```
Python 3.10.10 (main, Mar 21 2023, 18:45:11) [GCC 11.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.7.0 -- An enhanced Interactive Python. Type '?' for help.
```

In [ ]:
```python
import pathlib

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [ ]:
```python
cwd = pathlib.Path(__file__).parent
df_results = pd.read_csv(cwd.parent / 'local/eval_results/experiment_temporal_features.csv', index_col=0)
df_results["scope_estimator"] = pd.Categorical(df_results["scope_estimator"])
df_results["cols_used"] = df_results["cols_used"].str.replace('ft_numc_', '').fillna('None used')
df_results["uses_scope_1"] = df_results["cols_used"].str.contains('scope_1')
df_results["uses_scope_2"] = df_results["cols_used"].str.contains('scope_2')
df_results["uses_scope_3"] = df_results["cols_used"].str.contains('scope_3')
df_results["uses_year"] = df_results["cols_used"].str.contains('ft_numd_year')
df_results["uses_none"] = df_results["cols_used"].str.contains('None')
df_results["clueless_impact_sMAPE"] = df_results["clueless.sMAPE"] - df_results["raw.sMAPE"]
df_results["clueless_impact_jump_rate25"] = df_results["jump_rates.normal.percentile25"] - df_results["jump_rates.clueless.per
df_results["clueless_impact_jump_rate50"] = df_results["jump_rates.normal.percentile50"] - df_results["jump_rates.clueless.per
df_results["clueless_impact_jump_rate75"] = df_results["jump_rates.normal.percentile75"] - df_results["jump_rates.clueless.per
df_results["clueless_impact_ex_sMAPE"] = df_results["clueless.sMAPE"] / df_results["raw.sMAPE"]

df_results
```

Out[ ]:

	repetition	cols_used	time	scope	imputer	fin_transformer	cat_transformer	sc
0	1	None used	96.939466	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
1	1	ft_numd_year	106.558607	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
2	1	prior_tg_numc_scope_1	110.165539	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
3	1	prior_tg_numc_scope_3	107.040075	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
4	1	prior_tg_numc_scope_2	102.557714	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
...	...	...	...	...	...	...	...	...
315	20	ft_numd_year|prior_tg_numc_scope_1|prior_tg_nu...	76.379832	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
316	20	ft_numd_year|prior_tg_numc_scope_1|prior_tg_nu...	69.837858	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
317	20	ft_numd_year|prior_tg_numc_scope_3|prior_tg_nu...	79.947642	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
318	20	prior_tg_numc_scope_1|prior_tg_numc_scope_3|pr...	79.999450	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	
319	20	ft_numd_year|prior_tg_numc_scope_1|prior_tg_nu...	72.880073	1	RevenueQuantileBucketImputer	PowerTransformer	TargetEncoder	

320 rows × 514 columns

Correlation Matrix

Shows: A correlation matrix as heatmap showing the linear relationships between the different metrics used in this analysis

Notes:

- *raw.offset_percentiles* are the rate to which the prediction is different from the true value. For instance, at raw.offset_percentile 50% *the median rate of being off from the true value is 1.XX*
- Jump rates are similar in intuition but computed in a completely different way. They represent the percentile in jump-rates of scopes by yearly transition after scope imputations. These yearly transitions are not cleaned by whether the transition years where reported or not. For instance, a mean jump-rate of 1.5 says that after scope imputations of our model on average the scope of any year jumps 50% compared to the previous year.

- The jump-rates are normalised. Hence a jump-rate of 1.5 could mean a year increased or decreased by 50%.

Observations:

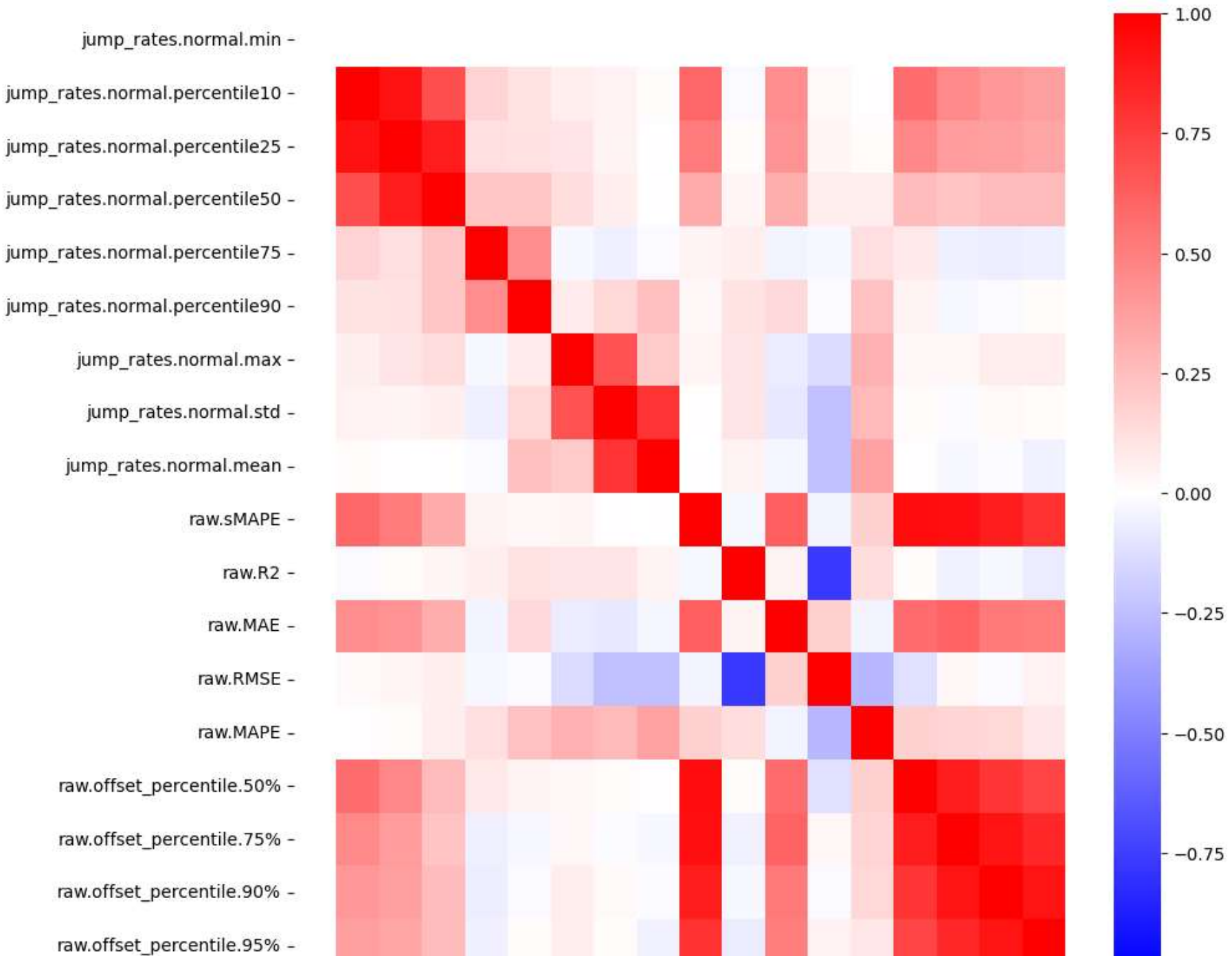
- sMAPE and MAE correlate with eachother. Hence, High sMAPE leads to high MAE
- sMAPE also correlates with jump rates in the lower percentiles but looses its influence for higher ones. In other words, where the maximum jump rate value lies is not dependent on how precise the model is in terms of sMAPE/MAE
- For R2 there seems to be the opposite relationship with the percentiles. Hence, the higher R2 the lower the max jump-rate. Ths relationship is weak though. Therefore, not very conclusive.
- The raw.offset percentiles have a correlation with the jump rate percentiles. A high jump-rate leads to high off-sets from the true value
- The minimum jump-rate is always 1. This is trivial, though.

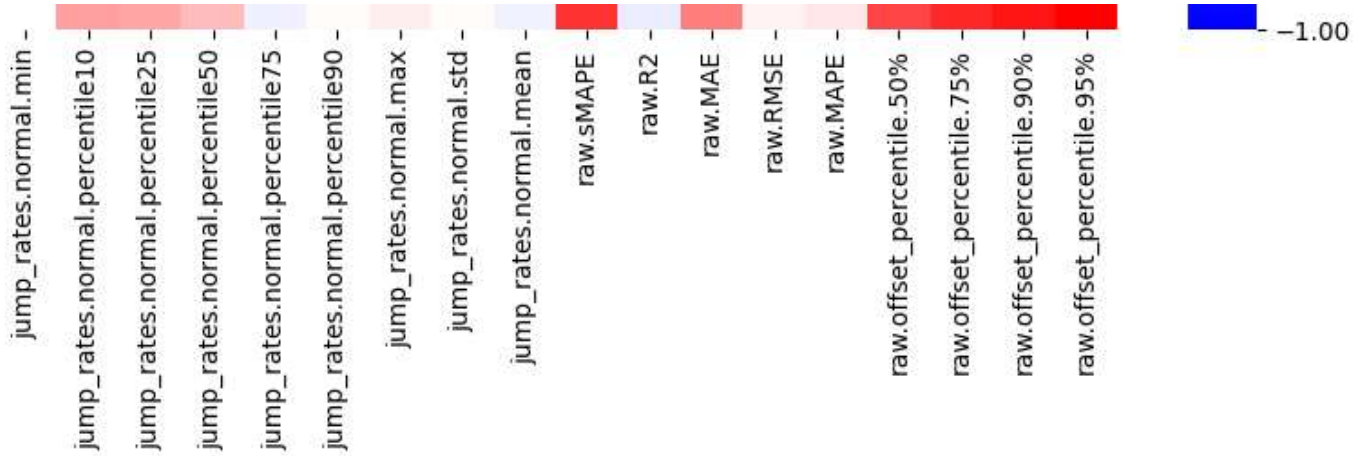
Interpretations:

- sMAPE and MAE can be interpreted interchangably.
- A relationship between the sMAPE metric and the jump-rate exists in the lower percentile region.
- The relationship between jump rates and offsets is important. It tells us that an improvement of our prediction offsets could lead to an improvement in jump rates. One could say the same about sMAPE/MAE, too. But only for the lower percentile regions of jump-rates.
- However, these relationships break down in higher percentile regions. Meaning, high yearly jumps do not appear to be an issue of prediction accuracy. Maybe those correlate with the lack of input data or the existence of missing values.

In [ ]:
```
fig = plt.figure(figsize=(10, 10))
corr_matrix = df_results.filter(regex="^(raw\.|jump_rates\.normal)").corr()
sns.heatmap(corr_matrix, cmap="bwr", vmax=1, vmin=-1)
plt.show()
```

```
<ipython-input-3-2ae8960ae2ff>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a futu
re version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  corr_matrix = df_results.filter(regex="^(raw\.|jump_rates\.normal)").corr()
```


jump_rates.normal.min
jump_rates.normal.percentile10
jump_rates.normal.percentile25
jump_rates.normal.percentile50
jump_rates.normal.percentile75
jump_rates.normal.percentile90
jump_rates.normal.max
jump_rates.normal.std
jump_rates.normal.mean
raw.sMAPE
raw.R2
raw.MAE
raw.RMSE
raw.MAPE
raw.offset_percentile.50%
raw.offset_percentile.75%
raw.offset_percentile.90%
raw.offset_percentile.95%
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



Effects of temporal hacks on sMAPE

Shows: Two bar plots showing the sMPAE in relationship to the temporal *hack* that was used to introduce time as a feature.

Notes

- The first plot shows the sMAPE on validation-data if the features remain present, and the second (*clueless*) if the features are not present.
- In the case of non-existing columns, they will be imputed using an imputation strategy.
- These are only results for *scope 1*. Doing the same for scope 2 and 3 would make the analysis more complex and time-consuming

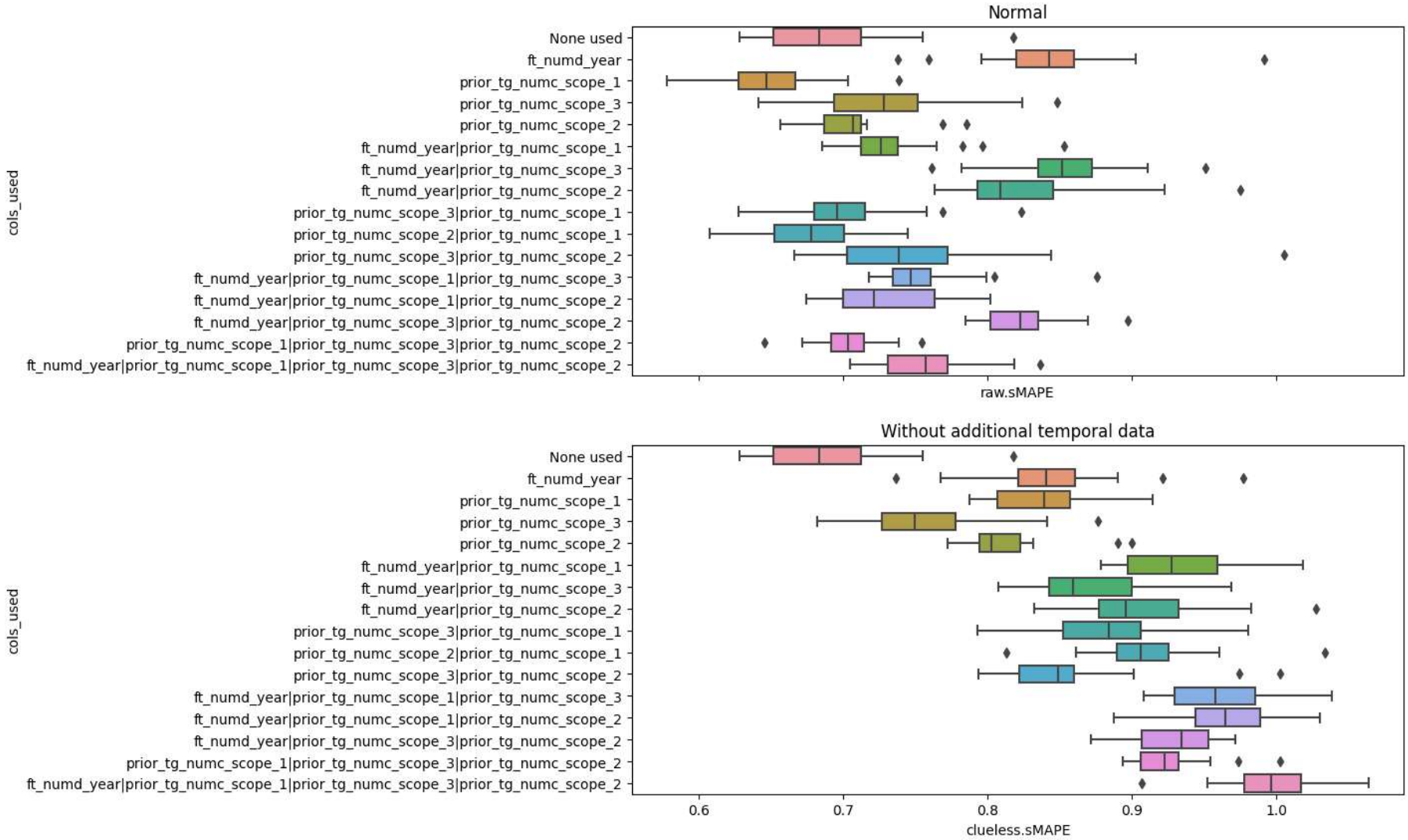
Observations

- Scope 1 sMAPE is lowest if the previous year was used as a feature during training.
- Introducing the year as feature always leads worse sMAPE results
- Introducing previous scope 2or 3 to predict scope 1 seems detrimental to the model.
- No usage of any hack does not impact the non-existence of the *hacky* features. That's quite trivial but an important sanity check. (Found a bug due to this earlier.)

Interpretations

- These results show that it might not be reasonable to introduce year as a feature.
- That previous scope 2 and scope 3 don't help improving scope 1 was expected.
- The use of previous scope 1 is not that clear, because the model does seem to over rely on that feature.

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10), sharex=True)

sns.boxplot(data=df_results, x="raw.sMAPE", y="cols_used", ax=ax1)
ax1.set_title('Normal')
sns.boxplot(data=df_results, x="clueless.sMAPE", y="cols_used", ax=ax2)
ax2.set_title('Without additional temporal data')
plt.show()
```



Impact on sMAPE if hacky features are omitted

Shows: Barplot which displays the clueness-less impact on the model

Notes

- Clueness-less means that the model was trained using a hack but during inference had no access to that feature. Hence, the model was forced to use an imputation strategy for the whole validation set.
- Lower number means lesser impact. More robust model.

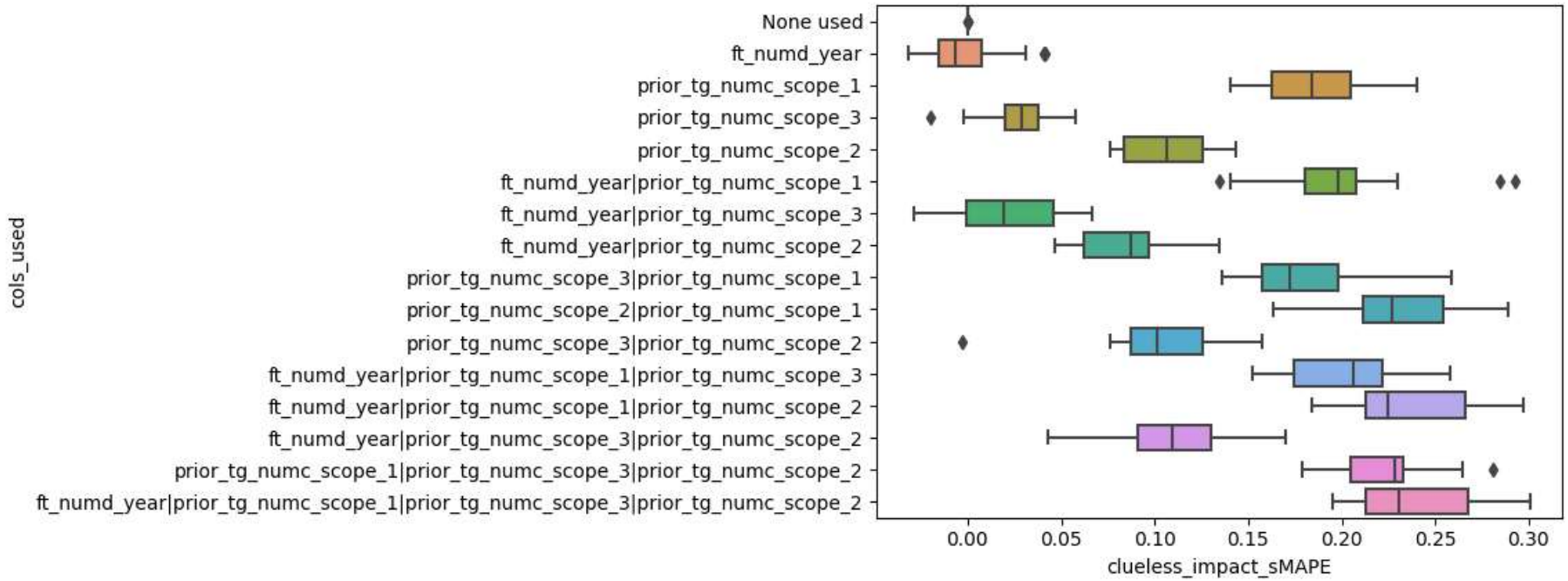
Observations

- If no hack is used, there is no impact if the features are missing. Also there's no variation. Again, trivial but important sanity check.
- Impact is also very low for using year as a feature. This time with variation. Meaning sometimes, the model becomes even better if trained with year as a feature and then omitting it.
- High impact for the prev-scope-1 column. Model loses around .20 in sMAPE without the feature.

Interpretations

The results of the year-feature are even more conclusive here. If the model is trained on year and then sometimes becomes better omitting it, the year is decisively bad for the model. The observation, that omitting the feature prev-scope-1 has such a high impact shows how much the model relies on the feature. It also shows how important previous years are. But this importance cannot be captured by just adding year.

In [ ]:
```
sns.boxplot(data=df_results, x="clueless_impact_sMAPE", y="cols_used")
plt.show()
```



Effects of temporal hacks on jump rates

Shows: Two bar plots showing the jump rates in relationship to the temporal *hack* that was used to introduce time as a feature.

Notes

- The plot shows the 50th percentile.

Observations

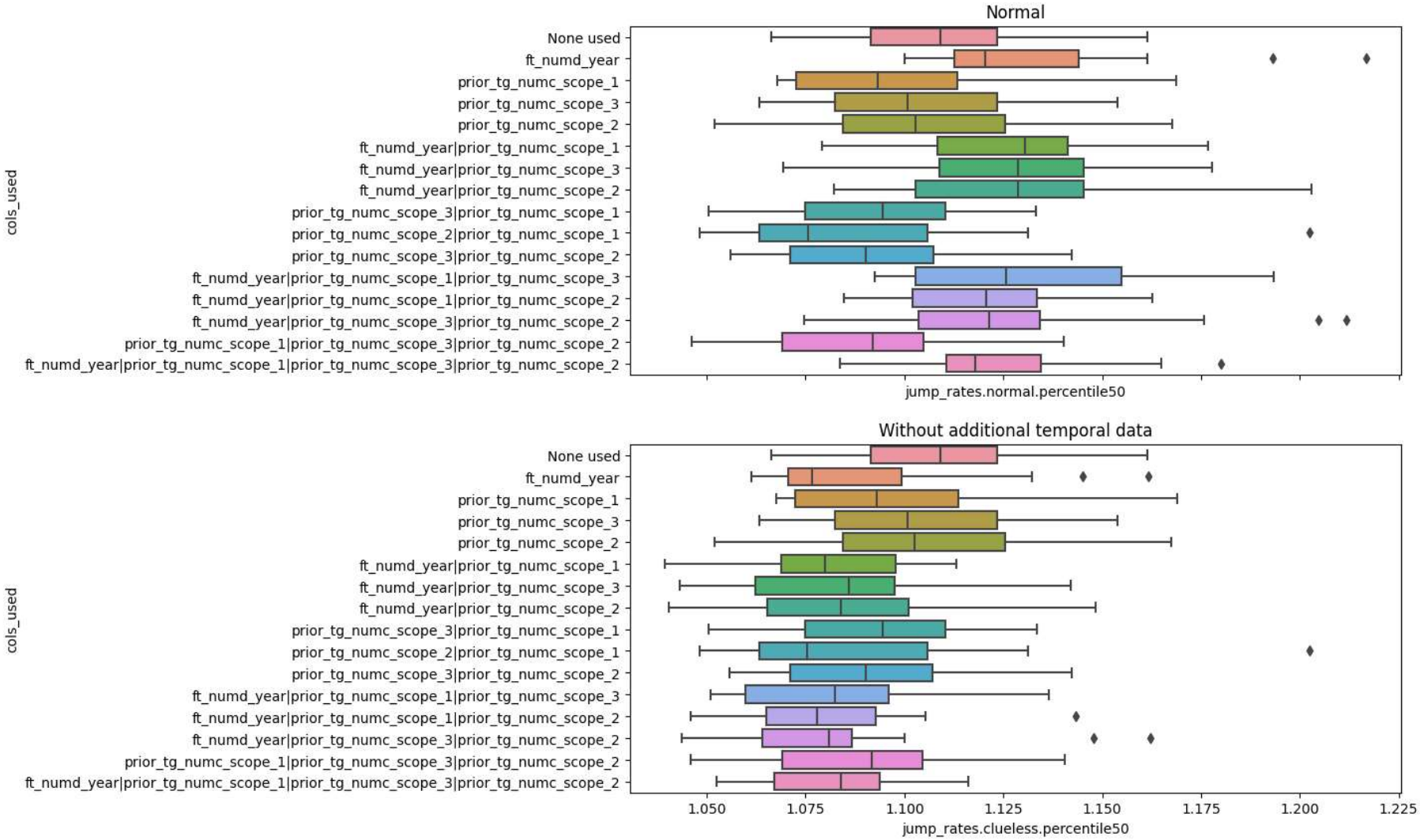
- Using the prev-scope-1 again improves on the jump-rates
- Year is again detrimental to the jump-rates
- Best improvement on jump-rates is when using prev-scope-2 and prev-scope-1 together.
- In the clueless case, introducing year has a stabilizing effect on the jump-rate spread.

- In the clueless case, training on any hack improves the jump rates. In other words, using a hack improves jump-rates, if the hack features are omitted. Weird...

Interpretations

- These results are hard to interpret.

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10), sharex=True)
sns.boxplot(data=df_results, x="jump_rates.normal.percentile50", y="cols_used", ax=ax1)
ax1.set_title('Normal')
sns.boxplot(data=df_results, x="jump_rates.clueless.percentile50", y="cols_used", ax=ax2)
ax2.set_title('Without additional temporal data')
plt.show()
```



Impact on jump rates if hacky features are omitted

Shows: Barplot which displays the clueness-less impact on the model

Notes

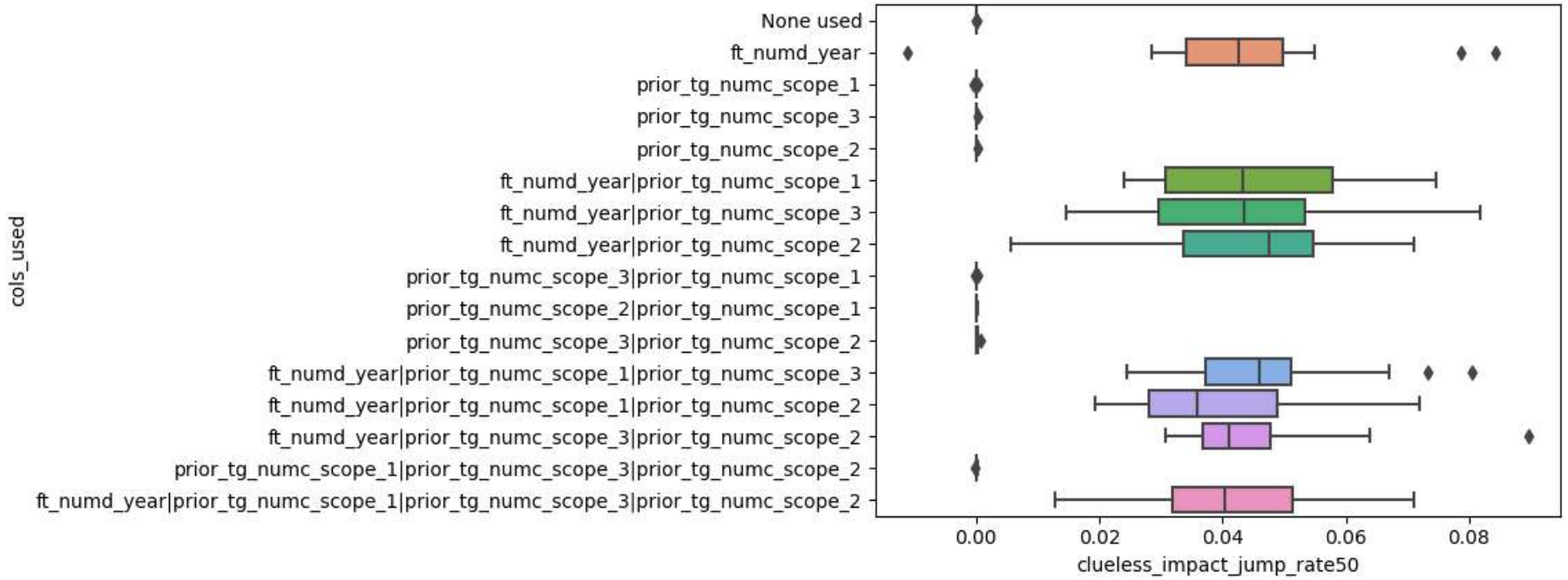
Observations

- In many cases omitting the hacky feature does not have an effect on the jump-rate unless year was involved.

Interpretations

- Need to reevaluate in light of the previous results

In [ ]:
```
sns.boxplot(data=df_results, x="clueless_impact_jump_rate50", y="cols_used")
plt.show()
```



Relationship between jump-rate and sMAPE

Shows: Scatterplot

Notes

This plot uses jump-rates of percentile 25. For, higher jump-rate percentile regions the relationship vanishes.

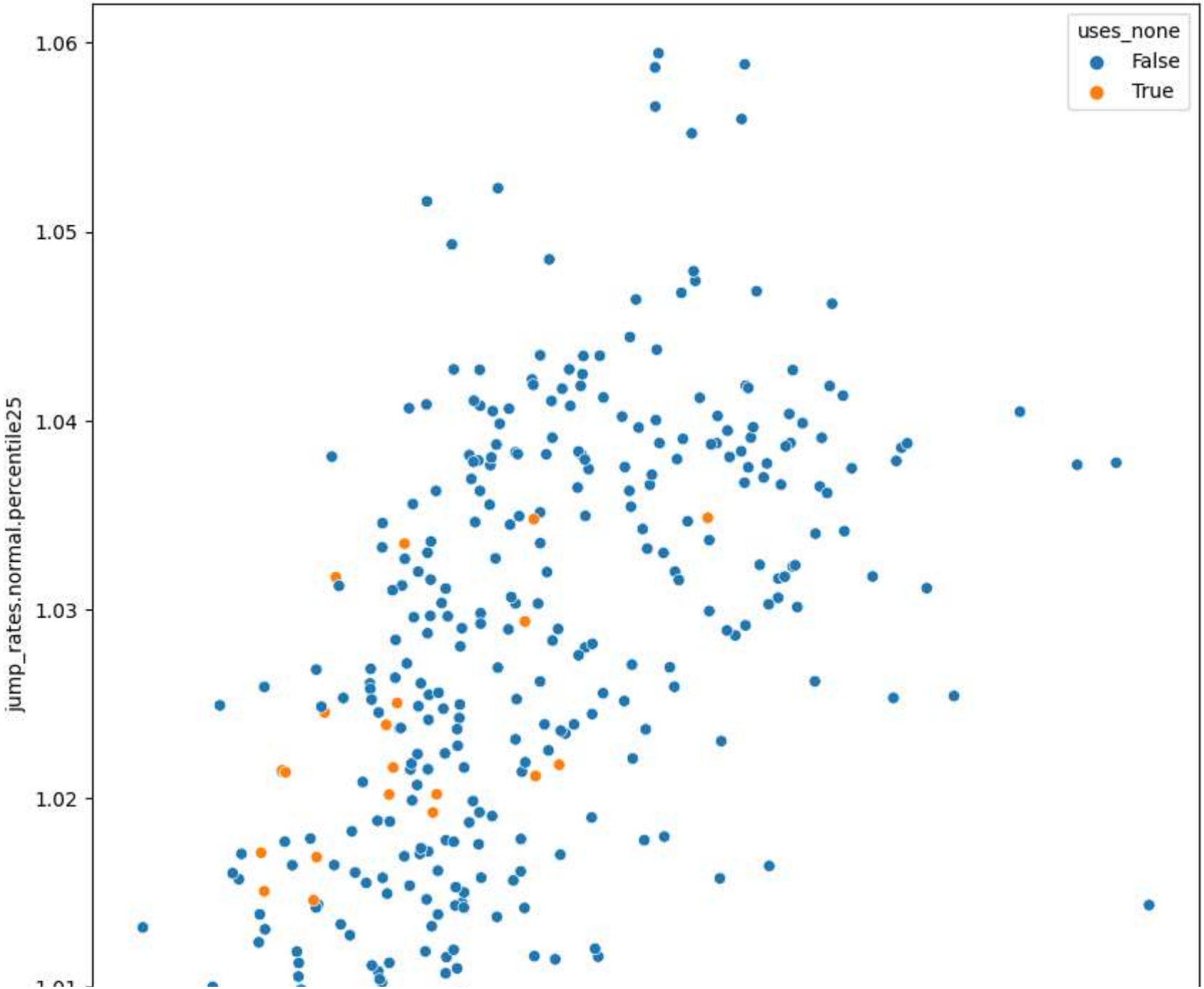
Observations

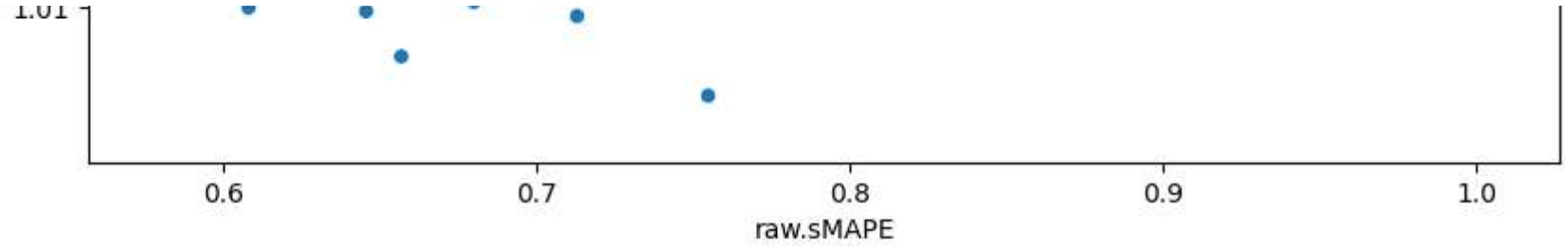
- Clear relationship between sMAPE and jump-rates on percentile 25

Interpretations

If we improve the sMAPE we might improve

In [ ]:
```
fig = plt.figure(figsize=(10, 10))
sns.scatterplot(data=df_results, y="jump_rates.normal.percentile25", x="raw.sMAPE", hue="uses_none")
plt.show()
```


uses_none
False
True
jump_rates.normal.percentile25
1.06
1.05
1.04
1.03
1.02



Marginal effect of the hacks

Shows: Bar plots of the marginal effect of introducing a temporal hack

Notes

- The next few plots all follow the same principal: Showing the effects of introducing a certain hack even if other hacks where also involved.Fon instance, if prev-scope-1 is the focus, all combinations with prev-scope-1 (prev-scope-1 with year, prev-scope-1 with prev-scope-2, etc.) are included.

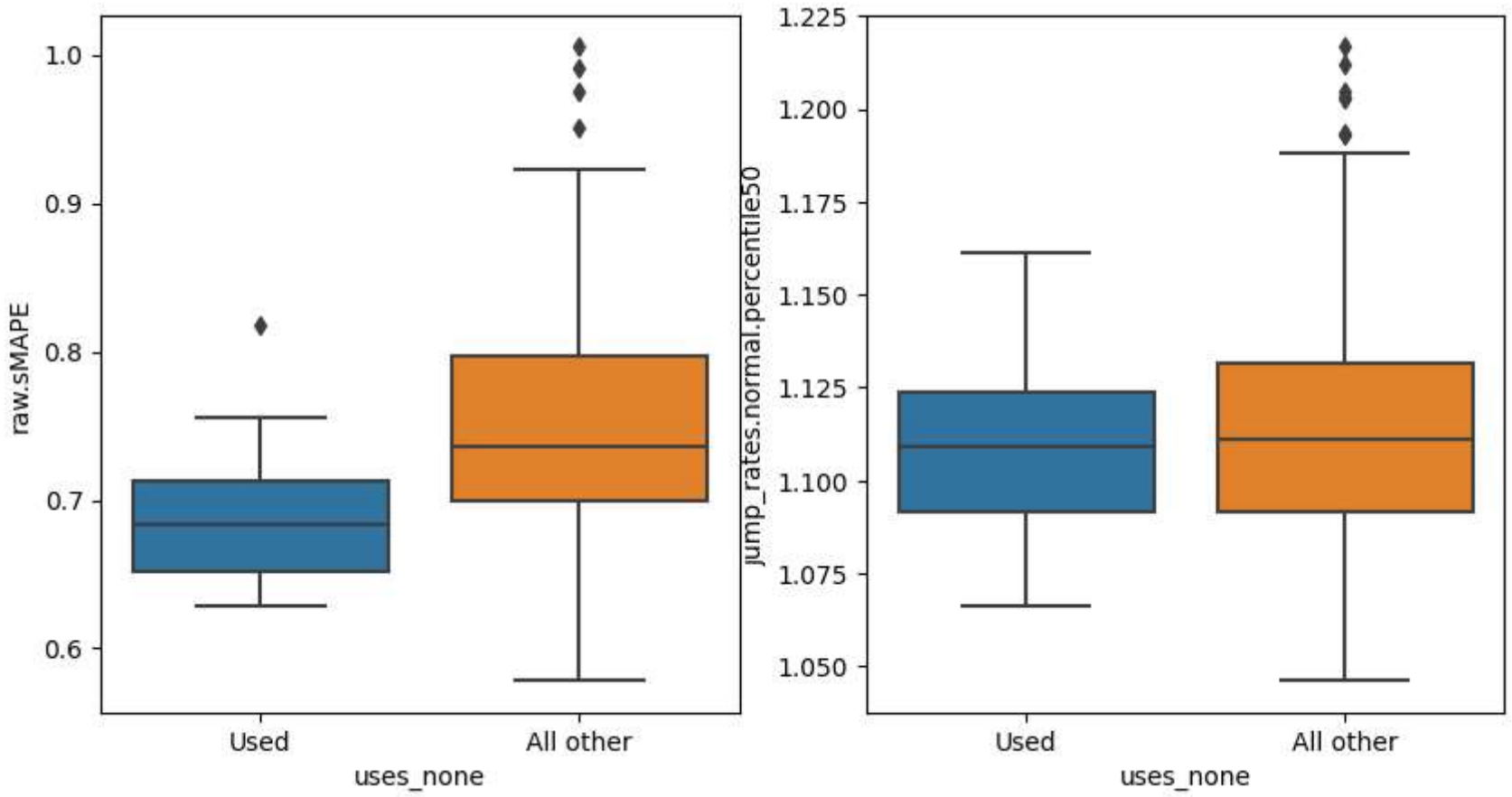
Observations

- Using no hack leads to a narrow range of sMAPE and jump-rates. Trivial result.
- Involving year leads to worse sMAPE and jump-rates
- Involving prev-scope-1 leads to strong improvements for sMAPE but small in terms of jump-rates
- Similar effects with prev-scope-2 as with prev-scope-1 but much much weaker
- Involvement of prev-scope-3 makes sMAPE worse but still improves a little on jump-rates

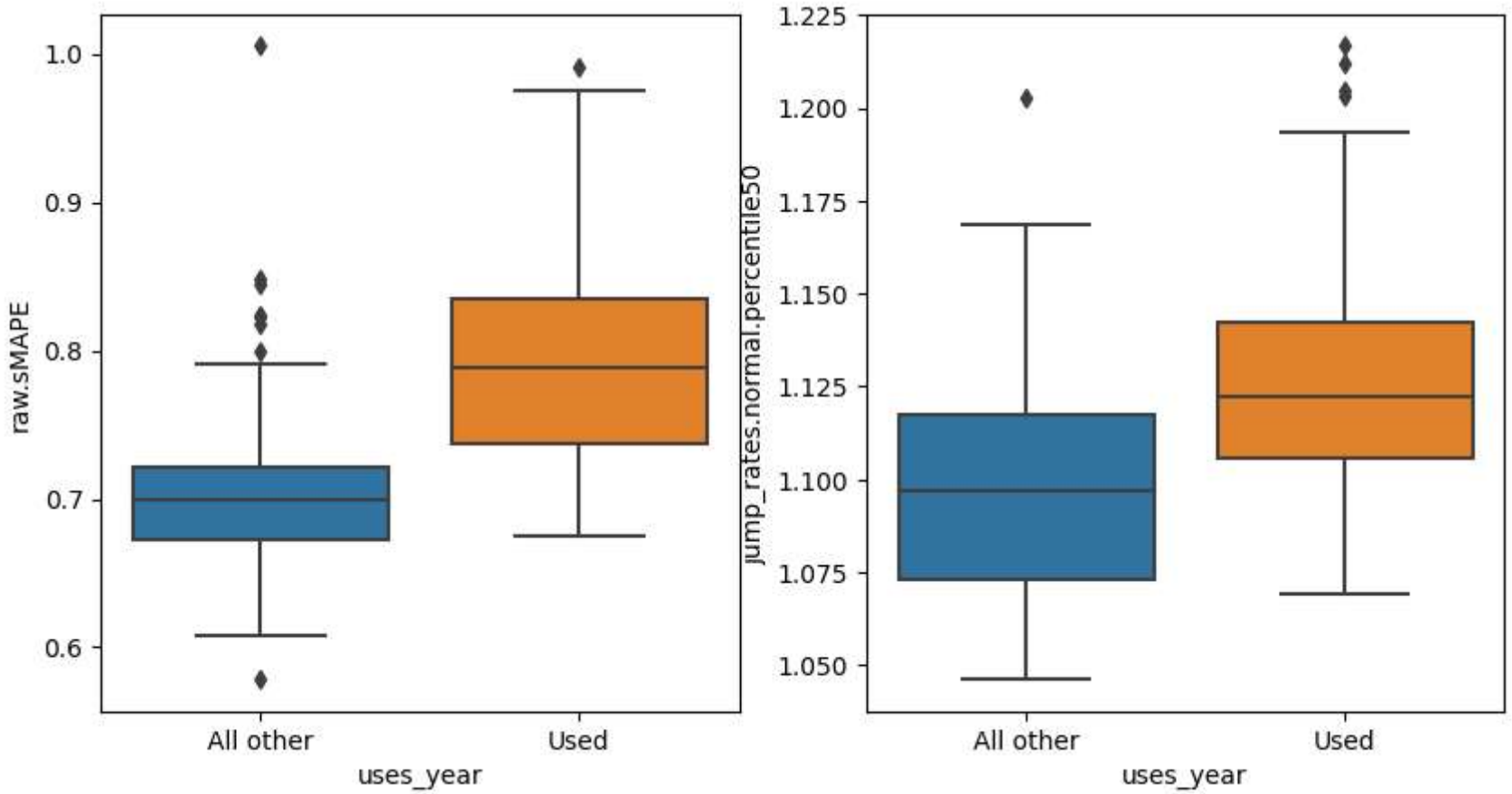
Interpretations

Many of the results were already drawn beforehand. However, one additional insight is given by the consistent improvement of jump-rates for all three prev-scope hacks. Cannot think of an explanation.

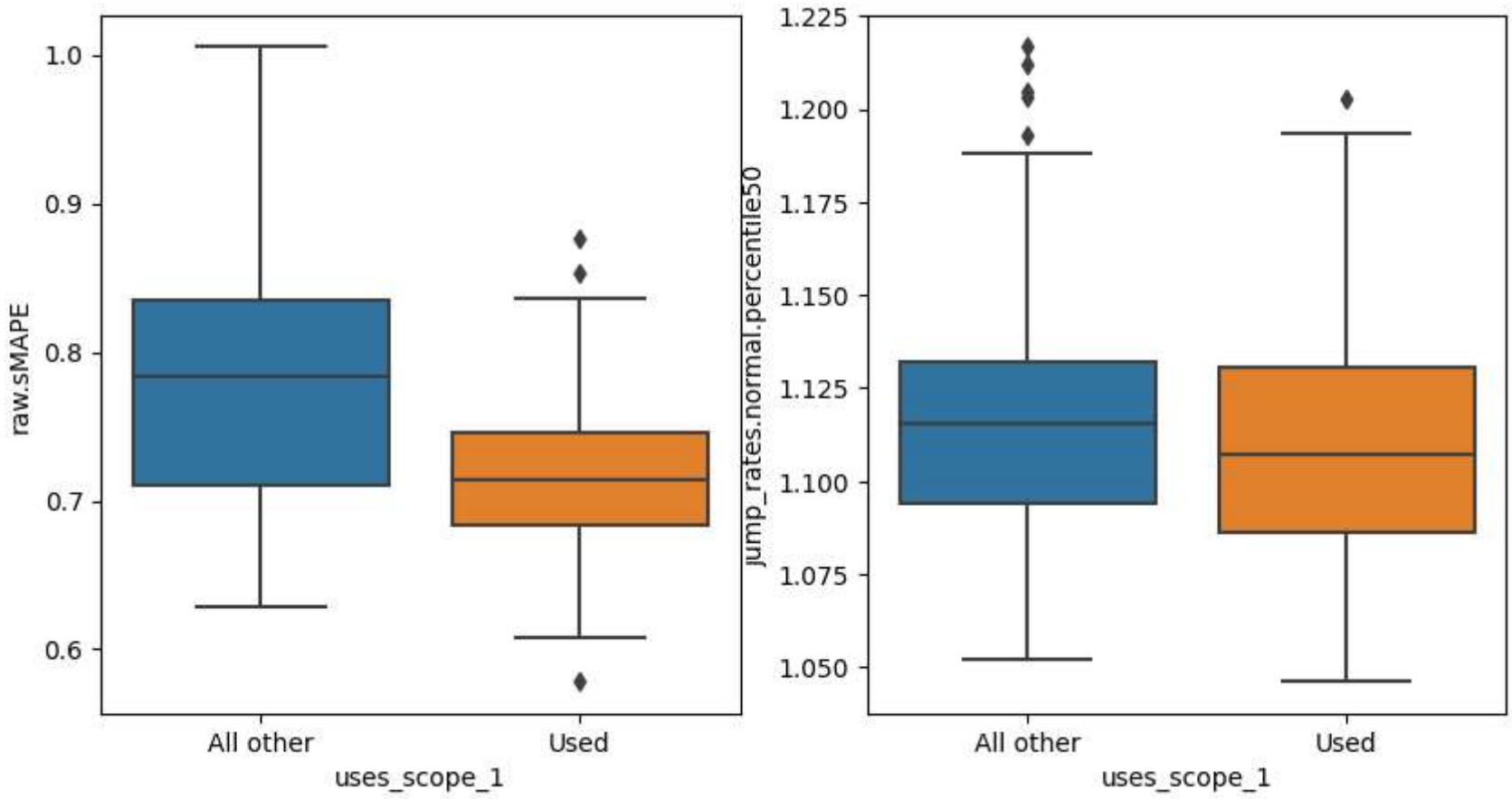
In [ ]:
```
col = "uses_none"
def plot_isolated_effect(df_results:pd.DataFrame, col):
    df = df_results.copy()
    df[col] = df[col].replace({True:"Used", False:"All other"})
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5), sharex=True)
    sns.boxplot(data=df, y="raw.sMAPE", x=col, ax=ax1)
    sns.boxplot(data=df, y="jump_rates.normal.percentile50", x=col, ax=ax2)

    plt.show()

plot_isolated_effect(df_results, "uses_none")
```



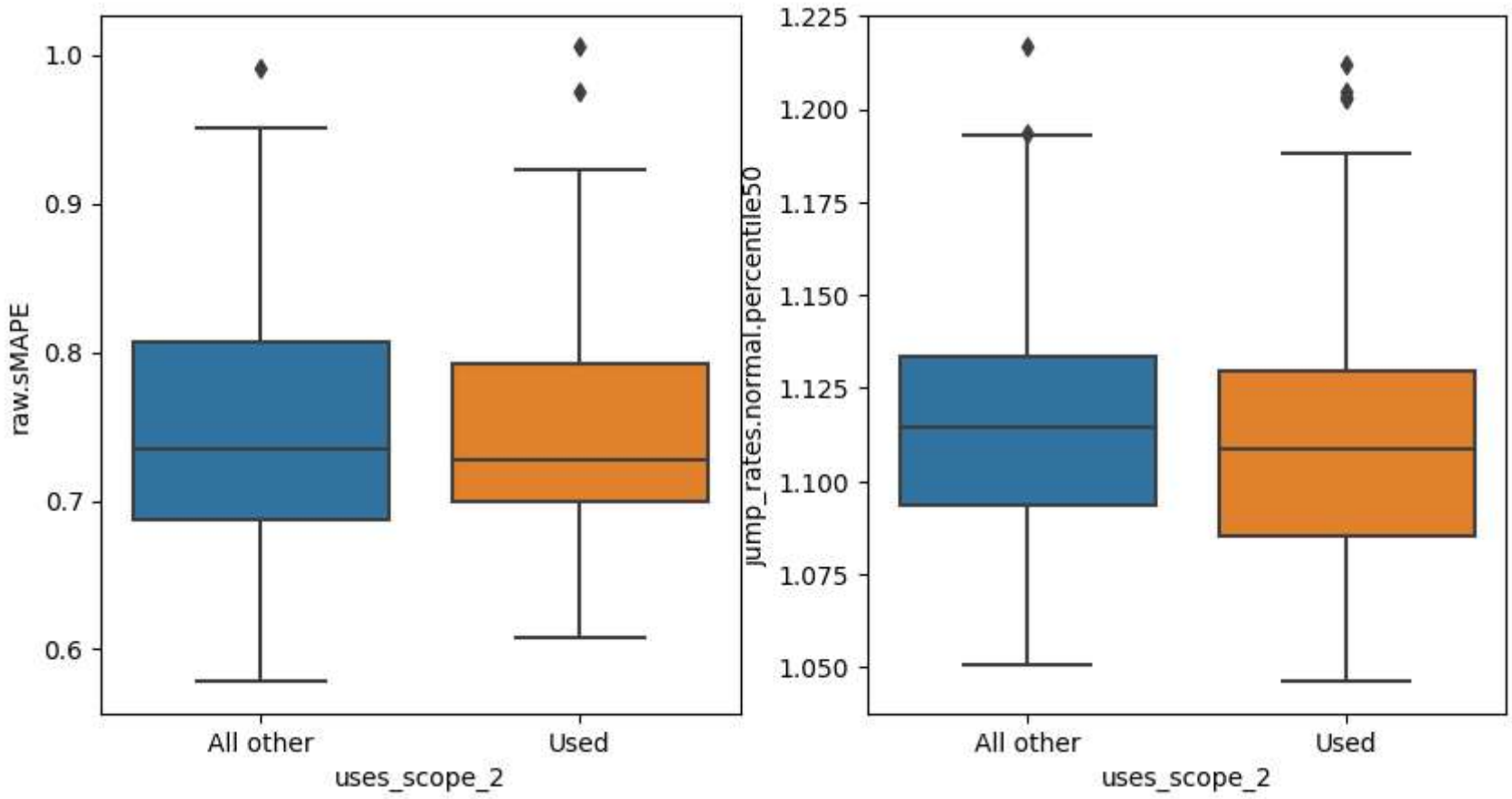
In [ ]:
```
plot_isolated_effect(df_results, "uses_year")
```



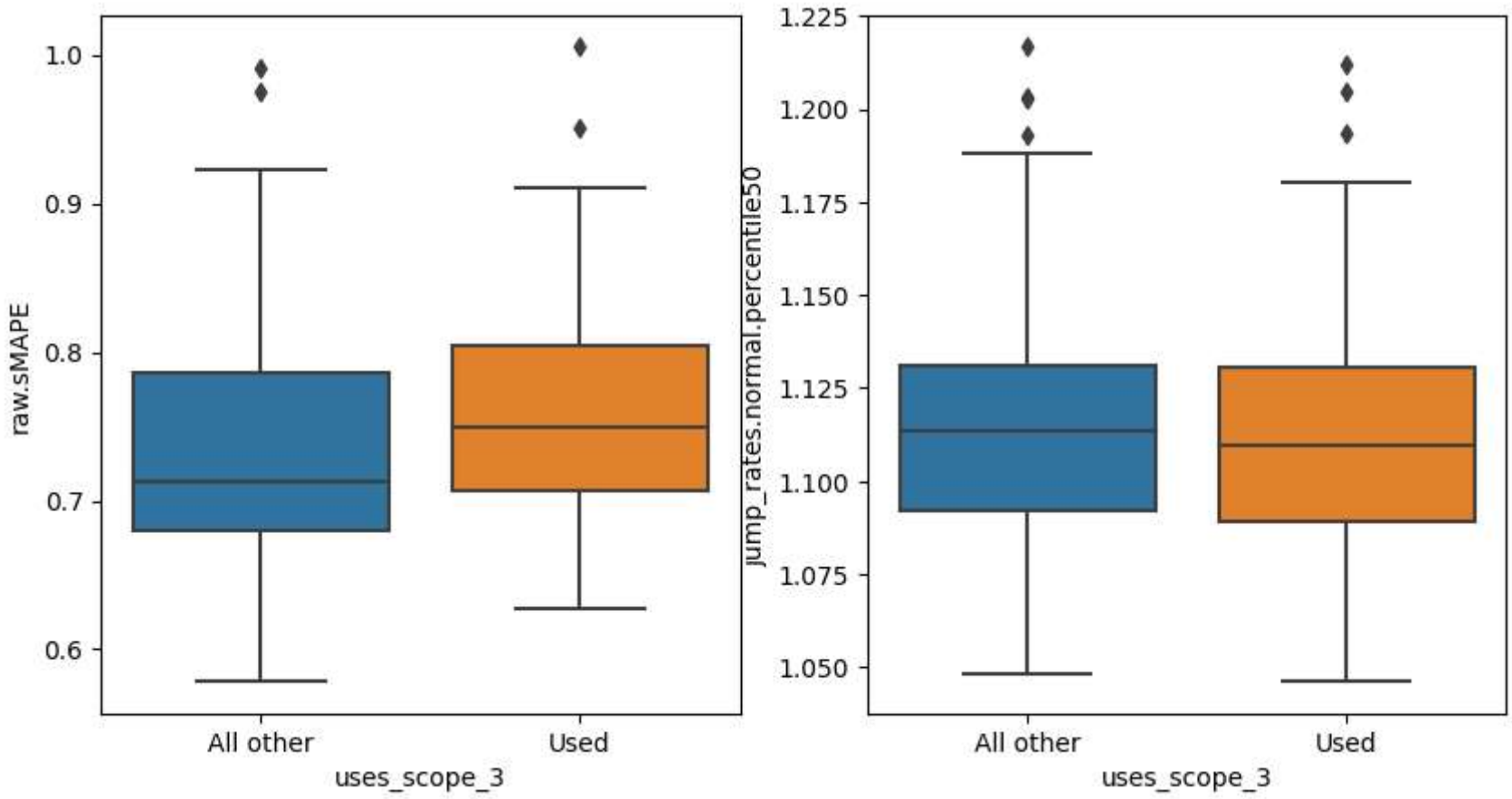
In [ ]:
```
plot_isolated_effect(df_results, "uses_scope_1")
```



In [ ]:
```
plot_isolated_effect(df_results, "uses_scope_2")
```



In [ ]:
```
plot_isolated_effect(df_results, "uses_scope_3")
```


raw.sMAPE
1.0
0.9
0.8
0.7
0.6
All other
Used
uses_scope_3
jump_rates.normal.percentile50
1.225
1.200
1.175
1.150
1.125
1.100
1.075
1.050
All other
Used
uses_scope_3