20418351e+00 -5.94317435e+00 -6.67899598e+00 -7.40922283e+00
 -8.12729576e+00 -8.80003723e+00 -8.80003723e+00 -8.12729576e+00
 -7.40922283e+00 -6.67899598e+00 -5.94317435e+00 -5.20418351e+00
 -4.46319267e+00 -3.72086049e+00 -2.97760060e+00 -2.23369584e+00
 -1.48935470e+00 -7.44742410e-01]
[ 2.          1.99314081  1.972425    1.93742355  1.88737056  1.82107277
  1.73674751  1.63172973  1.50191119  1.34055475  1.13536674  0.85914295
  0.41924946 -0.41924946 -0.85914295 -1.13536674 -1.34055475 -1.50191119
 -1.63172973 -1.73674751 -1.82107277 -1.88737056 -1.93742355 -1.972425
 -1.99314081 -2.         -1.99314081 -1.972425   -1.93742355 -1.88737056
 -1.82107277 -1.73674751 -1.63172973 -1.50191119 -1.34055475 -1.13536674
 -0.85914295 -0.41924946  0.41924946  0.85914295  1.13536674  1.34055475
  1.50191119  1.63172973  1.73674751  1.82107277  1.88737056  1.93742355
  1.972425    1.99314081]
```

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 error 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] and 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

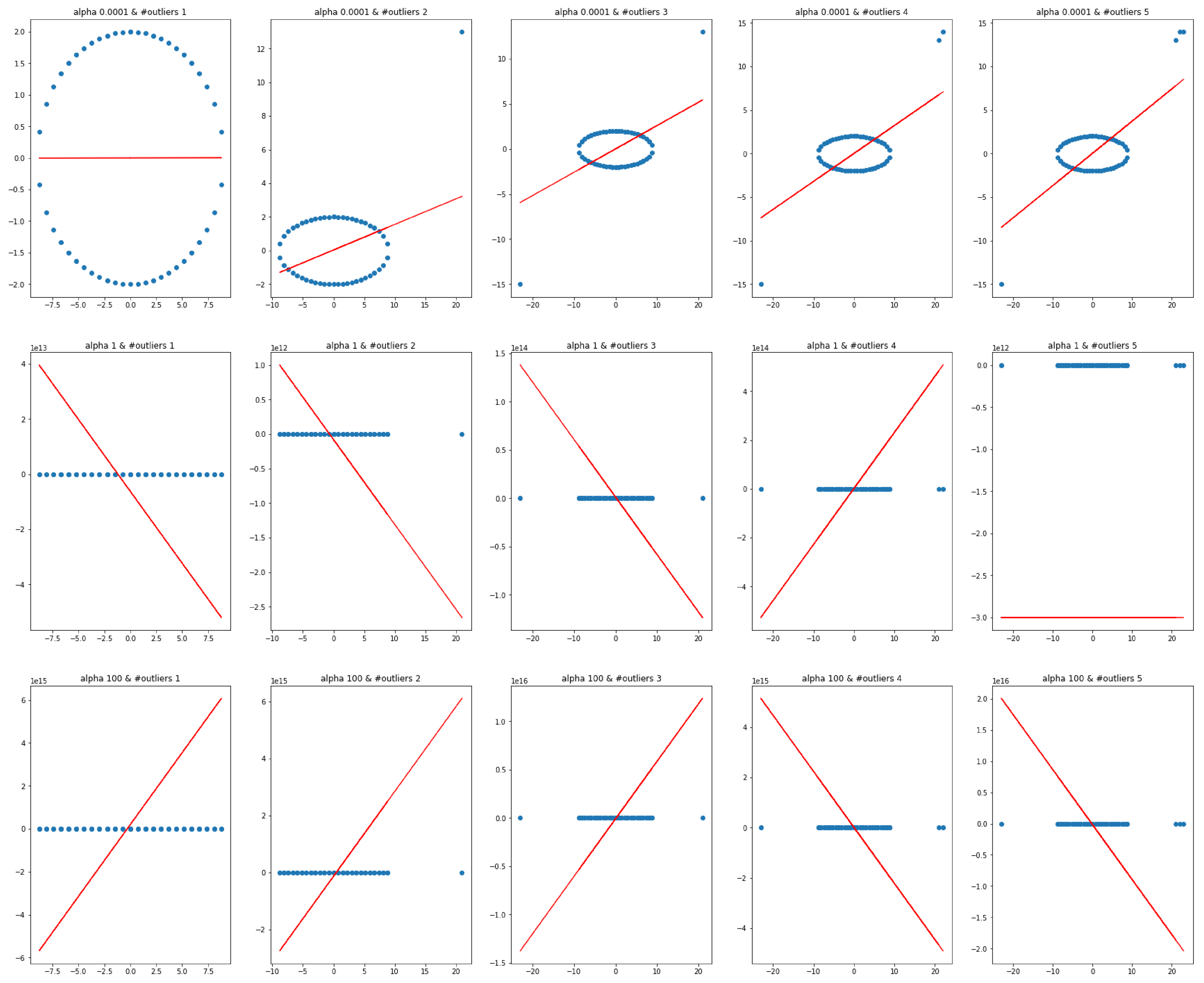
10. MAKE SURE YOU WRITE THE DETAILED OBSERVATIONS, PLEASE CHECK THE LOSS FUNCTION IN THE SKLEARN DOCUMENTATION (please do search for it).

GOAL :- Scatter PLOT | Grid PLOT | Regression Problem SIMPLE

In [5]:
```python
X.dtype
```

Out[5]:
```
dtype('float64')
```

In [20]:
```python
plt.figure(figsize=(30, 25))

regularizers = [0.0001, 1, 100]
outliers = [(0,2),(21, 13), (-23, -15), (22,14), (23, 14)]

# r -> rows, c -> columns
r, c = len(regularizers), len(outliers)


for i, alpha in enumerate(regularizers, start=1):
    # {alpha} is the Regularization Constant (ie lma in our case symbols)
    #! NOTE :- {alpha} may signifies the step-size during learning phase

    skip = c * (i-1)  # total graph already plotted
    newX = [*X]       # mutable X-vector
    newY = [*Y]       # mutable Y-vector

    # NOTE: In case regularization param can help to get the learning rate each time when
    #       learning rate is not constant
    #       But Regularization Param is not always Learning Rate, in equality
    learningRate = alpha

    for j, outlier in enumerate(outliers, start=0):
        newX.append(outlier[0])  # add outlier for current {alpha}
        newY.append(outlier[1])  # add outlier for current {alpha}
        mX = np.reshape(newX, (-1,1))  # need ndArray for sklearn's `fit()`

        # get region to plot current graph
        plt.subplot(r, c, skip + j + 1)
        plt.scatter(newX, newY)  # scatter plot
        # QUE ? Does alpha & eta0 needs to kept same all the time ?
        reg = SGDRegressor(alpha=alpha, eta0=learningRate, learning_rate='constant', random_state=0)
        reg.fit(mX, newY)
        plt.plot(mX, reg.predict(mX).ravel(), c='red')  # draw line
        plt.title(f'alpha {alpha} & #outliers {j+1}')
```



Observations

Learning Rate (ie Learning Step of Small Size) Helps to get Convergence better As Learnig Rate increases (ie Step Size increases) there can be speedup observed in Connvergence & due to which it may also lead to Oscillation around Converge Point. Thus it may figure out the best line in small time but may not be best line. (in case of large steps ie large alpha)

- When alpha is small it gets less affected by the outliers compare to larger alpha
- Also when alpha is large its not seems to be a good fit to line (as it may be due to oscillation effect occured due to large Steps whilst doing Gradient Descent)

In [ ]: