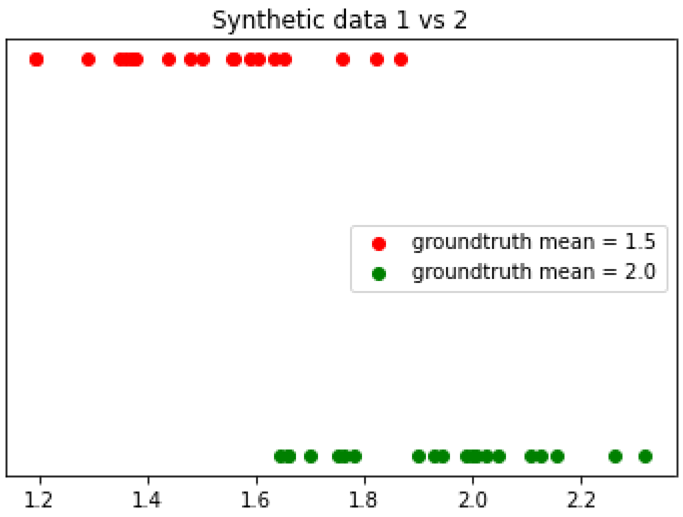
Part 1

In [1]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

rng = np.random.default_rng(seed=22197823)
```

q1.a

In [2]:
```
# generate sample data
std_gt = 0.2
mean_1_gt = 1.5
mean_2_gt = 2.0
N = 20

group1 = rng.normal(loc=mean_1_gt, scale=std_gt, size=N)
group2 = rng.normal(loc=mean_2_gt, scale=std_gt, size=N)

mean_1_calc = group1.mean()
mean_2_calc = group2.mean()
std_1_calc = group1.std()
std_2_calc = group2.std()

print(f'Data 1: mean : {mean_1_calc:.02f}, std: {std_1_calc:.02f}. Expected sum of
print(f'Data 2: mean : {mean_2_calc:.02f}, std: {std_2_calc:.02f}. Expected mean ar
print(f'\nExpected sum of square difference (SSD) from the mean : {(N * std_gt**2)
print(f'Data 1 SSD : {((mean_1_gt - group1)**2).sum():.2f}')
print(f'Data 2 SSD : {((mean_2_gt - group2)**2).sum():.2f}')

plt.scatter(group1, 2 * np.ones(N), color='r', label=f'groundtruth mean = {mean_1_g
plt.scatter(group2, 1 * np.ones(N), color='g', label=f'groundtruth mean = {mean_2_g
plt.yticks([])
plt.legend()
plt.title('Synthetic data 1 vs 2')
plt.show()
```

```
Data 1: mean : 1.50, std: 0.19. Expected sum of sqare: 1.5, 0.2
Data 2: mean : 1.94, std: 0.20. Expected mean and std: 2.0, 0.2

Expected sum of square difference (SSD) from the mean : 0.80
Data 1 SSD : 0.70
Data 2 SSD : 0.86
```



q1.b

In [3]:
```
t_statistic_gt, p_val = stats.ttest_ind(group1, group2)
print(f'T statistic: {t_statistic_gt:.2f}, p-value: {p_val:.12f}')
```

```
T statistic: -7.04, p-value: 0.000000021510
```

In [4]:
```
print('2 sided tail test')
2*stats.t.cdf(t_statistic_gt, 38)
```

```
2 sided tail test
```

Out[4]:
```
2.1510080928959593e-08
```

q1.c i)

In [5]:
```
# for Y = X1 * B1 + X2 * B2 + e, what is the design matrix?
# synthetic data is from 2 groups

# design matrix:
X = np.zeros(shape=(40,2))
X[:20, 0] = 1
X[20:, 1] = 1

Y = np.hstack([group1, group2])

rank = np.linalg.matrix_rank(X)
print(f'rank of C(X) = {rank}')
```

```
rank of C(X) = 2
```

q1.c ii)

In [6]:
```
Px = X @ np.linalg.inv(X.T @ X) @ X.T
print(Px)
print(f'Trace(Px) = {np.trace(Px):.2f}')
```

```
[[0.05 0.05 0.05 ... 0.   0.   0.  ]
 [0.05 0.05 0.05 ... 0.   0.   0.  ]
 [0.05 0.05 0.05 ... 0.   0.   0.  ]
 ...
 [0.   0.   0.   ... 0.05 0.05 0.05]
 [0.   0.   0.   ... 0.05 0.05 0.05]
 [0.   0.   0.   ... 0.05 0.05 0.05]]
Trace(Px) = 2.00
```

q1.c iii)

In [7]:

```
# use Px to find Y_hat
Y_hat = Px @ Y
print(f'error between Y and Y_hat: {(Y - Y_hat).sum():.2f}')
```

```
error between Y and Y_hat: -0.00
```

In [8]:

```
X.shape
```

Out[8]:

```
(40, 2)
```

q1.c iv)

In [9]:

```
d = Px.shape[0]
Rx = np.identity(d) - Px

eps = 1e-9

if np.abs((Rx @ Rx - Rx).sum()) < eps and np.abs((Rx - Rx.T).sum()) < eps:
    print(f'Rx = (I - Px) has passed numerical tests for being a perpendicular proj
else:
    print(f'Rx FAILED a numerical test for being a perpendicular projection operato
```

```
Rx = (I - Px) has passed numerical tests for being a perpendicular projection oper
ator
```

q1.c v)

In [10]:

```
error_hat = Rx @ Y
error_hat

error_space_dim = np.linalg.matrix_rank(Rx)

print(f'error space dim: {error_space_dim}')
text = 'error_hat = ['
for e in error_hat[:-1]:
    text += f'{e:.3f}, '
text += f'{error_hat[-1]:.3f}]'
print(text)
```

```
error space dim: 38
error_hat = [0.090, 0.263, 0.060, 0.001, -0.120, -0.129, -0.305, -0.140, -0.124,
0.106, 0.370, 0.058, 0.154, -0.019, -0.061, 0.324, -0.149, -0.210, -0.303, 0.135,
0.108, 0.169, -0.177, 0.323, 0.216, 0.007, 0.381, -0.279, 0.052, -0.009, -0.039, -
0.240, 0.070, -0.158, -0.280, -0.188, 0.086, 0.189, 0.062, -0.292]
```

q1.c vi)

In [11]:
```
# normalise the vectors then calc the angle
numerator = np.dot(error_hat, Y_hat)
divisor = np.sqrt(np.dot(error_hat, error_hat) * np.dot(Y_hat, Y_hat))
angle = np.arccos(numerator / divisor) / np.pi

print(f'angle between Y_hat and error_hat is {angle:.2f} * pi')
print(f'we expect error_hat and Y_hat to be perpendicular, so the angle should be 0
```

```
angle between Y_hat and error_hat is 0.50 * pi
we expect error_hat and Y_hat to be perpendicular, so the angle should be 0.5 * pi
```

q1.c vii)

In [12]:
```
M = np.linalg.inv(X.T @ X)
beta = M @ X.T @ Y
Y_hat_1 = X @ beta
diff = Y_hat - Y_hat_1
print(f'Difference when calculating Y_hat using Y = X @ Beta: {diff.sum():.2f}')
print(beta)
```

```
Difference when calculating Y_hat using Y = X @ Beta: -0.00
[1.49781955 1.93797768]
```

q1.c viii)

In [13]:
```
numerator = np.dot(error_hat, error_hat)
n = X.shape[0]
divisor = n - np.linalg.matrix_rank(X)
var_hat = numerator / divisor
var_hat
```

Out[13]:
```
0.0390570392238516
```

q1.c ix)

In [14]:
```
S_beta = var_hat * np.linalg.inv(X.T @ X)
std_beta1 = np.sqrt(S_beta[0,0])
std_beta2 = np.sqrt(S_beta[1,1])

print(f'standard deviation for Beta_1 : {std_beta1:.4f}, for Beta_2 : {std_beta2:.4
print(S_beta)
```

```
standard deviation for Beta_1 : 0.0442, for Beta_2 : 0.0442
[[0.00195285 0.        ]
 [0.         0.00195285]]
```

q1.c x)

In [15]:
```python
# calculate the contrast vector lmbda and the reduced model X_0

lmbda = np.asarray([1, -1])
X_0 = X @ np.asarray([1, 1])
X_0 = X_0.reshape((-1,1))
```

q1.c xi)

In [16]:
```python
# calculate the error from the reduced model

Px_0 = X_0 @ np.linalg.inv(X_0.T @ X_0) @ X_0.T
d = Px_0.shape[0]
I = np.identity(d)
Rx_0 = (I - Px_0)
error_0_hat = Rx_0 @ Y

# SSR = sum(Y_mean - Y_hat)**2
# we have error_hat = Y - Y_hat -> so introduce Y_error = Y_mean - Y
v1 = np.trace(Px - Px_0)
v2 = np.trace(I - Px)
Y_error = Y.mean() - Y
SSR_X0 = np.square(Y_error + error_0_hat).sum()
SSR_X = np.square(Y_error + error_hat).sum()

F_numerator = (SSR_X0 - SSR_X) / v1
F_denominator = SSR_X / v2

F_statistic = F_numerator / F_denominator
print(f'F statistic comparing the reduced model to the full model: {F_statistic:.2f

V = lmbda @ beta
S_V = np.sqrt(lmbda.reshape((2,1)).T @ S_beta @ lmbda.reshape((2,1)))
t_df = np.squeeze(V/S_V)

print(f'the degrees of freedom of the F statistic is ({v1:.0f}, {v2:.0f})')
print(f'p-value = {1 - stats.f.cdf(-F_statistic, v1, v2)}')
```

```
F statistic comparing the reduced model to the full model: -38.00
the degrees of freedom of the F statistic is (1, 38)
p-value = 3.3873272231588203e-07
```

q1.c xii)

In [17]:
```python
# calculate the t-statistic

numerator = lmbda @ beta
denominator = np.sqrt(lmbda.reshape((1,-1)) @ S_beta @ lmbda.reshape((-1,1)))[0,0]
t_statistic = numerator / denominator
print(f't-statistic for different means: {t_statistic:.2f}')
print(f'difference between t-statistic calculated at the begining : {t_statistic -
```

```
t-statistic for different means: -7.04
difference between t-statistic calculated at the begining : 0.00000
```

q1.c xiv)

In [18]:
```
# calcualte error from ground truth (Y_gt)
Y_gt = np.ones(Y.shape[0])
Y_gt[:20] = 1.5
Y_gt[20:] = 2.0

error = Y_gt - Y
error_projected_CX = Px @ error
error_projected_CX
```

Out[18]:
```
array([0.00218045, 0.00218045, 0.00218045, 0.00218045, 0.00218045,
       0.00218045, 0.00218045, 0.00218045, 0.00218045, 0.00218045,
       0.00218045, 0.00218045, 0.00218045, 0.00218045, 0.00218045,
       0.00218045, 0.00218045, 0.00218045, 0.00218045, 0.00218045,
       0.06202232, 0.06202232, 0.06202232, 0.06202232, 0.06202232,
       0.06202232, 0.06202232, 0.06202232, 0.06202232, 0.06202232,
       0.06202232, 0.06202232, 0.06202232, 0.06202232, 0.06202232,
       0.06202232, 0.06202232, 0.06202232, 0.06202232, 0.06202232])
```

In [19]:
```
beta - np.array([1.5,2])
```

Out[19]:
```
array([-0.00218045, -0.06202232])
```

q1.c xv)

In [20]:
```
# error projected into not(C(x))

error_projected_not_CX = Rx @ error
error_projected_not_CX

text = 'error_projected_not_CX = ['
for e in error_projected_not_CX[:-1]:
    text += f'{e:.3f}, '
text += f'{error_hat[-1]:.3f}]'
print(text)
print(f'error_hat diff to this: {(error_hat + error_projected_not_CX).sum()}')
```

```
error_projected_not_CX = [-0.090, -0.263, -0.060, -0.001, 0.120, 0.129, 0.305, 0.1
40, 0.124, -0.106, -0.370, -0.058, -0.154, 0.019, 0.061, -0.324, 0.149, 0.210, 0.3
03, -0.135, -0.108, -0.169, 0.177, -0.323, -0.216, -0.007, -0.381, 0.279, -0.052,
0.009, 0.039, 0.240, -0.070, 0.158, 0.280, 0.188, -0.086, -0.189, -0.062, -0.292]
error_hat diff to this: -5.4088677980956845e-15
```

q1.d i)

In [21]:
```
X_intercept = np.zeros(shape=(Y.shape[0], 3))
X_intercept[:,   0] = 1
X_intercept[:20, 1] = 1
X_intercept[20:, 2] = 1

print(f'design matrix X has rank {np.linalg.matrix_rank(X_intercept)}')
```

```
design matrix X has rank 2
```

q1.d ii)

In [22]:
```
Z = X_intercept
Px_intercept = Z @ np.linalg.pinv(Z.T @ Z) @ Z.T
Px_intercept
```

Out[22]:
```
array([[ 5.0000000e-02,  5.0000000e-02,  5.0000000e-02, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       [ 5.0000000e-02,  5.0000000e-02,  5.0000000e-02, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       [ 5.0000000e-02,  5.0000000e-02,  5.0000000e-02, ...,
         0.0000000e+00,  0.0000000e+00,  0.0000000e+00],
       ...,
       [-6.9388939e-18, -6.9388939e-18, -6.9388939e-18, ...,
         5.0000000e-02,  5.0000000e-02,  5.0000000e-02],
       [-6.9388939e-18, -6.9388939e-18, -6.9388939e-18, ...,
         5.0000000e-02,  5.0000000e-02,  5.0000000e-02],
       [-6.9388939e-18, -6.9388939e-18, -6.9388939e-18, ...,
         5.0000000e-02,  5.0000000e-02,  5.0000000e-02]])
```

q1.d iii)

In [23]:
```
lmbda_intercept = np.asarray([0,1,-1]).reshape((1,-1))
mult = np.asarray([[1, 0],
                   [0, 1],
                   [0, 1]])
X_0_intercept = X_intercept @ mult
X_0_intercept.shape
```

Out[23]:
```
(40, 2)
```

q1.d iv)

In [24]:
```
# calculate the t-statistic

d = Y.shape[0]
I = np.identity(d)
Rx_intercept = (I - Px_intercept)
error_hat_intercept = Rx_intercept @ Y
M = np.linalg.pinv(X_intercept.T @ X_intercept)
beta_intercept = M @ X_intercept.T @ Y

numerator = (np.dot(error_hat_intercept, error_hat_intercept))
denominator = d - np.linalg.matrix_rank(X_intercept)
var_hat_intercept = numerator / denominator

S_beta_intercept = var_hat_intercept * M

numerator = lmbda_intercept @ beta_intercept
denominator = np.sqrt(lmbda_intercept @ S_beta_intercept @ lmbda_intercept.T)
t_statistic_intercept = (numerator / denominator)[0,0]
t_statistic_intercept
```

Out[24]:
```
-7.043022482674422
```

In [25]:
```
beta_intercept
```

Out[25]: array([1.14526574, 0.3525538 , 0.79271194])

q1.e i)

In [26]:
```
X_e= np.zeros(shape=(Y.shape[0], 2))
X_e[:,    0] = 1
X_e[:20, 1] = 1

print(f'design matrix X has rank {np.linalg.matrix_rank(X_e)}')
```

```
design matrix X has rank 2
```

q1.e ii)

In [27]:
```
lmbda_e = np.asarray([0, 1]).reshape((2, -1))
```

q1.e iii)

In [28]:
```
# calculate the t-statistic
Z = X_e
Px_e = Z @ np.linalg.pinv(Z.T @ Z) @ Z.T
Px_e

d = Y.shape[0]
I = np.identity(d)
Rx_e = (I - Px_e)
error_hat_e = Rx_e @ Y
M = np.linalg.pinv(X_e.T @ X_e)
beta_e = M @ X_e.T @ Y

numerator = (np.dot(error_hat_e, error_hat_e))
denominator = d - np.linalg.matrix_rank(X_e)
var_hat_e = numerator / denominator

S_beta_e = var_hat_e * M

numerator = np.dot(lmbda_e.flatten(), beta_e)
denominator = np.sqrt(lmbda_e.T @ S_beta_e @ lmbda_e)
t_statistic_e = (numerator / denominator)[0,0]
print(t_statistic_e)
```

```
-7.043022482674408
```

q2.a i)

In [29]:
```
# now computing the ttest for null hypothesis 2 samples come from the same
# distribution with the same mean
t_statistic_1sample, p_val_1sample = stats.ttest_rel(group1, group2)
print(f'ttest for 1 sample distribution: t = {t_statistic_1sample:.2f}, p-value = 
print(f'ttest for 2 sample distribution: t = {t_statistic:.2f}, p-value = {p_val:.8
```

```
ttest for 1 sample distribution: t = -6.13, p-value = 0.00000678
ttest for 2 sample distribution: t = -7.04, p-value = 0.00000002
```

q2.b i)

In [30]:
```
# create the design matrix
X_2b = np.zeros(shape=(40,22))
X_2b[:,    0] = 1
X_2b[20:, 1] = 1
for i in range(20):
    X_2b[i,     i+2] = 1
    X_2b[20+i, i+2] = 1

print(f'rank of X is: {np.linalg.matrix_rank(X_2b)}')
```

```
rank of X is: 21
```

In [31]:
```
# create lmbda
lmbda_2b = np.zeros(22)
lmbda_2b[1] = -1
lmbda_2b = lmbda_2b.reshape((22,1))
```

q2.b iii)

In [32]:
```
# calculate the t-statistic
Z = X_2b
Px_2b = Z @ np.linalg.pinv(Z.T @ Z) @ Z.T
Px_2b

d = Y.shape[0]
I = np.identity(d)
Rx_2b = (I - Px_2b)
error_hat_2b = Rx_2b @ Y
M = np.linalg.pinv(X_2b.T @ X_2b)
beta_2b = M @ X_2b.T @ Y

numerator = (np.dot(error_hat_2b, error_hat_2b))
denominator = d - np.linalg.matrix_rank(X_2b)
var_hat_2b = numerator / denominator

S_beta_2b = var_hat_2b * M

numerator = np.dot(lmbda_2b.flatten(), beta_2b)
denominator = np.sqrt(lmbda_2b.T @ S_beta_2b @ lmbda_2b)
t_statistic_2b = (numerator / denominator)[0,0]
t_statistic_2b
```

Out[32]:
```
-6.13254658116648
```