CLASSIFICATION - HOUSE GRADE DATA

Build a predictive model to determine the Grade of house.

Importing libraries

In [1]:
```
#IMPORT REQUIRED LIBRARIES
import numpy as np
import pandas as pd
from numpy import mean
from numpy import std

import warnings
warnings.simplefilter(action='ignore')
```

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from matplotlib.colors import ListedColormap
from sklearn.metrics import precision_score, recall_score, accuracy_score ,f1_scor
```

In [3]:
```
#LOAD DATASET
house_df = pd.read_csv('DS3_C6_S2_Classification_HouseGrade_Data_Project.csv')
house_df.head(17)
```

Out[3]:

	Id	Area(total)	Trooms	Nbedrooms	Nbwashrooms	Twashrooms	roof	Roof(Area)	Lawn(Area)
0	1	339	6	5	4	6	NO	0	76
1	2	358	5	4	3	4	YES	71	96
2	3	324	7	5	4	5	YES	101	117
3	4	330	6	4	3	5	YES	101	82
4	5	320	7	4	4	5	NO	0	75
5	6	314	8	7	6	7	YES	81	93
6	7	332	9	8	7	9	YES	103	120
7	8	323	9	8	7	9	NO	0	95
8	9	351	8	6	6	8	YES	89	97
9	10	339	6	5	5	6	NO	0	111
10	11	308	5	3	2	3	YES	74	105
11	12	309	6	4	4	6	NO	0	115
12	13	324	6	5	5	7	NO	0	109
13	14	303	5	3	2	3	NO	0	84
14	15	321	9	6	5	7	NO	0	80
15	16	345	8	7	6	7	YES	116	83
16	17	307	7	6	6	7	NO	0	81

Data preprocessing

In [4]:
```
house_df.sample(7)
```

Out[4]:

	Id	Area(total)	Trooms	Nbedrooms	Nbwashrooms	Twashrooms	roof	Roof(Area)	Lawn(Area)
1537	1538	305	7	5	5	6	YES	86	71
195	196	308	7	4	3	4	YES	79	119
2868	2869	334	6	5	5	7	YES	109	80
2162	2163	322	9	8	8	10	YES	108	78
2987	2988	303	9	8	7	9	YES	97	85
2655	2656	293	5	3	3	5	YES	105	95
1757	1758	291	8	7	6	7	YES	75	98

In [5]:
```
house_df.shape
```

Out[5]:
```
(3000, 14)
```

In [6]:
```
house_df.dtypes
```

Out[6]:
```
Id                 int64
Area(total)        int64
Trooms             int64
Nbedrooms          int64
Nbwashrooms        int64
Twashrooms         int64
roof              object
Roof(Area)         int64
Lawn(Area)         int64
Nfloors            int64
API                int64
ANB                int64
Expected price     int64
Grade             object
dtype: object
```

In [7]:
```
house_df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Id	3000.0	1500.500000	866.169729	1.0	750.75	1500.5	2250.25	3000.0
Area(total)	3000.0	325.117000	20.507742	290.0	308.00	325.0	343.00	360.0
Trooms	3000.0	7.021667	1.421221	5.0	6.00	7.0	8.00	9.0
Nbedrooms	3000.0	5.023000	1.634838	2.0	4.00	5.0	6.00	8.0
Nbwashrooms	3000.0	4.513667	1.715263	1.0	3.00	4.0	6.00	8.0
Twashrooms	3000.0	6.010667	1.786136	2.0	5.00	6.0	7.00	10.0
Roof(Area)	3000.0	48.980667	48.746641	0.0	0.00	71.0	96.00	120.0
Lawn(Area)	3000.0	95.609333	14.837388	70.0	83.00	96.0	109.00	120.0
Nfloors	3000.0	4.013333	1.621532	1.0	3.00	4.0	5.00	7.0
API	3000.0	70.190667	17.563460	40.0	55.00	70.0	85.00	100.0
ANB	3000.0	3.479000	1.694260	1.0	2.00	4.0	5.00	6.0
Expected price	3000.0	3782.938333	567.189995	2504.0	3354.00	3771.0	4208.00	5216.0

In [8]:
```
# Checking for null value in each column
house_df.isnull().sum()
```

Out[8]:
```
Id                0
Area(total)       0
Trooms            0
Nbedrooms         0
Nbwashrooms       0
Twashrooms        0
roof              0
Roof(Area)        0
Lawn(Area)        0
Nfloors           0
API               0
ANB               0
Expected price    0
Grade             0
dtype: int64
```

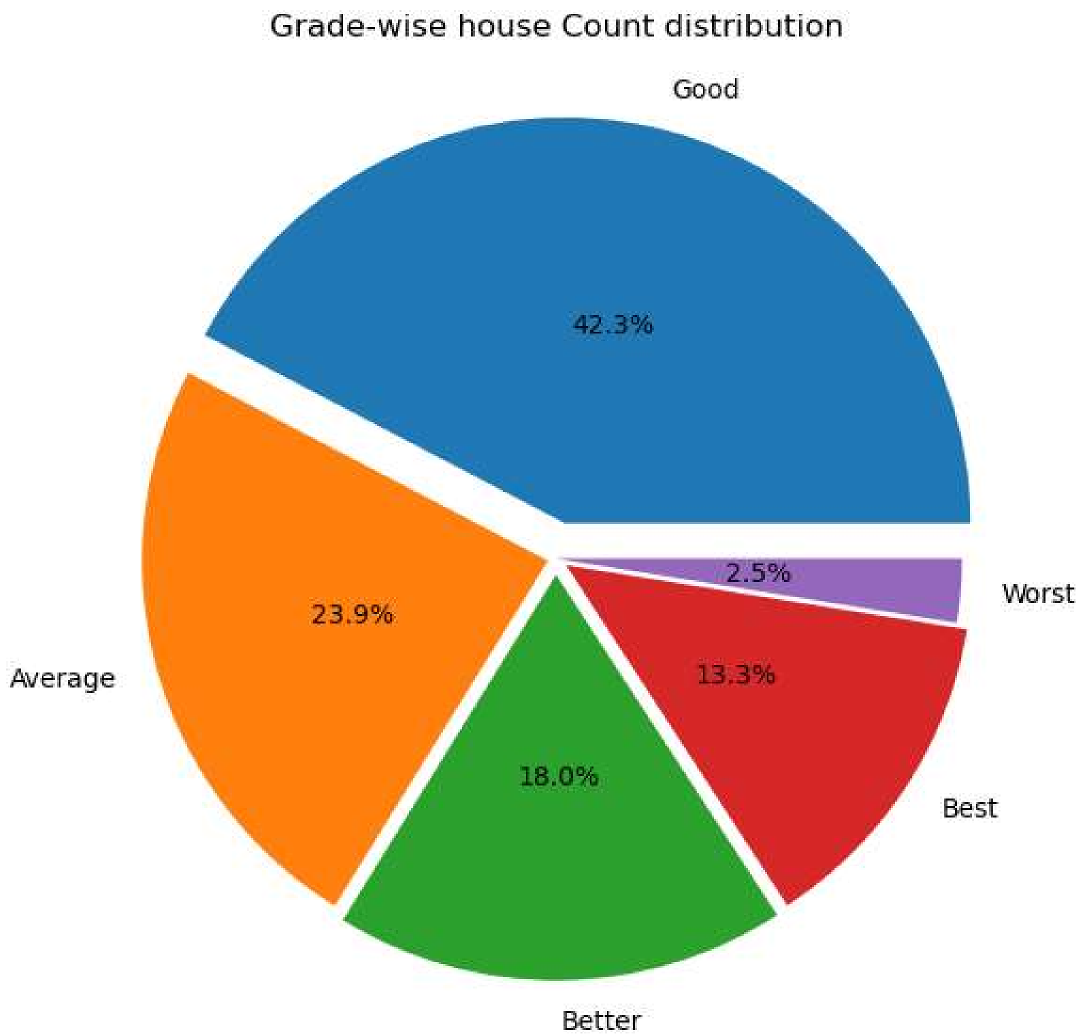
There are no null values in the House Grade Dataset

Data Visualization

In [9]:
```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

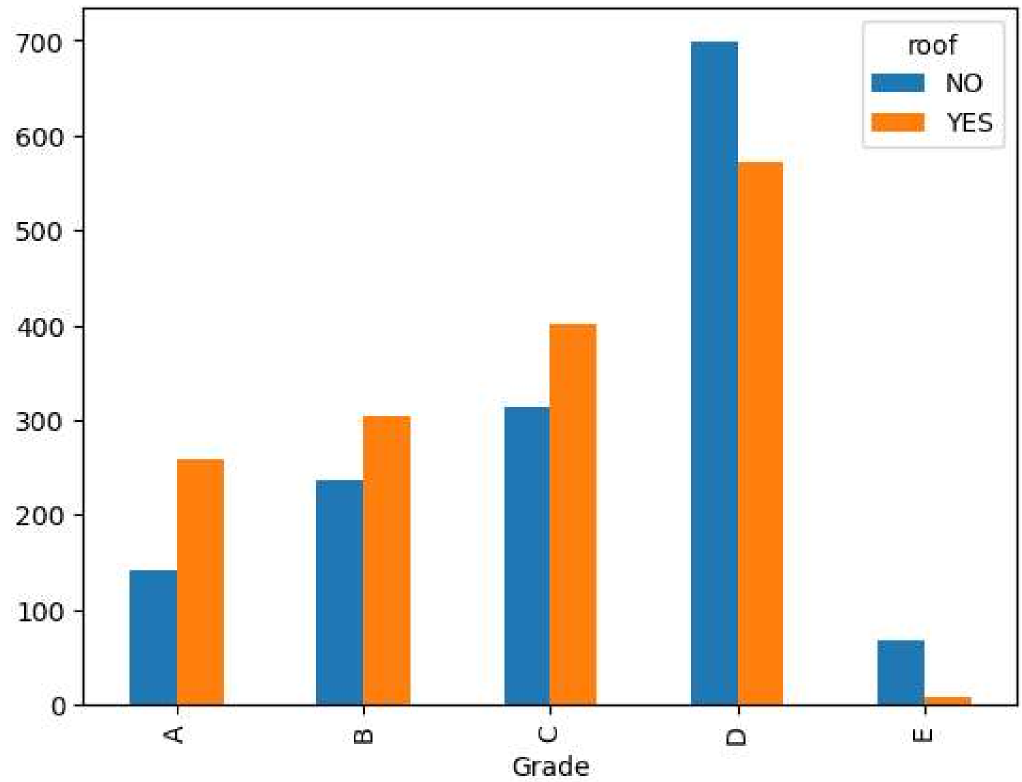
In [40]:

```
#Gradewise houses percentage
fig = plt.figure(figsize =(10, 7))

labels = ['Good','Average','Better','Best','Worst']
bins= [0,290 ,300,330 ,340 ,360]
#data = pd.cut(females["Grade"], labels = labels)
#data = data.value_counts()
data = pd.cut(house_df["Grade"],bins=bins, labels = labels)
data =house_df['Grade'].value_counts()

plt.pie( x = data, labels = labels,explode = [0.09, .02, 0.04, 0.03,0] , pctdistan
plt.title("Grade-wise house Count distribution")
plt.show()
```



Interpretation-Maximum number of house are of Good quality i.e, "C" Grade houses

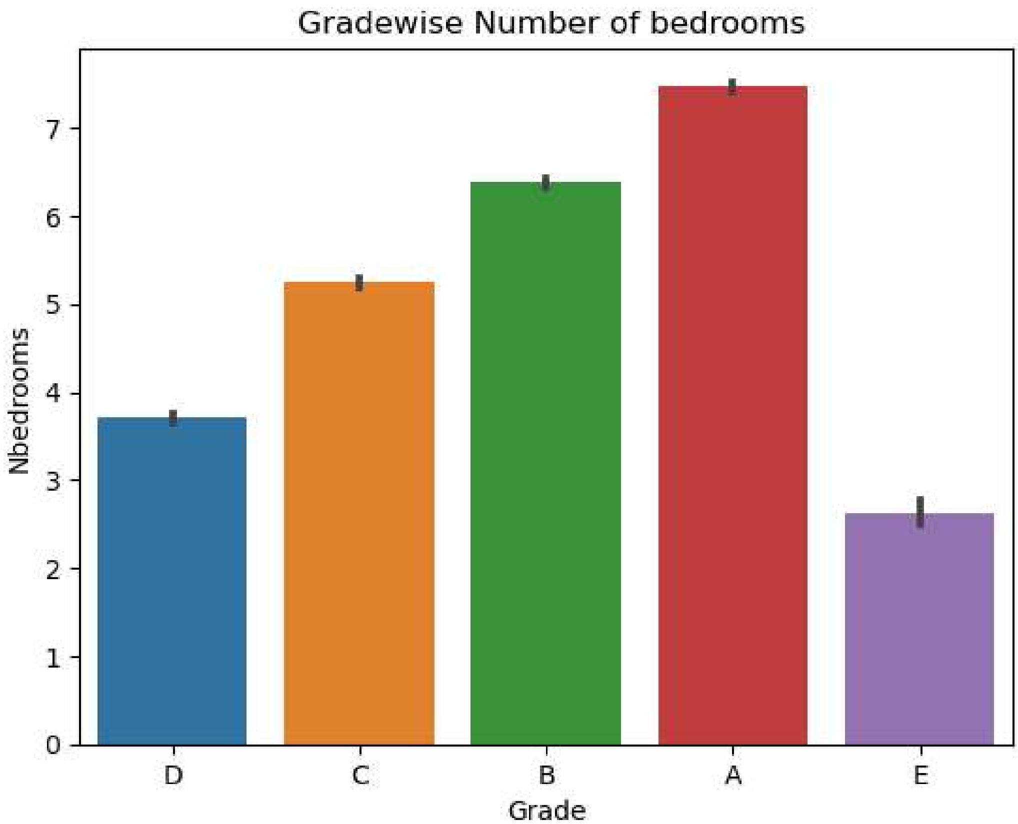
In [11]:
```
pd.crosstab(house_df['Grade'],house_df['roof']).plot(kind='bar');
```



Interpretation- Maximum Houses of Grade A ,B ,C have roofs whereas houses of Grade D and E don't have roofs

In [12]:
```
sns.barplot(house_df['Grade'] , house_df['Nbedrooms'])
plt.title(' Gradewise Number of bedrooms')
```

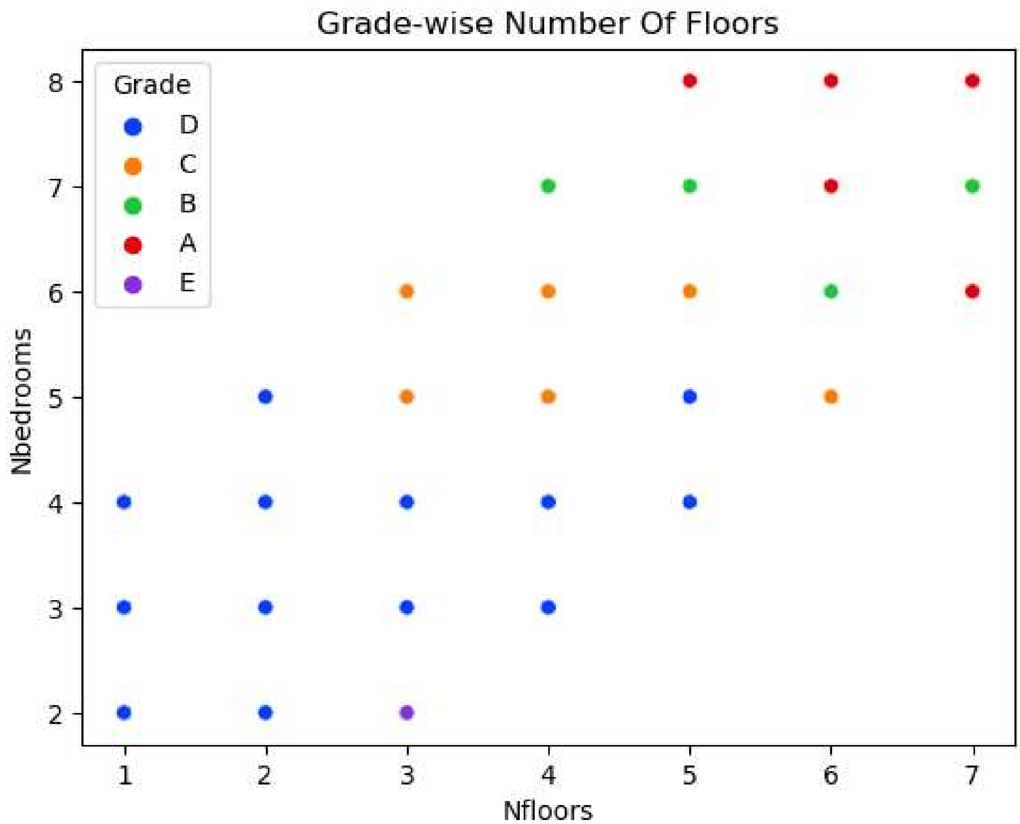
Out[12]:
```
Text(0.5, 1.0, ' Gradewise Number of bedrooms')
```



'A' grade houses have maximum number of rooms i.e,7 followed by 'B' Grade houses with 6 number of rooms

In [13]:

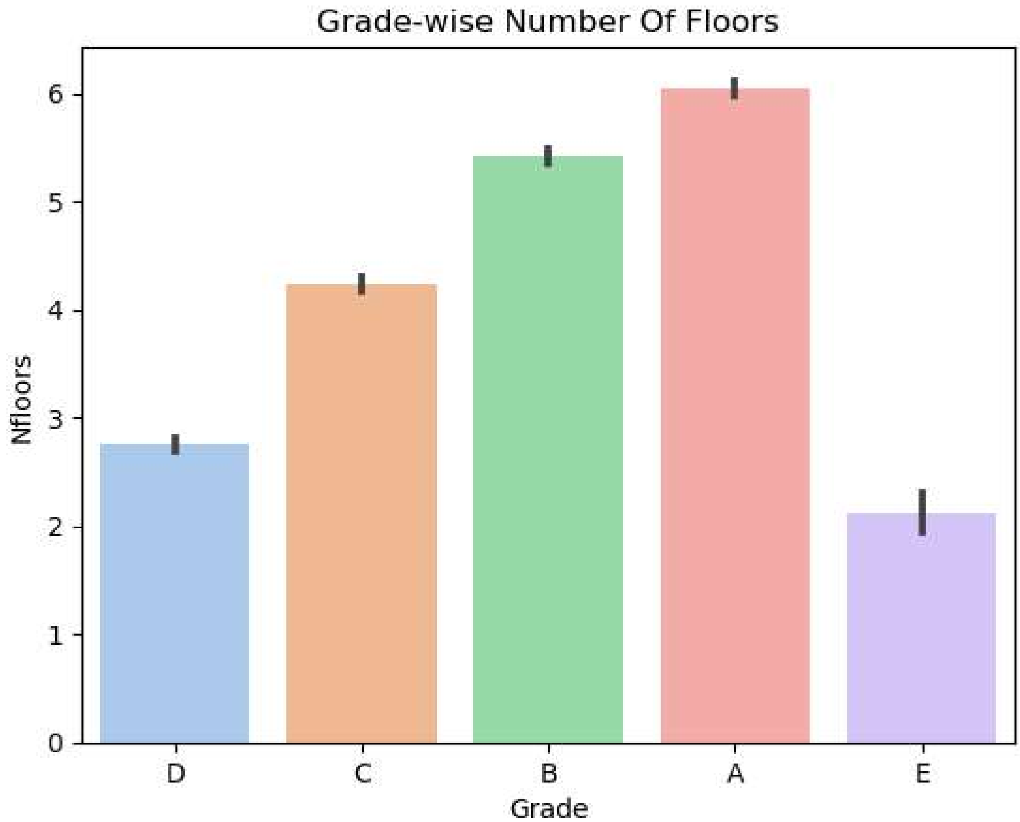
```
sns.scatterplot( y = "Nbedrooms",x = "Nfloors" , hue='Grade',data = house_df , pal
```



Interpretation- The Total Area wise and number of floor wise maximum houses are Good and minimum houses are of better quality .

In [14]:

```
sns.barplot(x ='Grade', y ='Nfloors', data = house_df, palette ='pastel').set(titl
```



Interpretation -'A' grade houses have maximum number of floors i.e,6 followed by 'B' Grade houses with 5 number of rooms

Label Encoding

In [15]:

```
#Label encoding
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()

obj = house_df.select_dtypes(include='object')
for i in obj:
    house_df[i]=le.fit_transform(house_df[i])
```

In [16]:
```
house_df.dtypes
```

Out[16]:
```
Id                int64
Area(total)       int64
Trooms            int64
Nbedrooms         int64
Nbwashrooms       int64
Twashrooms        int64
roof              int32
Roof(Area)        int64
Lawn(Area)        int64
Nfloors           int64
API               int64
ANB               int64
Expected price    int64
Grade             int32
dtype: object
```

In [17]:
```
house_df['Grade'].value_counts()
```

Out[17]:
```
3    1270
2     716
1     539
0     399
4      76
Name: Grade, dtype: int64
```

In Grade 'A' indicates are the best houses and 'E' indicates the worst houses

DATA SPLITTING

In [18]:
```
#2) Create a test-split with 30% test data and random state =11
from sklearn.model_selection import train_test_split

X = house_df.iloc[:, [1,3,5,6,7,8,9,10,12]].values
y = house_df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.80 ,rando

print('Size of training dataset: ', X_train.shape)
print('Size of test dataset: ', y_test.shape)
```

```
Size of training dataset:  (2400, 9)
Size of test dataset:  (600,)
```

In [19]:
```
#3) Normalizing and Standardizing using Standard Scalar

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

MODEL BUILDING

Naive Bayes Model

In [20]:
```
# Fitting Naive Bayes to the Training set
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[20]:
```
GaussianNB()
```

Evaluate the Model

In [21]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

```

Confusion Matrix

In [22]:
```
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[22]:
```
array([[ 72,  15,   0,   0,   0],
       [  1,  93,  12,   0,   0],
       [  0,  29,  81,  15,   0],
       [  0,   0,  28, 239,   5],
       [  0,   0,   0,   2,   8]], dtype=int64)
```

In [23]:
```
# Evaluate Accuracy Score
accuracy_score(y_test, y_pred)
```

Out[23]:
```
0.8216666666666667
```

Evaluation metrics

In [24]:
```
print('accuracy:', accuracy_score(y_test, y_pred))
print('recall:', recall_score(y_test, y_pred, average='weighted'))
print('f1-score:', f1_score(y_test, y_pred, average='weighted'))
print('precision:', precision_score(y_test, y_pred , average='weighted'))
```

```
accuracy: 0.8216666666666667
recall: 0.8216666666666667
f1-score: 0.8249197027023704
precision: 0.8358890927698394
```

Decision Tree Classifier

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

```

In [26]:
```
# define the model
model = DecisionTreeClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, erro
# report performance
print('Accuracy-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy-Score: 0.793 (0.022)
```

Evaluation metrics

In [27]:
```
print('accuracy:', accuracy_score(y_test, y_pred))
print('recall:', recall_score(y_test, y_pred, average='weighted'))
print('f1-score:', f1_score(y_test, y_pred, average='weighted'))
print('precision:', precision_score(y_test, y_pred , average='weighted'))
```

```
accuracy: 0.8216666666666667
recall: 0.8216666666666667
f1-score: 0.8249197027023704
precision: 0.8358890927698394
```

Evaluate the Model

In [28]:
```
# Predicting the Test set results
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Confusion Matrix

In [29]:
```
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[29]:
```
array([[ 72,  15,   0,   0,   0],
       [ 12,  82,  12,   0,   0],
       [  0,  18,  91,  16,   0],
       [  0,   0,  30, 241,   1],
       [  0,   0,   0,   4,   6]], dtype=int64)
```

Random Forest Classifier

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [31]:
```
# define the model
model2 = RandomForestClassifier()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
n_scores = cross_val_score(model2, X, y, scoring='accuracy', cv=cv, n_jobs=-1, err
# report performance
print('Accuracy-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy-Score: 0.860 (0.025)
```

In [32]:
```

print('accuracy:', accuracy_score(y_test, y_pred))
print('recall:', recall_score(y_test, y_pred, average='weighted'))
print('f1-score:', f1_score(y_test, y_pred, average='weighted'))
print('precision:', precision_score(y_test, y_pred , average='weighted'))
```

```
accuracy: 0.82
recall: 0.82
f1-score: 0.821891071775741
precision: 0.8256814495843139
```

Evaluate the Model

In [33]:
```
# Predicting the Test set results
model2.fit(X_train, y_train)
y_pred2 = model2.predict(X_test)
```

Confusion Matrix

In [34]:
```
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred2)
cm
```

Out[34]:
```
array([[ 76,  11,   0,   0,   0],
       [  5,  91,  10,   0,   0],
       [  0,  17,  91,  17,   0],
       [  0,   0,  10, 262,   0],
       [  0,   0,   0,   8,   2]], dtype=int64)
```

Adaboost Classifier

In [35]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [36]:

```
# define the model
model3 = AdaBoostClassifier()


# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
n_score = cross_val_score(model3, X, y, scoring='accuracy', cv=cv, n_jobs=-1, erro

# report performance
print('Accuracy-Score: %.3f (%.3f)' % (mean(n_score), std(n_score)))
```

```
Accuracy-Score: 0.688 (0.046)
```

Stacking

In [37]:

```
# required Python libraries
from sklearn.linear_model import LogisticRegression   #META MODEL
from sklearn.tree import DecisionTreeClassifier       #BASE MODEL
from sklearn.naive_bayes import GaussianNB            #BASE MODEL
from sklearn.neighbors import KNeighborsClassifier    #BASE MODEL
from sklearn.svm import SVC                           #BASE MODEL
from sklearn.ensemble import StackingClassifier       #BASE MODEL
```

In [38]:

```
#  get a stacking ensemble of models
def get_stacking():
    #BASE MODELS
    level0 = list()
    level0.append(('lr', LogisticRegression()))
    level0.append(('dt', DecisionTreeClassifier()))
    level0.append(('nb', GaussianNB()))
    level0.append(('knn', KNeighborsClassifier()))
    level0.append(('svm', SVC()))

    #META MODEL
    level1 = LogisticRegression()

    # ENSEMBLE STACKING
    model = StackingClassifier(estimators=level0 , final_estimator=level1, cv= 5)
    return model
```

In [39]:

```
# define the model
model = get_stacking()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
n_score = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_

# report performance
print('Accuracy-Score: %.3f (%.3f)' % (mean(n_score), std(n_score)))

```

```
Accuracy-Score: 0.838 (0.020)
```

Business Interpretation

- Maximum number of house are of average quality i.e, "C" Grade houses
- 'A' grade houses have maximum number of rooms i.e,7 followed by 'B' Grade houses with 6 number of rooms
- Maximum Houses of Grade A ,B ,C have roofs whereas houses of Grade D and E don't have roofs,
- Interpretation- The Total Area wise and number of floor wise maximum houses are Good and minimum houses are of better quality.
- 'A' grade houses have maximum number of floors i.e,6 followed by 'B' Grade houses with 5 number of rooms
- The Random forest Classifier Model performs the best for house grade dataset and gives the good accuracy score i.e, 86% among all the models (Naive Bayes Model ,Random forest Classifier Model,Decision Tree Classifier,Adaboost Classifier,Stacking)

In [ ]:

```

```