project

August 15, 2023

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from collections import Counter
```

[ ]:
```
# Load the Marvel dataset
marvel_url = "https://raw.githubusercontent.com/fivethirtyeight/data/master/
 ↪comic-characters/marvel-wikia-data.csv"
marvel_data = pd.read_csv(marvel_url)

# Load the DC dataset
dc_url = "https://raw.githubusercontent.com/fivethirtyeight/data/master/
 ↪comic-characters/dc-wikia-data.csv"
dc_data = pd.read_csv(dc_url)
```

[ ]:
```
from typing import Any
def convert_first_appearance(row: Any) -> float:
    """
    Converts the first appearance column to a float representing the year and␣
 ↪month.

    Params:
        - row (Any): The row to convert

    Returns:
        - float: The year and month as a float
    """
    if pd.isna(row):  # Handle NaN cases
        return row
    if type(row) == float: # Row has already been converted
        return row
    row = str(row)
```

```
    try:
        month_map = {
                        'January': 1, 'February': 2, 'March': 3, 'April': 4,
                        'May': 5, 'June': 6, 'July': 7, 'August': 8,
                        'September': 9, 'October': 10, 'November': 11,␣
 ↪'December': 12,
                        'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
                        'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
                        'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12,
                        'Holiday': 12.5
                    }
        if "-" in row:  # Marvel date format (e.g., 'Sep-75')
            parts = row.split("-")
            year = int("19" + parts[1] if int(parts[1]) > 30 else "20" +␣
 ↪parts[1])
            month_abbrev = parts[0]
            month = month_map.get(month_abbrev)
        else:  # DC date format (e.g., '1999, April')
            parts = row.split(", ")
            if len(parts) == 1:
                year = int(parts[0])
                month = 1
            else:
                year = int(parts[0])
                month_abbrev = parts[1]
                month = month_map.get(month_abbrev)

        return year + (month - 1) / 12  # Normalize month to a fraction of a␣
 ↪year
    except:
        print("Error parsing date: " + row)
        raise
```

[ ]:
```
# Convert the "first appearance" column to a float
marvel_data["FIRST APPEARANCE"] = marvel_data["FIRST APPEARANCE"].
 ↪apply(convert_first_appearance)
dc_data["FIRST APPEARANCE"] = dc_data["FIRST APPEARANCE"].
 ↪apply(convert_first_appearance)
```

[ ]:
```
# Rename the year column to YEAR
marvel_data.rename(columns={"Year": "YEAR"}, inplace=True)
```

[ ]:
```
# Combine datasets
combined_data = pd.concat([marvel_data, dc_data])
```

[ ]:
```
# Save the names of the characters for later
marvel_names = marvel_data["name"]
dc_names = dc_data["name"]
combined_names = combined_data["name"]
```

[ ]:
```
# Drop columns that are not useful for classification (year is redundant with␣
 ↪first appearance)
marvel_data = marvel_data.drop(["urlslug", "page_id", "name", "YEAR"], axis=1)
dc_data = dc_data.drop(["urlslug", "page_id", "name", "YEAR"], axis=1)
combined_data = combined_data.drop(["urlslug", "page_id", "name", "YEAR"],␣
 ↪axis=1)
```

[ ]:
```
# Count the number of each unique type in each column for Marvel
marvel_dfs = []
for column in marvel_data.columns:
    marvel_dfs.append((column, pd.DataFrame(marvel_data[column].
 ↪value_counts(dropna=False), columns=[f"count"], index=marvel_data[column].
 ↪unique())))
for column, df in marvel_dfs:
    print(f"Column: {column}", df, sep="\n", end="\n\n")
```

```
Column: ID
                              count
Secret Identity                6275
Public Identity                4528
No Dual Identity               1788
Known to Authorities Identity    15
NaN                            3770

Column: ALIGN
                     count
Good Characters       4636
Neutral Characters    2208
Bad Characters        6720
NaN                   2812

Column: EYE
                  count
Hazel Eyes           76
Blue Eyes          1962
Brown Eyes         1924
Green Eyes          613
Grey Eyes            95
Yellow Eyes         256
Gold Eyes            14
Red Eyes            508
Black Eyeballs        3
```

Amber Eyes	10
Variable Eyes	49
NaN	9767
Black Eyes	555
White Eyes	400
Orange Eyes	25
Silver Eyes	12
Purple Eyes	31
Pink Eyes	21
One Eye	21
Violet Eyes	11
Multiple Eyes	7
Magenta Eyes	2
Yellow Eyeballs	6
No Eyes	7
Compound Eyes	1

Column: HAIR

	count
Brown Hair	2339
White Hair	754
Black Hair	3755
Blond Hair	1582
No Hair	1176
Blue Hair	56
Red Hair	620
Bald	838
Auburn Hair	78
Grey Hair	531
Silver Hair	16
Purple Hair	47
Strawberry Blond Hair	47
Green Hair	117
Reddish Blond Hair	6
Gold Hair	8
NaN	4264
Orange Hair	43
Pink Hair	31
Variable Hair	32
Yellow Hair	20
Light Brown Hair	6
Magenta Hair	5
Bronze Hair	1
Dyed Hair	1
Orange-brown Hair	3

Column: SEX

	count

```
Male Characters         11638
Female Characters        3837
Genderfluid Characters      2
Agender Characters         45
NaN                       854

Column: GSM
                        count
NaN                     16286
Bisexual Characters        19
Transvestites               1
Homosexual Characters      66
Pansexual Characters        1
Transgender Characters      2
Genderfluid Characters      1

Column: ALIVE
                     count
Living Characters    12608
Deceased Characters   3765
NaN                      3

Column: APPEARANCES
        count
4043.0      1
3360.0      1
3061.0      1
2961.0      1
2258.0      1
…       …
4.0      1097
3.0      1419
2.0      2074
1.0      4810
NaN      1096

[359 rows x 1 columns]

Column: FIRST APPEARANCE
             count
1962.583333     19
1941.166667     53
1974.750000     18
1963.166667     16
1950.833333     12
…             …
1957.250000      2
1957.083333      1
```

```
1959.666667      1
1962.416667      4
1956.166667      1

[833 rows x 1 columns]
```

[ ]:
```python
# Count the number of each unique type in each column for DC
dc_dfs = []
for column in dc_data.columns:
    dc_dfs.append((column, pd.DataFrame(dc_data[column].
  ↪value_counts(dropna=False), columns=[f"count"], index=dc_data[column].
  ↪unique())))
for column, df in dc_dfs:
    print(f"Column: {column}", df, sep="\n", end="\n\n")
```

```
Column: ID
                  count
Secret Identity    2408
Public Identity    2466
NaN                2013
Identity Unknown      9

Column: ALIGN
                     count
Good Characters       2832
Bad Characters        2895
Neutral Characters     565
NaN                    601
Reformed Criminals       3

Column: EYE
                     count
Blue Eyes             1102
Brown Eyes             879
Green Eyes             291
Purple Eyes             14
Black Eyes             412
White Eyes             116
Red Eyes               208
Photocellular Eyes      48
Hazel Eyes              23
Amber Eyes               5
Yellow Eyes             86
NaN                   3628
Grey Eyes               40
Pink Eyes                6
Violet Eyes             12
```

```
Gold Eyes               9
Orange Eyes            10
Auburn Hair             7

Column: HAIR
                       count
Black Hair              1574
Brown Hair              1148
White Hair               346
Blond Hair               744
Red Hair                 461
NaN                     2274
Green Hair                42
Strawberry Blond Hair     28
Grey Hair                157
Silver Hair                3
Orange Hair               21
Purple Hair               32
Gold Hair                  5
Blue Hair                 41
Reddish Brown Hair         3
Pink Hair                 11
Violet Hair                4
Platinum Blond Hair        2

Column: SEX
                        count
Male Characters          4783
Female Characters        1967
NaN                       125
Genderless Characters      20
Transgender Characters      1

Column: GSM
                       count
NaN                     6832
Bisexual Characters       10
Homosexual Characters     54

Column: ALIVE
                     count
Living Characters     5200
Deceased Characters   1693
NaN                      3

Column: APPEARANCES
        count
3093.0      1
```

```
2496.0      1
1565.0      1
1316.0      1
1237.0      1
…         …
4.0       499
3.0       506
2.0       709
1.0      1002
NaN       355

[283 rows x 1 columns]

Column: FIRST APPEARANCE
             count
1939.333333      1
1986.750000     17
1959.750000      5
1987.083333     29
1940.250000      4
…              …
1946.583333      1
2012.333333      1
1975.166667      1
1974.083333      1
1946.250000      1

[755 rows x 1 columns]
```

[ ]:
```python
# Count the number of each unique type in each column for the combined dataset
combined_dfs = []
for column in combined_data.columns:
    combined_dfs.append((column, pd.DataFrame(combined_data[column].
 ↪value_counts(dropna=False), columns=[f"count"], index=combined_data[column].
 ↪unique())))
for column, df in combined_dfs:
    print(f"Column: {column}", df, sep="\n", end="\n\n")
```

```
Column: ID
                              count
Secret Identity                8683
Public Identity                6994
No Dual Identity               1788
Known to Authorities Identity    15
NaN                            5783
Identity Unknown                  9
```

Column: ALIGN

	count
Good Characters	7468
Neutral Characters	2773
Bad Characters	9615
NaN	3413
Reformed Criminals	3

Column: EYE

	count
Hazel Eyes	99
Blue Eyes	3064
Brown Eyes	2803
Green Eyes	904
Grey Eyes	135
Yellow Eyes	342
Gold Eyes	23
Red Eyes	716
Black Eyeballs	3
Amber Eyes	15
Variable Eyes	49
NaN	13395
Black Eyes	967
White Eyes	516
Orange Eyes	35
Silver Eyes	12
Purple Eyes	45
Pink Eyes	27
One Eye	21
Violet Eyes	23
Multiple Eyes	7
Magenta Eyes	2
Yellow Eyeballs	6
No Eyes	7
Compound Eyes	1
Photocellular Eyes	48
Auburn Hair	7

Column: HAIR

	count
Brown Hair	3487
White Hair	1100
Black Hair	5329
Blond Hair	2326
No Hair	1176
Blue Hair	97
Red Hair	1081
Bald	838

Auburn Hair	78
Grey Hair	688
Silver Hair	19
Purple Hair	79
Strawberry Blond Hair	75
Green Hair	159
Reddish Blond Hair	6
Gold Hair	13
NaN	6538
Orange Hair	64
Pink Hair	42
Variable Hair	32
Yellow Hair	20
Light Brown Hair	6
Magenta Hair	5
Bronze Hair	1
Dyed Hair	1
Orange-brown Hair	3
Reddish Brown Hair	3
Violet Hair	4
Platinum Blond Hair	2

Column: SEX

	count
Male Characters	16421
Female Characters	5804
Genderfluid Characters	2
Agender Characters	45
NaN	979
Genderless Characters	20
Transgender Characters	1

Column: GSM

	count
NaN	23118
Bisexual Characters	29
Transvestites	1
Homosexual Characters	120
Pansexual Characters	1
Transgender Characters	2
Genderfluid Characters	1

Column: ALIVE

	count
Living Characters	17808
Deceased Characters	5458
NaN	6

```
Column: APPEARANCES
        count
4043.0      1
3360.0      1
3061.0      1
2961.0      1
2258.0      1
…       …
167.0       1
162.0       1
159.0       1
154.0       1
123.0       3

[443 rows x 1 columns]

Column: FIRST APPEARANCE
             count
1962.583333     21
1941.166667     53
1974.750000     20
1963.166667     16
1950.833333     12
…             …
1958.166667      4
1936.666667      1
1988.958333      2
1949.916667      2
2013.750000      1

[881 rows x 1 columns]
```

[ ]:
```
# List of target features (exclude "url", "page_id", and "name")
target_features = ["ALIGN", "SEX", "EYE", "HAIR", "GSM", "ALIVE", "ID"]
```

[ ]:
```
# Create lists to hold the models
rf_models_marvel = []
rf_models_dc = []
rf_models_combined = []
```

[ ]:
```
# Iterate through each potential target feature, preprocess the data, and␣
 ↪create a model
for target_feature in target_features:
    print(f"""Target feature: {target_feature}""")

    # Remove rows with NaN values in the target feature
```

```
    data_marvel = marvel_data.dropna(subset=[target_feature])
    data_dc = dc_data.dropna(subset=[target_feature])
    data_combined = combined_data.dropna(subset=[target_feature])

    # Split data into features and targets
    features_marvel = data_marvel.drop(target_feature, axis=1)
    features_dc = data_dc.drop(target_feature, axis=1)
    features_combined = data_combined.drop(target_feature, axis=1)

    targets_marvel = data_marvel[target_feature]
    targets_dc = data_dc[target_feature]
    targets_combined = data_combined[target_feature]

    # Encode target feature, for each dataset (using a different encoder for␣
↪each dataset to preserve the mappings)
    le_marvel = LabelEncoder()
    le_dc = LabelEncoder()
    le_combined = LabelEncoder()
    y_marvel = le_marvel.fit_transform(targets_marvel)
    y_dc = le_dc.fit_transform(targets_dc)
    y_combined = le_combined.fit_transform(targets_combined)

    # One-hot encode categorical features (excluding the target feature)
    columns = [tf for tf in target_features if tf != target_feature]
    onehot_features_marvel = pd.get_dummies(features_marvel, columns=columns)
    onehot_features_dc = pd.get_dummies(features_dc, columns=columns)
    onehot_features_combined = pd.get_dummies(features_combined,␣
↪columns=columns)

    # Impute missing values (NaNs to mean, or most frequent value)
    imputer = SimpleImputer(strategy="mean")
    imputed_onehot_features_marvel = imputer.
↪fit_transform(onehot_features_marvel)
    imputed_onehot_features_dc = imputer.fit_transform(onehot_features_dc)
    imputed_onehot_features_combined = imputer.
↪fit_transform(onehot_features_combined)

    # Final features dataframe
    X_marvel = pd.DataFrame(imputed_onehot_features_marvel,␣
↪columns=onehot_features_marvel.columns)
    X_dc = pd.DataFrame(imputed_onehot_features_dc, columns=onehot_features_dc.
↪columns)
    X_combined = pd.DataFrame(imputed_onehot_features_combined,␣
↪columns=onehot_features_combined.columns)

    # Split data into training and testing sets
```

```
    X_train_marvel, X_test_marvel, y_train_marvel, y_test_marvel =␣
↪train_test_split(X_marvel, y_marvel, test_size=0.2, random_state=42)
    X_train_dc, X_test_dc, y_train_dc, y_test_dc = train_test_split(X_dc, y_dc,␣
↪test_size=0.2, random_state=42)
    X_train_combined, X_test_combined, y_train_combined, y_test_combined =␣
↪train_test_split(X_combined, y_combined, test_size=0.2, random_state=42)

    # Create RandomForest classifiers for each category
    rf_classifier_marvel = RandomForestClassifier(random_state=42)
    rf_classifier_dc = RandomForestClassifier(random_state=42)
    rf_classifier_combined = RandomForestClassifier(random_state=42)

    # Get the unique class labels from the original target data
    marvel_unique_class_labels = np.unique(y_marvel)
    dc_unique_class_labels = np.unique(y_dc)
    combined_unique_class_labels = np.unique(y_combined)

    # Convert the unique class labels to a list
    marvel_class_names = le_marvel.inverse_transform(marvel_unique_class_labels)
    dc_class_names = le_dc.inverse_transform(dc_unique_class_labels)
    combined_class_names = le_combined.
↪inverse_transform(combined_unique_class_labels)

    # Store the trained models in the respective lists
    rf_models_marvel.append((rf_classifier_marvel, X_train_marvel,␣
↪y_train_marvel, X_test_marvel, y_test_marvel, target_feature,␣
↪marvel_class_names))
    rf_models_dc.append((rf_classifier_dc, X_train_dc, y_train_dc, X_test_dc,␣
↪y_test_dc, target_feature, dc_class_names))
    rf_models_combined.append((rf_classifier_combined, X_train_combined,␣
↪y_train_combined, X_test_combined, y_test_combined, target_feature,␣
↪combined_class_names))
```

```
Target feature: ALIGN
Target feature: SEX
Target feature: EYE
Target feature: HAIR
Target feature: GSM
Target feature: ALIVE
Target feature: ID
```

[ ]:
```
# Define the parameter grid for the GridSearchCV
param_grid = {
    'n_estimators': [100, 150],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy'],
```

```
    'max_depth': [None, 10, 20],
}
```

[ ]:
```
for rf_model, X_train, y_train, X_test, y_test, target_feature, _ in␣
↪rf_models_marvel:
    try:
        # Find the least populated class in y_train
        class_counts = Counter(y_train)
        least_populated_class = min(class_counts, key=class_counts.get)

        # Check if the count of the least populated class is greater than or␣
↪equal to 2
        n = class_counts[least_populated_class]
        if n >= 2:
            print(f"Training model for Marvel {target_feature} predictions")
            grid_search_marvel = GridSearchCV(rf_model, param_grid,␣
↪cv=StratifiedKFold(n_splits=min(n, 5), shuffle=True, random_state=42),␣
↪n_jobs=-1)
            grid_search_marvel.fit(X_train, y_train)

            rf_model = grid_search_marvel.best_estimator_
            y_pred = rf_model.predict(X_test)

            report = classification_report(y_test, y_pred, zero_division=np.nan)
            print(f"""Classification report for Marvel {target_feature}␣
↪predictions:\n{report}""")
            print(f"Accuracy score for Marvel {target_feature} predictions:␣
↪{accuracy_score(y_test, y_pred)}")
            print(f"Best parameters for Marvel {target_feature} predictions:␣
↪{grid_search_marvel.best_params_}")
        else:
            print(f"Count of the least populated class for Marvel␣
↪{target_feature} predictions is < 2, training model with default parameters")


            rf_model.fit(X_train, y_train)
            y_pred = rf_model.predict(X_test)
            report = classification_report(y_test, y_pred, zero_division=np.nan)
            print(f"""Classification report for Marvel {target_feature}␣
↪predictions:\n{report}""")
            print(f"Accuracy score for Marvel {target_feature} predictions:␣
↪{accuracy_score(y_test, y_pred)}")
    except Exception as e:
        print(f"""Error training model for Marvel {target_feature} predictions␣
↪with error: {e}""")
```

```
Training model for Marvel ALIGN predictions
Classification report for Marvel ALIGN predictions:
              precision    recall  f1-score   support

           0       0.63      0.81      0.71      1356
           1       0.56      0.54      0.55       923
           2       0.42      0.08      0.13       434

    accuracy                           0.60      2713
   macro avg       0.54      0.48      0.47      2713
weighted avg       0.58      0.60      0.56      2713

Accuracy score for Marvel ALIGN predictions: 0.6030224843346849
Best parameters for Marvel ALIGN predictions: {'criterion': 'entropy',
'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators':
100}
Count of the least populated class for Marvel SEX predictions is < 2, training
model with default parameters
Classification report for Marvel SEX predictions:
              precision    recall  f1-score   support

           0       0.75      0.33      0.46         9
           1       0.45      0.30      0.36       774
           2        nan      0.00       nan         1
           3       0.79      0.88      0.83      2321

    accuracy                           0.73      3105
   macro avg       0.66      0.38      0.55      3105
weighted avg       0.70      0.73      0.71      3105

Accuracy score for Marvel SEX predictions: 0.7314009661835749
Training model for Marvel EYE predictions
Classification report for Marvel EYE predictions:
              precision    recall  f1-score   support

           0        nan      0.00       nan         2
           1        nan      0.00       nan         1
           2       0.14      0.01      0.02       122
           3       0.51      0.60      0.55       403
           4       0.47      0.82      0.60       386
           5        nan      0.00       nan         1
           6        nan      0.00       nan         3
           7       0.49      0.22      0.31       121
           8       0.00      0.00       nan        18
           9        nan      0.00       nan        10
          11        nan      0.00       nan         3
          13        nan      0.00       nan         2
          14        nan      0.00       nan         7
```

```
              15        nan      0.00       nan         4
              16        nan      0.00       nan         7
              17       0.32      0.34      0.33        86
              19       1.00      0.43      0.60        14
              20        nan      0.00       nan         3
              21       0.27      0.05      0.09        79
              23       0.00      0.00       nan        50

    accuracy                           0.47      1322
   macro avg       0.36      0.12      0.36      1322
weighted avg       0.41      0.47      0.44      1322

Accuracy score for Marvel EYE predictions: 0.4735249621785174
Best parameters for Marvel EYE predictions: {'criterion': 'gini', 'max_depth':
10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
Count of the least populated class for Marvel HAIR predictions is < 2, training
model with default parameters
Classification report for Marvel HAIR predictions:
              precision    recall  f1-score   support

           0       0.07      0.09      0.08        11
           1       0.13      0.10      0.11       168
           2       0.42      0.49      0.45       773
           3       0.24      0.24      0.24       323
           4       0.00      0.00       nan         7
           6       0.24      0.23      0.24       444
           8        nan      0.00       nan         3
           9       0.05      0.05      0.05        21
          10       0.07      0.05      0.06        99
          11       0.00      0.00       nan         2
          12       0.00      0.00       nan         2
          13       0.49      0.52      0.51       233
          14       0.00      0.00       nan        10
          15        nan      0.00       nan         1
          16       0.00      0.00       nan         9
          17       0.00      0.00       nan        12
          18       0.13      0.10      0.11       136
          19       0.00      0.00       nan         1
          20       0.00      0.00       nan         1
          21       0.00      0.00       nan         6
          22       0.67      0.44      0.53         9
          23       0.10      0.08      0.09       149
          24       0.00      0.00       nan         3

    accuracy                           0.30      2423
   macro avg       0.12      0.10      0.22      2423
weighted avg       0.28      0.30      0.30      2423
```

```
Accuracy score for Marvel HAIR predictions: 0.3033429632686752
Count of the least populated class for Marvel GSM predictions is < 2, training
model with default parameters
Classification report for Marvel GSM predictions:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           2       0.82      1.00      0.90        14
           5        nan      0.00       nan         1

    accuracy                           0.83        18
   macro avg       0.91      0.44      0.70        18
weighted avg       0.85      0.83      0.83        18

Accuracy score for Marvel GSM predictions: 0.8333333333333334
Training model for Marvel ALIVE predictions
Classification report for Marvel ALIVE predictions:
              precision    recall  f1-score   support

           0       0.46      0.05      0.08       754
           1       0.78      0.98      0.87      2521

    accuracy                           0.77      3275
   macro avg       0.62      0.52      0.48      3275
weighted avg       0.70      0.77      0.69      3275

Accuracy score for Marvel ALIVE predictions: 0.76793893129771
Best parameters for Marvel ALIVE predictions: {'criterion': 'gini', 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 150}
Training model for Marvel ID predictions
Classification report for Marvel ID predictions:
              precision    recall  f1-score   support

           0        nan      0.00       nan         3
           1       0.45      0.03      0.05       323
           2       0.51      0.47      0.49       908
           3       0.61      0.79      0.69      1288

    accuracy                           0.58      2522
   macro avg       0.52      0.32      0.41      2522
weighted avg       0.56      0.58      0.54      2522

Accuracy score for Marvel ID predictions: 0.578112609040444
Best parameters for Marvel ID predictions: {'criterion': 'entropy', 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
```

[ ]:
```
for rf_model, X_train, y_train, X_test, y_test, target_feature, _ in␣
↪rf_models_dc:
    try:
        # Find the least populated class in y_train
        class_counts = Counter(y_train)
        least_populated_class = min(class_counts, key=class_counts.get)

        # Check if the count of the least populated class is greater than or␣
↪equal to 2
        n = class_counts[least_populated_class]
        if n >= 2:
            print(f"Training model for DC {target_feature} predictions")
            grid_search_dc = GridSearchCV(rf_model, param_grid,␣
↪cv=StratifiedKFold(n_splits=min(n, 5), shuffle=True, random_state=42),␣
↪n_jobs=-1)
            grid_search_dc.fit(X_train, y_train)

            rf_model = grid_search_dc.best_estimator_
            y_pred = rf_model.predict(X_test)
            report = classification_report(y_test, y_pred, zero_division=np.nan)
            print(f"""Classification report for DC {target_feature} predictions:
↪\n{report}""")
            print(f"Accuracy score for DC {target_feature} predictions:␣
↪{accuracy_score(y_test, y_pred)}")
            print(f"Best parameters for DC {target_feature} predictions:␣
↪{grid_search_dc.best_params_}")
        else:
            print(f"Count of the least populated class for DC {target_feature}␣
↪predictions is < 2, training model with default parameters")

            rf_model.fit(X_train, y_train)
            y_pred = rf_model.predict(X_test)
            report = classification_report(y_test, y_pred, zero_division=np.nan)
            print(f"""Classification report for DC {target_feature} predictions:
↪\n{report}""")
            print(f"Accuracy score for DC {target_feature} predictions:␣
↪{accuracy_score(y_test, y_pred)}")
    except:
        print(f"""Error training model for DC {target_feature} predictions""")
```

```
Training model for DC ALIGN predictions
Classification report for DC ALIGN predictions:
              precision    recall  f1-score   support

           0       0.62      0.72      0.67       571
           1       0.63      0.65      0.64       571
           2        nan      0.00       nan       117
```

```
    accuracy                           0.62      1259
   macro avg       0.62      0.46      0.65      1259
weighted avg       0.62      0.62      0.65      1259

Accuracy score for DC ALIGN predictions: 0.6243050039714059
Best parameters for DC ALIGN predictions: {'criterion': 'gini', 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
Count of the least populated class for DC SEX predictions is < 2, training model
with default parameters
Classification report for DC SEX predictions:
              precision    recall  f1-score   support

           0       0.43      0.35      0.38       370
           1        nan      0.00       nan         5
           2       0.77      0.83      0.80       980

    accuracy                           0.69      1355
   macro avg       0.60      0.39      0.59      1355
weighted avg       0.68      0.69      0.68      1355

Accuracy score for DC SEX predictions: 0.6937269372693727
Training model for DC EYE predictions
Classification report for DC EYE predictions:
              precision    recall  f1-score   support

           0        nan      0.00       nan         1
           2       0.25      0.13      0.17        77
           3       0.43      0.67      0.53       217
           4       0.52      0.72      0.60       179
           5        nan      0.00       nan         3
           6       0.61      0.18      0.28        61
           7        nan      0.00       nan        11
           8        nan      0.00       nan         4
           9        nan      0.00       nan         3
          10       0.67      0.18      0.29        11
          12        nan      0.00       nan         1
          13       0.50      0.06      0.11        47
          14        nan      0.00       nan         3
          15        nan      0.00       nan        26
          16        nan      0.00       nan        10

    accuracy                           0.46       654
   macro avg       0.50      0.13      0.33       654
weighted avg       0.46      0.46      0.44       654

Accuracy score for DC EYE predictions: 0.4602446483180428
Best parameters for DC EYE predictions: {'criterion': 'entropy', 'max_depth':
```

```
20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 50}
Count of the least populated class for DC HAIR predictions is < 2, training
model with default parameters
Classification report for DC HAIR predictions:
              precision    recall  f1-score   support

           0       0.37      0.48      0.42       308
           1       0.24      0.25      0.25       147
           2       0.00      0.00       nan        15
           3       0.32      0.30      0.31       235
           4       0.00      0.00       nan         2
           5       0.25      0.10      0.14        10
           6       0.07      0.07      0.07        30
           7       0.25      0.33      0.29         3
           8       0.00      0.00       nan         4
          10       0.00      0.00       nan         8
          11       0.17      0.13      0.15        75
          12        nan      0.00       nan         1
          13        nan      0.00       nan         2
          14       0.00      0.00       nan         5
          16       0.14      0.06      0.09        80

    accuracy                           0.30       925
   macro avg       0.14      0.12      0.21       925
weighted avg       0.27      0.30      0.29       925

Accuracy score for DC HAIR predictions: 0.2972972972972973
Training model for DC GSM predictions
Classification report for DC GSM predictions:
              precision    recall  f1-score   support

           0        nan      0.00       nan         2
           1       0.85      1.00      0.92        11

    accuracy                           0.85        13
   macro avg       0.85      0.50      0.92        13
weighted avg       0.85      0.85      0.92        13

Accuracy score for DC GSM predictions: 0.8461538461538461
Best parameters for DC GSM predictions: {'criterion': 'gini', 'max_depth': None,
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Training model for DC ALIVE predictions
Classification report for DC ALIVE predictions:
              precision    recall  f1-score   support

           0       1.00      0.01      0.02       345
           1       0.75      1.00      0.86      1034
```

```
    accuracy                           0.75      1379
   macro avg       0.88      0.50      0.44      1379
weighted avg       0.81      0.75      0.65      1379

Accuracy score for DC ALIVE predictions: 0.7519941986947063
Best parameters for DC ALIVE predictions: {'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 50}
Training model for DC ID predictions
Classification report for DC ID predictions:
              precision    recall  f1-score   support

           1       0.65      0.68      0.67       493
           2       0.66      0.62      0.64       484

    accuracy                           0.65       977
   macro avg       0.65      0.65      0.65       977
weighted avg       0.65      0.65      0.65       977

Accuracy score for DC ID predictions: 0.6530194472876152
Best parameters for DC ID predictions: {'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
```

[ ]:
```
for rf_model, X_train, y_train, X_test, y_test, target_feature, _ in␣
↪rf_models_combined:
    try:
        # Find the least populated class in y_train
        class_counts = Counter(y_train)
        least_populated_class = min(class_counts, key=class_counts.get)

        # Check if the count of the least populated class is greater than or␣
↪equal to 2
        n = class_counts[least_populated_class]
        if n >= 2:
            print(f"Training model for Combined {target_feature} predictions")
            grid_search_combined = GridSearchCV(rf_model, param_grid,␣
↪cv=StratifiedKFold(n_splits=min(n, 5), shuffle=True, random_state=42),␣
↪n_jobs=-1)
            grid_search_combined.fit(X_train, y_train)

            rf_model = grid_search_combined.best_estimator_
            y_pred = rf_model.predict(X_test)
            report = classification_report(y_test, y_pred, zero_division=np.nan)
            print(f"""Classification report for Marvel {target_feature}␣
↪predictions:\n{report}""")
            print(f"Accuracy score for Combined {target_feature} predictions:␣
↪{accuracy_score(y_test, y_pred)}")
```

```
            print(f"Best parameters for Combined {target_feature} predictions:␣
↪{grid_search_combined.best_params_}")
        else:
            print(f"Count of the least populated class for DC {target_feature}␣
↪predictions is < 2, training model with default parameters")

            rf_model.fit(X_train, y_train)
            y_pred = rf_model.predict(X_test)
            report = classification_report(y_test, y_pred, zero_division=np.nan)
            print(f"""Classification report for Marvel {target_feature}␣
↪predictions:\n{report}""")
            print(f"Accuracy score for Combined {target_feature} predictions:␣
↪{accuracy_score(y_test, y_pred)}")
    except:
        print(f"""Error training model for Combined {target_feature}␣
↪predictions""")
```

```
Training model for Combined ALIGN predictions
Classification report for Marvel ALIGN predictions:
              precision    recall  f1-score   support

           0       0.61      0.82      0.70      1924
           1       0.60      0.55      0.57      1504
           2       0.27      0.01      0.01       543
           3        nan      0.00       nan         1

    accuracy                           0.61      3972
   macro avg       0.49      0.34      0.43      3972
weighted avg       0.56      0.61      0.56      3972

Accuracy score for Combined ALIGN predictions: 0.6062437059415912
Best parameters for Combined ALIGN predictions: {'criterion': 'gini',
'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators':
50}
Training model for Combined SEX predictions
Classification report for Marvel SEX predictions:
              precision    recall  f1-score   support

           0        nan      0.00       nan        10
           1       0.63      0.14      0.23      1178
           3        nan      0.00       nan         5
           4       0.76      0.97      0.85      3265
           5        nan      0.00       nan         1

    accuracy                           0.75      4459
   macro avg       0.69      0.22      0.54      4459
```

```
weighted avg       0.72      0.75      0.69      4459

Accuracy score for Combined SEX predictions: 0.7481498093742992
Best parameters for Combined SEX predictions: {'criterion': 'gini', 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}
Count of the least populated class for DC EYE predictions is < 2, training model
with default parameters
Classification report for Marvel EYE predictions:
              precision    recall  f1-score   support

           0       0.00      0.00       nan         1
           1        nan      0.00       nan         2
           3       0.20      0.17      0.18       188
           4       0.46      0.53      0.49       612
           5       0.50      0.56      0.53       568
           7       0.50      0.14      0.22         7
           8       0.28      0.21      0.24       182
           9       0.14      0.09      0.11        22
          10       0.00      0.00       nan        15
          11        nan      0.00       nan         1
          12       0.00      0.00       nan         1
          13       0.00       nan       nan         0
          14       0.00      0.00       nan         6
          15       0.00      0.00       nan         9
          16       0.62      0.31      0.42        16
          17       0.00      0.00       nan         4
          18       0.00      0.00       nan        13
          19       0.24      0.25      0.25       144
          20       0.00      0.00       nan         3
          21       0.86      0.46      0.60        13
          22       0.00      0.00       nan         7
          23       0.28      0.23      0.25        96
          24       0.00       nan       nan         0
          25       0.25      0.17      0.20        66

    accuracy                           0.40      1976
   macro avg       0.20      0.14      0.32      1976
weighted avg       0.38      0.40      0.40      1976

Accuracy score for Combined EYE predictions: 0.40384615384615385
Count of the least populated class for DC HAIR predictions is < 2, training
model with default parameters
Classification report for Marvel HAIR predictions:
              precision    recall  f1-score   support

           0       0.00      0.00       nan        18
           1       0.10      0.10      0.10       157
           2       0.41      0.43      0.42      1143
```

```
              3       0.25      0.26      0.26       447
              4       0.00      0.00       nan        19
              6       0.25      0.28      0.26       659
              8       1.00      0.33      0.50         3
              9       0.14      0.11      0.12        27
             10       0.12      0.08      0.10       136
             11       0.00      0.00       nan         1
             12       0.00       nan       nan         0
             13       0.44      0.46      0.45       225
             14       0.08      0.07      0.08        14
             15        nan      0.00       nan         1
             16       0.00      0.00       nan         9
             17        nan      0.00       nan         1
             18       0.08      0.05      0.06        19
             19       0.15      0.14      0.15       199
             20        nan      0.00       nan         2
             22       0.00      0.00       nan         4
             23       0.11      0.06      0.08        16
             24       0.60      0.67      0.63         9
             25        nan      0.00       nan         2
             26       0.11      0.08      0.09       233
             27       0.00      0.00       nan         3

    accuracy                           0.30      3347
   macro avg       0.18      0.13      0.24      3347
weighted avg       0.28      0.30      0.29      3347

Accuracy score for Combined HAIR predictions: 0.2954884971616373
Count of the least populated class for DC GSM predictions is < 2, training model
with default parameters
Classification report for Marvel GSM predictions:
              precision    recall  f1-score   support

           0       0.00      0.00       nan         3
           2       0.89      0.86      0.87        28

    accuracy                           0.77        31
   macro avg       0.44      0.43      0.87        31
weighted avg       0.80      0.77      0.87        31

Accuracy score for Combined GSM predictions: 0.7741935483870968
Training model for Combined ALIVE predictions
Classification report for Marvel ALIVE predictions:
              precision    recall  f1-score   support

           0       0.55      0.03      0.05      1072
           1       0.77      0.99      0.87      3582
```

```
    accuracy                           0.77      4654
   macro avg       0.66      0.51      0.46      4654
weighted avg       0.72      0.77      0.68      4654

Accuracy score for Combined ALIVE predictions: 0.7707348517404383
Best parameters for Combined ALIVE predictions: {'criterion': 'entropy',
'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5,
'n_estimators': 100}
Training model for Combined ID predictions
Classification report for Marvel ID predictions:
              precision    recall  f1-score   support

           0       nan      0.00       nan         3
           1       nan      0.00       nan         2
           2      1.00      0.02      0.04       361
           3      0.52      0.52      0.52      1412
           4      0.60      0.73      0.66      1720

    accuracy                           0.57      3498
   macro avg       0.71      0.25      0.41      3498
weighted avg       0.61      0.57      0.54      3498

Accuracy score for Combined ID predictions: 0.5714694110920526
Best parameters for Combined ID predictions: {'criterion': 'gini', 'max_depth':
None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 150}
```

[ ]:
```
import shap

shap.initjs()
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to
force console mode (e.g. in jupyter console)

<IPython.core.display.HTML object>
```

[ ]:
```
best_params_marvel = {
    "ALIGN": {},
    "SEX": {},
    "HAIR": {},
    "ID": {},
    "GSM": {},
    "ALIVE": {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 10, 'n_estimators': 150},
    "EYE": {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 2, 'n_estimators': 100},
    "ALIGN": {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 1,␣
 ↪'min_samples_split': 10, 'n_estimators': 100},
}
```

```
best_params_dc = {
    "ALIGN": {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 2, 'n_estimators': 100},
    "SEX": {},
    "EYE": {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 10, 'n_estimators': 50},
    "HAIR": {},
    "GSM": {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1,␣
 ↪'min_samples_split': 2, 'n_estimators': 50},
    "ALIVE": {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 4,␣
 ↪'min_samples_split': 2, 'n_estimators': 50},
    "ID": {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 1,␣
 ↪'min_samples_split': 10, 'n_estimators': 100}
}

best_params_combined = {
    "ALIGN": {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 5, 'n_estimators': 50},
    "SEX": {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 10, 'n_estimators': 100},
    "EYE": {},
    "HAIR": {},
    "GSM": {},
    "ALIVE": {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 5, 'n_estimators': 100},
    "ID": {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2,␣
 ↪'min_samples_split': 5, 'n_estimators': 150}
}
```

[ ]:
```
shap_models_marvel = []

for rf_model, X_train, y_train, X_test, y_test, target_feature, class_names in␣
 ↪rf_models_marvel:
    rf_model = RandomForestClassifier(**best_params_marvel[target_feature])
    rf_model.fit(X_train, y_train)

    print(f"""Creating SHAP values for Marvel {target_feature} predictions with␣
 ↪{len(X_test)} samples""")
    subset_size = min((int(len(X_test) * 0.1) if len(X_test) > 250 else␣
 ↪len(X_test)), 50)
    print(f"""Subset size: {subset_size}""")

    explainer = shap.TreeExplainer(rf_model)
    shap_values = explainer.shap_values(X_test[:subset_size])
```

```
    shap_models_marvel.append((explainer, shap_values, X_test[:subset_size],␣
 ↪target_feature, class_names))
    print(f"""SHAP values for Marvel {target_feature} predictions created""")

print("Marvel SHAP values created")
```

```
Creating SHAP values for Marvel ALIGN predictions with 2713 samples
Subset size: 50
SHAP values for Marvel ALIGN predictions created
Creating SHAP values for Marvel SEX predictions with 3105 samples
Subset size: 50
SHAP values for Marvel SEX predictions created
Creating SHAP values for Marvel EYE predictions with 1322 samples
Subset size: 50
SHAP values for Marvel EYE predictions created
Creating SHAP values for Marvel HAIR predictions with 2423 samples
Subset size: 50
SHAP values for Marvel HAIR predictions created
Creating SHAP values for Marvel GSM predictions with 18 samples
Subset size: 18
SHAP values for Marvel GSM predictions created
Creating SHAP values for Marvel ALIVE predictions with 3275 samples
Subset size: 50
SHAP values for Marvel ALIVE predictions created
Creating SHAP values for Marvel ID predictions with 2522 samples
Subset size: 50
SHAP values for Marvel ID predictions created
Marvel SHAP values created
```

[ ]:
```
# Create SHAP values for DC

shap_models_dc = []

for rf_model, X_train, y_train, X_test, y_test, target_feature, class_names in␣
 ↪rf_models_dc:
    rf_model = RandomForestClassifier(**best_params_dc[target_feature])
    rf_model.fit(X_train, y_train)

    print(f"""Creating SHAP values for DC {target_feature} predictions with␣
 ↪{len(X_test)} samples""")
    subset_size = min((int(len(X_test) * 0.1) if len(X_test) > 250 else␣
 ↪len(X_test)), 50)
    print(f"""Subset size: {subset_size}""")

    explainer = shap.TreeExplainer(rf_model)
    shap_values = explainer.shap_values(X_test[:subset_size])
```

```
    shap_models_dc.append((explainer, shap_values, X_test[:subset_size],␣
 ↪target_feature, class_names))
    print(f"""SHAP values for DC {target_feature} predictions created""")

print("DC SHAP values created")
```

```
Creating SHAP values for DC ALIGN predictions with 1259 samples
Subset size: 50
SHAP values for DC ALIGN predictions created
Creating SHAP values for DC SEX predictions with 1355 samples
Subset size: 50
SHAP values for DC SEX predictions created
Creating SHAP values for DC EYE predictions with 654 samples
Subset size: 50
SHAP values for DC EYE predictions created
Creating SHAP values for DC HAIR predictions with 925 samples
Subset size: 50
SHAP values for DC HAIR predictions created
Creating SHAP values for DC GSM predictions with 13 samples
Subset size: 13
SHAP values for DC GSM predictions created
Creating SHAP values for DC ALIVE predictions with 1379 samples
Subset size: 50
SHAP values for DC ALIVE predictions created
Creating SHAP values for DC ID predictions with 977 samples
Subset size: 50
SHAP values for DC ID predictions created
DC SHAP values created
```

[ ]:
```
# Create SHAP values for combined

shap_models_combined = []

for rf_model, X_train, y_train, X_test, y_test, target_feature, class_names in␣
 ↪rf_models_combined:
    rf_model = RandomForestClassifier(**best_params_combined[target_feature])
    rf_model.fit(X_train, y_train)

    print(f"""Creating SHAP values for combined {target_feature} predictions␣
 ↪with {len(X_test)} samples""")
    subset_size = min((int(len(X_test) * 0.1) if len(X_test) > 250 else␣
 ↪len(X_test)), 50)
    print(f"""Subset size: {subset_size}""")

    explainer = shap.TreeExplainer(rf_model)
    shap_values = explainer.shap_values(X_test[:subset_size])
```

```
    shap_models_combined.append((explainer, shap_values, X_test[:subset_size],␣
 ↪target_feature, class_names))
    print(f"""SHAP values for combined {target_feature} predictions created""")

print("Combined SHAP values created")
```

```
Creating SHAP values for combined ALIGN predictions with 3972 samples
Subset size: 50
SHAP values for combined ALIGN predictions created
Creating SHAP values for combined SEX predictions with 4459 samples
Subset size: 50
SHAP values for combined SEX predictions created
Creating SHAP values for combined EYE predictions with 1976 samples
Subset size: 50
SHAP values for combined EYE predictions created
Creating SHAP values for combined HAIR predictions with 3347 samples
Subset size: 50
SHAP values for combined HAIR predictions created
Creating SHAP values for combined GSM predictions with 31 samples
Subset size: 31
SHAP values for combined GSM predictions created
Creating SHAP values for combined ALIVE predictions with 4654 samples
Subset size: 50
SHAP values for combined ALIVE predictions created
Creating SHAP values for combined ID predictions with 3498 samples
Subset size: 50
SHAP values for combined ID predictions created
Combined SHAP values created
```

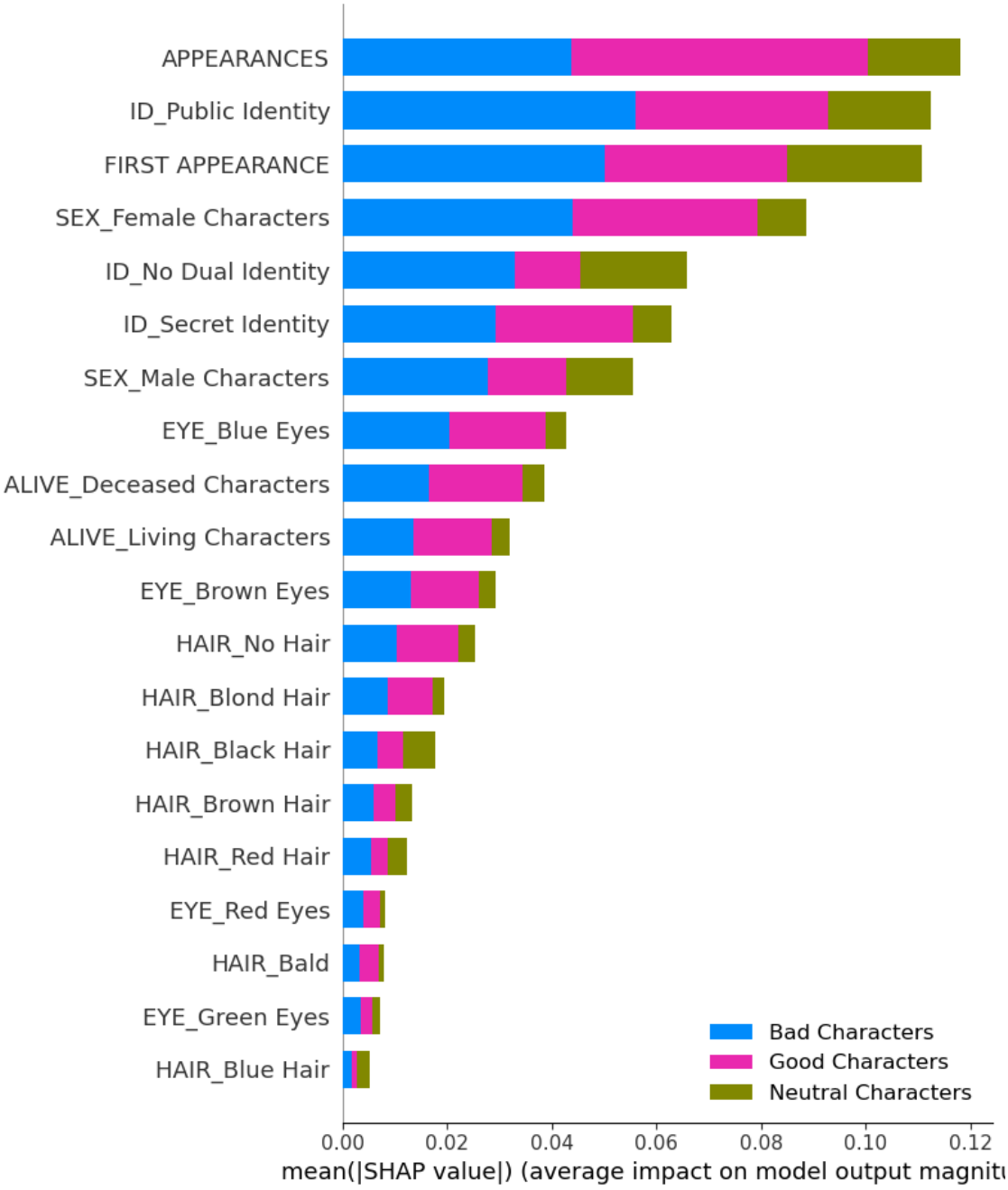
[ ]:
```
import random
import matplotlib.pyplot as plt
```

[ ]:
```
# Create plots for Marvel

for explainer, shap_values, feature_subset, target_feature, class_names in␣
 ↪shap_models_marvel:
    num_samples = len(feature_subset)
    sample_index = random.randint(0, num_samples - 1)  # Generate a random␣
 ↪index within the valid range

    print(f"Creating Summary plot for Marvel {target_feature} predictions")
    shap.summary_plot(shap_values, feature_subset, feature_names=feature_subset.
 ↪columns, show=False, class_names=class_names)
    plt.savefig(f"figs/summary/Marvel_{target_feature}_summary.png")
    plt.tight_layout()
    plt.show()
    print(f"Created Summary plot for Marvel {target_feature} predictions")
```

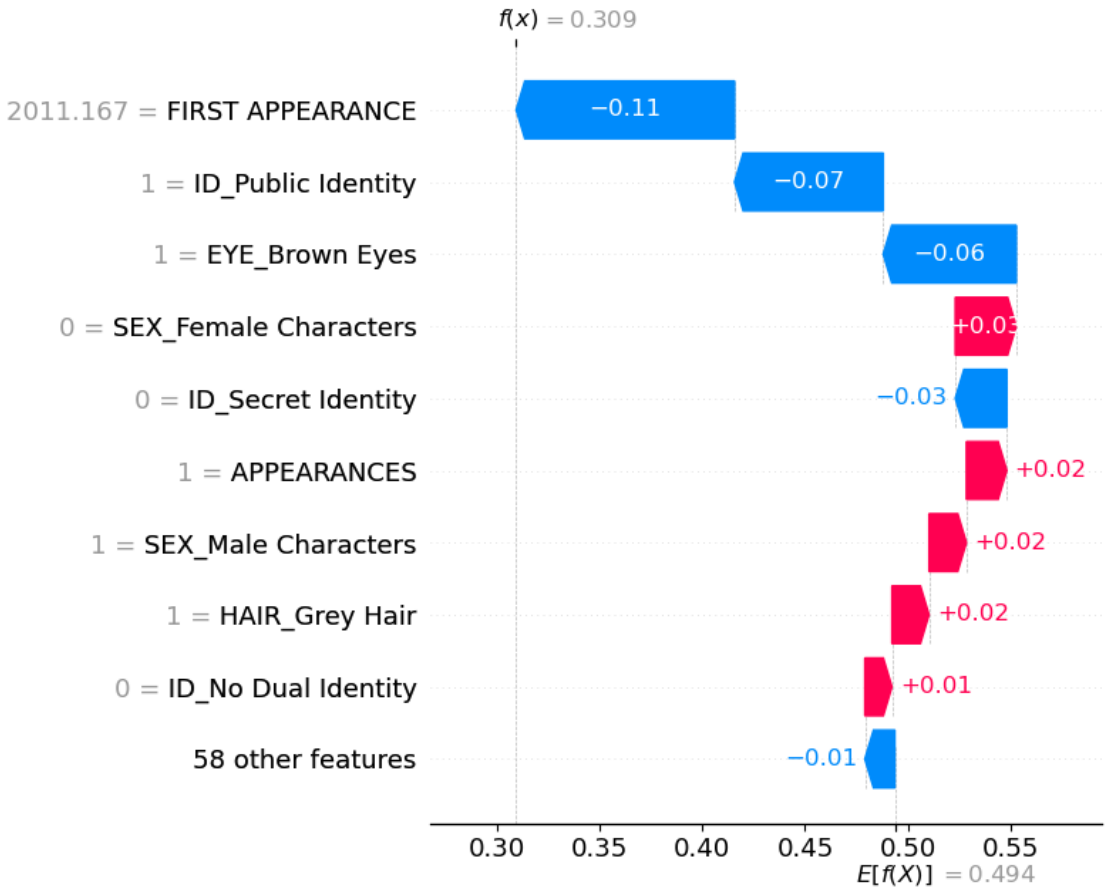
```
    print(f"Creating Waterfall plot for Marvel {target_feature} predictions")
    shap.plots.waterfall(shap.Explanation(values=shap_values[0][sample_index],␣
 ↪base_values=explainer.expected_value[0], data=feature_subset.
 ↪iloc[sample_index]), max_display=10, show=False)
    plt.savefig(f"figs/waterfall/
 ↪Marvel_{target_feature}_{sample_index}_waterfall.png")
    plt.tight_layout()
    plt.show()
    print(f"Created Waterfall plot for Marvel {target_feature} predictions")


print("Marvel plots created")
```

```
Creating Summary plot for Marvel ALIGN predictions
```

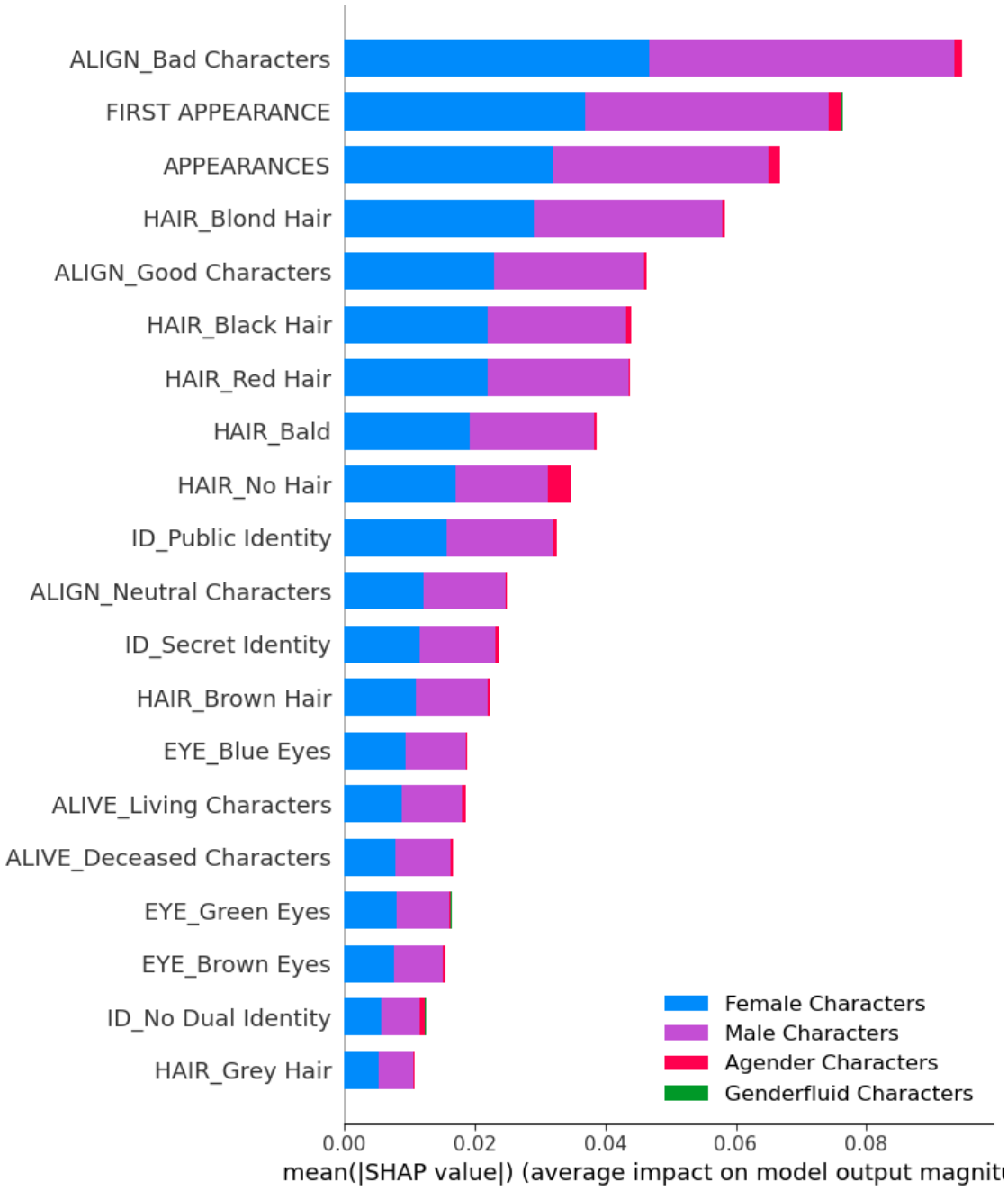
```
The figure layout has changed to tight
```



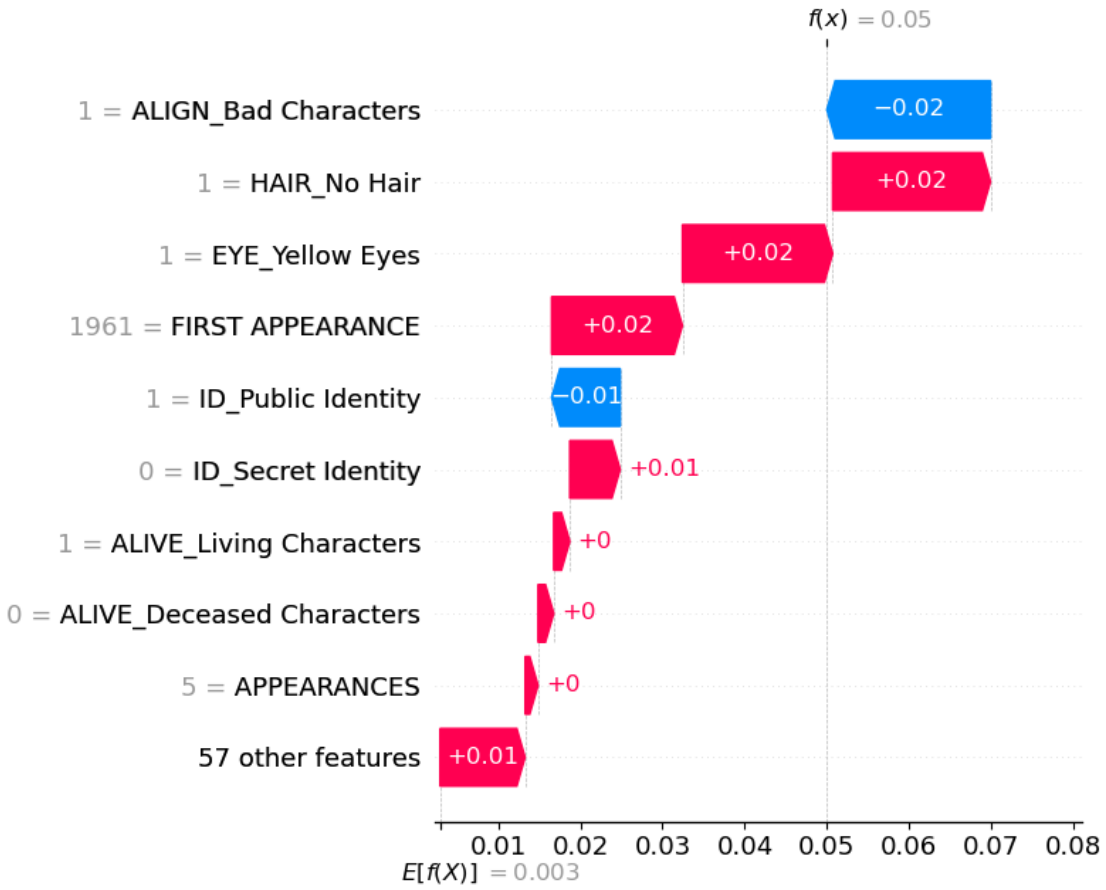

```
Created Summary plot for Marvel ALIGN predictions
Creating Waterfall plot for Marvel ALIGN predictions
```



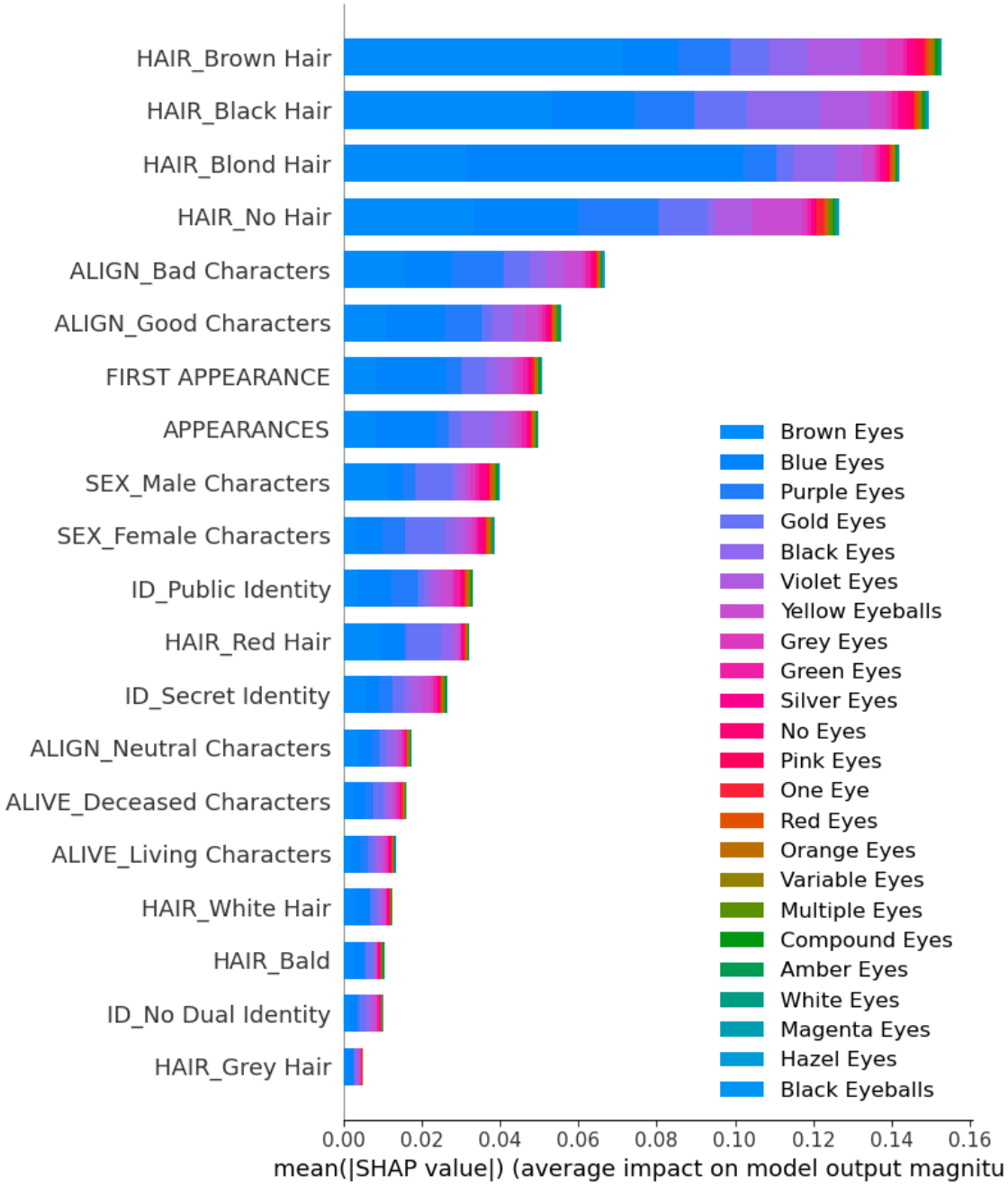

```
Created Waterfall plot for Marvel ALIGN predictions
Creating Summary plot for Marvel SEX predictions

The figure layout has changed to tight
```



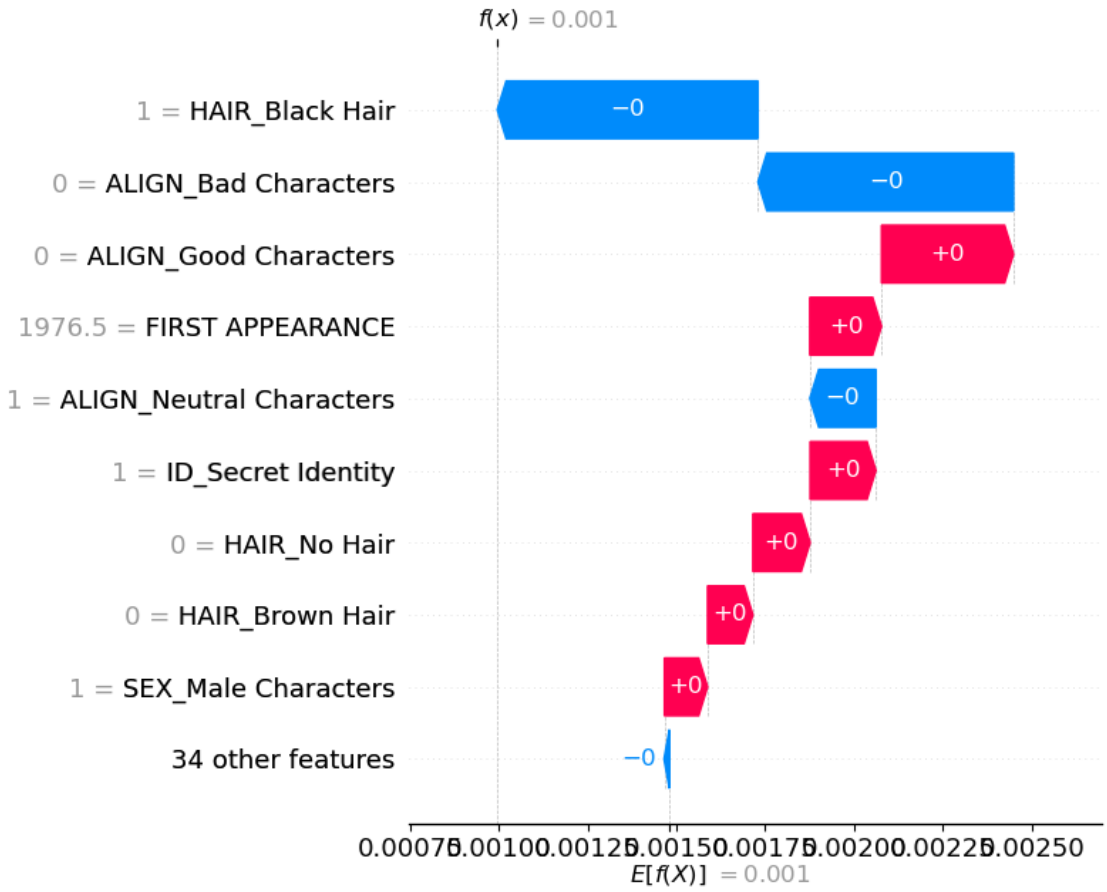

```
Created Summary plot for Marvel SEX predictions
Creating Waterfall plot for Marvel SEX predictions
```



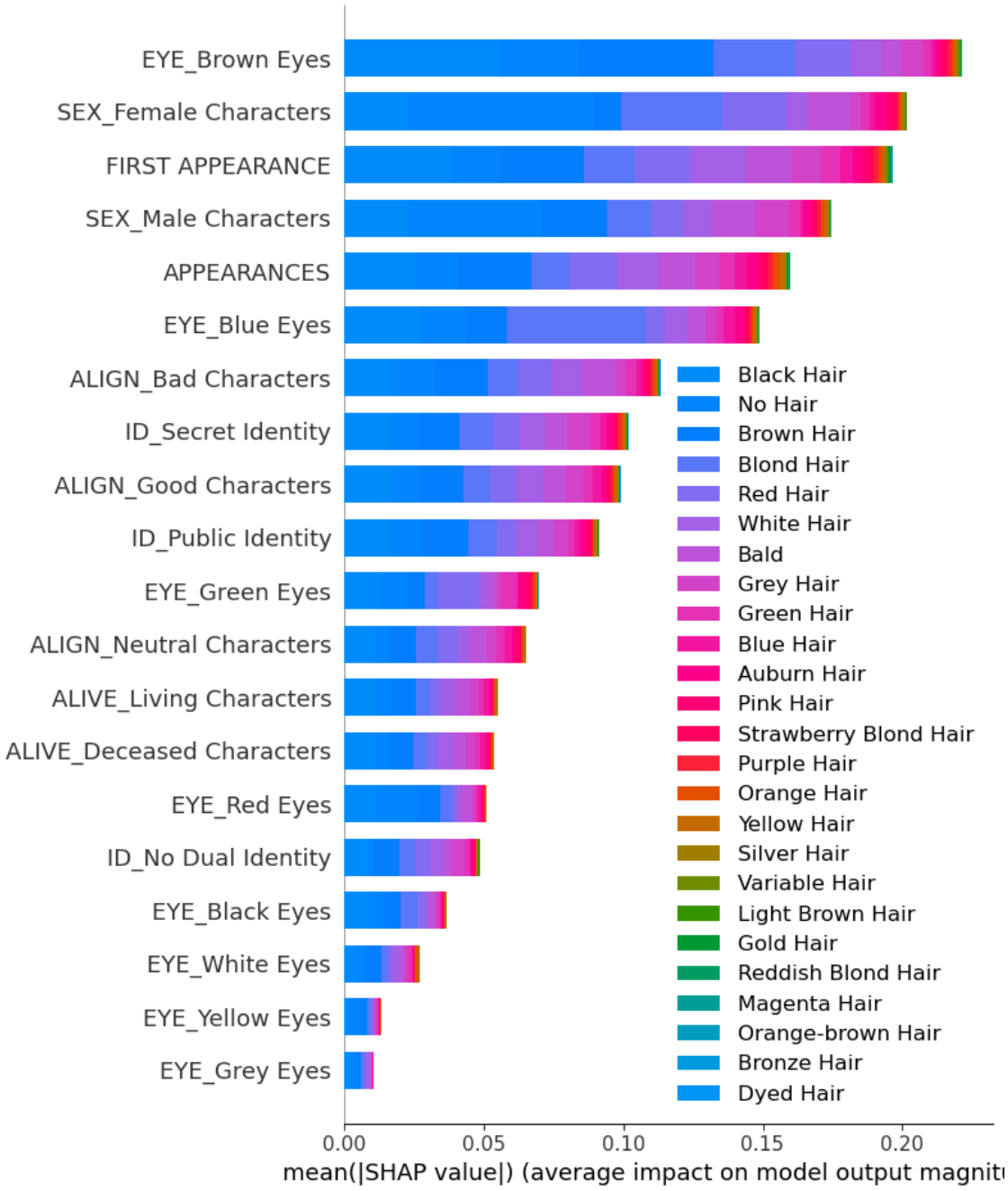

```
Created Waterfall plot for Marvel SEX predictions
Creating Summary plot for Marvel EYE predictions

The figure layout has changed to tight
```




```
Created Summary plot for Marvel EYE predictions
Creating Waterfall plot for Marvel EYE predictions
```

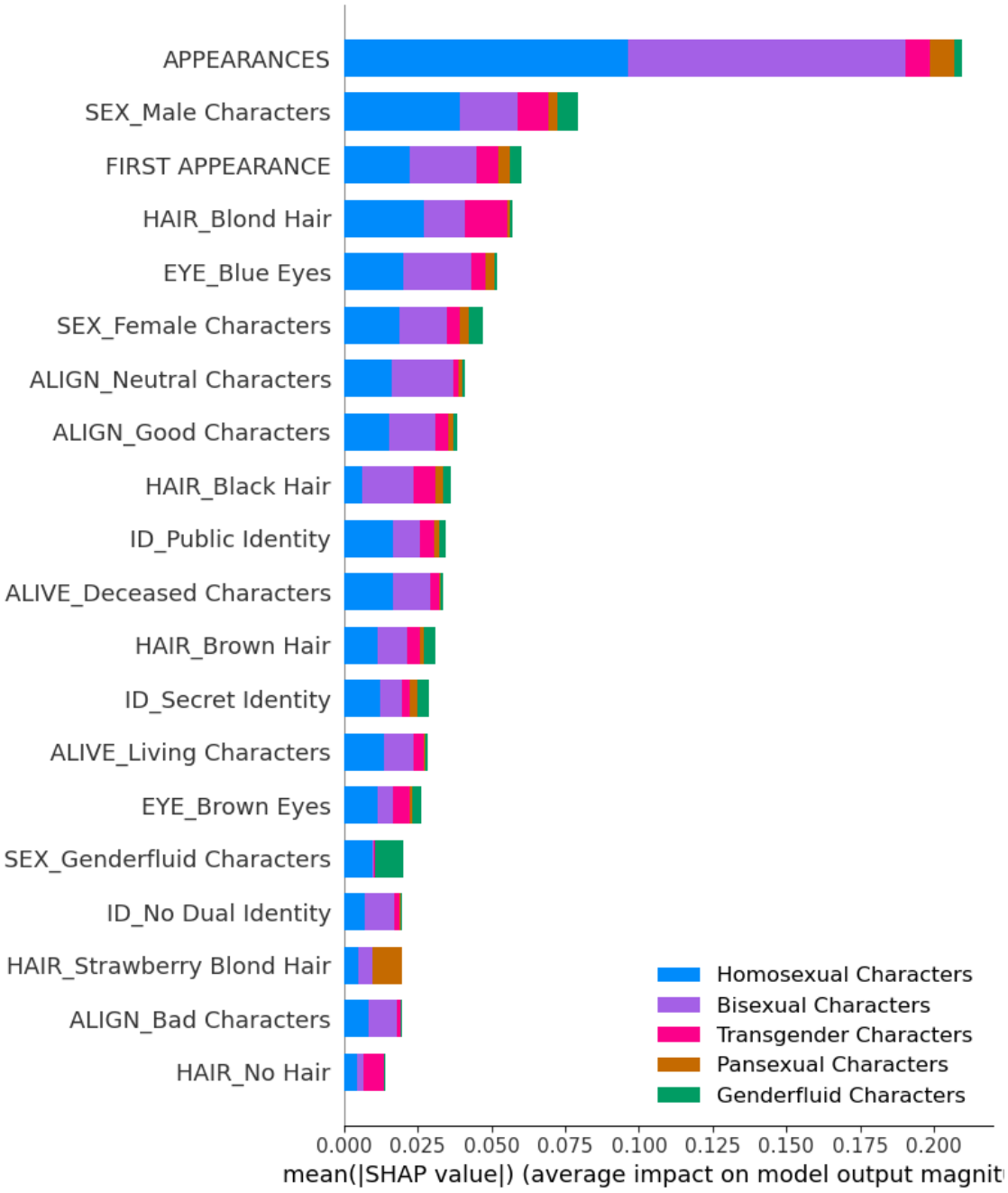



```
Created Waterfall plot for Marvel EYE predictions
Creating Summary plot for Marvel HAIR predictions

The figure layout has changed to tight
```

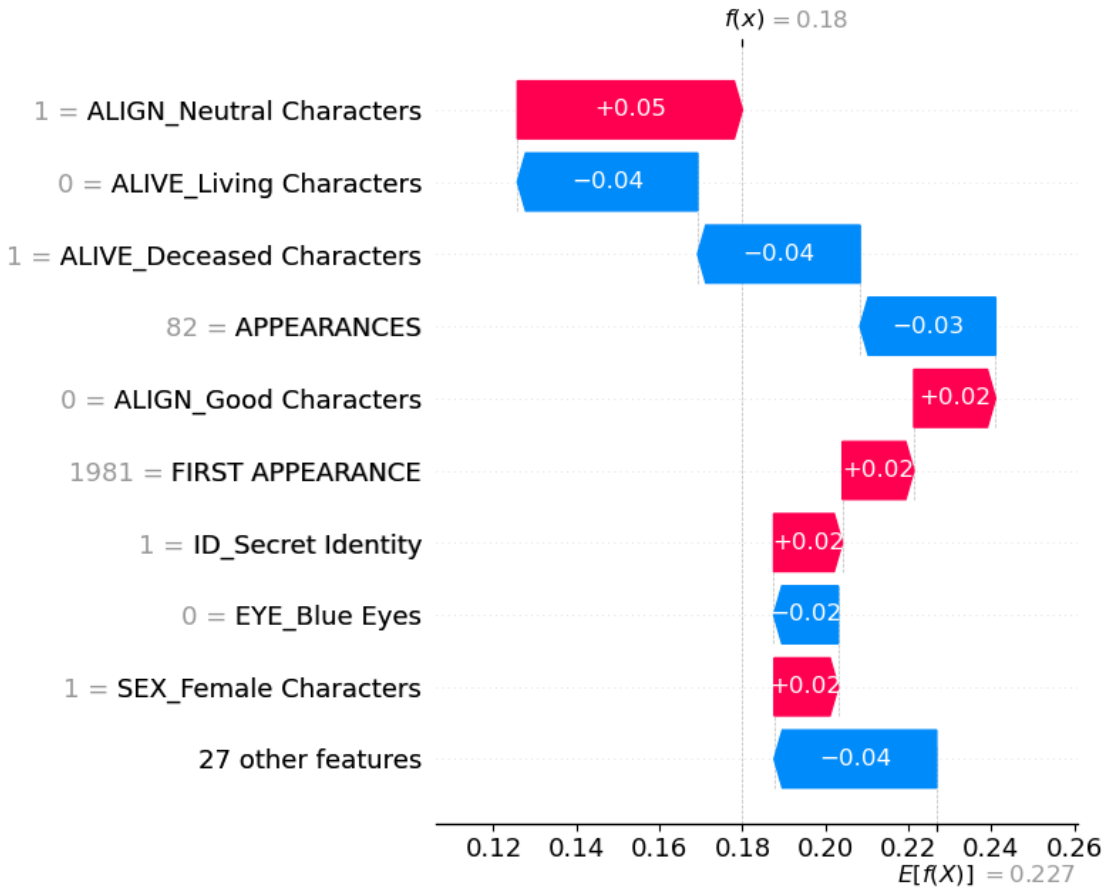



```
Created Summary plot for Marvel HAIR predictions
Creating Waterfall plot for Marvel HAIR predictions
```

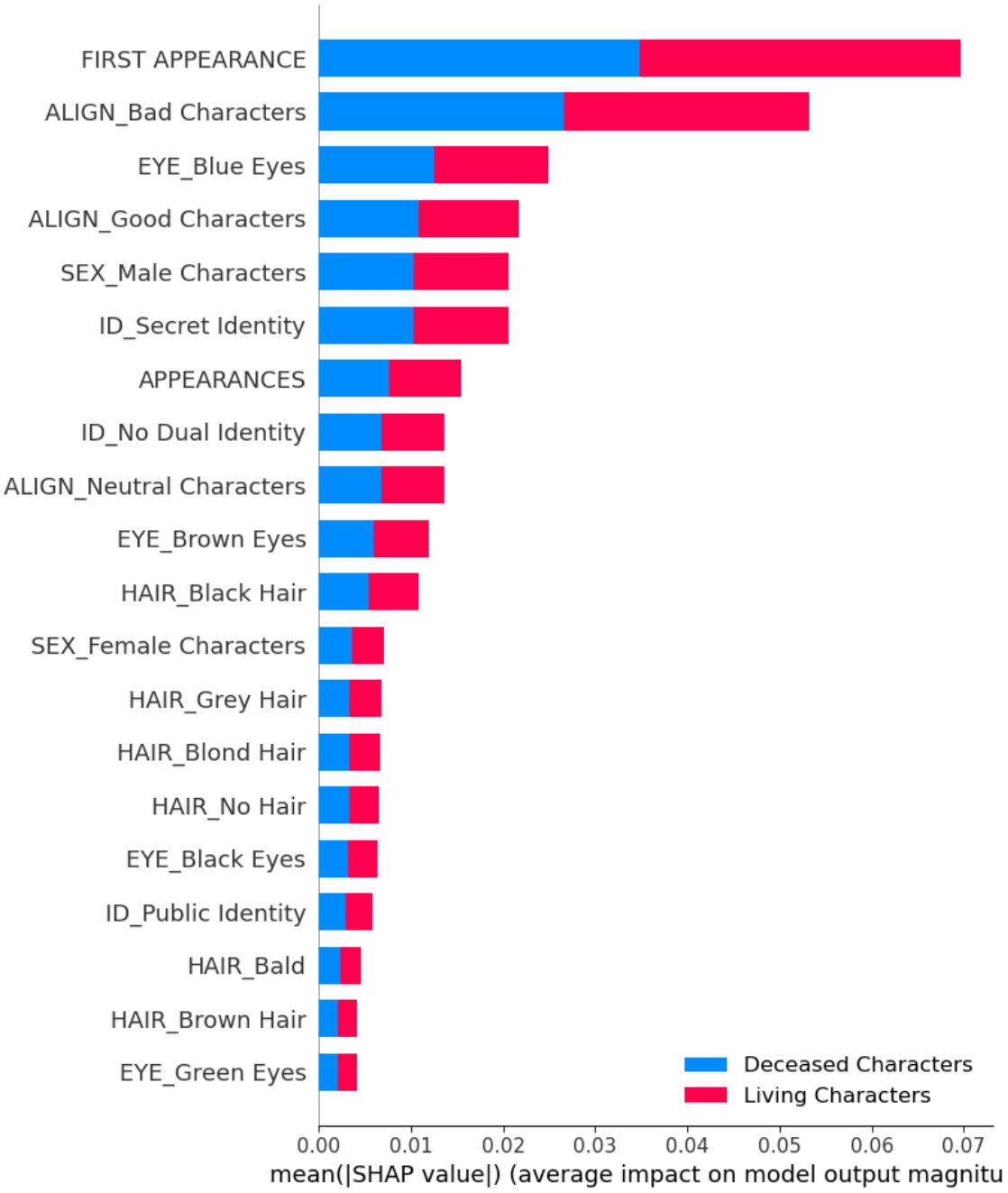

```
Created Waterfall plot for Marvel HAIR predictions
Creating Summary plot for Marvel GSM predictions

The figure layout has changed to tight
```



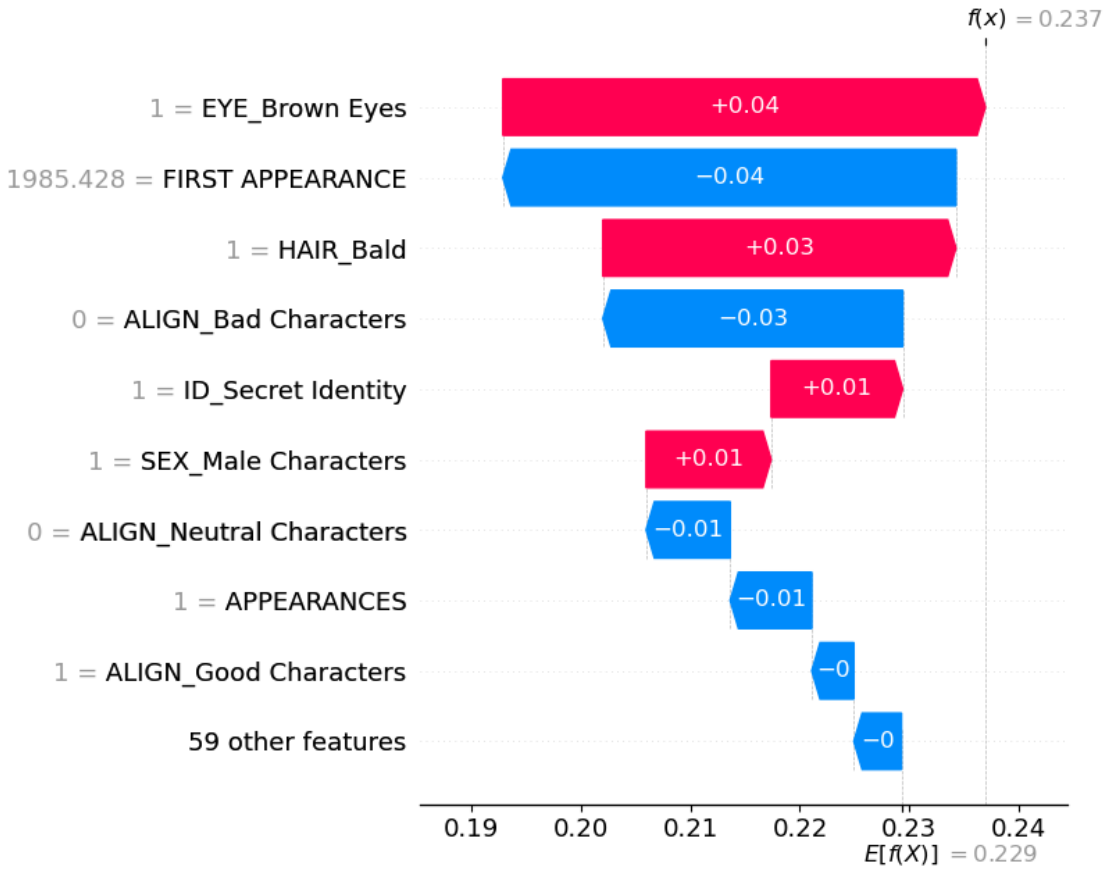

```
Created Summary plot for Marvel GSM predictions
Creating Waterfall plot for Marvel GSM predictions
```



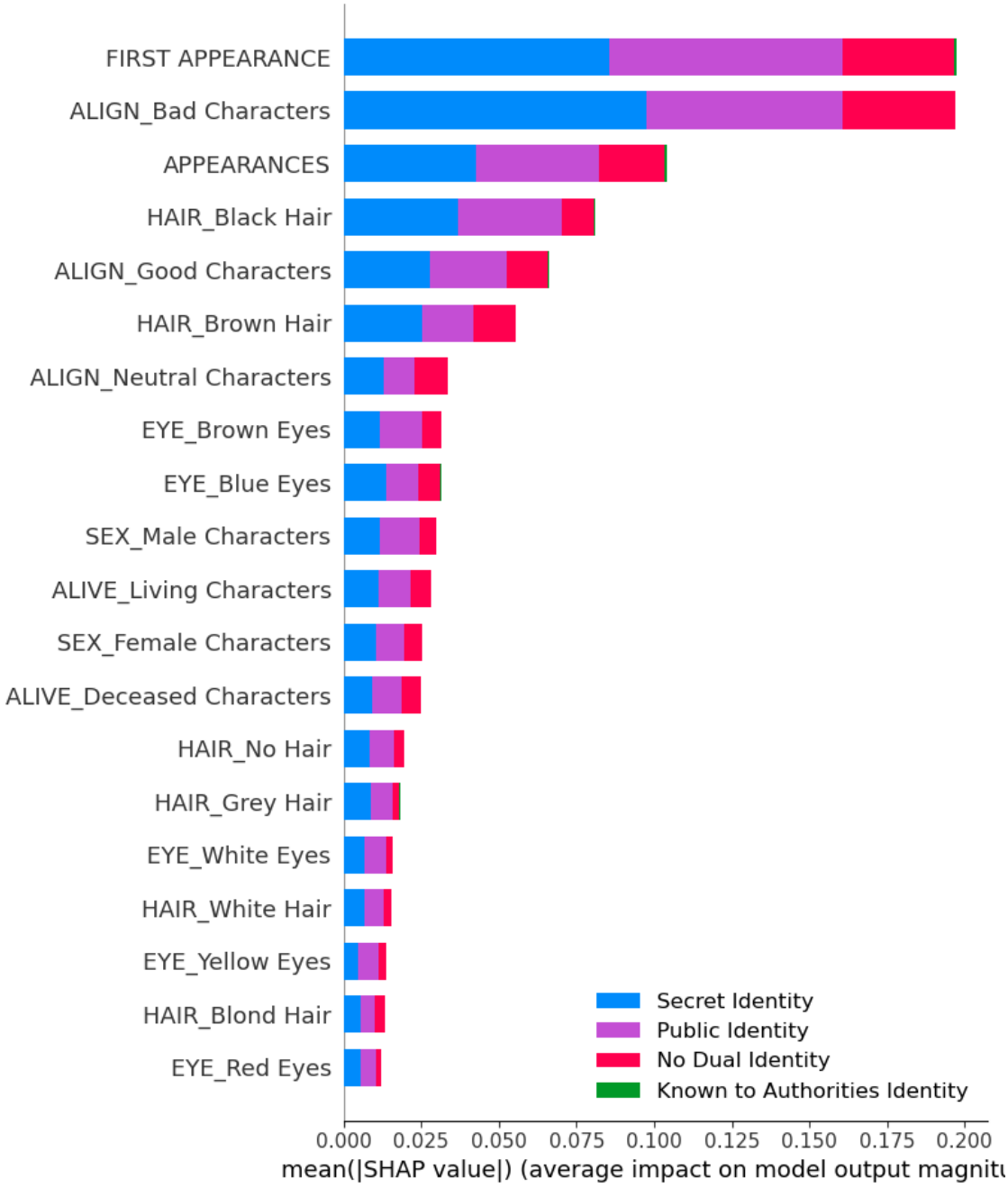

```
Created Waterfall plot for Marvel GSM predictions
Creating Summary plot for Marvel ALIVE predictions

The figure layout has changed to tight
```



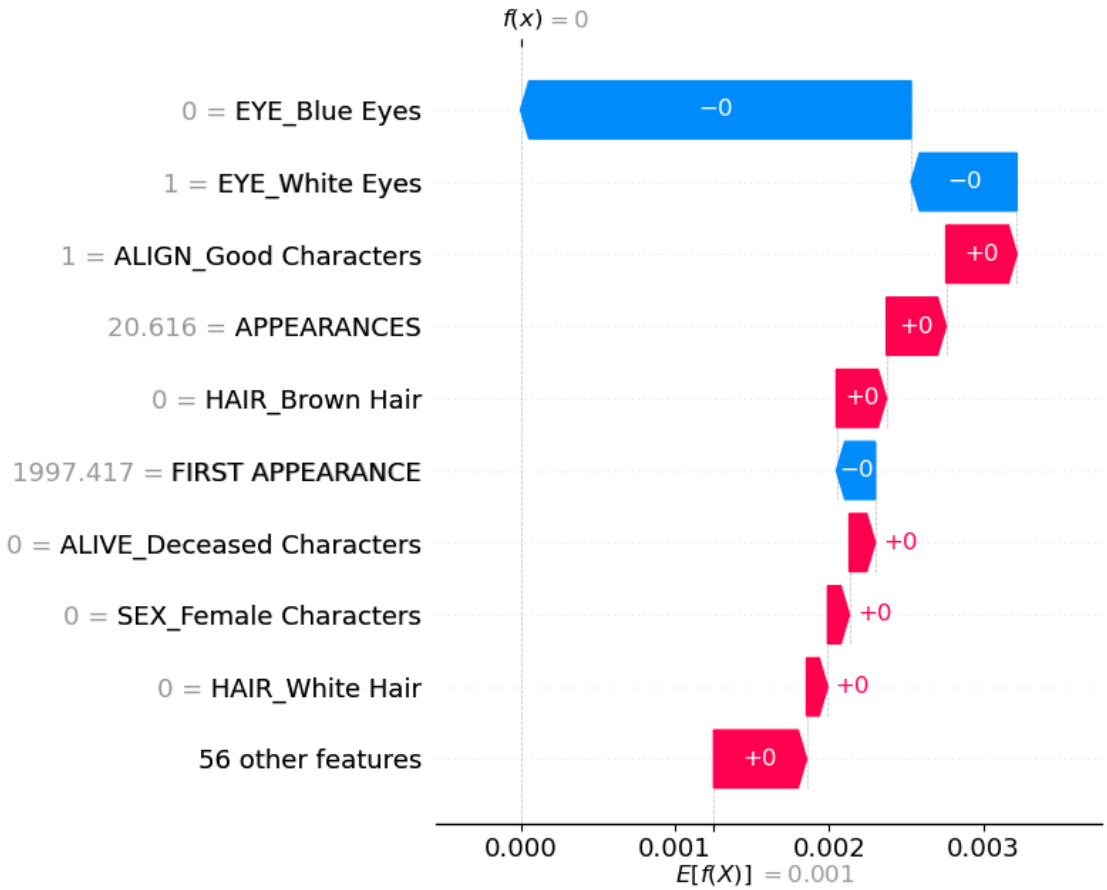

```
Created Summary plot for Marvel ALIVE predictions
Creating Waterfall plot for Marvel ALIVE predictions
```




```
Created Waterfall plot for Marvel ALIVE predictions
Creating Summary plot for Marvel ID predictions

The figure layout has changed to tight
```




```
Created Summary plot for Marvel ID predictions
Creating Waterfall plot for Marvel ID predictions
```



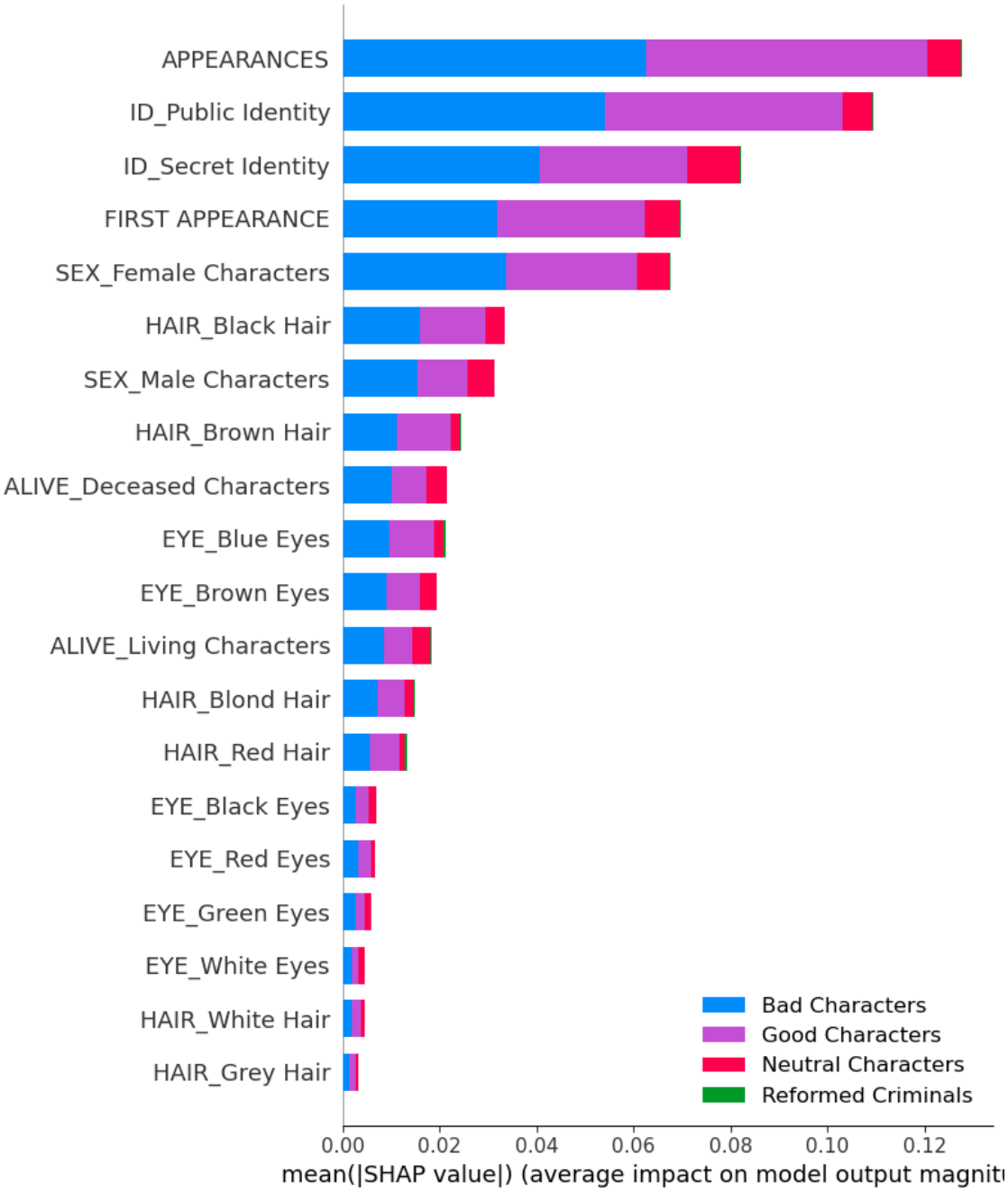

```
Created Waterfall plot for Marvel ID predictions
Marvel plots created
```

[ ]:
```python
# Create plots for DC

for explainer, shap_values, feature_subset, target_feature, class_names in␣
 ↪shap_models_dc:
    sample_index = random.randint(0, len(feature_subset) - 1) # Generate a␣
 ↪random index within the valid range

    print(f"""Creating Summary plot for DC {target_feature} predictions""")
    shap.summary_plot(shap_values, feature_subset, feature_names=feature_subset.
 ↪columns, show=False, class_names=class_names)
    plt.savefig(f"""figs/summary/DC_{target_feature}_summary.png""")
    plt.tight_layout()
    plt.show()
    print(f"""Created Summary plot for DC {target_feature} predictions""")

    print(f"Creating Waterfall plot for DC {target_feature} predictions")
```

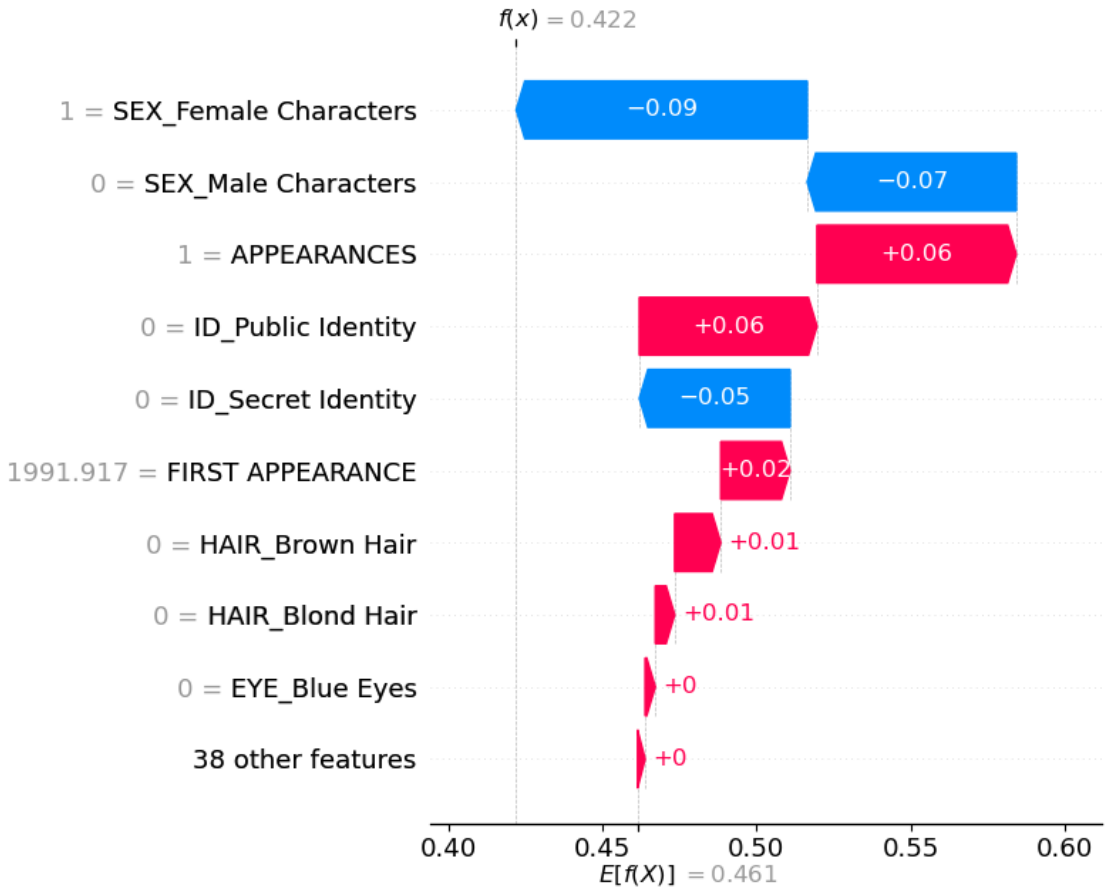
```
    shap.plots.waterfall(shap.Explanation(values=shap_values[0][sample_index],␣
 ↪base_values=explainer.expected_value[0], data=feature_subset.
 ↪iloc[sample_index]), max_display=10, show=False)
    plt.savefig(f"figs/waterfall/DC_{target_feature}_{sample_index}_waterfall.
 ↪png")
    plt.tight_layout()
    plt.show()
    print(f"Created Waterfall plot for DC {target_feature} predictions")


print("DC plots created")
```

```
Creating Summary plot for DC ALIGN predictions
```

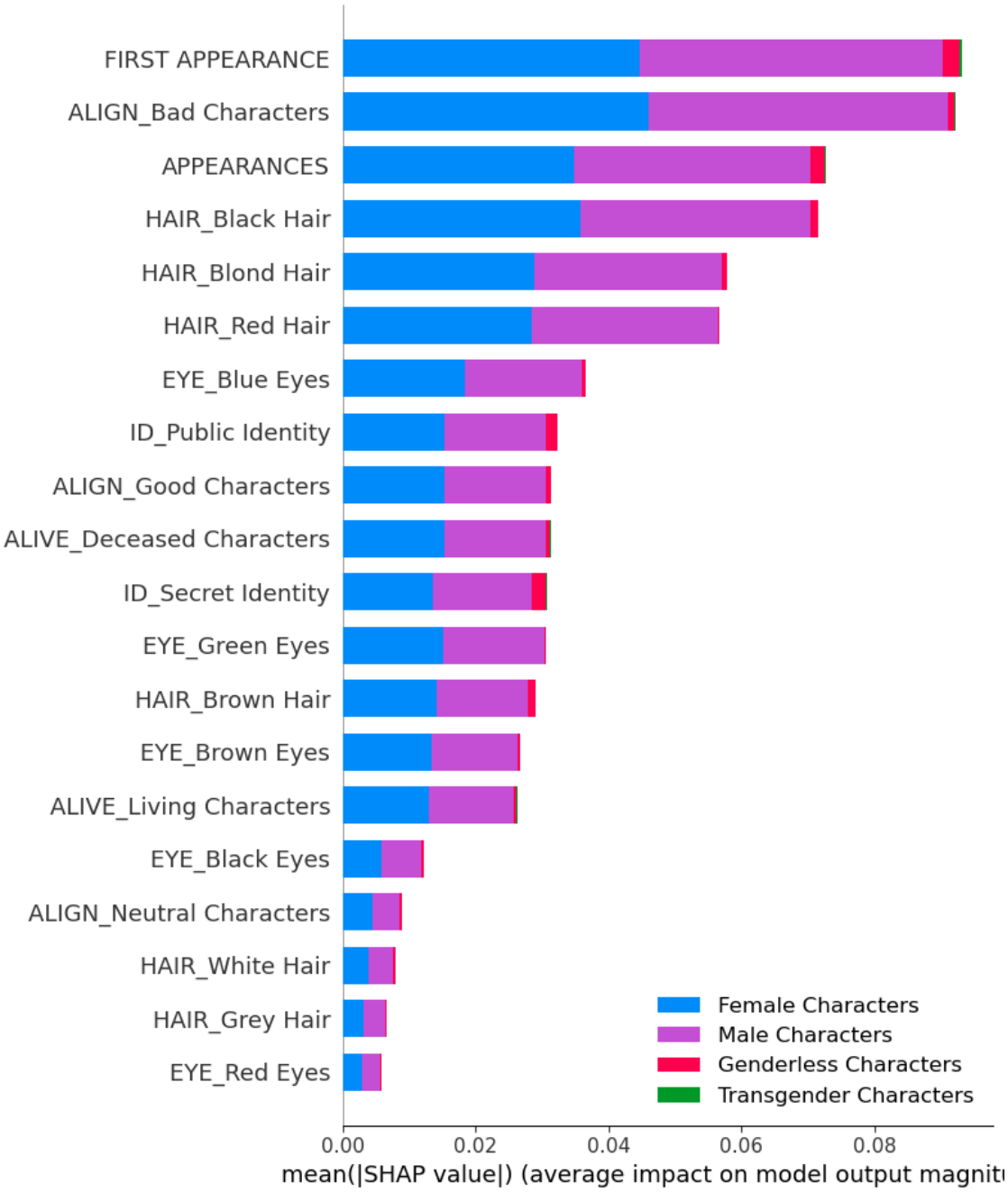
```
The figure layout has changed to tight
```



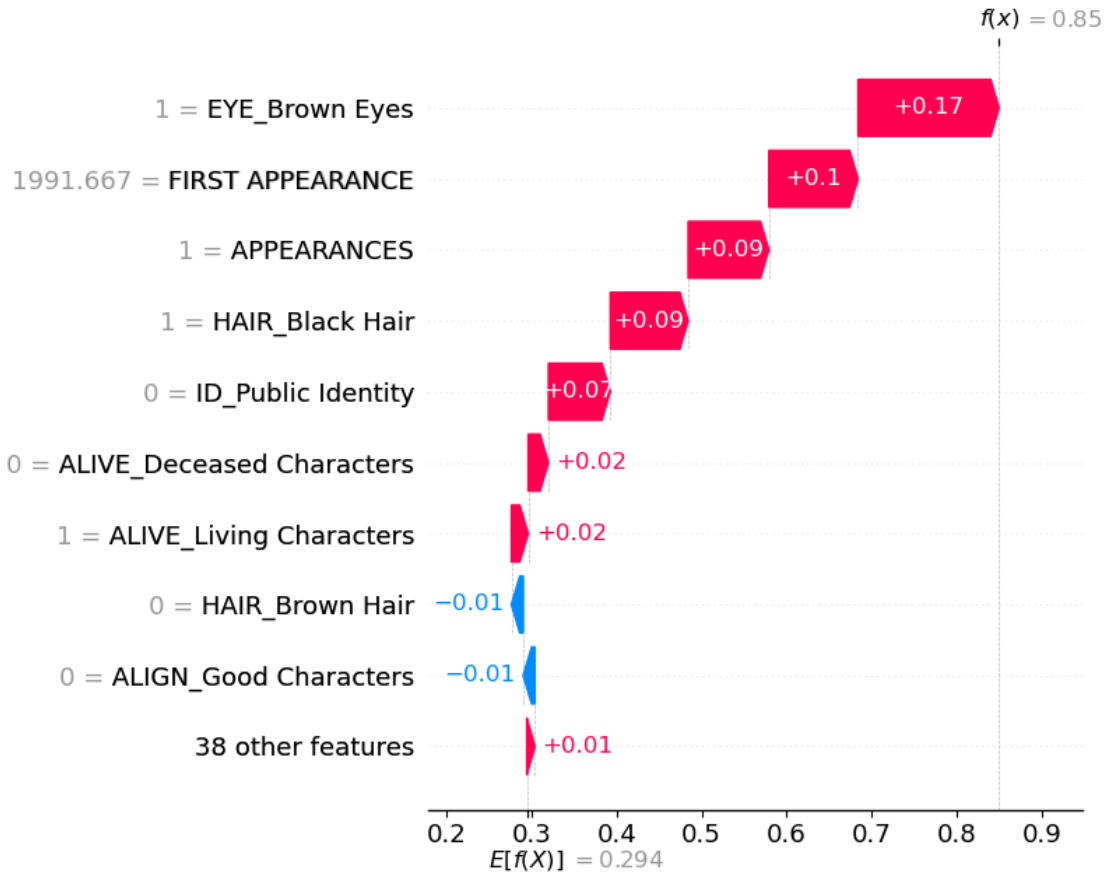

```
Created Summary plot for DC ALIGN predictions
Creating Waterfall plot for DC ALIGN predictions
```



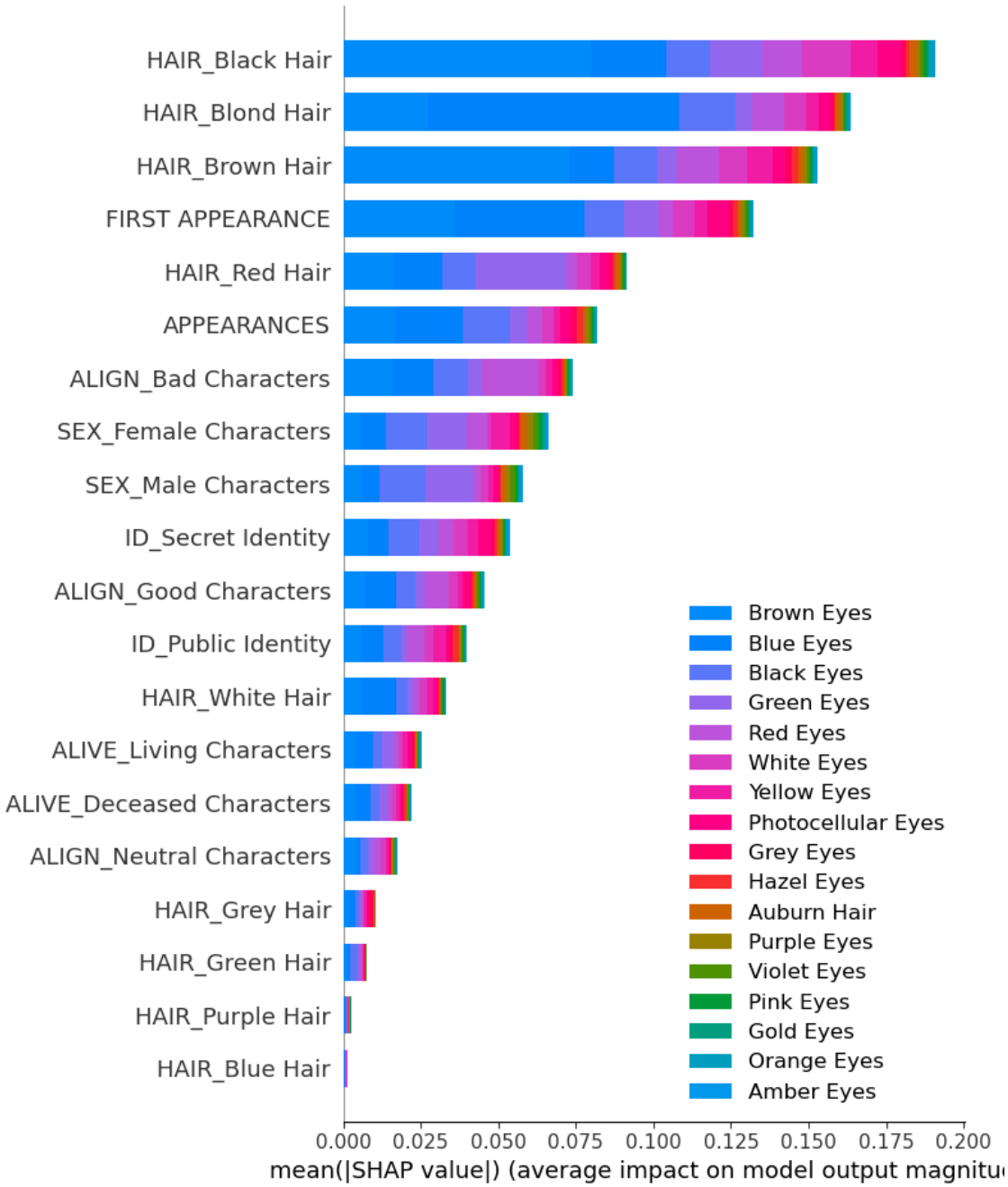

```
Created Waterfall plot for DC ALIGN predictions
Creating Summary plot for DC SEX predictions

The figure layout has changed to tight
```



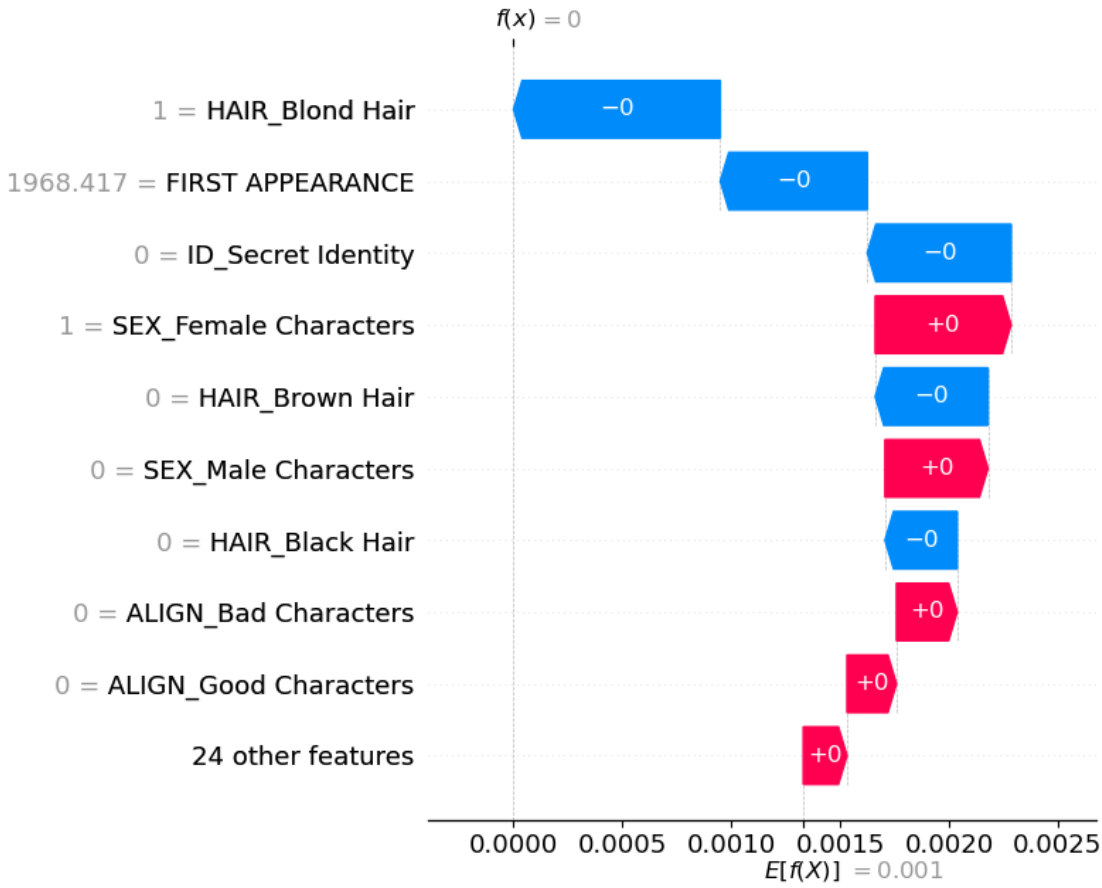

```
Created Summary plot for DC SEX predictions
Creating Waterfall plot for DC SEX predictions
```



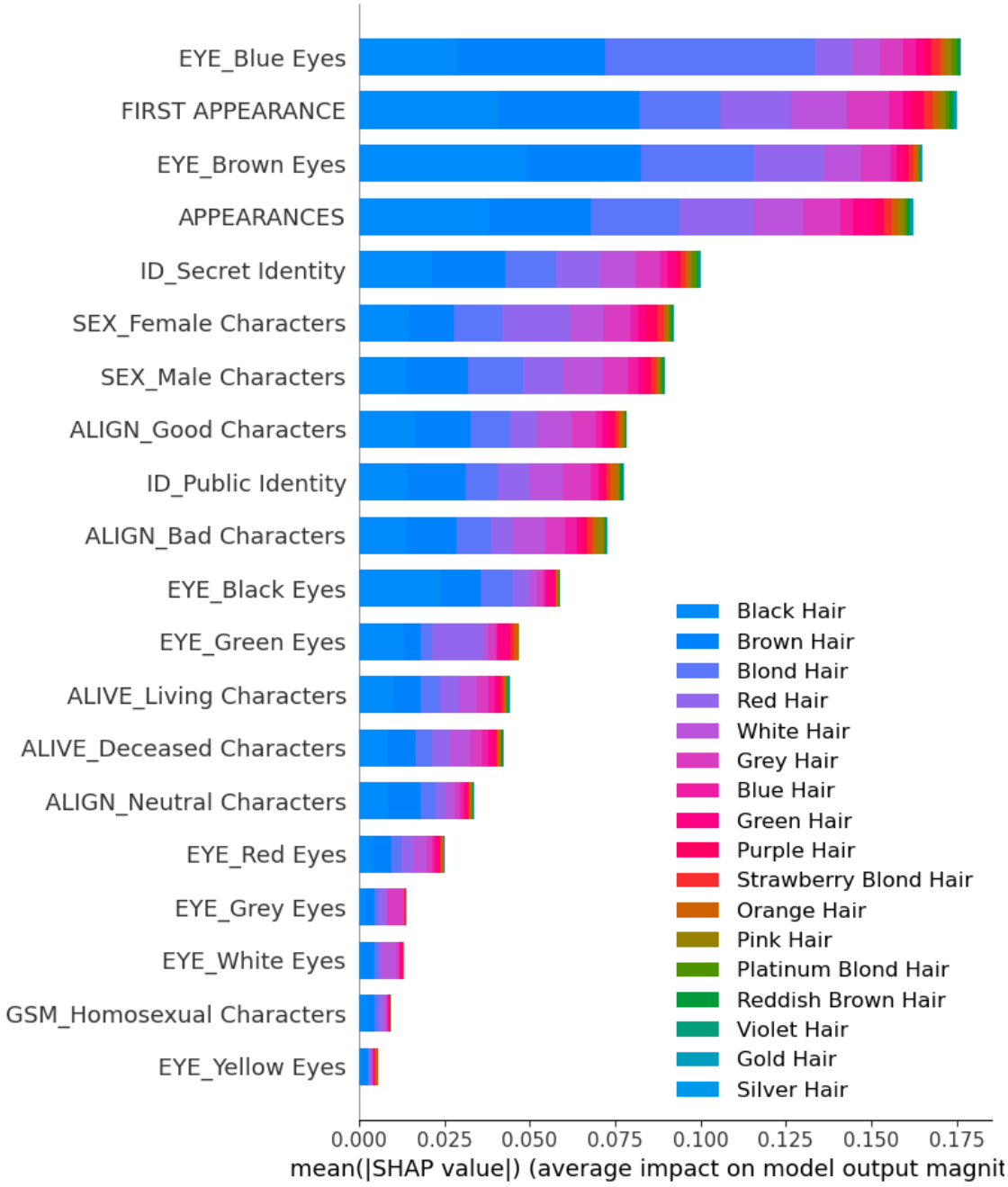

```
Created Waterfall plot for DC SEX predictions
Creating Summary plot for DC EYE predictions

The figure layout has changed to tight
```



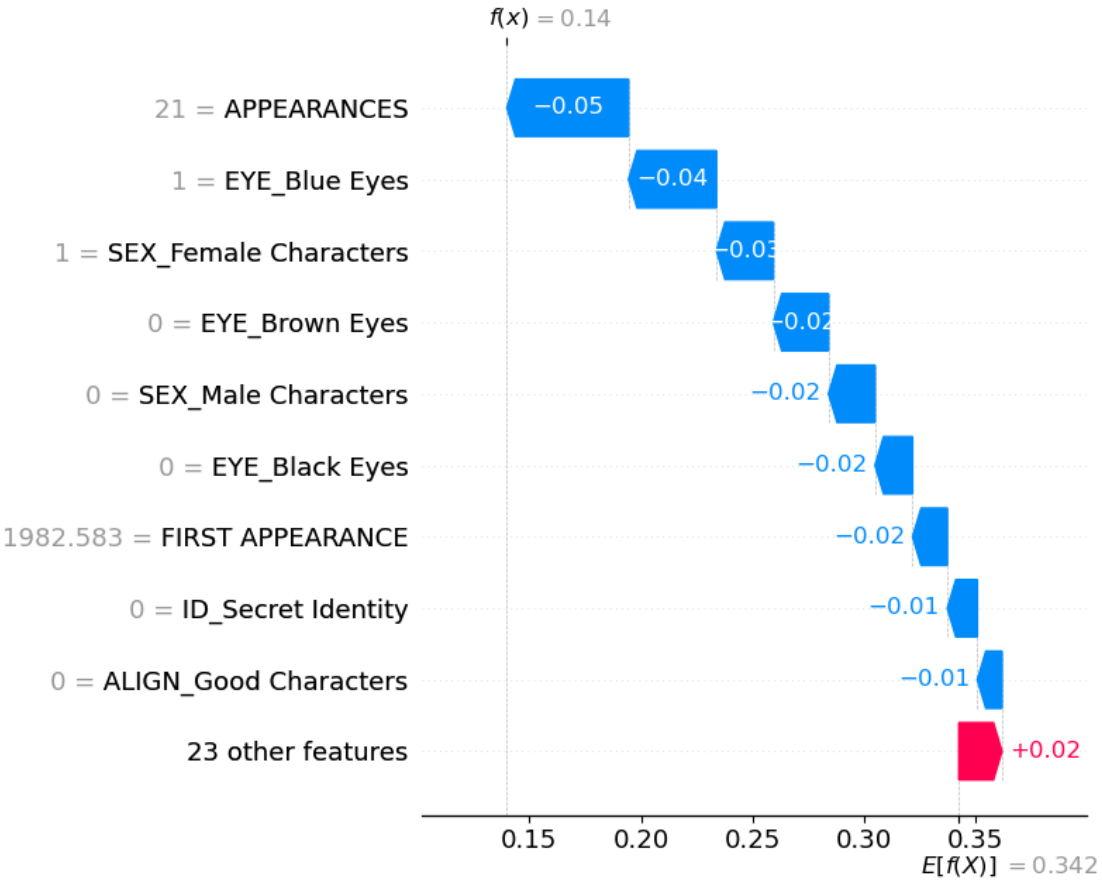

```
Created Summary plot for DC EYE predictions
Creating Waterfall plot for DC EYE predictions
```



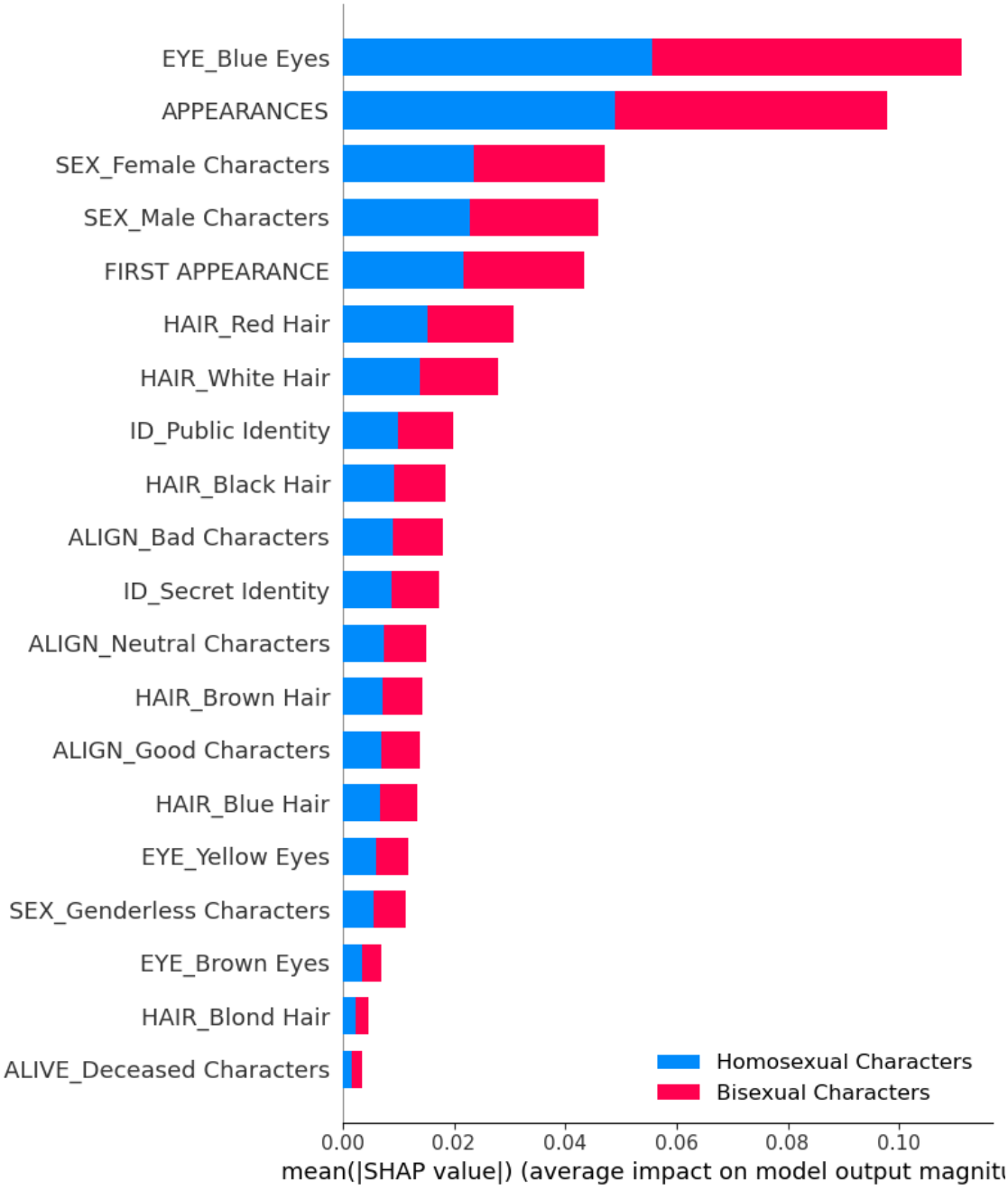

```
Created Waterfall plot for DC EYE predictions
Creating Summary plot for DC HAIR predictions

The figure layout has changed to tight
```



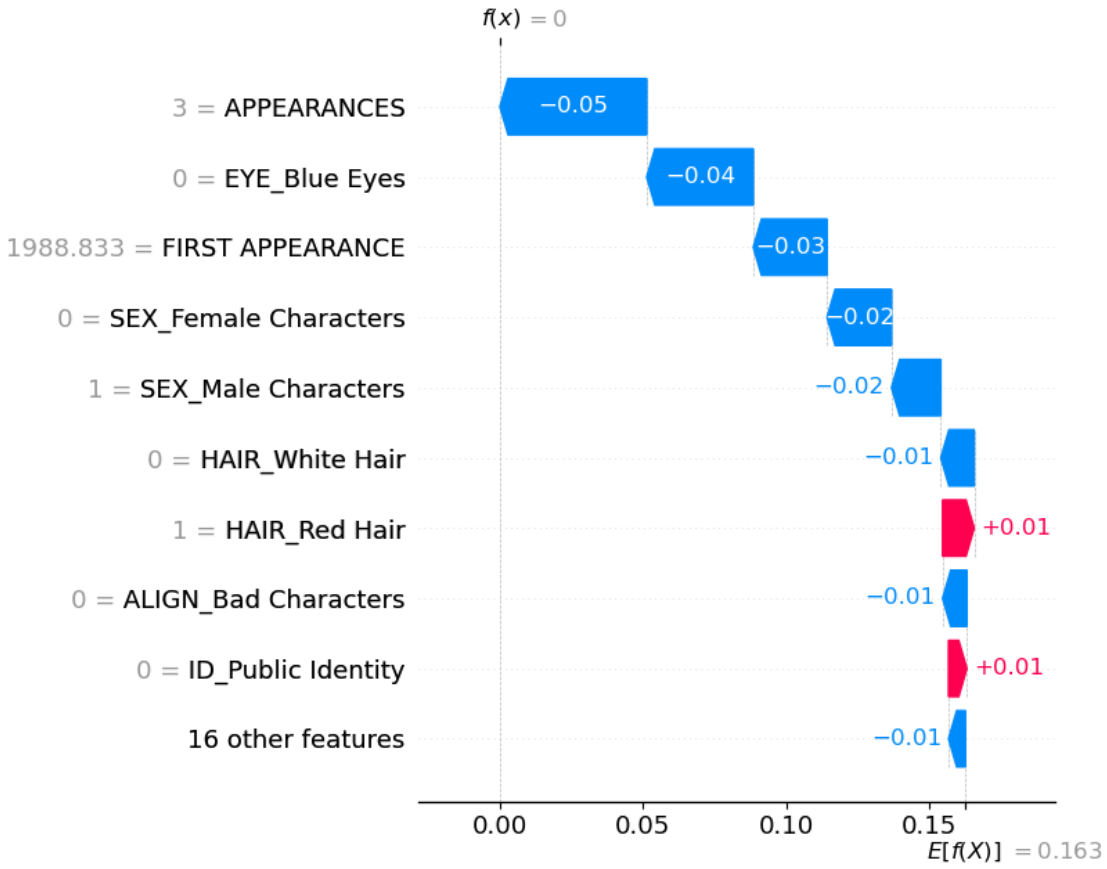

```
Created Summary plot for DC HAIR predictions
Creating Waterfall plot for DC HAIR predictions
```



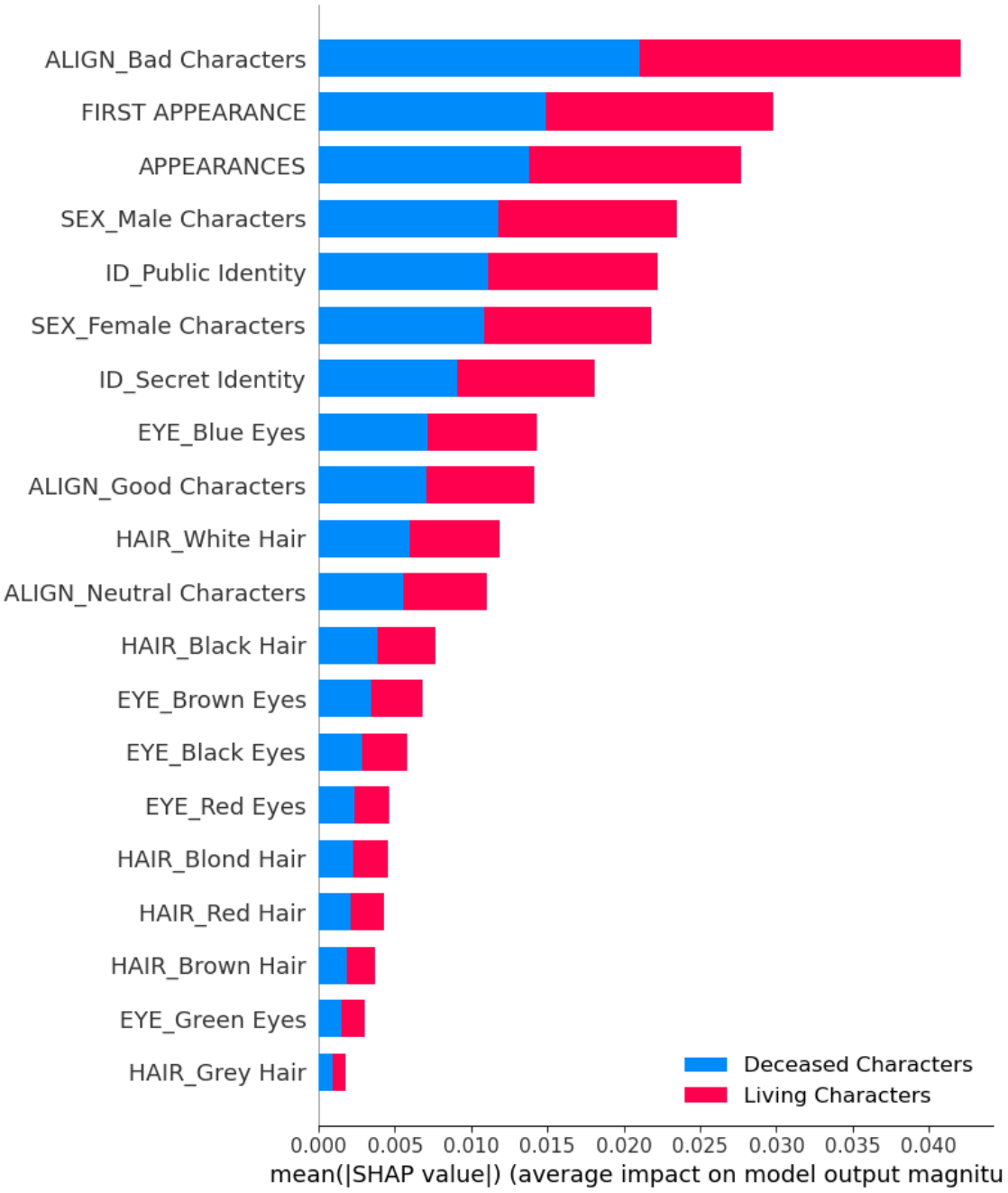

```
Created Waterfall plot for DC HAIR predictions
Creating Summary plot for DC GSM predictions

The figure layout has changed to tight
```



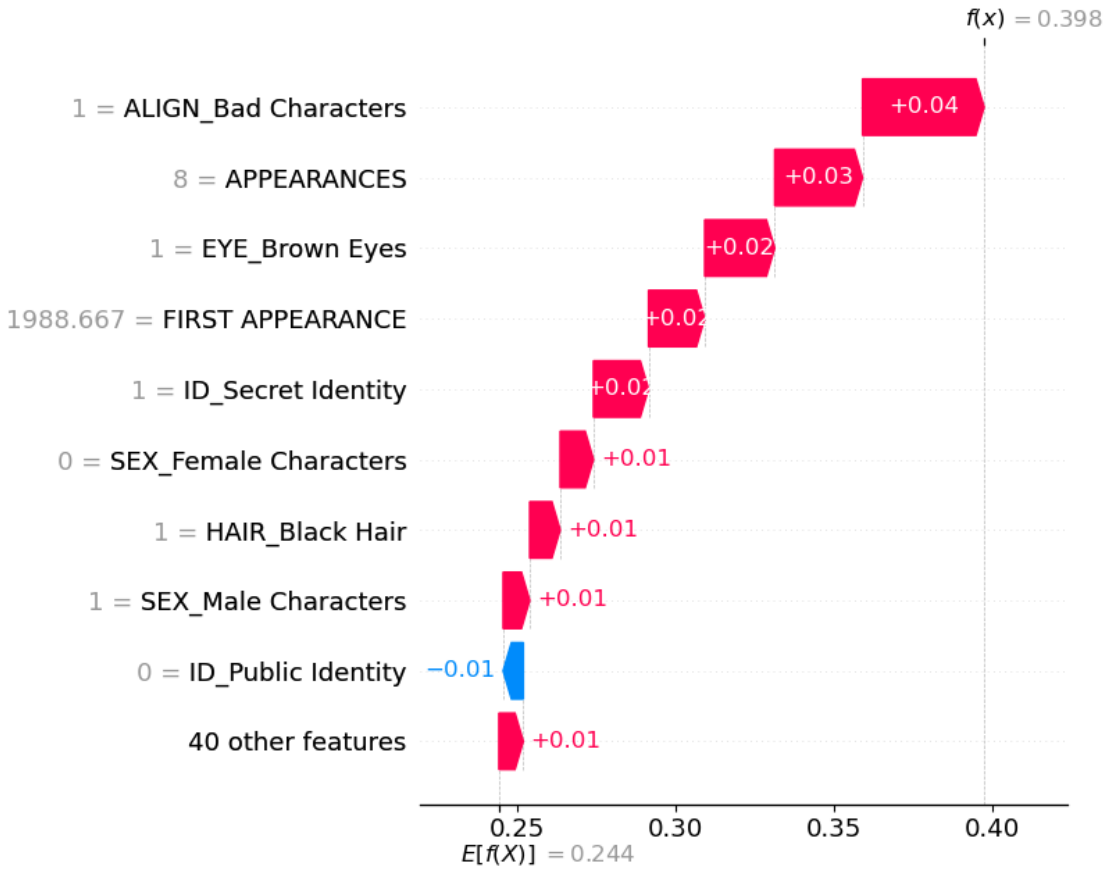

```
Created Summary plot for DC GSM predictions
Creating Waterfall plot for DC GSM predictions
```



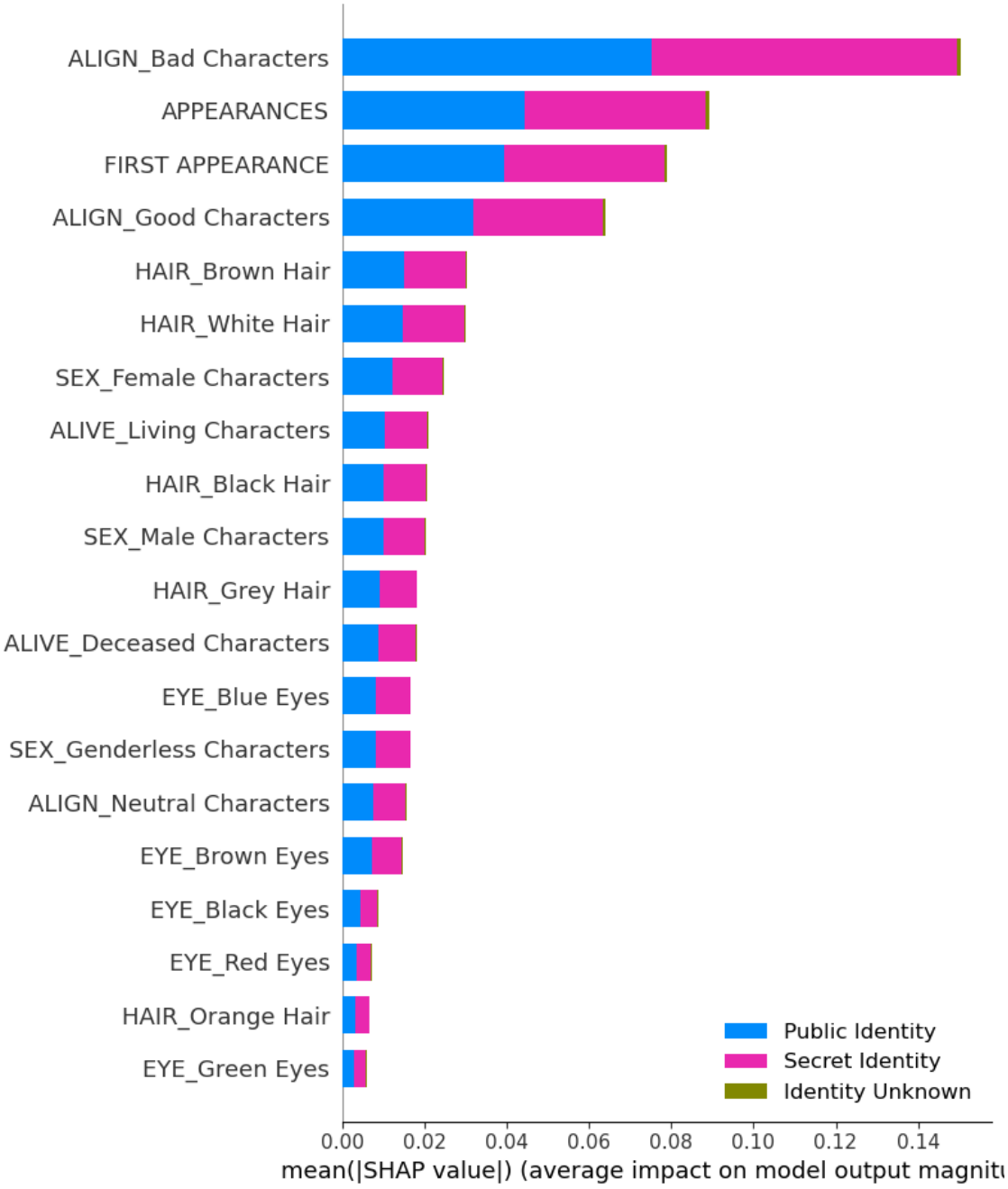

```
Created Waterfall plot for DC GSM predictions
Creating Summary plot for DC ALIVE predictions

The figure layout has changed to tight
```



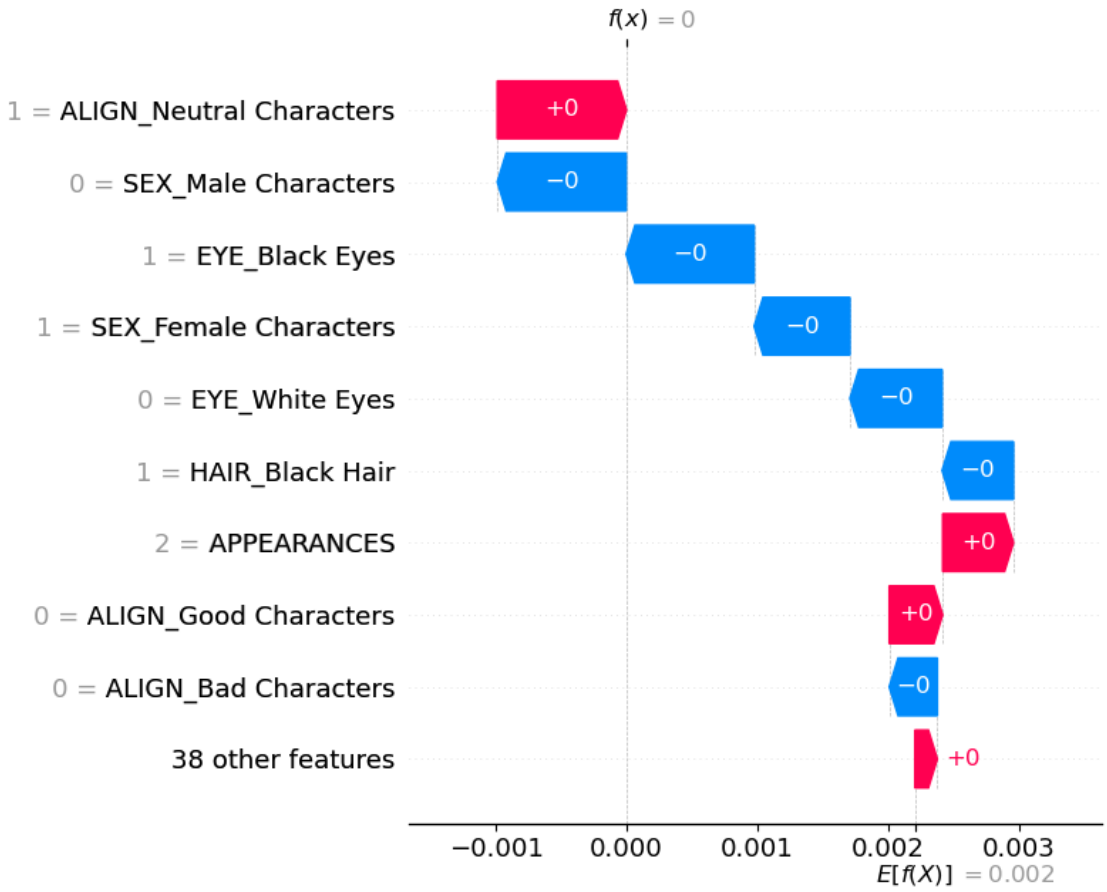

```
Created Summary plot for DC ALIVE predictions
Creating Waterfall plot for DC ALIVE predictions
```




```
Created Waterfall plot for DC ALIVE predictions
Creating Summary plot for DC ID predictions

The figure layout has changed to tight
```




```
Created Summary plot for DC ID predictions
Creating Waterfall plot for DC ID predictions
```



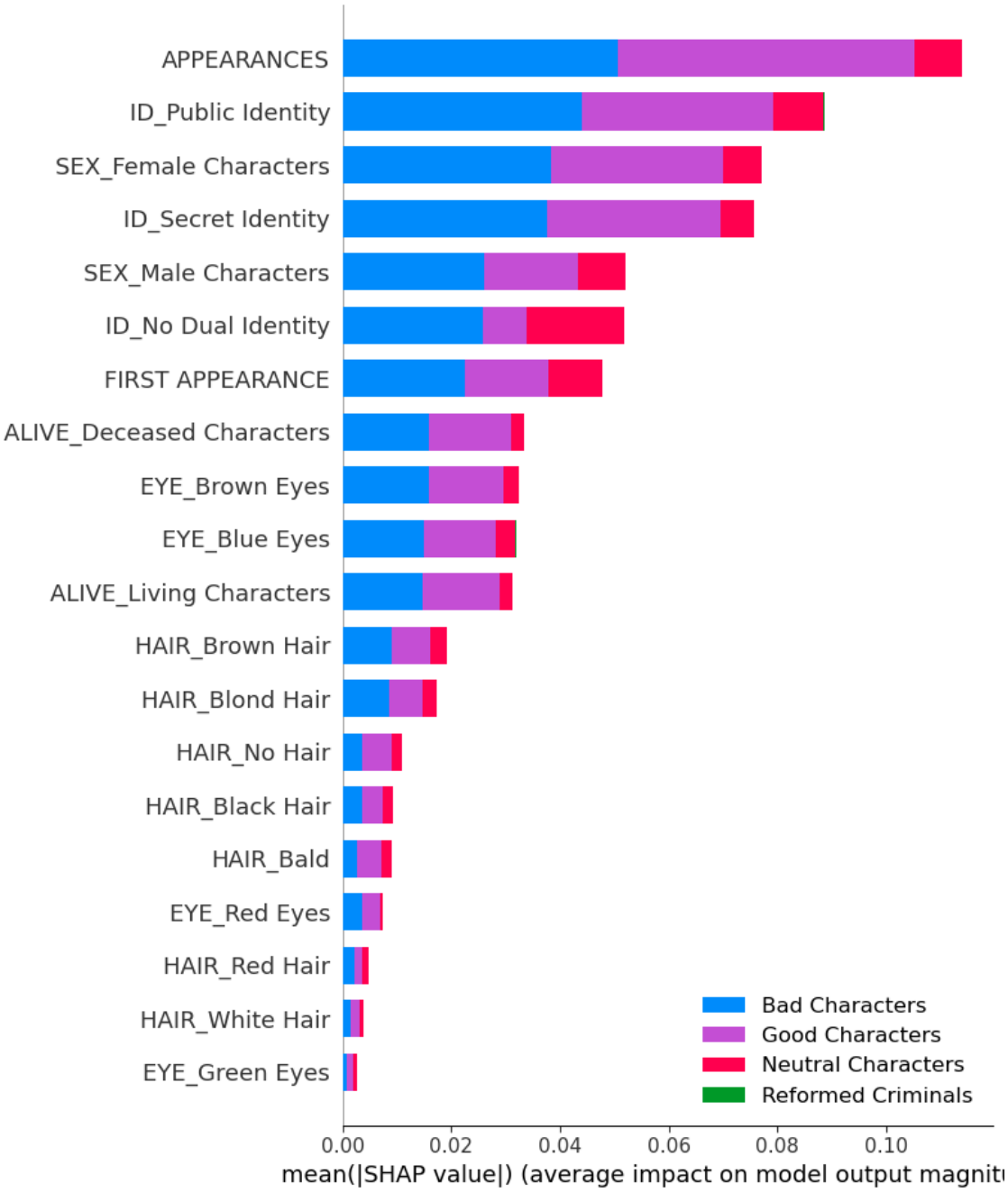

```
Created Waterfall plot for DC ID predictions
DC plots created
```

[ ]:
```
# Create plots for Combined

for explainer, shap_values, feature_subset, target_feature, class_names in␣
 ↪shap_models_combined:
    sample_index = random.randint(0, len(feature_subset) - 1) # Generate a␣
 ↪random index within the valid range

    print(f"""Creating Summary plot for Combined {target_feature}␣
 ↪predictions""")
    shap.summary_plot(shap_values, feature_subset, feature_names=feature_subset.
 ↪columns, show=False, class_names=class_names)
    plt.savefig(f"""figs/summary/Combined_{target_feature}_summary.png""")
    plt.tight_layout()
    plt.show()
    print(f"""Created Summary plot for Combined {target_feature} predictions""")
```

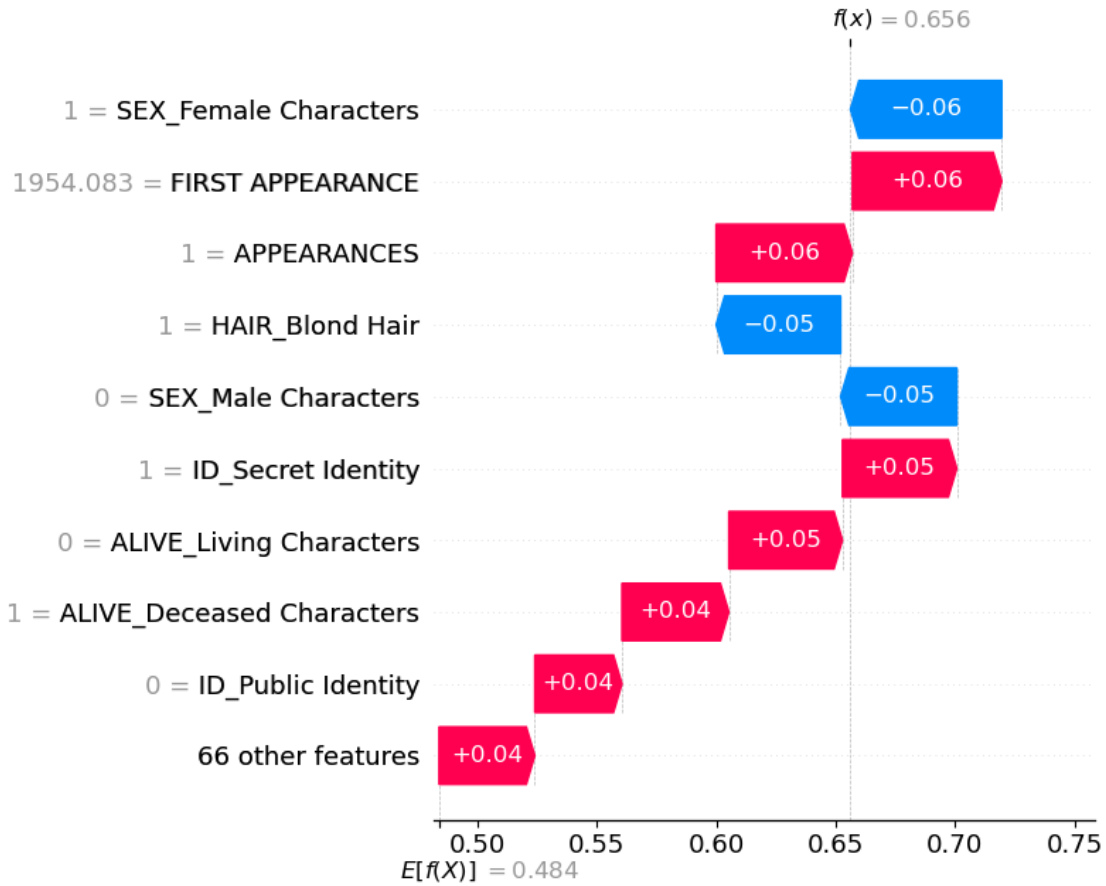
```
    print(f"Creating Waterfall plot for Combined {target_feature} predictions")
    shap.plots.waterfall(shap.Explanation(values=shap_values[0][sample_index],␣
 ↪base_values=explainer.expected_value[0], data=feature_subset.
 ↪iloc[sample_index]), max_display=10, show=False)
    plt.savefig(f"figs/waterfall/
 ↪Combined_{target_feature}_{sample_index}_waterfall.png")
    plt.tight_layout()
    plt.show()
    print(f"Created Waterfall plot for Combined {target_feature} predictions")

print("Combined plots created")
```

```
Creating Summary plot for Combined ALIGN predictions
```

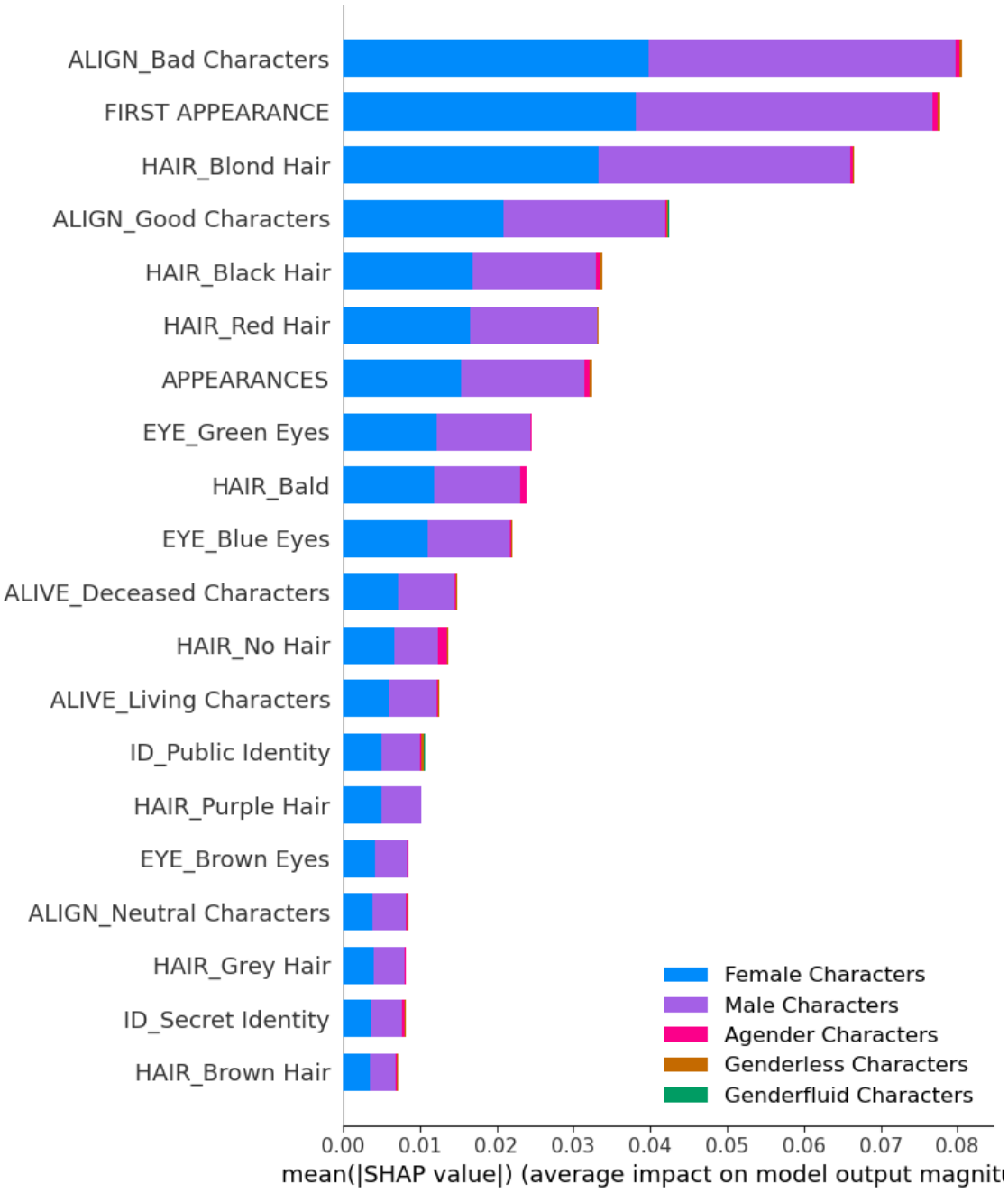
```
The figure layout has changed to tight
```



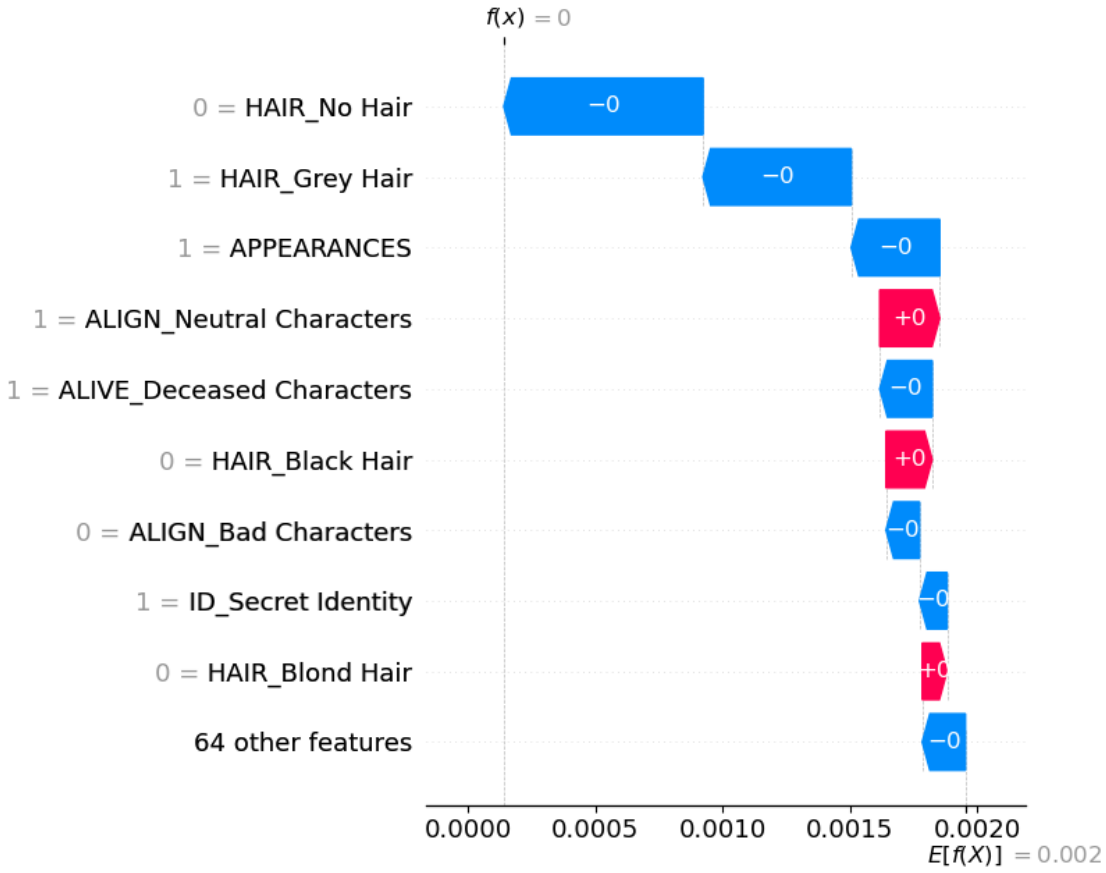

```
Created Summary plot for Combined ALIGN predictions
Creating Waterfall plot for Combined ALIGN predictions
```



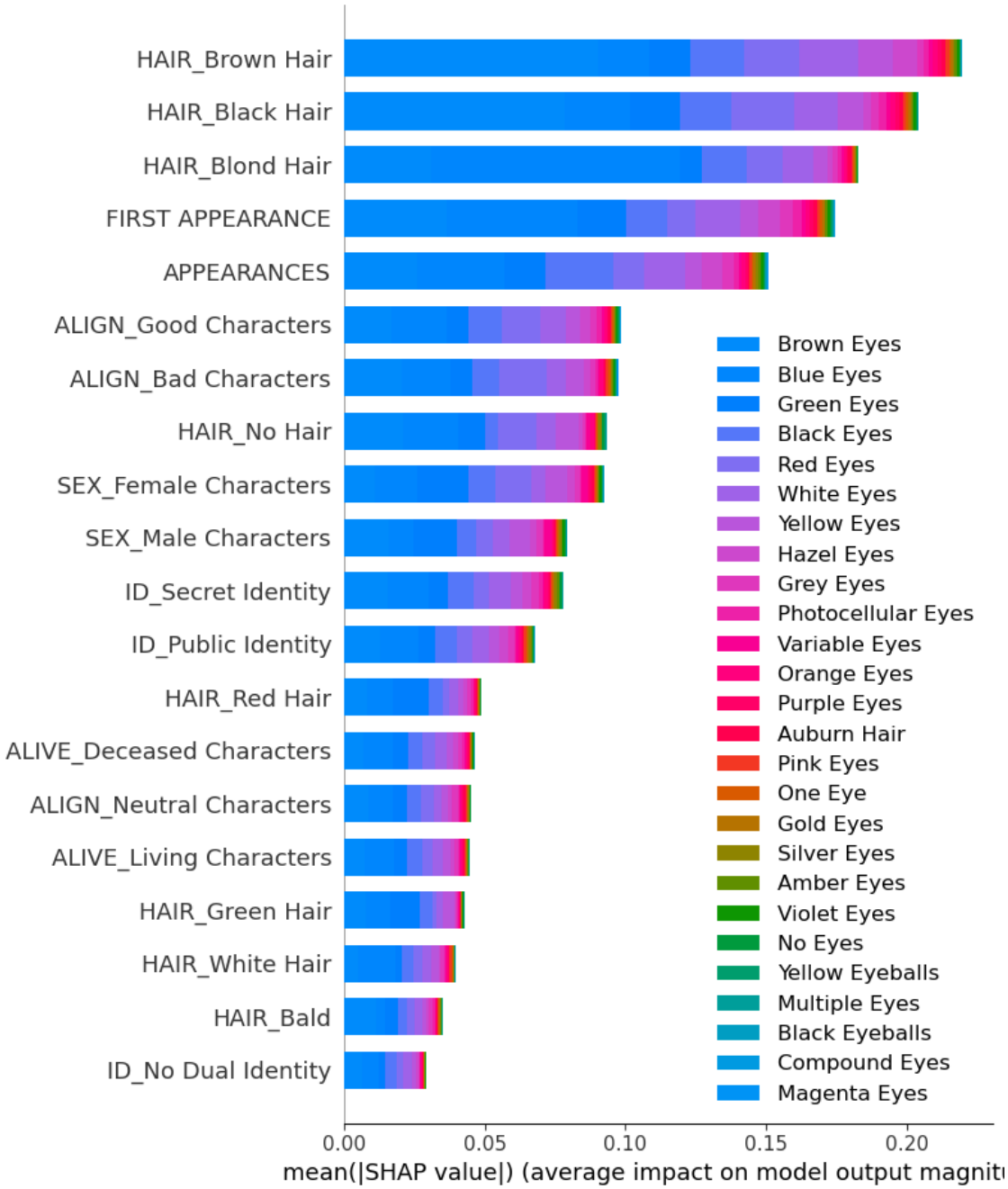

```
Created Waterfall plot for Combined ALIGN predictions
Creating Summary plot for Combined SEX predictions

The figure layout has changed to tight
```



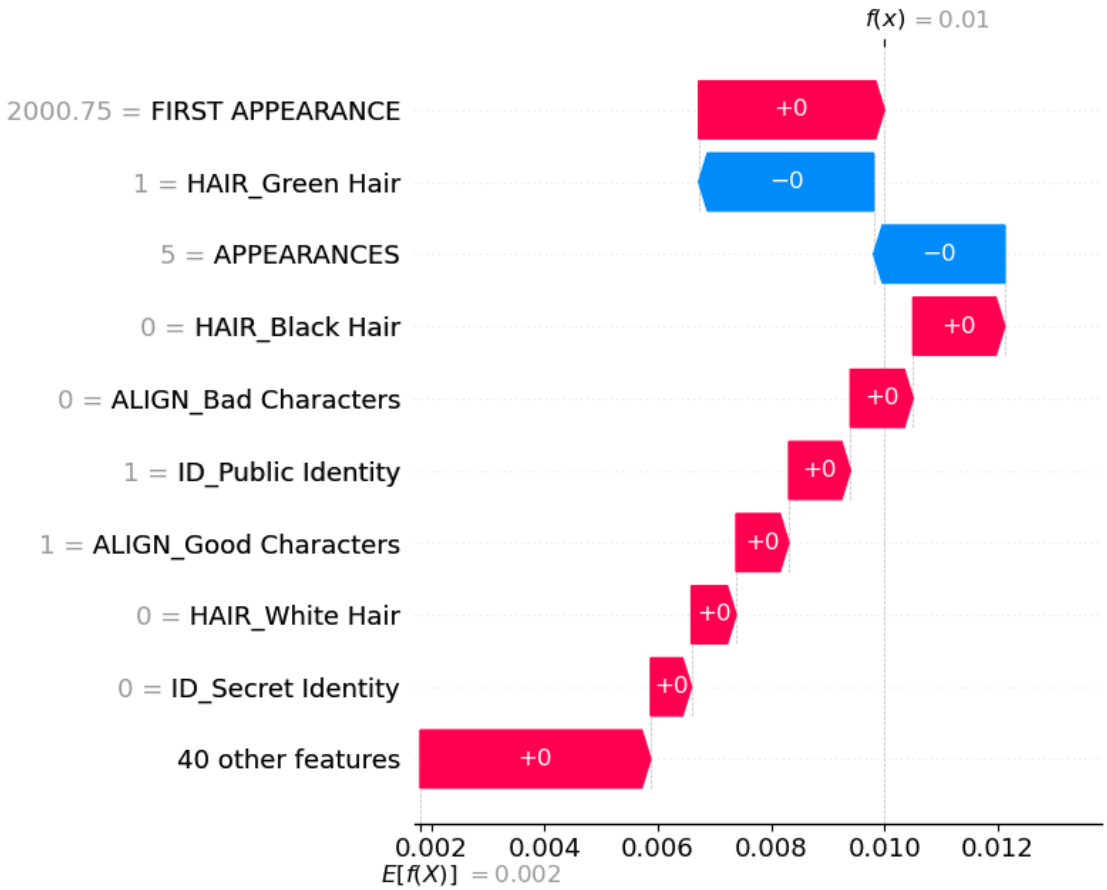

```
Created Summary plot for Combined SEX predictions
Creating Waterfall plot for Combined SEX predictions
```



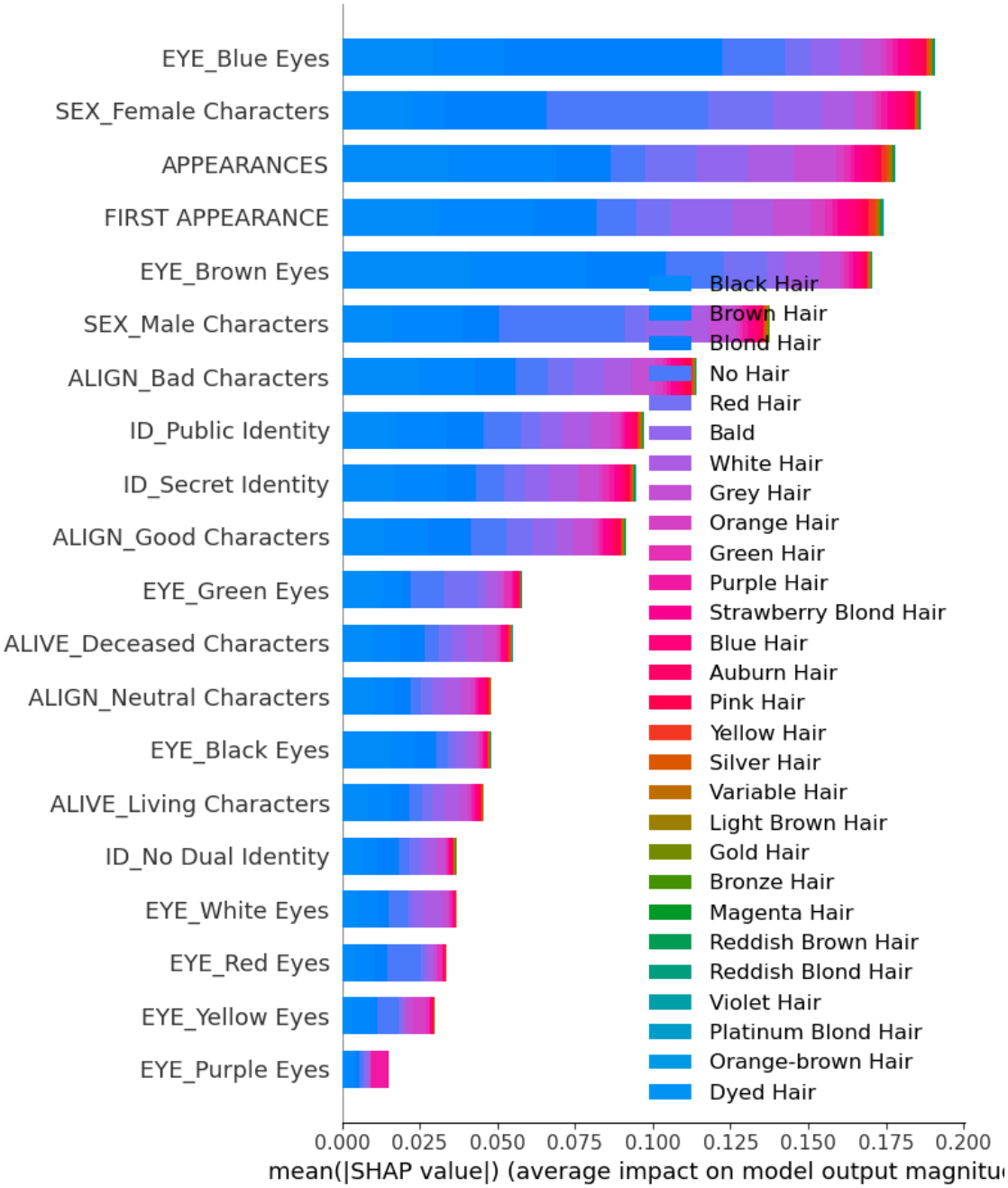

```
Created Waterfall plot for Combined SEX predictions
Creating Summary plot for Combined EYE predictions

The figure layout has changed to tight
```



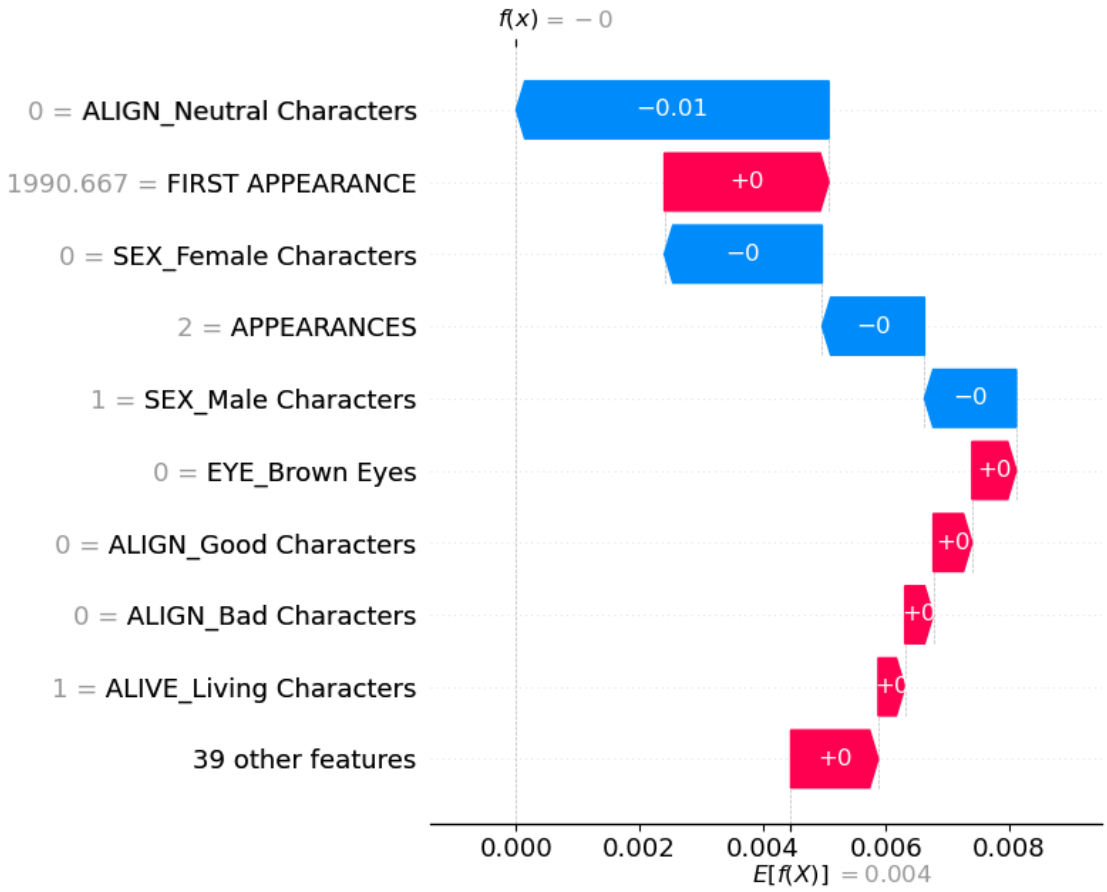

```
Created Summary plot for Combined EYE predictions
Creating Waterfall plot for Combined EYE predictions
```



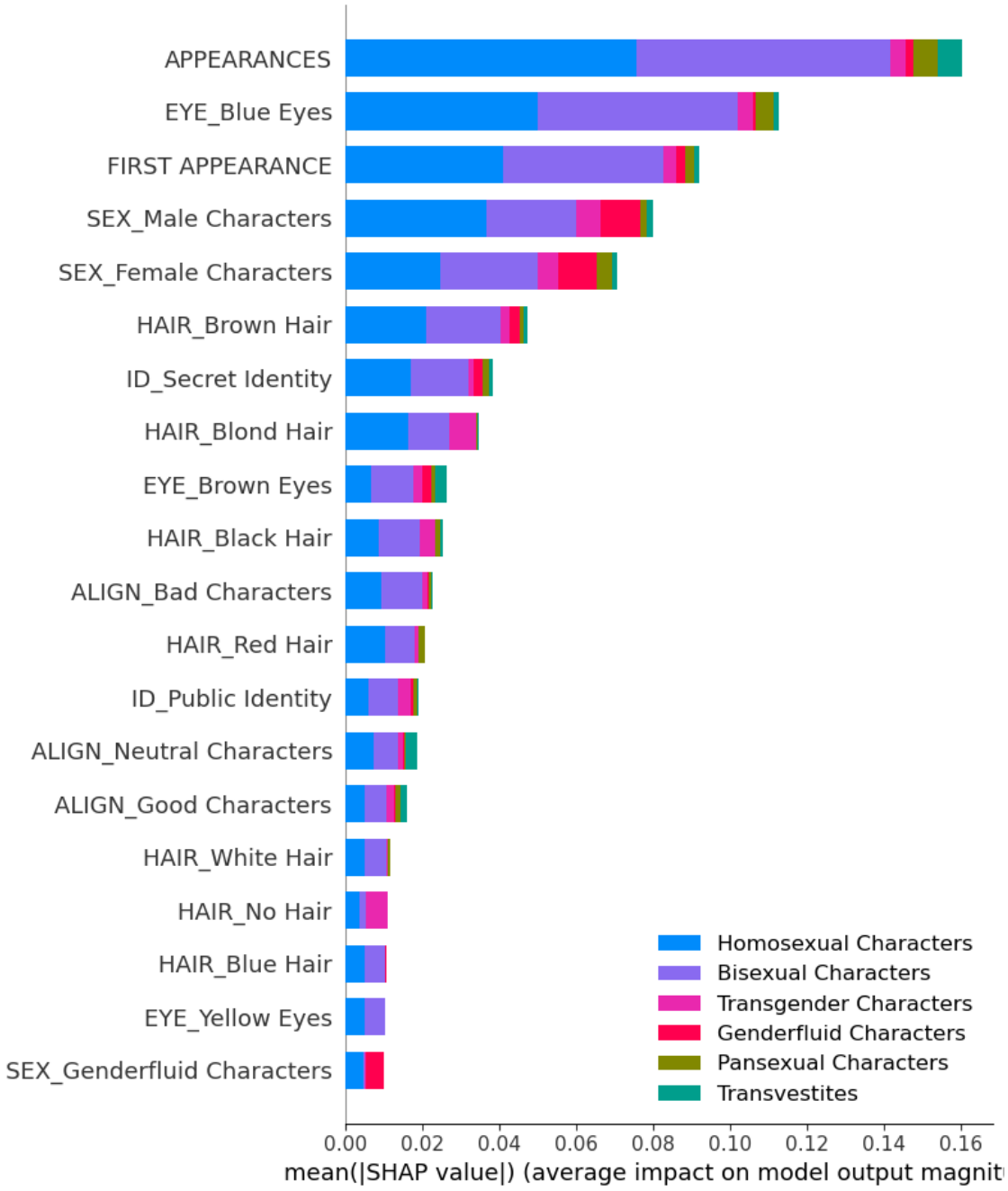

```
Created Waterfall plot for Combined EYE predictions
Creating Summary plot for Combined HAIR predictions

The figure layout has changed to tight
```



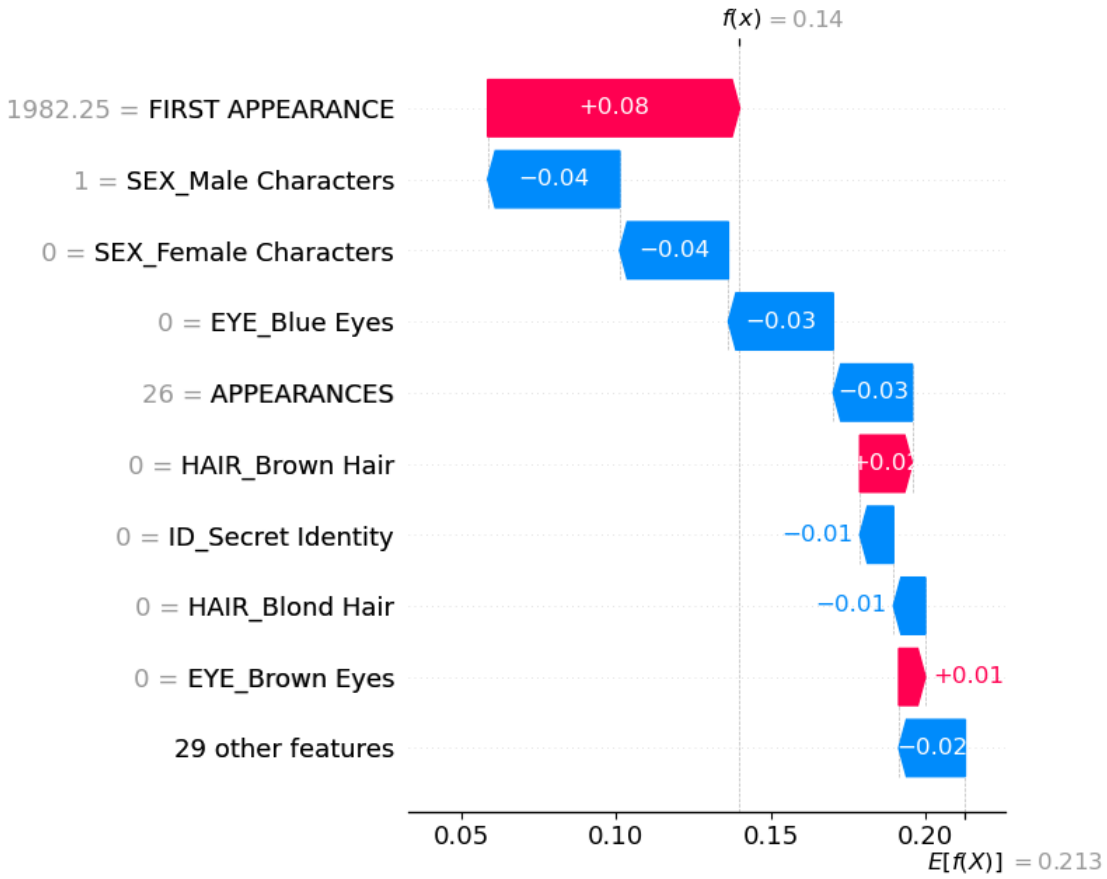

```
Created Summary plot for Combined HAIR predictions
Creating Waterfall plot for Combined HAIR predictions
```



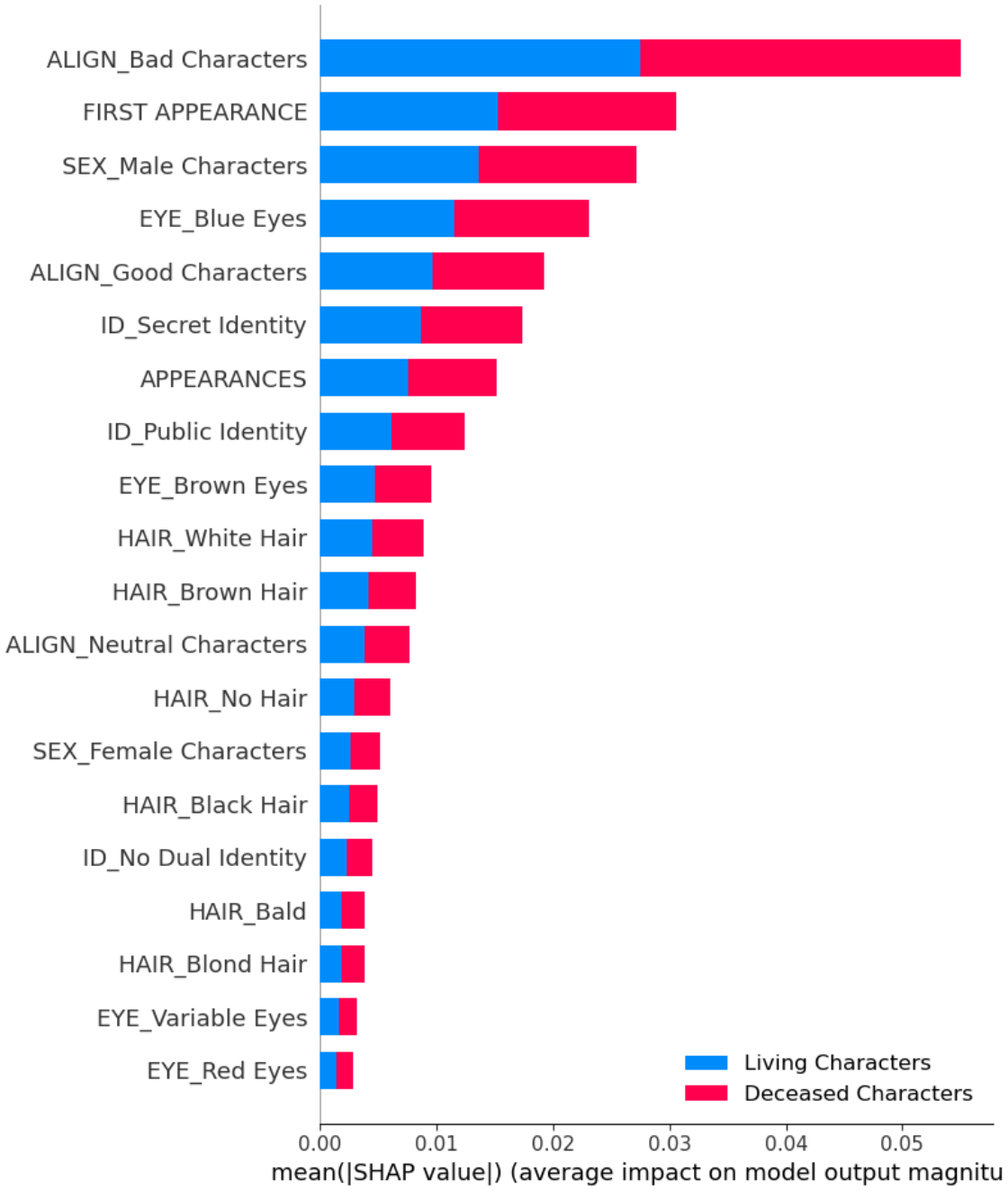

```
Created Waterfall plot for Combined HAIR predictions
Creating Summary plot for Combined GSM predictions

The figure layout has changed to tight
```



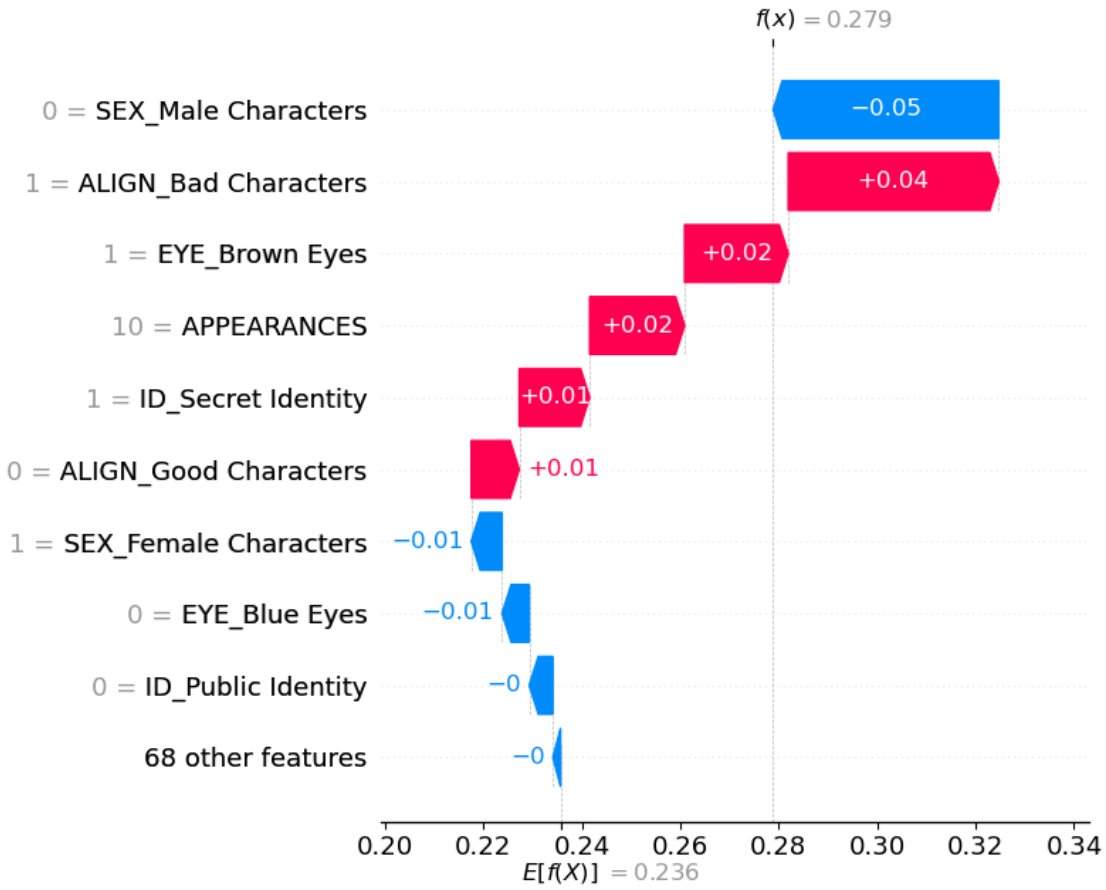

```
Created Summary plot for Combined GSM predictions
Creating Waterfall plot for Combined GSM predictions
```



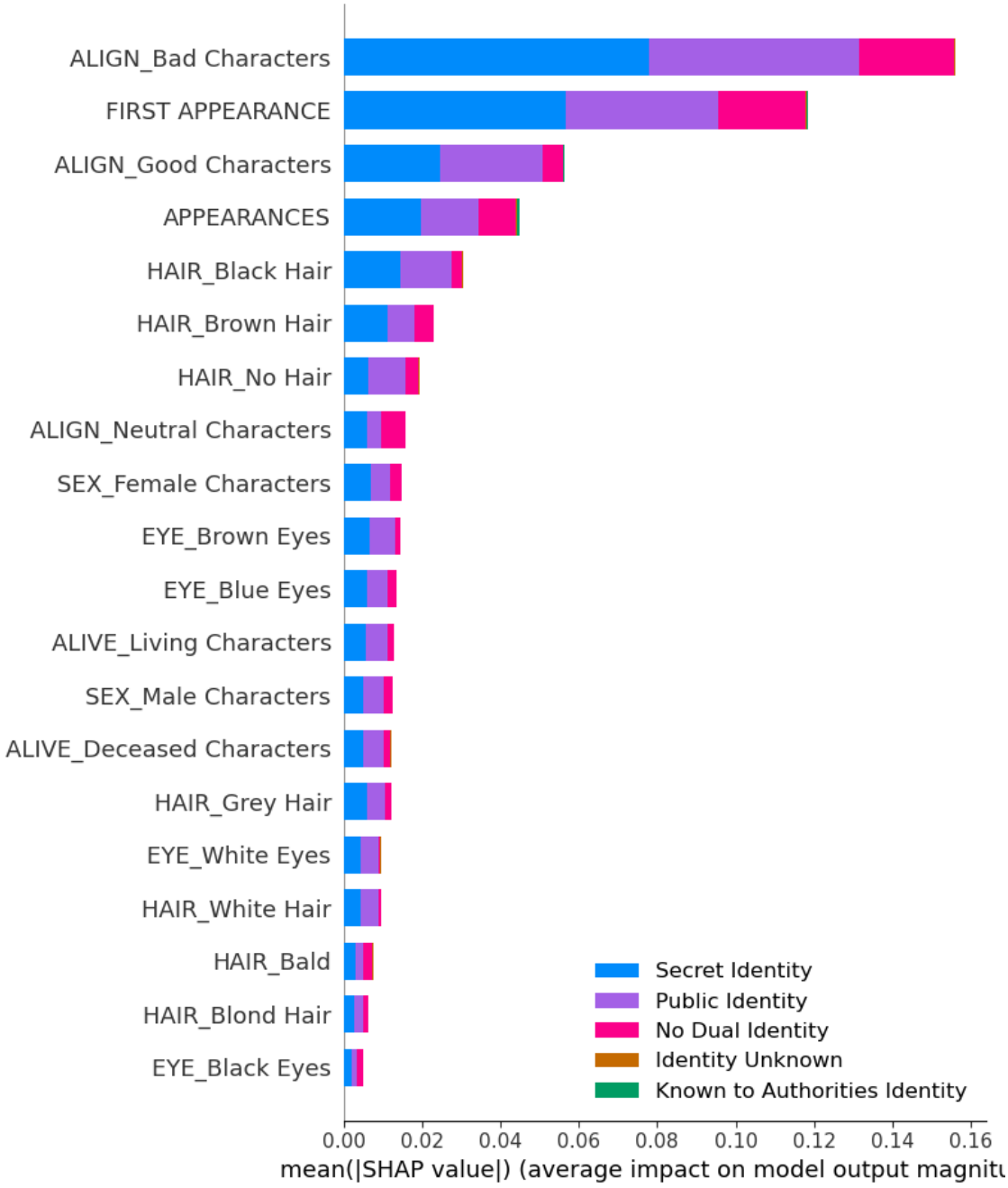

```
Created Waterfall plot for Combined GSM predictions
Creating Summary plot for Combined ALIVE predictions

The figure layout has changed to tight
```



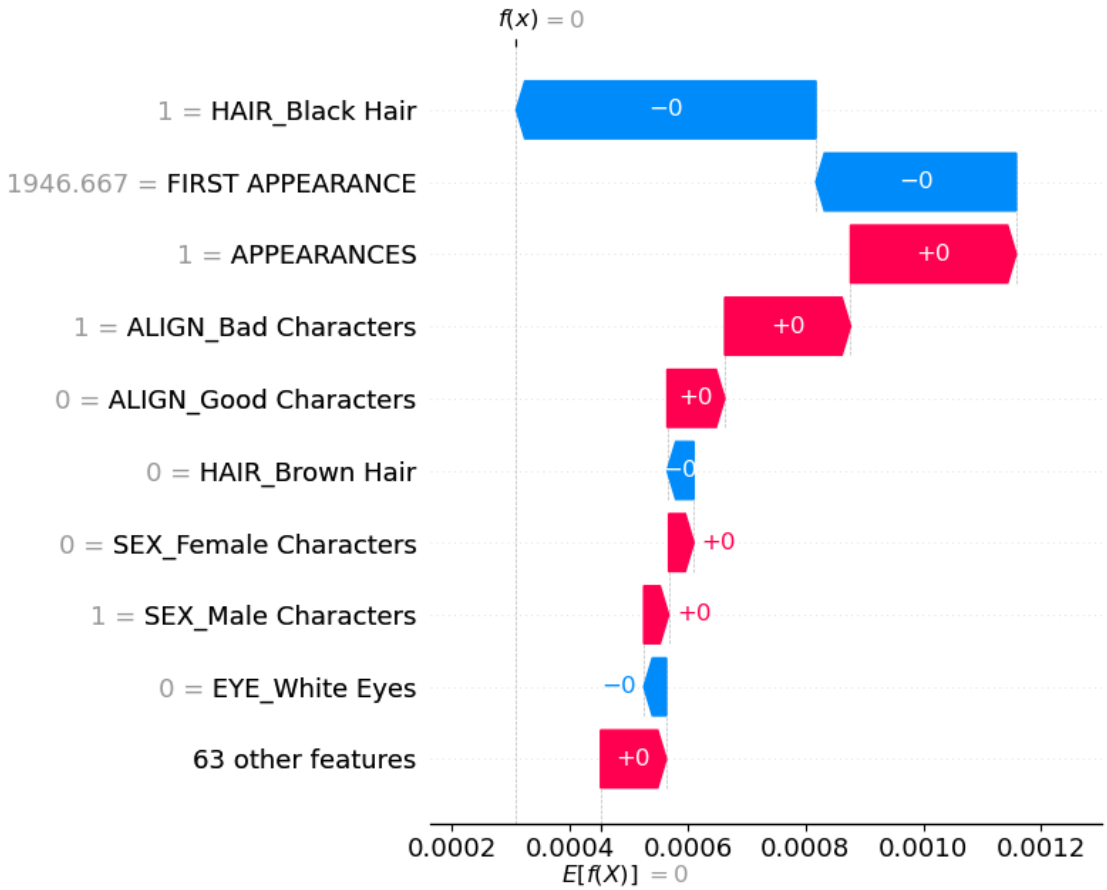

```
Created Summary plot for Combined ALIVE predictions
Creating Waterfall plot for Combined ALIVE predictions
```




```
Created Waterfall plot for Combined ALIVE predictions
Creating Summary plot for Combined ID predictions

The figure layout has changed to tight
```




```
Created Summary plot for Combined ID predictions
Creating Waterfall plot for Combined ID predictions
```

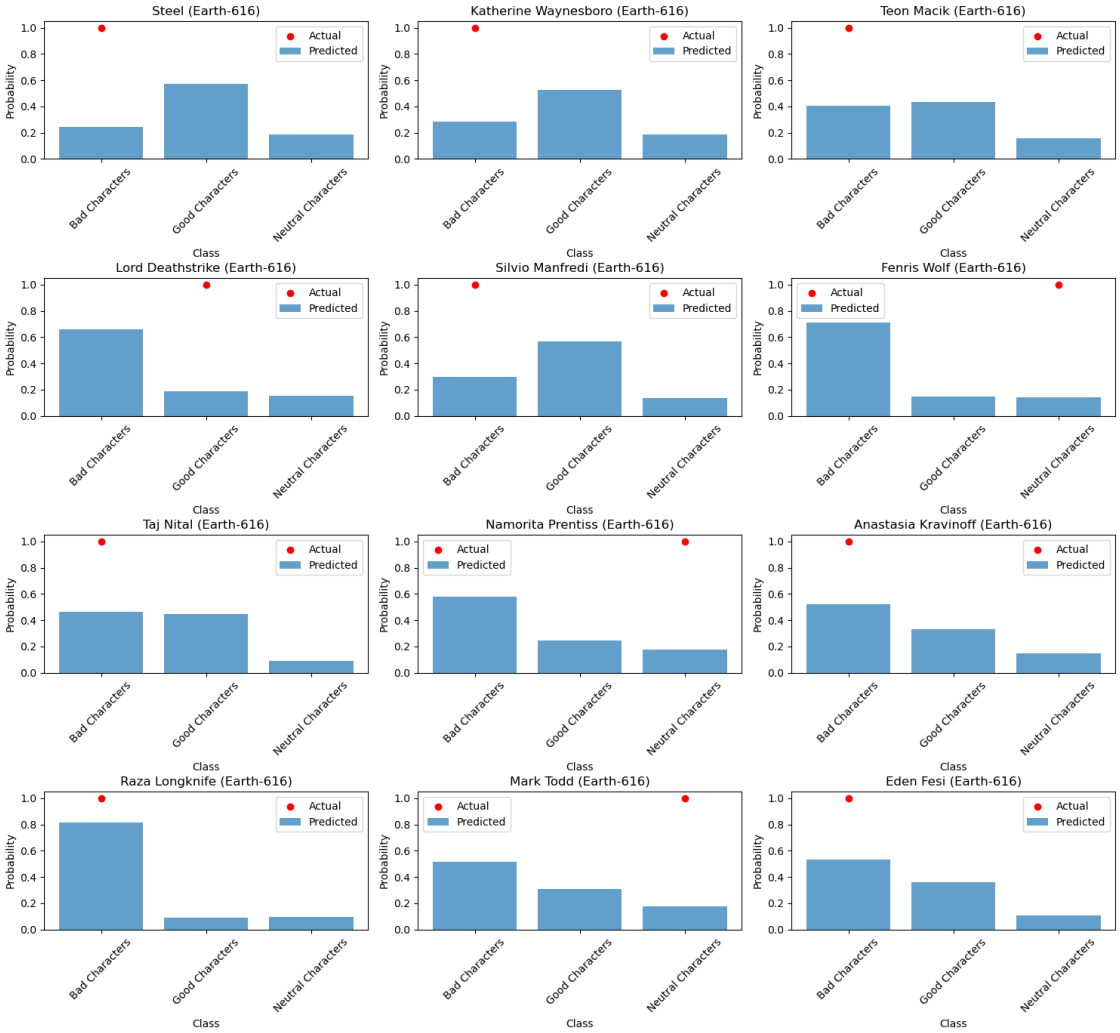



```
Created Waterfall plot for Combined ID predictions
Combined plots created
```

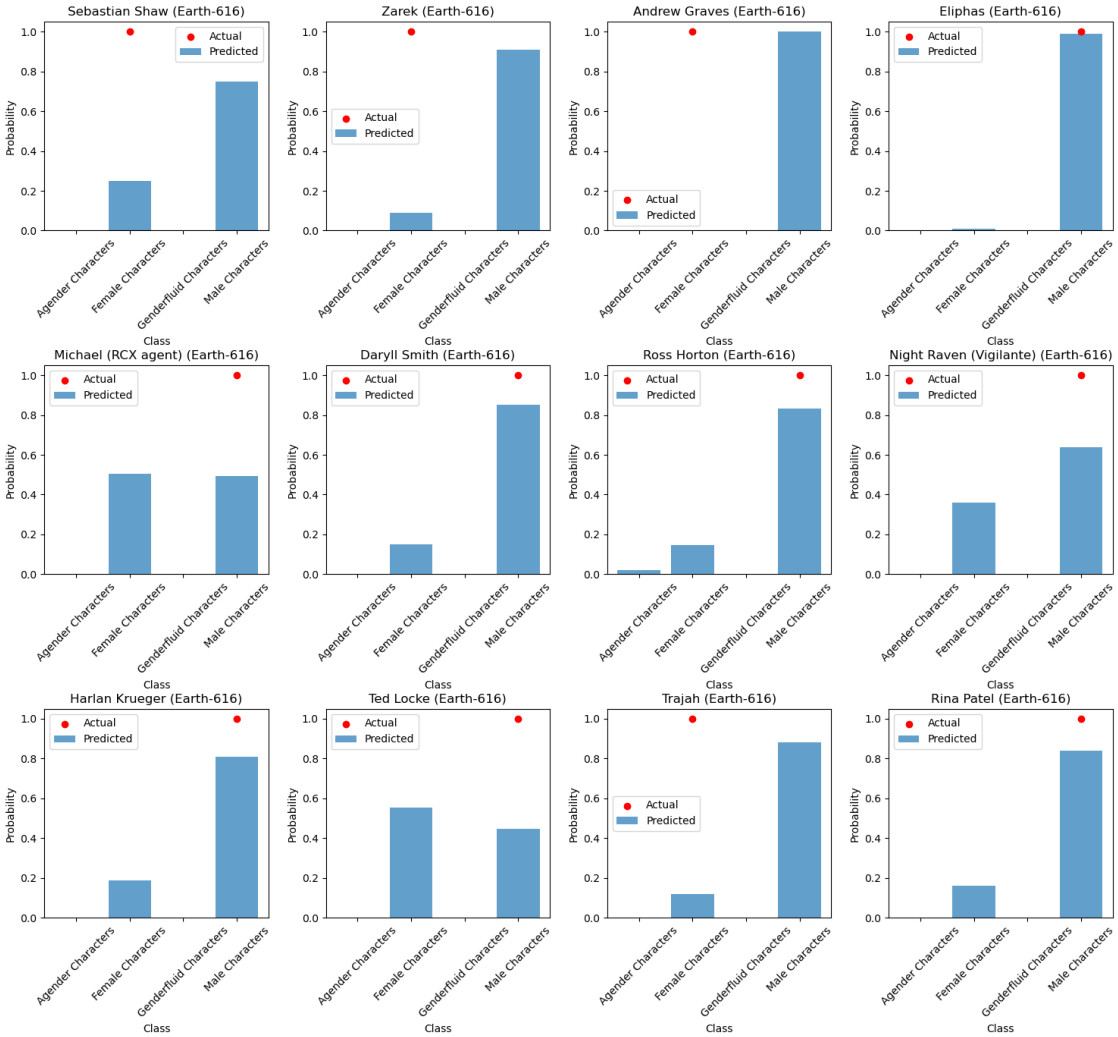
[ ]:
```
import math

# Loop through each dataset
for rf_model, X_train, y_train, X_test, y_test, target_feature, class_names in␣
 ↪rf_models_marvel:
    rf_model = RandomForestClassifier(**best_params_marvel[target_feature])
    rf_model.fit(X_train, y_train)


    # Select a random subset of instances from the test data
    num_instances = min((int(len(X_test) * 0.1) if len(X_test) > 250 else␣
 ↪len(X_test)), 12)
    selected_indices = np.random.choice(len(X_test), num_instances,␣
 ↪replace=False)
    X_test_subset = X_test.iloc[selected_indices]
    y_test_subset = y_test[selected_indices]
```

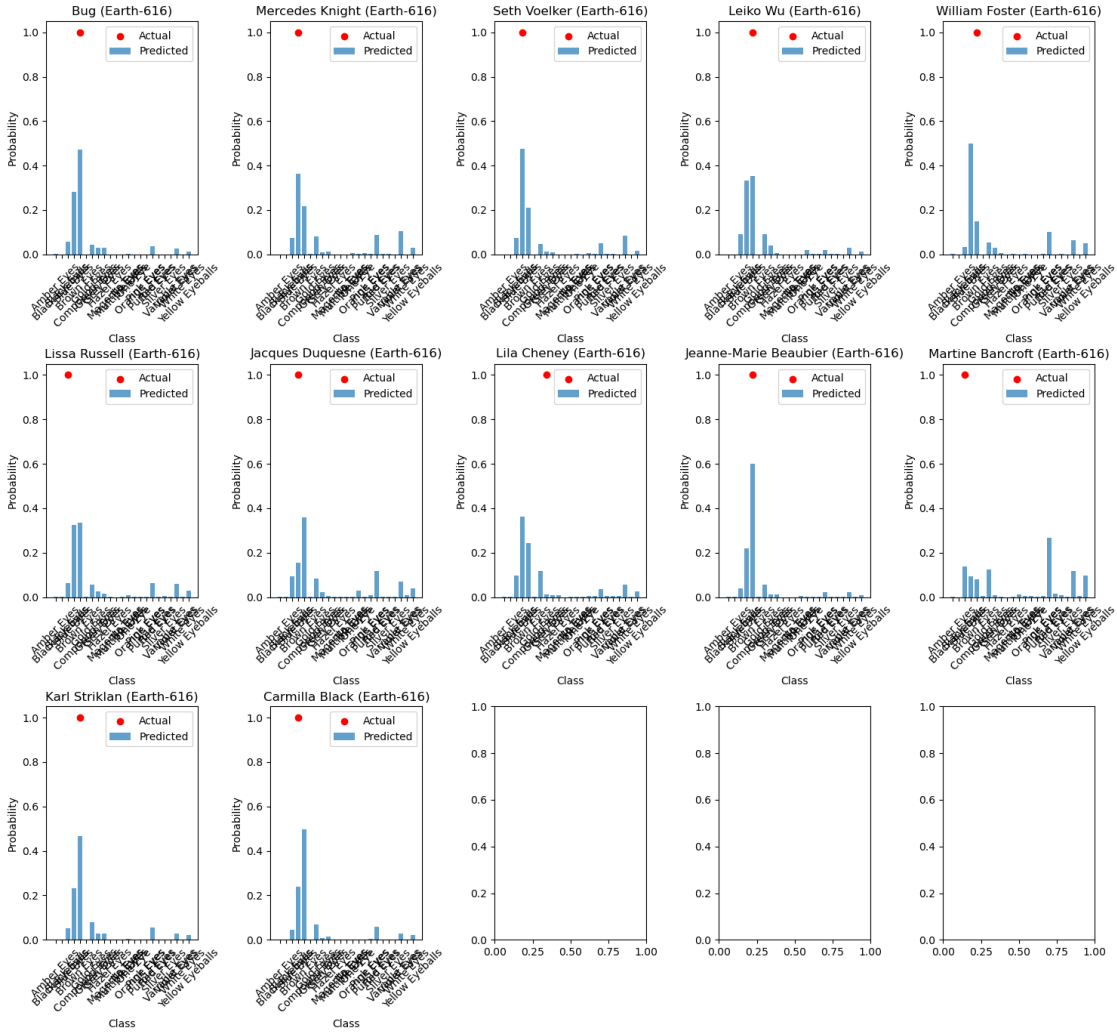
```
    # Get the predicted probabilities for the selected instances in the test␣
↪data
    predicted_probabilities = rf_model.predict_proba(X_test_subset)

    # Get the subset of class names corresponding to the number of classes
    subset_class_names = class_names[:predicted_probabilities.shape[1]]

    # Calculate the number of rows and columns for subplots
    num_instances_subset, num_classes = predicted_probabilities.shape
    num_columns = min(num_classes, 5)  # Limit the number of columns to avoid␣
↪too many subplots
    num_rows = math.ceil(num_instances_subset / num_columns)

    # Create a larger plot with subplots
    fig, axes = plt.subplots(num_rows, num_columns, figsize=(15, 15))
    fig.suptitle(f"Predicted Probabilities for {target_feature}", fontsize=16)

    # Loop through each instance and its corresponding predicted probabilities
    for i, (probs, ax) in enumerate(zip(predicted_probabilities, axes.
↪flatten())):
        character_name = str(marvel_names[selected_indices[i]])  # Convert to␣
↪string
        actual_class = y_test_subset[i]  # Get the actual class value

        # Plot the predicted probabilities as a bar chart
        ax.bar(subset_class_names, probs, label="Predicted", alpha=0.7)

        # Plot the actual class value as a point on top of the predicted␣
↪probabilities
        ax.scatter(actual_class, 1, color="red", label="Actual")

        ax.set_title(character_name)  # Set character name as the title
        ax.set_xlabel("Class")
        ax.set_ylabel("Probability")
        ax.set_xticks(range(len(subset_class_names)))  # Set tick positions
        ax.set_xticklabels(subset_class_names, rotation=45)  # Set tick labels
        ax.legend()  # Show legend

    # Adjust layout and display the plot
    plt.tight_layout()
    plt.subplots_adjust(top=0.9)
    plt.savefig(f"""figs/prob_charts/
↪Marvel_{target_feature}_{selected_indices}_probabilities.png""")
    plt.show()
```

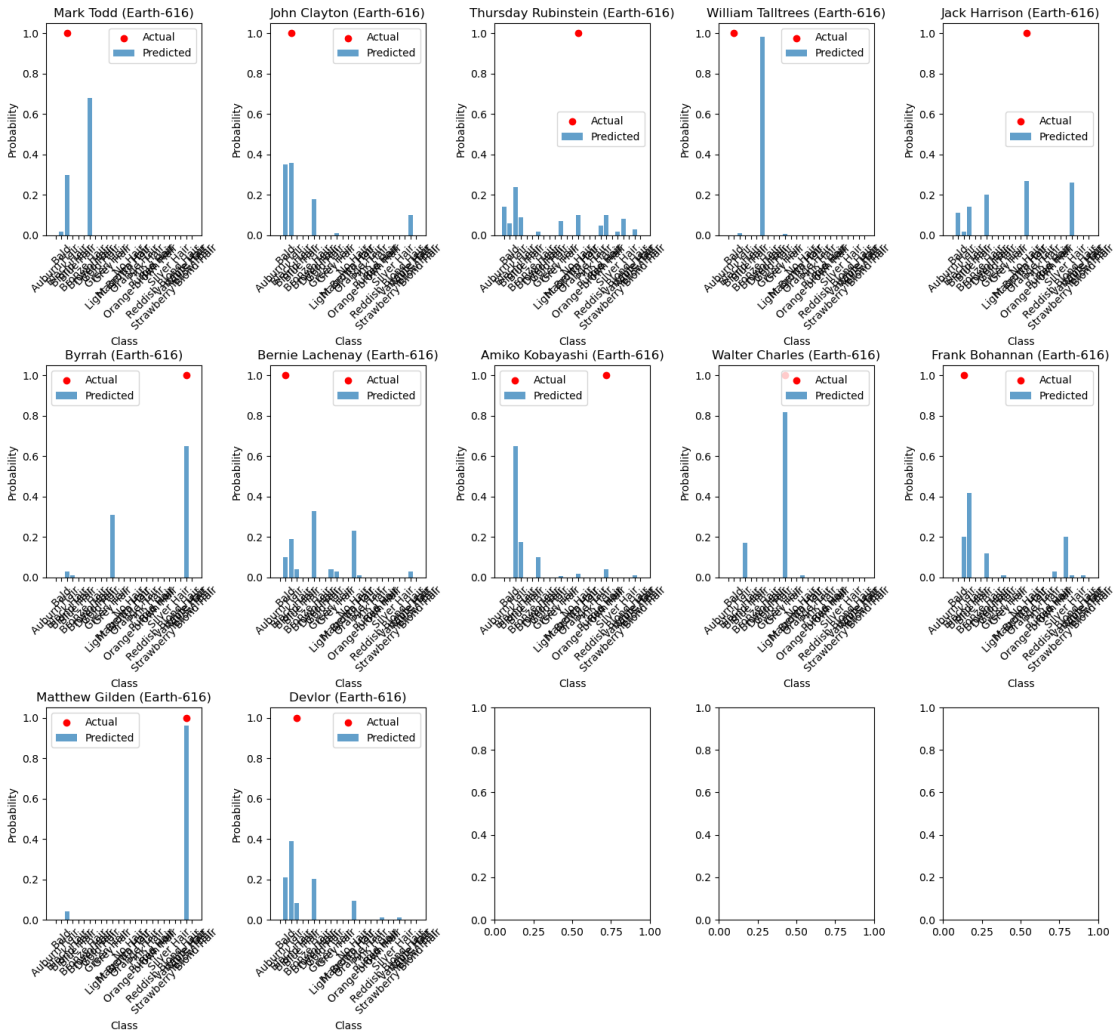
Predicted Probabilities for ALIGN

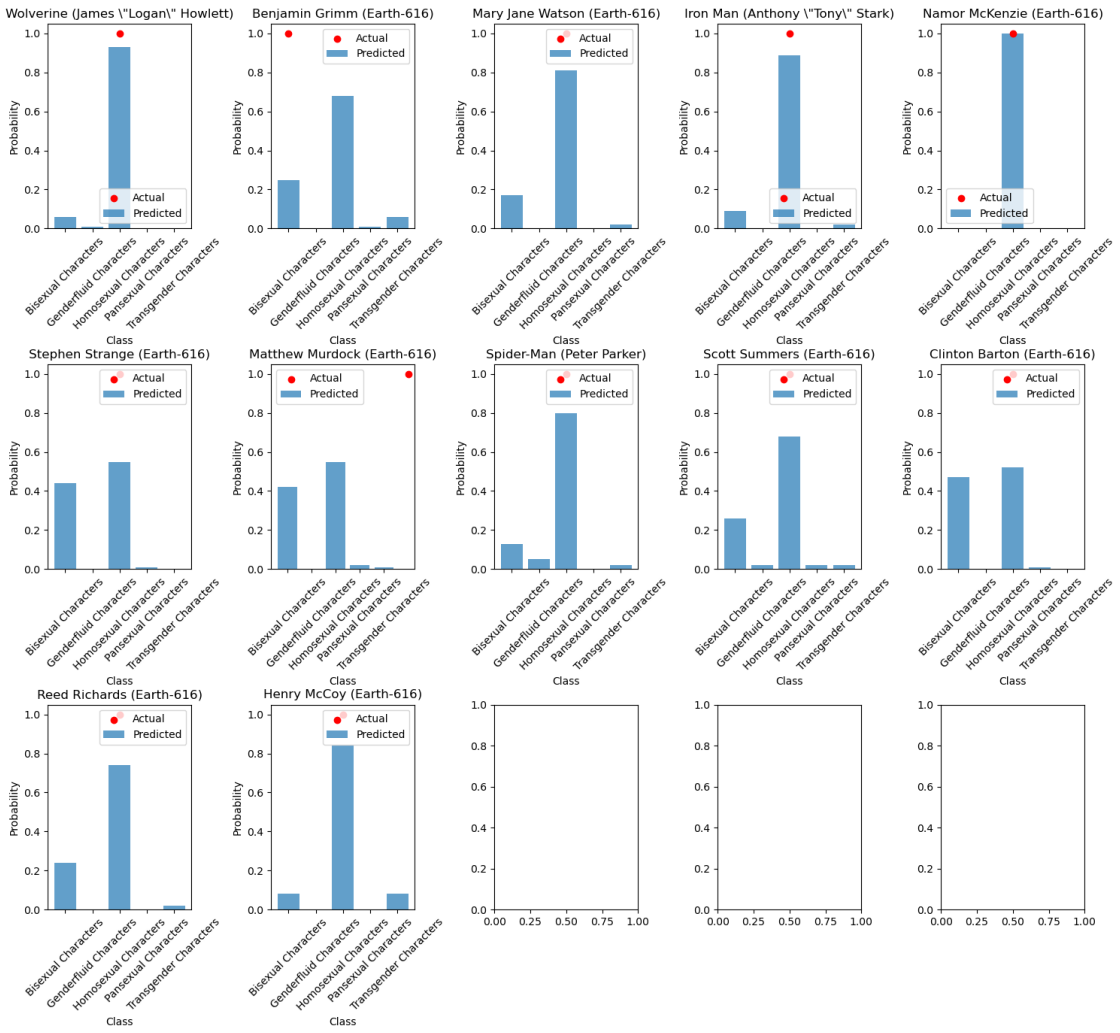


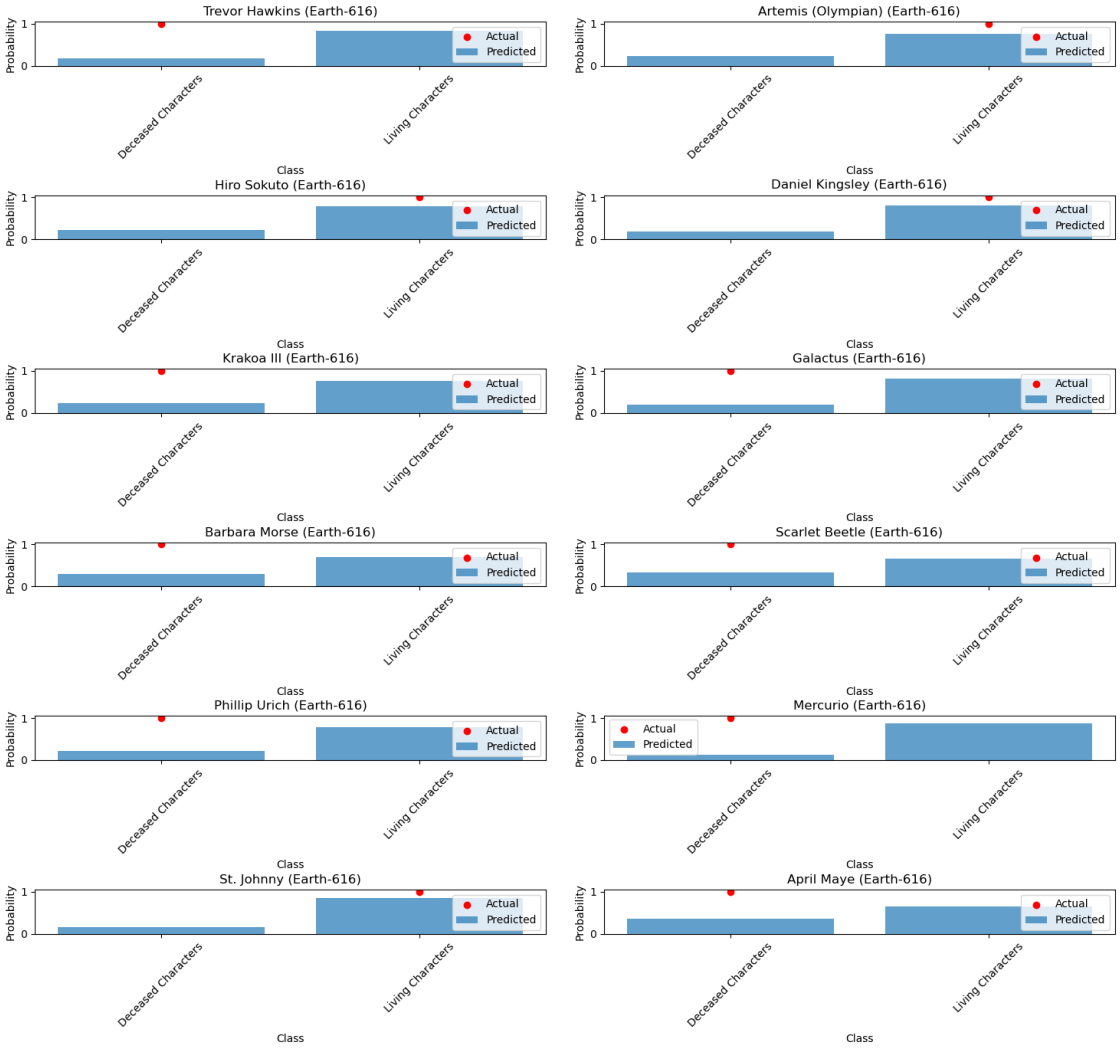
Predicted Probabilities for SEX

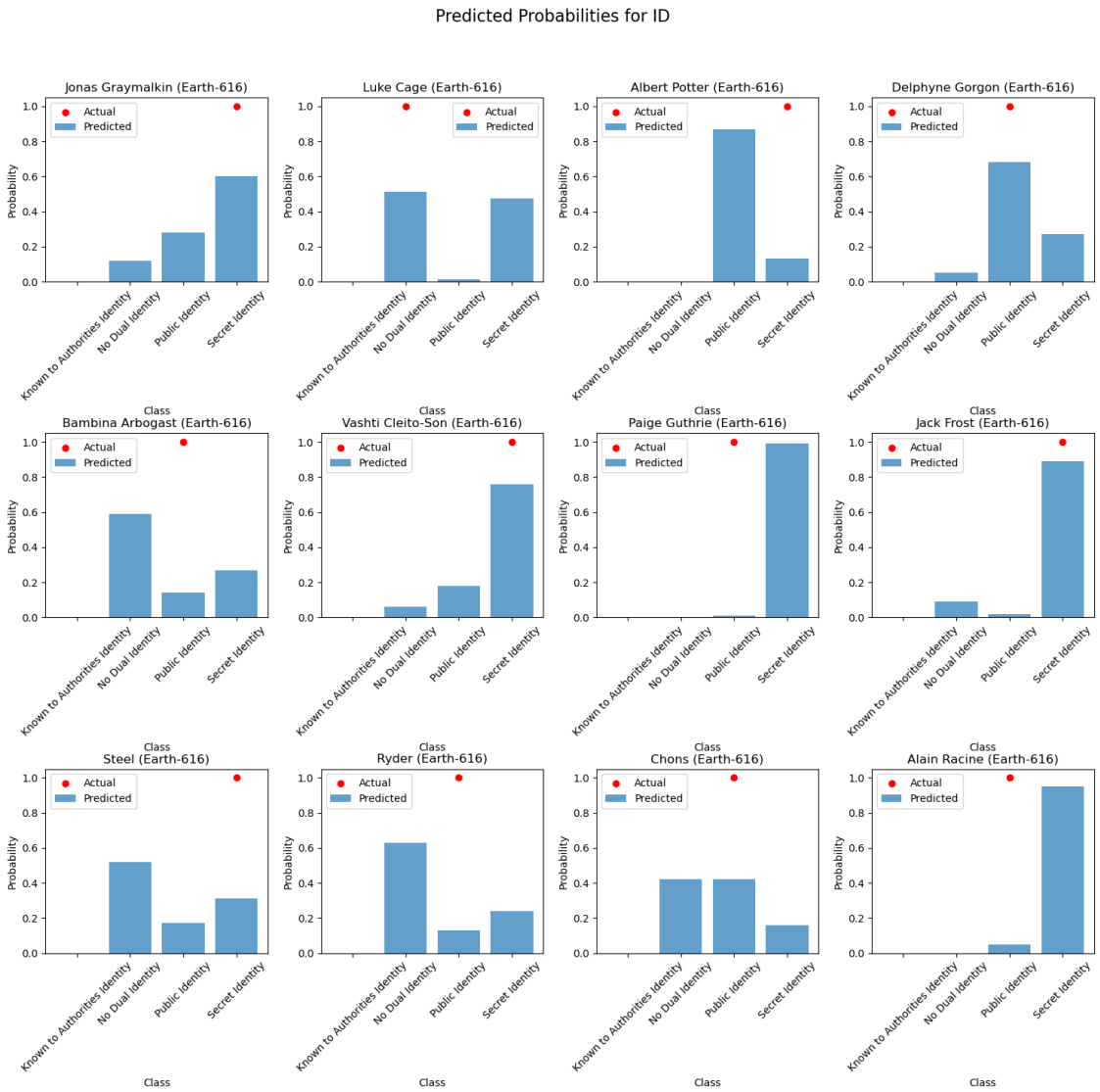



Predicted Probabilities for EYE
Bug (Earth-616)
Mercedes Knight (Earth-616)
Seth Voelker (Earth-616)
Leiko Wu (Earth-616)
William Foster (Earth-616)
Lissa Russell (Earth-616)
Jacques Duquesne (Earth-616)
Lila Cheney (Earth-616)
Jeanne-Marie Beaubier (Earth-616)
Martine Bancroft (Earth-616)
Karl Striklan (Earth-616)
Carmilla Black (Earth-616)
Actual
Predicted
Probability
Class
Amber Eyes
Black Eyes
Blue Eyes
Brown Eyes
Green Eyes
Orange Eyes
Pink Eyes
Purple Eyes
Red Eyes
Variable Eyes
White Eyes
Yellow Eyeballs


Predicted Probabilities for HAIR
Mark Todd (Earth-616)
John Clayton (Earth-616)
Thursday Rubinstein (Earth-616)
William Talltrees (Earth-616)
Jack Harrison (Earth-616)
Byrrah (Earth-616)
Bernie Lachenay (Earth-616)
Amiko Kobayashi (Earth-616)
Walter Charles (Earth-616)
Frank Bohannan (Earth-616)
Matthew Gilden (Earth-616)
Devlor (Earth-616)
Actual
Predicted
Probability
Class

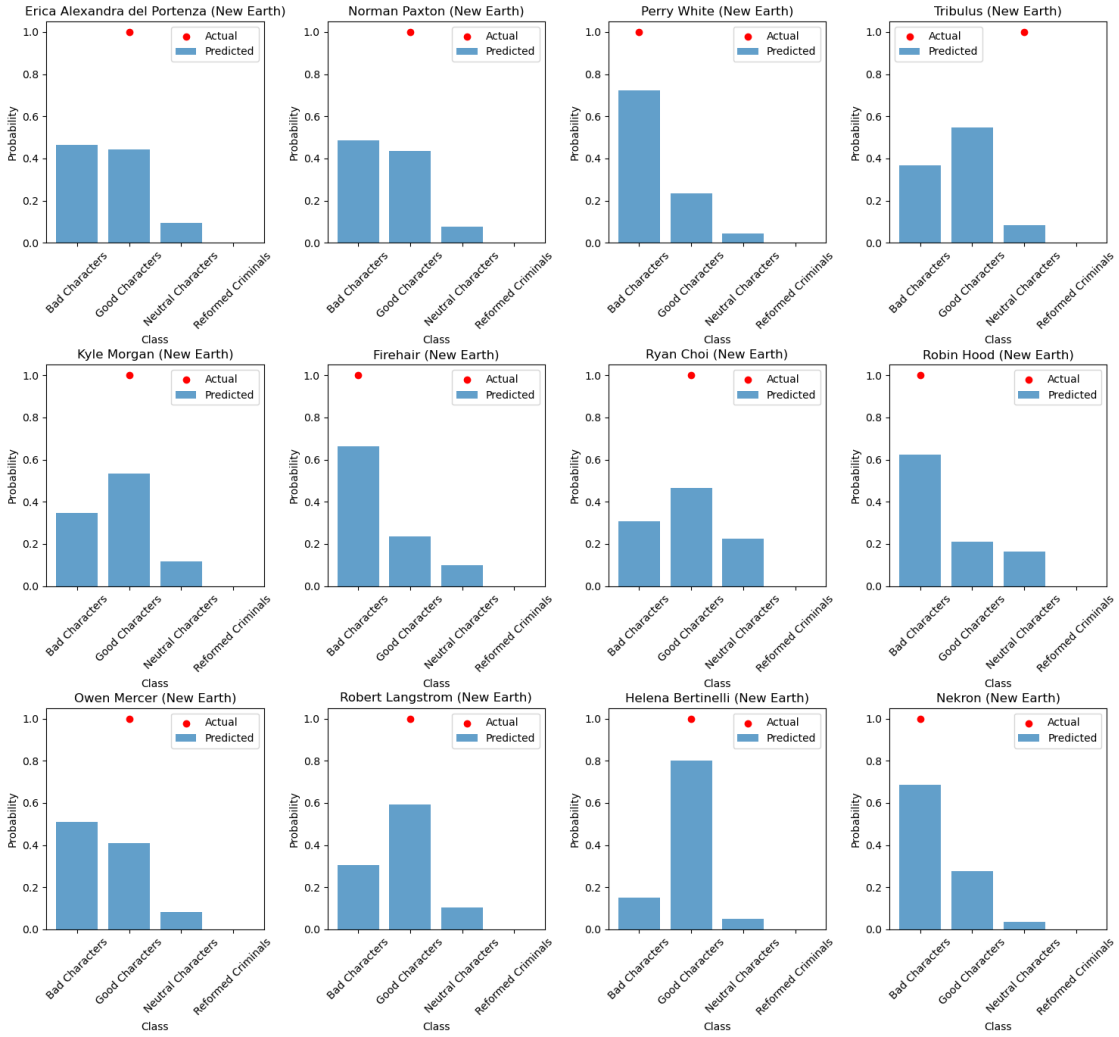

Predicted Probabilities for GSM
Wolverine (James \"Logan\" Howlett)
Benjamin Grimm (Earth-616)
Mary Jane Watson (Earth-616)
Iron Man (Anthony \"Tony\" Stark)
Namor McKenzie (Earth-616)
Stephen Strange (Earth-616)
Matthew Murdock (Earth-616)
Spider-Man (Peter Parker)
Scott Summers (Earth-616)
Clinton Barton (Earth-616)
Reed Richards (Earth-616)
Henry McCoy (Earth-616)
Actual
Predicted
Probability
Class
Bisexual Characters
Genderfluid Characters
Homosexual Characters
Pansexual Characters
Transgender Characters


Predicted Probabilities for ALIVE
Trevor Hawkins (Earth-616)
Artemis (Olympian) (Earth-616)
Hiro Sokuto (Earth-616)
Daniel Kingsley (Earth-616)
Krakoa III (Earth-616)
Galactus (Earth-616)
Barbara Morse (Earth-616)
Scarlet Beetle (Earth-616)
Phillip Urich (Earth-616)
Mercurio (Earth-616)
St. Johnny (Earth-616)
April Maye (Earth-616)
Probability
Class
Deceased Characters
Living Characters
Actual
Predicted

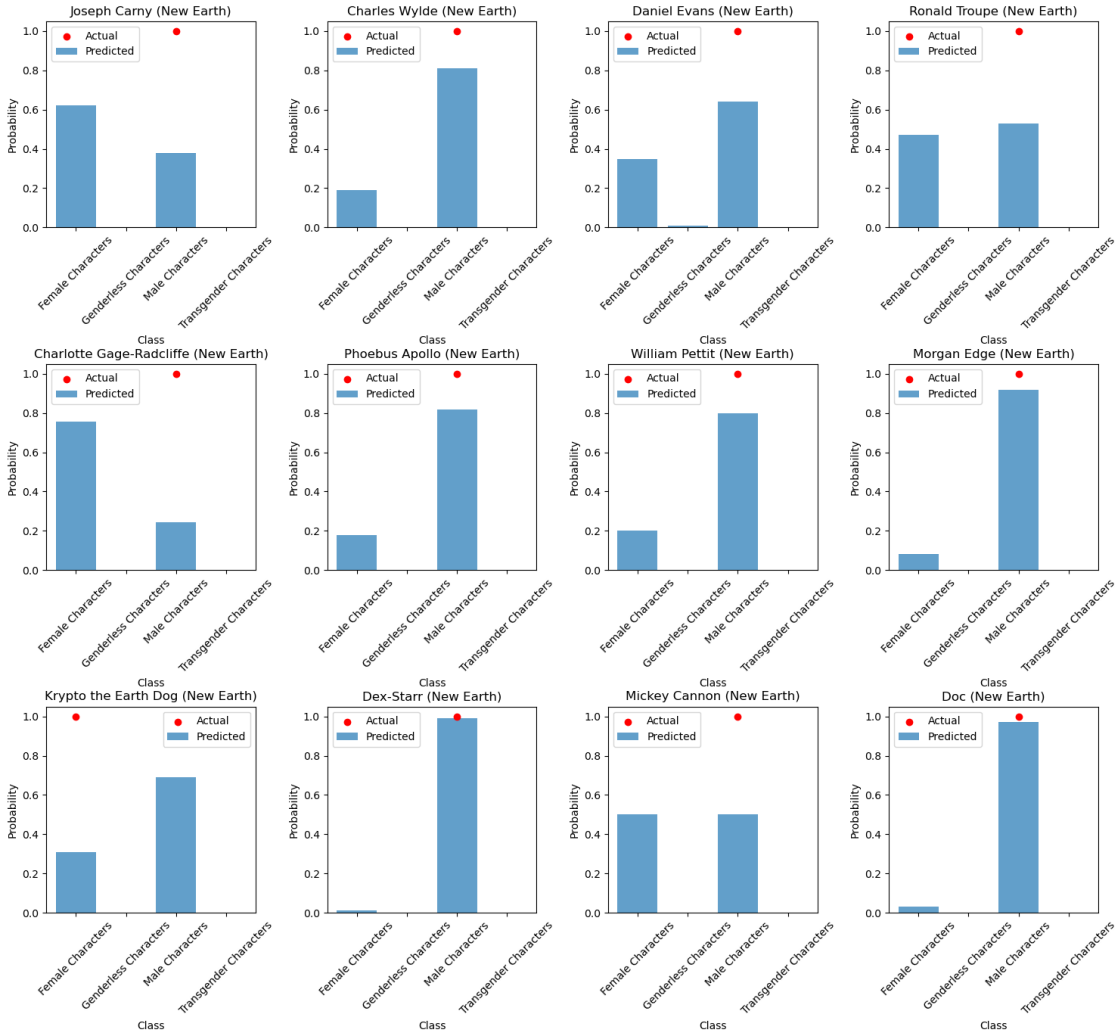


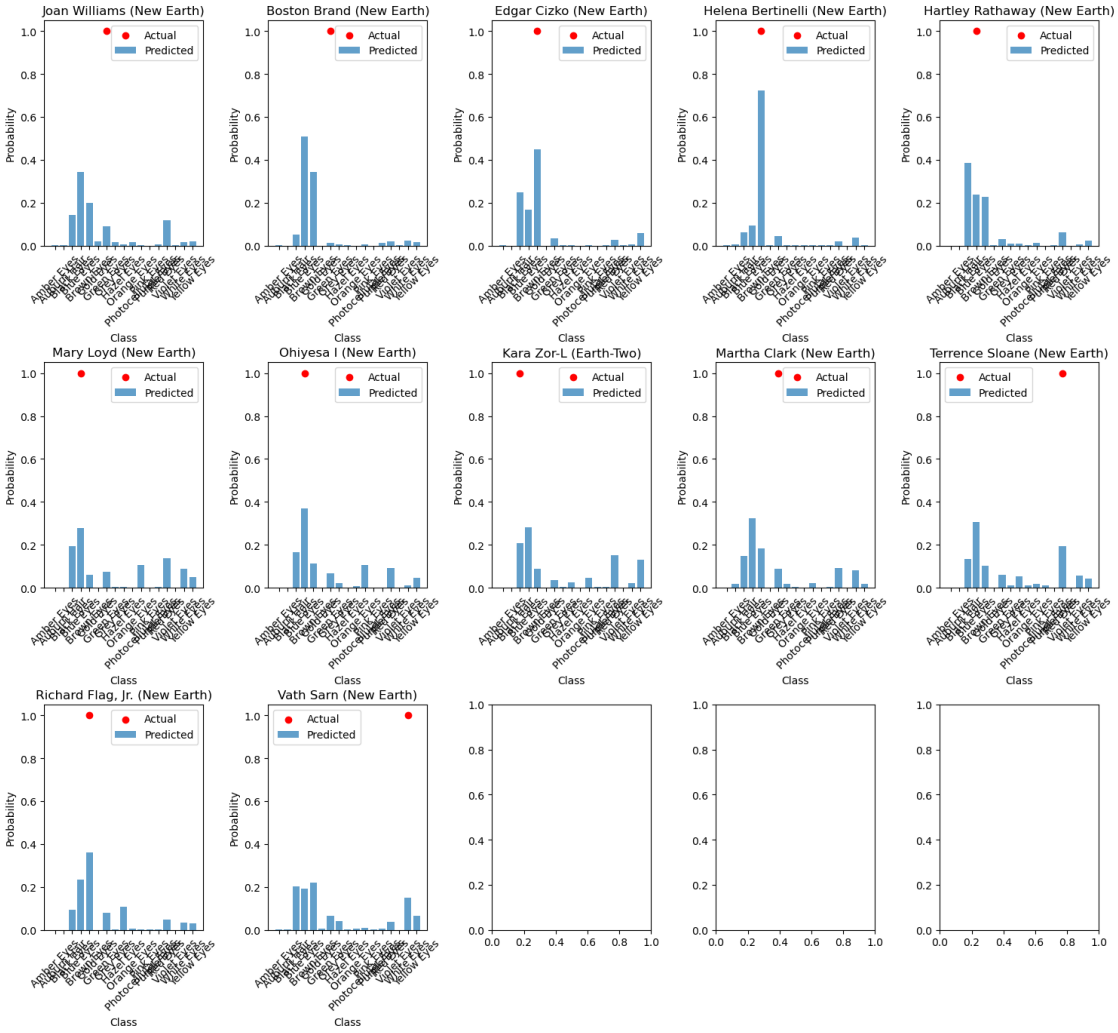
[ ]:
```
# Loop through each dataset
for rf_model, X_train, y_train, X_test, y_test, target_feature, class_names in␣
↪rf_models_dc:
    rf_model = RandomForestClassifier(**best_params_dc[target_feature])
    rf_model.fit(X_train, y_train)

    # Select a random subset of instances from the test data
    num_instances = min((int(len(X_test) * 0.1) if len(X_test) > 250 else␣
↪len(X_test)), 12)
    selected_indices = np.random.choice(len(X_test), num_instances,␣
↪replace=False)
    X_test_subset = X_test.iloc[selected_indices]
    y_test_subset = y_test[selected_indices]
```

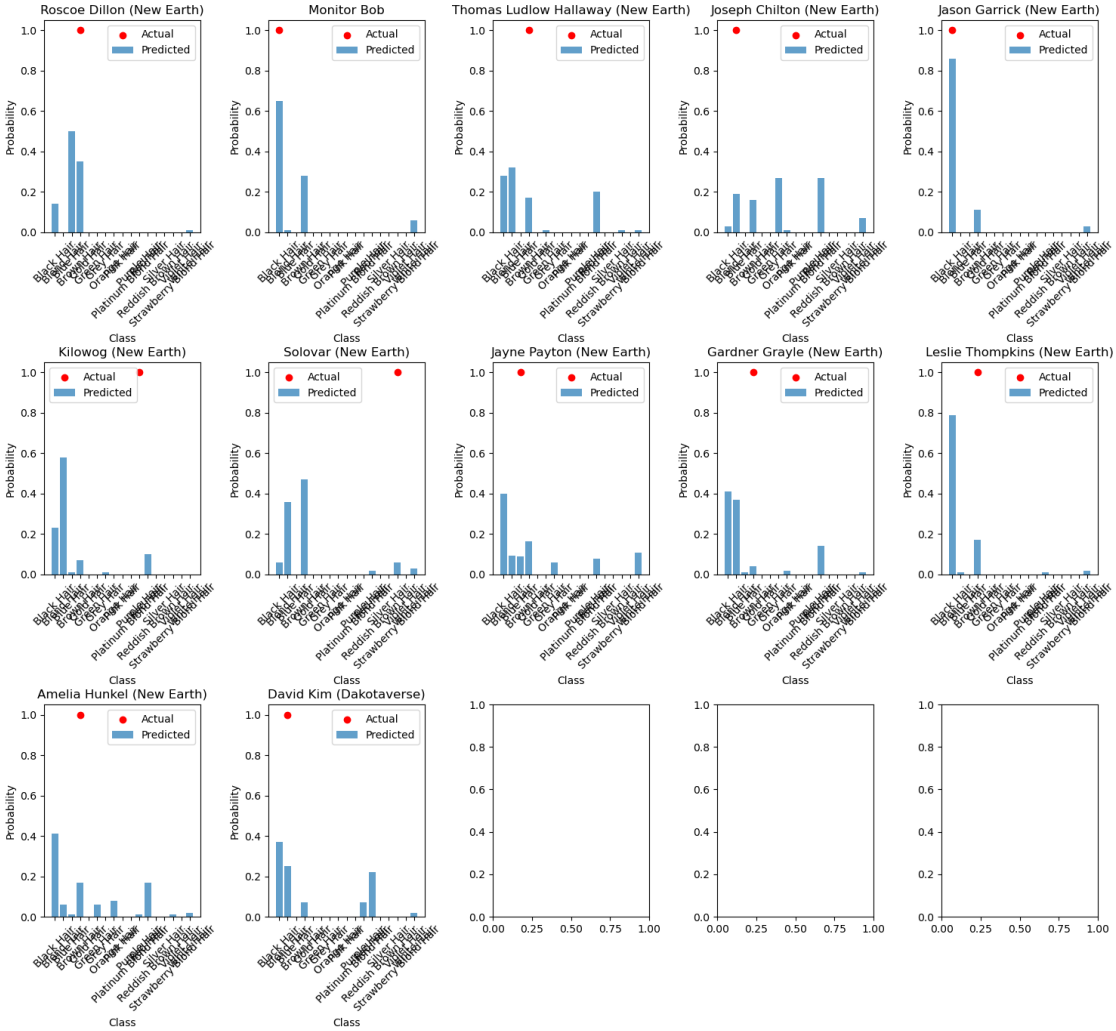
```
   # Get the predicted probabilities for the selected instances in the test␣
↪data
   predicted_probabilities = rf_model.predict_proba(X_test_subset)

   # Get the subset of class names corresponding to the number of classes
   subset_class_names = class_names[:predicted_probabilities.shape[1]]

   # Calculate the number of rows and columns for subplots
   num_instances_subset, num_classes = predicted_probabilities.shape
   num_columns = min(num_classes, 5)  # Limit the number of columns to avoid␣
↪too many subplots
   num_rows = math.ceil(num_instances_subset / num_columns)

   # Create a larger plot with subplots
   fig, axes = plt.subplots(num_rows, num_columns, figsize=(15, 15))
   fig.suptitle(f"Predicted Probabilities for {target_feature}", fontsize=16)

   # Loop through each instance and its corresponding predicted probabilities
   for i, (probs, ax) in enumerate(zip(predicted_probabilities, axes.
↪flatten())):
       character_name = str(dc_names[selected_indices[i]])  # Convert to string
       actual_class = y_test_subset[i]  # Get the actual class value

       # Plot the predicted probabilities as a bar chart
       ax.bar(subset_class_names, probs, label="Predicted", alpha=0.7)

       # Plot the actual class value as a point on top of the predicted␣
↪probabilities
       ax.scatter(actual_class, 1, color="red", label="Actual")

       ax.set_title(character_name)  # Set character name as the title
       ax.set_xlabel("Class")
       ax.set_ylabel("Probability")
       ax.set_xticks(range(len(subset_class_names)))  # Set tick positions
       ax.set_xticklabels(subset_class_names, rotation=45)  # Set tick labels
       ax.legend()  # Show legend

   # Adjust layout and display the plot
   plt.tight_layout()
   plt.subplots_adjust(top=0.9)
   plt.savefig(f"""figs/prob_charts/
↪DC_{target_feature}_{selected_indices}_probabilities.png""")
   plt.show()
```


Predicted Probabilities for ALIGN
Erica Alexandra del Portenza (New Earth)
Norman Paxton (New Earth)
Perry White (New Earth)
Tribulus (New Earth)
Kyle Morgan (New Earth)
Firehair (New Earth)
Ryan Choi (New Earth)
Robin Hood (New Earth)
Owen Mercer (New Earth)
Robert Langstrom (New Earth)
Helena Bertinelli (New Earth)
Nekron (New Earth)
Actual
Predicted
Probability
Class
Bad Characters
Good Characters
Neutral Characters
Reformed Criminals

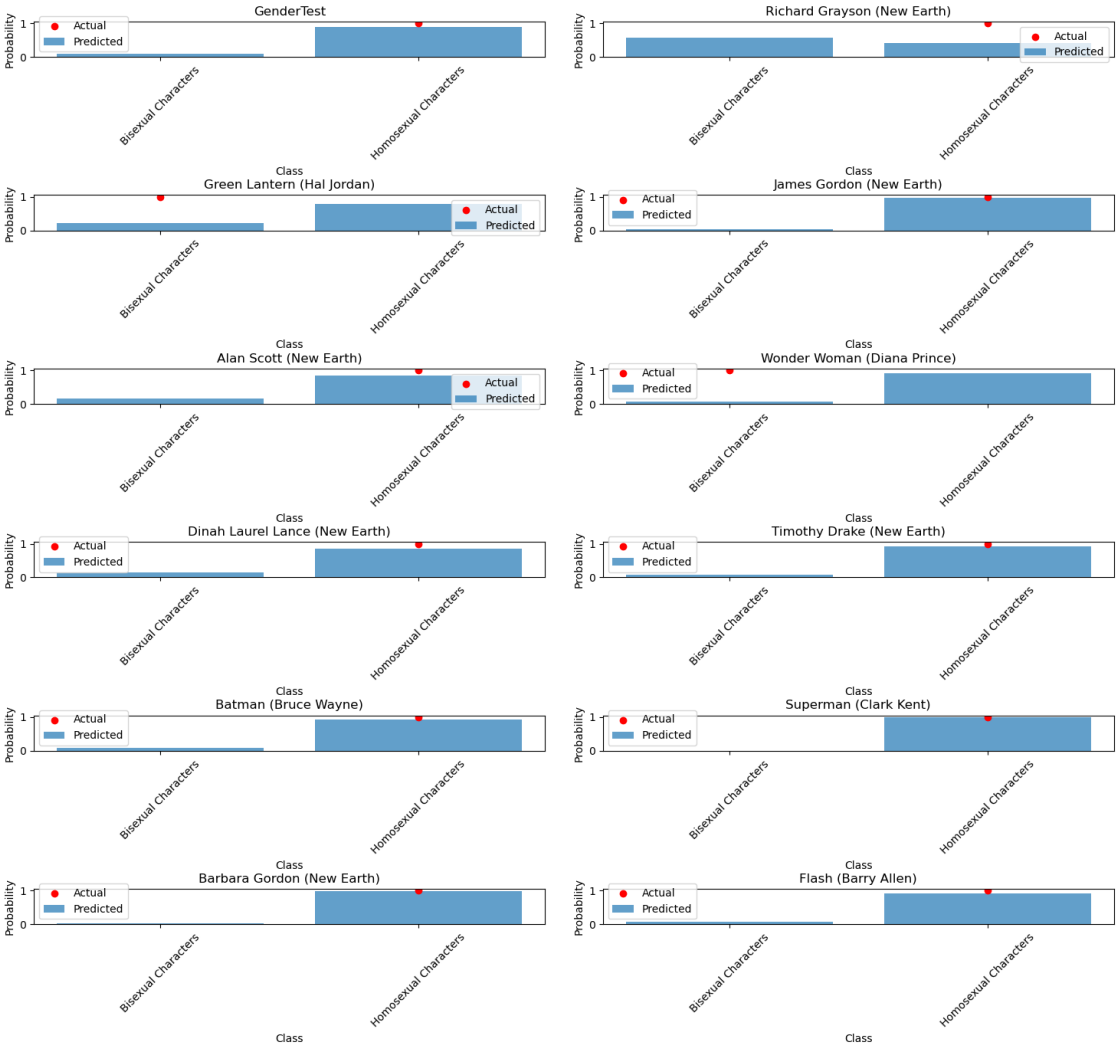
Predicted Probabilities for SEX



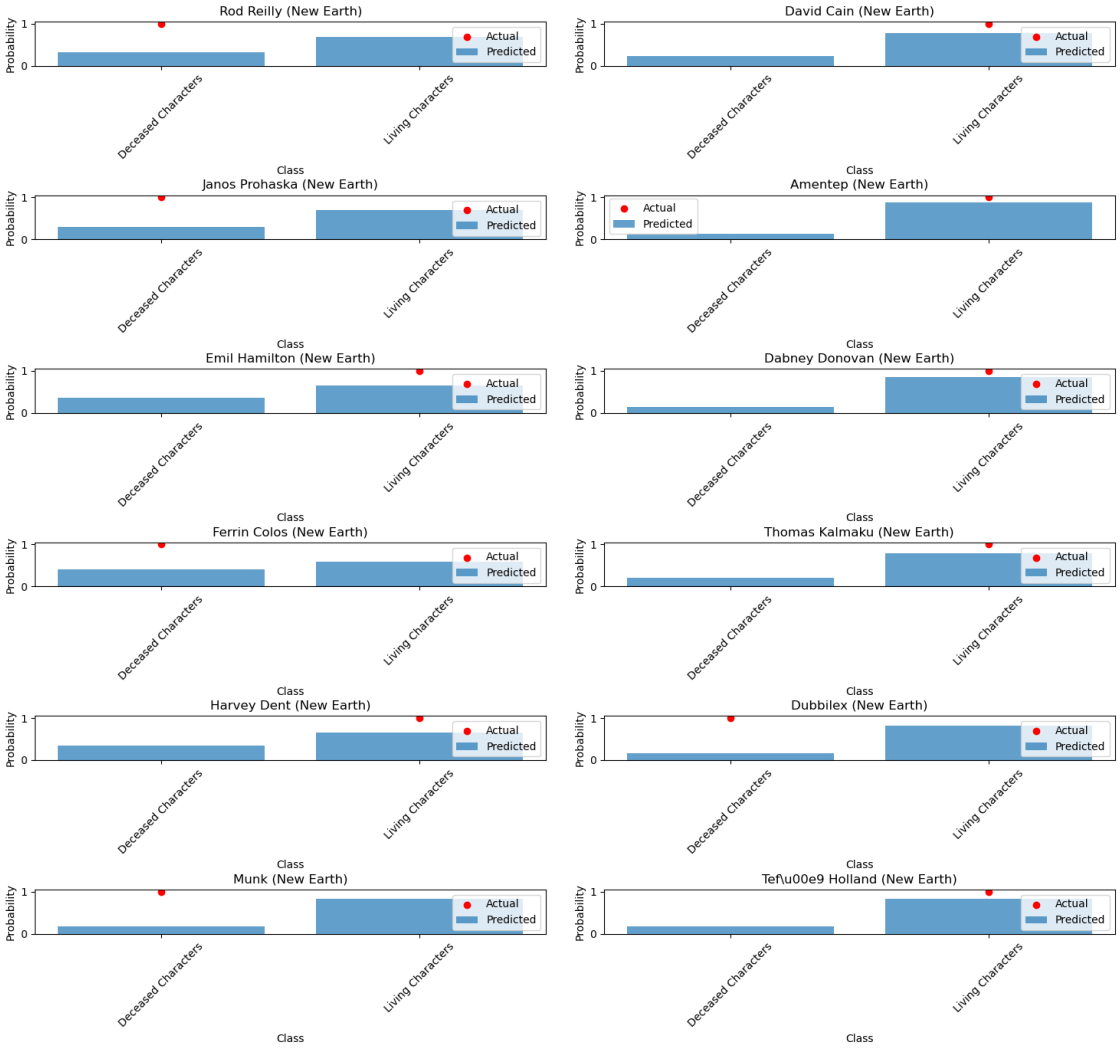

Predicted Probabilities for EYE
Joan Williams (New Earth)
Boston Brand (New Earth)
Edgar Cizko (New Earth)
Helena Bertinelli (New Earth)
Hartley Rathaway (New Earth)
Mary Loyd (New Earth)
Ohiyesa I (New Earth)
Kara Zor-L (Earth-Two)
Martha Clark (New Earth)
Terrence Sloane (New Earth)
Richard Flag, Jr. (New Earth)
Vath Sarn (New Earth)
Actual
Predicted
Probability
Class

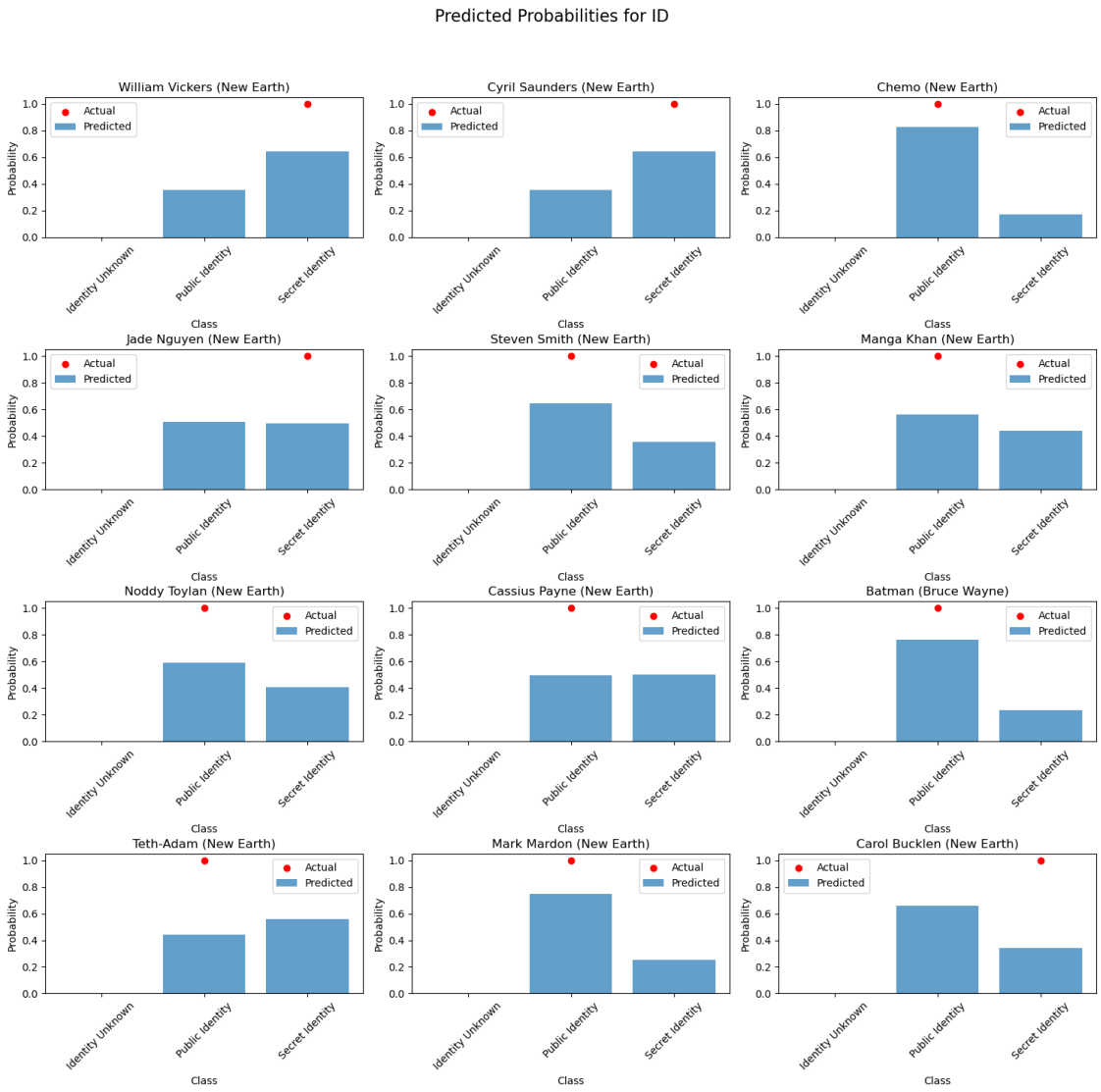

Predicted Probabilities for HAIR
Roscoe Dillon (New Earth)
Monitor Bob
Thomas Ludlow Hallaway (New Earth)
Joseph Chilton (New Earth)
Jason Garrick (New Earth)
Kilowog (New Earth)
Solovar (New Earth)
Jayne Payton (New Earth)
Gardner Grayle (New Earth)
Leslie Thompkins (New Earth)
Amelia Hunkel (New Earth)
David Kim (Dakotaverse)
Actual
Predicted
Probability
Class

Predicted Probabilities for GSM



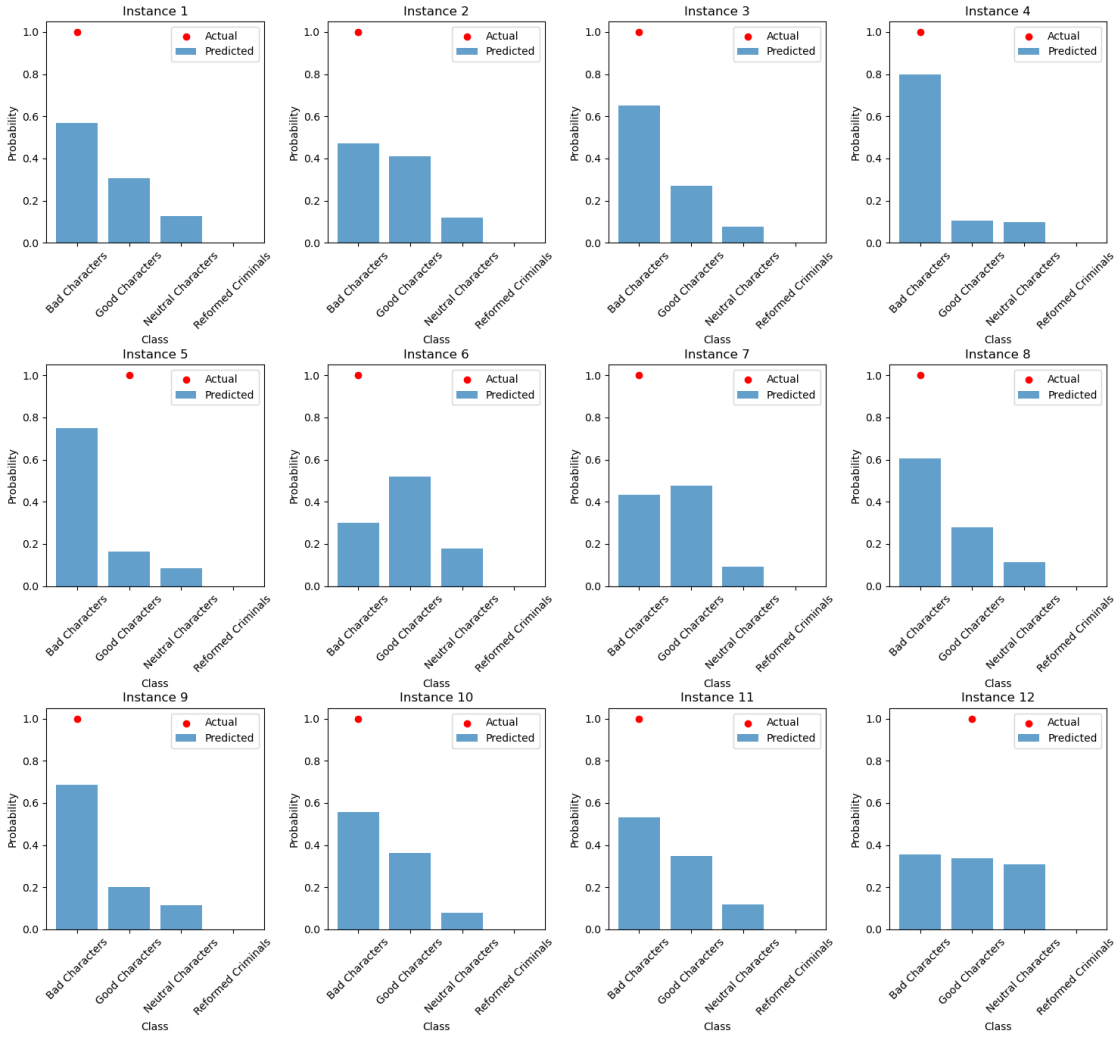
Predicted Probabilities for ALIVE

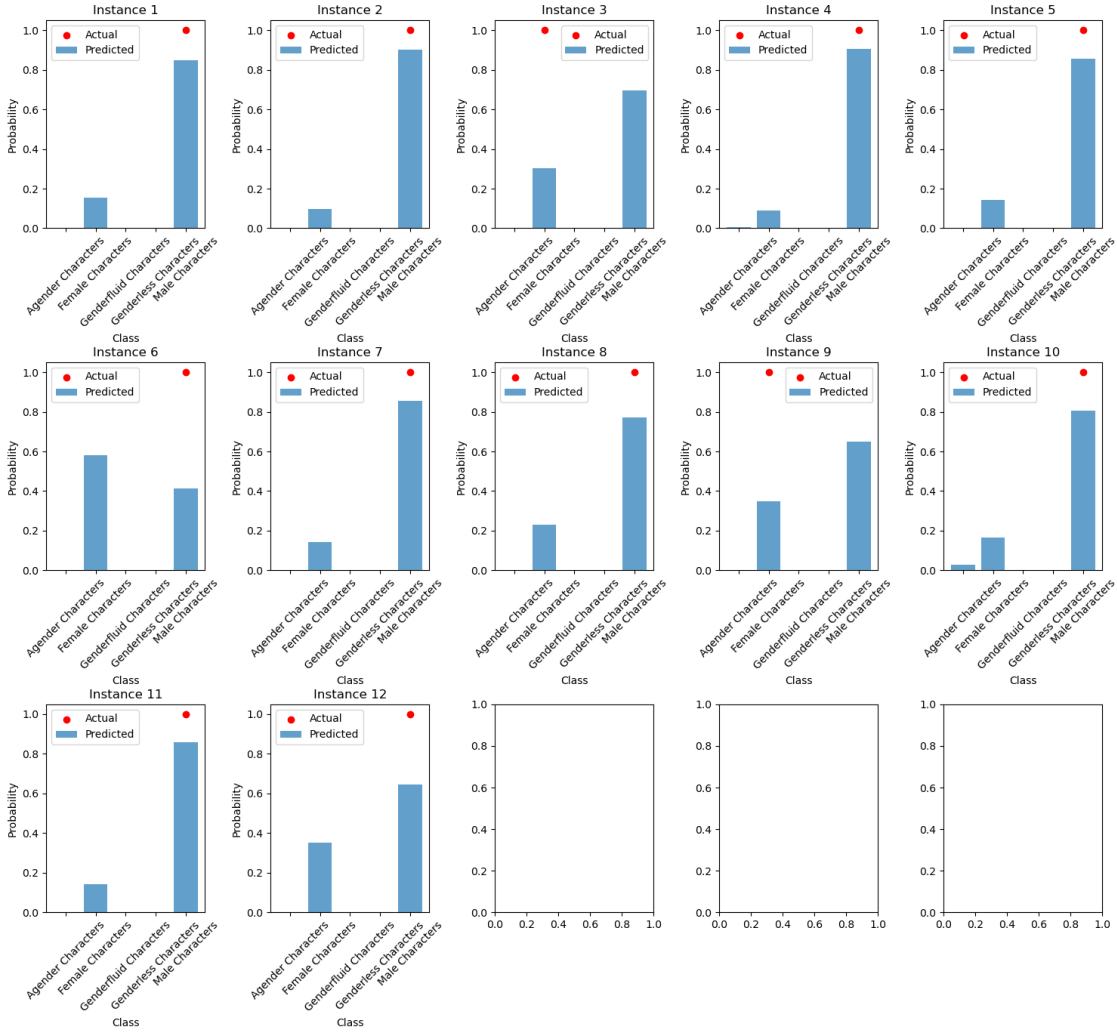


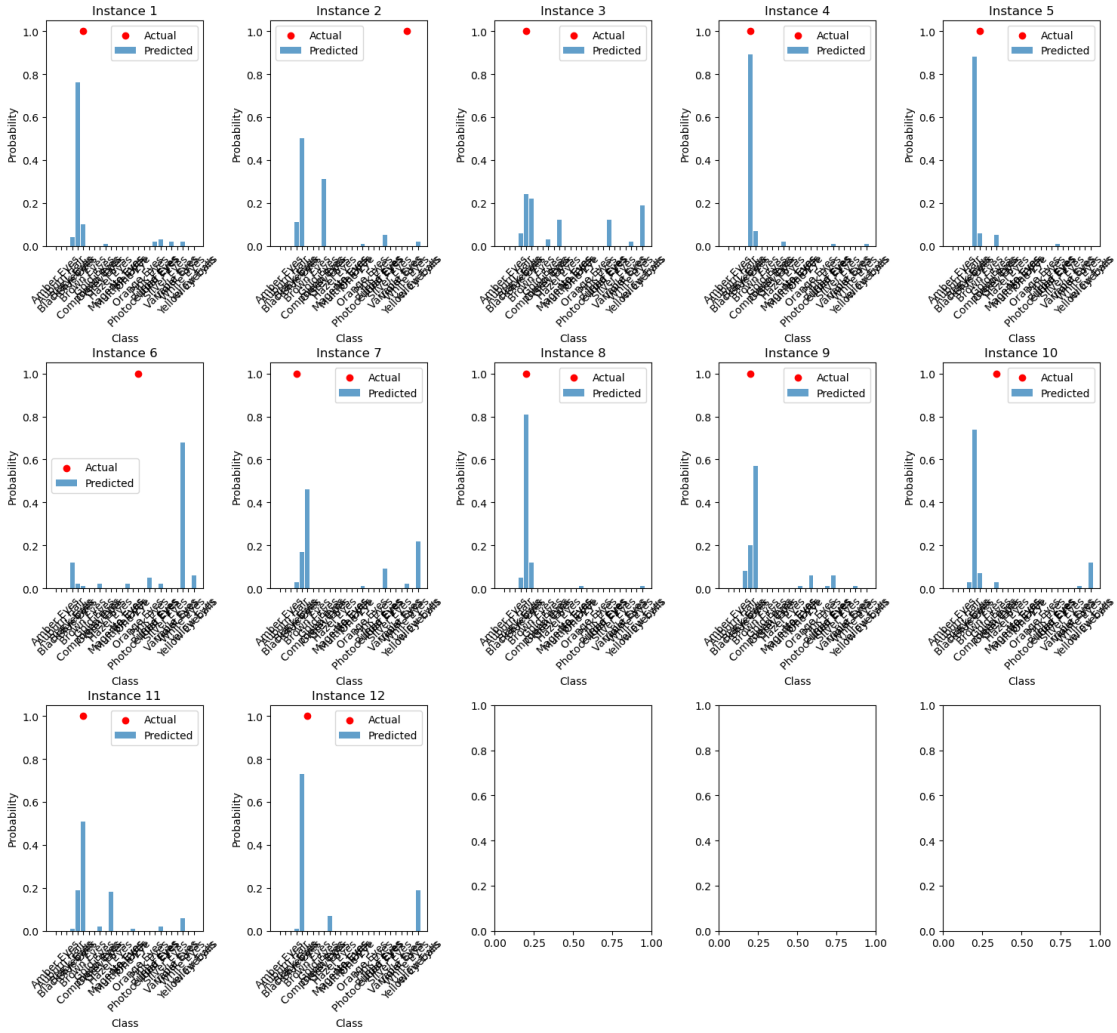


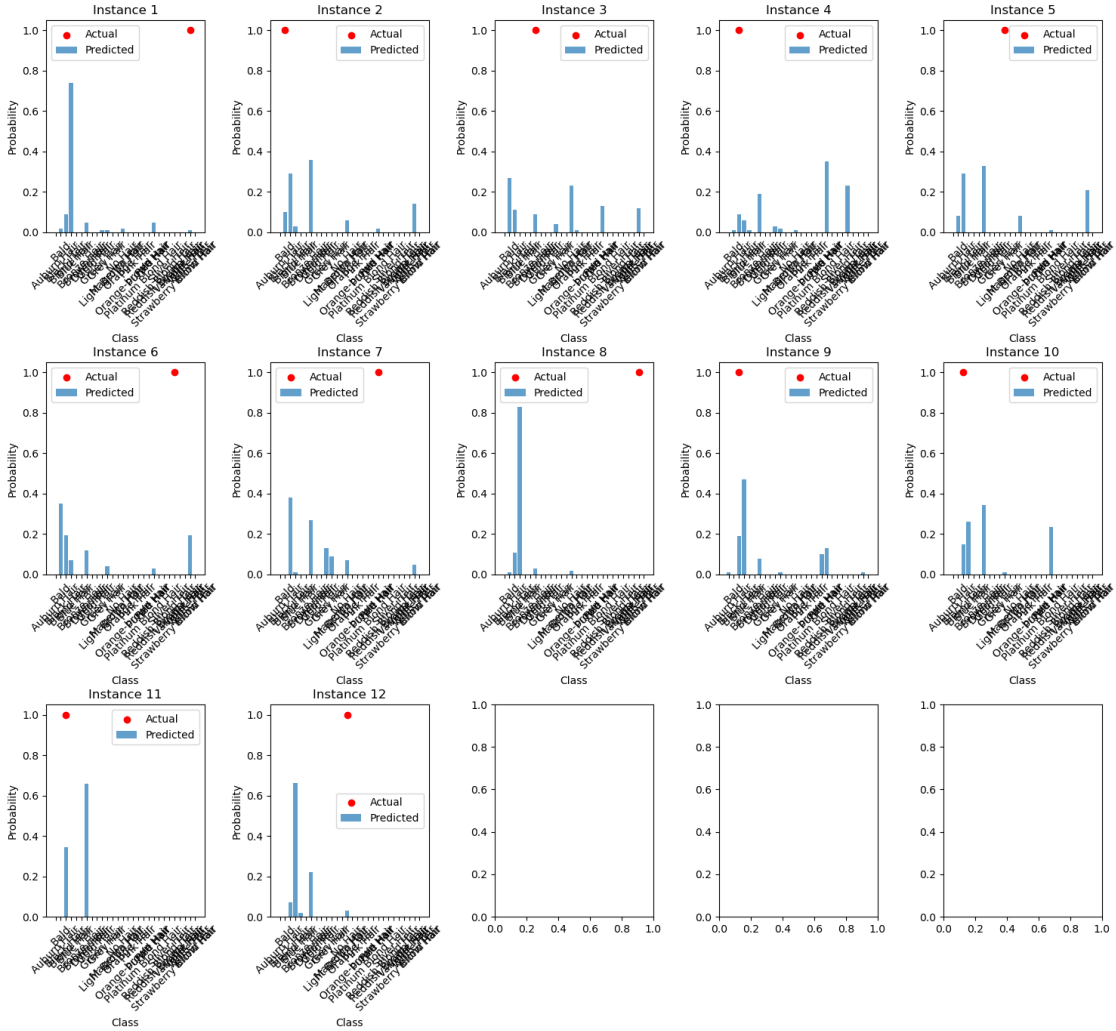
[ ]:
```
# Loop through each dataset
for rf_model, X_train, y_train, X_test, y_test, target_feature, class_names in␣
↪rf_models_combined:
    rf_model = RandomForestClassifier(**best_params_combined[target_feature])
    rf_model.fit(X_train, y_train)

    # Select a random subset of instances from the test data
    num_instances = min((int(len(X_test) * 0.1) if len(X_test) > 250 else␣
↪len(X_test)), 12)
    selected_indices = np.random.choice(len(X_test), num_instances,␣
↪replace=False)
    X_test_subset = X_test.iloc[selected_indices]
    y_test_subset = y_test[selected_indices]
```

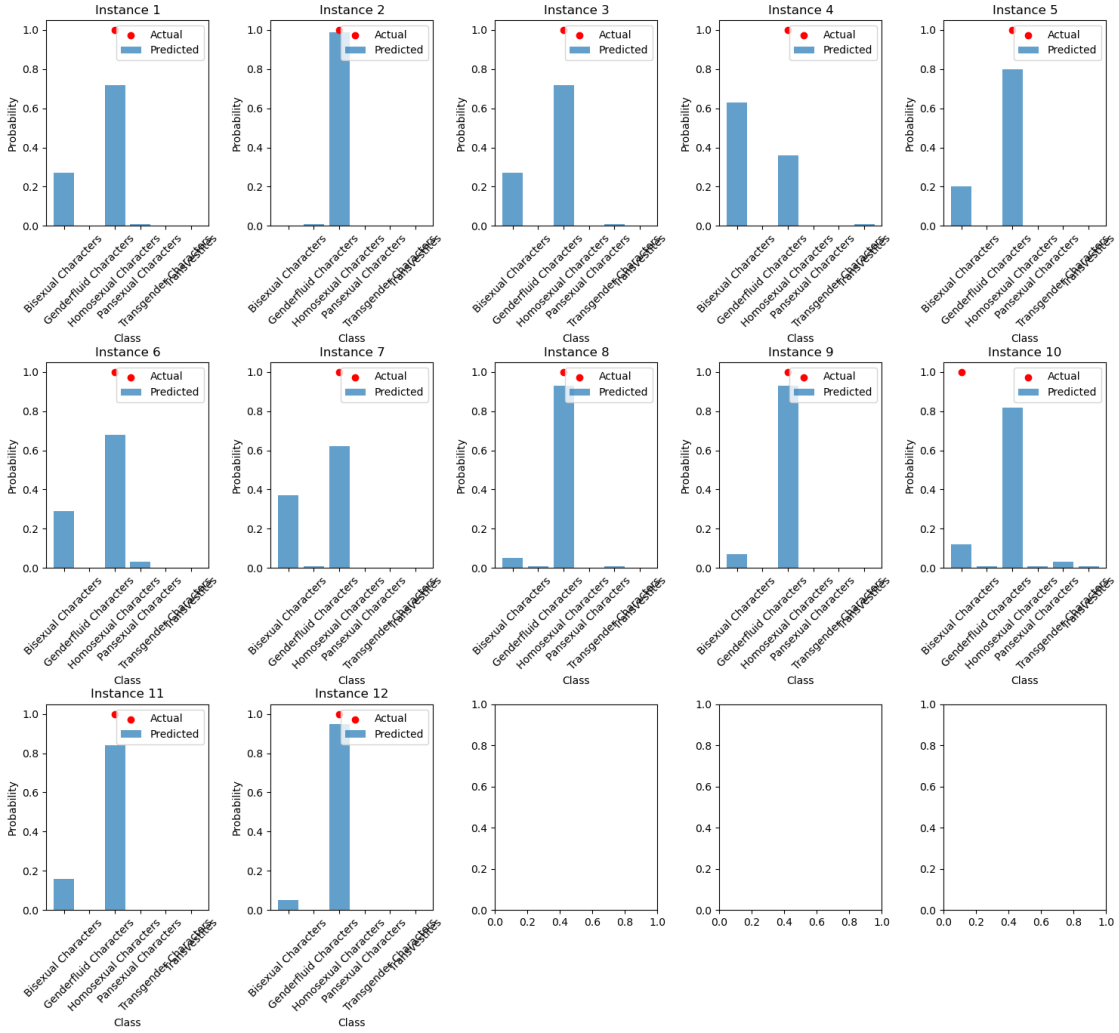
```
    # Get the predicted probabilities for the selected instances in the test data
    predicted_probabilities = rf_model.predict_proba(X_test_subset)

    # Get the subset of class names corresponding to the number of classes
    subset_class_names = class_names[:predicted_probabilities.shape[1]]

    # Calculate the number of rows and columns for subplots
    num_instances_subset, num_classes = predicted_probabilities.shape
    num_columns = min(num_classes, 5)  # Limit the number of columns to avoid too many subplots
    num_rows = math.ceil(num_instances_subset / num_columns)

    # Create a larger plot with subplots
    fig, axes = plt.subplots(num_rows, num_columns, figsize=(15, 15))
    fig.suptitle(f"Predicted Probabilities for {target_feature}", fontsize=16)

    # Loop through each instance and its corresponding predicted probabilities
    for i, (probs, ax) in enumerate(zip(predicted_probabilities, axes.flatten())):
        actual_class = y_test_subset[i]  # Get the actual class value

        # Plot the predicted probabilities as a bar chart
        ax.bar(subset_class_names, probs, label="Predicted", alpha=0.7)

        # Plot the actual class value as a point on top of the predicted probabilities
        ax.scatter(actual_class, 1, color="red", label="Actual")

        ax.set_title(f"Instance {i+1}")  # Set character name as the title
        ax.set_xlabel("Class")
        ax.set_ylabel("Probability")
        ax.set_xticks(range(len(subset_class_names)))  # Set tick positions
        ax.set_xticklabels(subset_class_names, rotation=45)  # Set tick labels
        ax.legend()  # Show legend

    # Adjust layout and display the plot
    plt.tight_layout()
    plt.subplots_adjust(top=0.9)
    plt.savefig(f"""figs/prob_charts/Combined_{target_feature}_{selected_indices}_probabilities.png""")
    plt.show()
```

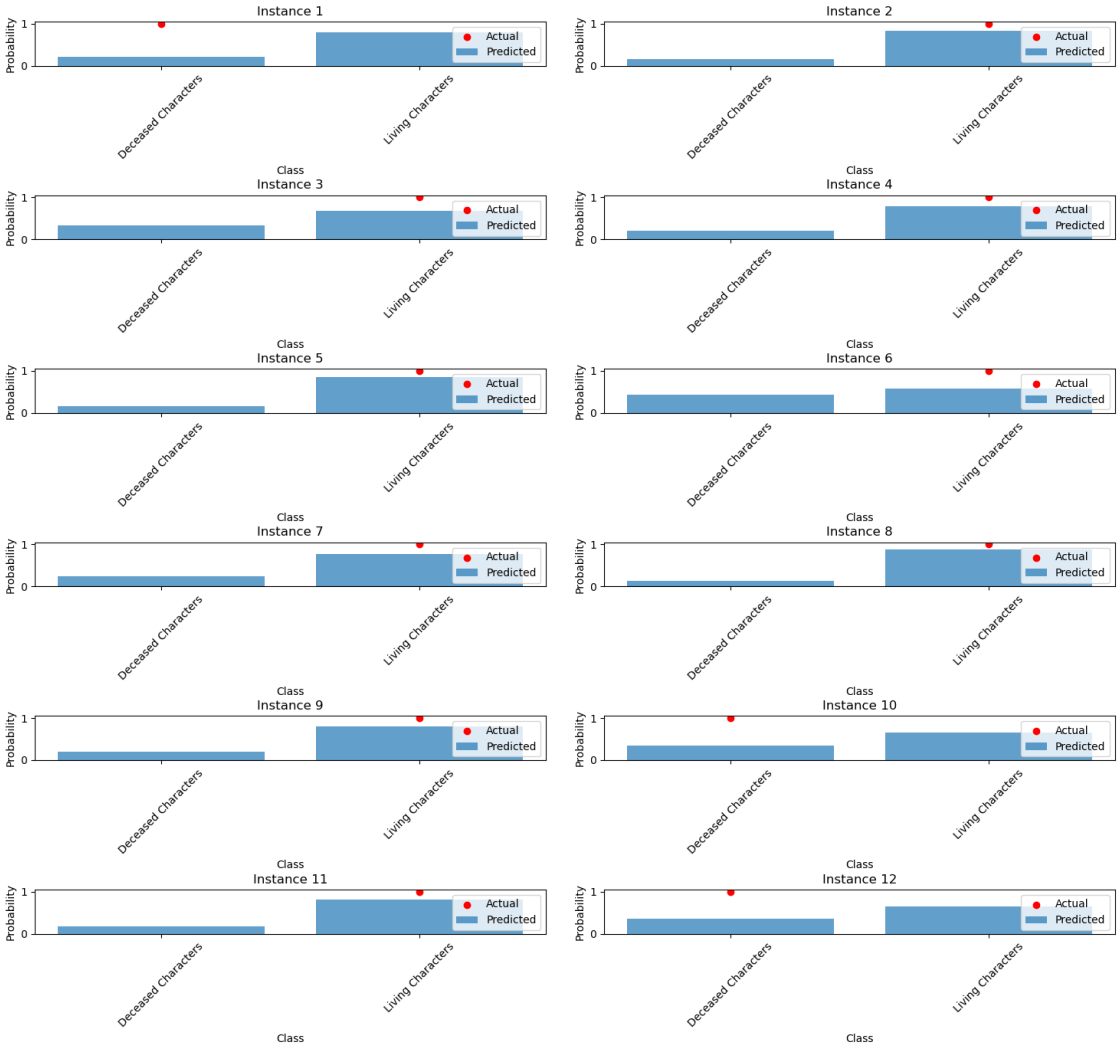

Predicted Probabilities for ALIGN
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class
Bad Characters
Good Characters
Neutral Characters
Reformed Criminals

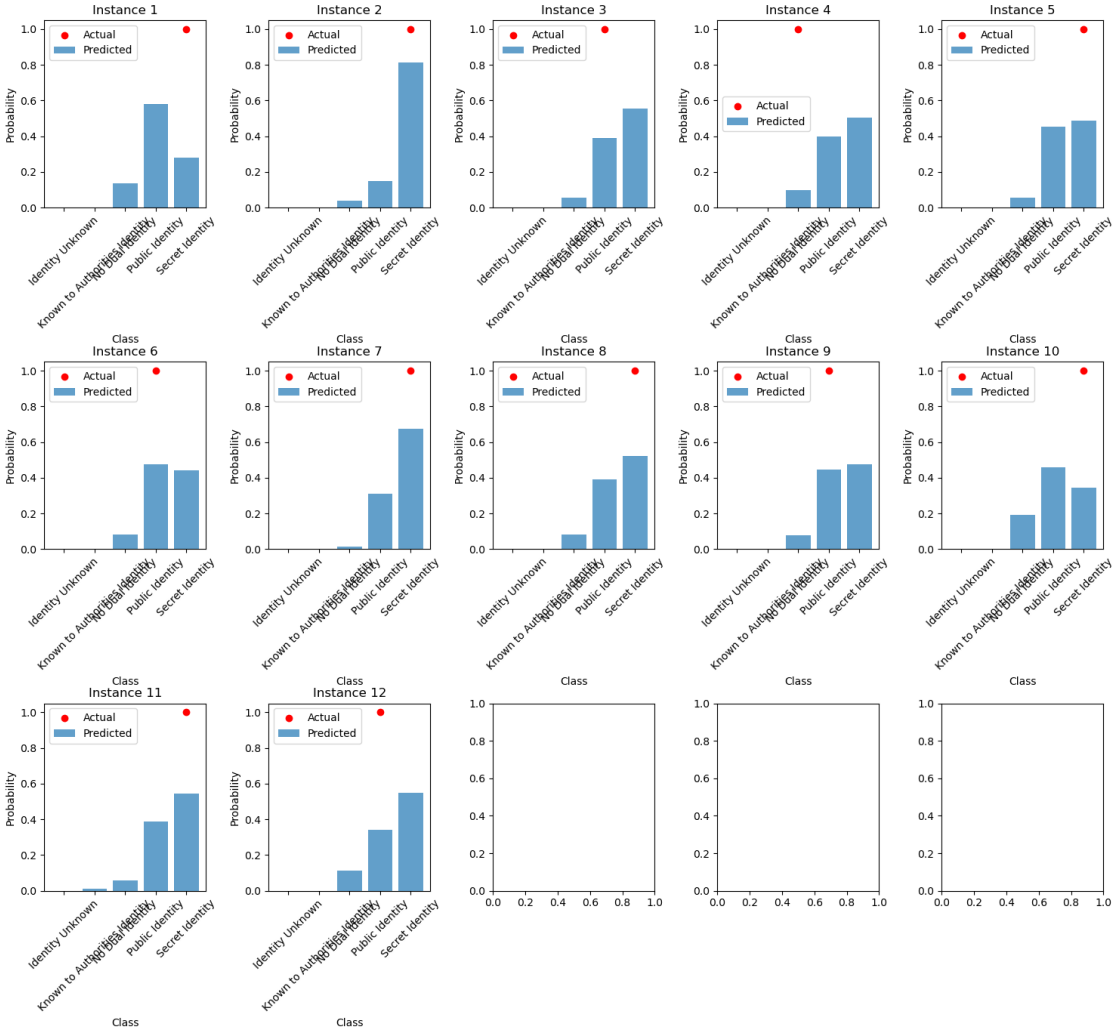

Predicted Probabilities for SEX
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class
Agender Characters
Female Characters
Genderfluid Characters
Genderless Characters
Male Characters


Predicted Probabilities for EYE
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class


Predicted Probabilities for HAIR
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class


Predicted Probabilities for GSM
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class
Bisexual Characters
Genderfluid Characters
Homosexual Characters
Pansexual Characters
Transgender Characters


Predicted Probabilities for ALIVE
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class
Deceased Characters
Living Characters
0
1


Predicted Probabilities for ID
Instance 1
Instance 2
Instance 3
Instance 4
Instance 5
Instance 6
Instance 7
Instance 8
Instance 9
Instance 10
Instance 11
Instance 12
Actual
Predicted
Probability
Class
Identity Unknown
Known to Authorities Identity
Public Identity
Secret Identity