In [ ]:

```python
import numpy as np
import pandas as pd
import regex as re
from datetime import datetime

import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report, roc_auc_score, precision_reca
from sklearn.cluster import KMeans
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA

from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
```

In [ ]:

```python
imonitor = pd.read_csv('data/imonitor_1703.csv')
imonitor.head()
```

```
C:\Users\mogam\AppData\Local\Temp\ipykernel_20704\2747984450.py:1: DtypeWarning: Columns (1,4,11,15,24,25,42,56,62,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84) have mixed types. Specify dtype option on import or set low_memory=False.
  imonitor = pd.read_csv('data/imonitor_1703.csv')
```

Out[ ]:

	Survey ID	Created Date	Facility name and MFL Code if applicable	Facility ownership	Please specify	County	What is your month; and year of birth	How do you consider yourself?	What hi le educ compl
0	2390063	04-Dec-23	BABA DOGO HEALTH CENTRE	GOK	NaN	Nairobi	1977-09-03	Male	Pı ş
1	2390062	04-Dec-23	BABA DOGO HEALTH CENTRE	GOK	NaN	Nairobi	1972-08-12	Female	Seco ş
2	2390061	04-Dec-23	BABA DOGO HEALTH CENTRE	GOK	NaN	Nairobi	1984-08-31	Female	Pı ş
3	2390060	04-Dec-23	BABA DOGO HEALTH CENTRE	GOK	NaN	Nairobi	1977-05-07	Female	Pı ş
4	2390059	04-Dec-23	BABA DOGO HEALTH CENTRE	GOK	NaN	Nairobi	1987-06-13	Male	Voca train tech

5 rows × 85 columns

In [ ]:
```
imonitor.shape
```

Out[ ]: (46549, 85)

In [ ]:
```
# Find and drop columns that contain "Please specify" or "Please Specify"
cols_to_drop = [col for col in imonitor.columns if "Please specify" in col or "P

# Drop these columns from the DataFrame in a single operation
imonitor.drop(cols_to_drop, axis=1, inplace=True)
```

In [ ]:
```
imonitor.shape
```

Out[ ]: (46549, 69)

In [ ]:
```
imonitor.columns = imonitor.columns.map(lambda x: x.strip())
```

In [ ]:

```python
columns_to_drop = [
    "Survey ID",
    "Facility name and MFL Code if applicable",
    "What is your month; and year of birth",
    "How do you consider yourself?",
    "What is the highest level of education you completed?",
    "What is your current marital status?",
    "Which county do you currently live in?",
    "What are your sources of income?",
    "Facility name",
    "What did you like about the services you received?",
    "What did you not like about the services you received?",
    "In your opinion what would you like to be improved?",
    "In your opinion what can be done to improve access to the services you seek
    "Facility name denied service",
    "Why",
    "Were reasons provided as to why these services were not available?",
    "Were reasons provided as to why these services were not available?.1",
    "What are the barriers to uptake of VMMC by males 25+years and above?",
    "What are some of the current site level practices that community members li
    "What would you like this facility to change/do better?",
    "Throughout your visit what did you find interesting/pleasing about this fac
    "What do you think can be improved",
    "Anything else that you would like to mention?",
    "What are the top 1-3 things you like about this facility with regards to ca
]

# Drop the columns
imonitor.drop(columns=columns_to_drop, axis=1, inplace=True)
```

In [ ]:

```python
column_name_mapping = {
    "Created Date": "Date",
    "Organization name coordinating the feedback from the clients": "OrgFeedback
    "Facility ownership": "FacilityOwnership",
    "County": "FacilityCounty",
    "For how long have you been accessing services (based on the expected packag
    "Are you aware of the package of services that you are entitled to?": "Servi
    "According to you; which HIV related services are you likely to receive in t
    "Is there a service that you needed that was not provided?": "UnprovidedServ
    "Facility name no service": "UnprovidedServiceFacilityName",
    "For that service that was not provided; were you referred?": "ReferralForUn
    "If referred; did you receive the service where you were referred to?": "Ref
    "If Yes which Service/Test/Medicine": "ReceivedServiceDetail",
    "On a scale of 1 to 5; how satisfied are you with the package of services re
    "Do you face any challenges when accessing the services at the facility?": "
    "Common issues that can be added in the drop-down box": "CommonIssuesDropdow
    "Was confidentiality considered while you were being served?": "Confidential
    "Are there age-appropriate health services for specific groups?": "AgeApprop
    "Does the facility allow you to share your concerns with the administration?
    "Do you know your health-related rights as a client of this facility?": "Rig
    "Have you ever been denied services at this facility?": "ServiceDenial",
    "Are you comfortable with getting services at this facility": "ComfortWithSe
    "Have you ever been counseled?": "CounselingReceived",
    "Did you identify any gaps in the facility when you tried to access the serv
    "Service type": "ServiceGapsType",
    "Are the HIV testing services readily available when required?": "HIVTesting
    "Have you ever Interrupted your treatment?": "TreatmentInterruption",
    "Are the PMTCT services readily available when required?": "PMTCTServiceAvai
    "Are the HIV prevention; testing; treatment and care services adequate for K
```

```
    "Facility Level": "FacilityLevel",
    "Facility Operation times": "OperationTimes",
    "Facility Operation Days": "OperationDays",
    "What are your preferred days of visiting the facility": "PreferredVisitDays
    "What are your preferred time of visiting the facility": "PreferredVisitTime
    "On a scale of 1-5; how clean do you find the facility?": "FacilityCleanline
    "How do you reach this facility?": "FacilityAccessMode",
    "How long does it take to reach this facility?": "FacilityAccessTime",
    "On a scale of 1-5; how accessible do you find this facility?": "FacilityAcc
    "Do you consider the waiting time to be seen at this facility long?": "Waiti
    "how long do you wait on average to get a service; which service was that?":
    "Do you consider the waiting time for lab test results long?": "LabResultsWa
    "how long do you wait on average to get your lab test result?": "AverageLabR
    "Does the facility offer support groups?": "SupportGroupAvailability",
    "Specify the support group you belong to": "SpecifySupportGroup",
    "In your opinion are the services offered at this facility youth friendly?":
    "What measures have been put in place to create GBV awareness and its harmfu
    "PWD In your opinion are the services offered at this facility persons-with-
    "What are the top 1-3 things you don't like about this facility with regards
}

# Assuming imonitor is your DataFrame
df = imonitor.rename(columns=column_name_mapping)
```

In [ ]:
```
columns_to_clean1 = [
    'WaitingTimeOpinion',
    'LabResultsWaitingTimeOpinion'
]

def replace_dont_know(df, column):
    df[column] = df[column].replace("Dont Know", "Do not know", regex=False)
    return df

for column in columns_to_clean1:
    df = replace_dont_know(df, column)
```

In [ ]:
```
columns_to_clean2 = [
    'FacilityCleanliness',
    'FacilityAccessibility'
    ]

def replace_mixed_with_text(df, column_name):
    def replace_value(value):
        satisfaction_map = {
            1: 'Very Unsatisfied',
            2: 'Unsatisfied',
            3: 'Okay',
            4: 'Satisfied',
            5: 'Very Satisfied'
        }
        if isinstance(value, str) and value[0].isdigit():
            num = int(value[0])
        elif isinstance(value, int):
            num = value
        else:
            return value

        return satisfaction_map.get(num, value)
```

```
    df[column_name] = df[column_name].apply(replace_value)
    return df

for column in columns_to_clean2:
    df = replace_mixed_with_text(df, column)
```

In [ ]:
```
def standardize_satisfaction(df, column_name):
    # Mapping for consolidating variations of satisfaction levels
    satisfaction_map = {
        '5': 'Very Satisfied',
        5.0: 'Very Satisfied',
        '4': 'Satisfied',
        4.0: 'Satisfied',
        '3': 'Okay',
        3.0: 'Okay',
        '2': 'Unsatisfied',
        2.0: 'Unsatisfied',
        '1': 'Very Unsatisfied',
        1.0: 'Very Unsatisfied',
        'Dissatisfied': 'Unsatisfied'
    }

    # Replace values based on the map
    df[column_name] = df[column_name].replace(satisfaction_map)
    return df

df = standardize_satisfaction(df, 'ServiceSatisfaction')
```

In [ ]:
```
print(df['FacilityLevel'].value_counts())
```

```
FacilityLevel
4.0    4802
3.0    4515
2.0    2889
5.0    2240
1.0     556
6.0      14
Name: count, dtype: int64
```

In [ ]:
```
def standardize_facility(df, column_name):
    # Mapping for consolidating variations of satisfaction levels
    satisfaction_map = {
        1.0: 'Community Health Unit',
        2.0: 'Dispensaries and Private Clinics',
        3.0: 'Health Centers',
        4.0: 'Sub-County Hospitals',
        5.0: 'County Referral Hospitals',
        6.0: 'National Referral Hospitals',
    }

    # Replace values based on the map
    df[column_name] = df[column_name].replace(satisfaction_map)
    return df

df = standardize_facility(df, 'FacilityLevel')
```

In [ ]:
```
def replace_symbols_and_words(df, column_name):
    df[column_name] = df[column_name].str.replace('<', 'Less than', regex=False)
    df[column_name] = df[column_name].str.replace('>', 'More than', regex=False)
```

```
    df[column_name] = df[column_name].str.replace('minutes', 'mins', regex=False
    return df

df = replace_symbols_and_words(df, 'FacilityAccessTime')
```

In [ ]:
```
def replace_symbols_and_words2(df, column_name):
    df[column_name] = df[column_name].str.replace('Less than 30mins', 'Less than
    df[column_name] = df[column_name].str.replace('More than45 mins', 'More than
    return df

df = replace_symbols_and_words2(df, 'FacilityAccessTime')
```

In [ ]:
```
def convert_mixed_dates(date_column):
    """
    This function takes a Pandas Series of mixed dates and Excel serial dates an

    Parameters:
    date_column (pd.Series): A pandas Series with mixed date formats and serial

    Returns:
    pd.Series: A pandas Series with all dates converted to datetime objects.
    """
    # Define the epoch start for Excel's serial date format
    excel_epoch = pd.Timestamp('1899-12-30')
    converted_dates = []

    for date in date_column:
        if isinstance(date, str) and re.match(r'^\d+(\.\d+)?$', date):
            # If it's a string that looks like a serial date, convert it
            serial_value = float(date)
            converted_date = excel_epoch + pd.to_timedelta(serial_value, unit='D
        elif isinstance(date, (int, float)):
            # If it's a numeric type, assume it's a serial date
            converted_date = excel_epoch + pd.to_timedelta(date, unit='D')
        else:
            # Otherwise, try to parse it as a regular date
            converted_date = pd.to_datetime(date, errors='coerce')

        # Append the result, which will be NaT (Not a Time) if parsing failed
        converted_dates.append(converted_date)

    return pd.Series(converted_dates)

# Example usage, assuming 'df' is your DataFrame and 'Date' is the column to be
df['Date'] = convert_mixed_dates(df['Date'])
```

In [ ]:
```
def standardize_gbv_awareness(df, column_name):
    df[column_name] = df[column_name].str.replace('Is there a desk to report GBV
    df[column_name] = df[column_name].str.replace('Are there training events on
    return df

df = standardize_gbv_awareness(df, 'GBVAwarenessMeasures')
```

In [ ]:
```
def encode_multi_select(df, columns):
    # Iterate over the specified columns
    for col in columns:
        # Remove all whitespaces within each value and split based on ';'
        # This creates a Series of lists
        split_series = df[col].str.replace(' ', '').str.split(';')
```

```
        # Use the str.get_dummies() method on the Series of lists to perform one
        # This approach handles the separation and encoding in one step
        encoded = split_series.str.join('|').str.get_dummies()

        # Prefix the encoded column names to indicate their origin
        encoded.columns = [f"{col}_{option}" for option in encoded.columns]

        # Join the encoded dataframe with the original dataframe
        df = df.join(encoded)

        # Optionally, drop the original column if no longer needed
        # df.drop(col, axis=1, inplace=True)

    return df

# Specify the columns to encode
columns_to_encode = ['ExpectedHIVServices', 'OperationTimes', 'OperationDays', '

# Apply the function
df2 = encode_multi_select(df, columns_to_encode)
```

In [ ]:
```
df2.drop(columns=columns_to_encode, axis=1, inplace=True)
```

In [ ]:
```
missing_percentage = df2.isnull().mean() * 100

threshold = 60

columns_to_drop = missing_percentage[missing_percentage > threshold].index.tolis

print("Columns to drop:", columns_to_drop)

print("Number of columns to drop:", len(columns_to_drop))

df2.drop(columns=columns_to_drop, axis=1, inplace=True)

print("DataFrame shape after dropping columns:", df2.shape)
```

```
Columns to drop: ['ReferralForUnprovidedService', 'ReferralServiceReceived', 'Rec
eivedServiceDetail', 'CommonIssuesDropdown', 'ServiceGapsType', 'HIVTestingAvaila
bility', 'TreatmentInterruption', 'PMTCTServiceAvailability', 'KPServiceAdequac
y', 'FacilityLevel', 'FacilityCleanliness', 'FacilityAccessMode', 'FacilityAccess
Time', 'FacilityAccessibility', 'WaitingTimeOpinion', 'AverageWaitingTime', 'LabR
esultsWaitingTimeOpinion', 'AverageLabResultsWaitingTime', 'SupportGroupAvailabil
ity', 'SpecifySupportGroup', 'YouthFriendlyServices', 'PWDFriendlyServicesOpinio
n', 'TopFacilityDislikes']
Number of columns to drop: 23
DataFrame shape after dropping columns: (46549, 66)
```

In [ ]:
```
threshold_percentage = 100

threshold = len(df2.columns) * (threshold_percentage / 100)

df3 = df2.dropna(thresh=threshold).copy()

print("Original DataFrame shape:", df2.shape)
print("Cleaned DataFrame shape:", df3.shape)
```

```
rows_dropped = df2.shape[0] - df3.shape[0]
print("Rows dropped:", rows_dropped)
```

```
Original DataFrame shape: (46549, 66)
Cleaned DataFrame shape: (39862, 66)
Rows dropped: 6687
```

In [ ]:

```
def divide_date_column(df):
    # Change Date column to datetime type
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

    # Extract year from Date and handle conditions
    df['Year'] = df['Date'].dt.year.fillna(0).astype(int).astype(str)

    # Replace year values not matching 2022, 2023, or 2024 with 'error'
    df['Year'] = df['Year'].apply(lambda x: x if x in ['2022', '2023', '2024'] e

    # Count number of rows with 'error' in 'Year'
    error_count = (df['Year'] == 'error').sum()

    # Delete rows with 'Year' == 'error' if error_count > 0
    if error_count > 0:
        df = df[df['Year'] != 'error']

    return df, error_count

# Applying the function to the dataframe
data, error_count = divide_date_column(df3)
data['Year'] = data['Year'].astype('object')
data.drop(columns=['Date'], inplace=True, axis=1)

print('Error count: ', error_count)
```

```
Error count:  0
```

In [ ]:

```
# Assuming 'data' is your DataFrame

# Separating features for preprocessing: Only identify categorical features sinc
categorical_features = data.select_dtypes(include=['object']).columns.tolist()

# If there's no preprocessing needed for numerical features, we can skip definin
# Defining the ColumnTransformer to apply preprocessing to only categorical data
preprocessor = ColumnTransformer(
    transformers=[
        # Only encode categorical features
        ('cat', OneHotEncoder(), categorical_features)],
    remainder='passthrough')  # 'remainder=passthrough' ensures that the rest of

# Creating the pipeline with preprocessing and the KMeans algorithm
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('cluster', KMeans(n_clusters=2))  # Adjust n_clusters as needed
])

# Fitting the pipeline to the data
pipeline.fit(data)

# Accessing the cluster labels assigned to each record
cluster_labels = pipeline.named_steps['cluster'].labels_
print(cluster_labels)
```

```
[0 0 0 ... 1 1 1]
```

In [ ]:

```python
# Extract the transformed dataset from the pipeline
transformed_data = pipeline.named_steps['preprocessor'].transform(data)

pca = PCA(n_components=2)
reduced_data = pca.fit_transform(transformed_data)

# Convert cluster labels to string to treat them as categorical data for colorin
cluster_labels_str = cluster_labels.astype(str)

# Create the Plotly scatter plot of the reduced data points, colored by their cl
fig = px.scatter(
    x=reduced_data[:, 0],
    y=reduced_data[:, 1],
    color=cluster_labels_str,
    color_continuous_scale='Viridis',
    labels={'color': 'Cluster Label'},
    title='Clusters after PCA Reduction'
)

fig.update_traces(marker=dict(size=12, line=dict(width=1, color='DarkSlateGrey')
fig.update_layout(xaxis_title='PCA Feature 1', yaxis_title='PCA Feature 2')
fig.show()
```

In [ ]:

```python
# Let's assume that after your analysis, you determine that:
# Cluster 0 corresponds to 'Not Satisfied'
# Cluster 1 corresponds to 'Satisfied'

# Accessing the cluster labels from your pipeline
cluster_labels = pipeline.named_steps['cluster'].labels_

# Mapping cluster labels to satisfaction scores
satisfaction_mapping = {0: 'Not Satisfied', 1: 'Satisfied'}
data['satisfaction_score'] = [satisfaction_mapping[label] for label in cluster_l

# Now 'data' has a new column 'satisfaction_score' with the satisfaction label
```

In [ ]:

```python
data.to_csv('data/cleanednonull.csv', index=False)
```

In [ ]:

```python
recategorization_mapping = {
    'Satisfied': 1,
    'Not Satisfied': 0
}

data.loc[:, 'satisfaction_score'] = data['satisfaction_score'].replace(recategor

# After replacement, you might want to ensure the data type is what you expect
# For example, if you want to ensure it's an integer (especially if NaN values a
data['satisfaction_score'] = data['satisfaction_score'].astype(int)

# Verify the changes
print(data['satisfaction_score'].value_counts())
```

```
satisfaction_score
1    25809
0    14053
Name: count, dtype: int64
```

```
C:\Users\mogam\AppData\Local\Temp\ipykernel_20704\1733675019.py:6: FutureWarning:

Downcasting behavior in `replace` is deprecated and will be removed in a future v
ersion. To retain the old behavior, explicitly call `result.infer_objects(copy=Fa
lse)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_dow
ncasting', True)`
```

In [ ]:

```
# Assuming subset_df is your DataFrame and 'ServiceSatisfaction' is the column o

class_1_df = data[data['satisfaction_score'] == 1]
class_0_df = data[data['satisfaction_score'] == 0]

# Get the target number of instances to match, which is the number of instances
target_number = class_0_df.shape[0]

# Randomly sample from classes 3 and 2 to match the number of instances in class
class_1_sampled_df = class_1_df.sample(n=target_number, random_state=42)

balanced_df = pd.concat([class_1_sampled_df, class_0_df])

balanced_df['satisfaction_score'].value_counts()
```

Out[ ]:

```
satisfaction_score
1    14053
0    14053
Name: count, dtype: int64
```

In [ ]:

```
ordinal_vars = balanced_df['satisfaction_score']
nominal_vars = [col for col in balanced_df.columns if balanced_df[col].dtype ==
encoded_data = pd.get_dummies(balanced_df, columns=nominal_vars)

# This automatically drops the original nominal columns and adds the one-hot enc
print("NaN counts after pandas get_dummies:", encoded_data.isnull().sum().sum())
```

```
NaN counts after pandas get_dummies: 0
```

In [ ]:

```
X = encoded_data.drop('satisfaction_score', axis=1)
y = encoded_data['satisfaction_score']
# Split the data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [ ]:

```
def test_models(X_train, y_train, X_test, y_test):
    # Models dictionary, assuming CatBoost and LightGBM handle categorical varia
    models = {
        'CatBoostClassifier': CatBoostClassifier(verbose=0),
        'LGBMClassifier': LGBMClassifier(),
        'XGBClassifier': XGBClassifier(use_label_encoder=False, eval_metric='log
    }

    best_model = None
    best_score = -1
    model_results = []
    for name, model in models.items():
        # For CatBoost, specify categorical features
        if name == 'CatBoostClassifier':
            model.set_params(cat_features=[col for col in X_train.columns if str

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
```

```
        roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]) if ha
        report = classification_report(y_test, y_pred, output_dict=True)

        model_result = {
            'Model': name,
            'ROC AUC': roc_auc,
            'Accuracy': report['accuracy'],
            'Precision': report['weighted avg']['precision'],
            'Recall': report['weighted avg']['recall'],
            'F1 Score': report['weighted avg']['f1-score'],
        }
        model_results.append(model_result)

        if roc_auc is not None and roc_auc > best_score:
            best_score = roc_auc
            best_model = model

    return pd.DataFrame(model_results), best_model

# Example usage:
results_df, best_model = test_models(X_train, y_train, X_test, y_test)
print(results_df)
print("Best model:", best_model)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 9824, number of negative: 9850
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.007428 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 200
[LightGBM] [Info] Number of data points in the train set: 19674, number of used f
eatures: 100
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499339 -> initscore=-0.002643
[LightGBM] [Info] Start training from score -0.002643
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
                Model   ROC AUC  Accuracy  Precision    Recall  F1 Score
0  CatBoostClassifier  0.999992  0.999288   0.999289  0.999288  0.999288
1      LGBMClassifier  0.999992  0.999407   0.999407  0.999407  0.999407
2       XGBClassifier  0.999995  0.999288   0.999289  0.999288  0.999288
Best model: XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=True, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=None, ...)
```

In [ ]:

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold

# Assuming 'model' is already defined (e.g., model = RandomForestClassifier())
# X is the feature set and y is the target for the entire dataset (not just the

# Define K-Fold cross-validation
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize an empty list to hold the ROC AUC scores
roc_auc_scores = []

# Perform K-Fold cross-validation
for train_index, test_index in kf.split(X, y):
    X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

    # Train the model on the training fold
    best_model.fit(X_train_fold, y_train_fold)

    # Make predictions on the test fold
    predictions_proba = best_model.predict_proba(X_test_fold)[:, 1]
```

```
    # Calculate the ROC AUC score and append to the list
    roc_auc = roc_auc_score(y_test_fold, predictions_proba)
    roc_auc_scores.append(roc_auc)

# Calculate average and standard deviation of ROC AUC scores across all folds
average_roc_auc = sum(roc_auc_scores) / len(roc_auc_scores)
std_dev_roc_auc = (sum((x - average_roc_auc) ** 2 for x in roc_auc_scores) / len

print(f"Average ROC AUC: {average_roc_auc:.4f}")
print(f"Standard Deviation of ROC AUC: {std_dev_roc_auc:.4f}")
```

```
Average ROC AUC: 1.0000
Standard Deviation of ROC AUC: 0.0000
```

In [ ]:

```
feature_importances = best_model.feature_importances_

# Create a Series for the feature importances
importances = pd.Series(feature_importances, index=X_train.columns)

# Sort the importances and select the top 10, then reverse the Series for plotti
top_10_importances = importances.sort_values(ascending=False)[:10][::-1]

# Create a bar chart using Plotly
fig = px.bar(top_10_importances, x=top_10_importances.values, y=top_10_importanc
             labels={'x': 'Importance', 'index': 'Feature'},
             title='Top 15 Feature Importances (Highest to Lowest)')

# Show the plot
fig.show()
```

In [ ]:

```
# Predict probabilities for the positive class
y_pred_probs = best_model.predict_proba(X_test)[:, 1]

# Calculate residuals (difference between true binary labels and predicted proba
residuals = y_test - y_pred_probs

# Assuming you have the true labels y_test and the predicted probabilities y_pre
# residuals = y_test - y_pred_probs  # Uncomment this line if you have y_test an

# Create the Plotly histogram of the residuals
fig = px.histogram(x=residuals, nbins=20, title='Residual Distribution')
fig.update_layout(xaxis_title='Residuals', yaxis_title='Frequency')
# Show the plot in your environment
fig.show()
```