Dataset Analysis

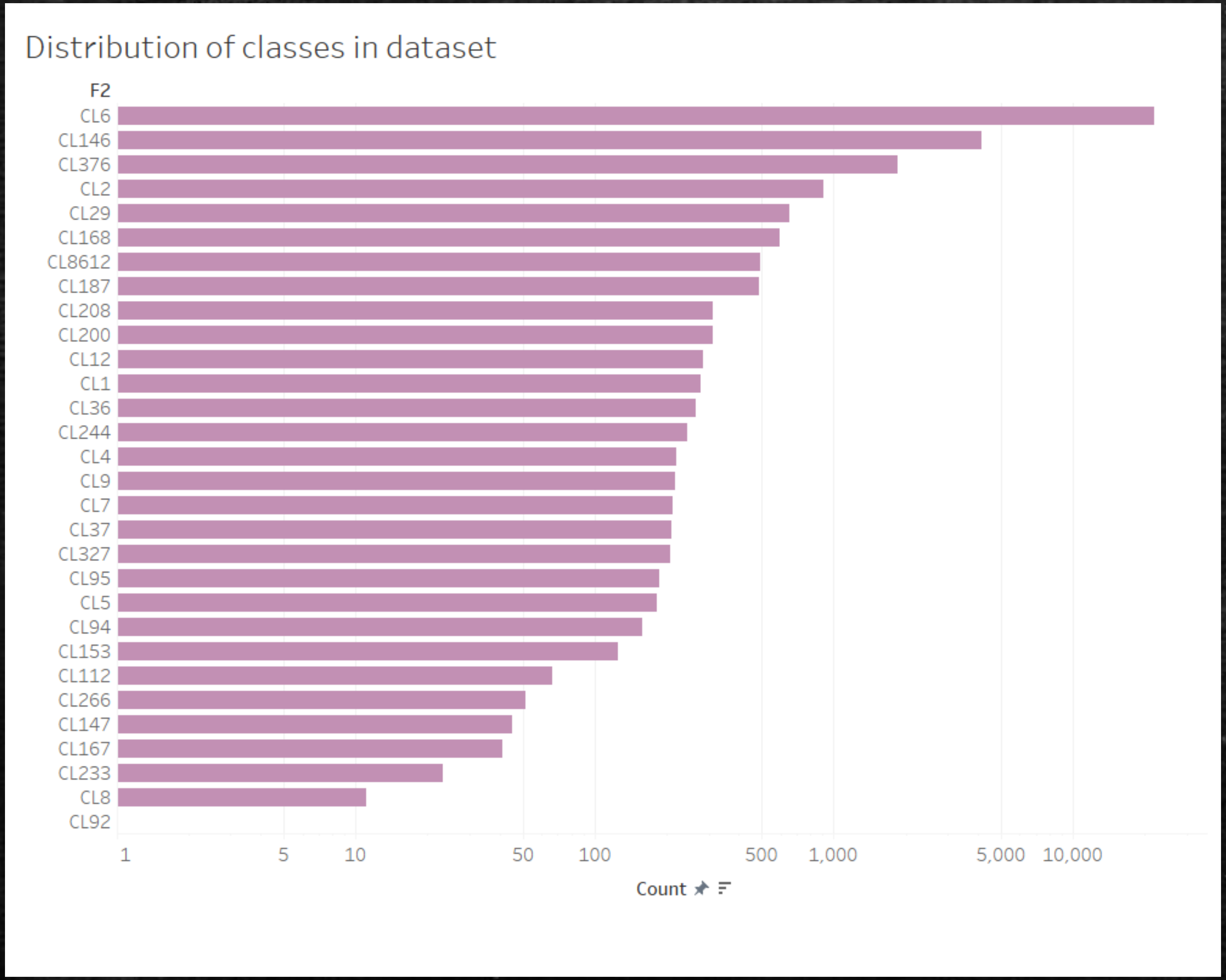
Data

X

y

Data

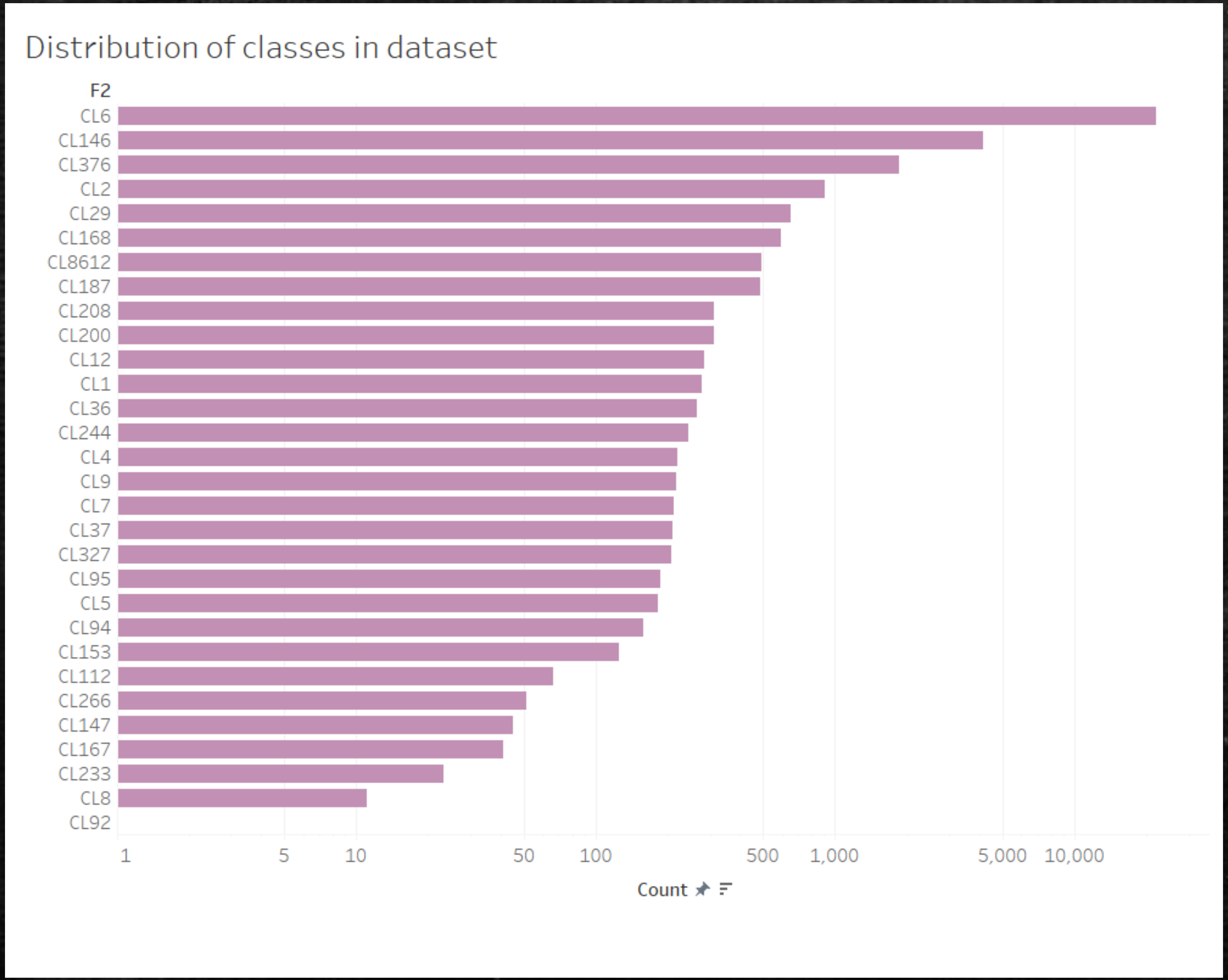
y



Multi Class Problem!

Data

y



Multi Class Problem!
Highly Imbalance !!

- Stratify while splitting
- Synthetically model the minority the classes

Data

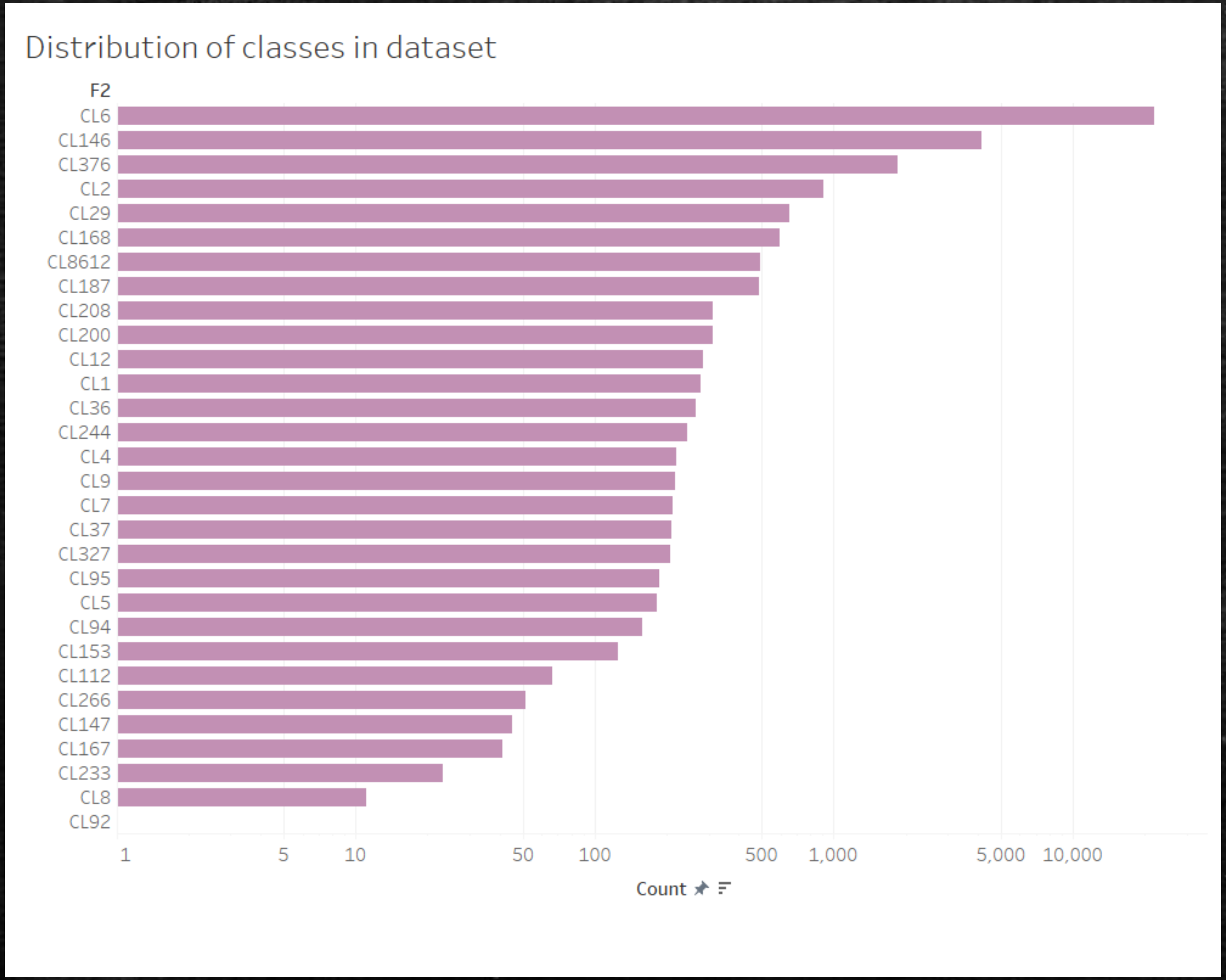
y

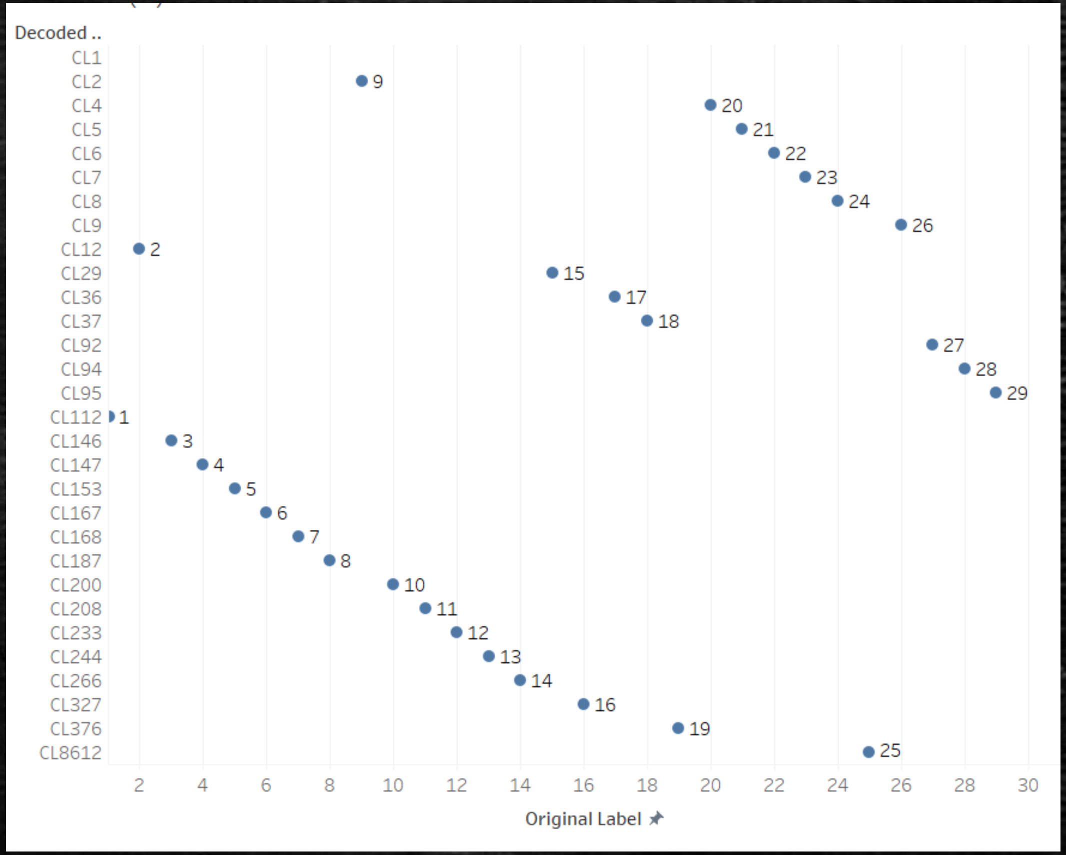
Multi Class Problem!
Highly Imbalance !!
Identify the metric – ' F1 – score' : average

Data

y

Multi Class Problem!
Highly Imbalance !!
Identify the metric – ' F1 – score' : average
Encode the labels





Data

X

34553 samples, 7672 features

Data

X

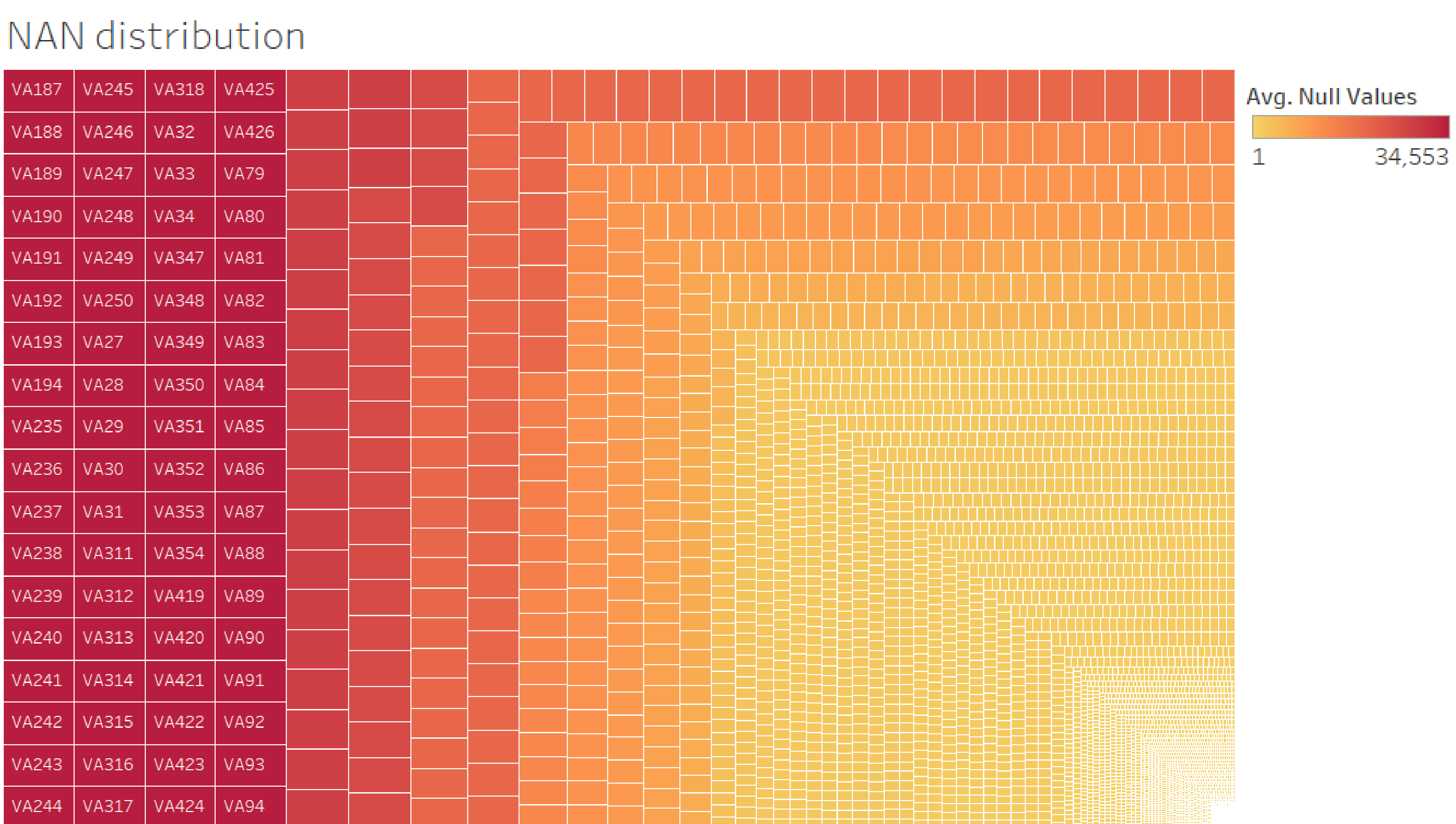
```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34553 entries, 0 to 34552
Columns: 7673 entries, GR1 to Class
dtypes: category(1), float64(7672)
memory usage: 2.0 GB
```

34553 samples, 7672 features
- Features – numeric
- No duplicated value

X – a lot of features, NANS

34553 samples, 7672 features
- Features – numeric
- No duplicated value

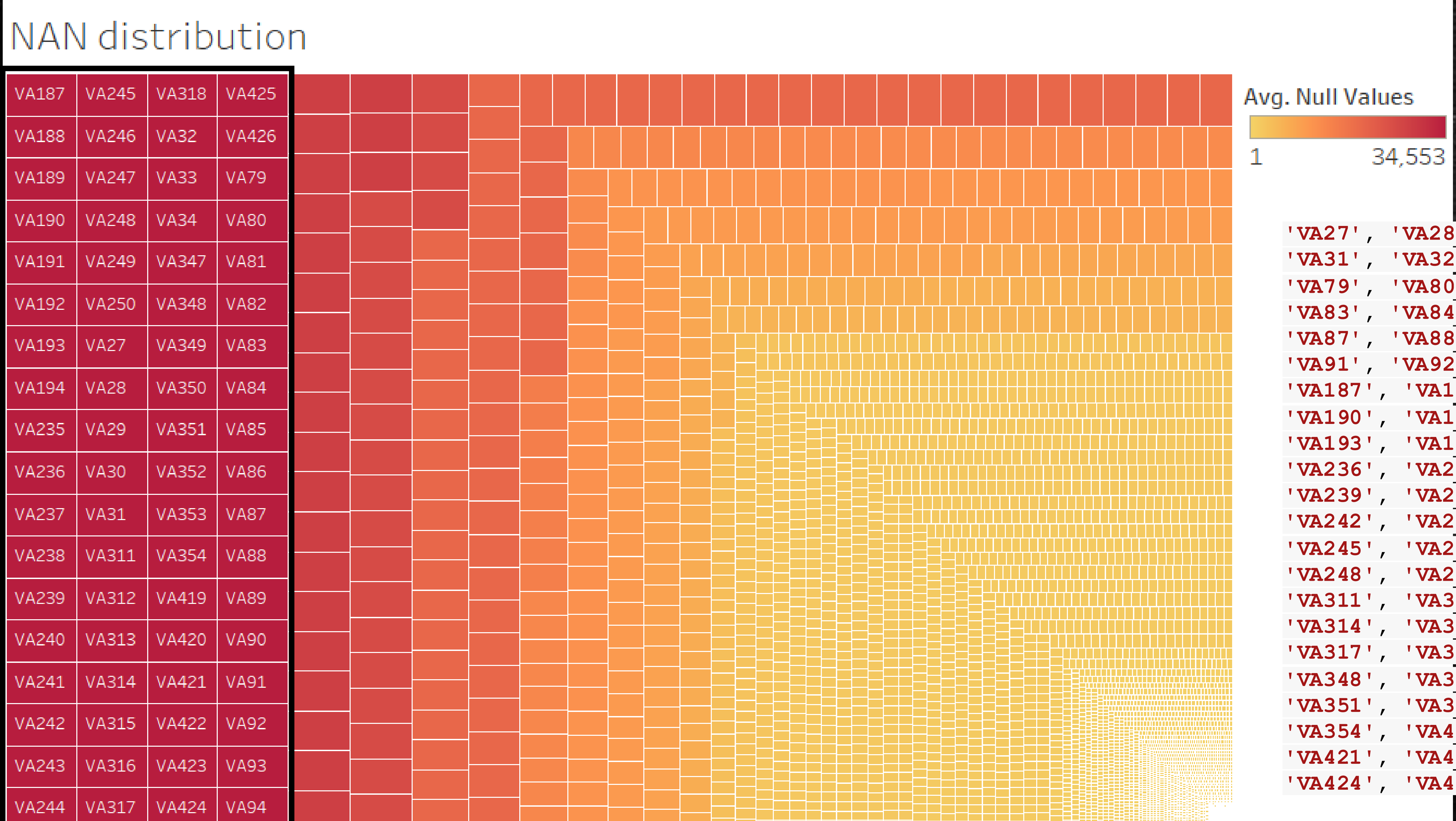
- NAN - Values



X – a lot of features, NANS

34553 samples, 7672 features
- Features – numeric
- No duplicated value

- NAN – Values – need to handle NANs properly



Dropped following cols

```
'VA27', 'VA28', 'VA29', 'VA30',
'VA31', 'VA32', 'VA33', 'VA34',
'VA79', 'VA80', 'VA81', 'VA82',
'VA83', 'VA84', 'VA85', 'VA86',
'VA87', 'VA88', 'VA89', 'VA90',
'VA91', 'VA92', 'VA93', 'VA94',
'VA187', 'VA188', 'VA189'
'VA190', 'VA191', 'VA192'
'VA193', 'VA194', 'VA235',
'VA236', 'VA237', 'VA238',
'VA239', 'VA240', 'VA241',
'VA242', 'VA243', 'VA244',
'VA245', 'VA246', 'VA247',
'VA248', 'VA249', 'VA250',
'VA311', 'VA312', 'VA313',
'VA314', 'VA315', 'VA316',
'VA317', 'VA318', 'VA347',
'VA348', 'VA349', 'VA350',
'VA351', 'VA352', 'VA353',
'VA354', 'VA419', 'VA420',
'VA421', 'VA422', 'VA423',
'VA424', 'VA425', 'VA426']
```

X – a lot of features, NANS, cols with unique value count 1 or 2

Dropped following cols. They had only one unique value.

Features : 7672 -> 7539

ME2	0.1158
GR2	0.1158
ME314	0.0000
ME313	0.0000
ME312	0.0000
ME311	0.0000
GR352	0.0000
GR351	0.0000
GR348	0.0000
GR347	0.0000
GR34	0.0000
GR33	0.0000
GR32	0.0000
GR318	0.0000
GR317	0.0000
GR316	0.0000
GR315	0.0000
GR31	0.0000
GR146	0.0000
GR145	0.0000
GR142	0.0000
GR141	0.0000
VAZ18	
VAZ16	

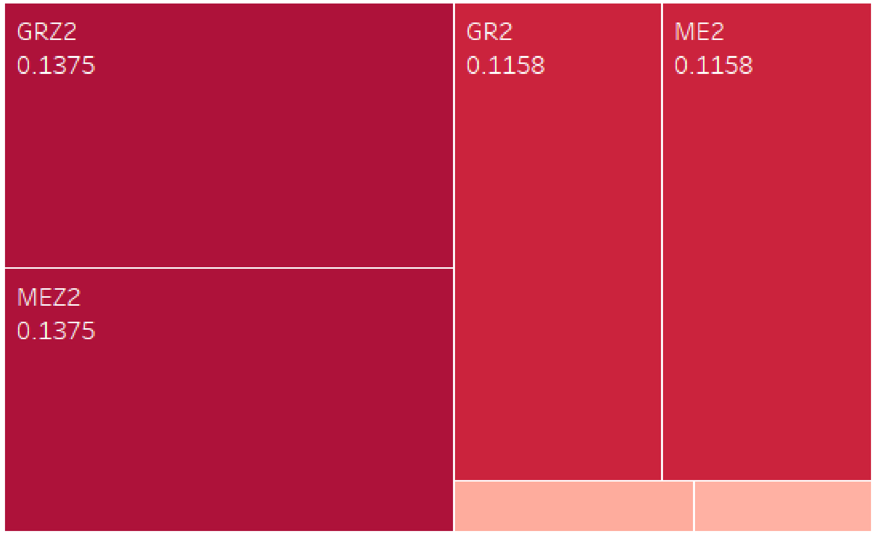
PL2
PL27
PL28
PL29
PL30
PL31
PL32
PL51
PL52
PL53
PL54
PL369
PL370
PL371
PL372
PL373
PL374
PL437
PL438
PL439
PL440
VA2
VA15
VAZ132
VAZ13
VA65

VA68
VA69
VA70
VA71
VA72
VA73
VA74
VA131
VA132
VA133
VA134
VA147
VA148
VA149
VA150
VA151
VA152
VA153
VA154
VA155
VA156
VA157
VA158
VA159
VA66
VA67

VA16
VA17
VA18
VA59
VA60
VA61
VA62
VA63
VA64
VA160
VA161
VA162
VAZ15
VAZ131

They had only two unique value. Dropped 82 columns

The features with 2 unique values and their average. - 82 null values (either filled with 0 or 0 and NaN)



X – a lot of features, NANS, cols with unique value count 1 or 2

Dropped following cols. They had only one unique value.

Features : 7672 -> 7539

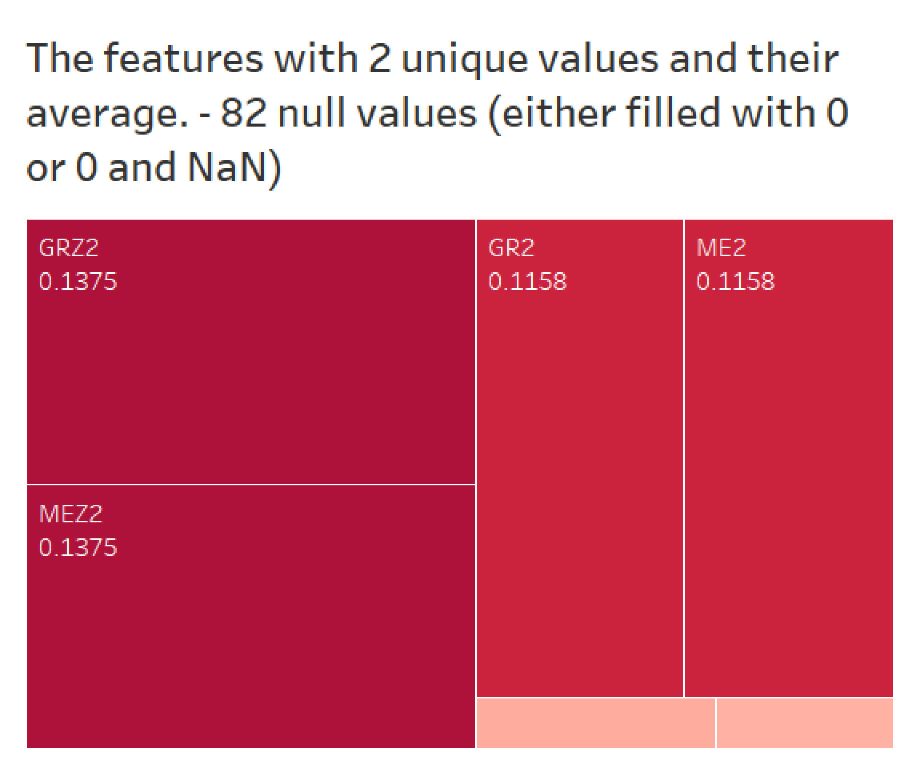
ME2	0.1158
GR2	0.1158
ME314	0.0000
ME313	0.0000
ME312	0.0000
ME311	0.0000
GR352	0.0000
GR351	0.0000
GR348	0.0000
GR347	0.0000
GR34	0.0000
GR33	0.0000
GR32	0.0000
GR318	0.0000
GR317	0.0000
GR316	0.0000
GR315	0.0000
GR31	0.0000
GR146	0.0000
GR145	0.0000
GR142	0.0000
GR141	0.0000
VAZ18	
VAZ16	

PL2
PL27
PL28
PL29
PL30
PL31
PL32
PL51
PL52
PL53
PL54
PL369
PL370
PL371
PL372
PL373
PL374
PL437
PL438
PL439
PL440
VA2
VA15
VAZ132
VAZ13
VA65

VA68
VA69
VA70
VA71
VA72
VA73
VA74
VA131
VA132
VA133
VA134
VA147
VA148
VA149
VA150
VA151
VA152
VA153
VA154
VA155
VA156
VA157
VA158
VA159
VA66
VA67

VA16
VA17
VA18
VA59
VA60
VA61
VA62
VA63
VA64
VA160
VA161
VA162
VAZ15
VAZ131

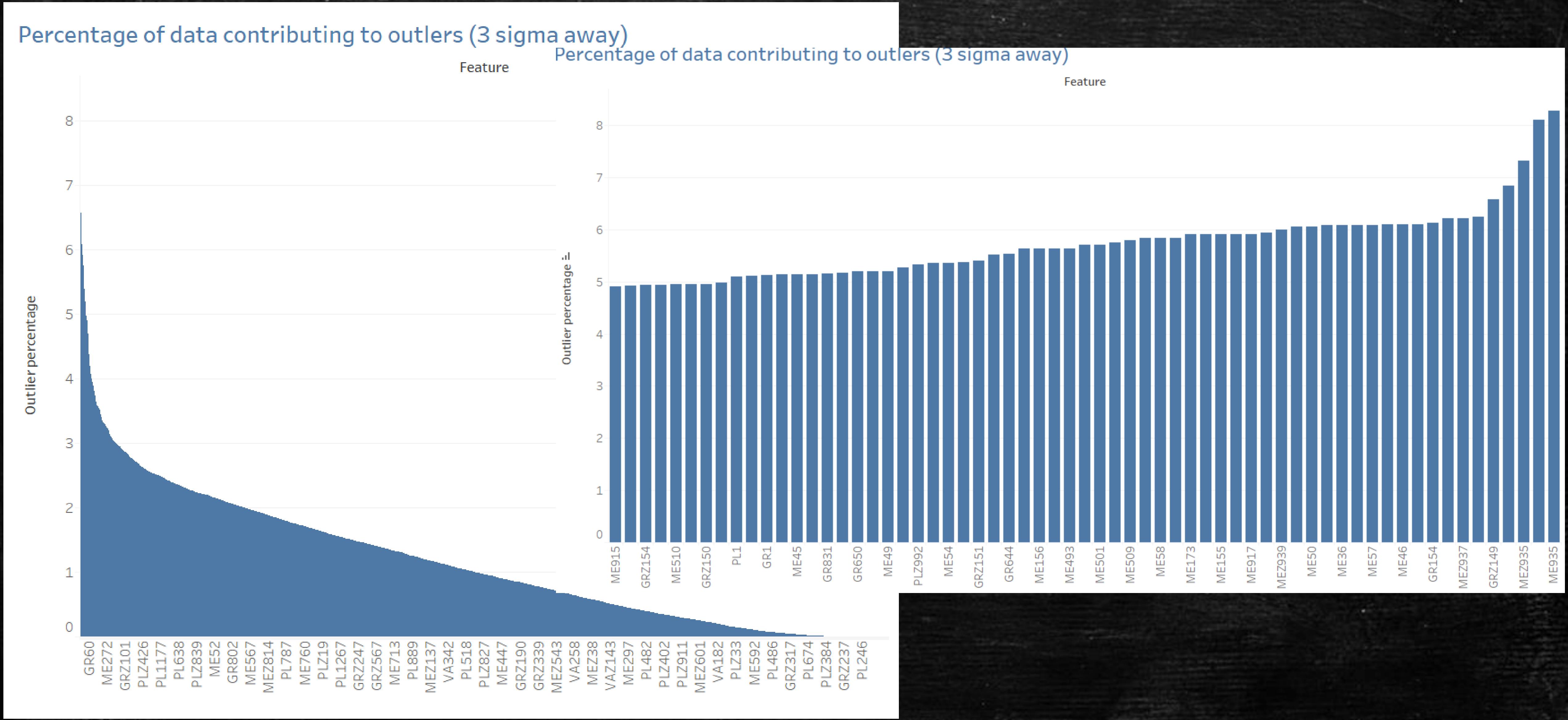
They had only two unique value. Dropped 82 columns



X – a lot of features, NANS, cols with unique value , Outliers

- Outlier – Values (3 sigma away)

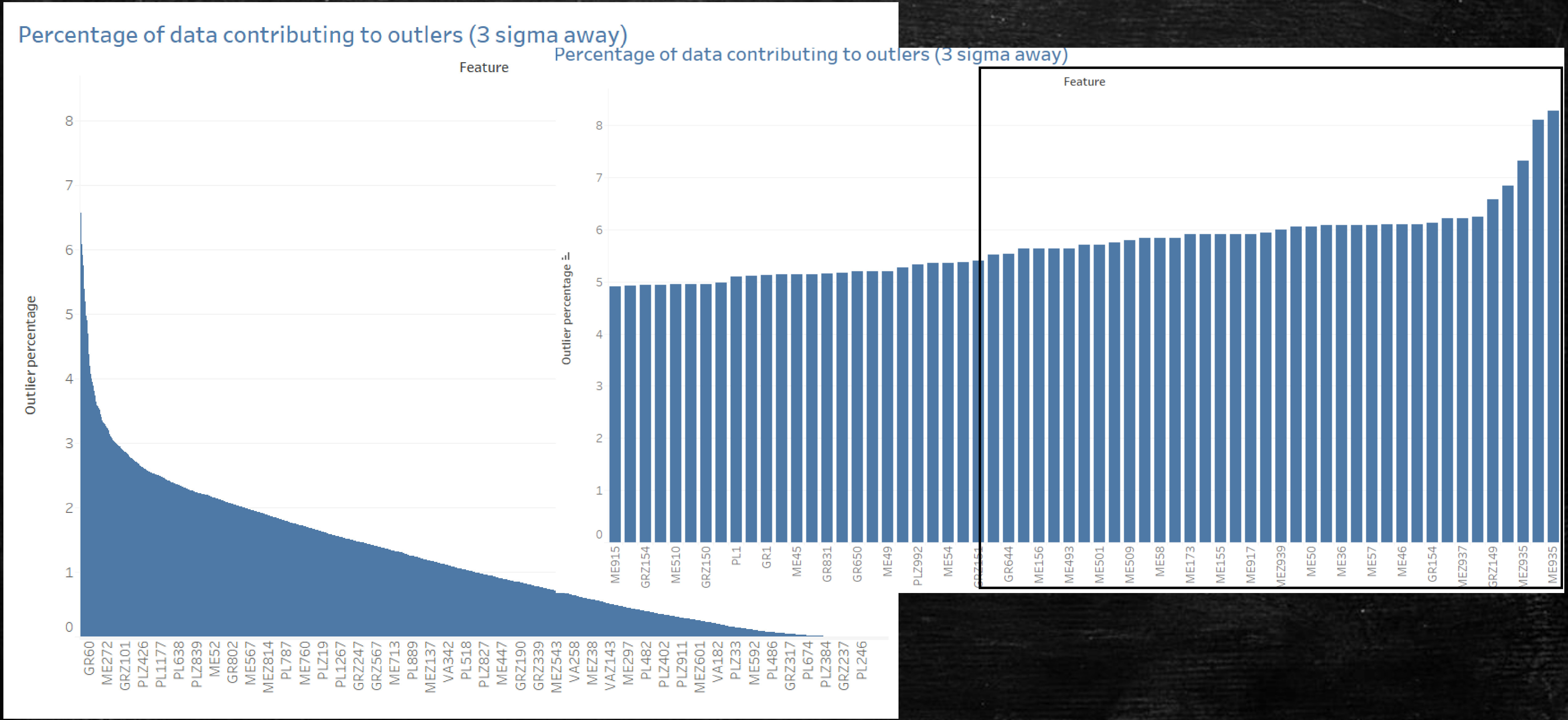
Features : 7672 -> 7539



X – a lot of features, NANS, cols with unique value , Outliers

- Outlier – Values (3 sigma away)

Features : 7672 -> 7539



X , y– split using stratification

X_train , y_train

X_test , y_test

80% : 20% stratified split

PL2
PL27
PL28
PL29
PL30
PL31
PL32
PL51
PL52
PL53
PL54
PL369
PL370
PL371
PL372
PL373
PL374
PL437
PL438
PL439
PL440
VA2
VA15
VA16
VA17
VA18
VA59
VA60
VA61
VA62
VA63
VA64
VA65
VA66
VA67
VA68
VA69
VA70
VA71
VA72
VA73
VA74
VA131
VA132
VA133
VA134
VA147
VA148
VA149
VA150
VA151
VA152
VA153
VA154
VA155
VA156
VA157
VA158
VA159
VA160
VA161
VA162
VAZ15
VAZ16
VAZ18
VAZ131
VAZ132
VAZ134

X – a lot of features, NANS

X : Imputaton

- Median
- Knn (nearest neighbor)
- Filled with zero

X – a lot of features, NANS, Scaling

X -> Imputed -> Scaled

- Robust Scaling
- MinMax Scaler
- Standard Scaler

X – a lot of features, NANS, Scaling

```python
def pre_process(df_train,  imputation, scaling, df_test):
   # Impute missing values
    if imputation == 'median':
        imputer = SimpleImputer(strategy=imputation)
        df_train_imputed = pd.DataFrame(imputer.fit_transform(df_train), columns=df_train.columns)
        df_test_imputed =  pd.DataFrame(imputer.transform(df_test), columns=df_train.columns)
        print(imputation)
    elif imputation == 'zero':
        print(imputation)
        df_train_imputed = df_train.fillna(0)
        df_test_imputed = df_test.fillna(0)

    elif imputation == 'knn':
        print(imputation)
        imputer = KNNImputer(n_neighbor = 5)
        df_train_imputed = pd.DataFrame(imputer.fit_transform(df_train), columns=df_train.columns)
        df_test_imputed =  pd.DataFrame(imputer.transform(df_test), columns=df_train.columns)

   # df_train_imputed = df_train
    if scaling == 'minmax':
        scaler = MinMaxScaler()
        X_t = scaler.fit_transform(df_train_imputed)
      #  df_train_imputed_scaled = pd.DataFrame(scaler.fit_transform(df_train), columns=df_train.
      columns)
        df_test_imputed_scaled = pd.DataFrame(scaler.transform(df_test_imputed), columns=df_train.
        columns)
    elif scaling == 'robust' :
        robust = RobustScaler()
        X_t = robust.fit_transform(df_train_imputed)
      #  df_train_imputed_scaled = pd.DataFrame(robust.fit_transform(df_train), columns=df_train.
      columns)


        df_test_imputed_scaled = pd.DataFrame(robust.transform(df_test_imputed), columns=df_test.
        columns)
    elif scaling == 'std':
        std = StandardScaler()
        df_train_imputed_scaled = pd.DataFrame(std.fit_transform(df_train_imputed), columns=df_train.
        columns)
        df_test_imputed_scaled = pd.DataFrame(std.transform(df_test_imputed), columns=df_test.columns)

        #del df_train_imputed
    else:
     #   raise ValueError("Invalid scaling method. Choose 'minmax'.")
    return  df_test_imputed_scaled, df_test_imputed

strategies = ['zero','median', 'knn' ]#'']
scalings = ['robust', 'minmax','std']#, 'robust', 'standard']
```

```python
for strat in strategies:
    for scaling in scalings:
        # Pre-process the data
        print(scaling)
        print(strat)
        df_test_imputed_scaled,df_test_imputed = pre_process(df_train = X_train,df_test = X_test,
        imputation=strat, scaling=scaling)
  #      print(f'shape : {df_train_imputed.shape}')
      #  df_train_imputed_scaled = pre_process(X_train,  scaling=scaling)
        print(f'shape : {df_test_imputed_scaled.shape}')

        # Save the pre-processed data
        #df_train_imputed.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_train_impute
        {imputation}_{scaling}.csv', index=False)
        #print('file 1 saved')
        df_test_imputed_scaled.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/
        X_test_imputed_{strat}_zero_{scaling}.csv', index=False)
        df_test_imputed.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_test_imputed_
        {strat}.csv', index=False)

        print('file2 saved')
exit()
```

X – a lot of features, NANS, Scaling

```python
def pre_process(df_train,  imputation, scaling, df_test):
    # Impute missing values
    if imputation == 'median':
        imputer = SimpleImputer(strategy=imputation)
        df_train_imputed = pd.DataFrame(imputer.fit_transform(df_train), columns=df_train.columns)
        df_test_imputed =  pd.DataFrame(imputer.transform(df_test), columns=df_train.columns)
        print(imputation)
    elif imputation == 'zero':
        print(imputation)
        df_train_imputed = df_train.fillna(0)
        df_test_imputed = df_test.fillna(0)

    elif imputation == 'knn':
        print(imputation)
        imputer = KNNImputer(n_neighbor = 5)
        df_train_imputed = pd.DataFrame(imputer.fit_transform(df_train), columns=df_train.columns)
        df_test_imputed =  pd.DataFrame(imputer.transform(df_test), columns=df_train.columns)

    # df_train_imputed = df_train
    if scaling == 'minmax':
        scaler = MinMaxScaler()
        X_t = scaler.fit_transform(df_train_imputed)
      #  df_train_imputed_scaled = pd.DataFrame(scaler.fit_transform(df_train), columns=df_train.columns)
        df_test_imputed_scaled = pd.DataFrame(scaler.transform(df_test_imputed), columns=df_train.columns)
    elif scaling == 'robust' :
        robust = RobustScaler()
        X_t = robust.fit_transform(df_train_imputed)
      #  df_train_imputed_scaled = pd.DataFrame(robust.fit_transform(df_train), columns=df_train.columns)


        df_test_imputed_scaled = pd.DataFrame(robust.transform(df_test_imputed), columns=df_test.columns)
    elif scaling == 'std':
        std = StandardScaler()
        df_train_imputed_scaled = pd.DataFrame(std.fit_transform(df_train_imputed), columns=df_train.columns)
        df_test_imputed_scaled = pd.DataFrame(std.transform(df_test_imputed), columns=df_test.columns)

        #del df_train_imputed
    else:
     #   raise ValueError("Invalid scaling method. Choose 'minmax'.")
    return  df_test_imputed_scaled, df_test_imputed

strategies = ['zero','median', 'knn' ]#'']
scalings = ['robust', 'minmax','std']#, 'robust', 'standard']

```

```python
for strat in strategies:
    for scaling in scalings:
        # Pre-process the data
        print(scaling)
        print(strat)
        df_test_imputed_scaled,df_test_imputed = pre_process(df_train = X_train,df_test = X_test, imputation=strat, scaling=scaling)
  #      print(f'shape : {df_train_imputed.shape}')
      #  df_train_imputed_scaled = pre_process(X_train,  scaling=scaling)
        print(f'shape : {df_test_imputed_scaled.shape}')

        # Save the pre-processed data
        #df_train_imputed.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_train_impute{imputation}_{scaling}.csv', index=False)
        #print('file 1 saved')
        df_test_imputed_scaled.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_test_imputed_{strat}_zero_{scaling}.csv', index=False)
        df_test_imputed.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_test_imputed_{strat}.csv', index=False)

        print('file2 saved')
exit()
```

X – Imputed, Scaled, CURSE OF DIMENSIONALITY

X -> Imputed -> Scaled

- Feature Extraction
- Feature selection

X – Imputed, Scaled, CURSE OF DIMENSIONALITY

X -> Imputed -> Scaled

Feature Extraction

- Variance Threshold
- Principal Component Analysis (using different nComponents)
- Kernel PCA

```
PCA : N Components = 876, 1500, 3000
kPCA : 875, 500
```

X – Imputed, Scaled, CURSE OF DIMENSIONALITY

```python
def dim_reduction(df_train, df_test, dimred_proc, nComp ):
    if dimred_proc == 'pca':
        pca_init = PCA(n_components = nComp)
        x_pca = pca_init.fit_transform(df_train)
        x_test = pca_init.transform(df_test)
       # pk.dump(x_pca, open(f"/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/p
        x_pca_df = pd.DataFrame(x_test, columns=[f'PC{i+1}' for i in range(nComp)])

    return x_pca_df
nComponents = [800,1500, 3000]
for comp in nComponents:
        print(comp)
        x_pca = dim_reduction(X_train_imputed_scaled,X_test, 'pca', comp)
        x_pca.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_test_im
        print('file 1 saved')
        del x_pca
print('done')
#import pickle
#print('loading pkl')
## Load the pickled object from the .pkl file
#with open('/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/pca_800.pkl', 'rb')
#    pca = pickle.load(f)
#
#print('loaded achar')# Run the loaded object
#result = pca.transform(X_test)
#print('result ')
#X_test_pca800 = pd.DataFrame(result, columns=[f'PC{i+1}' for i in range(800)])
```

```python
def dim_reduction(df_train,  dimred_proc, nComp ):
    if dimred_proc == 'kpca':
        kpca = KernelPCA(n_components=nComp, kernel='rbf', gamma = 0.00001)

# Fit and transform the data
        X_kpca = kpca.fit_transform(df_train)

        pk.dump(X_kpca, open(f"/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/k)
        x_pca_df = pd.DataFrame(X_kpca, columns=[f'PC{i+1}' for i in range(nComp)])

    return x_pca_df

nComponents = [500,875]
for comp in nComponents:
        print(comp)
        x_pca = dim_reduction(X_train_imputed_scaled, 'kpca', comp)
        x_pca.to_csv(f'/mnt/gpfs3_amd/scratch/rgu245/intel/processed_files/X_train_i
        print('file 1 saved')
        del x_pca
print('done')
```

X – Imputed, Scaled, Train Model

X -> Imputed -> Scaled

Feature Extraction

Train different models!

```
PCA : N Components = 875, 1500, 3000
kPCA : 875, 500
```

X – Imputed, Scaled, CURSE OF DIMENSIONALITY

X -> Imputed -> Scaled

Feature selection

Model selection using

- Random Forest
- XGBoost
- SHAP values

X – Imputed, Scaled, CURSE OF DIMENSIONALITY

X -> Imputed -> Scaled

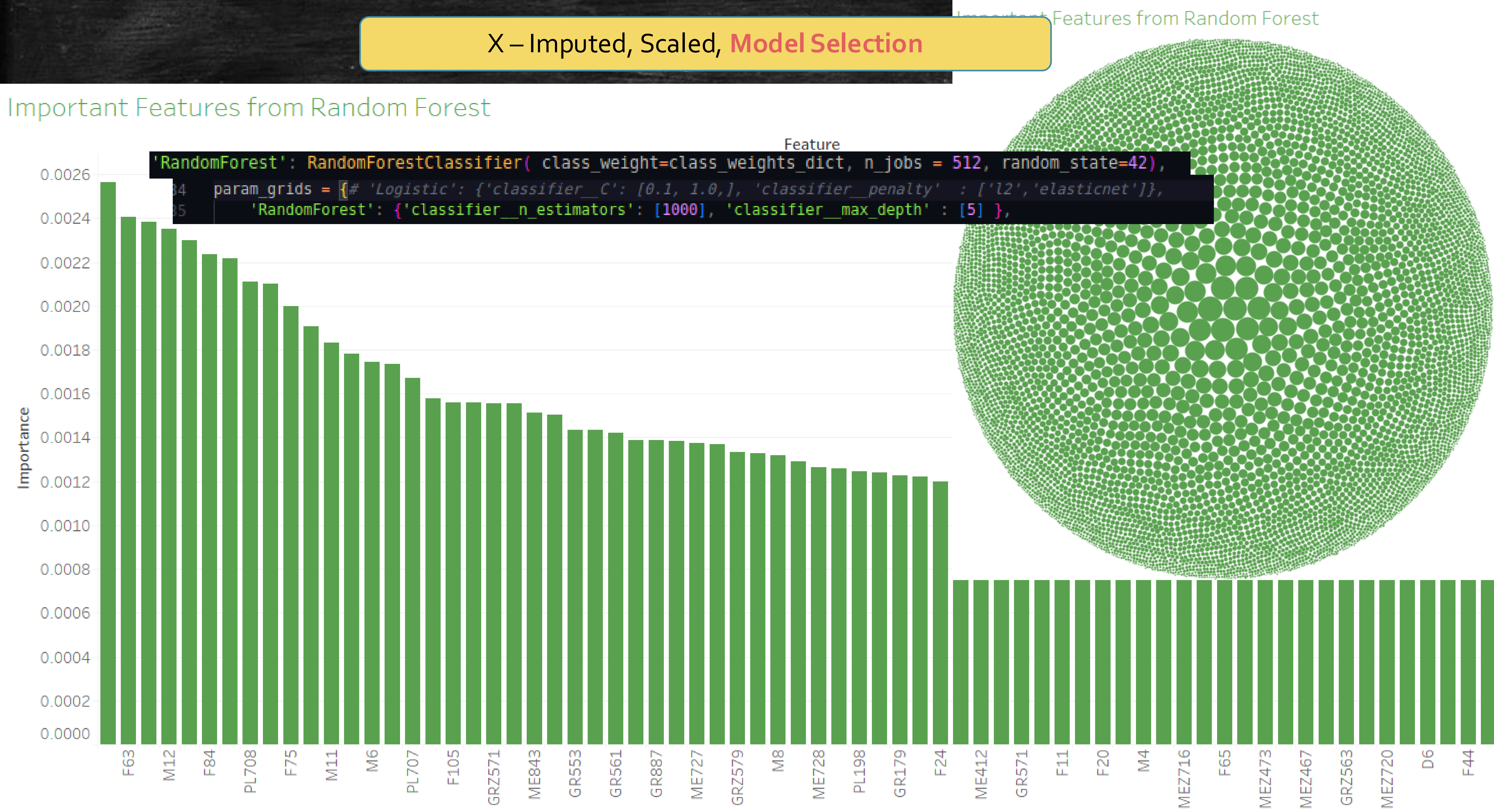
Feature selection

Model selection using

- Random Forest
- XGBoost
- SHAP values

X – Imputed, Scaled, Model Selection

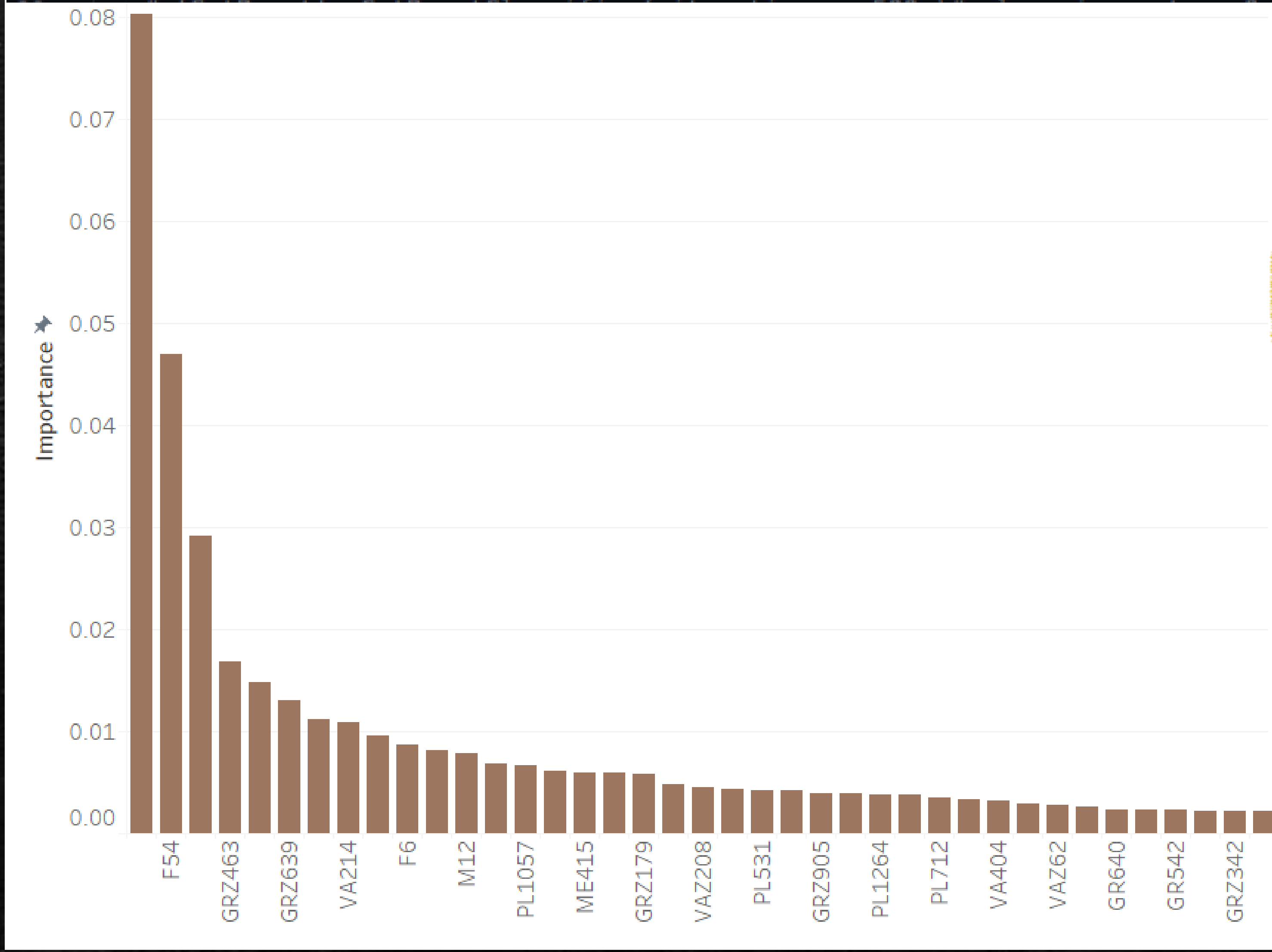
```
'RandomForest': RandomForestClassifier( class_weight=class_weights_dict, n_jobs = 512, random_state=42),
param_grids = {# 'Logistic': {'classifier__C': [0.1, 1.0,], 'classifier__penalty'  : ['l2','elasticnet']},
    'RandomForest': {'classifier__n_estimators': [1000], 'classifier__max_depth' : [5] },
```

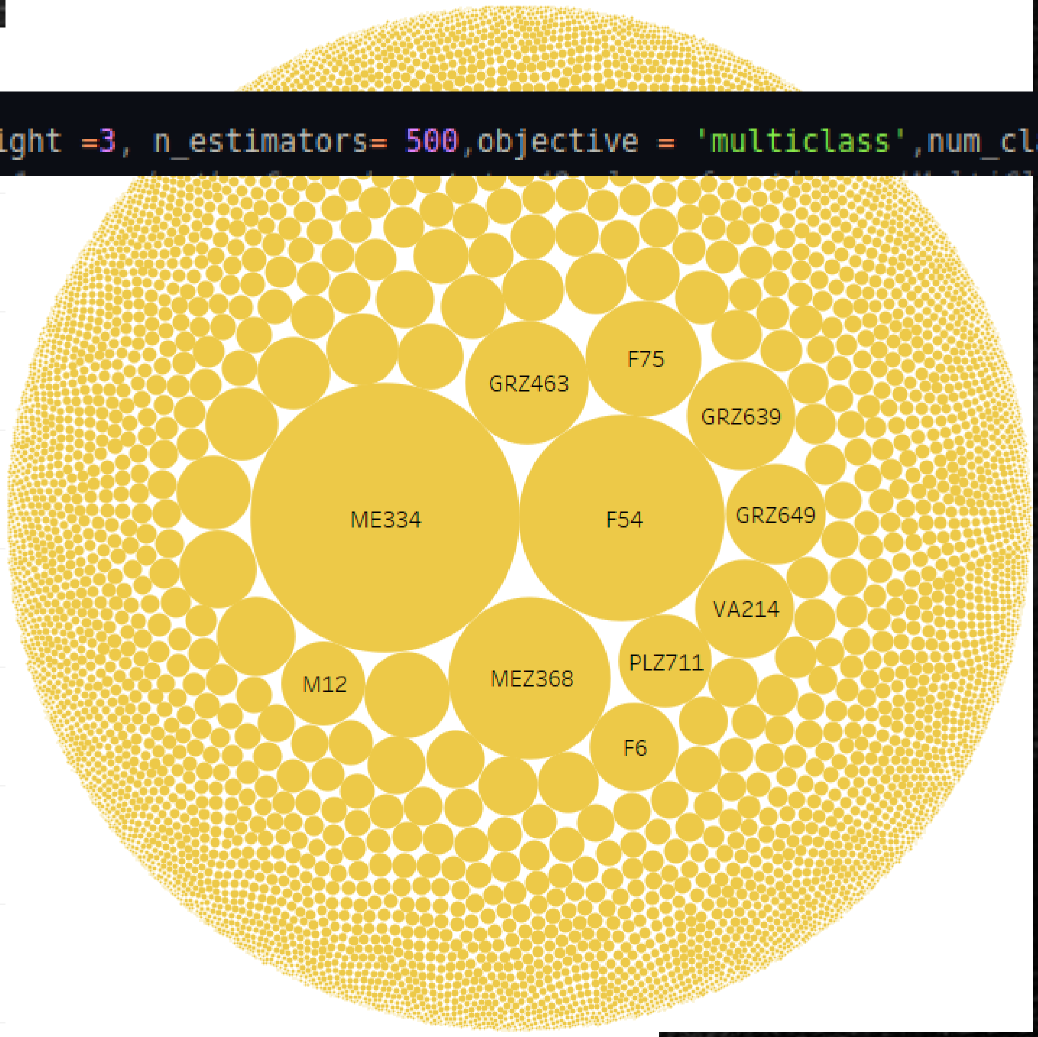


Feature Importance from XGB-boost

Feature Importance from XGB-boost

```
    'XGBoost': XGBClassifier(learning_rate =0.1, max_depth =6, min_child_weight =3, n_estimators= 500,objective = 'multiclass',num_cla
```





X – Imputed, Scaled, Features Reduced , TRAIN, stratified y

```python
# Compute class weights based on inverse class frequency
class_labels = np.unique(y_train_noC)  # Assuming y_train contains the target labels
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(y_train_noC), y=y_train_noC)
class_weights_dict = dict(zip(class_labels, class_weights))

print('dataset loaded')
```

Class weight distribution was estimated

```python
#-------------CLASSIFIERS------------------#
classifiers = {
    'Logistic': LogisticRegression(class_weight=class_weights_dict,multi_class='ovr',random_state=42),
    'RandomForest': RandomForestClassifier( class_weight=class_weights_dict, n_jobs = 512, random_state=42),
    'XGBoost': XGBClassifier(objective = 'multiclass',num_class = 29, random_state=42),
    'CatBoost': CatBoostClassifier( random_state=42, loss_function = 'MultiClass'),
    'NaiveBayes': GaussianNB(),
    'SVM': SVC(random_state=42),
    #'LightGBM': LGBMClassifier(objective = 'multiclass', class_weight = class_weights_dict,n_jobs = 512, random_state=42),
}


 param_grids = { 'Logistic': {'classifier__C': [0.1, 1.0,2.0], 'classifier__penalty'  : ['l2','elasticnet']},
     'RandomForest': {'classifier__n_estimators': [100,500,1000], 'classifier__max_depth' : [10,5,3,15] },

     'XGBoost': {'classifier__n_estimators': [100, 500, 1000], 'classifier__learning_rate': [0.1, 1.0, 10], 'classifier__max_depth' : [3,6, 9], 'classifier__min_child_weight' : [1,3]},# 'classifier__eval_metric' = ['merror', 'mlogloss']},

     'CatBoost': {'classifier__iterations': [100, 500, 1000], 'classifier__learning_rate': [0.1, 1.0, 0.5], 'classifier__max_depth' : [3,6, 9] },
    'NaiveBayes': {},  # Naive Bayes does not have hyperparameters to tune
     'SVM': {'classifier__C': [1.0, 10.0], 'classifier__decision_function_shape' : ['ovo', 'ovr']},
     'LightGBM': {'classifier__n_estimators': [100, 500], 'classifier__learning_rate': [ 0.1, 1.0], 'classifier__max_depth' : [3,5]},
 }
```

X – Imputed, Scaled, Features Reduced , TRAIN, stratified y

```
# Compute class weights based on inverse class frequency
class_labels = np.unique(y_train_noC)  # Assuming y_train contains the target labels
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(y_train_noC), y=y_train_noC)
class_weights_dict = dict(zip(class_labels, class_weights))

print('dataset loaded')
```

```
#-------------CLASSIFIERS------------------#
classifiers = {
    'Logistic': LogisticRegression(class_weight=class_weights_dict,multi_class='ovr',random_state=42),
    'RandomForest': RandomForestClassifier( class_weight=class_weights_dict, n_jobs = 512, random_state=42),
    'XGBoost': XGBClassifier(objective = 'multiclass',num_class = 29, random_state=42),
    'CatBoost': CatBoostClassifier( random_state=42, loss_function = 'MultiClass'),
    'NaiveBayes': GaussianNB(),
    'SVM': SVC(random_state=42),
    #'LightGBM': LGBMClassifier(objective = 'multiclass', class_weight = class_weights_dict,n_jobs = 512,
    random_state=42),
}

param_grids = { 'Logistic': {'classifier__C': [0.1, 1.0,2.0], 'classifier__penalty'  : ['l2','elasticnet']},
    'RandomForest': {'classifier__n_estimators': [100,500,1000], 'classifier__max_depth' : [10,5,3,15] },

    'XGBoost': {'classifier__n_estimators': [100, 500, 1000], 'classifier__learning_rate': [0.1, 1.0, 10],
    'classifier__max_depth' : [3,6, 9], 'classifier__min_child_weight' : [1,3]},# 'classifier__eval_metric' =
    ['merror', 'mlogloss']},

    'CatBoost': {'classifier__iterations': [100, 500, 1000], 'classifier__learning_rate': [0.1, 1.0, 0.5],
    'classifier__max_depth' : [3,6, 9] },
   'NaiveBayes': {},  # Naive Bayes does not have hyperparameters to tune
    'SVM': {'classifier__C': [1.0, 10.0], 'classifier__decision_function_shape' : ['ovo', 'ovr']},
    'LightGBM': {'classifier__n_estimators': [100, 500], 'classifier__learning_rate': [ 0.1, 1.0],
    'classifier__max_depth' : [3,5]},
}
```

Different classifiers were defined.

X – Imputed, Scaled, Features Reduced , TRAIN, stratified y

```python
# Compute class weights based on inverse class frequency
class_labels = np.unique(y_train_noC)  # Assuming y_train contains the target labels
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(y_train_noC), y=y_train_noC)
class_weights_dict = dict(zip(class_labels, class_weights))

print('dataset loaded')
```

```python
#-------------CLASSIFIERS------------------#
classifiers = {
    'Logistic': LogisticRegression(class_weight=class_weights_dict,multi_class='ovr',random_state=42),
    'RandomForest': RandomForestClassifier( class_weight=class_weights_dict, n_jobs = 512, random_state=42),
    'XGBoost': XGBClassifier(objective = 'multiclass',num_class = 29, random_state=42),
    'CatBoost': CatBoostClassifier( random_state=42, loss_function = 'MultiClass'),
    'NaiveBayes': GaussianNB(),
    'SVM': SVC(random_state=42),
    #'LightGBM': LGBMClassifier(objective = 'multiclass', class_weight = class_weights_dict,n_jobs = 512,
    random_state=42),
}

param_grids = { 'Logistic': {'classifier__C': [0.1, 1.0,2.0], 'classifier__penalty'  : ['l2','elasticnet']},
    'RandomForest': {'classifier__n_estimators': [100,500,1000], 'classifier__max_depth' : [10,5,3,15] },

    'XGBoost': {'classifier__n_estimators': [100, 500, 1000], 'classifier__learning_rate': [0.1, 1.0, 10],
    'classifier__max_depth' : [3,6, 9], 'classifier__min_child_weight' : [1,3]},# 'classifier__eval_metric' =
    ['merror', 'mlogloss']},

    'CatBoost': {'classifier__iterations': [100, 500, 1000], 'classifier__learning_rate': [0.1, 1.0, 0.5],
    'classifier__max_depth' : [3,6, 9] },
   'NaiveBayes': {},  # Naive Bayes does not have hyperparameters to tune
    'SVM': {'classifier__C': [1.0, 10.0], 'classifier__decision_function_shape' : ['ovo', 'ovr']},
    'LightGBM': {'classifier__n_estimators': [100, 500], 'classifier__learning_rate': [ 0.1, 1.0],
    'classifier__max_depth' : [3,5]},
}
```

Different parameters were tuned

X – Imputed, Scaled, Features Reduced , TRAIN

```python
#------------GRIDSEARCH---------------------#
nSplit = 5
print(f'nSplit : {nSplit}')
# initialize StratifiedKFold cross-validator
stratified_kfold = StratifiedKFold(n_splits=nSplit, shuffle=True, random_state=42)
results = []
# Fit and save the best model for each classifier
for metrics_name, metrictype in scorers.items():
    print(f'----{metrics_name}---')
    metric_results = []
    for name, pipeline in pipelines.items():
    # Define the parameter grid for the current classifier
        param_grid = param_grids.get(name, {})

    #for clf_name, clf in classifiers.items():
        print(f'----{name}---')

         # Get parameter grid for current classifier

        grid_search = GridSearchCV(pipeline, param_grid, cv=stratified_kfold, scoring=metrictype, n_jobs=512,
        verbose=4)
        grid_search.fit(X_train, y_train)
       # print(len(y_predicted))

    # Save the best model
        print('grid-search over')
        best_model = grid_search.best_estimator_
        print('best Model')
        dump(best_model, f'/mnt/gpfs3_amd/scratch/rgu245/intel/final/dim-red/models-best-params/f1/nsplit1{name}
        best_model_{metrics_name}_rob_med_pca1500.joblib')
         # Append results to the list for current metric
        metric_results.append({'Classifier': name,
                               'Metric': metrics_name,
                               'Best Score': grid_search.best_score_,
                               'Best Parameters': grid_search.best_params_})
        print(f'metric Result
        ----------------------------------------------------------------------------------------------------
        -{metric_results}')
```

- Jobs were run in parallel
- Grid search was performed across Stratified kFold to get the best parameters

```
matplotlib successful
libs imported
files loaded
dataset loaded
nSplit : 5
----f1_weighted---
----RandomForest---
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.605 total time=  18.2s
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.578 total time=  21.1s
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.606 total time=  35.1s
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.655 total time=  40.4s
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.619 total time=  41.2s
[CV 2/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.632 total time=  42.2s
F1 score on validation fold: 0.761370265637153
Fitting 5 folds for each of 9 candidates, totalling 45 fits
F1 score on validation fold: 0.7591858793118769
Fitting 5 folds for each of 9 candidates, totalling 45 fits
F1 score on validation fold: 0.7581896563000856
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.642 total time=  50.7s
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.593 total time=  52.3s
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.658 total time=  54.7s
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.654 total time= 1.0min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.746 total time= 1.0min
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.609 total time= 1.0min
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.606 total time= 1.1min
[CV 2/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.649 total time= 1.1min
F1 score on validation fold: 0.7618506629340698
Fitting 5 folds for each of 9 candidates, totalling 45 fits
F1 score on validation fold: 0.7624971668562314
Average F1 score for RandomForest: 0.7606187262078834
```

```
ifier__max_depth=5, classifier__n_estimators=500;, score=0.643 total time= 1.5min
ifier__max_depth=3, classifier__n_estimators=500;, score=0.600 total time=  52.4s
ifier__max_depth=10, classifier__n_estimators=500;, score=0.757 total time= 1.5min
ifier__max_depth=5, classifier__n_estimators=100;, score=0.632 total time=  16.9s
ifier__max_depth=10, classifier__n_estimators=100;, score=0.740 total time= 1.5min
ifier__max_depth=10, classifier__n_estimators=100;, score=0.749 total time= 1.6min
ifier__max_depth=3, classifier__n_estimators=500;, score=0.604 total time= 1.5min
ifier__max_depth=3, classifier__n_estimators=500;, score=0.596 total time=  29.3s
ifier__max_depth=10, classifier__n_estimators=500;, score=0.764 total time= 1.6min
ifier__max_depth=10, classifier__n_estimators=1000;, score=0.750 total time= 1.8min
ifier__max_depth=5, classifier__n_estimators=500;, score=0.667 total time= 1.6min
ifier__max_depth=5, classifier__n_estimators=1000;, score=0.682 total time= 1.7min
ifier__max_depth=3, classifier__n_estimators=1000;, score=0.603 total time= 1.6min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.760 total time= 1.6min
[CV 4/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.621 total time= 1.7min
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.609 total time= 1.7min
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.619 total time= 1.7min
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.594 total time=  11.5s
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.667 total time= 1.7min
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.604 total time= 1.3min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.756 total time= 1.6min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.751 total time= 1.8min
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.673 total time= 1.7min
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.597 total time= 1.3min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.754 total time= 1.7min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.767 total time= 1.8min
[CV 2/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.654 total time= 1.7min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.734 total time= 1.4min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.774 total time= 1.7min
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.585 total time=  12.8s
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.628 total time= 1.7min
[CV 4/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.628 total time= 1.7min
[CV 4/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.614 total time= 1.7min
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.754 total time= 1.7min
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.600 total time= 1.8min
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.616 total time= 1.0min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.773 total time= 1.8min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.759 total time= 1.8min
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.670 total time= 1.8min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.755 total time= 1.8min
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.632 total time= 1.8min
[CV 4/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.635 total time= 1.4min
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.756 total time= 1.8min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.738 total time= 1.4min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.763 total time= 1.8min
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.609 total time=  11.9s
[CV 4/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.644 total time= 1.8min
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.732 total time= 1.3min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.768 total time= 1.8min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.717 total time= 1.4min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.756 total time= 1.8min
```

```
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.587 total time= 1.4min
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.588 total time= 1.4min
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.621 total time= 1.5min
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.591 total time= 1.5min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.727 total time= 1.5min
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.617 total time= 1.5min
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.627 total time= 1.5min
[CV 4/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.604 total time= 1.5min
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.610 total time= 1.5min
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.593 total time= 1.5min
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.752 total time= 1.5min
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.592 total time= 1.5min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.741 total time= 1.5min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.753 total time= 1.5min
[CV 2/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.653 total time= 1.5min
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.619 total time= 1.5min
[CV 4/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.635 total time= 1.5min
[CV 2/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.659 total time= 1.6min
[CV 3/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.621 total time= 1.6min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.752 total time= 1.6min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=500;, score=0.754 total time= 1.6min
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.627 total time= 1.8min
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.755 total time= 1.8min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.752 total time= 1.8min
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.752 total time= 1.8min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.752 total time= 1.8min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.745 total time= 1.8min
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.593 total time= 1.1min
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.576 total time=   0.9s
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.600 total time= 1.2min
[CV 4/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.581 total time= 1.3min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.745 total time= 1.3min
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.728 total time= 1.3min
[CV 4/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.604 total time=  18.8s
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.745 total time= 1.3min
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.604 total time=  19.7s
[CV 4/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.745 total time= 1.3min
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.588 total time=  27.4s
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.663 total time= 1.3min
[CV 2/5] END classifier__max_depth=3, classifier__n_estimators=100;, score=0.615 total time=  24.8s
[CV 2/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.743 total time= 1.8min
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.730 total time=  37.1s
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.643 total time= 1.4min
[CV 5/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.596 total time=  45.1s
[CV 3/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.603 total time= 1.7min
[CV 1/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.749 total time=  48.8s
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=1000;, score=0.657 total time= 1.6min
[CV 5/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.625 total time=  47.7s
[CV 4/5] END classifier__max_depth=5, classifier__n_estimators=500;, score=0.622 total time= 1.5min
[CV 1/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.647 total time=  49.2s
[CV 5/5] END classifier__max_depth=10, classifier__n_estimators=1000;, score=0.760 total time= 1.8min
[CV 3/5] END classifier__max_depth=10, classifier__n_estimators=100;, score=0.738 total time=  54.0s
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=1000;, score=0.606 total time= 1.3min
[CV 1/5] END classifier__max_depth=3, classifier__n_estimators=500;, score=0.597 total time=  56.1s
[CV 4/5] END classifier__max_depth=5, classifier__n_estimators=100;, score=0.619 total time= 1.4min
```

X – Imputed, Scaled, Features Reduced , TRAIN**, MODEL EVALUATION (Best parameters)**

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg auc-roc score
0	minmax	1500	pca	logistic	0.958007458
0	minmax	1500	pca	RandomForest	0.938632022
0	minmax	1500	pca	XGBoost	0.970529713
0	minmax	1500	pca	CatBoost	0.976035182
0	minmax	1500	pca	Naive Bayes	0.874

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg auc-roc score
0	minmax	875	kpca	Logistic	0.895272466
0	minmax	875	kpca	RandomForest	0.950761852
0	minmax	875	kpca	XGBoost	0.972538066
0	minmax	875	kpca	CatBoost	0.976813318
0	minmax	875	kpca	Naive Bayes	0.876

X – Imputed, Scaled, Features Reduced , TRAIN, **MODEL EVALUATION (Best parameters)**

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg F1 score
median	robust	pca	1500	RandomForest	0.755166382
median	robust	pca	1500	XGBoost	0.871753945
median	robust	pca	1500	CatBoost	0.858095323
median	robust	pca	1500	Naive Bayes	0.783
median	robust	pca	1500	SVM	0.922

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg F1 score
0	stdscaled	3000	pca	RandomForest	0.831177229
0	stdscaled	3000	pca	XGBoost	0.868948278
0	stdscaled	3000	pca	CatBoost	0.873476535
0	stdscaled	3000	pca	Naive Bayes	0.855489329
0	stdscaled	3000	pca	svm	0

X – Imputed, Scaled, Features Reduced , TRAIN, **MODEL EVALUATION (Best parameters)**

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg F1 score
0	minmax	kpca	875	RandomForest	0.828768
0	minmax	kpca	875	XGBoost	0.889695
0	minmax	kpca	875	CatBoost	0.886021
0	minmax	kpca	875	Naive Bayes	0.783
0	minmax	kpca	875	SVM	0.922

Imputed	Scaling	nDimensions	Dim Reduction	Model	Avg F1 score
0	minmax	1500	pca	RandomForest	0.819196863
0	minmax	1500	pca	XGBoost	0.887455895
0	minmax	1500	pca	CatBoost	0.88271576
0	minmax	1500	pca	Naive Bayes	0.755606434
0	minmax	1500	pca	SVM	0.921611118

X – Imputed, Scaled, Features Reduced , TRAIN**, MODEL EVALUATION (Best parameters)**

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg F1 score
0	minmax	kpca	875	RandomForest	0.828768
0	minmax	kpca	875	XGBoost	0.889695
0	minmax	kpca	875	CatBoost	0.886021
0	minmax	kpca	875	Naive Bayes	0.783
0	minmax	kpca	875	SVM	0.922

Imputed	Scaling	nDimensions	Dim Reduction	Model	Avg F1 score
0	minmax	1500	pca	RandomForest	0.819196863
0	minmax	1500	pca	XGBoost	0.887455895
0	minmax	1500	pca	CatBoost	0.88271576
0	minmax	1500	pca	Naive Bayes	0.755606434
0	minmax	1500	pca	SVM	0.921611118

X – Imputed, Scaled, Features Reduced , TRAIN, **Best MODEL EVALUATION**

Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg F1 score
0	minmax	kpca	875	RandomForest	0.828768
0	minmax	kpca	875	XGBoost	0.889695
0	minmax	kpca	875	CatBoost	0.886021
0	minmax	kpca	875	Naive Bayes	0.783
0	minmax	kpca	875	SVM	0.922

Imputed	Scaling	nDimensions	Dim Reduction	Model	Avg F1 score
0	minmax	1500	pca	RandomForest	0.819196863
0	minmax	1500	pca	XGBoost	0.887455895
0	minmax	1500	pca	CatBoost	0.88271576
0	minmax	1500	pca	Naive Bayes	0.755606434
0	minmax	1500	pca	SVM	0.921611118

X – Imputed, Scaled, Features Reduced , **Synthetic Minority Oversampling, MODEL Performance**

```python
# Define the SMOTETomek sampler
smt_tomek = SMOTETomek(tomek=TomekLinks(sampling_strategy='majority'), n_jobs =
512)

```

```python
smote = SMOTE(random_state=42, sampling_strategy = 'not majority', n_jobs = 512)
_, y_train_resampled = smote.fit_resample(X_train, y_train)
```

- Under sampling and the over sampling was performed for the class distribution.
- Minority samples were SMOTEd only

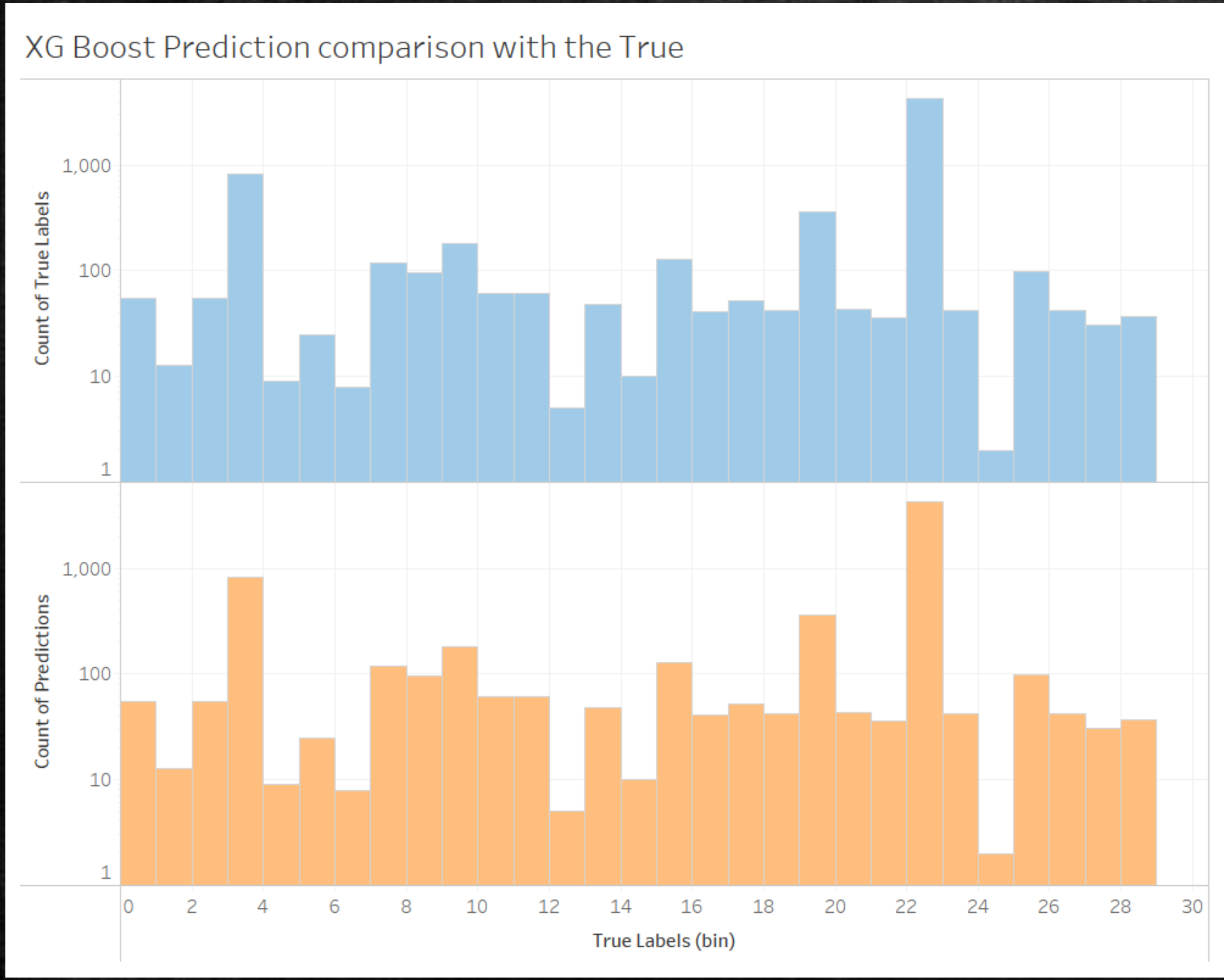
Imputed	Scaling	Dim Reduction	nDimensions	Model	Avg F1 score
0	minmax	modelseletcion - rfxgb	1000	Logistic Regression	0.892149496
0	minmax	modelseletcion - rfxgb	1000	RandomForest	0.855393666
0	minmax	modelseletcion - rfxgb	1000	XGBoost	0.978986299
0	minmax	modelseletcion – rfxgb	1000	CatBoost	0.912783372

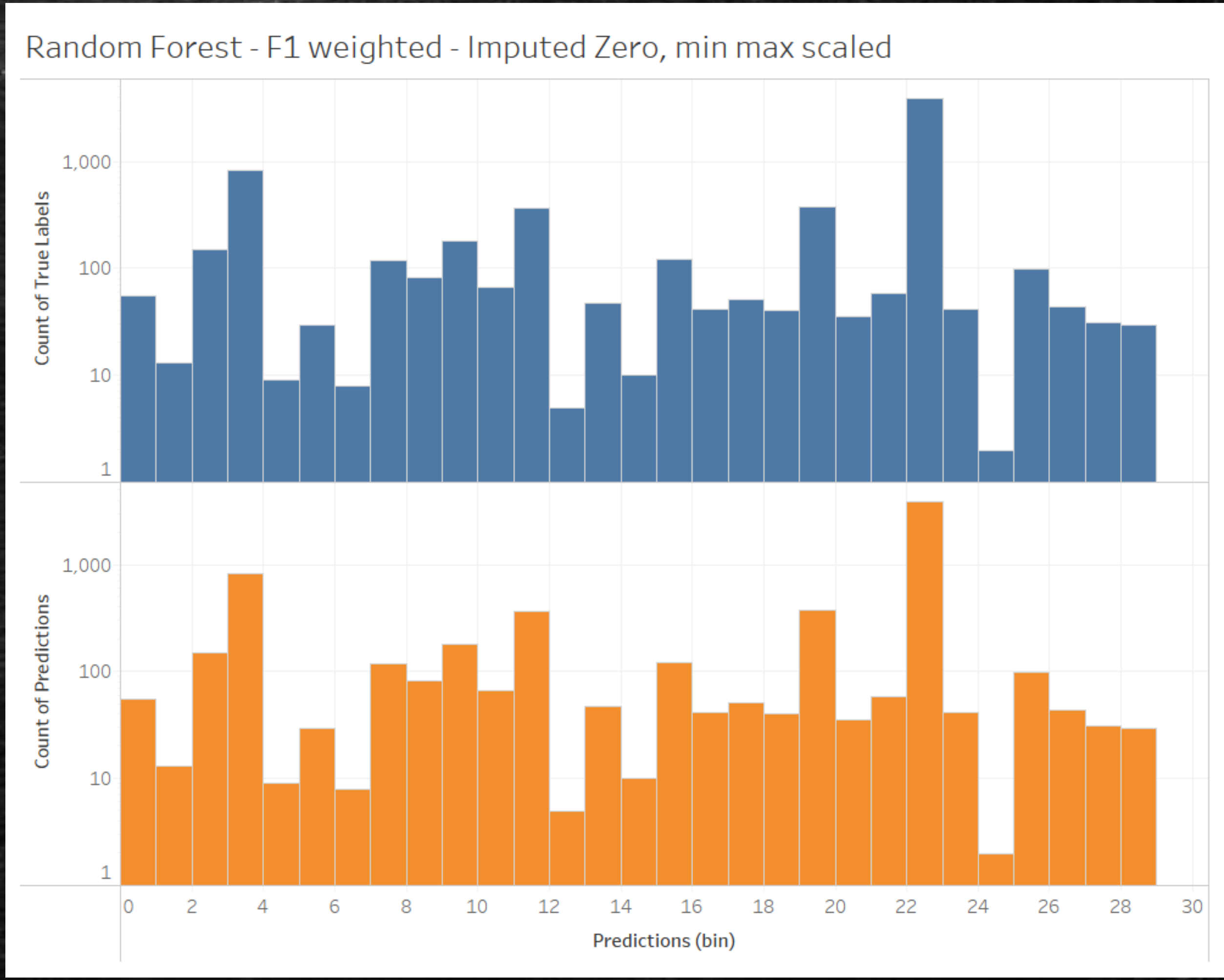
Training results for dataset that involved the synthetic generation of classes. The number of features were reduced to 1000 using Random Forest best features.

X – Imputed, Scaled, Features Reduced , TRAIN**, TESTING SET**

XGBoost : {'classifier__learning_rate': 0.1, 'classifier__max_depth': 6, 'classifier__min_child_weight': 3, 'classifier__n_estimators': 500} – scaled : MinMax, Nans – 0, PCA

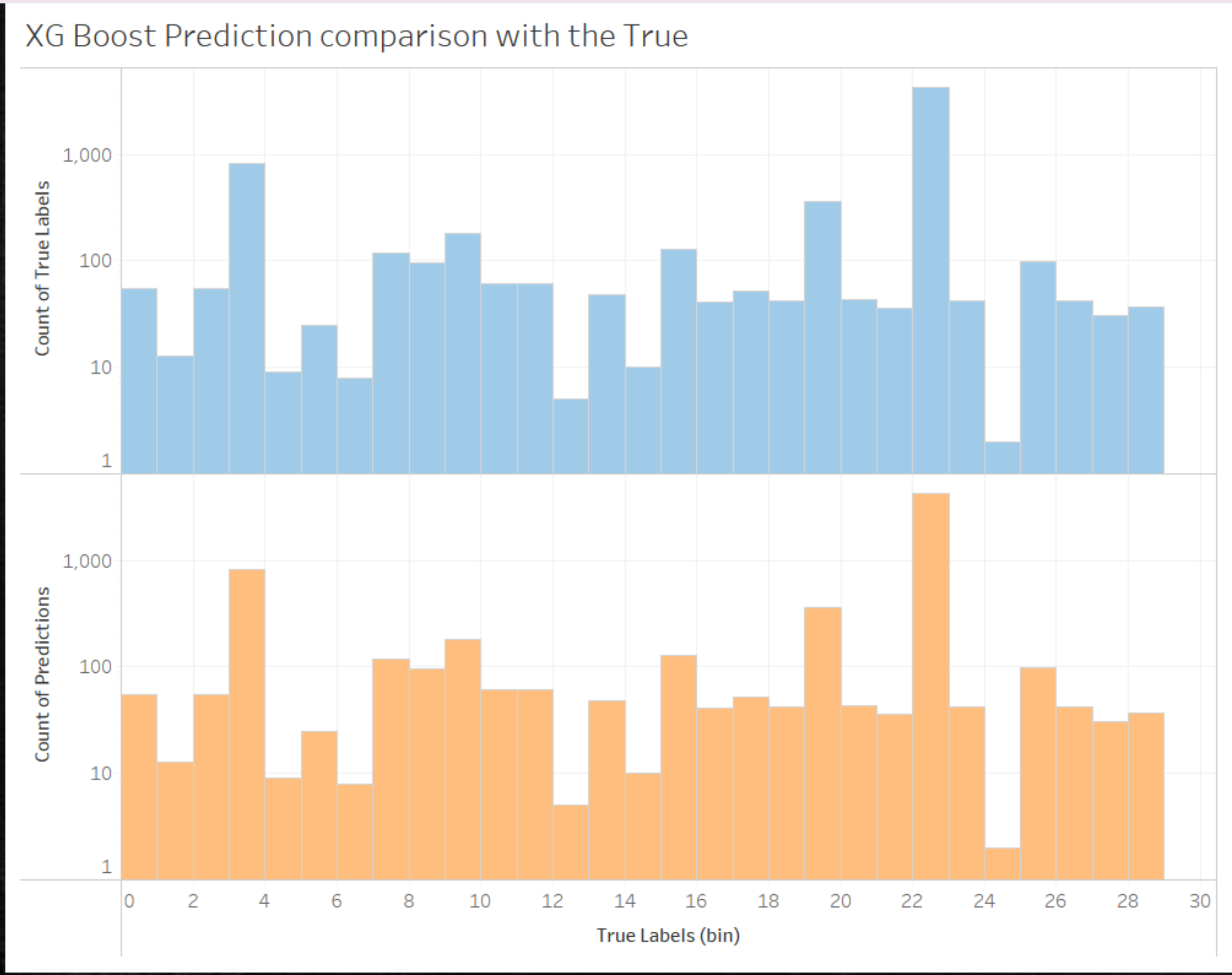
Random Forest : 'classifier__max_depth': 10, 'classifier__n_estimators': 1000





X – Imputed, Scaled, **Features Reduced PCA** , TRAIN, TESTING

XGBoost : {'classifier__learning_rate': 0.1, 'classifier__max_depth': 6, 'classifier__min_child_weight': 3, 'classifier__n_estimators': 500} – scaled : MinMax, Nans – 0
Average F1-score : 0.94



	precision	recall	f1-score	support
0	1.00	1.00	1.00	55
1	1.00	1.00	1.00	13
2	0.35	0.95	0.51	56
3	0.96	0.97	0.97	824
4	1.00	1.00	1.00	9
5	0.83	1.00	0.91	25
6	1.00	1.00	1.00	8
7	0.99	1.00	1.00	119
8	0.98	0.82	0.89	97
9	0.92	0.92	0.92	181
10	0.92	0.98	0.95	62
11	0.16	0.97	0.28	62
12	1.00	1.00	1.00	5
13	1.00	0.98	0.99	49
14	1.00	1.00	1.00	10
15	0.99	0.95	0.97	129
16	0.98	1.00	0.99	41
17	1.00	1.00	1.00	52
18	0.98	0.95	0.96	42
19	0.92	0.94	0.93	369
20	1.00	0.82	0.90	44
21	0.62	1.00	0.77	36
22	1.00	0.90	0.95	4369
23	1.00	1.00	1.00	42
24	1.00	1.00	1.00	2
25	0.99	1.00	0.99	99
26	0.89	0.91	0.90	43
27	1.00	1.00	1.00	31
28	1.00	0.81	0.90	37
accuracy			0.92	6911
macro avg	0.91	0.96	0.92	6911
eighted avg	0.97	0.92	0.94	6911

X – Imputed, Scaled, **Features Reduced model selection** , TRAIN, TESTING

The 1000 top features are extracted from the XGBoost model

	precision	recall	f1-score	support
0	1.00	1.00	1.00	55
1	1.00	1.00	1.00	13
2	0.21	0.91	0.34	56
3	0.96	0.97	0.97	824
4	1.00	1.00	1.00	9
5	0.86	1.00	0.93	25
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	119
8	0.92	0.88	0.90	97
9	0.91	0.91	0.91	181
10	0.83	0.97	0.90	62
11	0.26	0.97	0.41	62
12	1.00	1.00	1.00	5
13	0.98	0.98	0.98	49
14	1.00	1.00	1.00	10
15	0.98	0.96	0.97	129
16	0.98	1.00	0.99	41
17	1.00	1.00	1.00	52
18	0.97	0.93	0.95	42
19	0.92	0.93	0.92	369
20	1.00	0.80	0.89	44
21	0.60	1.00	0.75	36
22	1.00	0.91	0.95	4369
23	1.00	1.00	1.00	42
24	1.00	1.00	1.00	2
25	0.99	0.99	0.99	99
26	0.83	1.00	0.91	43
27	1.00	1.00	1.00	31
28	1.00	0.78	0.88	37
accuracy			0.93	6911
macro avg	0.90	0.96	0.91	6911
weighted avg	0.97	0.93	0.94	6911

The 1000 top features are extracted from the Random Forest model

	precision	recall	f1-score	support
0	1.00	1.00	1.00	55
1	1.00	1.00	1.00	13
2	0.24	0.95	0.38	56
3	0.96	0.97	0.97	824
4	1.00	1.00	1.00	9
5	0.81	1.00	0.89	25
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	119
8	0.95	0.89	0.91	97
9	0.90	0.92	0.91	181
10	0.87	0.95	0.91	62
11	0.22	0.97	0.35	62
12	1.00	1.00	1.00	5
13	1.00	0.98	0.99	49
14	0.91	1.00	0.95	10
15	0.98	0.95	0.97	129
16	0.98	1.00	0.99	41
17	1.00	1.00	1.00	52
18	1.00	0.93	0.96	42
19	0.92	0.93	0.92	369
20	1.00	0.75	0.86	44
21	0.60	1.00	0.75	36
22	1.00	0.91	0.95	4369
23	1.00	1.00	1.00	42
24	1.00	1.00	1.00	2
25	0.99	1.00	0.99	99
26	0.91	0.98	0.94	43
27	1.00	1.00	1.00	31
28	1.00	0.81	0.90	37
accuracy			0.93	6911
macro avg	0.90	0.96	0.91	6911
eighted avg	0.97	0.93	0.94	6911

X – Imputed, Scaled, Features Reduced model selection , TRAIN, TESTING

The 1000 top features are extracted from the XGBoost model

	precision	recall	f1-score	support
0	1.00	1.00	1.00	55
1	1.00	1.00	1.00	13
2	0.21	0.91	0.34	56
3	0.96	0.97	0.97	824
4	1.00	1.00	1.00	9
5	0.86	1.00	0.93	25
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	119
8	0.92	0.88	0.90	97
9	0.91	0.91	0.91	181
10	0.83	0.97	0.90	62
11	0.26	0.97	0.41	62
12	1.00	1.00	1.00	5
13	0.98	0.98	0.98	49
14	1.00	1.00	1.00	10
15	0.98	0.96	0.97	129
16	0.98	1.00	0.99	41
17	1.00	1.00	1.00	52
18	0.97	0.93	0.95	42
19	0.92	0.93	0.92	369
20	1.00	0.80	0.89	44
21	0.60	1.00	0.75	36
22	1.00	0.91	0.95	4369
23	1.00	1.00	1.00	42
24	1.00	1.00	1.00	2
25	0.99	0.99	0.99	99
26	0.83	1.00	0.91	43
27	1.00	1.00	1.00	31
28	1.00	0.78	0.88	37
accuracy			0.93	6911
macro avg	0.90	0.96	0.91	6911
veighted avg	0.97	0.93	0.94	6911

The 1000 top features are extracted from the Random Forest model

	precision	recall	f1-score	support
0	1.00	1.00	1.00	55
1	1.00	1.00	1.00	13
2	0.24	0.95	0.38	56
3	0.96	0.97	0.97	824
4	1.00	1.00	1.00	9
5	0.81	1.00	0.89	25
6	1.00	1.00	1.00	8
7	1.00	1.00	1.00	119
8	0.95	0.89	0.91	97
9	0.90	0.92	0.91	181
10	0.87	0.95	0.91	62
11	0.22	0.97	0.35	62
12	1.00	1.00	1.00	5
13	1.00	0.98	0.99	49
14	0.91	1.00	0.95	10
15	0.98	0.95	0.97	129
16	0.98	1.00	0.99	41
17	1.00	1.00	1.00	52
18	1.00	0.93	0.96	42
19	0.92	0.93	0.92	369
20	1.00	0.75	0.86	44
21	0.60	1.00	0.75	36
22	1.00	0.91	0.95	4369
23	1.00	1.00	1.00	42
24	1.00	1.00	1.00	2
25	0.99	1.00	0.99	99
26	0.91	0.98	0.94	43
27	1.00	1.00	1.00	31
28	1.00	0.81	0.90	37
accuracy			0.93	6911
macro avg	0.90	0.96	0.91	6911
ighted avg	0.97	0.93	0.94	6911

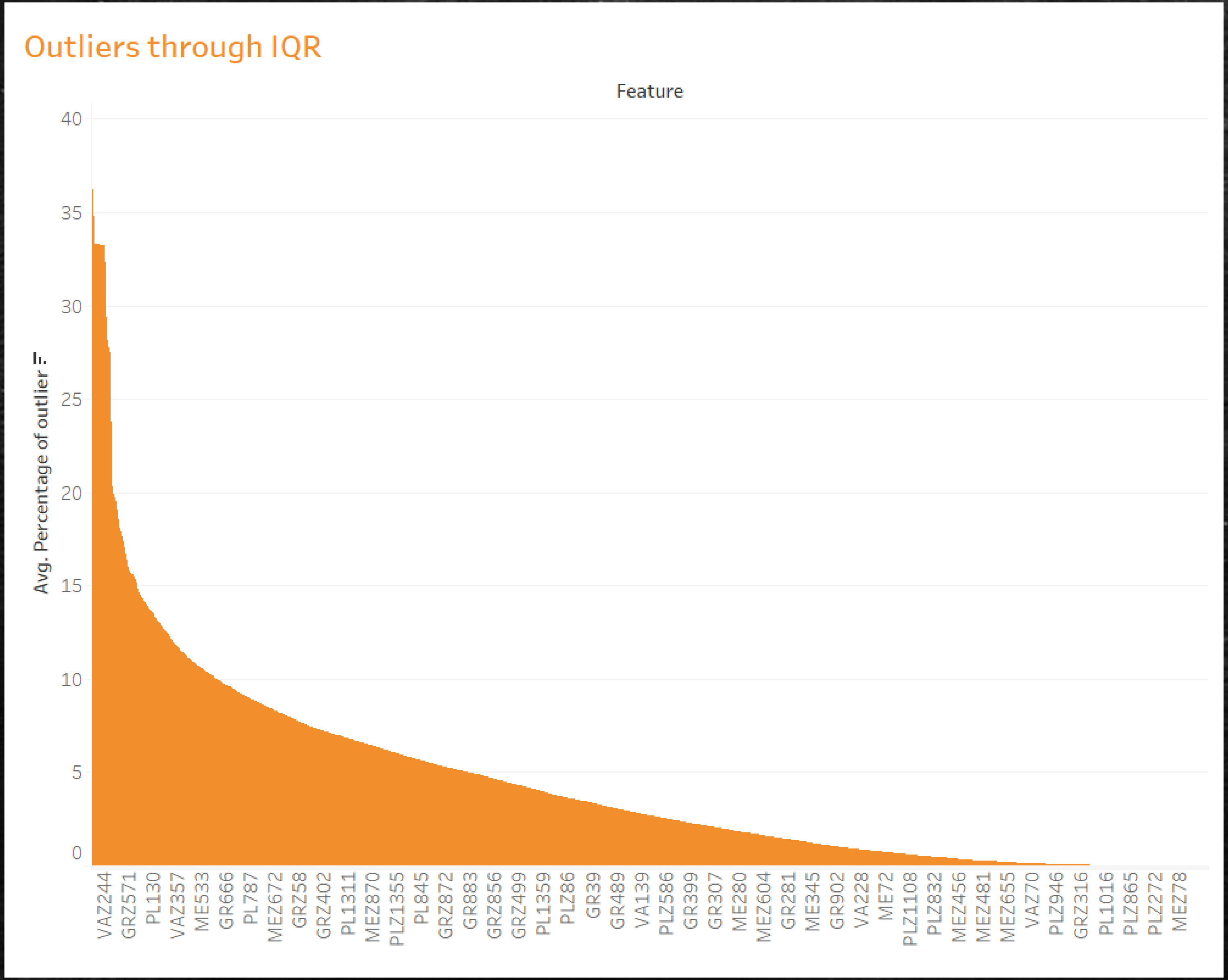
CONCLUSION

- Necessary EDA was done by handling NANs, scaling the data
- Dimension reduction was performed to train the dataset.
- Different models were trained on the cluster and XGBoost gave the best results with avg F1 score of 0.94 on test dataset.
- Model Selection was performed using best features from XGBoost and Random Forest. They gave the same average F1 score
- Models were also trained for synthetically generated minority classes

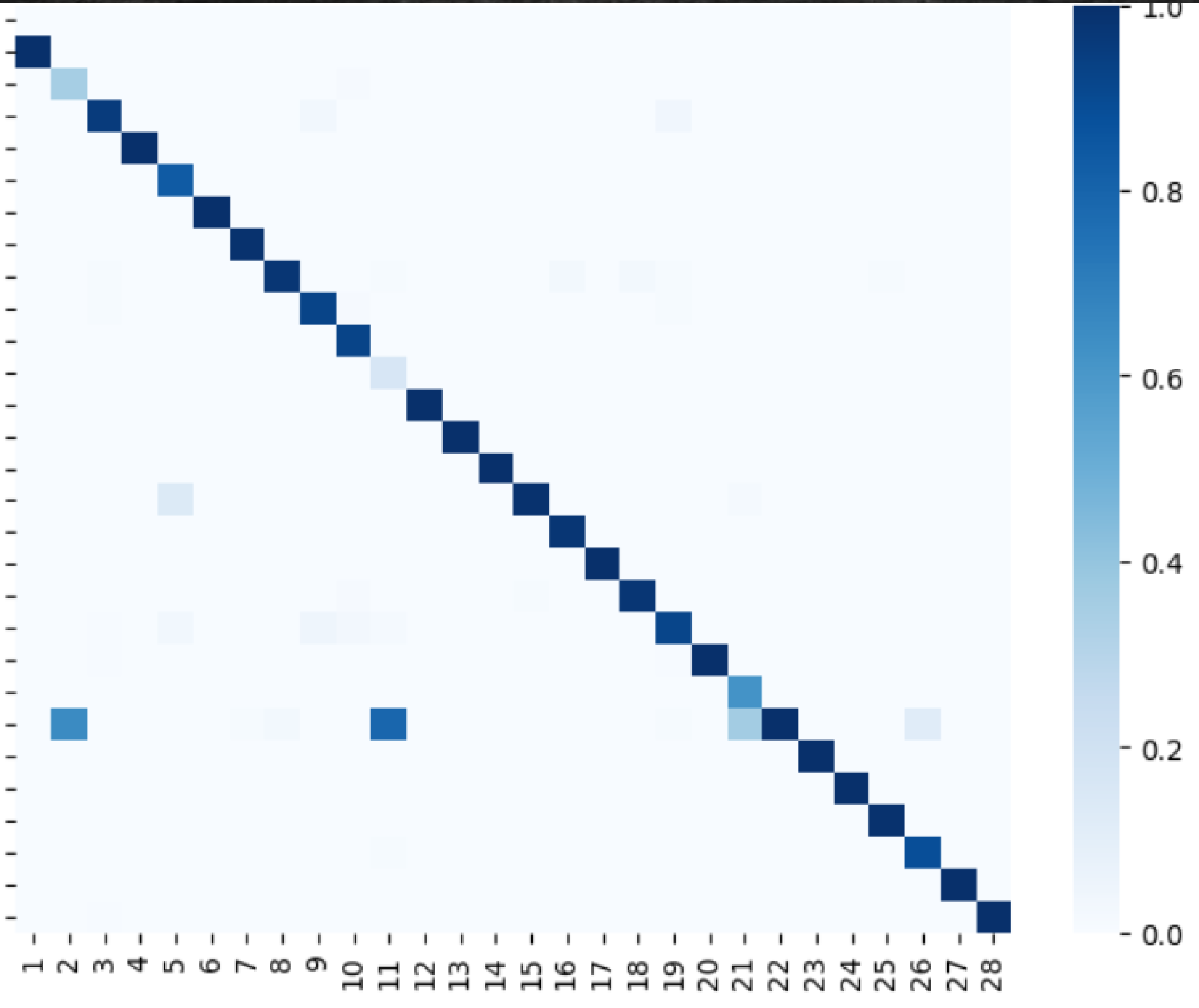
X – a lot of features, NANS, cols with unique value , Outliers

- Outlier – Values IQR

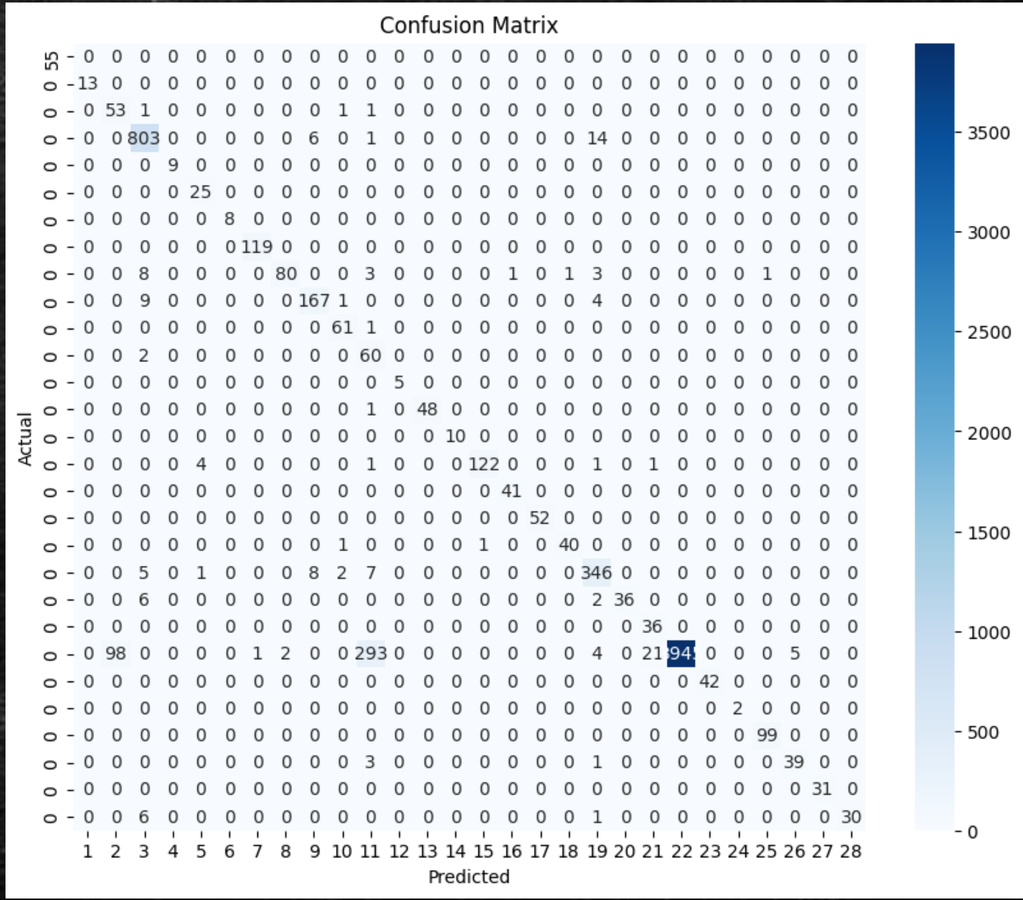
Features : 7672 -> 7539



Need to be careful while dealing with dataset



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
1.0
0.8
0.6
0.4
0.2
0.0



Confusion Matrix
Actual
Predicted
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
3500
3000
2500
2000
1500
1000
500
0