Name	ID
حازم احمد عبدالعال شاذلي	20210265
حامد وليد فتحي	20210273
حافظ عادل حافظ	20210272
عبدالرحمن عبد العزيز ابراهيم عبدالعزيز	20210514
محمود عبدالدايم ابو المجد حسين	20210867
جمال الدين ايمن عبدالرحمن	20210250
عبدالله محمود جمال الدين	20210558

GitHub Repository Link

https://github.com/gimmeursocks/UTKFaceClassifier_HousePricesRegressor

Numerical Dataset

House prices advanced regression

Image Dataset

UTK_Face Cropped

Numerical Data Set

(a) General Information

Name: House Prices Advanced Regression Techniques

Total Number of Samples: 1460

Number of Training: 80% -> 1168

Number of Testing: 20% -> 292

(b) Implementation details

Number of Features Extracted: 4

["HouseAge","GarageAge","TotalBathRooms","OutDoorArea"]

The dimension of resulted features:

```
features = house.columns.drop('SalePrice')

num_features = len(features)

print("Number of features:", num_features)
print('==============')
print(features)
```

✓ 0.0s

```
Number of features: 57
==============
Index(['LotArea', 'Street', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'YearRemodAdd', 'RoofStyle', 'RoofMatl',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual',
       'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
       'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtUnfSF',
       'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical',
       '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageType',
       'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond',
       'PavedDrive', 'SaleType', 'SaleCondition', 'HouseAge', 'GarageAge',
       'TotalBathrooms', 'OutdoorArea', 'Location Information'],
      dtype='object')
```

Is Cross Validation used? Yes

Number of Folds: 5

```
k_folds = KFold(n_splits=5, shuffle=True, random_state=42)
```

Ratio of Training/Validation:

```
for i, (train_index, val_index) in enumerate(k_folds.split(X, y)):
    train_size = len(train_index)
    val_size = len(val_index)
    print(f"Fold {i+1}: Training set size: {train_size}, Validation set size: {val_size}, Ratio of Training/Validation {train_size/val_size}")
```

✓ 0.0s

```
Fold 1: Training set size: 1168, Validation set size: 292, Ratio of Training/Validation 4.0
Fold 2: Training set size: 1168, Validation set size: 292, Ratio of Training/Validation 4.0
Fold 3: Training set size: 1168, Validation set size: 292, Ratio of Training/Validation 4.0
Fold 4: Training set size: 1168, Validation set size: 292, Ratio of Training/Validation 4.0
Fold 5: Training set size: 1168, Validation set size: 292, Ratio of Training/Validation 4.0
```

Hyperparameters used:

Number of Neighbors (`n_neighbors`): Defines the number of neighbors used for prediction.

Weights (`weights`): Specifies the weight function used in prediction.

'P': the 'p' hyperparameter is related to the choice of the Minkowski distance metric.

```
'n_neighbors': [4, 5, 6, 7],
'weights': ['uniform', 'distance'],
'p': [1, 2],
```

(c) Result details

Linear Regression Model:

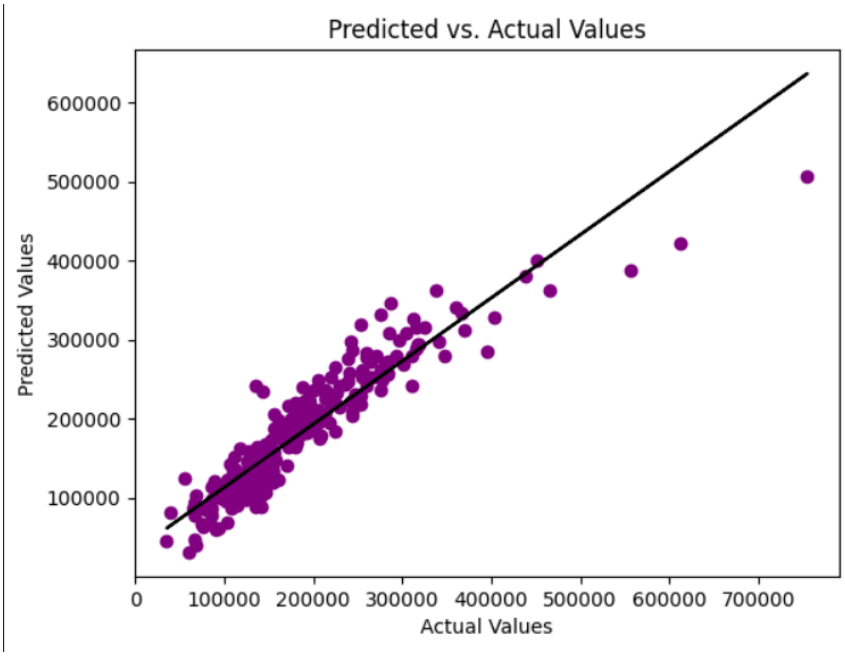
Accuracy:

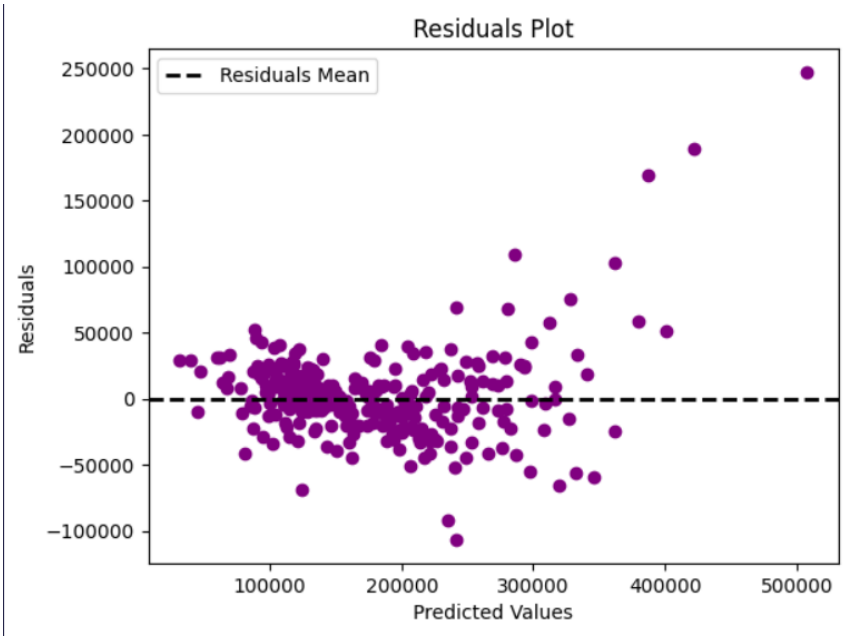
```
print(f"Training: {100*model.score(X_train, y_train):.2f}%")
print(f"Test: {100*model.score(X_test, y_test):.2f}%")
```

✓ 0.0s

```
Training: 83.49%
Test: 85.26%
```

Plots:





$$\text{Residual} = \text{Actual} - \text{Predicted}$$

KNN Model:

Preprocessing:

```
house.corr()['SalePrice'].sort_values(ascending=False)
columns_to_remove = house.corr()[((house.corr()['SalePrice']) <= 0.3) & ((house.corr()['SalePrice']) >= -0.3)].index
knn_house = house.drop(columns=columns_to_remove)
```
✓ 0.0s

```
knn_house.shape
```
✓ 0.0s

```
(1460, 24)
```

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```
✓ 0.0s

Accuracy (Without hyperparameter tuning):

```
print(f"Training: {100*knn.score(X_train, y_train):.2f}%")
print(f"Test: {100*knn.score(X_test, y_test):.2f}%")
```
✓ 0.0s

```
Training: 88.41%
Test: 80.29%
```

Hyperparameter tuning:

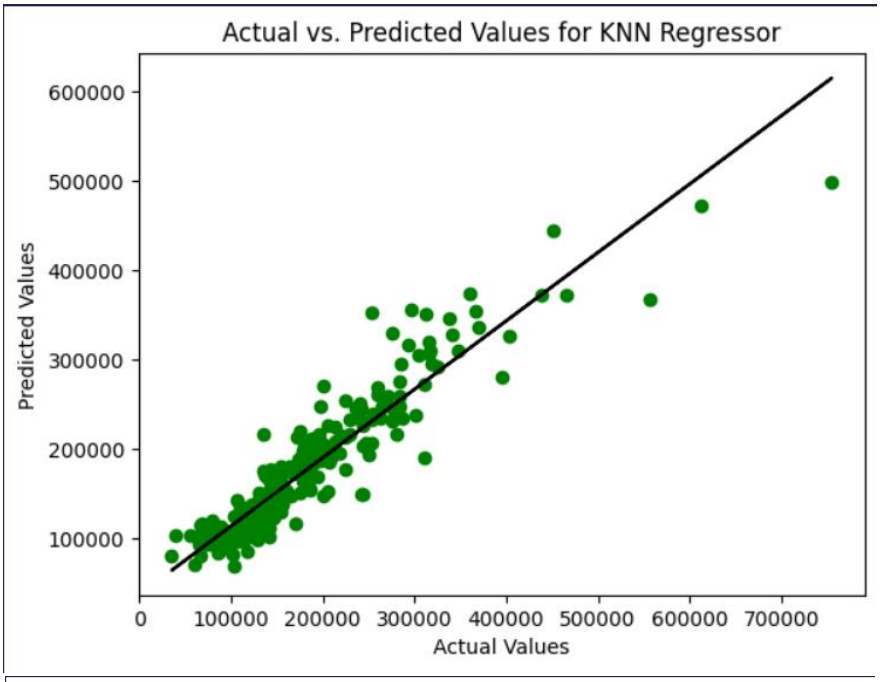
```
# Define the hyperparameter grid
param_grid = {
    'n_neighbors': [4, 5, 6, 7],  # Adjust as needed
    'weights': ['uniform', 'distance'],  # Adjust as needed
    'p': [1, 2],  # Adjust as needed (p=1 for Manhattan distance, p=2 for Euclidean distance)
}

# Create GridSearchCV
k_folds = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, scoring='accuracy', cv=k_folds, n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)
```
✓ 3.1s

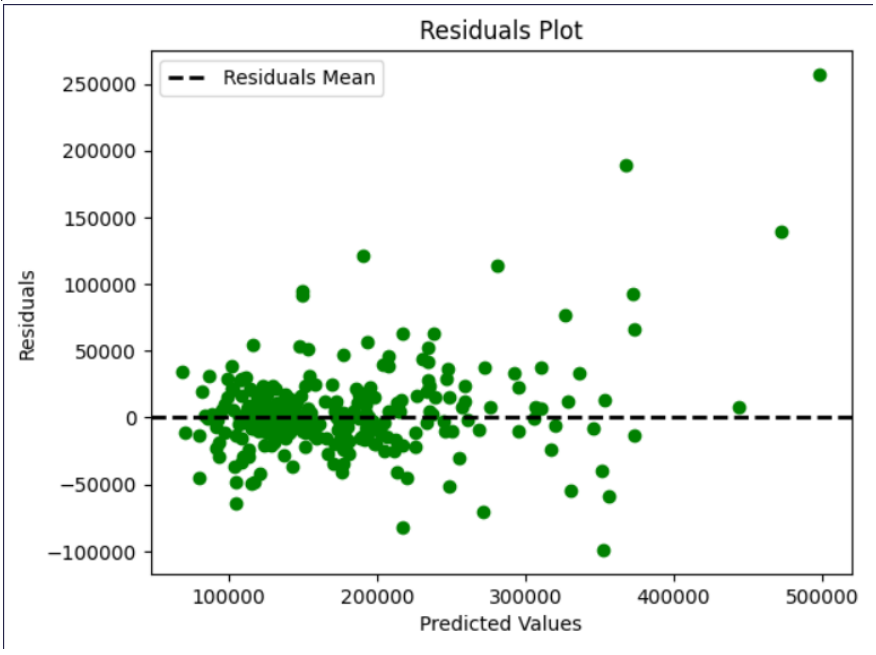
Accuracy (With hyperparameter tuning):

```
print(f"Training: {100*best_knn.score(X_train, y_train):.2f}%")
print(f"Test: {100*best_knn.score(X_test, y_test):.2f}%")
```
✓ 0.0s

```
Training: 88.13%
Test: 85.06%
```

Plots:





- In terms of overall accuracy, the KNN model performs slightly better than the linear regression model.

Image Dataset

(a) General Information

Explanation of data:

1 0 0 201612192 00250923.jpg.chip.jpg

Age

Gender (0: M, 1: F)

Race

Name: Age detection

Total Number of Samples: 14223 sample
(dynamic number) original number is 23k, used a subset of total number

Number of Classes: 4

- Kid 0-17
- Adult 18-49
- Old 50-79
- Senior 80+

•**Their Labels:** [1,2,3,4]

Number of Training: 56% -> 7964

Number of validation: 14% -> 1991

Number of Testing: 30% -> 4267

Image size: `200 x 200 x 3 pixels (Three for RGB channels)`

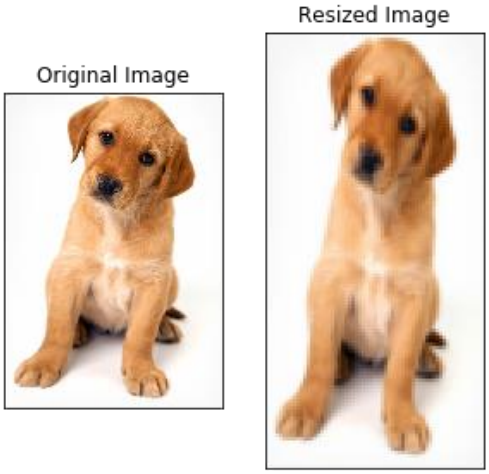
(b) Implementation details

Features Extracted Using Histogram of Oriented Gradients (HOG):

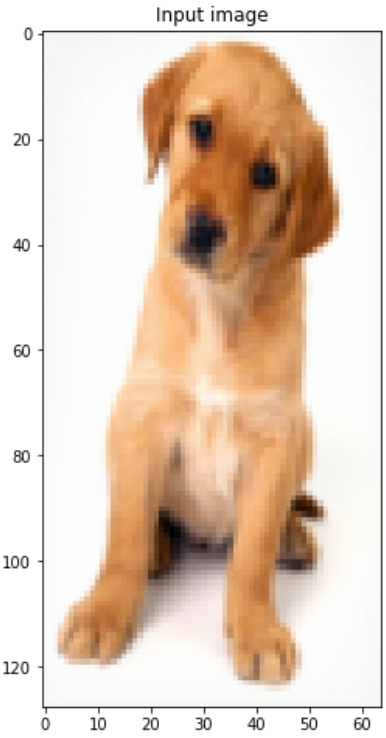
HOG is a feature descriptor that captures information about the gradients or edges in an image:

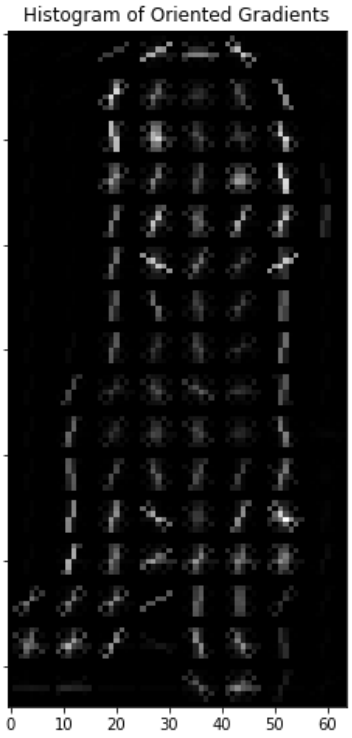
```
def compute_hog_features(image_paths):
    hog_features_list = []

    for path in image_paths:
        image = io.imread(path, as_gray=True)
        resized_image = transform.resize(image, (64, 64))
        hog_features = hog(resized_image, orientations=9, feature_vector=True)

        hog_features_list.append(hog_features)

    return hog_features_list
```

- The function initializes an empty list **`hog_features_list`** to store the HOG features for each image.
- It then iterates through each image path in the input **`image_paths`**.
- For each image, it uses **`io.imread`** to read the image.
- setting **`as_gray=True`** to convert the image to grayscale. Grayscale is often used for HOG feature extraction.
- The image is then resized to a fixed size of (64, 64) using **`transform.resize`**. This step is important because HOG works better with images of a consistent size.



- The `hog` function is applied to the resized image, specifying `orientations=9` to divide the gradient angles into 9 bins. The parameter `feature_vector=True` indicates that the output should be a 1D array (feature vector) instead of a 2D array.
- The computed HOG features for the image are appended to the `hog_features_list`.
- The function repeats this process for all images in the input list.
- Finally, the function returns the list of HOG features for all the input images.





The dimension of resulted features:

```
1  X.shape
✓ 0.0s

(14223, 2916)
```

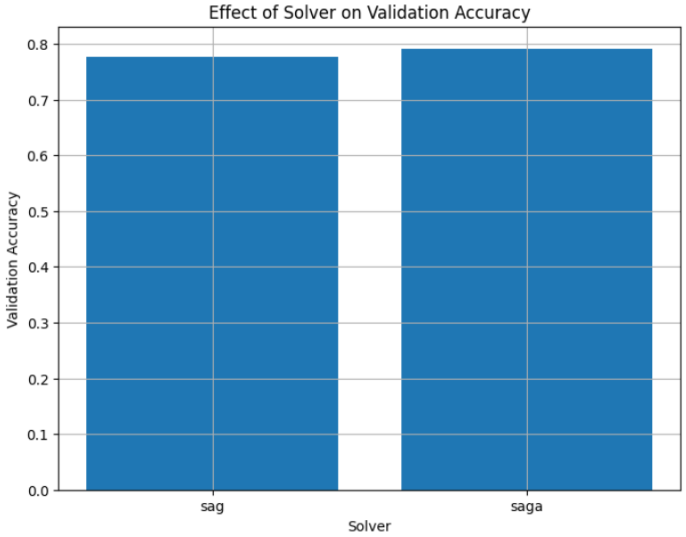
Hyperparameters used in LogisticRegression:

```
age_model = LogisticRegression(C=0.1, solver="saga", tol=0.1)
```

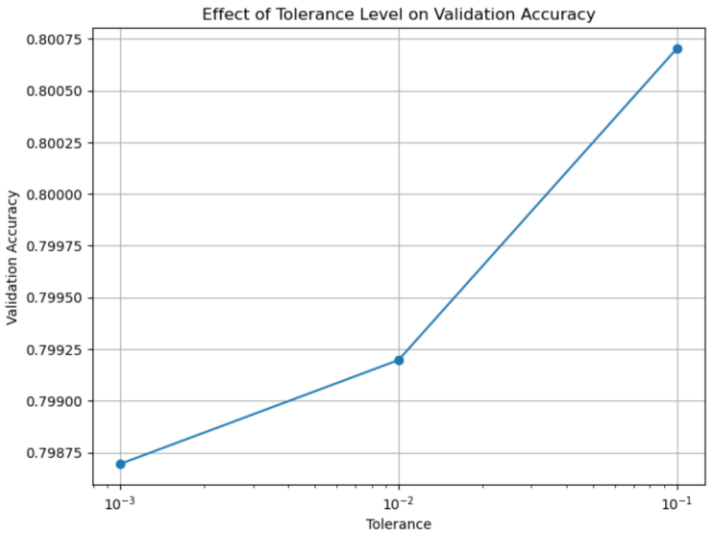
C (C): 0.1 'Regularization is a technique used to prevent overfitting in a model by penalizing the complexity of the model'

Solver('saga') : Stands for Stochastic Average Gradient, which is a variant of gradient descent that uses a stochastic average of past gradients. This solver is often suitable for large datasets.

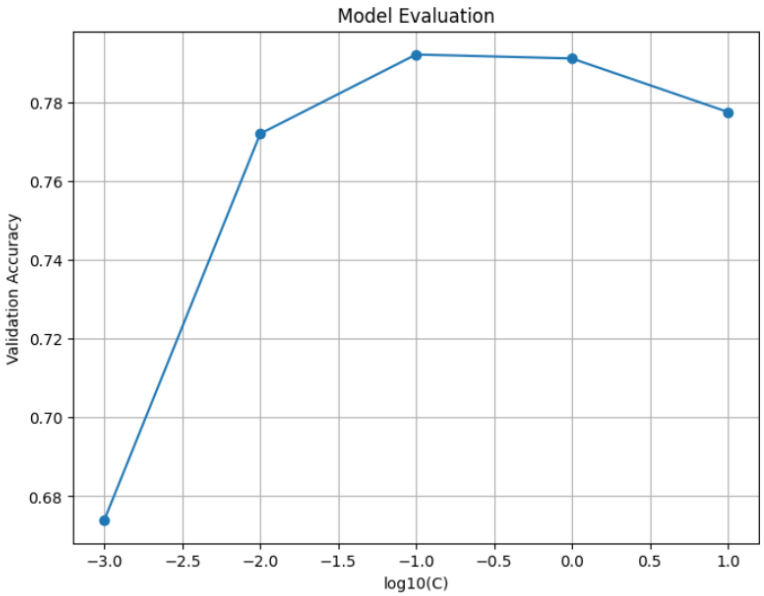
Tolerance (tol): 0.1 'refers to the stopping criterion based on the convergence of the optimization algorithm used to find the coefficients that best fit the data



Solver('saga') : Stands for Stochastic Average Gradient, which is a variant of gradient descent that uses a stochastic average of past gradients. This solver is often suitable for large datasets.



Tolerance (`tol`): 0.1 'refers to the stopping criterion based on the convergence of the optimization algorithm used to find the coefficients that best fit the data



C (C): 0.1 'Regularization is a technique used to prevent overfitting in a model by penalizing the complexity of the model'

(c) LogisticRegression Result details

1. Accuracy score in Logistic Regression:

```
accuracy = age_model.score(X_test,y_test)
print(f"Accuracy for Age Prediction: {accuracy:.2f}")
```

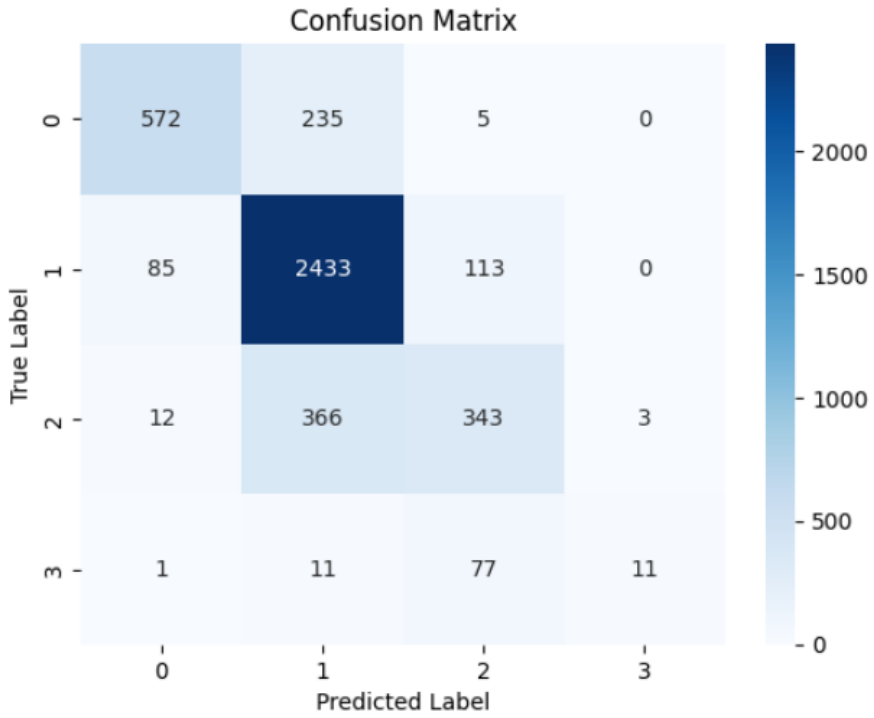
```
Accuracy for Age Prediction: 0.79
```

2. Accuracy score for each set (train,test,validation) in Logistic Regression:

```
train_predictions = age_model.predict(X_train)
train_accuracy = accuracy_score(y_train, train_predictions)

# Predictions on test set
test_predictions = age_model.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)


val_predictions = age_model.predict(X_val)
val_accuracy = accuracy_score(y_val, val_predictions)



print(f"Train Accuracy: {train_accuracy:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Validation Accuracy: {val_accuracy:.4f}")

```

```
Train Accuracy: 0.8226
Test Accuracy: 0.7893
Validation Accuracy: 0.8007
```

3. Confusion Matrix in Logistic Regression:



The elements of the confusion matrix are as follows:

- **Row 1 (Age Group 0):**
  - 572 instances were correctly predicted as Age Group 0.
  - 240 instances from Age Group 0 were misclassified as other groups. And so on

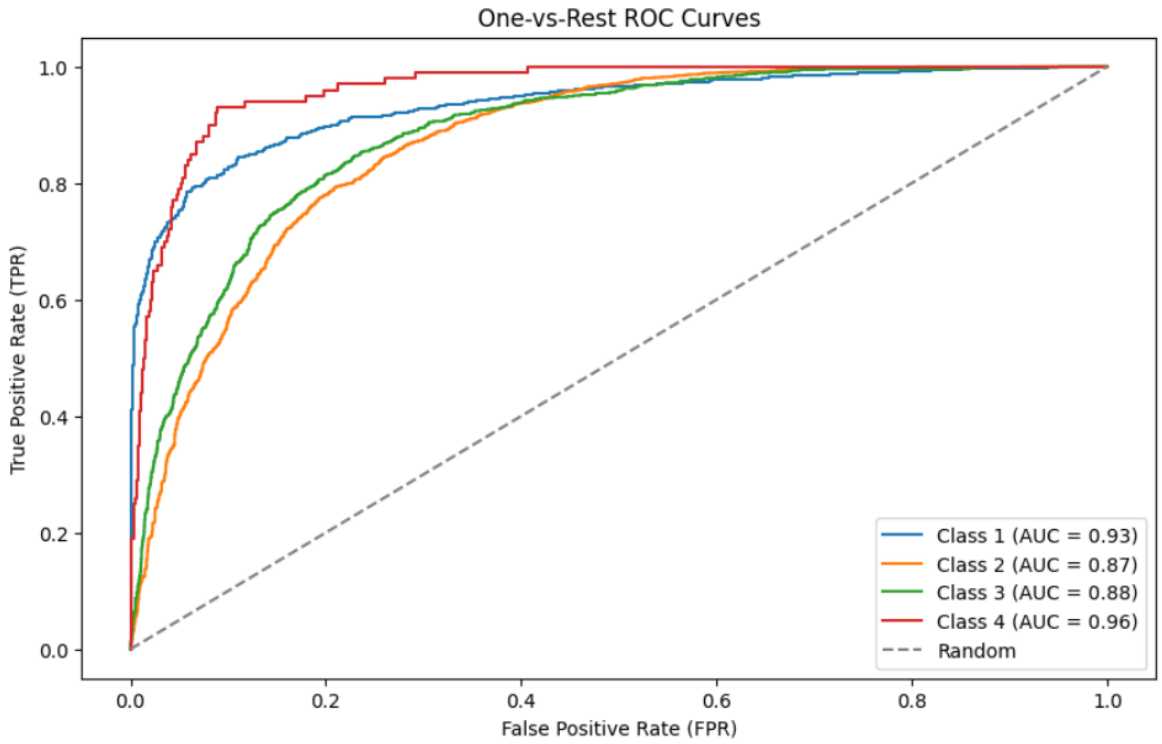
	precision	recall	f1-score
1	0.86	0.67	0.75
2	0.80	0.92	0.86
3	0.64	0.52	0.57
4	0.50	0.06	0.11
accuracy			0.79
macro avg	0.70	0.54	0.57
weighted avg	0.78	0.79	0.77

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures how many of the predicted positive instances are positive.

Recall is the ratio of correctly predicted positive observations to the total actual positives. It measures how many of the actual positive instances are correctly predicted.

The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall, particularly useful when there is an imbalance between the number of positive and negative instances.

4. ROC Curve in Logistic Regression:



True Positive Rate (TPR): The proportion of actual positive instances that are correctly identified by the model.

False Positive Rate (FPR): The proportion of actual negative instances that are incorrectly classified as positive.

- **Class 1 (AUC = 0.93):** A high AUC indicates good discrimination for class 1. The closer to 1, the better.

- **Class 2 (AUC = 0.87):** A good AUC but slightly lower than class 1. Still, it indicates reasonable discrimination.
- **Class 3 (AUC = 0.88):** Similar to class 2, indicating good discrimination.
- **Class 4 (AUC = 0.96):** The highest AUC among the classes, suggesting excellent discrimination for class 4.
- Senior(80+) is the easiest, Adult(18-49) is the hardest

K Means model

(b) Implementation details

Image resized:

```python
def resize_images():
    data = []
    for pic in df.image:
        img = io.imread(pic, as_gray=True)
        transformed_img = transform.resize(img,(64,64))
        transformed_img = transformed_img.reshape(1,4096)
        data.append(transformed_img)

    data = np.array(data).reshape(len(data),4096)
    return data
```

✓ 0.0s

The dimension of resulted features:

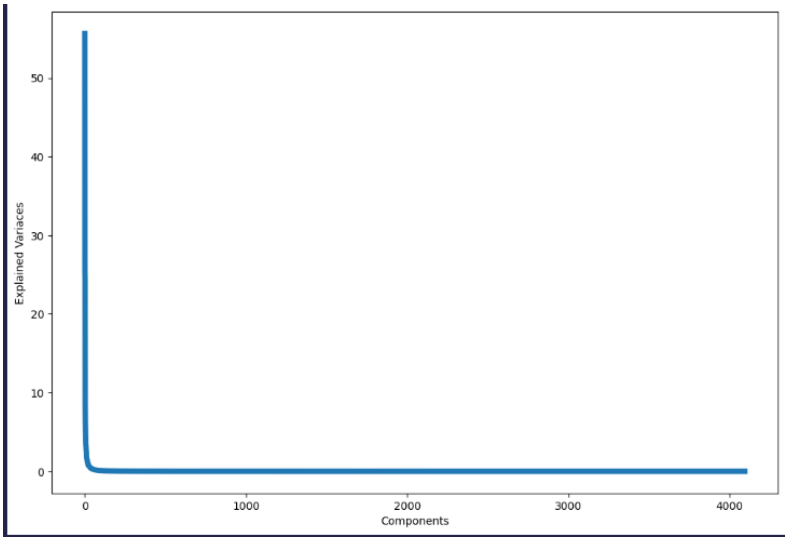
```python
X.shape
```

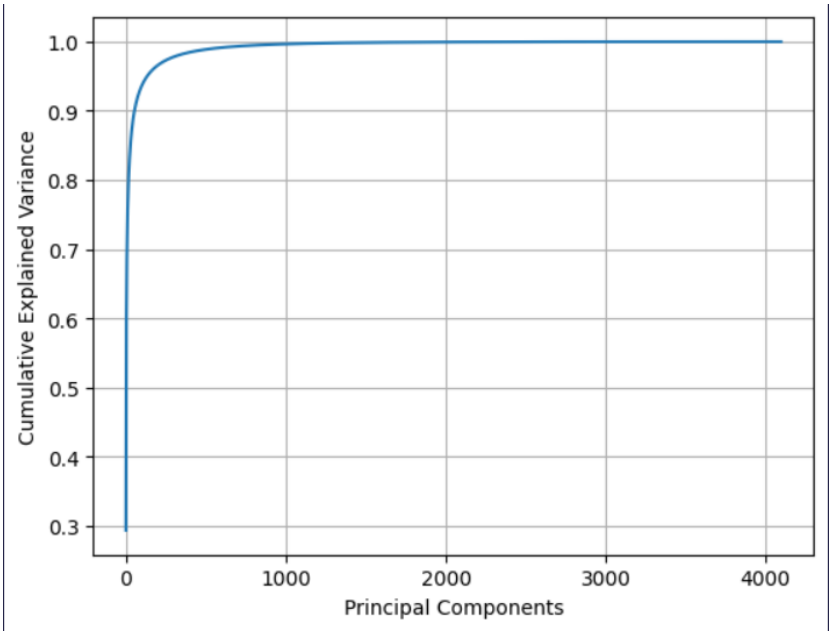
✓ 0.0s

```
(14223, 4096)
```

Preprocessing for KMeans:

PCA





Cutoff at 95% and more cumulative variance results in reduced features

134 features

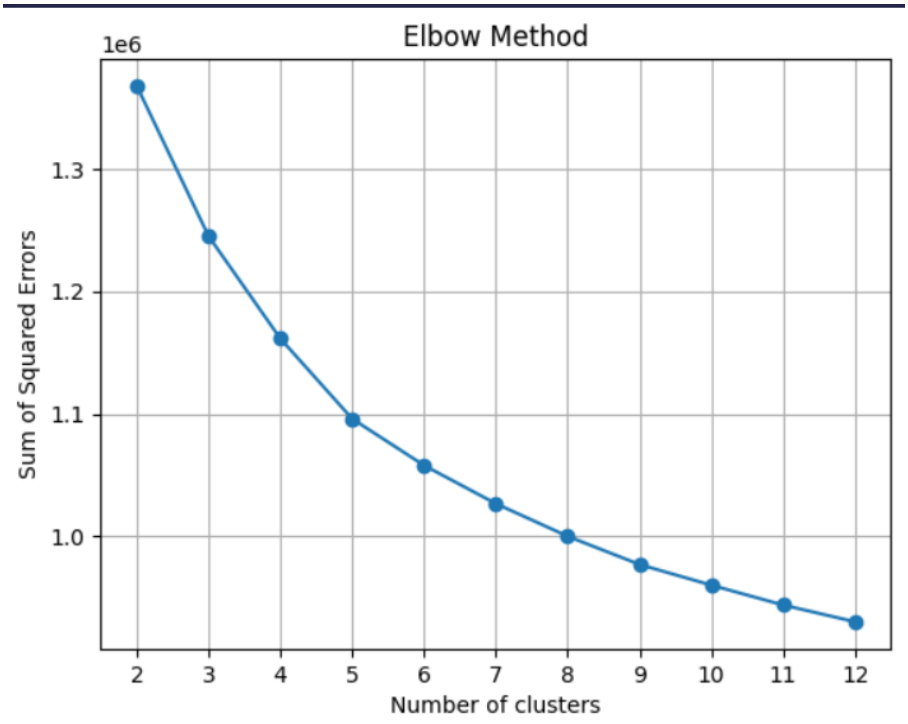
Hyperparameters used in KMeans Clustering:

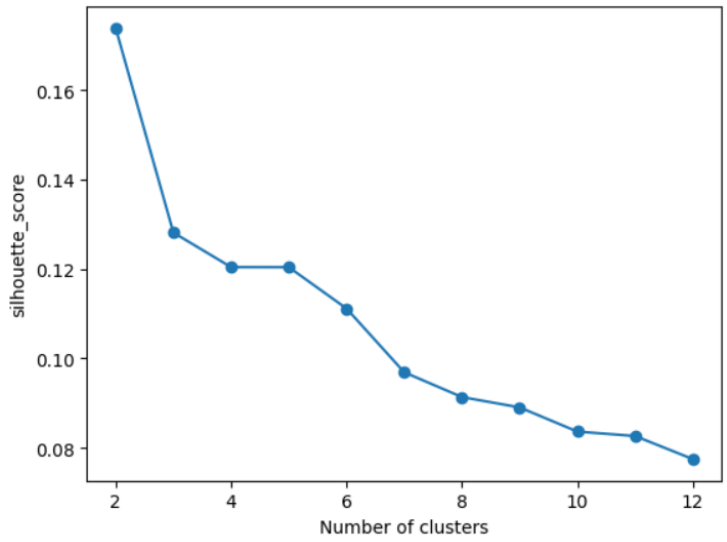
```
kmeans = KMeans(n_clusters=k, n_init=10, max_iter=400, init='k-means++', random_state=42)
```

Number of Clusters (`n_clusters`): 3 'Determines the number of clusters'

Maximum Number of Iterations (`max_iter`): 400 'The maximum number of iterations the K-Means algorithm will run.'

Initialization Method (`init`): 'k-means++' 'Specifies the method used to initialize the centroids of the clusters.'





Silhouette scores determine how similar a data point is within a cluster compared to other clusters. 1 is separated and -1 is displays cohesion.

(c) Result details

Confusion matrices, ROC curves, and loss curves are concepts more commonly associated with supervised learning, particularly in classification tasks. K-means clustering, on the other hand, is an unsupervised learning algorithm used for clustering data.