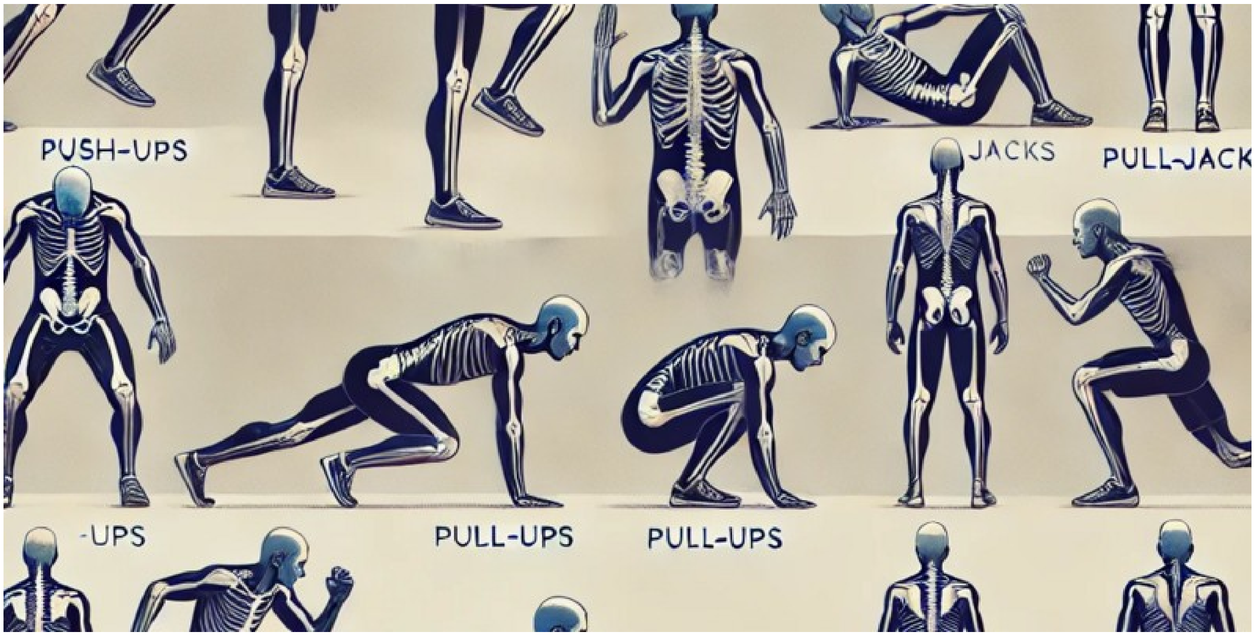
Real-Time Exercise Detection and Form Analysis Using Joint Angles and Deep Learning




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('exercise_detection.csv')

df.head()
```

```
   Side  Shoulder_Angle  Elbow_Angle   Hip_Angle  Knee_Angle
Ankle_Angle  \
0  left       10.639208   174.466813  174.785143  179.848140
179.419276
1  left       10.590342   174.428706  174.765041  179.775215
179.386147
2  left       10.546746   174.489431  174.785790  179.660017
179.333710
3  left       10.487682   174.614913  174.759542  179.614223
179.313926
4  left       10.412107   174.758503  174.737721  179.570564
179.298805
```

```
   Shoulder_Ground_Angle  Elbow_Ground_Angle  Hip_Ground_Angle  \
0                     90                  90                90   
1                     90                  90                90   
2                     90                  90                90   
3                     90                  90                90   
4                     90                  90                90   

   Knee_Ground_Angle  Ankle_Ground_Angle          Label  
0                 90                  90  Jumping Jacks  
1                 90                  90  Jumping Jacks  
2                 90                  90  Jumping Jacks  
3                 90                  90  Jumping Jacks  
4                 90                  90  Jumping Jacks  
```

```
df.tail()
```

```
       Side  Shoulder_Angle  Elbow_Angle   Hip_Angle  Knee_Angle
Ankle_Angle  \
31028  left       12.723974    81.226330  149.356832  154.358415
178.103121   
31029  left        9.080920    82.486551  148.100509  152.680540
178.625318   
31030  left        4.118076    85.164707  148.329461  152.458288
178.605852   
31031  left        0.558065    89.419330  146.742440  149.930599
179.604753   
31032  left        3.610121    89.992518  141.439189  144.633832
179.616705   

       Shoulder_Ground_Angle  Elbow_Ground_Angle  Hip_Ground_Angle  \
31028                     90                  90                90   
31029                     90                  90                90   
31030                     90                  90                90   
31031                     90                  90                90   
31032                     90                  90                90   

       Knee_Ground_Angle  Ankle_Ground_Angle           Label  
31028                 90                 -90  Russian twists  
31029                 90                 -90  Russian twists  
31030                 90                 -90  Russian twists  
31031                 90                 -90  Russian twists  
31032                 90                 -90  Russian twists  
```

```
df.shape
```

```
(31033, 12)
```

```
df.columns
```

```
Index(['Side', 'Shoulder_Angle', 'Elbow_Angle', 'Hip_Angle',
'Knee_Angle',
       'Ankle_Angle', 'Shoulder_Ground_Angle', 'Elbow_Ground_Angle',
       'Hip_Ground_Angle', 'Knee_Ground_Angle', 'Ankle_Ground_Angle',
'Label'],
      dtype='object')
```

```
df.duplicated().sum()
```

```
0
```

```
df.isnull().sum()
```

```
Side                     0
Shoulder_Angle           0
Elbow_Angle              0
Hip_Angle                0
Knee_Angle               0
Ankle_Angle              0
Shoulder_Ground_Angle    0
Elbow_Ground_Angle       0
Hip_Ground_Angle         0
Knee_Ground_Angle        0
Ankle_Ground_Angle       0
Label                    0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31033 entries, 0 to 31032
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Side                   31033 non-null  object 
 1   Shoulder_Angle         31033 non-null  float64
 2   Elbow_Angle            31033 non-null  float64
 3   Hip_Angle              31033 non-null  float64
 4   Knee_Angle             31033 non-null  float64
 5   Ankle_Angle            31033 non-null  float64
 6   Shoulder_Ground_Angle  31033 non-null  int64  
 7   Elbow_Ground_Angle     31033 non-null  int64  
 8   Hip_Ground_Angle       31033 non-null  int64  
 9   Knee_Ground_Angle      31033 non-null  int64  
 10  Ankle_Ground_Angle     31033 non-null  int64  
 11  Label                  31033 non-null  object 
dtypes: float64(5), int64(5), object(2)
memory usage: 2.8+ MB
```

```
df.describe()
```

```
       Shoulder_Angle   Elbow_Angle     Hip_Angle    Knee_Angle
Ankle_Angle  \
count    31033.000000  31033.000000  31033.000000  31033.000000
31033.000000
mean        66.522206    114.303010    137.466151    143.273623
135.211957
std         60.226756     57.906279     57.048278     48.041715
53.304068
min          0.002748      0.000974      0.006850      0.116036
0.031297
25%         17.852184     58.900491    111.556724    123.646144
106.740814
50%         40.585632    132.999090    168.374922    168.227063
162.926184
75%        121.209005    168.769517    175.656498    177.225089
175.735039
max        179.991577    179.998861    179.999848    179.999278
179.999942

       Shoulder_Ground_Angle  Elbow_Ground_Angle  Hip_Ground_Angle  \
count           31033.000000        31033.000000      31033.000000
mean               88.816743           88.926949         79.408694
std                14.546233           13.856550         42.359381
min               -90.000000          -90.000000        -90.000000
25%                90.000000           90.000000         90.000000
50%                90.000000           90.000000         90.000000
75%                90.000000           90.000000         90.000000
max                90.000000           90.000000         90.000000

       Knee_Ground_Angle  Ankle_Ground_Angle
count       31033.000000        31033.000000
mean           75.795121           68.985596
std            48.530150           57.802208
min           -90.000000          -90.000000
25%            90.000000           90.000000
50%            90.000000           90.000000
75%            90.000000           90.000000
max            90.000000           90.000000
```

```
df.nunique()
```

```
Side                         1
Shoulder_Angle           31032
Elbow_Angle              31033
Hip_Angle                31031
Knee_Angle               31031
Ankle_Angle              31031
Shoulder_Ground_Angle        2
Elbow_Ground_Angle           2
Hip_Ground_Angle             2
```

```
Knee_Ground_Angle            2
Ankle_Ground_Angle           2
Label                        5
dtype: int64
```

```
object_columns = df.select_dtypes(include=['object']).columns
print("Object type columns:")
print(object_columns)

numerical_columns = df.select_dtypes(include=['int64',
'float64']).columns
print("\nNumerical type columns:")
print(numerical_columns)
```

```
Object type columns:
Index(['Side', 'Label'], dtype='object')

Numerical type columns:
Index(['Shoulder_Angle', 'Elbow_Angle', 'Hip_Angle', 'Knee_Angle',
       'Ankle_Angle', 'Shoulder_Ground_Angle', 'Elbow_Ground_Angle',
       'Hip_Ground_Angle', 'Knee_Ground_Angle', 'Ankle_Ground_Angle'],
      dtype='object')
```

```
def classify_features(df):
    categorical_features = []
    non_categorical_features = []
    discrete_features = []
    continuous_features = []

    for column in df.columns:
        if df[column].dtype == 'object':
            if df[column].nunique() < 10:
                categorical_features.append(column)
            else:
                non_categorical_features.append(column)
        elif df[column].dtype in ['int64', 'float64']:
            if df[column].nunique() < 10:
                discrete_features.append(column)
            else:
                continuous_features.append(column)

    return categorical_features, non_categorical_features,
discrete_features, continuous_features

categorical, non_categorical, discrete, continuous =
classify_features(df)

print("Categorical Features:", categorical)
print("Non-Categorical Features:", non_categorical)
print("Discrete Features:", discrete)
print("Continuous Features:", continuous)
```

```
Categorical Features: ['Side', 'Label']
Non-Categorical Features: []
Discrete Features: ['Shoulder_Ground_Angle', 'Elbow_Ground_Angle',
'Hip_Ground_Angle', 'Knee_Ground_Angle', 'Ankle_Ground_Angle']
Continuous Features: ['Shoulder_Angle', 'Elbow_Angle', 'Hip_Angle',
'Knee_Angle', 'Ankle_Angle']
```

```
for i in discrete:
    print(i)
    print(df[i].unique())
    print()
```

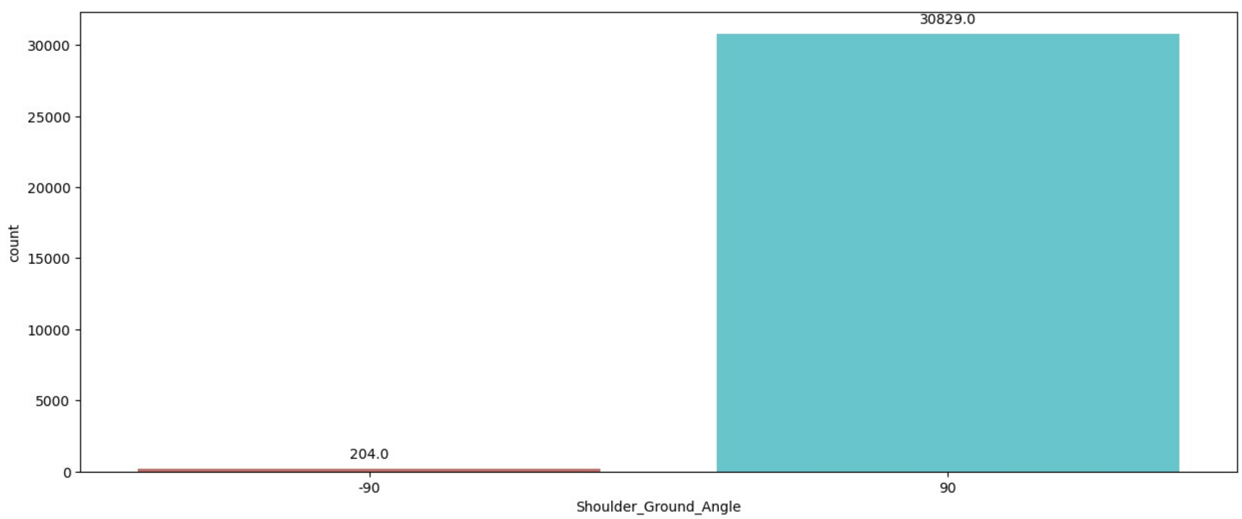
```
Shoulder_Ground_Angle
[ 90 -90]

Elbow_Ground_Angle
[ 90 -90]

Hip_Ground_Angle
[ 90 -90]

Knee_Ground_Angle
[ 90 -90]

Ankle_Ground_Angle
[ 90 -90]
```

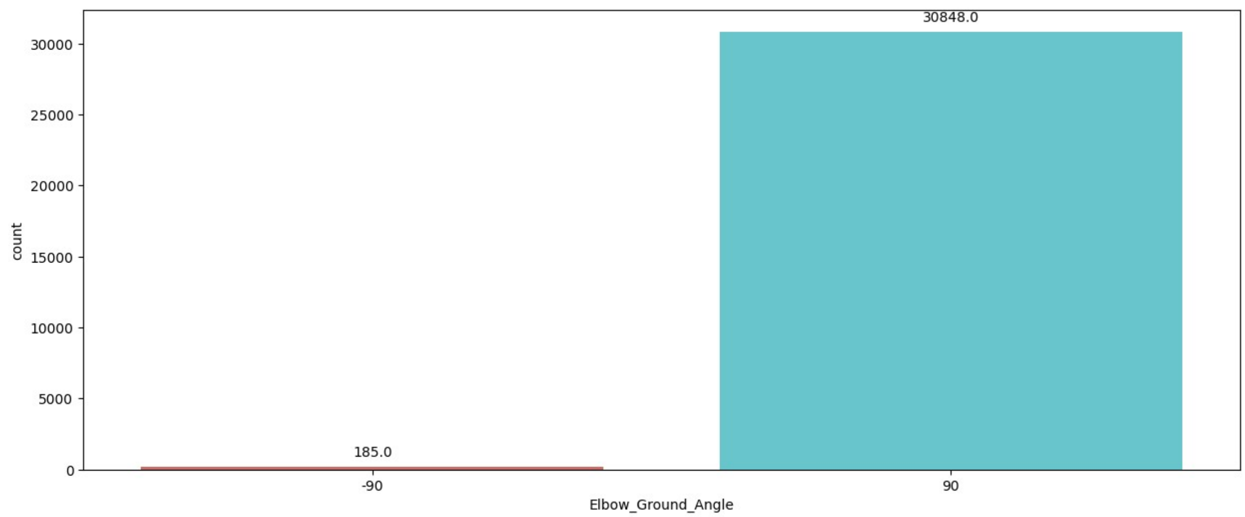
```
for i in discrete:
    print(df[i].value_counts())
    print()
```

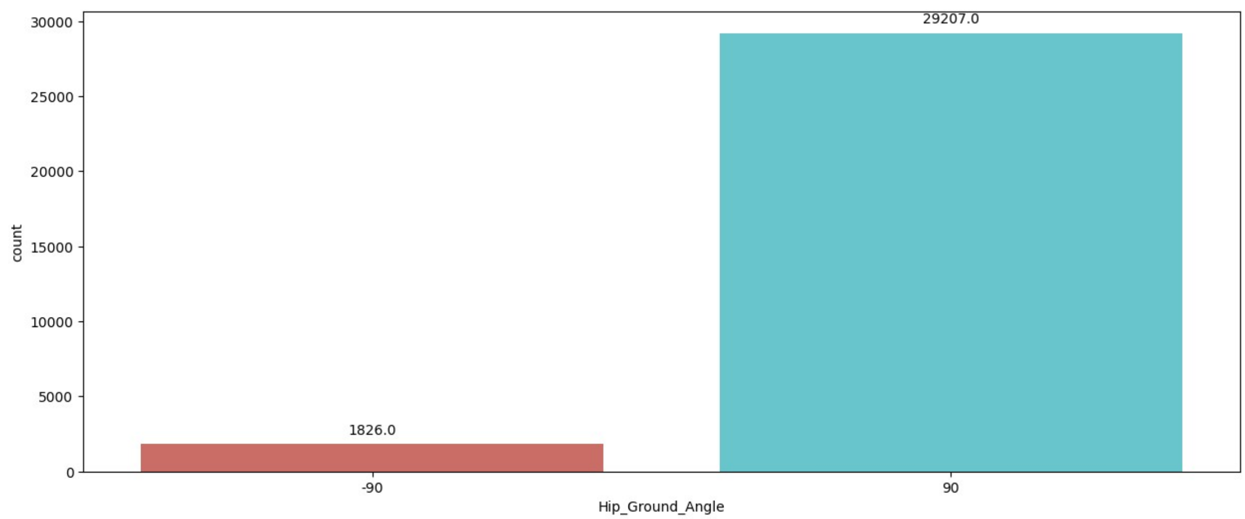
```
Shoulder_Ground_Angle
 90    30829
-90      204
Name: count, dtype: int64

Elbow_Ground_Angle
 90    30848
-90      185
Name: count, dtype: int64

Hip_Ground_Angle
 90    29207
-90     1826
Name: count, dtype: int64

Knee_Ground_Angle
 90    28584
-90     2449
Name: count, dtype: int64
```

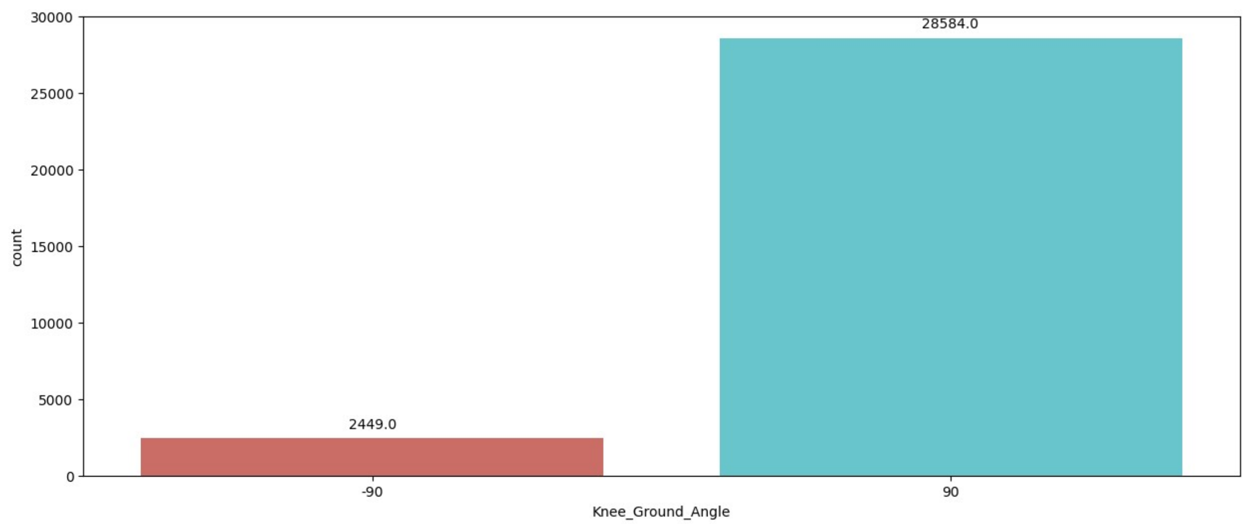
```
Ankle_Ground_Angle
 90     27410
-90      3623
Name: count, dtype: int64
```

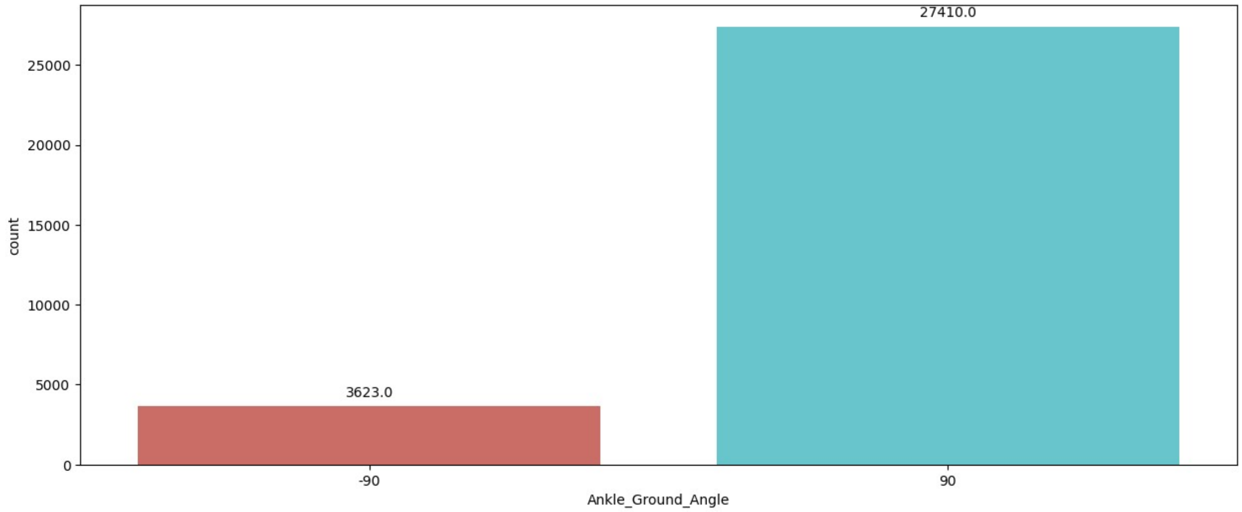
```
for i in discrete:
    plt.figure(figsize=(15, 6))
    ax = sns.countplot(x=i, data=df, palette='hls')

    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height}',
                    xy=(p.get_x() + p.get_width() / 2., height),
                    xytext=(0, 10),
                    textcoords='offset points',
                    ha='center', va='center')

    plt.show()
```



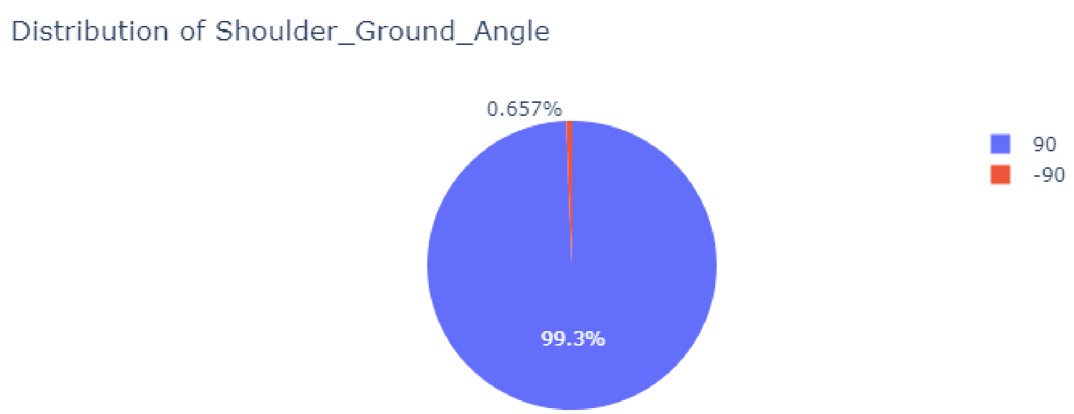

30848.0
185.0
count
-90
90
Elbow_Ground_Angle

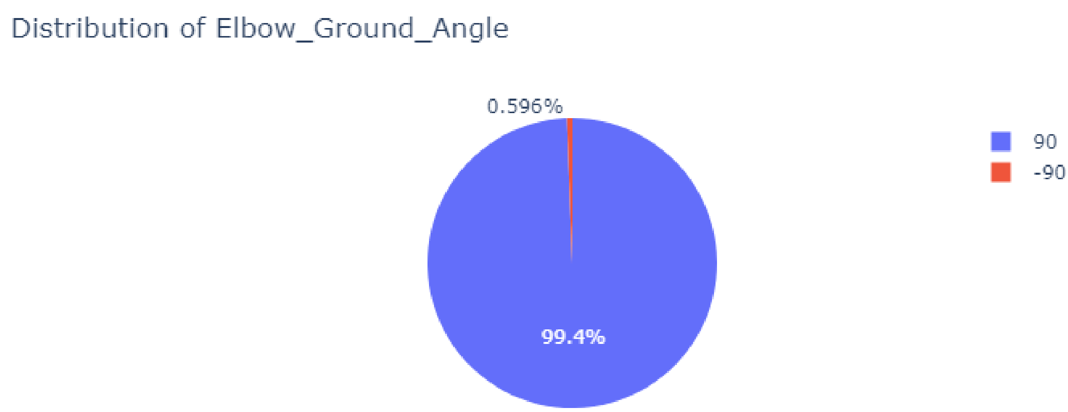

29207.0
1826.0
count
-90
90
Hip_Ground_Angle

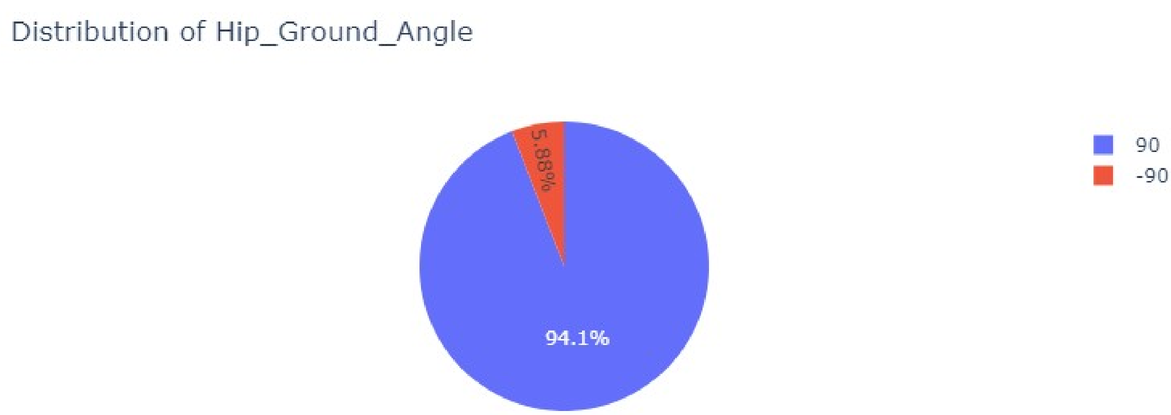

28584.0
2449.0
count
-90
90
Knee_Ground_Angle




```
import plotly.express as px

for i in discrete:
    counts = df[i].value_counts()
    fig = px.pie(counts, values=counts.values, names=counts.index,
title=f'Distribution of {i}')
    fig.show()
```

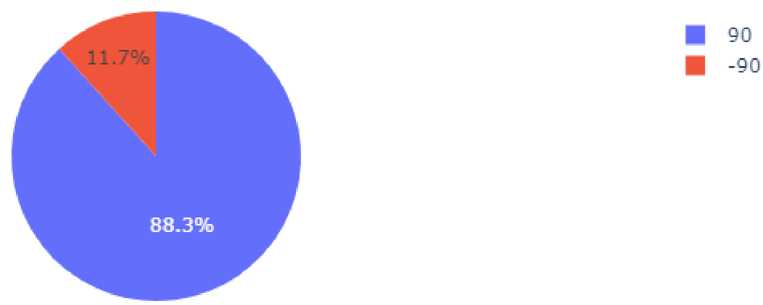



Distribution of Elbow_Ground_Angle
0.596%
99.4%
90
-90


Distribution of Hip_Ground_Angle
5.88%
94.1%
90
-90

Distribution of Knee_Ground_Angle

Distribution of Ankle_Ground_Angle

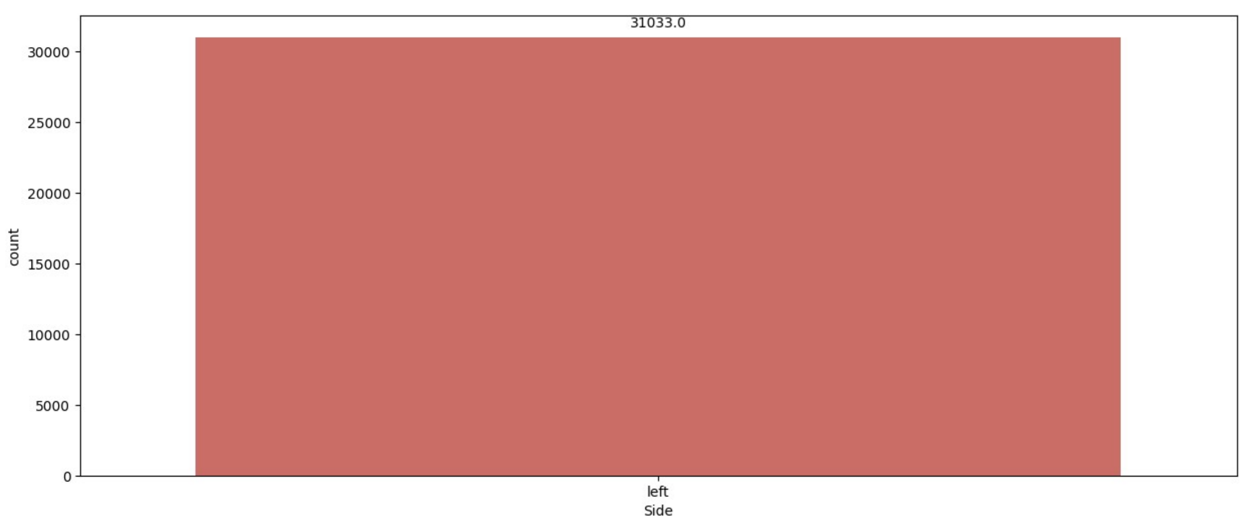


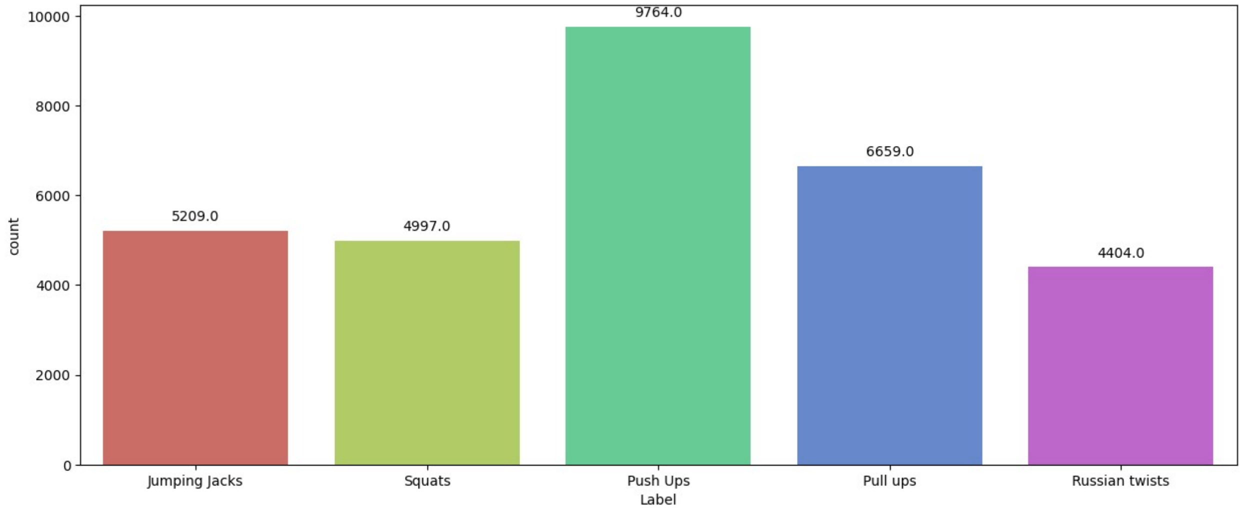

```
for i in categorical:
    print(i)
    print(df[i].unique())
    print()

Side
['left']

Label
['Jumping Jacks' 'Squats' 'Push Ups' 'Pull ups' 'Russian twists']


for i in categorical:
    print(i)
```

```
    print(df[i].value_counts())
    print()
```

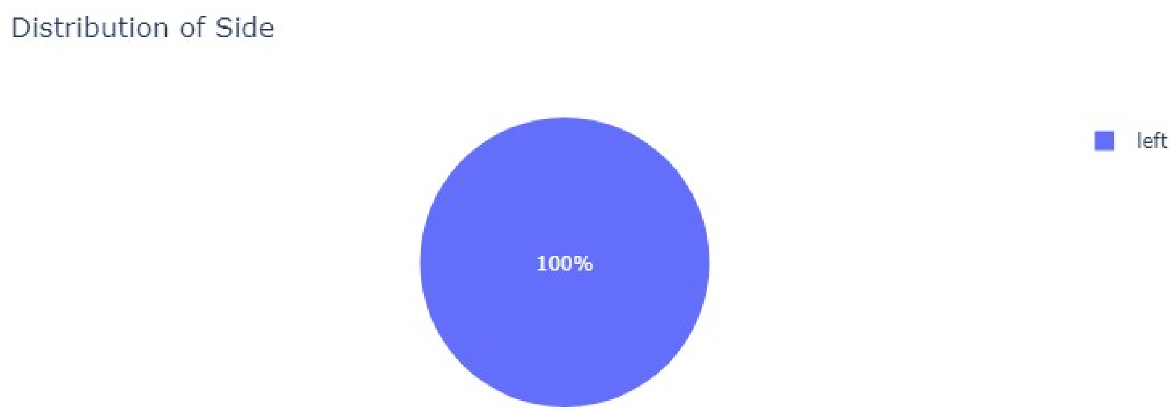
```
Side
Side
left    31033
Name: count, dtype: int64

Label
Label
Push Ups           9764
Pull ups           6659
Jumping Jacks      5209
Squats             4997
Russian twists     4404
Name: count, dtype: int64
```

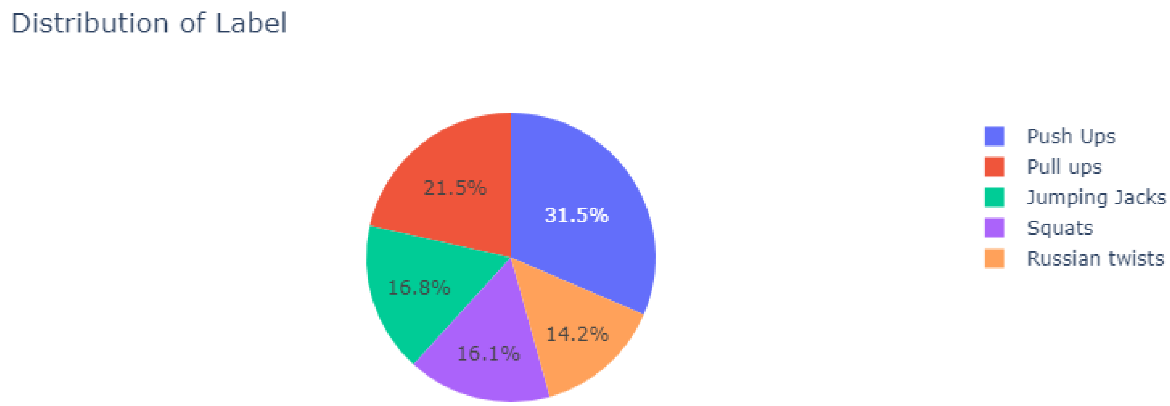
```
for i in categorical:
    plt.figure(figsize=(15, 6))
    ax = sns.countplot(x=i, data=df, palette='hls')

    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height}',
                    xy=(p.get_x() + p.get_width() / 2., height),
                    xytext=(0, 10),
                    textcoords='offset points',
                    ha='center', va='center')

    plt.show()
```



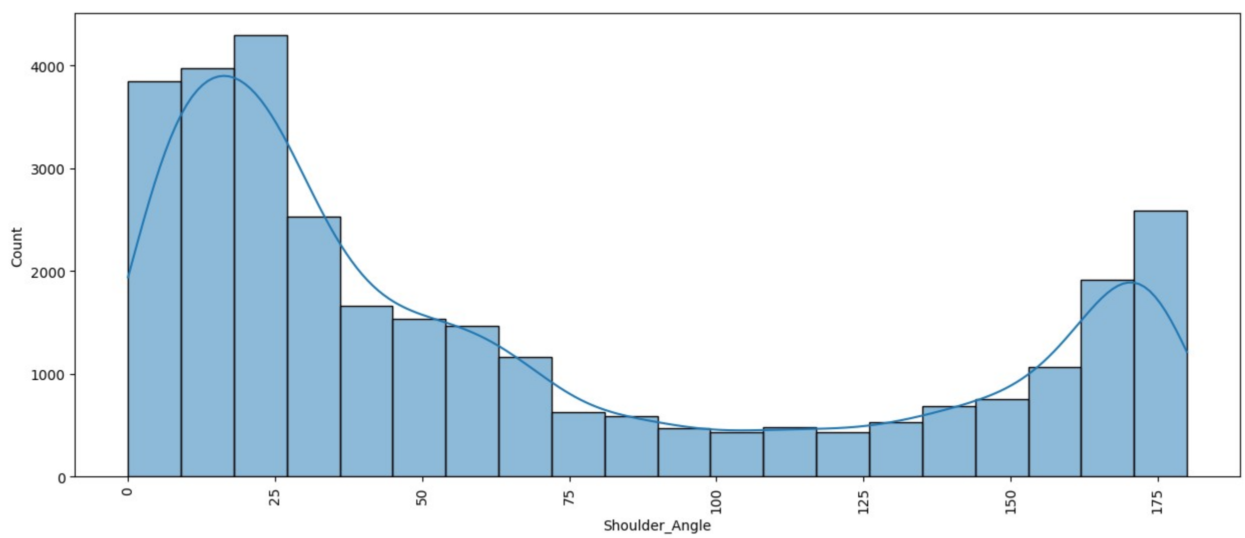


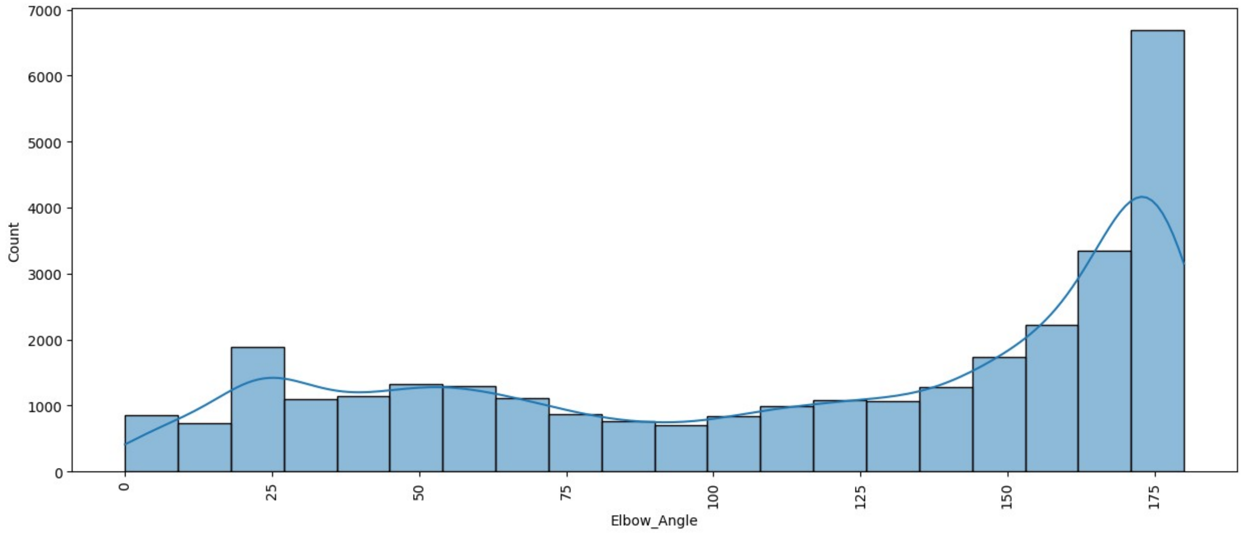

```
for i in categorical:
    counts = df[i].value_counts()
    fig = px.pie(counts, values=counts.values, names=counts.index, 
title=f'Distribution of {i}')
    fig.show()
```



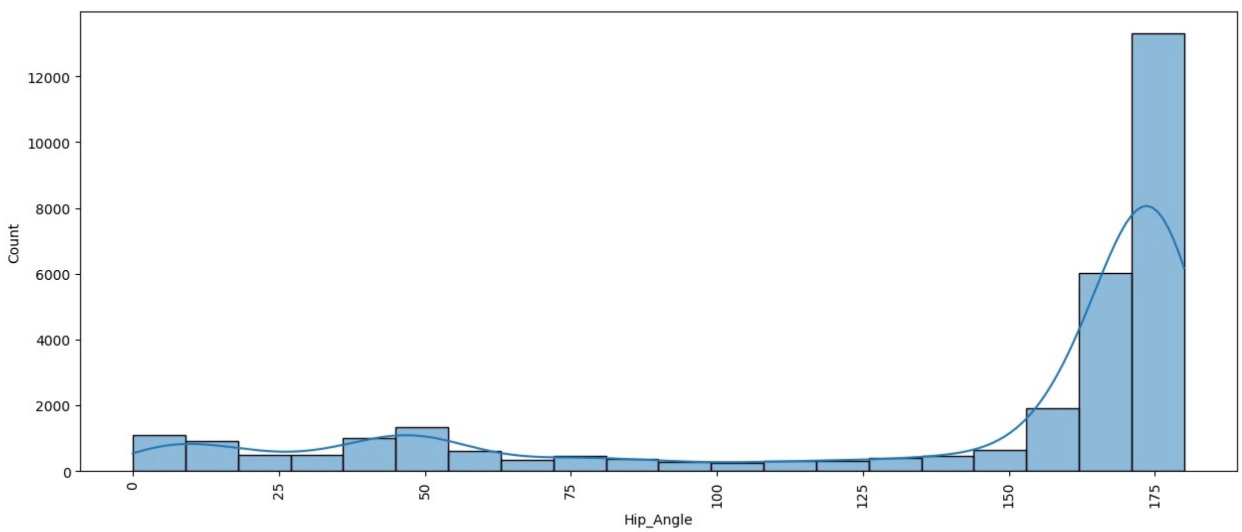



```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], bins = 20, kde = True, palette='hls')
    plt.xticks(rotation = 90)
    plt.show()
```

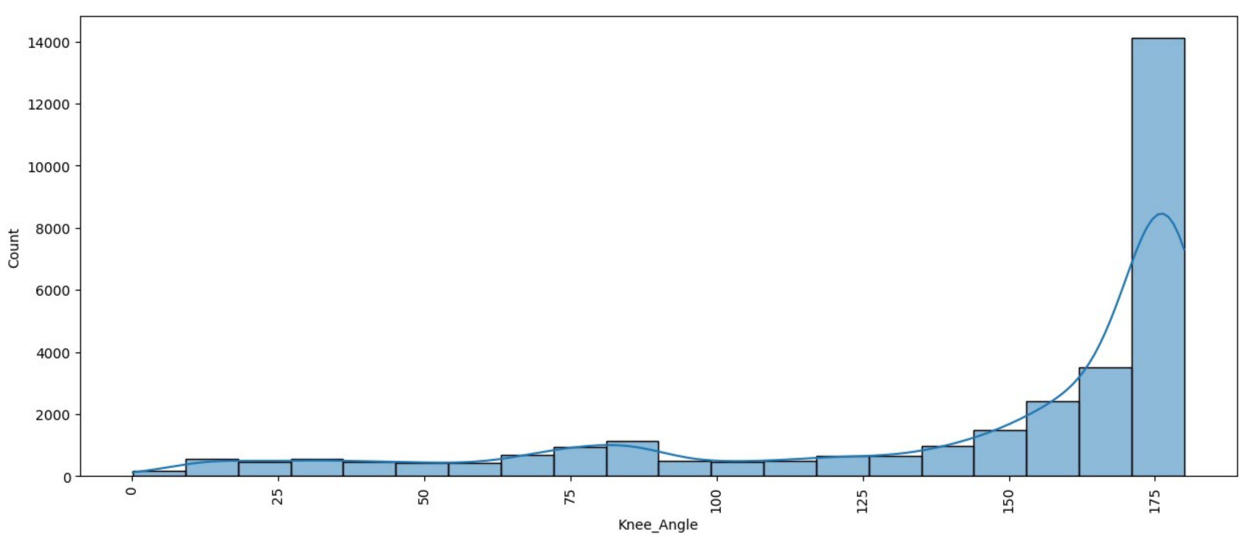




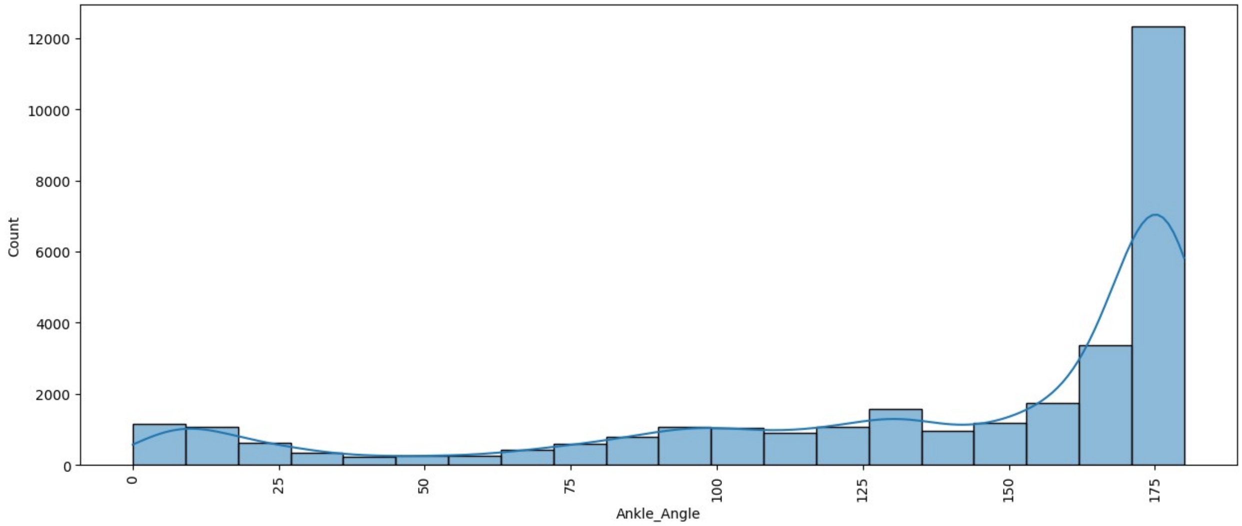
7000
6000
5000
4000
3000
2000
1000
0
Count
0
25
50
75
100
125
150
175
Elbow_Angle



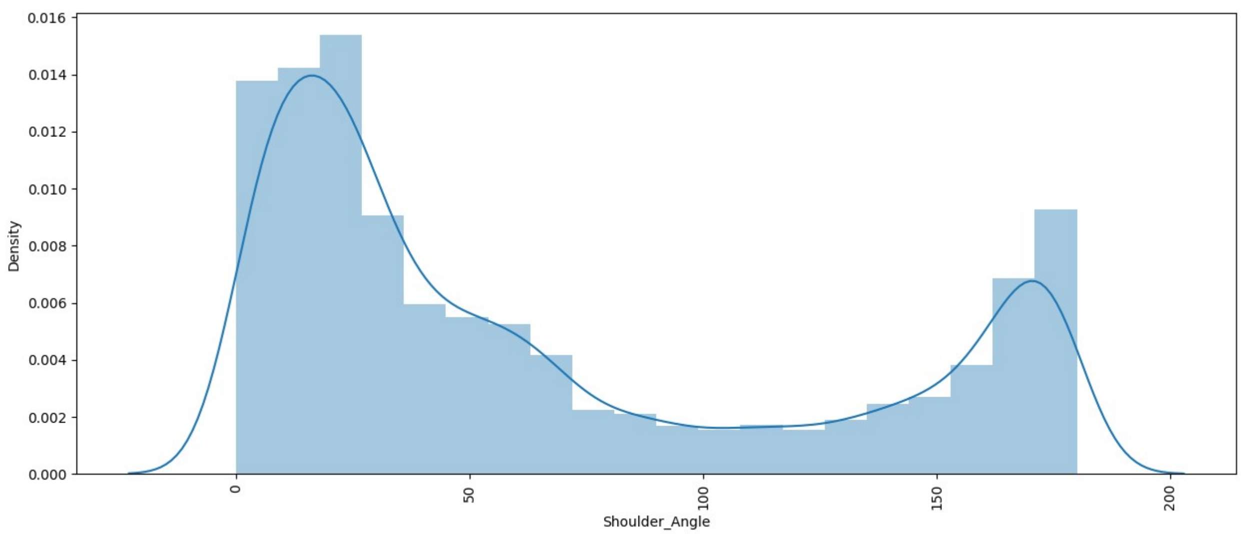
12000
10000
8000
6000
4000
2000
0
Count
0
25
50
75
100
125
150
175
Hip_Angle

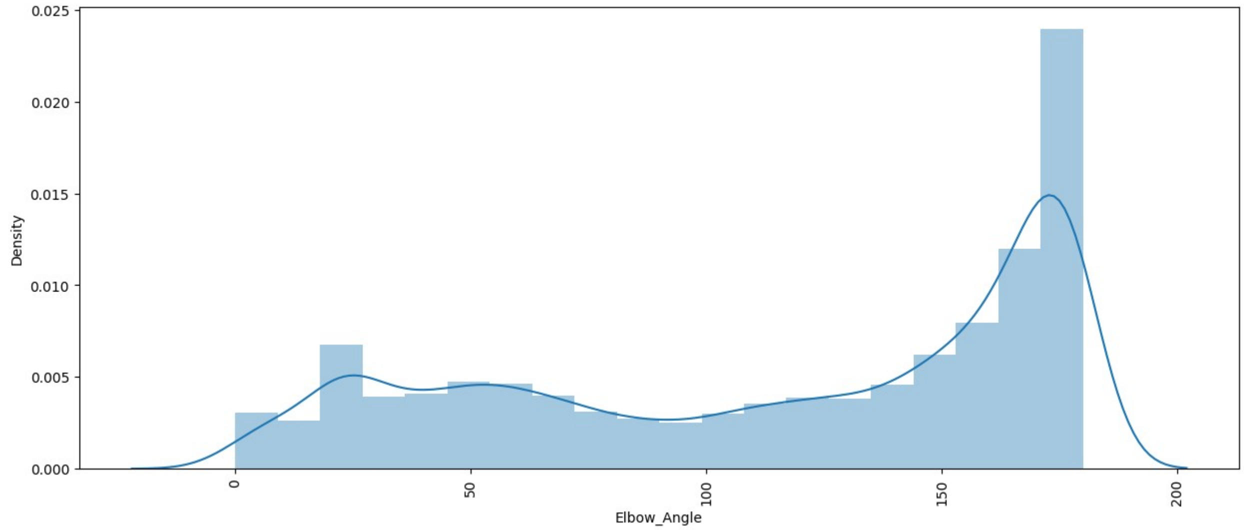


14000
12000
10000
8000
6000
4000
2000
0
Count
0
25
50
75
100
125
150
175
Knee_Angle

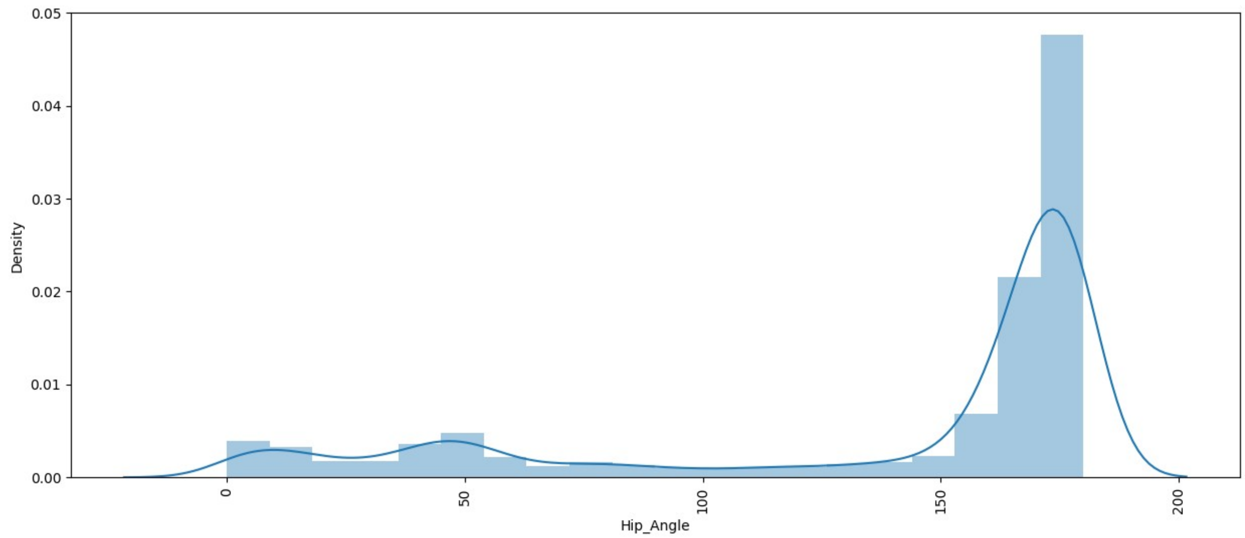



```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.distplot(df[i], bins = 20, kde = True)
    plt.xticks(rotation = 90)
    plt.show()
```

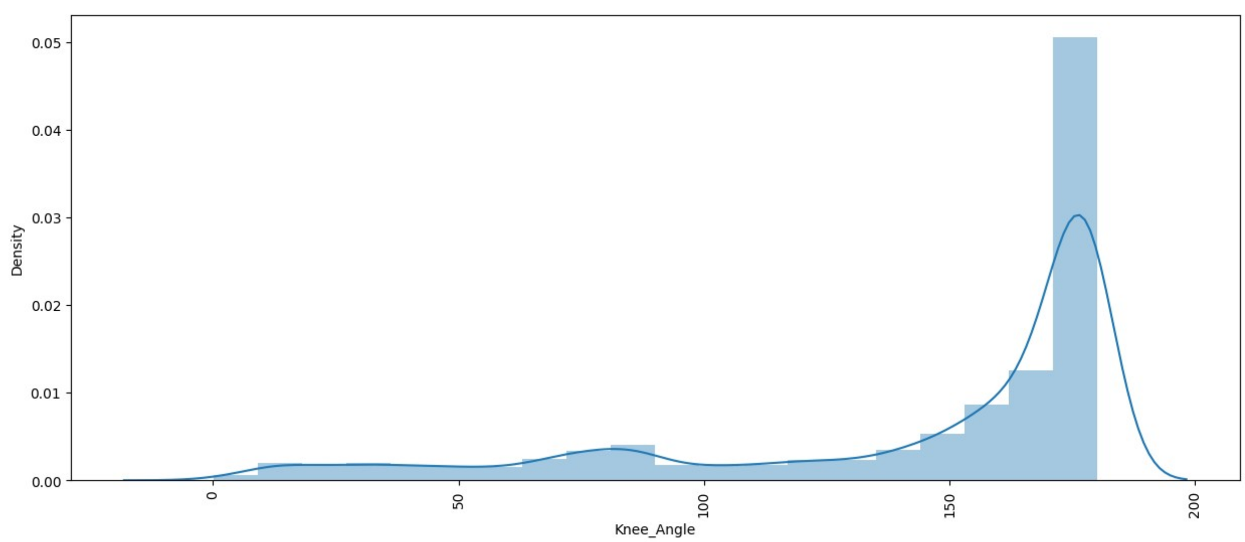




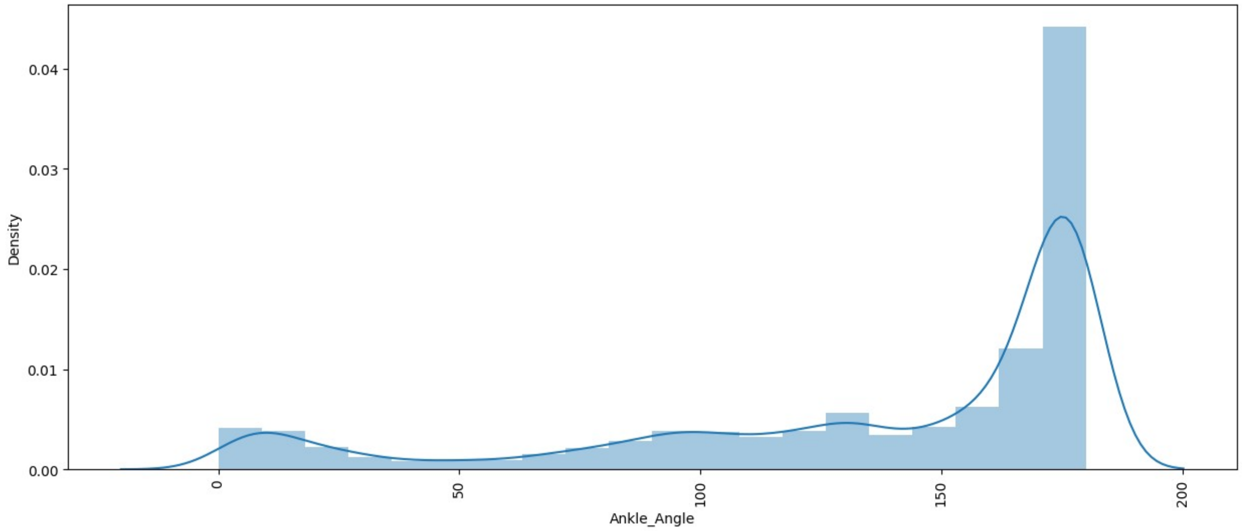
Density
Elbow_Angle
0.000
0.005
0.010
0.015
0.020
0.025
0
50
100
150
200



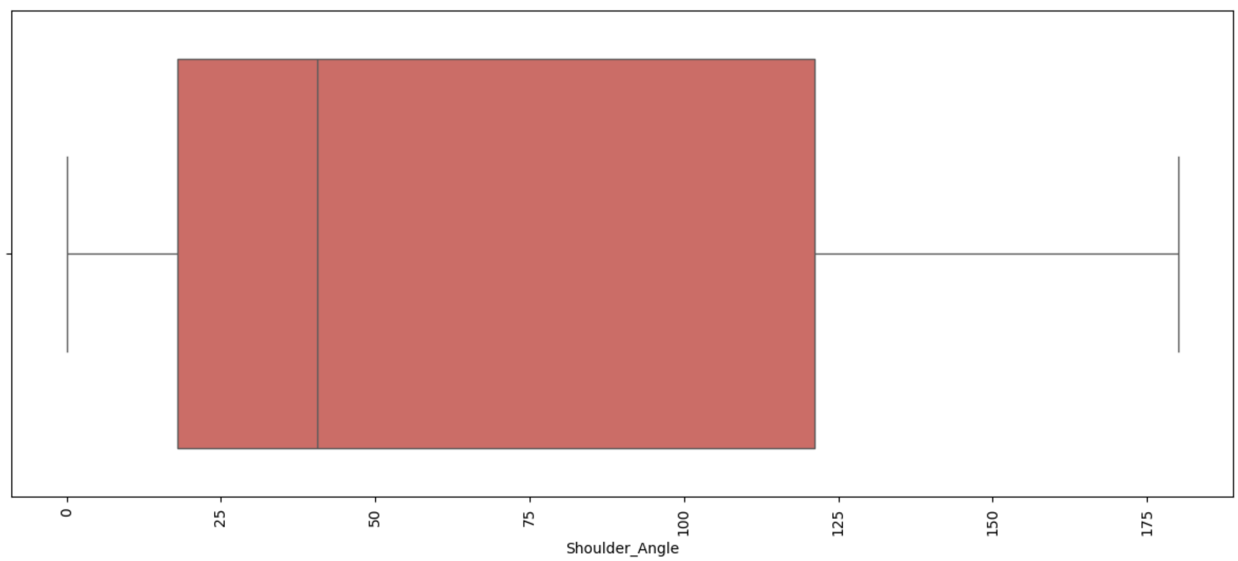
Density
Hip_Angle
0.00
0.01
0.02
0.03
0.04
0.05
0
50
100
150
200

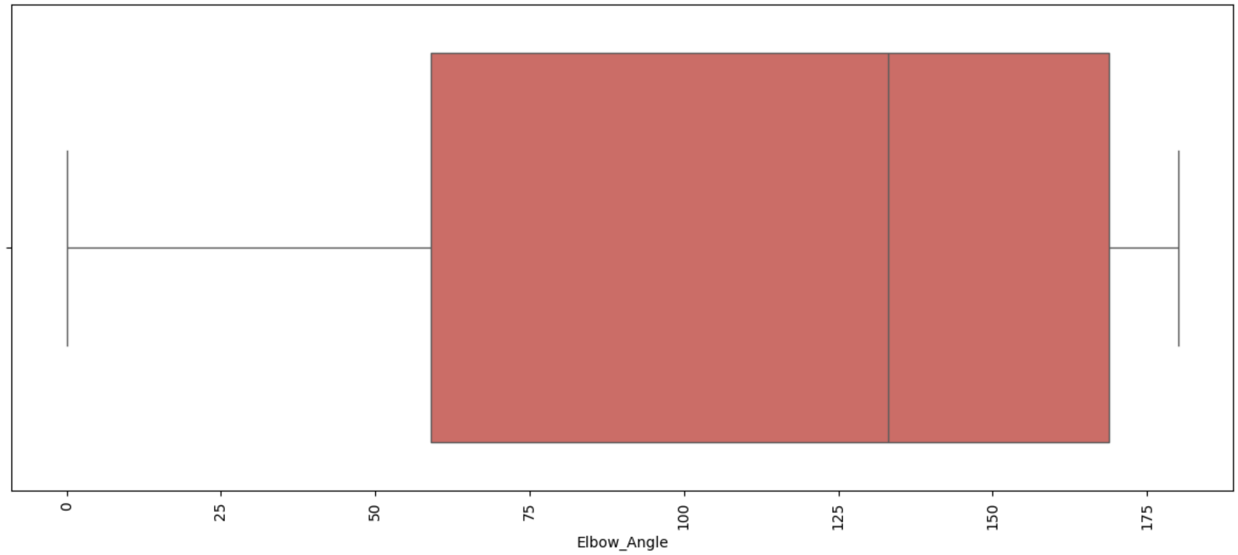


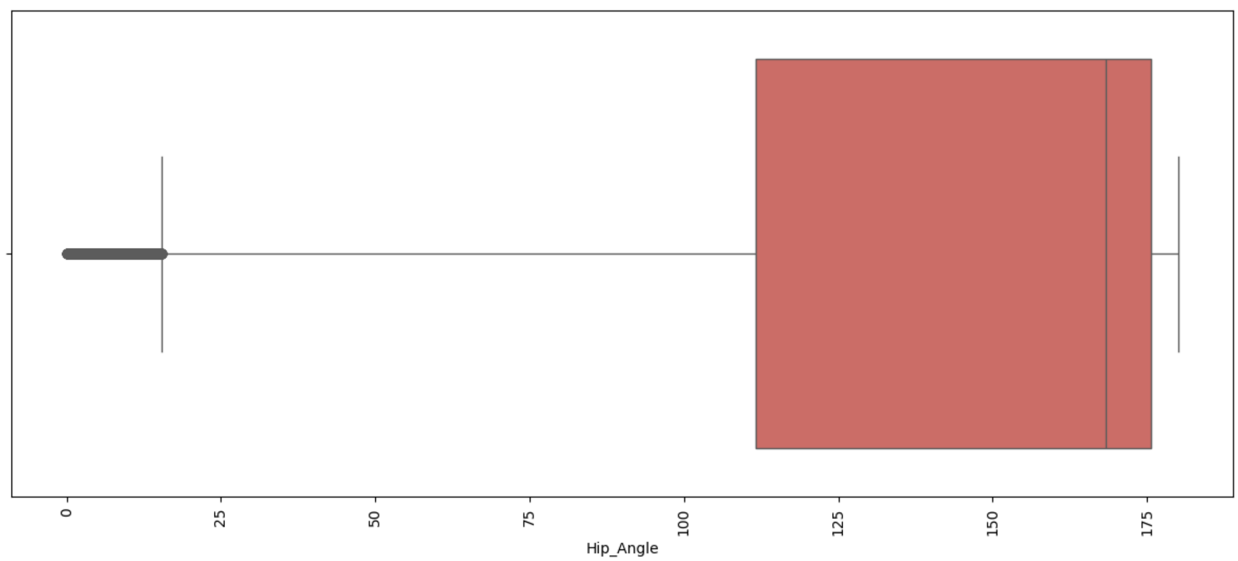
Density
Knee_Angle
0.00
0.01
0.02
0.03
0.04
0.05
0
50
100
150
200

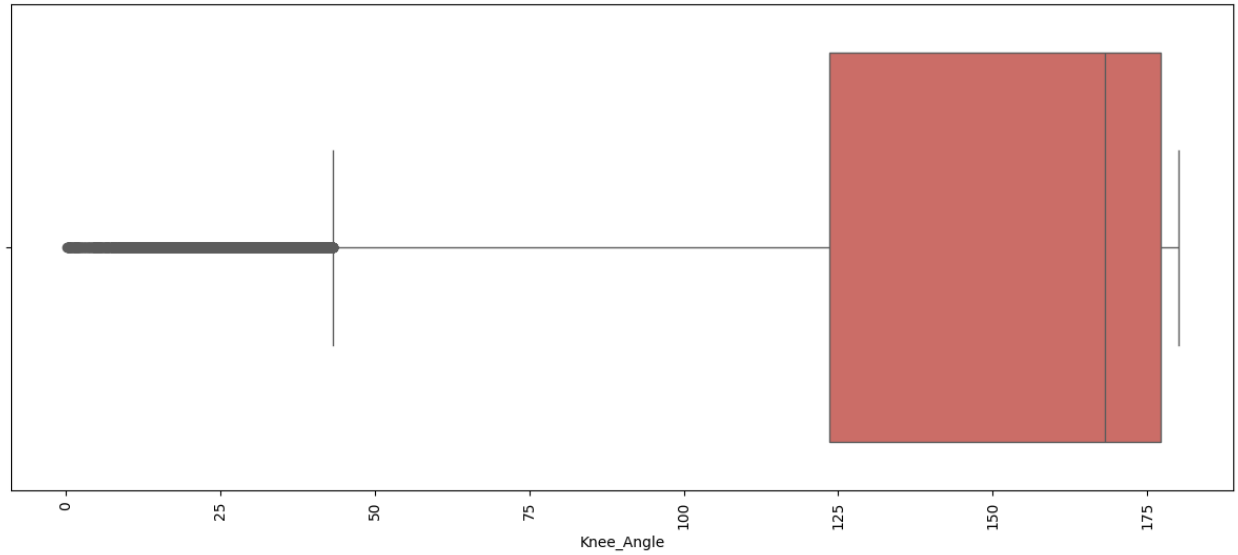


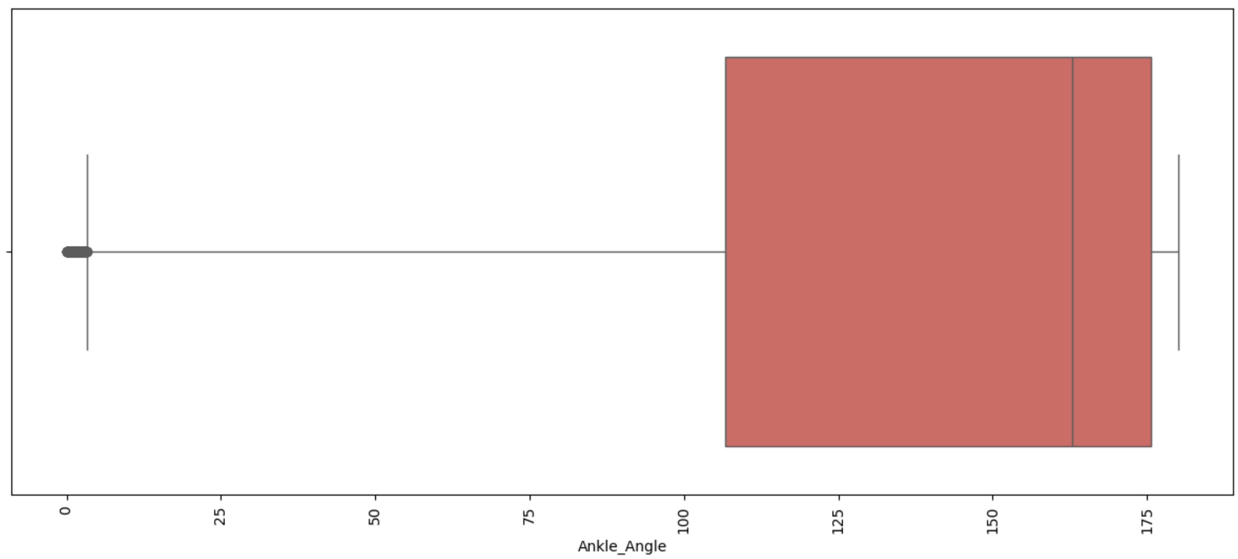

```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.boxplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```



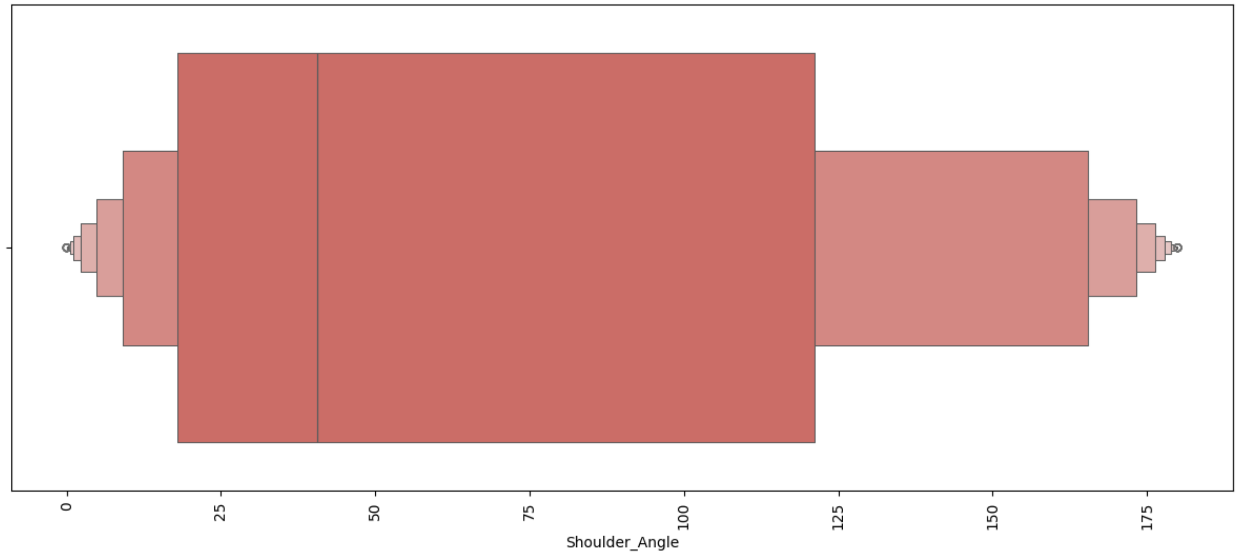

0
25
50
75
100
125
150
175
Elbow_Angle

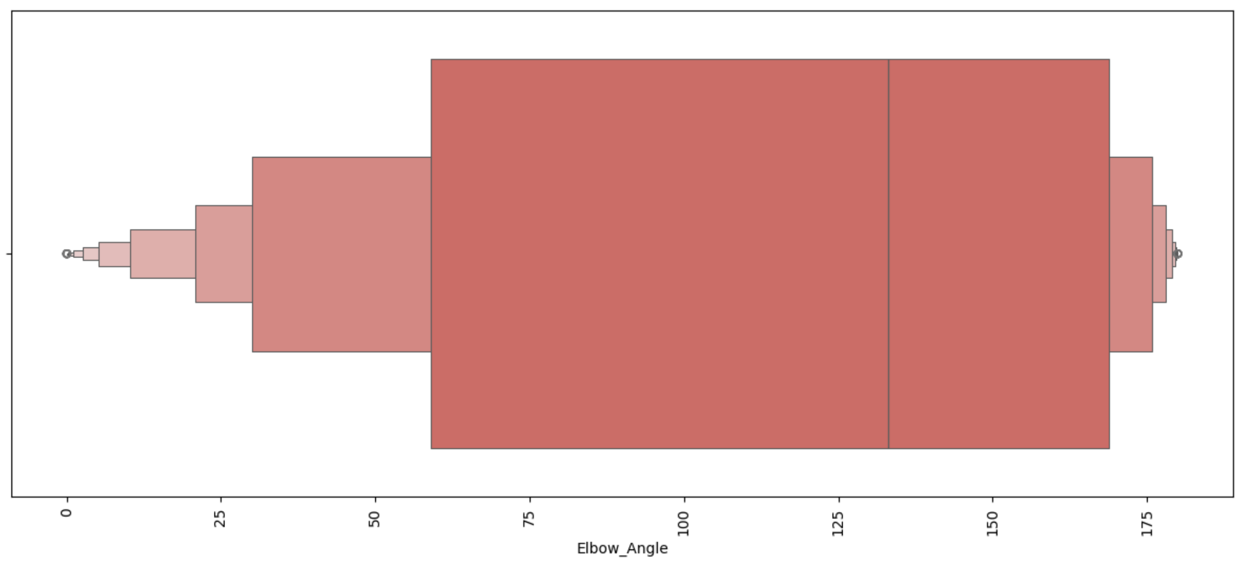

0
25
50
75
100
125
150
175
Hip_Angle

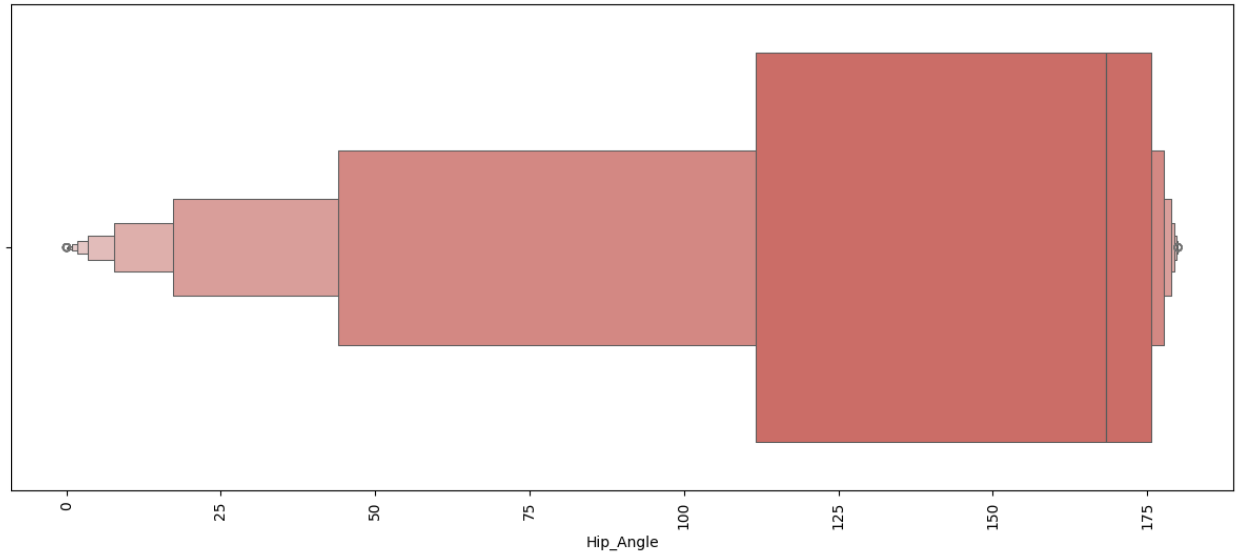


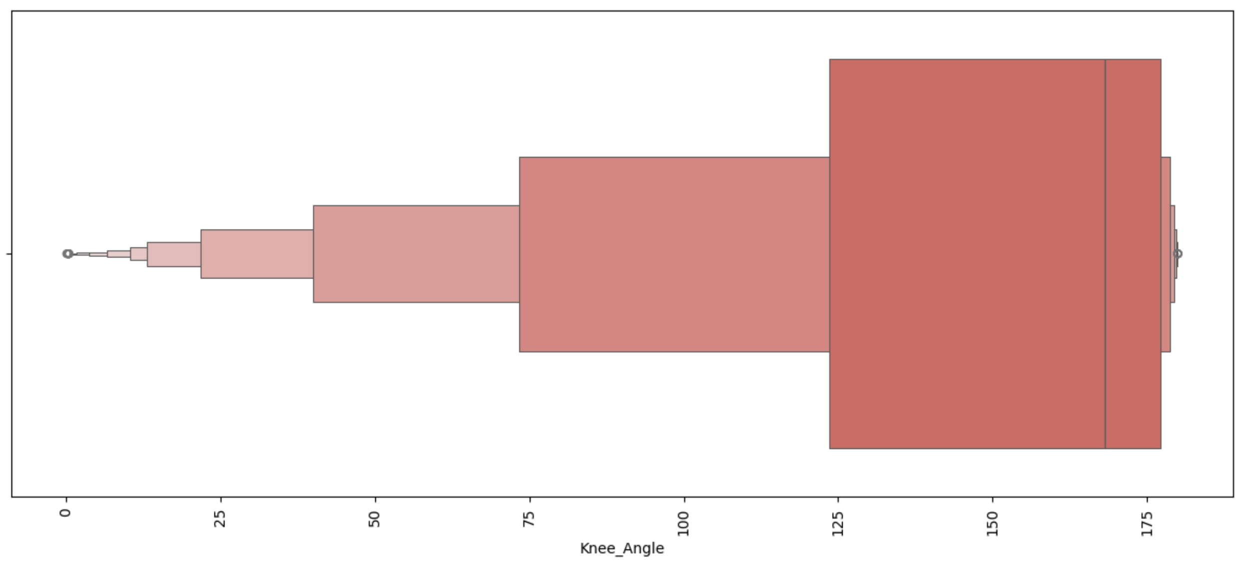


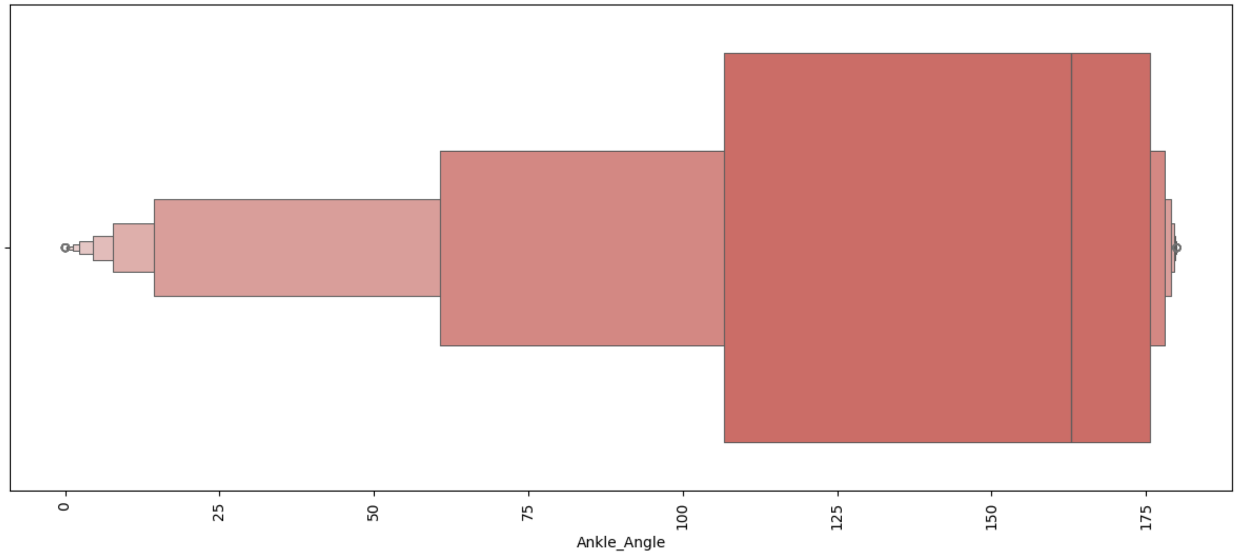

```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.boxenplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```


0
25
50
75
100
125
150
175
Shoulder_Angle

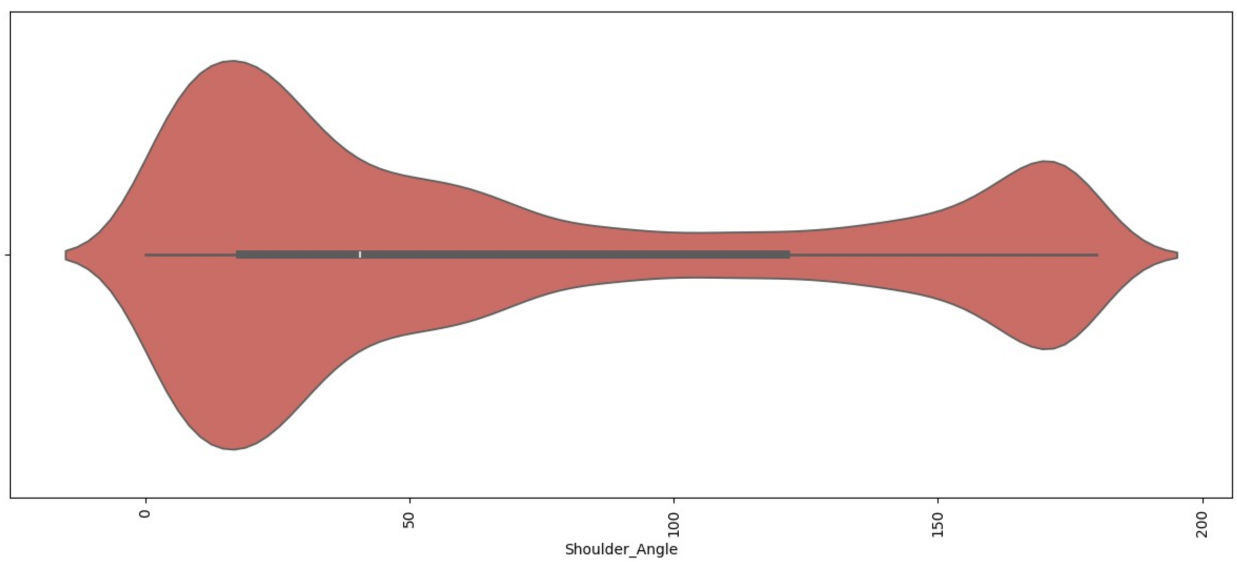

0
25
50
75
100
125
150
175
Elbow_Angle

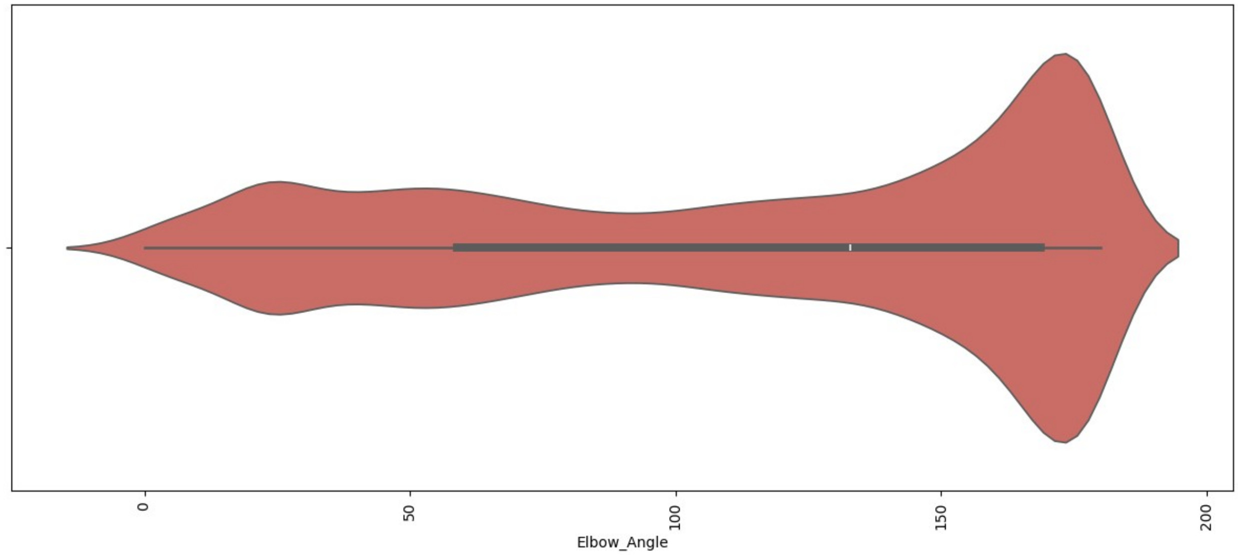

0
25
50
75
100
125
150
175
Hip_Angle

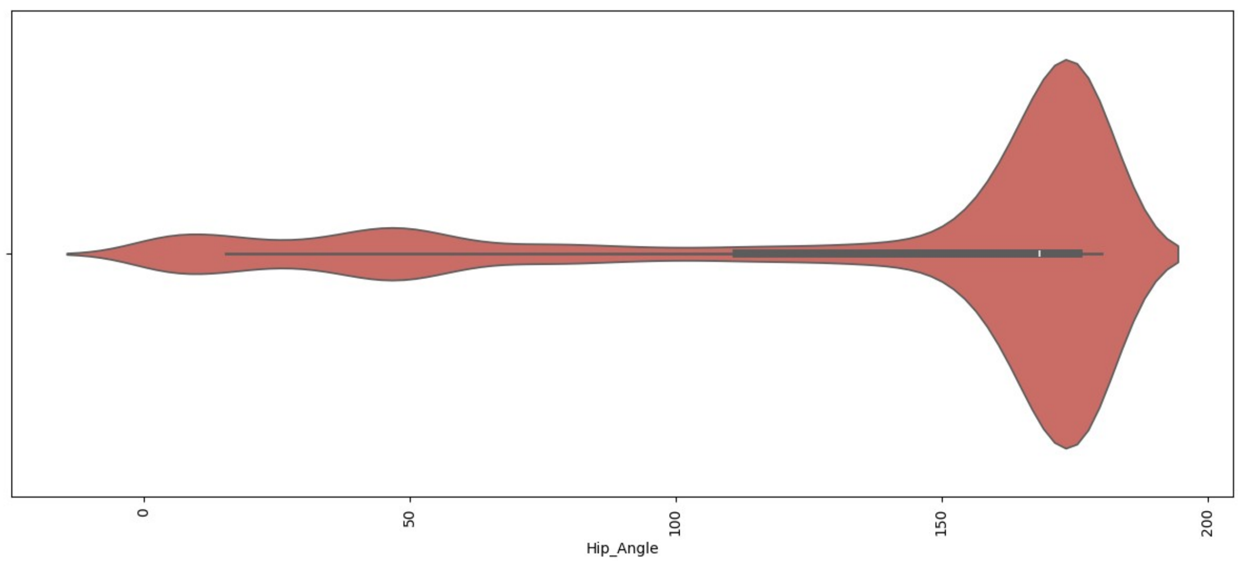

0
25
50
75
100
125
150
175
Knee_Angle

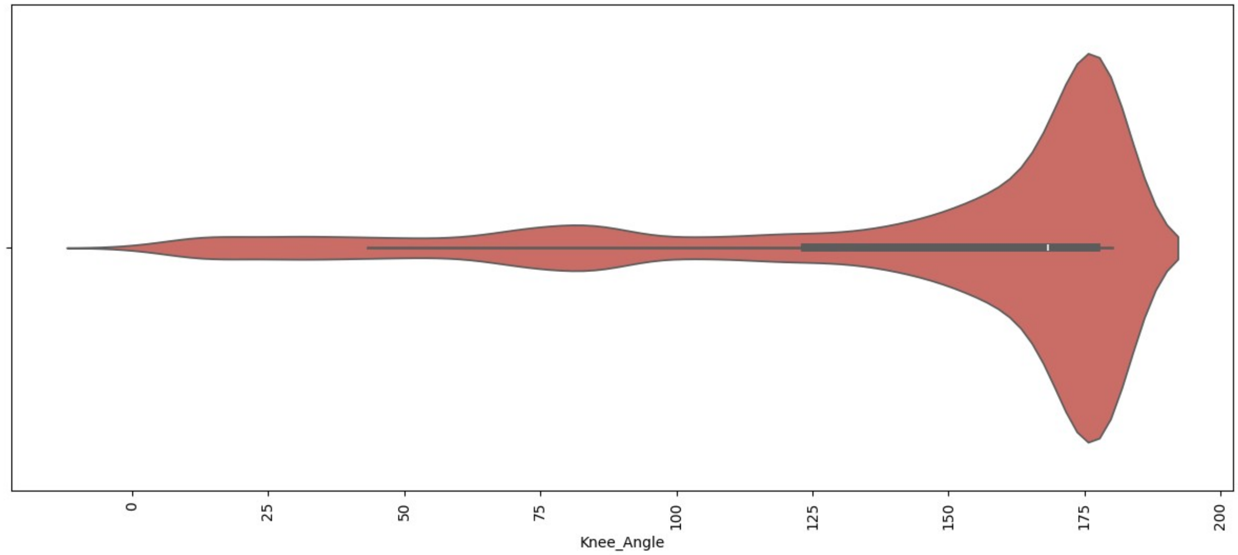


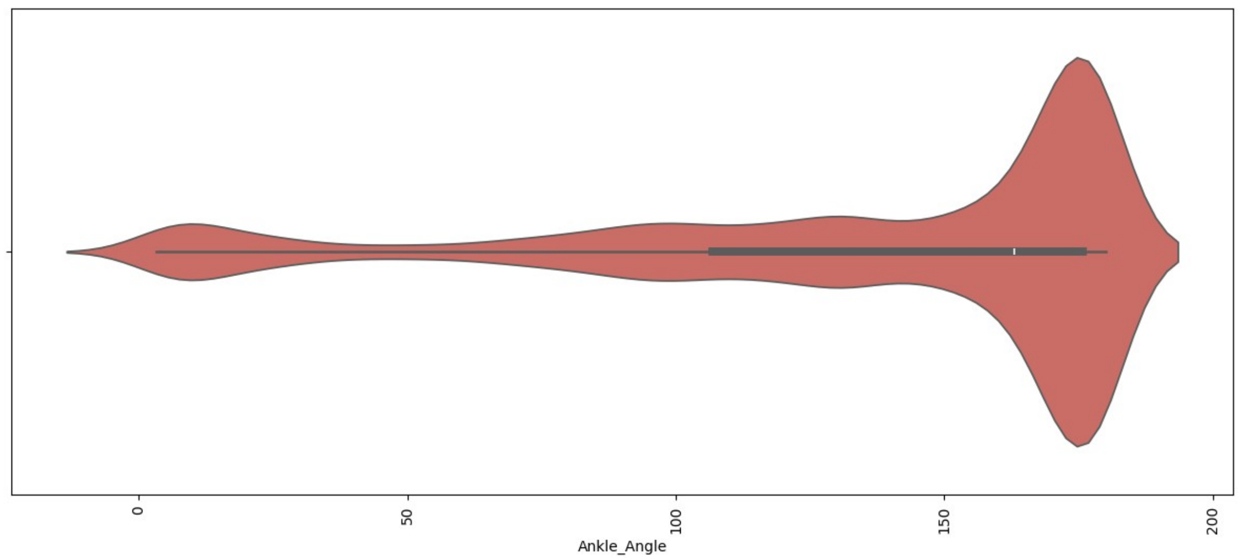

```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.violinplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```



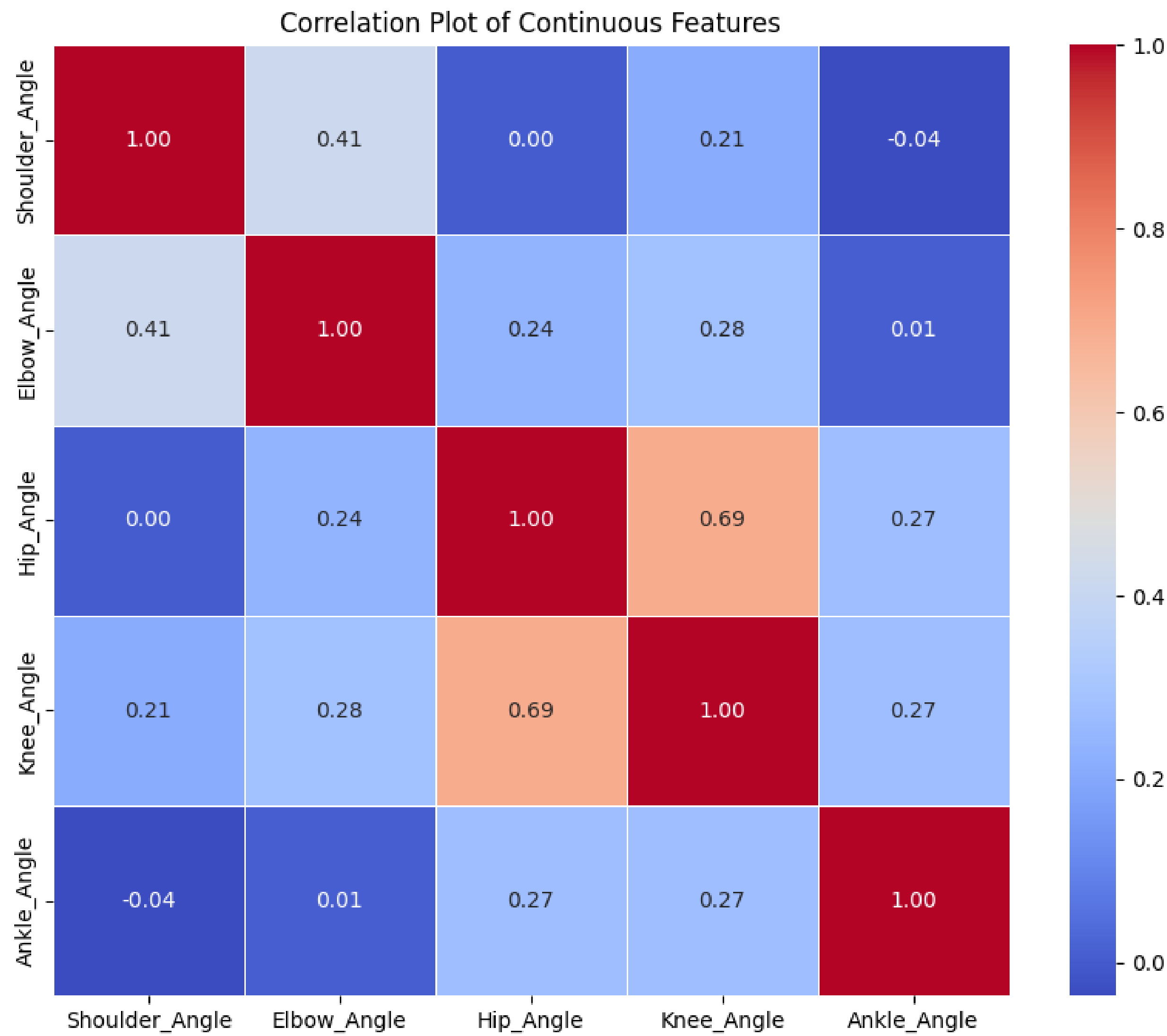

0
50
100
150
200
Elbow_Angle


0
50
100
150
200
Hip_Angle






```
corr_matrix = df[continuous].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5)
plt.title('Correlation Plot of Continuous Features')
plt.show()
```




```
corr_matrix

                Shoulder_Angle  Elbow_Angle  Hip_Angle  Knee_Angle  \
Shoulder_Angle        1.000000     0.414948   0.002495    0.214424
Elbow_Angle           0.414948     1.000000   0.236929    0.283897
Hip_Angle             0.002495     0.236929   1.000000    0.690139
Knee_Angle            0.214424     0.283897   0.690139    1.000000
Ankle_Angle          -0.036249     0.005913   0.274283    0.273995

                Ankle_Angle
Shoulder_Angle    -0.036249
Elbow_Angle        0.005913
Hip_Angle          0.274283
Knee_Angle         0.273995
Ankle_Angle        1.000000
```

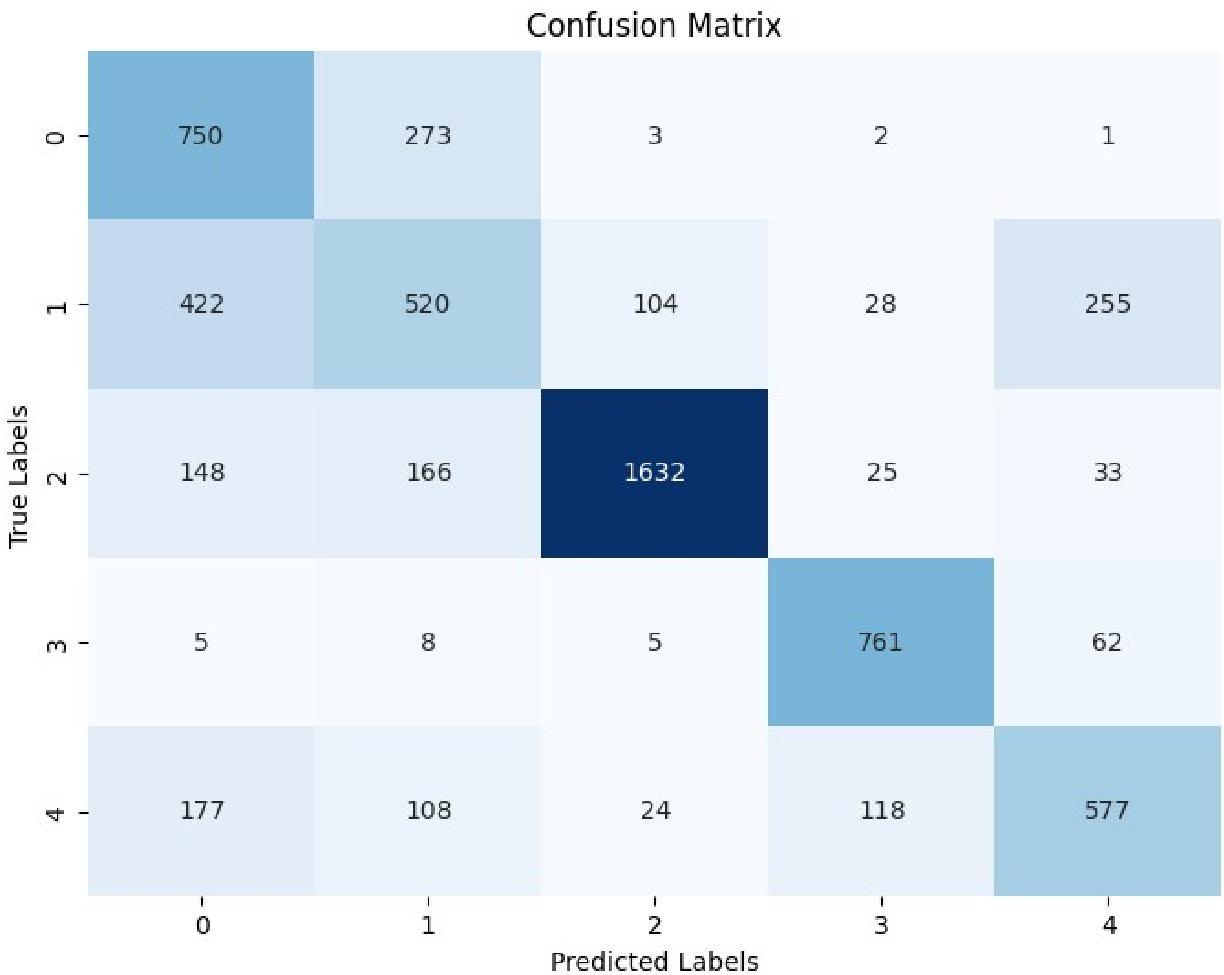
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

from imblearn.over_sampling import SMOTE
from collections import Counter

label_encoder = LabelEncoder()
df['Side'] = label_encoder.fit_transform(df['Side'])
df['Label'] = label_encoder.fit_transform(df['Label'])

X = df[['Shoulder_Ground_Angle', 'Elbow_Ground_Angle',
'Hip_Ground_Angle',
        'Knee_Ground_Angle', 'Ankle_Ground_Angle', 'Shoulder_Angle',
        'Elbow_Angle', 'Hip_Angle', 'Knee_Angle', 'Ankle_Angle',
'Side']]
y = df['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_smote, y_train_smote)
```

```
LogisticRegression(max_iter=1000)
```

```python
y_pred = model.predict(X_test)

print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

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

           0       0.50      0.73      0.59      1029
           1       0.48      0.39      0.43      1329
           2       0.92      0.81      0.87      2004
           3       0.81      0.90      0.86       841
           4       0.62      0.57      0.60      1004

    accuracy                           0.68      6207
   macro avg       0.67      0.68      0.67      6207
weighted avg       0.70      0.68      0.68      6207
```

```
Confusion Matrix:
 [[ 750  273    3    2    1]
 [ 422  520  104   28  255]
 [ 148  166 1632   25   33]
 [   5    8    5  761   62]
 [ 177  108   24  118  577]]

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```




```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```python
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_smote, y_train_smote)
```

```
DecisionTreeClassifier(random_state=42)
```

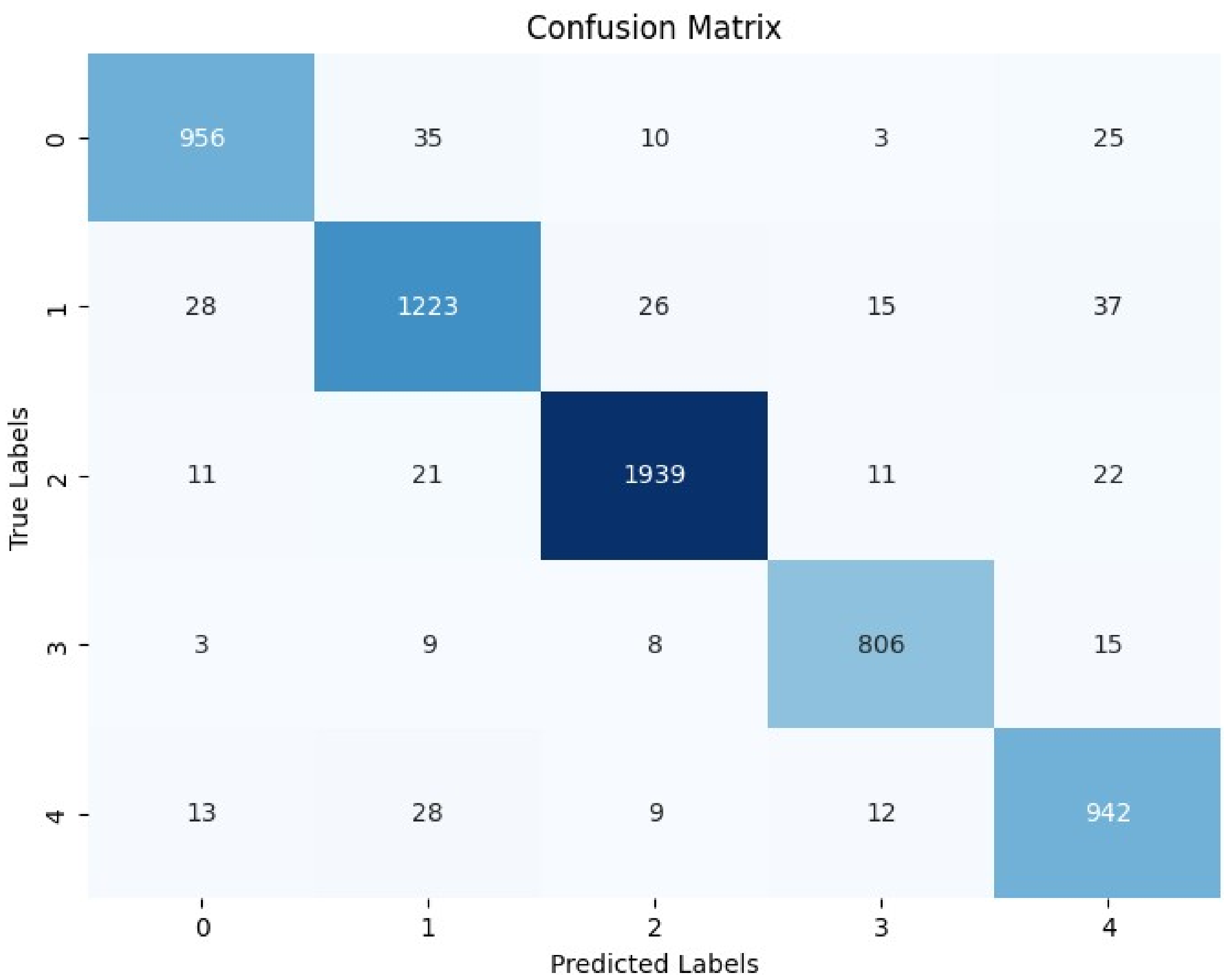
```python
y_pred = dt_model.predict(X_test)
```

```python
print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

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

           0       0.95      0.93      0.94      1029
           1       0.93      0.92      0.92      1329
           2       0.97      0.97      0.97      2004
           3       0.95      0.96      0.95       841
           4       0.90      0.94      0.92      1004

    accuracy                           0.95      6207
   macro avg       0.94      0.94      0.94      6207
weighted avg       0.95      0.95      0.95      6207

Confusion Matrix:
 [[ 956   35   10    3   25]
 [  28 1223   26   15   37]
 [  11   21 1939   11   22]
 [   3    9    8  806   15]
 [  13   28    9   12  942]]
```

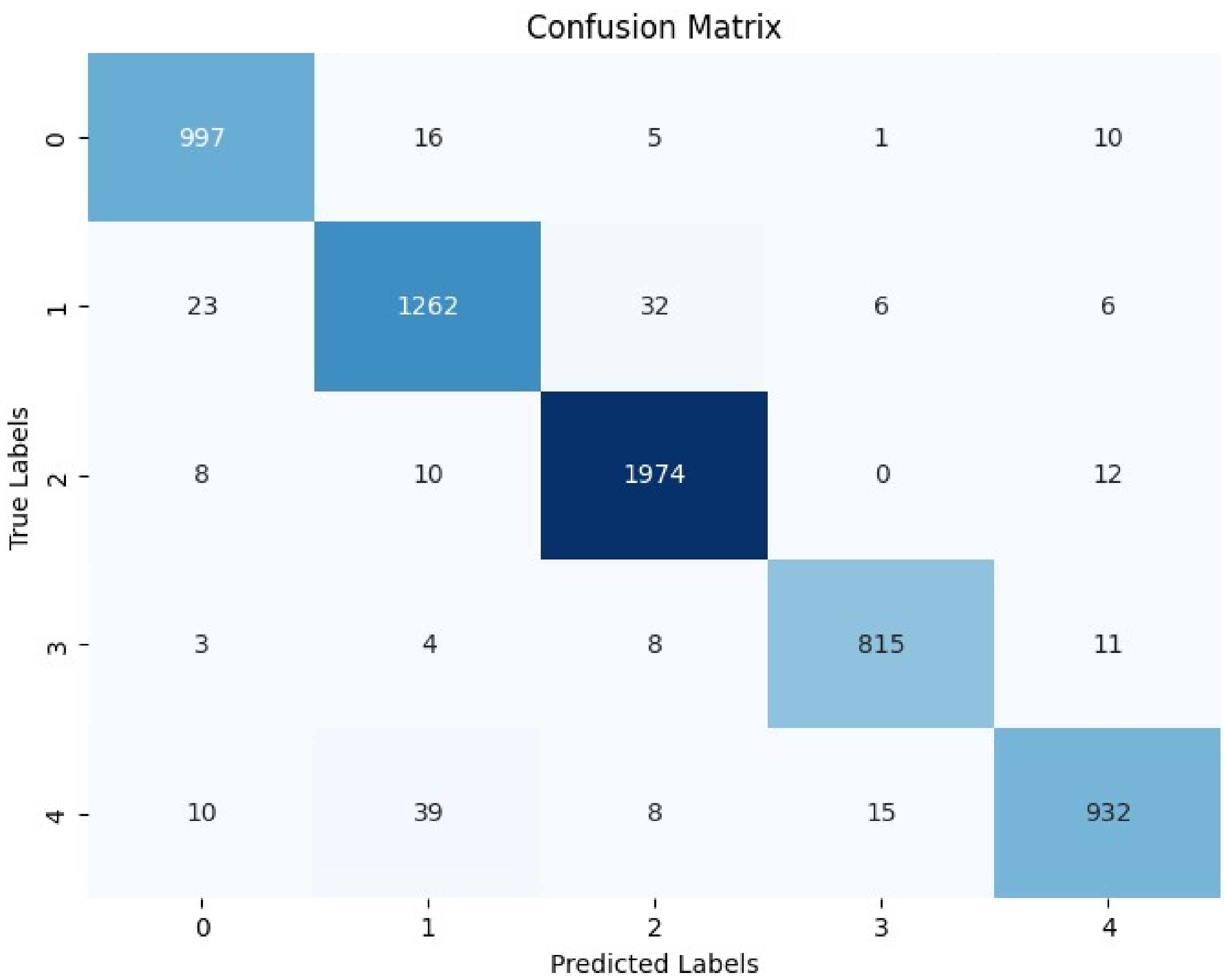
```python
cm = confusion_matrix(y_test, y_pred)
```

```python
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

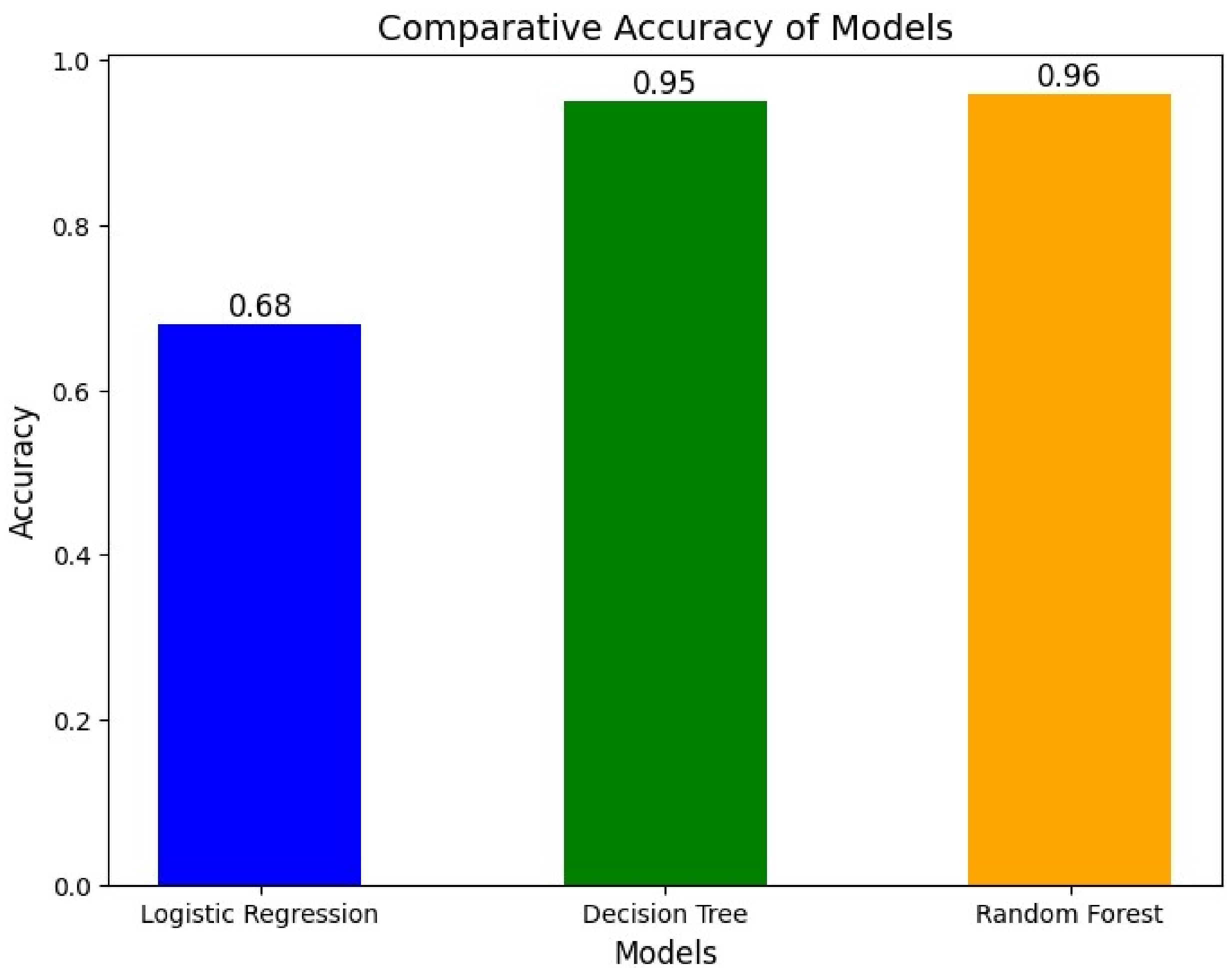



```
rf_model = RandomForestClassifier(n_estimators=10, random_state=42)
rf_model.fit(X_train_smote, y_train_smote)

RandomForestClassifier(n_estimators=10, random_state=42)

y_pred = rf_model.predict(X_test)

print("Classification Report:\n", classification_report(y_test,
y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.97      0.96      1029
           1       0.95      0.95      0.95      1329
           2       0.97      0.99      0.98      2004
           3       0.97      0.97      0.97       841
           4       0.96      0.93      0.94      1004
```

```
    accuracy                           0.96      6207
   macro avg       0.96      0.96      0.96      6207
weighted avg       0.96      0.96      0.96      6207

Confusion Matrix:
 [[ 997   16    5    1   10]
 [  23 1262   32    6    6]
 [   8   10 1974    0   12]
 [   3    4    8  815   11]
 [  10   39    8   15  932]]

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



```
accuracy_scores = {
    'Logistic Regression': 0.68,
    'Decision Tree': 0.95,
    'Random Forest': 0.96
}

fig, ax = plt.subplots(figsize=(8, 6))

model_names = list(accuracy_scores.keys())
accuracy_values = list(accuracy_scores.values())

bar_positions = np.arange(len(model_names))

ax.bar(bar_positions, accuracy_values, color=['blue', 'green',
'orange'], width=0.5)

ax.set_title('Comparative Accuracy of Models', fontsize=14)
ax.set_xlabel('Models', fontsize=12)
ax.set_ylabel('Accuracy', fontsize=12)

ax.set_xticks(bar_positions)
ax.set_xticklabels(model_names)

for i in range(len(accuracy_values)):
    ax.text(i, accuracy_values[i] + 0.01, f'{accuracy_values[i]:.2f}',
ha='center', fontsize=12)

plt.show()
```




```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import OneHotEncoder

y_train_smote_array = np.array(y_train_smote)
y_test_array = np.array(y_test)

encoder = OneHotEncoder(sparse=False)
y_train_encoded = encoder.fit_transform(y_train_smote_array.reshape(-
1, 1))  # Reshape for one-hot encoding
y_test_encoded = encoder.transform(y_test_array.reshape(-1, 1))

dl_model = Sequential()
dl_model.add(Dense(64, activation='relu',
input_dim=X_train_smote.shape[1]))
dl_model.add(Dense(32, activation='relu'))
dl_model.add(Dense(5, activation='softmax'))
```

```python
dl_model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])

history = dl_model.fit(X_train_smote, y_train_encoded, epochs=20,
batch_size=32, validation_data=(X_test, y_test_encoded), verbose=1)
```

```
Epoch 1/20
1213/1213 [==============================] - 2s 1ms/step - loss:
0.6033 - accuracy: 0.7755 - val_loss: 0.4117 - val_accuracy: 0.8436
Epoch 2/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.4068 - accuracy: 0.8541 - val_loss: 0.3624 - val_accuracy: 0.8816
Epoch 3/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.3537 - accuracy: 0.8776 - val_loss: 0.3380 - val_accuracy: 0.8937
Epoch 4/20
1213/1213 [==============================] - 3s 2ms/step - loss:
0.3233 - accuracy: 0.8876 - val_loss: 0.3040 - val_accuracy: 0.8974
Epoch 5/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.3012 - accuracy: 0.8936 - val_loss: 0.2896 - val_accuracy: 0.9029
Epoch 6/20
1213/1213 [==============================] - 2s 1ms/step - loss:
0.2856 - accuracy: 0.9007 - val_loss: 0.2825 - val_accuracy: 0.9020
Epoch 7/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2740 - accuracy: 0.9035 - val_loss: 0.2711 - val_accuracy: 0.9170
Epoch 8/20
1213/1213 [==============================] - 3s 2ms/step - loss:
0.2638 - accuracy: 0.9090 - val_loss: 0.2598 - val_accuracy: 0.9124
Epoch 9/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2547 - accuracy: 0.9114 - val_loss: 0.2559 - val_accuracy: 0.9161
Epoch 10/20
1213/1213 [==============================] - 3s 2ms/step - loss:
0.2484 - accuracy: 0.9139 - val_loss: 0.2621 - val_accuracy: 0.9124
Epoch 11/20
1213/1213 [==============================] - 3s 2ms/step - loss:
0.2426 - accuracy: 0.9157 - val_loss: 0.2375 - val_accuracy: 0.9212
Epoch 12/20
1213/1213 [==============================] - 4s 3ms/step - loss:
0.2363 - accuracy: 0.9173 - val_loss: 0.2418 - val_accuracy: 0.9238
Epoch 13/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2348 - accuracy: 0.9181 - val_loss: 0.2340 - val_accuracy: 0.9215
Epoch 14/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2292 - accuracy: 0.9193 - val_loss: 0.2467 - val_accuracy: 0.9148
Epoch 15/20
1213/1213 [==============================] - 2s 2ms/step - loss:
```

```
0.2243 - accuracy: 0.9216 - val_loss: 0.2453 - val_accuracy: 0.9201
Epoch 16/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2216 - accuracy: 0.9240 - val_loss: 0.2319 - val_accuracy: 0.9232
Epoch 17/20
1213/1213 [==============================] - 3s 3ms/step - loss:
0.2184 - accuracy: 0.9239 - val_loss: 0.2281 - val_accuracy: 0.9335
Epoch 18/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2147 - accuracy: 0.9260 - val_loss: 0.2299 - val_accuracy: 0.9222
Epoch 19/20
1213/1213 [==============================] - 3s 2ms/step - loss:
0.2131 - accuracy: 0.9249 - val_loss: 0.2236 - val_accuracy: 0.9309
Epoch 20/20
1213/1213 [==============================] - 2s 2ms/step - loss:
0.2095 - accuracy: 0.9283 - val_loss: 0.2327 - val_accuracy: 0.9233
```

```
dl_loss, dl_accuracy = dl_model.evaluate(X_test, y_test_encoded,
verbose=0)
print(f"Deep Learning Model Accuracy: {dl_accuracy:.2f}")
```

```
Deep Learning Model Accuracy: 0.92
```

```
y_pred_dl_prob = dl_model.predict(X_test)
y_pred_dl = y_pred_dl_prob.argmax(axis=1)
```

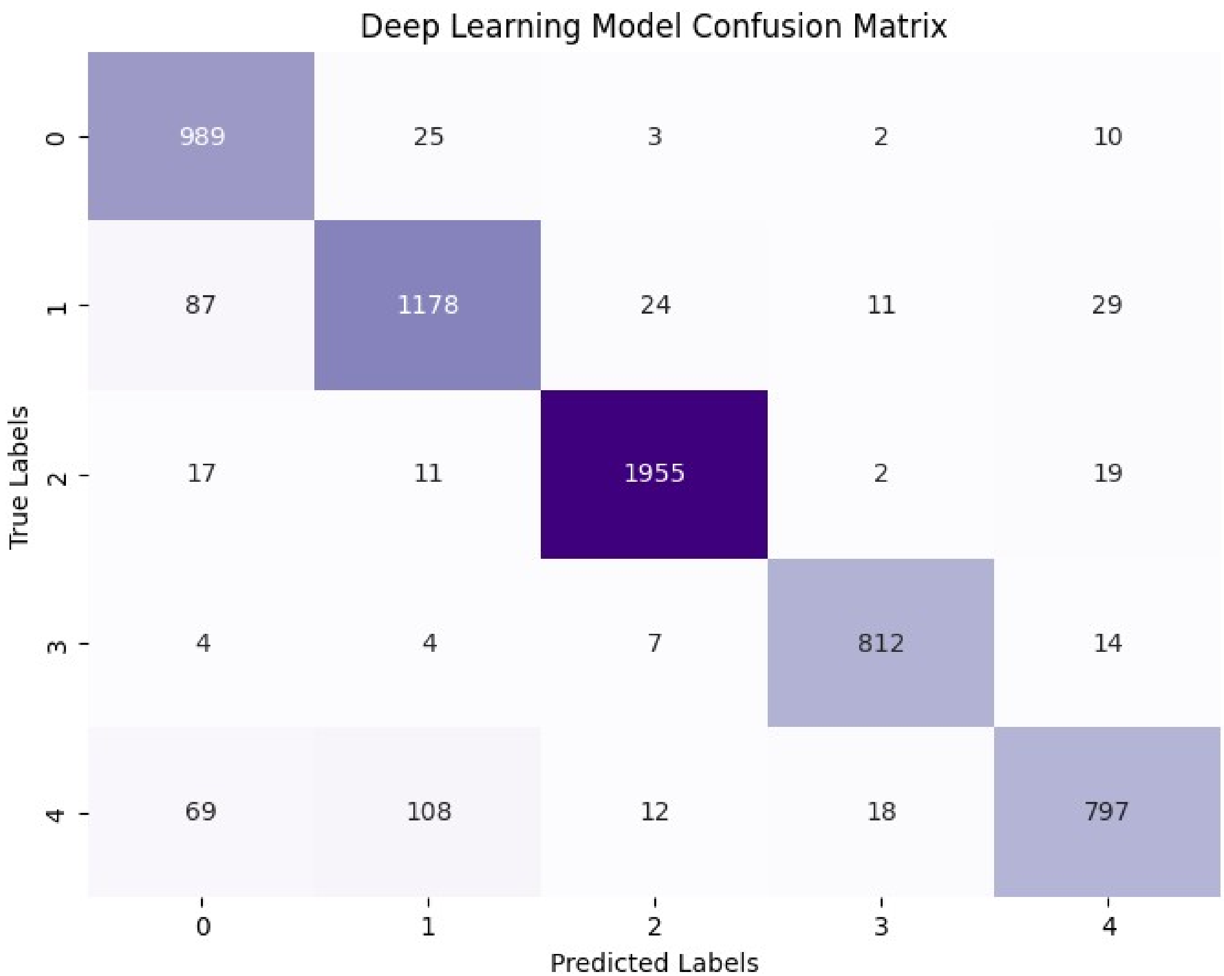
```
194/194 [==============================] - 0s 997us/step
```

```
print("Deep Learning Classification Report:\n",
classification_report(y_test, y_pred_dl))
```

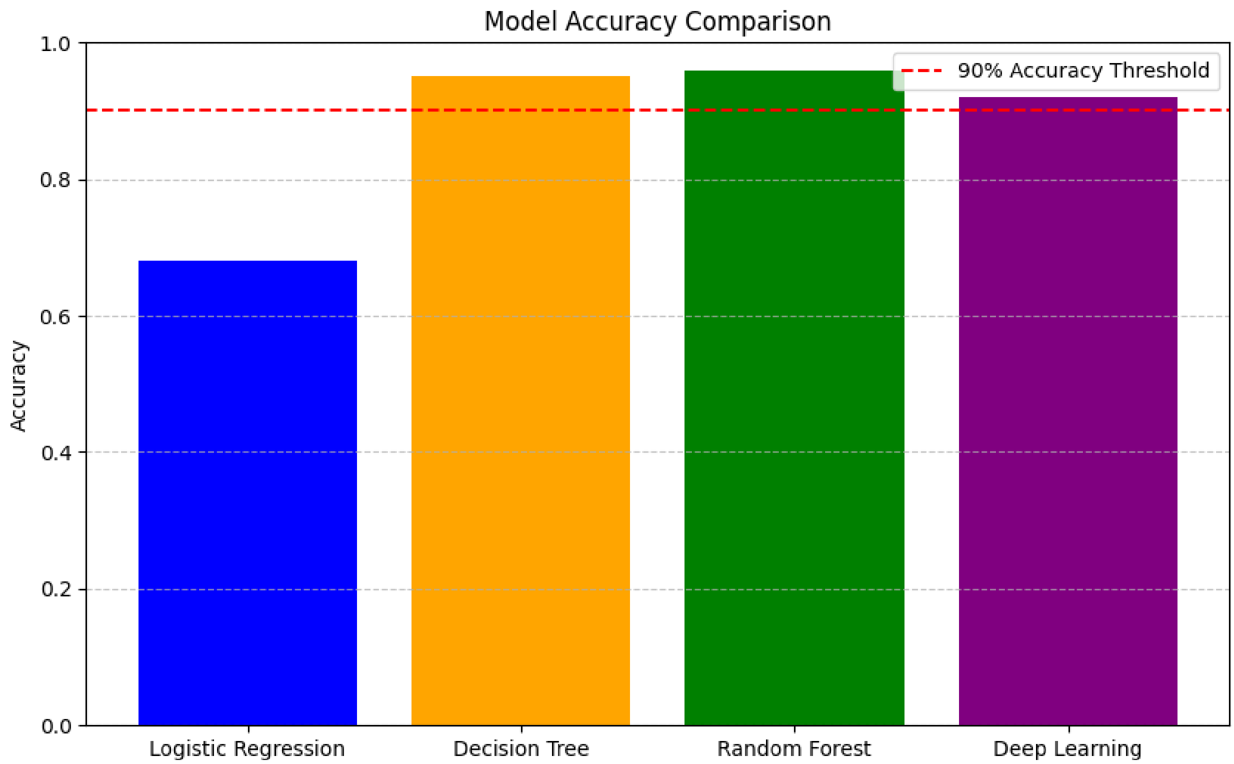
```
Deep Learning Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.96      0.90      1029
           1       0.89      0.89      0.89      1329
           2       0.98      0.98      0.98      2004
           3       0.96      0.97      0.96       841
           4       0.92      0.79      0.85      1004

    accuracy                           0.92      6207
   macro avg       0.92      0.92      0.92      6207
weighted avg       0.92      0.92      0.92      6207
```

```
cm_dl = confusion_matrix(y_test, y_pred_dl)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dl, annot=True, fmt='d', cmap='Purples', cbar=False)
plt.title('Deep Learning Model Confusion Matrix')
plt.xlabel('Predicted Labels')
```

```
plt.ylabel('True Labels')
plt.show()
```




```
models = ['Logistic Regression', 'Decision Tree', 'Random Forest',
'Deep Learning']
accuracies = [0.68, 0.95, 0.96, 0.92]

plt.figure(figsize=(10, 6))
plt.bar(models, accuracies, color=['blue', 'orange', 'green',
'purple'])
plt.ylim(0, 1)
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.axhline(y=0.9, color='red', linestyle='--', label='90% Accuracy
Threshold')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```


Model Accuracy Comparison
90% Accuracy Threshold
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
Decision Tree
Random Forest
Deep Learning

Thanks !!!