On the path to making a predictive model, we are sometimes faced with the choice to cherry pick amongst our list of features. (If the term cherrypick still gives you nightmares from your adventures in gitland then here's a fitbump 👊). Perhaps this is because of the high dimensionality of our data or just in the cyclic model hyperparameter finetuning. Regardless of the reason, learning the different ways you can employ to decide which features to use and which to not use can end up improving model performance or even reducing computational time and complexity.

EDA, Cleaning and Preprocessing

For this exercise, I employed the Financial Inclusion in Africa Competition in Zindi because of 2 reasons:

1. Readily available data
2. A starter notebook that deals with EDA and basic data cleaning

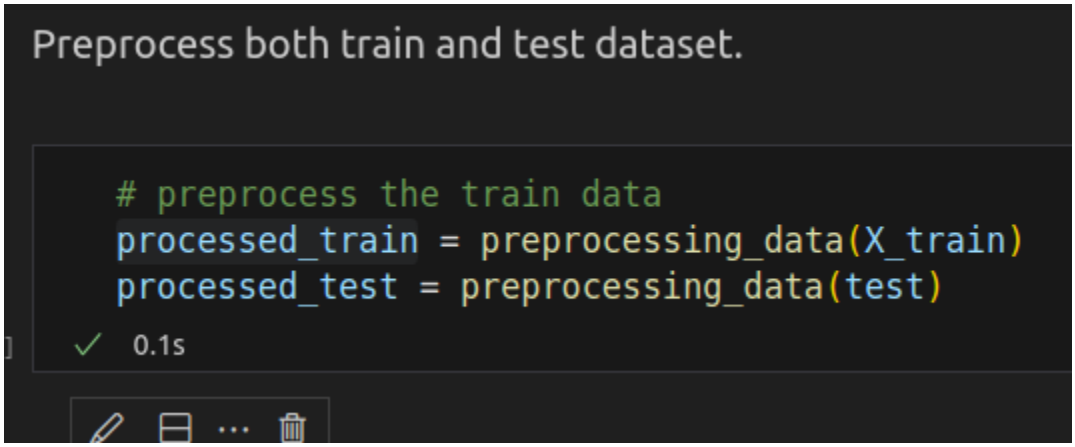
As such, the very next step from these offerings in the competition is to do feature engineering. If you want to join in for a follow-along kind of reading, feel free to download the data and starter Notebook from Zindi here (You might need to create a Zindi account though).

The starter notebook makes use of the following beautiful function that is used to transform both the test and the train datasets.

```python
# function to preprocess our data from train models
def preprocessing_data(data):

    # Convert the following numerical labels from interger to float
    float_array = data[["household_size", "age_of_respondent", "year"]].values.astype(float)

    # categorical features to be onverted to One Hot Encoding
    categ = ["relationship_with_head",
             "marital_status",
             "education_level",
             "job_type",
             "country"]

    # One Hot Encoding conversion
    data = pd.get_dummies(data, prefix_sep="_", columns=categ)

    # Label Encoder conversion
    data["location_type"] = le.fit_transform(data["location_type"])
    data["cellphone_access"] = le.fit_transform(data["cellphone_access"])
    data["gender_of_respondent"] = le.fit_transform(data["gender_of_respondent"])

    # drop uniquid column
    data = data.drop(["uniqueid"], axis=1)

    # scale our data into range of 0 and 1
    scaler = MinMaxScaler(feature_range=(0, 1))
    data = scaler.fit_transform(data)

    return data
```

Unfortunately, after the transformations, we have a long list of 37 features. Although we can use all of them it is advisable to select the best features which would help us best predict the target variable.

After the processing, we are left with two numpy array sets.



I converted the arrays to dataframes and then saved them to CSV files for easy processing in another notebook.

```
# Save to csv
processed_train = pd.DataFrame(processed_train)
processed_test = pd.DataFrame(processed_test)

processed_train.to_csv('data/preprocessed_train.csv', index = False)
processed_test.to_csv('data/preprocessed_test.csv', index = False)
```

✓ 0.9s

I could have simply used them in the starter Notebook but I wanted to have a saved version of my pre-processed data for future experimentation with feature Engineering and other techniques that would better model performance.

Feature Selection

In another notebook, I imported the required libraries as well as the datasets:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
X = pd.read_csv('data/preprocessed_train.csv')
train = pd.read_csv('data/Train.csv')
y = train['bank_account']

X.head()
```

Afterwards, I started the different ways to select features.

1. Univariate Statistics

Statistical tests can be used to select those features that have the strongest relationship with the output variable.

The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features.

Many different statistical test scan be used with this selection method. For example the ANOVA F-value method is appropriate for numerical inputs and categorical data. This can be used via the f_classif() function.

```
from sklearn.feature_selection import SelectKBest
from numpy import set_printoptions
from sklearn.feature_selection import f_classif

# feature extraction
test = SelectKBest(score_func=f_classif, k=4)
fit = test.fit(X, y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

We will use the chi method to select the 10 best features using this method in the example below.

```python
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
    Specs        Score
26     26  1398.459048
21     21  1309.442224
25     25  1287.543695
22     22  1224.908523
33     33   605.482163
17     17   386.875887
19     19   323.840989
20     20   295.349066
2       2   266.843216
5       5   190.722499
```

Alternatives to ch-squared and ANOVA F-value (all imported from sklearn.feature_selection)

1. *Mutual Information*: Measures the mutual dependence between two variables.

2. *Information Gain*: Measures the reduction in entropy achieved by splitting data on a particular feature.

3. *Correlation Coefficient*: Measures the linear relationship between two numerical variables.

4. *Distance Correlation*: Measures the dependence between two random variables.

5. *ReliefF*: Computes feature importance based on the ability to distinguish between instances of different classes.

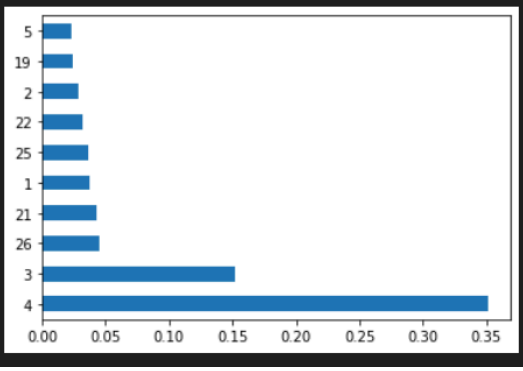
2. Feature Importance

Bagged decision trees like Random Forest and Extra Trees can be used to estimate the importance of features.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[1.05760727e-02 3.74495083e-02 2.83546148e-02 1.51631017e-01
 3.51043847e-01 2.33748542e-02 6.46679755e-03 1.67228763e-02
 2.13604723e-03 3.94363423e-03 3.84675759e-03 5.79380632e-03
 7.03031964e-03 1.77035601e-04 1.28480926e-02 1.06547351e-02
 5.78128207e-03 1.71716613e-02 9.63246421e-04 2.42933055e-02
 1.88041007e-02 4.28441332e-02 3.22379754e-02 1.23023694e-03
 8.92866561e-03 3.65443945e-02 4.50739490e-02 5.04969028e-03
 1.24935623e-02 1.67252352e-03 7.94029277e-03 9.56560149e-03
 1.56093011e-02 1.78868140e-02 6.93903571e-03 1.09375875e-02
 5.98262389e-03]
```



3. Recursive Feature Elimination

The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain.

It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

You can learn more about the RFE class in the scikit-learn documentation.

The example below uses RFE with the logistic regression algorithm to select the top 10 features. The choice of algorithm does not matter too much as long as it is skillful and consistent.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Assuming you have already defined X and y

# Create the RFE model and select 10 best features
rfe = RFE(estimator=LogisticRegression(), n_features_to_select=10)
fit = rfe.fit(X, y)

# Create DataFrame to store selected features and their rankings
selected_features = pd.DataFrame({'Feature': X.columns, 'Ranking': rfe.ranking_})

# Print the selected features
print(selected_features[selected_features['Ranking'] == 1])
```

```
    Feature  Ranking
2         2        1
4         4        1
17       17        1
19       19        1
21       21        1
22       22        1
25       25        1
26       26        1
29       29        1
35       35        1
```

4. Principal Component Analysis (PCA)

Principal Component Analysis(or PCA) uses linear algebra to transform the dataset into a compressed form.

Generally this is called a data reduction technique. A property of PCA is that you can choose the number of dimensions or principal component in the transformed result.

In the example below, we use PCA and select 3 principal components.

Learn more about the PCA class in scikit-learn by reviewing the PCA API. Dive deeper into the math behind PCA on the Principal Component Analysis Wikipedia article.

PCA Usefulness:

1. ***Dimension reduction***: When dealing with datasets containing a large number of features, PCA can help reduce the dimensionality while preserving most of the variability in the data. This can lead to simpler models, reduced computational complexity, and alleviation of the curse of dimensionality. An example is here where we have 37 features

2. ***Data exploration/visualization***: PCA can be used to visualize high-dimensional data in lower-dimensional space (e.g., 2D or 3D) for exploratory data analysis and visualization. This can help uncover patterns, clusters, and relationships between variables.

3. ***Noise reduction***: PCA identifies and removes redundant information (noise) in the data by focusing on the directions of maximum variance. This can lead to improved model performance by reducing overfitting and improving generalization

4. ***Feature Creation***: PCA can be used to create new composite features (principal components) that capture the most important information in the original features. These components may be more informative or less correlated than the original features, potentially enhancing the performance of machine learning algorithms.

5. ***Reducing computational Complexity***: In cases where the original dataset is large and computationally expensive to process, PCA can be used to reduce the size of the dataset without sacrificing much information. This can lead to faster training and inference times for machine learning models.

6. **Addressing multicollinearity in the features**: PCA can mitigate multicollinearity issues by transforming correlated features into orthogonal principal components. This can improve the stability and interpretability of regression models.

```python
from sklearn.decomposition import PCA

# Create the PCA model and specify the number of components
pca = PCA(n_components=10)
X_pca = pca.fit_transform(X)

# Create DataFrame to store principal components
pca_df = pd.DataFrame(data=X_pca, columns=[f'PC{i}' for i in range(1, 11)])

# Print the explained variance ratio of each principal component
print(pca.explained_variance_ratio_)

# Print the first few rows of the transformed data
print(pca_df.head())
```

```
[0.165 0.114 0.111 0.086 0.075 0.059 0.055 0.042 0.039 0.035]
        PC1       PC2       PC3       PC4       PC5       PC6       PC7  \
0 -0.072415  0.472525  1.597179  0.110169 -0.096852 -0.666902  0.002934
1  0.121378 -0.747126  0.669584  0.807667  1.212094  0.999779 -0.308788
2  1.147935 -0.259399  0.577820  0.373439 -0.660275 -0.420732 -0.169564
3 -0.227015 -0.533407  0.930931 -0.619797 -0.145277  0.513711 -0.232388
4  0.900289 -0.002575  0.200828  0.175920 -1.077896  1.097199  0.381326

        PC8       PC9      PC10
0  0.024823  0.466306 -0.190280
1  0.213120  0.109383 -0.091919
2  0.018762 -0.195644 -0.369087
3 -0.304223  0.203956 -0.155300
4  0.606125 -0.591878  0.233130
```

5. Correlation Matrix With HeatMap

A correlation matrix is a square matrix that shows the correlation coefficients between pairs of variables in a dataset. Each cell in the matrix represents the correlation coefficient between two variables.
The correlation coefficient ranges from -1 to 1, where:

- 1 indicates a perfect positive correlation,
- 0 indicates no correlation, and
- -1 indicates a perfect negative correlation.

A heatmap is a graphical representation of the correlation matrix, where each cell's color indicates the strength and direction of the correlation between two variables.
Darker colors (e.g., red) represent stronger positive correlations, while lighter colors (e.g., blue) represent stronger negative correlations.

The diagonal of the heatmap typically shows correlation values of 1, as each variable is perfectly correlated with itself.
By visualizing the correlation matrix as a heatmap, you can quickly identify patterns of correlation between features.

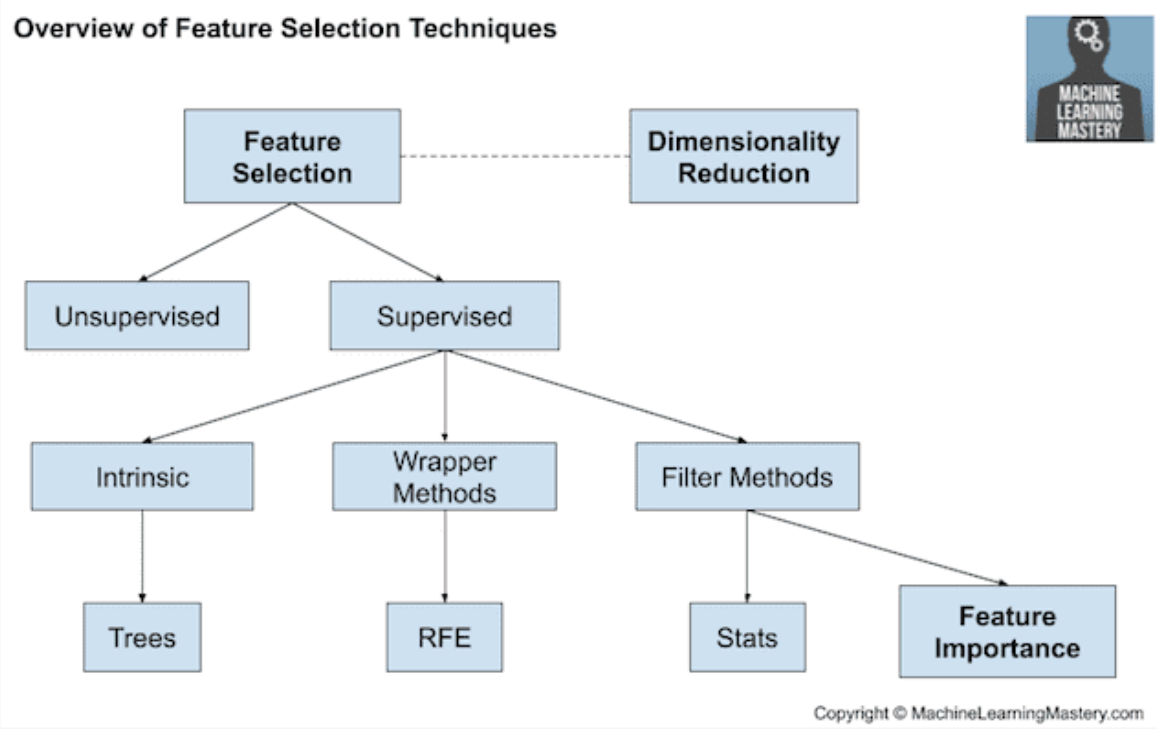
In the heatmap, you can look for clusters of high correlation (e.g., dark squares) to identify groups of features that are highly correlated with each other.
Once identified, you can decide whether to keep, remove, or transform these features based on their importance to the model and their contribution to multicollinearity.

5. Correlation Matrix with Heatmap

```
#get correlations of each features in dataset
corrmat = X.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(37,37))
#plot heat map
g=sns.heatmap(X[top_corr_features].corr(),annot=True,cmap="coolwarm")
```

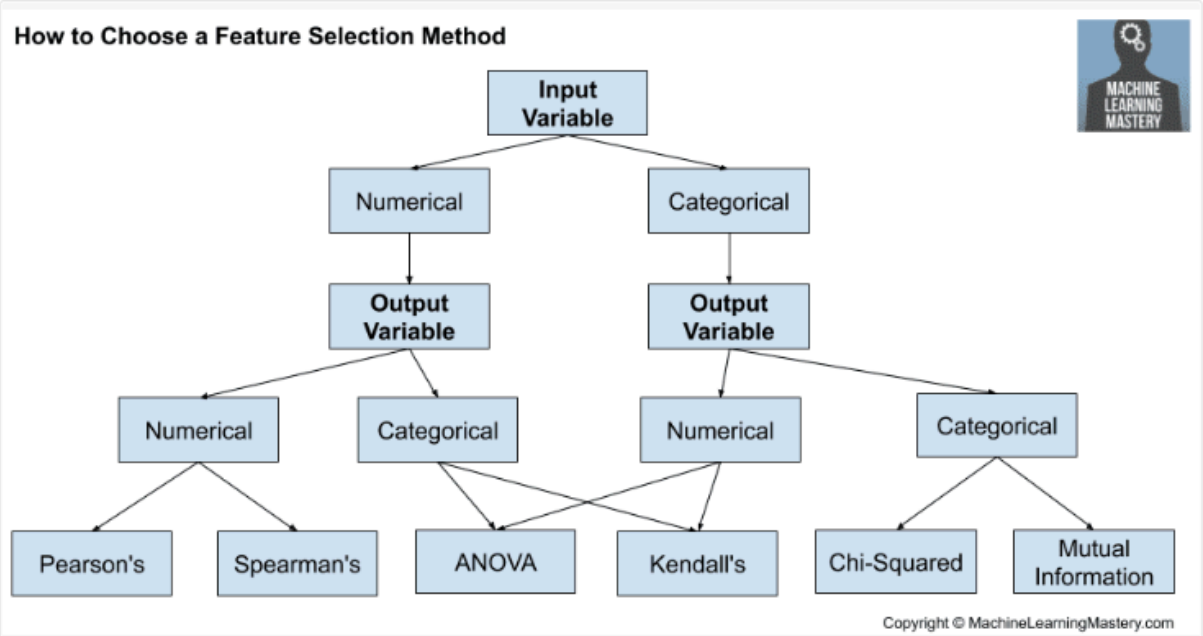
Conclusion

In Conclusion, feature selection is an important part of the machine learning process which can have a myriad of benefits. There are a multitude of ways to go about it but depending on your particular use case, be it supervised or unsupervised, some may be better than others. For instance, in supervised learning as was our case above, we compared the features against their usefulness in determining a target variable. However, in unsupervised learning, there is no target variable. As such, most feature selection algorithms that are useful here compare the variables to each other, helping cull out the ones with high multicollinearity.



Overview of Feature Selection Techniques

Another important matrix in determining the Feature selection algorithm to use is the data types of your features. In general, different data types need different selection algorithms. Please see below:



How to Choose Feature Selection Methods For Machine Learning

Have a happy time exploring the other algorithms and may the force be with you!