Results

Harpeth Lee

4/2/2022

Results

Data Simulation

Our data simulations are done based on the latent variable model that PLS methods assume. We generate $\mathbf{X}$ and $\mathbf{Y}$ using the following equations, $\mathbf{X} = \boldsymbol{\phi}\mathbf{A} + \boldsymbol{\epsilon}_X$ and $\mathbf{Y} = \boldsymbol{\phi}\mathbf{D} + \boldsymbol{\epsilon}_Y$. The $\mathbf{A}$ and $\mathbf{D}$ matrices are fixed, providing most of the structure seen in $\mathbf{X}$ and $\mathbf{Y}$. $\boldsymbol{\phi}$, $\boldsymbol{\epsilon}_X$, and $\boldsymbol{\epsilon}_Y$ are randomly generated. The method by which $\boldsymbol{\phi}$ is generated is less important as we want to approximate $\boldsymbol{\phi}$ with our final model. $\boldsymbol{\epsilon}_X$, and $\boldsymbol{\epsilon}_Y$ are sampled from a normal distribution.

Although $\boldsymbol{\phi}$ is generated using a similar method as $\boldsymbol{\epsilon}_X$, and $\boldsymbol{\epsilon}_Y$, it is important to note that $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$ are noise while $\boldsymbol{\phi}$ is not. Ideally, our models will be able to identify $\boldsymbol{\phi}$ while ignoring $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$. $\boldsymbol{\phi}$ is generated sampling from a multivariate normal distribution with uncorrelated samples(using the `mvrnorm` function in the `MASS` package in order to make sure columns are uncorrelated). We want to make sure columns of $\boldsymbol{\phi}$ are uncorrelated in order to ensure that $\mathbf{A}$ and $\mathbf{D}$ have the structure that we intend. If columns of $\boldsymbol{\phi}$ exhibited varying amounts of correlation, we would not be able to know the number of components that the model should build. $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$ are generated by samples from random distributions (using `rnorm`) as this will cause some random correlation in the noise. This random correlation is included as we want to test model's ability to distinguish true correlation from noise that appears correlated.

When generating the data, there are four variables related to the dimensionality of the matrices which must be chosen, n, p, q, and R. n is the number of observations generated, p is the number of predictors, and q is the number of response variables. Of these variables, R is the only one of these variables that is not clear from the generated data. R is the size of the dimension of $\mathbf{A}$,$\mathbf{D}$, and $\boldsymbol{\phi}$ that is lost when matrix multiplication is performed. Since we want $\mathbf{X}$ to be of dimension $n \times p$ and $\mathbf{Y}$ to be of dimension $n \times q$, $\boldsymbol{\phi}$ will be of dimension $n \times R$, $\mathbf{A}$ will be of dimension $R \times p$, and $\mathbf{D}$ will be of dimension $R \times q$. Although it may seem like we will want our final model to have R total components, we will want K to be the number of eigenvectors of the variance matrix of $\mathbf{X}$ that have non-null projections onto the covariance matrix of $\mathbf{X}$ and $\mathbf{Y}$. [Helland and Almoy]

There were a number of methods used to generate these matrices. $\boldsymbol{\phi}$ was always generated from uncorrelated normal distributions. Two methods were used to generate $\mathbf{A}$, the "simple" method and the "complex" method. In the simple method,

$$\mathbf{A} = \begin{pmatrix} \mathbf{1}_{\frac{3}{5}R \times 5} & \mathbf{0}_{\frac{3}{5}R \times 10} & \mathbf{0}_{\frac{3}{5}R \times (p-15)} \\ \mathbf{0}_{\frac{2}{5}R \times 5} & \mathbf{1}_{\frac{2}{5}R \times 10} & \mathbf{0}_{\frac{2}{5}R \times (p-15)} \end{pmatrix}$$

Note that $p \geq 15$ in order for this method to work. The dimensions of $\frac{3}{5}R$ and $\frac{2}{5}R$ are not exact, we round and then take the difference in order to make sure values are integers that add up to R.

Using the "complex" method,

$$\mathbf{A} = \begin{pmatrix} \mathbf{1}_{3\times 5} & \mathbf{0}_{3\times 10} & \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times (p-23)} \\ \mathbf{0}_{2\times 5} & \mathbf{1}_{2\times 10} & \mathbf{0}_{2\times 5} & \mathbf{0}_{2\times 1} & \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times (p-23)} \\ \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 10} & \mathbf{1}_{3\times 5} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 2} & \mathbf{0}_{3\times (p-23)} \\ \mathbf{0}_{2\times 5} & \mathbf{0}_{2\times 10} & \mathbf{0}_{2\times 5} & \mathbf{1}_{2\times 1} & \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times (p-23)} \\ \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 10} & \mathbf{0}_{3\times 5} & \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 2} & \mathbf{0}_{3\times (p-23)} \end{pmatrix}$$

Note that $p \geq 23$ in order for this method to work. Furthermore, $R = 13$ whenever this method is used.

A wider number of methods were used to generate the $\mathbf{D}$ matrix. We will call these methods "basic", "diagonal", "simple", "2-block", and "4-block". In the basic method,

$$\mathbf{D} = \mathbf{1}_{R\times q}$$

Using the "diagonal" method,

$$\mathbf{D} = \mathbf{I}_{R\times q}$$

Using the "simple" method

$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{R\times 1} & \mathbf{0}_{R\times (q-1)} \end{pmatrix}$$

Using the "2-block" method

$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{\frac{3}{5}R\times \frac{1}{4}q} & \mathbf{0}_{\frac{3}{5}R\times \frac{1}{4}q} & \mathbf{0}_{\frac{3}{5}\times \frac{1}{2}q} \\ \mathbf{0}_{\frac{2}{5}R\times \frac{1}{4}q} & \mathbf{1}_{\frac{2}{5}R\times \frac{1}{4}q} & \mathbf{0}_{\frac{2}{5}\times \frac{1}{2}q} \end{pmatrix}$$

Note the two blocks of 1s hence the name "2-block" method.

Using the "4-block" method

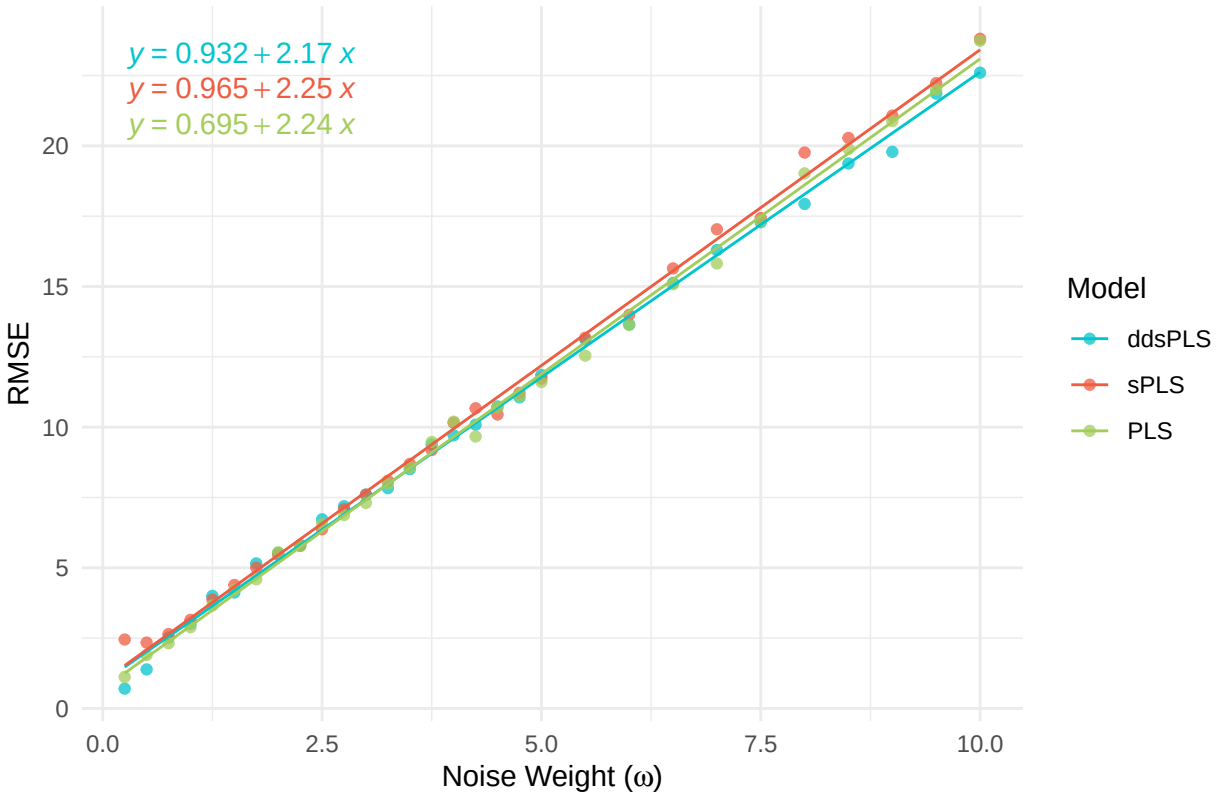
$$\mathbf{D} = \begin{pmatrix} \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{0}_{3\times 1} & \mathbf{1}_{3\times 1} & \mathbf{0}_{3\times (q-4)} \\ \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times 1} & \mathbf{0}_{1\times (q-4)} \end{pmatrix}$$

Note that $R = 13$ at all times when the "4-block" method is used.

The $\boldsymbol{\epsilon}_X$ and $\boldsymbol{\epsilon}_Y$ matrices are generated from samples from a standard normal distribution and then scaled by a factor ω, called the noise weight. $\boldsymbol{\epsilon}_X$ is of dimension $n \times p$ and $\boldsymbol{\epsilon}_Y$ is of dimension $n \times q$. Since the noise is randomly generated we expect to see some correlation occur both between predictors and between the predictors and the responses.

Noise

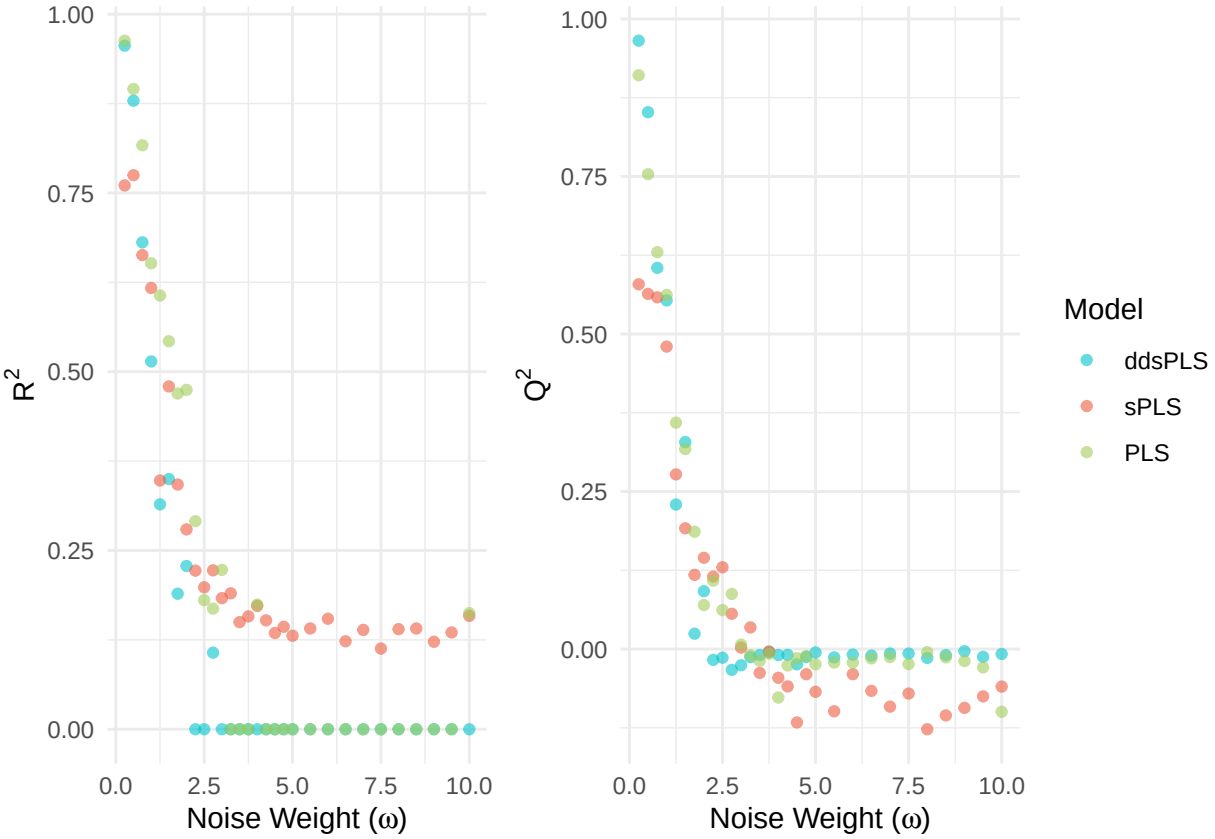
When examining model performance, it is important to consider how it performs under conditions where the amount of noise varies. The following graphs show the relationship between ω and the RMSE for the ddsPLS, sPLS, and PLS methods. Note the extremely linear relation displayed in all three.



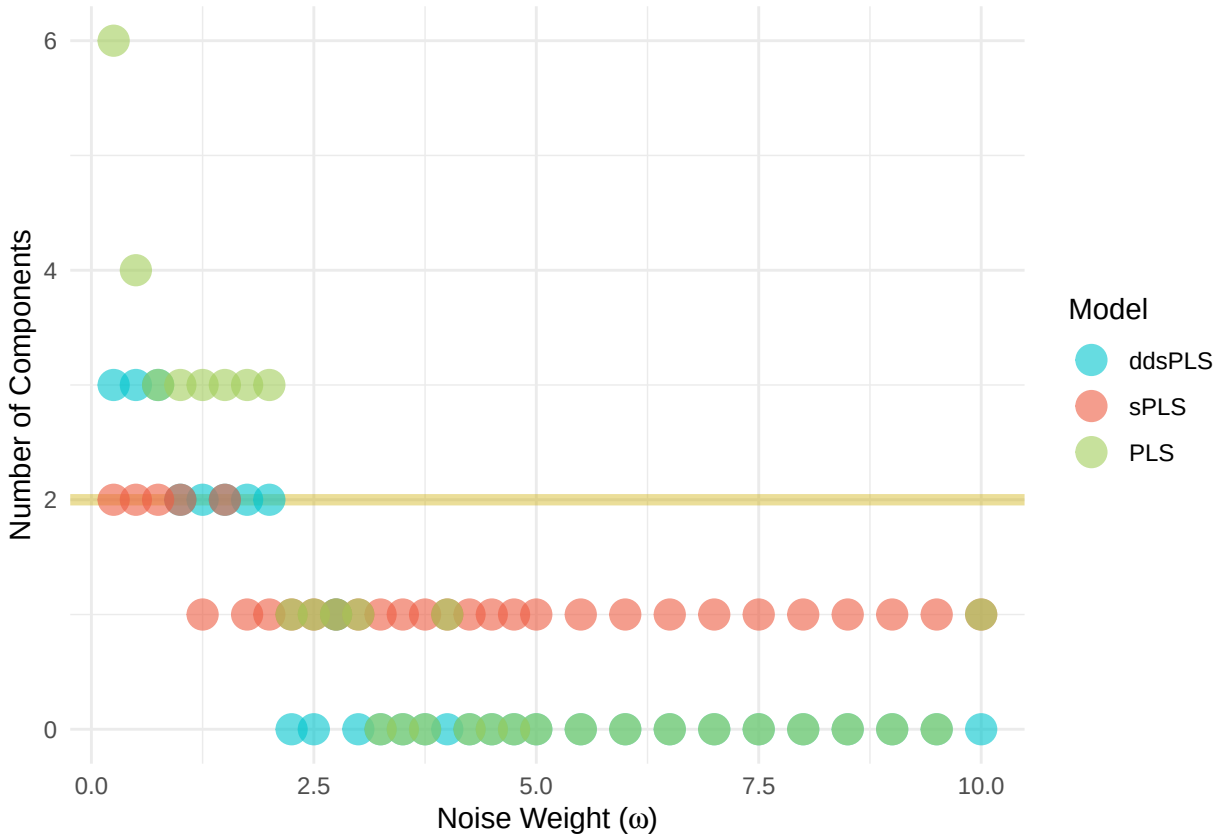
Relation between ω and the test set RMSE for the PLS model. n=100, p=100, q=5

Note the similar slopes of the line of best fit for all model types. Ultimately, these plots suggest that there is not a large difference in performance between model types as the amount of noise is increased. We should be careful about drawing any conclusions from results at higher values of ω as these models are usually just performing mean estimation and are not as dependent on the model used.

Now lets compare R^2 and Q^2 as ω changes.



Here we can see the issues that arise with the method sPLS uses to choose the number of components to build. When ddsPLS and PLS start performing mean estimation, sPLS continues to build a model. This leads to a higher R^2 but a lower Q^2 meaning models perform worse on test data than mean estimation. For $\omega > 2.5$, the noise drowns out the signal in the model

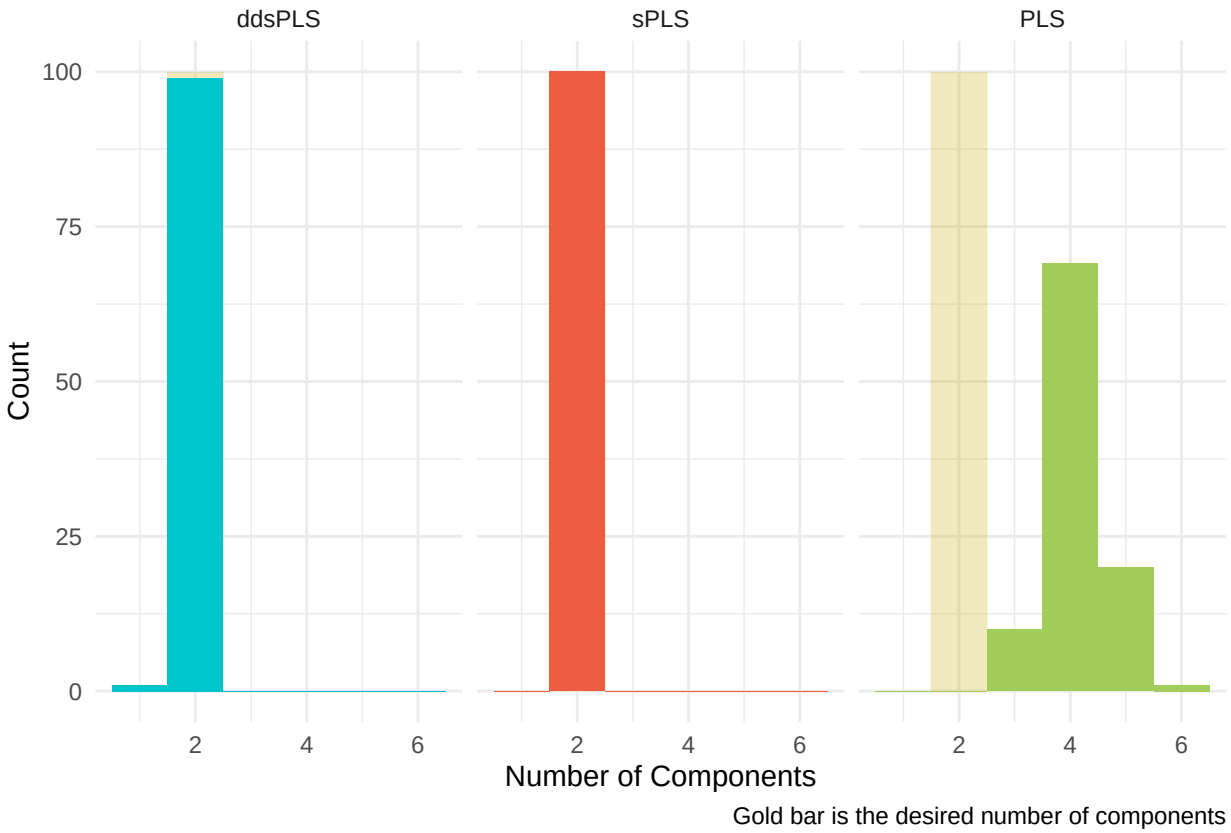


Here is the number of components built as the noise increases across model types. It is important to note that the sPLS tuning algorithm used always selects at least one component. When 0 components are included in the model this means that mean estimation performs better than the model built. Not building any components at high noise levels is not necessarily a bad sign as PLS methods are not designed to deal with problems due to high noise to signal ratios, it simply means that the noise included has drowned out the signal.

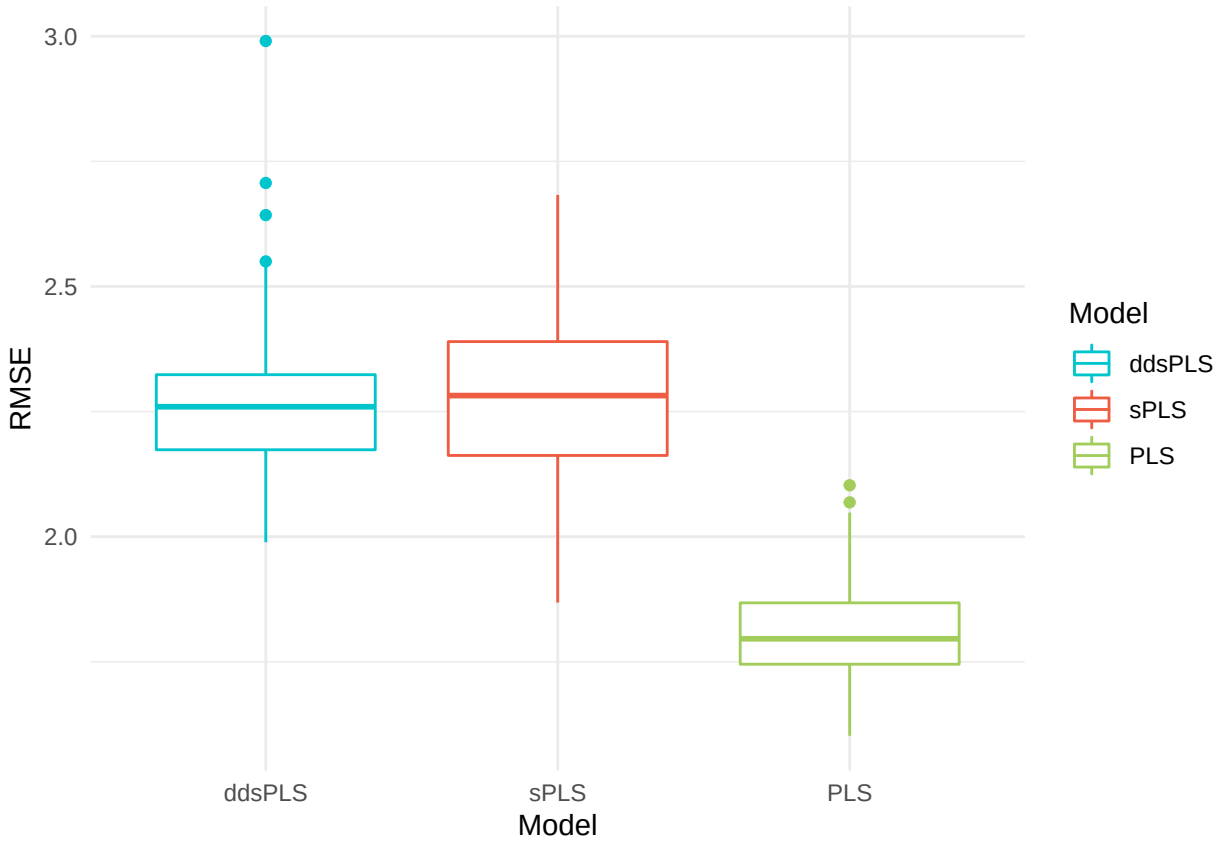
Here we can see that PLS tends to build more components than other models as we would expect since it makes no attempt to build sparser models.

Lets now compare model performance at fixed values of ω when $\mathbf{A}$ is generated using the complex method and $\mathbf{D}$ is generated using the 2-block method. For these trials we will fix $n = 100$, $p = 100$, and $q = 5$.

First let $\omega = 0.5$. At this noise level we can expect fairly robust and preictive models to be built.

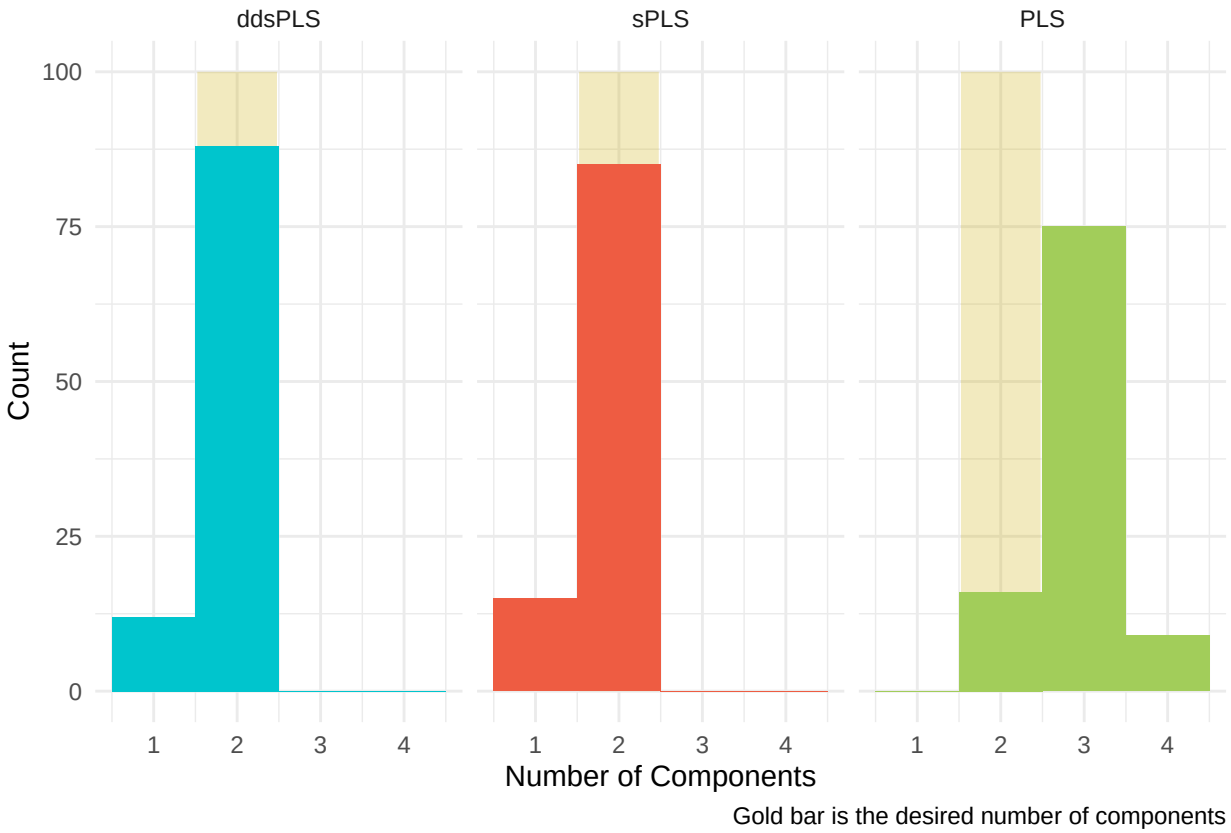


Due to the shape of the data, models should include 2 components. ddsPLS and sPLS both tend to build the correct number of components with sPLS building 2 components every time. PLS always builds too many components and never builds the correct number.

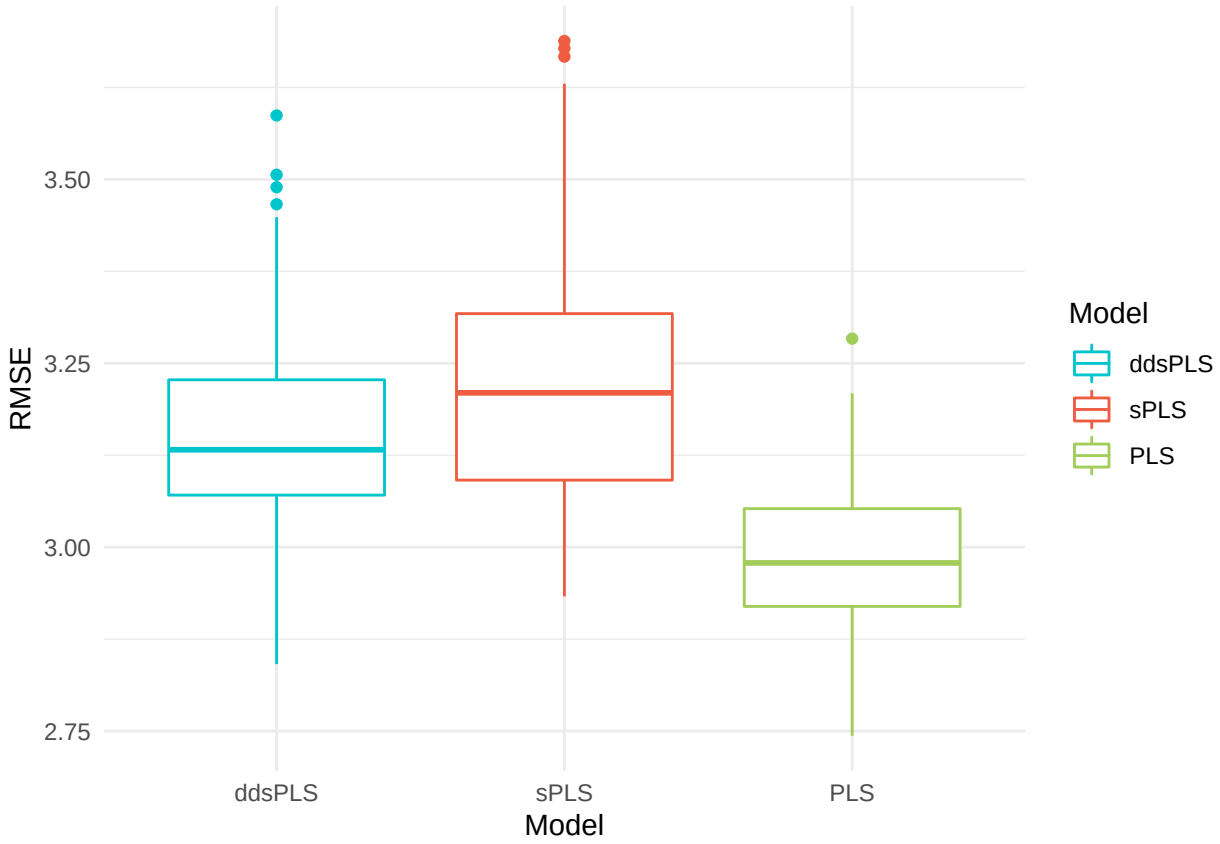


At $\omega = 0.5$, we can see that ddsPLS and sPLS perform similarly in regards to to RMSE while PLS performs significantly better.

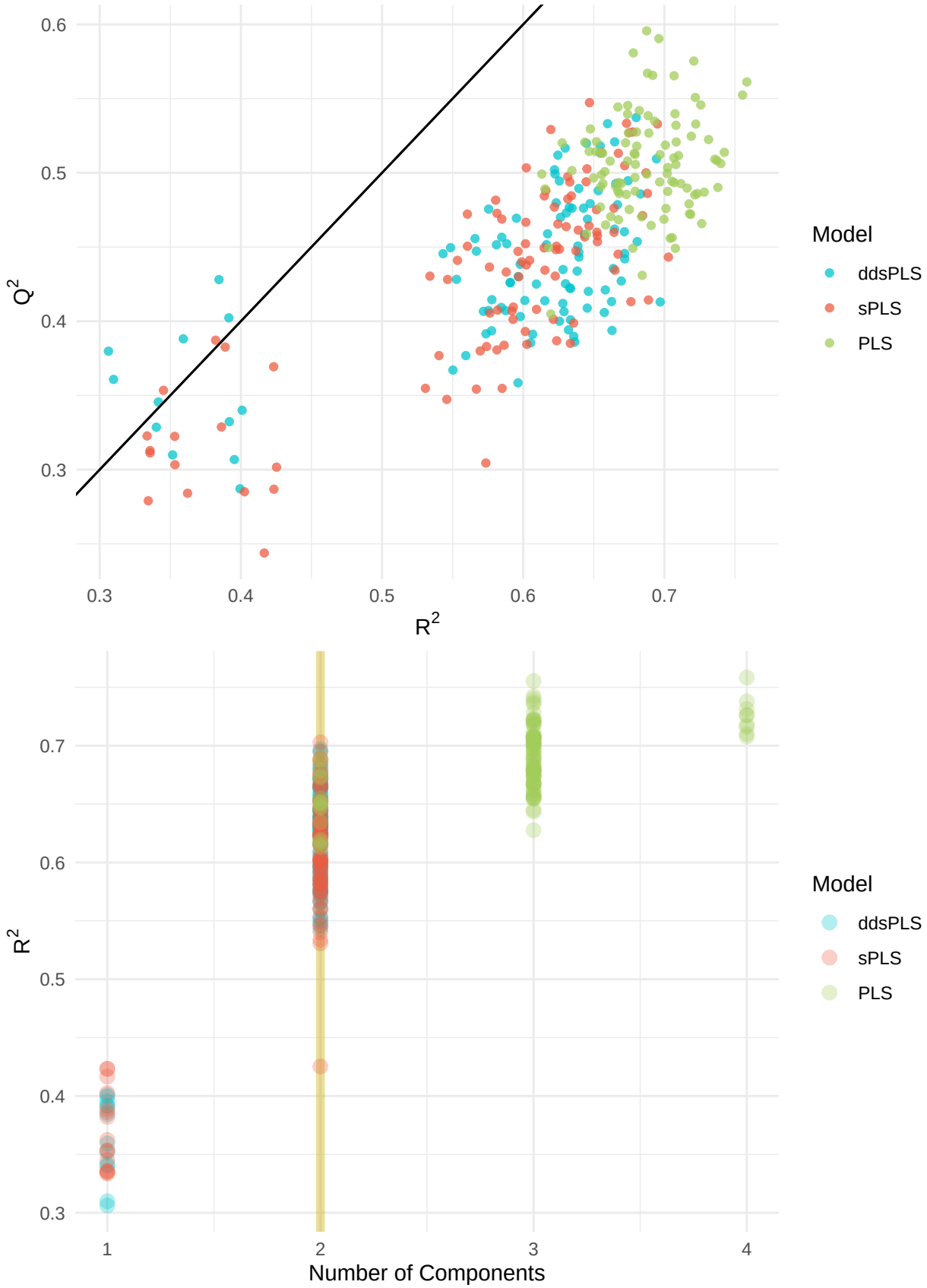
Now let $\omega = 1$.



ddsPLS and sPLS still tend to build the correct number of components when ω is increased. Both show an increased tendency to build too few components. PLS now builds fewer components but still tends to build too many.



Again PLS tends to have the lowest RMSE however it's performance is now more similar to the other models than with less noise. ddsPLS and sPLS still perform very similarly however we can see that ddsPLS now performs slightly better.


0.6
0.5
0.4
0.3
Q^2
0.3
0.4
0.5
0.6
0.7
R^2
Model
ddsPLS
sPLS
PLS
0.7
0.6
0.5
0.4
0.3
R^2
1
2
3
4
Number of Components
Model
ddsPLS
sPLS
PLS

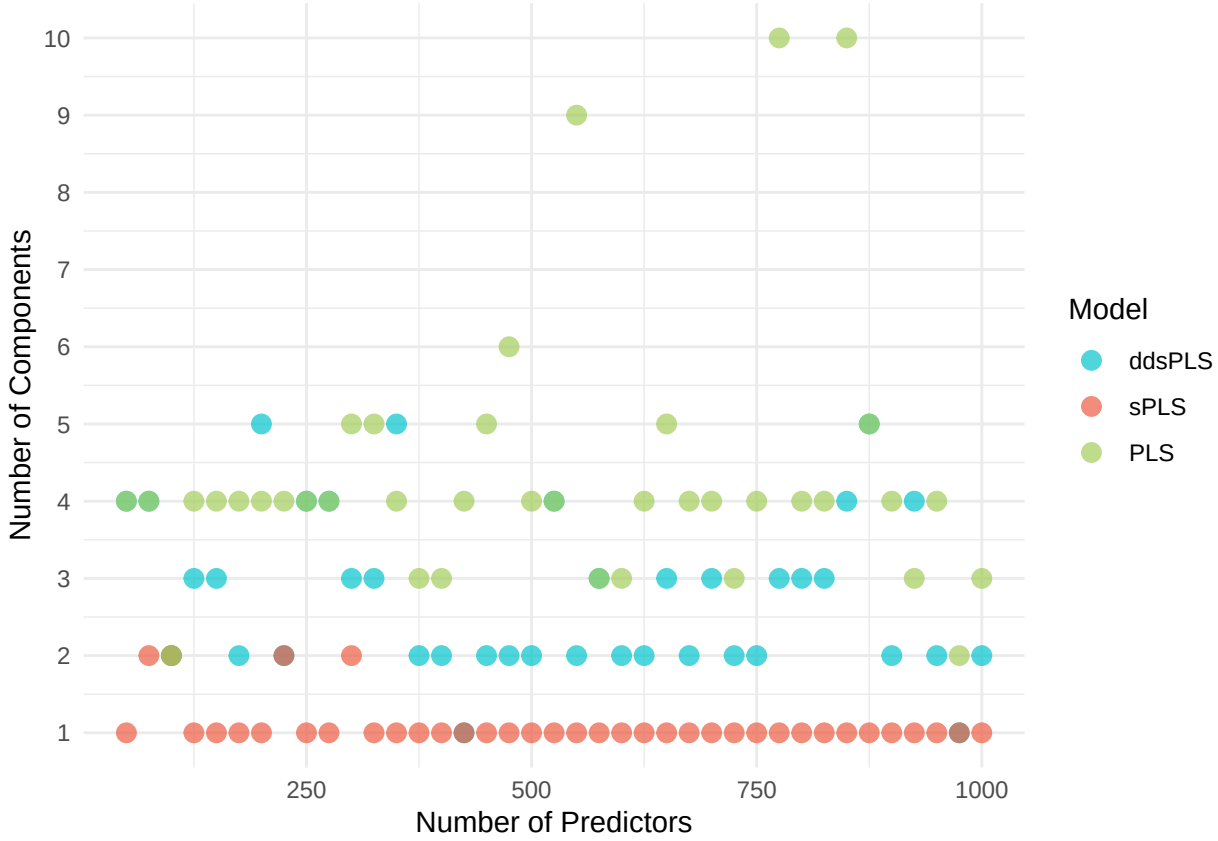
As this plot illustrates, there is a clear relationship between the number of components and the R^2 value. Almost all of the models that poorly explain the original data build only 1 component.

Here we will also introduce a problem that sparse models will suffer from. Looking at the results from PLS models, we see what we would hope to observe, models consistently have fairly high R^2 values while also having lower but still well performing Q^2 values. For the most part, ddsPLS and sPLS exhibit the same behavior however, we can notice an issue with the cluster of points in the lower left corner. These points exhibit cases where the original model failed to provide a good fit of the training data as evidenced by the noticeably lower R^2 value. While we expect sparse models to have a lower R^2 than PLS, the problems is that these models do not offer consistent performance on similar data.

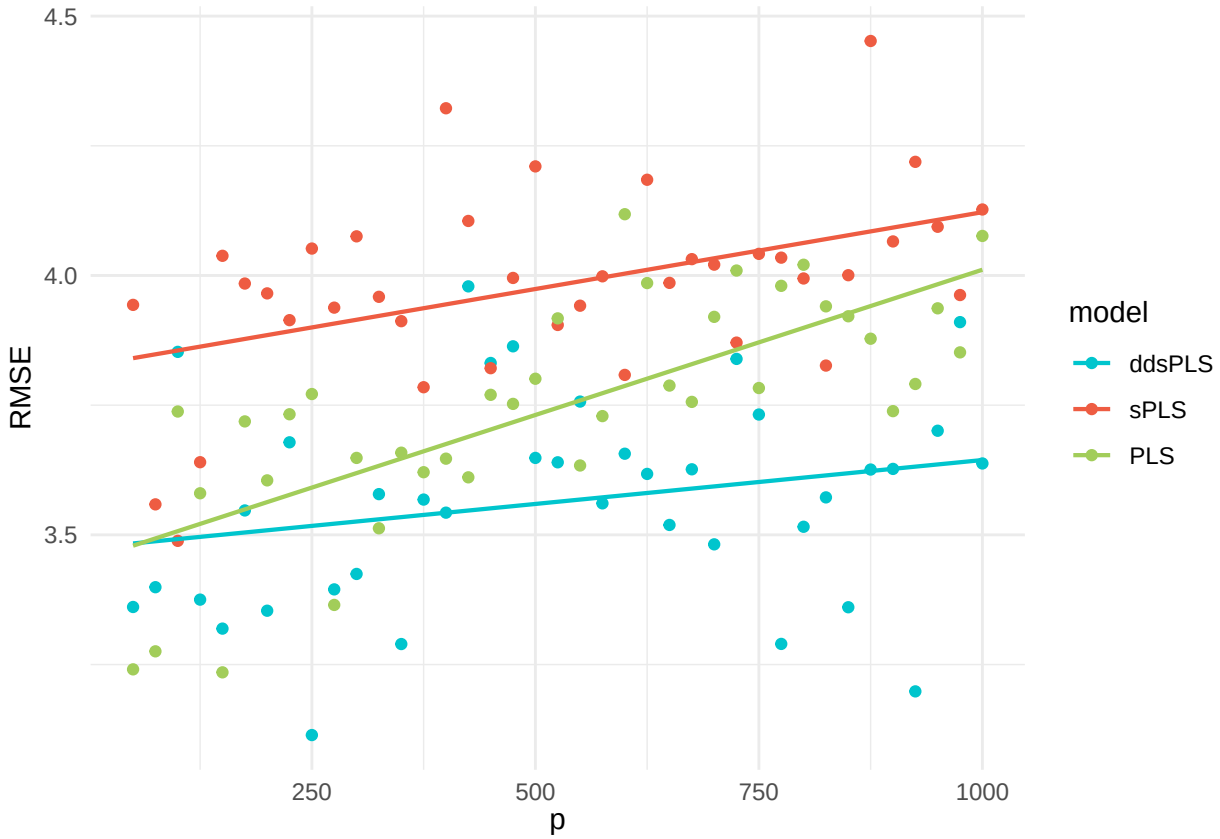
Due to the nature of these models, I would guess that this is due to the models overly penalizing model complexity. Due to the frequency of these models that underfit the data (12 of ddsPLS models and 16 of sPLS models) it is extremely unlikely that these results are due to random variation in the data that the different models fit. Since this problem is prevalent in both ddsPLS and sPLS, at least part of it is likely due to the models either heavily penalizing or selecting to remove relevant predictors.

Predictors

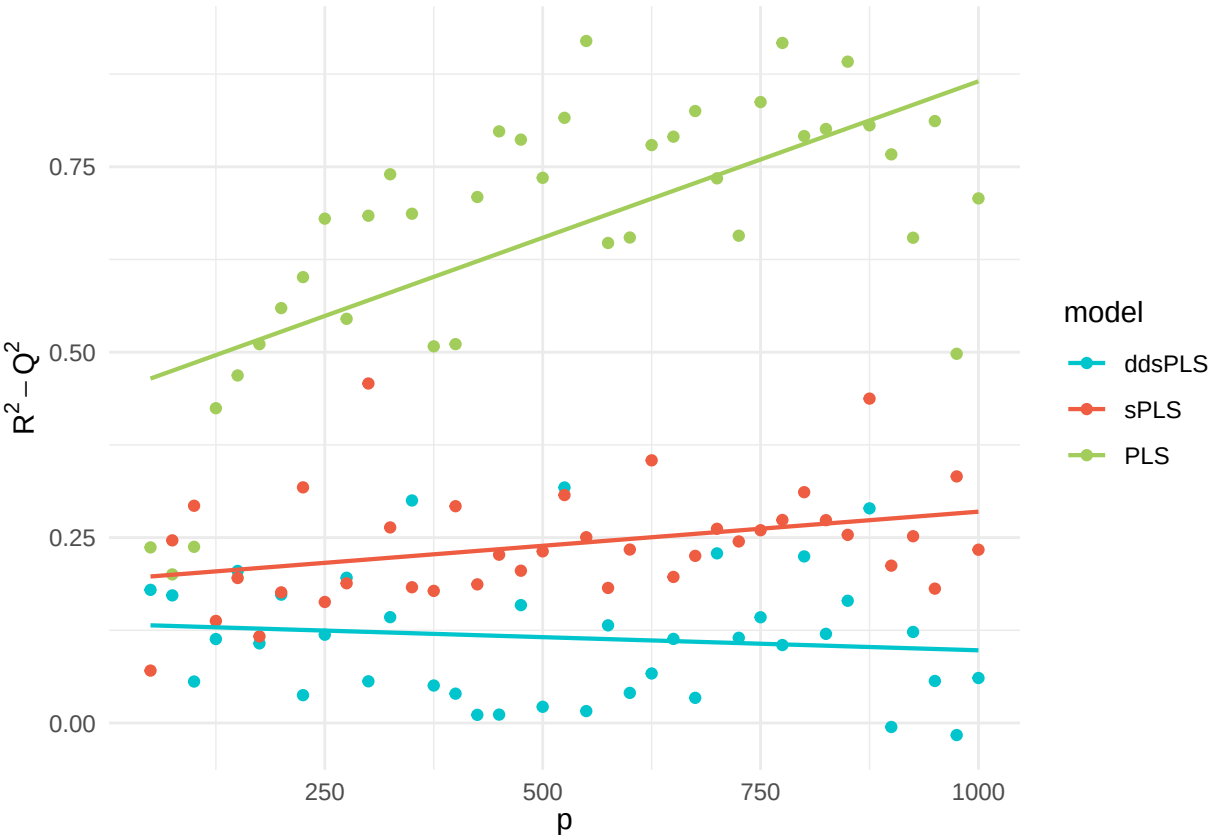
The following are for data generated with $n = 50$, $q = 5$, $\omega = 1$, **A** generated with a complex structure, and **D** generated with the 2-block method.



Here we can see the differing tendencies in the number of components that the models build. Ideally, models will build two components* in this situation. ddsPLS performs by far the best in this regard as PLS tends to overfit the data while sPLS underfits it.



As p increases all PLS models show some degree of decline in performance as we would expect. Both of the sparse models exhibit a noticeable but fairly moderate increase in the RMSE. Meanwhile PLS exhibits a more drastic increase in its RMSE.

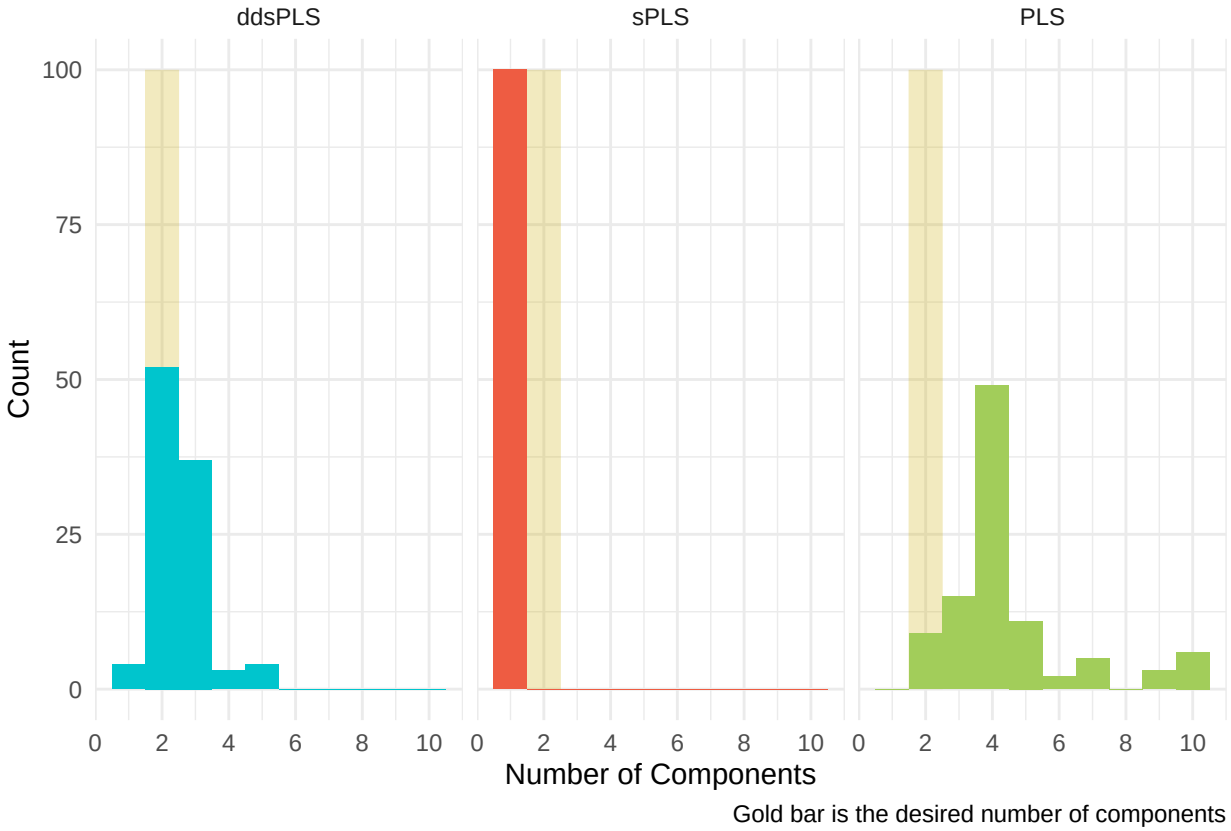


The increase in RMSE for the PLS model is due to it overfitting the data. As p increases, so does the difference in R^2 and Q^2. This is due to PLS fitting random correlation that appears in the noise that the sparse models are able to mostly ignore. Since ddsPLS is the only model to use $R^2 - Q^2$ as the metric for parameter selection it is unsurprising that it is the only one of the three models for which the metric decreases.

The following table shows the results of 100 replications with $p = 1000$ for each model type for the RMSE.

Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.569	3.454	3.565	3.674	0.0297
sPLS	4.052	3.954	4.041	4.142	0.0149
PLS	3.945	3.843	3.943	4.045	0.0185

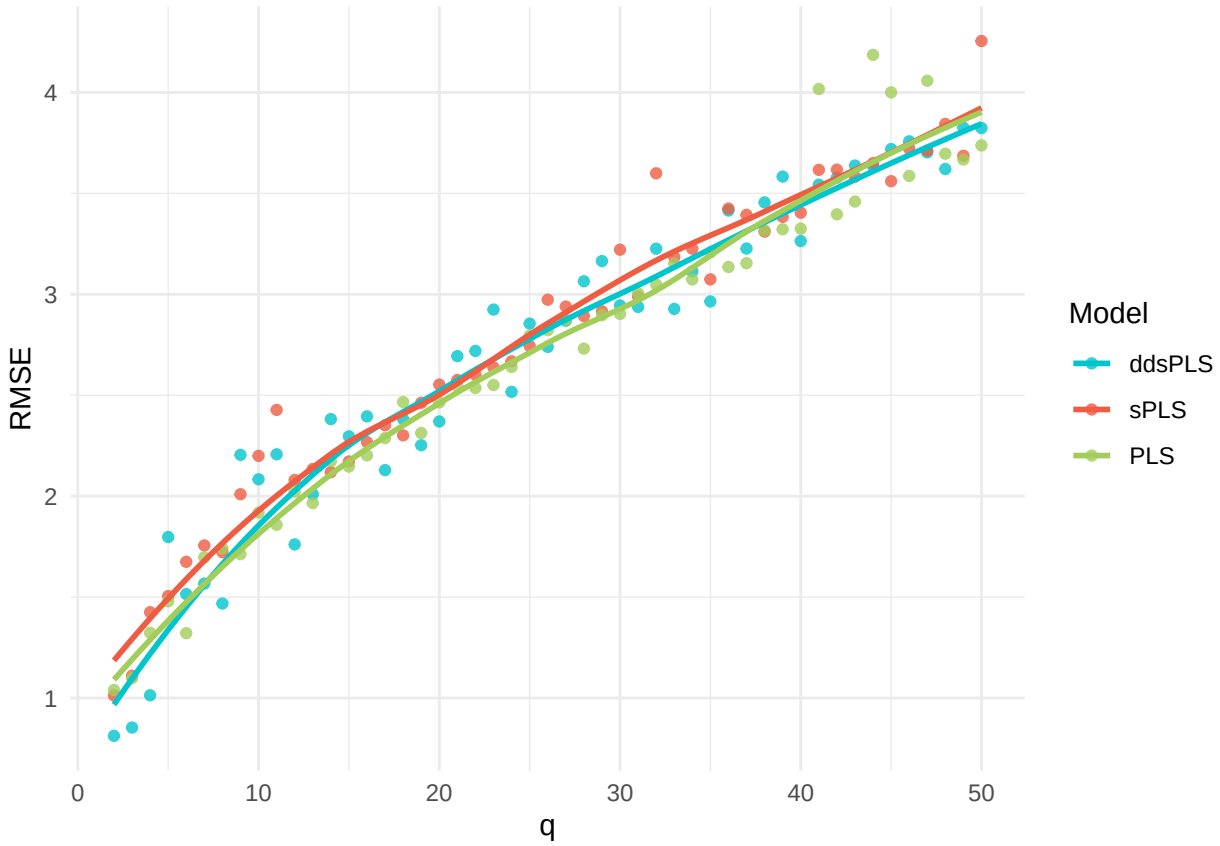
From this chart we can see that with a large number of predictors uncorrelated to the responses, ddsPLS performs significantly better than the other methods in regards to the RMSE. While the mean RMSE of PLS is significantly lower than that of sPLS the two often offer models that perform similarly. Furthermore, note the higher variance of the RMSE for ddsPLS models. This is representative of the performance of ddsPLS in general as models tend to be more variable in their performance on the test data.



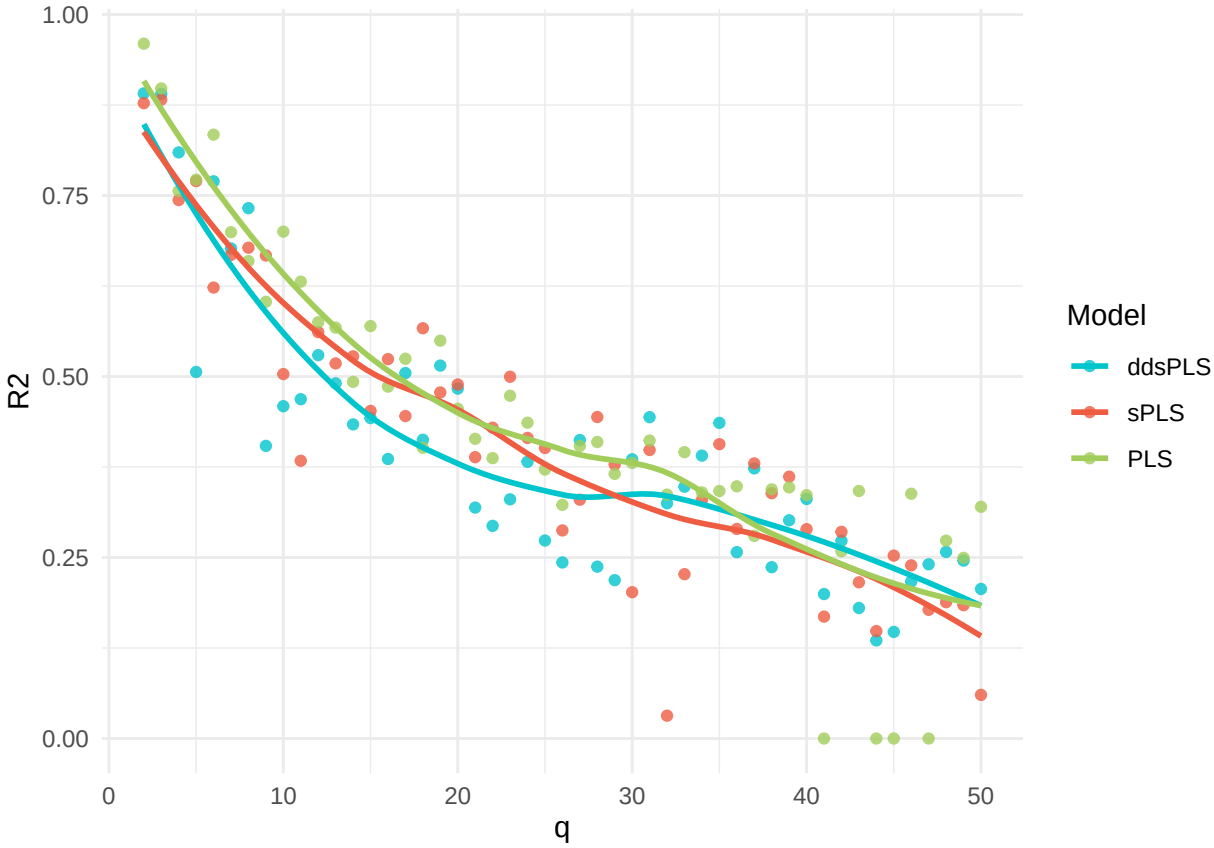
When looking at the number of components built during the replications, we can see that ddsPLS is the only model that tends to build the correct number of components. As in previous trials, sPLS uses too few components while PLS builds too many.

Responses

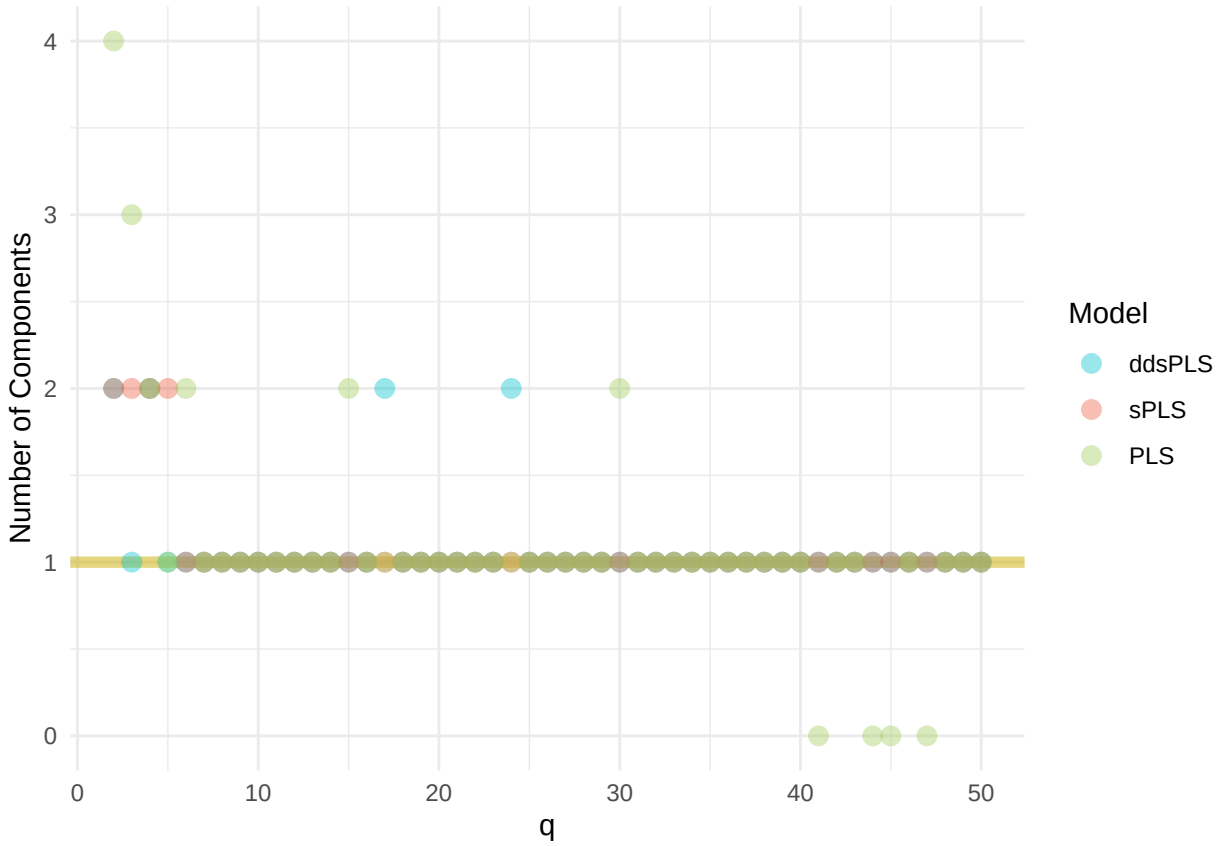
The following are for data generated with $n = 50$, $p = 100$, $\omega = 0.5$, $\mathbf{A}$ generated with a simple structure, and $\mathbf{D}$ generated with the simple method.



As expected, the RMSE increases as the number of response variables increase. More importantly, we can see that all models continue to perform similarly as the number of response variables increases.

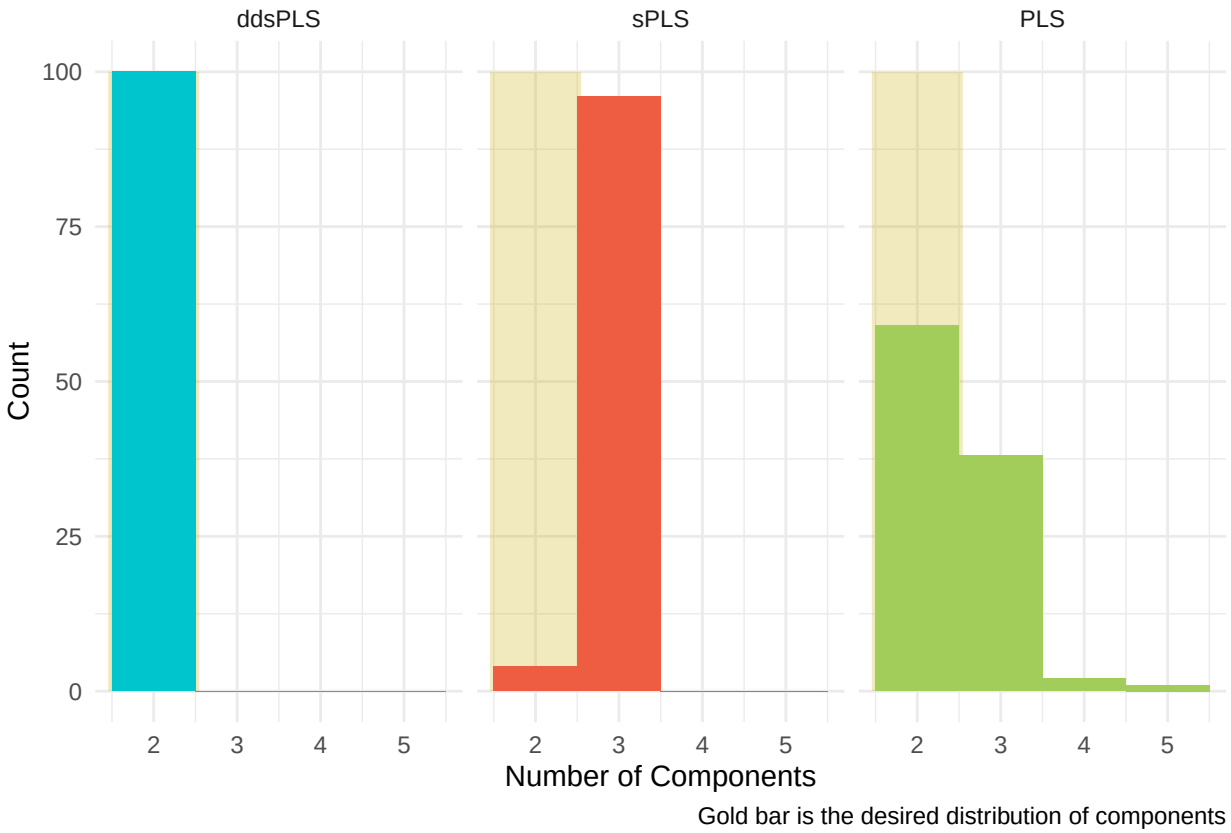


As q increases, we see all models perform similarily in their ability to fit the data.

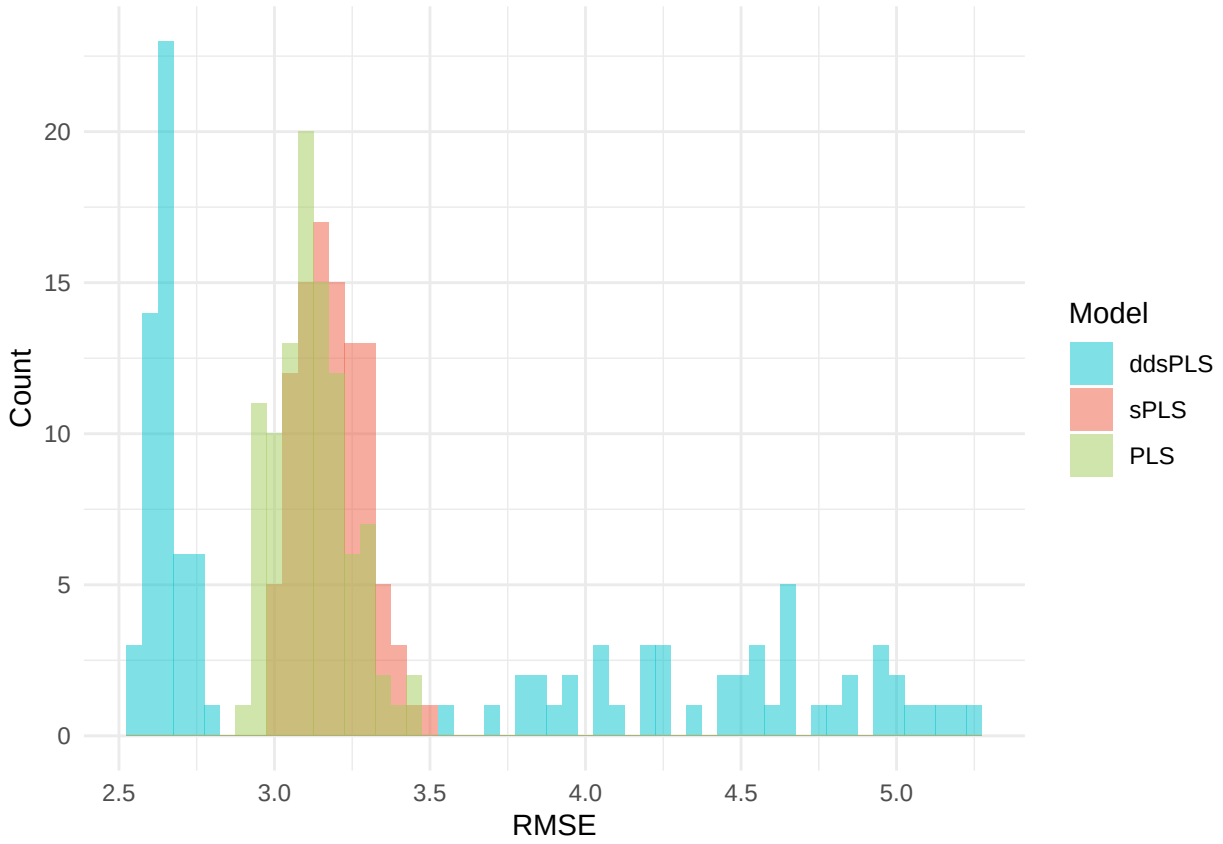


Models tend to build the correct number of components in this simple case when there are a small number of responses, we can see that all models have a slight tendency to build too many components however, this quickly stops at around $q = 5$.

The following results are for $q = 25$, $n = 50$, $p = 100$, $\omega = 0.5$, $\mathbf{A}$ generated with a simple structure, and $\mathbf{D}$ generated with the basic method.



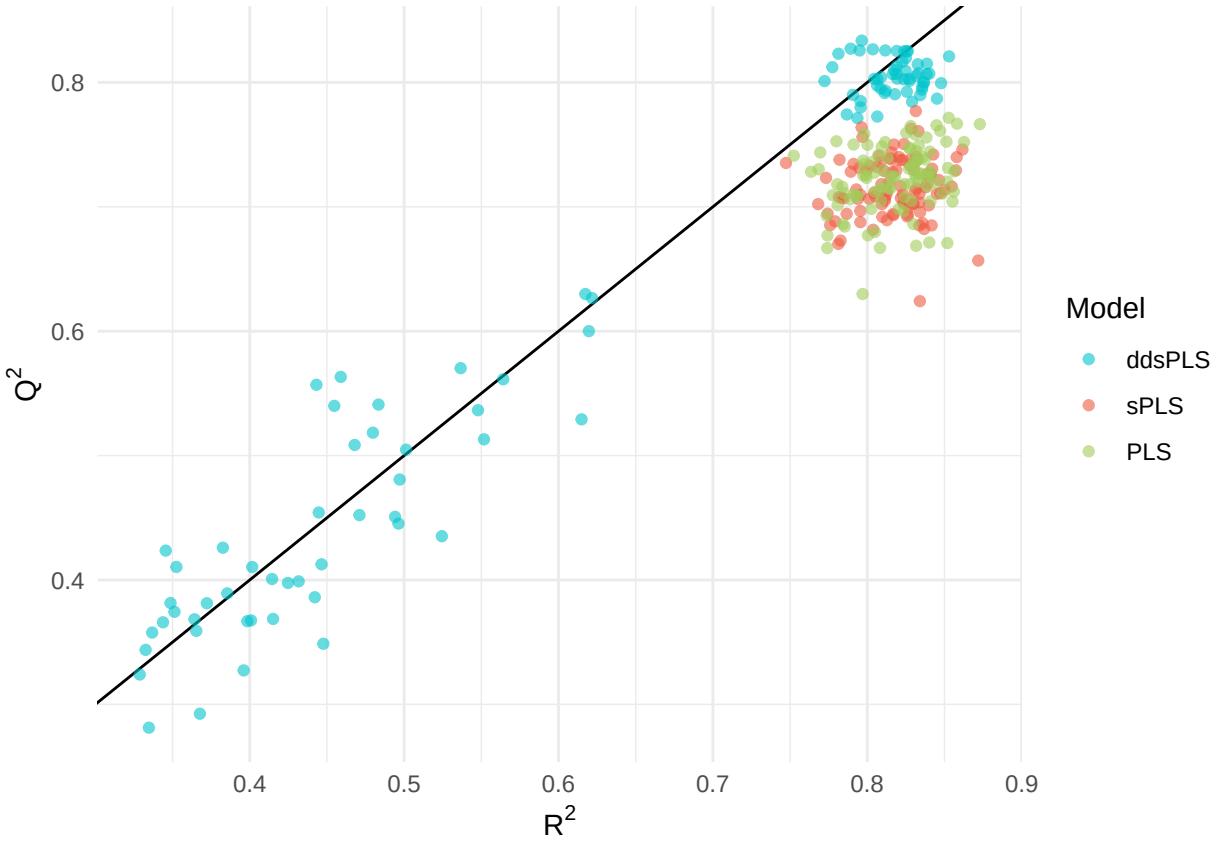
Interestingly, sPLS is most likely to build too many components in this scenario where there are a large number of predictors that are all strongly correlated. Here ddsPLS is able to build the correct number of components every time.



sPLS and PLS have a similar distribution of the RMSE with PLS performing slightly better. ddsPLS exhibits a much different behavior with models either performing notably better than the other two methods or far worse.

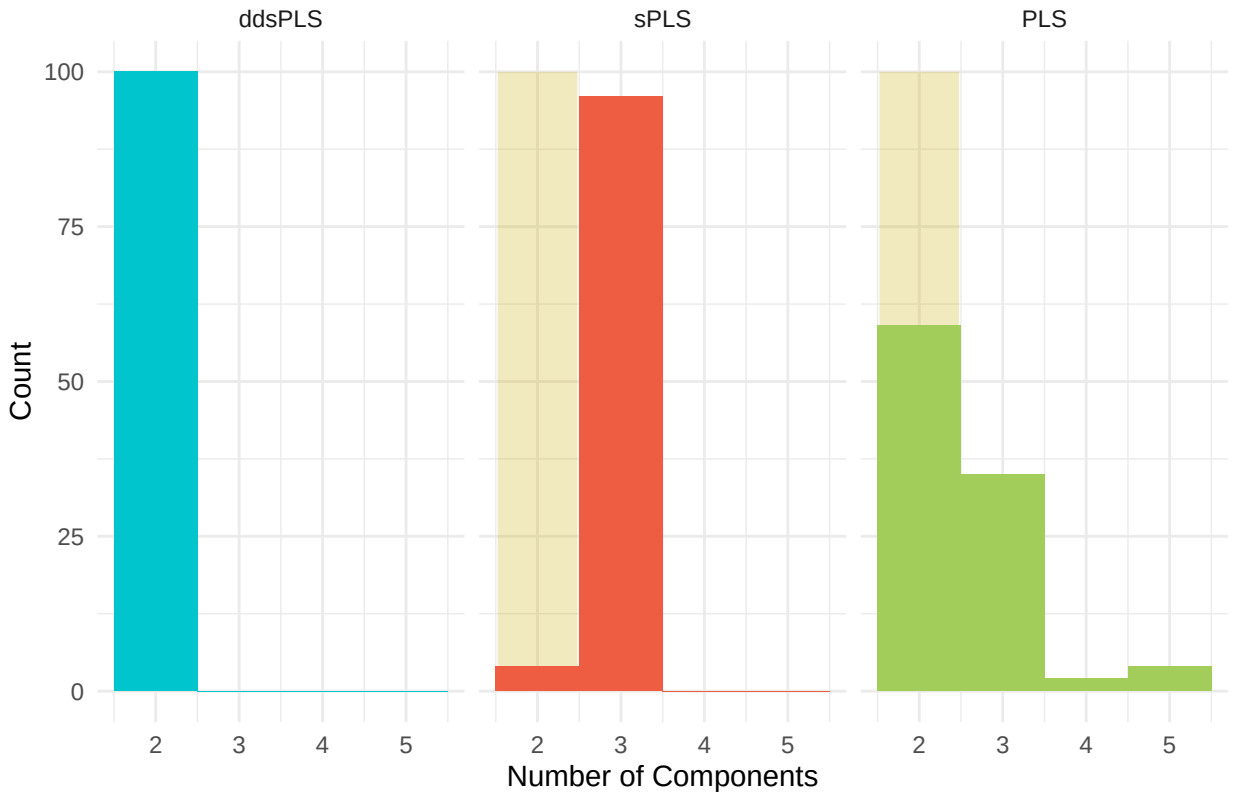
Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.502	2.644	2.756	4.448	0.9230
sPLS	3.187	3.107	3.178	3.257	0.0118
PLS	3.124	3.042	3.112	3.200	0.0145

Based on summary statistics of the distribution of RMSE we can see that the variance of ddsPLS performance is much higher than the other two models which perform fairly similarity. A slight majority of ddsPLS models perform better with the rest performing much worse. This seems to signal ddsPLS as a high-risk, high-reward model to use in these cases.

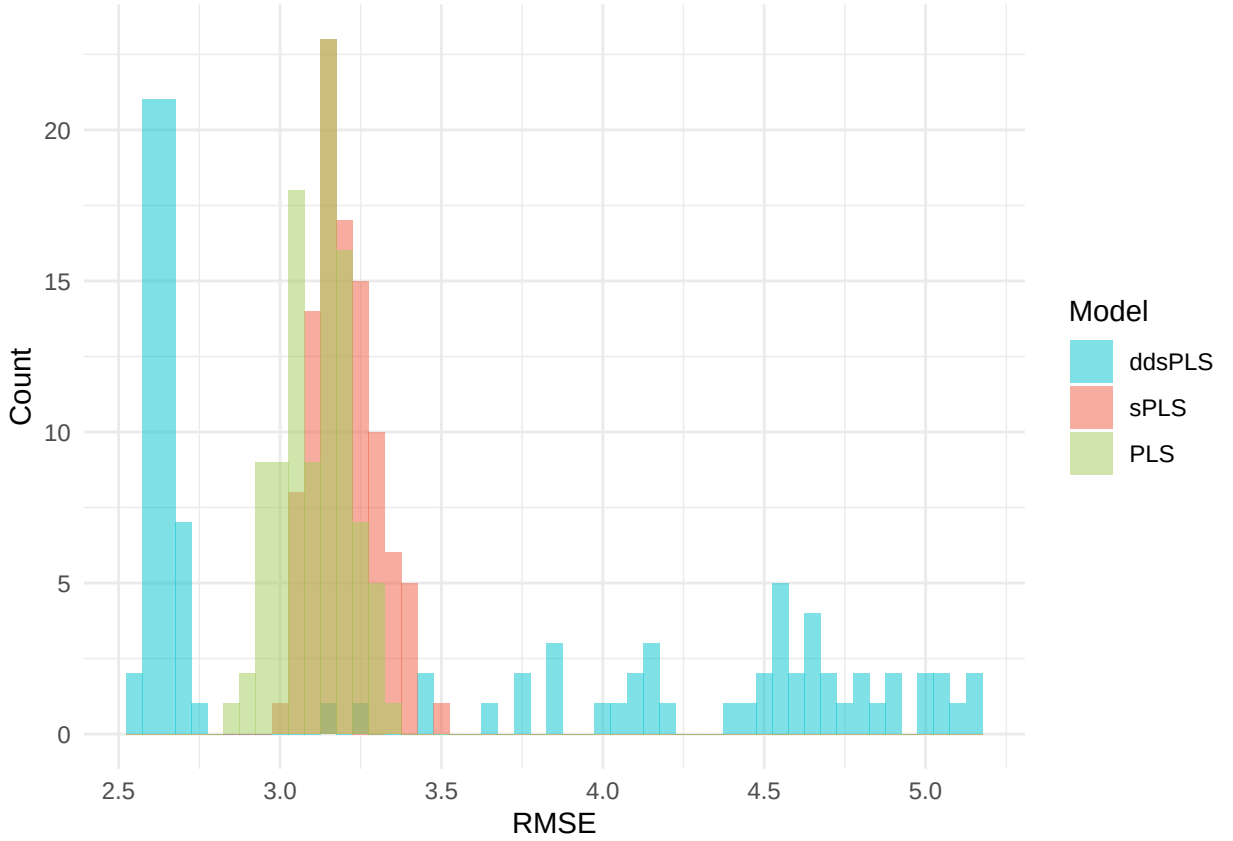


Here we can see that low R^2 values seem to be causing the high RMSE. Notice that for ddsPLS, $Q^2 \geq R^2$ in many cases, behavior that we do not expect to see. This suggests that the method is doing a good job of ignoring noise in the data and only reflecting parts of the underlying structure. The problem seems to be that ddsPLS is also ignoring signal at times and only capturing part of the underlying structure. In these instances, we see a lower R^2 value and thus a similar but also lower Q^2 value. Note for that sPLS and PLS, we always have $Q^2 < R^2$ showing that the models are to some degree fitting noise.

The following models are generated in a similar way as before, except this time $\mathbf{D}$ is generated with the 2-block method.



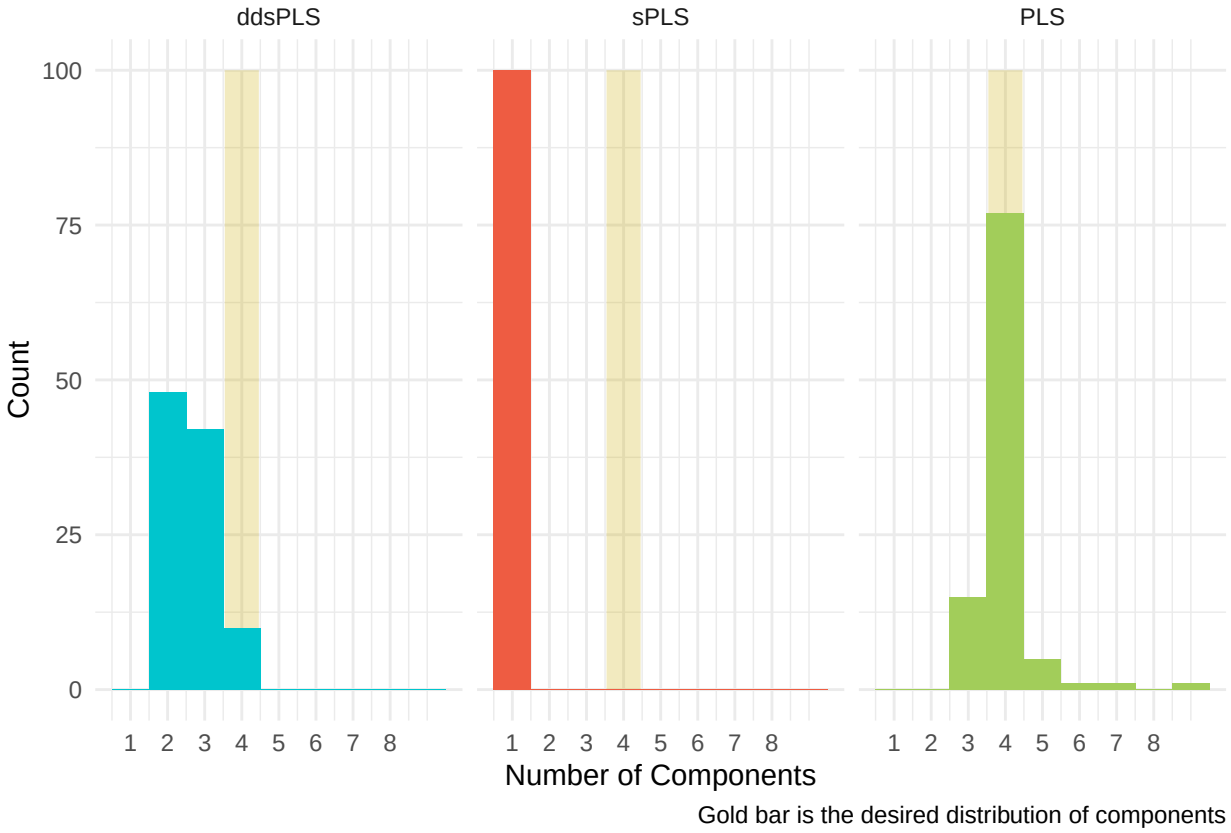
Gold bar is the desired distribution of components



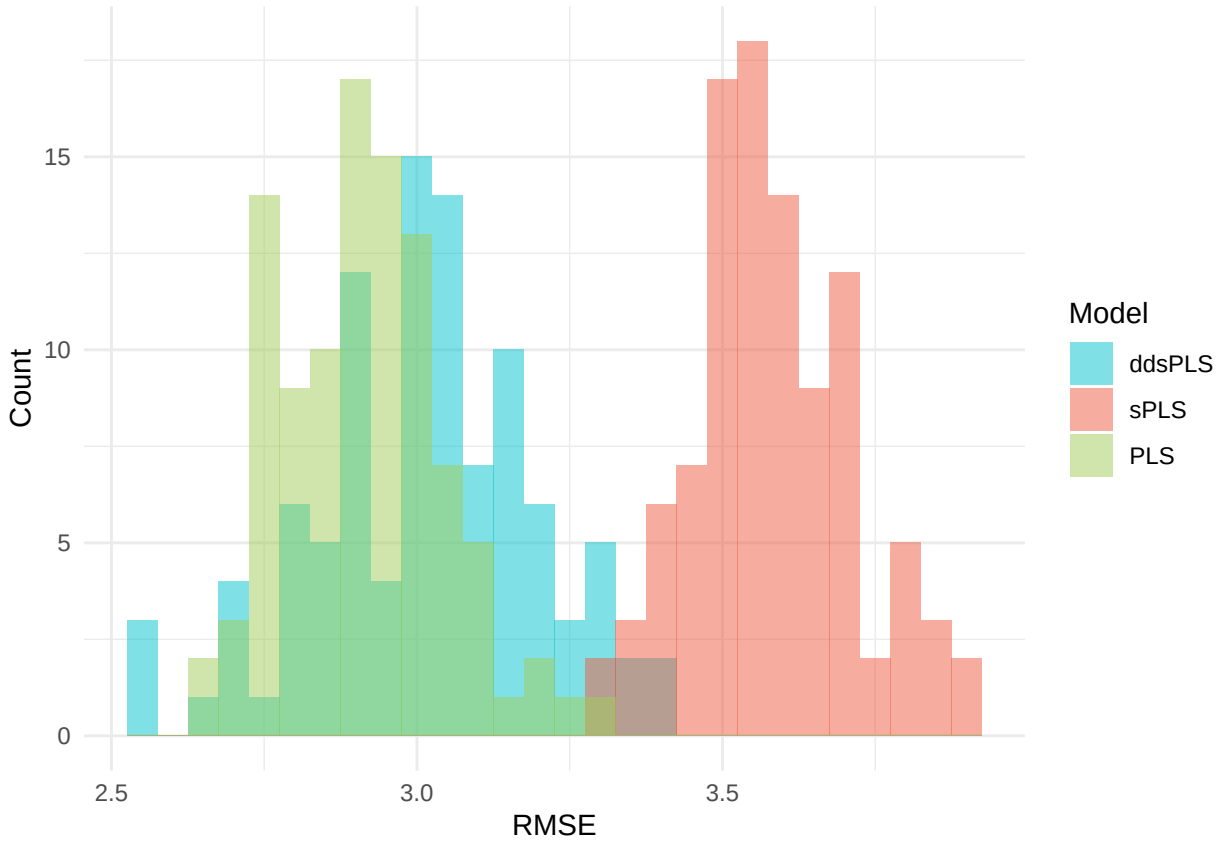
Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.480	2.627	2.718	4.531	0.9141
sPLS	3.197	3.134	3.181	3.263	0.0097
PLS	3.112	3.033	3.127	3.185	0.0110

Comparing these to the results of using the diagonal method to generate $\mathbf{D}$ we see little change. Nearly identical results appear both times.

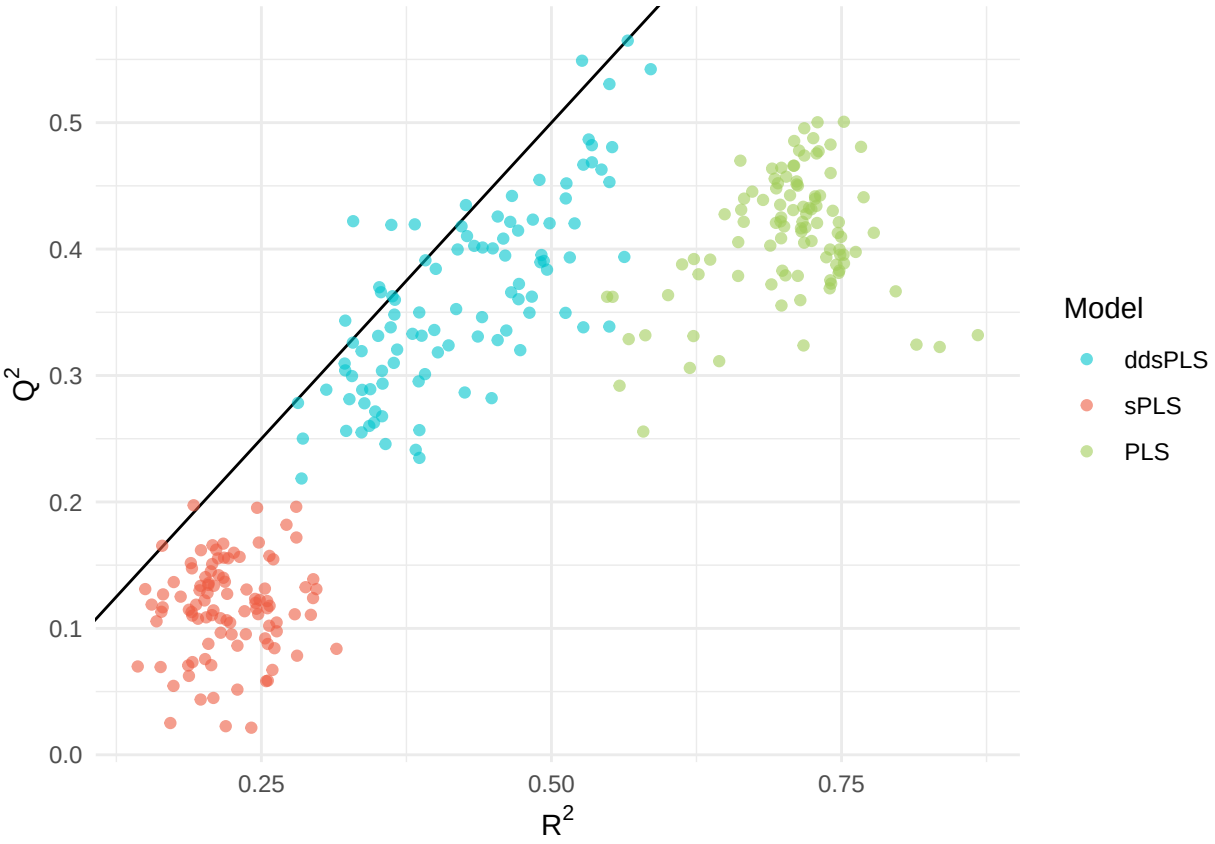
The following are for $q = 10$ with $\mathbf{A}$ generated with a complex structure and $\mathbf{D}$ generated with the 4-block method.



With a more complex structure to both the predictors and response variables, ddsPLS and sPLS tend to build too few elements. Surprisingly, PLS is best at building the correct number of components in this situation.



ddsPLS and PLS perform similarly in regards to the RMSE, with PLS having a slight edge in performance. sPLS performs significantly worse.

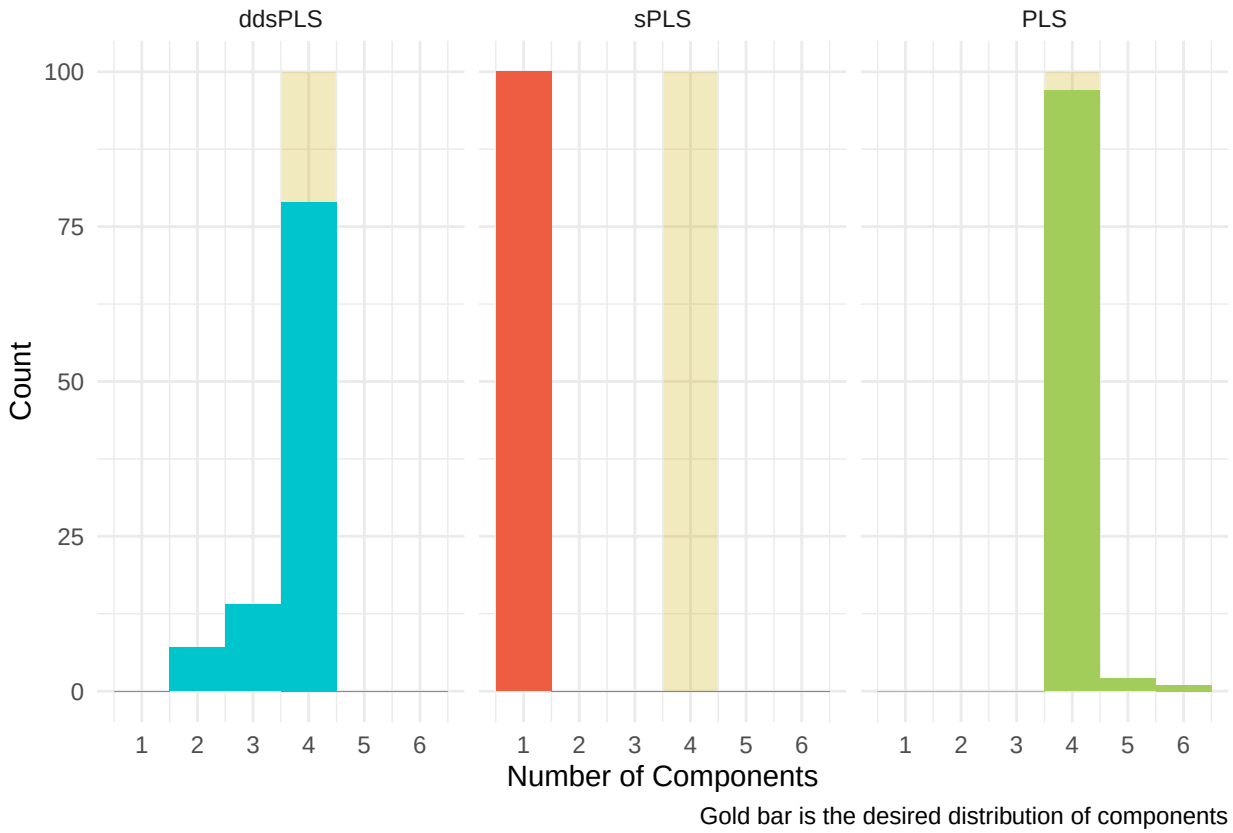


Both ddsPLS and sPLS exhibit an ability to avoid including noise in their models. Especially in the case of sPLS, this looks like it also causes them to miss some of the signal. While PLS models are more prone to include noise in the models as we can see in the larger difference between R^2 and Q^2. This also suggests that while PLS is building the correct number of components, these components may not be the most accurate.

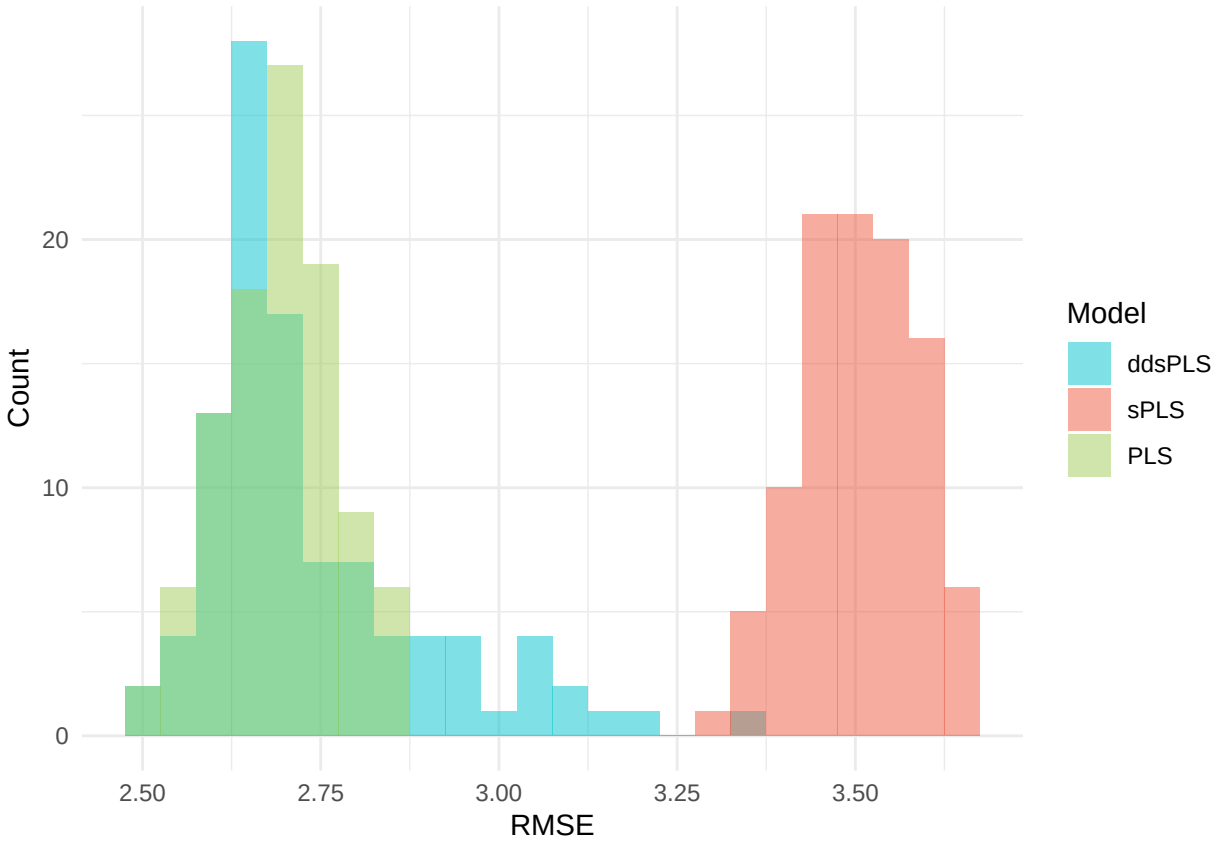
Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.016	2.914	3.024	3.132	0.0343
sPLS	3.582	3.496	3.572	3.671	0.0176
PLS	2.910	2.808	2.893	2.992	0.0169

While ddsPLS doesn't suffer as drastic a problem with underfitting as in previous trials, it still has the most variable model performance.

The following results are for a simulation of models built with the same structure as before this time with a sample size of $n = 100$.



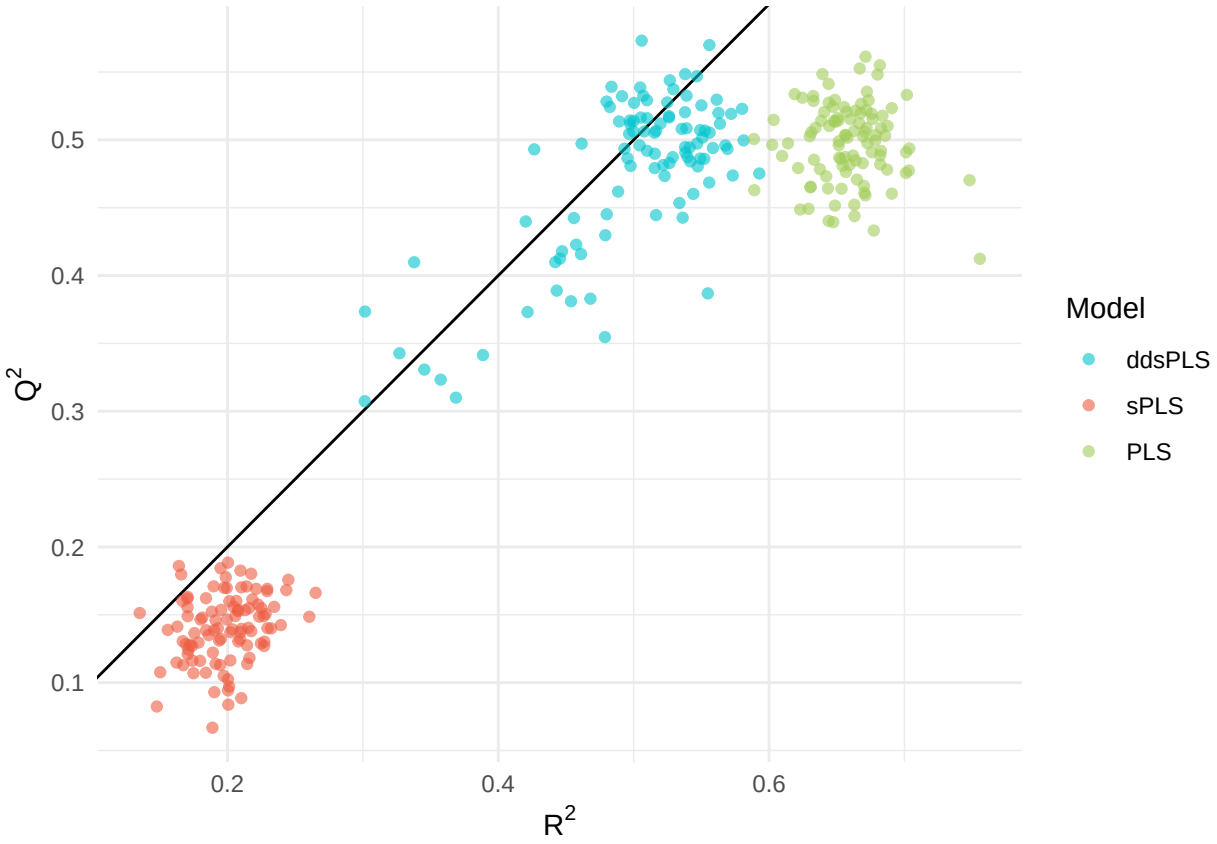
With a larger sample size, we now see that ddsPLS tends to build the correct number of components. This suggests that the number of components built when $n = 50$ was largely due to the sample size and less indicative of a problem with the performance of ddsPLS on more complex data structures. sPLS again builds only 1 components every time suggesting that the model does have issues on data with a more complex structure.



With the larger sample size the performance of ddsPLS and PLS becomes much more similar. sPLS doesn't exhibit the same decrease in RMSE that the other two models see. ddsPLS still exhibits some of the problem with variability in model performance that we have observed as is evidenced in the longer tail.

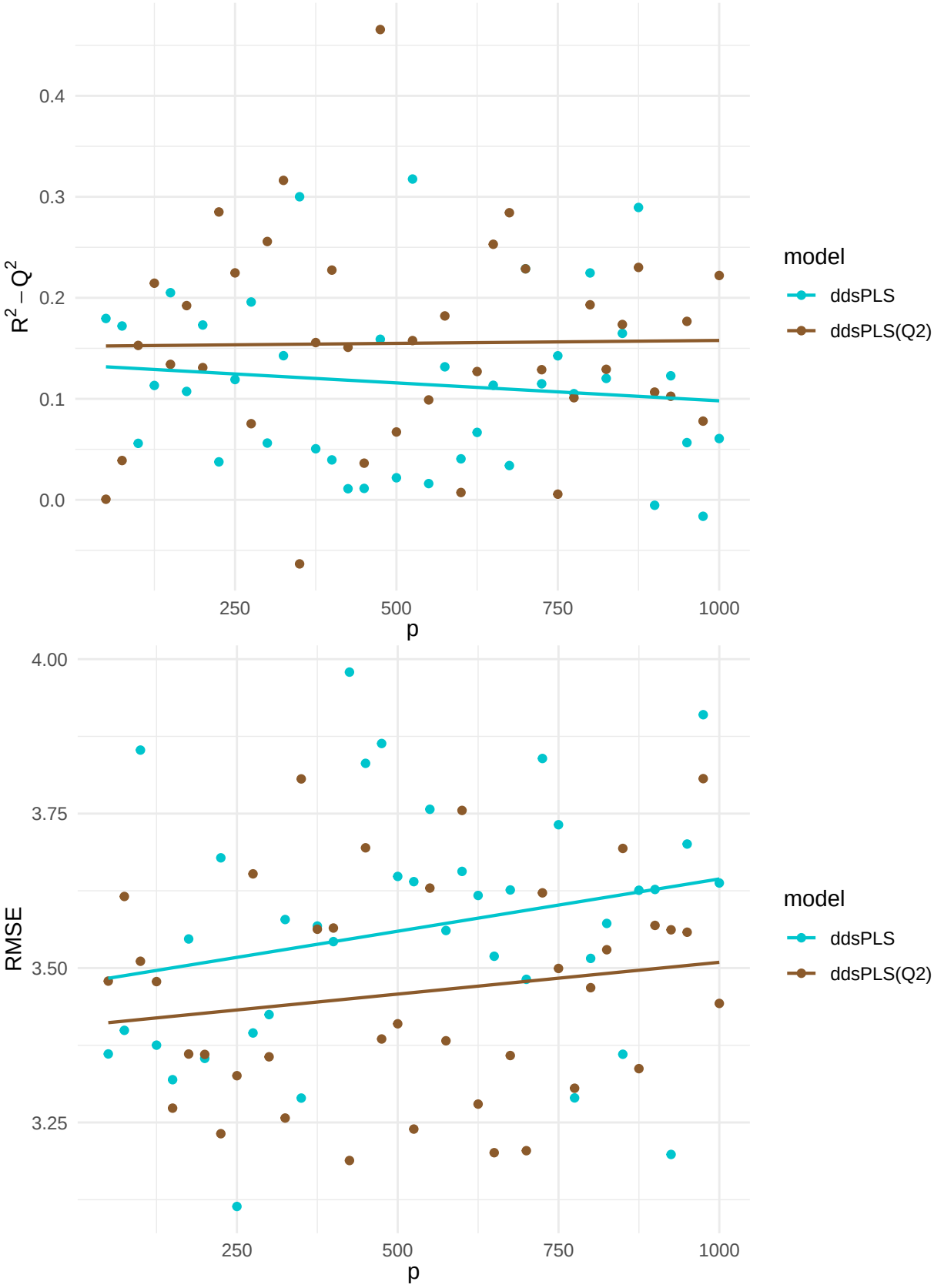
Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	2.737	2.640	2.684	2.809	0.0259
sPLS	3.507	3.449	3.511	3.563	0.0064
PLS	2.691	2.634	2.694	2.743	0.0061

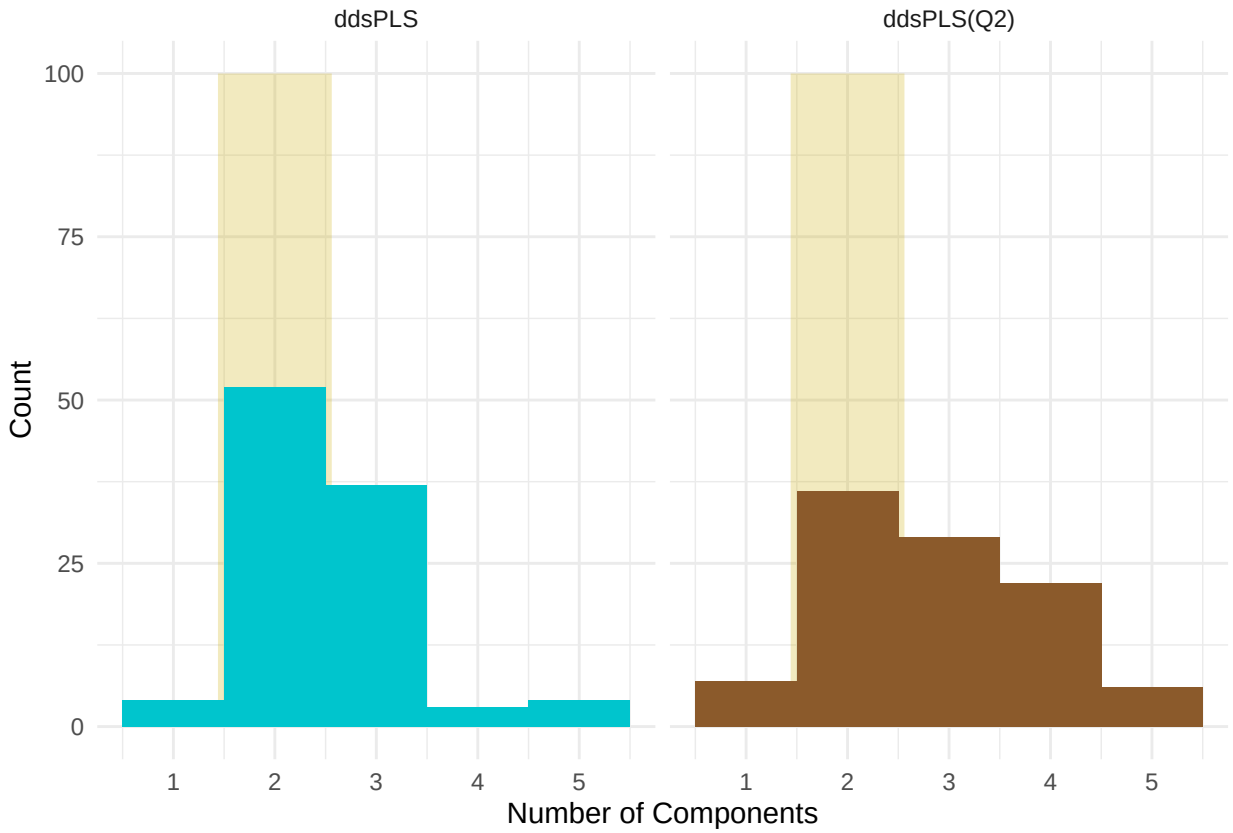
PLS still has a slightly lower RMSE than ddsPLS on average. Again, ddsPLS has a significantly higher variance than the other two models.



This plot clearly illustrates the problems of points that underfit the data. A number of ddsPLS models have noticeably lower R^2 values and thus lower Q^2 values. Both sPLS both have a more defined cluster of points, exhibiting close R^2 and Q^2 values across all models.

ddsPLS Parameter Selection





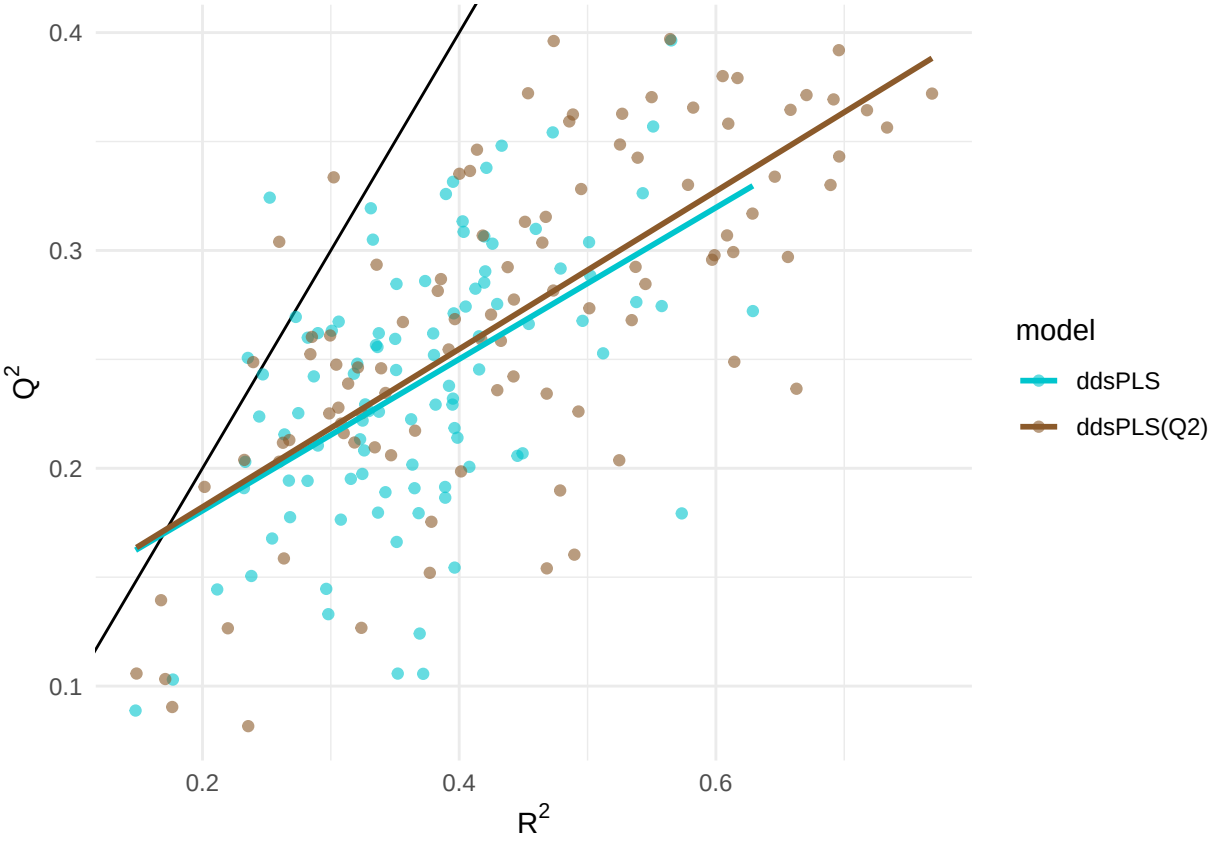
This plot illustrates a drawback of using Q^2 in order to select parameters. Models using Q^2 to select parameters are more likely to build too many components

```
reps_p %>%
  filter(model %in% c("ddsPLS", "ddsPLS(Q2)")) %>%
  group_by(model) %>%
  summarise(Mean = round(mean(rmse), 3),
            "25th quantile" = round(quantile(rmse, 0.25), 3),
            "50th quantile" = round(median(rmse), 3),
            "75th quantile" = round(quantile(rmse, 0.75), 3),
            Var = round(var(rmse), 4)) %>%
  gt() %>%
  cols_label(model = "Model")
```

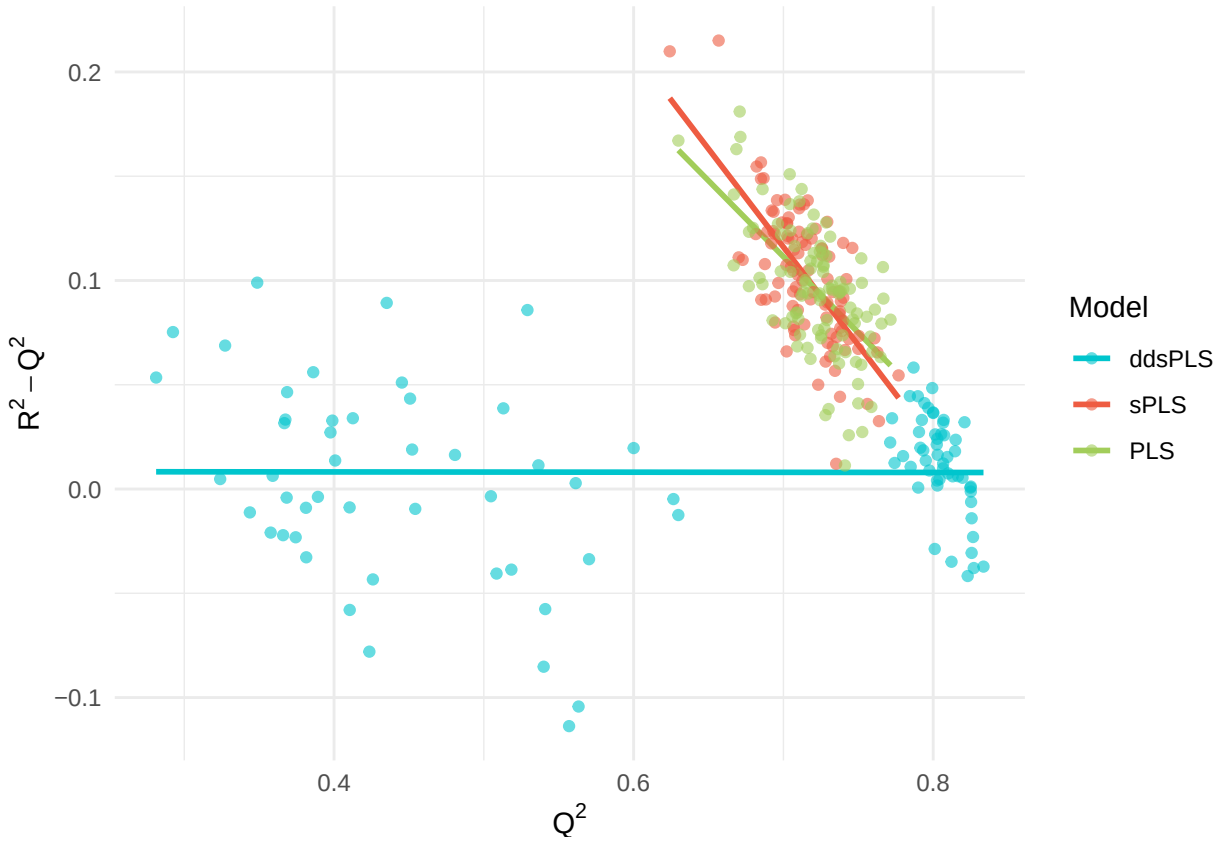
Model	Mean	25th quantile	50th quantile	75th quantile	Var
ddsPLS	3.569	3.454	3.565	3.674	0.0297
ddsPLS(Q2)	3.508	3.356	3.492	3.618	0.0385

```
reps_p %>%
  filter(model %in% c("ddsPLS", "ddsPLS(Q2)")) %>%
  ggplot(aes(x = R2, y = Q2, color = model)) +
  geom_abline(slope = 1, intercept = 0, size = 0.5) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "lm",
              formula = y~x,
              se = FALSE,
```

```
            alpha = 0.5) +
scale_color_manual(values = c(ddsPLS = "turquoise3",
                              "ddsPLS(Q2)" = "tan4")) +
labs(x = unname(TeX("$R^2$")),
     y = unname(TeX("$Q^2$"))) +
theme_minimal()
```



In this plot we see that the relation between R^2 and Q^2 is very similar for the two methods of parameter selection. When comparing ddsPLS to other models, we often saw that ddsPLS would have a closer association between R^2 and Q^2, i.e. if a PLS and ddsPLS model had the same R^2 value, then the ddsPLS model would tend to have a higher Q^2 value. However, this appears more to be a feature of how ddsPLS models rather than due to the fact that $R^2 - Q^2$ is used for parameter selection.



This is a plot of Q^2 and $R^2 - Q^2$ from simulations discussed in the responses section. For ddsPLS, there appears to be little relation between Q^2 and $R^2 - Q^2$. Finding a good $R^2 - Q^2$ value doesn't guarantee good performance. In these cases it seems that using $R^2 - Q^2$ to select parameters risks building models that miss part of the structure of the data.