In [1]:
```
# Import Required Packages

import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import math
import numpy as np
from statsmodels.multivariate.manova import MANOVA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classifica
tion_report
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
# Feature Dict

feat_dict = {'X1':'grt_len_skull'
            ,'X2':'grt_hrz_brdth_skull'
            ,'X3':'ht_skull'
            ,'X4':'upper_face_ht'
            ,'X5':'face_brdth_chick_bones'}

class_dict = {1 : 'Group A', 2 : 'Group B'}
```

Read and explore skull data

In [3]:
```
skull = pd.read_csv('../Assignment/skull_data.csv')
skull.head()
```

Out[3]:

	X1	X2	X3	X4	X5	Groups
0	190.5	152.5	145.0	73.5	136.5	1
1	172.5	132.0	124.5	63.0	121.0	1
2	167.0	130.0	125.5	69.5	119.5	1
3	169.5	150.5	133.5	64.5	128.0	1
4	175.0	138.5	126.0	77.5	135.5	1

In [4]:
```
# Groupwise Descriptive Statistics

skull_g1 = skull[skull['Groups'] == 1]
skull_g2 = skull[skull['Groups'] == 2]
```

In [5]:
```
skull_g1.iloc[:,0:5].describe()
```

Out[5]:

	X1	X2	X3	X4	X5
count	17.000000	17.000000	17.000000	17.000000	17.000000
mean	174.823529	139.352941	131.941176	69.823529	130.352941
std	6.747549	7.602970	6.079891	4.575550	8.137039
min	162.500000	126.500000	121.500000	62.000000	118.500000
25%	170.000000	135.000000	127.500000	66.000000	124.000000
50%	173.500000	139.000000	132.000000	70.500000	132.500000
75%	179.500000	142.500000	134.500000	73.500000	134.500000
max	190.500000	152.500000	145.000000	77.500000	146.500000

In [6]:
```
skull_g2.iloc[:,0:5].describe()
```

Out[6]:

	X1	X2	X3	X4	X5
count	15.000000	15.000000	15.000000	15.000000	15.00000
mean	185.733333	138.733333	134.766667	76.466667	137.50000
std	8.626924	6.111659	6.026331	3.911826	4.23843
min	173.500000	130.000000	123.500000	68.500000	131.50000
25%	180.250000	134.750000	131.250000	74.000000	135.00000
50%	184.500000	139.500000	135.000000	76.500000	136.50000
75%	193.250000	142.250000	139.250000	79.250000	140.25000
max	200.000000	153.000000	143.500000	82.500000	146.00000

Observations:

- Difference in mean of X2 for both the groups is very less.

Check for multicolinearity

Pooled Within group Correlation Matrix

In [7]:
```
corr = skull.iloc[:,0:5].corr()
corr
```

Out[7]:

	X1	X2	X3	X4	X5
X1	1.000000	0.108943	0.429852	0.754691	0.566653
X2	0.108943	1.000000	0.019795	0.085173	0.548589
X3	0.429852	0.019795	1.000000	0.294522	0.207615
X4	0.754691	0.085173	0.294522	1.000000	0.617267
X5	0.566653	0.548589	0.207615	0.617267	1.000000

In [8]:
```
corr.style.background_gradient()
```

Out[8]:

	X1	X2	X3	X4	X5
X1	1	0.108943	0.429852	0.754691	0.566653
X2	0.108943	1	0.0197949	0.0851727	0.548589
X3	0.429852	0.0197949	1	0.294522	0.207615
X4	0.754691	0.0851727	0.294522	1	0.617267
X5	0.566653	0.548589	0.207615	0.617267	1

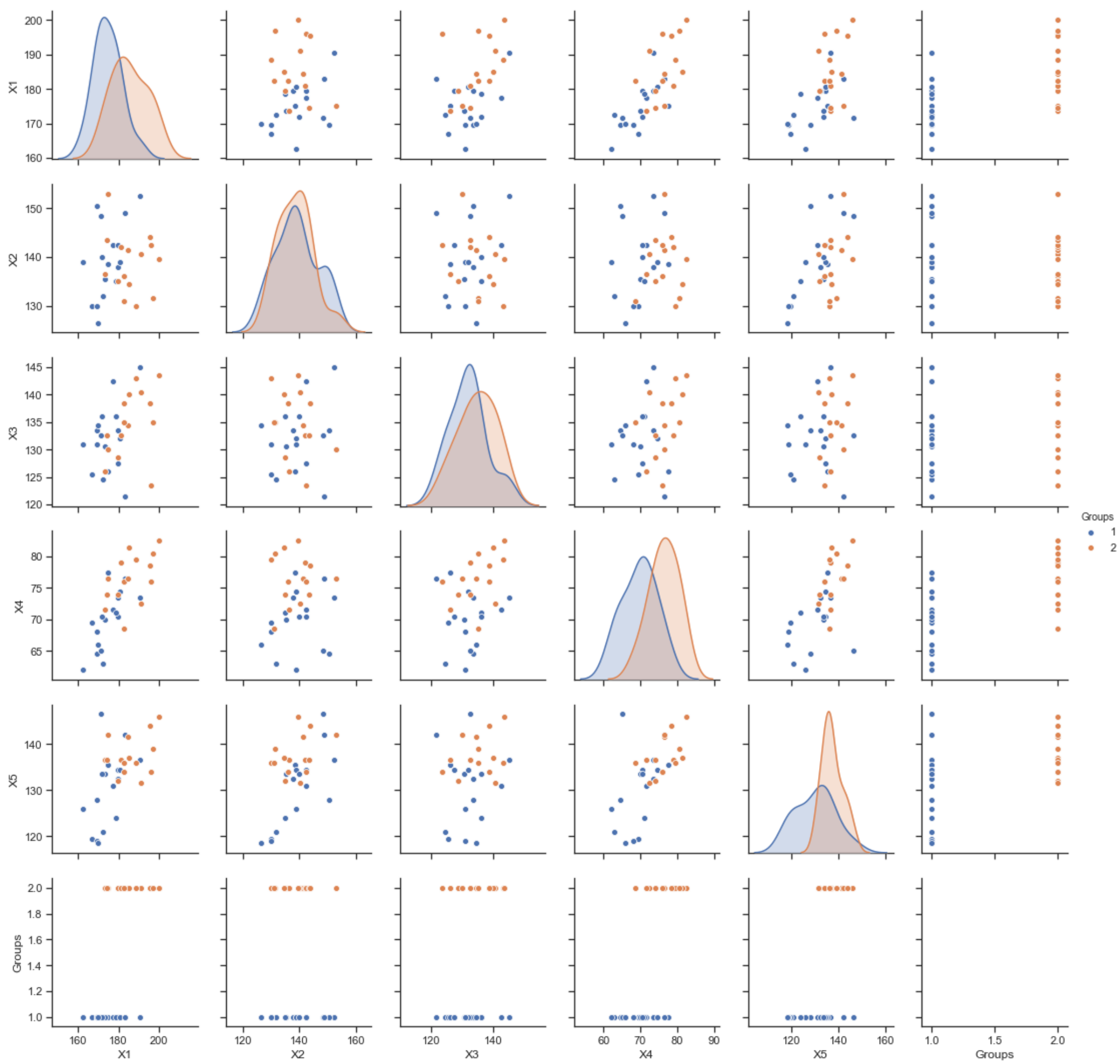
Observation

Here there is a larger correlation (0.754691) between X1 and X4. Hence there could be a problem of multi colinearity.

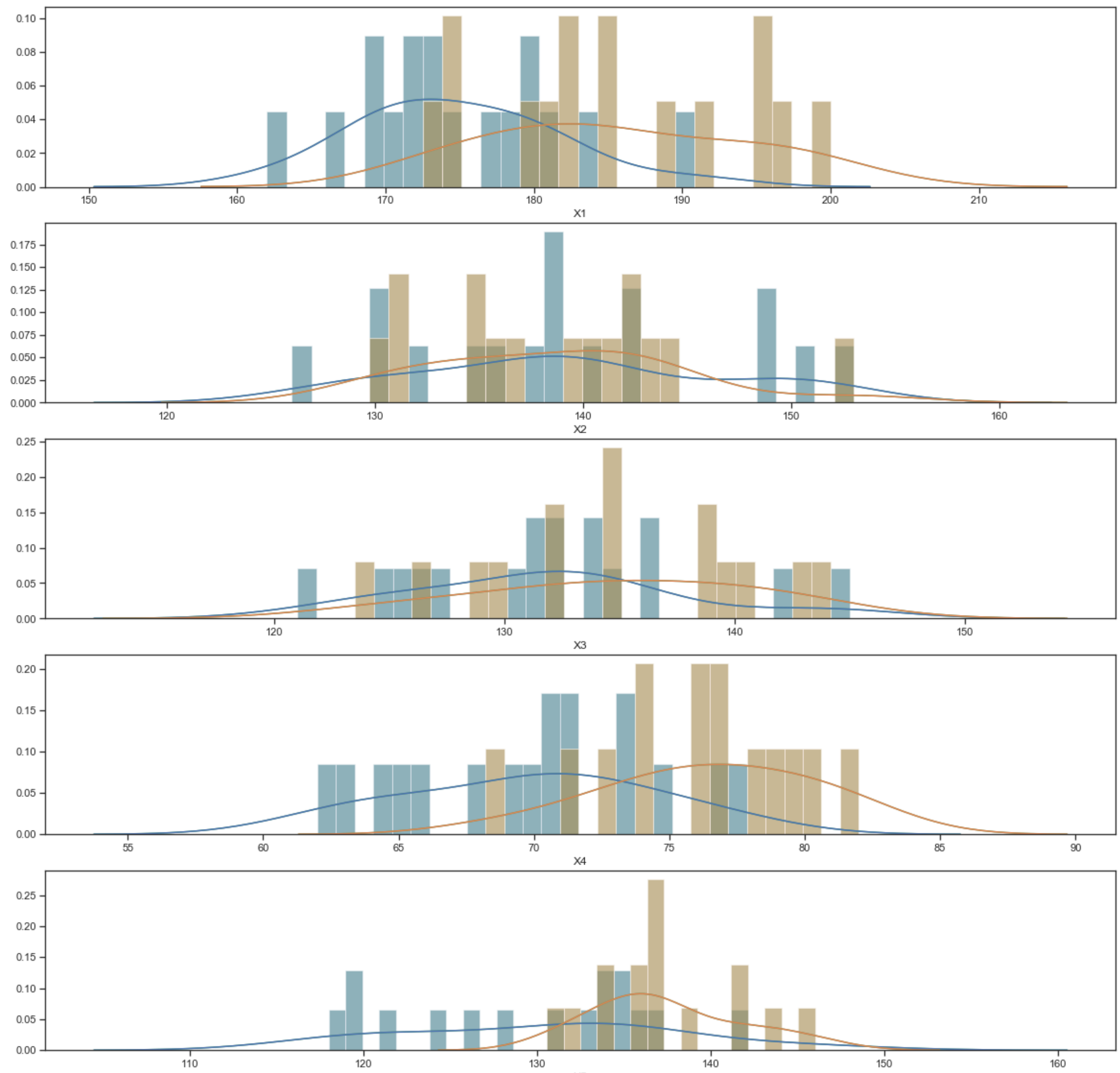
Scatterplot Matrix for the features

In [9]:
```
sns.set(style='ticks')
sns.pairplot(skull, hue="Groups");
```

```
/Applications/anaconda3/lib/python3.7/site-packages/statsmodels/nonpara
metric/kde.py:488: RuntimeWarning: invalid value encountered in true_di
vide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/Applications/anaconda3/lib/python3.7/site-packages/statsmodels/nonpara
metric/kdetools.py:34: RuntimeWarning: invalid value encountered in dou
ble_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```



In [10]:
```
fig, ax = plt.subplots(ncols=1, nrows=5, figsize=(20,20))

for col in range(1,6):
    x_min = math.floor(np.min(skull['X'+str(col)]))
    x_max = math.floor(np.max(skull['X'+str(col)]))
    num_bins = np.linspace(x_min,x_max,30)
    for grp in range(1,3):
        sns.distplot(skull[skull.Groups == grp]['X'+str(col)], ax=ax[col
-1], color='g', bins=num_bins)
        sns.distplot(skull[skull.Groups == grp]['X'+str(col)], ax=ax[col
-1], bins=num_bins)
```



Obersvations:

- From the scatter plot matrix, X1 has some correlation with X4 & X5.
- From the histogram plots, X5 seems to distinguish between the groups better than the other features.

LDA Analysis

Assumptions:

1. Covariance matrices of the two groups are equivalent.
2. Attributes follow a normal distribution

[1] Eigen Decomposition

[Step-1] Calculate the Feature Means for Every Class ==> Class wise mean vector

In [11]:
```
np.set_printoptions(precision=4)

X = skull[['X1','X2','X3','X4','X5']].values
y = skull['Groups'].values

grp_mean_vector = []

for grp in range(1,3):
    grp_mean_vector.append(np.mean(X[y==grp], axis=0))

    print(f"Group Mean Vector for Group {grp} is : {grp_mean_vector[grp-
1]}")

print('-'*40)
print('Overall Mean Vector : ')
print('-'*40)
print(grp_mean_vector)
print('-'*40)
```

```
Group Mean Vector for Group 1 is : [174.8235 139.3529 131.9412  69.8235
130.3529]
Group Mean Vector for Group 2 is : [185.7333 138.7333 134.7667  76.4667
137.5   ]
----------------------------------------
Overall Mean Vector : 
----------------------------------------
[array([174.8235, 139.3529, 131.9412,  69.8235, 130.3529]), array([185.
7333, 138.7333, 134.7667,  76.4667, 137.5   ])]
----------------------------------------
```

[Step-2] Get the Within Group and Between group Scatter Matrices

In [12]:
```
# Within class scatter matrix

S_W = np.zeros((5,5))

for grp,mv in zip(range(1,3), grp_mean_vector):

    grp_scatter_matrix = np.zeros((5,5))
    for row in X[y==grp]:

        row, mv = row.reshape((5,1)), mv.reshape((5,1))

        grp_scatter_matrix += (row-mv).dot((row-mv).T)

    S_W += grp_scatter_matrix

print(f"Within group Scatter Matrix is: \n {S_W}")
```

```
Within group Scatter Matrix is: 
 [[1770.4039  270.2422  518.8902  603.5873  603.3088]
 [ 270.2422 1447.8157   39.6696  130.1755  901.3824]
 [ 518.8902   39.6696 1099.8745  151.9569  132.6029]
 [ 603.5873  130.1755  151.9569  549.2039  389.5588]
 [ 603.3088  901.3824  132.6029  389.5588 1310.8824]]
```

In [13]:
```
# Between Class Scatter Matrix

S_B = np.zeros((5,5))

overall_mean = np.mean(X,axis=0)
print(overall_mean)

for i,mean_vec in enumerate(grp_mean_vector):
    n = X[y==i+1,:].shape[0]    # Number of elements in each class
    mean_vec = mean_vec.reshape(5,1) # make column vector
    overall_mean = overall_mean.reshape(5,1) # make column vector
    S_B += n * (mean_vec - overall_mean).dot((mean_vec - overall_mean).T
)

print(f'Between-Class Scatter Matrix:\n{S_B}')
```

```
[179.9375 139.0625 133.2656  72.9375 133.7031]
Between-Class Scatter Matrix:
[[948.4711 -53.8672 245.6411 577.5377 621.3474]
 [-53.8672   3.0593 -13.9509 -32.8005 -35.2886]
 [245.6411 -13.9509  63.6177 149.5744 160.9205]
 [577.5377 -32.8005 149.5744 351.6711 378.3474]
 [621.3474 -35.2886 160.9205 378.3474 407.0473]]
```

[Step-3] Get the linear discrimanants solving for Eignvector and eigen value of (INV(S_W).dot(S_B))

In [14]:
```
eig_val, eig_vec = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))

for i in range(len(eig_val)):

    eigvec_sc = eig_vec[:,i].reshape(5,1)
    print(f'\nEigenvector {i+1}: \n{eigvec_sc.real}')
    print(f'Eigenvalue {i}: {eig_val[i].real}')
```

```
Eigenvector 1:
[[-0.6172]
 [-0.415 ]
 [-0.0142]
 [ 0.4723]
 [ 0.4728]]
Eigenvalue 0: 0.0

Eigenvector 2:
[[ 0.2895]
 [-0.5049]
 [-0.0173]
 [ 0.5746]
 [ 0.5751]]
Eigenvalue 1: 0.9300824948618953

Eigenvector 3:
[[ 0.3735]
 [ 0.7871]
 [-0.0095]
 [-0.3224]
 [-0.1985]]
Eigenvalue 2: -1.6432123948888883e-17

Eigenvector 4:
[[ 0.3735]
 [ 0.7871]
 [-0.0095]
 [-0.3224]
 [-0.1985]]
Eigenvalue 3: -1.6432123948888883e-17

Eigenvector 5:
[[-0.6309]
 [-0.1524]
 [ 0.0429]
 [ 0.2223]
 [ 0.7262]]
Eigenvalue 4: 3.394097089567619e-17
```

[Step-4] Selecting the linear discriinants of new feature space

In [15]:
```
# Sort the Eigen Values in desceding order

eig_val
```

Out[15]:
```
array([ 0.0000e+00+0.0000e+00j,  9.3008e-01+0.0000e+00j,
       -1.6432e-17+2.7685e-17j, -1.6432e-17-2.7685e-17j,
        3.3941e-17+0.0000e+00j])
```

In [16]:
```
# Eigen Value and Eigen Vector pairs

eig_val_vec_pair = [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range(le
n(eig_val))]
eig_val_vec_pair
```

Out[16]:
```
[(0.0,
  array([-0.6172+0.j, -0.415 +0.j, -0.0142+0.j,  0.4723+0.j,  0.4728+0.
j])),
 (0.9300824948618953,
  array([ 0.2895+0.j, -0.5049+0.j, -0.0173+0.j,  0.5746+0.j,  0.5751+0.
j])),
 (3.219456881750208e-17,
  array([ 0.3735-0.1928j,  0.7871+0.j    , -0.0095+0.0516j, -0.3224+0.2
334j,
         -0.1985+0.0569j])),
 (3.219456881750208e-17,
  array([ 0.3735+0.1928j,  0.7871-0.j    , -0.0095-0.0516j, -0.3224-0.2
334j,
         -0.1985-0.0569j])),
 (3.394097089567619e-17,
  array([-0.6309+0.j, -0.1524+0.j,  0.0429+0.j,  0.2223+0.j,  0.7262+0.
j]))]
```

In [17]:
```
# Sort in descending order

eig_val_vec_pair = sorted(eig_val_vec_pair, reverse=True, key=lambda x :
x[0])
eig_val_vec_pair
```

Out[17]:
```
[(0.9300824948618953,
  array([ 0.2895+0.j, -0.5049+0.j, -0.0173+0.j,  0.5746+0.j,  0.5751+0.
j])),
 (3.394097089567619e-17,
  array([-0.6309+0.j, -0.1524+0.j,  0.0429+0.j,  0.2223+0.j,  0.7262+0.
j])),
 (3.219456881750208e-17,
  array([ 0.3735-0.1928j,  0.7871+0.j    , -0.0095+0.0516j, -0.3224+0.2
334j,
         -0.1985+0.0569j])),
 (3.219456881750208e-17,
  array([ 0.3735+0.1928j,  0.7871-0.j    , -0.0095-0.0516j, -0.3224-0.2
334j,
         -0.1985-0.0569j])),
 (0.0,
  array([-0.6172+0.j, -0.415 +0.j, -0.0142+0.j,  0.4723+0.j,  0.4728+0.
j]))]
```

In [18]:
```
# Variance Explained

print('Variance explained:\n')
eig_val_sum = sum(eig_val)
print(f'Total Variance : {eig_val_sum.real}')
for i,j in enumerate(eig_val_vec_pair):
    print('eigenvalue {0:}: {1:.2%}'.format(i+1, (j[0]/eig_val_sum).real
))
```

```
Variance explained:

Total Variance : 0.9300824948618953
eigenvalue 1: 100.00%
eigenvalue 2: 0.00%
eigenvalue 3: 0.00%
eigenvalue 4: 0.00%
eigenvalue 5: 0.00%
```

Note:

- First eigen value explains 100% variance

[Step-5] Choosing eigen vectors with largest eigen value

In [19]:
```
# Eigen vector matrix (Dimension : 1 X 5) # 1 -> One eigen value selecte
d above

W = eig_val_vec_pair[0][1].real
W
```

Out[19]:
```
array([ 0.2895, -0.5049, -0.0173,  0.5746,  0.5751])
```

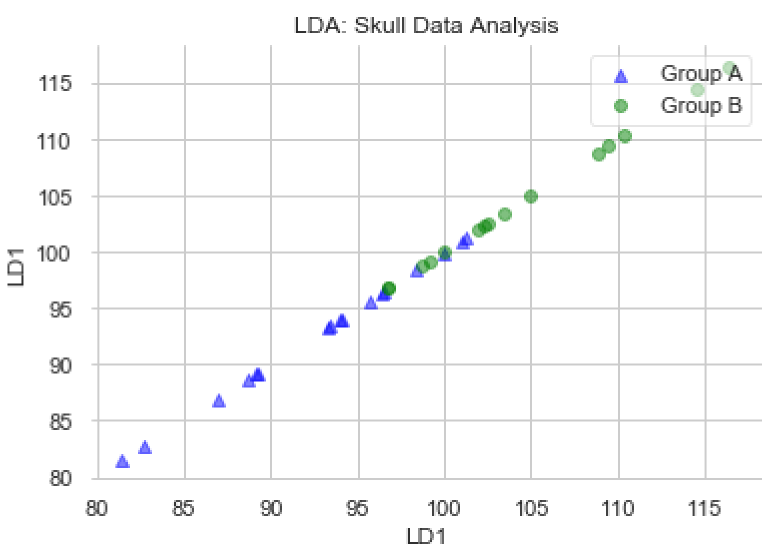
[Step-5] Transforming Data into new space

- Equation : Y = **X** X **W**

In [20]:
```
Y = X.dot(W)
Y
```

Out[20]:
```
array([ 96.3854,  86.9313,  89.2039,  81.4543, 101.0185,  93.4018,
        95.6783,  98.4203,  96.5613,  82.6906,  93.2772,  93.9875,
        99.9557, 101.2744,  88.6831,  94.0473,  89.0978, 102.5115,
       100.0212,  99.2166, 104.9833, 102.3131,  98.7215, 110.3616,
        96.792 , 103.397 , 116.3607, 108.8525,  96.8009, 109.4235,
       114.5032, 101.937 ])
```

In [21]:
```
def plot_step_lda():

    ax = plt.subplot(111)
    for label,marker,color in zip(
        range(1,3),('^','o'),('blue','green')):

        plt.scatter(x=Y[:][y == label],
                    y=Y[:][y == label],
                marker=marker,
                color=color,
                alpha=0.5,
                label=class_dict[label]
                )

    plt.xlabel('LD1')
    plt.ylabel('LD1')

    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title('LDA: Skull Data Analysis')

    # hide axis ticks
    plt.tick_params(axis="both", which="both", bottom=False, top=False,
            labelbottom=True, left=False, right=False, labelleft=True)

    # remove axis spines
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    plt.grid()
    plt.tight_layout
    plt.show()

plot_step_lda()
```



Testing for Significance of means of the features between groups (Wilk's Lambda Test)

In [22]:
```
maov = MANOVA.from_formula('X1 + X2 + X3 + X4 +X5 ~ Groups', data=skull)
```

In [23]:
```
print(maov.mv_test())
```

```
                   Multivariate linear model
===============================================================

---------------------------------------------------------------
       Intercept         Value   Num DF  Den DF  F Value  Pr > F
---------------------------------------------------------------
          Wilks' lambda   0.0093 5.0000 26.0000 554.4209 0.0000
         Pillai's trace   0.9907 5.0000 26.0000 554.4209 0.0000
 Hotelling-Lawley trace 106.6194 5.0000 26.0000 554.4209 0.0000
    Roy's greatest root 106.6194 5.0000 26.0000 554.4209 0.0000
---------------------------------------------------------------

---------------------------------------------------------------
            Groups         Value  Num DF  Den DF F Value Pr > F
---------------------------------------------------------------
             Wilks' lambda 0.5181 5.0000 26.0000  4.8364 0.0029
            Pillai's trace 0.4819 5.0000 26.0000  4.8364 0.0029
    Hotelling-Lawley trace 0.9301 5.0000 26.0000  4.8364 0.0029
       Roy's greatest root 0.9301 5.0000 26.0000  4.8364 0.0029
===============================================================
```

Testing if the LDA component is able to classify properly

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(Y, skull.Groups, ran
dom_state=1)
```

In [25]:
```
skull_dt = DecisionTreeClassifier()
skull_dt.fit(X_train.reshape(-1,1), y_train)
y_pred = skull_dt.predict(X_test.reshape(-1,1))
confusion_matrix(y_test, y_pred)
```

Out[25]:
```
array([[1, 0],
       [2, 5]])
```

In [26]:
```
print(f'Classification Accuracy : {accuracy_score(y_test,y_pred)}')
```

```
Classification Accuracy : 0.75
```

In [27]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.33      1.00      0.50         1
           2       1.00      0.71      0.83         7

   micro avg       0.75      0.75      0.75         8
   macro avg       0.67      0.86      0.67         8
weighted avg       0.92      0.75      0.79         8
```

[2] Discriminant Function Estimation

In [28]:
```
lda = LDA(solver='eigen')
lda.fit(skull.iloc[:,0:5], skull.Groups)
```

Out[28]:
```
LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=No
ne,
              solver='eigen', store_covariance=False, tol=0.0001)
```

In [29]:
```
lda.explained_variance_ratio_
```

Out[29]:
```
array([1.])
```

In [30]:
```
lda.coef_
```

Out[30]:
```
array([[ 3.2859, -5.7308, -0.1961,  6.5221,  6.5277]])
```

Observation

---

1. X1, X4, X5 seems to contribute more than other two features for the LDA classification.

Discriminant Function

In [31]:
```
coeff = lda.coef_

skull['score'] = coeff.flat[0]*skull['X1'] + coeff.flat[1]*skull['X2'] +
coeff.flat[2]*skull['X3'] + coeff.flat[3]*skull['X4'] + coeff.flat[4]*sk
ull['X5']

skull.head()
```

Out[31]:

	X1	X2	X3	X4	X5	Groups	score
0	190.5	152.5	145.0	73.5	136.5	1	1093.990799
1	172.5	132.0	124.5	63.0	121.0	1	986.684153
2	167.0	130.0	125.5	69.5	119.5	1	1012.479321
3	169.5	150.5	133.5	64.5	128.0	1	924.519605
4	175.0	138.5	126.0	77.5	135.5	1	1146.577251

In [32]:
```
# Cut off score

Z1 = skull[skull.Groups == 1]['score'].mean()
Z2 = skull[skull.Groups == 2]['score'].mean()

print(f'Mean Score for Group 1 : {Z1}\nMean Score for Group 2 : {Z2}')
```

```
Mean Score for Group 1 : 1056.2789479135024
Mean Score for Group 2 : 1185.1052120039014
```

In [33]:
```
# Number of elements in each group

n1 = skull[skull.Groups == 1].shape[0]
n2 = skull[skull.Groups == 2].shape[0]

print(f'Number of elements in Group 1 : {n1}\nNumber of elements in Grou
p 2 : {n2}')
```

```
Number of elements in Group 1 : 17
Number of elements in Group 2 : 15
```

In [34]:
```
# Cutoff Score

Z = ((n2*Z1)+(n1*Z2))/(n1+n2)

print(f'Cutoff Score : {Z}')
```

```
Cutoff Score : 1124.717900711527
```