Worksheet 21

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

Topics

- Logistic Regression
- Gradient Descent

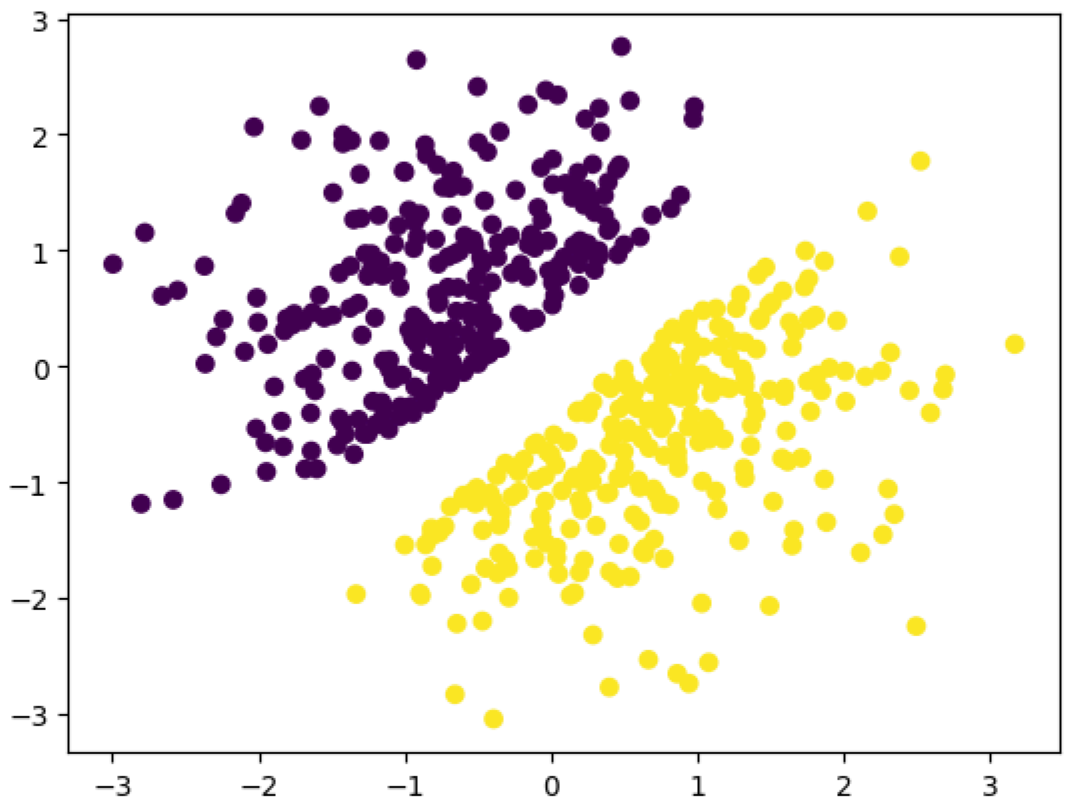
Logistic Regression

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

centers = [[0, 0]]
t, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=1, ra

# LINE
def generate_line_data():
    # create some space between the classes
    X = np.array(list(filter(lambda x : x[0] - x[1] < -.5 or x[0] - x[1] > .
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y

# CIRCLE
def generate_circle_data(t):
    # create some space between the classes
    X = np.array(list(filter(lambda x : (x[0] - centers[0][0])**2 + (x[1] - 
    Y = np.array([1 if (x[0] - centers[0][0])**2 + (x[1] - centers[0][1])**2
    return X, Y

# XOR
def generate_xor_data():
    X = np.array([
        [0,0],
        [0,1],
        [1,0],
        [1,1]])
    Y = np.array([x[0]^x[1] for x in X])
    return X, Y
```

a) Using the above code, generate and plot data that is linearly separable.

In [ ]:
```
X, Y = generate_line_data()
plt.scatter(X[:, 0], X[:, 1], c = Y)
```

```
plt.show()
```



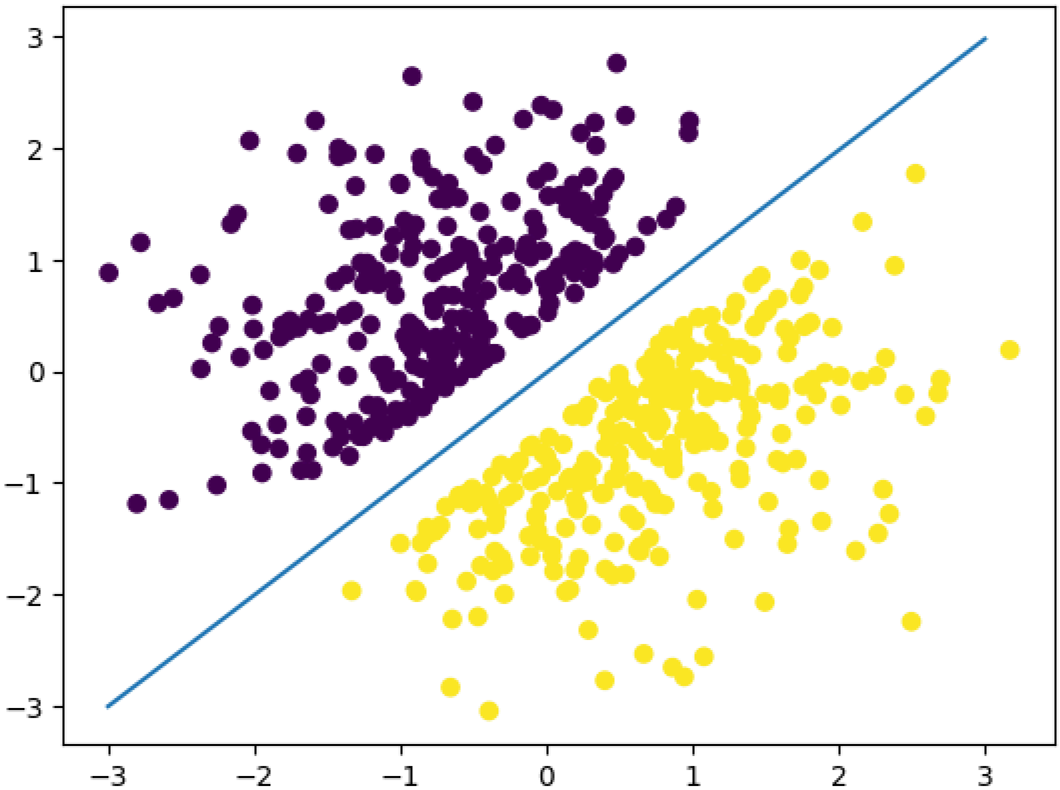
b) Fit a logistic regression model to the data a print out the coefficients.

In [ ]:
```
model = LogisticRegression().fit(X, Y)
print(f"The coefficient is {model.coef_}")
print(f"The intercept is {model.intercept_}")
```

```
The coefficient is [[ 4.11337993 -4.10105513]]
The intercept is [0.05839469]
```

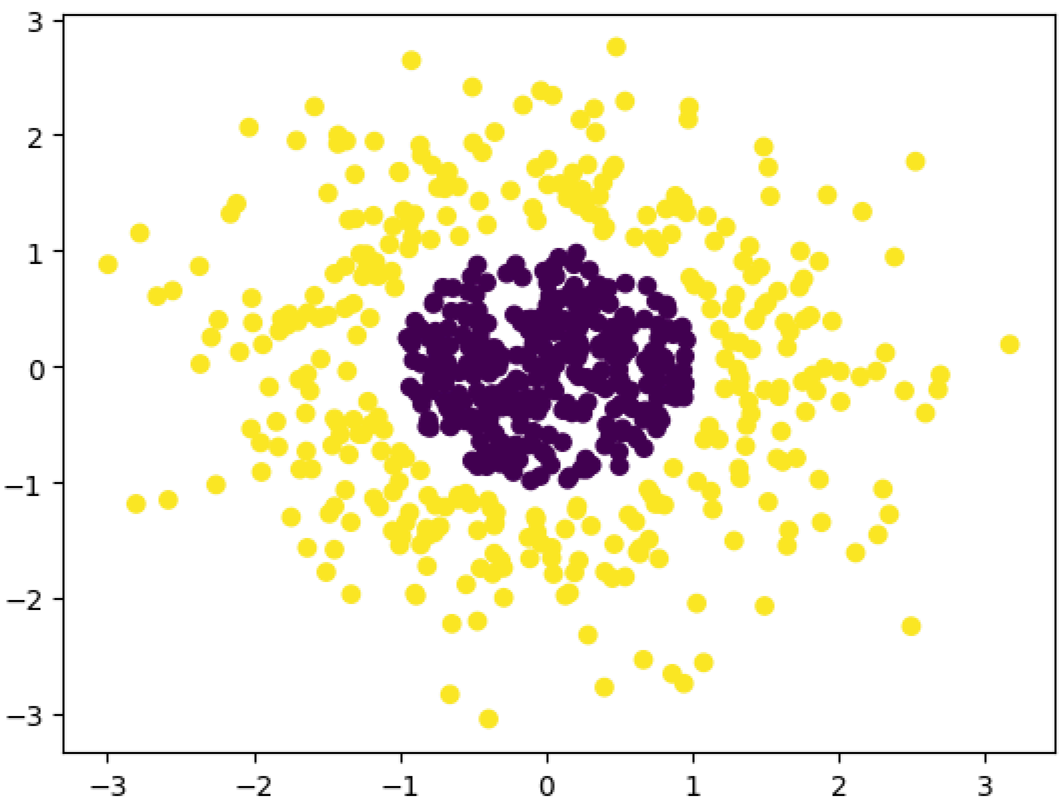
c) Using the coefficients, plot the line through the scatter plot you created in a). (Note: you need to do some math to get the line in the right form)

In [ ]:
```
modelCoef = model.coef_
modelIntercept = model.intercept_

spaces = np.linspace(-3, 3)
scatterLine = - modelIntercept / modelCoef[0][0] - modelCoef[0][1] * spaces
plt.scatter(X[:, 0], X[:, 1], c = Y)
plt.plot(spaces, scatterLine)
plt.show()
```



d) Using the above code, generate and plot the CIRCLE data.

In [ ]:
```
X, Y = generate_circle_data(t)
plt.scatter(X[:, 0], X[:, 1], c = Y)
plt.show()
```



e) Notice that the equation of an ellipse is of the form $$ax^2 + by^2 = c$$

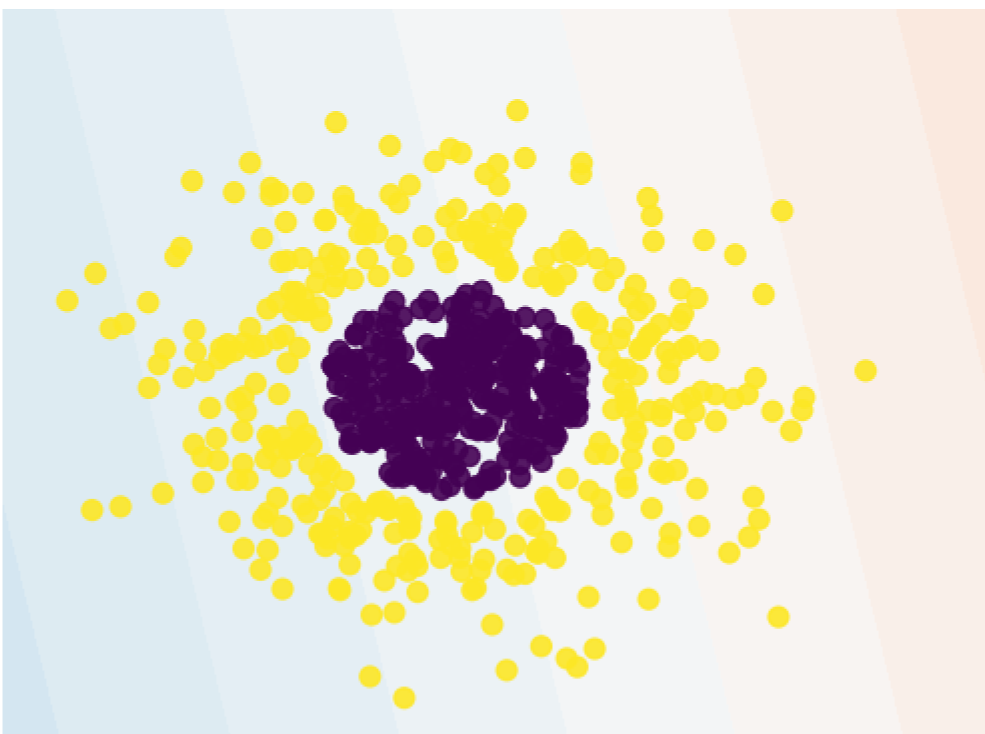
Fit a logistic regression model to an appropriate transformation of X.

In [ ]:
```
model = LogisticRegression().fit(X, Y)
print(f"The coefficient is {model.coef_}")
print(f"The intercept is {model.intercept_}")
```

```
The coefficient is [[-0.06183417 -0.01164363]]
The intercept is [0.09334628]
```

f) Plot the decision boundary using the code below.

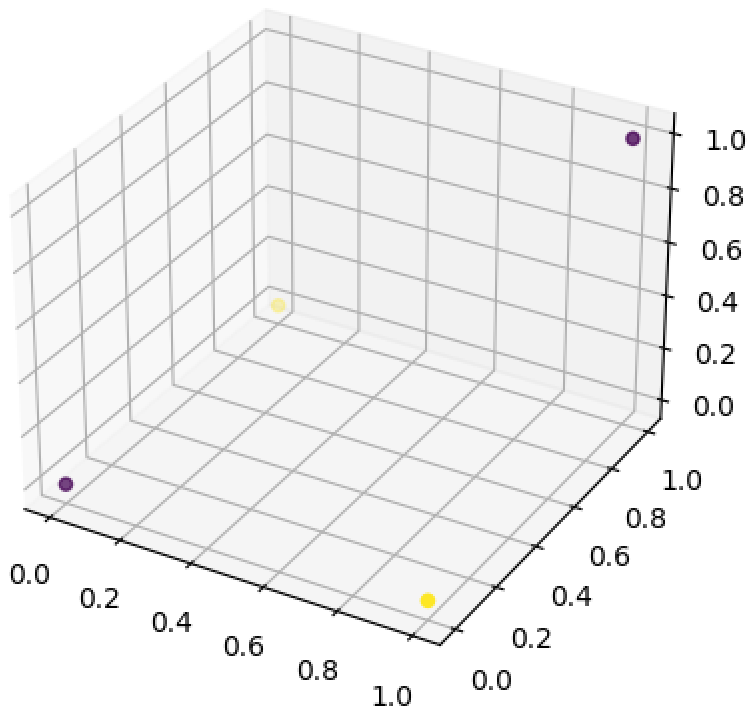
In [ ]:
```
# create a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
meshData = np.c_[xx.ravel(), yy.ravel()]

fig, ax = plt.subplots()
A = model.predict_proba(meshData)[:, 1].reshape(xx.shape)
Z = model.predict(meshData).reshape(xx.shape)
ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
ax.axis('off')

# Plot also the training points
ax.scatter(X[:, 0], X[:, 1], c=Y, s=50, alpha=0.9)
plt.show()
```



g) Plot the XOR data. In this 2D space, the data is not linearly separable, but by introducing a new feature $$x_3 = x_1 * x_2$$

(called an interaction term) we should be able to find a hyperplane that separates the data in 3D. Plot this new dataset in 3D.

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D

X, Y = generate_xor_data()
ax = plt.axes(projection='3d')
ax.scatter3D(X[: , 0], X[: , 1], X[: , 0]* X[: , 1], c=Y)
plt.show()
```



h) Apply a logistic regression model using the interaction term. Plot the decision boundary.

In [ ]:
```
poly = PolynomialFeatures(interaction_only=True)
lr = LogisticRegression(verbose=0)
model = make_pipeline(poly, lr).fit(X, Y)

# create a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
meshData = np.c_[xx.ravel(), yy.ravel()]

fig, ax = plt.subplots()
```

```
A = model.predict_proba(meshData)[:, 1].reshape(xx.shape)
Z = model.predict(meshData).reshape(xx.shape)
ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
ax.axis('off')

# Plot also the training points
ax.scatter(X[:, 0], X[:, 1], color=Y, s=50, alpha=0.9)
plt.show()
```

In [ ]:
```
!pip install ipympl
```

```
Collecting ipympl
  Downloading ipympl-0.9.4-py3-none-any.whl (516 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 516.3/516.3 kB 2.6 MB/s eta 0:0
0:00
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/
dist-packages (from ipympl) (0.2.0)
Requirement already satisfied: ipython<9 in /usr/local/lib/python3.10/dist-pa
ckages (from ipympl) (7.34.0)
Requirement already satisfied: ipywidgets<9,>=7.6.0 in /usr/local/lib/python
3.10/dist-packages (from ipympl) (7.7.1)
Requirement already satisfied: matplotlib<4,>=3.4.0 in /usr/local/lib/python
3.10/dist-packages (from ipympl) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packag
es (from ipympl) (1.25.2)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packa
ges (from ipympl) (9.4.0)
Requirement already satisfied: traitlets<6 in /usr/local/lib/python3.10/dist-
packages (from ipympl) (5.7.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/
dist-packages (from ipython<9->ipympl) (67.7.2)
Collecting jedi>=0.16 (from ipython<9->ipympl)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 9.6 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-pa
ckages (from ipython<9->ipympl) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-
packages (from ipython<9->ipympl) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 i
n /usr/local/lib/python3.10/dist-packages (from ipython<9->ipympl) (3.0.43)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-pac
kages (from ipython<9->ipympl) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-pac
kages (from ipython<9->ipympl) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.1
0/dist-packages (from ipython<9->ipympl) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-
packages (from ipython<9->ipympl) (4.9.0)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.10/
dist-packages (from ipywidgets<9,>=7.6.0->ipympl) (5.5.6)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/py
thon3.10/dist-packages (from ipywidgets<9,>=7.6.0->ipympl) (3.6.6)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/py
thon3.10/dist-packages (from ipywidgets<9,>=7.6.0->ipympl) (3.0.10)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
dist-packages (from matplotlib<4,>=3.4.0->ipympl) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
-packages (from matplotlib<4,>=3.4.0->ipympl) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib<4,>=3.4.0->ipympl) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib<4,>=3.4.0->ipympl) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib<4,>=3.4.0->ipympl) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
dist-packages (from matplotlib<4,>=3.4.0->ipympl) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python
```

```
3.10/dist-packages (from matplotlib<4,>=3.4.0->ipympl) (2.8.2)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/di
st-packages (from ipykernel>=4.5.1->ipywidgets<9,>=7.6.0->ipympl) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist
-packages (from ipykernel>=4.5.1->ipywidgets<9,>=7.6.0->ipympl) (6.3.3)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.
10/dist-packages (from jedi>=0.16->ipython<9->ipympl) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/d
ist-packages (from pexpect>4.3->ipython<9->ipympl) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-pack
ages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython<9->ipympl)
(0.2.13)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
kages (from python-dateutil>=2.7->matplotlib<4,>=3.4.0->ipympl) (1.16.0)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.10/d
ist-packages (from widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl)
(6.5.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packa
ges (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->i
pympl) (3.1.3)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dis
t-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=
7.6.0->ipympl) (23.2.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-
packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.
6.0->ipympl) (23.1.0)
Requirement already satisfied: jupyter-core>=4.6.1 in /usr/local/lib/python3.
10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets
<9,>=7.6.0->ipympl) (5.7.2)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-pac
kages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0-
>ipympl) (5.10.4)
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.10/dist
-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.
6.0->ipympl) (6.5.4)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.1
0/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<
9,>=7.6.0->ipympl) (1.6.0)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.1
0/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<
9,>=7.6.0->ipympl) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/
dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,
>=7.6.0->ipympl) (0.18.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.1
0/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<
9,>=7.6.0->ipympl) (0.20.0)
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/
dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,
>=7.6.0->ipympl) (1.0.0)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.1
0/dist-packages (from jupyter-core>=4.6.1->notebook>=4.4.1->widgetsnbextensio
n~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (4.2.0)
Requirement already satisfied: jupyter-server>=1.8 in /usr/local/lib/python3.
10/dist-packages (from nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~
=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.24.0)
```

```
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python
3.10/dist-packages (from nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextensio
n~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (0.2.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-package
s (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<
9,>=7.6.0->ipympl) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/di
st-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->i
pywidgets<9,>=7.6.0->ipympl) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidget
s<9,>=7.6.0->ipympl) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-p
ackages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywi
dgets<9,>=7.6.0->ipympl) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.1
0/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.
0->ipywidgets<9,>=7.6.0->ipympl) (0.4)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.
10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.
6.0->ipywidgets<9,>=7.6.0->ipympl) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/d
ist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->
ipywidgets<9,>=7.6.0->ipympl) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.1
0/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.
0->ipywidgets<9,>=7.6.0->ipympl) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/d
ist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->
ipywidgets<9,>=7.6.0->ipympl) (0.10.0)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python
3.10/dist-packages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=
3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.5.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-pac
kages (from nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidg
ets<9,>=7.6.0->ipympl) (1.2.1)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python
3.10/dist-packages (from nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0
->ipywidgets<9,>=7.6.0->ipympl) (2.19.1)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/d
ist-packages (from nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipyw
idgets<9,>=7.6.0->ipympl) (4.19.2)
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python
3.10/dist-packages (from argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.
6.0->ipywidgets<9,>=7.6.0->ipympl) (21.2.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dis
t-packages (from jsonschema>=2.6->nbformat->notebook>=4.4.1->widgetsnbextensi
on~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/l
ocal/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->notebook>=
4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.
10/dist-packages (from jsonschema>=2.6->nbformat->notebook>=4.4.1->widgetsnbe
xtension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (0.34.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/di
st-packages (from jsonschema>=2.6->nbformat->notebook>=4.4.1->widgetsnbextens
```

```
ion~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (0.18.0)
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/d
ist-packages (from jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->wi
dgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (3.7.1)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/
dist-packages (from jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->w
idgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.7.0)
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-
packages (from argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbe
xtension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.16.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dis
t-packages (from beautifulsoup4->nbconvert>=5->notebook>=4.4.1->widgetsnbexte
nsion~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (2.5)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist
-packages (from bleach->nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.
6.0->ipywidgets<9,>=7.6.0->ipympl) (0.5.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-pa
ckages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook
>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (3.6)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist
-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->noteb
ook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/di
st-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->not
ebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (1.2.
0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-pa
ckages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets<9,>=7.6.0->ipympl) (2.22)
Installing collected packages: jedi, ipympl
Successfully installed ipympl-0.9.4 jedi-0.19.1
```

In [ ]:

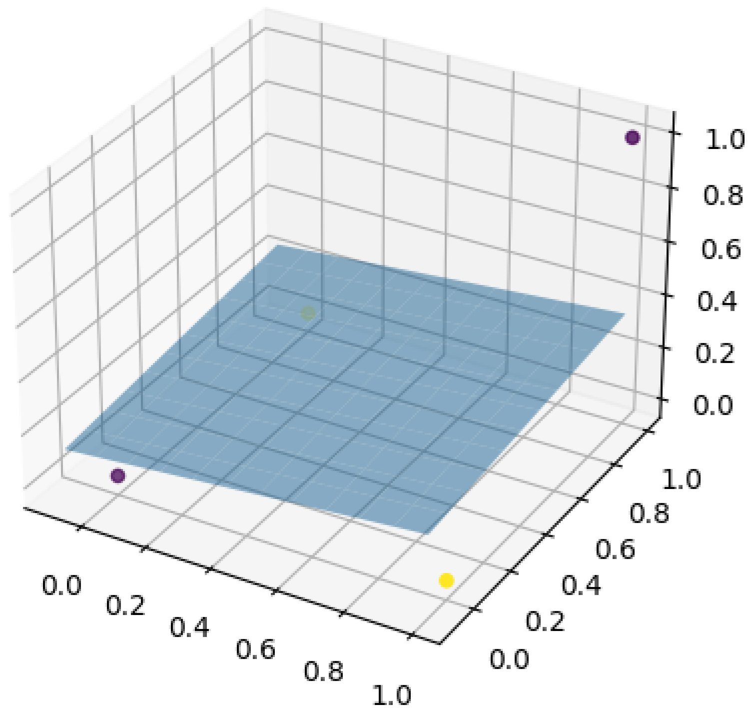
```
from google.colab import output
output.enable_custom_widget_manager()
```

In [ ]:

```
for i in range(20000):
    for solver in ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sa
        X_transform = PolynomialFeatures(interaction_only=True, include_bias
        model = LogisticRegression(verbose=0, solver=solver, random_state=i,
        model.fit(X_transform, Y)
        #print(model.score(X_transform, Y))
        if model.score(X_transform, Y) > .75:
            #print("random state = ", i)
            #print("solver = ", solver)
            break

print(model.coef_)
print(model.intercept_)

xx, yy = np.meshgrid([x / 10 for x in range(-1, 11)], [x / 10 for x in range
z = - model.intercept_ / model.coef_[0][2] - model.coef_[0][0] * xx / model.

ax = plt.axes(projection='3d')
ax.scatter3D(X[: , 0], X[: , 1], X[: , 0]* X[: , 1], c=Y)
```

```
ax.plot_surface(xx, yy, z, alpha=0.5)
plt.show()
```

```
[[ 0.04307198  0.04306862 -0.43033115]]
[0.06381617]
```



In [ ]:
```
from google.colab import output
output.disable_custom_widget_manager()
```

i) Using the code below that generates 3 concentric circles, fit a logisitc regression model to it and plot the decision boundary.

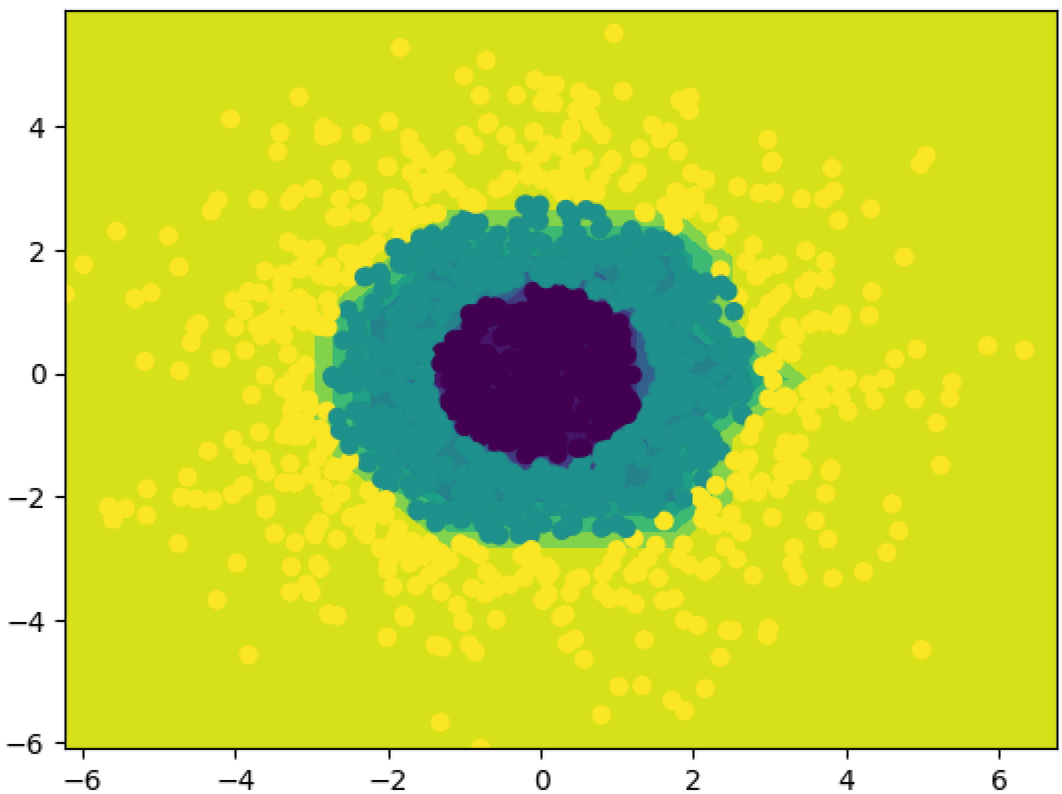
In [ ]:
```
t, _ = datasets.make_blobs(n_samples=1500, centers=centers, cluster_std=2,
                                random_state=0)

# CIRCLES
def generate_circles_data(t):
    def label(x):
        if x[0]**2 + x[1]**2 >= 2 and x[0]**2 + x[1]**2 < 8:
            return 1
        if x[0]**2 + x[1]**2 >= 8:
            return 2
        return 0
    # create some space between the classes
    X = np.array(list(filter(lambda x : (x[0]**2 + x[1]**2 < 1.8 or x[0]**2
    Y = np.array([label(x) for x in X])
    return X, Y

X, Y = generate_circles_data(t)

poly = PolynomialFeatures(2)
```

```
lr = LogisticRegression(verbose=2)
model = make_pipeline(poly, lr).fit(X, Y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:45
8: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  n_iter_i = _check_optimize_result(
```

In [ ]:

```
xx, yy = np.meshgrid(np.arange(X[:, 0].min(), X[:, 0].max() + 1), np.arange(
plt.contourf(xx, yy, Z.reshape(xx.shape))
plt.scatter(X[:, 0], X[:, 1], c = Y)
plt.show()
```



Gradient Descent

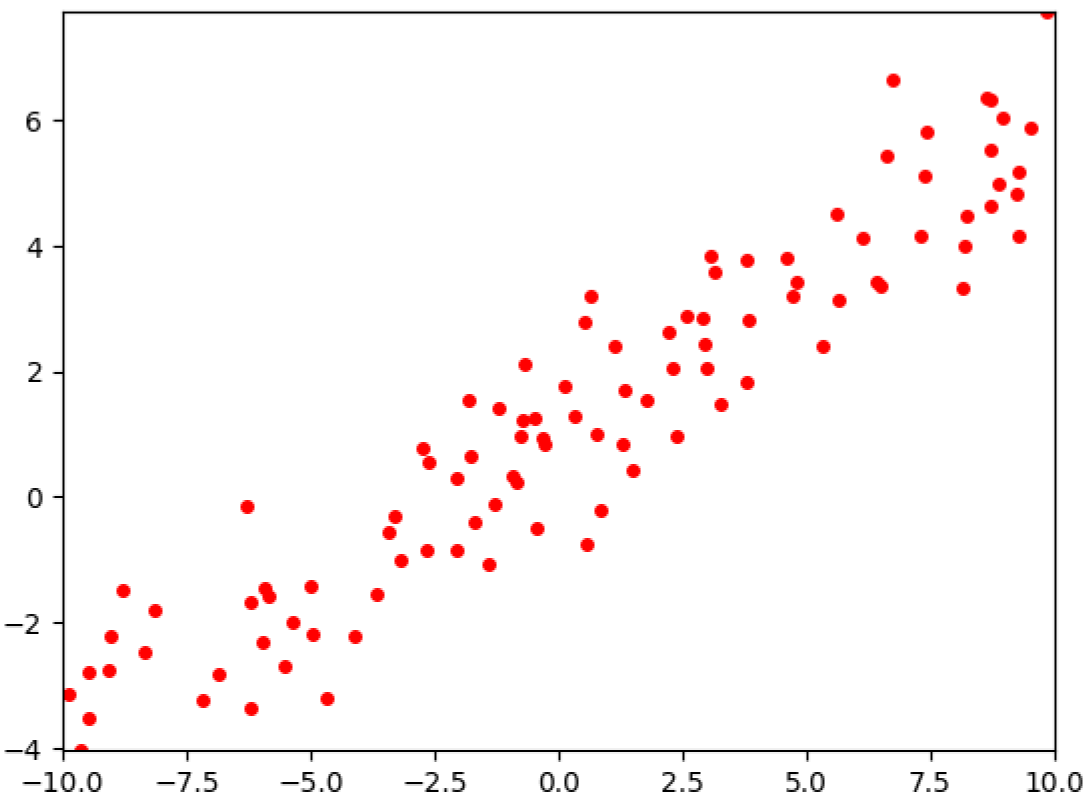
Recall in Linear Regression we are trying to find the line $$y = X \beta$$ that minimizes the sum of square distances between the predicted `y` and the `y` we observed in our dataset:

$$\mathcal{L}(\mathbf{\beta}) = \Vert \mathbf{y} - X\mathbf{\beta} \Vert^2$$

We were able to find a global minimum to this loss function but we will try to apply gradient descent to find that same solution.

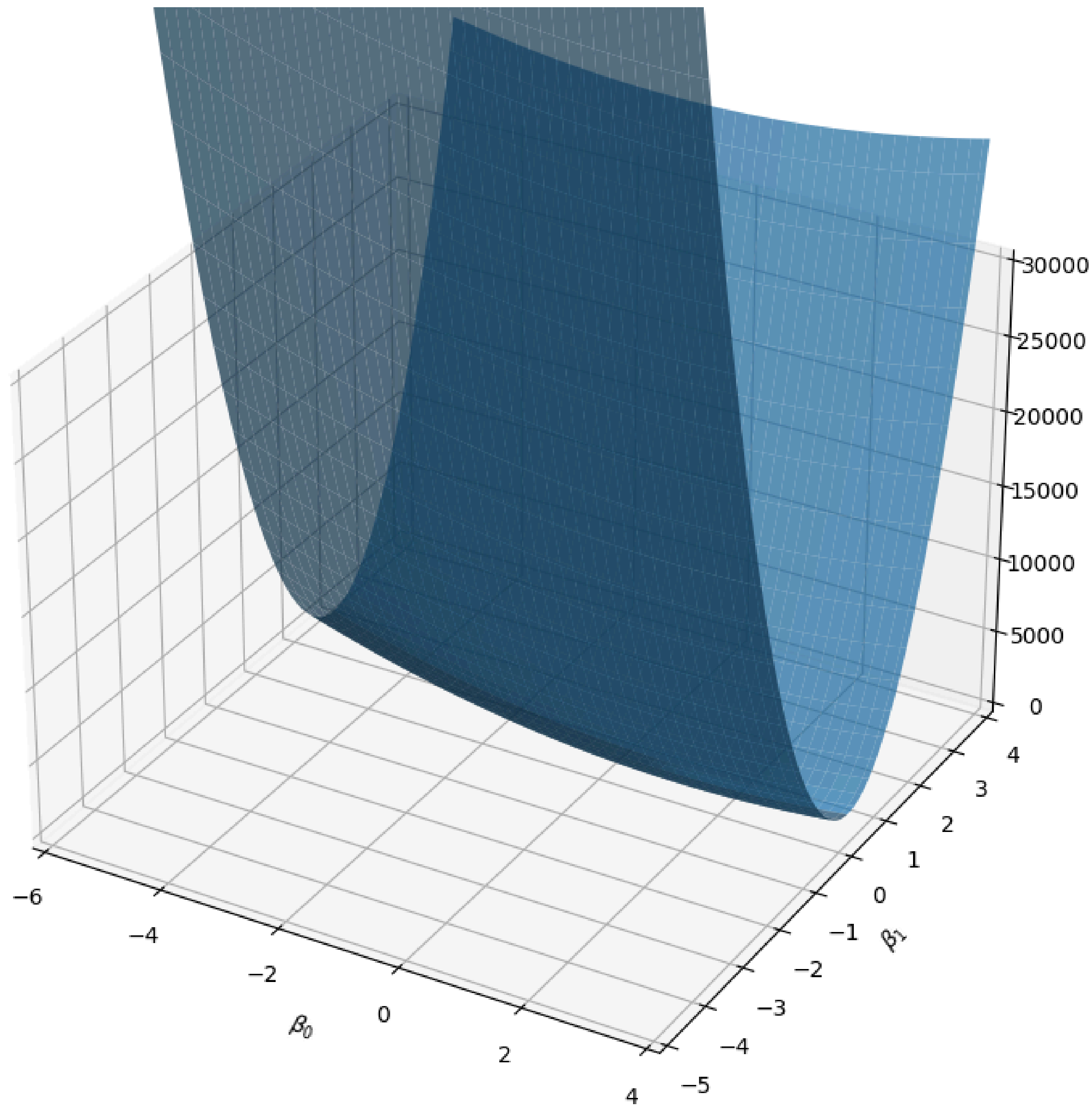
a) Implement the `loss` function to complete the code and plot the loss as a function of beta.

In [ ]:
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

beta = np.array([ 1 , .5 ])
xlin = -10.0 + 20.0 * np.random.random(100)
X = np.column_stack([np.ones((len(xlin), 1)), xlin])
y = beta[0]+(beta[1]*xlin)+np.random.randn(100)

fig, ax = plt.subplots()
ax.plot(xlin, y,'ro',markersize=4)
ax.set_xlim(-10, 10)
ax.set_ylim(min(y), max(y))
plt.show()
```



In [ ]:
```
b0 = np.arange(-5, 4, 0.1)
b1 = np.arange(-5, 4, 0.1)
b0, b1 = np.meshgrid(b0, b1)

def loss(X, y, beta):
    return beta.T @ X.T @ X @ beta - 2 * beta.T @ X.T @ y + y.T @ y

def get_cost(B0, B1):
    res = []
    for b0, b1 in zip(B0, B1):
        line = []
        for i in range(len(b0)):
```

```
            beta = np.array([b0[i], b1[i]])
            line.append(loss(X, y, beta))
        res.append(line)
    return np.array(res)

cost = get_cost(b0, b1)

# Creating figure
fig = plt.figure(figsize =(14, 9))
ax = plt.axes(projection ='3d')
ax.set_xlim(-6, 4)
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_ylim(-5, 4)
ax.set_zlim(0, 30000)

# Creating plot
ax.plot_surface(b0, b1, cost, alpha=.7)

# show plot
plt.show()
```



Since the loss is

$$\mathcal{L}(\mathbf{\beta}) = \Vert \mathbf{y} - X\mathbf{\beta} \Vert^2 = \beta^T X^T X \beta - 2\mathbf{\beta}^TX^T\mathbf{y} + \mathbf{y}^T\mathbf{y}$$

The gradient is

$$\nabla_\beta \mathcal{L}(\mathbf{\beta}) = 2X^T X \beta - 2X^T\mathbf{y}$$

b) Implement the gradient function below and complete the gradient descent algorithm

In [ ]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt

TEMPFILE = "temp.png"

def snap(betas, losses):
    # Creating figure
    fig = plt.figure(figsize =(14, 9))
```

```
    ax = plt.axes(projection ='3d')
    ax.view_init(20, -20)
    ax.set_xlim(-5, 4)
    ax.set_xlabel(r'$\beta_0$')
    ax.set_ylabel(r'$\beta_1$')
    ax.set_ylim(-5, 4)
    ax.set_zlim(0, 30000)

    # Creating plot
    ax.plot_surface(b0, b1, cost, color='b', alpha=.7)
    ax.plot(np.array(betas)[:,0], np.array(betas)[:,1], losses, 'o-', c='r',
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient(X, y, beta):
    return 2 * X.T @ X @ beta - 2 * X.T @ y


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap(betas, losses))
        beta_hat = beta_hat + learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)


beta_start = np.array([-5, -2])
learning_rate = 0.0002 # try .0005
images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 10, images

images[0].save(
    'gd.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```

c) Use the code above to create an animation of the linear model learned at every epoch.

In [ ]:
```
def snap_model(beta):
    xplot = np.linspace(-10,10,50)
    yestplot = beta[0] + beta[1] * xplot
    fig, ax = plt.subplots()
    ax.plot(xplot, yestplot,'b-',lw=2)
```

```
    ax.plot(xlin, y,'ro',markersize=4)
    ax.set_xlim(-10, 10)
    ax.set_ylim(min(y), max(y))
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap_model(beta_hat))
        beta_hat = beta_hat - learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)


images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 100, image

images[0].save(
    'model.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=200
)
```

In logistic regression, the `loss` is the negative log-likelihood

$$ \mathcal{l}(\mathbf{\beta}) = - \frac{1}{N} \sum_{i=1}^{N} y_i \log(\sigma(x_i \beta)) + (1 - y_i)\log(1 - \sigma(x_i \beta))$$

the gradient of which is:

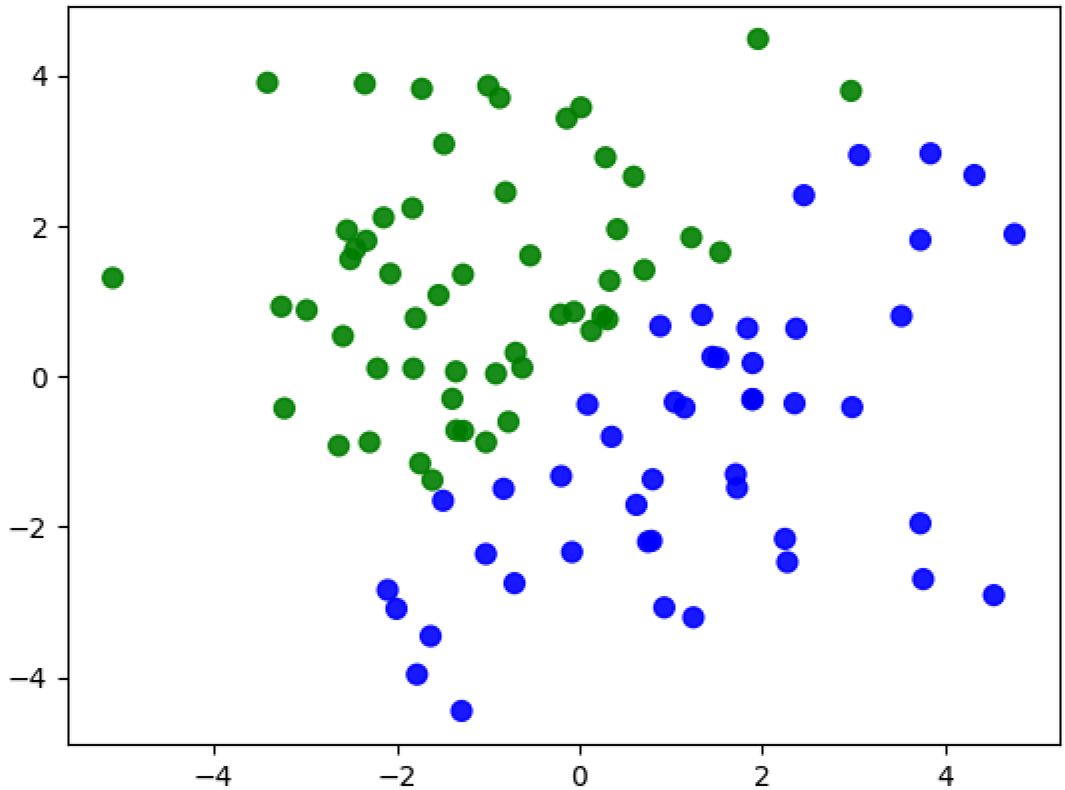
$$\nabla_\beta \mathcal{l}(\mathbf{\beta}) = \frac{1}{N} \sum_{i=1}^{N} x_i (y_i - \sigma(x_i \beta)) $$

d) Plot the loss as a function of b.

In [ ]:

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

centers = [[0, 0]]
t, _ = datasets.make_blobs(n_samples=100, centers=centers, cluster_std=2, ra

# LINE
def generate_line_data():
```

```
    # create some space between the classes
    X = t
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y

X, y = generate_line_data()

cs = np.array([x for x in 'gb'])
fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], color=cs[y].tolist(), s=50, alpha=0.9)
plt.show()
```

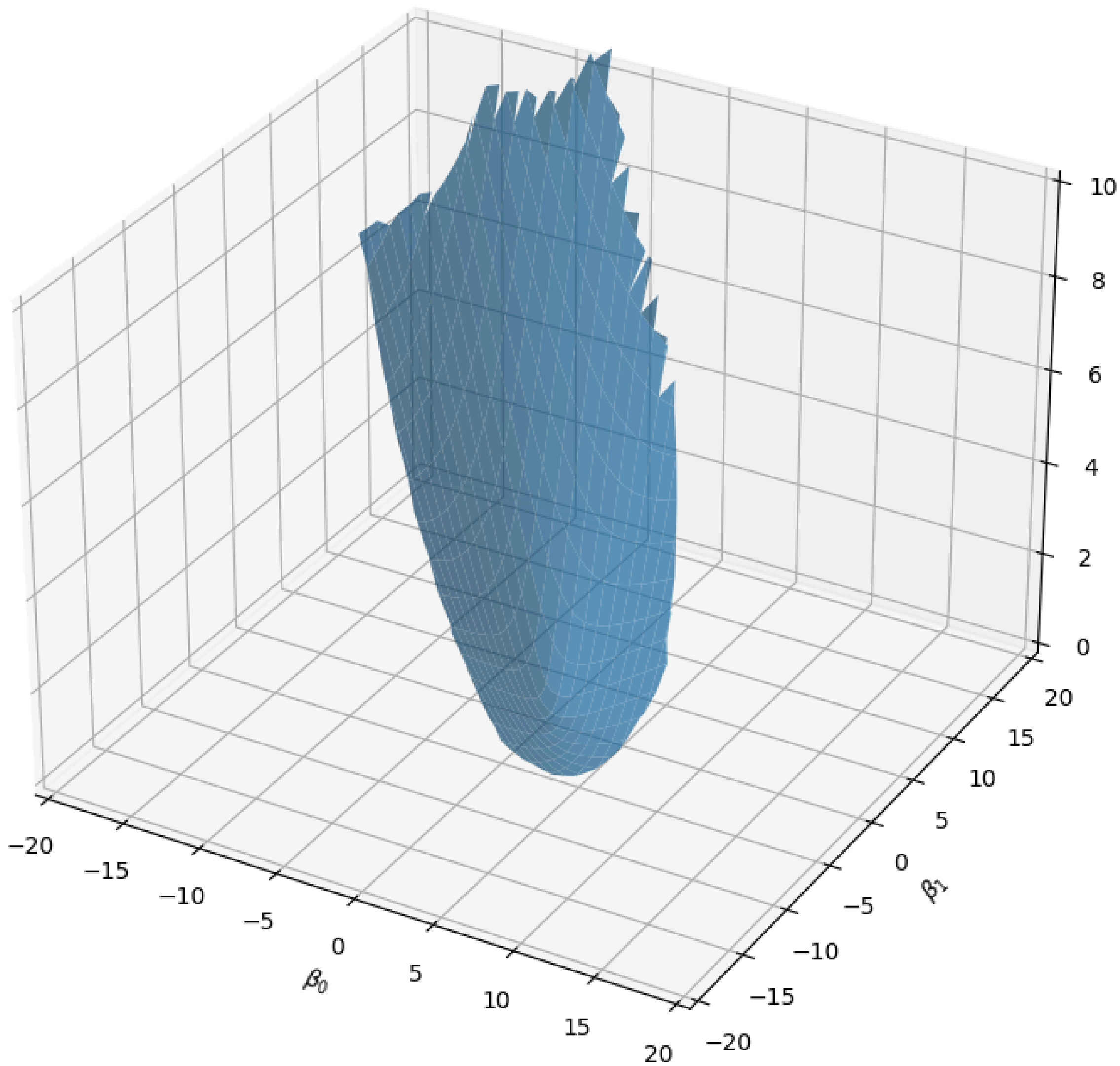


In [ ]:

```
b0 = np.arange(-20, 20, 0.1)
b1 = np.arange(-20, 20, 0.1)
b0, b1 = np.meshgrid(b0, b1)


def sigmoid(x):
    e = np.exp(x)
    return e / (1 + e)


def loss(X, y, beta):
    sigmoidfunc = sigmoid(np.dot(X, beta))
    return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfu

def get_cost(B0, B1):
    res = []
    for b0, b1 in zip(B0, B1):
        line = []
        for i in range(len(b0)):
```

```
            beta = np.array([b0[i], b1[i]])
            line.append(loss(X, y, beta))
        res.append(line)
    return np.array(res)

cost = get_cost(b0, b1)

# Creating figure
fig = plt.figure(figsize =(14, 9))
ax = plt.axes(projection ='3d')
ax.set_xlim(-20, 20)
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_ylim(-20, 20)
ax.set_zlim(0, 10)

# Creating plot
ax.plot_surface(b0, b1, cost, alpha=.7)

# show plot
plt.show()
```

```
<ipython-input-49-b928f83e3f8c>:13: RuntimeWarning: divide by zero encountere
d in log
  return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfun
c))
<ipython-input-49-b928f83e3f8c>:13: RuntimeWarning: invalid value encountered
in multiply
  return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfun
c))
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/art3d.py:1180: R
untimeWarning: invalid value encountered in subtract
  v1[poly_i, :] = ps[i1, :] - ps[i2, :]
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/art3d.py:1181: R
untimeWarning: invalid value encountered in subtract
  v2[poly_i, :] = ps[i2, :] - ps[i3, :]
/usr/local/lib/python3.10/dist-packages/numpy/core/numeric.py:1652: RuntimeWa
rning: invalid value encountered in subtract
  cp1 -= tmp
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/proj3d.py:180: R
untimeWarning: invalid value encountered in divide
  txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```



e) Plot the loss at each iteration of the gradient descent algorithm.

In [ ]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt

TEMPFILE = "tempForPartE.png"

def snap(betas, losses):
    # Creating figure
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    ax.view_init(10, 10)
    ax.set_xlabel(r'$\beta_0$')
    ax.set_ylabel(r'$\beta_1$')
    ax.set_ylim(-20, 20)
    ax.set_zlim(0, 10)

    # Creating plot
```

```
    ax.plot_surface(b0, b1, cost, color='b', alpha=.7)
    ax.plot(np.array(betas)[:,0], np.array(betas)[:,1], losses, 'o-', c='r',
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient(X, y, beta):
    sigmoidfunc = sigmoid(np.dot(X, beta))
    return np.mean(X.T * (y - sigmoidfunc))


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap(betas, losses))
        beta_hat = beta_hat - learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)


beta_start = np.array([-5, -2])
learning_rate = 0.1
images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 10, images

images[0].save(
    'gd_logit.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```

```
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/art3d.py:1180: R
untimeWarning: invalid value encountered in subtract
  v1[poly_i, :] = ps[i1, :] - ps[i2, :]
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/art3d.py:1181: R
untimeWarning: invalid value encountered in subtract
  v2[poly_i, :] = ps[i2, :] - ps[i3, :]
/usr/local/lib/python3.10/dist-packages/numpy/core/numeric.py:1652: RuntimeWa
rning: invalid value encountered in subtract
  cp1 -= tmp
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/proj3d.py:180: R
untimeWarning: invalid value encountered in divide
  txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```

f) Create an animation of the logistic regression fit at every epoch.

In [ ]:
```
def snap_model(beta):
    xplot = np.linspace(-10,10,50)
    yestplot = beta[0] + beta[1] * xplot
    fig, ax = plt.subplots()
    ax.plot(xplot, yestplot,'b-',lw=2)
    ax.plot(xlin, y,'ro',markersize=4)
    ax.set_xlim(-10, 10)
    ax.set_ylim(min(y), max(y))
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap_model(beta_hat))
        beta_hat = beta_hat - learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)


images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 100, image

images[0].save(
    'regressionPartF.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=200
)
```

```
<ipython-input-49-b928f83e3f8c>:13: RuntimeWarning: divide by zero encountere
d in log
  return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfun
c))
<ipython-input-49-b928f83e3f8c>:13: RuntimeWarning: invalid value encountered
in multiply
  return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfun
c))
```

g) Modify the above code to evaluate the gradient on a random batch of the data. Overlay the true loss curve and the approximation of the loss in your animation.

In [ ]:
```
def snap_model(beta):
    xplot = np.linspace(-10,10,50)
    yestplot = beta[0] + beta[1] * xplot
    fig, ax = plt.subplots()
```

```
    ax.plot(xplot, yestplot,'b-',lw=2)
    ax.plot(xlin, y,'ro',markersize=4)
    ax.set_xlim(-10, 10)
    ax.set_ylim(min(y), max(y))
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))

def gradient_descent(X, y, beta_hat, learning_rate, epochs, images, batch_si
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for epoch in range(epochs):
        batch_indices = np.random.choice(range(X.shape[0]), size = batch_siz
        X_batch = X[batch_indices]
        y_batch = y[batch_indices]

        images.append(snap_model(beta_hat))
        beta_hat = beta_hat - learning_rate * gradient(X_batch, y_batch, bet

        losses.append(loss(X_batch, y_batch, beta_hat))
        betas.append(beta_hat)



    return np.array(betas), np.array(losses), images




images = []
size = 32
betas, losses, images = gradient_descent(X, y, beta_start, learning_rate, 10

images[0].save(
    'regressionPartF.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=200
)
```

```
<ipython-input-49-b928f83e3f8c>:13: RuntimeWarning: divide by zero encountere
d in log
  return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfun
c))
<ipython-input-49-b928f83e3f8c>:13: RuntimeWarning: invalid value encountered
in multiply
  return -np.mean(y * np.log(sigmoidfunc) + (1 - y) * np.log(1 - sigmoidfun
c))
```

h) Below is a sandox where you can get intuition about how to tune gradient descent parameters:

In [ ]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt

TEMPFILE = "temp.png"

def snap(x, y, pts, losses, grad):
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    ax.view_init(20, -20)
    ax.plot_surface(x, y, loss(np.array([x, y])), color='r', alpha=.4)
    ax.plot(np.array(pts)[:,0], np.array(pts)[:,1], losses, 'o-', c='b', mar
    ax.plot(np.array(pts)[-1,0], np.array(pts)[-1,1], -1, 'o-', c='b', alpha

    # Plot Gradient Vector
    X, Y, Z = [pts[-1][0]], [pts[-1][1]], [-1]
    U, V, W = [-grad[0]], [-grad[1]], [0]
    ax.quiver(X, Y, Z, U, V, W, color='g')
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))

def loss(x):
    return np.tan(sum(x**2))

def gradient(x):
    return x * np.sin(sum(x**2))

def gradient_descent(x, y, init, learning_rate, epochs):
    images, losses, pts = [], [loss(init)], [init]
    for _ in range(epochs):
        grad = gradient(init)
        images.append(snap(x, y, pts, losses, grad))
        init = init - learning_rate * grad
        losses.append(loss(init))
        pts.append(init)
    return images

init = np.array([-.1, -.1])
learning_rate = 1.9
x, y = np.meshgrid(np.arange(-5, 5, 0.1), np.arange(-5, 5, 0.1))
images = gradient_descent(x, y, init, learning_rate, 12)

images[0].save(
    'gradient_descent.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```

In [ ]: