1.1) $\mathbb{E}\left[\sum_{i=1}^{n} W_i R_i\right] = \sum_{i=1}^{n} \mathbb{E}[W_i R_i]$

$= \sum_{i=1}^{n} \mathbb{E}\left[\mathbb{E}[W_i R_i | W_i]\right]$ Adam's Law

$= \sum_{i=1}^{n} \mathbb{E}\left[W_i \mathbb{E}[R|W_i]\right]$ Taking out what is known

$= \sum_{i=1}^{n} \mathbb{E}\left[W_i \left(0 \cdot P(R=0|W_i) + 1 \cdot P(R=1|W_i)\right)\right]$

$= \sum_{i=1}^{n} \mathbb{E}\left[W_i \pi(X_i)\right]$ def of π

$= \sum_{i=1}^{n} \mathbb{E}[1]$ def of $W_i = \frac{1}{\pi(X_i)}$

$= \sum_{i=1}^{n} 1$

$= n$

1.2.1) $\hat{m}_{cc} = \frac{\sum_{i=1}^{n} R_i Y_i}{\sum_{i=1}^{n} R_i}$. We apply WLLN on the numerator and denominator, then Slutsky's Theorem, as per the hint.

Numerator:

$R_1 Y_1, \ldots, R_n Y_n$ are i.i.d w/ $\mathbb{E}[R_i Y_i] < \infty \; \forall i$, so

$\frac{1}{n}\sum_{i=1}^{n} R_i Y_i \xrightarrow{p} \mathbb{E}[R_i Y_i] = \mathbb{E}\left[\mathbb{E}[R_i Y_i | X]\right]$ Adam Law

$= \mathbb{E}\left[\mathbb{E}[R_i|X]\,\mathbb{E}[Y_i|X]\right]$ MAR assumption

$= \mathbb{E}\left[\pi(X_i) m(X_i)\right]$ def of π, m

$= \mathbb{E}\left[\pi(X) m(X)\right]$ X_i, X are i.i.d.

Denominator:

Similarly,

$\frac{1}{n}\sum_{i=1}^{n} R_i \xrightarrow{p} \mathbb{E}[R_i] = \mathbb{E}\left[\mathbb{E}[R_i|X_i]\right]$ Adam's Law

$= \mathbb{E}\left[0 \cdot P(R_i=0|X_i) + 1 \cdot P(R_i=1|X_i)\right]$

$= \mathbb{E}\left[\pi(X_i)\right]$

$= \mathbb{E}\left[\pi(X)\right]$

Now, By Slutsky's Theorem,

$\hat{m}_{cc} = \frac{\sum_{i=1}^{n} R_i Y_i}{\sum_{i=1}^{n} R_i} = \frac{\frac{1}{n}\sum_{i=1}^{n} R_i Y_i}{\frac{1}{n}\sum_{i=1}^{n} R_i} \xrightarrow{p} \frac{\mathbb{E}[\pi(X) m(X)]}{\mathbb{E}[\pi(X)]}$

1.2.2)

$$X \sim \text{Unif}(\{0,1,2\})$$

$$Y|X=x \sim N(x,1)$$

$$R|X=x \sim \text{expit}(4-4x)$$

First, $\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y|X]]$

$= \mathbb{E}[\mathbb{E}[N(x,1)]]$

$= \mathbb{E}[X]$

$= 1$

Now, $\hat{\mu}_{ccm} \to \dfrac{\mathbb{E}[\pi(X)m(X)]}{\mathbb{E}[m(X)]}$ (by 1.2.1)

$\pi(x) = \text{expit}(4-4x)$, and $m(x) = \mathbb{E}[N(x,1)] = X$

$$\hat{\mu}_{ccm} \to \frac{\mathbb{E}[\text{expit}(4-4X)X]}{\mathbb{E}[\text{expit}(4-4X)]} = \left[\frac{\frac{1}{3}[.982 \cdot 0 + .5 \cdot 1 + .018 \cdot 2}{\frac{1}{3}[.018 + .5 + .982]}\right]$$

$$= .3573$$

1.3.1) $$\mu_S = \frac{\sum_{i=1}^{n} W_i R_i Y_i}{\sum_{i=1}^{n} W_i R_i}$$

$$\hat{\mu}_S(D+a) = \frac{\sum_{i=1}^{n} W_i R_i (Y_i + a)}{\sum_{i=1}^{n} W_i R_i} = \frac{\sum_{i=1}^{n} (W_i R_i Y_i + W_i R_i a)}{\sum_{i=1}^{n} W_i R_i}$$

$$= \frac{\sum_{i=1}^{n} (W_i R_i Y_i) + a \sum_{i=1}^{n} W_i R_i}{\sum_{i=1}^{n} W_i R_i}$$

$$= \frac{\sum_{i=1}^{n} W_i R_i Y_i}{\sum_{i=1}^{n} W_i R_i} + a$$

In the complete case estimator, W_i is the same for all i, so we would get

$$\hat{\mu}_{mean}(D+a) = \frac{W \sum_{i=1}^{n} R_i Y_i}{W \sum_{i=1}^{n} R_i} + a = \frac{\sum_{i=1}^{n} R_i Y_i}{\sum_{i=1}^{n} R_i} + a$$

which is equivariant.

1.3.2) $$\hat{\mu}_{ipw} = \frac{1}{n} \sum_{i=1}^{n} W_i R_i Y_i$$

$$\hat{\mu}_{ipw}(D+a) = \frac{1}{n} \left[\sum_{i=1}^{n} W_i R_i (Y_i + a) \right]$$

$$= \frac{1}{n} \left[\sum_{i=1}^{n} W_i R_i Y_i + W R_i a \right]$$

$$= \frac{1}{n} \left[\sum_{i=1}^{n} W_i R_i Y_i \right] + \frac{a}{n} \sum_{i=1}^{n} W_i R_i$$

If $W_i = 2 \ \forall i$, $\frac{a}{n} \sum_{i=1}^{n} 2 R_i \neq a$, since $\sum_{i=1}^{n} R_i$ is not necessarily exactly $\frac{n}{2}$

1.3.3) $\hat{\mu}_{ipw-a}(\mathcal{D}) := \hat{\mu}_{ipw}(\mathcal{D}+a) - a$

$$\mathbb{E}\left[\hat{\mu}_{ipw-a}(\mathcal{D})\right] = \mathbb{E}\left[\hat{\mu}_{ipw}\right] + \mathbb{E}\left[\frac{a}{n}\sum_{i=1}^{n} W_i R_i\right] - \mathbb{E}[a]$$

$$= \mathbb{E}[Y] + \frac{a}{n}\sum_{i=1}^{n}\mathbb{E}[W_i R_i] - \mathbb{E}[a]$$

from class: $\mathbb{E}[W_i R_i] = 1$

$$= \mathbb{E}[Y] + \frac{a}{n} \cdot n - a$$

$$= \mathbb{E}[Y]$$

Homework 1: Missing Data and inverse propensity weighting

In [1]:
```
import scipy

import pandas as pd
import numpy as np
from numpy.random import default_rng
import seaborn as sns
import matplotlib.pyplot as plt

from tools import KangSchafSampler, get_estimator_stats
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

#from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
```

Kang & Schafer Simulated Dataset

The sampling distribution provided in `sampler.py` (KangSchafSampler) was based on Kang and Schafer's Demystifying Double Robustness: A Comparison of Alternative Strategies for Estimating a Population Mean from Incomplete Data in Section 1.4. They describe the motivation for the simulated dataset in the discussion at the end:

> "We constructed our simulation to vaguely resemble a quasi-experiment to measure the effect of dieting on body mass index (BMI) in a large sample of high-school students. The study has a pre-post design. Covariates x_i measured at baseline include demographic variables, BMI, self-perceived weight and physical fitness, social acceptance and personality measures. The treatment t_i is dieting (0 = yes, 1 = no) and the outcome y_i is BMI one year later. The goal is to estimate an average causal effect of dieting among those who actually dieted. For that purpose, it suffices to treat the dieters as nonrespondents, set their BMI values to missing, and apply a missing-data method to estimate what the mean BMI for this group would have been had they not dieted."

In terms of the missing data and response bias problem that we've been studying, Kang and Schafer are interested in the situation where we are not given the propensity scores of our observations, but rather we have to estimate the propensity score function from data using a model. They consider two cases: 1) The model we're fitting is "correct", in the sense that the true propensity score function is in the parameterized set of functions of our model. In machine learning terminology, we would just say that the hypothesis space has 0 approximation error. 2) The model we're fitting is "incorrect", in the sense that the true propensity score function is not in our model (i.e. there is nonzero approximation error).

To simulate these two cases, they create a simple data generating distribution where each response y is an affine function of a covariate vector z, plus Gaussian noise. Similarly, the response probability is determined by a logistic regression model on z:

- Covariate: $z = (z_1, z_2, z_3, z_4) \sim N(0, I_{4x4})$
- Response: $y = 210 + 27.4z_1 + 13.7z_2 + 13.7z_3 + 13.7z_4 + \epsilon$ where $\epsilon \sim N(0, 1)$
- Propensity: $\pi = expit(-z_1 + 0.5z_2 - 0.25z_3 - 0.1z_4)$ which is called `obs_prob` in code below
- $r = Bernoull(\pi)$ is the response indicator, which is called `obs` in code below

Using logistic regression on z would be a "correct" model in this case. To simulate an "incorrect" model, they introduce a new covariate $x = (x_1, x_2, x_3, x_4)$, which is a complicated nonlinear transform of z. (See `transform_z` from `KangSchafSampler.get_x` in tools.py for the explicit mapping -- although in "real life" you would not know this transform function.) They then investigate using logistic regression on x to model the propensity score function. This is clearly an "incorrect" model, since logistic regression has a linear dependence on x, while the true model has a nonlinear dependence on x.

In the notebook below, we'll compare using incorrect and correct models in this setting, and we'll also see if we can improve the situation by using modern nonlinear ML models for the propensity function based on x.

For our sampling distribution:

- Overall mean of y = 210
- Mean of observed y = 200
- Mean of missing y = 220
- Response rate = 0.5

Our goal below is to use the observed data to estimate the overall mean of y. We'll evaluate performance using RMSE between our estimates and 210, which we know to be the actual overall mean.

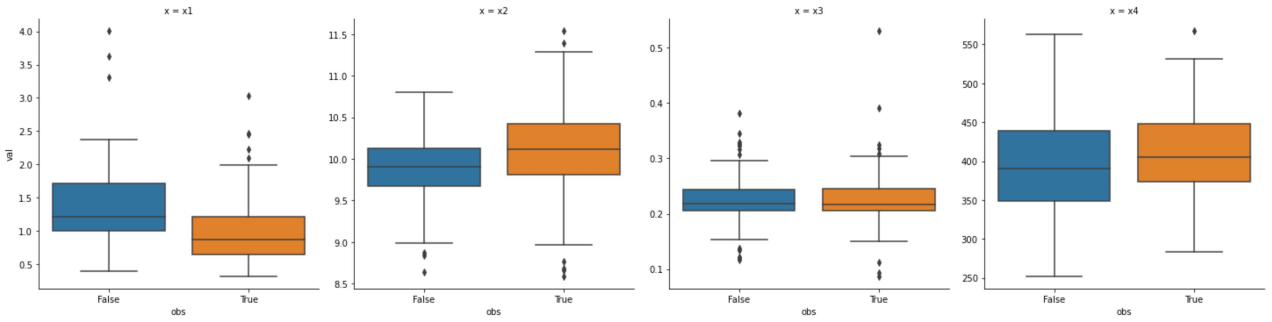
To start our investigation, we'll visualize the distribution of each component of x for the complete and incomplete cases (corresponding to `obs=True` and `obs=False`.

In [2]:
```
sampler = KangSchafSampler(rng=default_rng(27))
s = sampler.sample(n=200)
s.head()
```

Out[2]:

	z1	z2	z3	z4	x1	x2	x3	x4	obs_prob	obs
0	1.254448	0.776902	0.964881	-1.082178	1.872406	10.172419	0.272622	387.882139	0.269138	True
1	0.867017	-0.168161	-0.189661	1.120566	1.542661	9.950245	0.208974	439.003304	0.265858	True
2	-0.436779	-1.279645	-0.469500	-1.196827	0.803812	9.222627	0.224981	307.074015	0.508503	False
3	-1.839857	-0.145557	0.398279	-2.162061	0.398548	9.874394	0.185865	313.020372	0.868037	False
4	0.014477	0.255198	0.173191	-1.270094	1.007265	10.126675	0.216108	360.434196	0.549050	False

In [3]:
```
df_long = pd.melt(s[['x1', 'x2', 'x3', 'x4', 'obs']], "obs", var_name="x", value_name="
g = sns.catplot(x="obs", y="val", col="x", data=df_long, kind="box", sharey=False)
```



From the box plot, we can see from the x_1 plot, in particular, that values are not missing completely at random. This justifies the additional complexity of using estimators for the MAR setting, rather than just a complete-case estimator.

Problem 1

PART A: Create a function that calculates the IPW and self-normalized IPW mean estimators from a sample. Additionally, include a total of the weights associated to *observed* outcomes. We will use this function below.

In [4]:
```
def ipw_estimators(obs_prob, obs, outcome):
    """

    Args:
        obs_prob (pd.Series): probabilities that y_i were observed (i.e. propensities)
        obs (pd.Series): boolean series that tells whether value was observed or not
        outcome (pd.Series): the measurement of the outcome y_i

    Returns:
        dict: {'ipw': ipw, 'sn_ipw': sn_ipw, 'total_weight': total_weight}
    """
    ipw = outcome.dot(obs.divide(obs_prob))/len(outcome)
    sn_ipw = outcome.dot(obs.divide(obs_prob))/( (1/ obs_prob).dot(obs))
    total_weight = (obs.dot(1/obs_prob)).sum()

    return {'ipw': ipw,
            'sn_ipw':sn_ipw,
            'total_weight':total_weight}
```

PART B: Run 1000 trials of the following: 1) Draw a sample `sample_size=200` from the sampler defined above. 2) Get the ipw and sn_ipw estimates of the overall mean using the `ipw_estimators()` function you created above. For this simulation, use the true propensities for each observation, which are given in the `obs_prob` column of the dataframe returned by the sampler. Collect all these results into a dataframe. So that we can use the `get_estimator_stats` function below, each row of the resulting dataframe should correspond to a single trial and each column should correspond to an estimator (either ipw or sn_ipw).

In [5]:
```
ipw_list = []
```

```
sn_ipw_list = []
total_weight_list = []
for ii in range(1000):
    sampler = KangSchafSampler(rng=default_rng())
    s = sampler.sample(n=200)
    result = ipw_estimators(s['obs_prob'],s['obs'],s['y'])
    ipw_list.append(result['ipw'])
    sn_ipw_list.append(result['sn_ipw'])
    total_weight_list.append(result['total_weight'])
results_df = pd.DataFrame(
    {'ipw': ipw_list,
     'sn_ipw': sn_ipw_list,
     'total_weight': total_weight_list
    })
```

In [6]:
```
results_df.head()
```

Out[6]:

	ipw	sn_ipw	total_weight
0	271.301743	213.653230	253.964560
1	209.875618	210.419633	199.482923
2	214.865319	221.113900	194.348089
3	214.558093	207.182840	207.119559
4	233.525540	212.298157	219.997708

PART C: Compute the bias, variance, and RMSE of the IPW and the self-normalized IPW estimators using `get_estimator_stats` from tools.py (already imported). Briefly summarize your findings.

In [7]:
```
stat_df = get_estimator_stats(results_df,true_mean=210)
stat_df
```

Out[7]:

	stat	mean	SD	SE	bias	RMSE
0	ipw	211.083978	24.176362	0.764524	1.083978	24.200651
1	sn_ipw	210.041325	4.941520	0.156265	0.041325	4.941693
2	total_weight	200.752210	20.108181	0.635877	-9.247790	22.132794

Note that the true mean is 210. We can see that the IPW and SN_IPW estimators are both unbiased, and that the SN_IPW has lower variance, which leads to better RMSE results.

PART D: Show that for the ipw estimator, the estimates of SD, bias, and RMSE are compatible in terms of the bias-variance decomposition identity.

In [8]:
```
#RMSE = sqrt(bias^2 + SD^2)
RMSE = ((stat_df['mean'][0]-210)**2 + stat_df['SD'][0]**2)**.5
print(RMSE)
print(stat_df['RMSE'][0])
```

```
24.200650568551897
24.200650568551893
```

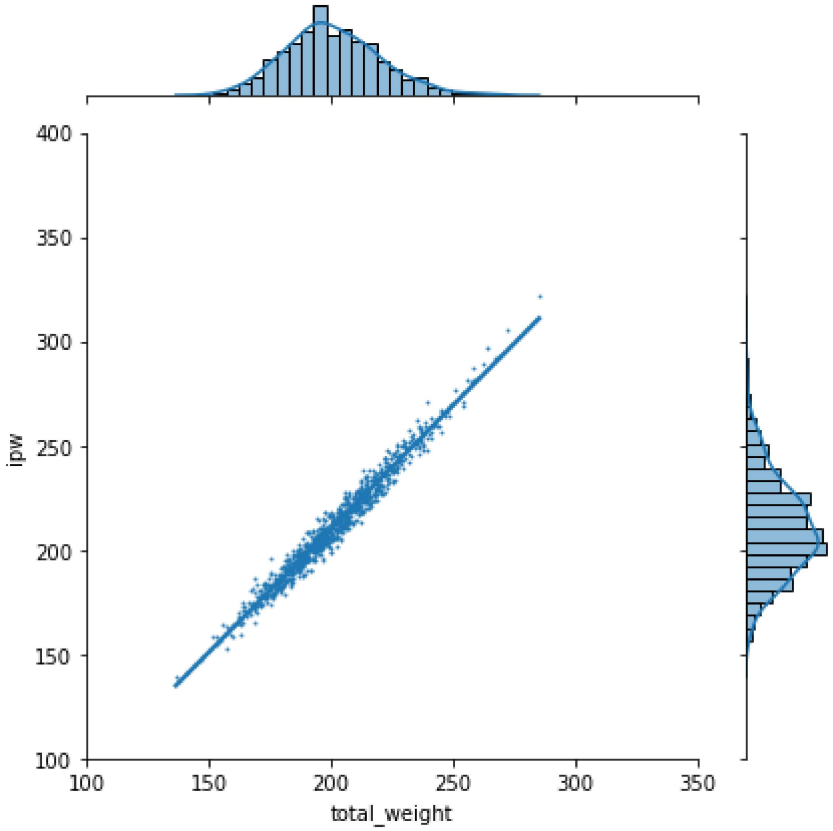
In [9]:
```
max(results_df['ipw'])
```

Out[9]:
```
321.78681805308327
```

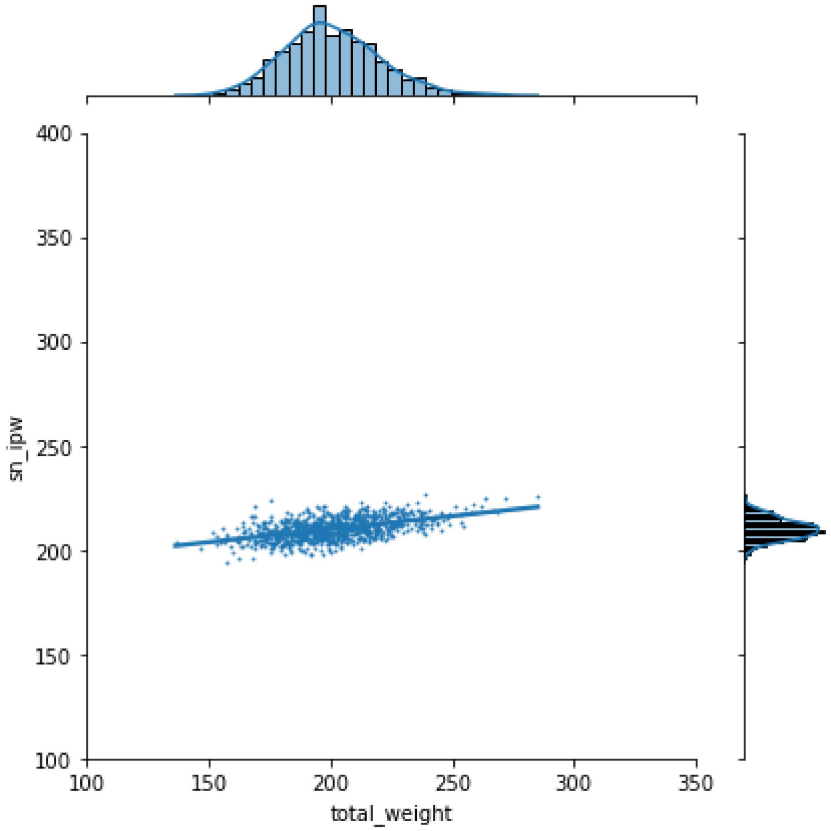
Problem 2

PART A: Graph the scatterplot of the IPW estimate vs total_weight across trials recorded in the results dataframe from Problem 1. *Do the same* for the self-normalized IPW estimate. Report the correlation for each. Briefly describe your findings.

In [10]:
```
plot = sns.jointplot(data=results_df, x='total_weight', y='ipw', kind="reg", x_jitter=0
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plt.tight_layout()
```



In [11]:
```
plot = sns.jointplot(data=results_df, x='total_weight', y='sn_ipw', kind="reg", x_jitte
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plt.tight_layout()
```



In [12]:
```
print(results_df.corr())
```

```
                   ipw    sn_ipw  total_weight
ipw           1.000000  0.652399      0.983997
sn_ipw        0.652399  1.000000      0.508583
total_weight  0.983997  0.508583      1.000000
```

The IPW is STRONGLY correlated with total weight (.98), whereas the SN_IPW is much less correlated (.5) with total weight.

\pagebreak

Problem 3

In the previous problems, we used the true propensities that our dataset generator provided us in the (`obs_prob`) column. In this problem, we'll investigate the impact of estimating the propensity score function from data, both for a "correct" model and an "incorrect" model.

PART A: Following the pattern in Problems 1.B and 1.C above, compare the performance of the IPW estimator using the true propensities to the performance using propensities predicted by a logistic regression model fit to the z_i's. Also compare to a logistic regression model fit to the x_i's. Repeat the comparison for the self-normalized IPW estimator. Summarize your findings. [Hint: Figuring out a good naming convention for these estimators is no joke. We like to use ipw_t for IPW using the true propensities, ipw_lr_x for IPW using a logistic regression estimate of propensity with the x_i covariates. Then sn_ipw_rf_z when we're using self-normalized IPW with a random forest propensity model based on covariate z.]

In [13]:
```
#@ignore_warnings(category=ConvergenceWarning)
```

```
def total_predicted_weight(s):
#Create X and y for Logisitic Regression DF
    #print('model_for_z')
    obs = s['obs']
    covariates=s[['z1','z2','z3','z4']]
    model = LogisticRegression(fit_intercept=True,max_iter=1000)
    model.fit(X=covariates,y=obs)
    true_index = np.where(model.classes_ == True)[0][0]
    pred_prob_z = model.predict_proba(X=covariates)[:,true_index]

    #print('model_for_x')

    obs = s['obs']
    covariates=s[['x1','x2','x3','x4']]
    model = LogisticRegression(fit_intercept=True,max_iter=1000)

    model.fit(X=covariates,y=obs)
    true_index = np.where(model.classes_ == True)[0][0]
    pred_prob_x = model.predict_proba(X=covariates)[:,true_index]

    return pred_prob_z, pred_prob_x
```

In [46]:

```
ipw_t_list = []
sn_ipw_t_list = []
total_weight_t_list = []
ipw_z_list = []
sn_ipw_z_list = []
total_weight_z_list = []
ipw_x_list = []
sn_ipw_x_list = []
total_weight_x_list = []
for ii in range(1000):
    #print(ii)
    sampler = KangSchafSampler(rng=default_rng())
    s = sampler.sample(n=200)

    pred_prob_z, pred_prob_x = total_predicted_weight(s)
    # True Weight
    result_t = ipw_estimators(s['obs_prob'],s['obs'],s['y'])
    ipw_t_list.append(result_t['ipw'])
    sn_ipw_t_list.append(result_t['sn_ipw'])
    total_weight_t_list.append(result_t['total_weight'])

    # Weight estimated from Z
    result_z = ipw_estimators(pd.Series(pred_prob_z),s['obs'],s['y'])
    ipw_z_list.append(result_z['ipw'])
    sn_ipw_z_list.append(result_z['sn_ipw'])
    total_weight_z_list.append(result_z['total_weight'])

    #Weight estimated from X
    result_x = ipw_estimators(pd.Series(pred_prob_x),s['obs'],s['y'])
    ipw_x_list.append(result_x['ipw'])
    sn_ipw_x_list.append(result_x['sn_ipw'])
    total_weight_x_list.append(result_x['total_weight'])



results_df = pd.DataFrame(
    {'ipw_t': ipw_t_list,
```

```
    'sn_ipw_t': sn_ipw_t_list,
    'total_weight_t': total_weight_t_list,
    'ipw_z': ipw_z_list,
    'sn_ipw_z': sn_ipw_z_list,
    'total_weight_z': total_weight_z_list,
    'ipw_x': ipw_x_list,
    'sn_ipw_x': sn_ipw_x_list,
    'total_weight_x': total_weight_x_list,
   })
```

In [47]:
```
results_df.head()
```

Out[47]:

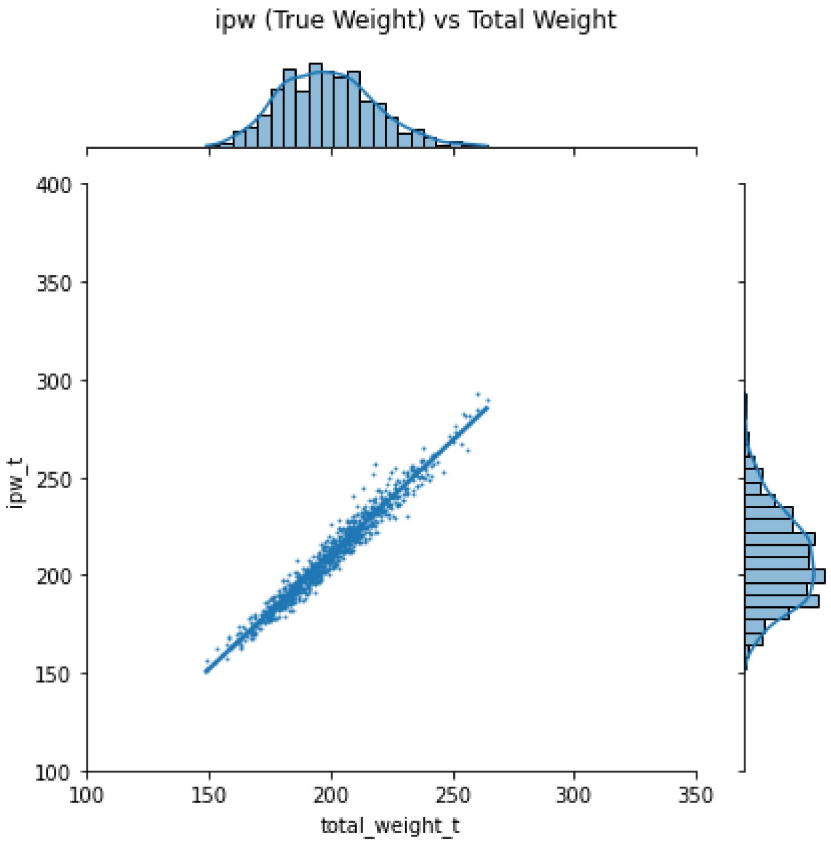
	ipw_t	sn_ipw_t	total_weight_t	ipw_z	sn_ipw_z	total_weight_z	ipw_x	sn_ipw_x
0	224.744809	213.500343	210.533441	208.699339	208.925548	199.783454	213.176367	209.92336(
1	235.188435	206.822032	227.430737	211.671096	204.372858	207.142081	211.111397	202.71504(
2	198.404857	208.561883	190.259940	190.460359	208.407592	182.776795	204.603848	212.43683(
3	223.510836	208.112801	214.797778	208.315937	207.764620	200.530713	210.963403	206.63049(
4	233.405693	211.086689	221.146767	208.255118	208.244618	200.010084	197.677125	205.40335(

In [48]:
```
#plot = sns.jointplot(data=results_df, x='total_weight_t', y='total_weight_z', kind="re
#plot.ax_marg_y.set_ylim(100,400)
#plot.ax_marg_x.set_xlim(100,350)
#plt.tight_layout()
```

In [49]:
```
#plot = sns.jointplot(data=results_df, x='total_weight_t', y='total_weight_x', kind="re
#plot.ax_marg_y.set_ylim(100,400)
#plot.ax_marg_x.set_xlim(100,350)
#plt.tight_layout()
```

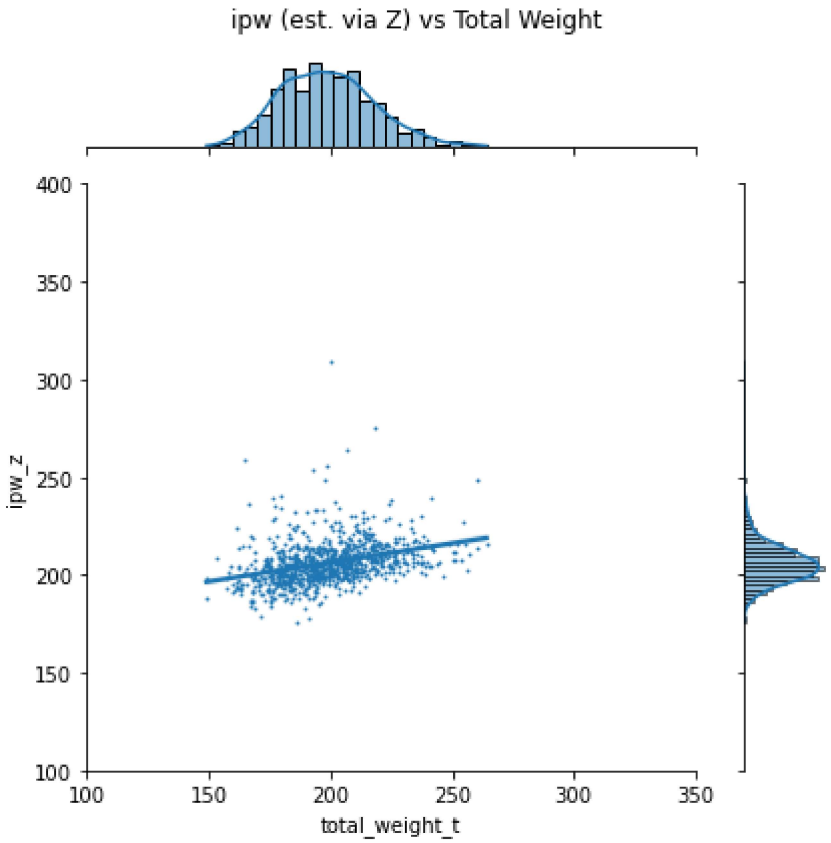
IPW (Not Normalized)

In [50]:
```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='ipw_t', kind="reg", x_jitt
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('ipw (True Weight) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



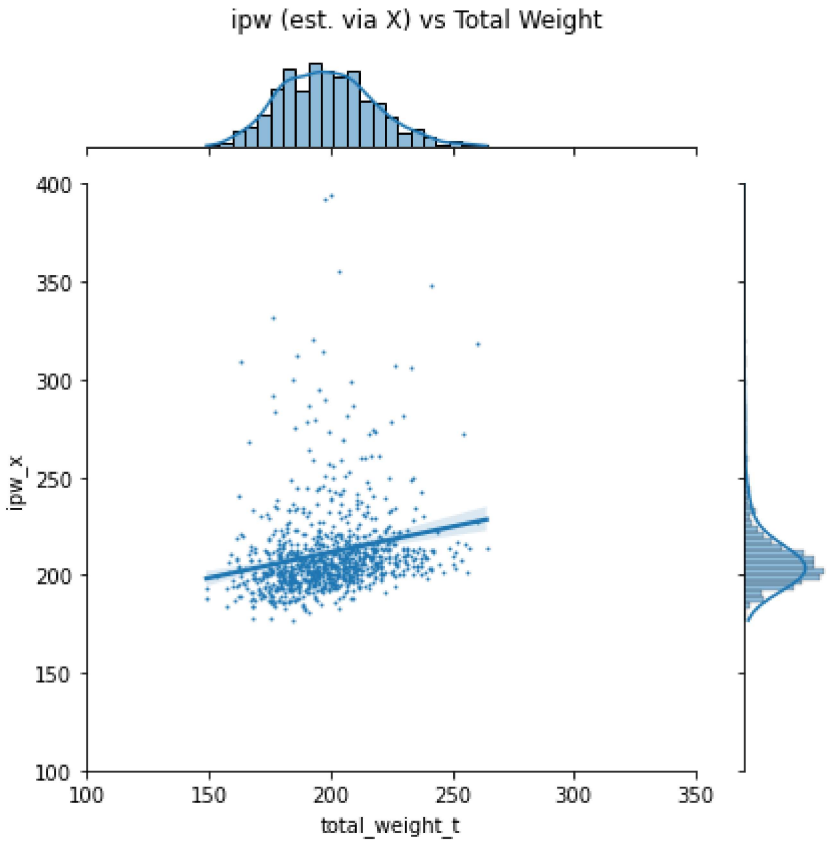
In [51]:

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='ipw_z', kind="reg", x_jitt
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('ipw (est. via Z) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [52]:

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='ipw_x', kind="reg", x_jitt
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('ipw (est. via X) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```

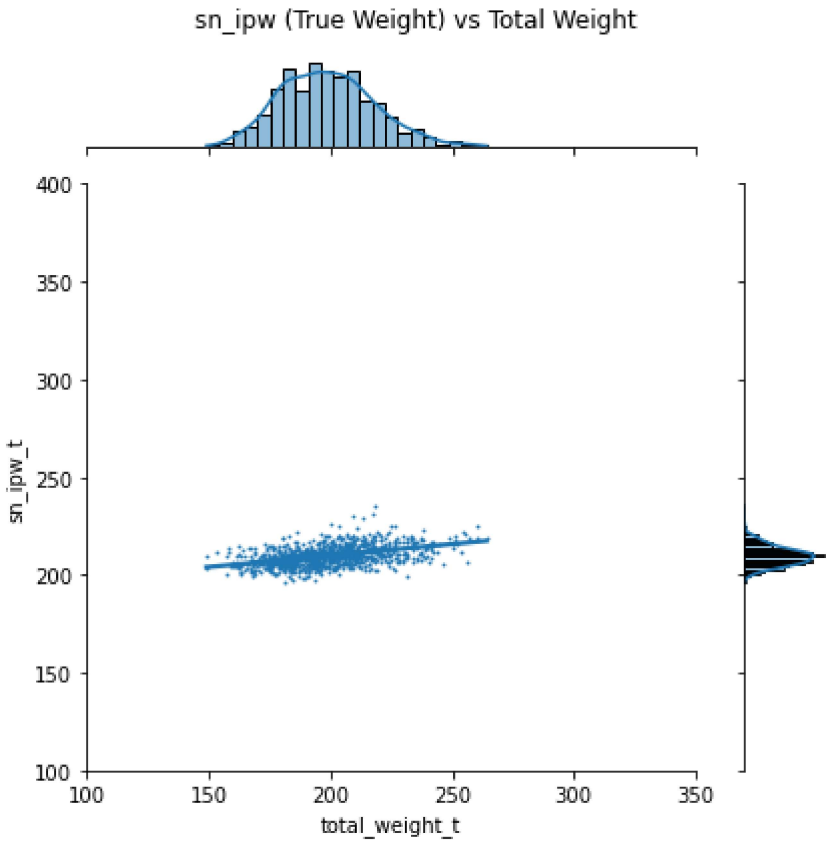


In [53]:
```
print(results_df[['total_weight_t','ipw_t','ipw_z','ipw_x']].corr())
```

```
                total_weight_t     ipw_t     ipw_z     ipw_x
total_weight_t        1.000000  0.980328  0.345903  0.166029
ipw_t                 0.980328  1.000000  0.427954  0.214852
ipw_z                 0.345903  0.427954  1.000000  0.527201
ipw_x                 0.166029  0.214852  0.527201  1.000000
```

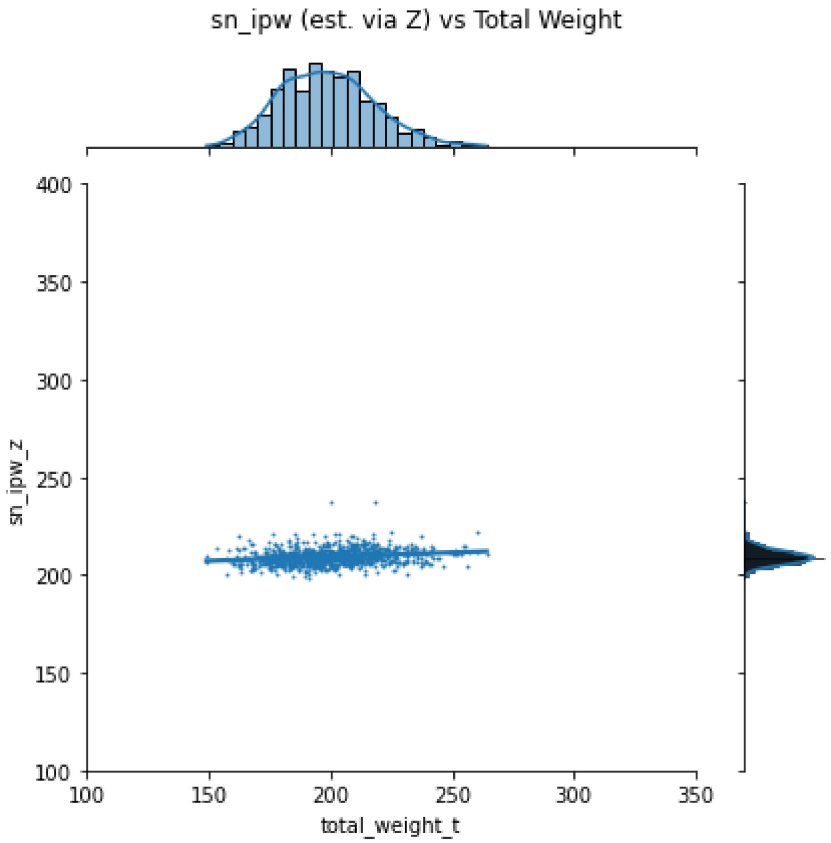
Self Normalized IPW

In [54]:
```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='sn_ipw_t', kind="reg", x_j
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('sn_ipw (True Weight) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



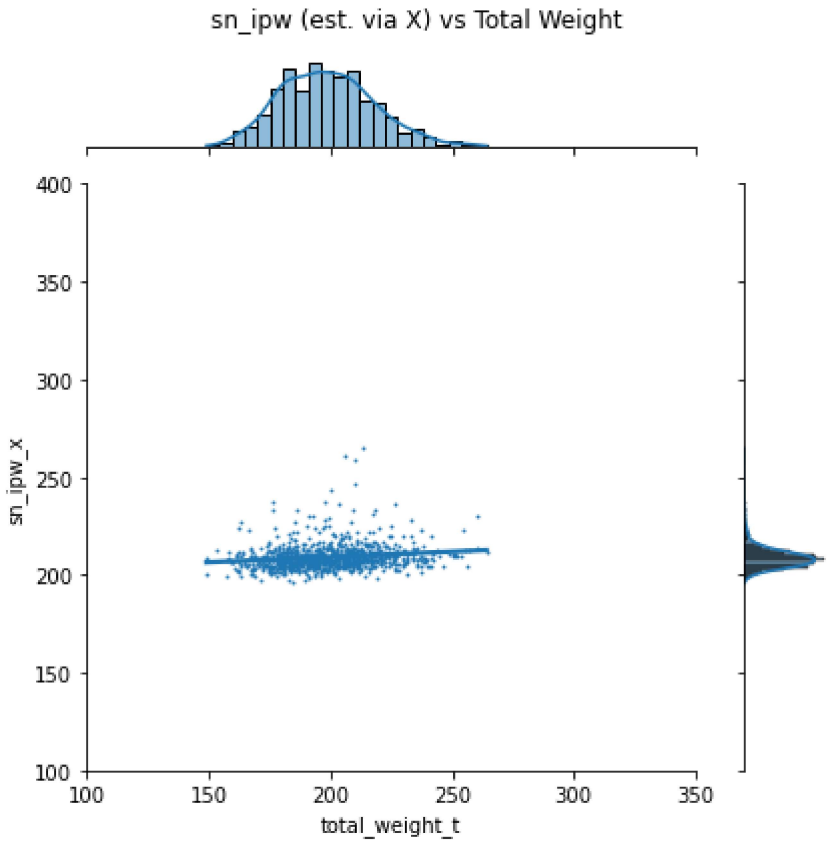
In [55]:

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='sn_ipw_z', kind="reg", x_j
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('sn_ipw (est. via Z) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [56]:

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='sn_ipw_x', kind="reg", x_j
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('sn_ipw (est. via X) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [ ]:

In [57]:
```
stat_df = get_estimator_stats(results_df[['ipw_t','ipw_z','ipw_x','sn_ipw_t','sn_ipw_z'
stat_df
```

Out[57]:

	stat	mean	SD	SE	bias	RMSE
0	ipw_t	208.849950	23.072752	0.729624	-1.150050	23.101396
1	ipw_z	206.186814	10.897674	0.344615	-3.813186	11.545548
2	ipw_x	211.270731	30.479905	0.963859	1.270731	30.506382
3	sn_ipw_t	209.801494	5.016435	0.158634	-0.198506	5.020361
4	sn_ipw_z	209.273570	3.868888	0.122345	-0.726430	3.936495
5	sn_ipw_x	209.131067	6.423865	0.203140	-0.868933	6.482367

We can see that in terms of RMSE, IPW and SN_IPW estimators that fit to Z have better fits than LR models fit to X. Surprisingly, the SN_IPW estimators fit to Z has a better RMSE than SN_IPW fit to the true weights.

In [58]:
```
results_df[['total_weight_t','ipw_t','ipw_z','ipw_x','sn_ipw_t','sn_ipw_z','sn_ipw_x',]
```

Out[58]:

	total_weight_t	ipw_t	ipw_z	ipw_x	sn_ipw_t	sn_ipw_z	sn_ipw_x
total_weight_t	1.000000	0.980328	0.345903	0.166029	0.450225	0.213916	0.164661

	total_weight_t	ipw_t	ipw_z	ipw_x	sn_ipw_t	sn_ipw_z	sn_ipw_x
ipw_t	0.980328	1.000000	0.427954	0.214852	0.616529	0.355546	0.267541
ipw_z	0.345903	0.427954	1.000000	0.527201	0.562621	0.773710	0.569474
ipw_x	0.166029	0.214852	0.527201	1.000000	0.309370	0.454324	0.875602
sn_ipw_t	0.450225	0.616529	0.562621	0.309370	1.000000	0.761350	0.556236
sn_ipw_z	0.213916	0.355546	0.773710	0.454324	0.761350	1.000000	0.708764
sn_ipw_x	0.164661	0.267541	0.569474	0.875602	0.556236	0.708764	1.000000

The correlations are also about what we expect - a higher correlation with the true weights, and lower correlation with fits with z and even lower with fits to x.

PART B: In this part we're going to try to get better performance using only the x_i's for our propensity model. Try using a nonlinear model (e.g. random forest, gradient boosted trees, etc...) to fit the propensity score function. Compare results using IPW and SN_IPW with both logistic regression and your nonlinear model for the propensity score function. Summarize your findings.

In [183…

```
#@ignore_warnings(category=ConvergenceWarning)
def total_predicted_weight_x_only(s):
    #Create X and y for sklearn models
    # print('model_for_RF')
    obs = s['obs']
    covariates=s[['x1','x2','x3','x4']]
    model = RandomForestClassifier(n_estimators=100, min_samples_leaf=15)
    model.fit(X=covariates,y=obs)
    true_index = np.where(model.classes_ == True)[0][0]
    pred_prob_x_rf = model.predict_proba(X=covariates)[:,true_index]

    # print('model_for_GB')

    obs = s['obs']
    covariates=s[['x1','x2','x3','x4']]
    model = GradientBoostingClassifier(n_estimators=10)
    model.fit(X=covariates,y=obs)
    true_index = np.where(model.classes_ == True)[0][0]
    pred_prob_x_gb = model.predict_proba(X=covariates)[:,true_index]

    return pred_prob_x_rf, pred_prob_x_gb
```

In [184…

```
ipw_t_list = []
sn_ipw_t_list = []
total_weight_t_list = []

ipw_x_rf_list = []
sn_ipw_x_rf_list = []
total_weight_x_rf_list = []

ipw_x_gb_list = []
sn_ipw_x_gb_list = []
total_weight_x_gb_list = []

for ii in range(1000):
    #Sample
```

```
    sampler = KangSchafSampler(rng=default_rng())
    s = sampler.sample(n=200)

    # Predict the propensity_scores
    pred_prob_x_rf, pred_prob_x_gb = total_predicted_weight_x_only(s)

    # True Weight
    result = ipw_estimators(s['obs_prob'],s['obs'],s['y'])
    ipw_t_list.append(result['ipw'])
    sn_ipw_t_list.append(result['sn_ipw'])
    total_weight_t_list.append(result['total_weight'])


    # Weight estimated from X (Random Forest)
    result_rf = ipw_estimators(pd.Series(pred_prob_x_rf),s['obs'],s['y'])
    ipw_x_rf_list.append(result_rf['ipw'])
    sn_ipw_x_rf_list.append(result_rf['sn_ipw'])
    total_weight_x_rf_list.append(result_rf['total_weight'])
    #print(pred_prop_x_rf)

    # Weight estimated from X (Gradient Boosting)
    result_gb = ipw_estimators(pd.Series(pred_prob_x_gb),s['obs'],s['y'])
    ipw_x_gb_list.append(result_gb['ipw'])
    sn_ipw_x_gb_list.append(result_gb['sn_ipw'])
    total_weight_x_gb_list.append(result_gb['total_weight'])
    #print(pred_prop_x_gb)



results_df = pd.DataFrame(
    {'ipw_t': ipw_t_list,
     'sn_ipw_t': sn_ipw_t_list,
     'total_weight_t': total_weight_t_list,
     'ipw_x_rf': ipw_x_rf_list,
     'sn_ipw_x_rf': sn_ipw_x_rf_list,
     'total_weight_x_rf': total_weight_x_rf_list,
     'ipw_x_gb': ipw_x_gb_list,
     'sn_ipw_x_gb': sn_ipw_x_gb_list,
     'total_weight_x_gb': total_weight_x_gb_list,
    })
```

In [200…

```
#results_df.head()
```

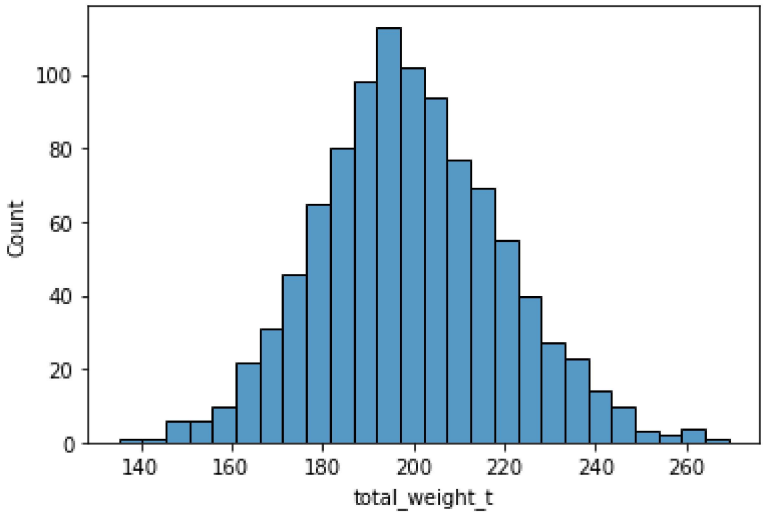
Out[200…

	ipw_t	sn_ipw_t	total_weight_t	ipw_x_rf	sn_ipw_x_rf	total_weight_x_rf	ipw_x_gb	sn_ipw
0	214.980387	202.955754	211.849512	183.465102	201.094889	182.466201	173.667664	200.1
1	170.357105	208.069072	163.750531	168.711536	203.408282	165.884628	164.316993	202.8
2	167.222255	206.483481	161.971558	173.864669	204.509865	170.030594	166.217342	204.5
3	232.967146	216.666524	215.046738	180.854048	205.055081	176.395578	173.363389	203.8
4	230.164666	210.560117	218.621332	179.710496	202.780455	177.246368	170.420614	201.5

Histograms of total weight

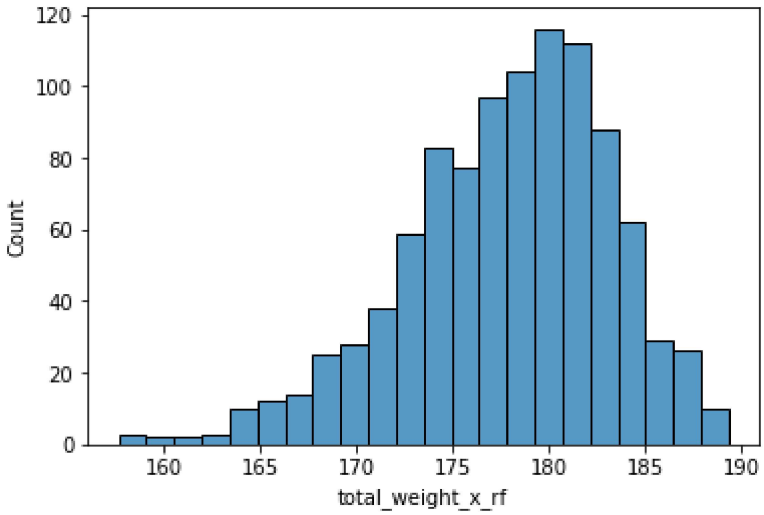
In [186…
```
sns.histplot(data=results_df['total_weight_t'])
```

Out[186… <AxesSubplot:xlabel='total_weight_t', ylabel='Count'>



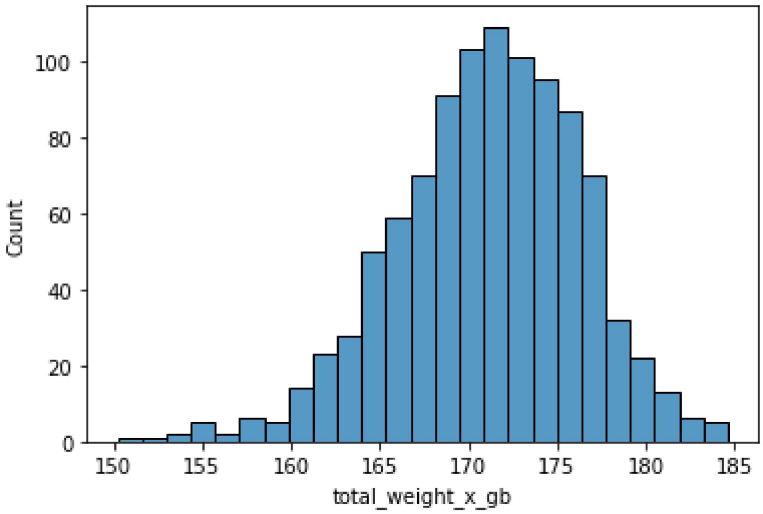
In [187…
```
sns.histplot(data=results_df['total_weight_x_rf'])
```

Out[187… <AxesSubplot:xlabel='total_weight_x_rf', ylabel='Count'>



In [188…
```
sns.histplot(data=results_df['total_weight_x_gb'])
```

Out[188… <AxesSubplot:xlabel='total_weight_x_gb', ylabel='Count'>

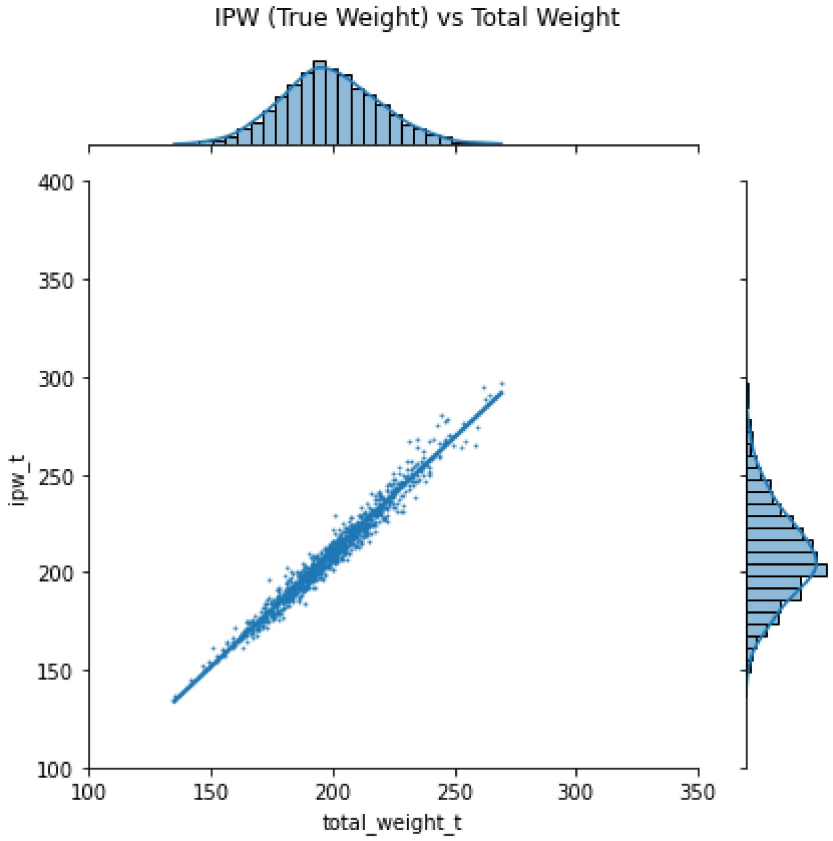


We plot these graphs to show that Random Forest and Gradient Boosted Trees seem to underpredict the total weight, which factors into the IPW/SN_IPW predictions later

IPW (not normalized)

In [189…

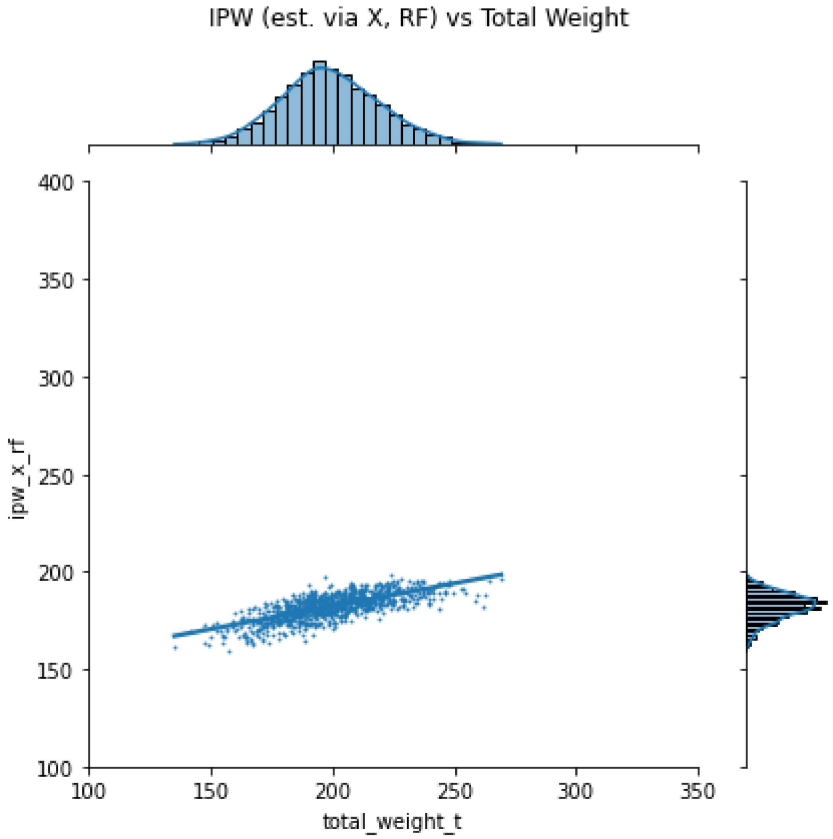
```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='ipw_t', kind="reg", x_jitt
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('IPW (True Weight) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [190…

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='ipw_x_rf', kind="reg", x_j
```

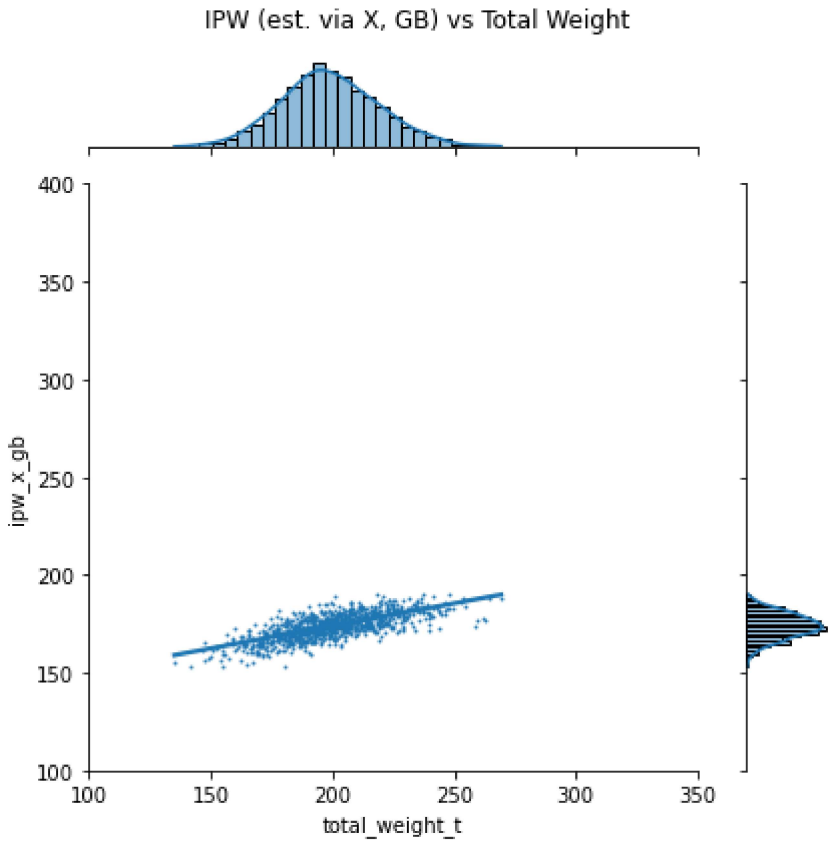
```
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('IPW (est. via X, RF) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [191…

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='ipw_x_gb', kind="reg", x_j
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('IPW (est. via X, GB) vs Total Weight')

plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [192…

Out[192…

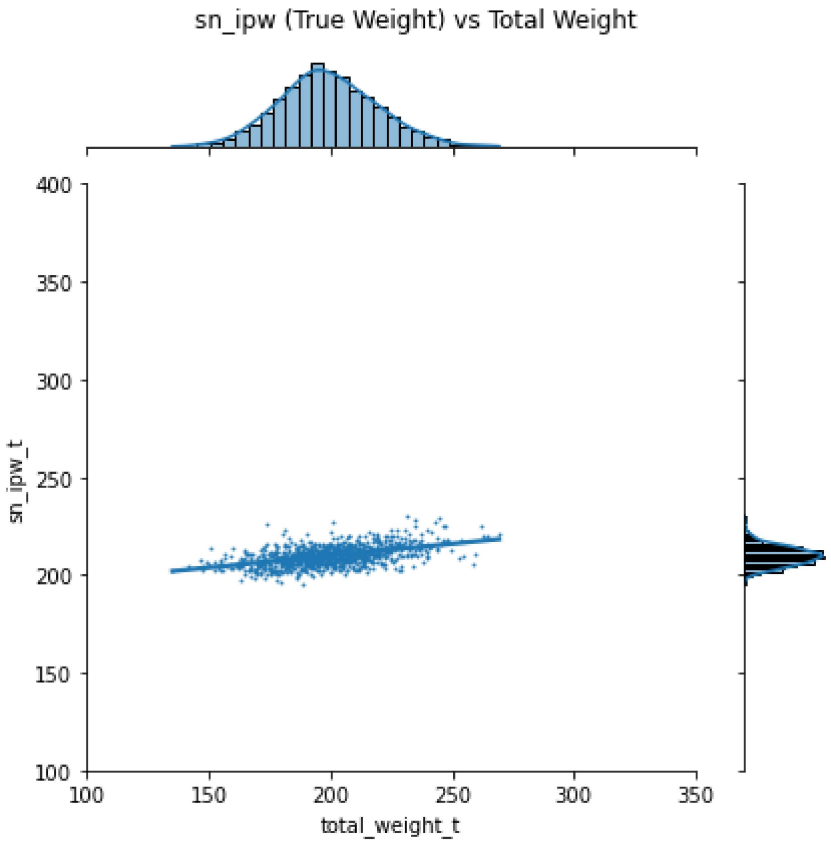
	stat	mean	SD	SE	bias	RMSE
0	ipw_t	209.210132	24.310610	0.768769	-0.789868	24.323438
1	ipw_x_rf	182.026869	6.596305	0.208593	-27.973131	28.740343
2	ipw_x_gb	173.795487	6.493729	0.205350	-36.204513	36.782268

In [ ]:

Self Normalized IPW

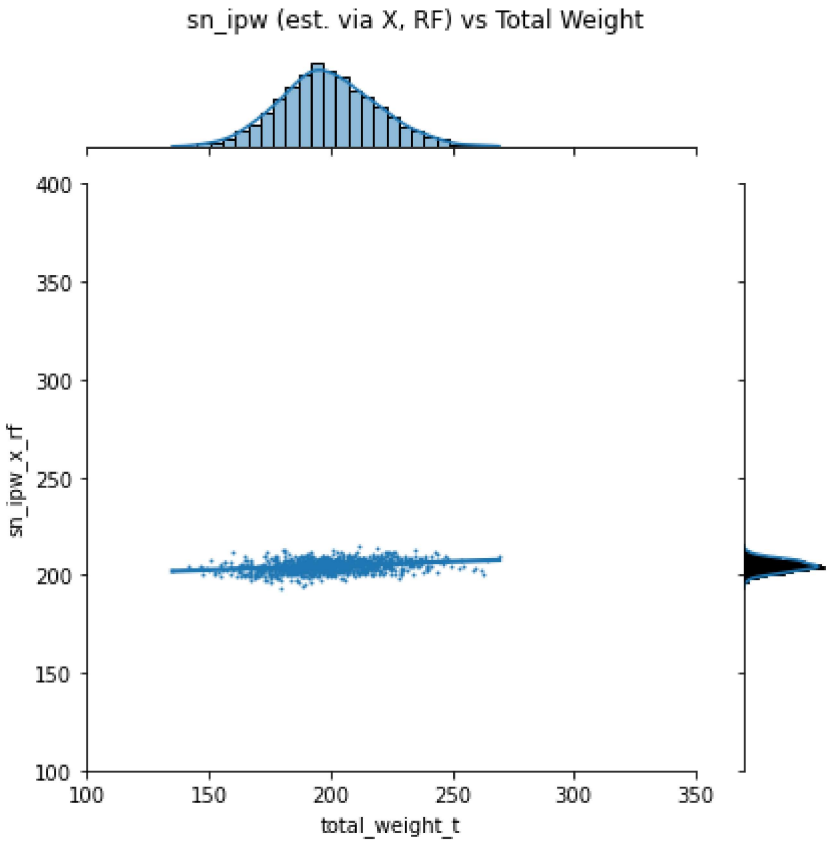
In [193…

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='sn_ipw_t', kind="reg", x_j
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('sn_ipw (True Weight) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



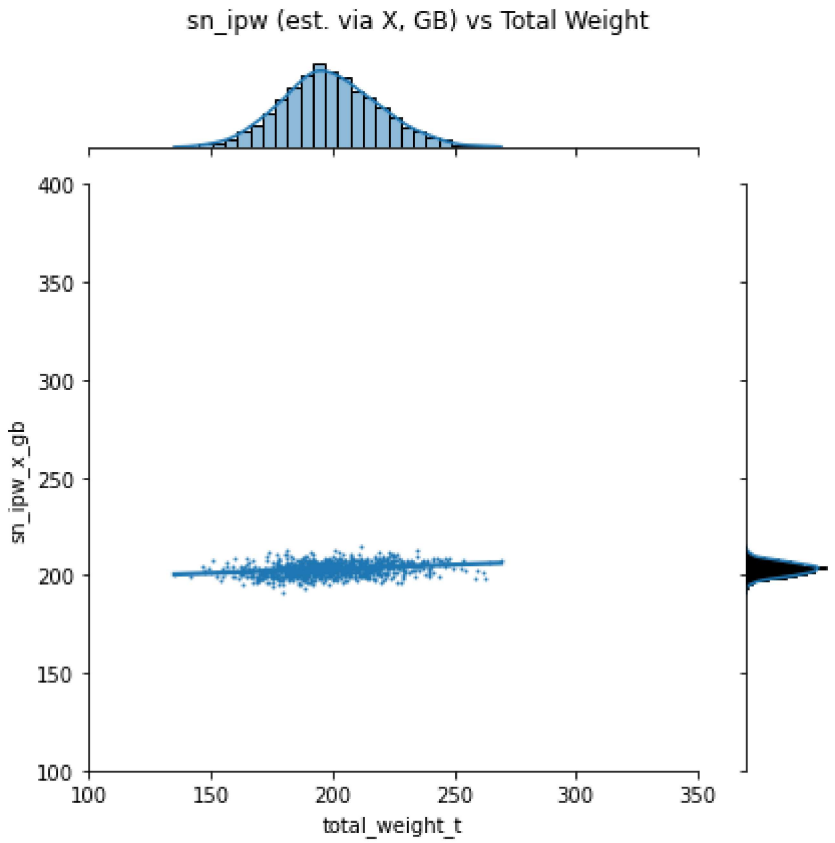
In [194…

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='sn_ipw_x_rf', kind="reg",
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('sn_ipw (est. via X, RF) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [195…

```
plot = sns.jointplot(data=results_df, x='total_weight_t', y='sn_ipw_x_gb', kind="reg",
plot.ax_marg_y.set_ylim(100,400)
plot.ax_marg_x.set_xlim(100,350)
plot.fig.suptitle('sn_ipw (est. via X, GB) vs Total Weight')
plot.fig.subplots_adjust(top=0.95)
plt.tight_layout()
```



In [207…
```
stats_df = get_estimator_stats(results_df[['ipw_t','ipw_x_rf','ipw_x_gb','sn_ipw_t','sn
stats_df
```

Out[207…

	stat	mean	SD	SE	bias	RMSE
0	ipw_t	209.210132	24.310610	0.768769	-0.789868	24.323438
1	ipw_x_rf	182.026869	6.596305	0.208593	-27.973131	28.740343
2	ipw_x_gb	173.795487	6.493729	0.205350	-36.204513	36.782268
3	sn_ipw_t	209.763565	4.984601	0.157627	-0.236435	4.990206
4	sn_ipw_x_rf	204.706058	3.077984	0.097334	-5.293942	6.123709
5	sn_ipw_x_gb	203.181064	3.275160	0.103570	-6.818936	7.564692

In [ ]:
```

```

In [197…
```
results_df[['total_weight_t','ipw_t','ipw_x_rf','ipw_x_gb','sn_ipw_t','sn_ipw_x_rf','sn
```

Out[197…

	total_weight_t	ipw_t	ipw_x_rf	ipw_x_gb	sn_ipw_t	sn_ipw_x_rf	sn_ipw_x_gb
total_weight_t	1.000000	0.983251	0.723394	0.723147	0.497058	0.280990	0.267441
ipw_t	0.983251	1.000000	0.750303	0.755843	0.645457	0.402017	0.394154
ipw_x_rf	0.723394	0.750303	1.000000	0.877939	0.568083	0.576777	0.547742
ipw_x_gb	0.723147	0.755843	0.877939	1.000000	0.592931	0.556321	0.613670

	total_weight_t	ipw_t	ipw_x_rf	ipw_x_gb	sn_ipw_t	sn_ipw_x_rf	sn_ipw_x_gb
sn_ipw_t	0.497058	0.645457	0.568083	0.592931	1.000000	0.762302	0.780597
sn_ipw_x_rf	0.280990	0.402017	0.576777	0.556321	0.762302	1.000000	0.941697
sn_ipw_x_gb	0.267441	0.394154	0.547742	0.613670	0.780597	0.941697	1.000000

We mentioned above that our random forest and gradient boosted tree models had difficulty predicting the right range of total weights, which leads to some bias in the results (We can see that for our IPW and SN_IPW estimators, they typically underestimate the mean.) The SN_IPW estimators has less bias than the IPW estimators. The IPW and SN_IPW estimators from the random forest model are a bit better, but its most likely largely due to the possiblity that the random forest model was just fit better, as we did no hyperparameter tuning.

For our Logistic regression fits, we were getting an RMSE of about 10 (IPW) and 4 (SN_IPW). Our random forest gets an RMSE of about 28 (IPW) and 6 (SN_IPW). Overall, I think the large difference in RMSE for IPW is caused by the two factors of the weight being under predicted, and the fact that IPW is correlated with weight.

For our correlations, we see the similar trend of the IPW estimators being more correlated with the total weight.

In [ ]:

In [ ]: