Data Science Lab: Lab 1

Submit:

A pdf of your notebook with solutions. A link to your colab notebook or also upload your .ipynb if not working on colab.

Goals of this Lab:

1. Review important results from probability, such as the CLT.
2. Connecting that review with basic Python commands.
3. Practice with Pandas, Numpy and Data Exploration.

Tony, Eric, Daniel

In [1]:

```
# Some useful libraries
# !pip install numpy
# !pip install pandas
# !pip install seaborn
# !pip install matplotlib
# !pip install scikit-learn

import numpy as np
from numpy.random import default_rng

#Pandas for data structure and analysis tools
import pandas as pd

#seaborn and matplotlib for plotting
import seaborn as sns
import matplotlib.pyplot as plt

#for nice vector graphics
%matplotlib inline

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png', 'pdf')

np.random.seed(42) # Fixed seed for reproducibility, do not change this value
rng = default_rng()
```

```
C:\Users\tonys\AppData\Local\Temp\ipykernel_26260\3657667379.py:22: DeprecationWarni
ng: `set_matplotlib_formats` is deprecated since IPython 7.23, directly use `matplot
lib_inline.backend_inline.set_matplotlib_formats()`
  set_matplotlib_formats('png', 'pdf')
```

In [ ]:

Problem 1

Part 1. Generate 1,000 samples of 2 dimensional data from the Gaussian distribution $\begin{pmatrix} X_i \\ Y_i \end{pmatrix} \sim N\left(\begin{pmatrix} -5 \\ 5 \end{pmatrix}, \begin{pmatrix} 2 & 0.8 \\ 0.8 & 3 \end{pmatrix}\right)$.

Part 2. Plot these points.

Part 3. Find the Eigenvectors and Eigenvalues of the covariance matrix using np.linalg.eig, or np.linalg.eigh, or something else of your choice.

Part 4. Now take the 1,000 points you generated in the first part, and use them to estimate the mean and covariance matrix for this multi-dimensional data using elementary numpy commands, i.e., addition, multiplication, division (do not use a command that takes data and returns the mean or standard deviation).

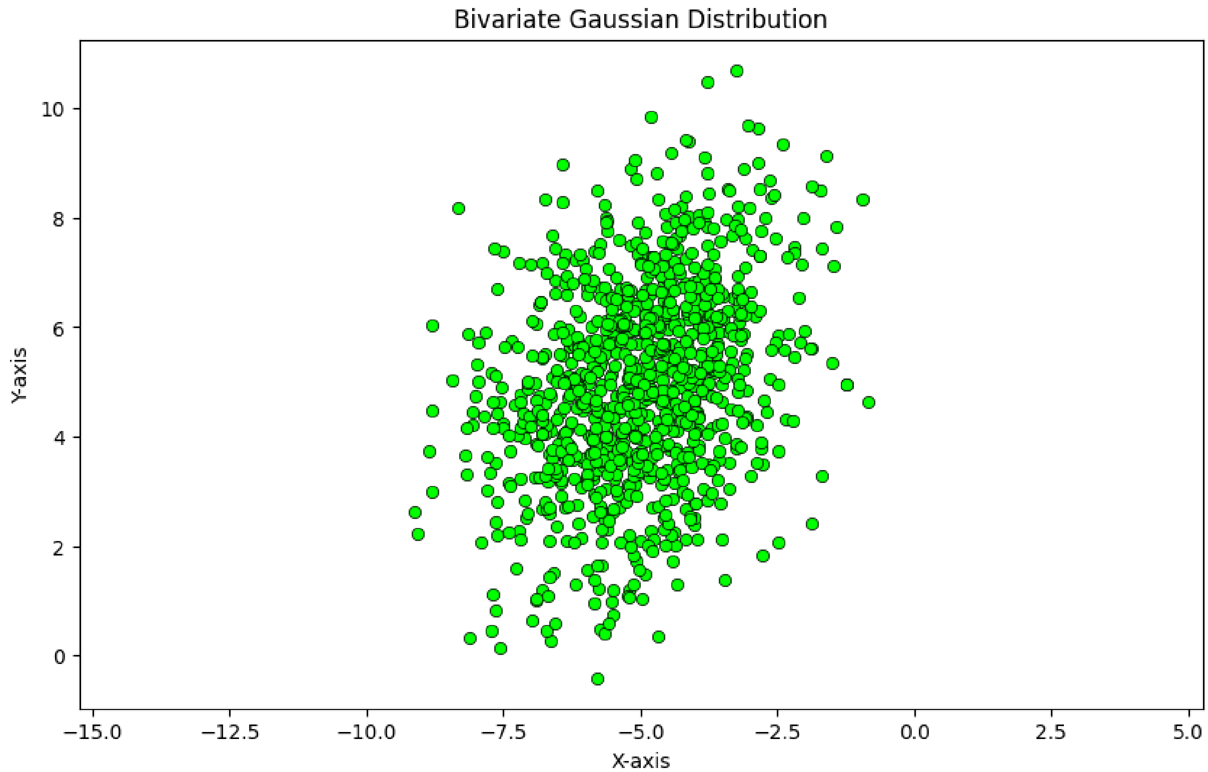
Remark: If you did this correctly: You should have made a number of observations. (i) The points you plotted should look like an elongated ellipse. (ii) The axis of elongation (the major axis of the ellipse) should be aligned with the eigenvector you computed that has the largest eigenvalue. The minor axis, should be aligned with the other eigenvector you computed. (iii) In the last part, you computed what is called the *empirical covariance* matrix. This should be quite close to the covariance matrix you used to generate the data. If we used more and more points (10,000, 100,000, etc.), then our empirical estimate would look more and more like what we used to generate the data.

Part 1,2)

In [3]:
```
from scipy.stats import multivariate_normal

cov=np.array([[2,0.8],[0.8,3]])
mean=(-5,5)
random_seed=44



plt.rcParams['figure.figsize']=10,6

#Easy Plotting without coloring(not showing the depth and height) + Generate 1000 s
distr = multivariate_normal(cov = cov, mean = mean,seed = random_seed)
data = distr.rvs(size = 1000)


# Plotting the generated samples
plt.plot(data[:,0],data[:,1], 'o', c='lime', markeredgewidth = 0.5, markeredgecolor
plt.title('Bivariate Gaussian Distribution')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.axis('equal') #using plt.axis('equal') ensures that the elliptical shape of the
# It accurately reflects the true relationship and spread between X and Y, make sur
```

```
# because X and Y are independent random variable and the axis distance should be a

print(data)
```

```
[[-4.23876104  2.98558259]
 [-5.63037218  7.9931902 ]
 [-8.19222642  3.6544736 ]
 ...
 [-2.40765191  9.35307396]
 [-4.9795823   3.67919912]
 [-6.39156512  4.96613628]]
```

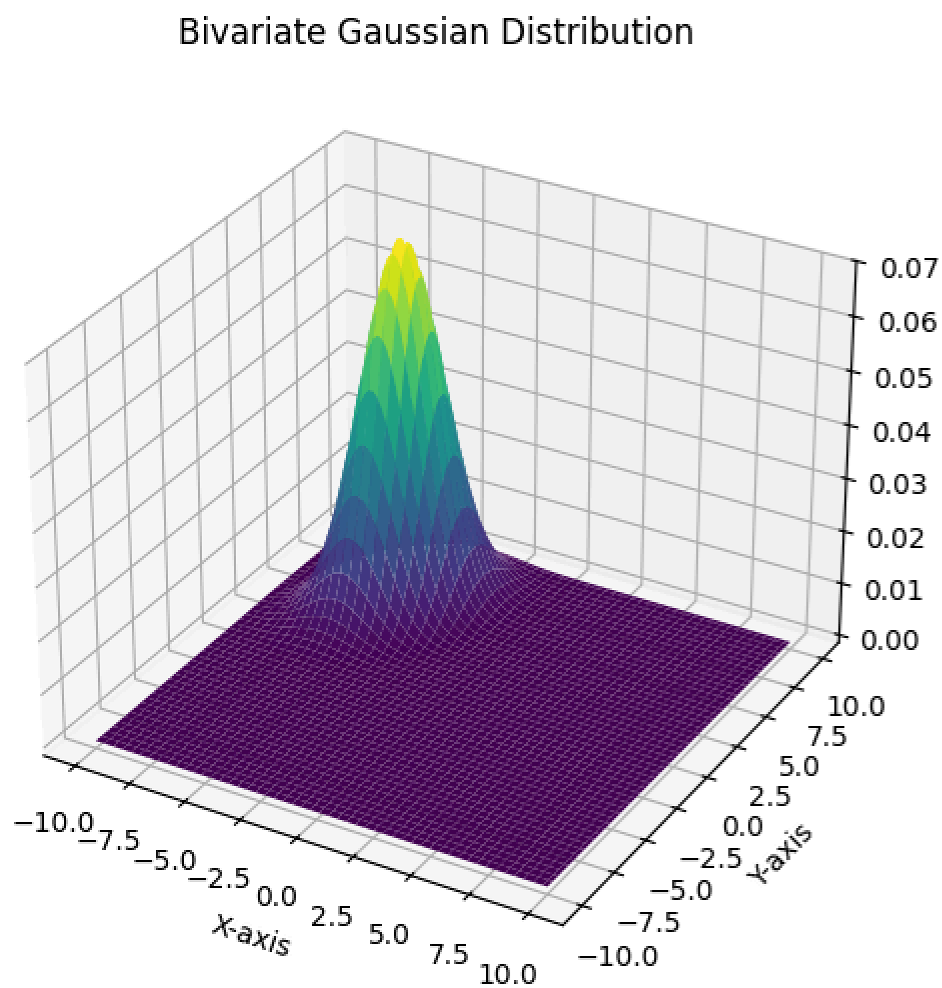


In [4]:
```
# Generating a meshgrid - 3D version

x = np.linspace(-10, 10, num=100)
y = np.linspace(-10, 10, num=100)
X, Y = np.meshgrid(x,y)

# pdf = np.zeros(X.shape)
# for i in range(X.shape[0]):
#     for j in range(X.shape[1]):
#         pdf[i,j] = distr.pdf([X[i,j], Y[i,j]])

pos=np.dstack((X,Y))
print(pos.shape)

pdf=distr.pdf(pos) #get the value of pdf from each position


# Plotting the density function values #3D version

ax = plt.subplot( projection = '3d')
ax.plot_surface(X, Y, pdf, cmap = 'viridis')
```

```
plt.title('Bivariate Gaussian Distribution')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
```

```
(100, 100, 2)
```

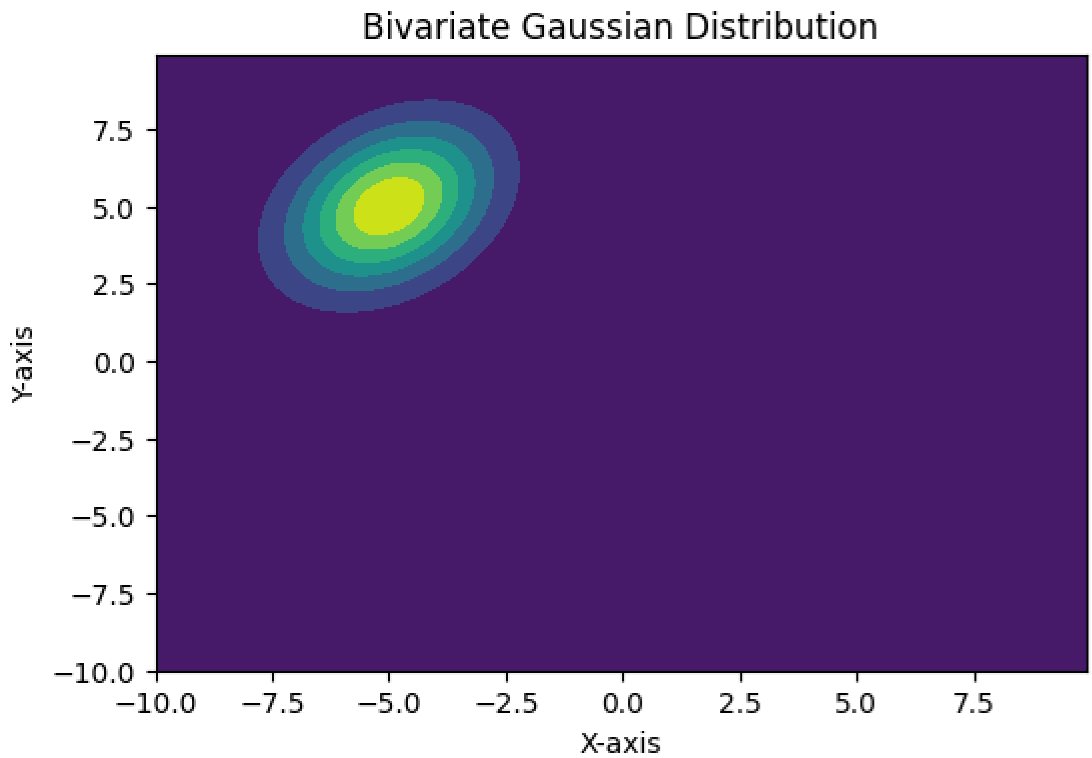
Out[4]: `Text(0.5, 0.5, 'Y-axis')`



In [48]:

```
#2D version
x, y = np.mgrid[-10:10:0.1, -10:10:0.1]

pos=np.dstack((x,y))

plt.figure(figsize=(6,4))

plt.contourf(x, y, distr.pdf(pos))
plt.title('Bivariate Gaussian Distribution')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

print(X_mean,Y_mean)
```

```
-5.0251430849779135 5.005249114743093
```

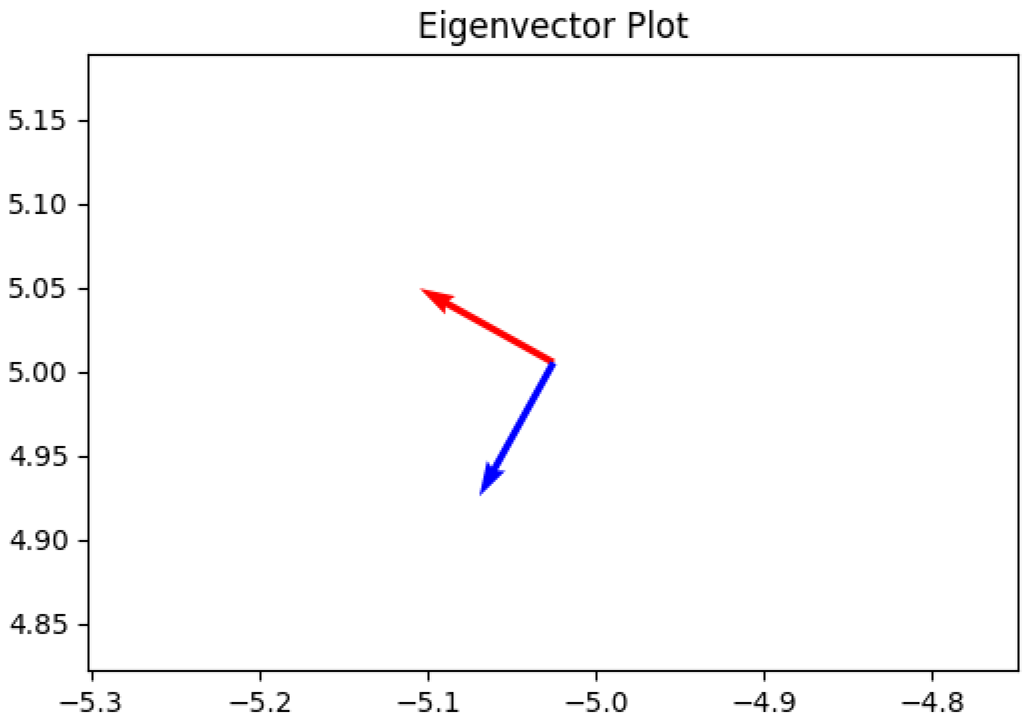


Part 3)

In [94]:
```
eigenvalues,eigenvectors=np.linalg.eig(cov)

print('eigenvalues:\n\n',eigenvalues,'\n\n','eigenvectors:\n\n',eigenvectors,sep=''

eigen_0=eigenvectors[:,0]
eigen_1=eigenvectors[:,1]


# origin = np.array([[X_mean, Y_mean],[X_mean, Y_mean]]) # origin point

plt.figure(figsize=(6,4))

# print(*origin)

# plt.quiver(*origin.T, eigen_0,eigen_1, color=['r','b'], scale=11,scale_units='xy'



# Plot first eigenvector
plt.quiver(X_mean, Y_mean, eigen_0[0], eigen_0[1], color='r', scale=11,scale_units=
# Plot second eigenvector
plt.quiver(X_mean, Y_mean, eigen_1[0], eigen_1[1], color='b', scale=11, scale_units

# Label the
plt.axis('equal')
plt.title("Eigenvector Plot")
```

```
eigenvalues:

[1.55660189 3.44339811]

eigenvectors:

[[-0.87464248 -0.48476853]
 [ 0.48476853 -0.87464248]]
```

Out[94]: Text(0.5, 1.0, 'Eigenvector Plot')



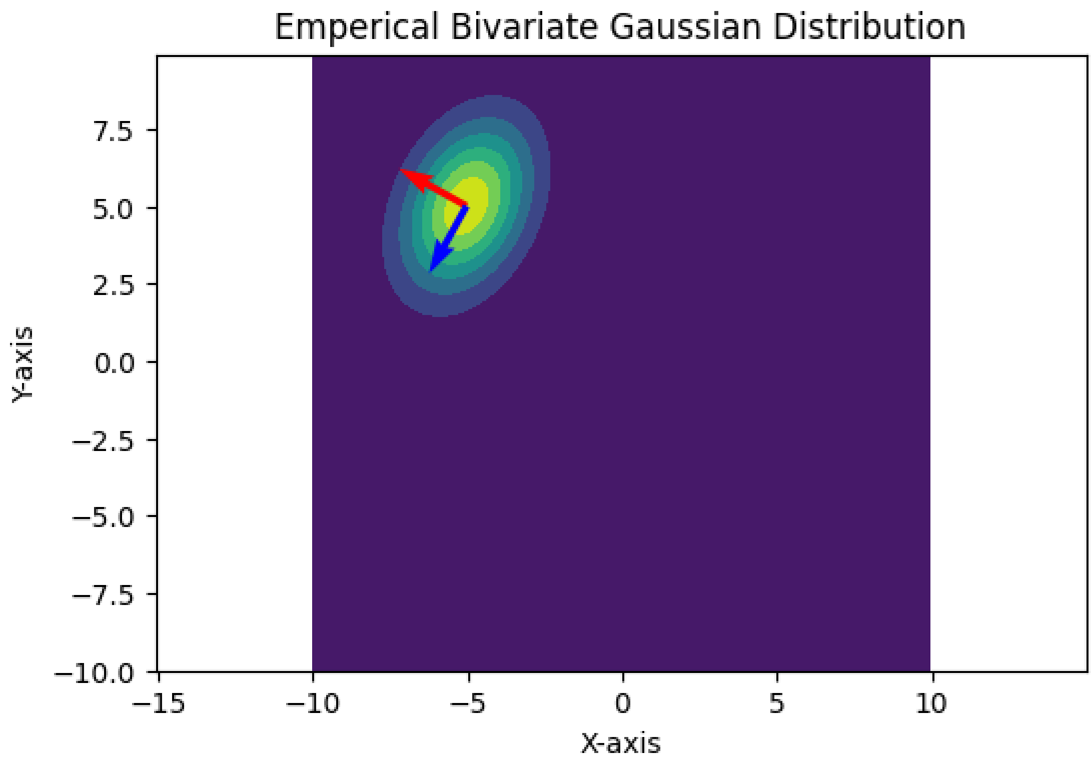
Part 4)

In [92]:
```
print("Data shape:\n\n", data,data.shape,type(data))


#Calculate mean,var to get emperical covariance matrix
X_mean=np.mean(data[:,0])
Y_mean=np.mean(data[:,1])

X_var=np.var(data[:,0])
Y_var=np.var(data[:,1])


emp_cov=np.sum((data[:,0]-X_mean)*(data[:,1]-Y_mean))/1000

emp_covmat=[[X_var,emp_cov],[emp_cov,Y_var]]

print("\n\nEmperical covariance matrix: \n\n",np.array(emp_covmat))

emp_distr=multivariate_normal(mean=[X_mean,Y_mean],cov=emp_covmat)
```

```
x, y = np.mgrid[-10:10:0.1, -10:10:0.1]

pos=np.dstack((x,y))

plt.figure(figsize=(6,4))

plt.contourf(x, y, emp_distr.pdf(pos))
plt.title('Emperical Bivariate Gaussian Distribution')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

# origin = np.array([[X_mean, Y_mean],[X_mean, Y_mean]]) # origin point
# plt.quiver(*origin.T, eigen_0,eigen_1, color=['r','b'], scale=0.5,scale_units='xy
# Plot first eigenvector
plt.quiver(X_mean, Y_mean, eigen_0[0], eigen_0[1], color='r', scale=0.4, scale_unit
# Plot second eigenvector
plt.quiver(X_mean, Y_mean, eigen_1[0], eigen_1[1], color='b', scale=0.4, scale_unit
plt.axis('equal') #to make vector algin with axis
```

```
Data shape:

 [[-4.23876104  2.98558259]
 [-5.63037218  7.9931902 ]
 [-8.19222642  3.6544736 ]
 ...
 [-2.40765191  9.35307396]
 [-4.9795823   3.67919912]
 [-6.39156512  4.96613628]] (1000, 2) <class 'numpy.ndarray'>


Emperical covariance matrix:

 [[1.88669212 0.80970696]
 [0.80970696 3.26673484]]
```

Out[92]:
```
(np.float64(-10.0),
 np.float64(9.900000000000002),
 np.float64(-10.0),
 np.float64(9.900000000000002))
```



Problem 2: Central Limit Theorem

Back in EE351K you learned the Law of Large Numbers, and the Central Limit Theorem, among many other things. The Law of Large Numbers says that if X_i are independent and identically distributed (iid) random variables, then $(1/N)\sum X_i$ converges to $\mathbb{E}[X]$. That's the law of large numbers.

You also learned the Central Limit Theorem. This says that if X_i are zero mean, have variance 1, and are iid, then $(1/\sqrt{N})\sum X_i$ converges to a random variable. Which random variable? A standard (zero mean, unit variance) Gaussian.

We're going to check the central limit theorem empirically, as an excuse to do more practice with Python and numpy and basic plotting.

Let X_i be an iid Bernoulli random variable with value {-1,1}. Look at the random variable $Z_n = \frac{1}{\sqrt{n}}\sum X_i$. By taking 1000 samples from Z_n, plot its histogram. {\bf Note:} To generate 1,000 samples from Z_n, you need to generate $1,000 \times n$ samples of X_i, since each Z needs $1,000$ X_i's. Now check that for small n (set $n = 5$) Z_n does not look that much like a Gaussian, but when n is bigger (set $n = 50$) it looks much more like a Gaussian. Check also for much bigger n: $n = 250$, to see that at this point, one can really see the bell curve.

In [162…

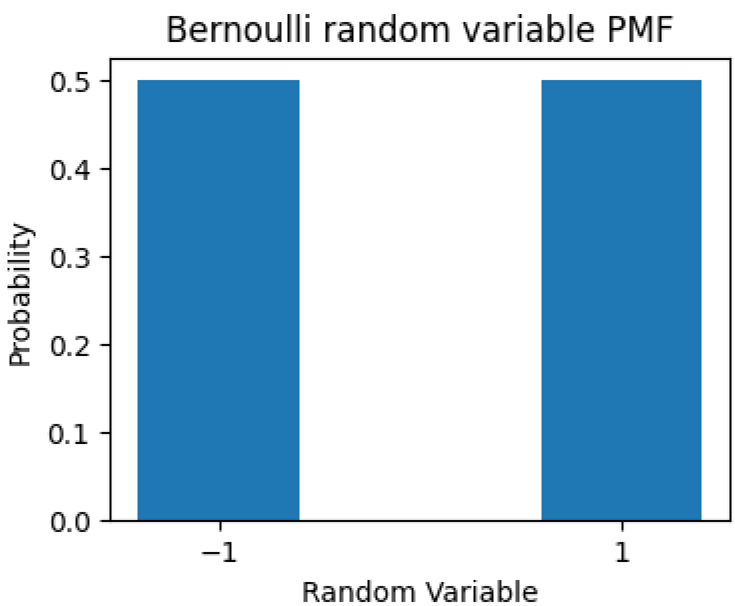
```
# plot Bernoulli random varaiable

# prob=np.random.rand()
prob=0.5
```

```python
#Let  Xi be an iid Bernoulli random variable with value {-1,1}.
print("probability for Bernoulli random variable: ",prob )

plt.figure(figsize=(4,3))
plt.bar([-1,1],[1-prob,prob])
plt.xticks([-1,1])
plt.ylabel("Probability")
plt.xlabel("Random Variable")
plt.title("Bernoulli random variable PMF")
```

```
probability for Bernoulli random variable:  0.5
```

Out[162…

```
Text(0.5, 1.0, 'Bernoulli random variable PMF')
```

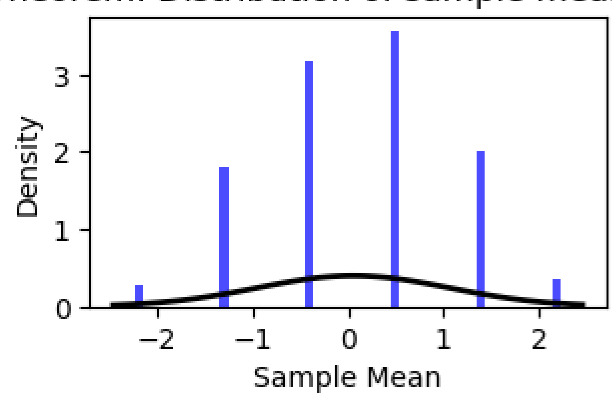


In [165…

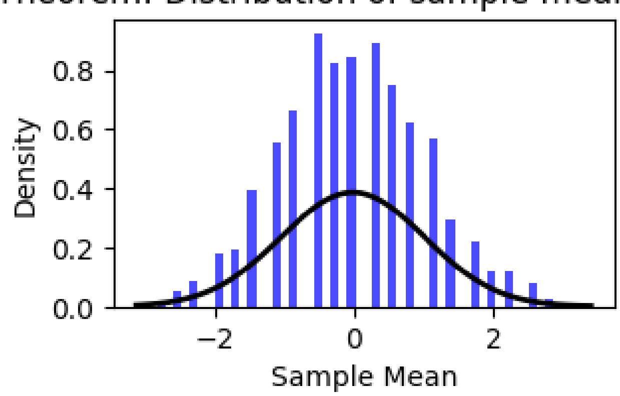
```python
# Calculate the mean of each sample by # of samples
# Parameters

sample_sizes=[5,50,100,250]# Number of samples per iteration
n_iterations = 1000  # Number of iterations(samples)


for i,sample_size in enumerate(sample_sizes):
    i+=1
    data=[]

    #Sampling by # of samples
    for j in range(n_iterations):
        sample = np.random.choice([-1,1], size=sample_size, p=[1-prob,prob])
        data.append(sample)


    #Calculate the means per sample[]
#     means = np.mean(data, axis=1)

    # Compute Zn = (1/sqrt(n)) * sum(Xi)
    means = np.sum(data, axis=1) / np.sqrt(sample_size)
```

```
#     print(means[0:10])
    plt.figure(figsize=(6,4))
    # Plot histogram of the sample means
    plt.subplot(2,2,i)
    plt.hist(means, bins=50, density=True, alpha=0.7, color='b')
    #denstiy=True: This normalizes the histogram so that the total area under all t

    # Plot the expected normal distribution for comparison
    mu, sigma = np.mean(means), np.std(means)
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    normf = np.exp(- (x - mu)**2 / (2 * sigma**2)) / (sigma * np.sqrt(2 * np.pi)) #
    plt.plot(x, normf, 'k', linewidth=2)

    plt.title(f'Central Limit Theorem: Distribution of sample means (sample num: {s
    plt.xlabel('Sample Mean')
    plt.ylabel('Density')
    plt.tight_layout()
```

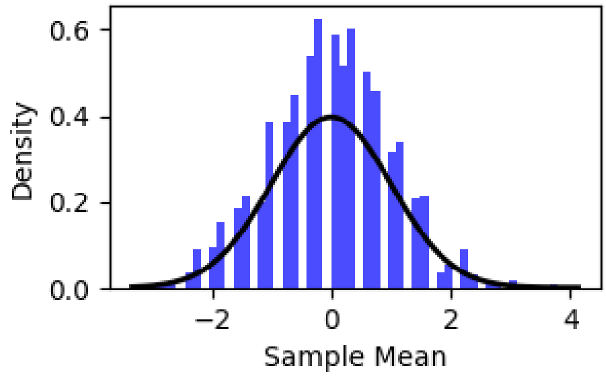
Central Limit Theorem: Distribution of sample means (sample num: 5)



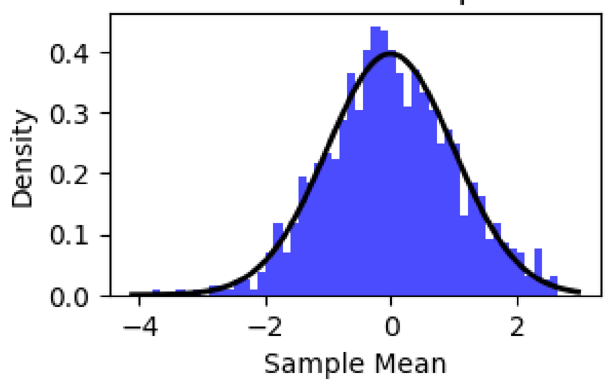
Central Limit Theorem: Distribution of sample means (sample num: 50)



Central Limit Theorem: Distribution of sample means (sample num: 100)



Central Limit Theorem: Distribution of sample means (sample num: 250)



Problem 3

Download from Canvas/Files the dataset PatientData.csv. Each row is a patient and the last column is the condition that the patient has. Do data exploration using Pandas and other visualization tools to understand what you can about the data set. For example:

Part 1. How many patients and how many features are there?

Part 2. What is the meaning of the first 4 features? See if you can understand what they mean.

Part 3. Are there missing values? Replace them with the average of the corresponding feature column and plot the feature histograms with

Part 4. How could you test which features strongly influence the patient condition and which do not? List what you think are the three most important features.

In [61]:
```
dataset=pd.read_csv('PatientData.csv',header=None)
dataset=pd.DataFrame(dataset)
dataset
```

Out[61]:

	0	1	2	3	4	5	6	7	8	9	...	270	271	272	273	274	275
0	75	0	190	80	91	193	371	174	121	-16	...	0.0	9.0	-0.9	0.0	0.0	0.9
1	56	1	165	64	81	174	401	149	39	25	...	0.0	8.5	0.0	0.0	0.0	0.2
2	54	0	172	95	138	163	386	185	102	96	...	0.0	9.5	-2.4	0.0	0.0	0.3
3	55	0	175	94	100	202	380	179	143	28	...	0.0	12.2	-2.2	0.0	0.0	0.4
4	75	0	190	80	88	181	360	177	103	-16	...	0.0	13.1	-3.6	0.0	0.0	-0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
447	53	1	160	70	80	199	382	154	117	-37	...	0.0	4.3	-5.0	0.0	0.0	0.7
448	37	0	190	85	100	137	361	201	73	86	...	0.0	15.6	-1.6	0.0	0.0	0.4
449	36	0	166	68	108	176	365	194	116	-85	...	0.0	16.3	-28.6	0.0	0.0	1.5
450	32	1	155	55	93	106	386	218	63	54	...	-0.4	12.0	-0.7	0.0	0.0	0.5
451	78	1	160	70	79	127	364	138	78	28	...	0.0	10.4	-1.8	0.0	0.0	0.5

452 rows × 280 columns

Part 1)

In [512…

```
print("There are 452 patients and 279 features, last columns=label(condition)")
```

```
There are 452 patients and 279 features, last columns=label(condition)
```

In [63]:

```
print(np.sort(dataset[279].unique()))
print(dataset[279].nunique())
dataset.info()
```

```
[ 1  2  3  4  5  6  7  8  9 10 14 15 16]
13
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Columns: 280 entries, 0 to 279
dtypes: float64(120), int64(155), object(5)
memory usage: 988.9+ KB
```

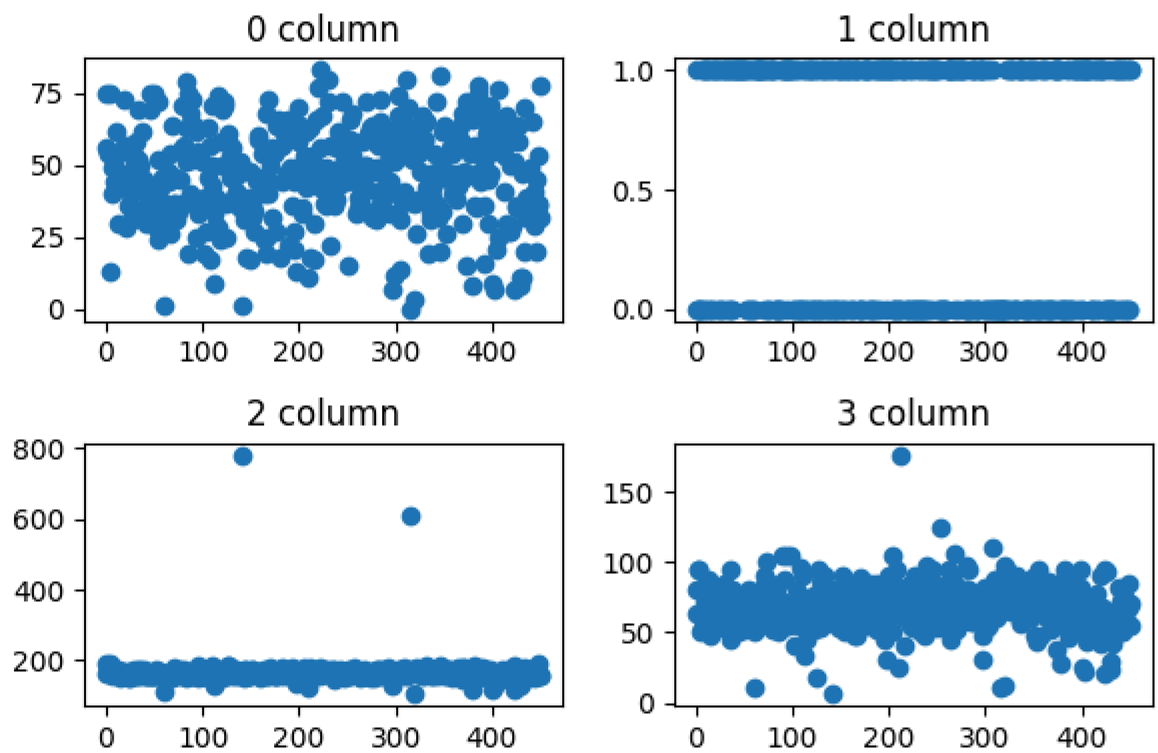
Part 2)

In [69]:

```
print(dataset[[0,1,2,3]].describe())

plt.figure(figsize=(6,4))
plt.subplot(2,2,1)
plt.scatter(np.arange(452),dataset[0])
plt.title('0 column')
```

```
plt.subplot(2,2,2)
plt.scatter(np.arange(452),dataset[1])
plt.title('1 column')

plt.subplot(2,2,3)
plt.scatter(np.arange(452),dataset[2])
plt.title('2 column')

plt.subplot(2,2,4)
plt.scatter(np.arange(452),dataset[3])
plt.title('3 column')
plt.tight_layout()
```

```
                0           1           2           3
count  452.000000  452.000000  452.000000  452.000000
mean    46.471239    0.550885  166.188053   68.170354
std     16.466631    0.497955   37.170340   16.590803
min      0.000000    0.000000  105.000000    6.000000
25%     36.000000    0.000000  160.000000   59.000000
50%     47.000000    1.000000  164.000000   68.000000
75%     58.000000    1.000000  170.000000   79.000000
max     83.000000    1.000000  780.000000  176.000000
```



In [70]:
```
print("Expected data for first four features --> \n")
print("\nfirst column: age" )
print("\nsecond column: sex(0 / 1)" )
print("\nthird column: height(cm) with two outlier" )
print("\nfourth column: weight(kg) with ourtlier" )
```

```
Expected data for first four features -->

first column: age

second column: sex(0 / 1)

third column: height(cm) with two outlier

fourth column: weight(kg) with ourtlier
```

In [71]:
```
dataset.tail()
```

Out[71]:

	0	1	2	3	4	5	6	7	8	9	...	270	271	272	273	274	275
447	53	1	160	70	80	199	382	154	117	-37	...	0.0	4.3	-5.0	0.0	0.0	0.7
448	37	0	190	85	100	137	361	201	73	86	...	0.0	15.6	-1.6	0.0	0.0	0.4
449	36	0	166	68	108	176	365	194	116	-85	...	0.0	16.3	-28.6	0.0	0.0	1.5
450	32	1	155	55	93	106	386	218	63	54	...	-0.4	12.0	-0.7	0.0	0.0	0.5
451	78	1	160	70	79	127	364	138	78	28	...	0.0	10.4	-1.8	0.0	0.0	0.5

5 rows × 280 columns

In [72]:
```
dataset.describe()
```

Out[72]:

	0	1	2	3	4	5	6
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000
mean	46.471239	0.550885	166.188053	68.170354	88.920354	155.152655	367.207965
std	16.466631	0.497955	37.170340	16.590803	15.364394	44.842283	33.385421
min	0.000000	0.000000	105.000000	6.000000	55.000000	0.000000	232.000000
25%	36.000000	0.000000	160.000000	59.000000	80.000000	142.000000	350.000000
50%	47.000000	1.000000	164.000000	68.000000	86.000000	157.000000	367.000000
75%	58.000000	1.000000	170.000000	79.000000	94.000000	175.000000	384.000000
max	83.000000	1.000000	780.000000	176.000000	188.000000	524.000000	509.000000

8 rows × 275 columns

Part 3)

In [74]:
```
#any missing values?
missing_cols=[]
for col in dataset.columns:
    if len(dataset[col])!=452:
        missing.append(col)

missing_values = dataset.isnull().sum()
sum(missing_values)


for col in dataset.columns:
    if dataset[col].dtype=='object':
        missing_cols.append(col)

print("missing columns name: ",missing_cols)
```

```
missing columns name:  [10, 11, 12, 13, 14]
```

In [75]:
```
dataset[missing_cols][dataset[missing_cols]!='?']
```

Out[75]:

	10	11	12	13	14
0	13	64	-2	NaN	63
1	37	-17	31	NaN	53
2	34	70	66	23	75
3	11	-5	20	NaN	71
4	13	61	3	NaN	NaN
...	...	...	...	...	...
447	4	40	-27	NaN	63
448	66	52	79	NaN	73
449	-19	-61	-70	84	84
450	29	-22	43	103	80
451	79	52	47	NaN	75

452 rows × 5 columns

In [76]:
```
for c in missing_cols:

    count=np.array([dataset[c]=='?']).sum()
    print(f'For column "{c}" missing count(?) = {count}')
```

```
For column "10" missing count(?) = 8
For column "11" missing count(?) = 22
For column "12" missing count(?) = 1
For column "13" missing count(?) = 376
For column "14" missing count(?) = 1
```

In [77]:

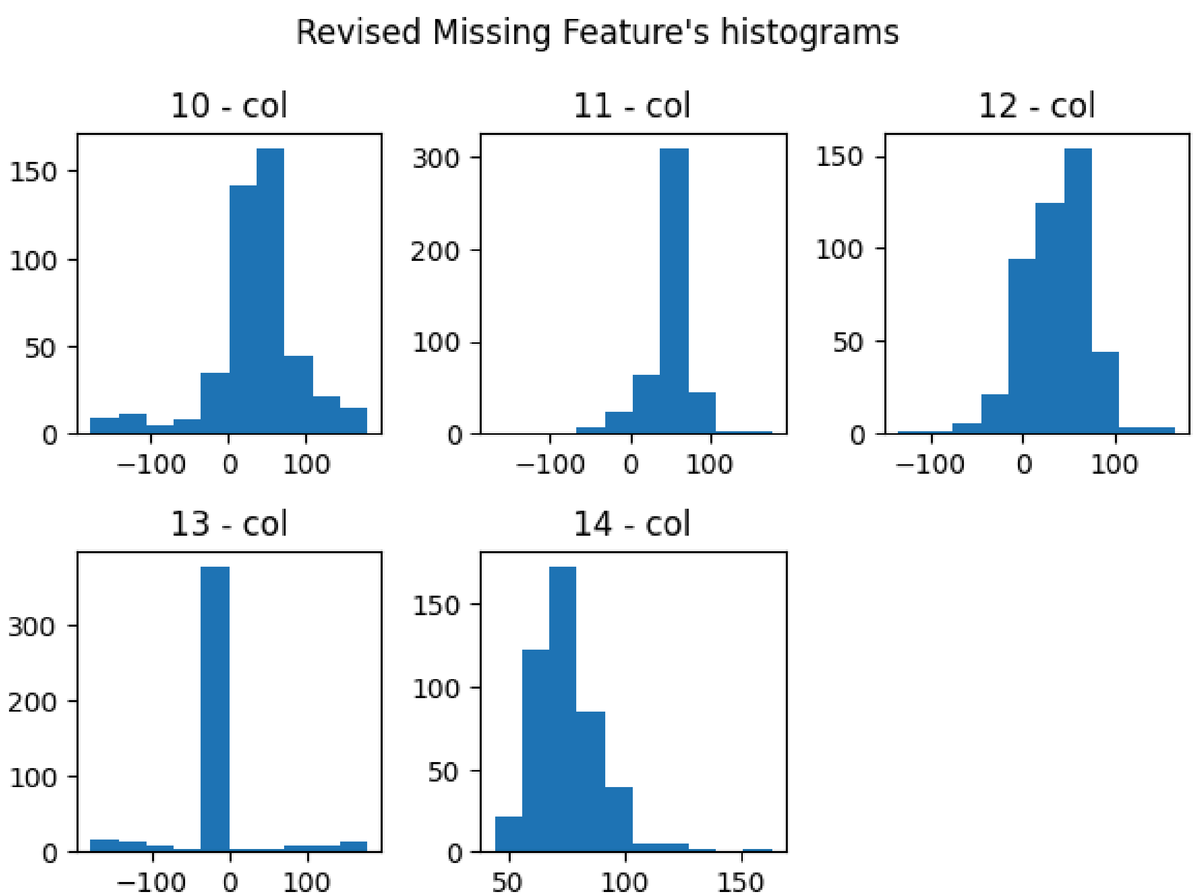
```python
#substitute NaN into 'mean of each columns vector'

for col in missing_cols:
    substitute_mean=dataset[col][dataset[col]!='?'].dropna(inplace=False).astype(fl

    dataset[col] = dataset[col].replace('?', np.nan).fillna(substitute_mean)

print(dataset[missing_cols])

for c in missing_cols:

    count=np.array([dataset[c]=='?']).sum()
    print(f'After Substitution!! \n\n For column "{c}" missing count(?) = {count}')
```

```
      10   11   12         13         14
0     13   64   -2 -13.592105         63
1     37  -17   31 -13.592105         53
2     34   70   66         23         75
3     11   -5   20 -13.592105         71
4     13   61    3 -13.592105  74.463415
..   ...  ...  ...        ...        ...
447    4   40  -27 -13.592105         63
448   66   52   79 -13.592105         73
449  -19  -61  -70         84         84
450   29  -22   43        103         80
451   79   52   47 -13.592105         75

[452 rows x 5 columns]
After Substitution!!

 For column "10" missing count(?) = 0
After Substitution!!

 For column "11" missing count(?) = 0
After Substitution!!

 For column "12" missing count(?) = 0
After Substitution!!

 For column "13" missing count(?) = 0
After Substitution!!

 For column "14" missing count(?) = 0
```

In [534…

```python
#plot the missing columns
plt.suptitle("Revised Missing Feature's histograms")

plt.subplot(2,3,1)
plt.hist(list(dataset[10].astype(float)))
```

```
plt.title('10 - col')

plt.subplot(2,3,2)
plt.hist(list(dataset[11].astype(float)))
plt.title('11 - col')

plt.subplot(2,3,3)
plt.hist(list(dataset[12].astype(float)))
plt.title('12 - col')

plt.subplot(2,3,4)
plt.hist(list(dataset[13].astype(float)))
plt.title('13 - col')

plt.subplot(2,3,5)
plt.hist(list(dataset[14].astype(float)))
plt.title('14 - col')

plt.tight_layout()
```



In [535…
```
# dataset[13].astype(float)
```

In [536…
```
# len(dataset.iloc[0])
```

Part 4)

In [548…

```python
# How to find the connection between features and patient condition? = covariance!

cov_list=[]
for col in dataset.columns[:-1]: #before last column = condition
#     print(col)
    relate = np.cov(dataset[col].astype(float),dataset[279])[0][1]
#     print(relate)
    cov_list.append(relate)
print(len(cov_list))
print(max(cov_list))

sorted_idx=np.argsort(cov_list) #to find first 5 maximum covariances
print(sorted_idx)

print("\n\nname of columns(sorted by strongly influenced): ",[92,4,90,56])
print("value of covariances: ",cov_list[92],cov_list[4],cov_list[90],cov_list[56])
```

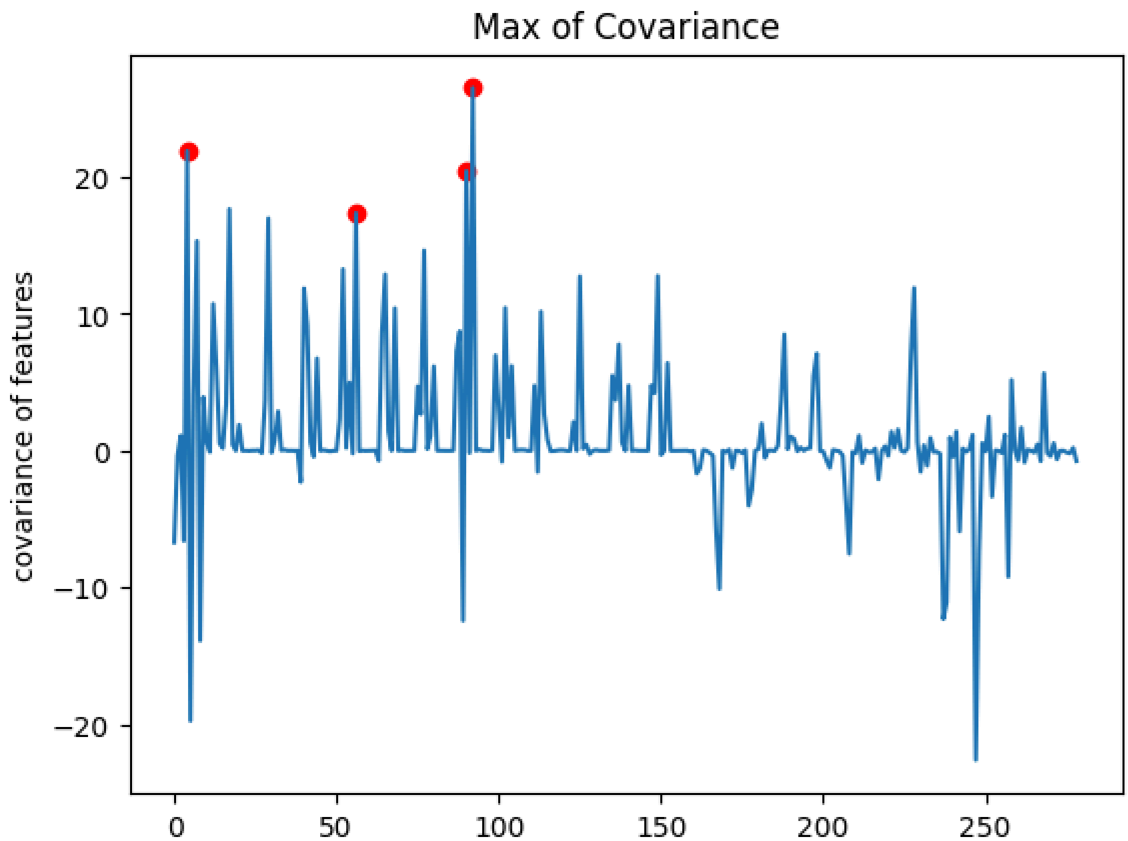
```
279
26.525910955006587
[247   5   8  89 237 238 168 257 208 248   0   3 167 242 177 207 252 178
  39 217 161 112 230 162 172 202 232 212 262 101 267 278  63 201 260 272
 182  43 240 270   1 220 206 166 150 128 236  55 210 276  27 175 269  91
 255 165  30 265 275 214 225 215 209  11 170 234 244 235 200 250 159 205
  48 218 192 116  57 174 117 185 199  23 160 121 109 122  97  37  59 146
 254 179 273  71 153  45  85 144  47  84 133 169  49  35  21 155 129 173
  72  50  60  61 105  38 110 274 264  19 131 145 143 139 141 157 132 151
 156  69 204  83 164  67 154  82  73  74 194  93  22  36  95  96 158  58
  81  25  24 124  86 118 184 183 224  62 253  33  98 213 134 176 142  26
 120  46 106  70 163 119 108 203 130  34 259 107 263 171  94 245 189  78
 180 126 216 195 243 226 196  15  53 277 222 193 219 186 231 127  18 266
  14 271 249  42 138 229  10  31 115 191 233 239 103 190   2 211  79 246
 256 221 241  66 223 261  20 181 123  51 251  76 114  32 100  16  28 187
 136   9   6 148  75 111 140 147  54 258 197 135 268  13  80 104 152  44
  99 198 227  87 137 188  64  88  41 113  68 102  12  40 228 125 149  65
  52  77   7  29  56  17  90   4  92]


name of columns(sorted by strongly influenced):  [92, 4, 90, 56]
value of covariances:  26.525910955006587 21.930596707415177 20.451268567392013 17.3
80099287718544
```

In [88]:

```python
plt.plot(cov_list)
plt.scatter([4,56,90,92],[cov_list[4],cov_list[56],cov_list[90],cov_list[92]],color
plt.title("Max of Covariance")
plt.ylabel("covariance of features")
```

Out[88]:

```
Text(0, 0.5, 'covariance of features')
```



In [ ]:

Problem 4

The goal of this exercise is for you to get more experience with Pandas, and to get a chance to explore a cool data set. Download the fileNames.zip from Canvas. This contains the frequency of all names that appeared more than 5 times on a social security application from 1880 through 2015.

Part 1. Write a program that on input k and XXXX, returns the top k names from year XXXX. Print out the top 100 names from the year 2000

Part 2. Write a program that on input Name returns the frequency for men and women of the name Name. Plot the frequency of the name "Alex" from the year 1880 to 2015

Part 3. It could be that names are more diverse now than they were in 1880, so that a name may be relatively the most popular, though its frequency may have been decreasing over the years. Modify the above to return the relative frequency. Plot the relative frequency of the name "Alex" from the year 1880 to 2015

Part 4. Find all the names that used to be more popular for one gender, but then became more popular for another gender and print out the first 100 names (alphabetized).

•(Optional) Find something cool about this data set.

In [123…

```
data=pd.read_csv('Names/yob1990.txt',header=None)
print(data.iloc[:5])
data[[0,2]]
```

```
          0  1      2
0   Jessica  F  46470
1    Ashley  F  45553
2  Brittany  F  36534
3    Amanda  F  34405
4  Samantha  F  25865
```

Out[123…

	0	2
0	Jessica	46470
1	Ashley	45553
2	Brittany	36534
3	Amanda	34405
4	Samantha	25865
...	...	...
24710	Zeus	5
24711	Ziyad	5
24712	Zoilo	5
24713	Zoran	5
24714	Zvi	5

24715 rows × 2 columns

Part 1)

In [105…

```
def top_name(k,year):
    data=pd.read_csv(f'Names/yob{year}.txt',header=None)
    return (data[[0,2]].iloc[:k])

top_name(100,2000)
```

Out[105...

	0	2
0	Emily	25953
1	Hannah	23075
2	Madison	19967
3	Ashley	17997
4	Sarah	17689
...	...	...
95	Leah	3395
96	Katie	3391
97	Gabriella	3369
98	Cheyenne	3367
99	Cassandra	3305

100 rows × 2 columns

In [ ]:

Part 2)

In [125...

```
print(data[0].nunique())
print(len(data[0]))
```

```
22675
24715
```

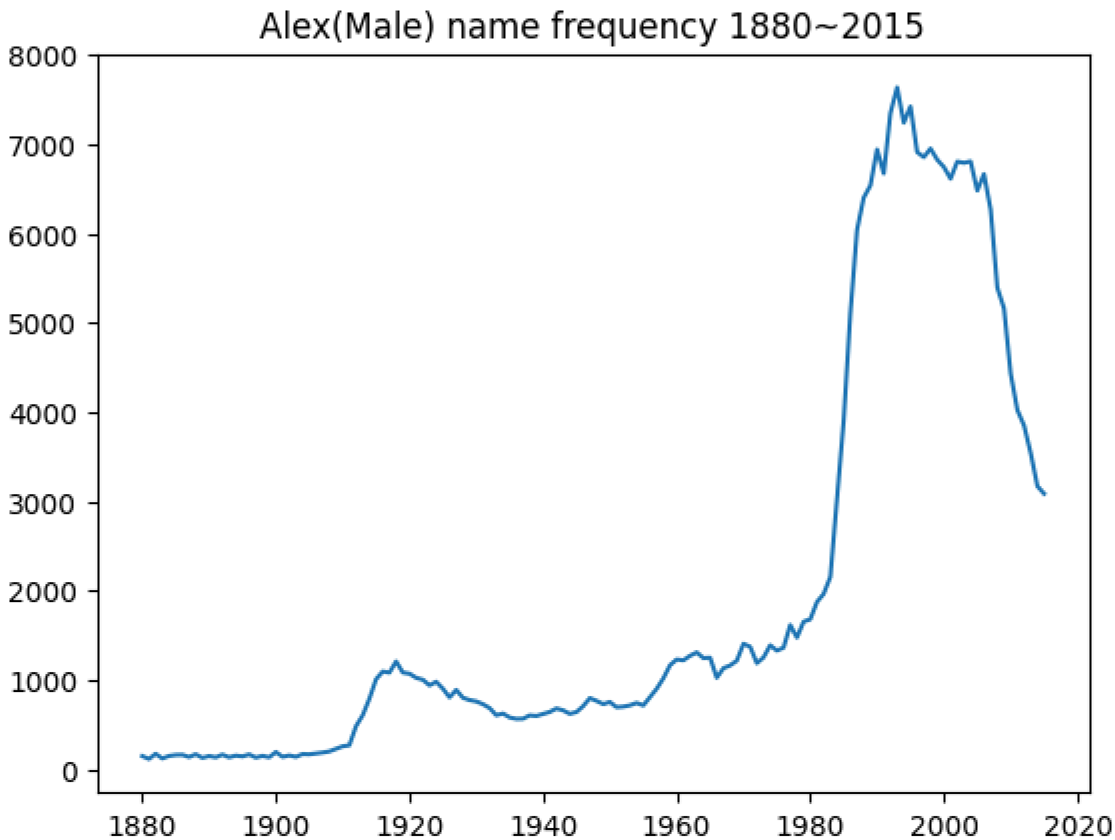
In [162...

```
def name_freq(name,sex='M', year_start=1880,year_end=2015):
    name_freq=[]
    for year in range(year_start,year_end+1):
        data=pd.read_csv(f'Names/yob{year}.txt',header=None)
        freq=data[2].loc[(data[0]==name) & (data[1]==sex)]
        name_freq.append(freq.item())
    return name_freq

count=name_freq('Alex','M',1880,2015)
print(count)
plt.plot(np.arange(1880,2016),count)
plt.title("Alex(Male) name frequency 1880~2015")
```

```
[147, 114, 172, 120, 148, 159, 161, 136, 167, 127, 147, 133, 163, 133, 152, 143, 16
5, 130, 149, 133, 192, 138, 154, 138, 169, 166, 176, 186, 199, 227, 256, 268, 482, 6
04, 789, 1008, 1094, 1081, 1207, 1082, 1070, 1023, 1002, 939, 979, 902, 804, 888, 80
0, 774, 759, 728, 684, 604, 623, 579, 563, 565, 603, 596, 619, 641, 680, 659, 620, 6
40, 706, 796, 765, 729, 752, 694, 700, 715, 742, 714, 809, 900, 1017, 1164, 1229, 12
19, 1270, 1308, 1244, 1249, 1024, 1131, 1162, 1215, 1405, 1371, 1187, 1252, 1388, 13
28, 1361, 1615, 1474, 1652, 1679, 1873, 1963, 2153, 3023, 3902, 5106, 6040, 6401, 65
38, 6941, 6672, 7348, 7636, 7240, 7422, 6911, 6855, 6951, 6826, 6744, 6613, 6805, 67
91, 6807, 6480, 6666, 6268, 5393, 5161, 4429, 4023, 3849, 3539, 3170, 3085]
```

Out[162…

```
Text(0.5, 1.0, 'Alex(Male) name frequency 1880~2015')
```

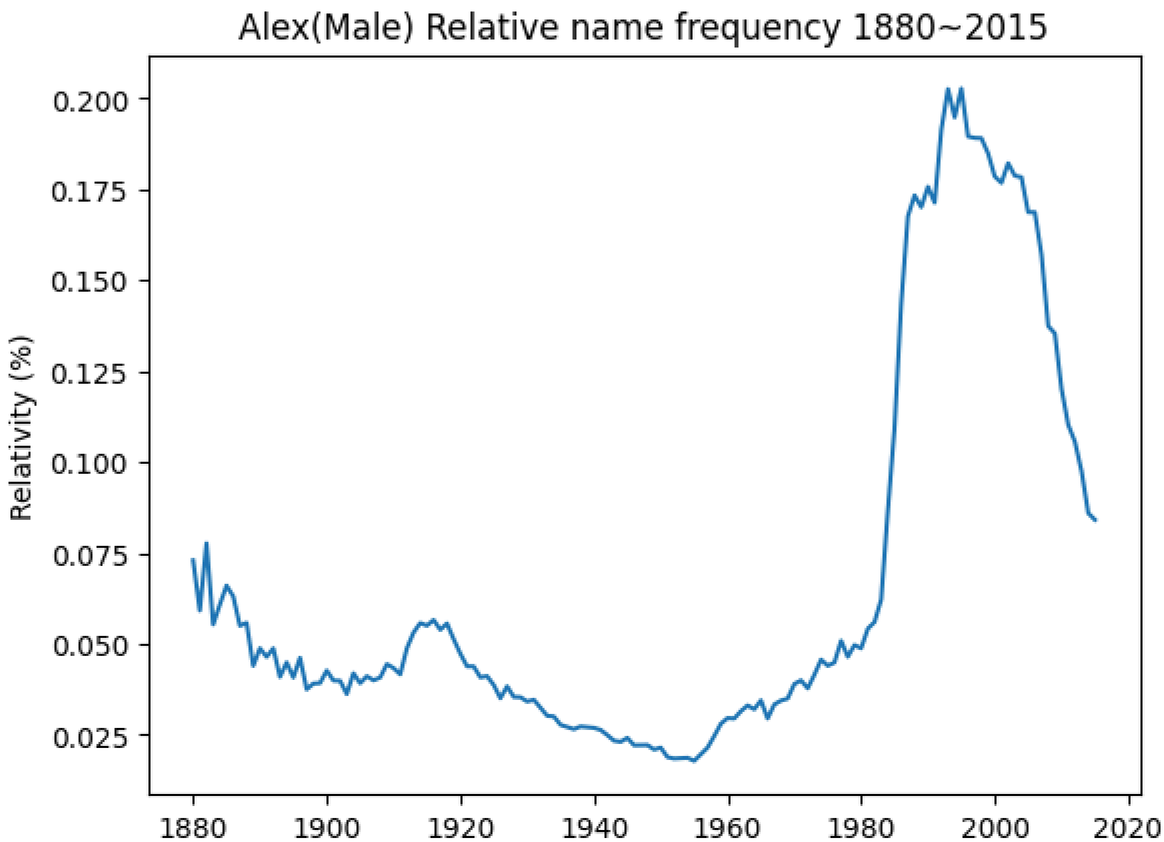


Part 3) Relative Frequency

In [172…

```python
def relative_name_freq(name,sex='M', year_start=1880,year_end=2015):
    rel_freq=[]
    for year in range(year_start,year_end+1):
        data=pd.read_csv(f'Names/yob{year}.txt',header=None)
        freq=data[2].loc[(data[0]==name) & (data[1]==sex)]
        total=data[2].sum()
        rel=freq/total
        rel_freq.append(rel.item()*100)
    return rel_freq

count=relative_name_freq('Alex','M',1880,2015)
# print(count)
plt.plot(np.arange(1880,2016),count)
plt.title("Alex(Male) Relative name frequency 1880~2015")
plt.ylabel("Relativity (%)")
```

Out[172… Text(0, 0.5, 'Relativity (%)')



Part 4)

In [549…

```
#part 4

# Helper function
import zipfile
def open_names_file(start_year, end_year = 0):
  years = {}
  if end_year == 0:
    end_year = start_year
  zip_file = 'Names.zip'
  with zipfile.ZipFile(zip_file, 'r') as zip_ref: # Opens the zipfile in read mode
    for year in range(start_year,end_year + 1):
      file_name = f'Names/yob{year}.txt'  # File path
      with zip_ref.open(file_name) as file:
          names = pd.read_csv(file, header=None, names=['Name', 'Gender', 'Frequenc
          years[year] = names
  if start_year == end_year:
    return names
  return years

def gender_change():
  gender_changes = []
  people = {}
  years = open_names_file(1880, 2015)
  for year in range(1880, 2015): # For every file
    for person in years[year].itertuples():
```

```
      name = person.Name
      frequency = person.Frequency
      gender = person.Gender
      if name not in people: # no occurences yet of name
        people[name] = {'Frequency': frequency, 'Gender': gender, 'Year': year}
      else:
        new_freq = frequency
        old_freq = people[name]['Frequency']
        new_gender = gender
        old_gender = people[name]['Gender']
        old_year = people[name]['Year']
        if new_freq > old_freq:
          people[name]['Frequency'] = new_freq
          if new_gender != old_gender:
            people[name]['Gender'] = new_gender
            if old_year < year:
              gender_changes.append(name)

  gender_changes =  sorted(set(gender_changes))
  return gender_changes[:100]


print(f"List of names that have changed which gender is most popular: {gender_chang
```

```
List of names that have changed which gender is most popular: ['Aalijah', 'Aamari',
'Aaren', 'Aareon', 'Aarian', 'Aarin', 'Aaris', 'Aavyn', 'Abba', 'Abbey', 'Abell', 'A
bey', 'Abir', 'Abrar', 'Abraxas', 'Abriel', 'Aby', 'Abyan', 'Acelin', 'Adair', 'Adal
i', 'Adama', 'Adar', 'Addis', 'Addison', 'Adel', 'Adi', 'Adis', 'Adisa', 'Adison',
'Adley', 'Adrean', 'Adryan', 'Aeon', 'Afsheen', 'Agam', 'Ahmari', 'Ahmi', 'Aideen',
'Aidyn', 'Aidynn', 'Aijalon', 'Aiman', 'Aimar', 'Aime', 'Aimen', 'Ainsley', 'Airen',
'Aires', 'Aivan', 'Ajai', 'Ajene', 'Aki', 'Akira', 'Akon', 'Alaa', 'Alai', 'Albany',
'Alder', 'Aldyn', 'Aleph', 'Alexandr', 'Alexi', 'Alexie', 'Alexis', 'Alexius', 'Alex
iz', 'Alexus', 'Alexy', 'Alfie', 'Alfonsa', 'Ali', 'Alijah', 'Alika', 'Alin', 'Ali
x', 'Aliyan', 'Allah', 'Allex', 'Alley', 'Allison', 'Allyn', 'Almer', 'Alonzia', 'Al
tair', 'Alter', 'Altonia', 'Alva', 'Alvern', 'Alvia', 'Aly', 'Alyjah', 'Alyn', 'Amad
i', 'Amahri', 'Amandeep', 'Amaree', 'Amari', 'Amarii', 'Amarri']
```

Problem 5

We looked at the MNIST data set in class. Recall that MNIST is a data set of handwritten digits. It is considered one of the ``easiest'' image recognition problems in computer vision. You can find the MNIST data set which we will use, here: https://www.openml.org/d/554. Though we haven't introduced decision trees formally, we have had a chance to see them in action in class. This exercise is an opportunity to play around with this data set, and in advance of when we get to talk about decision trees in detail, have a chance to see how they work. In short, this is an exercise in learning-by-doing.

Part 1. (Nothing to submit) Make sure you can run through the entire Colab notebook posted. Especially if you haven't used Python, try to understand what every line is doing.

Part 2. How many data points are there, how many features are there, and what do the features represent?

Part 3. Compute how many times each digit appears in the dataset.

Part 4. Read the documentation for sklearn.model_selection.train_test_split and explain what this does.

Part 5. Read the documentation for DecisionTreeClassifier, and explain what score means.

Part 6. What happens to the **training score** as you increase the depth of the tree? Explain.

Part 7. What happens to the difference between **training score** and **testing score** as you increase the depth of the tree? Explain.

Part 8. Fix the depth of the three, say, depth=7. Then plot the difference of training score - testing score when you train on: 100, 500, 5000, 10000, 15,0000, 20,000, 25,000 points, always computing testing score by evaluating on the complement of the training set. Plot this trend. Try to explain what you are seeing.

Part 1)

In [415…
```
"run complete"
```
Out[415…
```
'run complete'
```

Part 2)

In [417…
```
from sklearn import tree
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state

# Load data from https://www.openml.org/d/554
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)

# X contains the feature vectors, and y the labels

# Now let's put it into a numpy array
X = X.to_numpy()
y = y.to_numpy()
```

In [424…
```
print(X.shape,y.shape)
print('There are 70,000 datapoints, and 784 features. Each Feature is 28*28 image p
```

```
(70000, 784) (70000,)
There are 70,000 datapoints, and 784 features. Each Feature is 28*28 image pixel num
ber 0~255 grayscale
```

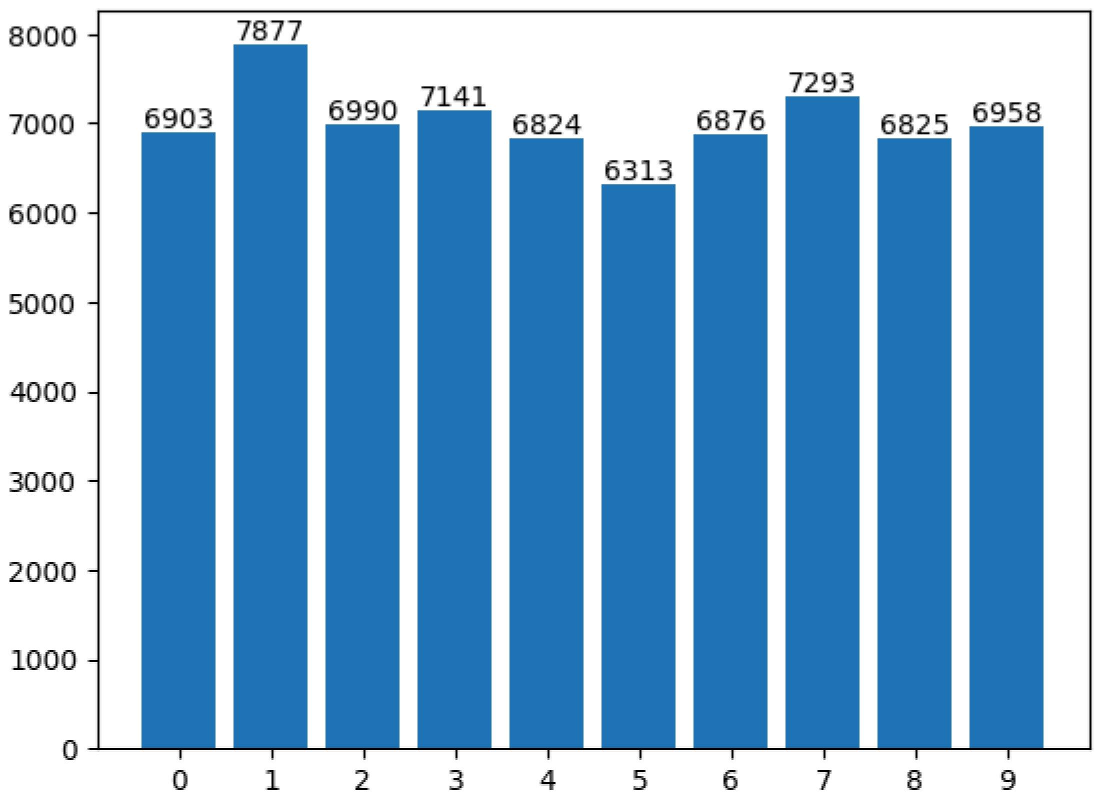
Part 3)

In [482…

```
each_num=pd.Series(y).value_counts().sort_index()
idx=each_num.index
value=each_num.values
plt.bar(idx,value)

# Add data points on top of each bar
for i, v in enumerate(value):
    plt.text(i, v + 50, str(v), ha='center')
each_num
```

Out[482…

```
0    6903
1    7877
2    6990
3    7141
4    6824
5    6313
6    6876
7    7293
8    6825
9    6958
Name: count, dtype: int64
```



Part 4)

In [490…

```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=44)

print("It splits and shuffle the dataset, train size: 80% of data, test: 20% of dat
print(f"random_state parameter set the random seed of shuffling dataset. \n\nDatase
```

```
It splits and shuffle the dataset, train size: 80% of data, test: 20% of data if we
set test_size=0.2
random_state parameter set the random seed of shuffling dataset.

Dataset shape: ((56000, 784), (56000,), (14000, 784), (14000,))
```

Part 5)

score(X, y, sample_weight=None)[source]

Return the **mean accuracy on the given test data and labels.**

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Part 6)

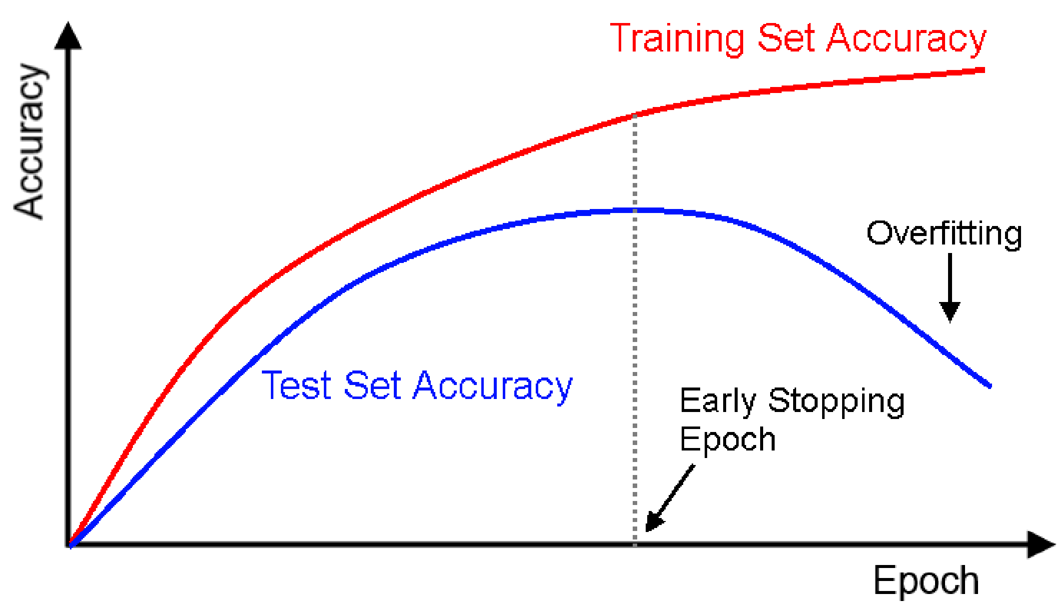
What happens to the training score as you increase the depth of the tree?

The train score will be in crease when we increase the depth, since the model would be more precise,sensitive to understand the small changes of the data.

Part 7)

What happens to the difference between training score and testing score as you increase the depth of the tree?

The train score will increase at first when we increase the depth, but after at some point, if depth become too complicated and deep, the test accuracy(unseen data) will be decrease. Which means overfitting. (Image below)

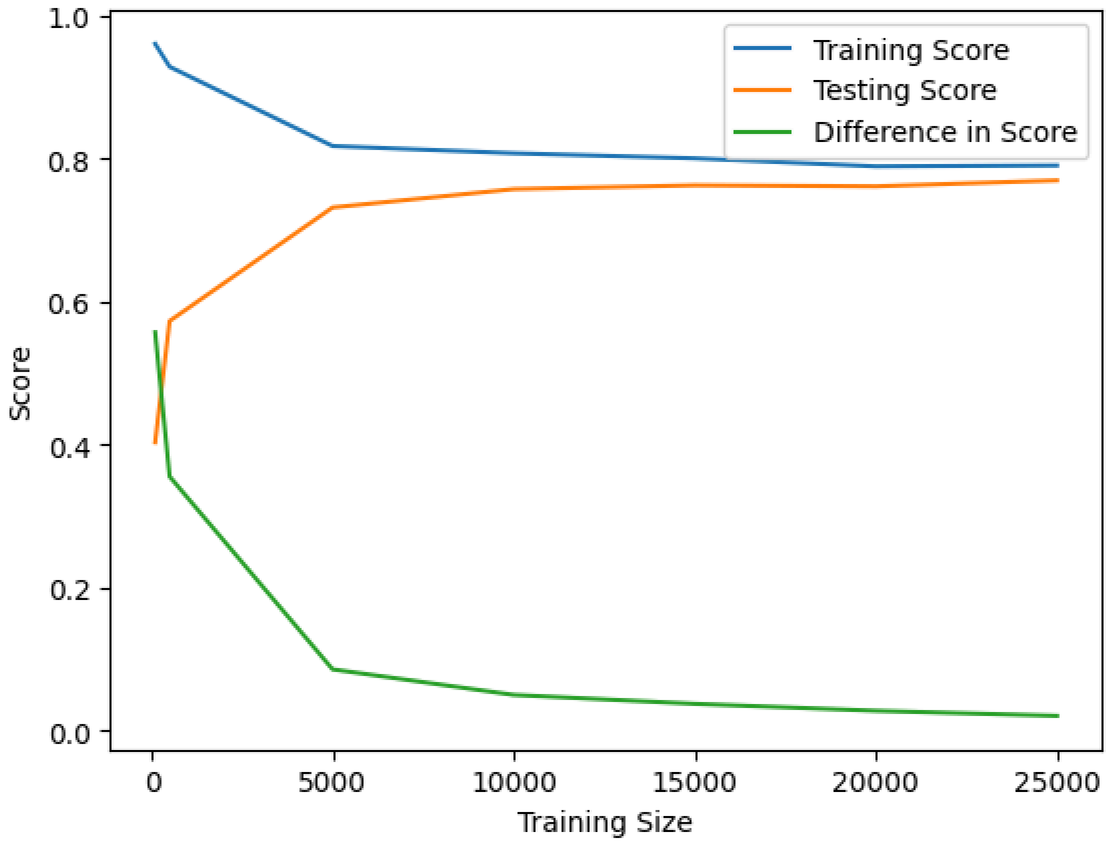


Part 8)

In [552…

```
import time
from sklearn import tree
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state

# Part 2
X, y = fetch_openml('mnist_784', version=1, return_X_y=True, parser='auto')


# Now let's put it into a numpy array
X = X.to_numpy() # the actual data
y = y.to_numpy() # the labels
data_points, num_features = X.shape
print(f"Number of data points: {data_points} and number of features: {num_features}

# Part 3
counters = {'0': 0, '1': 0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8': 0,
for num in y:
  counters[num] += 1
print(f"Number of each label: {counters}")

# Part 8
training_sizes = [100, 500, 5000, 10000, 15000, 20000, 25000]
train_score = []
test_score = []
for size in training_sizes:
  X_train, X_test, y_train, y_test = train_test_split(
      X, y, train_size=size, random_state=42)
  decision_tree = tree.DecisionTreeClassifier(max_depth=7)
  decision_tree = decision_tree.fit(X_train, y_train)
  train_score.append(decision_tree.score(X_train, y_train))
  test_score.append(decision_tree.score(X_test, y_test))
  difference = np.subtract(train_score,test_score)
plt.plot(training_sizes,train_score, label = "Training Score")
plt.plot(training_sizes, test_score, label="Testing Score")
plt.plot(training_sizes, difference, label="Difference in Score")
plt.legend()
plt.xlabel(f"Training Size")
plt.ylabel("Score")
plt.show()
```

```
Number of data points: 70000 and number of features: 784
Number of each label: {'0': 6903, '1': 7877, '2': 6990, '3': 7141, '4': 6824, '5': 6
313, '6': 6876, '7': 7293, '8': 6825, '9': 6958}
```



Explanation:

As the training dataset increases, the train accuracy(score)-test accuracy difference decreases. That means, As model is learning more data, the model is getting better performance to unseen data.

Problem 6

We now turn to a somewhat more sophisticated data set: CIFAR10. Here is an initial colab notebook: https://colab.research.google.com/drive/1H3a4yVuZLatBvFjrUp5aFBJn_vfmXj7o?usp=sharing

Part 1. How many data points are there, and how many labels? How many points for each label?

Part 2. There are two ``TO DOs'' listed in the colab notebook. Complete these.

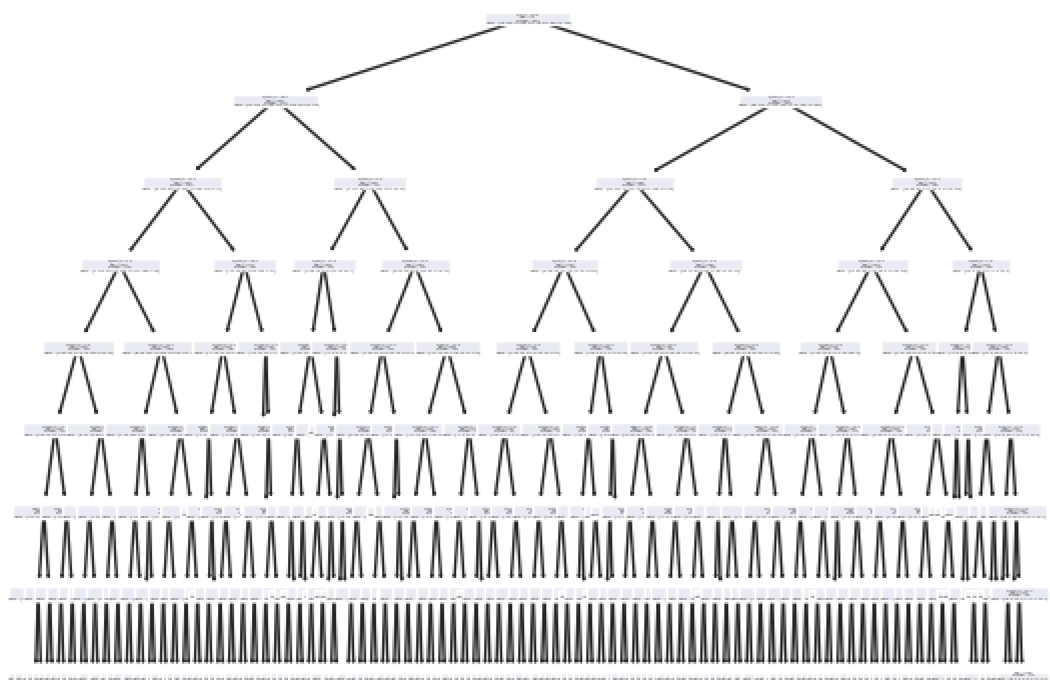
If you did this correctly and ran the notebook, you noticed that CIFAR10 indeed looks like a ``harder'' problem. Deep trees are again doing very well on the training set, and they do a little better than guessing on the testing data, but not as well as they do on MNIST. We will revisit CIFAR10 several times, as we develop more powerful tools. And we will see that we will do much better than deep decision trees!

Part 1)

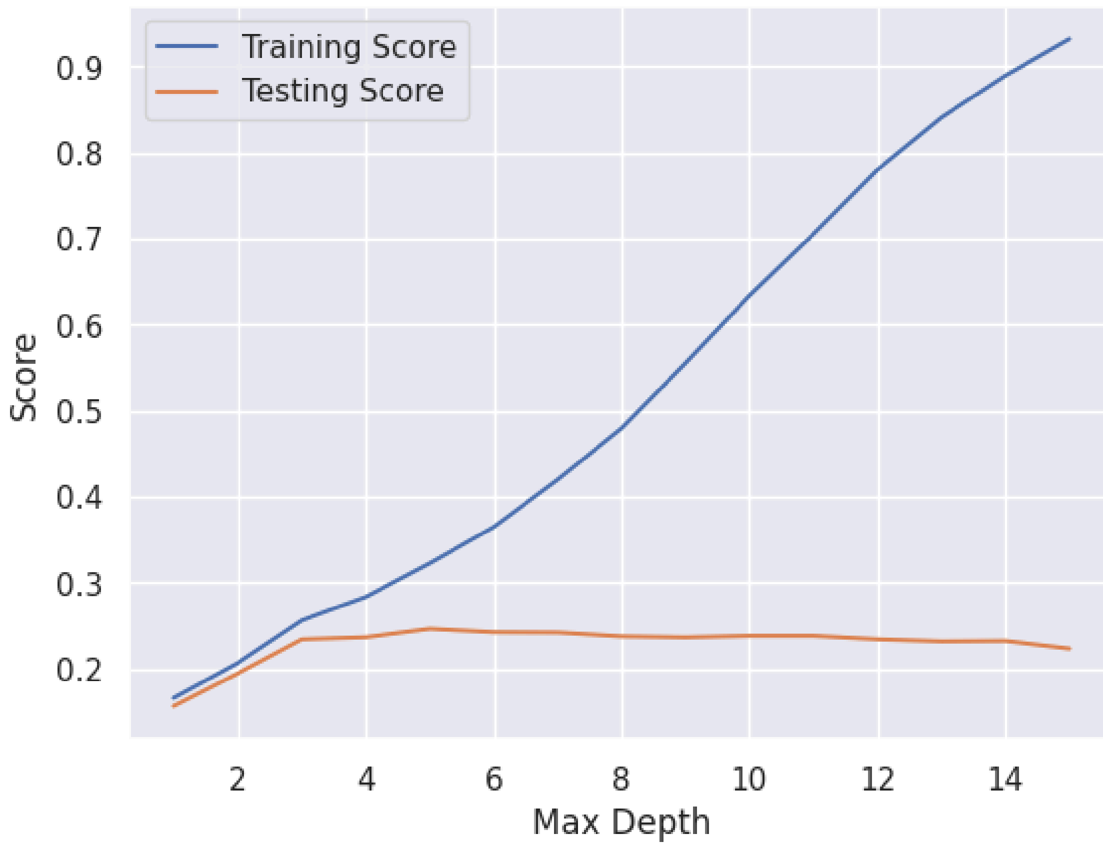
How many data points are there, and how many labels? How many points for each label?

There are 60000 data, 10 labels (each class have 6000 images) , 32 x 32 x 3 data points for each label.

Part 2)



Train score: 0.4794



Training Score
Testing Score
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Score
2
4
6
8
10
12
14
Max Depth