In [1]:
```
# import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import xgboost as xgb

from sklearn.metrics import precision_recall_curve

from yellowbrick.cluster import KElbowVisualizer

%matplotlib inline

plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams["figure.figsize"] = [12,10]
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Load data and see the race value counts
df = pd.read_csv("CombinedWaistDataWithCoordinates with weight.csv")
df_Front = pd.read_csv("CurveCoefficients.csv")
df.drop('Unnamed: 0', axis=1, inplace=True)
df_Front.drop('Unnamed: 0', axis=1, inplace=True)
print(df["Race"].value_counts());
```

```
White          397
AfricanAmer     66
Asian           56
Other           15
Hispanic         6
Name: Race, dtype: int64
```

In [3]:
```
# Merge curve coefficient into the original data through subject ID
df = pd.merge(df, df_Front, on = "Subject",  left_index=True, right_index=True, how=
df.isna().sum();
df = df.dropna();
df.reset_index(drop = True, inplace = True)
```

In [4]:
```
cols_x = [col for col in df.columns if col.endswith('_x')]
df = df.loc[:,~df.columns.isin(cols_x)]
df.columns
```

Out[4]:
```
Index(['Subject', 'weight', 'Curve_Coordinates', 'Race_y', 'BMI_y', 'Height_y',
       'Max_Hip_y', 'Anterior_posterior_Length_y', 'Depth_y',
```

```
       'Crotch_curve_length_at_back_waist_y', 'Front_Crotch_y',
       'Back_Crotch_y', 'Curve_Coord_Front', 'Curve_Coord_Back',
       'Coef_Front_2_3', 'intercept_Front_2_3', 'Coef_Back_2_3',
       'intercept_Back_2_3', 'Coef_Front_3_3', 'intercept_Front_3_3',
       'Coef_Back_3_3', 'intercept_Back_3_3', 'Coef_Front_2_4',
       'intercept_Front_2_4', 'Coef_Back_2_4', 'intercept_Back_2_4'],
      dtype='object')
```

In [5]:
```
# Create front and back curve coefficient columns.
df["f_coef_1"] = 1
df["f_coef_2"] = 1
df["f_coef_3"] = 1
df["f_coef_4"] = 1
df["b_coef_1"] = 1
df["b_coef_2"] = 1
df["b_coef_3"] = 1
df["b_coef_4"] = 1
```

In [6]:
```
# Add front curve coeffcient into the original data set
df["Coef_Front_2_3"];
for x in range(len(df)):
    df["f_coef_1"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[0]
    df["f_coef_2"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[1]
    df["f_coef_3"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[2]
    df["f_coef_4"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[3]

df["f_coef_1"] = df["f_coef_1"].astype("float64")
df["f_coef_2"] = df["f_coef_2"].astype("float64")
df["f_coef_3"] = df["f_coef_3"].astype("float64")
df["f_coef_4"] = df["f_coef_4"].astype("float64")

df["f_coef_1"] = df["f_coef_1"].apply(lambda x: format(x, '.3f'))
df["f_coef_2"] = df["f_coef_2"].apply(lambda x: format(x, '.3f'))
df["f_coef_3"] = df["f_coef_3"].apply(lambda x: format(x, '.3f'))
df["f_coef_4"] = df["f_coef_4"].apply(lambda x: format(x, '.3f'))
```

In [7]:
```
# Add back curve coeffcient into the original data set
df["Coef_Back_2_3"];
for x in range(len(df)):
    df["b_coef_1"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[0]
    df["b_coef_2"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[1]
    df["b_coef_3"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[2]
    df["b_coef_4"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[3]

df["b_coef_1"] = df["b_coef_1"].astype("float64")
df["b_coef_2"] = df["b_coef_2"].astype("float64")
df["b_coef_3"] = df["b_coef_3"].astype("float64")
df["b_coef_4"] = df["b_coef_4"].astype("float64")

df["b_coef_1"] = df["b_coef_1"].apply(lambda x: format(x, '.3f'))
df["b_coef_2"] = df["b_coef_2"].apply(lambda x: format(x, '.3f'))
df["b_coef_3"] = df["b_coef_3"].apply(lambda x: format(x, '.3f'))
df["b_coef_4"] = df["b_coef_4"].apply(lambda x: format(x, '.3f'))
```

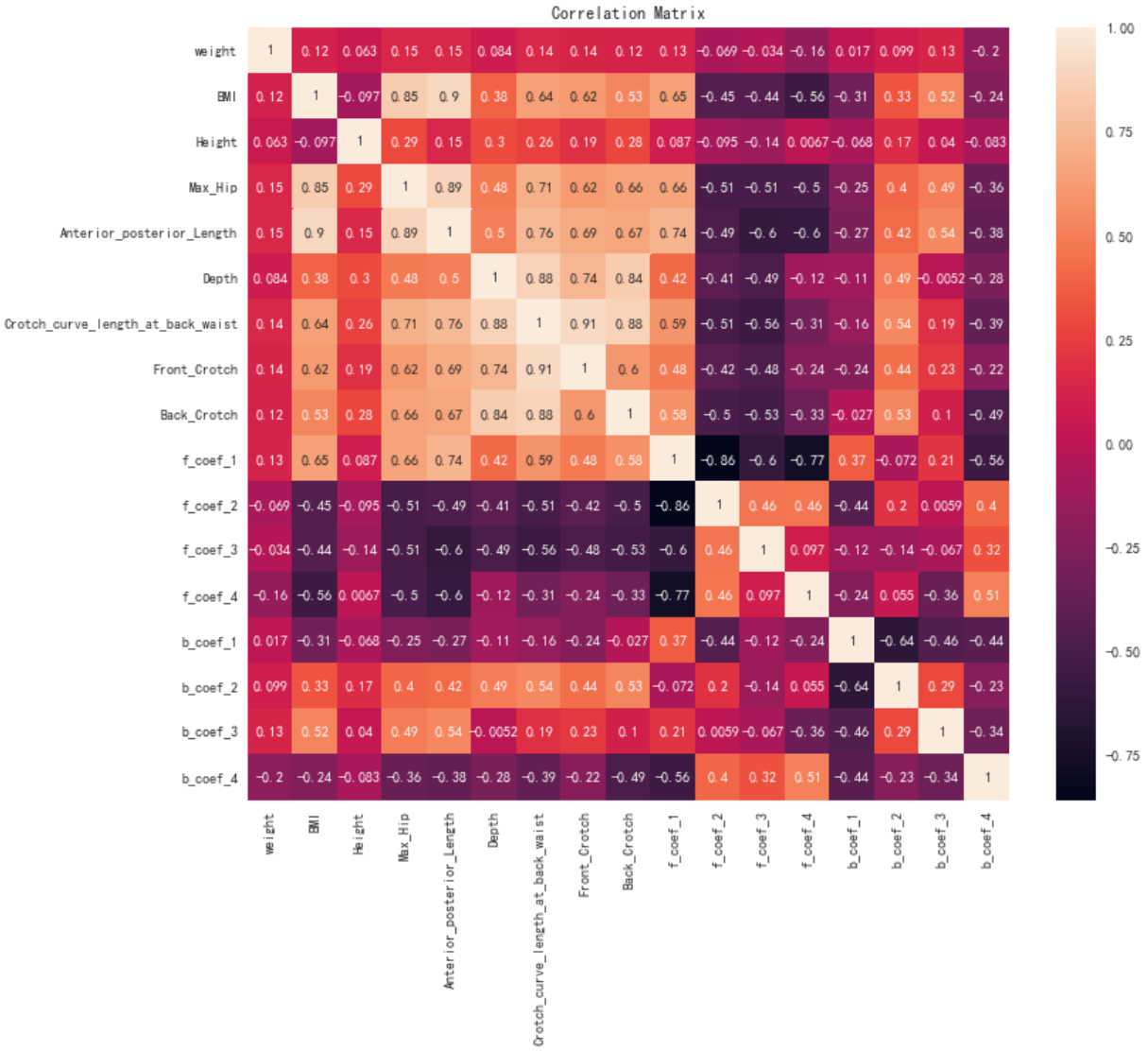
In [8]:
```
# Take out the columns we need
df = df.iloc[:,[0,1,3,4,5,6,7,8,9,10,11,26,27,28,29,30,31,32,33]]
df;
```

In [9]:
```
# Convert datatype to float
df["Height_y"] = df["Height_y"].astype("float64")
df["Max_Hip_y"] = df["Max_Hip_y"].astype("float64")
df["f_coef_1"] = df["f_coef_1"].astype("float64")
df["f_coef_2"] = df["f_coef_2"].astype("float64")
```

```
df["f_coef_3"] = df["f_coef_3"].astype("float64")
df["f_coef_4"] = df["f_coef_4"].astype("float64")
df["b_coef_1"] = df["b_coef_1"].astype("float64")
df["b_coef_2"] = df["b_coef_2"].astype("float64")
df["b_coef_3"] = df["b_coef_3"].astype("float64")
df["b_coef_4"] = df["b_coef_4"].astype("float64")
```

In [10]:
```
df.rename(columns = {"BMI_y":"BMI","Height_y":"Height","Max_Hip_y":"Max_Hip","Anteri
                     "Depth_y":"Depth","Crotch_curve_length_at_back_waist_y":"Crotch
                     "Back_Crotch_y":"Back_Crotch","Race_y":"Race"}, inplace = True)
```

In [11]:
```
# Correlation heatmap
corr = df.iloc[:,[1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]].corr()
sns.heatmap(corr, annot=True, xticklabels=corr.columns, yticklabels=corr.columns)
plt.title("Correlation Matrix");
```



In [12]:
```
# Only look at the White and African American people's data
df = df[(df["Race"]=="White") | (df["Race"]=="AfricanAmer")]
df.reset_index(inplace = True, drop = True)
df;
```

In [13]:
```
df2 = df.iloc[:,[1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]]
df2
```

Out[13]:

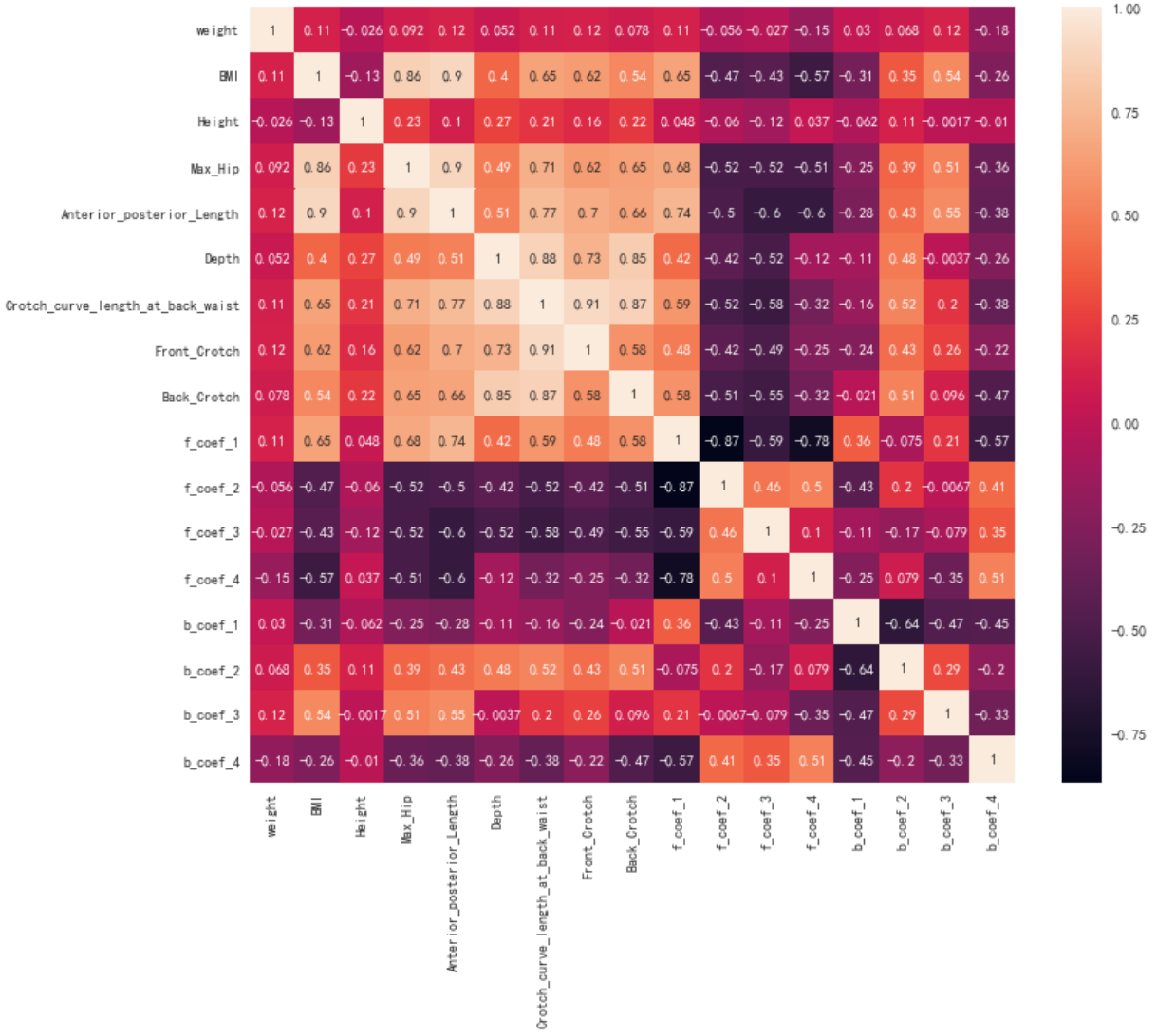
	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_length_a
0	76.19	21.400000	1601.0	934.0	254.2900	245.0120	

	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_length_a
1	74.06	26.000000	1685.0	1097.0	281.7800	234.3100	
2	85.92	32.900000	1644.0	1231.0	348.9300	249.7300	
3	94.30	23.700000	1603.0	1020.0	277.1100	313.6700	
4	78.04	24.100000	1554.0	1034.0	265.9500	267.7900	
...	...	...	...	...	...	...	
457	77.37	19.309389	1710.0	920.0	238.5356	281.6155	
458	79.68	22.761793	1685.0	1020.0	278.1208	216.5177	
459	75.14	28.309274	1601.0	1062.0	291.3675	228.6181	
460	80.39	21.376065	1769.0	1004.0	249.2301	211.0781	
461	85.18	23.999562	1692.0	1046.0	268.1176	243.5517	

462 rows × 17 columns

In [14]:
```
# Correlation heatmap
corr = df2.corr()
sns.heatmap(corr, annot=True, xticklabels=corr.columns, yticklabels=corr.columns)
```

Out[14]: <AxesSubplot:>



PCA AND K-Means

In [15]:
```
df2
```

Out[15]:

	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_length_a
0	76.19	21.400000	1601.0	934.0	254.2900	245.0120	
1	74.06	26.000000	1685.0	1097.0	281.7800	234.3100	
2	85.92	32.900000	1644.0	1231.0	348.9300	249.7300	
3	94.30	23.700000	1603.0	1020.0	277.1100	313.6700	
4	78.04	24.100000	1554.0	1034.0	265.9500	267.7900	
...	...	...	...	...	...	...	
457	77.37	19.309389	1710.0	920.0	238.5356	281.6155	
458	79.68	22.761793	1685.0	1020.0	278.1208	216.5177	
459	75.14	28.309274	1601.0	1062.0	291.3675	228.6181	
460	80.39	21.376065	1769.0	1004.0	249.2301	211.0781	
461	85.18	23.999562	1692.0	1046.0	268.1176	243.5517	

462 rows × 17 columns

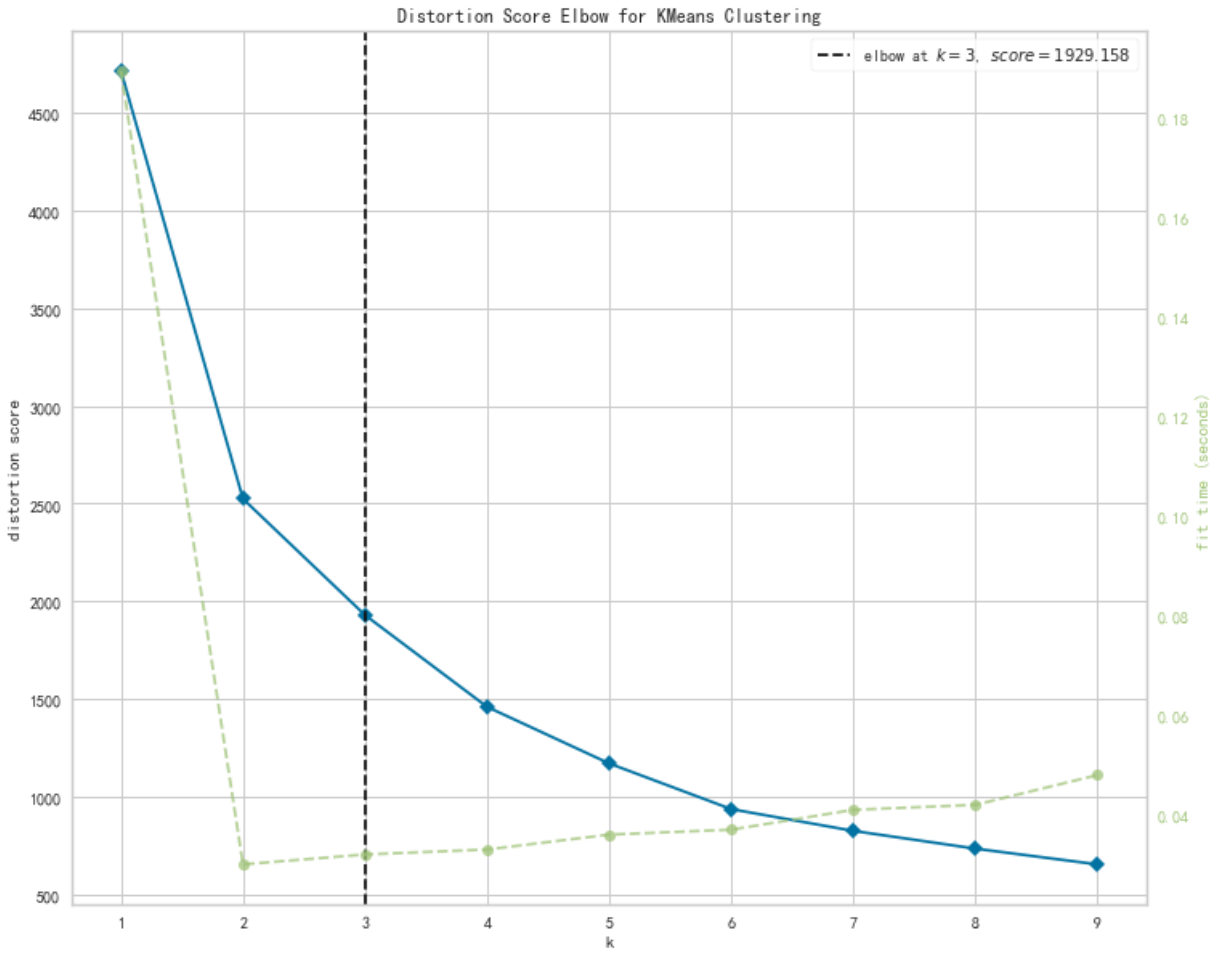
In [16]:
```
# Standardize the data
standard = preprocessing.scale(df2.values)
print(standard)
```

```
[[-0.54545592 -0.51668203 -0.78087315 ...  0.50804234  0.54779056
  -0.98278968]
 [-0.84160175  0.5933507   0.36870862 ... -1.01766718  0.75108099
  -0.50506977]
 [ 0.80736047  2.2583998  -0.19239676 ... -0.81379013  0.07906757
  -0.40841283]
 ...
 [-0.6914433   1.15060499 -0.78087315 ... -1.29576731  0.0840276
   0.59700984]
 [ 0.0384936  -0.52245782  1.5182904  ... -1.31981889 -0.52315113
  -0.36880755]
 [ 0.70447413  0.11062208  0.46450711 ...  0.57251476  0.21610615
   1.15604591]]
```

In [17]:
```
# Do PCA to reduce dimensional to 2
PCA_x = PCA(n_components = 2).fit_transform(standard)
```

In [18]:
```
PCA_r = pd.DataFrame(PCA_x)
PCA_r["Race"] = df["Race"]
```

In [19]:
```
# Elbow method to get the optimal value of K for K-means
km = KMeans(random_state=42)
visualizer = KElbowVisualizer(km, k=(1,10))

visualizer.fit(PCA_x)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```

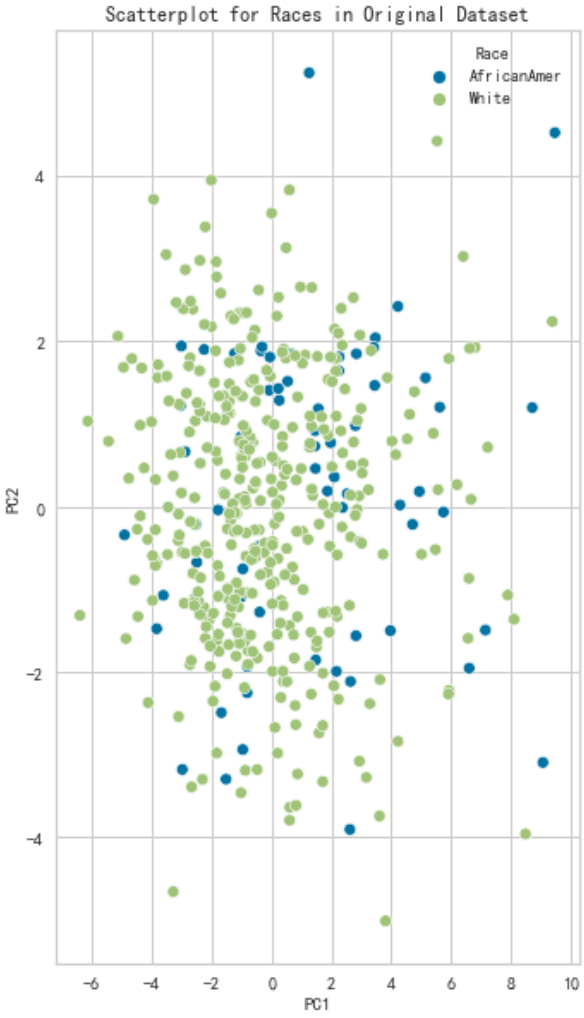


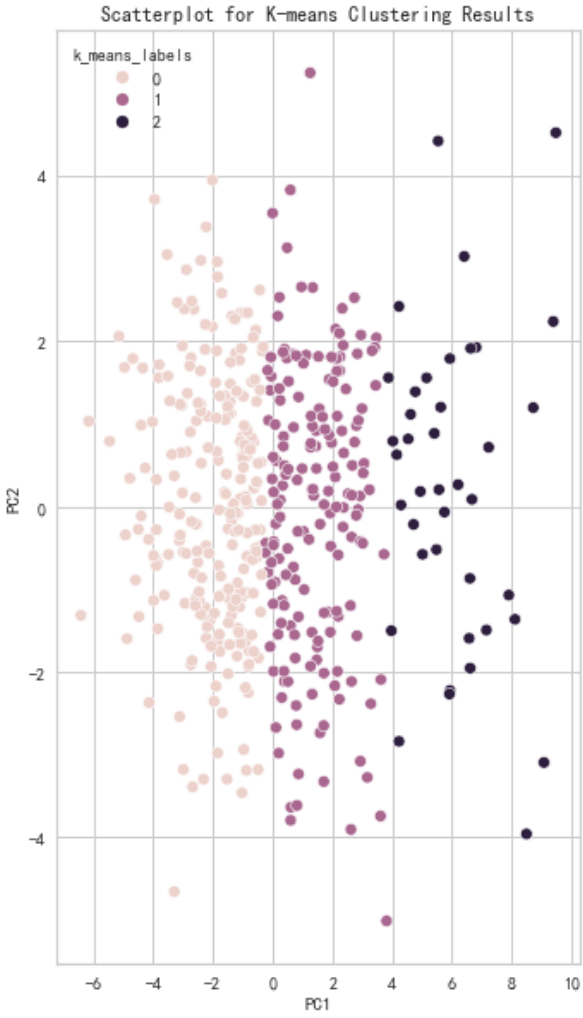
Out[19]: <AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>

In [20]:
```
# Do the K-means
clf = KMeans(n_clusters=3, init = "k-means++", max_iter = 100, random_state=0).fit(P
clf.cluster_centers_;
```

In [21]:
```
df3 = pd.DataFrame(PCA_x)
df3["k_means_labels"] = clf.labels_
df3['Race'] = PCA_r["Race"]
df_1 = df3[df3['k_means_labels'] == 0]
df_2 = df3[df3['k_means_labels'] == 1]
df3;
```

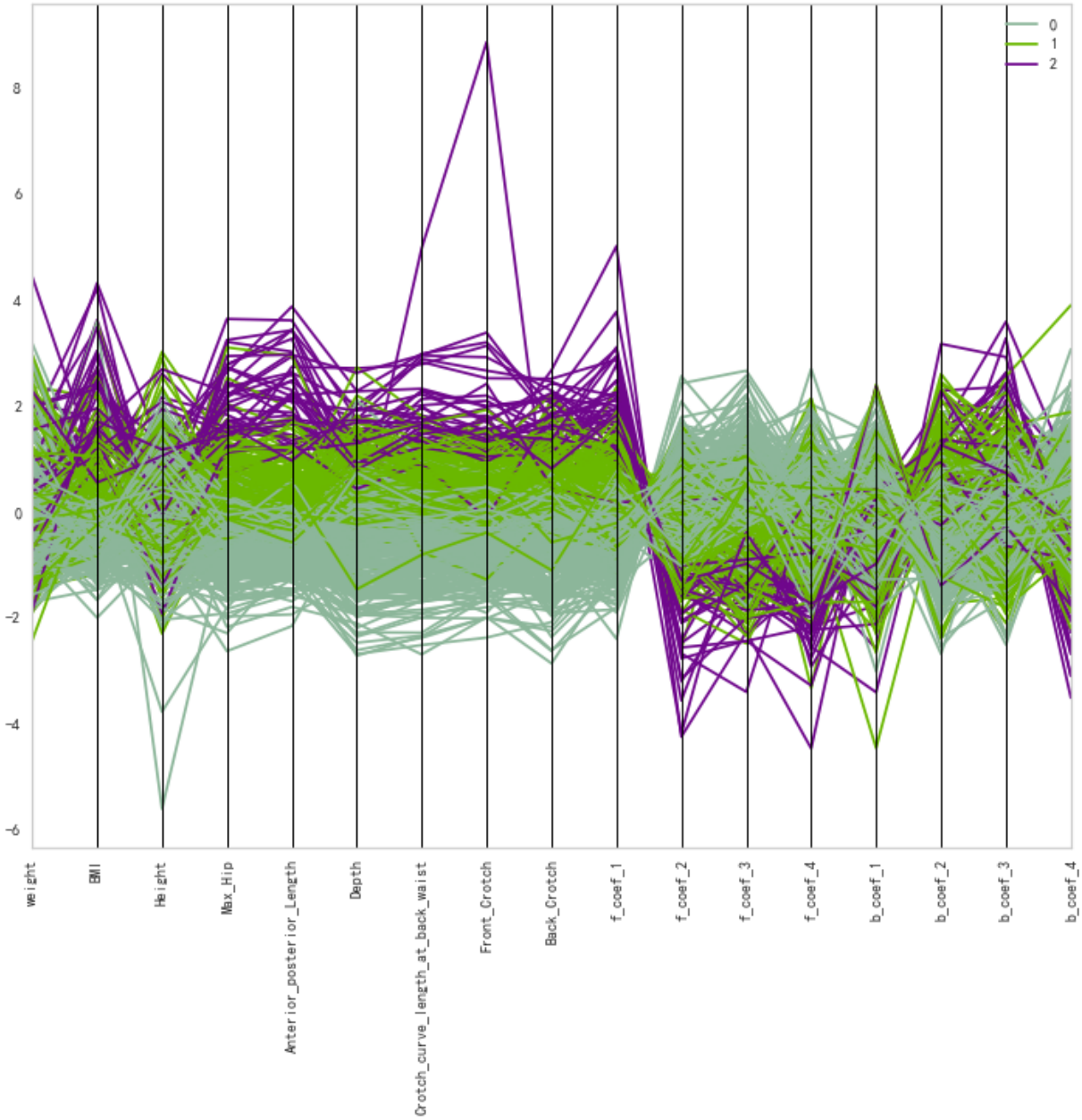
In [22]:
```
# Visualize the K-means results
fig = plt.figure()
ax1 = plt.subplot(121)
sns.scatterplot(df3[0], df3[1], hue = df3["Race"], ax = ax1)
plt.title("Scatterplot for Races in Original Dataset")
plt.xlabel("PC1")
plt.ylabel("PC2")

ax2 = plt.subplot(122)
sns.scatterplot(df3[0], df3[1], hue = df3["k_means_labels"], ax = ax2)
plt.title("Scatterplot for K-means Clustering Results")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```





In [23]:
```
X_clustered = pd.DataFrame(standard, index = df2.index, columns = df2.columns)
X_clustered["cluster"] = clf.labels_
```

In [24]:
```
# See the characteristics of each cluster
pd.plotting.parallel_coordinates(X_clustered,"cluster")
plt.xticks(rotation=90)
plt.show()
```



Heat Maps

In [25]:
```
standard_2 = preprocessing.scale(df.iloc[:,[1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
standard_2 = pd.DataFrame(standard_2)
standard_2 = standard_2.rename(columns={0:"weight",1:"BMI",2:"Height",3:"Max_Hip",4:
                                        5:"Depth",6:"Crotch_curve_length_at_back_waist",
                                        9:"f_coef_1", 10:"f_coef_2", 11:"f_coef_3", 12:"
                                        15:"b_coef_3", 16:"b_coef_4"})

standard_2;
standard_2['Race'] = df['Race'];
standard_2
```

Out[25]:

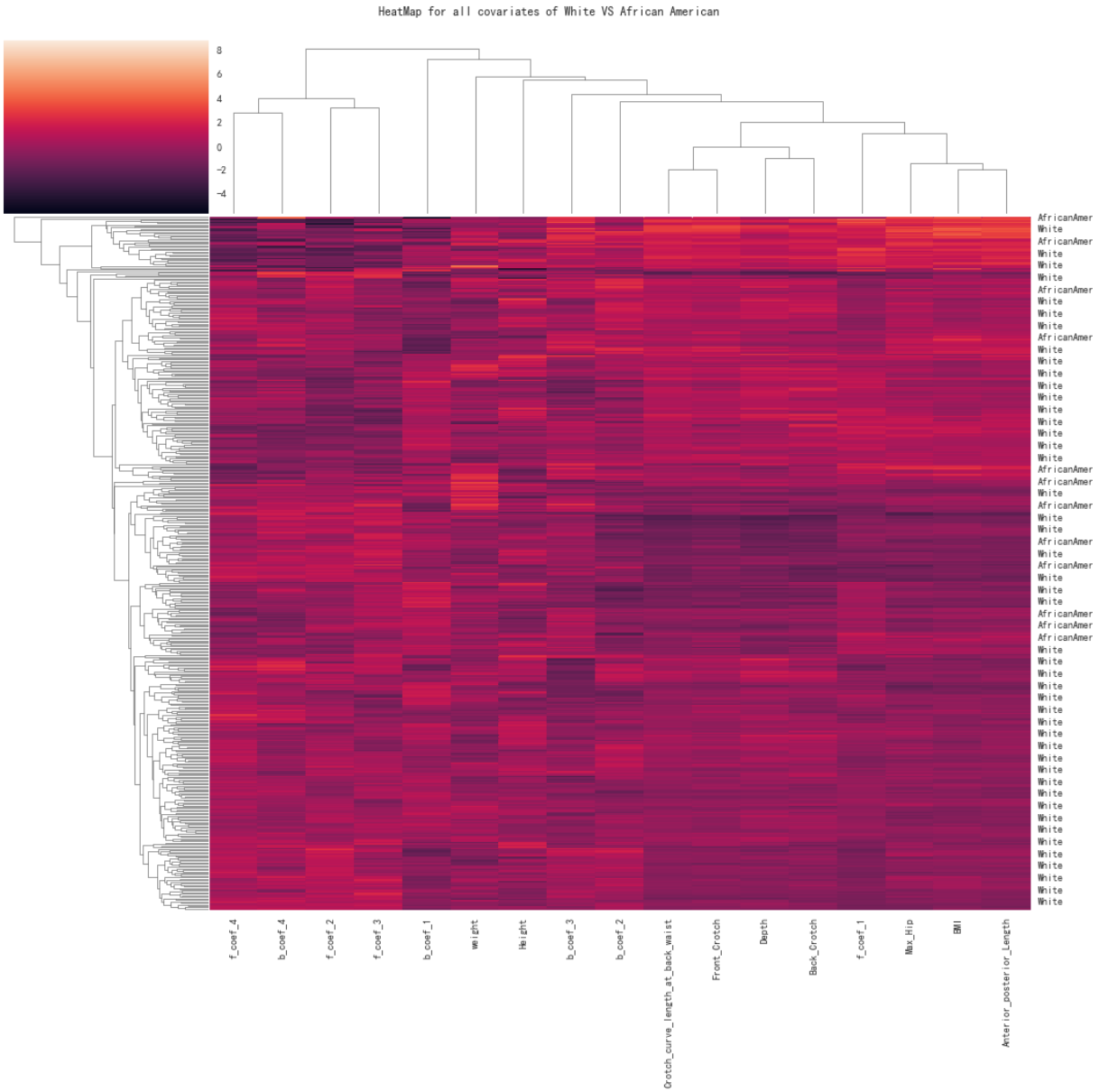
	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_le
0	-0.545456	-0.516682	-0.780873	-1.065765	-0.567946	-0.782420	
1	-0.841602	0.593351	0.368709	0.839436	0.206696	-1.227374	
2	0.807360	2.258400	-0.192397	2.405675	2.098918	-0.586262	
3	1.972479	0.038334	-0.753502	-0.060567	0.075099	2.072152	
4	-0.288240	0.134859	-1.424092	0.103070	-0.239379	0.164613	

	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_le
...	...	...	...	...	...	...	
457	-0.381394	-1.021170	0.710846	-1.229402	-1.011890	0.739432	
458	-0.060222	-0.188066	0.368709	-0.060567	0.103583	-1.967119	
459	-0.691443	1.150605	-0.780873	0.430344	0.476862	-1.464024	
460	0.038494	-0.522458	1.518290	-0.247580	-0.710529	-2.193279	
461	0.704474	0.110622	0.464507	0.243330	-0.178298	-0.843135	

462 rows × 18 columns

In [26]:
```
row_names = df['Race'].values.tolist()
standard_2.index = row_names
standard_2;
```

In [27]:
```
# Do the heatmap
g = sns.clustermap(standard_2.iloc[:,:-1], figsize=(15, 15))
g.fig.suptitle("HeatMap for all covariates of White VS African American")
g.fig.subplots_adjust(top=0.95)
```



Logistics Regression

In [28]:
```
df['Race_label'] = df['Race'].map({'AfricanAmer':0,'White':1})
df["Race"].value_counts()
```

Out[28]:
```
White          396
AfricanAmer     66
Name: Race, dtype: int64
```

In [29]:
```
X = df.drop(columns = ["Race","Race_label","Subject"], axis=1)
y = df["Race_label"]
```

In [30]:
```
# Use SMOTE method to figure out the data imbalanced problem
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_s
```

In [31]:
```
y_train.value_counts()
```

Out[31]:
```
1    320
0    313
Name: Race_label, dtype: int64
```

In [32]:
```
# Do the logistic regression
logistics = LogisticRegression(random_state = 42).fit(X_train,y_train)
y_pred = logistics.predict(X_test)
y_pred
```

Out[32]:
```
array([0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1], dtype=int64)
```

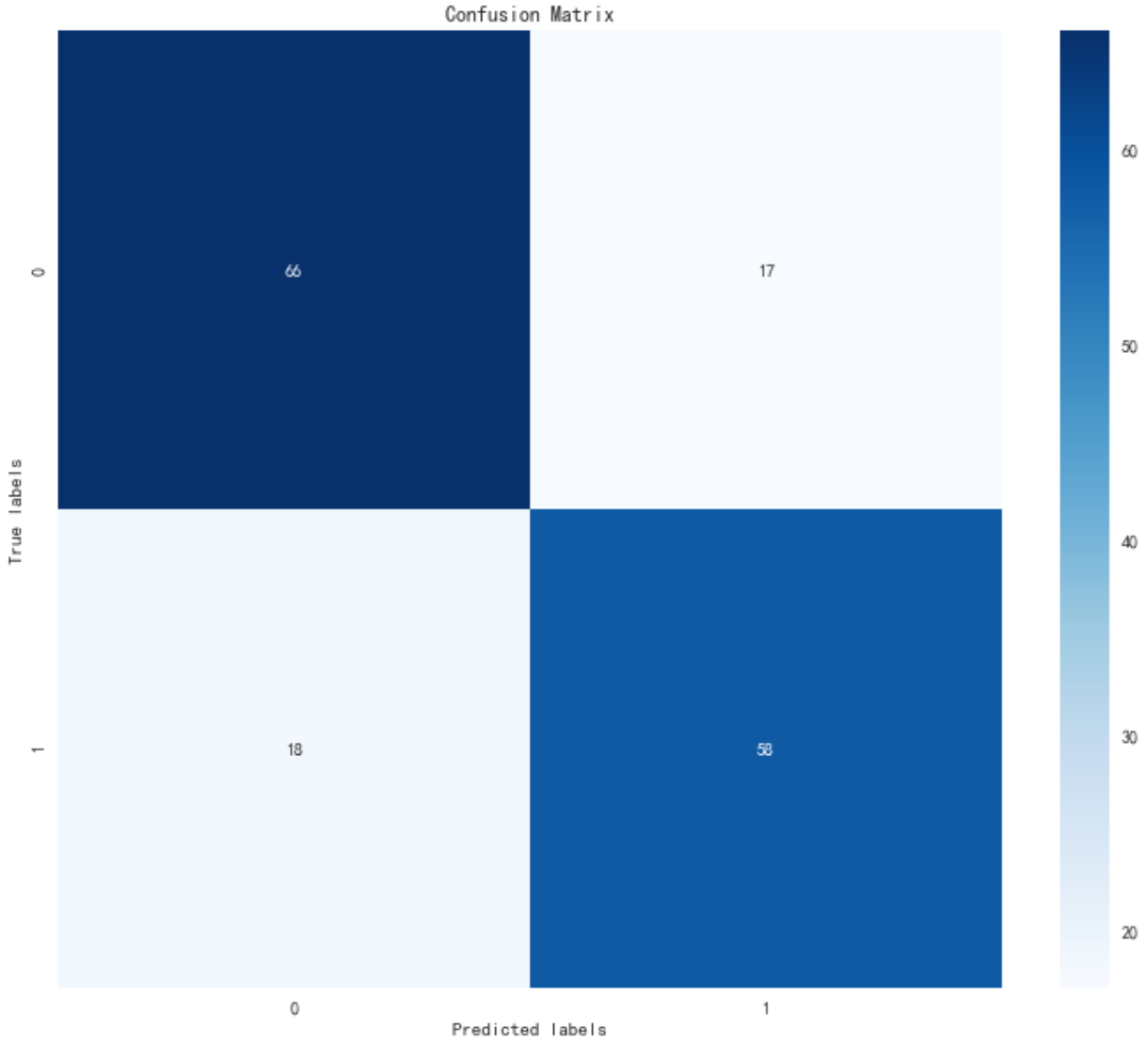
In [33]:
```
Logistic_result = pd.DataFrame(y_test)
Logistic_result = pd.DataFrame(np.concatenate((Logistic_result,pd.DataFrame(y_pred))
Logistic_result.rename(columns={0:"Original",1:"Predicted"},inplace = True)
```

In [34]:
```
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.80      0.79        83
           1       0.77      0.76      0.77        76

    accuracy                           0.78       159
   macro avg       0.78      0.78      0.78       159
weighted avg       0.78      0.78      0.78       159
```

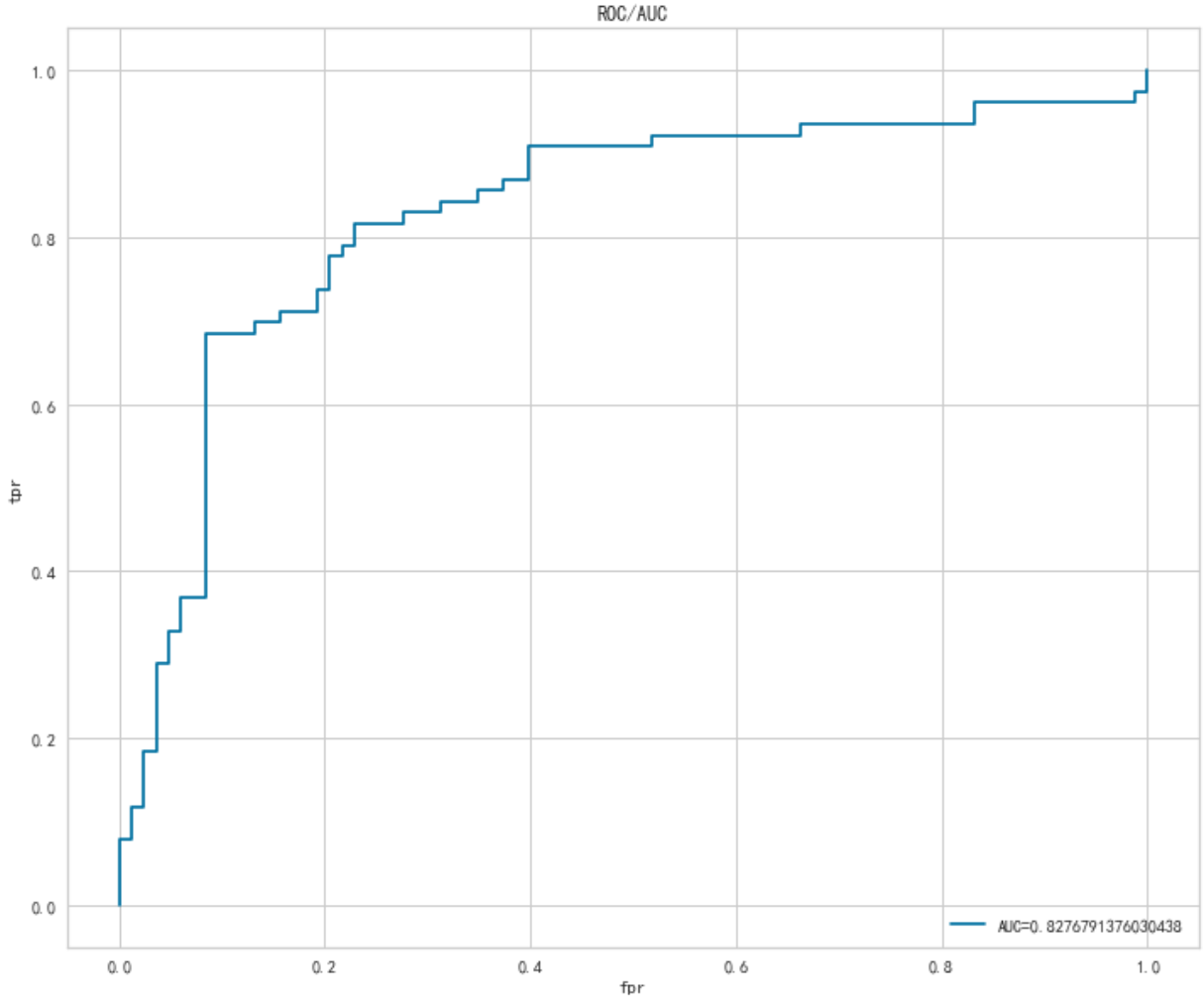
In [35]:
```
# Create the confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# Add axis labels and title
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```

```
plt.title('Confusion Matrix')
plt.show()
```



In [36]:

```
# Display ROC/AUC graph for the logistic regression
y_pred_proba = logistics.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.legend(loc=4)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC/AUC')
plt.show()
```



In [39]:
```
thresholds
```

Out[39]:
```
array([0, 1], dtype=int64)
```

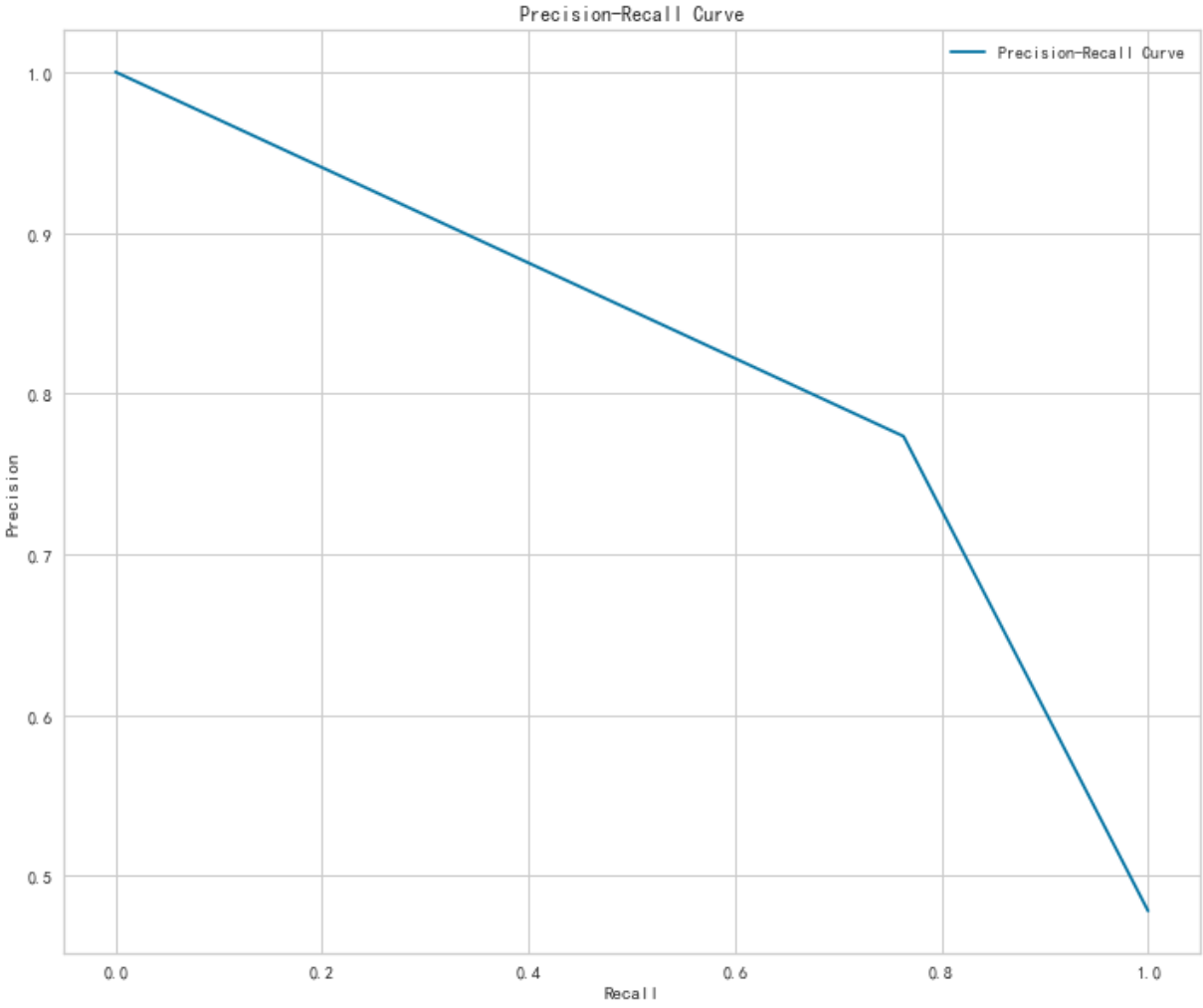
In [40]:
```
precision
```

Out[40]:
```
array([0.47798742, 0.77333333, 1.        ])
```

In [41]:
```
recall
```

Out[41]:
```
array([1.        , 0.76315789, 0.        ])
```

In [38]:
```
# Display Precison Recall graph for the logistic regression
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label='Precision-Recall Curve')
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



Random Forest

In [399…

```
# Do the random forest
rf = RandomForestClassifier()
n_estimators = [100, 300, 500, 1000]
max_features = ['auto', 'sqrt']
max_depth = [10, 30, 50]
min_samples_split = [2, 5, 8]
min_samples_leaf = [1, 2 , 3]
```

In [400…

```
# Do cross validation for each combination of hyperparameter
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter=100, cv=5, verbose=3, random_state=42, n_jobs=
rf_random.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[400…

▸ **RandomizedSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

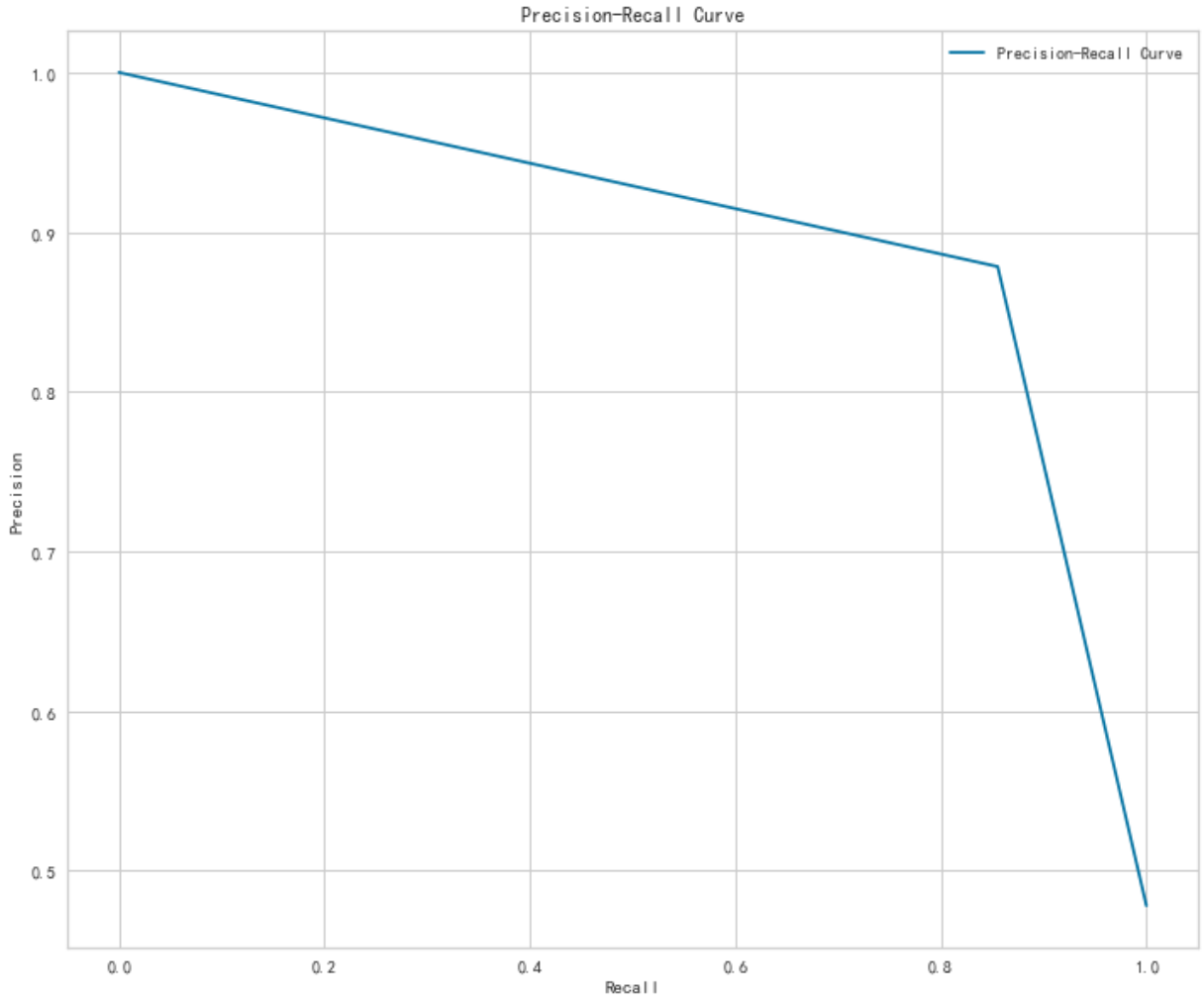
In [401…

```
print('Best hyperparameters:', rf_random.best_params_)
y_pred = rf_random.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Best hyperparameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_lea
f': 1, 'max_features': 'sqrt', 'max_depth': 50}
Accuracy: 0.8742138364779874
```

In [402…

```
# Display Precison Recall graph for the random forest
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label='Precision-Recall Curve')
plt.title("Precision-Recall Curve")
# set axis labels and legend
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



In [403…

```
rf_hp = pd.DataFrame(rf_random.best_estimator_.feature_importances_, index = X_train
```

In [404…

```
# Show the feature importance for random forest fit
rf_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[404…

	feature_importances
b_coef_3	0.199013

	feature_importances
b_coef_4	0.105003
f_coef_4	0.086917
Depth	0.072752
Anterior_posterior_Length	0.058878
BMI	0.051467
Height	0.049448
weight	0.045973
Crotch_curve_length_at_back_waist	0.045769
b_coef_2	0.044281
f_coef_1	0.042012
Back_Crotch	0.040262
b_coef_1	0.038592
f_coef_3	0.032383
Max_Hip	0.029450
Front_Crotch	0.029337
f_coef_2	0.028462

XGboost

In [405…

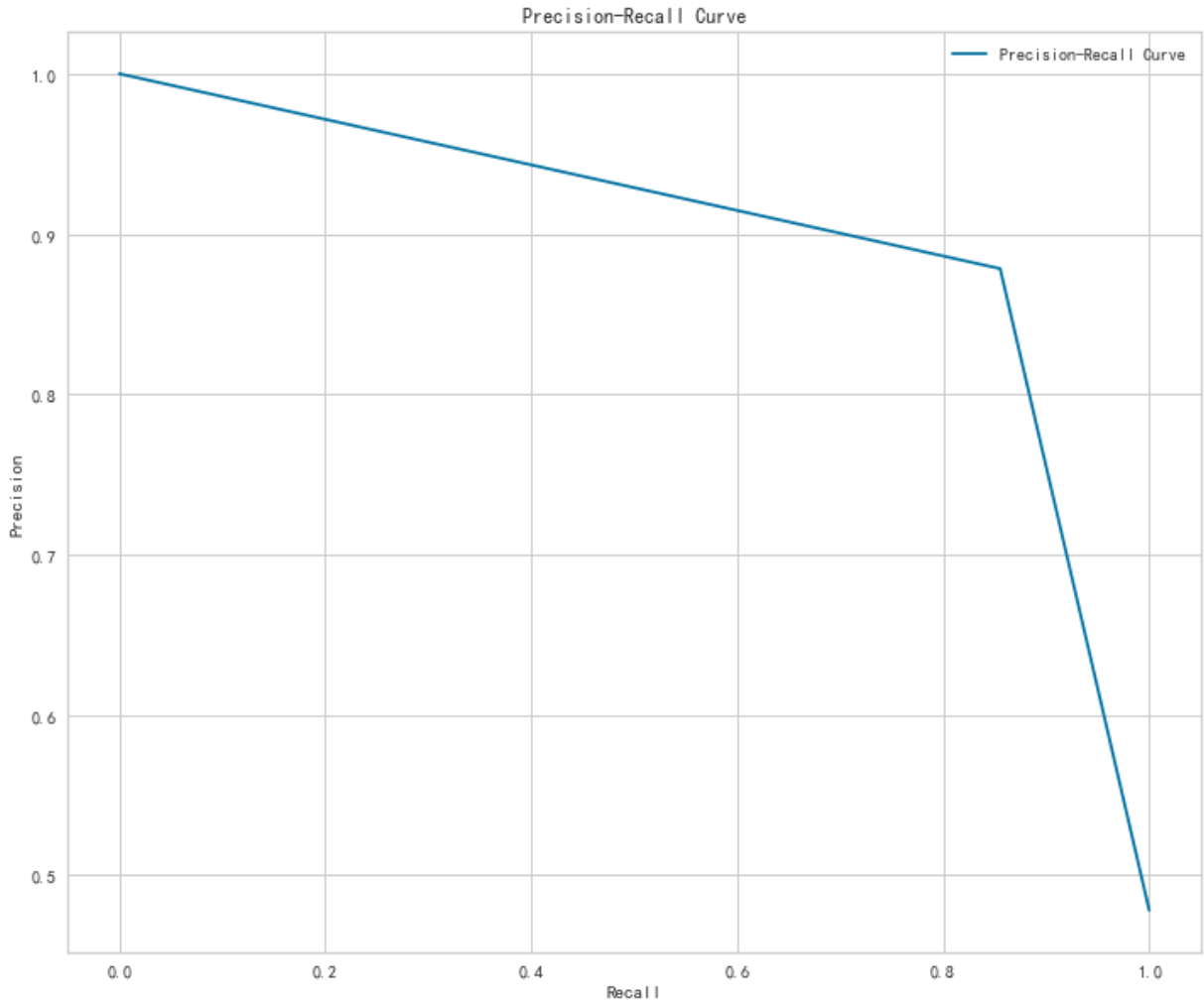
```
# Do the XGBoost model
xgb_model = xgb.XGBClassifier()

# Define the hyperparameters to tune
param_grid = {
    'max_depth': [3, 4, 5, 10],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 300, 500, 1000]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, verbose
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and score
print('Best parameters:', grid_search.best_params_)

# Make predictions on the test set with the best model
best_model = grid_search.best_estimator_
preds = best_model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best parameters: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 1000}
Accuracy: 0.8742138364779874
```

In [406…

```
# Display Precison Recall graph for the XGBoost model
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
```

```
plt.plot(recall, precision, label='Precision-Recall Curve')
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



In [407…
```
xgb_hp = pd.DataFrame(grid_search.best_estimator_.feature_importances_, index = X_tr
```

In [408…
```
# Show the feature importance for the XGBoost fit
xgb_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[408…

	feature_importances
b_coef_3	0.230318
b_coef_4	0.085182
f_coef_4	0.078491
Depth	0.072858
b_coef_2	0.066037
f_coef_1	0.058607
Height	0.056326
BMI	0.049110
f_coef_3	0.043781

	feature_importances
Crotch_curve_length_at_back_waist	0.042259
Back_Crotch	0.038142
Anterior_posterior_Length	0.037886
weight	0.034895
f_coef_2	0.034208
Front_Crotch	0.031730
b_coef_1	0.020436
Max_Hip	0.019734

In [ ]:

In [ ]:

Do the same thing for imbalanced data

In [409…
```
df['Race_label'] = df['Race'].map({'AfricanAmer':0,'White':1})
df["Race"].value_counts()
```

Out[409…
```
White          396
AfricanAmer     66
Name: Race, dtype: int64
```

In [410…
```
X = df.drop(columns = ["Race","Race_label","Subject"], axis=1)
y = df["Race_label"]
```

In [411…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

In [412…
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

In [413…
```
y_resampled.value_counts()
```

Out[413…
```
1    318
0    318
Name: Race_label, dtype: int64
```

In [414…
```
logistics = LogisticRegression(random_state = 42).fit(X_resampled,y_resampled)
y_pred = logistics.predict(X_test)
y_pred
```

Out[414…
```
array([1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 1], dtype=int64)
```

In [415…
```
Logistic_result = pd.DataFrame(y_test)
Logistic_result = pd.DataFrame(np.concatenate((Logistic_result,pd.DataFrame(y_pred))
Logistic_result.rename(columns={0:"Original",1:"Predicted"},inplace = True)
```

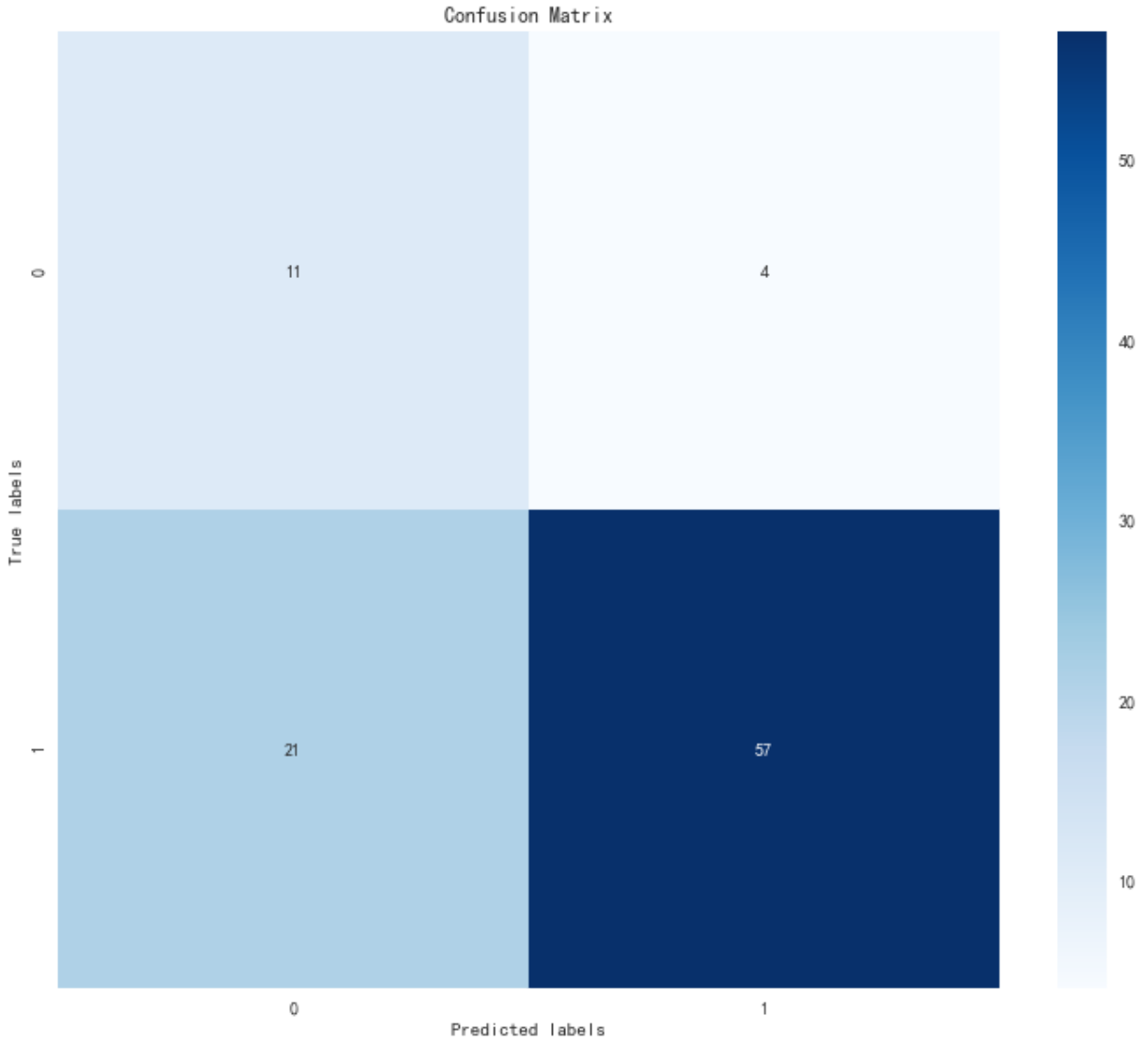
In [416…
```
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.34      0.73      0.47        15
              1       0.93      0.73      0.82        78

    accuracy                            0.73        93
   macro avg       0.64      0.73      0.64        93
weighted avg       0.84      0.73      0.76        93
```

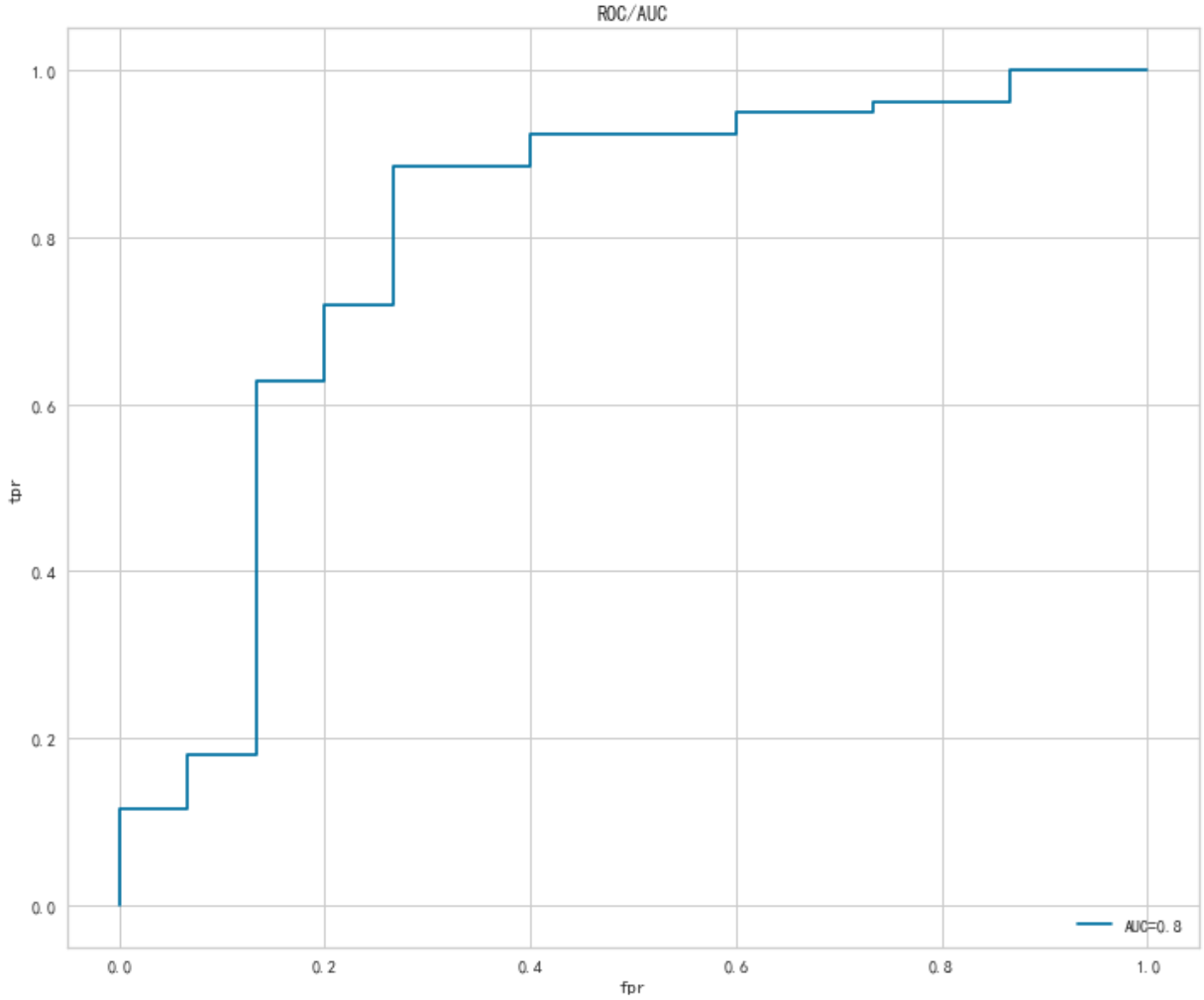
In [417…

```python
# Create the confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# Add axis labels and title
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



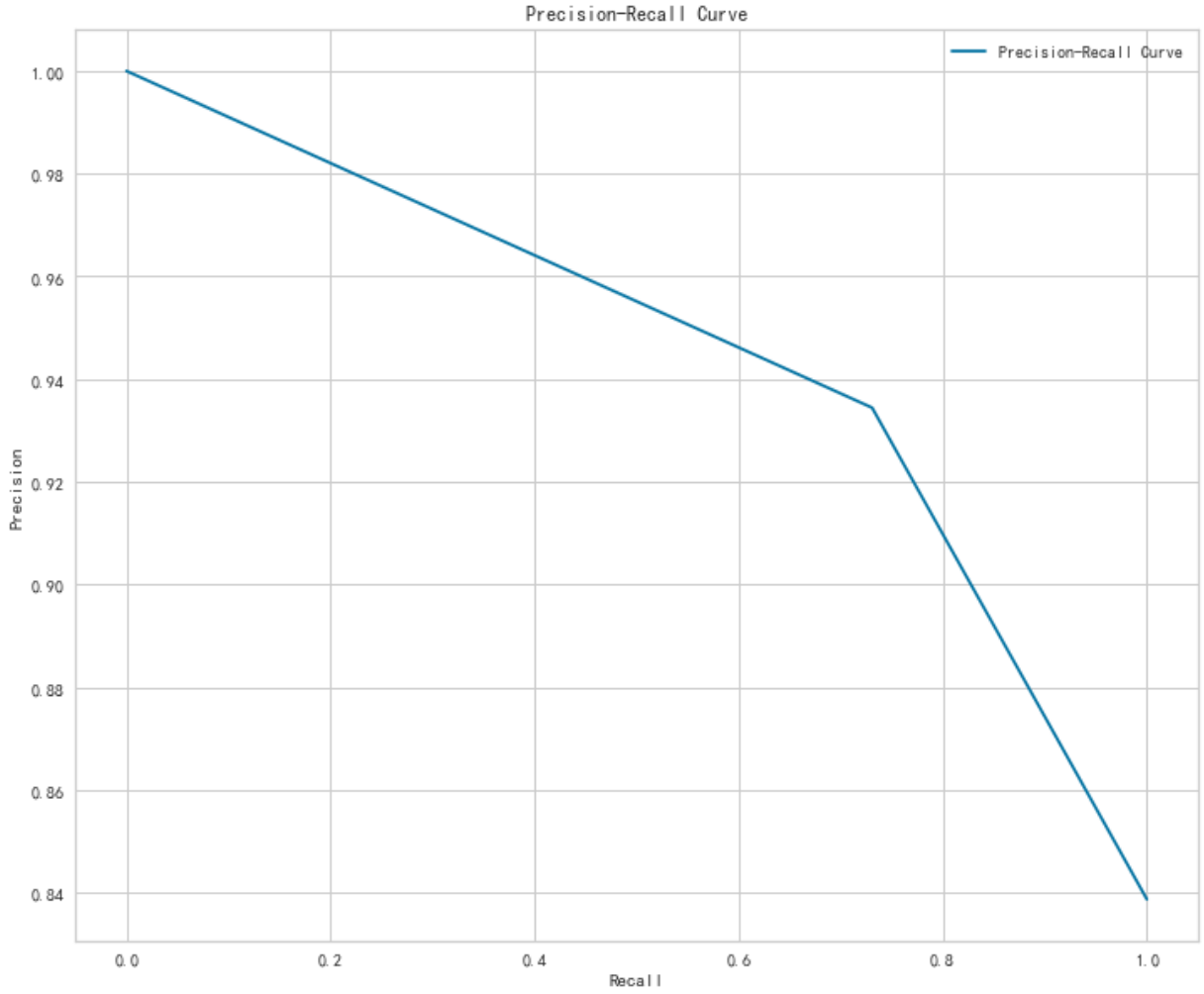
In [418…

```python
y_pred_proba = logistics.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.legend(loc=4)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC/AUC')
plt.show()
```



In [420…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label = "Precision-Recall Curve")
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



Random Forest

In [421…
```
rf = RandomForestClassifier()
n_estimators = [100, 300, 500, 1000]
max_features = ['auto', 'sqrt']
max_depth = [10, 30, 50]
min_samples_split = [2, 5, 8]
min_samples_leaf = [1, 2 , 3]
```

In [422…
```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter=100, cv=5, verbose=3, random_state=42, n_jobs=
rf_random.fit(X_resampled, y_resampled)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

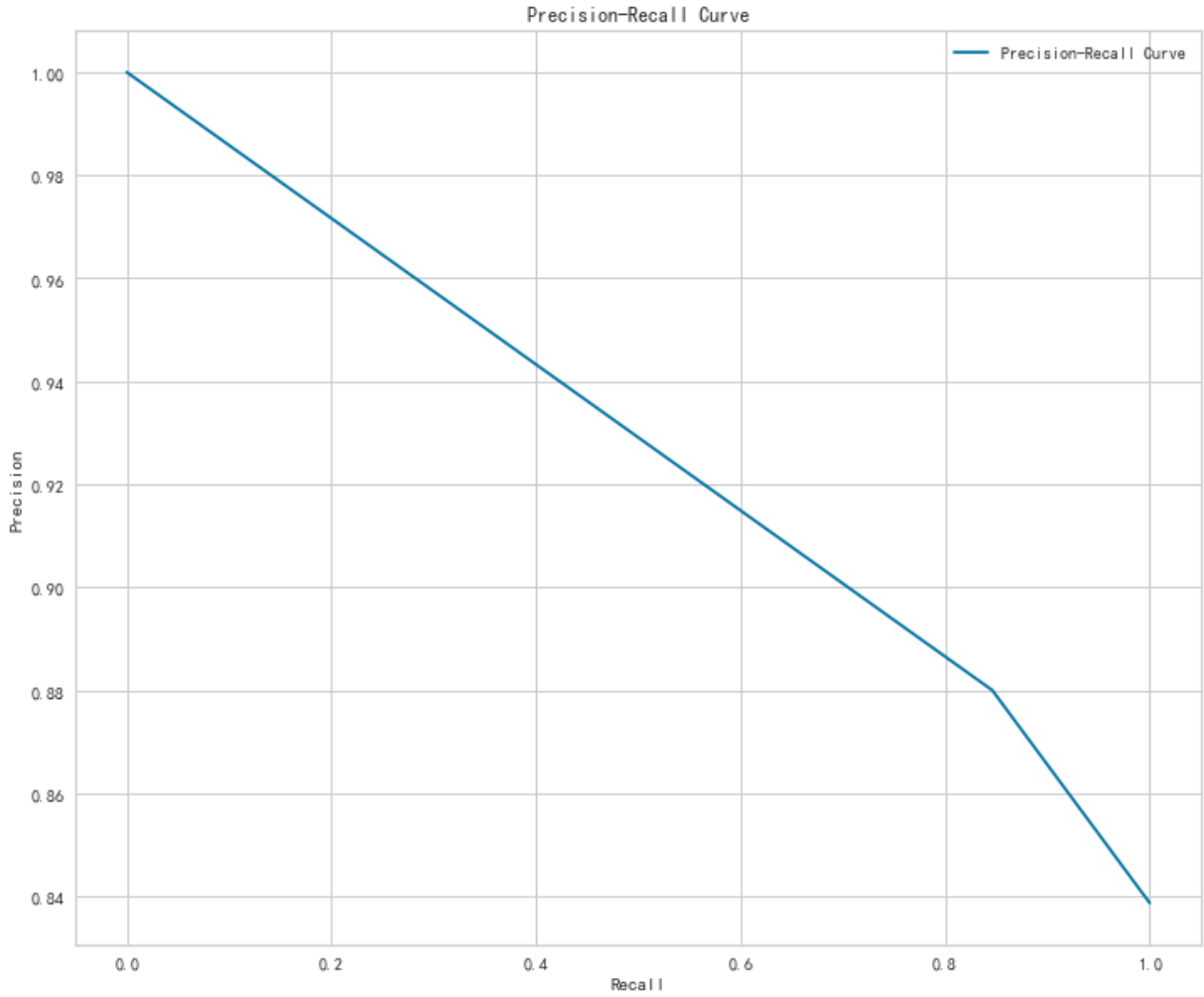
Out[422…
```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [423…
```
print('Best hyperparameters:', rf_random.best_params_)
y_pred = rf_random.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Best hyperparameters: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_lea
f': 1, 'max_features': 'auto', 'max_depth': 10}
Accuracy: 0.7741935483870968
```

In [424…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label='Precision-Recall Curve')
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



In [425…

```
rf_hp = pd.DataFrame(rf_random.best_estimator_.feature_importances_, index = X_train
```

In [426…

```
rf_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[426…

	feature_importances
b_coef_3	0.182649
b_coef_4	0.126609
f_coef_4	0.091740
Depth	0.062353
Anterior_posterior_Length	0.059638

	feature_importances
weight	0.054556
b_coef_2	0.046573
BMI	0.044585
Back_Crotch	0.044245
f_coef_1	0.043114
b_coef_1	0.042175
Crotch_curve_length_at_back_waist	0.037581
Height	0.037529
f_coef_3	0.036816
Front_Crotch	0.032857
Max_Hip	0.029766
f_coef_2	0.027214

XGboost

In [427…

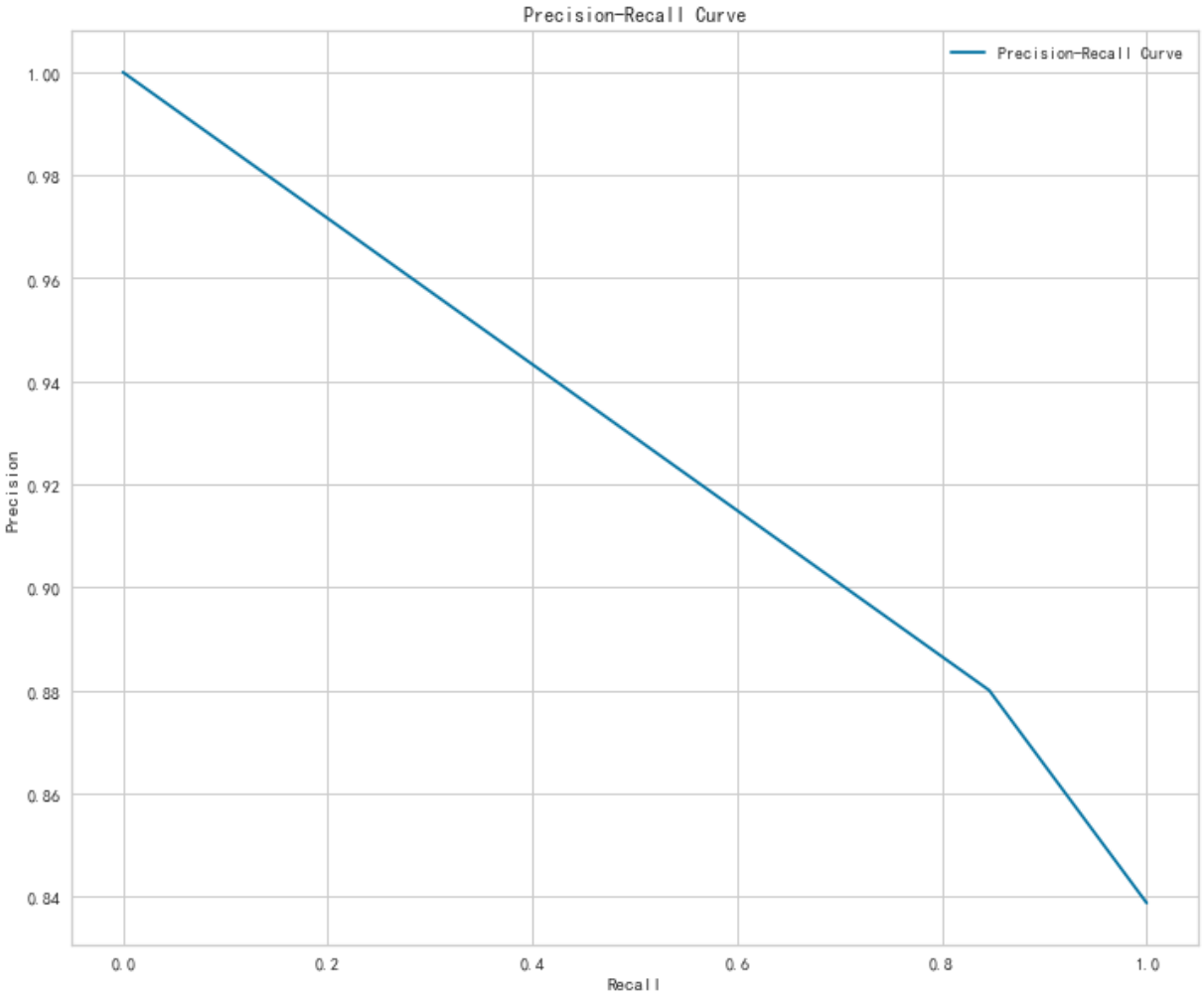
```
# Define the XGBoost model
xgb_model = xgb.XGBClassifier()

# Define the hyperparameters to tune
param_grid = {
    'max_depth': [3, 4, 5, 10],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 300, 500, 1000]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, verbose
grid_search.fit(X_resampled, y_resampled)

# Print the best hyperparameters and score
print('Best parameters:', grid_search.best_params_)

# Make predictions on the test set with the best model
best_model = grid_search.best_estimator_
preds = best_model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best parameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 300}
Accuracy: 0.7741935483870968
```

In [428…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label='Precision-Recall Curve')
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
```

```
# show plot
plt.show()
```



In [429…
```
xgb_hp = pd.DataFrame(grid_search.best_estimator_.feature_importances_, index = X_tr
```

In [430…
```
xgb_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[430…

	feature_importances
b_coef_3	0.205559
b_coef_4	0.090995
Depth	0.084274
f_coef_4	0.072702
b_coef_2	0.063989
Back_Crotch	0.059975
f_coef_3	0.052599
weight	0.048849
f_coef_1	0.046140
b_coef_1	0.045140
Anterior_posterior_Length	0.041066
Front_Crotch	0.040013
BMI	0.034043

	feature_importances
f_coef_2	0.032132
Crotch_curve_length_at_back_waist	0.029833
Height	0.029188
Max_Hip	0.023501

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Do the same thing but use Asian group as the sample set instead of African American

In [431…
```
# Load data and see the race value counts
df = pd.read_csv("CombinedWaistDataWithCoordinates with weight.csv")
df_Front = pd.read_csv("CurveCoefficients.csv")
df.drop('Unnamed: 0', axis=1, inplace=True)
df_Front.drop('Unnamed: 0', axis=1, inplace=True)
print(df["Race"].value_counts());
```

```
White          397
AfricanAmer     66
Asian           56
Other           15
Hispanic         6
Name: Race, dtype: int64
```

In [432…
```
# Merge curve coefficient into the original data through subject ID
df = pd.merge(df, df_Front, on = "Subject",  left_index=True, right_index=True, how=
df.isna().sum();
df = df.dropna();
df.reset_index(drop = True, inplace = True)
```

In [433…
```
cols_x = [col for col in df.columns if col.endswith('_x')]
df = df.loc[:,~df.columns.isin(cols_x)]
df.columns
```

Out[433…
```
Index(['Subject', 'weight', 'Curve_Coordinates', 'Race_y', 'BMI_y', 'Height_y',
       'Max_Hip_y', 'Anterior_posterior_Length_y', 'Depth_y',
       'Crotch_curve_length_at_back_waist_y', 'Front_Crotch_y',
       'Back_Crotch_y', 'Curve_Coord_Front', 'Curve_Coord_Back',
       'Coef_Front_2_3', 'intercept_Front_2_3', 'Coef_Back_2_3',
       'intercept_Back_2_3', 'Coef_Front_3_3', 'intercept_Front_3_3',
       'Coef_Back_3_3', 'intercept_Back_3_3', 'Coef_Front_2_4',
       'intercept_Front_2_4', 'Coef_Back_2_4', 'intercept_Back_2_4'],
      dtype='object')
```

In [434…
```
# Create front and back curve coefficient columns.
df["f_coef_1"] = 1
df["f_coef_2"] = 1
df["f_coef_3"] = 1
df["f_coef_4"] = 1
df["b_coef_1"] = 1
df["b_coef_2"] = 1
```

```
df["b_coef_3"] = 1
df["b_coef_4"] = 1
```

In [435…
```
# Add front curve coeffcient into the original data set
df["Coef_Front_2_3"];
for x in range(len(df)):
    df["f_coef_1"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[0]
    df["f_coef_2"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[1]
    df["f_coef_3"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[2]
    df["f_coef_4"][x] = df["Coef_Front_2_3"][x].strip("[]").split(",")[3]

df["f_coef_1"] = df["f_coef_1"].astype("float64")
df["f_coef_2"] = df["f_coef_2"].astype("float64")
df["f_coef_3"] = df["f_coef_3"].astype("float64")
df["f_coef_4"] = df["f_coef_4"].astype("float64")

df["f_coef_1"] = df["f_coef_1"].apply(lambda x: format(x, '.3f'))
df["f_coef_2"] = df["f_coef_2"].apply(lambda x: format(x, '.3f'))
df["f_coef_3"] = df["f_coef_3"].apply(lambda x: format(x, '.3f'))
df["f_coef_4"] = df["f_coef_4"].apply(lambda x: format(x, '.3f'))
```

In [436…
```
# Add back curve coeffcient into the original data set
df["Coef_Back_2_3"];
for x in range(len(df)):
    df["b_coef_1"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[0]
    df["b_coef_2"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[1]
    df["b_coef_3"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[2]
    df["b_coef_4"][x] = df["Coef_Back_2_3"][x].strip("[]").split(",")[3]

df["b_coef_1"] = df["b_coef_1"].astype("float64")
df["b_coef_2"] = df["b_coef_2"].astype("float64")
df["b_coef_3"] = df["b_coef_3"].astype("float64")
df["b_coef_4"] = df["b_coef_4"].astype("float64")

df["b_coef_1"] = df["b_coef_1"].apply(lambda x: format(x, '.3f'))
df["b_coef_2"] = df["b_coef_2"].apply(lambda x: format(x, '.3f'))
df["b_coef_3"] = df["b_coef_3"].apply(lambda x: format(x, '.3f'))
df["b_coef_4"] = df["b_coef_4"].apply(lambda x: format(x, '.3f'))
```

In [437…
```
# Take out the columns we need
df = df.iloc[:,[0,1,3,4,5,6,7,8,9,10,11,26,27,28,29,30,31,32,33]]
df;
```

In [438…
```
# Convert datatype to float
df["Height_y"] = df["Height_y"].astype("float64")
df["Max_Hip_y"] = df["Max_Hip_y"].astype("float64")
df["f_coef_1"] = df["f_coef_1"].astype("float64")
df["f_coef_2"] = df["f_coef_2"].astype("float64")
df["f_coef_3"] = df["f_coef_3"].astype("float64")
df["f_coef_4"] = df["f_coef_4"].astype("float64")
df["b_coef_1"] = df["b_coef_1"].astype("float64")
df["b_coef_2"] = df["b_coef_2"].astype("float64")
df["b_coef_3"] = df["b_coef_3"].astype("float64")
df["b_coef_4"] = df["b_coef_4"].astype("float64")
```

In [439…
```
df.rename(columns = {"BMI_y":"BMI","Height_y":"Height","Max_Hip_y":"Max_Hip","Anteri
                     "Depth_y":"Depth","Crotch_curve_length_at_back_waist_y":"Crotch
                     "Back_Crotch_y":"Back_Crotch","Race_y":"Race"}, inplace = True)
```

In [440…
```
df = df[(df["Race"]=="Asian") | (df["Race"]=="White")]
df.reset_index(inplace = True, drop = True)
df;
```

In [441…

```
df2 = df.iloc[:,[1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]]
df2
```

Out[441…

	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_length_a
0	77.52	22.152364	1647.0	975.0	258.6590	274.1830	
1	80.95	24.217396	1645.0	952.0	286.2070	299.7270	
2	70.63	20.530267	1541.0	967.0	246.1240	252.7480	
3	69.82	21.490804	1669.0	959.0	246.3330	252.9760	
4	77.37	20.205145	1695.0	991.0	245.9600	268.5630	
…	…	…	…	…	…	…	
445	77.37	19.309389	1710.0	920.0	238.5356	281.6155	
446	79.68	22.761793	1685.0	1020.0	278.1208	216.5177	
447	75.14	28.309274	1601.0	1062.0	291.3675	228.6181	
448	80.39	21.376065	1769.0	1004.0	249.2301	211.0781	
449	85.18	23.999562	1692.0	1046.0	268.1176	243.5517	

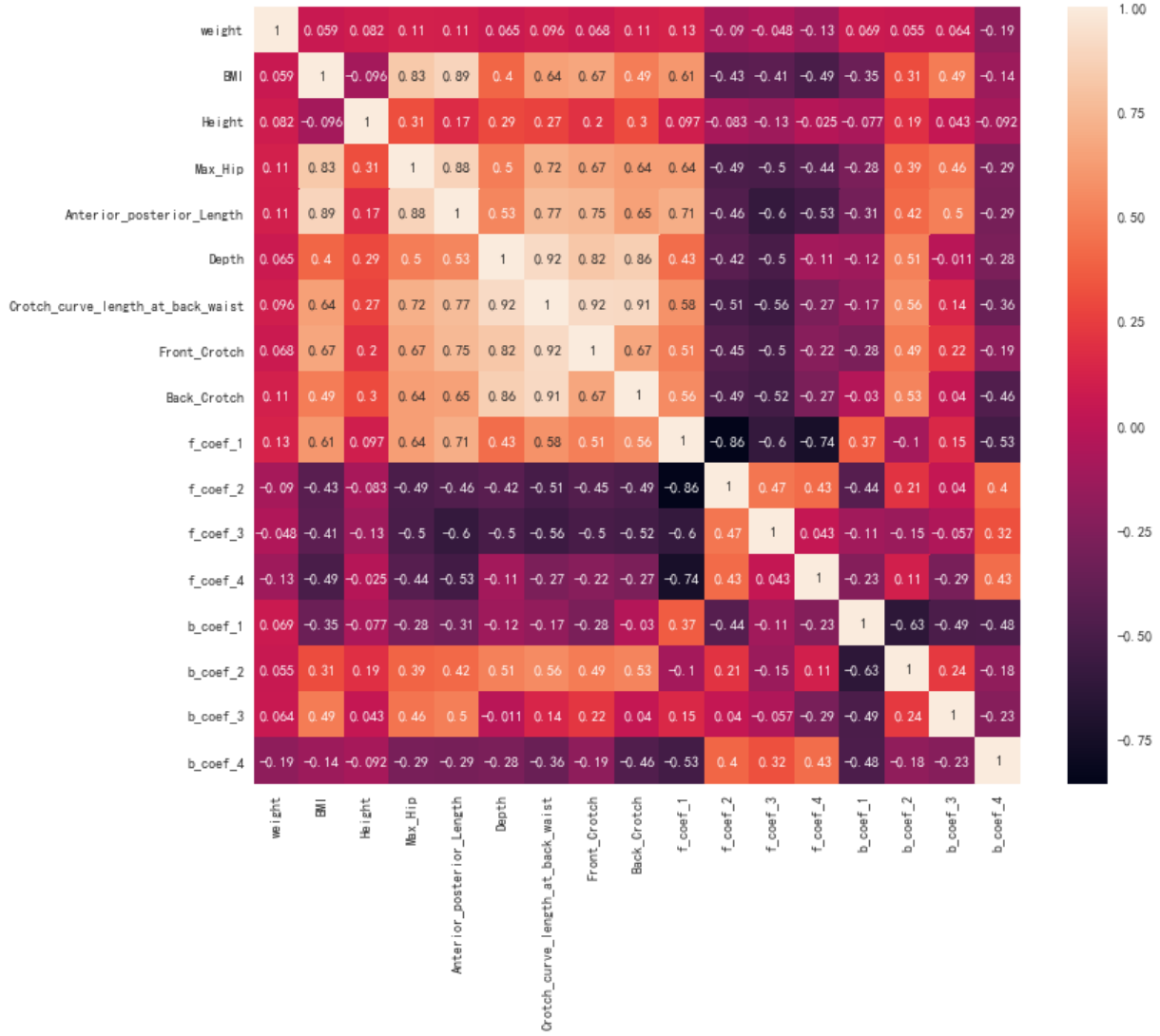
450 rows × 17 columns

In [442…

```
corr = df2.corr()
sns.heatmap(corr, annot=True, xticklabels=corr.columns, yticklabels=corr.columns)
```

Out[442…

```
<AxesSubplot:>
```



PCA AND K-Means

In [443...

```
df2
```

Out[443...

	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_length_a
0	77.52	22.152364	1647.0	975.0	258.6590	274.1830	
1	80.95	24.217396	1645.0	952.0	286.2070	299.7270	
2	70.63	20.530267	1541.0	967.0	246.1240	252.7480	
3	69.82	21.490804	1669.0	959.0	246.3330	252.9760	
4	77.37	20.205145	1695.0	991.0	245.9600	268.5630	
...	...	...	...	...	...	...	
445	77.37	19.309389	1710.0	920.0	238.5356	281.6155	
446	79.68	22.761793	1685.0	1020.0	278.1208	216.5177	
447	75.14	28.309274	1601.0	1062.0	291.3675	228.6181	
448	80.39	21.376065	1769.0	1004.0	249.2301	211.0781	
449	85.18	23.999562	1692.0	1046.0	268.1176	243.5517	

450 rows × 17 columns

In [444…
```
standard = preprocessing.scale(df2.values)
print(standard)
```

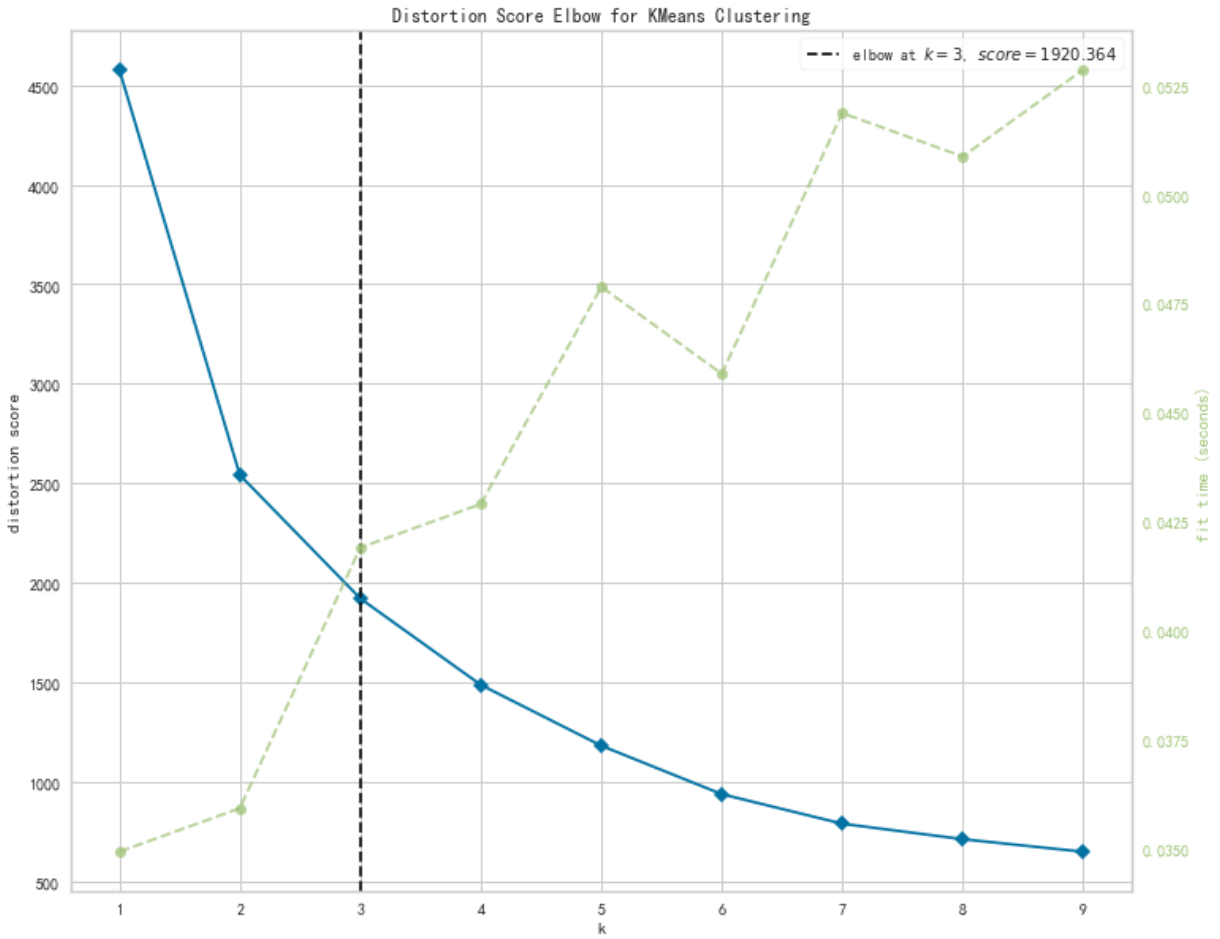
```
[[-0.2176293  -0.24441482 -0.05772807 ...  1.27195832  0.41947934
  -0.01164865]
 [ 0.28370769  0.30349152 -0.08485876 ...  0.56708316 -0.47757705
   0.83172037]
 [-1.22468816 -0.67479907 -1.49565452 ... -0.39799074  0.32978901
  -0.54134338]
 ...
 [-0.56549579  1.38917247 -0.68173389 ... -1.18474414  0.25410322
   0.45505565]
 [ 0.20185676 -0.45038701  1.59724387 ... -1.20883991 -0.40166168
  -0.57073537]
 [ 0.9019746   0.24569453  0.5527124  ...  0.68696996  0.39675065
   1.04880576]]
```

In [445…
```
PCA_x = PCA(n_components = 2).fit_transform(standard)
```

In [446…
```
PCA_r = pd.DataFrame(PCA_x)
PCA_r["Race"] = df["Race"]
```

In [447…
```
from yellowbrick.cluster import KElbowVisualizer
```

In [448…
```
km = KMeans(random_state=42)
visualizer = KElbowVisualizer(km, k=(1,10))

visualizer.fit(PCA_x)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```

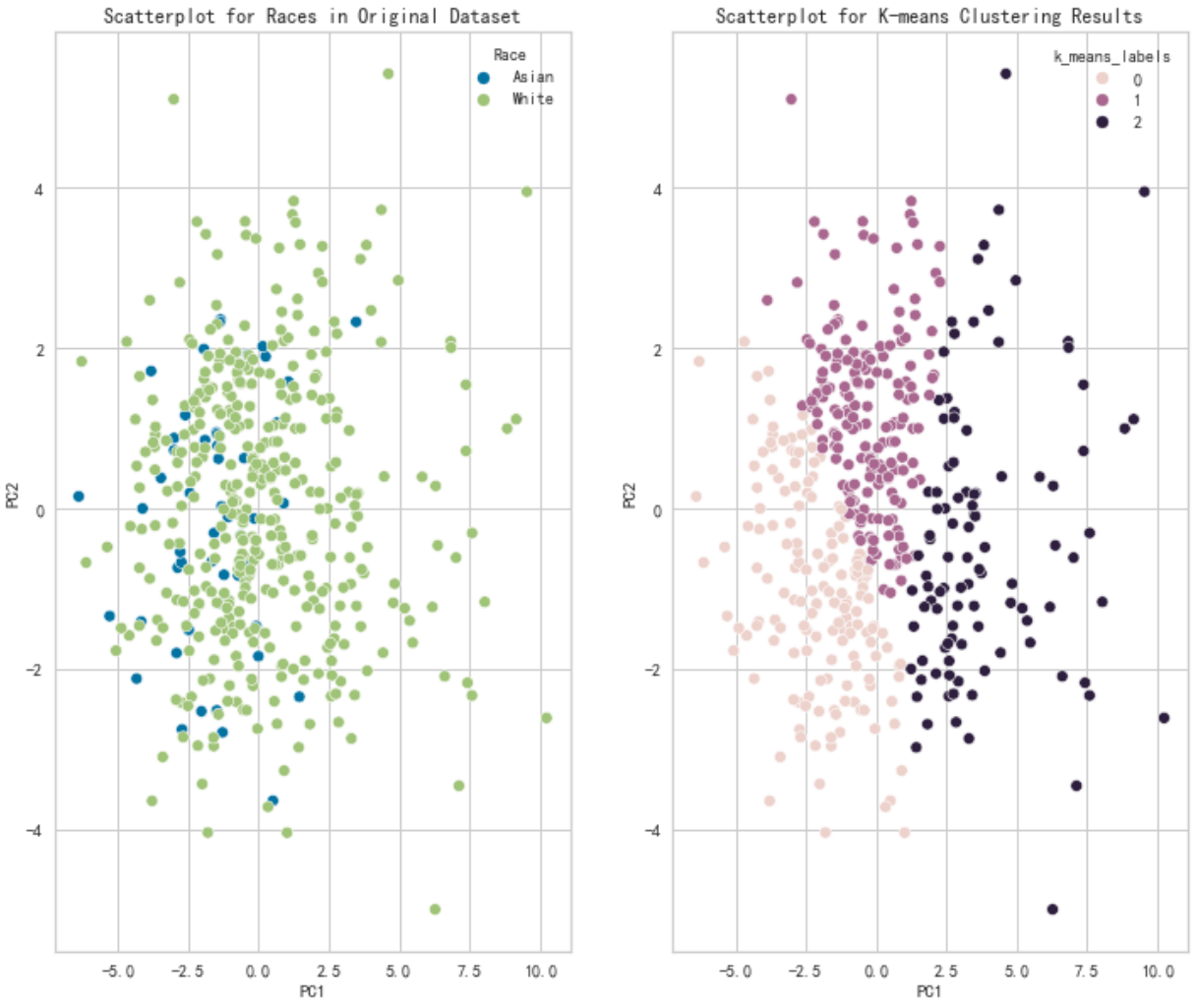


Out[448…
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel
='k', ylabel='distortion score'>
```

In [449…
```
clf = KMeans(n_clusters=3, init = "k-means++", max_iter = 100, random_state=0).fit(P
clf.cluster_centers_;
```

In [450…
```
df3 = pd.DataFrame(PCA_x)
df3["k_means_labels"] = clf.labels_
df3['Race'] = PCA_r["Race"]
df_1 = df3[df3['k_means_labels'] == 0]
df_2 = df3[df3['k_means_labels'] == 1]
df3;
```

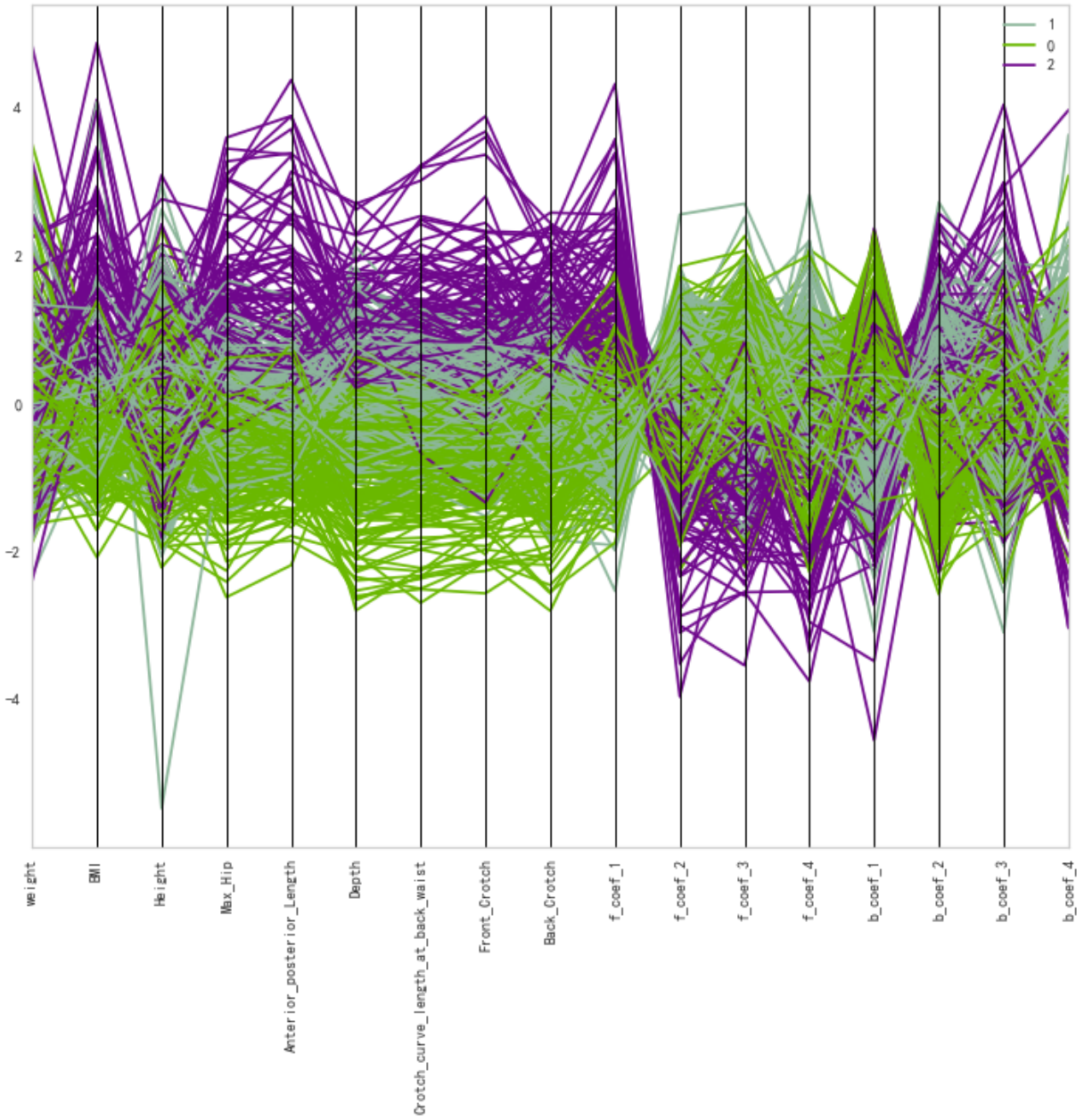
In [451…
```
# Visualize the K-means results
fig = plt.figure()
ax1 = plt.subplot(121)
sns.scatterplot(df3[0], df3[1], hue = df3["Race"], ax = ax1)
plt.title("Scatterplot for Races in Original Dataset")
plt.xlabel("PC1")
plt.ylabel("PC2")

ax2 = plt.subplot(122)
sns.scatterplot(df3[0], df3[1], hue = df3["k_means_labels"], ax = ax2)
plt.title("Scatterplot for K-means Clustering Results")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```



In [452…
```
X_clustered = pd.DataFrame(standard, index = df2.index, columns = df2.columns)
X_clustered["cluster"] = clf.labels_
```

In [453…

```
pd.plotting.parallel_coordinates(X_clustered,"cluster")
plt.xticks(rotation=90)
plt.show()
```



Heat Maps

In [454…

```
df
```

Out[454…

	Subject	weight	Race	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Croto
0	69	77.52	Asian	22.152364	1647.0	975.0	258.6590	274.1830	
1	70	80.95	Asian	24.217396	1645.0	952.0	286.2070	299.7270	
2	119	70.63	Asian	20.530267	1541.0	967.0	246.1240	252.7480	
3	160	69.82	Asian	21.490804	1669.0	959.0	246.3330	252.9760	
4	183	77.37	Asian	20.205145	1695.0	991.0	245.9600	268.5630	
…	…	…	…	…	…	…	…	…	
445	2979	77.37	White	19.309389	1710.0	920.0	238.5356	281.6155	

	Subject	weight	Race	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crot
446	2991	79.68	White	22.761793	1685.0	1020.0	278.1208	216.5177	
447	3006	75.14	White	28.309274	1601.0	1062.0	291.3675	228.6181	
448	3007	80.39	White	21.376065	1769.0	1004.0	249.2301	211.0781	
449	3018	85.18	White	23.999562	1692.0	1046.0	268.1176	243.5517	

450 rows × 19 columns

In [455…

```
standard_2 = preprocessing.scale(df.iloc[:,[1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
standard_2 = pd.DataFrame(standard_2)
standard_2 = standard_2.rename(columns={0:"weight",1:"BMI",2:"Height",3:"Max_Hip",4:
                                        5:"Depth",6:"Crotch_curve_length_at_back_waist",
                                        9:"f_coef_1", 10:"f_coef_2", 11:"f_coef_3", 12:"
                                        15:"b_coef_3", 16:"b_coef_4"})

standard_2;
standard_2['Race'] = df['Race'];
standard_2
```

Out[455…

	weight	BMI	Height	Max_Hip	Anterior_posterior_Length	Depth	Crotch_curve_le
0	-0.217629	-0.244415	-0.057728	-0.454968	-0.320991	0.463068	
1	0.283708	0.303492	-0.084859	-0.738997	0.522003	1.506109	
2	-1.224688	-0.674799	-1.495655	-0.553761	-0.704573	-0.412189	
3	-1.343080	-0.419944	0.240709	-0.652553	-0.698177	-0.402879	
4	-0.239554	-0.761062	0.593408	-0.257382	-0.709591	0.233586	
…	…	…	…	…	…	…	
445	-0.239554	-0.998730	0.796889	-1.134168	-0.936784	0.766560	
446	0.098081	-0.082718	0.457755	0.100741	0.274558	-1.891584	
447	-0.565496	1.389172	-0.681734	0.619403	0.679918	-1.397487	
448	0.201857	-0.450387	1.597244	-0.096844	-0.609523	-2.113699	
449	0.901975	0.245695	0.552712	0.421817	-0.031549	-0.787702	

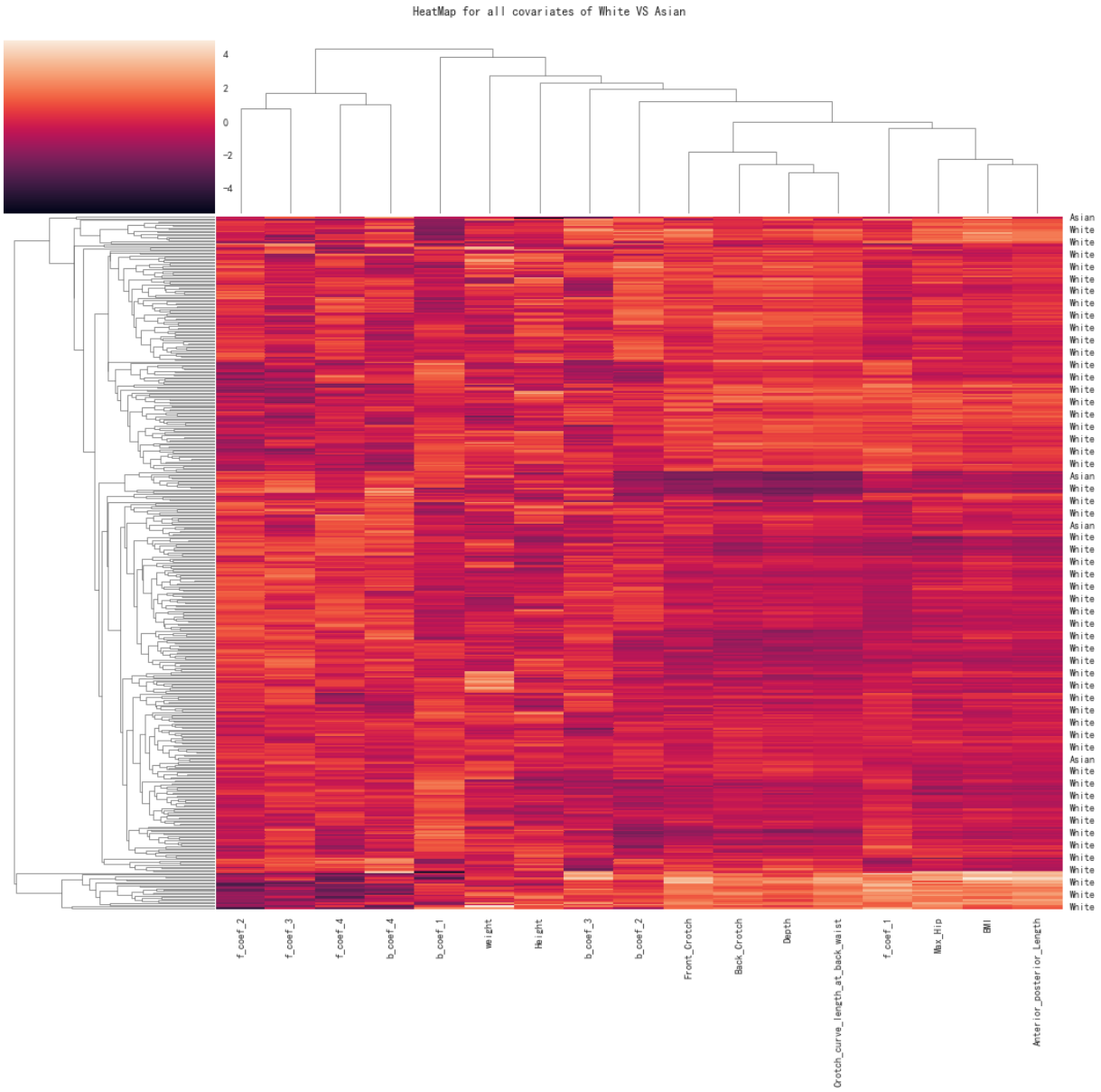
450 rows × 18 columns

In [456…

```
row_names = df['Race'].values.tolist()
standard_2.index = row_names
standard_2;
```

In [457…

```
# Do the heatmap
g = sns.clustermap(standard_2.iloc[:,:-1], figsize=(15, 15))
g.fig.suptitle("HeatMap for all covariates of White VS Asian")
g.fig.subplots_adjust(top=0.95)
```



Logistics Regression

In [458…
```
df['Race_label'] = df['Race'].map({'Asian':0,'White':1})
df["Race"].value_counts()
```

Out[458…
```
White    396
Asian     54
Name: Race, dtype: int64
```

In [459…
```
X = df.drop(columns = ["Race","Race_label","Subject"], axis=1)
y = df["Race_label"]
```

In [460…
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_s
```

In [461…
```
y_train.value_counts()
```

Out[461…
```
1    320
0    313
Name: Race_label, dtype: int64
```

In [462…
```
logistics = LogisticRegression(random_state = 42).fit(X_train,y_train)
y_pred = logistics.predict(X_test)
```

```
y_pred
```

Out[462…

```
array([0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 1], dtype=int64)
```

In [463…

```
Logistic_result = pd.DataFrame(y_test)
Logistic_result = pd.DataFrame(np.concatenate((Logistic_result,pd.DataFrame(y_pred))
Logistic_result.rename(columns={0:"Original",1:"Predicted"},inplace = True)
```

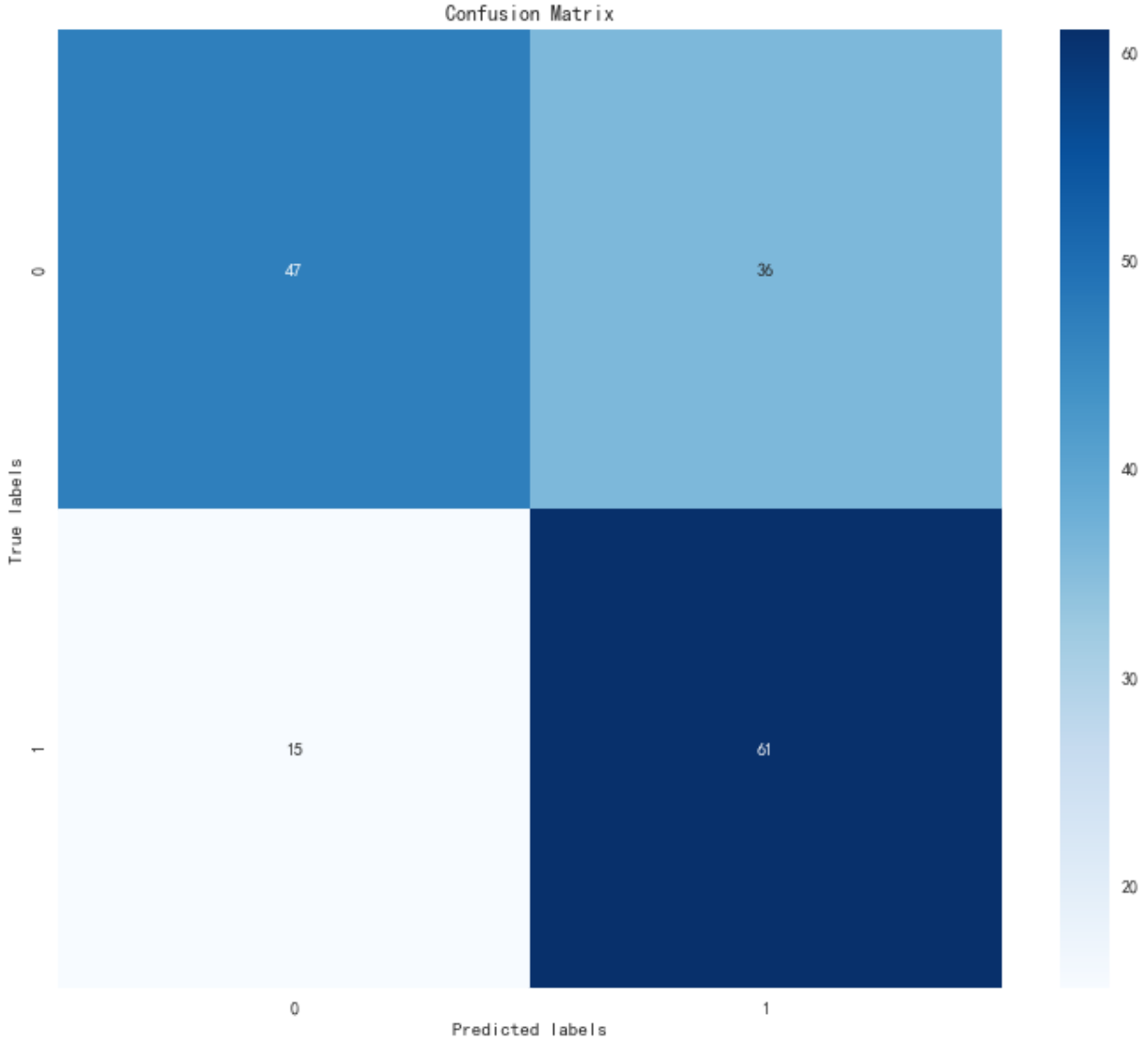
In [464…

```
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.57      0.65        83
           1       0.63      0.80      0.71        76

    accuracy                           0.68       159
   macro avg       0.69      0.68      0.68       159
weighted avg       0.70      0.68      0.68       159
```

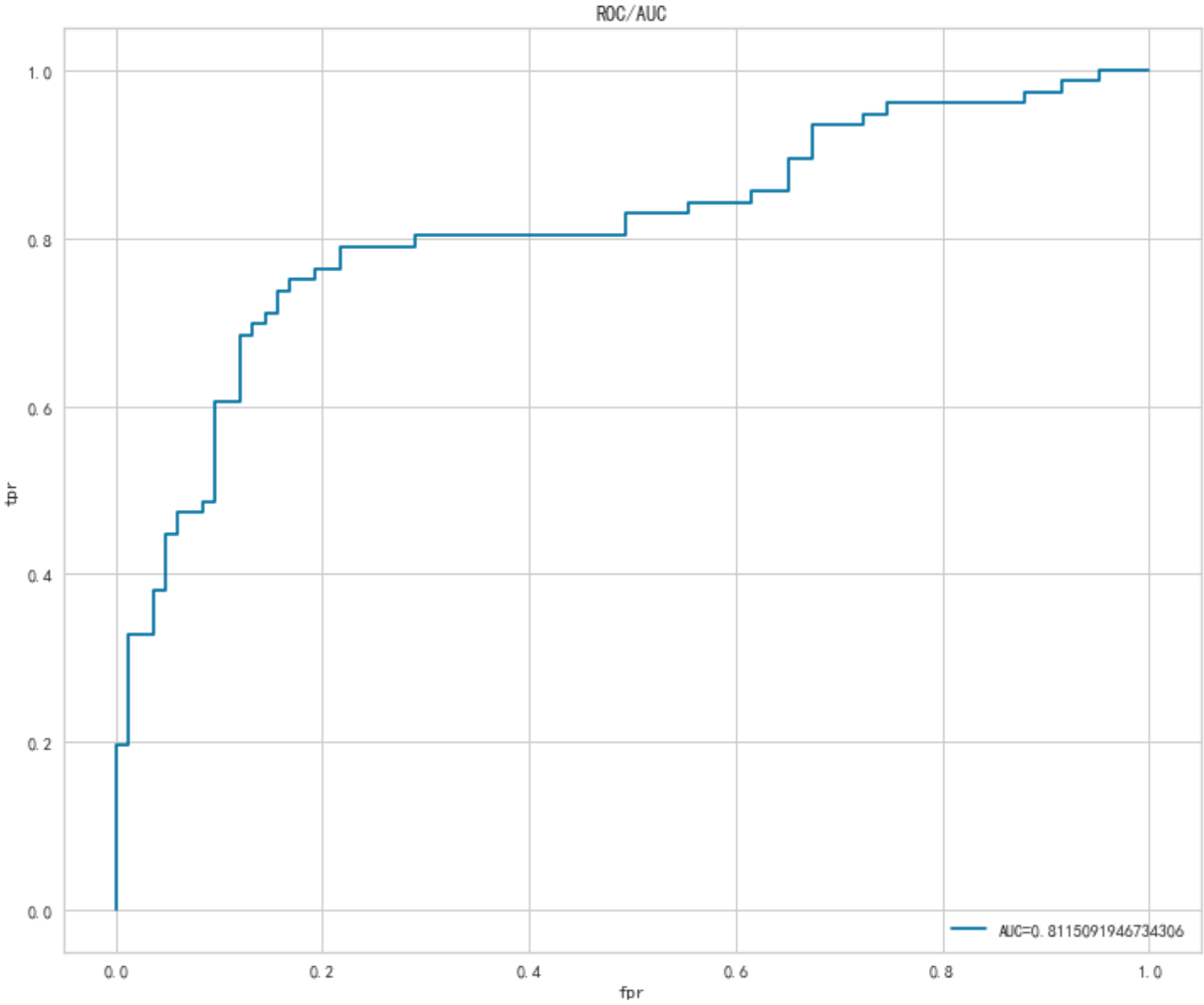
In [465…

```
# Create the confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# Add axis labels and title
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



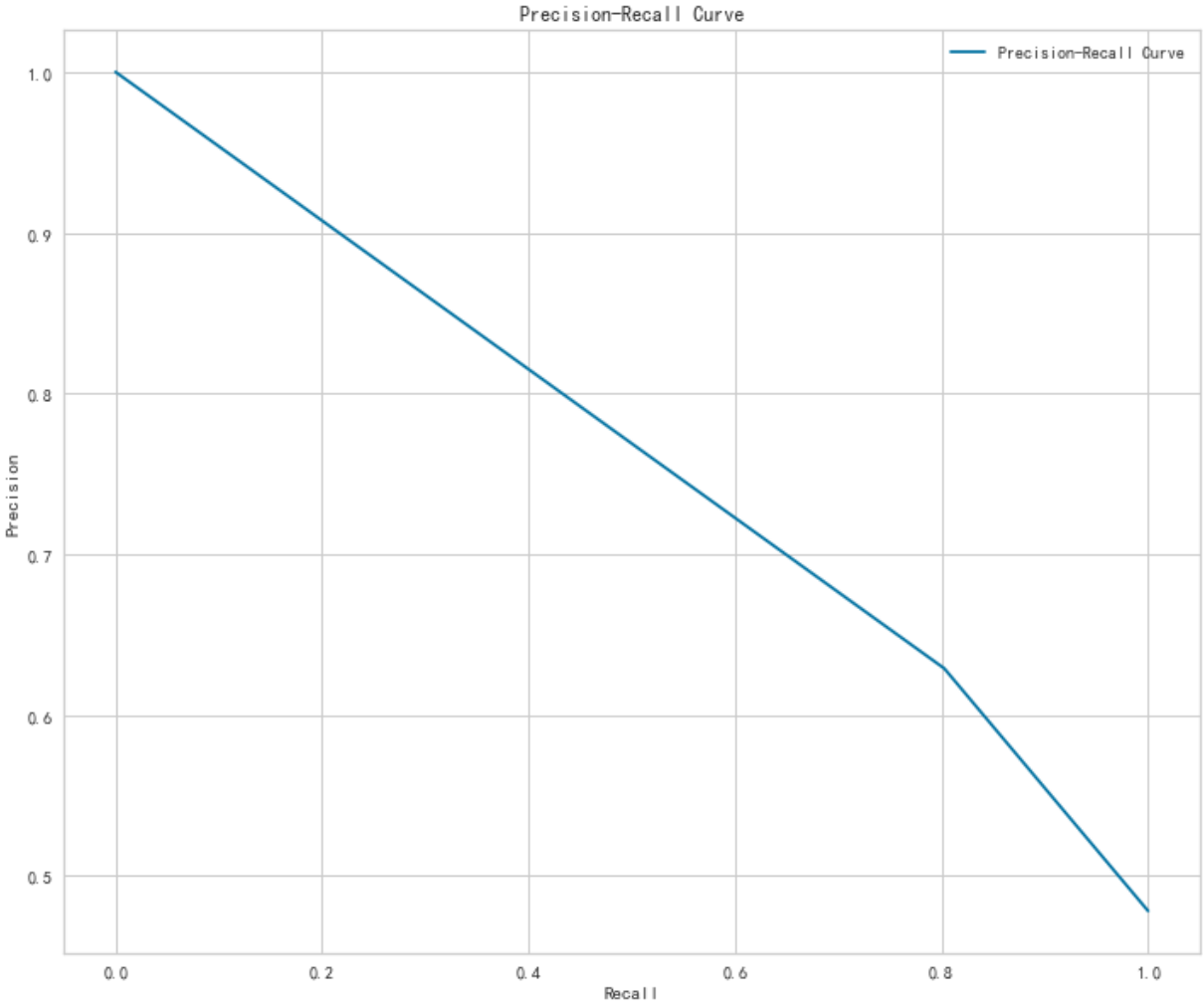
In [466…

```
y_pred_proba = logistics.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.legend(loc=4)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC/AUC')
plt.show()
```



In [467…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label = "Precision-Recall Curve")
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



Random Forest

In [468…
```
rf = RandomForestClassifier()
n_estimators = [100, 300, 500, 1000]
max_features = ['auto', 'sqrt']
max_depth = [10, 30, 50]
min_samples_split = [2, 5, 8]
min_samples_leaf = [1, 2 , 3]
```

In [469…
```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter=100, cv=5, verbose=3, random_state=42, n_jobs=
rf_random.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[469…

▸ **RandomizedSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

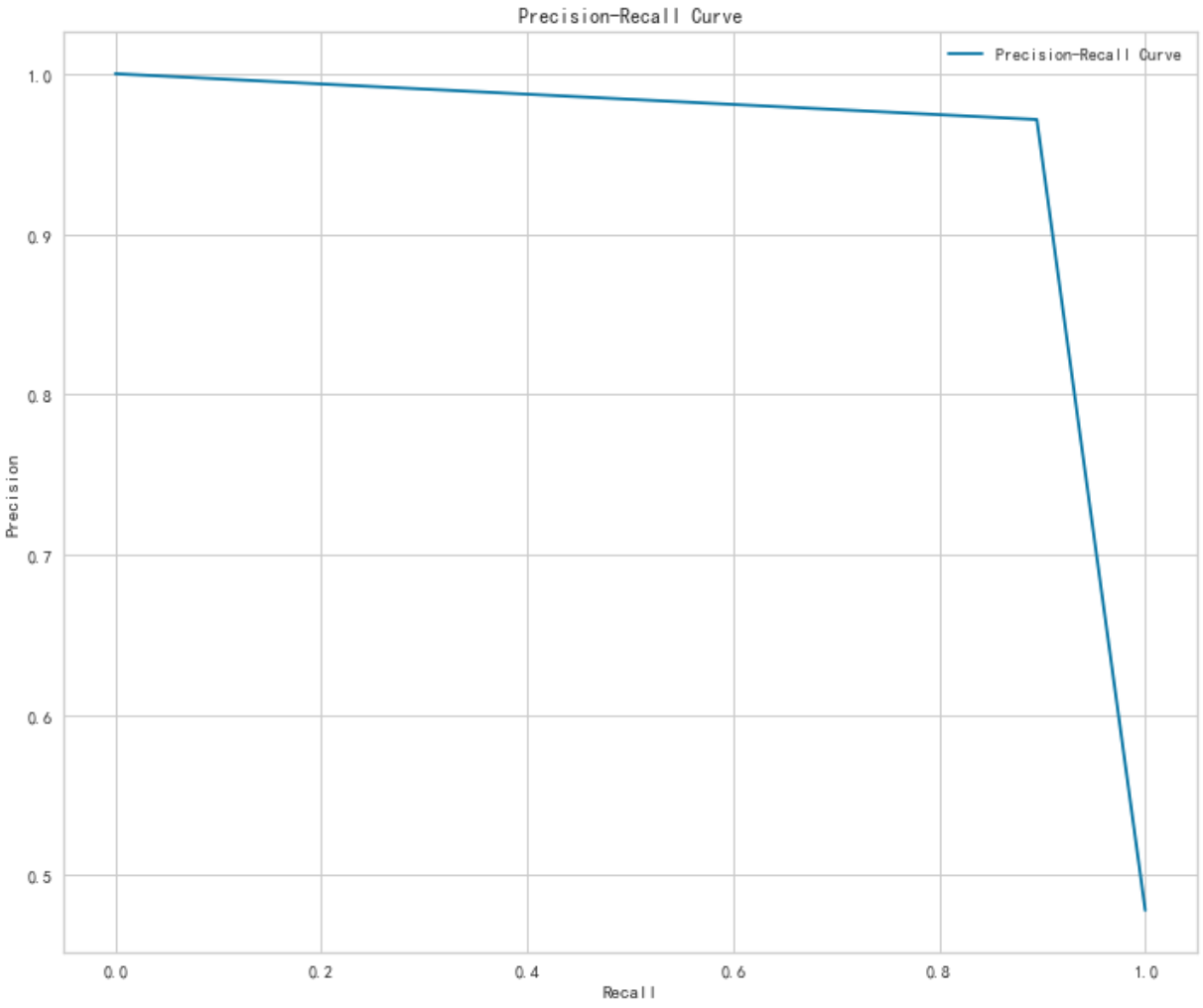
In [470…
```
print('Best hyperparameters:', rf_random.best_params_)
y_pred = rf_random.predict(X_test)
```

```
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Best hyperparameters: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_lea
f': 1, 'max_features': 'sqrt', 'max_depth': 50}
Accuracy: 0.9371069182389937
```

In [471…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label='Precision-Recall Curve')
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



In [472…

```
rf_hp = pd.DataFrame(rf_random.best_estimator_.feature_importances_, index = X_train
```

In [473…

```
rf_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[473…

	feature_importances
Height	0.236871
weight	0.113559
Back_Crotch	0.095086
Max_Hip	0.088442

	feature_importances
b_coef_4	0.050609
Crotch_curve_length_at_back_waist	0.045086
Anterior_posterior_Length	0.044442
Depth	0.044111
BMI	0.039959
f_coef_1	0.034731
Front_Crotch	0.034530
f_coef_2	0.033403
b_coef_2	0.032548
b_coef_3	0.032388
f_coef_4	0.028899
f_coef_3	0.024618
b_coef_1	0.020717

XGboost

In [474…

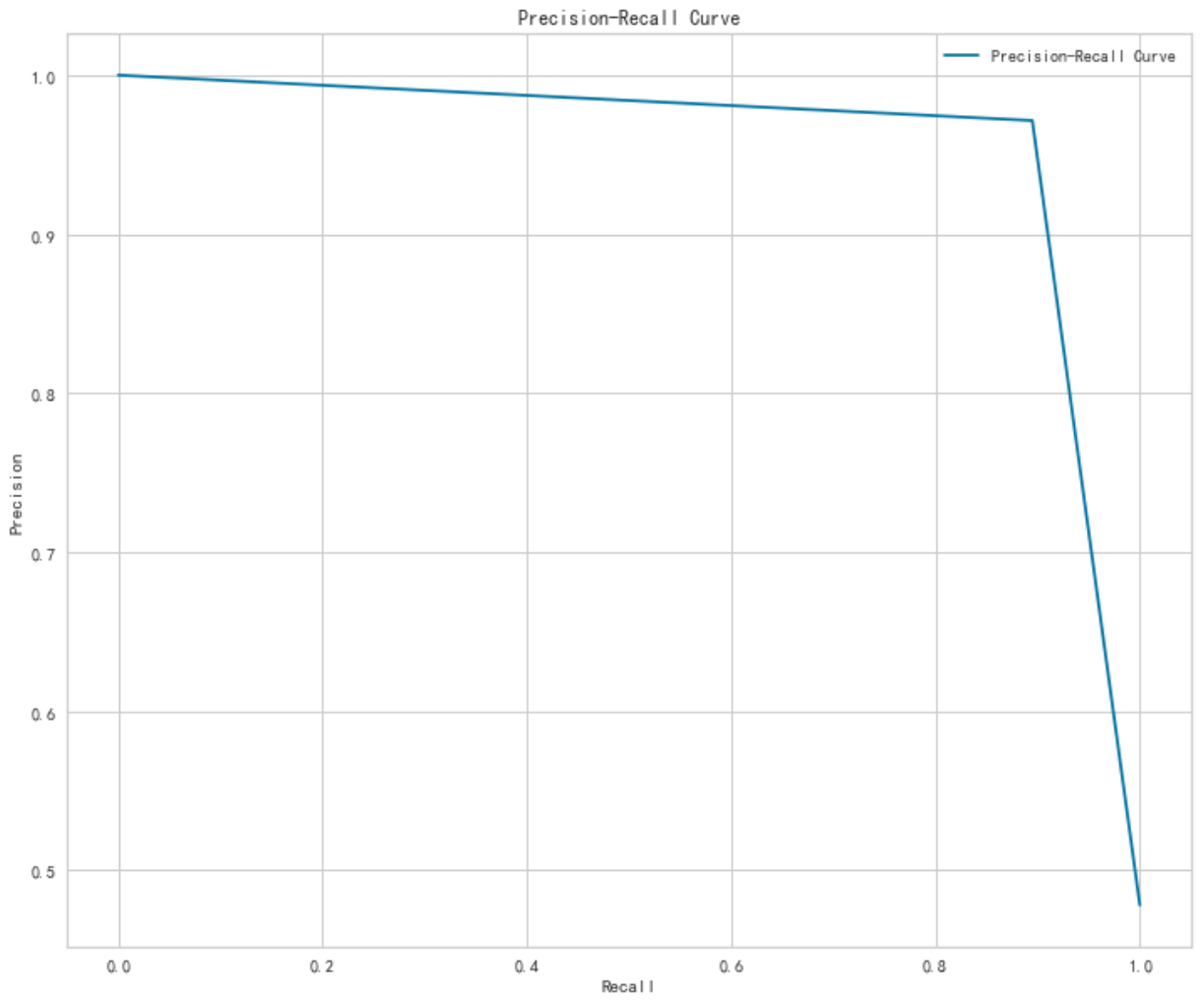
```
# Define the XGBoost model
xgb_model = xgb.XGBClassifier()

# Define the hyperparameters to tune
param_grid = {
    'max_depth': [3, 4, 5, 10],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 300, 500, 1000]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, verbose
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and score
print('Best parameters:', grid_search.best_params_)

# Make predictions on the test set with the best model
best_model = grid_search.best_estimator_
preds = best_model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best parameters: {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 1000}
Accuracy: 0.9371069182389937
```

In [475…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label ="Precision-Recall Curve")
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
```

```
# show plot
plt.show()
```



In [476…
```
xgb_hp = pd.DataFrame(grid_search.best_estimator_.feature_importances_, index = X_tr
```

In [477…
```
xgb_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[477…

	feature_importances
Height	0.204005
Crotch_curve_length_at_back_waist	0.124223
weight	0.107133
Max_Hip	0.074195
Back_Crotch	0.061916
f_coef_1	0.060594
b_coef_4	0.057069
BMI	0.050234
Depth	0.046393
Anterior_posterior_Length	0.038538
b_coef_3	0.032286
Front_Crotch	0.029537
b_coef_2	0.026666

	feature_importances
f_coef_3	0.024937
f_coef_4	0.023988
f_coef_2	0.021283
b_coef_1	0.017002

In [ ]:

In [ ]:

Do the same thing for imbalanced data

Logistic Regression

In [478…
```
df['Race_label'] = df['Race'].map({'Asian':0,'White':1})
df["Race"].value_counts()
```

Out[478…
```
White    396
Asian     54
Name: Race, dtype: int64
```

In [479…
```
X = df.drop(columns = ["Race","Race_label","Subject"], axis=1)
y = df["Race_label"]
```

In [480…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

In [481…
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

In [482…
```
y_resampled.value_counts()
```

Out[482…
```
1    318
0    318
Name: Race_label, dtype: int64
```

In [483…
```
logistics = LogisticRegression(random_state = 42).fit(X_resampled, y_resampled)
y_pred = logistics.predict(X_test)
y_pred
```

Out[483…
```
array([1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0], dtype=int64)
```

In [484…
```
Logistic_result = pd.DataFrame(y_test)
Logistic_result = pd.DataFrame(np.concatenate((Logistic_result,pd.DataFrame(y_pred))
Logistic_result.rename(columns={0:"Original",1:"Predicted"},inplace = True)
```

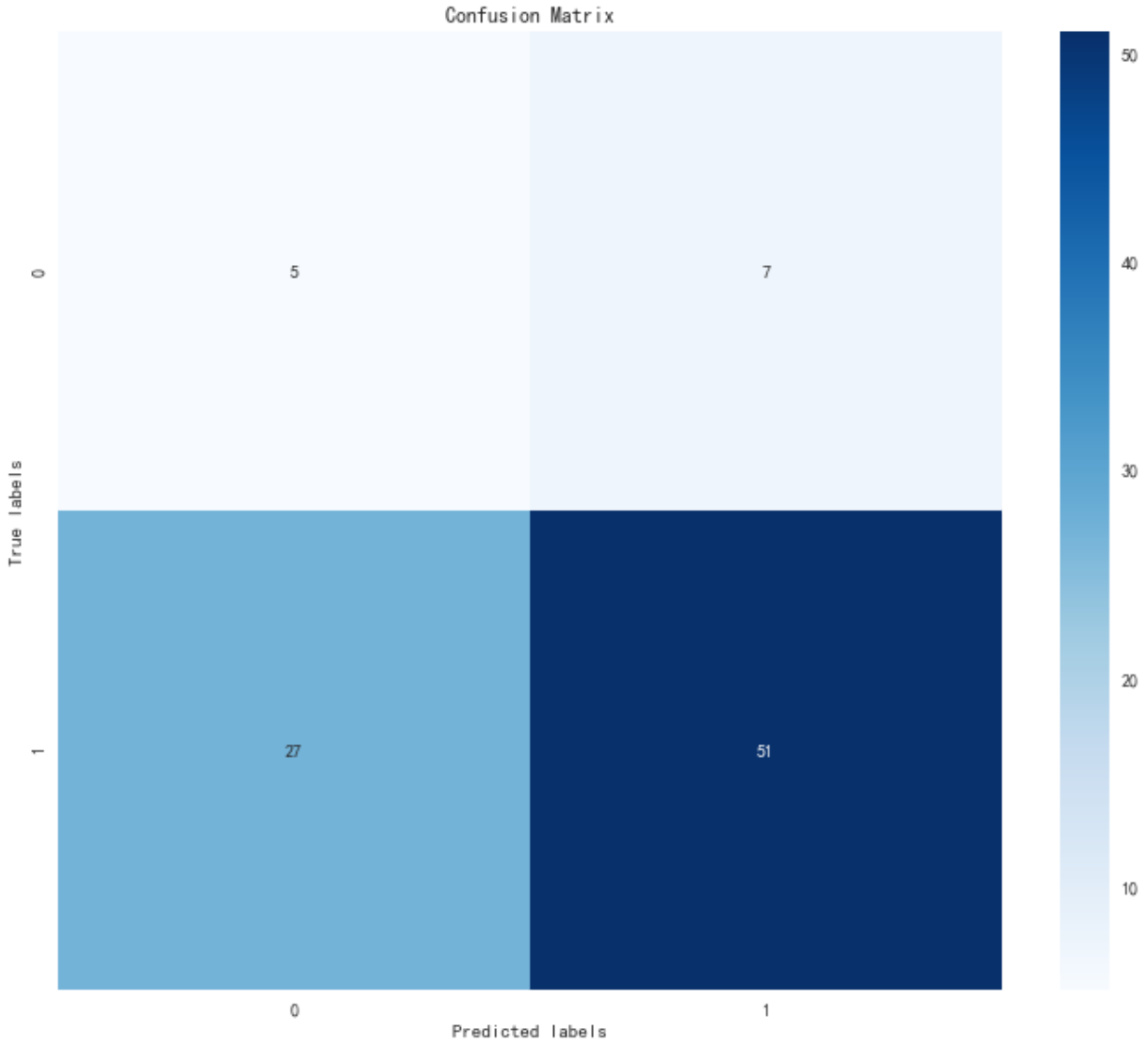
In [485…
```
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.16      0.42      0.23        12
           1       0.88      0.65      0.75        78
```

```
              accuracy                           0.62        90
             macro avg       0.52      0.54      0.49        90
          weighted avg       0.78      0.62      0.68        90
```

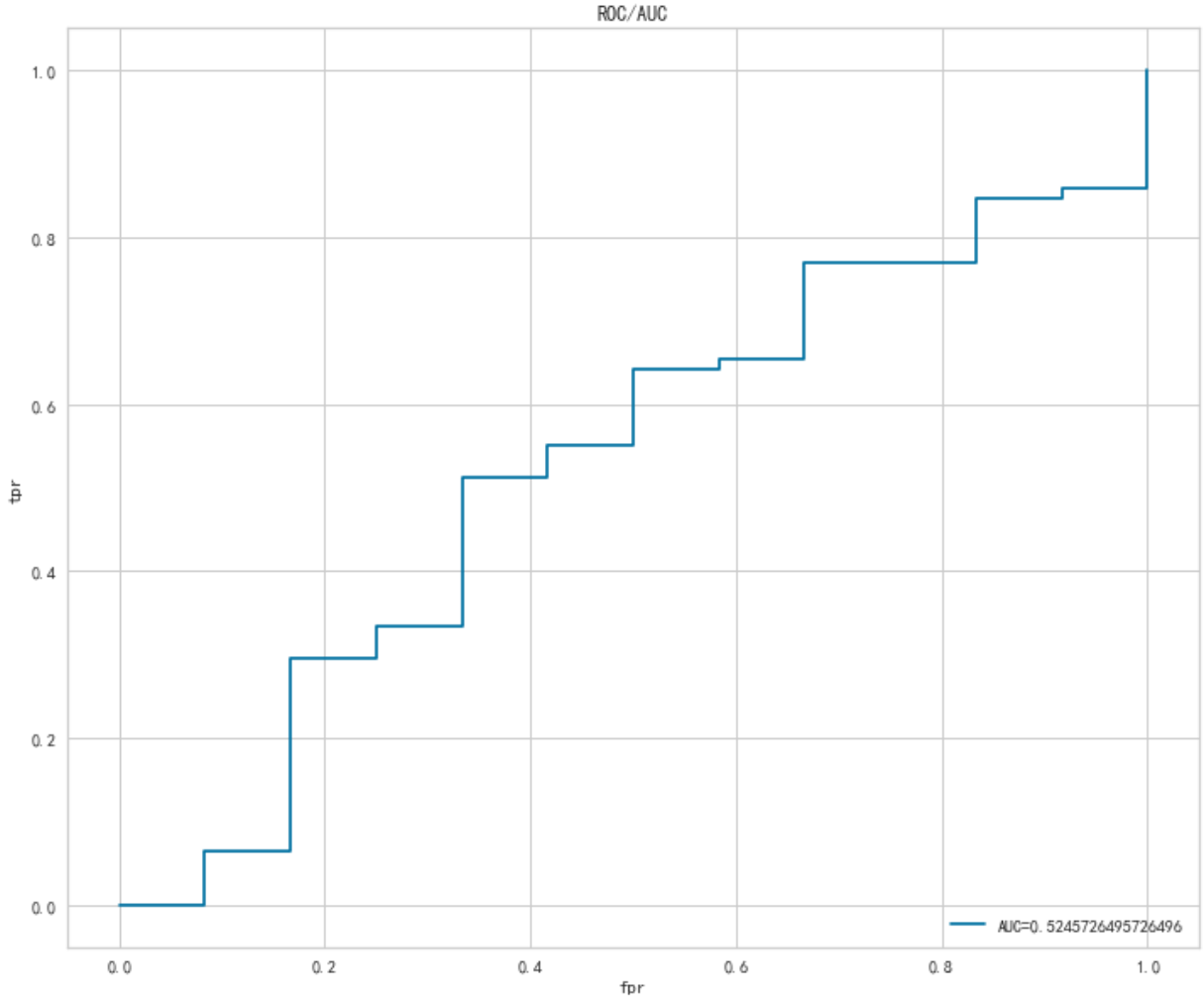
In [486…

```python
# Create the confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# Add axis labels and title
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



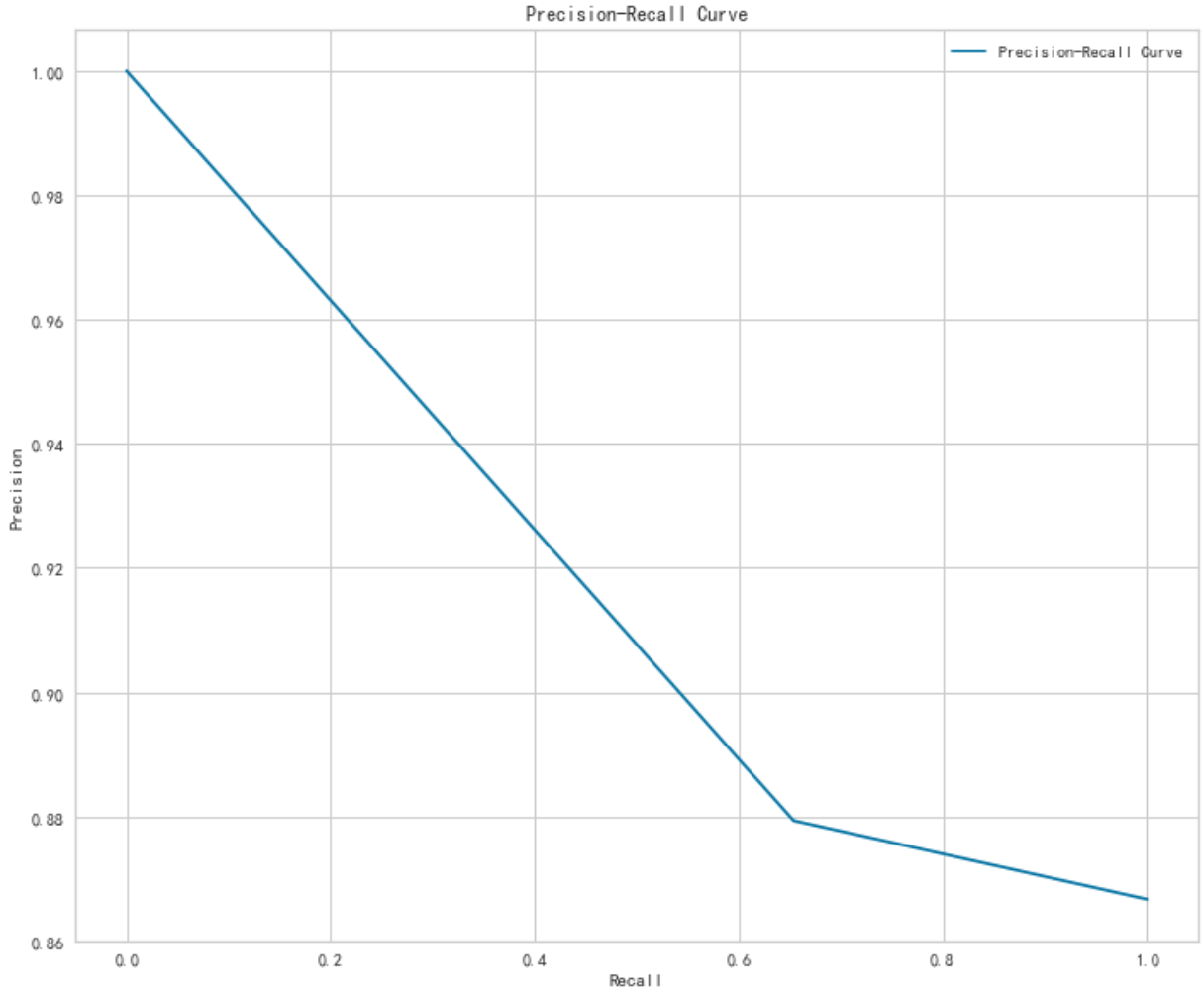
In [487…

```python
y_pred_proba = logistics.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.legend(loc=4)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC/AUC')
plt.show()
```



In [488…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label = "Precision-Recall Curve")
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



Random Forest

In [489…

```
rf = RandomForestClassifier()
n_estimators = [100, 300, 500, 1000]
max_features = ['auto', 'sqrt']
max_depth = [10, 30, 50]
min_samples_split = [2, 5, 8]
min_samples_leaf = [1, 2 , 3]
```

In [490…

```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter=100, cv=5, verbose=3, random_state=42, n_jobs=
rf_random.fit(X_resampled, y_resampled)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[490…

▸ **RandomizedSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

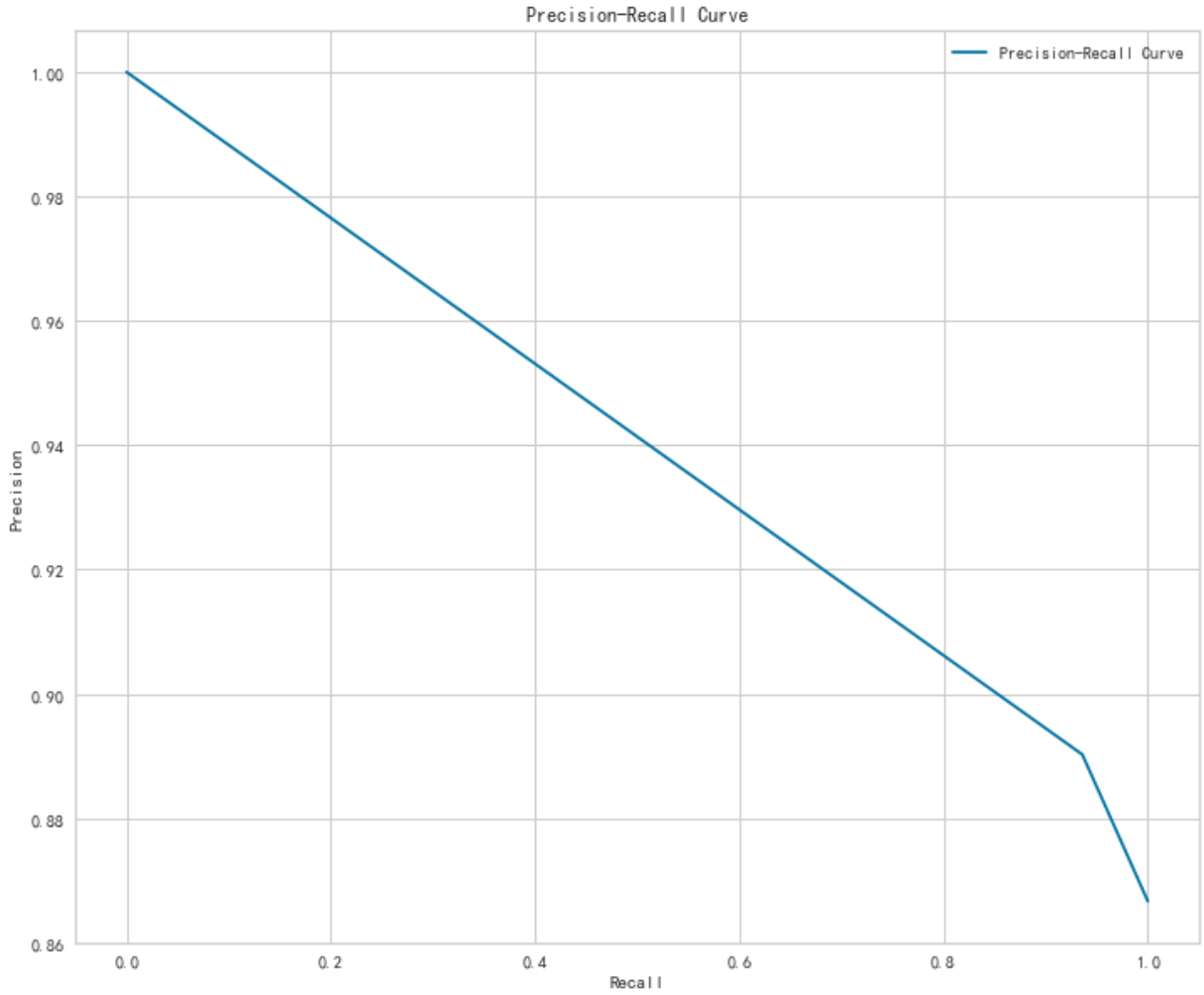
In [491…

```
print('Best hyperparameters:', rf_random.best_params_)
y_pred = rf_random.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Best hyperparameters: {'n_estimators': 1000, 'min_samples_split': 2, 'min_samples_le
af': 1, 'max_features': 'auto', 'max_depth': 50}
Accuracy: 0.8444444444444444
```

In [492…

```python
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label = "Precision-Recall Curve")
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()

# show plot
plt.show()
```



In [493…

```python
rf_hp = pd.DataFrame(rf_random.best_estimator_.feature_importances_, index = X_train
```

In [494…

```python
rf_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[494…

	feature_importances
Height	0.244045
weight	0.142684
Max_Hip	0.114930
Back_Crotch	0.066185
b_coef_4	0.061560

	feature_importances
f_coef_2	0.038607
Crotch_curve_length_at_back_waist	0.038080
BMI	0.036382
Anterior_posterior_Length	0.032881
Front_Crotch	0.032766
f_coef_4	0.031394
b_coef_3	0.030120
f_coef_1	0.030082
b_coef_2	0.030075
Depth	0.026151
f_coef_3	0.024979
b_coef_1	0.019078

XGboost

In [495…

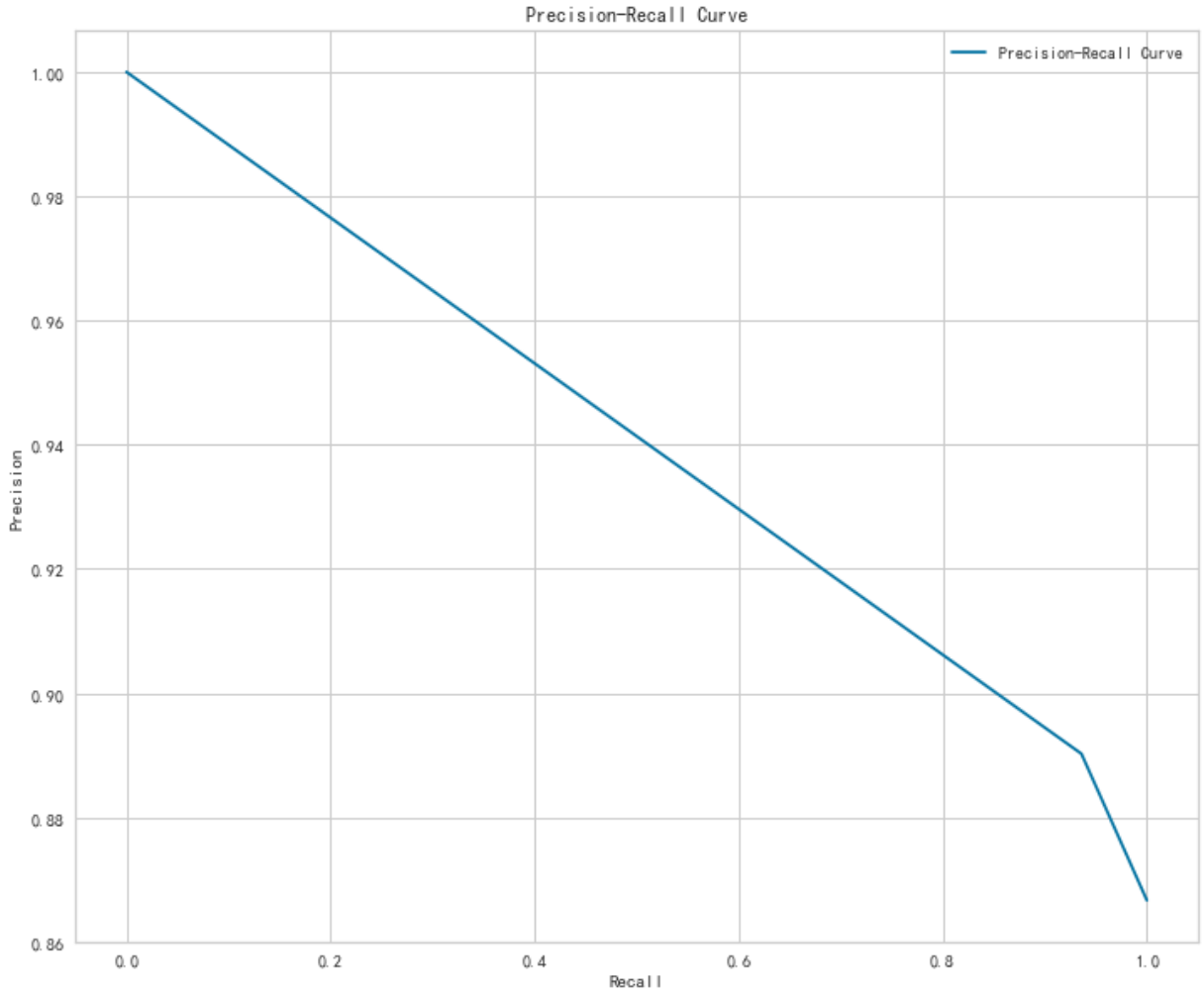
```
# Define the XGBoost model
xgb_model = xgb.XGBClassifier()

# Define the hyperparameters to tune
param_grid = {
    'max_depth': [3, 4, 5, 10],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 300, 500, 1000]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, verbose
grid_search.fit(X_resampled, y_resampled)

# Print the best hyperparameters and score
print('Best parameters:', grid_search.best_params_)

# Make predictions on the test set with the best model
best_model = grid_search.best_estimator_
preds = best_model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best parameters: {'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 500}
Accuracy: 0.8444444444444444
```

In [496…

```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

plt.plot(recall, precision, label='Precision-Recall Curve')
# set axis labels and legend
plt.title("Precision-Recall Curve")
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
```

```
# show plot
plt.show()
```



In [497…
```
xgb_hp = pd.DataFrame(grid_search.best_estimator_.feature_importances_, index = X_tr
```

In [498…
```
xgb_hp.sort_values(by = "feature_importances",ascending = False)
```

Out[498…

	feature_importances
Height	0.263086
weight	0.138149
f_coef_1	0.084276
Back_Crotch	0.064503
Max_Hip	0.062201
b_coef_4	0.052268
b_coef_3	0.044046
Depth	0.037384
f_coef_2	0.034485
Crotch_curve_length_at_back_waist	0.032932
b_coef_1	0.031730
f_coef_4	0.031588
Front_Crotch	0.028609

	feature_importances
BMI	0.026440
Anterior_posterior_Length	0.025322
b_coef_2	0.024331
f_coef_3	0.018649