Computing PAI with resampling methods:

In Linear Regression:

In [ ]:

```
import numpy as np
import pandas as pd
```

Old Data

In [ ]:

```
np.random.seed(123)

# Quantitative Response

Y_old = np.random.normal(loc=50, scale=10, size=500)

# Quantitative variables

X1_old = np.random.normal(loc=30, scale=25, size=500)

# Binary variables

X2_old = np.random.uniform(low=0.0, high=1.0, size=500).round()

# Multiclass categorical variables

X3_old = np.random.uniform(low=0, high=4, size=500).round()   # categories: 0,1,2,3,4
```

New Data (with a big change in X1 distribution)

In [ ]:

```
np.random.seed(666)

# Quantitative Response

Y_new = np.random.normal(loc=50, scale=10, size=500)

# Quantitative variables

X1_new = np.random.normal(loc=15, scale=60, size=500)

# Binary variables

X2_new = np.random.uniform(low=0.0, high=1.0, size=500).round()

# Multiclass categorical variables

X3_new = np.random.uniform(low=0, high=4, size=500).round()   # categories: 0,1,2,3,4
```

In [ ]:

```
df_Old = pd.DataFrame( {"Y":Y_old , "X1": X1_old , "X2": X2_old , "X3": X3_old} )

df_New = pd.DataFrame( {"Y":Y_new , "X1": X1_new , "X2": X2_new , "X3": X3_new} )
```

In [ ]:

```
from plotnine import ggplot, aes, geom_line, geom_point, geom_histogram, geom_bar, geom_boxplot, scale_y_continuous, scale_x_continuous, labs, after_stat,  geom_vline, scale_color_m
from mizani.formatters import percent_format
```

In [ ]:

```
import array as arr

df_Old_New = pd.concat([df_Old , df_New])

repeat_Old = ['Old Data']*len(df_Old)

repeat_New = ['New Dta']*len(df_New)

df_repeat_New = pd.DataFrame( {"group": repeat_New} )

df_repeat_Old = pd.DataFrame( {"group": repeat_Old} )

groups = pd.concat([df_repeat_Old , df_repeat_New])

df_Old_New_groups = pd.concat([df_Old_New , groups], axis=1 )

df_Old_New_groups
```

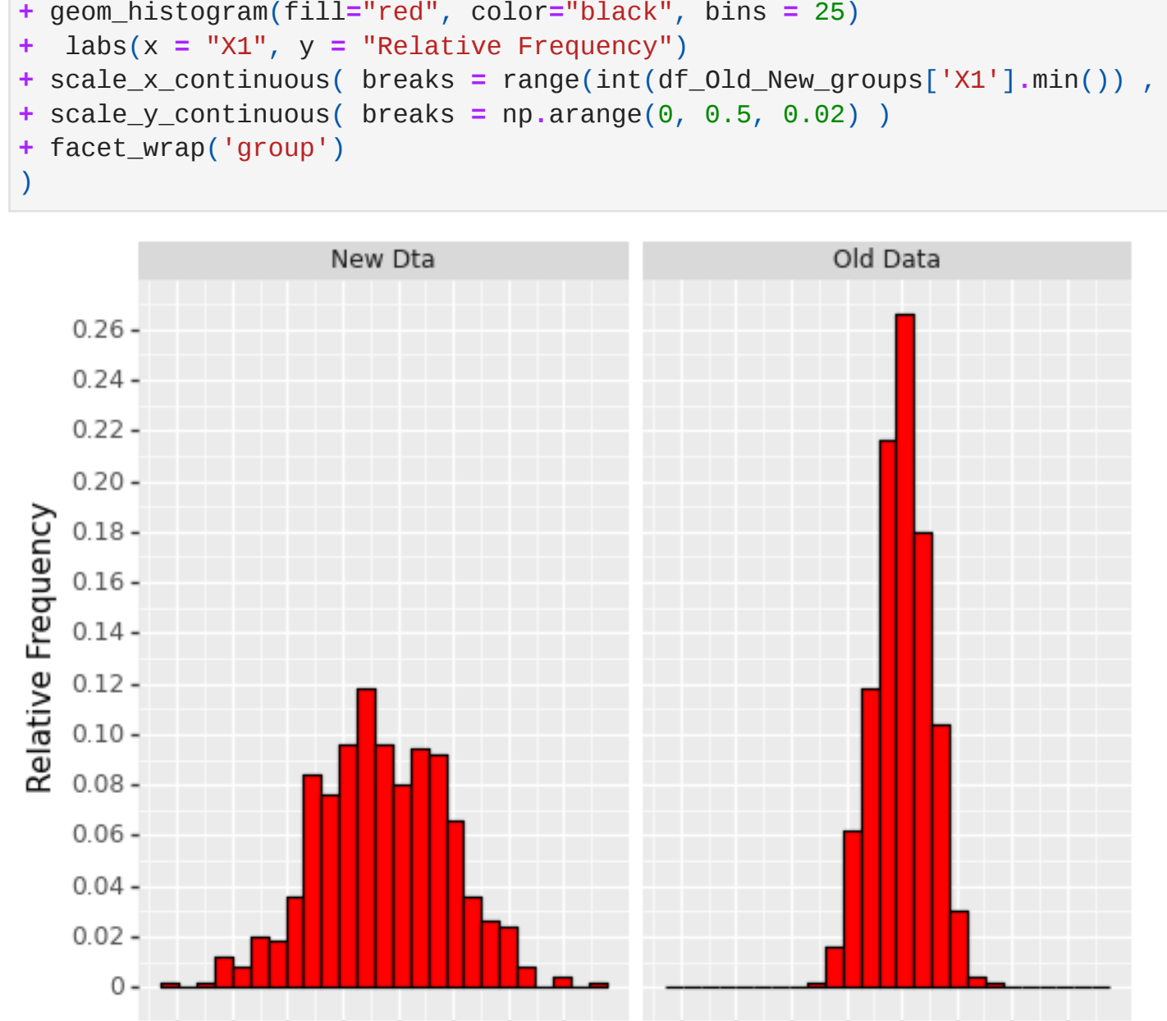
Out[ ]:

	Y	X1	X2	X3	group
0	39.143694	48.800842	1.0	1.0	Old Data
1	59.973454	31.741019	0.0	3.0	Old Data
2	52.829785	23.363859	0.0	4.0	Old Data
3	34.937053	53.239612	1.0	1.0	Old Data
4	44.213997	61.520532	1.0	3.0	Old Data
...	...	...	...	...	...
495	56.756956	12.495333	0.0	3.0	New Dta
496	36.846327	75.862895	1.0	1.0	New Dta
497	36.528625	-21.430239	0.0	0.0	New Dta
498	62.143112	-56.088634	0.0	2.0	New Dta
499	53.433602	10.075296	0.0	3.0	New Dta

1000 rows × 5 columns

In [ ]:

```
(
ggplot( df_Old_New_groups )
+  aes(x='X1' , y =  after_stat('width*density'))
+ geom_histogram(fill="red", color="black", bins = 25)
+  labs(x = "X1", y = "Relative Frequency")
+ scale_x_continuous( breaks = range(int(df_Old_New_groups['X1'].min()) , int(df_Old_New_groups['X1'].max()) , 50) )
+ scale_y_continuous( breaks = np.arange(0, 0.5, 0.02) )
+ facet_wrap('group')
)
```



Out[ ]:

```
<ggplot: (144229958650)>
```

We are going to consider the following definition of PAI (instead of use the variance, we will use the standard deviation. if we would consider the PAI definition with the variance, the process to compute it would have been very similar)

Where:

The numerator is computing by resampling methods using the model trained with the Old Data (for the response and the predictors) but predicting the response variable using the New Data for the predictors.

The denominator is computing by resampling methods using the model trained Old Data (for the response and the predictors) and also predicting the response variable using the Old Data for the predictors.

$$PAI = \frac{\frac{1}{N}\sum_{i \in NewData} \widehat{Var}(\hat{y}_i)}{\frac{1}{n}\sum_{i \in OldData} \widehat{Var}(\hat{y}_i)}$$

So, PAI is the quotient between the mean variance of the response predictions, using the model trained with the Old data but using the New data for the predictors to get the response predictions and the mean variance of the response predictions, using the model trained with the Old data and also using the Old data for the predictors to get the response predictions

In [ ]:

```
def varcharProcessing(X, varchar_process = "dummy_dropfirst"):

    dtypes = X.dtypes

    if varchar_process == "drop":
        X = X.drop(columns = dtypes[dtypes == np.object].index.tolist())

    elif varchar_process == "dummy":
        X = pd.get_dummies(X,drop_first=False)

    elif varchar_process == "dummy_dropfirst":
        X = pd.get_dummies(X,drop_first=True)

    else:
        X = pd.get_dummies(X,drop_first=True)

    X["intercept"] = 1
    cols = X.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    X = X[cols]

    return X
```

In [ ]:

```
df_Old['X2'] = df_Old['X2'].astype('category')
df_Old['X3'] = df_Old['X3'].astype('category')
```

In [ ]:

```
df_Old
```

Out[ ]:

	Y	X1	X2	X3
0	39.143694	48.800842	1.0	1.0
1	59.973454	31.741019	0.0	3.0
2	52.829785	23.363859	0.0	4.0
3	34.937053	53.239612	1.0	1.0
4	44.213997	61.520532	1.0	3.0
...	...	...	...	...
495	55.678801	45.869078	1.0	2.0
496	55.129828	56.747965	1.0	2.0
497	49.730774	7.266825	1.0	0.0
498	53.115815	41.756594	0.0	1.0
499	48.579493	2.214239	1.0	3.0

500 rows × 4 columns

In [ ]:

```
from sklearn.linear_model import LinearRegression
```

In [ ]:

```
B=100

y_predictions_Old_Data = np.zeros((B , len(df_Old)))

for i in range(0, B):

    df_Old_BOOT_SAMPLE = df_Old.sample( n=len(df_Old) , random_state=i , replace=True )    # i-th boot sample

    X_old_Boot_Sample = df_Old_BOOT_SAMPLE[['X1', 'X2', 'X3']]

    y_old_Boot_Sample = df_Old_BOOT_SAMPLE['Y']

    X_old_Boot_Sample = varcharProcessing(X_old_Boot_Sample, varchar_process = "dummy_dropfirst")

    # We train the model with i-th boot sample of the Old Data:

    Model_train_Old_data = LinearRegression().fit(X_old_Boot_Sample, y_old_Boot_Sample)

    # y predictions using Model_train_Old_data with the original Old Data for the predictors

    X_old = df_Old[['X1', 'X2', 'X3']]

    X_old = varcharProcessing(X_old, varchar_process = "dummy_dropfirst")

    # La idea es que con cada iteracion el modelo cambia (los parametros) ya que ha sido entrenado con diferentes data set (las muestras bootstrap)
    # Pero las predicciones se hacen usando siempre el mismo data set (Old Data), para asi asegurar que \hat{y}_i es siempre la prediccion de la respuesta
    # para el i-esimo individuo (cambiará porque al re-entrenar el modelo con las distintas muestras boot cambian los parametros de este, pero no cambia
    # el vector x_i de valores de los predictores del individuo i, con los que tambien se genera la prediccion)

    # Por tanto si hubiese mucha variabilidad entre los valores obtenidos de \hat{y}_i = \hat{\beta} x_i esto se deberia no a cambios en x_i (puesto que
    # en este programa no cambia), si no a cambios en \hat{\beta} debidos a las diferentes muestras boot usadas para entrenar el modelo.
    # Si en lugar del modelo de regresion lineal multiple usasemos otro de la misma indole (basicamente un modelo que pueda ser entrenado con unos datos
    # y pueda realizar predicciones sobre una respuesta usando otros datos), la idea seria la misma.


    y_predictions_Old_Data[i, :] = Model_train_Old_data.predict(X_old)
```

The (r,k) element of the Bxn matrix `y_predictions_Old_Data` is $\widehat{y_k}$ (the Y estimation for the k-th individual of the sample) when the model is trained with the r-th boot sample of `Old_Data_Set`

In other words:

$$y\_predictions\_Old\_Data = \begin{pmatrix} \widehat{y_1}^1 & \widehat{y_2}^1 & \dots & \widehat{y_n}^1 \\ \widehat{y_1}^2 & \widehat{y_2}^2 & \dots & \widehat{y_n}^2 \\ \dots & \dots & \dots & \dots \\ \widehat{y_1}^B & \widehat{y_2}^B & \dots & \widehat{y_n}^B \end{pmatrix}$$

Where:

$\widehat{y_k}^r$ is the response prediction for the k-th element of the original Old_Data_Set, obtained using the model trained with the r-th boot sample of Old_Data_Set

$n =$ len(Old_Data_Set)

$B =$ nº of boot samples

In [ ]:

```
y_predictions_Old_Data
```

Out[ ]:

```
array([[49.42508598, 50.11472721, 48.97935105, ..., 50.27502977,
         51.17601889, 47.40202062],
        [48.87265439, 50.14696569, 49.86794534, ..., 49.2737149 ,
         49.40472179, 49.47626932],
        [48.66018825, 50.95861995, 48.82043962, ..., 50.59214347,
         48.66699903, 49.84584848],
        ...,
        [49.19061918, 51.17108553, 50.27997571, ..., 49.4269939 ,
         49.36788126, 51.11558309],
        [48.43534486, 50.99945593, 50.51160695, ..., 50.99576546,
         47.97549545, 51.38434772],
        [49.00485734, 50.35848299, 50.47602223, ..., 49.57213947,
         50.16560803, 49.00704667]])
```

We compute the standard deviation of each column of the matrix , and we get an estimation of $Var(\hat{y}_i)$ for $i = 1, \dots, n$:

So, the i-th value of the following vector is

$$\widehat{Var}(\hat{y}_i)$$

In [ ]:

```
 # compute the variance by cols in an array

y_predictions_Old_Data.var(axis=0)
```

Out[ ]:

```
array([2.3736694 , 1.83531851, 1.49999452, 2.02032802, 2.03471412,
       2.59662433, 2.32653866, 2.10161238, 2.51028022, 2.48036944,
       1.62799249, 1.81614304, 2.08080234, 2.46520527, 1.90376617,
       1.39392336, 2.15432888, 2.81813501, 1.87711112, 1.85249005,
       1.52218726, 2.01087683, 2.40951067, 2.01183162, 1.66672321,
       1.94795425, 1.9963901 , 2.14494738, 2.13831145, 2.2681182 ,
       1.93516707, 2.03019929, 1.86845261, 1.88852273, 1.72905494,
       2.29470579, 2.23117035, 2.21600616, 2.52484136, 2.00556638,
       2.22137462, 1.90636041, 1.95191852, 2.20370171, 1.99852729,
       2.45884646, 1.65150188, 1.86239492, 2.05955444, 2.08133178,
       2.09107018, 2.09175392, 2.06124553, 2.37485167, 1.88766883,
       2.49074997, 2.03945508, 2.0099999 , 2.41058172, 2.5622215 ,
       2.13638402, 1.41088064, 1.94105678, 2.08061339, 2.22737029,
       1.7952498 , 2.36554675, 1.88416718, 2.11025968, 2.34593621,
       2.43984327, 2.30048053, 1.673033  , 2.0588727 , 1.88850036,
       1.84127261, 1.79411597, 2.04758911, 1.44696274, 2.19625116,
       2.15356721, 2.43663391, 1.84033922, 2.06567795, 1.90737313,
       2.16427552, 2.0374608 , 2.13522344, 2.32175074, 1.57980151,
       1.72416516, 2.01295333, 2.30454453, 2.15839882, 2.03820604,
       2.28739972, 1.59712926, 2.37949883, 2.34071099, 1.86147695,
       2.12172721, 2.14539257, 1.94091567, 1.78402293, 2.46771354,
       1.99270509, 1.53240014, 2.33558246, 2.07521755, 2.55004932,
       1.65392279, 2.12119825, 2.28159141, 1.65037548, 2.31523818,
       2.13680489, 1.87511976, 1.75703361, 2.0454583 , 1.53014661,
       2.13877468, 2.54344998, 1.58554634, 2.33854072, 1.67445219,
       2.02464584, 2.3964662 , 2.05904698, 2.08089888, 2.44399203,
       2.61004032, 1.50477361, 2.53486839, 2.53077422, 2.13594314,
       2.23053751, 1.92258065, 1.65479316, 2.43723677, 1.76171786,
       1.87181612, 2.85410433, 1.80040757, 1.9774813 , 2.50829495,
       1.82073469, 1.35377844, 1.64070325, 1.9955039 , 2.14942544,
       1.9044506 , 2.33536142, 1.99318704, 2.0761117 , 1.76030416,
       2.22150564, 1.70504076, 2.62562896, 1.50452379, 1.75079702,
       2.19909294, 2.03393218, 2.23558377, 2.44166098, 2.21683168,
       2.11363514, 2.25179509, 2.10235741, 1.82524657, 2.14479637,
       2.06238252, 1.99094124, 2.27144346, 1.93958067, 2.04749002,
       1.92035156, 2.04206279, 2.41948644, 1.92003609, 2.12542531,
       2.04362215, 2.13274738, 1.94066355, 2.28056378, 2.20246062,
       2.09574509, 2.14728063, 1.79952586, 1.90968647, 2.30099991,
       2.08461461, 2.06745162, 1.70799634, 1.58290215, 2.16830128,
       2.71768198, 1.88706053, 1.97205293, 1.54657729, 1.99446846,
       2.26345717, 1.82202836, 1.89404385, 1.87438696, 2.09730691,
       2.34628035, 2.47698091, 2.0823044 , 2.27619031, 2.36119729,
       2.13013124, 1.81366296, 1.57515908, 2.38465154, 1.99567066,
       1.89722962, 1.99436165, 1.98936705, 1.75289221, 1.69596951,
       1.92087342, 1.44895286, 2.29045143, 1.4558231 , 2.22876575,
       2.2043377 , 1.91871643, 1.84483136, 2.54140936, 1.70117437,
       2.05965361, 1.51705794, 1.81781142, 2.0291577 , 1.80783854,
       2.0138298 , 1.980037  , 1.85588429, 1.93060558, 1.34773886,
       1.91447034, 1.75978513, 1.82873048, 2.08978356, 1.26643026,
       2.48213802, 1.7683624 , 2.40895928, 2.62557341, 1.81964648,
       1.84348552, 2.43502039, 1.997738  , 1.98100989, 1.91718359,
       1.79334553, 1.93679284, 1.80742317, 1.74573569, 1.99522728,
       1.9001478 , 2.27293834, 1.98647476, 1.8150405 , 1.90720433,
       2.94119796, 2.17876897, 1.78138715, 1.67709711, 2.02725868,
       2.44678822, 2.03694414, 2.45989757, 1.74778493, 2.2518794 ,
       1.4898767 , 2.01469702, 2.38008713, 2.07756995, 2.26563375,
       2.12764036, 1.8845882 , 1.97192778, 2.30337725, 2.10122967,
       2.03942583, 2.14544503, 1.73591251, 2.20886238, 1.9728476 ,
       1.7450414 , 1.56217245, 2.14676333, 1.79893176, 2.19542478,
       2.6825844 , 1.79935841, 2.17114618, 1.46349292, 1.65724671,
       2.16626081, 2.09234108, 1.93827301, 1.96406104, 2.37485225,
       1.9537505 , 2.00596074, 2.16019626, 2.38724097, 2.22956641,
       2.26797791, 2.50453138, 1.91770088, 1.78979546, 1.50465685,
       1.87395415, 1.97592566, 2.48935403, 2.26473888, 1.63286279,
       2.21744957, 2.13937929, 2.06175399, 2.01967067, 2.06383905,
       2.25066181, 2.14990025, 2.19456881, 2.46169743, 2.41642957,
       2.20500691, 2.32417108, 2.23428504, 1.95356526, 1.70782296,
       2.67923403, 2.14299365, 2.00925918, 1.73751139, 2.13069998,
       2.64781339, 2.06361869, 1.72032094, 1.74286309, 1.69799942,
       1.95868322, 2.53519862, 2.13479637, 1.95793641, 2.12800488,
       2.01263571, 2.10727252, 1.91302864, 1.83486435, 2.16943251,
       2.14040675, 2.3763879 , 2.48511886, 1.89110727, 2.40598257,
       1.91144116, 2.47957096, 1.7461907 , 1.8994968 , 2.03663375,
       2.06548666, 2.39147523, 2.13028001, 1.9895038 , 2.11336714,
       2.06628776, 1.67736474, 1.6982976 , 2.23559273, 1.45225013,
       2.05293631, 2.09933561, 2.00132161, 2.02721961, 1.98032174,
       1.7617149 , 2.26029724, 2.00410287, 1.99882748, 1.82394422,
       1.87950783, 2.55704616, 2.07020351, 1.76886125, 1.74954093,
       2.69517098, 2.03615693, 2.60928485, 2.28406125, 2.17860311,
       2.11681652, 1.90374738, 1.90010615, 2.28947187, 1.67149551,
       1.86005984, 1.69373367, 2.15954387, 1.66653857, 2.0549499 ,
       2.0648361 , 2.6528081 , 1.62746167, 2.04650615, 1.76701236,
       1.95427137, 1.83815072, 1.29638993, 1.67176704, 2.18433503,
       3.07357893, 2.02822515, 2.10611903, 2.28984108, 1.97791392,
       1.87486014, 1.69433803, 2.56305081, 2.08959835, 1.87512355,
       1.93869501, 2.15874497, 2.09769598, 1.80132271, 2.51913779,
       1.92941412, 1.73694928, 2.0107068 , 2.29794933, 1.93119752,
       1.45788239, 2.20500358, 1.89032568, 2.29115143, 2.23884199,
       1.74264001, 2.73923327, 2.68588831, 1.74000188, 1.78868645,
       2.37355951, 2.20379106, 2.02378358, 2.40677578, 1.98724561,
       2.236324  , 2.34800342, 1.89604148, 1.97626666, 2.30193197,
       2.36817312, 2.19185091, 2.10060257, 2.19113991, 1.93710022,
       2.01165533, 1.94384554, 2.80405332, 2.24648819, 2.01963755,
       1.88669824, 1.92848419, 2.22336388, 2.01584707, 1.92979416,
       1.52671847, 2.59624977, 1.98359465, 1.94686938, 1.74162761,
       2.44356091, 2.19694542, 1.94772047, 2.2195643 , 2.04941136,
       2.17055792, 1.99614261, 1.94694232, 2.23885179, 2.14774963,
       2.06236136, 1.78791691, 1.59434478, 2.37666278, 2.64634448,
       2.05839346, 2.10317574, 1.97646033, 2.19247227, 1.92437924,
       2.06393185, 2.07380102, 2.57959118, 1.93219581, 2.48894233])
```

In [ ]:

```
len(y_predictions_Old_Data.var(axis=0))
```

Out[ ]:

```
500
```

Now, we compute the mean of the previous vector:

$$\frac{1}{n} \cdot \sum_{i=1,\dots,n} \widehat{Var}(\hat{y}_i)$$

In [ ]:

```
y_predictions_Old_Data.var(axis=0).mean()
```

Out[ ]:

```
1.3526733933331723
```

In [ ]:

```
PAI_denominator = y_predictions_Old_Data.var(axis=0).mean()
```

In [ ]:

```
PAI_denominator
```

Out[ ]:

```
1.3526733933331723
```

We repeat the previous process but now we get the response predictions for de predictors of the New_Data_Set (this is so important, taking into a count the PAI definitions).

In [ ]:

```
df_New['X2'] = df_New['X2'].astype('category')
df_New['X3'] = df_New['X3'].astype('category')
```

In [ ]:

```
B=100

y_predictions_New_Data = np.zeros((B , len(df_New)))

for i in range(0, B):

    # i-th boot sample of the Old Data

    df_Old_BOOT_SAMPLE = df_Old.sample( n=len(df_Old) , random_state=i , replace=True )



    X_old_Boot_Sample = df_Old_BOOT_SAMPLE[['X1', 'X2', 'X3']]

    y_old_Boot_Sample =  df_Old_BOOT_SAMPLE['Y']

    X_old_Boot_Sample = varcharProcessing(X_old_Boot_Sample, varchar_process = "dummy_dropfirst")



    X_new = df_New[['X1', 'X2', 'X3']]

    y_new = df_New['Y']

    X_new = varcharProcessing(X_new, varchar_process = "dummy_dropfirst")


    Model_Old_Boot_Sample = LinearRegression().fit(X_old_Boot_Sample, y_old_Boot_Sample)


    # y predictions for the New Data using the model trained with the Old Data Boot Sample
    # For this step with sk-learn is necessary X_new (test_set) columns have the same name as X_old (train set) columns

    y_predictions_New_Data[i, :] = Model_Old_Boot_Sample.predict(X_new)
```

In [ ]:

```
PAI_numerator = y_predictions_New_Data.var(axis=0).mean()
```

In [ ]:

```
PAI_numerator
```

Out[ ]:

```
2.406479599837219
```

In [ ]:

```
PAI_denominator
```

Out[ ]:

```
1.3526733933331723
```

In [ ]:

```
PAI = PAI_numerator / PAI_denominator

PAI
```

Out[ ]:

```
1.7790544352375588
```

Remember:

$$PAI = \frac{\frac{1}{N}\sum_{i \in NewData} \widehat{Var}(\hat{y}_i)}{\frac{1}{n}\sum_{i \in OldData} \widehat{Var}(\hat{y}_i)}$$

Then, in mean, the variance of the response predictions using the New Data Set is 1.78 times greater than the variance of the response predictions using the Old Data Set.

Following the interpretation "values less than 1.1 indicate no significant deterioration; values from 1.1 to 1.5 indicate a deterioration requiring further investigation, values exceeding 1.5 indicate the predictive accuracy of the model has deteriorated significantly" exposed in the paper `The Population Accuracy Index: A New Measure of Population Stability for Model Monitoring` , so, the PAI value that we have got (1.78) indicates that the predictive accuracy of the model has deteriorated significantly, so the model should be trained again using the New Data Set instead the Old.