Modelling

This note looks at fitting various models to the datasets.

I am going fit the following classifiers on each of the dataseets (with PCA and without PCA):

* Logistic Regression
* Random Forest
* Support Vector Machine (classifier)
* K-Nearest Neighbours
* Naive Bayes
* Decision Tree

Having reconsidered the situation, I am now going to split the training datasets to get a validation set and leave the test sets for final model evaluation. I would probably want to split into three datasets from the start next time - I went back and updated the previous code.

### Import libraries and datasets

import pandas as pd  
  
X\_train\_transformed = pd.read\_csv('../../data/X\_train\_transformed.csv')  
#X\_test\_transformed = pd.read\_csv('../../data/X\_test\_transformed.csv')  
X\_val\_transformed = pd.read\_csv('../../data/X\_val\_transformed.csv')  
X\_train\_pca = pd.read\_csv('../../data/X\_train\_pca.csv')  
#X\_test\_pca = pd.read\_csv('../../data/X\_test\_pca.csv')  
X\_val\_pca = pd.read\_csv('../../data/X\_val\_pca.csv')  
  
  
y\_train = pd.read\_csv('../../data/y\_train.csv')  
#y\_test = pd.read\_csv('../../data/y\_test.csv')  
y\_val = pd.read\_csv('../../data/y\_val.csv')

print(len(X\_train\_transformed))  
print(len(X\_test\_transformed))  
print(len(X\_train\_pca))  
print(len(X\_test\_pca))  
print(len(y\_train))  
print(len(y\_test))

18861  
6288  
18861  
6288  
18861  
6288

### Fitting Classifiers on multiclass outcome

First I will fit some classifiers on the multiclass outcome - that is ‘final\_result’ as it exists in the dataset. As mentioned elsewhere, I am also considering whether it is more appropriate to refactor the final\_result into a binary outcome of intervene or do\_not\_intervene which may be more applicable to a real world situation for an institution.

from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
  
# models  
models = [  
 LogisticRegression(max\_iter=1000),  
 DecisionTreeClassifier(),  
 RandomForestClassifier(),  
 SVC(),  
 KNeighborsClassifier(),  
 GaussianNB()  
]  
  
# iterate over the datasets  
for dataset\_name, X\_train, X\_val in [('X\_train\_transformed', X\_train\_transformed, X\_val\_transformed),  
 ('X\_train\_pca', X\_train\_pca, X\_val\_pca)]:  
 print(f"Results for {dataset\_name}:")  
   
 # iterate over the models  
 for model in models:  
 model\_name = type(model).\_\_name\_\_  
 print(f"Model: {model\_name}")  
   
 # fit model on the training data  
 model.fit(X\_train, y\_train.values.ravel())   
   
 # predictions on the validation set  
 y\_pred\_val = model.predict(X\_val)  
   
 # evaluation metrics on the validation set  
 accuracy\_val = accuracy\_score(y\_val, y\_pred\_val)  
 precision\_val = precision\_score(y\_val, y\_pred\_val, average='weighted', zero\_division=0)  
 recall\_val = recall\_score(y\_val, y\_pred\_val, average='weighted', zero\_division=0)  
 f1\_val = f1\_score(y\_val, y\_pred\_val, average='weighted', zero\_division=0)  
   
 print(f"Validation Accuracy: {accuracy\_val}")  
 print(f"Validation Precision: {precision\_val}")  
 print(f"Validation Recall: {recall\_val}")  
 print(f"Validation F1 Score: {f1\_val}")  
   
 # confusion matrix for the validation set  
 confusion\_matrix\_val = confusion\_matrix(y\_val, y\_pred\_val)  
 print("Confusion Matrix (Validation Set):")  
 print(confusion\_matrix\_val)  
   
 print()

Results for X\_train\_transformed:  
Model: LogisticRegression  
Validation Accuracy: 0.7014949109414759  
Validation Precision: 0.6789715814695912  
Validation Recall: 0.7014949109414759  
Validation F1 Score: 0.6773384147821265  
Confusion Matrix (Validation Set):  
[[ 213 3 347 2]  
 [ 0 430 344 607]  
 [ 116 95 2144 11]  
 [ 2 299 51 1624]]  
  
Model: DecisionTreeClassifier  
Validation Accuracy: 0.6372455470737913  
Validation Precision: 0.6348486963403792  
Validation Recall: 0.6372455470737913  
Validation F1 Score: 0.6359989029436383  
Confusion Matrix (Validation Set):  
[[ 246 15 299 5]  
 [ 8 560 283 530]  
 [ 309 251 1771 35]  
 [ 4 504 38 1430]]  
  
Model: RandomForestClassifier  
Validation Accuracy: 0.709764631043257  
Validation Precision: 0.6916675731434337  
Validation Recall: 0.709764631043257  
Validation F1 Score: 0.6936903848920055  
Confusion Matrix (Validation Set):  
[[ 241 2 321 1]  
 [ 0 522 312 547]  
 [ 161 80 2112 13]  
 [ 3 352 33 1588]]  
  
Model: SVC  
Validation Accuracy: 0.7056297709923665  
Validation Precision: 0.688544368998229  
Validation Recall: 0.7056297709923665  
Validation F1 Score: 0.6574618601310753  
Confusion Matrix (Validation Set):  
[[ 92 0 471 2]  
 [ 0 344 392 645]  
 [ 44 26 2281 15]  
 [ 1 197 58 1720]]  
  
Model: KNeighborsClassifier  
Validation Accuracy: 0.6752544529262087  
Validation Precision: 0.6584783316882634  
Validation Recall: 0.6752544529262087  
Validation F1 Score: 0.663263791537826  
Confusion Matrix (Validation Set):  
[[ 199 7 357 2]  
 [ 5 537 333 506]  
 [ 231 134 1991 10]  
 [ 3 403 51 1519]]  
  
Model: GaussianNB  
Validation Accuracy: 0.6423346055979644  
Validation Precision: 0.610418836619168  
Validation Recall: 0.6423346055979644  
Validation F1 Score: 0.6187649923984077  
Confusion Matrix (Validation Set):  
[[ 166 22 377 0]  
 [ 5 335 324 717]  
 [ 293 164 1905 4]  
 [ 2 300 41 1633]]  
  
Results for X\_train\_pca:  
Model: LogisticRegression  
Validation Accuracy: 0.6725508905852418  
Validation Precision: 0.6326335366435673  
Validation Recall: 0.6725508905852418  
Validation F1 Score: 0.6185307240243099  
Confusion Matrix (Validation Set):  
[[ 10 12 542 1]  
 [ 0 371 367 643]  
 [ 10 73 2267 16]  
 [ 0 348 47 1581]]  
  
Model: DecisionTreeClassifier  
Validation Accuracy: 0.6162531806615776  
Validation Precision: 0.6124247247084929  
Validation Recall: 0.6162531806615776  
Validation F1 Score: 0.6142057175359747  
Confusion Matrix (Validation Set):  
[[ 185 27 344 9]  
 [ 28 535 297 521]  
 [ 350 276 1702 38]  
 [ 6 458 59 1453]]  
  
Model: RandomForestClassifier  
Validation Accuracy: 0.6720737913486005  
Validation Precision: 0.6459885622575348  
Validation Recall: 0.6720737913486005  
Validation F1 Score: 0.6504465555751666  
Confusion Matrix (Validation Set):  
[[ 125 7 430 3]  
 [ 7 495 346 533]  
 [ 160 110 2072 24]  
 [ 1 389 52 1534]]  
  
Model: SVC  
Validation Accuracy: 0.6847964376590331  
Validation Precision: 0.6033183875959515  
Validation Recall: 0.6847964376590331  
Validation F1 Score: 0.6111383559359974  
Confusion Matrix (Validation Set):  
[[ 0 1 561 3]  
 [ 0 260 416 705]  
 [ 0 27 2319 20]  
 [ 0 175 74 1727]]  
  
Model: KNeighborsClassifier  
Validation Accuracy: 0.6673027989821882  
Validation Precision: 0.6463708280242947  
Validation Recall: 0.6673027989821882  
Validation F1 Score: 0.6519710711310617  
Confusion Matrix (Validation Set):  
[[ 155 7 400 3]  
 [ 17 518 337 509]  
 [ 201 148 2002 15]  
 [ 6 393 56 1521]]  
  
Model: GaussianNB  
Validation Accuracy: 0.6531488549618321  
Validation Precision: 0.6119033228473788  
Validation Recall: 0.6531488549618321  
Validation F1 Score: 0.6124341251402656  
Confusion Matrix (Validation Set):  
[[ 70 13 481 1]  
 [ 2 336 413 630]  
 [ 89 96 2168 13]  
 [ 2 337 104 1533]]

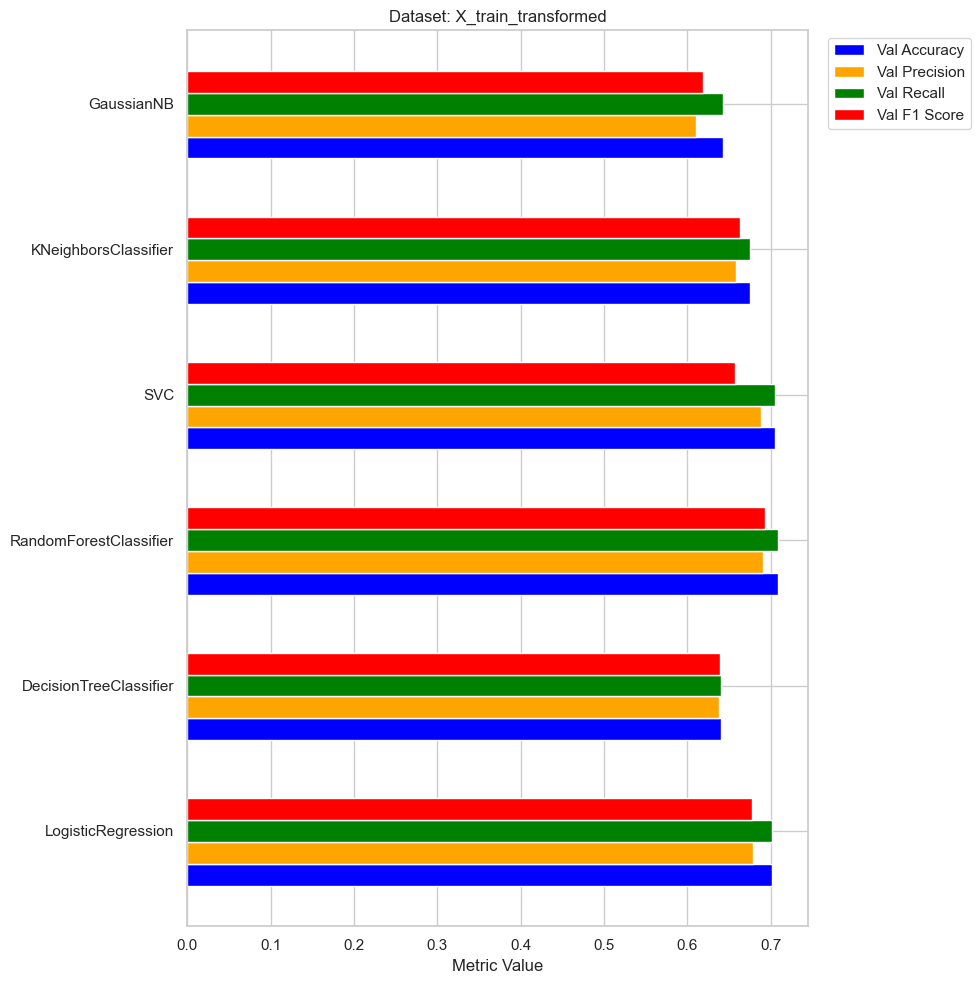
import pandas as pd  
import numpy as np  
  
# empty list of dictionaries for metrics data  
metrics\_data = [  
 {  
 'Model': '',  
 'Dataset': '',  
 'Val Accuracy': np.nan,  
 'Val Precision': np.nan,  
 'Val Recall': np.nan,  
 'Val F1 Score': np.nan  
 }  
]  
confusion\_matrices = {}  
  
# iterate over the datasets  
for dataset\_name, X\_train, X\_val in [('X\_train\_transformed', X\_train\_transformed, X\_val\_transformed),  
 ('X\_train\_pca', X\_train\_pca, X\_val\_pca)]:  
 print(f"\nDataset {dataset\_name}:\n")  
   
 # iterate over the models  
 for model in models:  
 model\_name = type(model).\_\_name\_\_  
 print(f"Saving results for: {model\_name}")  
   
 # fit model on the training data  
 model.fit(X\_train, y\_train.values.ravel())   
   
 # predictions on the validation set  
 y\_pred\_val = model.predict(X\_val)  
   
 # evaluation metrics on the validation set  
 accuracy\_val = accuracy\_score(y\_val, y\_pred\_val)  
 precision\_val = precision\_score(y\_val, y\_pred\_val, average='weighted', zero\_division=0)  
 recall\_val = recall\_score(y\_val, y\_pred\_val, average='weighted', zero\_division=0)  
 f1\_val = f1\_score(y\_val, y\_pred\_val, average='weighted', zero\_division=0)  
   
 # store metrics   
 metrics\_data.append({  
 'Model': model\_name,  
 'Dataset': dataset\_name,  
 'Val Accuracy': accuracy\_val,  
 'Val Precision': precision\_val,  
 'Val Recall': recall\_val,  
 'Val F1 Score': f1\_val  
 })  
   
 # confusion matrix for the validation set  
 confusion\_matrix\_val = confusion\_matrix(y\_val, y\_pred\_val)  
   
 # store confusion matrix using dataset and model as keys  
 confusion\_matrices[(dataset\_name, model\_name)] = confusion\_matrix\_val  
  
# metrics df  
metrics\_df = pd.DataFrame(metrics\_data)

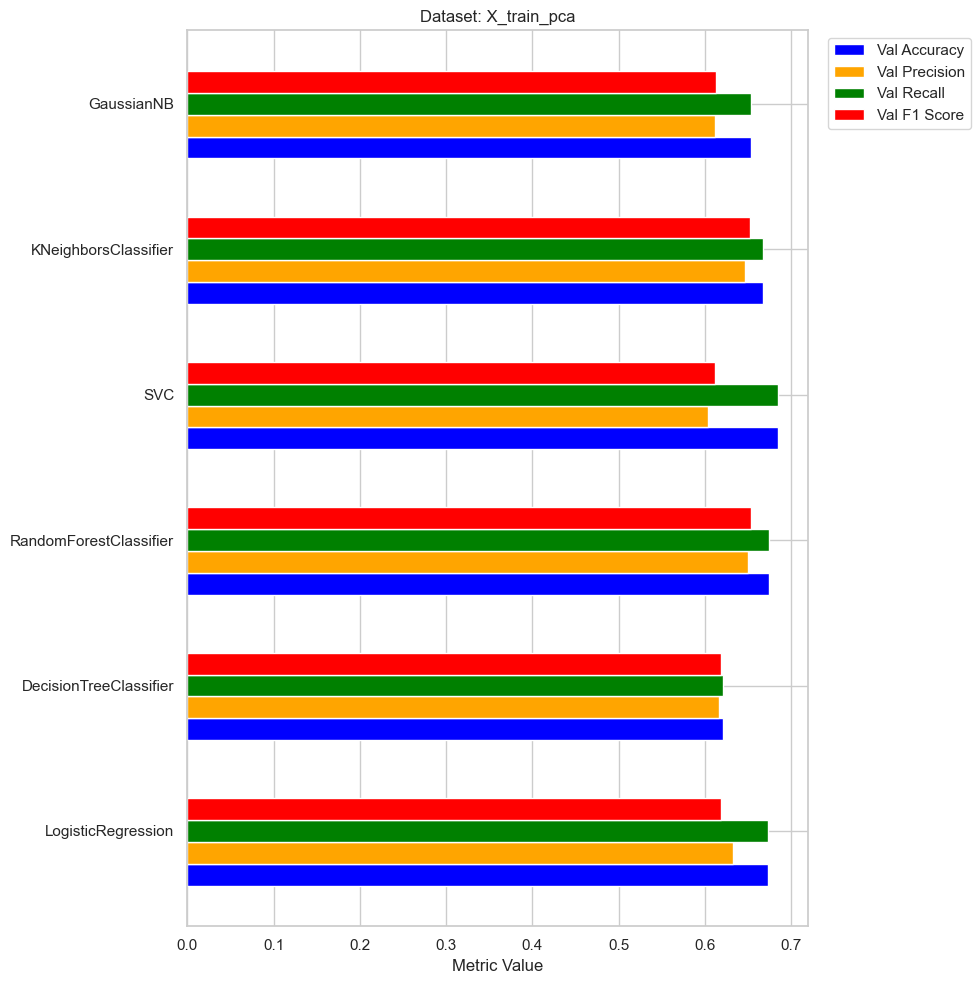
Dataset X\_train\_transformed:  
  
Saving results for: LogisticRegression  
Saving results for: DecisionTreeClassifier  
Saving results for: RandomForestClassifier  
Saving results for: SVC  
Saving results for: KNeighborsClassifier  
Saving results for: GaussianNB  
  
Dataset X\_train\_pca:  
  
Saving results for: LogisticRegression  
Saving results for: DecisionTreeClassifier  
Saving results for: RandomForestClassifier  
Saving results for: SVC  
Saving results for: KNeighborsClassifier  
Saving results for: GaussianNB

# Print  
print("Metrics DataFrame:")  
print(metrics\_df)  
  
print(metrics\_df.info())

Metrics DataFrame:  
 Model Dataset Val Accuracy Val Precision \  
0 NaN NaN   
1 LogisticRegression X\_train\_transformed 0.701495 0.678972   
2 DecisionTreeClassifier X\_train\_transformed 0.641062 0.638351   
3 RandomForestClassifier X\_train\_transformed 0.709128 0.690761   
4 SVC X\_train\_transformed 0.705630 0.688544   
5 KNeighborsClassifier X\_train\_transformed 0.675254 0.658478   
6 GaussianNB X\_train\_transformed 0.642335 0.610419   
7 LogisticRegression X\_train\_pca 0.672551 0.632634   
8 DecisionTreeClassifier X\_train\_pca 0.621183 0.615888   
9 RandomForestClassifier X\_train\_pca 0.674459 0.649413   
10 SVC X\_train\_pca 0.684796 0.603318   
11 KNeighborsClassifier X\_train\_pca 0.667303 0.646371   
12 GaussianNB X\_train\_pca 0.653149 0.611903   
  
 Val Recall Val F1 Score   
0 NaN NaN   
1 0.701495 0.677338   
2 0.641062 0.639654   
3 0.709128 0.692938   
4 0.705630 0.657462   
5 0.675254 0.663264   
6 0.642335 0.618765   
7 0.672551 0.618531   
8 0.621183 0.618309   
9 0.674459 0.652711   
10 0.684796 0.611138   
11 0.667303 0.651971   
12 0.653149 0.612434   
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 13 entries, 0 to 12  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Model 13 non-null object   
 1 Dataset 13 non-null object   
 2 Val Accuracy 12 non-null float64  
 3 Val Precision 12 non-null float64  
 4 Val Recall 12 non-null float64  
 5 Val F1 Score 12 non-null float64  
dtypes: float64(4), object(2)  
memory usage: 752.0+ bytes  
None

import matplotlib.pyplot as plt  
import numpy as np  
  
# separate dataset  
noPCA = metrics\_df[metrics\_df['Dataset'] == 'X\_train\_transformed']  
PCA = metrics\_df[metrics\_df['Dataset'] == 'X\_train\_pca']  
  
# figure size  
plt.figure(figsize=(10, 10))  
  
# metrics  
metrics = ['Val Accuracy', 'Val Precision', 'Val Recall', 'Val F1 Score']  
  
# colors  
colors = ['blue', 'orange', 'green', 'red']  
  
# positions of the bars on the y-axis  
y = np.arange(len(noPCA['Model']))  
  
# height of the bars  
height = 0.15  
  
# grouped bars for noPCA  
for i, metric in enumerate(metrics):  
 plt.barh(y + i\*height, dataset\_1[metric], height, label=metric, color=colors[i])  
  
# y-axis ticks and labels  
plt.yticks(y + height\*len(metrics)/2, noPCA['Model'])  
  
# x-axis label   
plt.xlabel('Metric Value')  
  
# legend to the side  
plt.legend(bbox\_to\_anchor=(1.02, 1), loc='upper left')  
  
# title   
plt.title('Dataset: X\_train\_transformed')  
  
# plot  
plt.tight\_layout()  
plt.show()  
  
# figure size for the second plot  
plt.figure(figsize=(10, 10))  
  
# positions of the bars on the y-axis second plot  
y = np.arange(len(PCA['Model']))  
  
# grouped bars for pca  
for i, metric in enumerate(metrics):  
 plt.barh(y + i\*height, dataset\_2[metric], height, label=metric, color=colors[i])  
  
# y-axis ticks and labels   
plt.yticks(y + height\*len(metrics)/2, PCA['Model'])  
  
# Sx-axis label second plot  
plt.xlabel('Metric Value')  
  
# legend to the side second plot  
plt.legend(bbox\_to\_anchor=(1.02, 1), loc='upper left')  
  
# title second plot  
plt.title('Dataset: X\_train\_pca')  
  
# plot  
plt.tight\_layout()  
plt.show()

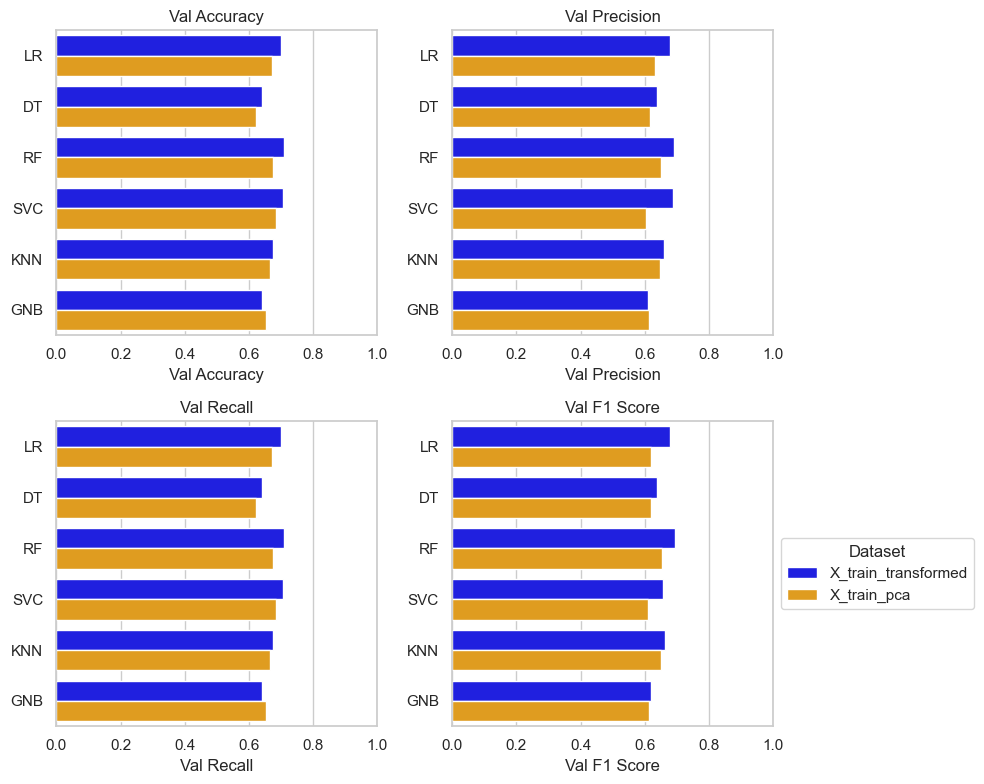




# make df  
metrics\_df = pd.DataFrame(metrics\_data)  
# drop the first row with NaN values  
metrics\_df = metrics\_df.drop(0)  
metrics\_df

|  | Model | Dataset | Val Accuracy | Val Precision | Val Recall | Val F1 Score |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | LogisticRegression | X\_train\_transformed | 0.701495 | 0.678972 | 0.701495 | 0.677338 |
| 2 | DecisionTreeClassifier | X\_train\_transformed | 0.641062 | 0.638351 | 0.641062 | 0.639654 |
| 3 | RandomForestClassifier | X\_train\_transformed | 0.709128 | 0.690761 | 0.709128 | 0.692938 |
| 4 | SVC | X\_train\_transformed | 0.705630 | 0.688544 | 0.705630 | 0.657462 |
| 5 | KNeighborsClassifier | X\_train\_transformed | 0.675254 | 0.658478 | 0.675254 | 0.663264 |
| 6 | GaussianNB | X\_train\_transformed | 0.642335 | 0.610419 | 0.642335 | 0.618765 |
| 7 | LogisticRegression | X\_train\_pca | 0.672551 | 0.632634 | 0.672551 | 0.618531 |
| 8 | DecisionTreeClassifier | X\_train\_pca | 0.621183 | 0.615888 | 0.621183 | 0.618309 |
| 9 | RandomForestClassifier | X\_train\_pca | 0.674459 | 0.649413 | 0.674459 | 0.652711 |
| 10 | SVC | X\_train\_pca | 0.684796 | 0.603318 | 0.684796 | 0.611138 |
| 11 | KNeighborsClassifier | X\_train\_pca | 0.667303 | 0.646371 | 0.667303 | 0.651971 |
| 12 | GaussianNB | X\_train\_pca | 0.653149 | 0.611903 | 0.653149 | 0.612434 |

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# abbreviate model names  
modified\_metrics\_df = metrics\_df.copy()  
modified\_metrics\_df['Model'] = modified\_metrics\_df['Model'].replace({  
 'LogisticRegression': 'LR',  
 'DecisionTreeClassifier': 'DT',  
 'RandomForestClassifier': 'RF',  
 'SVC': 'SVC',  
 'KNeighborsClassifier': 'KNN',  
 'GaussianNB': 'GNB'  
})  
  
# style  
sns.set(style="whitegrid")  
  
# figure size  
plt.figure(figsize=(10, 8))  
  
# metrics  
metrics = ['Val Accuracy', 'Val Precision', 'Val Recall', 'Val F1 Score']  
  
# Spalette  
dataset\_palette = {'X\_train\_transformed': 'blue', 'X\_train\_pca': 'orange'}  
  
# bar plots for each metric  
for i, metric in enumerate(metrics):  
 # subplot for each metric  
 ax = plt.subplot(2, 2, i+1)  
   
 # bar plot for each dataset  
 sns.barplot(x=metric, y='Model', hue='Dataset', data=modified\_metrics\_df, palette=dataset\_palette, orient='h', ax=ax)  
   
 # title  
 ax.set\_title(metric)  
   
 # x-axis label  
 ax.set\_xlabel(metric)  
   
 # remove the y-axis label  
 ax.set\_ylabel('')  
   
 # x-axis limit to 1.0  
 ax.set\_xlim(0, 1)  
   
 # remove the legend from first three plots  
 if i < 3:  
 ax.get\_legend().remove()  
  
# add a legend outside the subplots  
plt.legend(title='Dataset', loc='center left', bbox\_to\_anchor=(1, 0.5))  
  
# tight  
plt.tight\_layout()  
  
# plot  
plt.show()



After much plotting, we have something which will allow us to compare between datasets (PCA and non-PCA) and between models. I have visualised four performance metrics:

* **Accuracy** - the proportion of correctly classified samples - that is the overall correctness of the predictions. This is a simple metric, easy to understand and explain. However, it can be mislesding if the dataset is imbalanced - e.g. if the classes have different proportions.
* **Precision** - proportion of true positive predictions for a specific outcome over the total instances predicted to be that outcome - e.g. total predicted passed students who actually passed over the total predicted passed students. Precision is useful in multi-class problems as the model’s performance can vary for each specific class - thus it tells us about the model’s ability to discriminate between different classes and make accurate predictions. Precision can tell us if the model performs well on a particular outcome and whether it needs improvement for others.
* **Recall** - proportion of true positive predictions for a specific outcome over the total instances of that outcome - e.g. total predicted passed students who actually passed over the total passed students. Recall can be helpful where it is important in minimising false negatives - that is where it is important to not miss positive instances. Recall does not consider false positives, so it may lead to a high number of ‘false alarms’ or unnecessary action. In the case of student outcomes, focusing on recall alone as the performance metric might lead to unnecessarily contacting students for intervention, for example.
* **F1 score** - this is the harmonic mean of precision and recall, providing a balance between both metrics.

There is not much difference between the models - none of them are particularly good:

The most accurate model is RandomForestClassifier without PCA with an accuracy of 70.9% - that is, it got 70.9% of predictions correct. It’s F1 score was 0.693. The second most accurate classiifier for this dataset was Support Vector Classifier with an accuracy of 70.6% and an F1 score of 0.6657.

For the PCA transformed data, the SVC was most accuracte with an accuracy of 68.5% and an F1 score of 0.611. Random Forest classifier was second most accurate with an overall accuracy of 67.4% and a slightly better F1 score of 0.653.

Logistic Regression was the third best model for both datasets.

In conclusion, dimension reduced data via PCA transformation does not perform significantly worse than non-PCA transformed data so it may be worth considering PCA.

# confusion matrix   
dataset\_name = 'X\_train\_transformed'  
model\_name = 'RandomForestClassifier'  
confusion\_matrix\_val = confusion\_matrices[(dataset\_name, model\_name)]  
print(f"Confusion Matrix for {dataset\_name} - {model\_name}:")  
print(confusion\_matrix\_val)

Confusion Matrix for X\_train\_transformed - RandomForestClassifier:  
[[ 238 3 323 1]  
 [ 2 520 310 549]  
 [ 151 86 2118 11]  
 [ 2 360 31 1583]]

### Fitting Classifiers binary outcomes

#### Refactoring the ‘final\_result’ target variable

# copy the data  
y\_train\_binary = y\_train.copy()  
  
# map values to 'intervene' and 'no\_intervene'  
y\_train\_binary.replace({'Pass': 'no\_intervene', 'Distinction': 'no\_intervene',  
 'Withdrawn': 'intervene', 'Fail': 'intervene'}, inplace=True)  
  
y\_val\_binary = y\_val.copy()  
y\_val\_binary.replace({'Pass': 'no\_intervene', 'Distinction': 'no\_intervene',  
 'Withdrawn': 'intervene', 'Fail': 'intervene'}, inplace=True)

Redo models with binary outcome

from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix  
  
# models  
models = [  
 LogisticRegression(max\_iter=1000),  
 DecisionTreeClassifier(),  
 RandomForestClassifier(),  
 SVC(),  
 KNeighborsClassifier(),  
 GaussianNB()  
]  
  
# iterate over the datasets  
for dataset\_name, X\_train, X\_val in [('X\_train\_transformed', X\_train\_transformed, X\_val\_transformed),  
 ('X\_train\_pca', X\_train\_pca, X\_val\_pca)]:  
 print(f"Results for {dataset\_name}:")  
   
 # iterate over the models  
 for model in models:  
 model\_name = type(model).\_\_name\_\_  
 print(f"Model: {model\_name}")  
   
 # fit model on the training data  
 model.fit(X\_train, y\_train\_binary.values.ravel())   
   
 # predictions on the validation set  
 y\_pred\_val\_binary = model.predict(X\_val)  
   
 # evaluation metrics on the validation set  
 accuracy\_val = accuracy\_score(y\_val\_binary, y\_pred\_val\_binary)  
 precision\_val = precision\_score(y\_val\_binary, y\_pred\_val\_binary, average='weighted', zero\_division=0)  
 recall\_val = recall\_score(y\_val\_binary, y\_pred\_val\_binary, average='weighted', zero\_division=0)  
 f1\_val = f1\_score(y\_val\_binary, y\_pred\_val\_binary, average='weighted', zero\_division=0)  
   
 print(f"Validation Accuracy: {accuracy\_val}")  
 print(f"Validation Precision: {precision\_val}")  
 print(f"Validation Recall: {recall\_val}")  
 print(f"Validation F1 Score: {f1\_val}")  
   
 # confusion matrix for the validation set  
 confusion\_matrix\_val\_binary = confusion\_matrix(y\_val\_binary, y\_pred\_val\_binary)  
 print("Confusion Matrix (Validation Set):")  
 print(confusion\_matrix\_val\_binary)  
   
 print()

Results for X\_train\_transformed:  
Model: LogisticRegression  
Validation Accuracy: 0.9184160305343512  
Validation Precision: 0.9206324518592317  
Validation Recall: 0.9184160305343512  
Validation F1 Score: 0.9185097227087798  
Confusion Matrix (Validation Set):  
[[2996 361]  
 [ 152 2779]]  
  
Model: DecisionTreeClassifier  
Validation Accuracy: 0.901558524173028  
Validation Precision: 0.9016137033912648  
Validation Recall: 0.901558524173028  
Validation F1 Score: 0.9015778582910531  
Confusion Matrix (Validation Set):  
[[3038 319]  
 [ 300 2631]]  
  
Model: RandomForestClassifier  
Validation Accuracy: 0.9305025445292621  
Validation Precision: 0.932639841126152  
Validation Recall: 0.9305025445292621  
Validation F1 Score: 0.93058227544868  
Confusion Matrix (Validation Set):  
[[3036 321]  
 [ 116 2815]]  
  
Model: SVC  
Validation Accuracy: 0.9209605597964376  
Validation Precision: 0.9266898173768655  
Validation Recall: 0.9209605597964376  
Validation F1 Score: 0.9210153677910821  
Confusion Matrix (Validation Set):  
[[2935 422]  
 [ 75 2856]]  
  
Model: KNeighborsClassifier  
Validation Accuracy: 0.9173027989821882  
Validation Precision: 0.9201623803379486  
Validation Recall: 0.9173027989821882  
Validation F1 Score: 0.9173962470009592  
Confusion Matrix (Validation Set):  
[[2977 380]  
 [ 140 2791]]  
  
Model: GaussianNB  
Validation Accuracy: 0.9134860050890585  
Validation Precision: 0.9162128429898908  
Validation Recall: 0.9134860050890585  
Validation F1 Score: 0.9135844024585985  
Confusion Matrix (Validation Set):  
[[2968 389]  
 [ 155 2776]]  
  
Results for X\_train\_pca:  
Model: LogisticRegression  
Validation Accuracy: 0.9166666666666666  
Validation Precision: 0.9196146968137173  
Validation Recall: 0.9166666666666666  
Validation F1 Score: 0.9167603406282921  
Confusion Matrix (Validation Set):  
[[2973 384]  
 [ 140 2791]]  
  
Model: DecisionTreeClassifier  
Validation Accuracy: 0.8762722646310432  
Validation Precision: 0.8763868963388032  
Validation Recall: 0.8762722646310432  
Validation F1 Score: 0.8763095882996926  
Confusion Matrix (Validation Set):  
[[2953 404]  
 [ 374 2557]]  
  
Model: RandomForestClassifier  
Validation Accuracy: 0.9150763358778626  
Validation Precision: 0.917511336354585  
Validation Recall: 0.9150763358778626  
Validation F1 Score: 0.9151737809597927  
Confusion Matrix (Validation Set):  
[[2980 377]  
 [ 157 2774]]  
  
Model: SVC  
Validation Accuracy: 0.9165076335877863  
Validation Precision: 0.9232608491382319  
Validation Recall: 0.9165076335877863  
Validation F1 Score: 0.9165452504907223  
Confusion Matrix (Validation Set):  
[[2905 452]  
 [ 73 2858]]  
  
Model: KNeighborsClassifier  
Validation Accuracy: 0.9106234096692112  
Validation Precision: 0.9137472521506536  
Validation Recall: 0.9106234096692112  
Validation F1 Score: 0.9107226027053527  
Confusion Matrix (Validation Set):  
[[2950 407]  
 [ 155 2776]]  
  
Model: GaussianNB  
Validation Accuracy: 0.9013994910941476  
Validation Precision: 0.9061869240064168  
Validation Recall: 0.9013994910941476  
Validation F1 Score: 0.9014851619492598  
Confusion Matrix (Validation Set):  
[[2888 469]  
 [ 151 2780]]

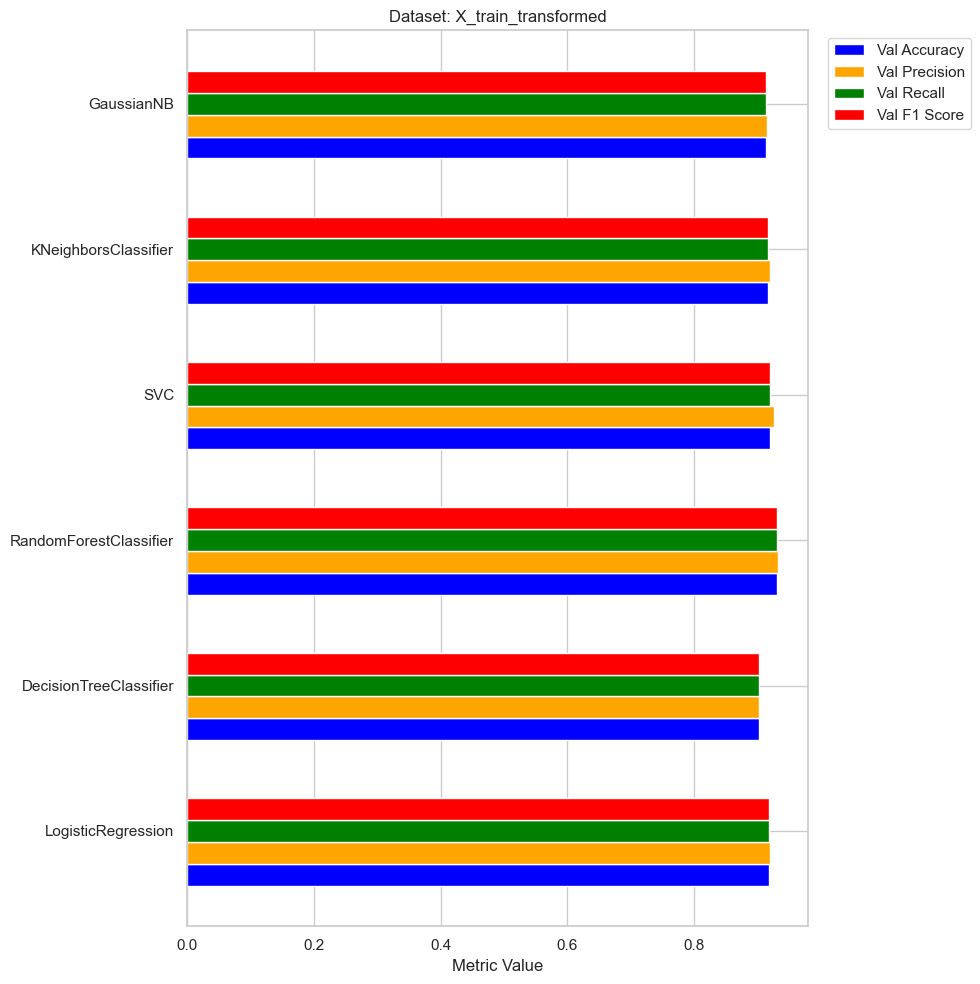
import pandas as pd  
import numpy as np  
  
# empty list of dictionaries for metrics data  
metrics\_data = [  
 {  
 'Model': '',  
 'Dataset': '',  
 'Val Accuracy': np.nan,  
 'Val Precision': np.nan,  
 'Val Recall': np.nan,  
 'Val F1 Score': np.nan  
 }  
]  
confusion\_matrices = {}  
  
# iterate over the datasets  
for dataset\_name, X\_train, X\_val in [('X\_train\_transformed', X\_train\_transformed, X\_val\_transformed),  
 ('X\_train\_pca', X\_train\_pca, X\_val\_pca)]:  
 print(f"\nDataset {dataset\_name}:\n")  
   
 # iterate over the models  
 for model in models:  
 model\_name = type(model).\_\_name\_\_  
 print(f"Saving results for: {model\_name}")  
   
 # fit model on the training data  
 model.fit(X\_train, y\_train\_binary.values.ravel())   
   
 # predictions on the validation set  
 y\_pred\_val\_binary = model.predict(X\_val)  
   
 # evaluation metrics on the validation set  
 accuracy\_val = accuracy\_score(y\_val\_binary, y\_pred\_val\_binary)  
 precision\_val = precision\_score(y\_val\_binary, y\_pred\_val\_binary, average='weighted', zero\_division=0)  
 recall\_val = recall\_score(y\_val\_binary, y\_pred\_val\_binary, average='weighted', zero\_division=0)  
 f1\_val = f1\_score(y\_val\_binary, y\_pred\_val\_binary, average='weighted', zero\_division=0)  
   
 # store metrics   
 metrics\_data.append({  
 'Model': model\_name,  
 'Dataset': dataset\_name,  
 'Val Accuracy': accuracy\_val,  
 'Val Precision': precision\_val,  
 'Val Recall': recall\_val,  
 'Val F1 Score': f1\_val  
 })  
   
 # confusion matrix for the validation set  
 confusion\_matrix\_val\_binary = confusion\_matrix(y\_val\_binary, y\_pred\_val\_binary)  
   
 # store confusion matrix using dataset and model as keys  
 confusion\_matrices[(dataset\_name, model\_name)] = confusion\_matrix\_val  
  
# metrics df  
metrics\_binary = pd.DataFrame(metrics\_data)

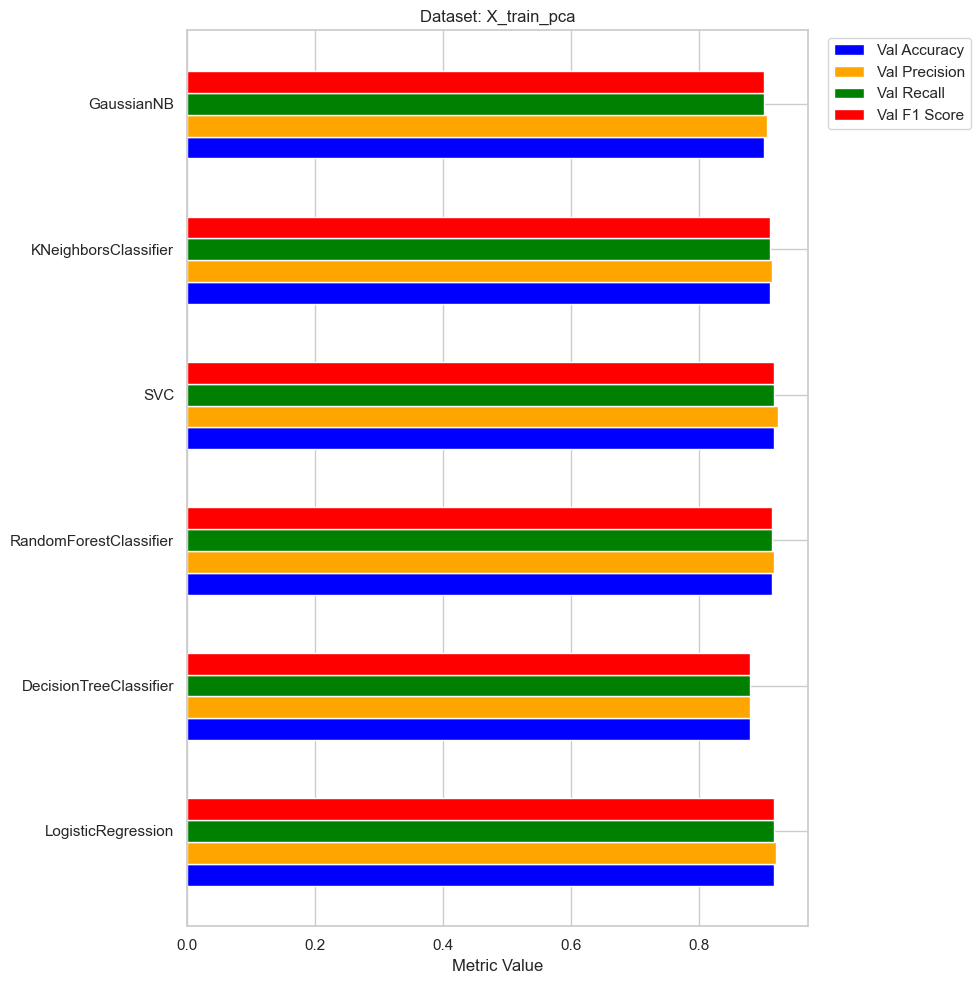
Dataset X\_train\_transformed:  
  
Saving results for: LogisticRegression  
Saving results for: DecisionTreeClassifier  
Saving results for: RandomForestClassifier  
Saving results for: SVC  
Saving results for: KNeighborsClassifier  
Saving results for: GaussianNB  
  
Dataset X\_train\_pca:  
  
Saving results for: LogisticRegression  
Saving results for: DecisionTreeClassifier  
Saving results for: RandomForestClassifier  
Saving results for: SVC  
Saving results for: KNeighborsClassifier  
Saving results for: GaussianNB

# Print  
print("Metrics DataFrame:")  
print(metrics\_binary)  
  
#print(metrics\_df.info())

Metrics DataFrame:  
 Model Dataset Val Accuracy Val Precision \  
0 NaN NaN   
1 LogisticRegression X\_train\_transformed 0.918416 0.920632   
2 DecisionTreeClassifier X\_train\_transformed 0.903308 0.903352   
3 RandomForestClassifier X\_train\_transformed 0.930980 0.933417   
4 SVC X\_train\_transformed 0.920961 0.926690   
5 KNeighborsClassifier X\_train\_transformed 0.917303 0.920162   
6 GaussianNB X\_train\_transformed 0.913486 0.916213   
7 LogisticRegression X\_train\_pca 0.916667 0.919615   
8 DecisionTreeClassifier X\_train\_pca 0.879453 0.879608   
9 RandomForestClassifier X\_train\_pca 0.913327 0.915948   
10 SVC X\_train\_pca 0.916508 0.923261   
11 KNeighborsClassifier X\_train\_pca 0.910623 0.913747   
12 GaussianNB X\_train\_pca 0.901399 0.906187   
  
 Val Recall Val F1 Score   
0 NaN NaN   
1 0.918416 0.918510   
2 0.903308 0.903324   
3 0.930980 0.931059   
4 0.920961 0.921015   
5 0.917303 0.917396   
6 0.913486 0.913584   
7 0.916667 0.916760   
8 0.879453 0.879498   
9 0.913327 0.913426   
10 0.916508 0.916545   
11 0.910623 0.910723   
12 0.901399 0.901485

import matplotlib.pyplot as plt  
import numpy as np  
  
# separate dataset  
noPCA = metrics\_binary[metrics\_binary['Dataset'] == 'X\_train\_transformed']  
PCA = metrics\_binary[metrics\_binary['Dataset'] == 'X\_train\_pca']  
  
# figure size  
plt.figure(figsize=(10, 10))  
  
# metrics  
metrics = ['Val Accuracy', 'Val Precision', 'Val Recall', 'Val F1 Score']  
  
# colors  
colors = ['blue', 'orange', 'green', 'red']  
  
# positions of the bars on the y-axis  
y = np.arange(len(noPCA['Model']))  
  
# height of the bars  
height = 0.15  
  
# grouped bars for noPCA  
for i, metric in enumerate(metrics):  
 plt.barh(y + i\*height, noPCA[metric], height, label=metric, color=colors[i])  
  
# y-axis ticks and labels  
plt.yticks(y + height\*len(metrics)/2, noPCA['Model'])  
  
# x-axis label   
plt.xlabel('Metric Value')  
  
# legend to the side  
plt.legend(bbox\_to\_anchor=(1.02, 1), loc='upper left')  
  
# title   
plt.title('Dataset: X\_train\_transformed')  
  
# plot  
plt.tight\_layout()  
plt.show()  
  
# figure size for the second plot  
plt.figure(figsize=(10, 10))  
  
# positions of the bars on the y-axis second plot  
y = np.arange(len(PCA['Model']))  
  
# grouped bars for pca  
for i, metric in enumerate(metrics):  
 plt.barh(y + i\*height, PCA[metric], height, label=metric, color=colors[i])  
  
# y-axis ticks and labels   
plt.yticks(y + height\*len(metrics)/2, PCA['Model'])  
  
# Sx-axis label second plot  
plt.xlabel('Metric Value')  
  
# legend to the side second plot  
plt.legend(bbox\_to\_anchor=(1.02, 1), loc='upper left')  
  
# title second plot  
plt.title('Dataset: X\_train\_pca')  
  
# plot  
plt.tight\_layout()  
plt.show()

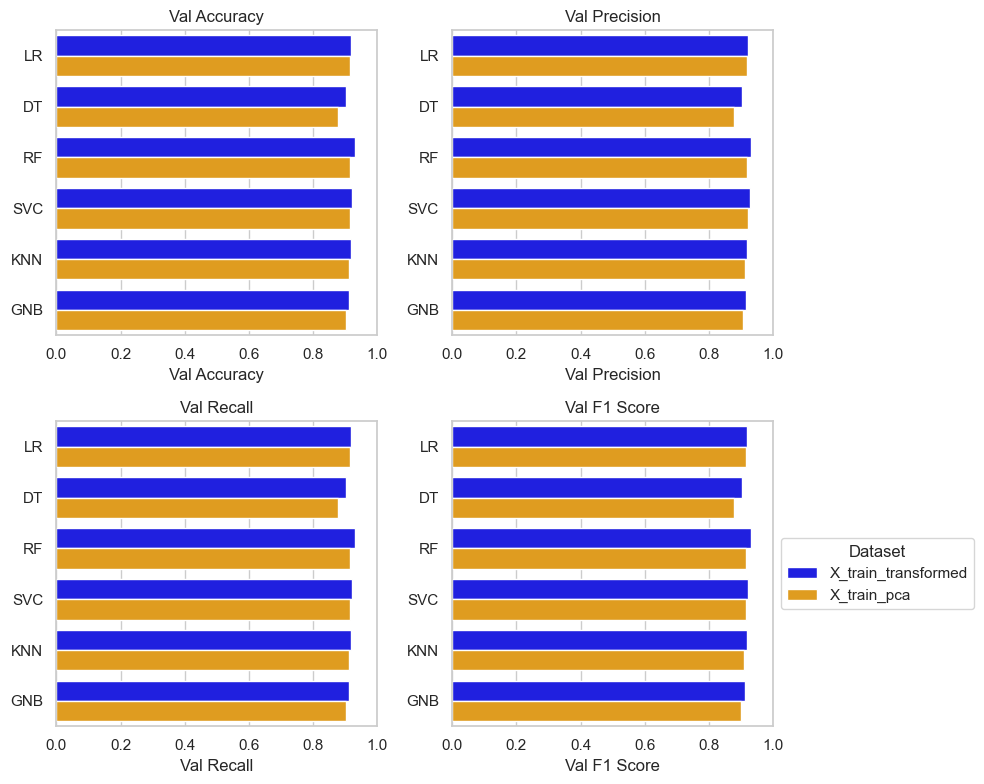




# make df  
metrics\_binary = pd.DataFrame(metrics\_data)  
# drop the first row with NaN values  
metrics\_binary = metrics\_df.drop(0)  
metrics\_binary

|  | Model | Dataset | Val Accuracy | Val Precision | Val Recall | Val F1 Score |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | LogisticRegression | X\_train\_transformed | 0.918416 | 0.920632 | 0.918416 | 0.918510 |
| 2 | DecisionTreeClassifier | X\_train\_transformed | 0.901559 | 0.901575 | 0.901559 | 0.901566 |
| 3 | RandomForestClassifier | X\_train\_transformed | 0.929866 | 0.932121 | 0.929866 | 0.929947 |
| 4 | SVC | X\_train\_transformed | 0.920961 | 0.926690 | 0.920961 | 0.921015 |
| 5 | KNeighborsClassifier | X\_train\_transformed | 0.917303 | 0.920162 | 0.917303 | 0.917396 |
| 6 | GaussianNB | X\_train\_transformed | 0.913486 | 0.916213 | 0.913486 | 0.913584 |
| 7 | LogisticRegression | X\_train\_pca | 0.916667 | 0.919615 | 0.916667 | 0.916760 |
| 8 | DecisionTreeClassifier | X\_train\_pca | 0.879135 | 0.879258 | 0.879135 | 0.879174 |
| 9 | RandomForestClassifier | X\_train\_pca | 0.915394 | 0.917911 | 0.915394 | 0.915491 |
| 10 | SVC | X\_train\_pca | 0.916508 | 0.923261 | 0.916508 | 0.916545 |
| 11 | KNeighborsClassifier | X\_train\_pca | 0.910623 | 0.913747 | 0.910623 | 0.910723 |
| 12 | GaussianNB | X\_train\_pca | 0.901399 | 0.906187 | 0.901399 | 0.901485 |

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# abbreviate model names  
modified\_metrics\_binary = metrics\_binary.copy()  
modified\_metrics\_binary['Model'] = modified\_metrics\_binary['Model'].replace({  
 'LogisticRegression': 'LR',  
 'DecisionTreeClassifier': 'DT',  
 'RandomForestClassifier': 'RF',  
 'SVC': 'SVC',  
 'KNeighborsClassifier': 'KNN',  
 'GaussianNB': 'GNB'  
})  
  
# style  
sns.set(style="whitegrid")  
  
# figure size  
plt.figure(figsize=(10, 8))  
  
# metrics  
metrics = ['Val Accuracy', 'Val Precision', 'Val Recall', 'Val F1 Score']  
  
# Spalette  
dataset\_palette = {'X\_train\_transformed': 'blue', 'X\_train\_pca': 'orange'}  
  
# bar plots for each metric  
for i, metric in enumerate(metrics):  
 # subplot for each metric  
 ax = plt.subplot(2, 2, i+1)  
   
 # bar plot for each dataset  
 sns.barplot(x=metric, y='Model', hue='Dataset', data=modified\_metrics\_binary, palette=dataset\_palette, orient='h', ax=ax)  
   
 # title  
 ax.set\_title(metric)  
   
 # x-axis label  
 ax.set\_xlabel(metric)  
   
 # remove the y-axis label  
 ax.set\_ylabel('')  
   
 # x-axis limit to 1.0  
 ax.set\_xlim(0, 1)  
   
 # remove the legend from first three plots  
 if i < 3:  
 ax.get\_legend().remove()  
  
# add a legend outside the subplots  
plt.legend(title='Dataset', loc='center left', bbox\_to\_anchor=(1, 0.5))  
  
# tight  
plt.tight\_layout()  
  
# plot  
plt.show()



metrics\_binary

|  | Model | Dataset | Val Accuracy | Val Precision | Val Recall | Val F1 Score |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | LogisticRegression | X\_train\_transformed | 0.918416 | 0.920632 | 0.918416 | 0.918510 |
| 2 | DecisionTreeClassifier | X\_train\_transformed | 0.901559 | 0.901575 | 0.901559 | 0.901566 |
| 3 | RandomForestClassifier | X\_train\_transformed | 0.929866 | 0.932121 | 0.929866 | 0.929947 |
| 4 | SVC | X\_train\_transformed | 0.920961 | 0.926690 | 0.920961 | 0.921015 |
| 5 | KNeighborsClassifier | X\_train\_transformed | 0.917303 | 0.920162 | 0.917303 | 0.917396 |
| 6 | GaussianNB | X\_train\_transformed | 0.913486 | 0.916213 | 0.913486 | 0.913584 |
| 7 | LogisticRegression | X\_train\_pca | 0.916667 | 0.919615 | 0.916667 | 0.916760 |
| 8 | DecisionTreeClassifier | X\_train\_pca | 0.879135 | 0.879258 | 0.879135 | 0.879174 |
| 9 | RandomForestClassifier | X\_train\_pca | 0.915394 | 0.917911 | 0.915394 | 0.915491 |
| 10 | SVC | X\_train\_pca | 0.916508 | 0.923261 | 0.916508 | 0.916545 |
| 11 | KNeighborsClassifier | X\_train\_pca | 0.910623 | 0.913747 | 0.910623 | 0.910723 |
| 12 | GaussianNB | X\_train\_pca | 0.901399 | 0.906187 | 0.901399 | 0.901485 |

Wow!

Predicting on binary outcomes results in much better performance. Of course, it is expected - it is much easier to predict one of two outcomes v one of four. There is also a better balance between the two outcomes in the dataset.

Thinking about the business scenario - being able to identify whether to intervene or not is much more useful than being able to predict the outcome…at least in the first instance. Being able to identify students who are not likely to continue their studies, either by eventual failure or withdrawal (which arguably is sometimes preempting failure) is the main goal of this model.

For the sake of the student, being able to intervene and potentially put measures in place to allow them to get back on track could have an immense outcome on their studies, prospects, opportunities, and life. The reasons for their discontinuation may be varied and complicated (financial, personal, academic, circumstantial, health, etc.) but early intervention will allow an institution to ensure that the student gets the best outcome for them.

For the institution, it is primarily about doing the right thing by the student. And by doing the right thing, the HE institution also benefits by way of better continuation rates, better retention rates, better outcomes for students, better reputation, better league table positions, better funding, etc.