```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import scipy.stats as stats
        from sklearn.linear_model import LogisticRegression
```

Part 1

```
In [2]: def eta(p):
            return np.log(p/(1-p))
```

1.

```
In [3]: B=400
        df = pd.read_csv('adult_clean.csv')
        df['income'] = df['income'].astype('category')
        df['income'] = df['income'].cat.codes
        df_200 = df.sample(n=200)
        df_200 = df_200[['educational-num', 'income']]
```

```
In [4]: df_1000 = df.drop(df_200.index).sample(n=1000)
```

2.a

```
In [5]: tau= df['educational-num'].median()
        print(tau)
```

```
10.0
```

```
In [6]: df_200_1 = df_200[df_200['income'] == 1]
        n_1 = df_200_1[df_200_1['educational-num'] > tau].count()[0]
        len1 = len(df_200_1['educational-num'])
        p_1 = n_1 / len1
        df_200_0 = df_200[df_200['income'] == 0]
        n_0 = df_200_0[df_200_0['educational-num'] > tau].count()[0]
        len0 = len(df_200_0['educational-num'])
        p_0 = n_0 / len0
```

We can use proportion as a plug-in estimator to calculate the estimator for ψ

```
In [7]: psi_hat = eta(p_1) - eta(p_0)
        print(psi_hat)
```

```
1.7346010553881066
```

```
In [8]: psi_boot = np.zeros((B))
        for b in range(B):
            df_boot = df_200.sample(n=200, replace=True)
            df_boot_1 = df_boot[df_boot['income'] == 1]
            len1_boot = len(df_boot_1['educational-num'])
            n_1_boot = df_boot_1[df_boot_1['educational-num'] > tau].count()[0]
            p_1_boot = n_1_boot / len1_boot
            df_boot_0 = df_boot[df_boot['income'] == 0]
            n_0_boot = df_boot_0[df_boot_0['educational-num'] > tau].count()[0]
            len0_boot = len(df_boot_0['educational-num'])
```

```
    p_0_boot = n_0_boot / len0_boot
    psi_boot[b] = eta(p_1_boot) - eta(p_0_boot)
```

We've decided to us percentile interval because our distribution is not Normal

In [9]:
```
psi_boot.sort()
conf_int = np.percentile(psi_boot, [2.5, 97.5])
print(conf_int)
```

```
[1.05006885 2.48509115]
```

1.b as seen in the lecture,we can use the uniform prior to calculate the posterior distribution as such: $p_1|X^n \sim Beta(S_X + 1, m - S_X + 1)$
$p_2|Y^n \sim Beta(S_Y + 1, n - S_Y + 1)$

In [10]:
```
beta_1 = stats.beta(a=n_1 + 1, b=len1 - n_1 + 1)
beta_0 = stats.beta(a=n_0 + 1, b=len0 - n_0 + 1)
# sample from the beta distribution 500 times
beta_1_sample = beta_1.rvs(size=B)
beta_0_sample = beta_0.rvs(size=B)
# calculate psi for each sample
psi_beta_interval = np.zeros((B))
for b in range(B):
    psi_beta_interval[b] = eta(beta_1_sample[b]) - eta(beta_0_sample[b])
```

Using Plug-in we can calculate the estimator as such: $\hat{\psi} = \frac{1}{B}\Sigma(\eta(p_1) - \eta(p_0))$ and the credibility interval will be calculated as instructed in the course's book

In [11]:
```
psi_beta_hat = np.mean(psi_beta_interval)
print(f"Psi estimator : {psi_beta_hat}")
psi_beta_interval.sort()
conf_int_beta = np.percentile(psi_beta_interval, [2.5, 97.5])
print(f"Credibility Interval : {conf_int_beta}")
```

```
Psi estimator : 1.734146528575526
Credibility Interval : [1.08271895 2.51666547]
```

2.c Because we use jeffery's prior we can calculate the posterior distribution as such: $p_1|X^n \sim Beta(S_X + 0.5, m - S_X + 0.5)$ $p_2|Y^n \sim Beta(S_Y + 0.5, n - S_Y + 0.5)$ and that's because jeffery's prior distribution is as follows: $\sqrt{\frac{1}{p*(1-p)}}$

In [12]:
```
beta_1_j = stats.beta(a=n_1 + 0.5, b=len1 - n_1 + 0.5)
beta_0_j = stats.beta(a=n_0 + 0.5, b=len0 - n_0 + 0.5)

beta_1_sample_j = beta_1_j.rvs(size=B)
beta_0_sample_j = beta_0_j.rvs(size=B)
# calculate psi for each sample
psi_beta_interval_j = np.zeros((B))
for b in range(B):
    psi_beta_interval_j[b] = eta(beta_1_sample_j[b]) - eta(beta_0_sample_j[b])
```

In [13]:
```
psi_beta_hat_j = np.mean(psi_beta_interval_j)
print(f"Psi estimator : {psi_beta_hat_j}")
psi_beta_interval_j.sort()
conf_int_beta_j = np.percentile(psi_beta_interval_j, [2.5, 97.5])
print(f"Credibility Estimator : {conf_int_beta_j}")
```

```
Psi estimator : 1.7184344434189336
Credibility Estimator : [1.01533065 2.42543798]
```

1.d First we will dichotomize our 1000 sample data

```
In [14]: df_1000_1 = df_1000[df_1000['income'] == 1]
n_1000_1 = df_1000_1[df_1000_1['educational-num'] > tau].count()[0]
len_1000_1 = len(df_1000_1['educational-num'])
p_1000_1 = n_1000_1 / len_1000_1
df_1000_0 = df_1000[df_1000['income'] == 0]
n_1000_0 = df_1000_0[df_1000_0['educational-num'] > tau].count()[0]
len_1000_0 = len(df_1000_0['educational-num'])
p_1000_0 = n_1000_0 / len_1000_0
```

Now, to construct a prior from these samples we will start from a uniform prior and do the same process we did in 2.b
$p_1|X^{m1000} \sim Beta(S1000_X + 1, m1000 - S1000_X + 1)$
$p_2|Y^{n1000} \sim Beta(S1000_Y + 1, n1000 - S1000_Y + 1)$ Now we will use these distributions as the priors for our next phase,we can calculate our next posterior distribution as such(as seen in the lecture questions):
$f(\theta|x^n) = L_n(\theta)\pi(\theta) \propto p^S(1-p)^{n-S}p^{\alpha-1}(1-p)^{\beta-1} = p^{S+\alpha+1}(1-p)^{n-S+\beta-1} \propto Beta($
Thus,we can write our posterior distributions as such using our 200 sample data:
$p_1|X^{m200} \sim Beta(S200_X + S1000_X + 1, m200 - S200_X + m1000 - S1000_X + 1)$
$p_2|Y^{n200} \sim Beta(S200_Y + S1000_Y + 1, n200 - S200_Y + n1000 - S1000_Y + 1)$

```
In [15]: beta_1000_1 = stats.beta(a=n_1+n_1000_1 + 1, b=len_1000_1 - n_1000_1 + 1+len1-n_
beta_1000_0 = stats.beta(a=n_1000_0 + 1+n_0, b=len_1000_0 - n_1000_0 + 1+len0-n_
```

```
In [16]: beta_1000_1_sample = beta_1000_1.rvs(size=B)
beta_1000_0_sample = beta_1000_0.rvs(size=B)
# calculate psi for each sample
psi_beta_interval_1000 = np.zeros((B))
for b in range(B):
    psi_beta_interval_1000[b] = eta(beta_1000_1_sample[b]) - eta(beta_1000_0_sam
```

```
In [17]: psi_beta_hat_1000 = np.mean(psi_beta_interval_1000)
print(f"Psi estimator : {psi_beta_hat_1000}")
psi_beta_interval_1000.sort()
conf_int_beta_1000 = np.percentile(psi_beta_interval_1000, [2.5, 97.5])
print(f"Credibility Interval : {conf_int_beta_1000}")
```

```
Psi estimator : 1.2918897119551247
Credibility Interval : [1.00041226 1.56788553]
```

2.e We can see that our Credibility Interval is tighter for our latest estimator while the estimator itself doesn't differ greatly from the previous ones.

Part 2

1.

```
In [18]: df1000_full = pd.read_csv('adult_clean.csv')
df1000_full['gender'] = df1000_full['gender'].astype('category')
```

```
df1000_full['gender'] = df1000_full['gender'].cat.codes
df1000_full = df1000_full[['age', 'educational-num', 'gender', 'hours-per-week']
# male==1,female==0
df1000_full = df1000_full.sample(n=1000)
df1000_full = df1000_full.reset_index(drop=True)
```

2. We can assume asymptotic normality for the coefficients during linear regression

In [19]:
```
X_1000_full = df1000_full[['age', 'educational-num', 'gender']]
Y_1000_full = df1000_full[['hours-per-week']]
X_1000_full.insert(0, 'Ones', 1)
beta1000_hat = np.linalg.inv(X_1000_full.T.dot(X_1000_full)).dot(X_1000_full.T).
print(f"Coefficients estimators for full DF :\n {beta1000_hat}")

# Covariance matrix
cov1000_hat = np.linalg.inv(X_1000_full.T.dot(X_1000_full))
CI1000_full = np.zeros((4, 2))
for i in range(4):
    CI1000_full[i, 0] = beta1000_hat[i] - 1.96 * np.sqrt(cov1000_hat[i, i])
    CI1000_full[i, 1] = beta1000_hat[i] + 1.96 * np.sqrt(cov1000_hat[i, i])
print(f"confidence interval for the full DF :\n {CI1000_full}")
```

```
Coefficients estimators for full DF :
 [[25.64049795]
 [ 0.10119782]
 [ 0.83127996]
 [ 4.97556858]]
confidence interval for the full DF :
 [[25.32371361 25.95728229]
 [ 0.096304    0.10609163]
 [ 0.8068002   0.85575973]
 [ 4.84154376  5.1095934 ]]
```

3.

In [20]:
```
# reorder df1000_full in ascending order of hours-per-week
df1000_full = df1000_full.sort_values(by=['hours-per-week'])

# list of 1000 increasing numbers starting from 1/5 to 4/5 where number 500 is 1
problist = np.linspace(1 / 5, 4 / 5, 1000)
indicators = np.zeros((1000, 1))
count = 0
for i in range(1000):
    # sample from bernoulli distribution with probability problist[i]
    indicators[i] = np.random.binomial(1, problist[i])
    if indicators[i] == 1:
        count += 1
    if count == 500:
        break

df_removed = df1000_full.copy()
for i in range(1000):
    if indicators[i] == 1:
        df_removed.loc[i, 'hours-per-week'] = np.nan

print(f"{count} rows were removed")
```

```
499 rows were removed
```

4.a once again we can assume asymptotic normality for the coefficients during linear regression

In [21]:
```
X_removed = df_removed[['age', 'educational-num', 'gender']]
X_removed.insert(0, 'Ones', 1)

# remove rows with nan values
df_dropped = df_removed.dropna()
X_dropped = df_dropped[['age', 'educational-num', 'gender']]
Y_dropped = df_dropped[['hours-per-week']]
X_dropped.insert(0, 'Ones', 1)
beta_dropped = np.linalg.inv(X_dropped.T.dot(X_dropped)).dot(X_dropped.T).dot(Y_
print(f"Coefficients estimators for dropped DF :\n {beta_dropped}")

# Covariance matrix
cov_dropped = np.linalg.inv(X_dropped.T.dot(X_dropped))
CI_dropped = np.zeros((4, 2))
for i in range(4):
    CI_dropped[i, 0] = beta_dropped[i] - 1.96 * np.sqrt(cov_dropped[i, i])
    CI_dropped[i, 1] = beta_dropped[i] + 1.96 * np.sqrt(cov_dropped[i, i])
print(f"confidence interval for the dropped DF :\n {CI_dropped}")
```

```
Coefficients estimators for dropped DF :
 [[27.14880685]
 [ 0.10804456]
 [ 0.73634326]
 [ 4.421321  ]]
confidence interval for the dropped DF :
 [[26.7008447  27.59676899]
 [ 0.10098164  0.11510748]
 [ 0.70195585  0.77073067]
 [ 4.22932484  4.61331716]]
```

4.b

In [22]:
```
# regress imputed values on the removed values
df_regimputed = df_removed.copy()
for i in range(1000):
    if indicators[i] == 1:
        # impute the removed values
        df_regimputed.loc[i, 'hours-per-week'] = X_removed.loc[i].dot(beta_dropp

X_regimputed = df_regimputed[['age', 'educational-num', 'gender']]
Y_regimputed = df_regimputed[['hours-per-week']]
X_regimputed.insert(0, 'Ones', 1)
beta_regimputed = np.linalg.inv(X_regimputed.T.dot(X_regimputed)).dot(X_regimput
print(f"Coefficients estimators for regression imputed DF :\n {beta_regimputed}"

# Covariance matrix
cov_regimputed = np.linalg.inv(X_regimputed.T.dot(X_regimputed))
CI_regimputed = np.zeros((4, 2))
for i in range(4):
    CI_regimputed[i, 0] = beta_regimputed[i] - 1.96 * np.sqrt(cov_regimputed[i,
    CI_regimputed[i, 1] = beta_regimputed[i] + 1.96 * np.sqrt(cov_regimputed[i,
print(f"confidence interval for the regression imputed DF :\n {CI_regimputed}")
```

```
Coefficients estimators for regression imputed DF :
 [[27.14880685]
 [ 0.10804456]
 [ 0.73634326]
 [ 4.421321  ]]
confidence interval for the regression imputed DF :
 [[26.8320225  27.46559119]
 [ 0.10315074  0.11293838]
 [ 0.7118635   0.76082303]
 [ 4.28729618  4.55534582]]
```

The estimators we got are very similar to the previous ones because we used them as the model,furthermore we have excluded the noise.

4.c as noted we will assume asymptotic normality

In [23]:
```
# multiple imputation using 10 imputed datasets
df_mi = [[0 for x in range(1000)] for y in range(10)]
for i in range(10):
    df_mi[i] = df_removed.copy()
    for j in range(1000):
        if indicators[j] == 1:
            # impute the removed values
            df_mi[i].loc[j, 'hours-per-week'] = X_removed.loc[j].dot(beta_droppe

beta_mi = []
cov_mi = []
M = 10
for i in range(M):
    df_mi[i]['hours-per-week'] = df_mi[i]['hours-per-week'] - np.random.normal(0
    X_mi = df_mi[i][['age', 'educational-num', 'gender']]
    Y_mi = df_mi[i][['hours-per-week']]
    X_mi.insert(0, 'Ones', 1)
    current_beta = np.linalg.inv(X_mi.T.dot(X_mi)).dot(X_mi.T).dot(Y_mi)
    beta_mi.append(current_beta)
    cov_mi.append(np.linalg.inv(X_mi.T.dot(X_mi)))
mean_beta_mi = np.mean(beta_mi, axis=0)
print(f"mean of coefficients estimators for multiple imputed DF :\n {mean_beta_m
# confidence interval for the multiple imputed DF
CI_mi = np.zeros((4, 2))
for i in range(4):
    CI_mi[i, 0] = mean_beta_mi[i] - 1.96 * np.sqrt(np.mean(cov_mi, axis=0)[i, i]
    CI_mi[i, 1] = mean_beta_mi[i] + 1.96 * np.sqrt(np.mean(cov_mi, axis=0)[i, i]
print(f"confidence interval for the multiple imputed DF :\n {CI_mi}")
```

```
mean of coefficients estimators for multiple imputed DF :
 [[27.09716417]
 [ 0.10952723]
 [ 0.73672757]
 [ 4.40580412]]
confidence interval for the multiple imputed DF :
 [[26.78037983 27.41394851]
 [ 0.10463341  0.11442104]
 [ 0.7122478   0.76120734]
 [ 4.2717793   4.53982894]]
```

4.d

In [24]:
```
beta_sum = 0
cov_sum = 0
beta_robin = [0 for x in range(4)]
for j in range(len(beta_mi[0])):
    beta_sum = 0
    cov_sum = 0
    for i in range(M):
        beta_sum += ((M + 1) / (M * (M - 1))) * (beta_mi[i][j] - mean_beta_mi[j]
        cov_sum += (1 / M) * cov_mi[i][j, j]

    beta_robin[j] = float(beta_sum) + float(cov_sum)

for b in beta_robin:
    b = np.sqrt(b)
print(f"Robin's estimator for multiple imputed DF :\n {beta_robin}")
```

```
Robin's estimator for multiple imputed DF :
 [0.05128777141808928, 1.1469900829551632e-05, 0.0002725869621702706, 0.0095301
360724276]
```

4.e We will print 10 examples for the probability $P(R=1|X_1.\dots.X_k)$

In [25]:
```
# logistic regression for the removed values
df_logistic = df_removed.copy()

# logistic regression for the removed values from sklearn
X_logistic = df_logistic[['age', 'educational-num', 'gender']]
X_logistic.insert(0, 'Ones', 1)
# change indicators to df
df_logistic['indicators'] = indicators
Y_logistic = df_logistic[['indicators']]
logreg = LogisticRegression()
logreg.fit(X_logistic, Y_logistic.values.ravel())
print(f"Coefficients estimators for logistic regression  :\n {logreg.coef_}")
# print the probability P(Y=1|X) on right side of the equation
print(f"Probability of P(Y=1|X1.......Xk) for logistic regression  :\n {logreg.p
```

```
Coefficients estimators for logistic regression  :
 [[-8.68157160e-05  8.29514205e-03  5.76663493e-02  3.32372399e-01]]
Probability of P(Y=1|X1.......Xk) for logistic regression  :
 [0.44089157 0.43089001 0.5232488  0.45104029 0.43292533 0.56102905
 0.63428897 0.358026   0.50065376 0.43743716]
```

4.f $min_{\beta}||\beta^T((X*diag(weights))-Y*diag(weights)||$

In [26]:
```
# TODO: linear regression as a least squares problem with IPW
IPW = 1 / logreg.predict_proba(X_dropped)[:, 1]


X_dropped_ipw = X_dropped.copy()
X_dropped_ipw = X_dropped_ipw.T * IPW
X_dropped_ipw = X_dropped_ipw.T
Y_dropped_ipw = Y_dropped.copy()
Y_dropped_ipw = Y_dropped_ipw.T * IPW
Y_dropped_ipw = Y_dropped_ipw.T

beta_ipw = np.linalg.inv(X_dropped.T.dot(X_dropped_ipw)).dot(X_dropped.T).dot(Y_
print(f"Coefficients estimators for IPW  :\n {beta_ipw}")
```

```
Coefficients estimators for IPW  :
 [[26.59969839]
 [ 0.12991635]
 [ 0.70811612]
 [ 4.40763903]]
```

4.g

In [27]:

```python
beta_ipw_boot = [0 for x in range(B)]
for b in range(B):
    df_logistic_boot = df_logistic.sample(n=len(df_logistic['age']), replace=Tru
    X_logistic_boot = df_logistic_boot[['age', 'educational-num', 'gender']]
    X_logistic_boot.insert(0, 'Ones', 1)
    Y_logistic_boot = df_logistic_boot[['indicators']]
    df_dropped_boot = df_logistic_boot.dropna()
    X_dropped_boot = df_dropped_boot[['age', 'educational-num', 'gender']]
    X_dropped_boot.insert(0, 'Ones', 1)
    Y_dropped_boot = df_dropped_boot[['hours-per-week']]
    logreg_boot = LogisticRegression()
    logreg_boot.fit(X_logistic_boot, Y_logistic_boot.values.ravel())
    IPW_boot = 1 / logreg_boot.predict_proba(X_dropped_boot)[:, 1]
    X_dropped_ipw_boot = X_dropped_boot.copy()
    X_dropped_ipw_boot = X_dropped_ipw_boot.T * IPW_boot
    X_dropped_ipw_boot = X_dropped_ipw_boot.T
    Y_dropped_ipw_boot = Y_dropped_boot.copy()
    Y_dropped_ipw_boot = Y_dropped_ipw_boot.T * IPW_boot
    Y_dropped_ipw_boot = Y_dropped_ipw_boot.T
    beta_ipw_boot[b] = np.linalg.inv(X_dropped_boot.T.dot(X_dropped_ipw_boot)).d
        Y_dropped_ipw_boot)

mean_beta_ipw_boot = np.mean(beta_ipw_boot, axis=0)
print(f"mean of coefficients estimators for IPW Using Bootstrap  :\n {mean_beta_
# confidence interval for the IPW Using quantile
beta_ipw_boot = np.array(beta_ipw_boot)
CI_ipw_boot = np.zeros((4, 2))
for i in range(4):
    CI_ipw_boot[i, 0] = mean_beta_ipw_boot[i] - 1.96 * np.sqrt(np.mean(cov_mi, a
    CI_ipw_boot[i, 1] = mean_beta_ipw_boot[i] + 1.96 * np.sqrt(np.mean(cov_mi, a
print(f"confidence interval for the IPW Using Bootstrap :\n {CI_ipw_boot}")
```

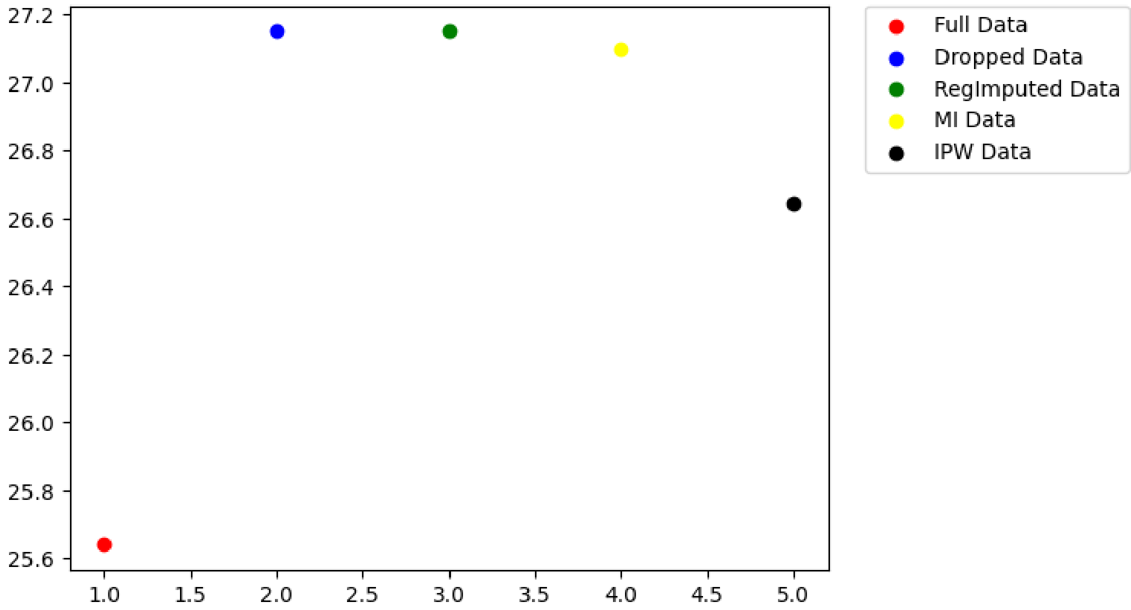
```
mean of coefficients estimators for IPW Using Bootstrap  :
 [[26.64416014]
 [ 0.12964963]
 [ 0.70736716]
 [ 4.35555918]]
confidence interval for the IPW Using Bootstrap :
 [[26.3273758  26.96094448]
 [ 0.12475581  0.13454344]
 [ 0.6828874   0.73184693]
 [ 4.22153437  4.489584  ]]
```

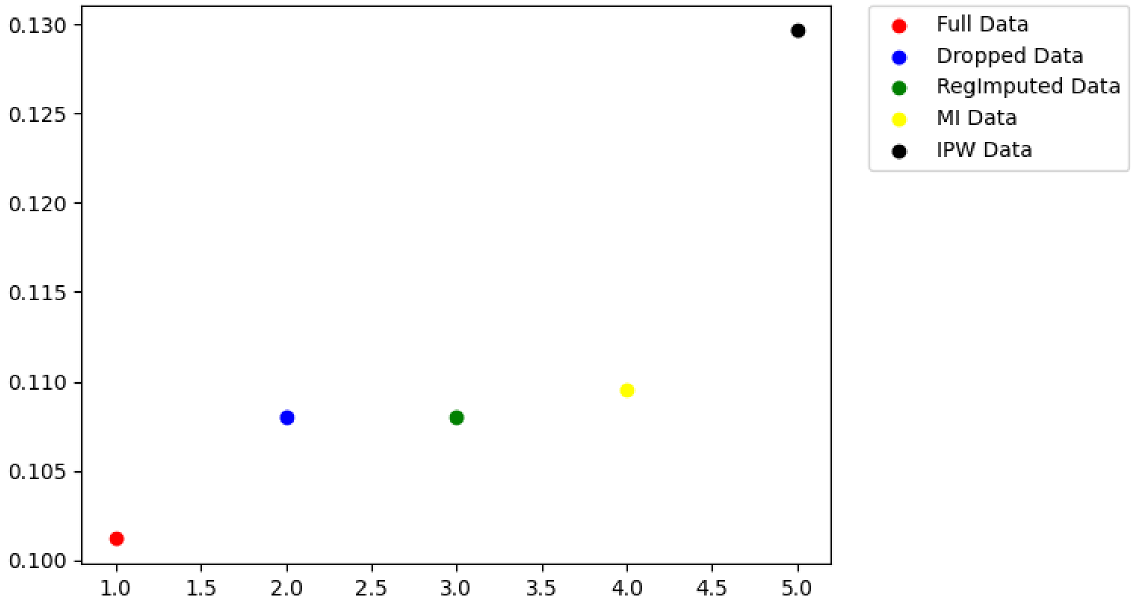
4.h.1

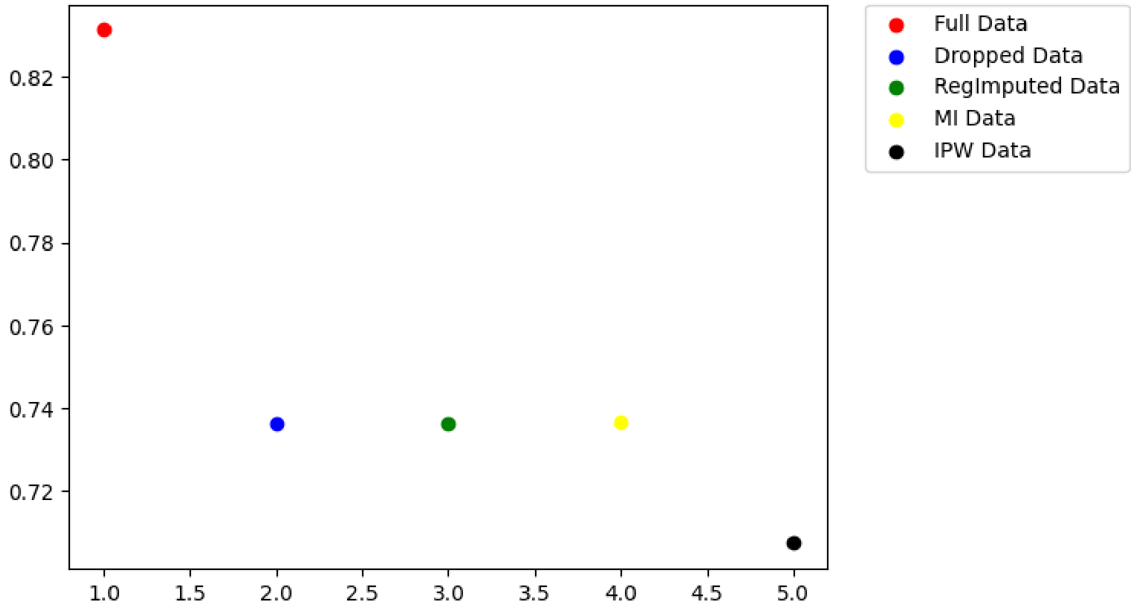
In [28]:

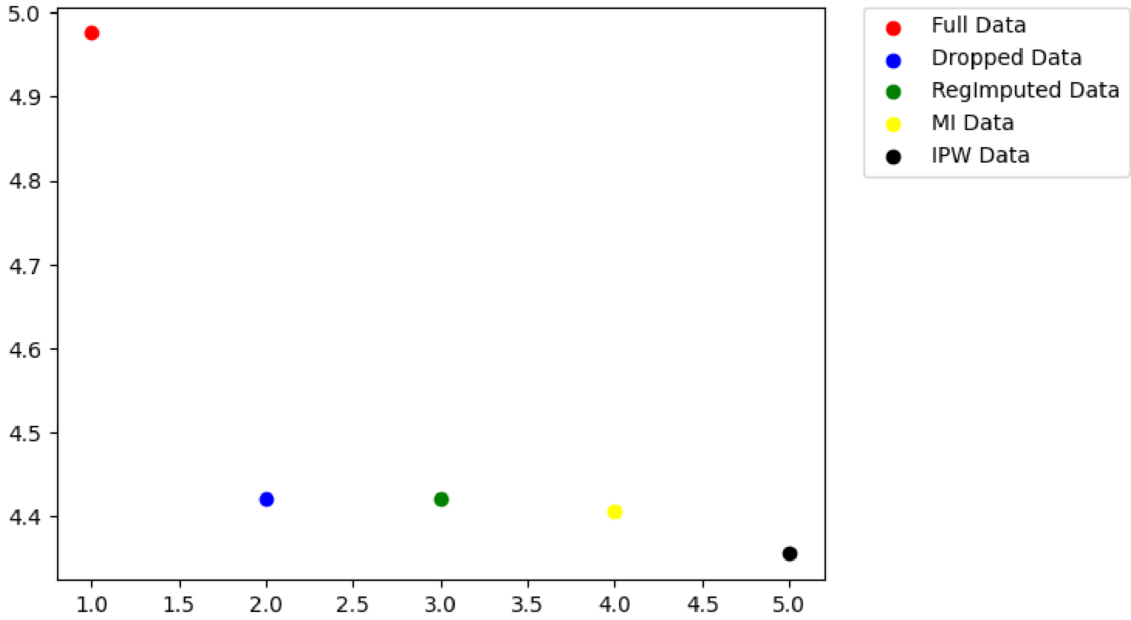
```python
for i in range(4):
    plt.scatter([1], [beta1000_hat[i] ], c=['red'],label='Full Data')
    plt.scatter([2], [ beta_dropped[i]], c=[ 'blue'],label='Dropped Data')
    plt.scatter([3], [ beta_regimputed[i]], c=[ 'green'],label='RegImputed Data'
    plt.scatter([4], [ mean_beta_mi[i]], c=[ 'yellow'],label='MI Data')
    plt.scatter([5], [ mean_beta_ipw_boot[i]], c=[ 'black'], label='IPW Data')
```

```
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
    plt.show()
```





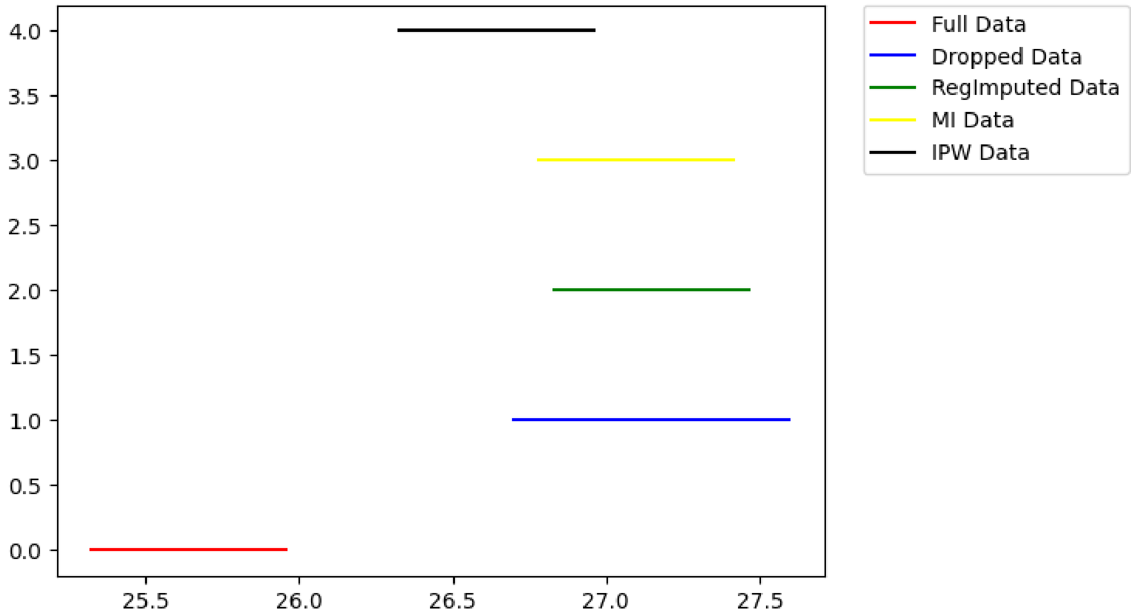


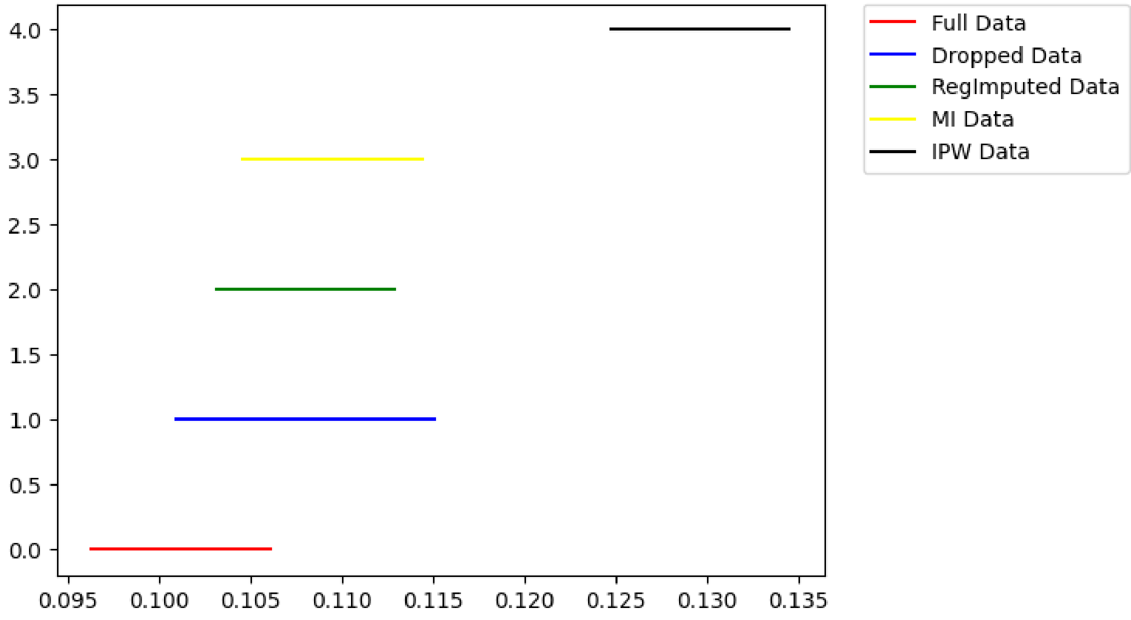


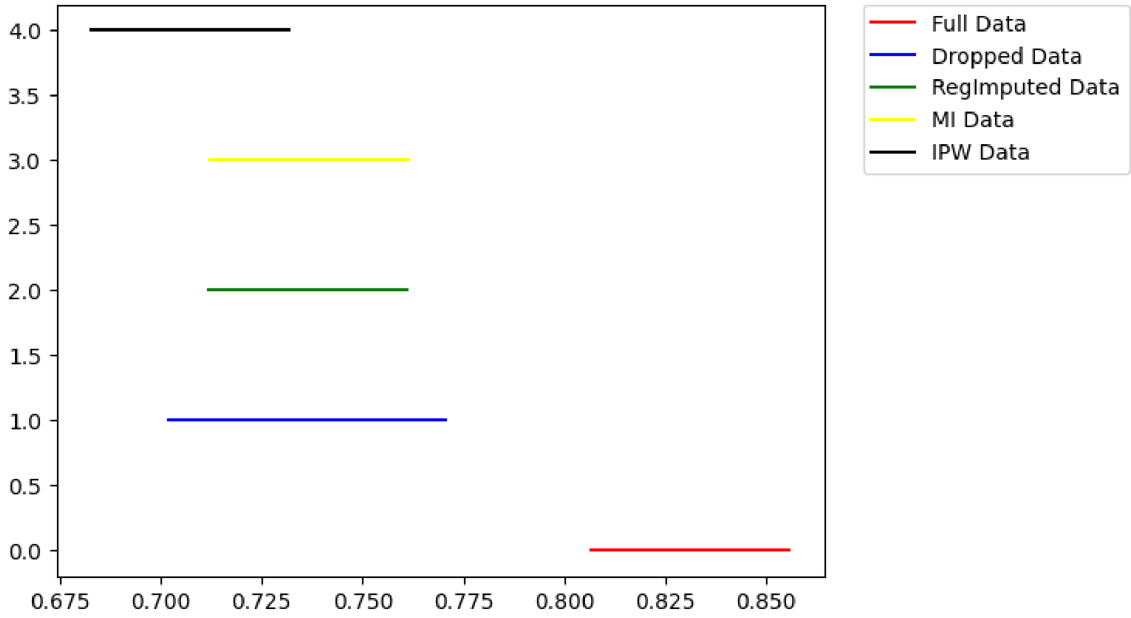
4.h.2

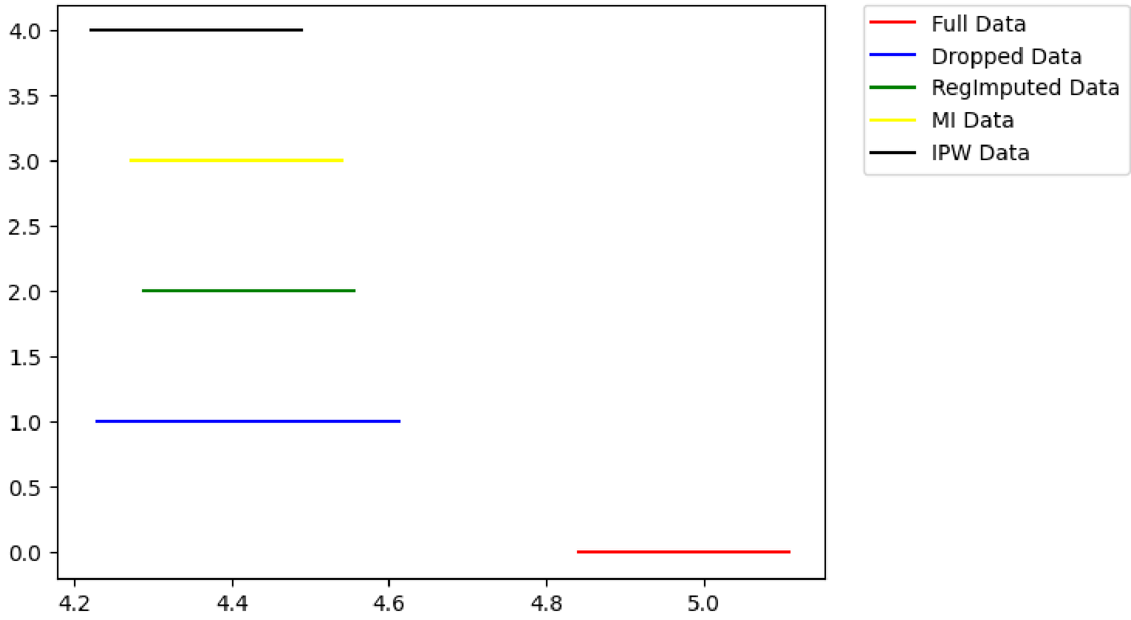
In [30]:

```
for i in range(4):
    plt.plot(CI1000_full[i],[0,0], label='Full Data',c='red')
    plt.plot(CI_dropped[i],[1,1], label='Dropped Data',c='blue')
    plt.plot(CI_regimputed[i],[2,2], label='RegImputed Data',c='green')
    plt.plot(CI_mi[i],[3,3], label='MI Data',c='yellow')
    plt.plot(CI_ipw_boot[i],[4,4], label='IPW Data',c='black')



    # Add a legend to the plot
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
    # Show the plot
    plt.show()
```









As we can see above both Estimators and Confidence Intervals of the IPW method are (except for the fourth coefficient) the closest to the full data.

In [30]:

```

```