Fall 2023 CS4641/CS7641 A Homework 3

Instructor: Dr. Mahdi Roozbahani

Deadline: Friday, November 10th, 11:59 pm EST

- No unapproved extension of the deadline is allowed. Submission past our 48-hour penalized acceptance period will lead to 0 credit.
- Discussion is encouraged on Ed as part of the Q/A. However, all assignments should be done individually.
- Plagiarism is a **serious offense**. You are responsible for completing your own work. You are not allowed to copy and paste, or paraphrase, or submit materials created or published by others, as if you created the materials. All materials submitted must be your own.
- All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures. If we observe any (even small) similarities/plagiarisms detected by Gradescope or our TAs, **WE WILL DIRECTLY REPORT ALL CASES TO OSI**, which may, unfortunately, lead to a very harsh outcome. **Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.**

Instructions for the assignment

- This assignment consists of both programming and theory questions.
- Unless a theory question explicitly states that no work is required to be shown, you must provide an explanation, justification, or calculation for your answer.
- To switch between cell for code and for markdown, see the menu -> Cell -> Cell Type
- You can directly type Latex equations into markdown cells.
- If a question requires a picture, you could use this syntax `<img src="" style="width: 300px;"/>` to include them within your ipython notebook.
- Your write up must be submitted in PDF form. You may use either Latex, markdown, or any word processing software. We will **NOT** accept handwritten work. Make sure that your work is formatted correctly, for example submit $\sum_{i=0} x_i$ instead of \ text{sum_{i=0} x_i}
- When submitting the non-programming part of your assignment, you must correctly map pages of your PDF to each question/subquestion to reflect where they appear.

Improperly mapped questions may not be graded correctly and/or will result in point deductions for the error.

- All assignments should be done individually, and each student must write up and submit their own answers.
- **Graduate Students**: You are required to complete any sections marked as Bonus for Undergrads

Using the autograder

- Grads will find three assignments on Gradescope that correspond to HW3: "Assignment 3 Programming", "Assignment 3 - Non-programming" and "Assignment 3 Programming - Bonus for all". Undergrads will find an additional assignment called "Assignment 3 Programming - Bonus for Undergrads".
- You will submit your code for the autograder in the Assignment 3 Programming sections. Please refer to the Deliverables and Point Distribution section for what parts are considered required, bonus for undergrads, and bonus for all.
- We provided you different .py files and we added libraries in those files please DO NOT remove those lines and add your code after those lines. Note that these are the only allowed libraries that you can use for the homework.
- You are allowed to make as many submissions until the deadline as you like. Additionally, note that the autograder tests each function separately, therefore it can serve as a useful tool to help you debug your code if you are not sure of what part of your implementation might have an issue.
- **For the "Assignment 3 - Non-programming" part, you will need to submit to Gradescope a PDF copy of your Jupyter Notebook with the cells ran. See this EdStem Post for multiple ways on to convert your .ipynb into a .pdf file.** Please refer to the **Deliverables and Point Distribution** section for an outline of the non-programming questions.
- **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem. The pages in the PDF should be of size 8.5" x 11", otherwise there may be a deduction in points for extra long sheets.**

Using the local tests

- For some of the programming questions we have included a local test using a small toy dataset to aid in debugging. The local test sample data and outputs are stored in .py files in the **local_tests_folder**. The actual local tests are stored in **localtests.py**.
- There are no points associated with passing or failing the local tests, you must still pass the autograder to get points.

- **It is possible to fail the local test and pass the autograder** since the autograder has a certain allowed error tolerance while the local test allowed error may be smaller. Likewise, passing the local tests does not guarantee passing the autograder.
- **You do not need to pass both local and autograder tests to get points, passing the Gradescope autograder is sufficient for credit.**
- It might be helpful to comment out the tests for functions that have not been completed yet.
- It is recommended to test the functions as it gets completed instead of completing the whole class and then testing. This may help in isolating errors. Do not solely rely on the local tests, continue to test on the autograder regularly as well.

Deliverables and Points Distribution

Q1: Image Compression [30pts]

Deliverables: imgcompression.py and printed results

- **1.1 Image Compression** [20 pts] - *programming*
  - svd [4pts]
  - compress [4pts]
  - rebuild_svd [4pts]
  - compression_ratio [4pts]
  - recovered_variance_proportion [4pts]
- **1.2 Black and White** [5 pts] *non-programming*
- **1.3 Color Image** [5 pts] *non-programming*

Q2: Understanding PCA [20pts]

Deliverables: pca.py and written portion

- **2.1 PCA Implementation** [10 pts] - *programming*
  - fit [5pts]
  - transform [2pts]
  - transform_rv [3pts]
- **2.2 Visualize** [5 pts] *programming and non-programming*
- **2.3 PCA Reduced Facemask Dataset Analysis** [5 pts] *non-programming*
- **2.4 PCA Exploration** [0 pts]

Q3: Regression and Regularization [80pts: 50pts + 20pts Bonus for Undergrads + 12pts Bonus for All]

Deliverables: regression.py and Written portion

- **3.1 Regression and Regularization Implementations** [50pts: 30pts + 20pts Bonus for Undergrad] - *programming*
  - RMSE [5pts]
  - Construct Poly Features 1D [2pts]
  - Construct Poly Features 2D [3pts]
  - Prediction [5pts]
  - Linear Fit Closed Form [5pts]
  - Ridge Fit Closed Form [5pts]
  - Cross Validation [5pts]
  - Linear Gradient Descent [5pts] **Bonus for Undergrad**
  - Linear Stochastic Gradient Descent [5pts] **Bonus for Undergrad**
  - Ridge Gradient Descent [5pts] **Bonus for Undergrad**
  - Ridge Stochastic Gradient Descent [5pts] **Bonus for Undergrad**
- **3.2 About RMSE** [3 pts] *non-programming*
- **3.3 Testing: General Functions and Linear Regression** [5 pts] *non-programming*
- **3.4 Testing: Ridge Regression** [5 pts + 2 pts Bonus for All] *non-programming*
- **3.5 Cross Validation** [7 pts] *non-programming*
- **3.6 Noisy Input Samples in Linear Regression** [10 pts] *non-programming* **BONUS FOR ALL**

Q4: Naive Bayes and Logistic Regression [35pts]

Deliverables: logistic_regression.py and Written portion

- **4.1 Llama Breed Problem using Naive Bayes** [5 pts] *non-programming*
- **4.2 News Data Sentiment Classification Using Logistic Regression** [30 pts] - *programming*
  - sigmoid [2 pts]
  - bias_augment [3 pts]

  - predict_probs [5 pts]
  - predict_labels [2 pts]
  - loss [3 pts]
  - gradient [3 pts]
  - accuracy [2 pts]
  - evaluate [5 pts]
  - fit [5 pts]

Q5: Noise in PCA and Linear Regression [15pts]

Deliverables: Written portion

- **5.1 Slope Functions** [5 pts] *non-programming*
- **5.2 Error in Y and Error in X and Y** [5 pts] *non-programming*
- **5.3 Analysis** [5 pts] *non-programming*

Q6: Feature Reduction.py [25pts Bonus for All]

Deliverables: feature_reduction.py and Written portion

- **6.1 Feature Reduction** [18 pts] - *programming*
  - forward_selection [9pts]
  - backward_elimination [9pts]
- **6.2 Feature Selection - Discussion** [7 pts] *non-programming*

Q7: Movie Recommendation with SVD [10pts Bonus for All]

Deliverables: svd_recommender.py and Written portion

- **7.1 SVD Recommender**
  - recommender_svd [5pts]
  - predict [5pts]
- **7.2 Visualize Movie Vectors** [0pts]

0 Set up

This notebook is tested under python 3._ ._, and the corresponding packages can be downloaded from miniconda. You may also want to get yourself familiar with several packages:

- jupyter notebook

- numpy
- matplotlib
- sklearn

There is also a VS Code and Anaconda Setup Tutorial on Ed under the "Links" category

Please implement the functions that have `raise NotImplementedError`, and after you finish the coding, please delete or comment out `raise NotImplementedError`.

Library imports

```
################################
### DO NOT CHANGE THIS CELL ###
################################
# This is cell which sets up some of the modules you might need
# Please do not change the cell or import any additional packages.

import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
from sklearn.feature_extraction import text
from sklearn.datasets import load_diabetes, load_breast_cancer,
load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score
import warnings
import sys

print('Version information')

print('python: {}'.format(sys.version))
print('matplotlib: {}'.format(matplotlib.__version__))
print('numpy: {}'.format(np.__version__))

warnings.filterwarnings('ignore')

%matplotlib inline
%load_ext autoreload
%autoreload 2

STUDENT_VERSION = 1
EO_TEXT, EO_FONT, EO_COLOR = 'TA VERSION', 'Chalkduster', 'gray',
EO_ALPHA, EO_SIZE, EO_ROT = 0.7, 90, 40
# Render types : 'browser', 'png', 'plotly_mimetype', 'jupyterlab',
pdf
rndr_type = 'png'
```

```
Version information
python: 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023,
```

```
13:26:23) [MSC v.1916 64 bit (AMD64)]
matplotlib: 3.7.2
numpy: 1.26.0
```

Q1: Image Compression [30 pts] **[P]** | **[W]**

Load images data and plot

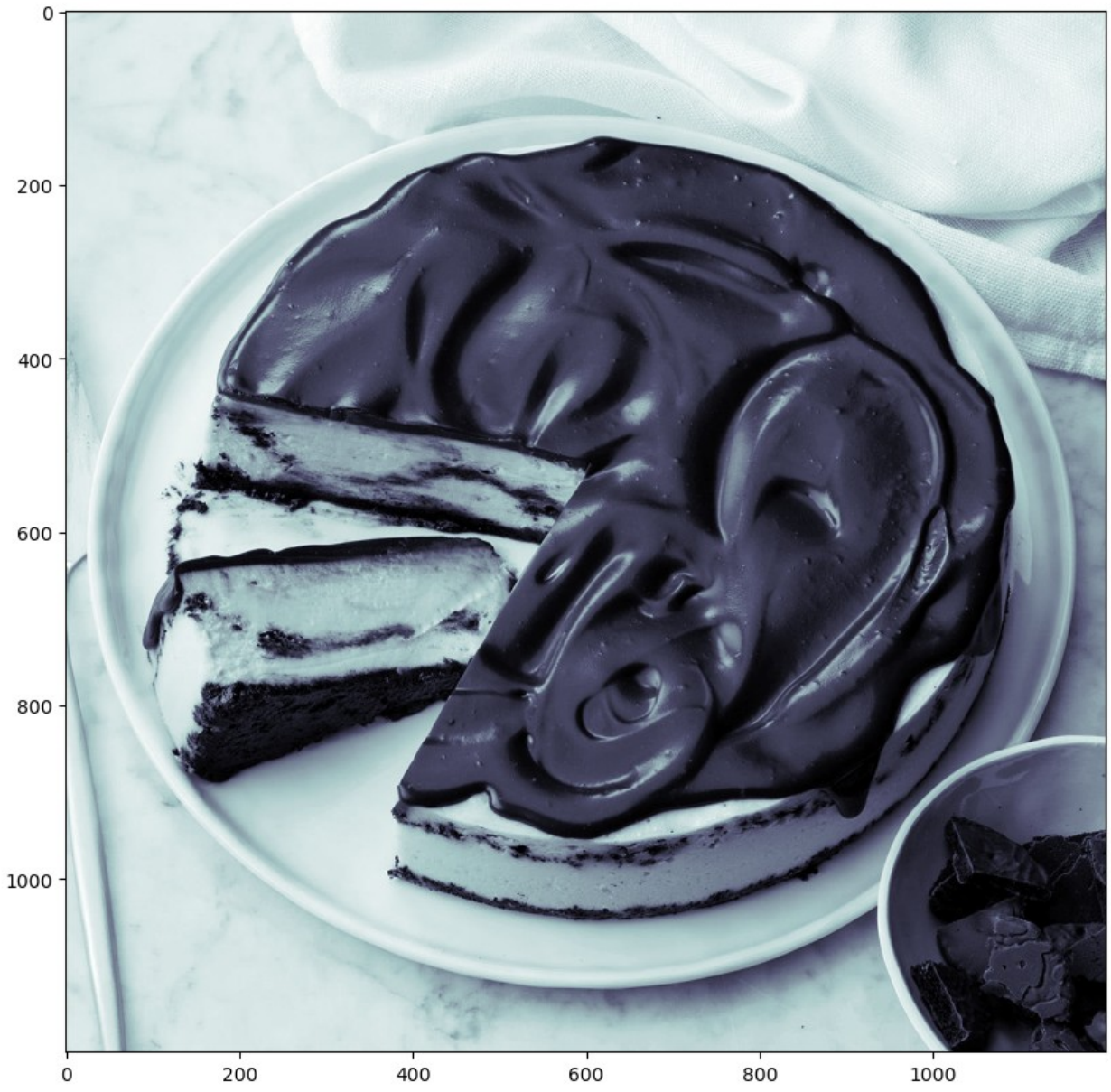
```
################################
### DO NOT CHANGE THIS CELL ###
################################
# load Image
image = plt.imread("./data/hw3_image_compression.jpeg") / 255
# plot image
fig = plt.figure(figsize=(10, 10))
if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE,
        color=EO_COLOR,
        alpha=EO_ALPHA,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT,
    )
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x20ac1864dd0>
```


```
################################
### DO NOT CHANGE THIS CELL ###
################################
def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.299, 0.587, 0.114])


fig = plt.figure(figsize=(10, 10))

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
```

```
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE,
        color=EO_COLOR,
        alpha=EO_ALPHA,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT,
    )
# plot several images
plt.imshow(rgb2gray(image), cmap=plt.cm.bone)
```

```
<matplotlib.image.AxesImage at 0x20ac1b4d090>
```



1.1 Image compression [20pts] **[P]**

SVD is a dimensionality reduction technique that allows us to compress images by throwing away the least important information.

Higher singular values capture greater variance and, thus, capture greater information from the corresponding singular vector. To perform image compression, apply SVD on each matrix and get rid of the small singular values to compress the image. The loss of information through this process is negligible, and the difference between the images can be hardly spotted.

For example, the proportion of variance captured by the first component is

$$\frac{\sigma_1^2}{\sum_{i=1}^{n} \sigma_i^2}$$

where σ_i is the i^{th} singular value.

In the imgcompression.py file, complete the following functions:

- **svd**: You may use np.linalg.svd in this function, and although the function defaults this parameter to true, you may explicitly set full_matrices=True using the optional full_matrices parameter. Hint 2 may be useful.
- **compress**
- **rebuild_svd**
- **compression_ratio**: Hint 1 may be useful
- **recovered_variance_proportion**: Hint 1 may be useful

HINT 1: http://timbaumann.info/svd-image-compression-demo/ is a useful article on image compression and compression ratio. You may find this article useful for implementing the functions compression_ratio and recovered_variance_proportion

HINT 2: If you have never used np.linalg.svd, it might be helpful to read Numpy's SVD documentation and note the particularities of the V matrix and that it is returned already transposed.

HINT 3: The shape of S resulting from SVD may change depending on if N > D, N < D, or N = D. Therefore, when checking the shape of S, note that min(N,D) means the value should be equal to whichever is lower between N and D.

1.1.1 Local Tests for Image Compression Black and White Case [No Points]

You may test your implementation of the functions contained in **imgcompression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestImgCompression

unittest_ic = TestImgCompression()
unittest_ic.test_svd_bw()
unittest_ic.test_compress_bw()
unittest_ic.test_rebuild_svd_bw()
unittest_ic.test_compression_ratio_bw()
unittest_ic.test_recovered_variance_proportion_bw()
```

```
UnitTest passed successfully for "SVD calculation - black and white
images"!
UnitTest passed successfully for "Image compression - black and white
```

```
images"!
UnitTest passed successfully for "SVD reconstruction - black and white
images"!
UnitTest passed successfully for "Compression ratio - black and white
images"!
UnitTest passed successfully for "Recovered variance proportion -
black and white images"!
```

1.1.2 Local Tests for Image Compression Color Case [No Points]

You may test your implementation of the functions contained in **imgcompression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from utilities.localtests import TestImgCompression

unittest_ic = TestImgCompression()

unittest_ic.test_svd_color()
unittest_ic.test_compress_color()
unittest_ic.test_rebuild_svd_color()
unittest_ic.test_compression_ratio_color()
unittest_ic.test_recovered_variance_proportion_color()
```

```
UnitTest passed successfully for "SVD calculation - color images"!
UnitTest passed successfully for "Image compression - color images"!
UnitTest passed successfully for "SVD reconstruction - color images"!
UnitTest passed successfully for "Compression ratio - color images"!
UnitTest passed successfully for "Recovered variance proportion -
color images"!
```

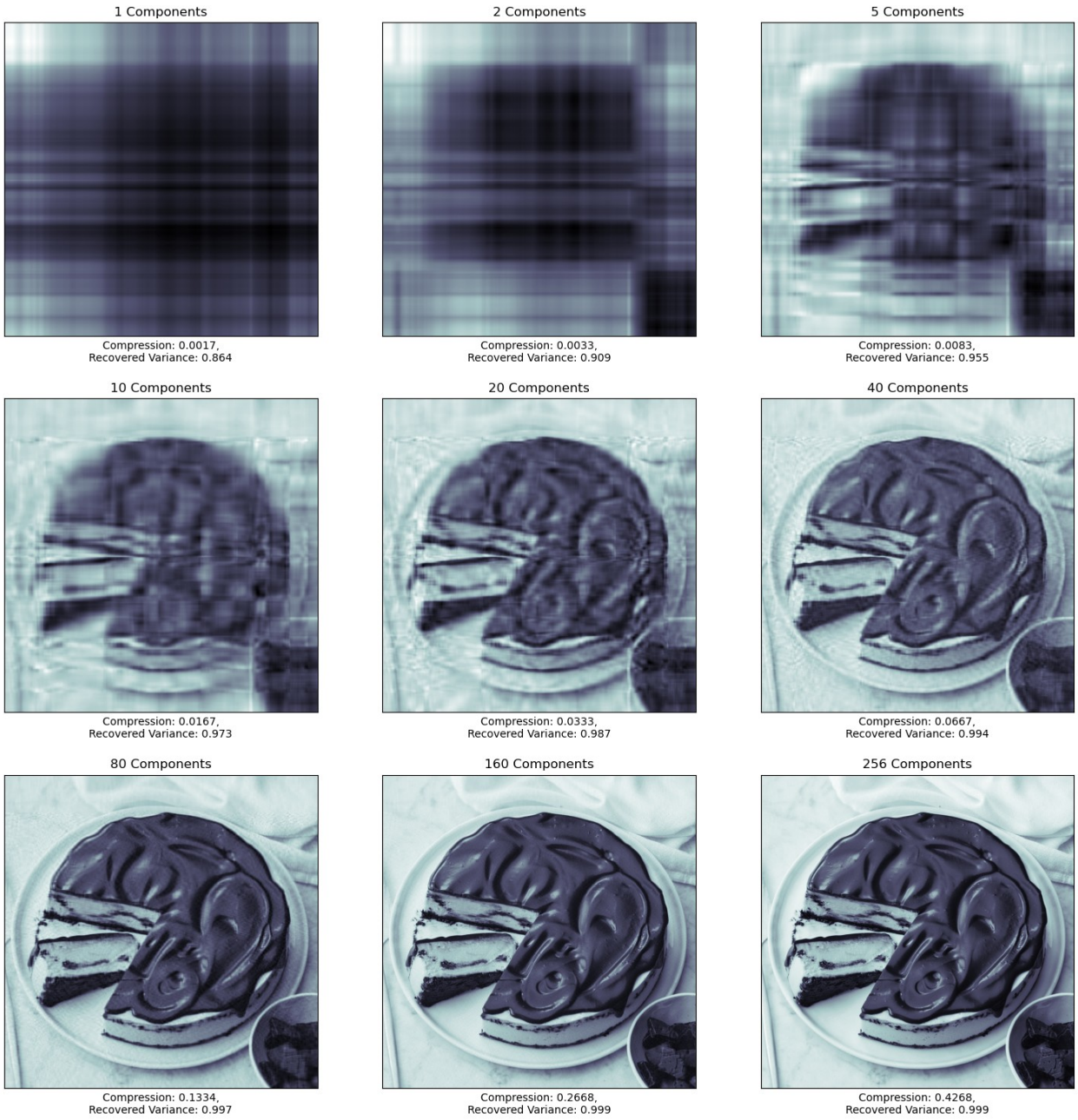
1.2.1 Black and white [5 pts] **[W]**

This question will use your implementation of the functions from Q1.1 to generate a set of images compressed to different degrees. You can simply run the below cell without making any changes to it, assuming you have implemented the functions in Q1.1.

Make sure these images are displayed when submitting the PDF version of the Juypter notebook as part of the non-programming submission of this assignment.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
from imgcompression import ImgCompression
```

```
imcompression = ImgCompression()
bw_image = rgb2gray(image)
U, S, V = imcompression.svd(bw_image)
component_num = [1, 2, 5, 10, 20, 40, 80, 160, 256]

fig = plt.figure(figsize=(18, 18))

# plot several images
i = 0
for k in component_num:
    U_compressed, S_compressed, V_compressed =
imcompression.compress(U, S, V, k)
    img_rebuild = imcompression.rebuild_svd(U_compressed,
S_compressed, V_compressed)
    c = np.around(imcompression.compression_ratio(bw_image, k), 4)
    r = np.around(imcompression.recovered_variance_proportion(S, k),
3)
    ax = fig.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    ax.imshow(img_rebuild, cmap=plt.cm.bone)
    ax.set_title(f"{k} Components")
    if not STUDENT_VERSION:
        ax.text(
            0.5,
            0.5,
            EO_TEXT,
            transform=ax.transAxes,
            fontsize=EO_SIZE / 2,
            color=EO_COLOR,
            alpha=EO_ALPHA,
            fontname=EO_FONT,
            ha="center",
            va="center",
            rotation=EO_ROT,
        )
    ax.set_xlabel(f"Compression: {c},\nRecovered Variance: {r}")
    i = i + 1
```



1.2.2 Black and White Compression Savings [No Points]

This question will use your implementation of the functions from Q1.1 to compare the number of bytes required to represent the SVD decomposition for the original image to the compressed image using different degrees of compression. You can simply run the below cell without making any changes to it, assuming you have implemented the functions in Q1.1.

Running this cell is primarily for your own understanding of the compression process.

```
################################
### DO NOT CHANGE THIS CELL ###
```

```
##############################
from imgcompression import ImgCompression

imcompression = ImgCompression()
bw_image = rgb2gray(image)
U, S, V = imcompression.svd(bw_image)

component_num = [1, 2, 5, 10, 20, 40, 80, 160, 256]

# Compare memory savings for BW image
for k in component_num:
    og_bytes, comp_bytes, savings =
imcompression.memory_savings(bw_image, U, S, V, k)
    comp_ratio = og_bytes / comp_bytes
    og_bytes = imcompression.nbytes_to_string(og_bytes)
    comp_bytes = imcompression.nbytes_to_string(comp_bytes)
    savings = imcompression.nbytes_to_string(savings)
    print(
        f"{k} components: Original Image: {og_bytes} -> Compressed
Image: {comp_bytes}, Savings: {savings}, Compression Ratio
{comp_ratio:.1f}:1"
    )
```

```
1 components: Original Image: 10.986 MB -> Compressed Image: 18.758
KB, Savings: 10.968 MB, Compression Ratio 599.8:1
2 components: Original Image: 10.986 MB -> Compressed Image: 37.516
KB, Savings: 10.95 MB, Compression Ratio 299.9:1
5 components: Original Image: 10.986 MB -> Compressed Image: 93.789
KB, Savings: 10.895 MB, Compression Ratio 120.0:1
10 components: Original Image: 10.986 MB -> Compressed Image: 187.578
KB, Savings: 10.803 MB, Compression Ratio 60.0:1
20 components: Original Image: 10.986 MB -> Compressed Image: 375.156
KB, Savings: 10.62 MB, Compression Ratio 30.0:1
40 components: Original Image: 10.986 MB -> Compressed Image: 750.312
KB, Savings: 10.254 MB, Compression Ratio 15.0:1
80 components: Original Image: 10.986 MB -> Compressed Image: 1.465
MB, Savings: 9.521 MB, Compression Ratio 7.5:1
160 components: Original Image: 10.986 MB -> Compressed Image: 2.931
MB, Savings: 8.055 MB, Compression Ratio 3.7:1
256 components: Original Image: 10.986 MB -> Compressed Image: 4.689
MB, Savings: 6.297 MB, Compression Ratio 2.3:1
```

1.3.1 Color image [5 pts] **[W]**

This section will use your implementation of the functions from Q1.1 to generate a set of images compressed to different degrees. You can simply run the below cell without making any changes to it, assuming you have implemented the functions in Q1.1.

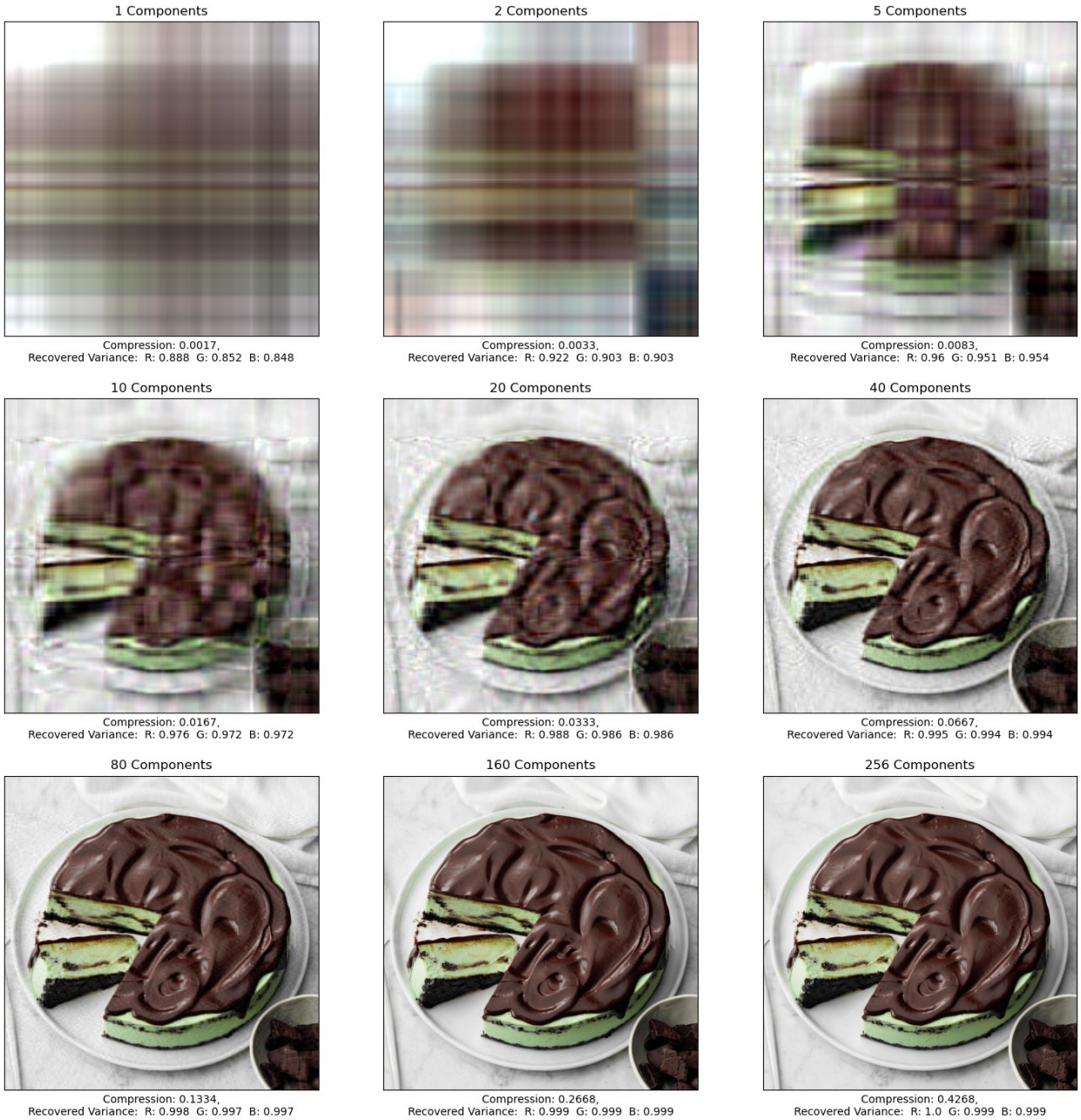
Make sure these images are displayed when submitting the PDF version of the Juypter notebook as part of the non-programming submission of this assignment.

NOTE: You might get warning "Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)." This warning is acceptable since some of the pixels may go above 1.0 while rebuilding. You should see similar images to original even with such clipping.

HINT 1: Make sure your implementation of recovered_variance_proportion returns an array of 3 floats for a color image.
HINT 2: Try performing SVD on the individual color channels and then stack the individual channel U, S, V matrices.
HINT 3: You may need separate implementations for a color or grayscale image in the same function.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from imgcompression import ImgCompression

imcompression = ImgCompression()
image_rolled = np.moveaxis(image, -1, 0)
U, S, V = imcompression.svd(image_rolled)

component_num = [1, 2, 5, 10, 20, 40, 80, 160, 256]

fig = plt.figure(figsize=(18, 18))

# plot several images
i = 0
for k in component_num:
    U_compressed, S_compressed, V_compressed =
imcompression.compress(U, S, V, k)
    img_rebuild = np.clip(
        imcompression.rebuild_svd(U_compressed, S_compressed,
V_compressed), 0, 1
    )
    img_rebuild = np.moveaxis(img_rebuild, 0, -1)
    c = np.around(imcompression.compression_ratio(image_rolled, k), 4)
    r = np.around(imcompression.recovered_variance_proportion(S, k),
3)
    ax = fig.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    ax.imshow(img_rebuild)
    ax.set_title(f"{k} Components")
    if not STUDENT_VERSION:
        ax.text(
            0.5,
            0.5,
            EO_TEXT,
            transform=ax.transAxes,
            fontsize=EO_SIZE / 2,
            color=EO_COLOR,
            alpha=EO_ALPHA,
```

```
            fontname=EO_FONT,
            ha="center",
            va="center",
            rotation=EO_ROT,
        )
    ax.set_xlabel(
        f"Compression: {np.around(c,4)},\nRecovered Variance:  R:
{r[0]}  G: {r[1]}  B: {r[2]}"
    )
    i = i + 1
```



1.3.2 Color Compression Savings [No Points]

This question will use your implementation of the functions from Q1.1 to compare the number of bytes required to represent the SVD decomposition for the original image to the compressed image using different degrees of compression. You can simply run the below cell without making any changes to it, assuming you have implemented the functions in Q1.1.

Running this cell is primarily for your own understanding of the compression process.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
from imgcompression import ImgCompression

imcompression = ImgCompression()
U, S, V = imcompression.svd(image_rolled)

component_num = [1, 2, 5, 10, 20, 40, 80, 160, 256]

# Compare the memory savings of the color image
i = 0
for k in component_num:
    og_bytes, comp_bytes, savings = imcompression.memory_savings(
        image_rolled, U, S, V, k
    )
    comp_ratio = og_bytes / comp_bytes
    og_bytes = imcompression.nbytes_to_string(og_bytes)
    comp_bytes = imcompression.nbytes_to_string(comp_bytes)
    savings = imcompression.nbytes_to_string(savings)
    print(
        f"{k} components: Original Image: {og_bytes} -> Compressed
Image: {comp_bytes}, Savings: {savings}, Compression Ratio
{comp_ratio:.1f}:1"
    )
```

```
1 components: Original Image: 32.959 MB -> Compressed Image: 56.273
KB, Savings: 32.904 MB, Compression Ratio 599.8:1
2 components: Original Image: 32.959 MB -> Compressed Image: 112.547
KB, Savings: 32.849 MB, Compression Ratio 299.9:1
5 components: Original Image: 32.959 MB -> Compressed Image: 281.367
KB, Savings: 32.684 MB, Compression Ratio 120.0:1
10 components: Original Image: 32.959 MB -> Compressed Image: 562.734
KB, Savings: 32.409 MB, Compression Ratio 60.0:1
20 components: Original Image: 32.959 MB -> Compressed Image: 1.099
MB, Savings: 31.86 MB, Compression Ratio 30.0:1
40 components: Original Image: 32.959 MB -> Compressed Image: 2.198
MB, Savings: 30.761 MB, Compression Ratio 15.0:1
80 components: Original Image: 32.959 MB -> Compressed Image: 4.396
MB, Savings: 28.563 MB, Compression Ratio 7.5:1
160 components: Original Image: 32.959 MB -> Compressed Image: 8.793
```

```
MB, Savings: 24.166 MB, Compression Ratio 3.7:1
256 components: Original Image: 32.959 MB -> Compressed Image: 14.068
MB, Savings: 18.891 MB, Compression Ratio 2.3:1
```

Q2: Understanding PCA [20 pts] **[P]** | **[W]**

Principal Component Analysis (PCA) is another dimensionality reduction technique that reduces dimensions or features while still preserving the maximum (or close-to) amount of information. This is useful when analyzing large datasets that contain a high number of dimensions or features that may be correlated. PCA aims to eliminate features that are highly correlated and only retain the important/uncorrelated ones that can describe most or all the variance in the data. This enables better interpretability and visualization of the multi-dimensional data. In this problem, we will investigate how PCA can be used to improve features for regression and classification tasks and how the data itself affects the behavior of PCA.

Here, we will employ Singular Value Decomposition (SVD) for PCA. In PCA, we first center the data by subtracting the mean of each feature. SVD is well suited for this task since each singular value tells us the amount of variance captured in each component for a given matrix (e.g. image). Hence, we can use SVD to extract data only in directions with high variances using either a threshold of the amount of variance or the number of bases/components. Here, we will reduce the data to a set number of components.

Recall from class that in PCA, we project the original matrix X into new components, each one corresponding to an eigenvector of the covariance matrix $X^T X$. We know that SVD decomposes X into three matrices U, S, and V^T. We can find the SVD decomposition of X^T*X using the decomposition for X as follows:

$$X^T X = \left(U S V^T\right)^T U S V^T = \left(V S^T U^T\right) U S V^T = V S^2 V^T$$

This means two important things for us:

- The matrix V^T, often referred to as the *right singular vectors* of X, is equivalent to the *eigenvectors* of $X^T X$.
- S^2 is equivalent to the *eigenvalues* of $X^T X$.

So the first n-principal components are obtained by projecting X by the first n vectors from V^T. Similarly, S^2 gives a measure of the variance retained.

2.1 Implementation [10 pts] **[P]**

Implement PCA. In the pca.py file, complete the following functions:

- fit: You may use np.linalg.svd. Set full_matrices=False. Hint 1 may be useful.
- transform
- transform_rv: You may find np.cumsum helpful for this function.

Assume a dataset is composed of N datapoints, each of which has D features with D < N. The dimension of our data would be D. However, it is possible that many of these dimensions

contain redundant information. Each feature explains part of the variance in our dataset, and some features may explain more variance than others.

HINT 1: Make sure you remember to first center your data by subtracting the mean of each feature.

2.1.1 Local Tests for PCA [No Points]

You may test your implementation of the functions contained in **pca.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
###############################
### DO NOT CHANGE THIS CELL ###
###############################

from utilities.localtests import TestPCA

unittest_pca = TestPCA()
unittest_pca.test_pca()
unittest_pca.test_transform()
unittest_pca.test_transform_rv()
```

```
UnitTest passed successfully for "PCA fit"!
UnitTest passed successfully for "PCA transform"!
UnitTest passed successfully for "PCA transform with recovered
variance"!
```

2.2 Visualize [5 pts] **[W]**

PCA is used to transform multivariate data tables into smaller sets so as to observe the hidden trends and variations in the data. It can also be used as a feature extractor for images. Here you will visualize two datasets using PCA, first is the iris dataset and then a dataset of masked and unmasked images.

In the **pca.py**, complete the following function:

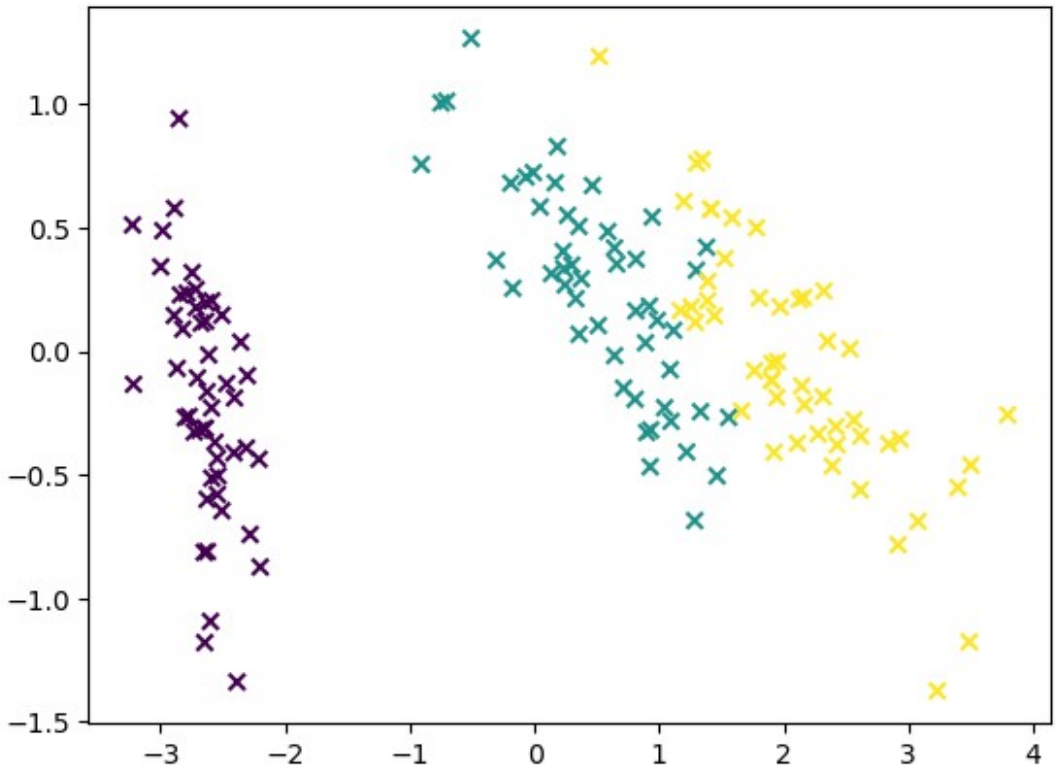
- **visualize**: Use your implementation of PCA and reduce the datasets such that they contain only two features. Using Plotly's Express make a 2d and 3d scatterplot of the data points using these features. Make sure to differentiate the data points according to their true labels using color. We recommend converting the data to a pandas dataframe before plotting.

The datasets have already been loaded for you in the subsequent cells.

NOTE: Here, we won't be testing for accuracy. Even with correct implementations of PCA, the accuracy can differ from the TA solution. That is fine as long as the visualizations come out similar.

Iris Dataset

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Use PCA for visualization of iris dataset

from pca import PCA

iris_data = load_iris(return_X_y=True)

X = iris_data[0]
y = iris_data[1]

fig_title = "Iris Dataset with Dimensionality Reduction"
PCA().visualize(X, y, fig_title)
```



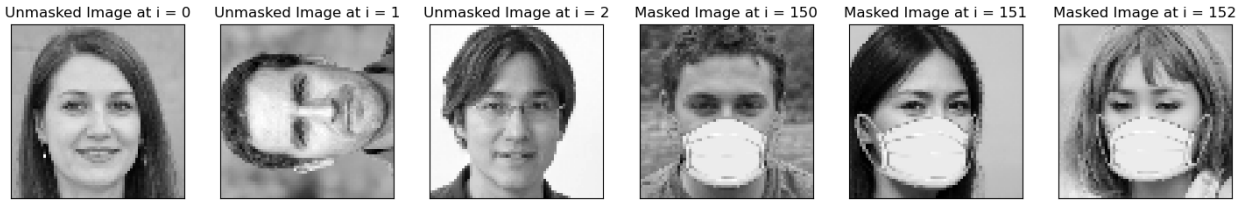
2.3 PCA Reduced Facemask Dataset Analysis [5 pts] **[W]**

Facemask Dataset

The masked and unmasked dataset is made up of grayscale images of human faces facing forward. Half of these images are faces that are completely unmasked, and the remaining images show half of the face covered with an artificially generated face mask. The images have already been preprocessed, they are also reduced to a small size of 64x64 pixels and then

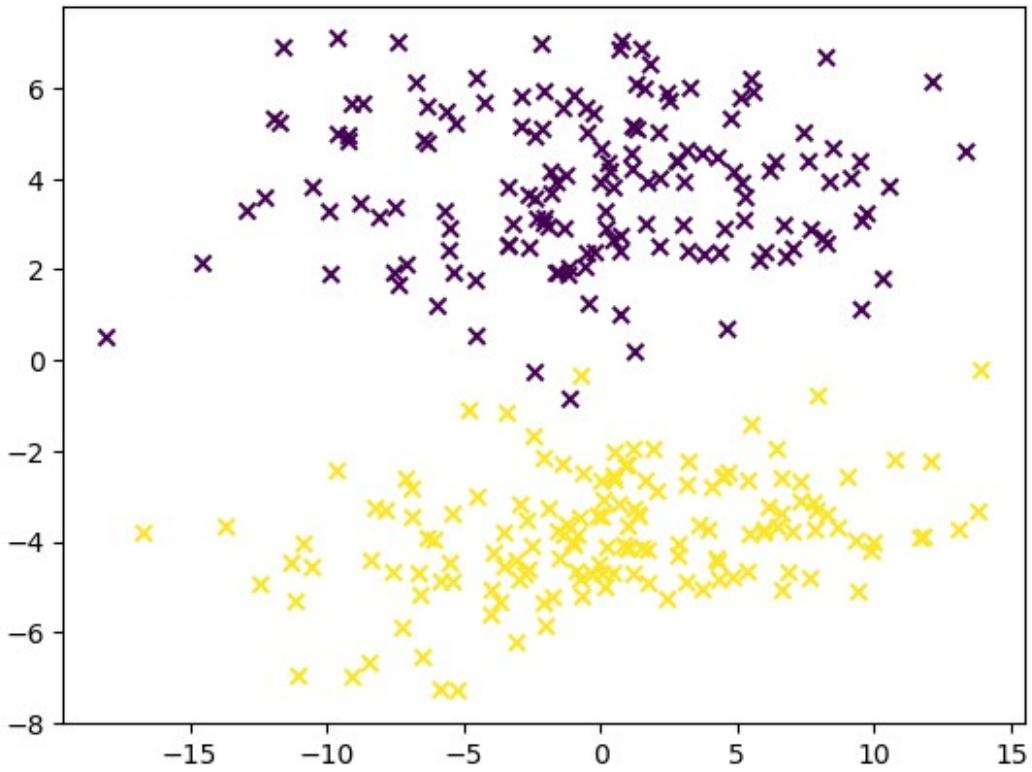
reshaped into a feature vector of 4096 pixels. Below is a sample of some of the images in the dataset.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

X = np.load("./data/smallflat_64.npy")
y = np.load("./data/masked_labels.npy").astype("int")
i = 0
fig = plt.figure(figsize=(18, 18))
for idx in [0, 1, 2, 150, 151, 152]:
    ax = fig.add_subplot(6, 6, i + 1, xticks=[], yticks=[])
    image = (
        np.rot90(X[idx].reshape(64, 64), k=1)
        if idx % 2 == 1 and idx < 150
        else X[idx].reshape(64, 64)
    )
    m_status = "Unmasked" if idx < 150 else "Masked"
    ax.imshow(image, cmap="gray")
    ax.set_title(f"{m_status} Image at i = {idx}")
    i += 1
```




```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Use PCA for visualization of masked and unmasked images

X = np.load("./data/smallflat_64.npy")
y = np.load("./data/masked_labels.npy")

fig_title = "Facemask Dataset Visualization with Dimensionality
Reduction"
PCA().visualize(X, y, fig_title)


print(
    "*In this plot, the 0 points are unmasked images and the 1 points
are masked images."
)
```

```
*In this plot, the 0 points are unmasked images and the 1 points are
masked images.
```



What do you think of this 2 dimensional plot, knowing that the original dataset was originally a set of flattened image vectors that had 4096 pixels/features?.

1. Look at the 2-dimensional plot above. If the *facemask* dataset that has been reduced to 2 features was fed into a classifier, do you think the classifier would produce high accuracy or low accuracy in comparison to the original dataset which had 4096 pixels/features? Why? You can refer to the 2D visualization made above (One or two sentences will suffice for this question) **(3 pts)**

   Answer Feeding the facemask dataset with 2 reduced features into a classifier is likely to result in lower accuracy compared to the original dataset with 4096 features because the reduced dataset may lose important information and details necessary for accurate classification.

2. Assuming an equal rate of accuracy, what do you think is the main advantage in feeding a classifier a dataset with 2 features vs a dataset with 4096 features? (One sentence will suffice for this question.) **(2 pts)**

   Answer Feeding a classifier a dataset with 2 features is advantageous due to reduced dimensionality, which can lead to faster training and lower computational costs, while a dataset with 4096 features may have a higher potential for capturing complex patterns but at the cost of increased computational requirements and potential overfitting.

2.4 PCA Exploration [No Points]

Note The accuracy can differ from the TA solution and this section is not graded.

Emotion Dataset [No Points]

Now you will use PCA on an actual real-world dataset. We will use your implementation of PCA function to reduce the dataset with 99% retained variance and use it to obtain the reduced features. On the reduced dataset, we will use logistic and linear regression to compare results between PCA and non-PCA datasets. Run the following cells to see how PCA works on regression and classification tasks.

The first dataset we will use is an emotion dataset made up of grayscale images of human faces faces that are visibly happy and visibly sad. Note how Accuracy increases after reducing the number of features used.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

X = np.load("./data/emotion_features.npy")
y = np.load("./data/emotion_labels.npy").astype("int")
i = 0
fig = plt.figure(figsize=(18, 18))
for idx in [0, 1, 2, 150, 151, 152]:
    ax = fig.add_subplot(6, 6, i + 1, xticks=[], yticks=[])
    image = (
        np.rot90(X[idx].reshape(64, 64), k=1)
        if idx % 2 == 1 and idx < 150
        else X[idx].reshape(64, 64)
    )
    m_status = "Unmasked" if idx < 150 else "Masked"
    ax.imshow(image, cmap="gray")
    m_status = "Sad" if idx < 150 else "Happy"
    ax.set_title(f"{m_status} Image at i = {idx}")
    i += 1
```


```
################################
### DO NOT CHANGE THIS CELL ###
################################

X = np.load("./data/emotion_features.npy")
y = np.load("./data/emotion_labels.npy").astype("int")
```

```
print("Not Graded - Data shape before PCA ", X.shape)

pca = PCA()
pca.fit(X)

X_pca = pca.transform_rv(X, retained_variance=0.99)

print("Not Graded - Data shape with PCA ", X_pca.shape)
```

```
Not Graded - Data shape before PCA  (600, 4096)

Not Graded - Data shape with PCA  (600, 150)
```

```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Train, test splits
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)

# Use logistic regression to predict classes for test set
clf = LogisticRegression()
clf.fit(X_train, y_train)
preds = clf.predict_proba(X_test)
print(
    "Not Graded - Accuracy before PCA: {:.5f}".format(
        accuracy_score(y_test, preds.argmax(axis=1))
    )
)
```

```
Not Graded - Accuracy before PCA: 0.95000
```

```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Train, test splits
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.3, stratify=y, random_state=42
)

# Use logistic regression to predict classes for test set
clf = LogisticRegression()
clf.fit(X_train, y_train)
preds = clf.predict_proba(X_test)
print(
    "Not Graded - Accuracy after PCA: {:.5f}".format(
        accuracy_score(y_test, preds.argmax(axis=1))
    )
)
```

```
Not Graded - Accuracy after PCA: 0.95556
```

Now we will explore sklearn's Diabetes dataset using PCA dimensionality reduction and regression. Notice the RMSE score reduction after we apply PCA.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
from sklearn.linear_model import RidgeCV


def apply_regression(X_train, y_train, X_test):
    ridge = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1])
    clf = ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)

    return y_pred

################################
### DO NOT CHANGE THIS CELL ###
################################
# load the dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

print(X.shape, y.shape)

pca = PCA()
pca.fit(X)

X_pca = pca.transform_rv(X, retained_variance=0.9)
print("Not Graded - data shape with PCA ", X_pca.shape)
```

```
(442, 10) (442,)
Not Graded - data shape with PCA  (442, 7)
```

```
################################
### DO NOT CHANGE THIS CELL ###
################################
# Train, test splits
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Ridge regression without PCA
y_pred = apply_regression(X_train, y_train, X_test)

# calculate RMSE
rmse_score = np.sqrt(mean_squared_error(y_pred, y_test))
```

```
print(
    "Not Graded - RMSE score using Ridge Regression before PCA:
{:.5}".format(
        rmse_score
    )
)
```

```
Not Graded - RMSE score using Ridge Regression before PCA: 53.101
```

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################
# Ridge regression with PCA
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.3, random_state=42
)

# use Ridge Regression for getting predicted labels
y_pred = apply_regression(X_train, y_train, X_test)

# calculate RMSE
rmse_score = np.sqrt(mean_squared_error(y_pred, y_test))
print(
    "Not Graded - RMSE score using Ridge Regression after PCA:
{:.5}".format(rmse_score)
)
```

```
Not Graded - RMSE score using Ridge Regression after PCA: 53.024
```

Q3 Polynomial regression and regularization [80pts: 50pts + 20pts Bonus for Undergrads + 10pts Bonus for All] **[P]** | **[W]**

3.1 Regression and regularization implementations [50pts: 30 pts + 20 pts bonus for CS 4641] **[P]**

We have three methods to fit linear and ridge regression models: 1) closed form solution; 2) gradient descent (GD); 3) stochastic gradient descent (SGD). Some of the functions are bonus, see the below function list on what is required to be implemented for graduate and undergraduate students. We use the term weight in the following code. Weights and parameters (θ) have the same meaning here. We used parameters (θ) in the lecture slides.

In the regression.py file, complete the Regression class by implementing the listed functions below. We have provided the Loss function, L, associated with the GD and SGD function for Linear and Ridge Regression for deriving the gradient update.

- **rmse**

- **construct_polynomial_feats**
- **predict**
- **linear_fit_closed**: You should use np.linalg.pinv in this function
- **linear_fit_GD** (bonus for undergrad, **required for grad**):

$$L*\text{linear, GD}(\theta)=\frac{1}{2N}\sum*i=0^{N}\left[y_i-\hat{y}_i(\theta)\right)^2 y_i=\text{label}, \hat{y}_i(\theta)=\text{prediction}$$

- **linear_fit_SGD** (bonus for undergrad, **required for grad**):

$$L_{\text{linear, SGD}}(\theta)=\frac{1}{2}\left[y_i-\hat{y}_i(\theta)\right)^2 y_i=\text{label}, \hat{y}_i(\theta)=\text{prediction}$$

- **ridge_fit_closed**: You should adjust your I matrix to handle the bias term differently than the rest of the terms
- **ridge_fit_GD** (bonus for undergrad, **required for grad**):

$$L*\text{ridge, GD}(\theta)=L*\text{linear, GD}(\theta)+\frac{c_\lambda}{2N}\theta^T\theta$$

- **ridge_fit_SGD** (bonus for undergrad, **required for grad**):

$$L*\text{ridge, SGD}(\theta)=L*\text{linear, SGD}(\theta)+\frac{c_\lambda}{2N}\theta^T\theta$$

- **ridge_cross_validation**: Use `ridge_fit_closed` for this function

IMPORTANT NOTE:

- Use your RMSE function to calculate actual loss when coding GD and SGD, but use the loss listed above to derive the gradient update.
- In **ridge_fit_GD** and **ridge_fit_SGD**, you should avoid applying regularization to the bias term in the gradient update.

The points for each function is in the **Deliverables and Points Distribution** section.

3.1.1 Local Tests for Helper Regression Functions [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestRegression
```

```
unittest_reg = TestRegression()
unittest_reg.test_rmse()
unittest_reg.test_construct_polynomial_feats()
unittest_reg.test_predict()
```

```
UnitTest passed successfully for "RMSE"!
UnitTest passed successfully for "Polynomial feature construction"!
UnitTest passed successfully for "Linear regression prediction"!
```

3.1.2 Local Tests for Linear Regression Functions [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestRegression

unittest_reg = TestRegression()
unittest_reg.test_linear_fit_closed()
```

```
[[ 3.11428866e+06]
 [-7.09956538e+07]
 [ 1.18326090e+08]
 [ 9.27018941e+07]
 [-6.87882565e+06]
 [-1.35395305e+08]]
UnitTest passed successfully for "Closed form linear regression"!
```

3.1.3 Local Tests for Ridge Regression Functions [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestRegression

unittest_reg = TestRegression()
unittest_reg.test_ridge_fit_closed()
unittest_reg.test_ridge_cross_validation()
```

```
UnitTest passed successfully for "Closed form ridge regression"!
UnitTest passed successfully for "Ridge regression cross validation"!
```

3.1.4 Local Tests for Gradient Descent and SGD (Bonus for Undergrad Tests) [No Points]

You may test your implementation of the functions contained in **regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestRegression

unittest_reg = TestRegression()
#unittest_reg.test_linear_fit_GD()
unittest_reg.test_linear_fit_SGD()
#unittest_reg.test_ridge_fit_GD()
#unittest_reg.test_ridge_fit_SGD()
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call
last)
c:\Users\leque\Downloads\course_files_export\HW3\hw3_code\
FALL2023_HW3_Student.ipynb Cell 56 line 9
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y106sZmlsZQ%3D%3D?
line=6'>7</a> unittest_reg = TestRegression()
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y106sZmlsZQ%3D%3D?
line=7'>8</a> #unittest_reg.test_linear_fit_GD()
----> <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y106sZmlsZQ%3D%3D?
line=8'>9</a> unittest_reg.test_linear_fit_SGD()

File c:\Users\leque\Downloads\course_files_export\HW3\hw3_code\
utilities\localtests.py:514, in
TestRegression.test_linear_fit_SGD(self)
    512 lsgd_test = np.allclose(linear_SGD, test_reg.linear_SGD)
    513 lsgd_loss_test = np.allclose(linear_SGD_loss,
test_reg.linear_SGD_loss)
--> 514 self.assertTrue(lsgd_test, "Weights are incorrect")
    515 self.assertTrue(lsgd_loss_test, "Loss is incorrect")
```

```
    517 success_msg = "Stochastic gradient descent linear regression"

File c:\Users\leque\miniforge3\envs\ml_hw3\Lib\unittest\case.py:715,
in TestCase.assertTrue(self, expr, msg)
    713 if not expr:
    714     msg = self._formatMessage(msg, "%s is not true" %
safe_repr(expr))
--> 715     raise self.failureException(msg)

AssertionError: False is not true : Weights are incorrect
```

3.2 About RMSE [3 pts] **[W]**

What is a good RMSE value?

If we normalize our labels such that the true labels y and the model outputs $\hat{y}$ can only be between 0 and 1, what does it mean when the RMSE = 1? Please provide an example with your explanation.

Answer A good RMSE (Root Mean Square Error) value is one that is as low as possible. In general, a lower RMSE indicates a better fit of the model to the data.

If the RMSE is 1 when both the true labels (y) and model outputs (ŷ) are normalized to the range [0, 1], it means that, on average, the model's predictions are off by 1 unit within the normalized scale. In this context, it's important to consider the range and scale of the data.

For example, let's say we're predicting the prices of houses on a scale of 0 to 1, where 0 represents the lowest possible price, and 1 represents the highest possible price. An RMSE of 1 in this case means that, on average, the model's predictions are off by the equivalent of the entire price range. This is not very accurate and suggests that the model's predictions have a significant error relative to the scale of the problem. If the true price of a house is 0.6 (60% of the highest price), the model might predict 0.7, on average. This indicates that the model is typically making predictions that are 10% higher (0.7 - 0.6 = 0.1) than the true values in terms of the normalized scale.

3.3 Testing: General Functions and Linear Regression [5 pts] **[W]**

In this section. we will test the performance of the linear regression. As long as your test RMSE score is close to the TA's answer (TA's answer ± 0.05), you can get full points. Let's first construct a dataset for polynomial regression.

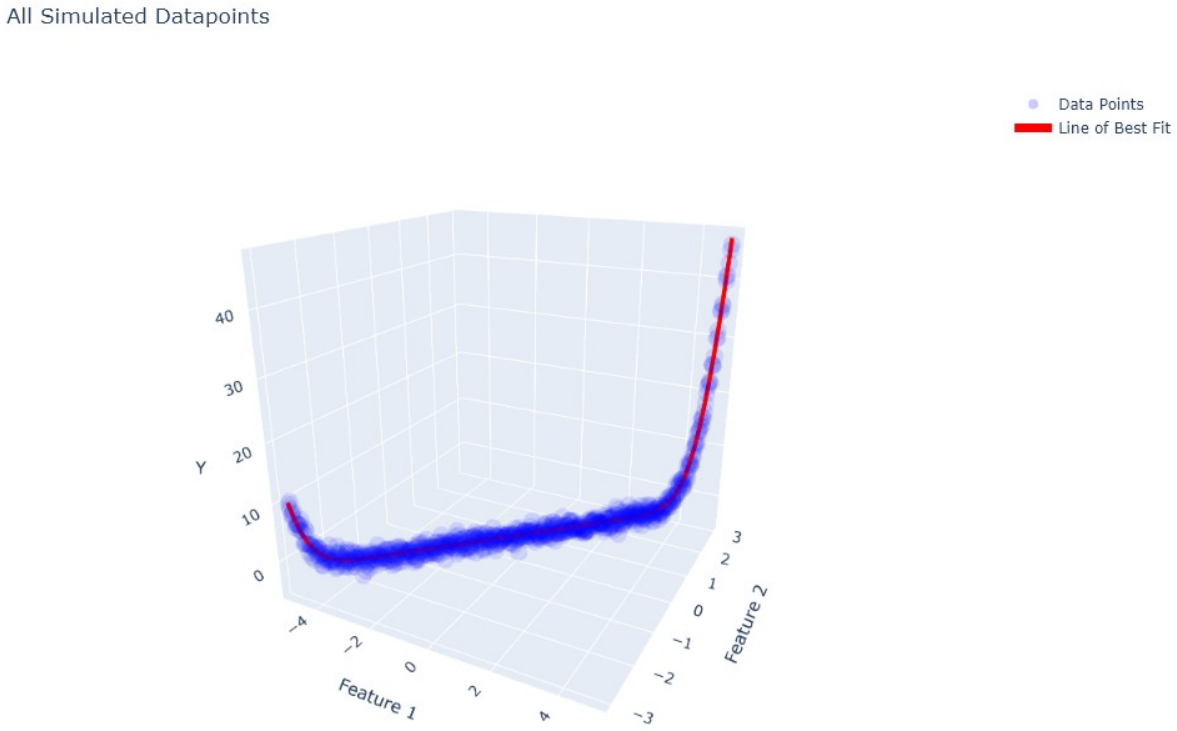
In this case, we construct the polynomial features up to degree 5. Each data sample consists of two features $[a, b)$. We compute the polynomial features of both a and b in order to yield the vectors $\left[1, a, a^2, a^3, \ldots, a^{\text{degree}}\right)$ and $\left[1, b, b^2, b^3, \ldots, b^{\text{degree}}\right)$. We train our model with the cartesian product of these polynomial features. The cartesian product generates a new feature vector consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.

For example, if degree = 2, we will have the polynomial features $\left[1, a, a^2\right)$ and $\left[1, b, b^2\right)$ for the datapoint $\left[a, b\right)$. The cartesian product of these two vectors will be $\left[1, a, b, a\,b, a^2, b^2\right)$. We do not generate a^3 and b^3 since their degree is greater than 2 (specified degree).

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

from regression import Regression
from plotter import Plotter

##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Generate a sample regression dataset with polynomial features
# using the student's regression implementation.

POLY_DEGREE = 8

reg = Regression()
plotter = Plotter(
    regularization=reg,
    poly_degree=POLY_DEGREE,
    student_version=STUDENT_VERSION,
    eo_params=(EO_TEXT, EO_FONT, EO_COLOR, EO_ALPHA, EO_SIZE, EO_ROT),
)

x_all, y_all, p, x_all_feat, x_cart_flat = plotter.create_data()
```

```
x_all: 700 (rows/samples) 2 (columns/features)
y_all: 700 (rows/samples) 1 (columns/features)
```

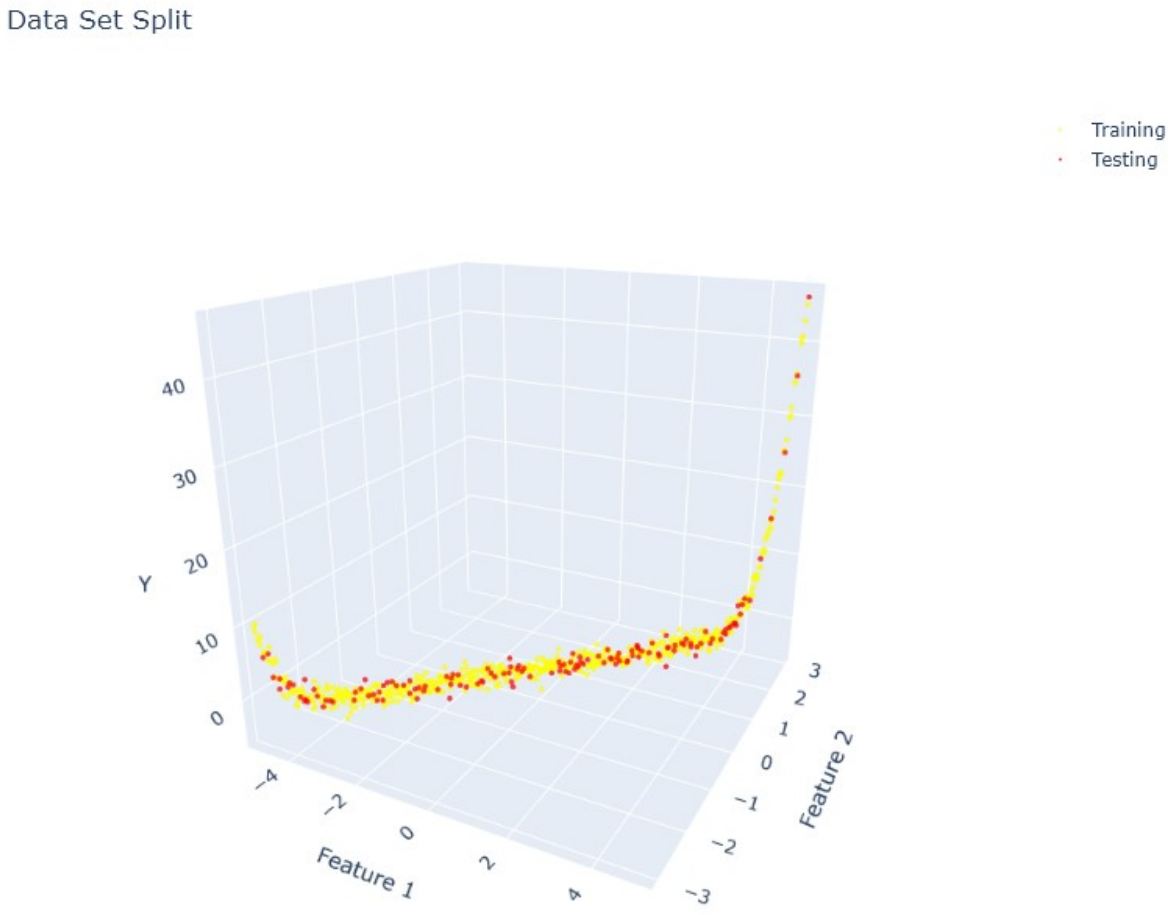
```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

# Visualize simulated regression data

plotter.plot_all_data(x_all, y_all, p)
```



In the figure above, the red curve is the true fuction we want to learn, while the blue dots are the noisy data points. The data points are generated by $Y = X\theta + \epsilon$, where $\epsilon_i \sim N(0, 1)$ are i.i.d. generated noise.

Now let's split the data into two parts, the training set and testing set. The yellow dots are for training, while the black dots are for testing.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

xtrain, ytrain, xtest, ytest, train_indices, test_indices =
plotter.split_data(
    x_all, y_all
)

plotter.plot_split_data(xtrain, xtest, ytrain, ytest)
```



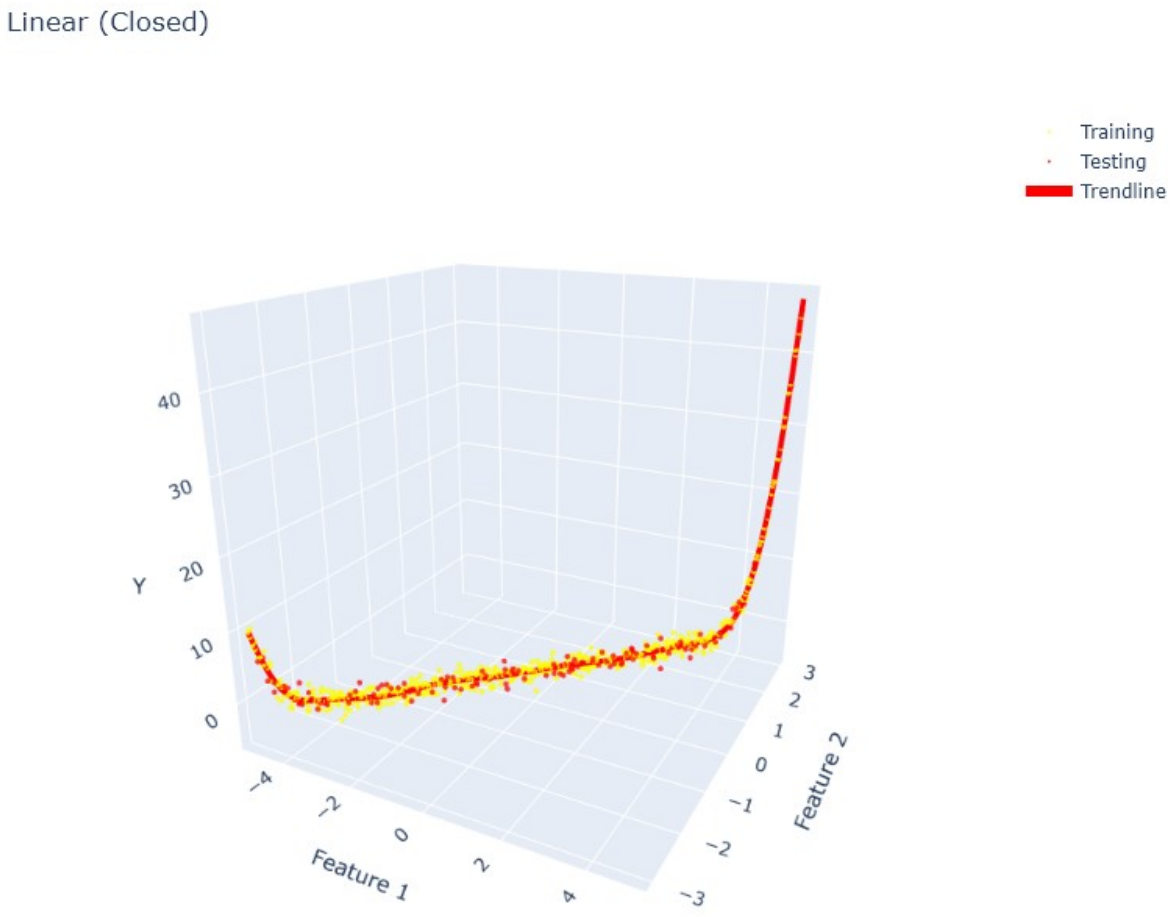
Now let us train our model using the training set and see how our model performs on the testing set. Observe the red line, which is our model's learned function.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for both Grad and Undergrad

weight = reg.linear_fit_closed(x_all_feat[train_indices],
y_all[train_indices])
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
y_pred = reg.predict(x_all_feat, weight)
print("Linear (closed) RMSE: %.4f" % test_rmse)
```

```
plotter.plot_linear_closed(xtrain, xtest, ytrain, ytest, x_all,
y_pred)
```

```
[[ 1.72223512e+04]
 [ 6.81847430e+05]
 [ 6.58368678e+04]
 [-1.15881468e+06]
 [ 9.84668926e+03]
 [ 3.96338682e+05]
 [-2.80543488e+04]
 [-6.63236359e+04]
 [ 4.15997430e+05]
 [ 4.63910911e+04]
 [-6.88399839e+05]
 [ 1.27969835e+04]
 [ 2.44692180e+05]
 [-9.94363863e+03]
 [-3.29052105e+04]
 [ 3.07065567e+03]
 [ 3.47236250e+04]
 [-4.06150933e+05]
 [ 1.45671607e+04]
 [ 1.53704279e+05]
 [ 9.22787355e+02]
 [-1.28541559e+04]
 [ 8.73136397e+03]
 [ 1.24366482e+03]
 [-2.36801589e+05]
 [ 1.56292672e+04]
 [ 9.91115379e+04]
 [ 7.44264300e+03]
 [-8.23522975e+02]
 [ 1.21277890e+04]
 [ 7.63516946e+03]
 [-9.76617582e+03]
 [ 1.62665308e+04]
 [ 6.63558932e+04]
 [ 1.13545564e+04]
 [ 6.39485683e+03]
 [ 1.41656440e+04]
 [ 1.14700723e+04]
 [ 1.02926514e+03]
 [-8.08593895e+03]
 [ 4.67025066e+04]
 [ 1.37017045e+04]
 [ 1.07258847e+04]
 [ 1.53883571e+04]
 [ 1.37710140e+04]
 [ 7.50652955e+03]
 [ 2.03740722e+03]
```

```
 [-5.28589548e+03]
 [ 1.51099932e+04]
 [ 1.33245014e+04]
 [ 1.61219848e+04]
 [ 1.51515789e+04]
 [ 1.13928883e+04]
 [ 8.11141493e+03]
 [ 3.71743305e+03]
 [-3.00373161e+03]
 [ 1.48836714e+04]
 [ 1.65621614e+04]
 [ 1.59799180e+04]
 [ 1.37247037e+04]
 [ 1.17558196e+04]
 [ 9.11943044e+03]
 [ 5.08673160e+03]
 [-7.95331161e+00]
 [-6.36385826e+03]
 [ 1.68262675e+04]]
Linear (closed) RMSE: 1.0072
```



HINT: If your RMSE is off, make sure to follow the instruction given for `linear_fit_closed` in the list of functions to implement above.

Now let's use our linear gradient descent function with the same setup. Observe that the trendline is now less optimal, and our RMSE increased. Do not be alarmed.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only
# This cell may take more than 1 minute
weight, _ = reg.linear_fit_GD(
    x_all_feat[train_indices], y_all[train_indices], epochs=50000,
learning_rate=1e-8
)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
```

```
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
print("Linear (GD) RMSE: %.4f" % test_rmse)
y_pred = reg.predict(x_all_feat, weight)
y_pred = np.reshape(y_pred, (y_pred.size,))

plotter.plot_linear_gd(xtrain, xtest, ytrain, ytest, x_all, y_pred)
```

We must tune our epochs and learning_rate. As we tune these parameters our trendline will approach the trendline generated by the linear closed form solution. Observe how we slowly tune (increase) the epochs and learning_rate below to create a better model.

Note that the closed form solution will always give the most optimal/overfit results. We cannot outperform the closed form solution with GD. We can only approach closed forms level of optimality/overfitness. We leave the reasoning behind this as an exercise to the reader.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only
# This cell may take more than 1 minute

learning_rates = [1e-8, 1e-6, 1e-4]
weights = np.zeros((3, POLY_DEGREE**2 + 2))

for ii in range(len(learning_rates)):
    weights[ii, :] = reg.linear_fit_GD(
        x_all_feat[train_indices],
        y_all[train_indices],
        epochs=50000,
        learning_rate=learning_rates[ii],
    )[0].ravel()
    y_test_pred = reg.predict(
        x_all_feat[test_indices],
weights[ii, :].reshape((POLY_DEGREE**2 + 2, 1))
    )
    test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
    print("Linear (GD) RMSE: %.4f (learning_rate=%s)" % (test_rmse,
learning_rates[ii]))

plotter.plot_linear_gd_tuninglr(
    xtrain, xtest, ytrain, ytest, x_all, x_all_feat, learning_rates,
weights
)
```

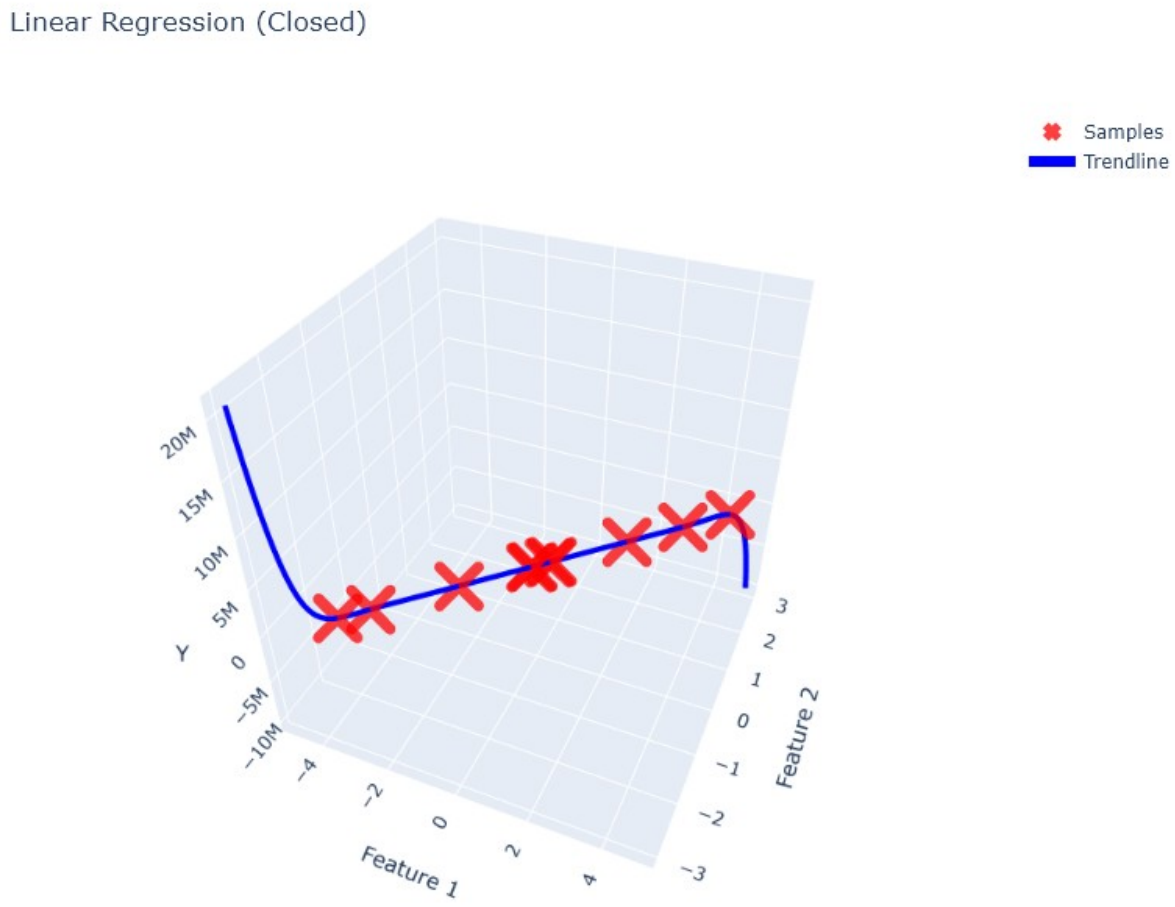
And what if we just use the first 10 data points to train?

Linear Closed 10 Samples

```
################################
### DO NOT CHANGE THIS CELL ###
################################
rng = np.random.RandomState(seed=3)
y_all_noisy = np.dot(x_cart_flat, np.zeros((POLY_DEGREE**2 + 2, 1))) +
rng.randn(
    x_all_feat.shape[0], 1
)
sub_train = train_indices[10:20]

################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for both Grad and Undergrad

weight = reg.linear_fit_closed(x_all_feat[sub_train],
y_all_noisy[sub_train])
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print("Linear (closed) 10 Samples RMSE: %.4f" % test_rmse)

plotter.plot_linear_closed_10samples(x_all, y_all_noisy, sub_train,
y_pred)
```

```
[[-6.16672873e+07]
 [ 2.61522528e+05]
 [-3.19695185e+08]
 [ 8.75762498e+08]
 [ 2.51218061e+08]
 [ 6.75412852e+07]
 [ 2.39480880e+07]
 [-4.63322856e+07]
 [-2.45099894e+07]
 [-2.16484203e+08]
 [ 5.00790447e+08]
 [ 1.26063783e+08]
 [ 1.58577170e+07]
 [-1.02981991e+07]
 [-5.24664241e+07]
 [ 9.71905036e+06]
 [-1.54557568e+08]
 [ 2.75807217e+08]
 [ 5.09712255e+07]
 [-1.51524114e+07]
 [-3.08459665e+07]
 [-5.61469000e+07]
 [-1.88356202e+07]]
```

```
 [-1.95812923e+07]
 [ 1.40817282e+08]
 [ 5.91568883e+06]
 [-3.37585044e+07]
 [-4.31746320e+07]
 [-5.83551922e+07]
 [-3.59684210e+07]
 [-3.64158288e+07]
 [ 1.25380284e+07]
 [-2.11176346e+07]
 [-4.49221457e+07]
 [-5.05718343e+07]
 [-5.96801752e+07]
 [-4.62481016e+07]
 [-4.65165456e+07]
 [-1.71442352e+07]
 [ 5.84927832e+07]
 [-5.16203326e+07]
 [-5.50101386e+07]
 [-6.04751548e+07]
 [-5.24159150e+07]
 [-5.25769714e+07]
 [-3.49535910e+07]
 [ 1.04286236e+07]
 [ 2.16351041e+07]
 [-5.76731291e+07]
 [-6.09521399e+07]
 [-5.61165988e+07]
 [-5.62132387e+07]
 [-4.56392003e+07]
 [-1.84098742e+07]
 [-1.16859997e+07]
 [ 6.76400998e+06]
 [-6.12383278e+07]
 [-5.83370067e+07]
 [-5.83949931e+07]
 [-5.20505707e+07]
 [-3.57129806e+07]
 [-3.16786421e+07]
 [-2.06086540e+07]
 [ 3.63914507e+04]
 [ 5.73101659e+07]
 [-5.96692485e+07]]
Linear (closed) 10 Samples RMSE: 2207393.1849
```



Did you see a worse performance? Let's take a closer look at what we have learned.

3.4 Testing: Testing ridge regression [5 pts] **[W]**

3.4.1 [3pts] **[W]**

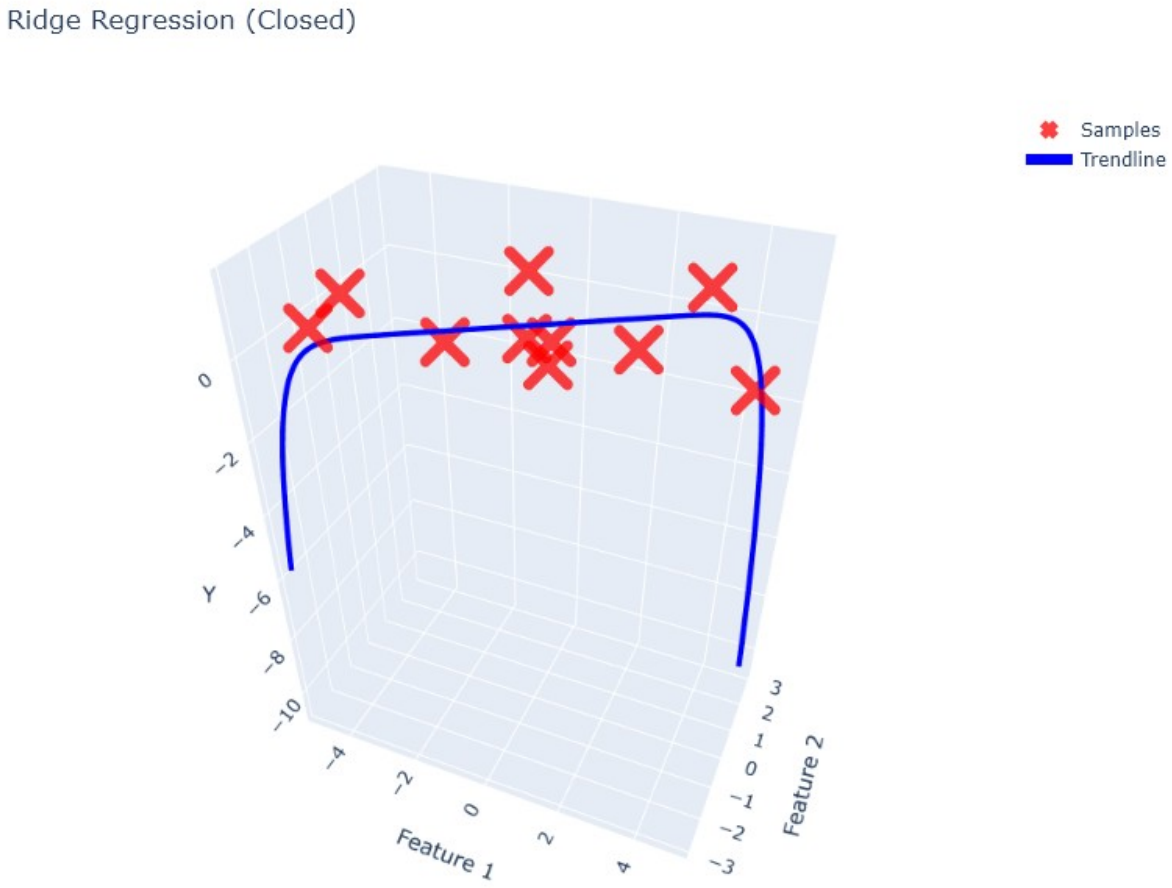
Now let's try ridge regression. Like before, undergraduate students need to implement the closed form, and graduate students need to implement all three methods. We will call the prediction function from linear regression part. As long as your test RMSE score is close to the TA's answer (TA's answer ± 0.05), you can get full points.

Again, let's see what we have learned. **You only need to run the cell corresponding to your specific implementation.**

```
################################
### DO NOT CHANGE THIS CELL ###
################################
```

```
# Required for both Grad and Undergrad
weight = reg.ridge_fit_closed(
    x_all_feat[sub_train], y_all_noisy[sub_train], c_lambda=10
)
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print("Ridge Regression (closed) RMSE: %.4f" % test_rmse)

plotter.plot_ridge_closed_10samples(x_all, y_all_noisy, sub_train,
y_pred)
```

```
Ridge Regression (closed) RMSE: 1.8283
```



HINT: Make sure to follow the instruction given for `ridge_fit_closed` in the list of functions to implement above.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only

weight, _ = reg.ridge_fit_GD(
    x_all_feat[sub_train], y_all_noisy[sub_train], c_lambda=20,
learning_rate=1e-5
)
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print("Ridge Regression (GD) RMSE: %.4f" % test_rmse)

plotter.plot_ridge_gd_10samples(x_all, y_all_noisy, sub_train, y_pred)
```

```
---------------------------------------------------------------------
-----
NotImplementedError                       Traceback (most recent call
last)
c:\Users\leque\Downloads\course_files_export\HW3\hw3_code\
FALL2023_HW3_Student.ipynb Cell 80 line 7
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=0'>1</a> ################################
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=1'>2</a> ### DO NOT CHANGE THIS CELL ###
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=2'>3</a> ################################
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=3'>4</a>
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=4'>5</a> # Required for Grad Only
----> <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=6'>7</a> weight, _ = reg.ridge_fit_GD(
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
```

```
line=7'>8</a>     x_all_feat[sub_train], y_all_noisy[sub_train],
c_lambda=20, learning_rate=1e-5
      <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=8'>9</a> )
     <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=9'>10</a> y_pred = reg.predict(x_all_feat, weight)
     <a
href='vscode-notebook-cell:/c%3A/Users/leque/Downloads/course_files_ex
port/HW3/hw3_code/FALL2023_HW3_Student.ipynb#Y142sZmlsZQ%3D%3D?
line=10'>11</a> y_test_pred = reg.predict(x_all_feat[test_indices],
weight)

File c:\Users\leque\Downloads\course_files_export\HW3\hw3_code\
regression.py:289, in Regression.ridge_fit_GD(self, xtrain, ytrain,
c_lambda, epochs, learning_rate)
    262 def ridge_fit_GD(
    263     self,
    264     xtrain: np.ndarray,
   (...)
    268     learning_rate: float = 1e-7,
    269 ) -> Tuple[np.ndarray, List[float]]:  # [5pts]
    270     """
    271     Fit a ridge regression model using gradient descent.
    272     Although there are many valid initializations, to pass the
local tests
   (...)
    287         - You should avoid applying regularization to the bias
term in the gradient update
    288     """
--> 289     raise NotImplementedError

NotImplementedError:
```

```
################################
### DO NOT CHANGE THIS CELL ###
################################

# Required for Grad Only

weight, _ = reg.ridge_fit_SGD(
    x_all_feat[sub_train], y_all_noisy[sub_train], c_lambda=20,
learning_rate=1e-5
)
y_pred = reg.predict(x_all_feat, weight)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
```

```
print("Ridge Regression (SGD) RMSE: %.4f" % test_rmse)

plotter.plot_ridge_sgd_10samples(x_all, y_all, sub_train, y_pred)
```

Linear vs. Ridge Regression

Regression technique comparison

Analyze the difference in performance between the linear and ridge regression methods given the output RMSE **from the testing on 10 samples** and their corresponding approximation plots.

3.4.2 Why does ridge regression achieve a lower RMSE than linear regression on 10 sample points? [1pts] **[W]**

3.4.3 Describe and contrast two scenarios (real life applications): One where linear is more suitable than ridge, and one in which ridge is better choice than linear. Explain why. [1 pts] **[W]**

3.4.4 [Bonus pts] What is the impact of having some highly correlated features on the data set in terms of linear algebra? Mathematically explain (include expressions) how ridge has an advantage on this in comparison to linear regression. Include the idea of numerical stability. [2pts Bonus For All] **[W]**

3.4.1. **Answer** The RSME for linear is so much larger compared to the RSME for ridge, however, the treadline for linear captures the samples better than ridge.

3.4.2. **Answer**

Ridge regression can achieve a lower RMSE than linear regression on a small dataset with 10 sample points because it is designed to address the problem of overfitting, which can occur when the number of features (or parameters) exceeds the number of data points.

3.4.3. **Answer**

Predicting the Weight of Objects on a Kitchen Scale

Linear regression is more suitable than ridge regression. Linear regression works well when there is a simple and direct relationship between the input features and the target variable, and overfitting is not a significant concern. In the context of predicting the weight of objects on a kitchen scale, the relationship is straightforward: the more an object weighs, the further the needle on the scale moves. This relationship is linear and doesn't involve many complex interactions or high-dimensionality features.

Predicting Housing Prices in a Metropolitan Area

Ridge regression is a better choice than linear regression. Ridge regression is particularly useful when dealing with high-dimensional data or when there are potentially collinear features. Predicting housing prices in a metropolitan area often involves numerous features like square footage, number of bedrooms, location, proximity to amenities, and more. Many of these features can be correlated, and the dataset may include a large number of variables.

3.4.4. **Answer**

When we have highly correlated features, it often leads to multicollinearity in the design matrix X. In the context of linear regression, the solution involves the inversion of the matrix $X^T X$ (where X^T is the transpose of X). The problem arises when $X^T X$ is close to being singular, making its inverse unstable and prone to large numerical errors. Ridge regression addresses this issue by adding a regularization term $\left(X^T X + \lambda I\right)^{-1}$
I is the identity matrix, while λ controls the strength of the regularization. When λ is greater than zero, it ensures that the matrix X+λI is always invertible, even if $X^T X$ is close to being singular.

3.5 Cross validation [7 pts] **[W]**

Let's use Cross Validation to search for the best value for `c_lambda` in ridge regression.

Imagine we have a dataset of 10 points [1,2,3,4,5,6,7,8,9,10] and we want to do 5-fold cross validation.

- The first iteration we would train with [3,4,5,6,7,8,9,10] and test (validate) with [1,2]
- The second iteration we would train with [1,2,5,6,7,8,9,10] and test (validate) with [3,4]
- The third iteration we would train with [1,2,3,4,7,8,9,10] and test (validate) with [5,6]
- The fourth iteration we would train with [1,2,3,4,5,6,9,10] and test (validate) with [7,8]
- The fifth iteration we would train with [1,2,3,4,5,6,7,8] and test (validate) with [9,10]

We provided a list of possible values for λ, and you will use them in cross validation. For cross validation, use **10-fold** method and only use it for your training data (you already have the `train_indices` to get training data). For the training data, split them in 10 folds which means that use 10 percent of training data for test and 90 percent for training. For each λ, you will have calculated 10 RMSE values. Compute the mean of the 10 RMSE values. Then pick the λ with the lowest mean RMSE.

HINTS:

- `np.concatenate` is your friend
- Make sure to follow the instruction given for `ridge_fit_closed` in the list of functions to implement above.
- To use the 10-fold method, that would include looping over all the data 10 times, where we split a different 10% of the data at every iteration. So the first iteration extracts the first 10% to testing and the remaining 90% for training.The second iteration splits the second 10% of data for testing and the (different) remaining 90% for testing. If we have the array of elements 1 - 10, the second iteration would extract the number "2" because that's in the second 10% of the array.
- The `hyperparameter_search` function will handle averaging the errors, so don't average the errors in `ridge_cross_validation`. We've done this so you can see your error across every fold when using the gradescope tests.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
lambda_list = [0.0001, 0.001, 0.1, 1, 5, 10, 50, 100, 1000, 10000]
kfold = 10
```

```
best_lambda, best_error, error_list = reg.hyperparameter_search(
    x_all_feat[train_indices], y_all[train_indices], lambda_list,
kfold
)
for lm, err in zip(lambda_list, error_list):
    print("Lambda: %.4f" % lm, "RMSE: %.6f" % err)

print("Best Lambda: %.4f" % best_lambda)
weight = reg.ridge_fit_closed(
    x_all_feat[train_indices], y_all_noisy[train_indices],
c_lambda=best_lambda
)
y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all_noisy[test_indices])
print("Best Test RMSE: %.4f" % test_rmse)
```

```
Lambda: 0.0001 RMSE: 0.957365
Lambda: 0.0010 RMSE: 0.955850
Lambda: 0.1000 RMSE: 0.953591
Lambda: 1.0000 RMSE: 0.951137
Lambda: 5.0000 RMSE: 0.949576
Lambda: 10.0000 RMSE: 0.949279
Lambda: 50.0000 RMSE: 0.949647
Lambda: 100.0000 RMSE: 0.951808
Lambda: 1000.0000 RMSE: 1.162351
Lambda: 10000.0000 RMSE: 3.065832
Best Lambda: 10.0000
Best Test RMSE: 1.0463
```

3.6 Noisy Input Samples in Linear Regression [10 pts Bonus for All] **[W]**

Consider a linear model of the form:

$$y(x_n, \theta) = \theta_0 + \sum_{d=1}^{D} \theta_d x_{nd}$$

where $x_n = (x_{n1}, \ldots, x_{nD}) \in R^D$ and weights $\theta = (\theta_0, \ldots, \theta_D) \in R^D$. Given the the D-dimension input sample set $x = \{x_1, \ldots, x_n\}$ with corresponding target value $y = \{y_1, \ldots, y_n\}$, the sum-of-squares error function is:

$$E_D(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left[y(x_n, \theta) - y_n \right]^2$$

Now, suppose that Gaussian noise $\epsilon_n \in R^D$ is added independently to each of the input sample x_n to generate a new sample set $x' = \{x_1 + \epsilon_1, \ldots, x_n + \epsilon_n\}$. Here, ϵ_{ni} (an entry of ϵ_n) has zero mean and variance σ^2. For each sample x_n, let $x_n' = (x_{n1} + \epsilon_{n1}, \ldots, x_{nD} + \epsilon_{nd})$, where n and d is independent across both n and d indices.

1. (3pts) Show that $y(x_n', \theta) = y(x_n, \theta) + \sum_{d=1}^{D} \theta_d \epsilon_{nd}$

2. (7pts) Assume the sum-of-squares error function of the noise sample set $x' = \{x_1 + \epsilon_1, \ldots, x_n + \epsilon_n\}$ is $E_D(\theta)'$. Prove the expectation of $E_D(\theta)'$ is equivalent to the sum-of-squares error $E_D(\theta)$ for noise-free input samples with the addition of a weight-decay regularization term (e.g. ℓ_2 norm) , in which the bias parameter θ_0 is omitted from the regularizer. In other words, show that

$$E\left[E_D(\theta)'\right] = E_D(\theta) + \text{Regularizer}.$$

N.B. You should be incorporating your solution from the first part of this problem into the given sum of squares equation for the second part.

Write your responses below using LaTeX in Markdown.

HINT:

- During the class, we have discussed how to solve for the weight θ for ridge regression, the function looks like this:

  $$E(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[y(x_i, \theta) - y_i\right]^2 + \frac{\lambda}{N} \sum_{i=1}^{d} |\theta_i|^2$$

  where the first term is the sum-of-squares error and the second term is the regularization term. N is the number of samples. In this question, we use another form of the ridge regression, which is:

  $$E(\theta) = \frac{1}{2} \sum_{i=1}^{N} \left[y(x_i, \theta) - y_i\right]^2 + \frac{\lambda}{2} \sum_{i=1}^{d} |\theta_i|^2$$

- For the Gaussian noise ϵ_n, we have $E[\epsilon_n] = 0$

- Assume the noise $\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ are **independent** to each other, we have

  $$E[\epsilon_n \epsilon_m] = \begin{cases} \sigma^2 & ¿ m = n \\ 0 & ¿ m \neq n \end{cases}$$

1. **Answer** ...

$$ y\left(x_n^{\prime}, \theta\right) =\theta_0+\sum_{d=1}^D \theta_d\left(x_{n d}+\epsilon_{n d}\right)\\ =\theta_0+\sum_{d=1}^D \theta_d x_{n d}+\sum_{d=1}^D \theta_d \epsilon_{n d} =y\left(x_n, \theta\right)+\sum_{d=1}^D \theta_d \epsilon_{n d} $$

1. **Answer** ...

$$E_D(\theta)' \qquad \text{¿}\frac{1}{2}\sum_{n=1}^{N} y(x_n,\theta)' - y_n{}^2$$

$$\text{¿} \qquad \text{¿}\frac{1}{2}\sum_{n=1}^{N} y(x_n,\theta) - y_n{}^2 + \sum_{n=1}^{N} y(x_n,\theta) - y_n \sum_{d=1}^{D} \theta_d \epsilon_{nd} + \frac{1}{2}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} \theta_d \epsilon_{nd}\right)^2$$

Since ϵ_n is independent across both n and d indices,

$$E[\epsilon_n] = 0$$

$$E[\epsilon_n \epsilon_m] = \begin{cases} \sigma^2 & m = n \\ 0 & m \neq n \end{cases}$$

$$E\left[\sum_{n=1}^{N} y(x_n,\theta) - y_n \sum_{d=1}^{D} \theta_d \epsilon_{nd}\right] = \sum_{n=1}^{N} y(x_n,\theta) - y_n \sum_{d=1}^{D} \theta_d E[\epsilon_{nd}] = 0$$

$$E\left[\frac{1}{2}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} \theta_d \epsilon_{nd}\right)^2\right] = \frac{1}{2}\sum_{n=1}^{N} E\left[\left(\sum_{d=1}^{D} \theta_d \epsilon_{nd}\right)^2\right] = \frac{1}{2}\sum_{n=1}^{N}\sum_{d=1}^{D} E[\theta_d^2 \epsilon_{nd}^2] = \frac{N\sigma^2}{2}\sum_{d=1}^{D} \theta_d^2$$

$$E[E_D(\theta)'] = E\left[\frac{1}{2}\sum_{n=1}^{N} y(x_n,\theta) - y_n{}^2 + \sum_{n=1}^{N} y(x_n,\theta) - y_n \sum_{d=1}^{D} \theta_d \epsilon_{nd} + \frac{1}{2}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} \theta_d \epsilon\, nd\right)^2\right] = E\,D(\theta) + \frac{N\sigma^2}{2}\sum d=1^{D}\theta d^2$$

Q4: Naive Bayes and Logistic Regression [35pts] **[P] | [W]**

In Bayesian classification, we're interested in finding the probability of a label given some observed feature vector $x = [x_1, \ldots, x_d)$, which we can write as $P(y \vee x_1, \ldots, x_d)$. Bayes's theorem tells us how to express this in terms of quantities we can compute more directly:

$$P(y \vee x_1, \ldots, x_d) = \frac{P(x_1, \ldots, x_d \vee y) P(y)}{P(x_1, \ldots, x_d)}$$

The main assumption in Naive Bayes is that, given the label, the observed features are conditionally independent i.e.

$$P(x_1, \ldots, x_d \vee y) = P(x_1 \vee y) \times \ldots \times P(x_d \vee y)$$

Therefore, we can rewrite Bayes rule as

$$P(y \vee x_1, \ldots, x_d) = \frac{P(x_1 \vee y) \times \ldots \times P(x_d \vee y) P(y)}{P(x_1, \ldots, x_d)}$$

Training Naive Bayes

One way to train a Naive Bayes classifier is done using frequentist approach to calculate probability, which is simply going over the training data and calculating the frequency of different observations in the training set given different labels. For example,

$$P(x_1 = i \vee y = j) = \frac{P(x_1 = i, y = j)}{P(y = j)} = \frac{\text{Number of times in training data } x_1 = i \text{ and } y = j}{\text{Total number of times in training data } y = j}$$

Testing Naive Bayes

During the testing phase, we try to estimate the probability of a label given an observed feature vector. We combine the probabilities computed from training data to estimate the probability of a given label. For example, if we are trying to decide between two labels y_1 and y_2, then we compute the ratio of the posterior probabilities for each label:

$$\frac{P(y_1 \vee x_1, \ldots, x_d)}{P(y_2 \vee x_1, .., x_d)} = \frac{P(x_1, \ldots, x_d \vee y_1)}{P(x_1, \ldots, x_d \vee y_2)} \frac{P(y_1)}{P(y_2)} = \frac{P(x_1 \vee y_1) \times \ldots \times P(x_d \vee y_1) P(y_1)}{P(x_1 \vee y_2) \times \ldots \times P(x_d \vee y_2) P(y_2)}$$

All we need now is to compute $P(x_1 \vee y_i), \ldots, P(x_d \vee y_i)$ and $P(y_i)$ for each label by plugging in the numbers we got during training. The label with the higher posterior probabilities is the one that is selected.

4.1 Llama Breed Problem using Naive Bayes [5pts] [W]

Above are images of two different breeds of llamas – the Suri and the Wooly. The difference between these two breeds is subtle, as these two breeds are often mixed up. However the Suri Llama is vastly more valuable than the Wooly llama. You devise a way to determine with some confidence, which is which – without the need for expensive genetic testing.

You look at four key features of the llama: {curly hair, over 14 inch tail, over 400 pounds, extremely shy}.

You only have 7 randomly chosen llamas to work with, and their breed as the ground truth. You record the data as vectors with the entry 1 if true and 0 if false. For example a llama with vector {1,1,0,1} would have curly hair, a tail over 14 inches, be **less** than 400 pounds, and be extremely shy.

The **Suri Llamas** yield the following data: {1, 0, 1, 0}, {0, 1, 0, 1}, {1, 1, 1, 1}, {0, 0, 0, 1}

The **Wooly Llamas** yield the following data: {0, 0, 1, 0}, {1, 1, 0, 0}, {1, 0, 1, 1}.

Now is the time to test your method!

You see a new llama you are interested in that **has** curly hair, **does** have a tail over 14 inches, is **more than** 400 pounds, and is **not** shy.

Using Naive Bayes, **is this a Suri or a Wooly Llama?**

NOTE: We expect students to show their work (Naive Bayes calculations) and not just the final answer.

Answer ...

LLama **has** curly hair, **does** have a tail over 14 inches, is **more than** 400 pounds, and is **not** shy. {1, 1, 0, 0}

P(S) = Number of Suri Llamas / Total number of llamas = 4 / 7

P(W) = Number of Wooly Llamas / Total number of llamas = 3 / 7

conditional probabilities:

P(has curly hair | Suri) = Number of Suri's with curly hair/total suri's = 2/4

P(has curly hair | Wooly) = Number of Wooly's with curly hair/total wooly's = 2/3

P(tail over 14in | Suri) = Number of Suri's with tail over 14in /total suri's = 2/4

P(tail over 14in | Wooly) = Number of Wooly's with tail over 14in /total wooly's = 1/3

P(less than 400 pounds | Suri) = Number of Suri's less than 400 pounds/total suri's = 2/4

P(less than 400 pounds | Wooly) = Number of Wooly's less than 400 pounds/total wooly's = 1/3

P(not shy | Suri) = Number of Suri's not shy/total suri's = 1/4

P(not shy | Wooly) = Number of Wooly's not shy/total wooly's = 2/3

Finally, P(Suri | {1, 1, 0, 0}) = P(Suri) * all conditional probabilities of suri = 4/7 * 2/4 * 2/4 * 2/4 * 1/4 = 1/56 ≈ 0.0179

P(Wooly | {1, 1, 0, 0}) = P(Wooly) * all conditional probabilities of wooly = 3/7 * 2/3 * 1/3 * 1/3 * 2/3 = 4/189 ≈ 0.021

Since P(Wooly | {1, 1, 0, 0}) > P(Suri | {1, 1, 0, 0}), we can conclude that the given llama is a Wooly llama.

4.2 News Data Sentiment Classification via Logistic Regression [30pts] **[P]**

This dataset contains the sentiments for financial news headlines from the perspective of a retail investor. The sentiment of news has 3 classes, negative, positive and neutral. In this problem, we only use the negative (class label = 0) and positive (class label = 1) classes for binary logistic regression. For data preprocessing, we remove the duplicate headlines and remove the neutral class to get 1967 unique news headlines. Then we randomly split the 1967 headlines into training set and evaluation set with 8:2 ratio. We use the training set to fit a binary logistic regression model.

The code which is provided loads the documents, preprocess the data, builds a "bag of words" representation of each document. Your task is to complete the missing portions of the code in logisticRegression.py to determine whether a news headline is negative or positive.

In logistic_regression.py file, complete the following functions:

- sigmoid: transform $s = x\theta$ to probability of being positive using sigmoid function, which is $\frac{1}{1+e^{-s}}$.
- **bias_augment**: augment x with 1's to account for bias term in θ
- **predict_probs**: predicts the probability of positive label $P(y=1 \vee x)$

- **predict_labels**: predicts labels
- loss: calculates binary cross-entropy loss
- gradient: calculate the gradient of the loss function with respect to the parameters θ.
- accuracy: calculate the accuracy of predictions
- **evaluate**: gives loss and accuracy for a given set of points
- fit: fit the logistic regression model on the training data.

Logistic Regression Overview:

1. In logistic regression, we model the conditional probability using parameters θ, which includes a bias term b.

$$p(y_i=1 \vee x_i;\theta)=h_\theta(x_i)=\sigma(x\theta)$$
$$p(y_i=0 \vee x_i;\theta)=1-h_\theta(x_i)$$

where $\sigma(\cdot)$ is the sigmoid function as follows:

$$\sigma(s)=\frac{1}{1+e^{-s}}$$

1. The conditional probabilities of the positive class $(y=1)$ and the negative class $(y=0)$ of the sample x_i attributes are combined into one equation as follows:

$$p(y*i \vee x_i;\theta)=(h*\theta(x*i))^{y_i}(1-h*\theta(x_i))^{1-y_i}$$

1. Assuming that the samples are independent of each other, the likelihood of the entire dataset is the product of the probabilities of all samples. We use maximum likehood estimation to estimate the model parameters θ. The negative log likelihood (scaled by the dataset size N) is given by:

$$L(\theta \mid X,Y)=-\frac{1}{N}\sum_{i=1}^{N} y_i \log h_\theta(x*i)+(1-y_i)\log(1-h*\theta(x_i))$$

where:

$N=$ ¿ number of training samples
$x_i=$ ¿ bag of words features of the i-th training sample
$y_i=$ ¿ label of the i-th training sample

Note that this will be our model's loss function

1. Then calculate the gradient $\nabla_\theta L$ and use gradient descent to optimize the loss function:

$$\theta_{t+1}=\theta_t-\eta\cdot\nabla_\theta L(\theta_t \mid X,Y)$$

where η is the learning rate and the gradient $\nabla_\theta L$ is given by:

$$\nabla_\theta L(\theta \mid X,Y)=\frac{1}{N}\sum_{i=1}^{N} x_i^\top(h_\theta(x_i)-y_i)$$

4.2.1 Local Tests for Logistic Regression [No Points]

You may test your implementation of the functions contained in **logistic_regression.py** in the cell below. Feel free to comment out tests for functions that have not been completed yet. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestLogisticRegression

unittest_lr = TestLogisticRegression()
unittest_lr.test_gradient()
unittest_lr.test_sigmoid()
unittest_lr.test_bias_augment()
unittest_lr.test_loss()
unittest_lr.test_predict_probs()
unittest_lr.test_predict_labels()
unittest_lr.test_loss()
unittest_lr.test_accuracy()
unittest_lr.test_evaluate()
unittest_lr.test_fit()
```

```
UnitTest passed successfully for "Logistic Regression gradient"!
UnitTest passed successfully for "Logistic Regression sigmoid"!
UnitTest passed successfully for "Logistic Regression bias_augment"!
UnitTest passed successfully for "Logistic Regression loss"!
UnitTest passed successfully for "Logistic Regression predict_probs"!
UnitTest passed successfully for "Logistic Regression predict_labels"!
UnitTest passed successfully for "Logistic Regression loss"!
UnitTest passed successfully for "Logistic Regression accuracy"!
UnitTest passed successfully for "Logistic Regression evaluate"!
Epoch 0:
     train loss: 0.675     train acc: 0.7
     val loss:   0.675     val acc:   0.7
UnitTest passed successfully for "Logistic Regression fit"!
```

4.2.2 Logistic Regression Model Training [No Points]

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from logistic_regression import LogisticRegression as LogReg

################################
### DO NOT CHANGE THIS CELL ###
################################

news_data = pd.read_csv("./data/news-data.csv", encoding="cp437",
```

```
header=None)

class_to_label_mappings = {"negative": 0, "positive": 1}

label_to_class_mappings = {0: "negative", 1: "positive"}

news_data.columns = ["Sentiment", "News"]
news_data.drop_duplicates(inplace=True)

news_data = news_data[news_data.Sentiment != "neutral"]

news_data["Sentiment"] =
news_data["Sentiment"].map(class_to_label_mappings)

vectorizer = text.CountVectorizer(stop_words="english")

X = news_data["News"].values
y = news_data["Sentiment"].values.reshape(-1, 1)

RANDOM_SEED = 5
BOW = vectorizer.fit_transform(X).toarray()
indices = np.arange(len(news_data))
X_train, X_test, y_train, y_test, indices_train, indices_test =
train_test_split(
    BOW, y, indices, test_size=0.2, random_state=RANDOM_SEED
)
```

Fit the model to the training data Try different learning rates `lr` and number of `epochs` to achieve >80% test accuracy.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

model = LogReg()
lr = 0.048
epochs = 11000
theta = model.fit(X_train, y_train, X_test, y_test, lr, epochs)
```

```
Epoch 0:
     train loss: 0.69train acc: 0.7
     val loss:   0.691     val acc:   0.665
Epoch 1000:
     train loss: 0.44train acc: 0.791
     val loss:   0.535     val acc:   0.698
Epoch 2000:
     train loss: 0.368     train acc: 0.841
     val loss:   0.487     val acc:   0.744
Epoch 3000:
     train loss: 0.323     train acc: 0.87
```

```
     val loss:   0.459     val acc:   0.761
Epoch 4000:
     train loss: 0.291     train acc: 0.891
     val loss:   0.441     val acc:   0.769
Epoch 5000:
     train loss: 0.266     train acc: 0.909
     val loss:   0.428     val acc:   0.779
Epoch 6000:
     train loss: 0.246     train acc: 0.923
     val loss:   0.418     val acc:   0.789
Epoch 7000:
     train loss: 0.23train acc: 0.93
     val loss:   0.411     val acc:   0.797
Epoch 8000:
     train loss: 0.216     train acc: 0.941
     val loss:   0.406     val acc:   0.802
Epoch 9000:
     train loss: 0.205     train acc: 0.947
     val loss:   0.401     val acc:   0.797
Epoch 10000:
     train loss: 0.194     train acc: 0.952
     val loss:   0.398     val acc:   0.802
```

4.2.3 Logistic Regression Model Evaluation [No Points]

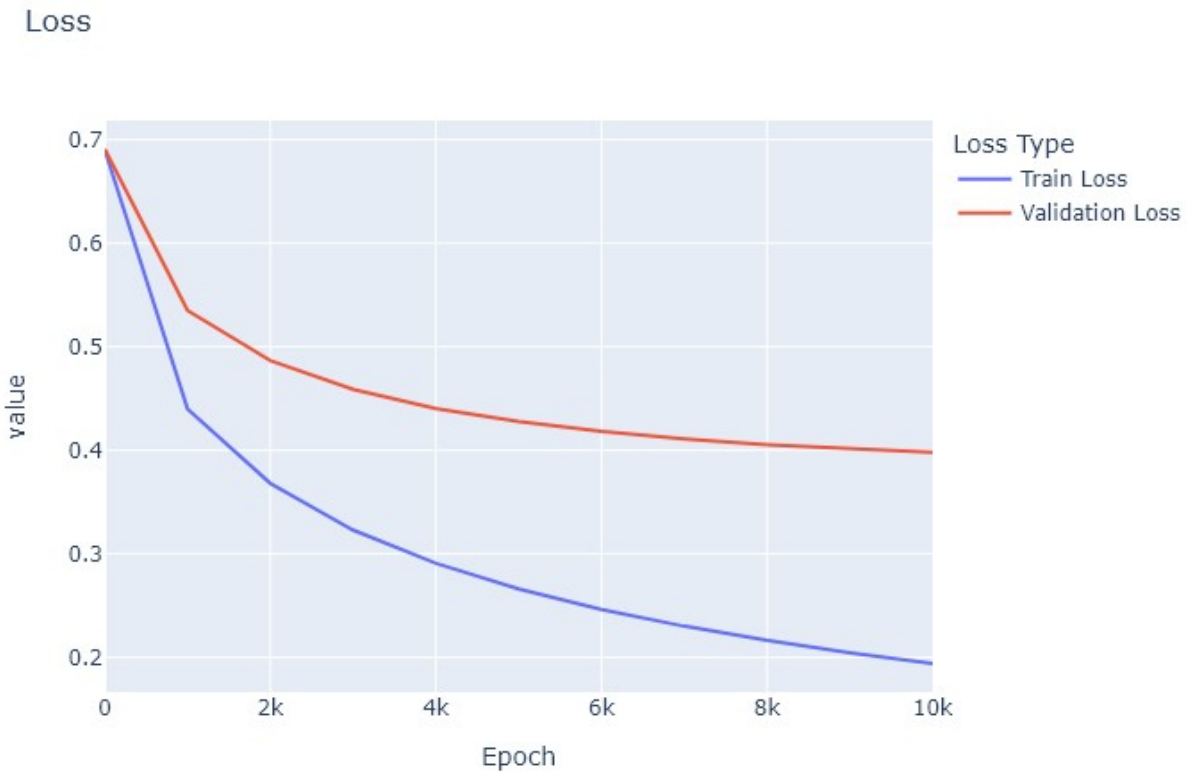
Evaluate the model on the test dataset

```
################################
### DO NOT CHANGE THIS CELL ###
################################

test_loss, test_acc = model.evaluate(X_test, y_test, theta)
print(f"Test Dataset Accuracy: {round(test_acc, 3)}")
```

```
Test Dataset Accuracy: 0.812
```

Plotting the loss function on the training data and the test data for every 100th epoch

```
################################
### DO NOT CHANGE THIS CELL ###
################################

model.plot_loss()
```



Plotting the accuracy function on the training data and the test data for each epoch

```
################################
### DO NOT CHANGE THIS CELL ###
################################

model.plot_accuracy()
```


```
np.reshape(X_test[0], (1, X_test.shape[1])).shape
```

```
(1, 5286)
```

Check out sample evaluations from the test set.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

num_samples = 10
for i in range(10):
    rand_index = np.random.randint(0, len(X_test))
    x_test = np.reshape(X_test[rand_index], (1, X_test.shape[1]))
    prob = model.predict_probs(model.bias_augment(x_test), theta)
    pred = model.predict_labels(prob)
    print(f"Input News: {X[indices_test[rand_index]]}\n")
    print(f"Predicted Sentiment: {label_to_class_mappings[pred[0]
[0]]}")
    print(f"Actual Sentiment:
{label_to_class_mappings[y_test[rand_index][0]]}\n")
```

```
Input News: In a release , the Company said that Vocollect T2
customers in North America can now take advantage of its Voice
Directed Warehousing solution and increase warehouse efficiency and
productivity gains at a discounted price .

Predicted Sentiment: positive
Actual Sentiment: positive

Input News: Svyturys-Utenos Alus , which is controlled by the Nordic
group Baltic Beverages Holding ( BBH ) , posted a 4.7-per-cent growth
in beer sales for January-May to 46.22 million litres .

Predicted Sentiment: positive
Actual Sentiment: positive

Input News: At 1.33 pm , the OMX Helsinki 25 was 0.30 pct lower at
2,463.67 and the OMX Helsinki was down 0.37 pct at 8,537.42 on volume
of 256 mln eur .

Predicted Sentiment: negative
Actual Sentiment: negative

Input News: However , the company saw its net profit for the third
quarter down to EUR 1.4 million from EUR 1.5 million for the
corresponding period of 2009 .

Predicted Sentiment: positive
Actual Sentiment: negative

Input News: Key shareholders of Finnish IT services provider
TietoEnator Oyj on Friday rejected a hostile EUR1 .08 billion $ 1.67
billion offer from buyout shop Nordic Capital , giving new life to a
possible counter offer from Blackstone Group LP and Norwegian telecom
Telenor ASA .

Predicted Sentiment: positive
Actual Sentiment: positive

Input News: ` This is a repeat order to follow successfully installed
159 elevators in the same Delhi metro system , ' Kone spokeswoman told
Thomson Financial News .

Predicted Sentiment: positive
Actual Sentiment: positive

Input News: Israeli cable network operator HOT Telecom has chosen
Teleste , an international technology group , as its exclusive
provider of FTTx equipment in Israel , according to Teleste today .

Predicted Sentiment: positive
Actual Sentiment: positive
```

```
Input News: A portion , $ 12.5 million , will be recorded as part of
its winnings in a prior patent dispute with Finnish phone maker Nokia
Oyj .

Predicted Sentiment: positive
Actual Sentiment: positive

Input News: Also Lemmink+┼inen 's profit for accounting period went up
to EUR 3.1 mn from EUR -24.5 mn a year ago .

Predicted Sentiment: positive
Actual Sentiment: positive

Input News: After the reporting period , BioTie North American
licensing partner Somaxon Pharmaceuticals announced positive results
with nalmefene in a pilot Phase 2 clinical trial for smoking cessation
.

Predicted Sentiment: positive
Actual Sentiment: positive
```

Q5: Noise in PCA and Linear Regression [15pts] **[W]**

Both PCA and least squares regression can be viewed as algorithms for inferring (linear) relationships among data variables. In this part of the assignment, you will develop some intuition for the differences between these two approaches and develop an understanding of the settings that are better suited to using PCA or better suited to using the least squares fit.

The high level bit is that PCA is useful when there is a set of latent (hidden/underlying) variables, and all the coordinates of your data are linear combinations (plus noise) of those variables. The least squares fit is useful when you have direct access to the independent variables, so any noisy coordinates are linear combinations (plus noise) of known variables.

5.1 Slope Functions [5 pts] **[W]**

In the **following cell**, complete the following:

1. **pca_slope**: For this function, assume that X is the first feature and y is the second feature for the data. Write a function, that takes in the first feature vector X and the second feature vector y. Stack these two feature vectors into a single N x 2 matrix and use this to determine the first principal component vector of this dataset. Be careful of how you are stacking the two vectors. You can check the output by printing it which should help you debug. Finally, return the slope of this first component. You should use the PCA implementation from Q2.

2. **lr_slope**: Write a function that takes X and y and returns the slope of the least squares fit. You should use the Linear Regression implementation from Q3 but do not use any kind of regularization. Think about how weight could relate to slope.

In later subparts, we consider the case where our data consists of noisy measurements of x and y. For each part, we will evaluate the quality of the relationship recovered by PCA, and that recovered by standard least squares regression.

As a reminder, least squares regression minimizes the squared error of the dependent variable from its prediction. Namely, given (x_i, y_i) pairs, least squares returns the line $l(x)$ that minimizes $\sum_i \left(y_i - l(x_i)\right)^2$.

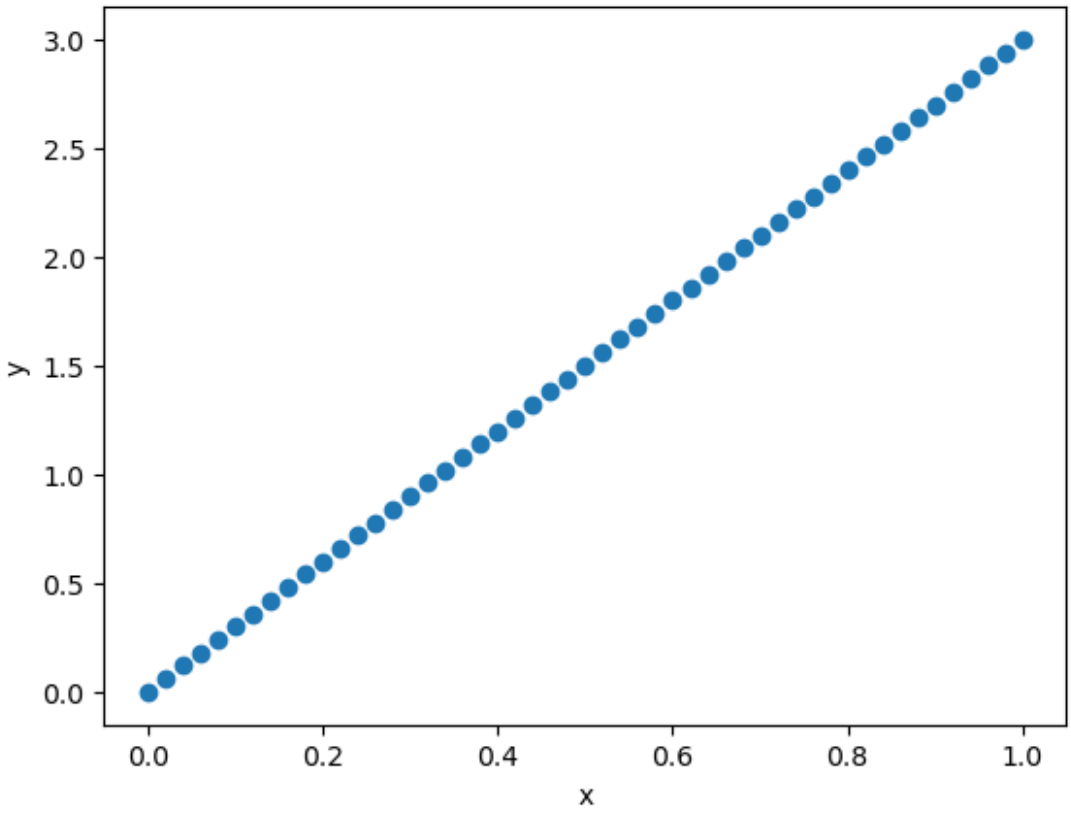
```
import numpy as np
from pca import PCA
from regression import Regression


def pca_slope(X, y):
    """
    Calculates the slope of the first principal component given by PCA

    Args:
        x: N x 1 array of feature x
        y: N x 1 array of feature y
    Return:
        slope: (float) scalar slope of the first principal component
    """
    data = np.column_stack((X, y))

    pca = PCA()
    pca.U, pca.S, pca.V = np.linalg.svd(data, full_matrices=False)

    pc1 = pca.V[0]

    return pc1[1] / pc1[0]


def lr_slope(X, y):
    """
    Calculates the slope of the best fit returned by
linear_fit_closed()

    For this function don't use any regularization

    Args:
        X: N x 1 array corresponding to a dataset
        y: N x 1 array of labels y
    Return:
        slope: (float) slope of the best fit
```

```
    """

    # Compute the mean of X and y
    Xbiased=np.concatenate((np.ones((X.shape[0],1)), X), axis=1)
    regression=Regression()
    weight=regression.linear_fit_closed(Xbiased, y)
    slope=round(float(weight[1]),1)
    return slope
```

We will consider a simple example with two variables, x and y, where the true relationship between the variables is $y=3x$. Our goal is to recover this relationship—namely, recover the coefficient "3". We set $X=[0,.02,.04,.06,\ldots,1)$ and $y=3x$. Make sure both functions return 3.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
x = np.arange(0, 1.02, 0.02).reshape(-1, 1)

y = 3 * np.arange(0, 1.02, 0.02).reshape(-1, 1)

print("Slope of first principal component", pca_slope(x, y))

print("Slope of best linear fit", lr_slope(x, y))

fig = plt.figure()
plt.scatter(x, y)
plt.xlabel("x")
plt.ylabel("y")

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT * 0.8,
    )

plt.show()
```

```
Slope of first principal component 3.000000000000001
[[2.72351586e-16]
 [3.00000000e+00]]
Slope of best linear fit 3.0
```



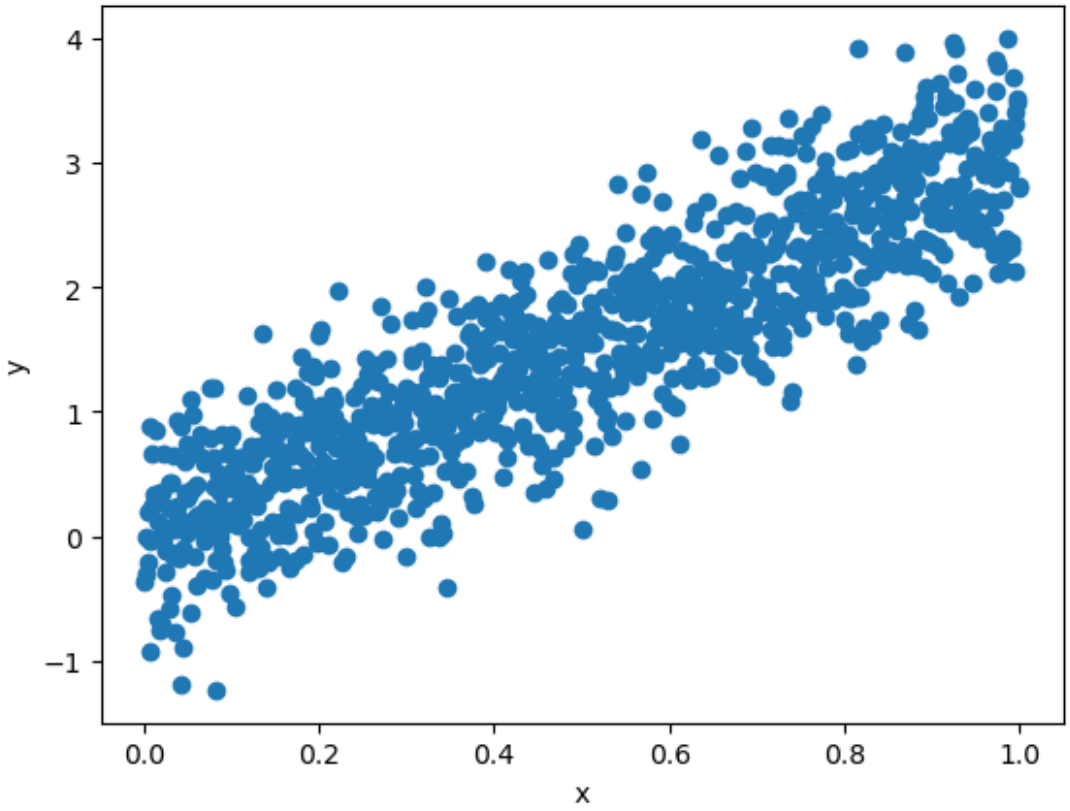
5.2 Analysis Setup [5 pts] **[W]**

Error in y

In this subpart, we consider the setting where our data consists of the actual values of x, and noisy estimates of y. Run the following cell to see how the data looks when there is error in y.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
base = np.arange(0.001, 1.001, 0.001).reshape(-1, 1)
c = 0.5
X = base
y = 3 * base + np.random.normal(loc=[0], scale=c, size=base.shape)

fig = plt.figure()
plt.scatter(X, y)
plt.xlabel("x")
plt.ylabel("y")

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
```

```
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT,
    )

plt.show()
```



In following cell, you will implement the addNoise function:

1. Create a vector X where $X = [x_1, x_2, \ldots, x_{1000}] = [.001, .002, .003, \ldots, 1]$.
2. For a given noise level c, set $ \hat{y}_i ~ 3x_i + \mathcal{N}(0, c) = 3i/1000 + \mathcal{N}(0, c)$, and $\hat{Y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{1000}]$. You can use the `np.random.normal` function, where scale is equal to noise level, to add noise to your points.
3. Notice the parameter x*noise in the addNoise function. When this parameter is set to $True$, you will have to add noise to X. For a given noise level c, let $\hat{x}_i \sim x_i + N(0, c) = i/1000 + N(0, c)$, and $\hat{X} = [\hat{x}_1, \hat{x}_2, \ldots . \hat{x} * 1000]$

4. Return the pca_slope and lr_slope values of this X and $\hat{Y}$ dataset you have created where $\hat{Y}$ has noise ($X = X$ or $\hat{X}$ depending on the problem).

Hint 1: Refer to the above example on how to add noise to X or Y

Hint 2: Be careful not to add double noise to X or Y

```
def addNoise(c, x_noise=False, seed=1):
    """
    Creates a dataset with noise and calculates the slope of the
dataset
    using the pca_slope and lr_slope functions implemented in this
class.

    Args:
        c: (float) scalar, a given noise level to be used on Y and/or
X
        x_noise: (Boolean) When set to False, X should not have noise
added
                 When set to True, X should have noise.
                 Note that the noise added to X should be different
from the
                 noise added to Y. You should NOT use the same noise
you add
                 to Y here.
        seed: (int) Random seed
    Return:
        pca_slope_value: (float) slope value of dataset created using
pca_slope
        lr_slope_value: (float) slope value of dataset created using
lr_slope

    """
    np.random.seed(seed)  #### DO NOT CHANGE THIS ####
    ############## START YOUR CODE BELOW #############
    # TODO: Implement this function
    X=np.linspace(0.001,1,1000).reshape(-1,1)
    if x_noise:
        x_noise=X+np.random.normal(0, c, size=X.shape)
    else:
        x_noise=X
    Y_noise=(3*X+np.random.normal(0, c, size=X.shape))
+np.random.normal(0, c, size=x_noise.shape)
    pca_slope_value=pca_slope(x_noise, Y_noise)
    lr_slope_value=lr_slope(x_noise, Y_noise)

    ############## END YOUR CODE ABOVE #############
    return pca_slope_value, lr_slope_value
```

A scatter plot with c on the horizontal axis and the output of pca_slope and lr_slope on the vertical axis has already been implemented for you.

A sample $\hat{Y}$ has been taken for each c in $[0, 0.05, 0.1, \ldots, 0.95, 1.0)$. The output of pca_slope is plotted as a red dot, and the output of lr_slope as a blue dot. This has been repeated 30 times, you can see that we end up with a plot of 1260 dots, in 21 columns of 60, half red and half blue. Note that the plot you get might not look exactly like the TA version and that is fine because you might have randomized the noise slightly differently than how we did it.

NOTE: Here, `x_noise = False` since we only want Y to have noise.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
pca_slope_values = []
linreg_slope_values = []
c_values = []
s_idx = 0

for i in range(30):
    for c in np.arange(0, 1.05, 0.05):
        # Calculate pca_slope_value (psv) and lr_slope_value (lsv)
        psv, lsv = addNoise(c, seed=s_idx)

        # Append pca and lr slope values to list for plot function
        pca_slope_values.append(psv)
        linreg_slope_values.append(lsv)

        # Append c value to list for plot function
        c_values.append(c)

        # Increment random seed index
        s_idx += 1

fig = plt.figure()
plt.scatter(c_values, pca_slope_values, c="r")
plt.scatter(c_values, linreg_slope_values, c="b")
plt.xlabel("c")
plt.ylabel("slope")

if not STUDENT_VERSION:
    fig.text(
        0.6,
        0.4,
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA * 0.5,
        fontname=EO_FONT,
```

```
        ha="center",
        va="center",
        rotation=EO_ROT,
    )

plt.show()
```

```
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.00659601]
 [2.99342835]]
[[-0.00686805]
 [ 3.00202311]]
[[-0.01309136]
 [ 3.00834896]]
[[0.02793374]
 [2.96550187]]
[[0.01364546]
 [2.98206731]]
[[0.01314135]
 [2.99162222]]
[[-0.03153234]
 [ 3.01167523]]
[[0.03624293]
 [2.99260103]]
[[0.04660975]
 [2.89771898]]
[[0.07582357]
 [2.82288181]]
[[0.01133277]
 [2.99912086]]
[[-0.10897438]
 [ 3.13348867]]
[[0.00478438]
 [2.95964847]]
[[-0.09019413]
 [ 3.08366945]]
[[-0.00351808]
 [ 2.92083121]]
[[0.02156053]
 [2.96592475]]
[[0.04120154]
 [3.06718596]]
[[0.11935153]
 [2.82951567]]
[[-0.18009383]
 [ 3.29592422]]
[[0.00719978]
 [2.96100334]]
[[9.99200722e-16]
```

```
 [3.00000000e+00]]
[[0.00449117]
 [2.98752882]]
[[-0.0140057 ]
 [ 3.01385239]]
[[-0.01295163]
 [ 3.01569808]]
[[-0.0167135 ]
 [ 3.05052751]]
[[-0.01786592]
 [ 3.01945043]]
[[0.02586458]
 [2.96526402]]
[[-0.00708533]
 [ 3.02720969]]
[[0.00660587]
 [3.02741314]]
[[-0.03920546]
 [ 3.10864718]]
[[0.04865478]
 [2.94368066]]
[[1.26423226e-03]
 [2.95375105e+00]]
[[-0.0082633 ]
 [ 3.01589418]]
[[0.00299455]
 [2.90381087]]
[[-0.01183394]
 [ 2.99596873]]
[[0.00979642]
 [2.97529226]]
[[0.01780245]
 [2.95914585]]
[[0.10782712]
 [2.91156432]]
[[-0.04350946]
 [ 2.96369703]]
[[0.07140746]
 [3.03782351]]
[[-0.09294571]
 [ 3.17303331]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.00611278]
 [2.99187538]]
[[0.00385927]
 [2.9875419 ]]
[[0.00683366]
 [2.98126223]]
```

```
[[0.01965939]
 [2.96018051]]
[[0.02283532]
 [2.99607706]]
[[0.00731021]
 [3.00240662]]
[[-0.00317421]
 [ 3.00095091]]
[[-0.02132006]
 [ 3.03761853]]
[[-0.05075687]
 [ 3.04318806]]
[[0.02892048]
 [2.90740056]]
[[0.01549167]
 [2.91899138]]
[[-0.09284835]
 [ 3.08512174]]
[[-0.00662798]
 [ 3.03814995]]
[[0.09404345]
 [2.84137249]]
[[0.01771774]
 [2.96174604]]
[[-0.10826044]
 [ 3.09653186]]
[[0.02525577]
 [3.042568  ]]
[[-0.02458795]
 [ 2.94939507]]
[[-0.0840989 ]
 [ 3.03564173]]
[[0.02743345]
 [3.02891067]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-0.00358176]
 [ 3.00086553]]
[[9.92390272e-04]
 [2.99722290e+00]]
[[9.34570383e-04]
 [3.00280063e+00]]
[[-0.02249014]
 [ 3.02452283]]
[[0.02414977]
 [2.95966555]]
[[-0.02597946]
 [ 3.03195504]]
[[0.03742797]
```

```
 [2.91407206]]
[[0.03077118]
 [2.92641215]]
[[-0.03292665]
 [ 3.0714882 ]]
[[-0.03046085]
 [ 3.05634903]]
[[0.03882211]
 [2.92178672]]
[[0.05662722]
 [2.93766079]]
[[-0.04089948]
 [ 3.08816379]]
[[-0.00718186]
 [ 3.01614753]]
[[0.01015137]
 [3.02642895]]
[[0.05282062]
 [2.99815612]]
[[0.05536966]
 [2.98226898]]
[[0.00415118]
 [2.98956841]]
[[-0.02197891]
 [ 3.02955779]]
[[0.01341721]
 [2.98535933]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[9.90012991e-04]
 [3.00070339e+00]]
[[0.00868226]
 [2.97951078]]
[[-0.01026265]
 [ 3.04318232]]
[[0.00922403]
 [3.03197948]]
[[-0.02711003]
 [ 3.05683968]]
[[-0.01510369]
 [ 3.06162815]]
[[0.02460725]
 [2.94764748]]
[[-0.03021042]
 [ 3.0744132 ]]
[[0.02718061]
 [2.92714277]]
[[-0.04357243]
 [ 3.10428522]]
```

```
[[0.06678385]
 [2.81506085]]
[[-0.01243682]
 [ 3.04135947]]
[[-0.02811934]
 [ 3.08735508]]
[[-0.11795659]
 [ 3.22624798]]
[[0.07853088]
 [2.93864705]]
[[0.06234178]
 [2.86275226]]
[[0.02765109]
 [3.07162601]]
[[0.07961688]
 [2.92540074]]
[[0.0695839 ]
 [3.02413013]]
[[-0.03136482]
 [ 3.13682413]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[2.40403093e-03]
 [2.99815150e+00]]
[[-1.68236591e-03]
 [ 2.99679001e+00]]
[[0.0133816 ]
 [2.95755226]]
[[0.06195866]
 [2.92437225]]
[[-0.00424903]
 [ 3.0078703 ]]
[[-0.00611937]
 [ 3.02970166]]
[[0.03866248]
 [2.93423754]]
[[0.06502095]
 [2.9527844 ]]
[[0.01215357]
 [3.0028825 ]]
[[0.07054248]
 [2.88319321]]
[[0.04526013]
 [2.93798856]]
[[0.17078815]
 [2.73382439]]
[[-0.03381659]
 [ 3.03908876]]
[[0.10211165]
```

```
 [2.85220468]]
[[0.04225001]
 [2.99818717]]
[[0.01176605]
 [2.86852542]]
[[-0.08181167]
 [ 3.0747803 ]]
[[0.03473081]
 [2.87454889]]
[[0.06966587]
 [2.96059275]]
[[0.0692587 ]
 [3.02537594]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-0.00447567]
 [ 3.00452435]]
[[0.0106022]
 [2.9927212]]
[[-0.03111578]
 [ 3.03505963]]
[[-0.02006641]
 [ 3.00821954]]
[[-0.01429617]
 [ 2.99076986]]
[[-0.00372518]
 [ 3.01043907]]
[[0.05051171]
 [2.93199709]]
[[0.01969541]
 [3.0111905 ]]
[[-0.04094905]
 [ 3.06217654]]
[[-0.08983497]
 [ 3.05175298]]
[[0.01897793]
 [2.97470304]]
[[-0.03631923]
 [ 3.02626239]]
[[0.02746822]
 [2.82849015]]
[[0.1427348 ]
 [2.85185324]]
[[0.04668708]
 [2.86539297]]
[[0.10402872]
 [2.81122184]]
[[0.13280772]
 [2.84431965]]
```

```
[[0.04250754]
 [2.86212016]]
[[0.02482208]
 [2.87752315]]
[[0.00414964]
 [3.08372777]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[4.45108753e-04]
 [2.99877392e+00]]
[[1.24583653e-04]
 [2.99423667e+00]]
[[-0.01565141]
 [ 3.02811328]]
[[0.01838521]
 [2.96606637]]
[[-0.05357435]
 [ 3.09487701]]
[[6.73359219e-04]
 [2.99163888e+00]]
[[0.01901154]
 [2.96639995]]
[[-0.01223914]
 [ 3.02500103]]
[[-0.04569658]
 [ 3.12103485]]
[[2.38830178e-03]
 [2.92650301e+00]]
[[3.43161106e-04]
 [3.03013771e+00]]
[[-0.03921684]
 [ 3.05123795]]
[[0.09889056]
 [2.81205418]]
[[-0.04597864]
 [ 3.10454989]]
[[0.02039077]
 [2.96018751]]
[[0.03163082]
 [2.95916671]]
[[-0.09945039]
 [ 3.16384575]]
[[0.02773074]
 [2.94586936]]
[[-0.08443   ]
 [ 3.21259707]]
[[-0.02499241]
 [ 3.03676636]]
[[9.99200722e-16]
```

```
 [3.00000000e+00]]
[[-0.00885536]
 [ 3.01692928]]
[[-2.77411753e-03]
 [ 3.00118674e+00]]
[[0.00515622]
 [3.02902502]]
[[2.40383562e-03]
 [2.98066959e+00]]
[[0.01993409]
 [2.97268971]]
[[-0.03589585]
 [ 3.08588579]]
[[-5.95859624e-05]
 [ 2.98104525e+00]]
[[0.02099012]
 [2.95925567]]
[[8.38921372e-04]
 [2.93591365e+00]]
[[0.01704189]
 [3.02372439]]
[[0.05431998]
 [2.8812448 ]]
[[0.04790806]
 [2.95084562]]
[[-0.04182212]
 [ 3.06567684]]
[[0.08385238]
 [2.97091059]]
[[0.04169503]
 [3.01122376]]
[[0.13199279]
 [2.82027721]]
[[0.1307235 ]
 [2.81527234]]
[[-0.03804242]
 [ 3.05052954]]
[[-0.27863369]
 [ 3.35741792]]
[[0.18965377]
 [2.85919133]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-1.72524923e-03]
 [ 2.99818450e+00]]
[[-0.02038282]
 [ 3.01854164]]
[[0.00860734]
 [2.96750819]]
```

```
[[0.00735869]
 [2.99714089]]
[[-0.00659306]
 [ 3.00595873]]
[[0.01698059]
 [2.97514552]]
[[0.01051543]
 [2.96649844]]
[[0.0476895 ]
 [2.94776239]]
[[-0.03556877]
 [ 3.04011653]]
[[0.05584665]
 [2.86956719]]
[[0.12568425]
 [2.74595453]]
[[-0.02806408]
 [ 3.03464867]]
[[-0.07271957]
 [ 3.10248663]]
[[0.06750695]
 [2.96212107]]
[[0.1409074 ]
 [2.69787479]]
[[0.03977642]
 [3.0266697 ]]
[[-0.08610128]
 [ 3.11244516]]
[[-0.01558427]
 [ 2.99455007]]
[[-0.01239452]
 [ 3.01314615]]
[[0.09607719]
 [2.82070206]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[1.59371201e-03]
 [3.00053041e+00]]
[[0.01343277]
 [2.99324284]]
[[-0.01204345]
 [ 3.01370401]]
[[-0.01708436]
 [ 2.98068963]]
[[7.70848129e-04]
 [3.00573895e+00]]
[[-0.0061741 ]
 [ 2.99750665]]
[[0.00862708]
```

```
 [2.99789655]]
[[0.01253154]
 [2.99356517]]
[[-0.03411558]
 [ 3.07301382]]
[[0.03295303]
 [2.96446295]]
[[0.09458178]
 [2.87034014]]
[[-0.05882048]
 [ 3.08038195]]
[[-0.09120023]
 [ 3.13586489]]
[[0.00814068]
 [2.97782634]]
[[0.04700389]
 [2.97821808]]
[[0.0204905 ]
 [2.96668779]]
[[0.08615559]
 [2.79293659]]
[[0.15171523]
 [2.65007271]]
[[0.05367863]
 [2.79898005]]
[[0.04885548]
 [2.88814998]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-1.21224487e-03]
 [ 3.00092468e+00]]
[[0.00913187]
 [3.00148355]]
[[-1.92587183e-03]
 [ 3.00196017e+00]]
[[0.01438327]
 [2.96761192]]
[[-0.01537386]
 [ 3.03070077]]
[[-0.04436914]
 [ 3.03582459]]
[[0.01340907]
 [3.00207563]]
[[0.01050715]
 [2.95243942]]
[[0.08064692]
 [2.87000394]]
[[-0.00866796]
 [ 3.02176916]]
```

```
[[0.03979798]
 [2.86423077]]
[[-0.0263264 ]
 [ 3.04085827]]
[[-0.08675902]
 [ 3.10477307]]
[[-0.14721061]
 [ 3.37286915]]
[[0.05830174]
 [2.9712099 ]]
[[-0.12246526]
 [ 3.19027331]]
[[0.02831262]
 [2.9777554 ]]
[[-0.02454905]
 [ 3.09437521]]
[[0.01818859]
 [2.89926287]]
[[-0.02426906]
 [ 3.18261483]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.0034165]
 [2.9952853]]
[[0.00641962]
 [2.99004361]]
[[-0.00751528]
 [ 3.00156998]]
[[-0.01307708]
 [ 3.04050943]]
[[-0.01391538]
 [ 3.02273291]]
[[-0.00615546]
 [ 2.99485761]]
[[0.01027745]
 [2.97826094]]
[[0.03921187]
 [2.97738344]]
[[0.01306586]
 [3.00521556]]
[[0.05980659]
 [2.87744232]]
[[-0.04977304]
 [ 3.05666556]]
[[0.02078606]
 [2.9536063 ]]
[[0.0461545 ]
 [2.90741389]]
[[0.02957677]
 [3.0435354 ]]
```

```
[[0.01203177]
 [3.02588992]]
[[0.07475538]
 [2.80262404]]
[[-0.0703396 ]
 [ 3.12062231]]
[[0.05233301]
 [2.91421193]]
[[0.01286902]
 [3.00144567]]
[[0.01700171]
 [2.98385412]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-0.00703896]
 [ 3.00743653]]
[[0.01879141]
 [2.97388179]]
[[0.00892822]
 [2.97991765]]
[[0.00716245]
 [2.99193625]]
[[0.00538913]
 [3.00427586]]
[[0.02684779]
 [2.96247018]]
[[0.0102472 ]
 [2.99970679]]
[[-0.00487545]
 [ 3.00767133]]
[[2.52840797e-03]
 [2.99341058e+00]]
[[-0.04971039]
 [ 3.06732172]]
[[-0.04873111]
 [ 3.08030235]]
[[-0.04851593]
 [ 2.97489062]]
[[-0.11432375]
 [ 3.20927173]]
[[0.09312475]
 [2.8070942 ]]
[[-0.02511058]
 [ 3.11385943]]
[[0.10834491]
 [2.88881604]]
[[-0.06079109]
 [ 3.09865752]]
[[-6.58859759e-04]
```

```
 [ 2.93803109e+00]]
[[0.02419351]
 [3.05096389]]
[[-0.08657952]
 [ 3.03661466]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-8.60744692e-04]
 [ 2.99753202e+00]]
[[-0.00808214]
 [ 3.00754568]]
[[-0.00833676]
 [ 3.02704655]]
[[-0.01422657]
 [ 3.03491243]]
[[-0.05113936]
 [ 3.07981607]]
[[0.02522328]
 [2.94740025]]
[[-0.02469344]
 [ 3.02491319]]
[[-0.08410574]
 [ 3.15435366]]
[[0.00504466]
 [2.99525411]]
[[-0.04576304]
 [ 3.0792764 ]]
[[0.05419595]
 [2.93769364]]
[[-0.08278303]
 [ 3.11290342]]
[[0.00362812]
 [3.0205482 ]]
[[-0.02018834]
 [ 2.98688703]]
[[0.0127935 ]
 [3.02138687]]
[[-0.11669051]
 [ 3.12465235]]
[[-0.17642139]
 [ 3.25449329]]
[[0.07423444]
 [2.8231086 ]]
[[-0.07149302]
 [ 3.07963966]]
[[0.10661724]
 [2.77153733]]
[[9.99200722e-16]
 [3.00000000e+00]]
```

```
[[1.43127144e-03]
 [2.99293487e+00]]
[[-0.01205942]
 [ 3.02109718]]
[[0.0040748 ]
 [2.97395299]]
[[0.02178445]
 [2.97217233]]
[[0.01749555]
 [2.92866951]]
[[0.01968123]
 [2.96028907]]
[[-0.03430134]
 [ 3.0644934 ]]
[[0.02461332]
 [2.95393714]]
[[0.00498424]
 [2.99285334]]
[[0.06001632]
 [2.95215092]]
[[0.03011313]
 [2.84482773]]
[[-0.03535962]
 [ 2.99700264]]
[[0.03917248]
 [2.95818162]]
[[0.01750425]
 [2.96237876]]
[[-0.08514348]
 [ 3.24655026]]
[[0.05811649]
 [2.88866794]]
[[-0.06430215]
 [ 3.09050184]]
[[0.1027578 ]
 [2.76653338]]
[[0.0654038 ]
 [2.94992146]]
[[-0.00800741]
 [ 3.10460278]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[1.67618354e-03]
 [2.99877399e+00]]
[[-4.34779944e-04]
 [ 3.00157783e+00]]
[[-0.02109413]
 [ 3.03294555]]
[[-0.03968274]
```

```
 [ 3.03359739]]
[[-1.36548519e-03]
 [ 2.99145117e+00]]
[[-0.01204672]
 [ 3.06727113]]
[[0.04430175]
 [2.99245504]]
[[0.00455575]
 [2.98042691]]
[[-0.02949868]
 [ 3.02063301]]
[[-0.01655317]
 [ 3.02733888]]
[[-0.01067104]
 [ 2.98611682]]
[[-0.02901484]
 [ 3.09515532]]
[[0.07067176]
 [2.88473434]]
[[-0.00968644]
 [ 2.98872276]]
[[-0.01547832]
 [ 3.03635742]]
[[0.01715955]
 [2.95818022]]
[[0.01332259]
 [3.04358133]]
[[-0.110956  ]
 [ 3.17278468]]
[[0.12234778]
 [2.87317309]]
[[-0.02671536]
 [ 3.09337026]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[1.96543200e-03]
 [2.99733176e+00]]
[[6.67378313e-05]
 [3.00333049e+00]]
[[0.0040841 ]
 [2.98299742]]
[[-0.01226882]
 [ 2.99851215]]
[[0.02395145]
 [2.97335027]]
[[0.04102113]
 [2.9492219 ]]
[[-0.02121668]
 [ 3.09074416]]
```

```
[[-0.0087087 ]
 [ 3.02380813]]
[[0.01843774]
 [2.99322519]]
[[0.04547157]
 [2.85460181]]
[[0.02307463]
 [2.94832074]]
[[-0.01083373]
 [ 2.93783018]]
[[-0.04343144]
 [ 3.02462644]]
[[0.03575456]
 [2.93644392]]
[[-0.01508455]
 [ 3.01023975]]
[[-0.15412006]
 [ 3.27247291]]
[[5.08999050e-04]
 [2.85879601e+00]]
[[0.08517648]
 [2.84592585]]
[[-0.03712139]
 [ 3.09766984]]
[[0.00625265]
 [2.94544354]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[1.01289872e-03]
 [3.00273531e+00]]
[[-0.00892463]
 [ 3.02579002]]
[[-1.54985005e-03]
 [ 2.98775740e+00]]
[[-0.00517674]
 [ 3.06367912]]
[[7.80800933e-04]
 [2.97909723e+00]]
[[0.00488315]
 [3.02253943]]
[[-0.03074309]
 [ 3.01673725]]
[[-0.06803311]
 [ 3.08162802]]
[[-0.09237395]
 [ 3.10408021]]
[[0.03520819]
 [2.88935661]]
[[-0.12645465]
```

```
 [ 3.27904258]]
[[0.01755898]
 [2.9712973 ]]
[[0.00546992]
 [3.08755208]]
[[-2.57702572e-03]
 [ 2.90388873e+00]]
[[-0.0507664]
 [ 3.063065 ]]
[[0.03442407]
 [2.89068016]]
[[-0.20374933]
 [ 3.31470579]]
[[0.13425481]
 [2.91203023]]
[[0.02491308]
 [2.95295615]]
[[-8.62782606e-04]
 [ 3.03604482e+00]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-0.00348403]
 [ 3.00277319]]
[[0.0168438 ]
 [2.96532738]]
[[0.01226092]
 [2.98867807]]
[[-0.0118929 ]
 [ 3.00015278]]
[[0.01895726]
 [2.96166263]]
[[0.00951026]
 [3.01351303]]
[[0.05513331]
 [2.87414428]]
[[-0.0278087 ]
 [ 3.09062428]]
[[-0.01212068]
 [ 3.03093685]]
[[-0.0067195 ]
 [ 3.01876144]]
[[0.04363375]
 [2.8836676 ]]
[[0.07299813]
 [2.85391684]]
[[0.02162992]
 [2.87926051]]
[[-0.01281471]
 [ 3.0636138 ]]
```

```
[[0.04056255]
 [2.93861858]]
[[-0.0977124 ]
 [ 3.06395771]]
[[0.11860799]
 [2.8309308 ]]
[[-0.0541785 ]
 [ 3.10387234]]
[[-0.07804578]
 [ 3.12604843]]
[[-2.62682594e-03]
 [ 2.91133187e+00]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.0045616 ]
 [2.99404045]]
[[1.73761588e-03]
 [2.98822595e+00]]
[[0.01499901]
 [2.98552608]]
[[2.51605790e-03]
 [2.99620398e+00]]
[[-1.39867614e-04]
 [ 3.01065375e+00]]
[[2.88900663e-03]
 [3.00258151e+00]]
[[0.04489954]
 [2.93123549]]
[[0.0448118 ]
 [2.93311643]]
[[0.05812638]
 [2.96185859]]
[[-0.01297064]
 [ 3.00946186]]
[[0.05506554]
 [2.93378592]]
[[0.04033352]
 [2.97307021]]
[[1.74914076e-03]
 [3.00278456e+00]]
[[-0.04622447]
 [ 3.08924546]]
[[0.01666944]
 [3.06984542]]
[[-0.05347963]
 [ 3.01372986]]
[[-0.13833376]
 [ 3.20570021]]
[[-0.07203822]
```

```
 [ 3.18804882]]
[[0.06355228]
 [2.89676612]]
[[-0.04155285]
 [ 3.06117994]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-2.67763740e-03]
 [ 3.00795063e+00]]
[[0.01262199]
 [2.97866706]]
[[-0.00663928]
 [ 2.98536019]]
[[-1.04592198e-03]
 [ 3.00586112e+00]]
[[-0.01212825]
 [ 2.99156597]]
[[0.02776405]
 [2.9860136 ]]
[[-0.01621673]
 [ 3.07068175]]
[[-0.02545628]
 [ 3.03058305]]
[[0.01547397]
 [2.96533635]]
[[-0.0234484 ]
 [ 3.07795045]]
[[-0.05151893]
 [ 3.10948751]]
[[0.0031021 ]
 [2.93458808]]
[[-0.10724227]
 [ 3.26595009]]
[[-2.03786326e-03]
 [ 3.01894157e+00]]
[[0.04205669]
 [2.95876162]]
[[0.08245692]
 [2.9505003 ]]
[[-0.03208428]
 [ 3.06225873]]
[[0.11561138]
 [2.75505909]]
[[-0.01036229]
 [ 2.96310633]]
[[0.00650299]
 [3.1163906 ]]
[[9.99200722e-16]
 [3.00000000e+00]]
```

```
[[0.00473822]
 [2.99383966]]
[[-0.0038352 ]
 [ 2.99186115]]
[[0.00346131]
 [2.99649959]]
[[0.01495776]
 [2.97397966]]
[[0.00335723]
 [2.99954319]]
[[-0.04045306]
 [ 3.04094527]]
[[-0.0454256 ]
 [ 3.10292398]]
[[-0.0393124 ]
 [ 3.05407847]]
[[-0.0405685 ]
 [ 3.05241121]]
[[-0.08241019]
 [ 3.14098495]]
[[0.08150336]
 [2.84435966]]
[[-0.04871551]
 [ 2.98096794]]
[[0.01692367]
 [2.97657491]]
[[0.0448513 ]
 [2.95728499]]
[[0.00581409]
 [3.04819096]]
[[-0.04223447]
 [ 2.99441784]]
[[0.11458785]
 [2.83751085]]
[[0.0253501]
 [2.9636671]]
[[0.12759004]
 [2.75575308]]
[[0.09769126]
 [2.83798656]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-2.82288861e-03]
 [ 3.00278062e+00]]
[[-1.19620539e-03]
 [ 3.00249531e+00]]
[[-0.0118166 ]
 [ 3.01174107]]
[[0.01199083]
```

```
 [2.97369342]]
[[-0.03576341]
 [ 3.07403126]]
[[-0.04501276]
 [ 3.02755764]]
[[0.07272144]
 [2.83844101]]
[[-0.0199127]
 [ 3.033018 ]]
[[-0.00980142]
 [ 3.0288004 ]]
[[-0.0123662 ]
 [ 3.02504093]]
[[0.10336223]
 [2.86149636]]
[[0.06293259]
 [2.90184922]]
[[7.31174182e-04]
 [2.97321815e+00]]
[[-0.05491132]
 [ 3.05464593]]
[[0.00348384]
 [3.02502206]]
[[-0.04687027]
 [ 3.04382425]]
[[0.02770874]
 [2.9743496 ]]
[[2.61431522e-03]
 [2.99940076e+00]]
[[0.00318124]
 [2.89269567]]
[[-0.01967682]
 [ 2.97214753]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.005754  ]
 [2.99724315]]
[[0.00331477]
 [3.00473744]]
[[0.00795379]
 [2.9860211 ]]
[[-0.02517606]
 [ 3.036104  ]]
[[0.03965151]
 [2.91164792]]
[[0.01242443]
 [2.97129011]]
[[0.01402286]
 [2.95807744]]
```

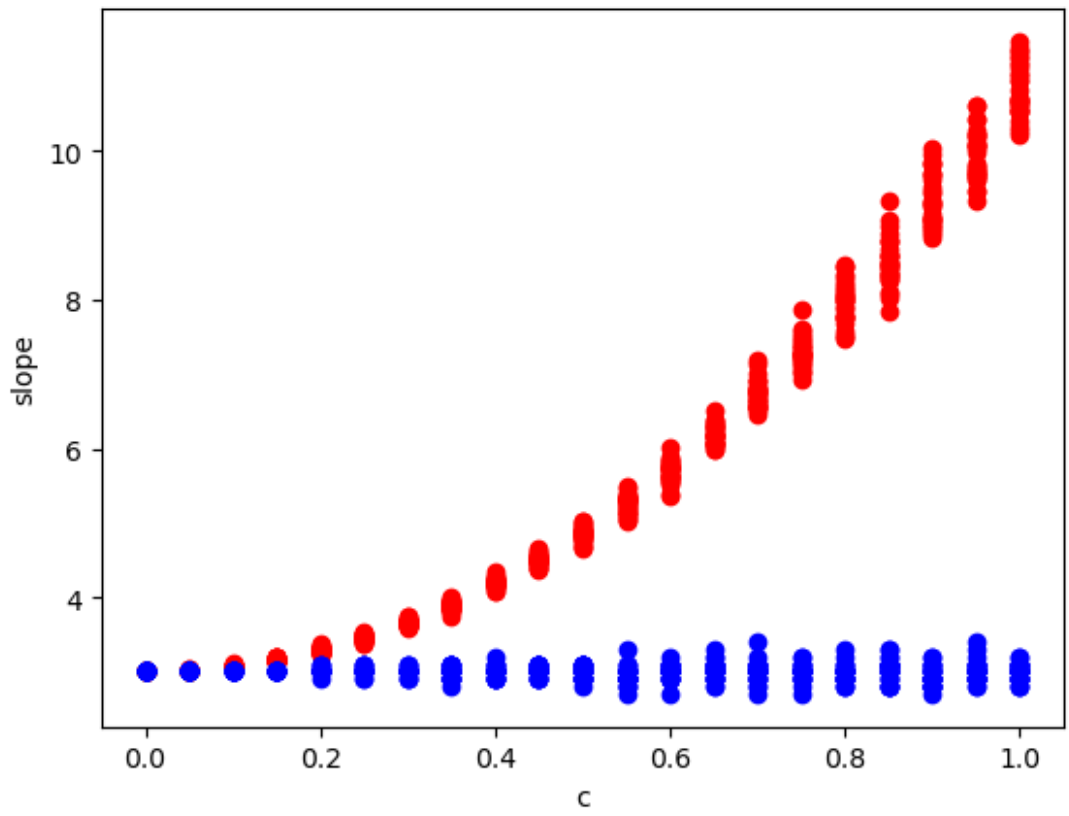
```
[[0.00379565]
 [2.99917687]]
[[-0.04271319]
 [ 3.07823057]]
[[0.01932265]
 [2.97440685]]
[[0.04978455]
 [2.88085827]]
[[0.0492816 ]
 [2.91070558]]
[[0.07850287]
 [2.91891175]]
[[-0.08319892]
 [ 3.14818482]]
[[0.04974719]
 [3.01148823]]
[[0.0877562 ]
 [2.85378421]]
[[-0.05092829]
 [ 3.01648237]]
[[-0.10458676]
 [ 3.03382114]]
[[-0.00680221]
 [ 2.88550386]]
[[0.1054171 ]
 [2.90005067]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[2.74045753e-03]
 [2.98977479e+00]]
[[0.00783358]
 [2.97982165]]
[[0.00324298]
 [2.97989719]]
[[0.00406729]
 [2.98321767]]
[[-0.00761746]
 [ 3.01466617]]
[[0.01450648]
 [2.9750313 ]]
[[-0.03123866]
 [ 3.03261121]]
[[0.01681058]
 [2.95821555]]
[[0.07524913]
 [2.88553463]]
[[-0.07941589]
 [ 3.09793933]]
[[-0.02686922]
 [ 3.01822493]]
```

```
[[-0.01399412]
 [ 3.02497094]]
[[-0.08792819]
 [ 3.19309067]]
[[-0.03565771]
 [ 3.03104628]]
[[0.05408485]
 [2.98675505]]
[[0.04133485]
 [2.85594993]]
[[-0.01824936]
 [ 2.99509736]]
[[0.07538013]
 [2.83688002]]
[[0.0051372 ]
 [2.96363221]]
[[0.03649166]
 [3.05883015]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-2.22130570e-03]
 [ 3.00630873e+00]]
[[0.01088889]
 [2.97596338]]
[[0.02250339]
 [2.97348386]]
[[-0.03342512]
 [ 3.04978325]]
[[0.0030431 ]
 [2.99324663]]
[[-1.75741435e-03]
 [ 3.03555242e+00]]
[[-0.04118809]
 [ 3.07180861]]
[[0.05487677]
 [2.93948642]]
[[-0.0291812]
 [ 3.0322885]]
[[0.0073468 ]
 [2.94353572]]
[[0.0676849 ]
 [2.88615063]]
[[-0.04270705]
 [ 3.09704822]]
[[0.01912067]
 [2.90667408]]
[[-0.01998737]
 [ 3.05893788]]
[[-0.14402517]
```

```
 [ 3.18926231]]
[[0.0331089 ]
 [2.96226426]]
[[0.0747329 ]
 [2.92067443]]
[[0.05186145]
 [3.05467991]]
[[0.0169248 ]
 [2.97189608]]
[[0.07676457]
 [2.90499795]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[-1.26802074e-03]
 [ 3.00402970e+00]]
[[0.00413211]
 [2.98848325]]
[[0.00673849]
 [2.99309258]]
[[-0.0374622 ]
 [ 3.08277477]]
[[-0.00894171]
 [ 2.98057125]]
[[0.06029072]
 [2.92309629]]
[[-0.01868438]
 [ 3.07885636]]
[[0.02949984]
 [2.93145996]]
[[-0.01428215]
 [ 2.98364426]]
[[-0.01064136]
 [ 3.05973454]]
[[0.01069043]
 [3.01300737]]
[[-0.01154476]
 [ 2.96935008]]
[[-0.05566136]
 [ 3.03865875]]
[[-0.00944027]
 [ 3.04984761]]
[[-0.00590668]
 [ 2.9346895 ]]
[[-0.04020437]
 [ 2.97954288]]
[[-0.05648805]
 [ 3.11570449]]
[[0.02206213]
 [2.91592024]]
```

```
[[0.02555536]
 [2.8841893 ]]
[[0.02598322]
 [2.91946491]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[1.85534550e-03]
 [3.00268499e+00]]
[[-0.01647916]
 [ 3.02679019]]
[[6.88428447e-04]
 [3.02604152e+00]]
[[-6.82848714e-04]
 [ 2.97741961e+00]]
[[0.02713006]
 [2.97592906]]
[[-0.02298753]
 [ 3.01736411]]
[[0.05135098]
 [2.94723203]]
[[0.03956369]
 [2.94409765]]
[[0.01881716]
 [2.97117808]]
[[-3.78374105e-04]
 [ 3.02416491e+00]]
[[-0.07801967]
 [ 2.97135053]]
[[-0.13878102]
 [ 3.19456836]]
[[-0.04847788]
 [ 3.06637357]]
[[-0.03711315]
 [ 2.93640567]]
[[0.11360191]
 [2.79404671]]
[[0.10970678]
 [2.86339029]]
[[0.01246293]
 [2.99977702]]
[[0.02745508]
 [2.92680485]]
[[-0.08005543]
 [ 3.10578122]]
[[0.00643881]
 [3.04490844]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[7.80442512e-04]
```

```
 [3.00182423e+00]]
[[2.06555691e-03]
 [2.99732735e+00]]
[[-0.01076286]
 [ 3.03172784]]
[[0.01745102]
 [2.96775102]]
[[0.0042699 ]
 [2.97761463]]
[[0.03768353]
 [2.97360514]]
[[-0.03449152]
 [ 3.07144185]]
[[0.00697774]
 [2.97282162]]
[[-0.02012734]
 [ 3.01021263]]
[[0.04356801]
 [3.03938447]]
[[0.01733262]
 [2.99851693]]
[[0.02619721]
 [3.02420437]]
[[0.02334331]
 [2.96409775]]
[[0.13597231]
 [2.73047378]]
[[-0.03137675]
 [ 3.05946011]]
[[-0.03738247]
 [ 2.98944655]]
[[-0.07733472]
 [ 3.1041812 ]]
[[0.13663092]
 [2.78421981]]
[[-0.04094446]
 [ 3.1130888 ]]
[[0.03329056]
 [3.01194969]]
```

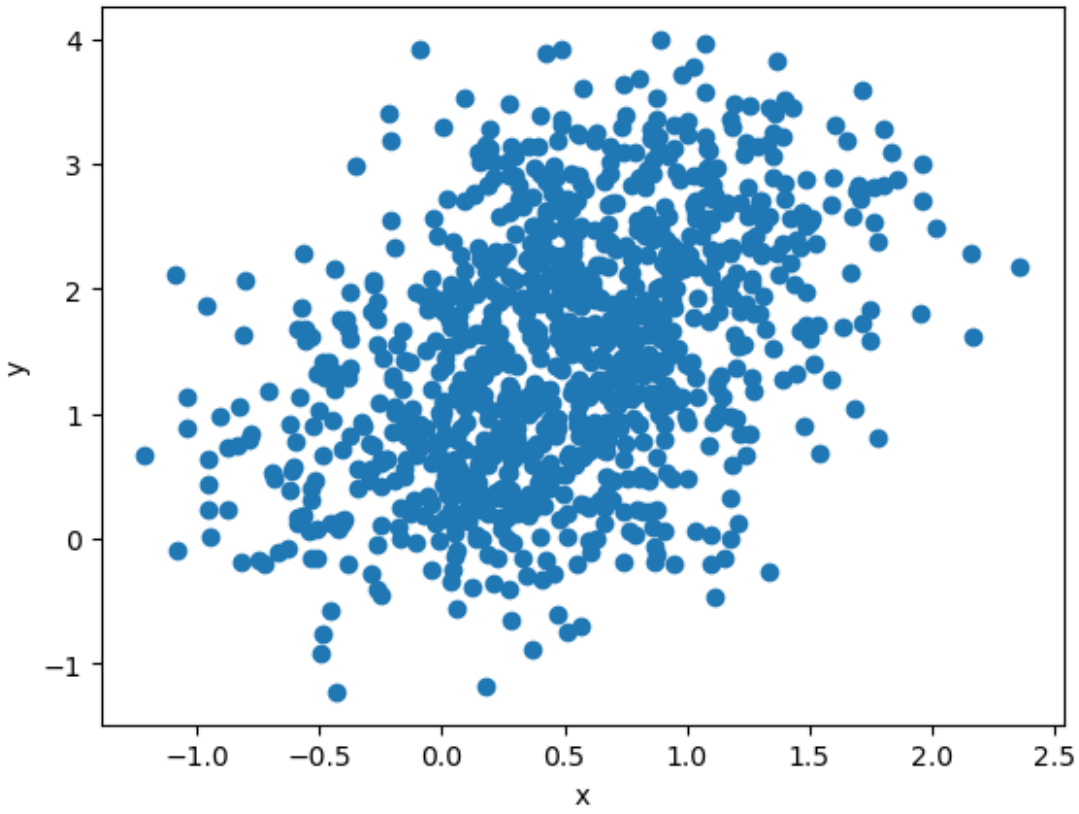


Error in x and y

We will now examine the case where our data consists of noisy estimates of both x and y. Run the following cell to see how the data looks when there is error in both.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
base = np.arange(0.001, 1, 0.001).reshape(-1, 1)
c = 0.5
X = base + np.random.normal(loc=[0], scale=c, size=base.shape)
y = 3 * base + np.random.normal(loc=[0], scale=c, size=base.shape)

fig = plt.figure()
plt.scatter(X, y)
plt.xlabel("x")
plt.ylabel("y")

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
        EO_TEXT,
        transform=fig.transFigure,
```

```
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA * 0.8,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT,
    )

plt.show()
```



In the below cell, we graph the predicted PCA and LR slopes on the vertical axis against the value of c on the horizontal axis. Note that the graph you get might not look exactly like the TA version and that is fine because you might have randomized the noise slightly differently than how we did it.

```
################################
### DO NOT CHANGE THIS CELL ###
################################
pca_slope_values = []
linreg_slope_values = []
c_values = []
s_idx = 0
```

```
for i in range(30):
    for c in np.arange(0, 1.05, 0.05):
        # Calculate pca_slope_value (psv) and lr_slope_value (lsv),
notice x_noise = True
        psv, lsv = addNoise(c, x_noise=True, seed=s_idx)

        # Append pca and lr slope values to list for plot function
        pca_slope_values.append(psv)
        linreg_slope_values.append(lsv)

        # Append c value to list for plot function
        c_values.append(c)

        # Increment random seed index
        s_idx += 1

fig = plt.figure()
plt.scatter(c_values, pca_slope_values, c="r")
plt.scatter(c_values, linreg_slope_values, c="b")
plt.xlabel("c")
plt.ylabel("slope")

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA * 0.5,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT,
    )

plt.show()
```

```
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03526142]
 [2.91874573]]
[[0.17059934]
 [2.68604519]]
[[0.30947436]
 [2.34772221]]
[[0.42473086]
 [2.10238862]]
[[0.60458945]
```

```
 [1.7423829 ]]
[[0.77732046]
 [1.51294021]]
[[0.90281067]
 [1.19734642]]
[[0.98804766]
 [1.02659893]]
[[1.08856579]
 [0.7614957 ]]
[[1.08299768]
 [0.81592036]]
[[1.20480884]
 [0.63552409]]
[[1.16112046]
 [0.59173937]]
[[1.28422784]
 [0.46863033]]
[[1.34984627]
 [0.4583038 ]]
[[1.33918785]
 [0.34470047]]
[[1.35447305]
 [0.35611685]]
[[1.39971399]
 [0.24153257]]
[[1.42573372]
 [0.21291581]]
[[1.38164757]
 [0.29991245]]
[[1.39771009]
 [0.25468812]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03297911]
 [2.92056316]]
[[0.15512973]
 [2.70865115]]
[[0.31491445]
 [2.33908262]]
[[0.50250361]
 [1.95118436]]
[[0.61659747]
 [1.744486  ]]
[[0.8186055 ]
 [1.40043258]]
[[0.85474171]
 [1.26542529]]
[[0.99404451]
 [1.00805958]]
```

```
[[1.05859785]
 [0.89834208]]
[[1.11398796]
 [0.78549224]]
[[1.15381214]
 [0.67710192]]
[[1.19542389]
 [0.62816606]]
[[1.24261074]
 [0.58202016]]
[[1.34865015]
 [0.35826646]]
[[1.31344989]
 [0.3639787 ]]
[[1.30627141]
 [0.34124624]]
[[1.41041802]
 [0.26103599]]
[[1.37958855]
 [0.27017972]]
[[1.39050914]
 [0.28557316]]
[[1.41576116]
 [0.24614757]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04517722]
 [2.9151194 ]]
[[0.15495587]
 [2.69108491]]
[[0.33036571]
 [2.35883273]]
[[0.5077058 ]
 [1.99399228]]
[[0.62887011]
 [1.71038864]]
[[0.77519677]
 [1.38205847]]
[[0.88291566]
 [1.2280253 ]]
[[0.99362905]
 [1.08473444]]
[[1.0395291 ]
 [0.87637551]]
[[1.14262027]
 [0.69522337]]
[[1.20625908]
 [0.5965095 ]]
[[1.14318226]
```

```
 [0.67578177]]
[[1.26246827]
 [0.47467253]]
[[1.3400089 ]
 [0.40807617]]
[[1.33640292]
 [0.37175644]]
[[1.37796085]
 [0.24194823]]
[[1.34763039]
 [0.27367569]]
[[1.32028989]
 [0.27918015]]
[[1.35530266]
 [0.19920862]]
[[1.3363449 ]
 [0.34494957]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.06185853]
 [2.89538513]]
[[0.1833985 ]
 [2.65015784]]
[[0.28201617]
 [2.39951847]]
[[0.50655335]
 [2.03553089]]
[[0.63068023]
 [1.71517375]]
[[0.80397205]
 [1.41712754]]
[[0.8959485 ]
 [1.18956176]]
[[0.96695284]
 [1.0508706 ]]
[[1.09527971]
 [0.84761995]]
[[1.13176798]
 [0.79880804]]
[[1.14828967]
 [0.66313503]]
[[1.19333056]
 [0.60502614]]
[[1.28746124]
 [0.48724791]]
[[1.30026839]
 [0.45079401]]
[[1.31876844]
 [0.4061595 ]]
```

```
[[1.33907362]
 [0.32659717]]
[[1.29941502]
 [0.3086141 ]]
[[1.40734381]
 [0.25983484]]
[[1.36939575]
 [0.2554502 ]]
[[1.35581651]
 [0.29124161]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.06008847]
 [2.88583543]]
[[0.15037429]
 [2.68624047]]
[[0.2913147 ]
 [2.35927263]]
[[0.4842117 ]
 [1.98682116]]
[[0.61532397]
 [1.72760194]]
[[0.78563745]
 [1.36778354]]
[[0.88113377]
 [1.23498589]]
[[0.92766054]
 [1.09608865]]
[[1.05888419]
 [0.90944881]]
[[1.07942706]
 [0.80336167]]
[[1.20362037]
 [0.62404434]]
[[1.18629709]
 [0.57042609]]
[[1.25424355]
 [0.53076137]]
[[1.2370568 ]
 [0.49906208]]
[[1.32081197]
 [0.42665321]]
[[1.43539216]
 [0.27139286]]
[[1.47975287]
 [0.19792767]]
[[1.35075829]
 [0.30426341]]
[[1.39851404]
```

```
 [0.29355101]]
[[1.3585118 ]
 [0.26054417]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.05808961]
 [2.89390281]]
[[0.17724955]
 [2.65782232]]
[[0.2762967 ]
 [2.45805924]]
[[0.51529489]
 [1.96011252]]
[[0.64264942]
 [1.70022265]]
[[0.80338623]
 [1.40202769]]
[[0.90831006]
 [1.22848415]]
[[0.97077941]
 [1.04871389]]
[[1.07049437]
 [0.89892414]]
[[1.14093164]
 [0.74592877]]
[[1.20490019]
 [0.65444187]]
[[1.26173293]
 [0.470392  ]]
[[1.2815824 ]
 [0.42140819]]
[[1.27354389]
 [0.44384971]]
[[1.30689083]
 [0.36880395]]
[[1.35834515]
 [0.29641169]]
[[1.37168883]
 [0.30720803]]
[[1.4338895 ]
 [0.18171635]]
[[1.46340487]
 [0.23647787]]
[[1.49716599]
 [0.15480865]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04994501]
 [2.90739551]]
```

```
[[0.1339126 ]
 [2.70237523]]
[[0.37535231]
 [2.3457527 ]]
[[0.4472388 ]
 [2.14436715]]
[[0.64548993]
 [1.70474033]]
[[0.72736568]
 [1.50156944]]
[[0.90671755]
 [1.1888271 ]]
[[0.96549276]
 [1.07015145]]
[[0.9874335 ]
 [0.95248307]]
[[1.05702463]
 [0.76316233]]
[[1.13909459]
 [0.65292659]]
[[1.15507964]
 [0.63630818]]
[[1.29909606]
 [0.41098052]]
[[1.30237737]
 [0.43462863]]
[[1.36295431]
 [0.35253004]]
[[1.38386588]
 [0.23070805]]
[[1.38656459]
 [0.32740547]]
[[1.40613219]
 [0.19107299]]
[[1.33421074]
 [0.32667782]]
[[1.35147124]
 [0.22772308]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03122348]
 [2.93316057]]
[[0.14624342]
 [2.72152144]]
[[0.31758924]
 [2.37793   ]]
[[0.4756857 ]
 [2.01533877]]
[[0.66184454]
```

```
 [1.69460786]]
[[0.75468363]
 [1.4986581 ]]
[[0.9048576 ]
 [1.17925028]]
[[0.97305957]
 [1.07374708]]
[[1.06122464]
 [0.91483795]]
[[1.15077767]
 [0.69488103]]
[[1.17087179]
 [0.65068531]]
[[1.18657885]
 [0.60420951]]
[[1.31227468]
 [0.39590345]]
[[1.27765957]
 [0.4639824 ]]
[[1.26987557]
 [0.3631508 ]]
[[1.36162024]
 [0.26047426]]
[[1.36042704]
 [0.24968689]]
[[1.4656567]
 [0.2000098]]
[[1.3658849 ]
 [0.32626975]]
[[1.28265294]
 [0.3554288 ]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04743948]
 [2.9024436 ]]
[[0.13889384]
 [2.70758881]]
[[0.31665678]
 [2.34192229]]
[[0.48238371]
 [2.04651096]]
[[0.62064157]
 [1.74611194]]
[[0.78367214]
 [1.45582374]]
[[0.90410302]
 [1.21760318]]
[[0.93477984]
 [1.06910785]]
```

```
[[1.048591  ]
 [0.94943691]]
[[1.10064051]
 [0.76561853]]
[[1.18568479]
 [0.60573557]]
[[1.2412731 ]
 [0.50168412]]
[[1.24027314]
 [0.56453804]]
[[1.27776834]
 [0.44682548]]
[[1.33251827]
 [0.43278203]]
[[1.33071612]
 [0.36212327]]
[[1.3340535 ]
 [0.35808686]]
[[1.42843934]
 [0.28973273]]
[[1.36639427]
 [0.31287443]]
[[1.45146682]
 [0.20987005]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03865758]
 [2.91959208]]
[[0.1602983 ]
 [2.69503735]]
[[0.29587781]
 [2.37737414]]
[[0.48538178]
 [2.02951094]]
[[0.59763554]
 [1.7657522 ]]
[[0.75809558]
 [1.47101629]]
[[0.88511122]
 [1.2365915 ]]
[[0.9748358 ]
 [1.02092417]]
[[1.06865325]
 [0.88033888]]
[[1.16925374]
 [0.72314138]]
[[1.19074745]
 [0.62877331]]
[[1.25401791]
```

```
 [0.58767947]]
[[1.27197551]
 [0.47339104]]
[[1.26033625]
 [0.42418693]]
[[1.26793825]
 [0.41127927]]
[[1.33023291]
 [0.34474655]]
[[1.3312626 ]
 [0.29313543]]
[[1.33823034]
 [0.30011995]]
[[1.32697201]
 [0.30369379]]
[[1.40285785]
 [0.19439627]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04564016]
 [2.90439631]]
[[0.13207045]
 [2.70213001]]
[[0.33324314]
 [2.38386793]]
[[0.54025554]
 [1.98035103]]
[[0.68689001]
 [1.67007734]]
[[0.75724616]
 [1.56096069]]
[[0.9141263 ]
 [1.22862373]]
[[0.98677391]
 [1.06555777]]
[[1.06424675]
 [0.89086026]]
[[1.15102253]
 [0.69987868]]
[[1.19344646]
 [0.63315467]]
[[1.17580538]
 [0.63244403]]
[[1.28068947]
 [0.44010445]]
[[1.29276232]
 [0.42604623]]
[[1.31316498]
 [0.38500184]]
```

```
[[1.3645385 ]
 [0.32264795]]
[[1.34631037]
 [0.27392442]]
[[1.33564871]
 [0.28042409]]
[[1.33350785]
 [0.23251759]]
[[1.45520877]
 [0.21314761]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03716298]
 [2.92291022]]
[[0.16165668]
 [2.6854678 ]]
[[0.33455032]
 [2.33178482]]
[[0.51522252]
 [1.9828982 ]]
[[0.65290625]
 [1.6731522 ]]
[[0.80884928]
 [1.46531583]]
[[0.89893563]
 [1.21069824]]
[[1.00633339]
 [1.01921174]]
[[1.06442896]
 [0.89964506]]
[[1.12726917]
 [0.77255909]]
[[1.11640054]
 [0.63758438]]
[[1.20244827]
 [0.52313272]]
[[1.25669892]
 [0.53248893]]
[[1.25949874]
 [0.56401637]]
[[1.32019431]
 [0.34507414]]
[[1.26772599]
 [0.37760316]]
[[1.38598092]
 [0.2671193 ]]
[[1.35634249]
 [0.32154486]]
[[1.36952535]
 [0.22892606]]
```

```
[[1.43188951]
 [0.21003208]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03717274]
 [2.92348701]]
[[0.14789377]
 [2.71669415]]
[[0.30817664]
 [2.38316088]]
[[0.50672982]
 [1.96058949]]
[[0.64858515]
 [1.76810632]]
[[0.80866854]
 [1.39467831]]
[[0.87800183]
 [1.25624841]]
[[1.00363038]
 [0.97090998]]
[[1.09272006]
 [0.89909064]]
[[1.142555  ]
 [0.74072563]]
[[1.13153397]
 [0.68312363]]
[[1.18676296]
 [0.58440378]]
[[1.2794115 ]
 [0.47101824]]
[[1.22666679]
 [0.47212909]]
[[1.35083107]
 [0.3148149 ]]
[[1.39341233]
 [0.29184196]]
[[1.37591886]
 [0.35453901]]
[[1.42014873]
 [0.34366902]]
[[1.45448675]
 [0.24179091]]
[[1.24451551]
 [0.26272368]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.05866718]
 [2.88773702]]
[[0.12210157]
```

```
 [2.72562382]]
[[0.30607501]
 [2.37442708]]
[[0.47908488]
 [2.05124221]]
[[0.60552521]
 [1.79488907]]
[[0.79044554]
 [1.3794268 ]]
[[0.8416108 ]
 [1.29213228]]
[[0.99036664]
 [1.07718092]]
[[1.10387944]
 [0.89278078]]
[[1.12984518]
 [0.72606053]]
[[1.16181262]
 [0.66051578]]
[[1.23438385]
 [0.50976494]]
[[1.24806387]
 [0.53097876]]
[[1.29264227]
 [0.37996619]]
[[1.27590583]
 [0.41691182]]
[[1.37173929]
 [0.39927604]]
[[1.28631704]
 [0.32871382]]
[[1.35051117]
 [0.3841973 ]]
[[1.47339833]
 [0.21398997]]
[[1.32103191]
 [0.28296996]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.05974756]
 [2.90663322]]
[[0.16803939]
 [2.69179427]]
[[0.29782422]
 [2.37577276]]
[[0.52039749]
 [1.97803997]]
[[0.65678882]
 [1.69596313]]
```

```
[[0.77901609]
 [1.43922444]]
[[0.87958082]
 [1.2725892 ]]
[[0.94527226]
 [1.10109684]]
[[0.98478083]
 [0.97025077]]
[[1.10893553]
 [0.79165034]]
[[1.22034918]
 [0.55522369]]
[[1.20700172]
 [0.62206556]]
[[1.20540216]
 [0.55844125]]
[[1.36955073]
 [0.38042083]]
[[1.31196105]
 [0.32810094]]
[[1.31736519]
 [0.38285845]]
[[1.37832559]
 [0.27635329]]
[[1.39796426]
 [0.25077905]]
[[1.3478047 ]
 [0.20665371]]
[[1.4872933]
 [0.1127482]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.060089  ]
 [2.88971764]]
[[0.1552645 ]
 [2.66408812]]
[[0.31940339]
 [2.35855086]]
[[0.49436453]
 [2.00825355]]
[[0.69125219]
 [1.66295835]]
[[0.76266778]
 [1.41866452]]
[[0.89719989]
 [1.25704592]]
[[0.99413936]
 [1.02214723]]
[[1.00156246]
```

```
 [0.91772266]]
[[1.14153427]
 [0.73055798]]
[[1.21342759]
 [0.53534231]]
[[1.23338409]
 [0.49905893]]
[[1.30520418]
 [0.4083104 ]]
[[1.21698507]
 [0.45150511]]
[[1.31284916]
 [0.44529692]]
[[1.32866141]
 [0.31663117]]
[[1.35609058]
 [0.30387   ]]
[[1.37609357]
 [0.29391258]]
[[1.47776447]
 [0.24868885]]
[[1.2711124]
 [0.3422219]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04125037]
 [2.92040534]]
[[0.16354656]
 [2.68764884]]
[[0.32122572]
 [2.39830641]]
[[0.51917165]
 [2.02954198]]
[[0.64622273]
 [1.71315227]]
[[0.74949445]
 [1.48017011]]
[[0.92564245]
 [1.18441373]]
[[0.99806698]
 [1.07233545]]
[[1.05366572]
 [0.88320903]]
[[1.11524224]
 [0.6949204 ]]
[[1.10408922]
 [0.71211418]]
[[1.23047557]
 [0.52781654]]
```

```
[[1.2940336 ]
 [0.48572272]]
[[1.3299174]
 [0.4180338]]
[[1.29035959]
 [0.41213325]]
[[1.30395806]
 [0.38079757]]
[[1.34668785]
 [0.31552265]]
[[1.35371513]
 [0.19832807]]
[[1.37897668]
 [0.24268603]]
[[1.44477318]
 [0.19602296]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04301375]
 [2.90874784]]
[[0.1587867 ]
 [2.68108464]]
[[0.31059033]
 [2.40223299]]
[[0.52245505]
 [1.95428786]]
[[0.621496  ]
 [1.72738294]]
[[0.67995066]
 [1.56436129]]
[[0.91619911]
 [1.19788176]]
[[0.9696611 ]
 [1.11497492]]
[[1.1187438 ]
 [0.78831711]]
[[1.17942479]
 [0.69119902]]
[[1.16809636]
 [0.61321675]]
[[1.25365859]
 [0.57044346]]
[[1.2122234]
 [0.5545619]]
[[1.2139454 ]
 [0.50755176]]
[[1.32339547]
 [0.38638911]]
[[1.25853243]
```

```
 [0.41838005]]
[[1.36832737]
 [0.26763865]]
[[1.41333752]
 [0.1786874 ]]
[[1.46435192]
 [0.26182228]]
[[1.33525292]
 [0.24412289]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03904285]
 [2.92418845]]
[[0.17764371]
 [2.65527088]]
[[0.32833109]
 [2.38010012]]
[[0.47951044]
 [2.0142886 ]]
[[0.67962273]
 [1.65484215]]
[[0.74836311]
 [1.46571067]]
[[0.84913403]
 [1.23757922]]
[[0.97081553]
 [1.057951  ]]
[[1.05143324]
 [0.92302086]]
[[1.08198345]
 [0.73398841]]
[[1.08888914]
 [0.70295081]]
[[1.24186822]
 [0.52600352]]
[[1.2869785 ]
 [0.49102063]]
[[1.35189898]
 [0.36172049]]
[[1.35165828]
 [0.3533014 ]]
[[1.27971794]
 [0.39269783]]
[[1.38073363]
 [0.31369633]]
[[1.37690032]
 [0.27359025]]
[[1.3353536 ]
 [0.26733595]]
```

```
[[1.38683307]
 [0.22098545]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.03664847]
 [2.92267924]]
[[0.18066271]
 [2.64403957]]
[[0.33147551]
 [2.31045303]]
[[0.5201933]
 [2.0066715]]
[[0.62268086]
 [1.72603199]]
[[0.7853939 ]
 [1.43627692]]
[[0.91757352]
 [1.17124752]]
[[0.91993326]
 [1.08100798]]
[[1.0591831 ]
 [0.91150244]]
[[1.14435336]
 [0.74617976]]
[[1.13743526]
 [0.69483308]]
[[1.19664167]
 [0.6434366 ]]
[[1.26075733]
 [0.47971014]]
[[1.29115776]
 [0.42165341]]
[[1.30570733]
 [0.43761064]]
[[1.35964439]
 [0.28464756]]
[[1.33616772]
 [0.28136485]]
[[1.24135456]
 [0.30997546]]
[[1.33149383]
 [0.29448209]]
[[1.29149661]
 [0.2610683 ]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.0363582 ]
 [2.93642212]]
[[0.18337254]
```

```
 [2.65504794]]
[[0.33343491]
 [2.32733976]]
[[0.50290204]
 [2.03617664]]
[[0.65878179]
 [1.6823125 ]]
[[0.81783617]
 [1.4210724 ]]
[[0.89813214]
 [1.17747761]]
[[1.00653439]
 [1.02794358]]
[[1.0360721 ]
 [0.83702399]]
[[1.15817052]
 [0.73718711]]
[[1.18447838]
 [0.60805203]]
[[1.19726283]
 [0.56705877]]
[[1.31160381]
 [0.50128043]]
[[1.32455131]
 [0.44285443]]
[[1.34686963]
 [0.38081265]]
[[1.30703244]
 [0.39934138]]
[[1.30333267]
 [0.4158623 ]]
[[1.37984708]
 [0.28036912]]
[[1.3381569 ]
 [0.28612715]]
[[1.38336989]
 [0.27501919]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.06164716]
 [2.87238235]]
[[0.17488456]
 [2.6570903 ]]
[[0.28897898]
 [2.37708942]]
[[0.52279461]
 [2.0027172 ]]
[[0.65277822]
 [1.6809074 ]]
```

```
[[0.75802047]
 [1.46042577]]
[[0.88913163]
 [1.28199308]]
[[1.02984206]
 [1.03845949]]
[[1.09387908]
 [0.77806679]]
[[1.12196428]
 [0.75837722]]
[[1.16318314]
 [0.68437126]]
[[1.1715652 ]
 [0.64360761]]
[[1.2757958 ]
 [0.48213767]]
[[1.32653032]
 [0.43350968]]
[[1.31623314]
 [0.37955886]]
[[1.34199931]
 [0.35500162]]
[[1.33943984]
 [0.28512662]]
[[1.34755561]
 [0.25462855]]
[[1.37258124]
 [0.29238941]]
[[1.44498231]
 [0.21295286]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.02424342]
 [2.93464385]]
[[0.13920863]
 [2.72729916]]
[[0.34204847]
 [2.33753166]]
[[0.4655034 ]
 [2.07645022]]
[[0.65641495]
 [1.70937179]]
[[0.78786438]
 [1.45480653]]
[[0.91883597]
 [1.21012447]]
[[0.93431157]
 [1.07412168]]
[[1.04967518]
```

```
 [0.89893066]]
[[1.1282734 ]
 [0.72091509]]
[[1.22876389]
 [0.52160748]]
[[1.22407535]
 [0.53886443]]
[[1.2053371 ]
 [0.59203073]]
[[1.28870517]
 [0.42599226]]
[[1.31034729]
 [0.37946278]]
[[1.28976734]
 [0.41461993]]
[[1.34692433]
 [0.2000633 ]]
[[1.33504193]
 [0.25611328]]
[[1.42018118]
 [0.19392945]]
[[1.42350425]
 [0.16296258]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.05680428]
 [2.88935973]]
[[0.11479345]
 [2.74881065]]
[[0.34935723]
 [2.32935412]]
[[0.48592828]
 [2.05230126]]
[[0.67008742]
 [1.68008134]]
[[0.78265835]
 [1.44447954]]
[[0.94810724]
 [1.14308294]]
[[0.97691742]
 [1.06218784]]
[[1.05779919]
 [0.91431657]]
[[1.11000124]
 [0.77676918]]
[[1.20744431]
 [0.64010242]]
[[1.24583911]
 [0.52284785]]
```

```
[[1.21264449]
 [0.51726673]]
[[1.27106226]
 [0.39201322]]
[[1.28395862]
 [0.38479599]]
[[1.28165868]
 [0.37780334]]
[[1.28931519]
 [0.30110331]]
[[1.42927035]
 [0.190002  ]]
[[1.38650043]
 [0.25443002]]
[[1.37876724]
 [0.17614395]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.02712969]
 [2.93876295]]
[[0.15583622]
 [2.68477074]]
[[0.31427163]
 [2.38024487]]
[[0.45511131]
 [2.0385426 ]]
[[0.66014363]
 [1.70978458]]
[[0.77041214]
 [1.41104748]]
[[0.83513904]
 [1.27788597]]
[[0.96451243]
 [1.0695223 ]]
[[1.05667178]
 [0.8861621 ]]
[[1.17268575]
 [0.70328588]]
[[1.11823561]
 [0.73106213]]
[[1.19993736]
 [0.50705063]]
[[1.2952171 ]
 [0.48545231]]
[[1.26015984]
 [0.44566235]]
[[1.26990991]
 [0.37642209]]
[[1.32610065]
 [0.40075598]]
```

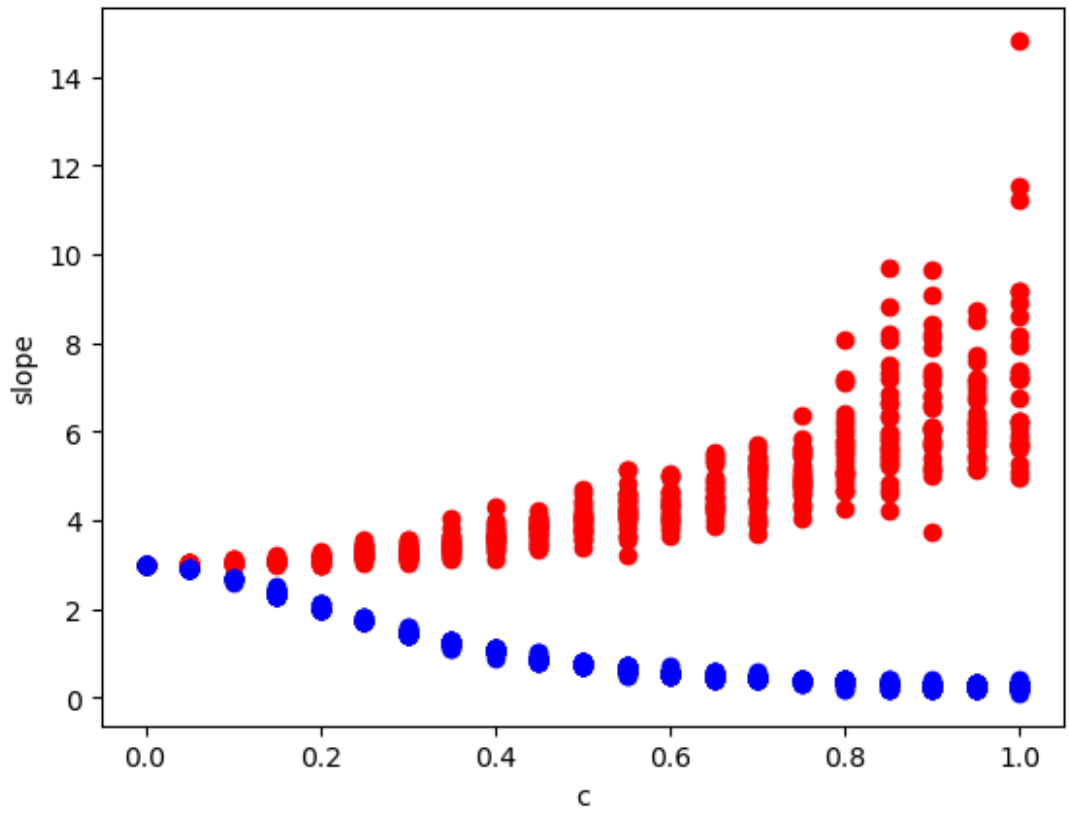
```
[[1.34371547]
 [0.34410646]]
[[1.40372795]
 [0.24543831]]
[[1.39712385]
 [0.21874764]]
[[1.36596126]
 [0.26732045]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04886881]
 [2.9050601 ]]
[[0.16348635]
 [2.70714791]]
[[0.2745057 ]
 [2.39213884]]
[[0.49406706]
 [2.02737348]]
[[0.63657477]
 [1.70347192]]
[[0.77841987]
 [1.42518255]]
[[0.8969649 ]
 [1.20795927]]
[[0.92670087]
 [1.08015521]]
[[1.13482977]
 [0.76081696]]
[[1.16096581]
 [0.7507976 ]]
[[1.19633808]
 [0.69118926]]
[[1.18730699]
 [0.53410181]]
[[1.20490323]
 [0.52144478]]
[[1.23880875]
 [0.48230424]]
[[1.2579776 ]
 [0.40824336]]
[[1.34977392]
 [0.30328768]]
[[1.31786504]
 [0.31273816]]
[[1.29440195]
 [0.32813041]]
[[1.37190913]
 [0.26496511]]
[[1.37637267]
```

```
 [0.31786233]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04341319]
 [2.90875381]]
[[0.12250463]
 [2.71695799]]
[[0.32479419]
 [2.35094476]]
[[0.50185009]
 [2.03822644]]
[[0.62266777]
 [1.70678528]]
[[0.77191691]
 [1.49058653]]
[[0.86143518]
 [1.27191985]]
[[0.9719428 ]
 [1.04938686]]
[[1.09422304]
 [0.86864263]]
[[1.1202872 ]
 [0.75955782]]
[[1.17752913]
 [0.63020704]]
[[1.18845967]
 [0.64789354]]
[[1.34032667]
 [0.43686456]]
[[1.31286084]
 [0.40926962]]
[[1.26555718]
 [0.44216778]]
[[1.34989426]
 [0.3982043 ]]
[[1.34483367]
 [0.29000482]]
[[1.40644445]
 [0.28428844]]
[[1.41990029]
 [0.20650417]]
[[1.44201966]
 [0.19893207]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.06285991]
 [2.8832898 ]]
[[0.15557811]
 [2.71047892]]
```

```
[[0.27086895]
 [2.4277262 ]]
[[0.47667999]
 [1.99554519]]
[[0.65334987]
 [1.74771033]]
[[0.83321786]
 [1.37097277]]
[[0.85679618]
 [1.31970206]]
[[1.02123131]
 [0.9424604 ]]
[[1.05538469]
 [0.88087169]]
[[1.06728101]
 [0.8078735 ]]
[[1.16533453]
 [0.63026595]]
[[1.22727179]
 [0.57326538]]
[[1.3206171 ]
 [0.40954102]]
[[1.3262796 ]
 [0.38415627]]
[[1.39394043]
 [0.35624757]]
[[1.35893521]
 [0.29514185]]
[[1.34867662]
 [0.3138374 ]]
[[1.39289828]
 [0.23049849]]
[[1.40182112]
 [0.25728297]]
[[1.4513405 ]
 [0.15934444]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.01884609]
 [2.94957031]]
[[0.1584839 ]
 [2.68334807]]
[[0.37740741]
 [2.27508605]]
[[0.48992036]
 [2.00580889]]
[[0.63736816]
 [1.70871163]]
[[0.80116568]
```

```
 [1.4638407 ]]
[[0.92553373]
 [1.11374076]]
[[0.97916941]
 [1.04579095]]
[[1.0328467 ]
 [0.86947027]]
[[1.02526449]
 [0.83574836]]
[[1.21638879]
 [0.60179436]]
[[1.20625791]
 [0.59634115]]
[[1.22575936]
 [0.50366106]]
[[1.30892582]
 [0.39336384]]
[[1.2794596 ]
 [0.37737675]]
[[1.35053125]
 [0.30695601]]
[[1.38613707]
 [0.18597964]]
[[1.36287346]
 [0.27607473]]
[[1.37431906]
 [0.28502344]]
[[1.42287707]
 [0.2110249 ]]
[[9.99200722e-16]
 [3.00000000e+00]]
[[0.04938301]
 [2.91621853]]
[[0.15774615]
 [2.6969745 ]]
[[0.26782085]
 [2.44419557]]
[[0.53652276]
 [1.96485454]]
[[0.57529918]
 [1.80632804]]
[[0.77632923]
 [1.40559691]]
[[0.8863069 ]
 [1.24956038]]
[[1.04203135]
 [1.00894415]]
[[1.05016879]
 [0.90879702]]
```

```
[[1.14574714]
 [0.7010546 ]]
[[1.20267734]
 [0.6057998 ]]
[[1.28729101]
 [0.5721187 ]]
[[1.31499901]
 [0.45973071]]
[[1.31094822]
 [0.43787513]]
[[1.38738897]
 [0.35419137]]
[[1.31882305]
 [0.36137337]]
[[1.35201547]
 [0.33176105]]
[[1.44900012]
 [0.25107   ]]
[[1.34404398]
 [0.28588401]]
[[1.42544564]
 [0.18362556]]
```



5.3. Analysis [5 pts] **[W]**

Based on your observations from previous subsections answer the following questions about the two cases (error in Y and error in both X and Y) in 2-3 lines.

NOTE:

1. You don't need to provide a mathematical proof for this question.
2. Understanding how PCA and Linear Regression work should help you decipher which case was better for which algorithm. Base your answer on this understanding of how either algorithms works.

QUESTIONS:

1. Based on the obtained plots, how can you determine which technique (PCA/Linear regression) is performing better in comparison? (1 Pt)
2. In case-1 where there is error in Y which technique gave better performance and why do you think it performed better in this case? (2 Pts)
3. In case-2 where there is error in both X and Y which technique gave better performance and why do you think it performed better in this case? (2 Pts)

Answer ...

1. When looking at linear regression, if the residuals are randomly distributed around zero, linear regression is likely a good fit, while for PCA, if the scatter plot of data points along the principal components reveals a structured pattern, PCA is a good fit.
2. Linear regression performs better when error is added to just Y. This is because, even when error is added to Y, it does not change the relationship of X and Y since the relationship between X and Y is linear,
3. When error is added to both X and Y, it increases the varinace in both X and Y similarly, which allows PCA to extract the directions leading to a better performance.

Q6 Feature Reduction Implementation [25pts Bonus for All] **[P]** | **[W]**

6.1 Implementation [18 Points] **[P]**

Feature selection is an integral aspect of machine learning. It is the process of selecting a subset of relevant features that are to be used as the input for the machine learning task. Feature selection may lead to simpler models for easier interpretation, shorter training times, avoidance of the curse of dimensionality, and better generalization by reducing overfitting.

In the feature_reduction.py file, complete the following functions:

- forward_selection
- backward_elimination

These functions should each output a list of features.

Forward Selection:

In forward selection, we start with a null model, start fitting the model with one individual feature at a time, and select the feature with the minimum p-value. We continue to do this until we have a set of features where one feature's p-value is less than the confidence level.

Steps to implement it:

1. Choose a significance level (given to you).
2. Fit all possible simple regression models by considering one feature at a time.
3. Select the feature with the lowest p-value.
4. Fit all possible models with one extra feature added to the previously selected feature(s).
5. Select the feature with the minimum p-value again. if p_value < significance, go to Step 4. Otherwise, terminate.

Backward Elimination:

In backward elimination, we start with a full model, and then remove the insignificant feature with the highest p-value (that is greater than the significance level). We continue to do this until we have a final set of significant features.

Steps to implement it:

1. Choose a significance level (given to you).
2. Fit a full model including all the features.
3. Select the feature with the highest p-value. If (p-value > significance level), go to Step 4, otherwise terminate.
4. Remove the feature under consideration.
5. Fit a model without this feature. Repeat entire process from Step 3 onwards.

HINT 1: The p-value is known as the observed significance value for a null hypothesis. In our case, the p-value of a feature is associated with the hypothesis $H_0 : \beta_j = 0$. If $\beta_j = 0$, then this feature contributes no predictive power to our model and should be dropped. We reject the null hypothesis if the p-value is smaller than our significance level. More briefly, a p-value is a measure of how much the given feature significantly represents an observed change. A lower p-value represents higher significance. Some more information about p-values can be found here: https://towardsdatascience.com/what-is-a-p-value-b9e6c207247f

HINT 2: For this function, you will have to install statsmodels if not installed already. To do this, run `pip install statsmodels` in command line/terminal. In the case that you are using an Anaconda environment, run `conda install -c conda-forge statsmodels` in the command line/terminal. For more information about installation, refer to https://www.statsmodels.org/stable/install.html. The statsmodels library is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. You will have to use this library to choose a regression model to fit your data against. Some more information about this module can be found here: https://www.statsmodels.org/stable/index.html

HINT 3: For step 2 in each of the forward and backward selection functions, you can use the `sm.OLS` function as your regression model. Also, do not forget to add a bias to your regression model. A function that may help you is the `sm.add_constants` function.

TIP 4: You should be able to implement these function using only the libraries provided in the cell below.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from feature_reduction import FeatureReduction

################################
### DO NOT CHANGE THIS CELL ###
################################
bc_dataset = load_breast_cancer()
bc = pd.DataFrame(bc_dataset.data, columns=bc_dataset.feature_names)
bc["Diagnosis"] = bc_dataset.target
# print(bc)
X = bc.drop("Diagnosis", axis=1)
y = bc["Diagnosis"]
featureselection = FeatureReduction()
# Run the functions to make sure two lists are generated, one for each
method
print(
    "Features selected by forward selection:",
FeatureReduction.forward_selection(X, y)
)
print(
    "Features selected by backward elimination:",
    FeatureReduction.backward_elimination(X, y),
)
```

```
Features selected by forward selection: ['worst concave points',
'worst radius', 'worst texture', 'worst area', 'smoothness error',
'worst symmetry', 'compactness error', 'radius error', 'worst fractal
dimension', 'mean compactness', 'mean concave points', 'worst
concavity', 'concavity error', 'area error']
Features selected by backward elimination: ['mean radius', 'mean
compactness', 'mean concave points', 'radius error', 'smoothness
error', 'concavity error', 'concave points error', 'worst radius',
'worst texture', 'worst area', 'worst concavity', 'worst symmetry',
'worst fractal dimension']
```

6.2 Feature Selection - Discussion [7pts] **[W]**

Question 6.2.1:

We have seen two regression methods namely Lasso and Ridge regression earlier in this assignment. Another extremely important and common use-case of these methods is to perform feature selection. Considering there are no restrictions set on the dataset, according to you, which of these two methods is more appropriate for feature selection generally (choose one method)? Why? **(3 pts)**

Answer ...

In feature selection, we want to retain the most significant features and get rid of the ones that do not help our model. Lasso regression will be more appropriate since it focuses on sparsity and assigns zeros as coefficients for some features, thus getting rid of the features that are not positively contributing to the performance of our model.

Question 6.2.2:

We have seen that we use different subsets of features to get different regression models. These models depend on the relevant features that we have selected. Using forward selection, what fraction of the total possible models can we explore? Assume that the total number of features that we have at our disposal is N. Remember that in stepwise feature selection (like forward selection and backward elimination), we always include an intercept in our model, so you only need to consider the N features. **(4 pts)**

Answer ...

There are 2^N possible total models that can be explored. Number of models that can be explored during forward selection are

$$N+(N-1)+(N-2)+\ldots.+1=\frac{N(N+1)}{2}$$

Q7: Netflix Movie Recommendation Problem Solved using SVD [10pts Bonus for All] **[P]**

Let us try to tackle the famous problem of movie recommendation using just our SVD functions that we have implemented. We are given a table of reviews that 600+ users have provided for close to 10,000 different movies. Our challenge is to predict how much a user would rate a movie that they have not seen (or rated) yet. Once we have these ratings, we would then be able to predict which movies to recommend to that user.

Understanding How SVD Helps in Movie Recommendation

We are given a dataset of user-movie ratings (R) that looks like the following:

Ratings in the matrix range from 1-5. In addition, the matrix contains `nan` wherever there is no rating provided by the user for the corresponding movie. One simple way to utilize this matrix to predict movie ratings for a given user-movie pair would be to fill in each row / column with the average rating for that row / column. For example: For each movie, if any rating is missing, we could just fill in the average value of all available ratings and expect this to be around the actual / expected rating.

While this may sound like a good approximation, it turns out that by just using SVD we can improve the accuracy of the predicted rating.

How does SVD fit into this picture?

Recall how we previously used SVD to compress images by throwing out less important information. We could apply the same idea to our above matrix (R) to generate another matrix ($R_¿$) which will provide the same information, i.e ratings for any user-movie pairs but by combining only the most important features.

Let's look at this with an example:

Assume that decomposition of matrix R looks like:

$$R = U \Sigma V^T$$

We can re-write this decomposition as follows:

$$R = U \sqrt{\Sigma} \sqrt{\Sigma} V^T$$

If we were to take only the top K singular values from this matrix, we could again write this as:

$$R_¿ U \sqrt{\Sigma}_k \sqrt{\Sigma}_k V^T$$

Thus we have now effectively separated our ratings matrix R into two matrices given by: $ U_k = U_{[:k]}\sqrt\Sigma_k $ and $ V_k = \sqrt\Sigma_k V_{[:k]}^{T} $

There are many ways to visualize the importance of U and V matrices but with respect to our context of movie ratings, we can visualize these matrices as follows:

We can imagine each row of U_k to be holding some information how much each user likes a particular feature (feature1, feature2, feature 3...feature k). On the contrary, we can imagine each column of V_k^T to be holding some information about how much each movie relates to the given features (feature 1, feature 2, feature 3 ... feature k).

Lets denote the row of U_k by u_i and the column of V_k^T by m_j. Then the dot-product: $u_i \cdot m_j$ can provide us with information on how much a user *i* likes movie *j*.

What have we achieved by doing this?

Starting with a matrix R containing very few ratings, we have been able to summarize the sparse matrix of ratings into matrices U_k and V_k which each contain feature vectors about the Users and the Movies. Since these feature vectors are summarized from only the most important K features (by our SVD), we can predict any User-Movie rating that is closer to the actual value

than just taking any average rating of a row / column (recall our brute force solution discussed above).

Now this method in practice is still not close to the state-of-the-art but for a naive and simple method we have used, we can still build some powerful visualizations as we will see in part 3.

We have divided the task into 3 parts:

1. Implement `recommender_svd` to return matrices U_k and V_k
2. Implement `predict` to predict top 3 movies a given user would watch
3. *(Ungraded)* Feel free to run the final cell labeled to see some visualizations of the feature vectors you have generated

Hint: Movie IDs are IDs assigned to the movies in the dataset and can be greater than the number of movies. This is why we have given movies_index and users_index as well that map between the movie IDs and the indices in the ratings matrix. Please make sure to use this as well.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from svd_recommender import SVDRecommender
from regression import Regression

################################
### DO NOT CHANGE THIS CELL ###
################################

recommender = SVDRecommender()
recommender.load_movie_data()
regression = Regression()
# Read the data into the respective train and test dataframes
train, test = recommender.load_ratings_datasets()
print("-------------------------------------------")
print("Train Dataset Stats:")
print("Shape of train dataset: {}".format(train.shape))
print("Number of unique users (train):
{}".format(train["userId"].unique().shape[0]))
print("Number of unique users (train):
{}".format(train["movieId"].unique().shape[0]))
print("Sample of Train Dataset:")
print("-----------------------------------------")
print(train.head())
print("-----------------------------------------")
print("Test Dataset Stats:")
print("Shape of test dataset: {}".format(test.shape))
print("Number of unique users (test):
{}".format(test["userId"].unique().shape[0]))
```

```
print("Number of unique users (test):
{}".format(test["movieId"].unique().shape[0]))
print("Sample of Test Dataset:")
print("----------------------------------------")
print(test.head())
print("----------------------------------------")

# We will first convert our dataframe into a matrix of Ratings: R
# R[i][j] will indicate rating for movie:(j) provided by user:(i)
# users_index, movies_index will store the mapping between array
indices and actual userId / movieId
R, users_index, movies_index =
recommender.create_ratings_matrix(train)
print("Shape of Ratings Matrix (R): {}".format(R.shape))

# Replacing `nan` with average rating given for the movie by all users
# Additionally, zero-centering the array to perform SVD
mask = np.isnan(R)
masked_array = np.ma.masked_array(R, mask)
r_means = np.array(np.mean(masked_array, axis=0))
R_filled = masked_array.filled(r_means)
R_filled = R_filled - r_means
```

7.1.1 Implement the `recommender_svd` method to use SVD for Recommendation [5pts] **[P]**

In svd_recommender.py file, complete the following function:

- recommender_svd: Use the above equations to output U_k and V_k. You can utilize the `svd` and `compress` methods from `imgcompression.py` to retrieve your initial U, Σ and V matrices. Then, calculate U_k and V_k based on the decomposition example above.

Local Test for `recommender_svd` Function [No Points]

You may test your implementation of the function in the cell below. See Using the Local Tests for more details.

```
################################
### DO NOT CHANGE THIS CELL ###
################################

from utilities.localtests import TestSVDRecommender

unittest_svd_rec = TestSVDRecommender()
unittest_svd_rec.test_recommender_svd()

################################
### DO NOT CHANGE THIS CELL ###
################################
```

```
# Implement the method `recommender_svd` and run it for the following
values of features
no_of_features = [2, 3, 8, 15, 18, 25, 30]
test_errors = []

for k in no_of_features:
    U_k, V_k = recommender.recommender_svd(R_filled, k)
    pred = []  # to store the predicted ratings
    for _, row in test.iterrows():
        user = row["userId"]
        movie = row["movieId"]
        u_index = users_index[user]
        # If we have a prediction for this movie, use that
        if movie in movies_index:
            m_index = movies_index[movie]
            pred_rating = np.dot(U_k[u_index, :], V_k[:, m_index]) +
r_means[m_index]
        # Else, use an average of the users ratings
        else:
            pred_rating = np.mean(np.dot(U_k[u_index], V_k)) +
r_means[m_index]
        pred.append(pred_rating)
    test_error = regression.rmse(test["rating"], pred)
    test_errors.append(test_error)
    print("RMSE for k = {} --> {}".format(k, test_error))
```

Plot the Test Error over the different values of `k`

```
################################
### DO NOT CHANGE THIS CELL ###
################################
fig = plt.figure()
plt.plot(no_of_features, test_errors, "bo")
plt.plot(no_of_features, test_errors)
plt.xlabel("Value for k")
plt.ylabel("RMSE on Test Dataset")
plt.title("SVD Recommendation Test Error with Different k values")

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA * 0.5,
        fontname=EO_FONT,
```

```
        ha="center",
        va="center",
        rotation=EO_ROT,
    )

plt.show()
```

7.1.2 Implement the `predict` method to find which movie a user is interested in watching next [5pts] **[P]**

Our goal here is to predict movies that a user would be interested in watching next. Since our dataset contains a large list of movies and our model is very naive, filtering among this huge set for top 3 movies can produce results that we may not correlate immediately. Therefore, we'll restrict this prediction to only movies among a subset as given by movies_pool.

Let us consider a user (ID: 660) who has already watched and rated well (>3) on the following movies:

- Iron Man (2008)
- Thor: The Dark World (2013)
- Avengers, The (2012)

The following cell tries to predict which among the movies given by the list below, the user would be most interested in watching next: `movies_pool`:

- Ant-Man (2015)
- Iron Man 2 (2010)
- Avengers: Age of Ultron (2015)
- Thor (2011)
- Captain America: The First Avenger (2011)
- Man of Steel (2013)
- Star Wars: Episode IV - A New Hope (1977)
- Ladybird Ladybird (1994)
- Man of the House (1995)
- Jungle Book, The (1994)

In svd_recommender.py file, complete the following function:

- predict: Predict the next 3 movies that the user would be most interested in watching among the ones above.

HINT: You can use the method `get_movie_id_by_name` to convert movie names into movie IDs and vice-versa.

NOTE: The user may have already watched and rated some of the movies in `movies_pool`. Remember to filter these out before returning the output. The original Ratings Matrix, R might come in handy here along with `np.isnan`

Local Test for `predict` Functions [No Points]

You may test your implementation of the function in the cell below. See Using the Local Tests for more details.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

unittest_svd_rec.test_predict()
```

7.2 Visualize Movie Vectors [No Points]

Our model is still a very naive model, but it can still be used for some powerful analysis such as clustering similar movies together based on user's ratings.

We have said that our matrix V_k that we have generated above contains information about movies. That is, each column in V_k contains (feature 1, feature 2, ... feature k) for each movie. We can also say this in other terms that V_k gives us a feature vector (of length k) for each movie that we can visualize in a k-dimensional space. For example, using this feature vector, we can find out which movies are similar or vary.

While we would love to visualize a k-dimensional space, the constraints of our 2D screen wouldn't really allow us to do so. Instead let us set $K = 2$ and try to plot the feature vectors for just a couple of these movies.

As a fun activity run the following cell to visualize how our model separates the two sets of movies given below.

NOTE: There are 2 possible visualizations. Your plot could be the one that's given on the expected PDF or the one where the y-coordinates are inverted.

```
##############################
### DO NOT CHANGE THIS CELL ###
##############################

marvel_movies = [
    "Thor: The Dark World (2013)",
    "Avengers: Age of Ultron (2015)",
    "Ant-Man (2015)",
    "Iron Man 2 (2010)",
    "Avengers, The (2012)",
    "Thor (2011)",
    "Captain America: The First Avenger (2011)",
]
marvel_labels = ["Blue"] * len(marvel_movies)
star_wars_movies = [
    "Star Wars: Episode IV - A New Hope (1977)",
    "Star Wars: Episode V - The Empire Strikes Back (1980)",
```

```
    "Star Wars: Episode VI - Return of the Jedi (1983)",
    "Star Wars: Episode I - The Phantom Menace (1999)",
    "Star Wars: Episode II - Attack of the Clones (2002)",
    "Star Wars: Episode III - Revenge of the Sith (2005)",
]
star_wars_labels = ["Green"] * len(star_wars_movies)


movie_titles = star_wars_movies + marvel_movies
genre_labels = star_wars_labels + marvel_labels

movie_indices = [
    movies_index[recommender.get_movie_id_by_name(str(x))] for x in
movie_titles
]

_, V_k = recommender.recommender_svd(R_filled, k=2)
x, y = V_k[0, movie_indices], V_k[1, movie_indices]
fig = plt.figure()
plt.scatter(x, y, c=genre_labels)
for i, movie_name in enumerate(movie_titles):
    plt.annotate(movie_name, (x[i], y[i]))

if not STUDENT_VERSION:
    fig.text(
        0.5,
        0.5,
        EO_TEXT,
        transform=fig.transFigure,
        fontsize=EO_SIZE / 2,
        color=EO_COLOR,
        alpha=EO_ALPHA * 0.5,
        fontname=EO_FONT,
        ha="center",
        va="center",
        rotation=EO_ROT,
    )
```