Business Understanding

The local police department has required the development of a binary prediction automated system that could determine the sex of individuals from the footprints that have been left at crime scenes, for the automated model, the local police force requires, needs to be able to make reasonably high predictions accuracy, within a limited of time, that will be used on a new device and to help the investigation team to narrow down suspects on the initial stages.

To achieve these targets, we have been given a set of data that contains 18 landmarks in the form of X and y coordinates, the report below will provide a detailed examination of the data and its findings, the decision-making of each process, and recommendations for potential improvements and future work.

step 0: Prepareing

At step zero, we will first be setting up the necessary components for the work to work seamlessly and error free.

local RUN setup

```
import zipfile
```

```
pip install kaggle pandas joblib numpy matplotlib seaborn xgboost scipy statsmodels
```

```
!pip install kaggle xgboost joblib statsmodels

pip install --upgrade kaggle pandas joblib numpy matplotlib seaborn xgboost scipy st
```

```
pip install --upgrade kaggle pandas joblib numpy matplotlib seaborn xgboost scipy st
```

Import List

```
import kaggle
import pandas as pd
from joblib import dump, load
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier, p
from scipy import stats
from scipy.stats import spearmanr
from scipy.stats.mstats import winsor
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import classific
from sklearn.impute import KNNImputer
from sklearn.experimental import enab
from sklearn.impute import IterativeI
from sklearn.preprocessing import Rob
from sklearn.naive_bayes import Gauss
from sklearn.ensemble import Gradient
from sklearn.ensemble import Isolatic
from sklearn.neighbors import LocalOu
from sklearn.model_selection import t
from sklearn.linear_model import Logi
from sklearn import linear_model
from sklearn.neighbors import KNeighb
from sklearn.tree import DecisionTree
from sklearn.metrics import accuracy_
```

funstion list

In function list it will hold all the implement function for features use and robustness.

```
X_train_variants = {}
X_test_variants = {}
y_train_variants = {}
y_test_variants = {}
```

```
def my_plot_importance(booster, figsiz
    plt.rcParams["figure.figsize"]
    plot_importance(booster=booster)
```

```
def plot_footprint(footprint_row, titl

    x_values = [footprint_row[f'x
    y_values = [footprint_row[f'y

    plt.figure(figsize=(8, 12))
    plt.scatter(x_values, y_values,

    for i, (x, y) in enumerate(
        plt.text(x, y, str(i),

    plt.xlabel('Width (pixels)')
    plt.ylabel('Height (pixels)')
    plt.title(title)
    plt.gca().invert_yaxis()
    plt.show()
```

distance clataiton

```
def euclidean_distance(df, x1, y1, x
    return np.sqrt((df[x1] - df[x
```

lengths and widths

```
def lengths_widths_calculation(df):
    df_lengths_widths = df.copy()
    df_lengths_widths['lengths'] =
    df_lengths_widths['widths'] =
    return df_lengths_widths
```

7 foot point

```
def point7_calculation(df):
    df_7_point_footprints = df.cop
    df_7_point_footprints['T1'] =
    df_7_point_footprints['T2'] =
    df_7_point_footprints['T3'] =
    df_7_point_footprints['T4'] =
    df_7_point_footprints['T5'] =
    df_7_point_footprints['BAB'] =
    df_7_point_footprints['BAH'] =
```

```
        return df_7_point_footprints
```

IQR missing value function

```
def IQR(df):
        df_outlier_IQR = df.copy()
        for column in df_outlier_IQR.
                Q1 = df_outlier_IQR[cc
                Q3 = df_outlier_IQR[cc
                IQR = Q3 - Q1
                lower_bound = Q1 - 3
                upper_bound = Q3 + 3

                df_outlier_IQR[column]

        return df_outlier_IQR
```

Cap Outliers and Apply Robust and standard Scaling

```
def cap_outliers_and_scale(df):
        df_outlier_capped_scale = df.c

        for column in df_outlier_capp
                Q1 = df_outlier_cappec
                Q3 = df_outlier_cappec
                IQR = Q3 - Q1
                lower_bound = Q1 - 1
                upper_bound = Q3 + 1
                df_outlier_capped_scale[

        robust_scaler = RobustScaler()
        df_outlier_capped_scale = robu

        standard_scaler = StandardScal
        df_outlier_capped_scale = star

        return pd.DataFrame(df_outlier_
```

Winsorization

```
def Winsorization(df):
        df_winsorized = df.copy()
        for column in df_winsorized.c
                df_winsorized[column] =
        return df_winsorized
```

z_score

```
def z_score(df):
    df_z_score = df.copy()
    z_threshold = 4
    for column in df_z_score.colu
        z_scores = stats.zscor
        df_z_score[column] = r
        df_z_score[column] = r
    return df_z_score
```

isolation_forest

```
def isolation_forest(df, contaminatior
    df_isolation = df.copy()
    model = IsolationForest(contam

    model.fit(df_isolation)

    outlier_predictions = model.pr

    for column in df_isolation.cc
        median_value = df_isol
        df_isolation[column] =

    return df_isolation
```

step 1: Understanding the data

```
開始使用 AI 編寫或生成程式碼。
```

Although all landmarks are provided, it does not necessarily mean all of them will be positive for the model, therefore we will implement features engineering, This involves both adding new features and feature selection to improve model learning, details on feature engineering will be discussed in a later section.

The data has been standardized between 0 and 1, if needed, we can recover to the original values by scaling back to 2240x3200, this will bring us back to its true data form, for more data understanding.

The dataset contains 2,000 entries, which will be used to train the model and between them, x1 to y17 contain 6 to 17 missing values in between that require handling to ensure the data quality, and we will experiment with different imputation methods in step 3.

```
footprints_data = pd.read_csv('SexLanc
print(footprints_data.info())
footprints_data.head()
```

```
footprints_data.isnull().sum()
```

In this step, on "Box Plots for Outliers", outliers are present on the dataset, for early outlier handling, we can scale back the standardized data and calculate basic length and width, as it is difficult to gain meaningful information from the basic box plots, by doing so, we can identify extreme outliers more easily and correct them manually if needed, this method allows us to clean data more consistently, as leaving unreasonable extreme outliers most likely hurt the robustness of the dataset and effectiveness of the deployment.

```
plt.figure(figsize=(15, 10))
sns.boxplot(data=footprints_data)
plt.xticks(rotation=90)
plt.title("Box Plots for Outliers")
plt.show()
```

```
width, height = 2240, 3200
```

```
original_scaled_data = footprints_data

for column in original_scaled_data.cc
    if column.startswith('x'):
        original_scaled_data[col
    elif column.startswith('y'):
        original_scaled_data[col

print(original_scaled_data.head())
```

```
plt.figure(figsize=(15, 10))
sns.boxplot(data=original_scaled_data)
plt.xticks(rotation=90)
plt.title("Box Plots for Outliers")
plt.show()
```

```
original_scaled_data_with_lengths_widths
```

```
for name, group in original_scaled_c
    plt.plot(group.lengths, group.v
plt.legend()
```

The graph below shows the length and width of each footprint, Based on it we can observe extreme outliers, we will check if should we remove or correct these outliers, based on the landmark and dose it relistic.

```
lengths_upper_threshold = original_sca
lengths_lower_threshold = original_sca
widths_upper_threshold = original_scal
widths_lower_threshold = original_scal


big_feet = original_scaled_data_with_l
        (original_scaled_data_with_lengt
        (original_scaled_data_with_lengt
]

small_feet = original_scaled_data_with
        (original_scaled_data_with_lengt
        (original_scaled_data_with_lengt
]


print("Big Feet Data Points:")
print(big_feet)

print("\nSmall Feet Data Points:")
print(small_feet)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))


plt.scatter(original_scaled_data_with_le


plt.scatter(big_feet['lengths'], big_fe
plt.scatter(small_feet['lengths'], smal

plt.xlabel('Lengths')
plt.ylabel('Widths')
plt.legend()
plt.show()
```

```
small_foot_1 = small_feet[
        (small_feet['lengths'] > 1600)
        (small_feet['widths'] > 0) &
]

small_foot_2 = small_feet[
        (small_feet['lengths'] > 2000)
        (small_feet['widths'] > 400)
]

big_foot_1 = big_feet[
        (big_feet['lengths'] > 3100)
        (big_feet['widths'] > 1900) &
]

big_foot_2 = big_feet[
        (big_feet['lengths'] > 2200)
        (big_feet['widths'] > 100) &
]
```

As shown in the graph below, the coordinates of the small feet, has shown a spread that are hardly can be recognized as human, therefore drop these data point from the dataset should improve the dataset.

On the other hand both of the big foot seems to be showing a normal spared therefore they will be kept.

Hill Lam
2024年11月6日

I have use iqr base dataset for all of them now to just have atry, if need change it back

```
plot_footprint(small_foot_1.iloc[0], 'S
```

```
plot_footprint(small_foot_2.iloc[0], 'S

plot_footprint(big_foot_1.iloc[0], 'Big

plot_footprint(big_foot_2.iloc[0], 'Big
```

```
indices_to_drop = [small_foot_1.index[

footprints_data = footprints_data.drop
```

```
footprints_data.describe().T
```

The graph below shown there is a class imbalance on the dataset, it will be the best practice to implement the Synthetic Minority Over-sampling Technique (SMOTE) to prevent model bias. SMOTE will generate synthetic samples for the minority class, this can help to balance the dataset and improve the model's ability to generalize both classes.

```
barplot=(sns.countplot(data= footprints
plt.title('0 v/s 1\n')
```

```
corr = footprints_data.corr(method='sp

triangle = np.triu(corr)

plt.figure(figsize=(16, 7))
sns.heatmap(data=corr, annot=True, mas
```

```
plt.figure(figsize=(20,12))
sns.set_context('notebook',font_scale =
sns.heatmap(footprints_data.corr(),annot
plt.tight_layout()
```

```
ax = sns.heatmap(
        corr,
        vmin=-1, vmax=1, center=0,
        cmap=sns.diverging_palette(20,
        square=True
)
ax.set_xticklabels(
        ax.get_xticklabels(),
        rotation=45,
```

```
            horizontalalignment='right'
    );
    ax
```

Th is dataset has shown there is no duplicated, therefore no action needed

```
footprints_data.duplicated().value_count
```

step 2: data processing

outliers handling

```
footprints_data_df = footprints_data.c
footprints_data_df.describe().T
```

we will uses 4 method to handle outliers, and we will not be dropping outliers, because as seen there is meaningful data with in the outliers, therefore Dropping them could bring loss of important patterns.

1.Basic IQR Method:

- The Interquartile Range (IQR) is a standard technique used to identify outliers, the outliers will be capped to a bounds, to limit their range.

2.Cap Outliers and Apply Robust Scaling

- Similar to the IQR method but apply robusts and standard scaling to create deviation of the data.

3.Winsorization

- limits exteme values by capping them within specified boundaries.

4.Use Z score

- uses standard deviation to identify outliers, which are then replaced replaced with the median to reduce their effect.

```
footprints_data_df = footprints_data.c
```

use IQR for outliners

```
footprints_data_df = footprints_data.c
footprints_data_df.describe().T
```

```
X = footprints_data_df.drop('sex', ax
y = footprints_data_df['sex']
```

```
X_train, X_test, y_train, y_test =

X_train_iqr = IQR(X_train)
X_test_iqr = IQR(X_test)

y_train_iqr = y_train.loc[X_train_iqr.
y_test_iqr = y_test.loc[X_test_iqr.inc
```

```
X_train_variants['IQR'] = X_train_iqr
X_test_variants['IQR'] = X_test_iqr
y_train_variants['IQR'] = y_train_iqr
y_test_variants['IQR'] = y_test_iqr
```

```
import matplotlib.pyplot as plt
lengths_widths_df_iqr = pd.concat([X_t

for name, group in lengths_widths_df
        plt.plot(group.x1, group.x14,
plt.legend()
```

```
X_train_iqr.describe()
```

Cap Outliers and Apply Robust and standard Scaling

```
X_train, X_test, y_train, y_test =

X_train_robust = cap_outliers_and_scal
X_test_robust = cap_outliers_and_scale
y_train = y_train.reset_index(drop=Tru
y_test = y_test.reset_index(drop=True)

y_train_robust = y_train.loc[X_train_r
y_test_robust = y_test.loc[X_test_robu
```

```
X_train_variants['RobustScaling'] = X_
X_test_variants['RobustScaling'] = X_t
y_train_variants['RobustScaling'] = y_
y_test_variants['RobustScaling'] = y_t
```

```
lengths_widths_df_robust = pd.concat([

for name, group in lengths_widths_df
        plt.plot(group.x1, group.x14,
plt.legend()
```

```
lengths_widths_df_robust.describe()
```

for now we have done with the grouping onto lengths and widths which we have mentioned earlier and we have use 2 ways to deal with outliers

Winsorization

```
X_train, X_test, y_train, y_test =

X_train_Winsorization = Winsorization(
X_test_Winsorization = Winsorization(X

y_train_Winsorization = y_train.loc[X_
y_test_Winsorization = y_test.loc[X_te
```

```
X_train_variants['Winsorization'] = X_
X_test_variants['Winsorization'] = X_t
y_train_variants['Winsorization'] = y_
y_test_variants['Winsorization'] = y_t
```

```
lengths_widths_df_Winsorization = pd.c

for name, group in lengths_widths_df
```

```
        plt.plot(group.x1,  group.x14,
plt.legend()
```

```
lengths_widths_df_Winsorization.describe
```

z_score_df

```
X_train,  X_test,  y_train,  y_test  =

X_train_z_score  =  z_score(X_train)
X_test_z_score  =  z_score(X_test)
y_train  =  y_train.reset_index(drop=Tru
y_test  =  y_test.reset_index(drop=True)

y_train_z_score  =  y_train.loc[X_train_
y_test_z_score  =  y_test.loc[X_test_rob
```

```
X_train_variants['z_score']  =  X_train_
X_test_variants['z_score']  =  X_test_z_
y_train_variants['z_score']  =  y_train_
y_test_variants['z_score']  =  y_test_z_
```

```
lengths_widths_df_z_score  =  pd.concat(

for  name,  group  in  lengths_widths_df
        plt.plot(group.x1,  group.x14,
plt.legend()
```

```
lengths_widths_df_z_score.describe()
```

step 3: missing value handle

At this step, we group the data that has been processed for outliers handling, assign key values for easier management, then we apply KNN Imputer and Iterative Imputer, this avoids data leakage meanwhile being efficient. These two imputation methods are chosen because:

1. KNN Imputer:

- The KNN fills up missing values by averaging the values from the nearest neighbours, this helps missing values while keeping the patterns related to those neighbours.

2. Iterative Imputer:

- The Iterative predicts each missing value by running an iterative regression.

```
datasets = {
        'IQR': (X_train_iqr, X_test_ic
        'RobustScaling': (X_train_robus
        'Winsorization': (X_train_Winsc
        'Zscore': (X_train_z_score, X_
}

KNNImputer = KNNImputer(n_neighbors=4)
IterativeImputer = IterativeImputer(ma

imputed_variants = {}

for variant_name, (X_train, X_test,

        X_train_imputed = pd.DataFrame
        X_test_imputed = pd.DataFrame(

        key = f"{variant_name}_Iterati
        imputed_variants[key] = (X_tra

for variant_name, (X_train, X_test,

        X_train_imputed = pd.DataFrame
        X_test_imputed = pd.DataFrame(

        key = f"{variant_name}_knn"
        imputed_variants[key] = (X_tra

print(imputed_variants.keys())
```

step 4: baseline test before features engineering

In this step, we will prepare a baseline test for the performance of the models, as at this point we have already cleaned up our data with the basic method we have covered, and now the data are already for a baseline test and we will choose models that perform well for further development.

```
models = {
        "Logistic Regression": Logisti
        "Gaussian Naïve Bayes": Gauss
        "Support Vector Machine": SVC
        "KNN Classifier": KNeighborsCl
        "Decision Tree": DecisionTree(
        "Random Forest": RandomForest(
        "Gradient Boosting": GradientE
        "AdaBoost": AdaBoostClassifier(
        "XGBoost": XGBClassifier(use_la
        "SGDOneClassSVM":linear_model.S(
}
```

The results from the baseline test has shown a acceptable performance concider we have only processed with basic methodology to clean the data.

The list below has shown the results in order of accuracy score, along with the model, variant of the dataset, outliers method and imputation method. According to the results, the best-performing model so far is XGBoost, which uses IQR_Iterative and it able to achieve 0.8250, This suggests further development of XGBoost will be worthwhile, followed by Random Forest and Gradient Boosting, with a different set of variants, it has also shown there is not yet have a clear idea of which variants will be the best for us to achieve our goal there for more test will be needed in future steps.

```
results = []

for model_name, model in models.iten
    for dataset_name, (X_train, )

        model.fit(X_train, y_tr
        y_pred = model.predict

        accuracy = accuracy_sc

        results.append({
            "Model": model_
            "Variant": data
            "Outlier Handli
            "Imputation Met
            "Accuracy": acc
        })

results_df = pd.DataFrame(results)

results_df = results_df.sort_values(by

pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', Nc
pd.set_option('display.width', 1000)
print(results_df)
```

step 5: features engineering

At this step we will implement two different kinds of feature engineering, first, we will simply calculate the fundamental lengths and widths of the footprints, to obtain extra measurements that can help understanding of the size.

Second approach involves using a more unique feature extraction technique based on research of Abledu et al. (2015), published by the

NIH (National Library of Medicine), they have implemented an calculation of Seven dimensions–length of each toe to the bottom (t1 to t5), breadth at the ball (BAB) and breadth at heel (BAH), will this approach they have able to achieve a remarkable accuracy in a similar tasks, therefore we will implement this along with the basic lengths and widths calculation.

ref

Abledu, J. K., Abledu, G. K., Offei, E. B., and Antwi, E. M., 2015. Determination of sex from footprint dimensions in a Ghanaian population [online]. PloS one. Available from: https://pmc.ncbi.nlm.nih.gov/articles/PMC4596846/ [Accessed 5 Nov 2024].

```
feature_engineered_variants = {}

for variant_name, (X_train, X_test,

    X_train_lengths = lengths_widt
    X_test_lengths = lengths_widtl

    key = f"{variant_name}_lengths
    feature_engineered_variants[key]

for variant_name, (X_train, X_test,

    X_train_point7 = point7_calcul
    X_test_point7 = point7_calcula

    key = f"{variant_name}_point7"
    feature_engineered_variants[key]

print(feature_engineered_variants.keys()
```

step 6: Testing all the model after features

engineering (baseline)

As we have now implement features engineering, it will be beneficial to did an other baseline test to have a better understanding dose the features we create bring positive or negative impact to the model learning

```
X_train_point7, X_test_point7, y_trair
```

```
engineered_results = []

for model_name, model in models.iten
    for dataset_name, (X_train, Y

        model.fit(X_train, y_tr
        y_pred = model.predict

        accuracy = accuracy_sc

        engineered_results.apper
            "Model": model_
            "Variant": data
            "Outlier Handli
            "Imputation Met
            "Feature Engine
            "Accuracy": acc
        })

engineered_results_df = pd.DataFrame(e

engineered_results_df = engineered_res

pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', Nc
pd.set_option('display.width', 1000)
print(engineered_results_df)
```

step 7: Hyperparameter Tuning For model

```
print("Imputed Variants:")
print(imputed_variants.keys())

print("Feature-Engineered Variants:")
print(feature_engineered_variants.keys()
```

```
print("Holdout Imputed Variants:")
print(imputed_variants_holdout.keys())

print("Holdout Feature-Engineered Vari
print(holdout_feature_engineered_variant
```

Hyperparameter Tuning for XGBoost (kaggle 0.8334)

```
from sklearn.model_selection import F

best_variant_name = 'Winsorization_knn
X_train_best, X_test_best, y_train_bes

param_grid = {
    'learning_rate': [0.01, 0.05,
    'max_depth': [3, 5, 7, 9, 1
    'n_estimators': [50, 100, 20(
    'subsample': [0.5, 0.7, 0.9,
    'colsample_bytree': [0.5, 0.7,
    'colsample_bylevel': [0.5, 0.7
    'colsample_bynode': [0.5, 0.7,
    'min_child_weight': [1, 3, 5,
    'gamma': [0, 0.1, 0.3, 0.5,
    'reg_lambda': [0.5, 1, 1.5],
    'reg_alpha': [0, 0.5, 1, 1.5
    'booster': ['gbtree', 'dart'],
    'tree_method': ['auto', 'exact
}

xgb = XGBClassifier(use_label_encoder=

grid_search = RandomizedSearchCV(
    xgb, param_grid, n_iter=50, c
)
grid_search.fit(X_train_best, y_train_b

print("Best Parameters:", grid_search.
print("Best Accuracy from Grid Searc
```

```
best_variant_name = 'Winsorization_Ite
X_train_best, X_test_best, y_train_bes

param_grid = {
    'n_estimators': [50, 100, 20(
    'learning_rate': [0.05, 0.1,
    'max_depth': [3, 5, 7],
    'subsample': [0.5, 0.6, 1.0],
    'colsample_bytree': [0.5, 0.7,
    'gamma': [0, 0.1, 0.3, 0.5,
    'min_child_weight': [1, 3, 5,
    'reg_alpha': [0, 0.1, 1],
    'reg_lambda': [0.5, 1, 2, 5]
```

```
    }

    xgb = XGBClassifier(use_label_encoder=

    grid_search = GridSearchCV(xgb, param
    grid_search.fit(X_train_best, y_train_b

    print("Best Parameters:", grid_search.
    print("Best Accuracy from Grid Searc

    best_model = grid_search.best_estimato
    y_pred_best = best_model.predict(X_tes
    accuracy_best = accuracy_score(y_test_
    print(f"Test Accuracy for Best Model
```

RandomizedSearchCV first than GridSearchCV to safe time as there will be less to try on and close down the candidates

```
    best_variant_name = 'Winsorization_knn
    X_train, X_test, y_train, y_test =
    print(f"Running model for variant: 

    all_accuracies = []
    num_runs = 10

    for i in range(num_runs):
        X_train_best, X_test_best, y_t
            X_train, y_train, test
        )

        print(f"Features before fittin

        model = XGBClassifier(
            learning_rate=0.05,
            max_depth=5,
            n_estimators=100,
            subsample=0.5,
            eval_metric='logloss',
            reg_lambda=0.5,
            reg_alpha=1,
            min_child_weight=5,
            gamma=0.1,
            colsample_bytree=0.9,
            random_state=i,
        )

        model.fit(X_train_best, y_train
        y_pred = model.predict(X_test_

        accuracy = accuracy_score(y_te
        all_accuracies.append(accuracy)

        scores = cross_validate(
```

```
                model,  X_train_best,  y
        )

        print(f"Run  {i  +  1}:")
        print(f"Accuracy  (Testing):  {a
        print(f"Accuracy  (CV  Mean):  -

conf_matrix  =  confusion_matrix(y_test_
sns.heatmap(conf_matrix,  annot=True,  f
plt.title('Confusion  Matrix')
plt.show()

print("\nSummary  of  accuracies  across
print(f"Mean  accuracy  over  {num_runs}

print(classification_report(y_test_best,
```

training to test on the robustness of the process we are getting 85% with almost the same CV mean which means it is generalising well

```
best_xgb  =  XGBClassifier(
        learning_rate=0.05,
        max_depth=5,
        n_estimators=100,
        subsample=0.5,
        eval_metric='logloss',
        reg_lambda=0.5,
        reg_alpha=1,
        min_child_weight=5,
        gamma=0.1,
        colsample_bytree=0.9,
)

best_xgb.fit(X_train_best,  y_train_best)
```

model output for more data understanding later

XGBoost play ground

```
def  my_plot_importance(booster,  figsiz
        plt.rcParams["figure.figsize"]
        plot_importance(booster=booster)

my_plot_importance(best_xgb,  figsize=(1
```

```
import  shap
```

```
explainer = shap.TreeExplainer(best_xg
shap_values = explainer.shap_values(X_
shap.summary_plot(shap_values, X_test_b
```

```
from sklearn.inspection import permut

result = permutation_importance(best_x
for i in result.importances_mean.args
    print(f"{X_test_best.columns[i]}
```

up to now we can see which features are more imporatnat which are not, it will allow us to do features selection, base on the infrmation above

```
data_imbalance = pd.concat([X_train_be

for name, group in data_imbalance.gr
    plt.plot(group.BAB, group.HB_ir
plt.legend()
```

```
barplot=(sns.countplot(data= data_imbal
plt.title('0 v/s 1\n')
```

as shown above there is heavy data imblance and there is ouliers with in the engineered features, to move forword for better XGBoost performacne we will implnemnt 3 different ways for outliners and ways uses SMOTE for class imblance than perform feature selection and compaire there perfomance together

```
X_train, X_test, y_train, y_test =

X_train_point7_iqr = IQR(X_train)
X_test_point7_iqr = IQR(X_test)

y_train_point7_iqr = y_train.loc[X_tra
y_test_point7_iqr = y_test.loc[X_test_
```

```
point7_df_iqr = pd.concat([X_train_poi

for name, group in point7_df_iqr.gro
```

```
        plt.plot(group.BAB, group.HB_ir
plt.legend()
```

```
X_train_point7_z_score = z_score(X_tra
X_test_point7_z_score = z_score(X_test
y_train = y_train.reset_index(drop=Tru
y_test = y_test.reset_index(drop=True)

y_train_point7_z_score = y_train.loc[X
y_test_point7_z_score = y_test.loc[X_t
```

```
point7_df_z_score = pd.concat([X_trair

for name, group in point7_df_z_score
        plt.plot(group.BAB, group.HB_ir
plt.legend()
```

```
X_train_point7_isolation_forest = isol
X_test_point7_isolation_forest = isola
y_train = y_train.reset_index(drop=Tru
y_test = y_test.reset_index(drop=True)

y_train_point7_isolation_forest = y_tr
y_test_point7_isolation_forest = y_tes
```

```
point7_df_isolation_forest = pd.concat

for name, group in point7_df_isolati
        plt.plot(group.BAB, group.HB_ir
plt.legend()
```

do the same for hold out

now we have done all 3 ways that we have talked about for daeling with the ouliners, and each of them have perfrom abit different which it will be provide a good variants on the outcome

```
from imblearn.over_sampling import SM
from imblearn.combine import SMOTETom
```

```
from imblearn.over_sampling import SM
from imblearn.combine import SMOTETom

point7_datasets = {
        'point7_IQR': (X_train_point7_i
```

```
        'point7_Zscore': (X_train_point
        'point7_isolationforest': (X_tr
}

XGBoost_outliers_variants = {}

for dataset_name, (X_train, X_test,

        smote = SMOTE()
        X_train_resampled, y_train_resa

        X_train_imputed = X_train_resa
        X_test_imputed = X_test

        key = f"{dataset_name}_SMOTE"
        XGBoost_outliers_variants[key]

for dataset_name, (X_train, X_test,

        smote = BorderlineSMOTE()
        X_train_resampled, y_train_resa

        X_train_imputed = X_train_resa
        X_test_imputed = X_test

        key = f"{dataset_name}_Borderl
        XGBoost_outliers_variants[key]

for dataset_name, (X_train, X_test,

        smote = SVMSMOTE()
        X_train_resampled, y_train_resa

        X_train_imputed = X_train_resa
        X_test_imputed = X_test

        key = f"{dataset_name}_SVMSMOT
        XGBoost_outliers_variants[key]

for dataset_name, (X_train, X_test,

        smote = RandomOverSampler(ranc
        X_train_resampled, y_train_resa

        X_train_imputed = X_train_resa
        X_test_imputed = X_test

        key = f"{dataset_name}_RandomS
        XGBoost_outliers_variants[key]

for dataset_name, (X_train, X_test,

        smote = SMOTETomek(random_stat
        X_train_resampled, y_train_resa

        X_train_imputed = X_train_resa
        X_test_imputed = X_test
```

```
        key = f"{dataset_name}_SMOTETc
        XGBoost_outliers_variants[key]

    print(XGBoost_outliers_variants.keys())
```

```
    X_train_point7_IQR_SMOTE, X_test_point7

    train_data_point7_IQR_SMOTE = pd.conca
    test_data_point7_IQR_SMOTE = pd.concat

    barplot=(sns.countplot(data= train_data
    barplot=(sns.countplot(data= test_data_
    plt.title('0 v/s 1\n')
```

as shown the smote is applied only to the test set to avoid data leakage

```
    point7_smote_engineered_results_df = [

    for model_name, model in models.item
        for dataset_name, (X_train, X

            model.fit(X_train, y_tr
            y_pred = model.predict

            accuracy = accuracy_sc

            point7_smote_engineered_
                "Model": model_
                "Variant": data
                "Feature Engine
                "Imputation Met
                "Smote Method":
                "Accuracy": acc
            })

    point7_smote_engineered_df = pd.DataFr

    point7_smote_engineered_results_df = p

    pd.set_option('display.max_rows', 1000)
    pd.set_option('display.max_columns', No
    pd.set_option('display.width', 1000)

    print(point7_smote_engineered_results_df
```

for now the Accuracy seems like the same as before but we should try on XGBoost to get more inforemation about its performacne

```
from sklearn.metrics import classific

best_variant_name = 'point7_Zscore_SM(
X_train, X_test, y_train, y_test = 
print(f"Running model for variant: 

all_accuracies = []
num_runs = 10

for i in range(num_runs):
    X_train_best, X_test_best, y_t
        X_train, y_train, test
    )

    print(f"Features before fittir

    model = XGBClassifier(
        learning_rate=0.05,
        max_depth=5,
        n_estimators=100,
        subsample=0.5,
        eval_metric='logloss',
        reg_lambda=0.5,
        reg_alpha=1,
        min_child_weight=5,
        gamma=0.1,
        colsample_bytree=0.9,'
        random_state=43,
    )

    model.fit(X_train_best, y_trair
    y_pred = model.predict(X_test_

    accuracy = accuracy_score(y_te
    all_accuracies.append(accuracy)

    scores = cross_validate(
        model, X_train_best, y
    )

    print(f"Run {i + 1}:")
    print(f"Accuracy (Testing): {a
    print(f"Accuracy (CV Mean): 

conf_matrix = confusion_matrix(y_test_
sns.heatmap(conf_matrix, annot=True, f
plt.title('Confusion Matrix')
plt.show()

print("\nSummary of accuracies across
print(f"Mean accuracy over {num_runs}
print(classification_report(y_test_best,
```

we can see our performance have largely increased on different areas with out features selection, now we will move onto features selection.

```
smote_best_xgb = XGBClassifier(
        learning_rate=0.05,
        max_depth=5,
        n_estimators=100,
        subsample=0.5,
        eval_metric='logloss',
        reg_lambda=0.5,
        reg_alpha=1,
        min_child_weight=5,
        gamma=0.1,
        colsample_bytree=0.9,
)

smote_best_xgb.fit(X_train_best, y_train_best)
```

```
explainer = shap.TreeExplainer(smote_b
shap_values = explainer.shap_values(X_
shap.summary_plot(shap_values, X_test_b
```

we can see that even after we done isoforest and smotie, it stay the same as before because (look at i can talk about for smote)

```
def my_plot_importance(booster, figsiz
        plt.rcParams["figure.figsize"]
        plot_importance(booster=booster)

my_plot_importance(smote_best_xgb, figs
```

```
from sklearn.feature_selection import
from sklearn.model_selection import S

XGBoost_outliers_variants_features_selec

threshold_importance = 0.9
n_features_to_select = 25
cv = StratifiedKFold(n_splits=5, shuf

for dataset_name, (X_train, X_test,
        model = XGBClassifier()
        model.fit(X_train, y_train)

        importance_scores = model.get_

        importance_df = pd.DataFrame(I

        selected_features = importance
```

```
        feature_indices = [list(model.
        X_train_selected = X_train.ilc
        X_test_selected = X_test.iloc[

        key = f"{dataset_name}_importa
        XGBoost_outliers_variants_featur


for dataset_name, (X_train, X_test,
        model = XGBClassifier()
        model.fit(X_train, y_train)

        importance_scores = model.get_
        importance_df = pd.DataFrame(I
        selected_features = importance

        feature_indices = [list(model.
        X_train_filtered = X_train.ilc
        X_test_filtered = X_test.iloc[

        estimator = XGBClassifier()
        rfe = RFE(estimator, n_featur
        rfe.fit(X_train_filtered, y_tra

        rfe_selected_features_mask = X

        X_train_rfe = X_train_filtered
        X_test_rfe = X_test_filtered.I

        key = f"{dataset_name}_rfeAfte
        XGBoost_outliers_variants_featur

for dataset_name, (X_train, X_test,

        estimator = XGBClassifier()
        sfs = SequentialFeatureSelecto
        sfs.fit(X_train, y_train)

        X_train_sfs = X_train.loc[:,
        X_test_sfs = X_test.loc[:, sf

        key = f"{dataset_name}_sfsforw
        XGBoost_outliers_variants_featur


for dataset_name, (X_train, X_test,
        estimator = XGBClassifier(ranc

        rfecv = RFECV(
                estimator=estimator,
                step=1,
                cv=cv,
                scoring='accuracy',
                min_features_to_select=1
        )

        rfecv.fit(X_train, y_train)
```

```
        optimal_feature_count = rfecv.
        feature_ranking = rfecv.rankir
        total_mean_score = np.mean(rfe

        X_train_rfecv = X_train.loc[:,
        X_test_rfecv = X_test.loc[:,

        key = f"{dataset_name}_rfecv"
        XGBoost_outliers_variants_featu

        print(f"Dataset: {dataset_name}
        print(f"Optimal number of fea
        print(f"Cross-validation scores
        print(f"Total mean score: {tc


print(XGBoost_outliers_variants_features
print(importance_df)
```

```
print(imputed_variants.keys())
print(feature_engineered_variants.keys()
print(XGBoost_outliers_variants.keys())
print(XGBoost_outliers_variants_features
```

```
X_train_test, X_test_test, y_train_tes

data_check = pd.concat([X_train_test,

data_check.describe().T
```

```
X_train_test, X_test_test, y_train_tes

X_train_test = pd.DataFrame(X_train_te

y_train_test = y_train_test.reset_inde

data_check = pd.concat([X_train_test,

data_check.describe().T
```

```
model_results = []

for dataset_name, (X_train, X_test,

                smote_best_xgb.fit(X_tra
                y_pred = smote_best_xg

                accuracy = accuracy_sc

                model_results.append({
                        "Model": model_
                        "Variant": data
                        "Feature Engine
```

```
                    "Imputation Met
                    "Smote Method":
                    "Features Selec
                    "Accuracy": acc
                })

    model_results_df = pd.DataFrame(model_

    model_results_df = model_results_df.sc

    pd.set_option('display.max_rows', 1000(
    pd.set_option('display.max_columns', Nc
    pd.set_option('display.width', 10000)
    print(model_results_df)
```

point7_Zscore_SMOTETomek_importanceScore 88%

point7_IQR_SMOTETomek_importanceScore and 88%

point7_IQR_SMOTETomek_rfecv 89% (have the best for now overall)

point7_IQR_SMOTETomek_rfeAfterImportanceFiltered 89%

```
    best_variant_name = 'point7_Zscore_SM(
    X_train, X_test, y_train, y_test =
    print(f"Running model for variant:

    all_accuracies = []
    num_runs = 10

    for i in range(num_runs):
        X_train_best, X_test_best, y_t
            X_train, y_train, test
        )

        print(f"Features before fittir

        model = XGBClassifier(
            learning_rate=0.05,
            max_depth=11,
            n_estimators=200,
            subsample=0.5,
            eval_metric='logloss',
            reg_lambda=1.5,
            reg_alpha=0.5,
            min_child_weight=10,
            gamma=0.5,
            colsample_bytree=1,
            colsample_bynode=1,
            colsample_bylevel=0.5,
```

```
            booster='gbtree',
            random_state=42,
        )

        model.fit(X_train_best, y_trair
        y_pred = model.predict(X_test_

        accuracy = accuracy_score(y_te
        all_accuracies.append(accuracy)

        scores = cross_validate(
            model, X_train_best, y
        )

        print(f"Run {i + 1}:")
        print(f"Accuracy (Testing): {a
        print(f"Accuracy (CV Mean): 

    conf_matrix = confusion_matrix(y_test_
    sns.heatmap(conf_matrix, annot=True, f
    plt.title('Confusion Matrix')
    plt.show()

    print("\nSummary of accuracies across
    print(f"Mean accuracy over {num_runs}

    print(classification_report(y_test_best,
```

```
all_runs_results = []

num_runs = 10

for dataset_name, (X_train, X_test, y_train, y_test) in XGBoost_outliers_variants_featur

    all_accuracies = []
    for i in range(num_runs):

        X_train_best, X_test_best, y_train_best, y_test_best = train_test_split(
            X_train, y_train, test_size=0.2, random_state=i
        )

        print(f"Features before fitting (run {i + 1}): {X_train_best.columns}")
        print(f"Running model for variant: {dataset_name}")

        model = XGBClassifier(
            learning_rate=0.05,
            max_depth=10,
            n_estimators=200,
            subsample=0.8,
            objective='reg:squarederror',
            reg_lambda=1.5,
            reg_alpha=1.5,
            min_child_weight=10,
            gamma=0.5,
            colsample_bytree=1,
            colsample_bynode=1,
            colsample_bylevel=0.5,
            booster='gbtree',
```

```
                random_state=i
            )

            model.fit(X_train_best, y_train_best)

            y_pred = model.predict(X_test_best)
            accuracy = accuracy_score(y_test_best, y_pred)
            all_accuracies.append(accuracy)

            scores = cross_validate(
                model, X_train_best, y_train_best, cv=5, return_train_score=True,
            )

            print(f"\nRun {i + 1}:")
            print(f"Accuracy (Testing): {accuracy:.2f}")
            print(f"Accuracy (CV Mean): {np.mean(scores['test_score']):.2f} (+/- {np.s

        mean_accuracy = np.mean(all_accuracies)
        std_accuracy = np.std(all_accuracies)
        all_runs_results.append({
            "Dataset": dataset_name,
            "Mean Accuracy": mean_accuracy,
            "Std Accuracy": std_accuracy,
            "Details": X_train.columns.tolist()
        })

    print("\nSummary of accuracies across runs:")
    for result in all_runs_results:
        print(f"Dataset: {result['Dataset']}, Mean Accuracy: {result['Mean Accuracy']:.2f}

    results_df = pd.DataFrame(all_runs_results)
    sorted_results_df = results_df.sort_values(by="Mean Accuracy", ascending=False)

    print("\nSorted Results by Accuracy:")
    print(sorted_results_df)

    import ace_tools as tools
    tools.display_dataframe_to_user(name="Sorted Model Results by Accuracy", dataframe=sorted_
```

output model and safe work state

```
    best_variant_name = 'point7_Zscore_SM(
    XGB_finial_X_train, X_test, XGB_finial

    best_XGB_After_proccess = XGBClassifie
                learning_rate=0.05,
                max_depth=11,
                n_estimators=200,
                subsample=0.5,
                eval_metric='logloss',
                reg_lambda=1.5,
                reg_alpha=0.5,
                min_child_weight=10,
                gamma=0.5,
                colsample_bytree=1,
                colsample_bynode=1,
```

```
                colsample_bylevel=0.5,
                booster='gbtree',
                random_state=42,
        )

best_XGB_After_proccess.fit(XGB_finial_)
```

```
import joblib
joblib.dump(best_XGB_After_proccess, 'b
```

```
best_XGB_After_process = joblib.load('
y_pred = best_XGB_After_process.predic
```

```
y_pred = best_XGB_After_process.predic
y_pred
```

```
joblib.dump(best_XGB_After_process, 'be
```

```
print(imputed_variants.keys())
print(feature_engineered_variants.keys()
print(XGBoost_outliers_variants.keys())
print(XGBoost_outliers_variants_features
```

Hyperparameter Tuning for Gradient Boosting (kaggle 0.8505)

```
from skopt import BayesSearchCV

best_variant_name = 'point7_IQR_SMOTET
X_train, X_test, y_train, y_test =

param_space = {
        'n_estimators': (50, 100, 20(
        'learning_rate': (0.01, 0.05,
        'max_depth': (3, 5, 7),
        'min_samples_split': (2, 5, 1
        'min_samples_leaf': (1, 2, 5,
        'max_features': ['sqrt', 'log2
        'subsample': (0.7, 0.8, 1.0),
        'loss': ['log_loss', 'exponent
        'min_impurity_decrease': (0.001
        'warm_start': [True, False],
        'max_leaf_nodes': [None, 10,
        'n_iter_no_change': [None, 5,
        'tol': (0.0001, 0.001, 0.01)
}

model = GradientBoostingClassifier(ran
```

```
bayes_opt = BayesSearchCV(
        estimator=model,
        search_spaces=param_space,
        n_iter=50,
        scoring='accuracy',
        n_jobs=-1,
        cv=3,
        verbose=1,
        random_state=42
)

bayes_opt.fit(X_train, y_train)

print("Best Parameters: ", bayes_opt.
print("Best Accuracy from Grid Searc
```

```
best_variant_name = 'point7_IQR_SMOTET
X_train, X_test, y_train, y_test =

param_grid = {
        'n_estimators': [50, 100, 20(
        'learning_rate': [0.01, 0.05,
        'max_depth': [3, 5, 7],
        'min_samples_split': [2, 5, 1
        'min_samples_leaf': [1, 2, 5,
        'max_features': ['sqrt', 'log2
        'subsample': [0.7, 0.8, 1.0]
}


gb_model = GradientBoostingClassifier(

grid_search = GridSearchCV(estimator=g






grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.
print("Best Score:", grid_search.best_
```

```
all_runs_results = []

num_runs = 10

for dataset_name, (X_train, X_test,

        all_accuracies = []
        for i in range(num_runs):

                X_train_best, X_test_be
                        X_train, y_trai
                )
```

```
            print(f"Features before
            print(f"Running model

            model = GradientBoosti
                learning_rate=0.
                max_depth=7,
                n_estimators=50,
                subsample=1.0,
                max_features='lc
                min_samples_leaf
                min_samples_spli
                random_state=42,
                warm_start=False
                tol=0.001,
                min_impurity_dec
                max_leaf_nodes=N
                loss='exponentia
                n_iter_no_change
            )

            model.fit(X_train_best,

            y_pred = model.predict
            accuracy = accuracy_sc
            all_accuracies.append(ac

            scores = cross_validat
                model, X_train_
            )

            print(f"\nRun {i + 1}
            print(f"Accuracy (Testi
            print(f"Accuracy (CV M

        mean_accuracy = np.mean(all_ac
        std_accuracy = np.std(all_accu
        all_runs_results.append({
            "Dataset": dataset_name
            "Mean Accuracy": mean_
            "Std Accuracy": std_ac
            "Details": X_train.colu
        })

    print("\nSummary of accuracies across
    for result in all_runs_results:
        print(f"Dataset: {result['Datas

    results_df = pd.DataFrame(all_runs_res
    sorted_results_df = results_df.sort_va

    print("\nSorted Results by Accuracy:"
    print(sorted_results_df)

    import ace_tools as tools
    tools.display_dataframe_to_user(name="Sc
```

```
    best_variant_name = 'point7_Zscore_SMOTETomek_rfecv'
```

```
    X_train, X_test, y_train, y_test = XGBoost_outliers_variants_features_selected[best_varian
    print(f"Running model for variant: {best_variant_name}")

    all_accuracies = []
    num_runs = 10

    for i in range(num_runs):
        X_train_best, X_test_best, y_train_best, y_test_best = train_test_split(
            X_train, y_train, test_size=0.2, random_state=i
        )

        print(f"Features before fitting (run {i + 1}): {X_train_best.columns}")

        model = GradientBoostingClassifier(
            learning_rate=0.2,
            max_depth=7,
            n_estimators=50,
            subsample=1.0,
            max_features='log2',
            min_samples_leaf=1,
            min_samples_split=10,
            random_state=42,
            warm_start=False,
            tol=0.001,
            min_impurity_decrease=0.001,
            max_leaf_nodes=None,
            loss='exponential',
            n_iter_no_change=None
        )

        model.fit(X_train_best, y_train_best)
        y_pred = model.predict(X_test_best)

        accuracy = accuracy_score(y_test_best, y_pred)
        all_accuracies.append(accuracy)

        scores = cross_validate(
            model, X_train_best, y_train_best, cv=5, return_train_score=True, return_e
        )

        print(f"Run {i + 1}:")
        print(f"Accuracy (Testing): {accuracy:.2f}")
        print(f"Accuracy (CV Mean): {np.mean(scores['test_score']):.2f} (+/- {np.std(score

    conf_matrix = confusion_matrix(y_test_best, y_pred)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.show()

    print("\nSummary of accuracies across runs:")
    print(f"Mean accuracy over {num_runs} runs: {np.mean(all_accuracies):.2f} (+/- {np.std(a

    print(classification_report(y_test_best, y_pred))
```

```
    best_variant_name = 'point7_Zscore_SM(
    GB_finial_X_train, X_test, GB_finial_y
```

```
best_GB_After_proccess = GradientBoost
                learning_rate=0.2,
                max_depth=7,
                n_estimators=50,
                subsample=1.0,
                max_features='log2',
                min_samples_leaf=1,
                min_samples_split=10,
                random_state=42,
                warm_start=False,
                tol=0.001,
                min_impurity_decrease=0.
                max_leaf_nodes=None,
                loss='exponential',
                n_iter_no_change=None
        )

best_GB_After_proccess.fit(GB_finial_X_t
```

```
joblib.dump(best_GB_After_proccess, 'be
```

```
best_XGB_After_process = joblib.load('
y_pred = best_XGB_After_process.predic
```

```
y_pred = best_XGB_After_process.predic
y_pred
```

```
y_pred = best_gb_model.predict(X_test)
y_pred
```

Gradient Boosting play ground

```
X = p7_point_footprints_df_iqr_knn_bla
y = p7_point_footprints_df_iqr_knn_bla
x = 0
count = 0
num_runs = 1

for x in range (num_runs):

        count += 1
        model = GradientBoostingClassi






        SMOTE_iso_X_train, SMOTE_iso_X_
```

```
        model.fit(SMOTE_iso_X_train,  SM
        SMOTE_iso_y_pred  =  model.predi

scores  =  cross_validate(model,  X,  y,
precision  =  precision_score(SMOTE_iso_
recall  =  recall_score(SMOTE_iso_y_test

print(metrics.confusion_matrix(SMOTE_isc
print("\nAccuracy  (Testing):    %0.2f
print("Accuracy  (Testing):    %0.2f  (+
print("count:"  ,  count)
print("Precision:  %.2f"  %  precision)
print("recall:  %.2f"  %  recall)

from  sklearn.metrics  import  confusion
print(confusion_matrix(SMOTE_iso_y_test,
sns.heatmap(confusion_matrix(SMOTE_iso_y
```

```
SMOTE_iso_best_gb_model  =  GradientBoos
        n_estimators=100,
        learning_rate=1,
        max_depth=5,
        min_samples_split=5,
        min_samples_leaf=2,
)

SMOTE_iso_best_gb_model.fit(SMOTE_iso_X_
```

Hyperparameter Tuning for Support Vector Machines

SVM overall will do better after StandardScaler there for we will use it to improve SVM score

```
from  sklearn.preprocessing  import  Sta
from  sklearn.decomposition  import  PCA
from  sklearn.svm  import  SVC
from  skopt  import  BayesSearchCV

best_variant_name  =  'point7_IQR_SMOTET
X_train,  X_test,  y_train,  y_test  =


scaler  =  StandardScaler()
X_train_scaled  =  scaler.fit_transform(
X_test_scaled  =  scaler.transform(X_tes

pca  =  PCA(n_components=0.95)
X_train_pca  =  pca.fit_transform(X_trai
```

```
X_test_pca = pca.transform(X_test_scal

param_grid = {
        'C': (0.1, 1000, 'log-uniform
        'gamma': (0.001, 10, 'log-uni
        'kernel': ['rbf'],
        'tol': (1e-4, 1e-2, 'log-unif
        'max_iter': (1000, 10000),
        'class_weight': [None, 'balanc
}

model = SVC(random_state=42)

bayes_opt = BayesSearchCV(
        estimator=model,
        search_spaces=param_grid,
        n_iter=50,
        scoring='accuracy',
        n_jobs=-1,
        cv=3,
        verbose=1,
        random_state=42
)

bayes_opt.fit(X_train_pca, y_train)

print("Best Parameters:", bayes_opt.be
print("Best Score:", bayes_opt.best_sc
```

按兩下 (或按 Enter 鍵) 即可編輯

```
XGBoost_outliers_features_selected_scale

scalers = {
        'StandardScaler': StandardScale
        'RobustScaler': RobustScaler()
}

for scaler_name, scaler in scalers.i
        for variant_name, (X_train, Y

                X_train_scaled = pd.Da
                X_test_scaled = pd.Dat

                key = f"{variant_name}
                XGBoost_outliers_feature

print(XGBoost_outliers_features_selectec
```

```
X_train_test, X_test_test, y_train_tes

X_train_test = pd.DataFrame(X_train_te

y_train_test = y_train_test.reset_inde
```

```
data_check = pd.concat([X_train_test,

data_check.describe().T
```

```
X_train_test, X_test_test, y_train_tes

X_train_test = pd.DataFrame(X_train_te

y_train_test = y_train_test.reset_inde

data_check = pd.concat([X_train_test,

data_check.describe().T
```

```
all_runs_results = []

num_runs = 10

best_params = {
        'C': 1.1930801848463657,
        'class_weight': 'balanced',
        'gamma': 0.120601154417892,
        'kernel': 'rbf',
        'max_iter': 1000,
        'tol': 0.0001
}

for dataset_name, (X_train, X_test,

        all_accuracies = []
        for i in range(num_runs):

                X_train_best, X_test_be
                        X_train, y_trai
                )

                print(f"Features before
                print(f"Running model

                model = SVC(
                        C=best_params['C
                        kernel=best_para
                        gamma=best_param
                        class_weight=bes
                        max_iter=best_pa
                        tol=best_params[
                        random_state=42
                )

                model.fit(X_train_best,

                y_pred = model.predict
                accuracy = accuracy_sc
                all_accuracies.append(ac
```

```
            scores = cross_validat
                    model, X_train_
            )

            print(f"\nRun {i + 1}
            print(f"Accuracy (Testi
            print(f"Accuracy (CV M

        mean_accuracy = np.mean(all_ac
        std_accuracy = np.std(all_accu
        all_runs_results.append({
            "Dataset": dataset_name
            "Mean Accuracy": mean_
            "Std Accuracy": std_ac
            "Details": X_train.colu
        })

    print("\nSummary of accuracies across
    for result in all_runs_results:
        print(f"Dataset: {result['Datas

    results_df = pd.DataFrame(all_runs_res
    sorted_results_df = results_df.sort_va

    print("\nSorted Results by Accuracy:"
    print(sorted_results_df)

    tools.display_dataframe_to_user(name="Sc
```

```
    best_variant_name = 'point7_Zscore_SMC
    X_train, X_test, y_train, y_test =
    print(f"Running model for variant: 

    all_accuracies = []
    num_runs = 10

    for i in range(num_runs):
        X_train_best, X_test_best, y_t
            X_train, y_train, test
        )

        print(f"Features before fittin

        model = SVC(
            C =1.9650743261576813,
            class_weight = 'balanc
            gamma = 0.090070545592
            kernel = 'rbf',
            max_iter = 4979,
            tol = 0.01,
            random_state=42
        )

        model.fit(X_train_best, y_train
        y_pred = model.predict(X_test_

        accuracy = accuracy_score(y_te
```

```
        all_accuracies.append(accuracy)

        scores = cross_validate(
                model, X_train_best, y
        )

        print(f"Run {i + 1}:")
        print(f"Accuracy (Testing): {a
        print(f"Accuracy (CV Mean): 

conf_matrix = confusion_matrix(y_test_
sns.heatmap(conf_matrix, annot=True, f
plt.title('Confusion Matrix')
plt.show()

print("\nSummary of accuracies across
print(f"Mean accuracy over {num_runs}

print(classification_report(y_test_best,
```

Support Vector Machines play ground

```
best_variant_name = 'point7_Zscore_SMC
SCV_finial_X_train, X_test, SCV_finial

best_SVC_After_proccess = SVC(
                C =1.9650743261576813,
                class_weight = 'balanc
                gamma = 0.090070545592
                kernel = 'rbf',
                max_iter = 4979,
                tol = 0.01,
                random_state=42,
                probability=True
        )

best_SVC_After_proccess.fit(SCV_finial_X
```

```
joblib.dump(best_SVC_After_proccess, 'b
```

```
best_SVC_After_proccess = joblib.load(
y_pred = best_SVC_After_proccess.predi
```

```
print(y_pred)
```

Hyperparameter Tuning for Random Forest

```
best_variant_name = 'point7_Zscore_SMC
X_train, X_test, y_train, y_test =
```

```
param_grid = {
    'n_estimators': [50, 100, 20(
    'max_depth': [5, 10, 15, 20,
    'min_samples_split': [2, 5, 1
    'min_samples_leaf': [1, 2, 5,
    'max_features': ['sqrt', 'log2
    'bootstrap': [True],
    'max_leaf_nodes': [10, 20, 5(
    'min_impurity_decrease': [0.0,
    'criterion': ['gini', 'entropy
    'class_weight': [None, 'balanc
    'oob_score': [True, False]
}

model = RandomForestClassifier(random_

bayes_opt = BayesSearchCV(
    estimator=model,
    search_spaces=param_grid,
    n_iter=50,
    scoring='accuracy',
    n_jobs=-1,
    cv=3,
    verbose=1,
    random_state=42
)

bayes_opt.fit(X_train_pca, y_train)

print("Best Parameters:", bayes_opt.be
print("Best Score:", bayes_opt.best_sc
```

```
best_variant_name = 'point7_Zscore_SM(
X_train, X_test, y_train, y_test =
print(f"Running model for variant:

all_accuracies = []
num_runs = 10

for i in range(num_runs):
    X_train_best, X_test_best, y_t
        X_train, y_train, test
    )

    print(f"Features before fittir

    model = RandomForestClassifier
        class_weight =None,
        criterion = 'entropy',
        max_depth = None,
        max_features = None,
        max_leaf_nodes = 100,
        min_impurity_decrease =
        min_samples_leaf = 2,
        min_samples_split = 2,
        n_estimators = 200,
```

```
                oob_score  =  False,
                random_state=42
        )

        model.fit(X_train_best,  y_trair
        y_pred  =  model.predict(X_test_

        accuracy  =  accuracy_score(y_te
        all_accuracies.append(accuracy)

        scores  =  cross_validate(
                model,  X_train_best,  y
        )

        print(f"Run  {i  +  1}:")
        print(f"Accuracy  (Testing):  {a
        print(f"Accuracy  (CV  Mean):  

conf_matrix  =  confusion_matrix(y_test_
sns.heatmap(conf_matrix,  annot=True,  f
plt.title('Confusion  Matrix')
plt.show()

print("\nSummary  of  accuracies  across
print(f"Mean  accuracy  over  {num_runs}

print(classification_report(y_test_best,
```

```
best_variant_name  =  'point7_Zscore_SM(
RF_finial_X_train,  X_test,  RF_finial_y

best_RF_After_proccess  =  RandomForest(
                class_weight  =None,
                criterion  =  'entropy',
                max_depth  =  None,
                max_features  =  None,
                max_leaf_nodes  =  100,
                min_impurity_decrease  =
                min_samples_leaf  =  2,
                min_samples_split  =  2,
                n_estimators  =  200,
                oob_score  =  False,
                random_state=42
        )

best_RF_After_proccess.fit(RF_finial_X_t
```

step 8: Ensemble Learning

```
XGB_model  =  XGBClassifier(best_XGB_Aft
GBC_model  =  GradientBoostingClassifier
SVM_model  =  SVC(best_SVC_After_procces
RF_model  =  RandomForestClassifier(best
```

```
from sklearn.ensemble import VotingCl
from sklearn.metrics import accuracy_

voting_clf = VotingClassifier(
    estimators=[
        ('xgb', best_XGB_After_
        ('gbc', best_GB_After_p
        ('svm', best_SVC_After_
        ('rf', best_RF_After_pr
    ],
    voting='soft'
)

voting_clf.fit(X_train, y_train)

y_pred = voting_clf.predict(X_test)

print("Ensemble Model Accuracy:", acc
print("Confusion Matrix:\n", confusion
print("Classification Report:\n", clas
```

```
print(imputed_variants.keys())
print(feature_engineered_variants.keys()
print(XGBoost_outliers_variants.keys())
print(XGBoost_outliers_features_selectec
print(XGBoost_outliers_variants_features
```

```
XGBoost_outliers_variants = {'point7_l
```

```
from sklearn.ensemble import Stacking
from sklearn.linear_model import Logi

stacking_clf = StackingClassifier(
    estimators=[
        ('xgb', best_XGB_After_
        ('gbc', best_GB_After_p
        ('svm', best_SVC_After_
        ('rf', best_RF_After_pr
    ],
    final_estimator=LogisticRegressi
)
stacking_clf.fit(X_train, y_train)
y_pred_stack = stacking_clf.predict(X_
print("Stacking Ensemble Accuracy:",
print("Confusion Matrix:\n", confusion
print("Classification Report:\n", clas
```

```
X_train.describe().T
```

```
from sklearn.ensemble import VotingCl

ensemble_model = VotingClassifier(
    estimators=[
```

```
            ('xgb', best_xgb),
            ('gb', best_gb_model),

        ],
        voting='soft'
)

ensemble_model.fit(X_train, y_train)

y_pred_ensemble = ensemble_model.predi

accuracy_ensemble = metrics.accuracy_s
print("Ensemble Model Accuracy:", acc

print("Confusion Matrix:", metrics.cor
print("Classification Report:", metric
```

```
best_ensemble_Voting_model =VotingClass
                    estimators=[
                        ('xgb', best_X(
                        ('gbc', best_GE
                        ('svm', best_S\
                        ('rf', best_RF_
                    ],
                    voting='soft'
            )



# Assume X_train and y_train are t
best_ensemble_Voting_model.fit(X_train,
```

```
from sklearn.ensemble import Stacking

base_models = [
            ('xgb', best_XGB_After_proc
            ('gbc', best_GB_After_procc
            ('svm', best_SVC_After_proc
            ('rf', best_RF_After_procce
]

meta_model = LogisticRegression()

stacking_model = StackingClassifier(es
stacking_model.fit(X_train, y_train)
y_pred = stacking_model.predict(X_test

accuracy_ensemble = metrics.accuracy_s
print("Ensemble Model Accuracy:", acc

print("Confusion Matrix:", metrics.cor
print("Classification Report:", metric
```

```
best_ensemble_Stacking_model =Stacking(
                    estimators=[
                        ('xgb', best_X(
                        ('gbc', best_GE
                        ('svm', best_S\
                        ('rf', best_RF_
                    ],
            )

best_ensemble_model.fit(X_train, y_trai
```

hold_out set change

```
hold_out = pd.read_csv('SexLandmarks-t
```

```
hold_out_data_df = hold_out.copy()
```

```
hold_out_data_df = IQR(hold_out_data_c
```

```
hold_out_scaled_data = hold_out.copy()

for column in hold_out_scaled_data.cc
        if column.startswith('x'):
                hold_out_scaled_data[col
        elif column.startswith('y'):
                hold_out_scaled_data[col

print(hold_out_scaled_data.head())
```

```
lengths_upper_threshold = hold_out_sca
lengths_lower_threshold = hold_out_sca
widths_upper_threshold = hold_out_scal
widths_lower_threshold = hold_out_scal

big_feet = hold_out_scaled_data_with_l
        (hold_out_scaled_data_with_lengt
        (hold_out_scaled_data_with_lengt
]

small_feet = hold_out_scaled_data_witl
        (hold_out_scaled_data_with_lengt
        (hold_out_scaled_data_with_lengt
]

print("Big Feet Data Points:")
print(big_feet)

print("\nSmall Feet Data Points:")
print(small_feet)

import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(10, 6))

plt.scatter(hold_out_scaled_data_with_le

plt.scatter(big_feet['lengths'], big_fe
plt.scatter(small_feet['lengths'], smal

plt.xlabel('Lengths')
plt.ylabel('Widths')
plt.legend()
plt.show()
```

```
big_foot_1 = big_feet[
        (big_feet['lengths'] > 2200)
        (big_feet['widths'] > 1000) &
]
```

```
if not small_foot_1.empty:
        plot_footprint(small_foot_1.iloc
```

```
holdout_datasets = {
        'IQR': IQR(hold_out_data_df),
        'RobustScaling': cap_outliers_a
        'Winsorization': Winsorization(
        'Zscore': z_score(hold_out_data
}

imputed_variants_holdout = {}

for variant_name, hold_out_data in h
        hold_out_imputed = pd.DataFram
        key = f"{variant_name}_Iterati
        imputed_variants_holdout[key] =

for variant_name, hold_out_data in h
        hold_out_imputed = pd.DataFram
        key = f"{variant_name}_knn"
        imputed_variants_holdout[key] =

print(imputed_variants_holdout.keys())
```

```
holdout_feature_engineered_variants =

lengths_widths_temp_dict = {}

for variant_name, hold_out_data in i

        hold_out_lengths = lengths_wic

        key = f"{variant_name}_lengths

        lengths_widths_temp_dict[key] =

holdout_feature_engineered_variants.upda
```

```
point7_temp_dict = {}

for variant_name, hold_out_data in i

    hold_out_point7 = point7_calcu
    key = f"{variant_name}_point7'
    point7_temp_dict[key] = hold_c

holdout_feature_engineered_variants.upda

print(holdout_feature_engineered_variant
```

```
print(holdout_feature_engineered_variant
print(imputed_variants_holdout.keys())
```

```
hold_out_XGBoost_outliers_variants = ·

hold_out_data = holdout_feature_engine

hold_out_point7_datasets = {
    'point7_IQR': (IQR(Winsorizatic
    'point7_Zscore': (z_score(Winsc
    'point7_isolationforest': (isol
}

print(hold_out_point7_datasets.keys())
```

```
if 'point7_Zscore' in hold_out_point7

    display(hold_out_point7_datasets
else:
    print("The key 'point7_Zscore'
```

```
if 'point7_IQR' in hold_out_point7_da

    display(hold_out_point7_datasets
else:
    print("The key 'point7_IQR' c
```

```
if 'point7_isolationforest' in hold_c

    display(hold_out_point7_datasets
else:
    print("The key 'point7_isolati
```

```
selected_features = [ #it its from
        'x0', 'y0', 'x1', 'y2', 'x3'
        'x7', 'y7', 'x8', 'y8', 'x1(
        'x13', 'y13', 'y14', 'x17',
        'BAH', 'HB_index'
]
```

```
hold_out_data_filtered_unscaled = holc
```

```
hold_out_data_filtered_unscaled.describe
```

```
Used_in_model = 'point7_Zscore_SMOTETc
x_train_scale, x_test_scale, y_train_s
```

```
scaler = StandardScaler()
scaler.fit(x_test_scale)

hold_out_data_filtered_unscaled = holc
hold_out_data_filtered_unscaled = holc

try:
        hold_out_data_filtered_scaled =
                scaler.transform(hold_ou
                columns=hold_out_data_fi
        )
        print("Scaling successful.")
except ValueError as e:
        print("Error during scaling:",
```

```
print(hold_out_data_filtered_unscaled.he
print(hold_out_data_filtered_scaled.heac
```

```
print("Mean used by scaler: ", scal
print("Scale used by scaler: ", sca
```

```
scaler = StandardScaler()
scaler.fit(X_train_best)

hold_out_data_filtered_scaled = pd.Dat
        scaler.transform(hold_out_data_f
        columns=hold_out_data_filtered_u
)

print(hold_out_data_filtered_scaled.desc
```

```
print(hold_out_data_filtered_unscaled.he
print(hold_out_data_filtered_scaled.heac
```

Try to submitting it to kaggle

```
Used_in_model = 'point7_Zscore_SMOTETc
x_train_submit, x_test_submit, y_trair
```

```
print(x_train_submit.shape)
```

```
scaler = StandardScaler()
scaler.fit(x_train_submit)
```

```
x_train_submit_scaled = pd.DataFrame(s
x_train_submit_unscaled = x_train_subm
```

```
print(x_train_submit.shape)
print(y_train_submit.shape)
```

```
y_train_submit = pd.Series(y_train_sub
y_train_submit.reset_index(drop=True, i
y_train_submit = y_train_submit.values
```

```
best_XGB_After_proccess.fit(x_train_subm
best_GB_After_proccess.fit(x_train_submi

best_RF_After_proccess.fit(x_train_submi
best_SVC_After_proccess.fit(x_train_subm
```

best_ensemble_Voting_model submit

```
training_columns = x_train_submit.colu
hold_out_data_filtered_unscaled = holc

hold_out_data_filtered_scaled = pd.Dat
        scaler.transform(hold_out_data_f
        columns=hold_out_data_filtered_u
)
```

```
xgb_pred = best_XGB_After_proccess.pre
gbc_pred = best_GB_After_proccess.prec

rf_pred = best_RF_After_proccess.predi
svm_pred = best_SVC_After_proccess.pre
```

```
hold_out_pred = best_ensemble_Voting_n
```

```
RowID = np.array(hold_out_data_filtere
```

```
results = pd.DataFrame({'RowID': RowI
```

```
print(results)
```

```
results.to_csv('results.csv', index=Fal
```

```
'''!kaggle competitions submit -c bu
```

best_GB_After_proccess submit

```
best_GB_After_proccess.fit(x_train_submi

gb_pred = best_GB_After_proccess.predi

results = pd.DataFrame({
        'RowID': np.array(hold_out_data
        'Sex': gb_pred
})

results.to_csv('best_GB_After_proccess.c
print(results.head)
```

```
results.to_csv('best_GB_After_proccess.c
```

```
!kaggle competitions submit -c budm-
```

best_XGB_After_proccess submit

```
best_XGB_After_proccess.fit(x_train_subm

gb_pred = best_XGB_After_proccess.prec

results = pd.DataFrame({
        'RowID': np.array(hold_out_data
        'Sex': gb_pred
})

results.to_csv('best_XGB_After_proccess.
print(results.head)
```

```
results.to_csv('best_XGB_After_proccess.
```

```
!kaggle competitions submit -c budm-
```

best_RF_After_proccess submit

```
best_RF_After_proccess.fit(x_train_submi

gb_pred = best_RF_After_proccess.predi

results = pd.DataFrame({
        'RowID': np.array(hold_out_data
        'Sex': gb_pred
})

results.to_csv('best_RF_After_proccess.c
print(results.head)
```

```
results.to_csv('best_RF_After_proccess.c
```

```
!kaggle competitions submit -c budm-
```

best_SVC_After_proccess submit

```
best_SVC_After_proccess.fit(x_train_subm

gb_pred = best_SVC_After_proccess.prec
```