TEAM1

December 8, 2024

PRML Project Assignment - BMI Faces

TEAM MEMBERS:

Kathiravan S (CS22B2052)
Avinaash A (CS22B1064)
Ashwinth Anbu (CS22B2055)

[1]:
```
import torch
from facenet_pytorch import MTCNN, InceptionResnetV1
from PIL import Image
import os
import pandas as pd
import numpy as np
from torchvision import transforms
from tqdm import tqdm
import warnings
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBRegressor
from skimage import io
warnings.filterwarnings('ignore')
```

Setting Basic Parameters

[28]:
```
BASE_PATH = "/media/kathir/Apps and Games/prml project/illinois_doc_dataset"
MAX_PERSONS = 60000
OUTPUT_FILE = "face_features_60000.csv"
```

Feature Extraction - FaceNet Features (512 x 2 = 1024 Features per Person)

[29]:
```python
def load_and_process_image(image_path, device, mtcnn, resnet):
    try:
        img = Image.open(image_path)
        img_cropped = mtcnn(img)

        if img_cropped is None:
            print(f"No face detected in {image_path}")
            return None

        img_cropped = torch.unsqueeze(img_cropped, 0).to(device)
        with torch.no_grad():
            features = resnet(img_cropped)
        return features.cpu().numpy().flatten()

    except Exception as e:
        print(f"Error processing {image_path}: {str(e)}")
        return None
```

[30]:
```python
def extract_features(base_path, max_persons=1000):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"Using device: {device}")

    mtcnn = MTCNN(device=device)
    resnet = InceptionResnetV1(pretrained='vggface2').eval().to(device)

    front_path = os.path.join(base_path, 'front')
    front_files = sorted(os.listdir(front_path))

    if max_persons: front_files = front_files[:max_persons]

    all_features = []
    processed_files = []

    for front_file in tqdm(front_files, desc="Processing images"):
        side_file = front_file  # Same filename in side folder

        front_features = load_and_process_image(
            os.path.join(front_path, front_file),
            device, mtcnn, resnet
        )

        side_features = load_and_process_image(
            os.path.join(base_path, 'side', side_file),
            device, mtcnn, resnet
        )

        if front_features is not None and side_features is not None:
```

```
            combined_features = np.concatenate([front_features, side_features])
            all_features.append(combined_features)
            processed_files.append(front_file)

    # Create feature column names
    front_cols = [f'front_feature_{i}' for i in range(512)]  # FaceNet outputs␣
↪512-D vectors
    side_cols = [f'side_feature_{i}' for i in range(512)]
    all_cols = front_cols + side_cols

    df = pd.DataFrame(all_features, columns=all_cols)
    df.insert(0, 'id', processed_files)

    return df
```

Extracting...

[ ]:
```
print("Starting feature extraction...")
features_df = extract_features(BASE_PATH, MAX_PERSONS)
```

[6]:
```
features_df.to_csv(OUTPUT_FILE, index=False)
print(f"Features saved to {OUTPUT_FILE}")
print(f"Processed {len(features_df)} persons successfully")
```

```
Features saved to face_features_65000.csv
Processed 59558 persons successfully
```

Initializing Training and Testing Parameters

[110]:
```
MAX_PERSONS = 50000
```

[146]:
```
persons = pd.read_csv("/media/kathir/Apps and Games/prml project/
↪illinois_doc_dataset/label.csv",delimiter = ";")
features = pd.read_csv("face_features_60000.csv")[:MAX_PERSONS]
features['id'] = features['id'].str.split('.').str[0]
filtered_df = pd.merge(features, persons, on='id')
filtered_df['bmi'] = (filtered_df['weight']*703)/(filtered_df['height']**2)
xs = np.array(filtered_df.iloc[:, 1:1025])
xs_train = xs[:int(0.8*len(xs))]
xs_test = xs[int(0.8*len(xs)):]
filtered_df['bmi'].fillna((filtered_df['bmi'].mean()), inplace=True)
ys = np.array(filtered_df['bmi'])
print(len(xs), len(ys))
ys_train = ys[:int(0.8*len(ys))]
ys_test = ys[int(0.8*len(ys)):]
```

```
50000 50000
```

[112]:
```
persons
```

[112]:

	id	name	date_of_birth	weight	\
0	A00147	MCCUTCHEON, JOHN	06/14/1949	185.0	
1	A00220	WALKER, ISIAH	03/30/1957	155.0	
2	A00360	BELL, HOWARD	12/18/1946	167.0	
3	A00367	GARVIN, RAYMOND	01/12/1954	245.0	
4	A01054	TIPTON, DARNELL	03/25/1954	166.0	
…	…	…	…	…	
61105	Y25363	RANEY, JEFFERY D.	05/15/1986	170.0	
61106	Y25364	CALDWELL, KIMBERLY D.	02/23/1972	112.0	
61107	Y25365	WEIGAND, KARLEE R.	06/16/1992	158.0	
61108	Y25366	CROY, DAVID W.	03/04/1949	220.0	
61109	Y25367	BROWN, DANDRE R.	06/18	NaN	

	hair	sex	height	race	eyes	admission_date	\
0	Brown	Male	67.0	White	Blue	02/16/1983	
1	Black	Male	73.0	Black	Brown	05/19/2016	
2	Gray or Partially Gray	Male	69.0	White	Green	02/26/1988	
3	Black	Male	72.0	Black	Brown	11/09/2017	
4	Salt and Pepper	Male	67.0	Black	Brown	12/23/1988	
…	…	…	…	…	…	…	
61105	Brown	Male	71.0	White	Brown	10/25/2017	
61106	Brown	Female	62.0	White	Green	10/25/2017	
61107	Brown	Female	63.0	White	Brown	10/25/2017	
61108	Gray or Partially Gray	Male	67.0	White	Blue	10/25/2017	
61109	NaN	NaN	NaN	NaN	NaN	NaN	

	…	projected_discharge_date	parole_date	electronic_detention_date	\
0	…	10/06/2036	NaN	NaN	
1	…	NaN	NaN	NaN	
2	…	TO BE DETERMINED	10/02/2017	NaN	
3	…	11/20/2020	NaN	NaN	
4	…	08/14/2068	NaN	NaN	
…	…	…	…	…	
61105	…	07/05/2019	07/05/2018	NaN	
61106	…	TO BE DETERMINED	10/25/2017	NaN	
61107	…	04/07/2020	04/07/2018	NaN	
61108	…	03/22/2025	NaN	NaN	
61109	…	NaN	NaN	NaN	

	discharge_date	parent_institution	offender_status	\
0	NaN	DIXON CORRECTIONAL CENTER	IN CUSTODY	
1	NaN	STATEVILLE CORRECTIONAL CENTER	NON-IDOC CUSTODY	
2	NaN	PINCKNEYVILLE CORRECTIONAL CENTER	PAROLE	
3	NaN	WESTERN ILLINOIS CORRECTIONAL CENTER	IN CUSTODY	
4	NaN	MENARD CORRECTIONAL CENTER	IN CUSTODY	
…	…	…	…	
61105	NaN	SOUTHWESTERN CORRECTIONAL CENTER	PAROLE	

```
61106            NaN             LOGAN CORRECTIONAL CENTER         ABSCONDER
61107            NaN           VIENNA CORRECTIONAL CENTER            PAROLE
61108            NaN         ROBINSON CORRECTIONAL CENTER        IN CUSTODY
61109            NaN                                  NaN               NaN

                       location sex_offender_registry_required  \
0                         DIXON                            True
1      ILL/OTH STATE/FED CONCURR                             NaN
2              PAROLE DISTRICT 1                             NaN
3               WESTERN ILLINOIS                             NaN
4                         MENARD                            True
…                             …                             …
61105          PAROLE DISTRICT 5                             NaN
61106                     PAROLE                             NaN
61107          PAROLE DISTRICT 2                             NaN
61108                   ROBINSON                             NaN
61109                        NaN                             NaN

                                                 alias Unnamed: 21
0                                                  NaN         NaN
1                                                  NaN         NaN
2      HOWARD R BELL | DONALD  BROADSTONE | RONALD  B…         NaN
3                                                  NaN         NaN
4                                                  NaN         NaN
…                                                  …           …
61105                                              NaN         NaN
61106                                              NaN         NaN
61107                                              NaN         NaN
61108                                              NaN         NaN
61109                                              NaN         NaN

[61110 rows x 22 columns]
```

[113]:
```
len(xs_train),len(ys_train)
```

[113]:
```
(40000, 40000)
```

Loading Team Member Face Images

[114]:
```
our_features = extract_features("/media/kathir/Apps and Games/prml project/
 ↪check", 3).iloc[:,1:]
prefix = "/media/kathir/Apps and Games/prml project/check/front"
our_paths = [f"{prefix}/kathir.jpg", f"{prefix}/avinaash.jpg", f"{prefix}/
 ↪ashwinth.jpg"]
our_bmis = {'id':[1, 2, 3], 'bmi': [23.5, 23.2, 23.1]}
our_bmis = pd.DataFrame(our_bmis)
```

```
Using device: cpu
```

```
Processing images:
100%|                                    | 3/3
[00:00<00:00,  4.84it/s]
```

Diagnostic Functions

[115]:
```python
def diagnostics():
    plt.figure(figsize=(10, 6))

    # Plotting Actual vs Predicted
    plt.subplot(1, 2, 1)
    plt.scatter(ys, predictions, color='teal', alpha=0.6)
    plt.plot([min(ys), max(ys)], [min(ys), max(ys)], color='red',␣
 ↪linestyle='--', linewidth=2)
    plt.title('Actual vs Predicted BMI')
    plt.xlabel('Actual BMI')
    plt.ylabel('Predicted BMI')

    # Residuals plot (Actual - Predicted)
    residuals = ys - predictions
    plt.subplot(1, 2, 2)
    sns.scatterplot(x=predictions, y=residuals, color='red', alpha=1)
    plt.axhline(0, color='black', linestyle='--', linewidth=2)  # Horizontal␣
 ↪line at 0
    plt.title('Residuals vs Predicted BMI')
    plt.xlabel('Predicted BMI')
    plt.ylabel('Residuals')

    plt.tight_layout()
    plt.show()

def imgdisplay(our_preds):
    plt.figure(figsize=(15, 5))

    for i in range(len(our_preds)):
        img = io.imread(our_paths[i])
        plt.subplot(1, 3, i + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"Actual BMI: {our_bmis.iloc[i]['bmi']}\nPredicted BMI:␣
 ↪{our_preds[i]:.2f}")

    plt.tight_layout()
    plt.show()
```

Linear Regression

[116]:
```
#CUSTOM Linear Regression Implementation   -- SLOW, use with persons < 1000
'''class LinearRegression:
    def __init__(self, learning_rate=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, xs, ys):
        # Initialize weights and bias
        n_samples, n_features = len(xs), len(xs[0])
        self.weights = [0] * n_features  # Zero weights
        self.bias = 0  # Zero bias

        # Gradient Descent
        for _ in range(self.epochs):
            # Calculate predictions: y_pred = X.w + b
            y_pred = [self._predict_single(x) for x in xs]

            # Compute gradients
            dw = [0] * n_features
            db = 0
            for i in range(n_samples):
                error = ys[i] - y_pred[i]
                db += -2 * error  # Gradient of bias
                for j in range(n_features):
                    dw[j] += -2 * error * xs[i][j]  # Gradient of weights

            # Average the gradients
            db /= n_samples
            dw = [grad / n_samples for grad in dw]

            # Update weights and bias
            self.bias -= self.learning_rate * db
            self.weights = [w - self.learning_rate * grad for w, grad in␣
 ↪zip(self.weights, dw)]

    def predict(self, xs):
        # Predict for multiple samples
        return [self._predict_single(x) for x in xs]

    def _predict_single(self, x):
        # Predict for a single sample
        return sum(w * xi for w, xi in zip(self.weights, x)) + self.bias

lr_model = LinearRegression(learning_rate=0.01, epochs=1000)
lr_model.fit(xs_train, ys_train)
```

```
predictions = lr_model.predict(xs_test)

print("MSE: ",mean_squared_error(predictions,ys_test))
print("MAE: ",mean_absolute_error(predictions,ys_test))
print("R2 score: ",r2_score(predictions,ys_test))
corr, p_value = pearsonr(y_true, y_pred)
print("Pearson's correlation coefficient: ",corr)'''
```

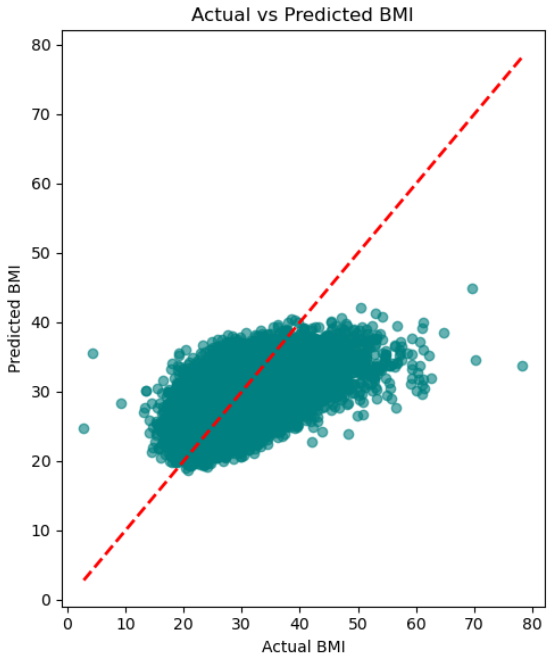
[116]:
```
'class LinearRegression:\n    def __init__(self, learning_rate=0.01,
epochs=1000):\n        self.learning_rate = learning_rate\n        self.epochs =
epochs\n        self.weights = None\n        self.bias = None\n\n    def
fit(self, xs, ys):\n        # Initialize weights and bias\n        n_samples,
n_features = len(xs), len(xs[0])\n        self.weights = [0] * n_features  #
Zero weights\n        self.bias = 0  # Zero bias\n\n        # Gradient Descent\n
for _ in range(self.epochs):\n            # Calculate predictions: y_pred = X.w
+ b\n            y_pred = [self._predict_single(x) for x in xs]\n\n            #
Compute gradients\n            dw = [0] * n_features\n            db = 0\n
for i in range(n_samples):\n                error = ys[i] - y_pred[i]\n
db += -2 * error  # Gradient of bias\n                for j in
range(n_features):\n                    dw[j] += -2 * error * xs[i][j]  #
Gradient of weights\n\n            # Average the gradients\n            db /=
n_samples\n            dw = [grad / n_samples for grad in dw]\n\n            #
Update weights and bias\n            self.bias -= self.learning_rate * db\n
self.weights = [w - self.learning_rate * grad for w, grad in zip(self.weights,
dw)]\n\n    def predict(self, xs):\n        # Predict for multiple samples\n
return [self._predict_single(x) for x in xs]\n\n    def _predict_single(self,
x):\n        # Predict for a single sample\n        return sum(w * xi for w, xi
in zip(self.weights, x)) + self.bias\n\nlr_model =
LinearRegression(learning_rate=0.01, epochs=1000)\nlr_model.fit(xs_train,
ys_train)\npredictions = lr_model.predict(xs_test)\n\nprint("MSE:
",mean_squared_error(predictions,ys_test))\nprint("MAE:
",mean_absolute_error(predictions,ys_test))\nprint("R2 score:
",r2_score(predictions,ys_test))\ncorr, p_value = pearsonr(y_true,
y_pred)\nprint("Pearson\'s correlation coefficient: ",corr)'
```

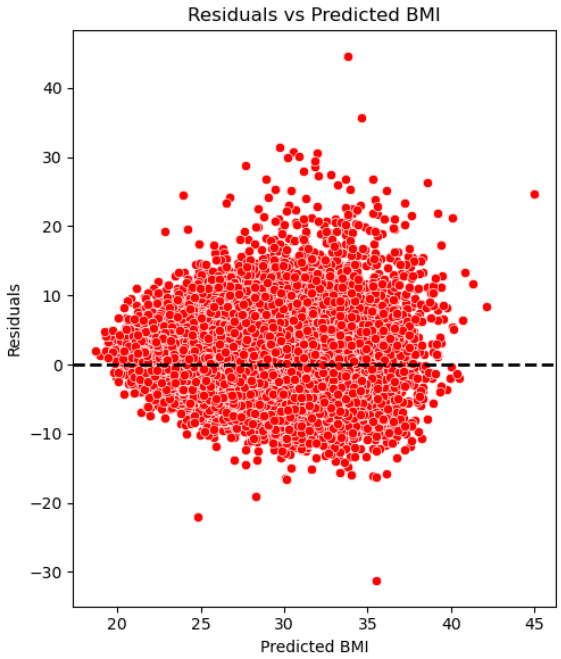
[117]:
```
#Linear Regression  -- Inbuilt

lr_model = LinearRegression()
lr_model.fit(xs_train, ys_train)
predictions = lr_model.predict(xs)
```

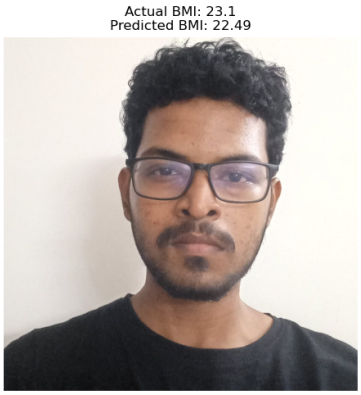
[118]:
```
mse = mean_squared_error(ys, predictions)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
mae = mean_absolute_error(ys, predictions)
print("Mean Absolute Error (MAE):", mae)
r2 = r2_score(ys, predictions)
```

```
print("R^2 Score:", r2)
pearson_corr, _ = pearsonr(ys, predictions)
print("Pearson Correlation Coefficient:", pearson_corr)
diagnostics()

our_preds = lr_model.predict(our_features)
imgdisplay(our_preds)
```

```
Root Mean Squared Error (RMSE): 4.153204989430456
Mean Absolute Error (MAE): 3.155997749708187
R^2 Score: 0.3441508998029317
Pearson Correlation Coefficient: 0.5867312724682521
```





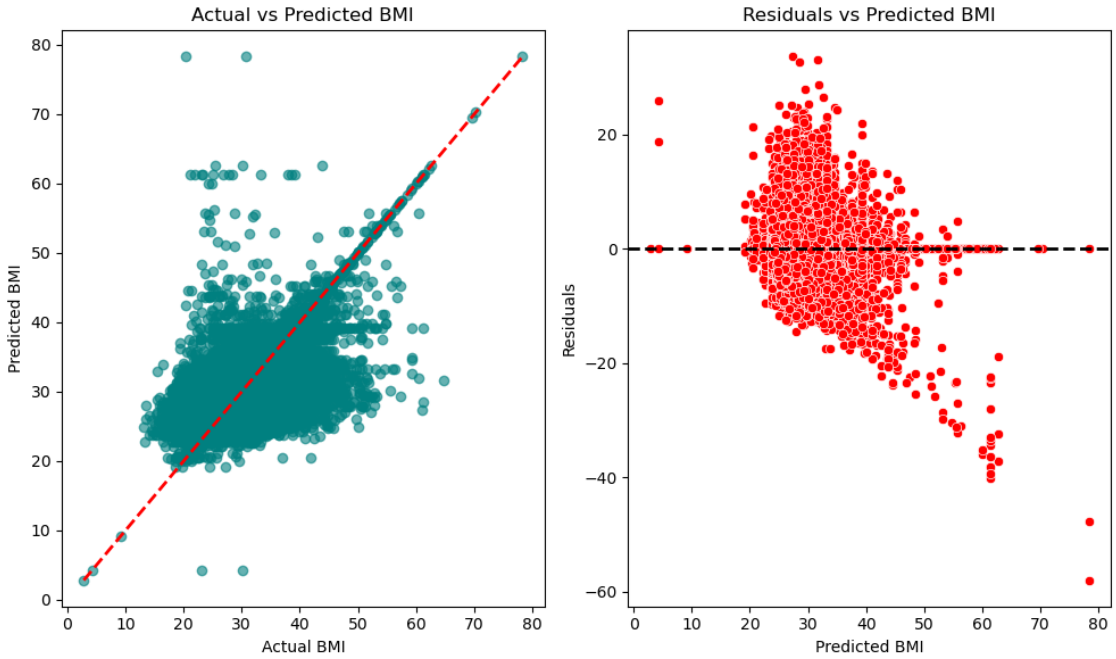


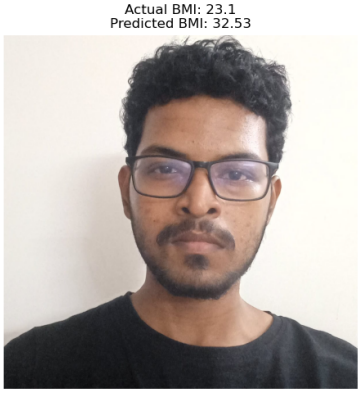
Decision Tree Regression

[119]:
```
#Decision Tree

tree_model = DecisionTreeRegressor(max_depth=10, random_state=42)
tree_model.fit(xs_train, ys_train)
predictions = tree_model.predict(xs)
```

[120]:
```
mse = mean_squared_error(ys, predictions)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
mae = mean_absolute_error(ys, predictions)
print("Mean Absolute Error (MAE):", mae)
r2 = r2_score(ys, predictions)
print("R^2 Score:", r2)
pearson_corr, _ = pearsonr(ys, predictions)
print("Pearson Correlation Coefficient:", pearson_corr)
diagnostics()

our_preds = tree_model.predict(our_features)
imgdisplay(our_preds)
```

```
Root Mean Squared Error (RMSE): 4.31561801479256
Mean Absolute Error (MAE): 3.2340569716840455
R^2 Score: 0.2918533784991194
Pearson Correlation Coefficient: 0.5491104578609672
```



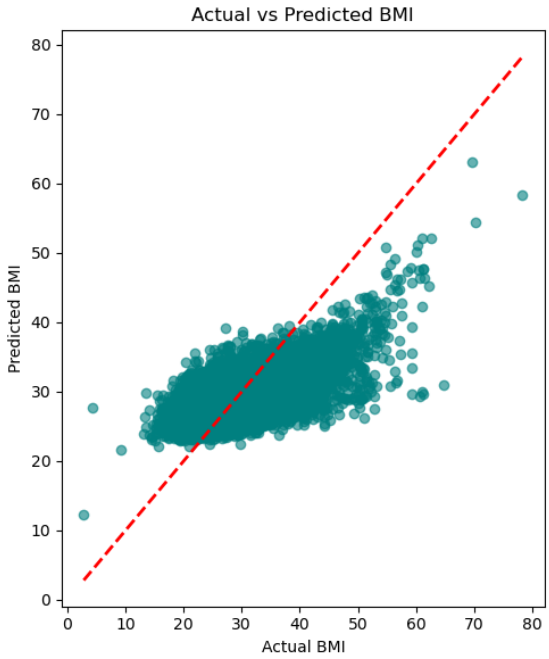


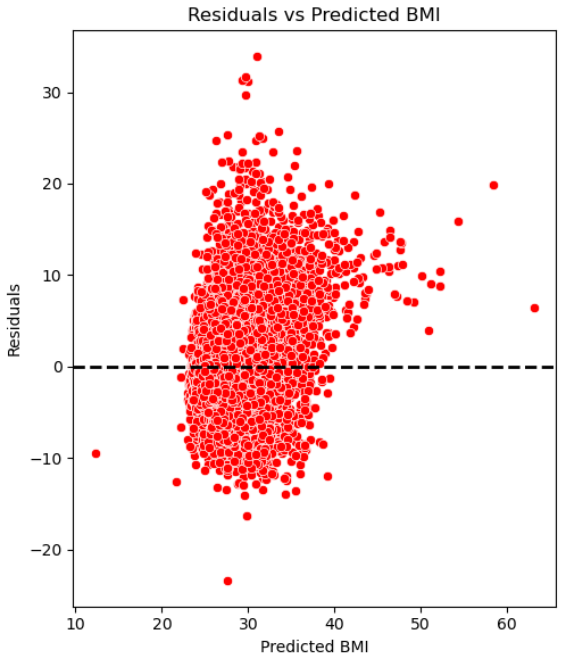
Random Forest Regression

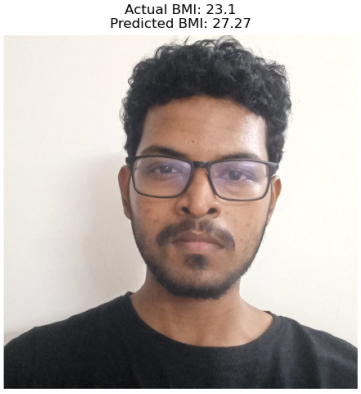
[121]:
```
#Random Forest
rf_model = RandomForestRegressor(n_estimators=10, max_depth=10, random_state=42)
rf_model.fit(xs_train, ys_train)
predictions = rf_model.predict(xs)
```

[122]:
```
mse = mean_squared_error(ys, predictions)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
mae = mean_absolute_error(ys, predictions)
print("Mean Absolute Error (MAE):", mae)
r2 = r2_score(ys, predictions)
print("R^2 Score:", r2)
pearson_corr, _ = pearsonr(ys, predictions)
print("Pearson Correlation Coefficient:", pearson_corr)
diagnostics()

our_preds = rf_model.predict(our_features)
imgdisplay(our_preds)
```

```
Root Mean Squared Error (RMSE): 3.9533769067127613
Mean Absolute Error (MAE): 3.0569944703227208
R^2 Score: 0.40574391840248303
Pearson Correlation Coefficient: 0.6686268883433503
```





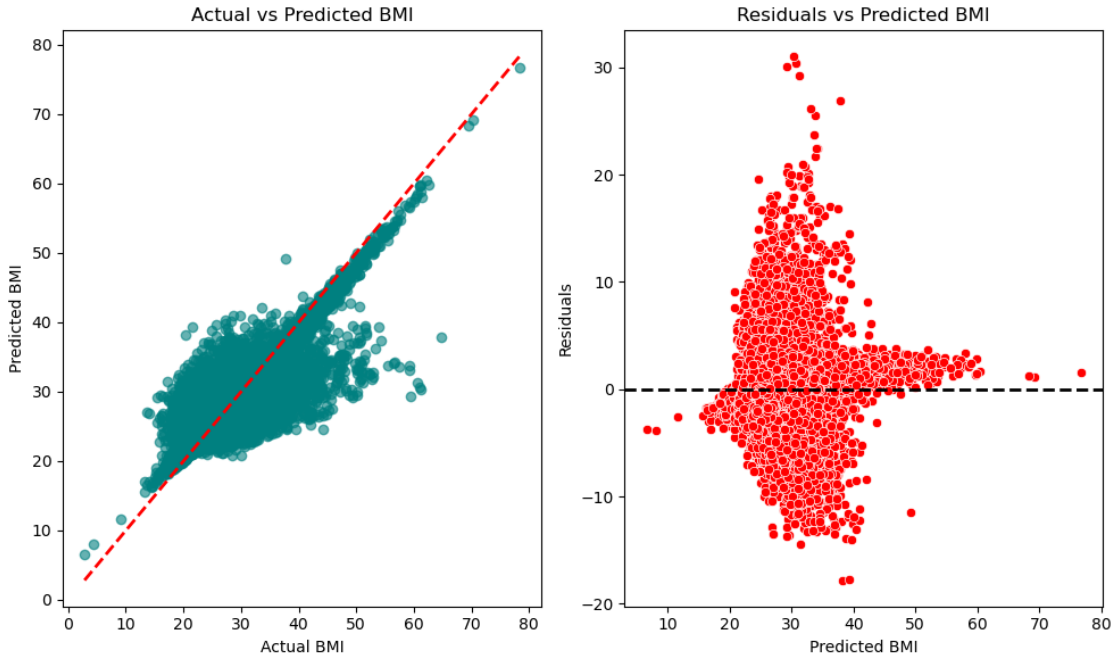


XG Boost Regression

[123]:
```
xgb_model = XGBRegressor(
    n_estimators=1000,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    random_state=42
)
xgb_model.fit(xs_train, ys_train)
predictions = xgb_model.predict(xs)
```

[124]:
```
mse = mean_squared_error(ys, predictions)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
mae = mean_absolute_error(ys, predictions)
print("Mean Absolute Error (MAE):", mae)
r2 = r2_score(ys, predictions)
print("R^2 Score:", r2)
pearson_corr, _ = pearsonr(ys, predictions)
print("Pearson Correlation Coefficient:", pearson_corr)
diagnostics()

our_preds = xgb_model.predict(our_features)
imgdisplay(our_preds)
```

```
Root Mean Squared Error (RMSE): 2.290183098716382
Mean Absolute Error (MAE): 1.3583405043203558
R^2 Score: 0.8005759171143986
Pearson Correlation Coefficient: 0.8988354661490192
```



Classification Into Obese, Normal or Underweight using Regression Values

[152]:
```
newpersons = filtered_df.copy()
cols = [i for i in range(1, 1025)]
newpersons.drop(newpersons.columns[cols],axis=1,inplace=True)
newpersons = newpersons[['id', 'name', 'weight', 'height', 'sex']]
newpersons['calculated bmi'] = (newpersons['weight']*703)/
 ↪(newpersons['height']**2)
newpersons.head()
```

[152]:

	id	name	weight	height	sex	calculated bmi
0	A00147	MCCUTCHEON, JOHN	185.0	67.0	Male	28.971931
1	A00360	BELL, HOWARD	167.0	69.0	Male	24.658895
2	A00367	GARVIN, RAYMOND	245.0	72.0	Male	33.224344
3	A01072	BRISBON, HENRY	195.0	69.0	Male	28.793321
4	A01077	JONES, ROBERT	180.0	68.0	Male	27.365917

Considering BMI>30 to be Obese and <18.5 to be Underweight

[153]:
```
def getclass(x):
    if x > 30: return "Obese"
    elif x > 18.5: return "Normal"
    else: return "Underweight"
vals = np.array(features.iloc[:,1:])
predicted_vals = xgb_model.predict(vals)
newpersons = newpersons.iloc[:len(predicted_vals)]
newpersons['predicted_bmi'] = predicted_vals
newpersons['weight class'] = newpersons['predicted_bmi'].apply(getclass)
newpersons
```

[153]:

	id	name	weight	height	sex	calculated bmi	\
0	A00147	MCCUTCHEON, JOHN	185.0	67.0	Male	28.971931	
1	A00360	BELL, HOWARD	167.0	69.0	Male	24.658895	
2	A00367	GARVIN, RAYMOND	245.0	72.0	Male	33.224344	

```
3      A01072       BRISBON, HENRY   195.0    69.0  Male       28.793321
4      A01077        JONES, ROBERT   180.0    68.0  Male       27.365917
…       …                   …      …     …   …              …
49995  Y16005   ALLEN, TERRANCE T.   169.0    71.0  Male       23.568141
49996  Y16006       EVANS, KENNETH   156.0    70.0  Male       22.381224
49997  Y16009   PHILLIPS, NEHEMIAH   150.0    72.0  Male       20.341435
49998  Y16011  CIESLOWSKI, PAUL T.   152.0    72.0  Male       20.612654
49999  Y16012  ALMARAZ, ROBERTO C.   160.0    69.0  Male       23.625289

       predicted_bmi weight class
0          28.908102       Normal
1          24.451565       Normal
2          31.511412        Obese
3          27.517553       Normal
4          25.842659       Normal
…                …           …
49995      24.061115       Normal
49996      27.334667       Normal
49997      23.795485       Normal
49998      26.680717       Normal
49999      24.108101       Normal

[50000 rows x 8 columns]
```

[154]:
```
print("Total Number: ", len(newpersons))
print("Number of Normal:", len(newpersons[newpersons['weight class'] ==␣
 ↪'Normal']))
print("Number of Underweight:", len(newpersons[newpersons['weight class'] ==␣
 ↪'Underweight']))
print("Number of Obese:", len(newpersons[newpersons['weight class'] ==␣
 ↪'Obese']))
```

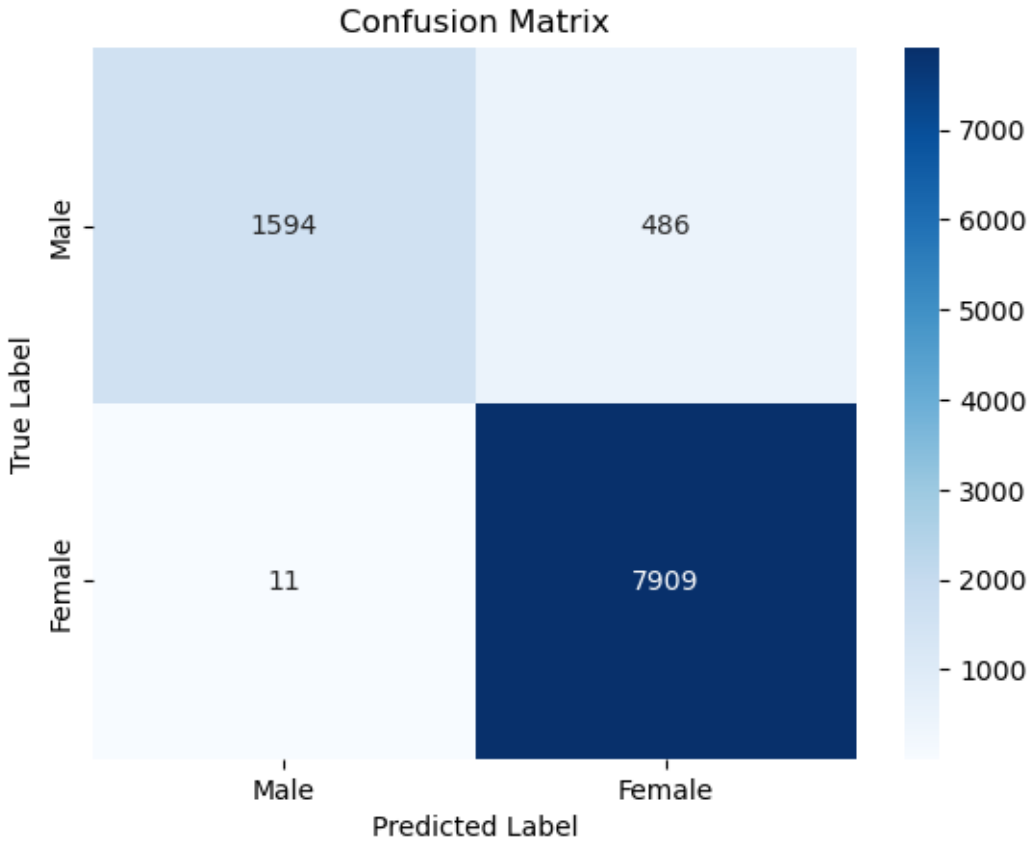
```
Total Number:  50000
Number of Normal: 37088
Number of Underweight: 31
Number of Obese: 12881
```

Logistic Regression for Gender Classification

[155]:
```
logreg = LogisticRegression()
yq_train = newpersons['sex'][:int(0.8*(len(newpersons)))]
yq_test = newpersons['sex'][int(0.8*(len(newpersons))):]
logreg.fit(xs_train, yq_train)

yq_pred = logreg.predict(xs_test)
accuracy = accuracy_score(yq_test, yq_pred)
print(f"Accuracy: {accuracy:.2f}")
```

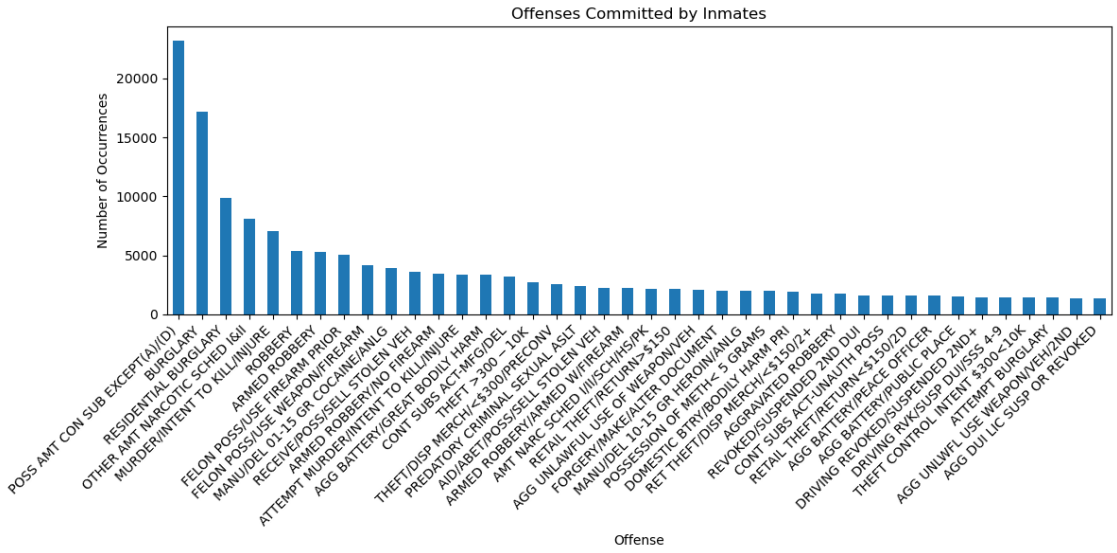
```
print("Classification Report:\n", classification_report(yq_test, yq_pred))

conf_matrix = confusion_matrix(yq_test, yq_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",␣
 ↪xticklabels=["Male", "Female"], yticklabels=["Male", "Female"])
plt.title("Confusion Matrix")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

      Female       0.99      0.77      0.87      2080
        Male       0.94      1.00      0.97      7920

    accuracy                           0.95     10000
   macro avg       0.97      0.88      0.92     10000
weighted avg       0.95      0.95      0.95     10000
```



Plotting of Offenses

[156]:
```
offenses = pd.read_csv('/media/kathir/Apps and Games/prml project/
 ↪illinois_doc_dataset/sentencing.csv', sep=';')
offenses = offenses['offense'].value_counts()
top_n = 40
n_offences = offenses.head(top_n)

# Plot the distribution
plt.figure(figsize=(12, 6))
n_offences.plot(kind='bar')
plt.title('Offenses Committed by Inmates')
plt.xlabel('Offense')
plt.ylabel('Number of Occurrences')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[ ]:

[ ]: