Task-C: Regression outlier effect.

Objective:Visualization best fit linear regression line for different scenarios

In [1]:

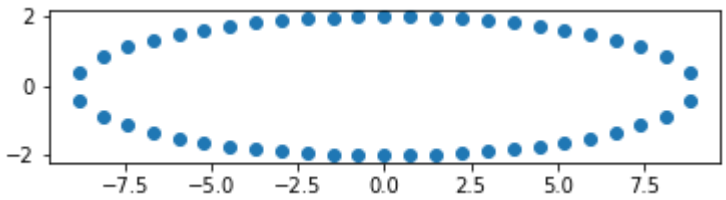
```
# you should not import any other packages
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from sklearn.linear_model import SGDRegressor
```

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)
```

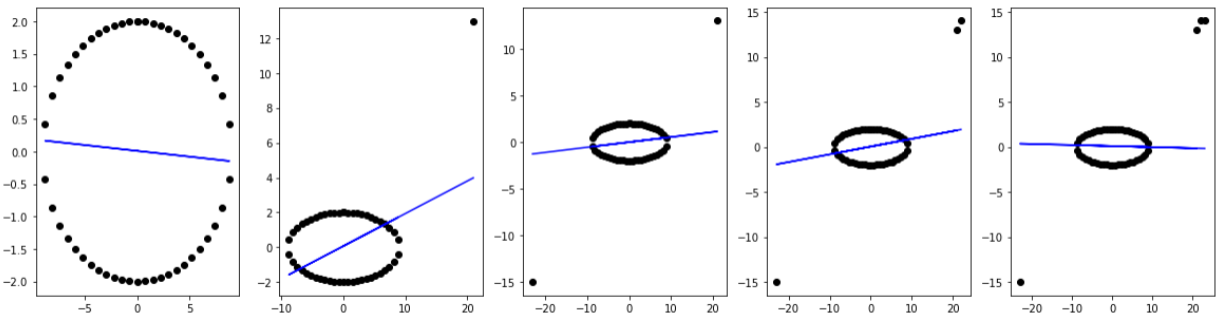
1. As a part of this assignment you will be working the regression problem and how regularization helps to get rid of outliers

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 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 ***etao*** *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:

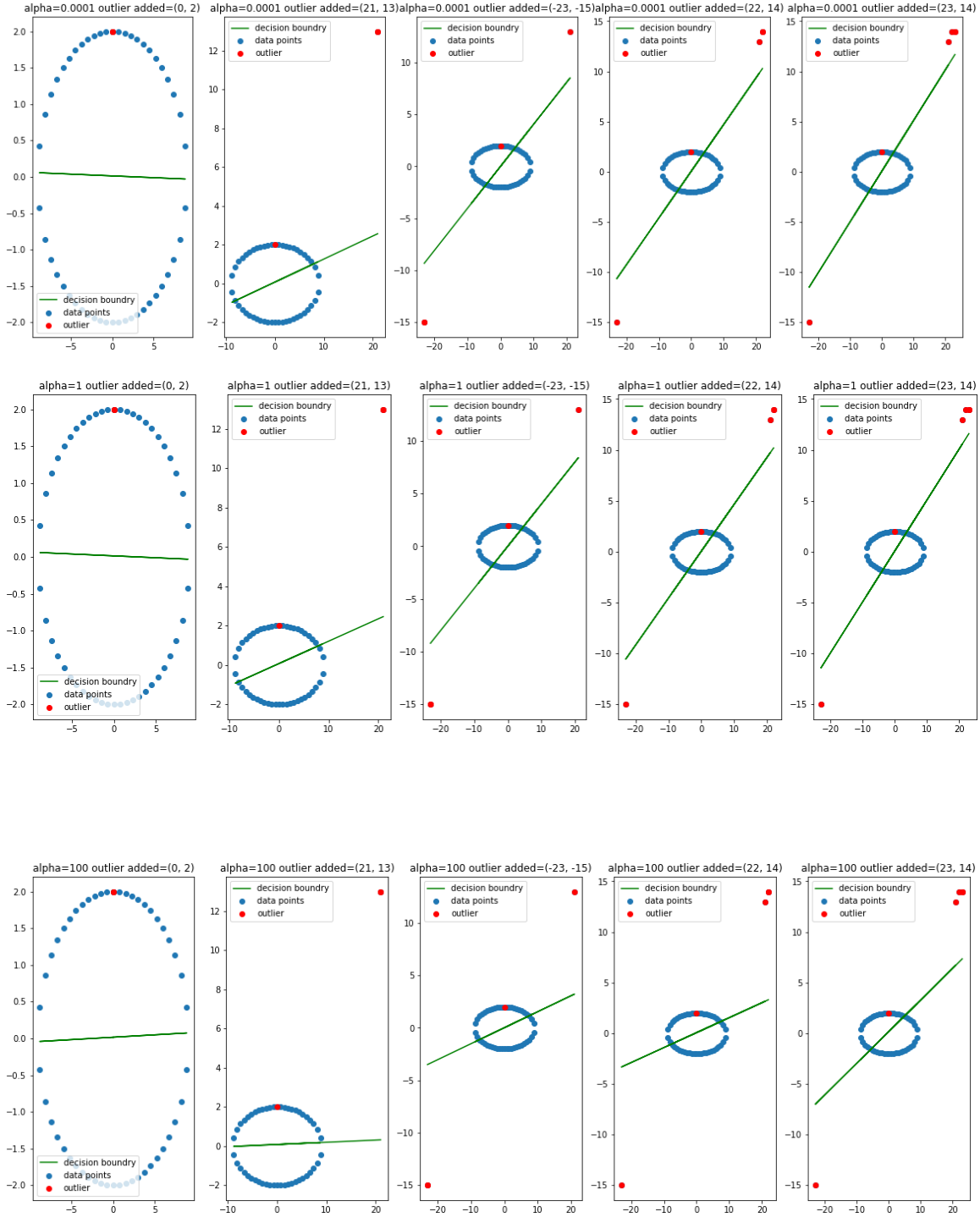
for each regularizer:
for each outlier:
#add the outlier to the data
#fit the linear regression to the updated data
#get the hyper plane
#plot the hyperplane along with the data points

10. MAKE SURE YOU WRITE THE DETAILED OBSERVATIONS, PLEASE CHECK THE LOSS FUN

CTION IN THE SKLEARN DOCUMENTATION
(please do search for it).

In [6]:

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

for a in alpha:
  fig = plt.figure(figsize = (20,24))
  count= 0
  x_temp = X
  y_temp = Y

  for outlier in outliers:
    plt.subplot(3, 5, count+1)
    count += 1
    x_temp = np.append(x_temp, outlier[0]).reshape(-1, 1)
    y_temp = np.append(y_temp, outlier[1]).reshape(-1, 1)
    model = SGDRegressor(alpha=a, eta0=0.001, learning_rate='constant',random_st
ate=0)
    model.fit(x_temp,y_temp)

    predictions = model.predict(x_temp.reshape(-1,1))
    plt.plot(x_temp, predictions, color="green", label = "decision boundry")


    plt.scatter(x_temp,y_temp,label="data points")
    plt.title('alpha='+str(a)+' outlier added='+str(outlier))

    # just of highlighting outliers in red color
    for i in range(0, count):
      plt.scatter(outliers[i][0], outliers[i][1], color="red")

    plt.scatter(outliers[i][0], outliers[i][1], color="red",label="outlier")
    plt.legend()

  plt.show()
```



In [7]:

```
model
```

Out[7]:

```
SGDRegressor(alpha=100, average=False, early_stopping=False, epsilon
=0.1,
             eta0=0.001, fit_intercept=True, l1_ratio=0.15,
             learning_rate='constant', loss='squared_loss', max_iter
=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_
state=0,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbo
se=0,
             warm_start=False)
```

SGD regressor basically implements a plain SGD learning routine supporting various loss functions and penalties to fit linear regression models. Scikit-learn provides SGDRegressor module to implement SGD regression.

Outliers and loss functions are intertwined. Whatever we do with outliers has a direct mapping to the loss function we use.

Here we have squared loss function which is affected by outliers a lot, and hence we use Regularization. When we increase regularization paramater we are giving more penalties to outliers.

Aplha is constant that multiplies the regularization term. The higher the value, the stronger the regularization.

Least-Squares Loss L(yi, f(xi)) = 1/2 (yi - f(xi))^2

More the outliers -> more the Loss

With outlier (0,2), decision boundry not changing much on increasing Regularization parameter as well becasue the point is not far away from the actual data point.

On adding 1 more outlier (21, 13) which is far away from the datapoint, decision boundry now bend little bit because of outlier.

On adding 1 more outlier (-23, -15) which is opposite side of the previously added outlier, impact the devision boundry direction a bit.

On adding 2 more outlier one by one (22,14), (23, 14) impacts the decision boundry a bit and withincrease in regularization parameter its not changing much after that.