Analysis

February 8, 2024

```
[ ]: import pandas as pd
     import plotly.graph_objects as go
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import LabelEncoder, StandardScaler
     from sklearn.pipeline import Pipeline
     from sklearn.feature_selection import mutual_info_classif
     from sklearn.feature_selection import f_classif
     from feature_engine.selection import DropCorrelatedFeatures,␣
      ↪SmartCorrelatedSelection
     from sklearn.feature_selection import SelectKBest
     from sklearn.model_selection import train_test_split
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import classification_report
     from venny4py.venny4py import *
```

0.1 Heatmap of Numerical Features

```
[ ]: original_df = pd.read_csv('Original_Data.csv')
     numerical_columns = original_df.select_dtypes(include=['int64', 'float64']).
      ↪columns
     fig = go.Figure(data=go.Heatmap(
         z=original_df.values,
         x=numerical_columns,
         y=numerical_columns,
     ))

     fig.update_layout(
         title='<b>Heatmap'
     )

     fig.show()
```

0.2 Enconding Categorical Data

```
[ ]: categorical_columns = ['type_of_meal_plan', 'room_type_reserved',␣
      ↪'market_segment_type', 'repeated_guest', 'booking_status']
     labelencoder = LabelEncoder()
     original_df[categorical_columns] = original_df[categorical_columns].
      ↪apply(labelencoder.fit_transform)

     original_df.head()
```

```
[ ]:   Booking_ID  no_of_adults  no_of_children  no_of_weekend_nights  \
     0   INN00001             2               0                     1
     1   INN00002             2               0                     2
     2   INN00003             1               0                     2
     3   INN00004             2               0                     0
     4   INN00005             2               0                     1

        no_of_week_nights  type_of_meal_plan  required_car_parking_space  \
     0                  2                  0                           0
     1                  3                  3                           0
     2                  1                  0                           0
     3                  2                  0                           0
     4                  1                  3                           0

        room_type_reserved  lead_time  arrival_year  arrival_month  arrival_date  \
     0                   0        224          2017             10             2
     1                   0          5          2018             11             6
     2                   0          1          2018              2            28
     3                   0        211          2018              5            20
     4                   0         48          2018              4            11

        market_segment_type  repeated_guest  no_of_previous_cancellations  \
     0                    3               0                             0
     1                    4               0                             0
     2                    4               0                             0
     3                    4               0                             0
     4                    4               0                             0

        no_of_previous_bookings_not_canceled  avg_price_per_room  \
     0                                     0               65.00
     1                                     0              106.68
     2                                     0               60.00
     3                                     0              100.00
     4                                     0               94.50

        no_of_special_requests  booking_status
     0                       0               1
```

```
1                          1                 1
2                          0                 0
3                          0                 0
4                          0                 0
```

0.3 Normalizing

[ ]:
```
columns_to_normalize = ['lead_time', 'arrival_year', 'arrival_month',␣
↪'arrival_date', 'avg_price_per_room', 'no_of_special_requests']
scaler = StandardScaler()
fitting = scaler.fit(original_df[columns_to_normalize])
original_df[columns_to_normalize] = fitting.
↪transform(original_df[columns_to_normalize])
original_df.head()
```

[ ]:
```
  Booking_ID  no_of_adults  no_of_children  no_of_weekend_nights  \
0   INN00001             2               0                     1
1   INN00002             2               0                     2
2   INN00003             1               0                     2
3   INN00004             2               0                     0
4   INN00005             2               0                     1

   no_of_week_nights  type_of_meal_plan  required_car_parking_space  \
0                  2                  0                           0
1                  3                  3                           0
2                  1                  0                           0
3                  2                  0                           0
4                  1                  3                           0

   room_type_reserved  lead_time  arrival_year  arrival_month  arrival_date  \
0                   0   1.614896     -2.137469       0.839242     -1.555662
1                   0  -0.933701      0.467843       1.164990     -1.098013
2                   0  -0.980250      0.467843      -1.766747      1.419055
3                   0   1.463610      0.467843      -0.789501      0.503757
4                   0  -0.433291      0.467843      -1.115250     -0.525952

   market_segment_type  repeated_guest  no_of_previous_cancellations  \
0                    3               0                             0
1                    4               0                             0
2                    4               0                             0
3                    4               0                             0
4                    4               0                             0

   no_of_previous_bookings_not_canceled  avg_price_per_room  \
0                                     0           -1.095033
1                                     0            0.092806
2                                     0           -1.237528
```

```
3                                    0          -0.097567
4                                    0          -0.254312

   no_of_special_requests  booking_status
0                -0.78814               1
1                 0.48376               1
2                -0.78814               0
3                -0.78814               0
4                -0.78814               0
```

0.4 Classification by Different Algorithms

[ ]:
```
original_df.drop(['Booking_ID'],axis=1, inplace=True)
```

[ ]:
```
column_names = original_df.columns.tolist()
column_names = column_names[:-1]
print(column_names)
```

```
['no_of_adults', 'no_of_children', 'no_of_weekend_nights', 'no_of_week_nights',
'type_of_meal_plan', 'required_car_parking_space', 'room_type_reserved',
'lead_time', 'arrival_year', 'arrival_month', 'arrival_date',
'market_segment_type', 'repeated_guest', 'no_of_previous_cancellations',
'no_of_previous_bookings_not_canceled', 'avg_price_per_room',
'no_of_special_requests']
```

[ ]:
```
def plot_conf_matrix (conf_matrix):

    plt.figure(figsize=(15,10))
    sns.heatmap(conf_matrix, annot=True,  fmt="d")
    plt.title('Confusion Matrix')

    plt.show()
```

Splitting Data

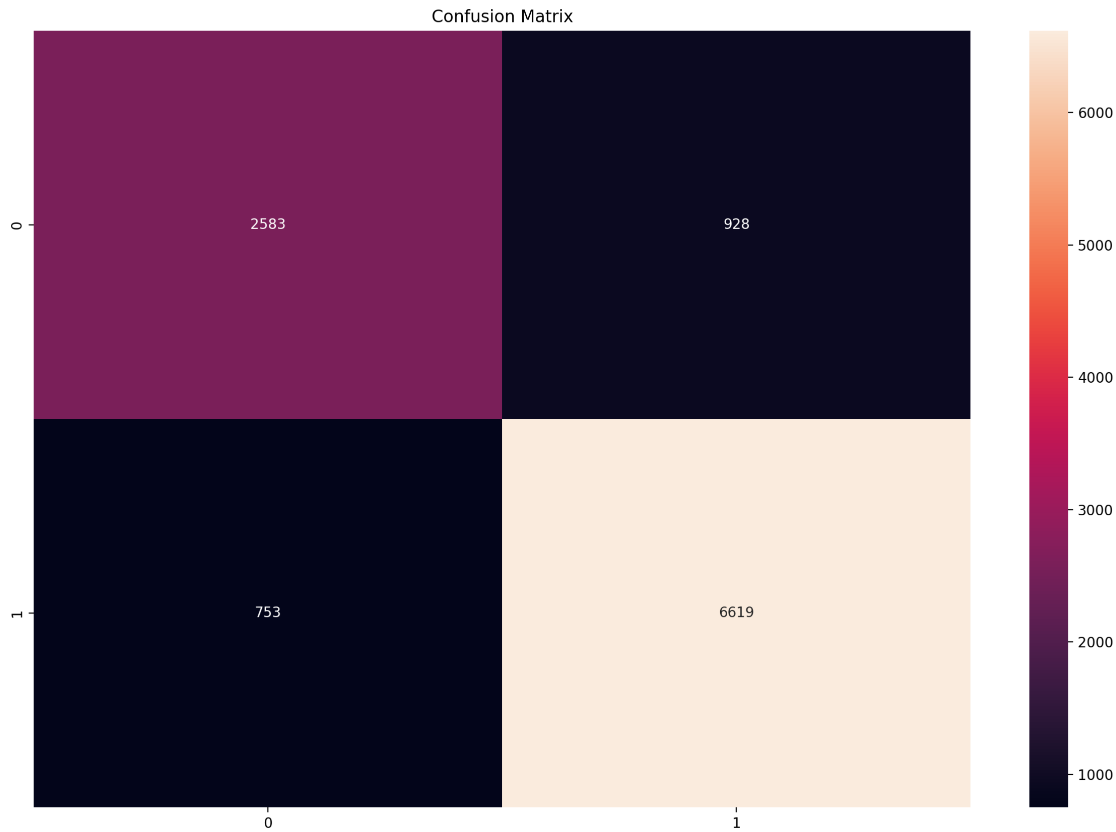
[ ]:
```
X = original_df.drop(['booking_status'], axis=1).values
y = original_df['booking_status'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

[ ]:
```
KNN = KNeighborsClassifier()
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)

KNN_score = KNN.score(X_train, y_train)
KNN_test = KNN.score(X_test, y_test)

conf_matrix = confusion_matrix(y_test, y_pred)
```

```python
print('Training Score', KNN_score)
print('Testing Score', KNN_test)
```

```
Training Score 0.8925645872715816
Testing Score 0.8455389139024166
```

[ ]:
```python
plot_conf_matrix(conf_matrix)
```



[ ]:
```python
print(classification_report(y_test,y_pred))
```

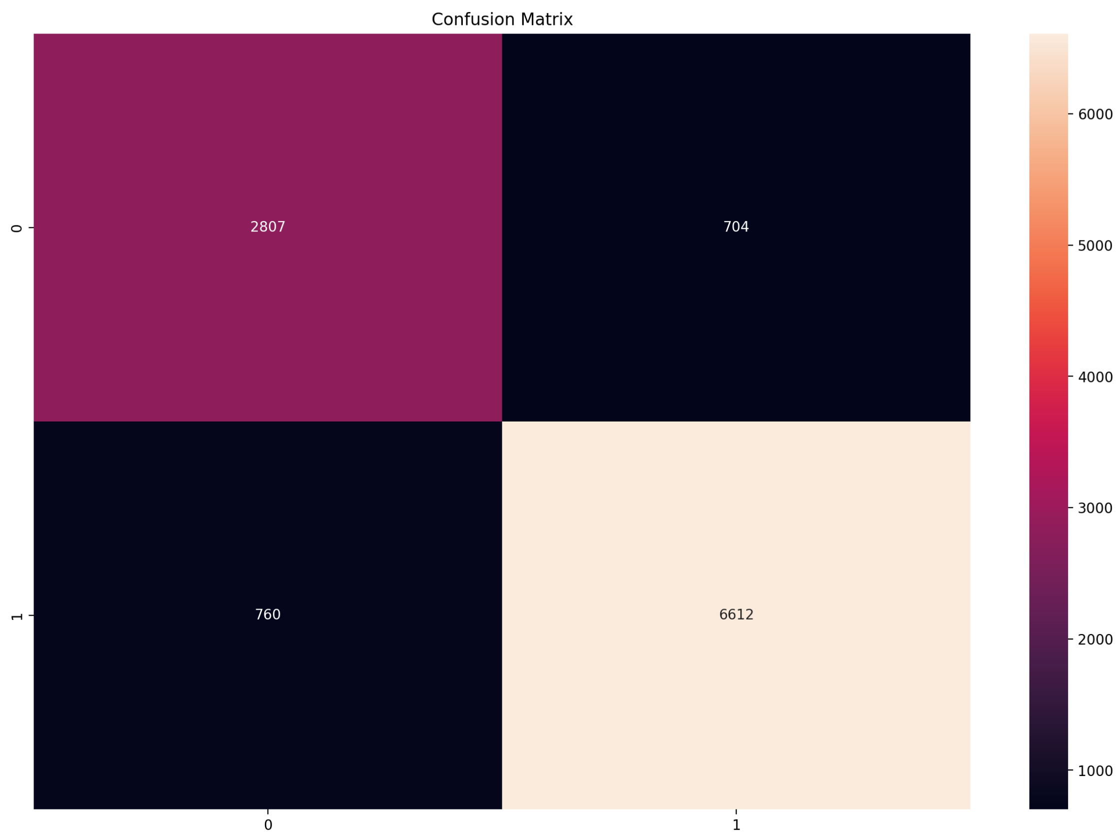
```
              precision    recall  f1-score   support

           0       0.77      0.74      0.75      3511
           1       0.88      0.90      0.89      7372

    accuracy                           0.85     10883
   macro avg       0.83      0.82      0.82     10883
weighted avg       0.84      0.85      0.84     10883
```

0.4.1 Decision Tree

[ ]:
```
DecisionTree = DecisionTreeClassifier(random_state=1)
DecisionTree.fit(X_train, y_train)
y_pred = DecisionTree.predict(X_test)

DecisionTree_score = DecisionTree.score(X_train, y_train)
DecisionTree_test = DecisionTree.score(X_test, y_test)

conf_matrix = confusion_matrix(y_test, y_pred)

print('Training Score', DecisionTree_score)
print('Testing Score', DecisionTree_test)
```

```
Training Score 0.993935097668557
Testing Score 0.8654782688596894
```

[ ]:
```
plot_conf_matrix(conf_matrix)
```



[ ]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support
```

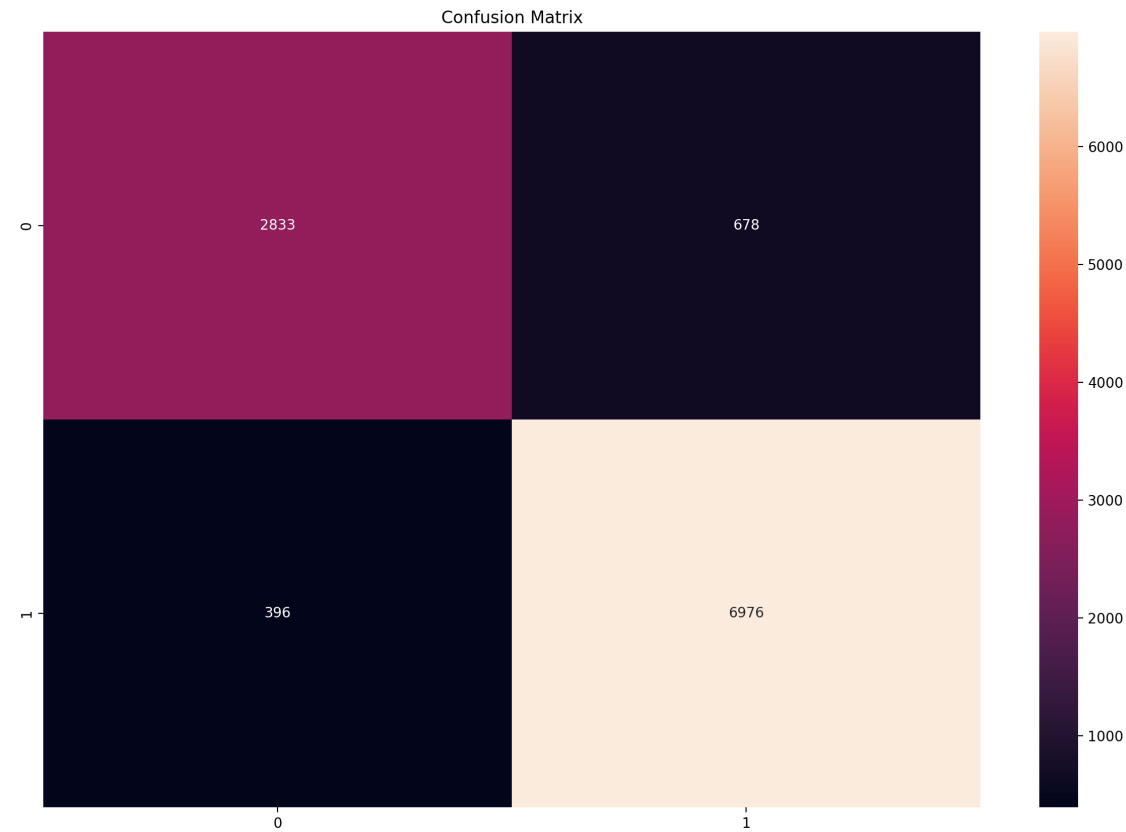
```
           0       0.79      0.80      0.79      3511
           1       0.90      0.90      0.90      7372

    accuracy                           0.87     10883
   macro avg       0.85      0.85      0.85     10883
weighted avg       0.87      0.87      0.87     10883
```

0.4.2 Random Forest

[ ]:
```python
RandomForest = RandomForestClassifier(n_estimators = 100)

RandomForest.fit(X_train, y_train)
RandomForest_score = RandomForest.score(X_train, y_train)
RandomForest_test = RandomForest.score(X_test, y_test)

y_pred = RandomForest.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)

print('Training Score',RandomForest_score)
print('Testing Score',RandomForest_test)
```

```
Training Score 0.9938957151858853
Testing Score 0.9013139759257558
```

[ ]:
```python
plot_conf_matrix(conf_matrix)
```




```
[ ]: print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.81      0.84      3511
           1       0.91      0.95      0.93      7372

    accuracy                           0.90     10883
   macro avg       0.89      0.88      0.88     10883
weighted avg       0.90      0.90      0.90     10883
```

0.5 Classification Using Feature Selection

```
[ ]: def plot_conf_matrixes(conf1, title1, conf2, title2, conf3, title3):
         fig, axes = plt.subplots(1, 3, figsize=(18, 6))

         sns.heatmap(conf1, ax=axes[0], annot=True, fmt="d")
         sns.heatmap(conf2, ax=axes[1], annot=True, fmt="d")
         sns.heatmap(conf3, ax=axes[2], annot=True, fmt="d")
```

```
    axes[0].set_title(title1)
    axes[1].set_title(title2)
    axes[2].set_title(title3)

    plt.suptitle('Confusion Matrices', fontsize=16)
    plt.tight_layout()
    plt.show()
```

[ ]:
```
dropC = DropCorrelatedFeatures(
    threshold=0.8,
    method='pearson'
)
```

Mutual Information, Anova and Smart Correlated Groups

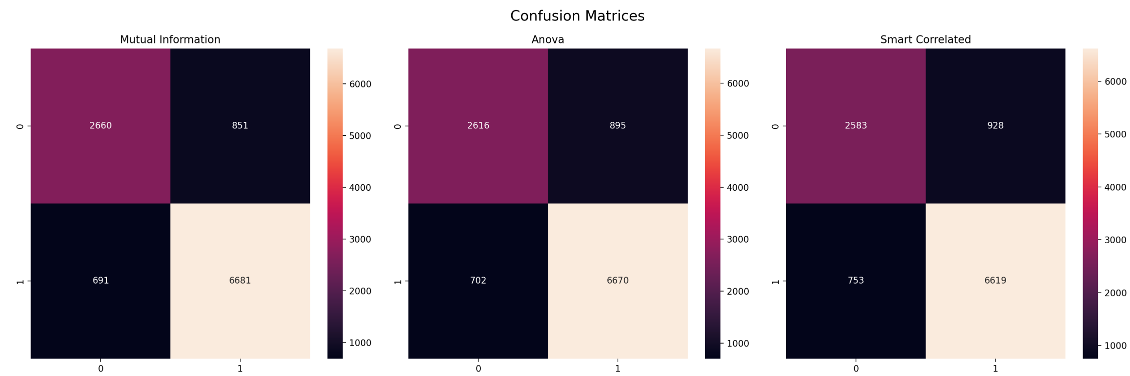
[ ]:
```
MutualInformation = SelectKBest(mutual_info_classif, k=10)
Anova = SelectKBest(f_classif, k=10)
SmartCorr = SmartCorrelatedSelection(
    method='pearson',
    threshold=0.8,
    selection_method='variance',
    estimator=None
)
```

0.5.1 K-Nearest Neighbours

[ ]:
```
MI_KNN = Pipeline(
    [('Mutual Information', MutualInformation),
     ('K-Nearest Neighbours', KNN)]
)

MI_KNN.fit(X_train, y_train)
MI_KNN_pred = MI_KNN.predict(X_test)

MI_KNN_conf_matrix = confusion_matrix(y_test, MI_KNN_pred)

AN_KNN = Pipeline(
    [('Anova', Anova),
     ('K-Nearest Neighbours', KNN)]
)

AN_KNN.fit(X_train, y_train)
AN_KNN_pred = AN_KNN.predict(X_test)

AN_KNN_conf_matrix = confusion_matrix(y_test, AN_KNN_pred)

SC_KNN = Pipeline(
    [('Smart Correlated Groups', SmartCorr),
```

```
    ('K-Nearest Neighbours', KNN)]
)

SC_KNN.fit(X_train, y_train)
SC_KNN_pred = SC_KNN.predict(X_test)

SC_KNN_conf_matrix = confusion_matrix(y_test, SC_KNN_pred)
```

[ ]:
```
plot_conf_matrixes(MI_KNN_conf_matrix, 'Mutual Information',␣
 ↪AN_KNN_conf_matrix, 'Anova', SC_KNN_conf_matrix, 'Smart Correlated')
```

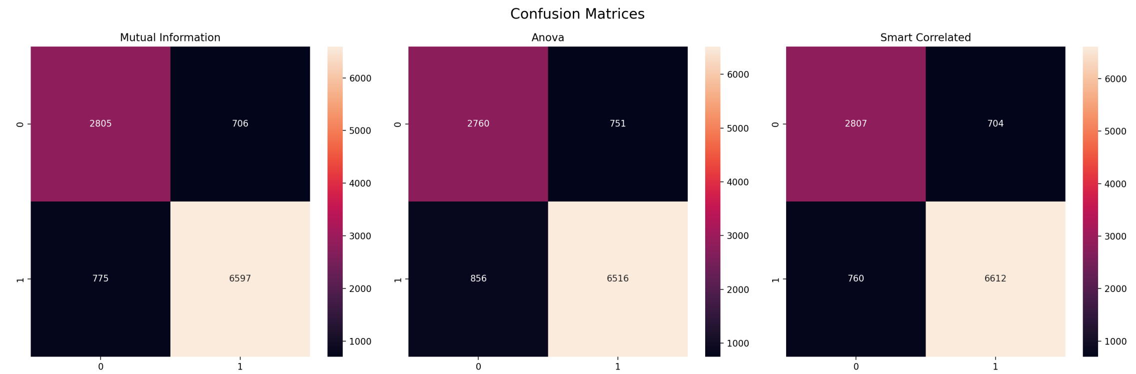


0.5.2 Decision Tree

[ ]:
```
MI_DT = Pipeline(
    [('Mutual Information', MutualInformation),
     ('Decision Tree', DecisionTree)]
)

MI_DT.fit(X_train, y_train)
MI_DT_pred = MI_DT.predict(X_test)

MI_DT_conf_matrix = confusion_matrix(y_test, MI_DT_pred)

AN_DT = Pipeline(
    [('Anova', Anova),
     ('Decision Tree', DecisionTree)]
)

AN_DT.fit(X_train, y_train)
AN_DT_pred = AN_DT.predict(X_test)

AN_DT_conf_matrix = confusion_matrix(y_test, AN_DT_pred)
```

```
SC_DT = Pipeline(
    [('Smart Correlated Groups', SmartCorr),
     ('Decision Tree', DecisionTree)]
)

SC_DT.fit(X_train, y_train)
SC_DT_pred = SC_DT.predict(X_test)

SC_DT_conf_matrix = confusion_matrix(y_test, SC_DT_pred)
```

[ ]:
```
plot_conf_matrixes(MI_DT_conf_matrix, 'Mutual Information', AN_DT_conf_matrix,␣
 ↪'Anova', SC_DT_conf_matrix, 'Smart Correlated')
```



0.5.3 Random Forest

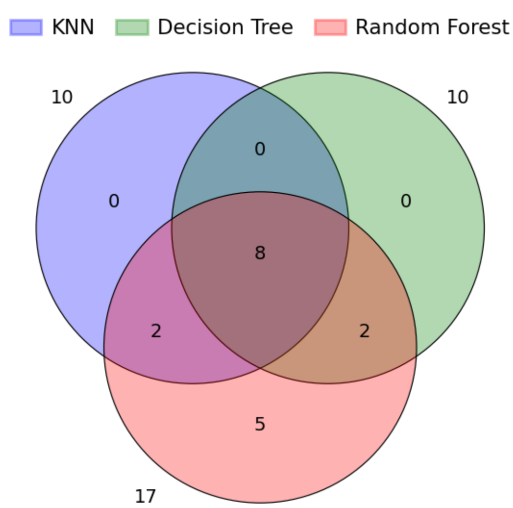
[ ]:
```
MI_RF = Pipeline(
    [('Mutual Information', MutualInformation),
     ('Random Forest', RandomForest)]
)

MI_RF.fit(X_train, y_train)
MI_RF_pred = MI_RF.predict(X_test)

MI_RF_conf_matrix = confusion_matrix(y_test, MI_RF_pred)

AN_RF = Pipeline(
    [('Anova', Anova),
     ('Random Forest', RandomForest)]
)

AN_RF.fit(X_train, y_train)
AN_RF_pred = AN_RF.predict(X_test)
```

```
AN_RF_conf_matrix = confusion_matrix(y_test, AN_RF_pred)

SC_RF = Pipeline(
    [('Smart Correlated Groups', SmartCorr),
     ('Random Forest', RandomForest)]
)

SC_RF.fit(X_train, y_train)
SC_RF_pred = SC_RF.predict(X_test)

SC_RF_conf_matrix = confusion_matrix(y_test, SC_RF_pred)
```

0.5.4 Common Features

[ ]:
```
# MI_KNN = pd.DataFrame(MI_KNN, columns = column_names)

features_MI_KNN = MI_KNN[:-1].get_feature_names_out(input_features=column_names)
features_AN_KNN = AN_KNN[:-1].get_feature_names_out(input_features=column_names)
features_SC_KNN = SC_KNN[:-1].get_feature_names_out(input_features=column_names)
```

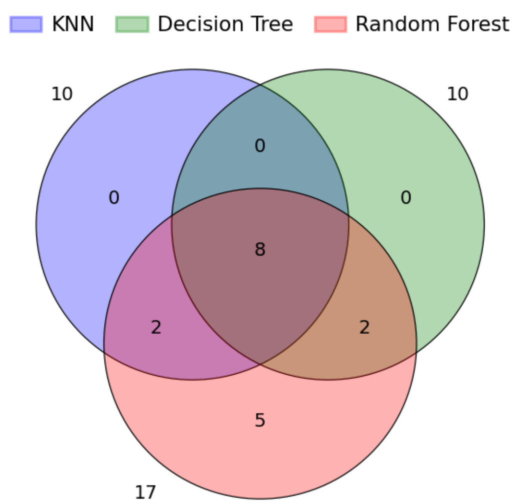
[ ]:
```
sets = {
    'KNN': set(features_MI_KNN),
    'Decision Tree': set(features_AN_KNN),
    'Random Forest': set(features_SC_KNN)
}

venny4py(sets=sets)
```



```
[ ]: set(features_MI_KNN).intersection(features_AN_KNN, features_SC_KNN)
```

```
[ ]: {'arrival_year',
      'avg_price_per_room',
      'lead_time',
      'market_segment_type',
      'no_of_adults',
      'no_of_special_requests',
      'no_of_weekend_nights',
      'required_car_parking_space'}
```

```
[ ]: features_MI_DT = MI_DT[:-1].get_feature_names_out(input_features=column_names)
     features_AN_DT = AN_DT[:-1].get_feature_names_out(input_features=column_names)
     features_SC_DT = SC_DT[:-1].get_feature_names_out(input_features=column_names)

     sets = {
         'KNN': set(features_MI_DT),
         'Decision Tree': set(features_AN_DT),
         'Random Forest': set(features_SC_DT)
     }

     venny4py(sets=sets)
```

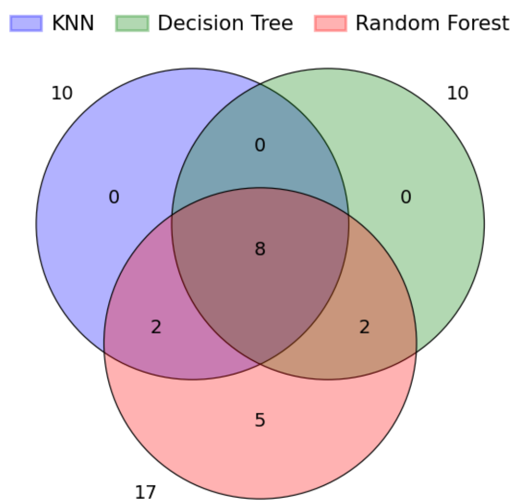



```
[ ]: set(features_MI_DT).intersection(features_AN_DT, features_SC_DT)
```

```
[ ]: {'arrival_year',
      'avg_price_per_room',
      'lead_time',
      'market_segment_type',
```

```
 'no_of_adults',
 'no_of_special_requests',
 'no_of_weekend_nights',
 'required_car_parking_space'}
```

[ ]:
```
features_MI_RF = MI_RF[:-1].get_feature_names_out(input_features=column_names)
features_AN_RF = AN_RF[:-1].get_feature_names_out(input_features=column_names)
features_SC_RF = SC_RF[:-1].get_feature_names_out(input_features=column_names)

sets = {
    'KNN': set(features_MI_RF),
    'Decision Tree': set(features_AN_RF),
    'Random Forest': set(features_SC_DT)
}

venny4py(sets=sets)
```



[ ]:
```
set(features_MI_RF).intersection(features_AN_RF, features_SC_RF)
```

[ ]:
```
{'arrival_year',
 'avg_price_per_room',
 'lead_time',
 'market_segment_type',
 'no_of_adults',
 'no_of_special_requests',
 'no_of_weekend_nights',
 'required_car_parking_space'}
```