ta5y9cuw4

October 20, 2023

[3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
colNames = ["Rock_number","Category_number",
"Subtype_number","Token_number",
"Stimulus_Type","Frequency_responded_Igneous",
"Frequency_responded_Metamorphic","Frequency_responded_Sedimentary",
"Frequency_responded_Old","Frequency_responded_New",
"Proportion_responded_Igneous","Proportion_responded_Metamorphic",
"Proportion_responded_Sedimentary","Proportion_responded_Old"]

df = pd.read_excel("aggregateRockData-1.xlsx",
                   header=None,
                   names = colNames
)
```

[5]:
```
df
```

[5]:
```
     Rock_number  Category_number  Subtype_number  Token_number  \
0              1                1               1             1
1              2                1               1             2
2              3                1               1             3
3              4                1               1             4
4              5                1               1             5
..           …              …             …           …
535          716                3              28            18
536          717                3              29            17
537          718                3              29            18
538          719                3              30            17
539          720                3              30            18

     Stimulus_Type  Frequency_responded_Igneous  \
0                2                           57
1                3                           59
2                3                           37
3                3                           41
```

```
4                3                          42
..             …                          …
535              4                           1
536              4                           2
537              4                          25
538              4                          16
539              4                           6

     Frequency_responded_Metamorphic  Frequency_responded_Sedimentary  \
0                                 19                                6
1                                 10                               13
2                                 28                               17
3                                 11                               30
4                                  6                               34
..                               …                                …
535                                0                               81
536                               44                               36
537                                9                               48
538                               46                               20
539                               36                               40

     Frequency_responded_Old  Frequency_responded_New  \
0                         46                       36
1                         24                       58
2                         20                       62
3                         29                       53
4                         25                       57
..                       …                        …
535                       57                       25
536                       40                       42
537                       27                       55
538                       23                       59
539                       30                       52

     Proportion_responded_Igneous  Proportion_responded_Metamorphic  \
0                        0.695122                          0.231707
1                        0.719512                          0.121951
2                        0.451220                          0.341463
3                        0.500000                          0.134146
4                        0.512195                          0.073171
..                            …                             …
535                      0.012195                          0.000000
536                      0.024390                          0.536585
537                      0.304878                          0.109756
538                      0.195122                          0.560976
539                      0.073171                          0.439024
```

```
     Proportion_responded_Sedimentary  Proportion_responded_Old
0                            0.073171                  0.560976
1                            0.158537                  0.292683
2                            0.207317                  0.243902
3                            0.365854                  0.353659
4                            0.414634                  0.304878
..                                ...                       ...
535                          0.987805                  0.695122
536                          0.439024                  0.487805
537                          0.585366                  0.329268
538                          0.243902                  0.280488
539                          0.487805                  0.365854

[540 rows x 14 columns]
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 540 entries, 0 to 539
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Rock_number                      540 non-null    int64
 1   Category_number                  540 non-null    int64
 2   Subtype_number                   540 non-null    int64
 3   Token_number                     540 non-null    int64
 4   Stimulus_Type                    540 non-null    int64
 5   Frequency_responded_Igneous      540 non-null    int64
 6   Frequency_responded_Metamorphic  540 non-null    int64
 7   Frequency_responded_Sedimentary  540 non-null    int64
 8   Frequency_responded_Old          540 non-null    int64
 9   Frequency_responded_New          540 non-null    int64
 10  Proportion_responded_Igneous     540 non-null    float64
 11  Proportion_responded_Metamorphic 540 non-null    float64
 12  Proportion_responded_Sedimentary 540 non-null    float64
 13  Proportion_responded_Old         540 non-null    float64
dtypes: float64(4), int64(10)
memory usage: 59.2 KB
```

[7]:
```
df_norm540 = pd.read_csv("norm540.txt",
                         sep="\t",
                         header=None,
                         names = ["Rock_number","Subtype_number","Token_number",
                                  ␣
 ↪"Porphyritic_texture","Presence_of_holes","Salient_green_hue","Pegmatitic_texture","Conchoi
```

```
                                    ␣
↪"Rounded_fragments","Straight_stripes","Curved_stripes","Physical_layers","Veins","Oily/
↪shimmery_texture",
                                    ␣
↪"Splotchy_texture","Single_translucent_crystal","Multiple_cubic_crystals","Sandy_texture","
                                     "Crystals_(disjunctive)"]
                          )
```

[8]:
```
df_norm540
```

[8]:

	Rock_number	Subtype_number	Token_number	Porphyritic_texture	\
0	1	1	1	1.690468	
1	2	1	2	1.690468	
2	3	1	3	1.665576	
3	4	1	4	2.233118	
4	5	1	5	2.213204	
..	…	…	…	…	
535	716	28	18	-0.037053	
536	717	29	17	-0.584681	
537	718	29	18	-0.559789	
538	719	30	17	-0.753949	
539	720	30	18	-0.968022	

	Presence_of_holes	Salient_green_hue	Pegmatitic_texture	\
0	-0.159688	-0.646115	-0.252007	
1	-0.159688	-0.530724	0.127922	
2	-0.407623	0.858984	-0.631936	
3	-0.407623	-0.415333	-0.424702	
4	-0.159688	1.129901	-0.044773	
..	…	…	…	
535	-0.159688	-0.435401	-0.410886	
536	-0.407623	-0.957168	-0.493780	
537	0.881638	0.066299	-1.053311	
538	-0.407623	-0.029024	-0.701014	
539	-0.407623	-0.686250	-1.011865	

	Conchoidal_fracture	Angular_fragments	Rounded_fragments	…	\
0	-0.609794	0.579927	0.375313	…	
1	-0.482150	2.865772	0.375313	…	
2	-0.443857	2.611790	-0.405184	…	
3	-1.120369	0.071962	4.017633	…	
4	-1.082076	1.341876	3.757467	…	
..	…	…	…	…	
535	-1.126751	-0.436004	-0.405184	…	
536	0.066717	-0.436004	-0.405184	…	
537	-1.088458	-0.436004	-0.405184	…	
538	0.366680	-0.436004	-0.405184	…	

```
539             1.387829          -0.436004          -0.405184  …

     Physical_layers     Veins  Oily/shimmery_texture  Splotchy_texture  \
0          -0.759128 -0.013842              -0.540653          0.946521
1          -0.529150 -0.512160              -0.540653         -0.249084
2          -0.529150 -0.512160              -0.540653          1.245422
3          -0.529150 -0.512160              -0.540653         -0.249084
4          -0.759128 -0.512160              -0.540653         -0.249084
..               …       …                      …                 …
535        -0.759128 -0.512160              -0.540653         -0.547986
536        -0.069195 -0.013842              -0.540653         -0.846887
537        -0.759128 -0.512160              -0.540653         -0.846887
538         1.770626  0.484475              -0.353270         -0.547986
539         1.540648 -0.013842              -0.165887         -0.846887

     Single_translucent_crystal  Multiple_cubic_crystals  Sandy_texture  \
0                     -0.227922                -0.225045      -0.116312
1                     -0.227922                 0.185510      -0.401124
2                     -0.227922                -0.225045      -0.401124
3                     -0.227922                -0.225045      -0.116312
4                     -0.227922                -0.225045      -0.401124
..                          …                       …              …
535                   -0.227922                 7.986072      -0.685937
536                   -0.227922                -0.225045       1.592561
537                   -0.227922                -0.225045       5.010309
538                   -0.227922                -0.225045      -0.685937
539                   -0.227922                -0.225045      -0.116312

     Fragments_(disjunctive)  Stripes_(disjunctive)  Crystals_(disjunctive)
0                   0.635812              -0.409247               -0.310419
1                   2.042938              -0.409247               -0.034059
2                   1.665865              -0.409247               -0.310419
3                   2.640737              -0.409247               -0.310419
4                   2.659131              -0.409247               -0.310419
..                       …                     …                       …
535                -0.541391              -0.409247                5.216791
536                -0.541391               3.054169               -0.310419
537                -0.541391              -0.409247               -0.310419
538                -0.541391              -0.409247               -0.310419
539                -0.541391               0.368255               -0.310419

[540 rows x 22 columns]
```

[9]:
```
# categoryDict = {1: "Igneous", 2: "Metamorphic", 3:"Sedimentary"}
# stimulusDict = {1:"Non_parent_training",2:"Parent_training", 3:
 ↪"Standard_transfer",4:"HSN_transfer"}
```

```
# def discriptiveCols(row):
#   cat_num = row["Category_number"]
#   stimulus_num = row["Stimulus_Type"]
#   return categoryDict.get(cat_num),stimulusDict.get(stimulus_num)

# df[["category","stimulus"]] = df.apply(lambda row:␣
 ↪discriptiveCols(row),axis=1, result_type = "expand")
```

[10]:
```
df
```

[10]:

	Rock_number	Category_number	Subtype_number	Token_number	\
0	1	1	1	1	
1	2	1	1	2	
2	3	1	1	3	
3	4	1	1	4	
4	5	1	1	5	
..	…	…	…	…	
535	716	3	28	18	
536	717	3	29	17	
537	718	3	29	18	
538	719	3	30	17	
539	720	3	30	18	

	Stimulus_Type	Frequency_responded_Igneous	\
0	2	57	
1	3	59	
2	3	37	
3	3	41	
4	3	42	
..	…	…	
535	4	1	
536	4	2	
537	4	25	
538	4	16	
539	4	6	

	Frequency_responded_Metamorphic	Frequency_responded_Sedimentary	\
0	19	6	
1	10	13	
2	28	17	
3	11	30	
4	6	34	
..	…	…	
535	0	81	
536	44	36	
537	9	48	
538	46	20	

```
539                              36                              40

     Frequency_responded_Old  Frequency_responded_New  \
0                         46                       36
1                         24                       58
2                         20                       62
3                         29                       53
4                         25                       57
..                       …                        …
535                       57                       25
536                       40                       42
537                       27                       55
538                       23                       59
539                       30                       52

     Proportion_responded_Igneous  Proportion_responded_Metamorphic  \
0                        0.695122                          0.231707
1                        0.719512                          0.121951
2                        0.451220                          0.341463
3                        0.500000                          0.134146
4                        0.512195                          0.073171
..                            …                                 …
535                      0.012195                          0.000000
536                      0.024390                          0.536585
537                      0.304878                          0.109756
538                      0.195122                          0.560976
539                      0.073171                          0.439024

     Proportion_responded_Sedimentary  Proportion_responded_Old
0                            0.073171                  0.560976
1                            0.158537                  0.292683
2                            0.207317                  0.243902
3                            0.365854                  0.353659
4                            0.414634                  0.304878
..                                …                         …
535                          0.987805                  0.695122
536                          0.439024                  0.487805
537                          0.585366                  0.329268
538                          0.243902                  0.280488
539                          0.487805                  0.365854

[540 rows x 14 columns]
```

[11]:
```
df = df.merge(df_norm540, how = "left",␣
 ↪on=["Rock_number","Subtype_number","Token_number"])
```

[12]:
```
colNames.remove("Category_number")
df.drop(columns=colNames, inplace =True)
```

[13]:
```
df
```

[13]:

	Category_number	Porphyritic_texture	Presence_of_holes	\
0	1	1.690468	-0.159688	
1	1	1.690468	-0.159688	
2	1	1.665576	-0.407623	
3	1	2.233118	-0.407623	
4	1	2.213204	-0.159688	
..	…	…	…	
535	3	-0.037053	-0.159688	
536	3	-0.584681	-0.407623	
537	3	-0.559789	0.881638	
538	3	-0.753949	-0.407623	
539	3	-0.968022	-0.407623	

	Salient_green_hue	Pegmatitic_texture	Conchoidal_fracture	\
0	-0.646115	-0.252007	-0.609794	
1	-0.530724	0.127922	-0.482150	
2	0.858984	-0.631936	-0.443857	
3	-0.415333	-0.424702	-1.120369	
4	1.129901	-0.044773	-1.082076	
..	…	…	…	
535	-0.435401	-0.410886	-1.126751	
536	-0.957168	-0.493780	0.066717	
537	0.066299	-1.053311	-1.088458	
538	-0.029024	-0.701014	0.366680	
539	-0.686250	-1.011865	1.387829	

	Angular_fragments	Rounded_fragments	Straight_stripes	Curved_stripes	\
0	0.579927	0.375313	-0.352386	-0.260224	
1	2.865772	0.375313	-0.352386	-0.260224	
2	2.611790	-0.405184	-0.352386	-0.260224	
3	0.071962	4.017633	-0.352386	-0.260224	
4	1.341876	3.757467	-0.352386	-0.260224	
..	…	…	…	…	
535	-0.436004	-0.405184	-0.352386	-0.260224	
536	-0.436004	-0.405184	4.102756	0.045922	
537	-0.436004	-0.405184	-0.352386	-0.260224	
538	-0.436004	-0.405184	-0.352386	-0.260224	
539	-0.436004	-0.405184	0.433816	0.045922	

	Physical_layers	Veins	Oily/shimmery_texture	Splotchy_texture	\
0	-0.759128	-0.013842	-0.540653	0.946521	
1	-0.529150	-0.512160	-0.540653	-0.249084	

```
2          -0.529150 -0.512160              -0.540653          1.245422
3          -0.529150 -0.512160              -0.540653         -0.249084
4          -0.759128 -0.512160              -0.540653         -0.249084
..               …       …                    …                …
535        -0.759128 -0.512160              -0.540653         -0.547986
536        -0.069195 -0.013842              -0.540653         -0.846887
537        -0.759128 -0.512160              -0.540653         -0.846887
538         1.770626  0.484475              -0.353270         -0.547986
539         1.540648 -0.013842              -0.165887         -0.846887

     Single_translucent_crystal  Multiple_cubic_crystals  Sandy_texture  \
0                     -0.227922                -0.225045      -0.116312
1                     -0.227922                 0.185510      -0.401124
2                     -0.227922                -0.225045      -0.401124
3                     -0.227922                -0.225045      -0.116312
4                     -0.227922                -0.225045      -0.401124
..                          …                       …            …
535                   -0.227922                 7.986072      -0.685937
536                   -0.227922                -0.225045       1.592561
537                   -0.227922                -0.225045       5.010309
538                   -0.227922                -0.225045      -0.685937
539                   -0.227922                -0.225045      -0.116312

     Fragments_(disjunctive)  Stripes_(disjunctive)  Crystals_(disjunctive)
0                   0.635812              -0.409247               -0.310419
1                   2.042938              -0.409247               -0.034059
2                   1.665865              -0.409247               -0.310419
3                   2.640737              -0.409247               -0.310419
4                   2.659131              -0.409247               -0.310419
..                       …                     …                       …
535                -0.541391              -0.409247                5.216791
536                -0.541391               3.054169               -0.310419
537                -0.541391              -0.409247               -0.310419
538                -0.541391              -0.409247               -0.310419
539                -0.541391               0.368255               -0.310419

[540 rows x 20 columns]
```

[14]:
```
df.shape
```

[14]:
```
(540, 20)
```

[15]:
```
df.columns
```

[15]:
```
Index(['Category_number', 'Porphyritic_texture', 'Presence_of_holes',
       'Salient_green_hue', 'Pegmatitic_texture', 'Conchoidal_fracture',
       'Angular_fragments', 'Rounded_fragments', 'Straight_stripes',
```

```
       'Curved_stripes', 'Physical_layers', 'Veins', 'Oily/shimmery_texture',
       'Splotchy_texture', 'Single_translucent_crystal',
       'Multiple_cubic_crystals', 'Sandy_texture', 'Fragments_(disjunctive)',
       'Stripes_(disjunctive)', 'Crystals_(disjunctive)'],
      dtype='object')
```

[16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 540 entries, 0 to 539
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Category_number             540 non-null    int64
 1   Porphyritic_texture         540 non-null    float64
 2   Presence_of_holes           540 non-null    float64
 3   Salient_green_hue           540 non-null    float64
 4   Pegmatitic_texture          540 non-null    float64
 5   Conchoidal_fracture         540 non-null    float64
 6   Angular_fragments           540 non-null    float64
 7   Rounded_fragments           540 non-null    float64
 8   Straight_stripes            540 non-null    float64
 9   Curved_stripes              540 non-null    float64
 10  Physical_layers             540 non-null    float64
 11  Veins                       540 non-null    float64
 12  Oily/shimmery_texture       540 non-null    float64
 13  Splotchy_texture            540 non-null    float64
 14  Single_translucent_crystal  540 non-null    float64
 15  Multiple_cubic_crystals     540 non-null    float64
 16  Sandy_texture               540 non-null    float64
 17  Fragments_(disjunctive)     540 non-null    float64
 18  Stripes_(disjunctive)       540 non-null    float64
 19  Crystals_(disjunctive)      540 non-null    float64
dtypes: float64(19), int64(1)
memory usage: 88.6 KB
```

What attributes/features are continuous valued?

- All the attributes are numerical attributes.
- The only categorical attributes is the rock category which the label for this data set.

[17]:
```
#Checking if there any null values in the dataset
df.isna().sum()
```

[17]:
```
Category_number             0
Porphyritic_texture         0
Presence_of_holes           0
Salient_green_hue           0
```

```
Pegmatitic_texture            0
Conchoidal_fracture           0
Angular_fragments             0
Rounded_fragments             0
Straight_stripes              0
Curved_stripes                0
Physical_layers               0
Veins                         0
Oily/shimmery_texture         0
Splotchy_texture              0
Single_translucent_crystal    0
Multiple_cubic_crystals       0
Sandy_texture                 0
Fragments_(disjunctive)       0
Stripes_(disjunctive)         0
Crystals_(disjunctive)        0
dtype: int64
```

No null values are found in the given dataset

0.1 Part 1: Display the statistical values for each of the attributes, along with visualizations (e.g., histogram) of the distributions for each attribute. Are there any attributes that might require special treatment? If so, what special treatment might they require?

[18]:
```
df.describe()
```

[18]:

	Category_number	Porphyritic_texture	Presence_of_holes	\
count	540.000000	5.400000e+02	5.400000e+02	
mean	2.000000	-1.851852e-09	9.629630e-08	
std	0.817254	1.000000e+00	1.000000e+00	
min	1.000000	-1.321491e+00	-4.076230e-01	
25%	1.000000	-8.236470e-01	-4.076230e-01	
50%	2.000000	-3.009100e-01	-4.076230e-01	
75%	3.000000	7.644770e-01	-1.596880e-01	
max	3.000000	2.422299e+00	4.551072e+00	

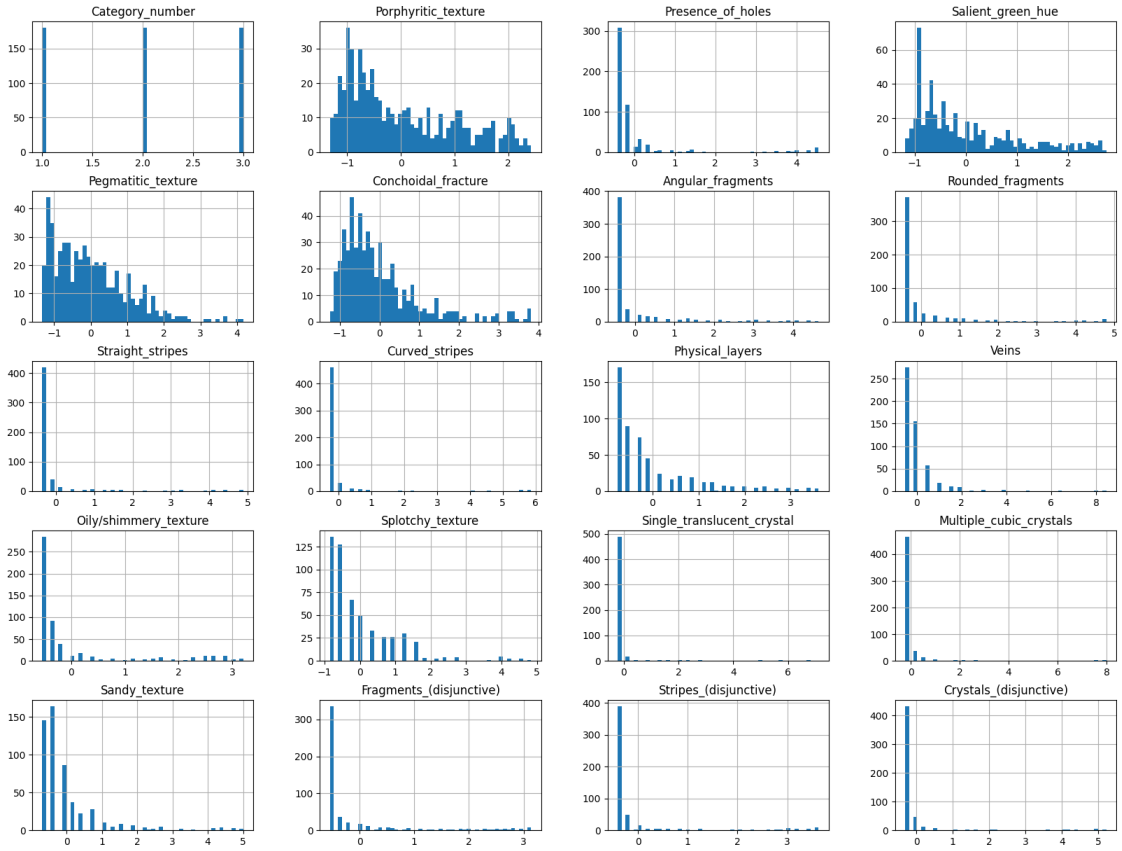
	Salient_green_hue	Pegmatitic_texture	Conchoidal_fracture	\
count	5.400000e+02	5.400000e+02	5.400000e+02	
mean	5.370370e-08	-6.296296e-08	-4.074074e-08	
std	1.000000e+00	1.000000e+00	1.000000e+00	
min	-1.187950e+00	-1.322715e+00	-1.248012e+00	
25%	-7.615050e-01	-8.046310e-01	-6.991450e-01	
50%	-3.751970e-01	-1.829290e-01	-2.715380e-01	
75%	5.843033e-01	5.769290e-01	3.571070e-01	
max	2.750390e+00	4.175892e+00	3.813059e+00	

```
       Angular_fragments  Rounded_fragments  Straight_stripes  Curved_stripes  \
count       5.400000e+02       5.400000e+02      5.400000e+02    5.400000e+02
mean       -1.537037e-07      -1.666667e-08     -1.814815e-07   -1.481481e-08
std         1.000000e+00       1.000000e+00      1.000000e+00    1.000000e+00
min        -4.360040e-01      -4.051840e-01     -3.523860e-01   -2.602240e-01
25%        -4.360040e-01      -4.051840e-01     -3.523860e-01   -2.602240e-01
50%        -4.360040e-01      -4.051840e-01     -3.523860e-01   -2.602240e-01
75%        -1.820210e-01      -1.450180e-01     -3.523860e-01   -2.602240e-01
max         4.643652e+00       4.798130e+00      4.888957e+00    5.862693e+00

       Physical_layers         Veins  Oily/shimmery_texture  Splotchy_texture  \
count     5.400000e+02  5.400000e+02           5.400000e+02      5.400000e+02
mean     -1.166667e-07 -1.629630e-07           1.703704e-07     -6.851852e-08
std       1.000000e+00  1.000000e+00           9.999999e-01      1.000000e+00
min      -7.591280e-01 -5.121600e-01          -5.406530e-01     -8.468870e-01
25%      -7.591280e-01 -5.121600e-01          -5.406530e-01     -8.468870e-01
50%      -2.991730e-01 -5.121600e-01          -5.406530e-01     -2.490840e-01
75%       3.907600e-01 -1.384200e-02          -1.658870e-01      3.487180e-01
max       3.610446e+00  8.457556e+00           3.207009e+00      4.832237e+00

       Single_translucent_crystal  Multiple_cubic_crystals  Sandy_texture  \
count                5.400000e+02             5.400000e+02   5.400000e+02
mean                -5.555556e-09             3.388889e-07   1.203704e-07
std                  1.000000e+00             9.999999e-01   1.000000e+00
min                 -2.279220e-01            -2.250450e-01  -6.859370e-01
25%                 -2.279220e-01            -2.250450e-01  -6.859370e-01
50%                 -2.279220e-01            -2.250450e-01  -4.011240e-01
75%                 -2.279220e-01            -2.250450e-01   1.685000e-01
max                  7.120010e+00             7.986072e+00   5.010309e+00

       Fragments_(disjunctive)  Stripes_(disjunctive)  Crystals_(disjunctive)
count             5.400000e+02           5.400000e+02            5.400000e+02
mean             -1.148148e-07          -1.759259e-07            3.018519e-07
std               1.000000e+00           1.000000e+00            9.999999e-01
min              -5.413910e-01          -4.092470e-01           -3.104190e-01
25%              -5.413910e-01          -4.092470e-01           -3.104190e-01
50%              -5.413910e-01          -4.092470e-01           -3.104190e-01
75%               1.042300e-02          -2.072980e-01           -3.104190e-01
max               3.137369e+00           3.629722e+00            5.216791e+00
```

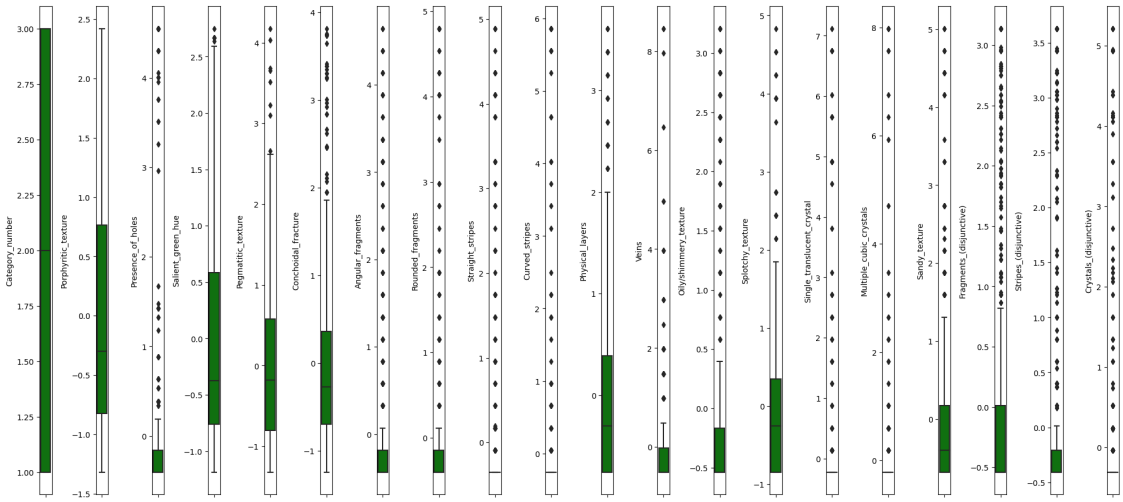
[19]:
```
describeNum = df.describe(include =['float64', 'int64', 'float', 'int'])
describeNum.T.style.background_gradient(cmap='viridis',low=0.2,high=0.1)
```

[19]:
```
<pandas.io.formats.style.Styler at 0x7d00d5c82260>
```

[20]:
```
df.hist(bins=50, figsize=(20,15))
plt.show()
```



[21]:
```
plt.figure(figsize=(20, 9))
for i, column in enumerate(df.columns):
    plt.subplot(1, len(df.columns), i + 1)
    sns.boxplot(y=df[column], color='green', orient='v')
plt.tight_layout()
```



- The data distributions show significant skewness in the data, which requires us to employ feature scaling techniques like normalization or standardization to mitigate this skewness.
- Additionally, we have dropped the rows that contain missing values.
- Upon examining the distributions as described, we have identified outliers in each attribute. We have opted to retain these outliers since they could potentially play a valuable role in our predictions for rock categories. Another reason to retain the outliers is that we have limited data points, hence we can't drop it.

0.2 Part 2: Analyze and discuss the relationships between the data attributes, and between the data attributes and label. This involves computing the Pearson Correlation Coefficient (PCC) and generating scatter plots.

[22]:
```
df.corr()
```

[22]:

	Category_number	Porphyritic_texture	\
Category_number	1.000000	-0.305296	
Porphyritic_texture	-0.305296	1.000000	
Presence_of_holes	-0.301462	0.017601	
Salient_green_hue	-0.104748	0.255830	
Pegmatitic_texture	-0.164925	0.534070	
Conchoidal_fracture	0.103781	-0.505542	
Angular_fragments	-0.123388	0.645724	
Rounded_fragments	0.009450	0.653319	
Straight_stripes	0.086923	-0.117642	
Curved_stripes	-0.016680	-0.026782	
Physical_layers	0.262855	-0.490498	
Veins	0.130094	-0.189307	
Oily/shimmery_texture	0.007679	-0.310966	
Splotchy_texture	-0.242921	0.373241	
Single_translucent_crystal	0.253550	-0.163897	
Multiple_cubic_crystals	0.121163	0.046817	
Sandy_texture	0.132240	-0.226060	
Fragments_(disjunctive)	-0.068022	0.819815	
Stripes_(disjunctive)	0.061825	-0.108380	
Crystals_(disjunctive)	0.258386	-0.086319	

	Presence_of_holes	Salient_green_hue	\
Category_number	-0.301462	-0.104748	
Porphyritic_texture	0.017601	0.255830	
Presence_of_holes	1.000000	-0.026329	
Salient_green_hue	-0.026329	1.000000	
Pegmatitic_texture	-0.199425	0.126113	
Conchoidal_fracture	-0.248047	-0.189701	
Angular_fragments	-0.026176	0.064291	
Rounded_fragments	-0.026623	0.186064	

Straight_stripes	-0.112504	-0.061815
Curved_stripes	-0.086442	0.006080
Physical_layers	-0.210501	-0.050614
Veins	-0.155518	-0.033938
Oily/shimmery_texture	-0.154314	-0.131127
Splotchy_texture	-0.098999	0.089565
Single_translucent_crystal	-0.082330	-0.168250
Multiple_cubic_crystals	-0.044729	0.011505
Sandy_texture	0.283791	-0.072024
Fragments_(disjunctive)	-0.035854	0.169734
Stripes_(disjunctive)	-0.132162	-0.042696
Crystals_(disjunctive)	-0.087471	-0.109629

	Pegmatitic_texture	Conchoidal_fracture	\
Category_number	-0.164925	0.103781	
Porphyritic_texture	0.534070	-0.505542	
Presence_of_holes	-0.199425	-0.248047	
Salient_green_hue	0.126113	-0.189701	
Pegmatitic_texture	1.000000	0.055266	
Conchoidal_fracture	0.055266	1.000000	
Angular_fragments	0.450381	-0.210173	
Rounded_fragments	0.321181	-0.260734	
Straight_stripes	-0.081682	-0.019918	
Curved_stripes	0.006070	0.058211	
Physical_layers	-0.334001	0.275226	
Veins	-0.099182	0.058838	
Oily/shimmery_texture	0.180904	0.641196	
Splotchy_texture	0.141837	-0.240125	
Single_translucent_crystal	0.128090	0.075173	
Multiple_cubic_crystals	0.237183	-0.088640	
Sandy_texture	-0.442259	-0.196296	
Fragments_(disjunctive)	0.498259	-0.293316	
Stripes_(disjunctive)	-0.055575	0.018864	
Crystals_(disjunctive)	0.243675	-0.002322	

	Angular_fragments	Rounded_fragments	\
Category_number	-0.123388	0.009450	
Porphyritic_texture	0.645724	0.653319	
Presence_of_holes	-0.026176	-0.026623	
Salient_green_hue	0.064291	0.186064	
Pegmatitic_texture	0.450381	0.321181	
Conchoidal_fracture	-0.210173	-0.260734	
Angular_fragments	1.000000	0.260913	
Rounded_fragments	0.260913	1.000000	
Straight_stripes	-0.137379	-0.125969	
Curved_stripes	-0.103715	-0.084503	
Physical_layers	-0.229081	-0.236336	

Veins	-0.130497	-0.143923
Oily/shimmery_texture	-0.152724	-0.136893
Splotchy_texture	0.055986	0.009402
Single_translucent_crystal	-0.086229	-0.090216
Multiple_cubic_crystals	-0.027110	-0.045775
Sandy_texture	-0.164748	-0.133824
Fragments_(disjunctive)	0.783714	0.794627
Stripes_(disjunctive)	-0.161878	-0.143357
Crystals_(disjunctive)	-0.078127	-0.093649

	Straight_stripes	Curved_stripes	Physical_layers	\
Category_number	0.086923	-0.016680	0.262855	
Porphyritic_texture	-0.117642	-0.026782	-0.490498	
Presence_of_holes	-0.112504	-0.086442	-0.210501	
Salient_green_hue	-0.061815	0.006080	-0.050614	
Pegmatitic_texture	-0.081682	0.006070	-0.334001	
Conchoidal_fracture	-0.019918	0.058211	0.275226	
Angular_fragments	-0.137379	-0.103715	-0.229081	
Rounded_fragments	-0.125969	-0.084503	-0.236336	
Straight_stripes	1.000000	0.178444	0.006392	
Curved_stripes	0.178444	1.000000	-0.125805	
Physical_layers	0.006392	-0.125805	1.000000	
Veins	0.055417	0.043942	0.074632	
Oily/shimmery_texture	-0.102114	-0.069111	0.179334	
Splotchy_texture	-0.140431	-0.116719	-0.301661	
Single_translucent_crystal	-0.014729	-0.055039	-0.095277	
Multiple_cubic_crystals	-0.071865	-0.057039	-0.136120	
Sandy_texture	0.106271	-0.076266	-0.051544	
Fragments_(disjunctive)	-0.168211	-0.120035	-0.297611	
Stripes_(disjunctive)	0.805348	0.715588	-0.062761	
Crystals_(disjunctive)	-0.055964	-0.076533	-0.156928	

	Veins	Oily/shimmery_texture	Splotchy_texture	\
Category_number	0.130094	0.007679	-0.242921	
Porphyritic_texture	-0.189307	-0.310966	0.373241	
Presence_of_holes	-0.155518	-0.154314	-0.098999	
Salient_green_hue	-0.033938	-0.131127	0.089565	
Pegmatitic_texture	-0.099182	0.180904	0.141837	
Conchoidal_fracture	0.058838	0.641196	-0.240125	
Angular_fragments	-0.130497	-0.152724	0.055986	
Rounded_fragments	-0.143923	-0.136893	0.009402	
Straight_stripes	0.055417	-0.102114	-0.140431	
Curved_stripes	0.043942	-0.069111	-0.116719	
Physical_layers	0.074632	0.179334	-0.301661	
Veins	1.000000	-0.072125	0.031088	
Oily/shimmery_texture	-0.072125	1.000000	-0.218891	
Splotchy_texture	0.031088	-0.218891	1.000000	

```
Single_translucent_crystal -0.011992             -0.017060         -0.071954
Multiple_cubic_crystals    -0.092319             -0.050390         -0.072552
Sandy_texture              -0.063367             -0.311432         -0.165969
Fragments_(disjunctive)    -0.173014             -0.179942          0.044657
Stripes_(disjunctive)       0.066635             -0.112371         -0.170788
Crystals_(disjunctive)     -0.066545             -0.045595         -0.096146

                          Single_translucent_crystal  \
Category_number                             0.253550
Porphyritic_texture                        -0.163897
Presence_of_holes                          -0.082330
Salient_green_hue                          -0.168250
Pegmatitic_texture                          0.128090
Conchoidal_fracture                         0.075173
Angular_fragments                          -0.086229
Rounded_fragments                          -0.090216
Straight_stripes                           -0.014729
Curved_stripes                             -0.055039
Physical_layers                            -0.095277
Veins                                      -0.011992
Oily/shimmery_texture                      -0.017060
Splotchy_texture                           -0.071954
Single_translucent_crystal                  1.000000
Multiple_cubic_crystals                     0.085457
Sandy_texture                              -0.130810
Fragments_(disjunctive)                    -0.112346
Stripes_(disjunctive)                      -0.040831
Crystals_(disjunctive)                      0.769259

                          Multiple_cubic_crystals  Sandy_texture  \
Category_number                          0.121163       0.132240
Porphyritic_texture                      0.046817      -0.226060
Presence_of_holes                       -0.044729       0.283791
Salient_green_hue                        0.011505      -0.072024
Pegmatitic_texture                       0.237183      -0.442259
Conchoidal_fracture                     -0.088640      -0.196296
Angular_fragments                       -0.027110      -0.164748
Rounded_fragments                       -0.045775      -0.133824
Straight_stripes                        -0.071865       0.106271
Curved_stripes                          -0.057039      -0.076266
Physical_layers                         -0.136120      -0.051544
Veins                                   -0.092319      -0.063367
Oily/shimmery_texture                   -0.050390      -0.311432
Splotchy_texture                        -0.072552      -0.165969
Single_translucent_crystal               0.085457      -0.130810
Multiple_cubic_crystals                  1.000000      -0.124932
Sandy_texture                           -0.124932       1.000000
```

Fragments_(disjunctive)	-0.042715	-0.193201
Stripes_(disjunctive)	-0.085779	0.034719
Crystals_(disjunctive)	0.697034	-0.174296

	Fragments_(disjunctive)	Stripes_(disjunctive)	\
Category_number	-0.068022	0.061825	
Porphyritic_texture	0.819815	-0.108380	
Presence_of_holes	-0.035854	-0.132162	
Salient_green_hue	0.169734	-0.042696	
Pegmatitic_texture	0.498259	-0.055575	
Conchoidal_fracture	-0.293316	0.018864	
Angular_fragments	0.783714	-0.161878	
Rounded_fragments	0.794627	-0.143357	
Straight_stripes	-0.168211	0.805348	
Curved_stripes	-0.120035	0.715588	
Physical_layers	-0.297611	-0.062761	
Veins	-0.173014	0.066635	
Oily/shimmery_texture	-0.179942	-0.112371	
Splotchy_texture	0.044657	-0.170788	
Single_translucent_crystal	-0.112346	-0.040831	
Multiple_cubic_crystals	-0.042715	-0.085779	
Sandy_texture	-0.193201	0.034719	
Fragments_(disjunctive)	1.000000	-0.194927	
Stripes_(disjunctive)	-0.194927	1.000000	
Crystals_(disjunctive)	-0.106859	-0.084039	

	Crystals_(disjunctive)
Category_number	0.258386
Porphyritic_texture	-0.086319
Presence_of_holes	-0.087471
Salient_green_hue	-0.109629
Pegmatitic_texture	0.243675
Conchoidal_fracture	-0.002322
Angular_fragments	-0.078127
Rounded_fragments	-0.093649
Straight_stripes	-0.055964
Curved_stripes	-0.076533
Physical_layers	-0.156928
Veins	-0.066545
Oily/shimmery_texture	-0.045595
Splotchy_texture	-0.096146
Single_translucent_crystal	0.769259
Multiple_cubic_crystals	0.697034
Sandy_texture	-0.174296
Fragments_(disjunctive)	-0.106859
Stripes_(disjunctive)	-0.084039
Crystals_(disjunctive)	1.000000

[23]:
```
df_label = df["Category_number"]
df_attributes = df.drop(columns = ["Category_number"])
```

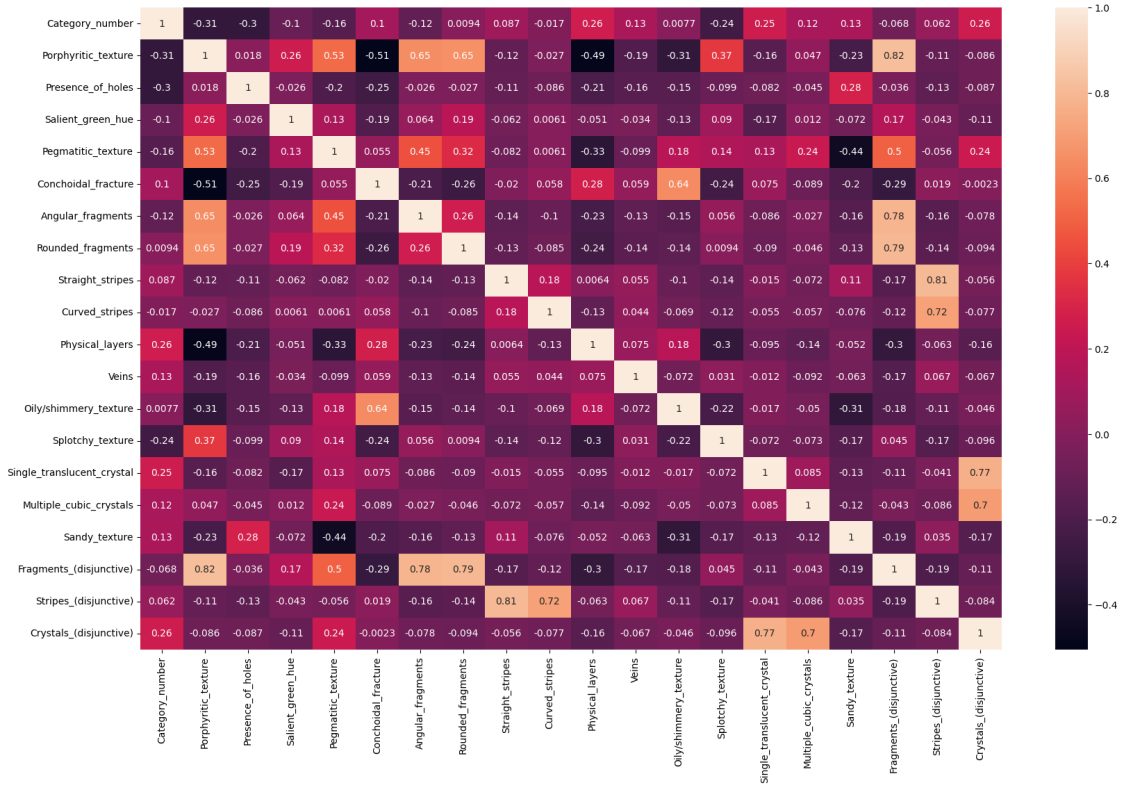
[24]:
```
import pandas as pd
from scipy import stats

# Create a function to calculate biserial correlation
def biserial_corr(x, y):
    corr, _ = stats.pointbiserialr(x, y)
    return corr

# Calculate biserial correlation for each attribute with the label
biserial_corr_values = df_attributes.apply(lambda col: biserial_corr(col,␣
 ↪df_label))

# Create a DataFrame to hold the biserial correlation values
biserial_corr_df = pd.DataFrame({'Biserial Correlation': biserial_corr_values})

# Display the correlation values in a table
print(biserial_corr_df)
```

```
                          Biserial Correlation
Porphyritic_texture                  -0.305296
Presence_of_holes                    -0.301462
Salient_green_hue                    -0.104748
Pegmatitic_texture                   -0.164925
Conchoidal_fracture                   0.103781
Angular_fragments                    -0.123388
Rounded_fragments                     0.009450
Straight_stripes                      0.086923
Curved_stripes                       -0.016680
Physical_layers                       0.262855
Veins                                 0.130094
Oily/shimmery_texture                 0.007679
Splotchy_texture                     -0.242921
Single_translucent_crystal            0.253550
Multiple_cubic_crystals               0.121163
Sandy_texture                         0.132240
Fragments_(disjunctive)              -0.068022
Stripes_(disjunctive)                 0.061825
Crystals_(disjunctive)                0.258386
```

Based on the above correlation table, we can identify that * Physical Layers, Single_translucent_crystal, Crystals_(disjunctive) are positive correlated with the rock categories. * On the other hand, Porphyritic_texture, Presence_of_holes and Splotchy_texture are strongly negatively correlated with the rock category

[25]:
```
fig,ax=plt.subplots(figsize=(20,12))
sns.heatmap(df.corr(),annot=True)
```

[25]: <Axes: >



Analyzing the correlations between the attributies we can infer the following:

- The correlation between similar attributes such as Fragements, crystals and stripes are highly correlated amongst each other.
- Porphyritic_texture is highly correlated with fragments.
- Conchoidal_fracture and Oily/Shimmery texture are correlated with each other

[26]:
```
df_label
```

[26]:
```
0      1
1      1
2      1
3      1
4      1
      ..
535    3
536    3
537    3
```

```
538    3
539    3
Name: Category_number, Length: 540, dtype: int64
```

[27]:
```
df_attributes
```

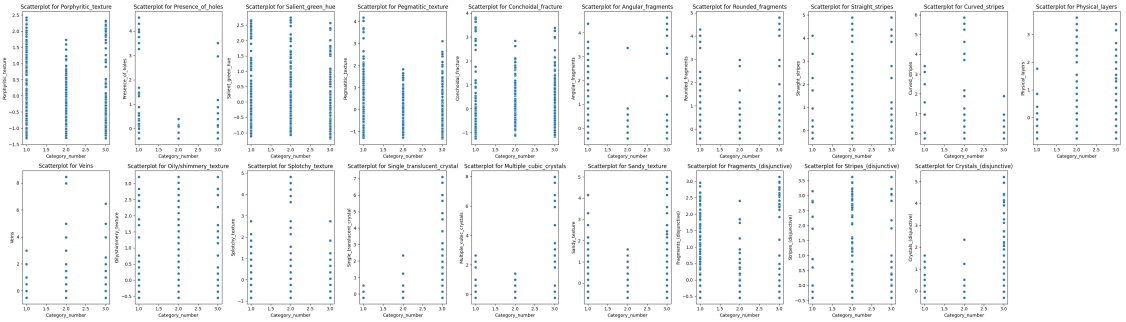
[27]:
```
     Porphyritic_texture  Presence_of_holes  Salient_green_hue  \
0               1.690468          -0.159688          -0.646115
1               1.690468          -0.159688          -0.530724
2               1.665576          -0.407623           0.858984
3               2.233118          -0.407623          -0.415333
4               2.213204          -0.159688           1.129901
..                   …                  …                  …
535            -0.037053          -0.159688          -0.435401
536            -0.584681          -0.407623          -0.957168
537            -0.559789           0.881638           0.066299
538            -0.753949          -0.407623          -0.029024
539            -0.968022          -0.407623          -0.686250

     Pegmatitic_texture  Conchoidal_fracture  Angular_fragments  \
0             -0.252007            -0.609794           0.579927
1              0.127922            -0.482150           2.865772
2             -0.631936            -0.443857           2.611790
3             -0.424702            -1.120369           0.071962
4             -0.044773            -1.082076           1.341876
..                  …                    …                  …
535           -0.410886            -1.126751          -0.436004
536           -0.493780             0.066717          -0.436004
537           -1.053311            -1.088458          -0.436004
538           -0.701014             0.366680          -0.436004
539           -1.011865             1.387829          -0.436004

     Rounded_fragments  Straight_stripes  Curved_stripes  Physical_layers  \
0             0.375313         -0.352386       -0.260224        -0.759128
1             0.375313         -0.352386       -0.260224        -0.529150
2            -0.405184         -0.352386       -0.260224        -0.529150
3             4.017633         -0.352386       -0.260224        -0.529150
4             3.757467         -0.352386       -0.260224        -0.759128
..                 …                 …               …                …
535          -0.405184         -0.352386       -0.260224        -0.759128
536          -0.405184          4.102756        0.045922        -0.069195
537          -0.405184         -0.352386       -0.260224        -0.759128
538          -0.405184         -0.352386       -0.260224         1.770626
539          -0.405184          0.433816        0.045922         1.540648

        Veins  Oily/shimmery_texture  Splotchy_texture  \
0   -0.013842              -0.540653          0.946521
```

```
1   -0.512160             -0.540653         -0.249084
2   -0.512160             -0.540653          1.245422
3   -0.512160             -0.540653         -0.249084
4   -0.512160             -0.540653         -0.249084
..        …                     …                 …
535 -0.512160             -0.540653         -0.547986
536 -0.013842             -0.540653         -0.846887
537 -0.512160             -0.540653         -0.846887
538  0.484475             -0.353270         -0.547986
539 -0.013842             -0.165887         -0.846887

     Single_translucent_crystal  Multiple_cubic_crystals  Sandy_texture  \
0                     -0.227922                -0.225045      -0.116312
1                     -0.227922                 0.185510      -0.401124
2                     -0.227922                -0.225045      -0.401124
3                     -0.227922                -0.225045      -0.116312
4                     -0.227922                -0.225045      -0.401124
..                          …                       …              …
535                   -0.227922                 7.986072      -0.685937
536                   -0.227922                -0.225045       1.592561
537                   -0.227922                -0.225045       5.010309
538                   -0.227922                -0.225045      -0.685937
539                   -0.227922                -0.225045      -0.116312

     Fragments_(disjunctive)  Stripes_(disjunctive)  Crystals_(disjunctive)
0                   0.635812              -0.409247               -0.310419
1                   2.042938              -0.409247               -0.034059
2                   1.665865              -0.409247               -0.310419
3                   2.640737              -0.409247               -0.310419
4                   2.659131              -0.409247               -0.310419
..                       …                     …                       …
535                -0.541391              -0.409247                5.216791
536                -0.541391               3.054169               -0.310419
537                -0.541391              -0.409247               -0.310419
538                -0.541391              -0.409247               -0.310419
539                -0.541391               0.368255               -0.310419

[540 rows x 19 columns]
```

[28]:
```
#sns.pairplot(df_attributes)
```

[29]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define the number of rows and columns for subplots
num_rows = 2  # Adjust this based on your preference
num_cols = (len(df_attributes.columns) + num_rows - 1) // num_rows
```

```
# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(35, 10))

# Flatten the axes for easy iteration
axes = axes.ravel()

# Loop through the columns and create scatter plots
for i, colname in enumerate(df_attributes.columns):
    sns.scatterplot(data=df, x="Category_number", y=colname, ax=axes[i])
    axes[i].set_title(f"Scatterplot for {colname}")
    axes[i].set_xlabel("Category_number")
    axes[i].set_ylabel(colname)

# Remove any empty subplots
for i in range(len(df_attributes.columns), len(axes)):
    fig.delaxes(axes[i])

# Adjust spacing and display the subplots
plt.tight_layout()
plt.show()
```



0.3 Part 3: Select 20% of the data for testing and 20% for validation and use the remaining 60% of the data for training. Describe how you did that and verify that your test and validation portions of the data are representative of the entire dataset.

[30]:
```
from sklearn.model_selection import train_test_split
```

[31]:
```
X_train, X_temp, y_train, y_temp = train_test_split(df_attributes, df_label,␣
↪test_size=0.4, random_state=33)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
↪random_state=33)
```

The dataset was initially divided into three subsets: a training set, a temporary dataset, and a

test set. Sixty percent of the data was allocated to the training set. The temporary dataset was subsequently split into two parts, with 20% designated for validation and another 20% allocated to the test set.

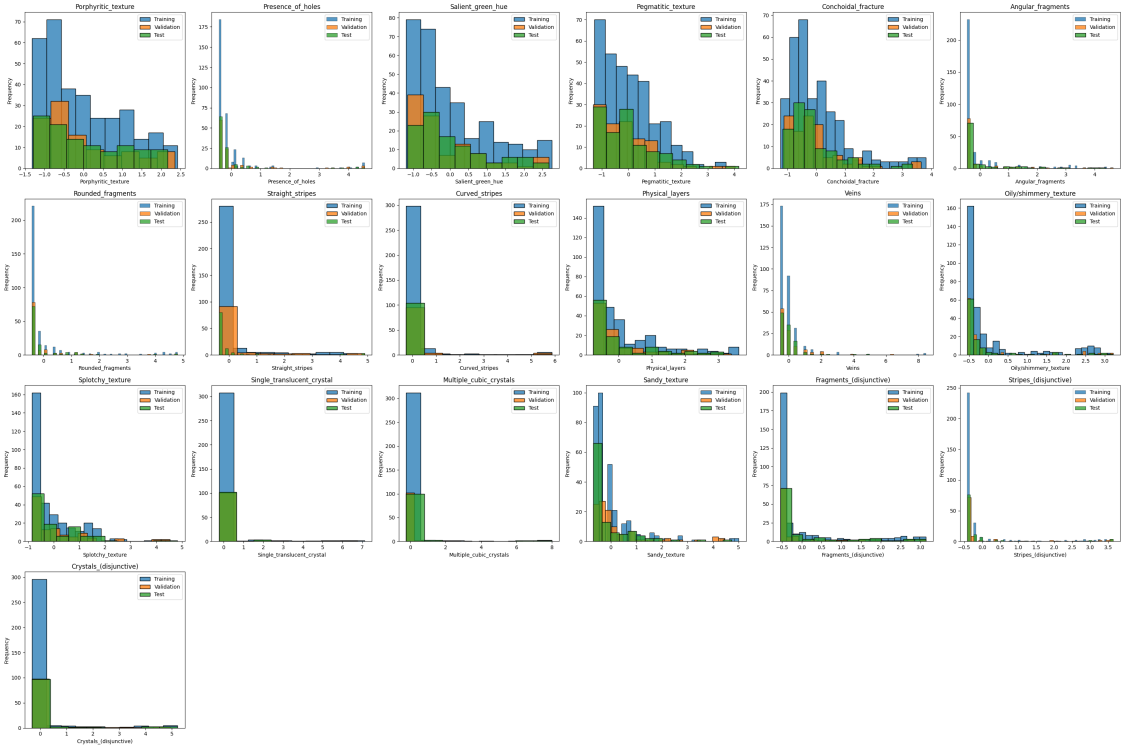
[32]:
```
#Checking for any imbalance in the dataset
df_label.value_counts()
```

[32]:
```
1    180
2    180
3    180
Name: Category_number, dtype: int64
```

[33]:
```
import numpy as np
X_train_numerical = X_train.select_dtypes(include=np.number)
X_val_numerical = X_val.select_dtypes(include=np.number)
X_test_numerical = X_test.select_dtypes(include=np.number)
```

[34]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import math

# Create subplots
num_cols = len(df_attributes.columns)
graphs_per_row = 6
num_rows = math.ceil(num_cols / graphs_per_row)

fig, axes = plt.subplots(num_rows, graphs_per_row, figsize=(30, 5 * num_rows))

datasets = [X_train_numerical, X_val_numerical, X_test_numerical]
dataset_names = ['Training', 'Validation', 'Test']

# Loop through columns and create histograms
for i, colname in enumerate(df_attributes.columns):
    row_idx = i // graphs_per_row
    col_idx = i % graphs_per_row
    ax = axes[row_idx, col_idx]
    for j, dataset in enumerate(datasets):
        sns.histplot(data=dataset, x=colname, ax=ax, label=dataset_names[j])
    ax.set_title(colname)
    ax.set_xlabel(colname)
    ax.set_ylabel("Frequency")
    ax.legend()

# Remove any empty subplots
for i in range(num_cols, num_rows * graphs_per_row):
    row_idx = i // graphs_per_row
    col_idx = i % graphs_per_row
    fig.delaxes(axes[row_idx, col_idx])
```

```
# Adjust spacing and display the subplots
plt.tight_layout()
plt.show()
```



[35]:
```
y_train.value_counts(normalize = True)
```

[35]:
```
1    0.345679
2    0.339506
3    0.314815
Name: Category_number, dtype: float64
```

[36]:
```
y_val.value_counts(normalize = True)
```

[36]:
```
2    0.361111
3    0.324074
1    0.314815
Name: Category_number, dtype: float64
```

[37]:
```
y_test.value_counts(normalize = True)
```

[37]:
```
3    0.398148
1    0.314815
```

```
2    0.287037
Name: Category_number, dtype: float64
```

From the above attribute and label dostriubutions, we can identify that the validation and test set is the same proportion of the train set and is representative of the entire dataset.

0.4 Part 4: Train different classifiers and tweak the hyperparameters to improve performance (you can use the grid search if you want or manually try different values). Report training, validation and testing performance (classification accuracy, precision, recall and F1 score) and discuss the impact of the hyperparameters (use markdown cells in Jupyter Notebook to clearly indicate each solution):

Normalizing the features because SVM is sensitive to scale.

[38]:
```
from sklearn.preprocessing import MinMaxScaler

num_cols = X_train.columns

for i in num_cols:

  scale = MinMaxScaler().fit(X_train[[i]])

  X_train[i] = scale.transform(X_train[[i]])
  X_val[i] = scale.transform(X_val[[i]])
  X_test[i] = scale.transform(X_test[[i]])
```

Function to evaluate classification models

[39]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

def compute_evaluation_metrics(y_true, y_pred):

    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average = "macro")
    recall = recall_score(y_true, y_pred, average = "macro")
    f1 = f1_score(y_true, y_pred, average = "macro")

    return acc, precision, recall, f1
```

0.5 A. Multinomial Logistic Regression (softmax regression); hyperparameters to explore: C, solver, max number of iterations.

[40]:
```
from sklearn.linear_model import LogisticRegression
logreg_orig=LogisticRegression(solver = 'lbfgs', max_iter = 1000, penalty =␣
 ↪'l2')
logreg_orig.fit(X=X_train, y=y_train)
```

[40]:
```
LogisticRegression(max_iter=1000)
```

[41]:
```
y_pred_train_lr = logreg_orig.predict(X_train)
y_pred_val_lr = logreg_orig.predict(X_val)
y_pred_test_lr = logreg_orig.predict(X_test)

acc_train_lr, prec_train_lr, rec_train_lr, f1_train_lr =␣
 ↪compute_evaluation_metrics(y_train, y_pred_train_lr)
acc_val_lr, prec_val_lr, rec_val_lr, f1_val_lr =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_lr)
acc_test_lr, prec_test_lr, rec_test_lr, f1_test_lr =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_lr)

print('Logistic Regression Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_lr)
print('Precision: ', prec_train_lr)
print('Recall: ', rec_train_lr)
print('F1-Score: ', f1_train_lr)
print()

print('Logistic Regression Model Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_lr)
print('Precision: ', prec_val_lr)
print('Recall: ', rec_val_lr)
print('F1-Score: ', f1_val_lr)
print()

print('Logistic Regression Model Evaluation - Test Set')
print('-----------------------------------------------')
print('Accuracy: ', acc_test_lr)
print('Precision: ', prec_test_lr)
print('Recall: ', rec_test_lr)
print('F1-Score: ', f1_test_lr)
print()
```

```
Logistic Regression Model Evaluation - Training Set
---------------------------------------------------
```

```
Accuracy:  0.7006172839506173
Precision:  0.7020489495700227
Recall:  0.6976689160512689
F1-Score:  0.6973899963774929

Logistic Regression Model Evaluation - Validation Set
------------------------------------------------------
Accuracy:  0.75
Precision:  0.75
Recall:  0.7467571644042232
F1-Score:  0.7444253203289347

Logistic Regression Model Evaluation - Test Set
-----------------------------------------------
Accuracy:  0.7407407407407407
Precision:  0.7759728783058583
Recall:  0.7638380183281114
F1-Score:  0.737777777777778
```

[42]:
```
#Applying Grid search for Hyperparametre tuning and testing the model on Train,␣
 ↪Validationa and Test

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import PredefinedSplit
from sklearn.linear_model import LogisticRegression

split_index = [-1 if x in X_train.index else 0 for x in pd.concat([X_train,␣
 ↪X_val]).index]

# use the list to create PredefinedSplit
pds = PredefinedSplit(test_fold = split_index)

lr_params = {'C': [0.001, 0.01, 0.1, 1, 5, 10, 100],
             'solver': ['newton-cg','lbfgs','saga','sag','liblinear'],
             'max_iter': [100, 200, 500, 1000, 5000, 10000]}

lr_grid_search = GridSearchCV(LogisticRegression(random_state = 42),
                              param_grid=lr_params,
                              cv=pds,
                              n_jobs=-1)
lr_grid_search.fit(pd.concat([X_train, X_val]),
                   pd.concat([y_train, y_val]))
lr = lr_grid_search.best_estimator_

print("Best Hyperparameters for Logistic Regression: ", lr)
print("Best Score: ", lr_grid_search.best_score_)
```

```
y_pred_train_lr = lr.predict(X_train)
y_pred_val_lr = lr.predict(X_val)
y_pred_test_lr = lr.predict(X_test)

acc_train_lr, prec_train_lr, rec_train_lr, f1_train_lr =␣
 ↪compute_evaluation_metrics(y_train, y_pred_train_lr)
acc_val_lr, prec_val_lr, rec_val_lr, f1_val_lr =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_lr)
acc_test_lr, prec_test_lr, rec_test_lr, f1_test_lr =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_lr)

print('Logistic Regression Model Evaluation - Training Set')
print('--------------------------------------------------')
print('Accuracy: ', acc_train_lr)
print('Precision: ', prec_train_lr)
print('Recall: ', rec_train_lr)
print('F1-Score: ', f1_train_lr)
print()

print('Logistic Regression Model Evaluation - Validation Set')
print('--------------------------------------------------------')
print('Accuracy: ', acc_val_lr)
print('Precision: ', prec_val_lr)
print('Recall: ', rec_val_lr)
print('F1-Score: ', f1_val_lr)
print()

print('Logistic Regression Model Evaluation - Test Set')
print('---------------------------------------------')
print('Accuracy: ', acc_test_lr)
print('Precision: ', prec_test_lr)
print('Recall: ', rec_test_lr)
print('F1-Score: ', f1_test_lr)
print()
```

```
Best Hyperparameters for Logistic Regression:  LogisticRegression(C=100,
random_state=42, solver='newton-cg')
Best Score:  0.7685185185185185
Logistic Regression Model Evaluation - Training Set
--------------------------------------------------
Accuracy:  0.7006172839506173
Precision:  0.6999195307371407
Recall:  0.6995490620490621
F1-Score:  0.69961386933864

Logistic Regression Model Evaluation - Validation Set
```

```
--------------------------------------------------
Accuracy:  0.7685185185185185
Precision:  0.7701370468611848
Recall:  0.7658047834518422
F1-Score:  0.7646396884573146

Logistic Regression Model Evaluation - Test Set
--------------------------------------------------
Accuracy:  0.7777777777777778
Precision:  0.7897435897435897
Recall:  0.7907418030978333
F1-Score:  0.7772994129158514
```

Base Model Performance:

- Training Set: The base logistic regression model achieved an accuracy of 70.06% on the training set. The precision, recall, and F1-score were all around 70%, indicating balanced performance.
- Validation Set: On the validation set, the accuracy was 75%. The precision and recall were also around 75%, with an F1-score of 74.44%.
- Test Set: The model performed similarly on the test set with an accuracy of 74.07%. The precision was 77.60%, recall was 76.38%, and the F1-score was 73.78%.

Hyperparameter Tuned Model Performance:

- Best Hyperparameters: After grid search, the best hyperparameters for logistic regression were found to be C=100, random_state=42, and solver='newton-cg'.
- Training Set: The accuracy of the tuned model on the training set remained approximately the same as the base model, around 70%. However, the precision, recall, and F1-score showed slight improvements, reaching around 70%.
- Validation Set: The tuned model performed significantly better on the validation set, with an accuracy of 76.85%, a precision of 77.01%, recall of 76.58%, and an F1-score of 76.46%. These improvements suggest that the hyperparameter tuning helped the model generalize better to unseen data.
- Test Set: Similar to the validation set, the test set performance of the tuned model was notably improved. The accuracy increased to 77.78%, the precision was 78.97%, recall was 79.07%, and the F1-score was 77.30%.

Discussion of Hyperparameters: 1. Regularization Strength (C):Lower values of C (e.g., C=0.01) increase the regularization strength, leading to a simpler model with smaller coefficients. This may help prevent overfitting, but it can result in reduced training set accuracy and, potentially, underfitting. Higher values of C (e.g., C=100) decrease the regularization strength, allowing the model to fit the training data more closely. This can lead to higher training set accuracy but may risk overfitting. 2. Different hyperparameters can impact the logistic regression model's performance in various ways. Applying Grid research helped us choose the required, which helped improve the performance of the model.

0.6 B. Support vector machines (make sure to try using kernels); hyperparameters to explore: C, kernel, degree of polynomial kernel, gamma.

[54]:
```
from sklearn.svm import SVC
svc_orig=SVC()
svc_orig.fit(X=X_train, y=y_train)

y_pred_train_svc = svc_orig.predict(X_train)
y_pred_val_svc = svc_orig.predict(X_val)
y_pred_test_svc = svc_orig.predict(X_test)

acc_train_svc, prec_train_svc, rec_train_svc, f1_train_svc =␣
 ↪compute_evaluation_metrics(y_train, y_pred_train_svc)
acc_val_svc, prec_val_svc, rec_val_svc, f1_val_svc =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_svc)
acc_test_svc, prec_test_svc, rec_test_svc, f1_test_svc =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_svc)

print('Support Vector Machine Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_svc)
print('Precision: ', prec_train_svc)
print('Recall: ', rec_train_svc)
print('F1-Score: ', f1_train_svc)
print()

print('Support Vector Machine Model Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_svc)
print('Precision: ', prec_val_svc)
print('Recall: ', rec_val_svc)
print('F1-Score: ', f1_val_svc)
print()

print('Support Vector Machine Model Evaluation - Test Set')
print('-----------------------------------------------')
print('Accuracy: ', acc_test_svc)
print('Precision: ', prec_test_svc)
print('Recall: ', rec_test_svc)
print('F1-Score: ', f1_test_svc)
print()
```

```
Support Vector Machine Model Evaluation - Training Set
-----------------------------------------------------
Accuracy:  0.7777777777777778
Precision:  0.7868686868686869
Recall:  0.7753490790255496
```

```
F1-Score:  0.7769471083987214

Support Vector Machine Model Evaluation - Validation Set
--------------------------------------------------------
Accuracy:  0.7685185185185185
Precision:  0.7707892882311488
Recall:  0.7667815844286432
F1-Score:  0.7676252881931805

Support Vector Machine Model Evaluation - Test Set
--------------------------------------------------
Accuracy:  0.7407407407407407
Precision:  0.7743517430473953
Recall:  0.7617860347439801
F1-Score:  0.7416782333932291
```

[55]:
```
from sklearn.svm import SVC

svc_params = {
    'C': [0.001, 0.01, 0.1, 1, 2,5],
    'kernel': ['poly', 'rbf', 'sigmoid'],
    'degree': [2, 3, 4, 5, 10],
    'gamma': ['scale', 'auto'] + [ 0.001, 0.01, 0.1, 1, 10, 100]
}

svc_grid_search = RandomizedSearchCV(SVC(random_state = 33, probability=True),
                                     param_distributions=svc_params,
                                     cv=pds,
                                     n_jobs=-1,
                                     verbose=2)
svc_grid_search.fit(pd.concat([X_train, X_val]),
                    pd.concat([y_train, y_val]))
svc = svc_grid_search.best_estimator_

print("Best Hyperparameters for Support Vector Machine: ", svc)
print("Best Score: ", svc_grid_search.best_score_)

y_pred_train_svc = svc.predict(X_train)
y_pred_val_svc = svc.predict(X_val)
y_pred_test_svc = svc.predict(X_test)

acc_train_svc, prec_train_svc, rec_train_svc, f1_train_svc =␣
 ↪compute_evaluation_metrics(y_train, y_pred_train_svc)
acc_val_svc, prec_val_svc, rec_val_svc, f1_val_svc =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_svc)
```

```
acc_test_svc, prec_test_svc, rec_test_svc, f1_test_svc =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_svc)

print('Support Vector Machine Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_svc)
print('Precision: ', prec_train_svc)
print('Recall: ', rec_train_svc)
print('F1-Score: ', f1_train_svc)
print()

print('Support Vector Machine Model Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_svc)
print('Precision: ', prec_val_svc)
print('Recall: ', rec_val_svc)
print('F1-Score: ', f1_val_svc)
print()

print('Support Vector Machine Model Evaluation - Test Set')
print('-----------------------------------------------')
print('Accuracy: ', acc_test_svc)
print('Precision: ', prec_test_svc)
print('Recall: ', rec_test_svc)
print('F1-Score: ', f1_test_svc)
print()
```

```
Fitting 1 folds for each of 10 candidates, totalling 10 fits
Best Hyperparameters for Support Vector Machine:  SVC(C=2, degree=10, gamma=1,
probability=True, random_state=33)
Best Score:  0.7592592592592593
Support Vector Machine Model Evaluation - Training Set
-----------------------------------------------------
Accuracy:  0.7870370370370371
Precision:  0.7868270728751948
Recall:  0.7872379254732196
F1-Score:  0.7866101234514952

Support Vector Machine Model Evaluation - Validation Set
-----------------------------------------------------
Accuracy:  0.8518518518518519
Precision:  0.8517991812109459
Recall:  0.8526395173453997
F1-Score:  0.8520995912300261

Support Vector Machine Model Evaluation - Test Set
-----------------------------------------------
```

```
Accuracy:  0.8333333333333334
Precision:  0.8294483294483294
Recall:  0.8342526808172631
F1-Score:  0.8306837979094076
```

Base Model Performance:

The base SVM model showed the following performance on the training, validation, and test sets:
- Training Set: Accuracy of 77.78%, precision of 78.69%, recall of 77.53%, and F1-Score of 77.69%.
- alidation Set: Accuracy of 76.85%, precision of 77.08%, recall of 76.68%, and F1-Score of 76.76%.
- Test Set: Accuracy of 74.07%, precision of 77.44%, recall of 76.18%, and F1-Score of 74.17%.

Impact of Hyperparameters on SVM Performance:

1. C (Regularization Parameter): Lower C values increase the regularization strength, which may lead to a simpler decision boundary, potentially underfitting the data. This can result in lower training set accuracy but can be beneficial for generalization. Higher C values reduce the regularization effect, allowing the SVM to fit the training data more closely. This can result in higher training set accuracy but may risk overfitting, as the model might capture noise in the data. Additionally, as we kept the outliers in the dataset, a higher hyperparameter was ideal as we chose to retain the outliers.

2. Degree (for Polynomial Kernel): For polynomial kernels, the 'degree' hyperparameter controls the degree of the polynomial used to separate data points. A lower degree may lead to simpler decision boundaries which may lead to underfitting the dataset, while a higher degree can create more complex decision boundaries. The choice depends on the dataset's characteristics.

3. Gamma (Kernel Coefficient): The 'gamma' hyperparameter controls the shape of the decision boundary. A low 'gamma' leads to a wider, more flexible decision boundary, while a high 'gamma' leads to a narrower, more rigid boundary. Low 'gamma' values might lead to underfitting, while high 'gamma' values could cause overfitting.

4. Probability Estimation (probability): Enabling probability estimation can be computationally expensive but can provide probability scores in addition to class predictions. Further, we mentioned probability true, as we would be applying soft voting classifier in the ensemble model.

5. Impact on Performance: After performing randomized search and finding the best hyperparameters (C=2, degree=10, gamma=1, probability=True), the SVM model's performance improved significantly.

Performance Post-Tuning:

The tuned SVM model showed improved performance: - Training Set: Accuracy increased to 78.70%, precision remained high at 78.68%, and recall improved to 78.72%. The F1-Score also increased to 78.66%. - Validation Set: Accuracy increased to 85.19%, precision to 85.18%, recall to 85.26%, and the F1-Score improved to 85.21%. - Test Set: Accuracy improved to 83.33%, precision was 82.94%, recall increased to 83.43%, and the F1-Score improved to 83.07%.

Discussion:

Proper hyperparameter tuning resulted in a better balance between bias and variance, leading to improved accuracy, precision, recall, and F1-Score, particularly on the validation and test sets.

0.7 C. Random Forest classifier (also analyze feature importance); hyperparameters to explore: the number of trees, max depth, the minimum number of samples required to split an internal node, the minimum number of samples required to be at a leaf node

[56]:
```
from pickle import NONE
from sklearn.ensemble import RandomForestClassifier
rf_orig=RandomForestClassifier(max_features= None)
rf_orig.fit(X=X_train, y=y_train)

y_pred_train_rf = rf_orig.predict(X_train)
y_pred_val_rf = rf_orig.predict(X_val)
y_pred_test_rf = rf_orig.predict(X_test)

acc_train_rf, prec_train_rf, rec_train_rf, f1_train_rf =␣
 ↪compute_evaluation_metrics(y_train, y_pred_train_rf)
acc_val_rf, prec_val_rf, rec_val_rf, f1_val_rf =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_rf)
acc_test_rf, prec_test_rf, rec_test_rf, f1_test_rf =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_rf)

print('Random Forest Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_rf)
print('Precision: ', prec_train_rf)
print('Recall: ', rec_train_rf)
print('F1-Score: ', f1_train_rf)
print()

print('Random Forest Model Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_rf)
print('Precision: ', prec_val_rf)
print('Recall: ', rec_val_rf)
print('F1-Score: ', f1_val_rf)
print()

print('Random Forest Model Evaluation - Test Set')
print('-----------------------------------------------')
print('Accuracy: ', acc_test_rf)
print('Precision: ', prec_test_rf)
print('Recall: ', rec_test_rf)
print('F1-Score: ', f1_test_rf)
print()
```

```
Random Forest Model Evaluation - Training Set
-----------------------------------------------------
```

```
Accuracy:  1.0
Precision:  1.0
Recall:  1.0
F1-Score:  1.0

Random Forest Model Evaluation - Validation Set
--------------------------------------------------
Accuracy:  0.7870370370370371
Precision:  0.7856164583151207
Recall:  0.7851325145442791
F1-Score:  0.7840909090909091

Random Forest Model Evaluation - Test Set
--------------------------------------------------
Accuracy:  0.7870370370370371
Precision:  0.7922829950693728
Recall:  0.7913890237265199
F1-Score:  0.7852688374427504
```

[57]:
```python
from sklearn.ensemble import RandomForestClassifier

rf_params = {
    'n_estimators': [50, 100, 150, 200],  # Number of trees in the forest
    'max_depth': [None, 6, 8, 10, 15, 20],  # Max depth of each tree
    'min_samples_split': [2, 5, 8, 10],  # Minimum number of samples required␣
 ↪to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be␣
 ↪at a leaf node
    'criterion' :['gini', 'entropy'],
    'max_features': ['auto', 'sqrt', 'log2']
}

grid_search_rf = RandomizedSearchCV(RandomForestClassifier(random_state=33),
                                    param_distributions=rf_params,
                                    cv=pds,
                                    n_jobs=-1)
grid_search_rf.fit(pd.concat([X_train, X_val]),
                   pd.concat([y_train, y_val]))
rf = grid_search_rf.best_estimator_

print("Best Hyperparameters for Random Forest: ", rf)
print("Best Score: ", grid_search_rf.best_score_)

y_pred_train_rf = rf.predict(X_train)
y_pred_val_rf = rf.predict(X_val)
y_pred_test_rf = rf.predict(X_test)
```

```
acc_train_rf, prec_train_rf, rec_train_rf, f1_train_rf = compute_evaluation_metrics(y_train, y_pred_train_rf)
acc_val_rf, prec_val_rf, rec_val_rf, f1_val_rf = compute_evaluation_metrics(y_val, y_pred_val_rf)
acc_test_rf, prec_test_rf, rec_test_rf, f1_test_rf = compute_evaluation_metrics(y_test, y_pred_test_rf)

print('Random Forest Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_rf)
print('Precision: ', prec_train_rf)
print('Recall: ', rec_train_rf)
print('F1-Score: ', f1_train_rf)
print()

print('Random Forest Model Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_rf)
print('Precision: ', prec_val_rf)
print('Recall: ', rec_val_rf)
print('F1-Score: ', f1_val_rf)
print()

print('Random Forest Model Evaluation - Test Set')
print('-----------------------------------------------')
print('Accuracy: ', acc_test_rf)
print('Precision: ', prec_test_rf)
print('Recall: ', rec_test_rf)
print('F1-Score: ', f1_test_rf)
print()
```

```
Best Hyperparameters for Random Forest:  RandomForestClassifier(max_depth=10,
max_features='log2', min_samples_leaf=4,
                       min_samples_split=8, n_estimators=150, random_state=33)
Best Score:  0.8148148148148148
Random Forest Model Evaluation - Training Set
-----------------------------------------------------
Accuracy:  0.8672839506172839
Precision:  0.8708926275297957
Recall:  0.8659387997623291
F1-Score:  0.8669266518151403

Random Forest Model Evaluation - Validation Set
-----------------------------------------------------
Accuracy:  0.9444444444444444
Precision:  0.9484745712943387
```

```
Recall:  0.9422969187675069
F1-Score:  0.9439388423734983

Random Forest Model Evaluation - Test Set
-------------------------------------------------
Accuracy:  0.7685185185185185
Precision:  0.7890964053754752
Recall:  0.785990615300884
F1-Score:  0.7690648394873746
```

Base Model Performance: The base Random Forest model exhibits high accuracy and F1-Score on the training set, but it's important to evaluate how it performs on the validation and test sets to determine whether it generalizes well.

- Training Set: The base model achieves perfect accuracy, precision, recall, and F1-Score (all 1.0). This suggests that it's highly overfitting the training data, capturing noise, and may not generalize well to new data.
- Validation Set: On the validation set, the base model's performance is still relatively good with an accuracy of 78.70% and F1-Score of 78.41%. While it doesn't suffer from severe overfitting, there's room for improvement in generalization.
- Test Set: The base model's performance on the test set is similar to the validation set, indicating that it generalizes reasonably well. However, further improvements are possible.

Performance After Hyperparameter Tuning:

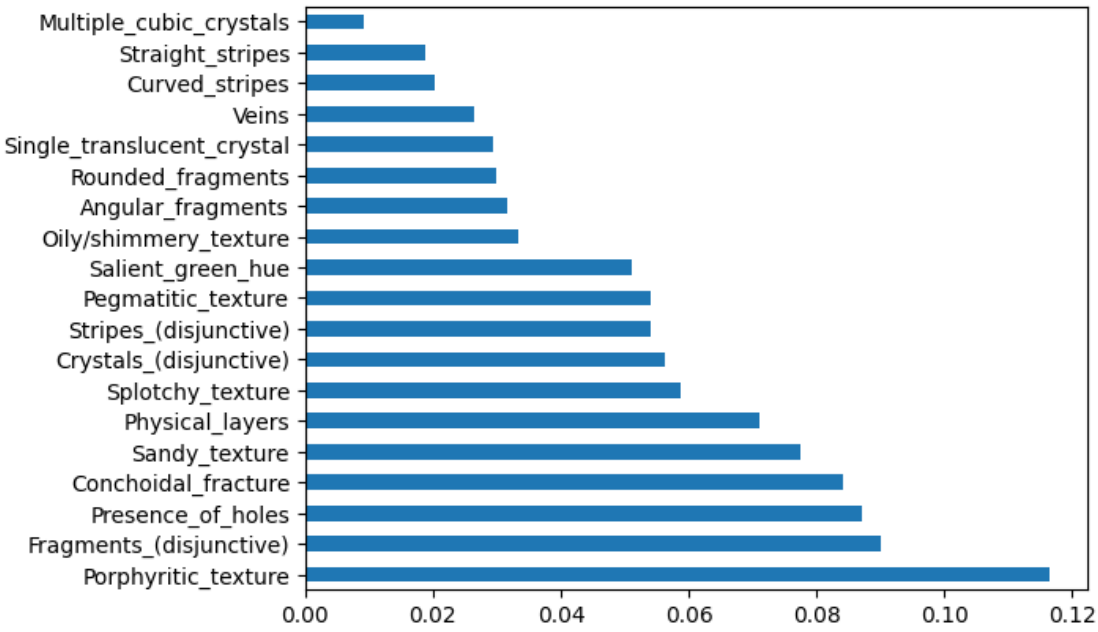
- Training Set: Post-tuning, the training set accuracy decreased from 100% to 86.73%, indicating reduced overfitting. Precision, recall, and F1-Score also show improvement, but not at the expense of overfitting.
- Validation Set: The validation set performance significantly improved, with an accuracy of 94.44% and an F1-Score of 94.39%. This indicates that the hyperparameter tuning led to a model that generalizes better to unseen data.
- Test Set: The test set accuracy decreased slightly, but the model's F1-Score remains improved at 76.91%. It demonstrates that the model is better at making accurate predictions on new data.

Generalization and Model Complexity:

The hyperparameter-tuned model has better generalization abilities, as seen in the improved validation set performance. A good combination of hyperparameters balances model complexity and performance, leading to a more robust and reliable model.

[58]:
```
feat_importances = pd.Series(rf.feature_importances_, index=X_train.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

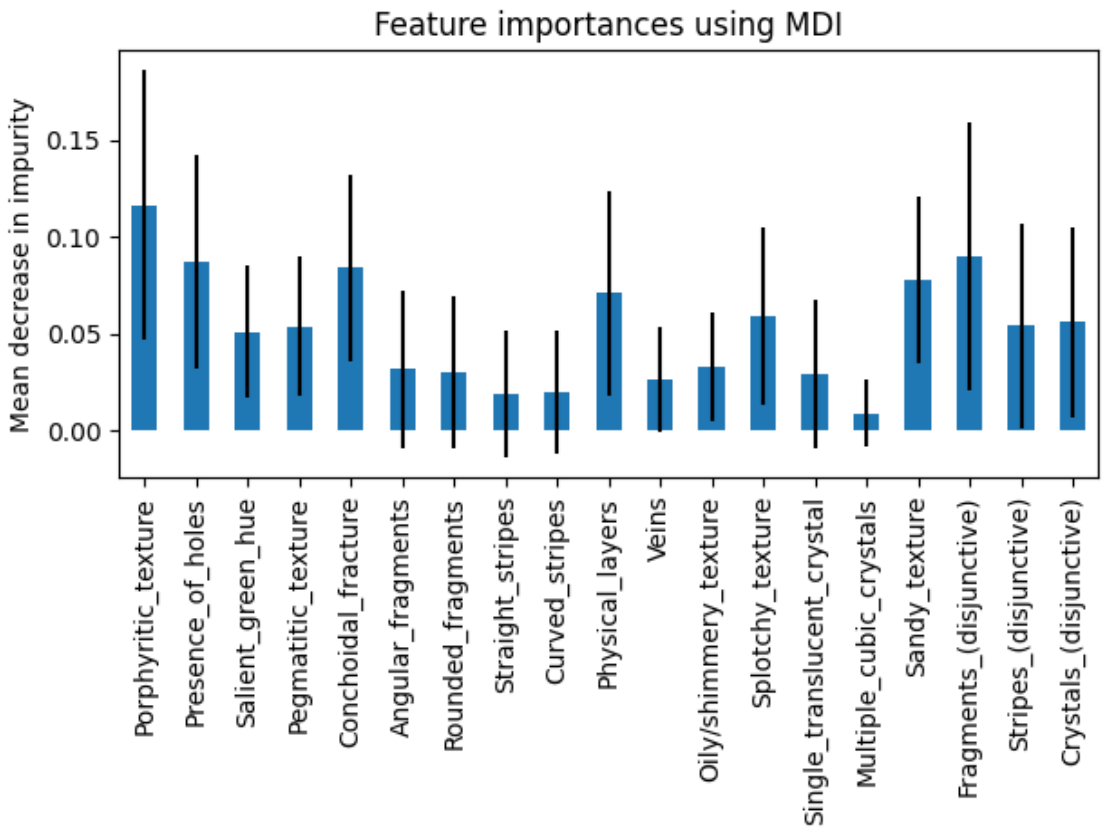
[58]:
```
<Axes: >
```



[59]:
```
import time

import numpy as np

start_time = time.time()
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)
elapsed_time = time.time() - start_time

print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds")
```

```
Elapsed time to compute the importances: 0.032 seconds
```

[60]:
```
import pandas as pd

forest_importances = pd.Series(importances, index=df_attributes.columns)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



1 Part 5: Combine your classifiers into an ensemble and try to outperform each individual classifier on the validation set (try to get above 80% accuracy). Once you have found a good one, try it on the test set. Describe and discuss your findings.

[61]:
```
from sklearn.ensemble import VotingClassifier

# Create a list of tuples, each containing a classifier's name and the trained␣
 ↪model
ensemble_estimators = [
                        ('Logistic Regression', lr),
                        ('SVM Classifier', svc),
                        ('Random Forest', rf)
                      ]

ensemble = VotingClassifier(estimators=ensemble_estimators, voting='soft')

ensemble.fit(X_train, y_train)
```

```
y_pred_train_ensemble = ensemble.predict(X_train)
y_pred_val_ensemble = ensemble.predict(X_val)
y_pred_test_ensemble = ensemble.predict(X_test)

acc_train_ensemble, prec_train_ensemble, rec_train_ensemble, f1_train_ensemble␣
 ↪= compute_evaluation_metrics(y_train, y_pred_train_ensemble)
acc_val_ensemble, prec_val_ensemble, rec_val_ensemble, f1_val_ensemble =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_ensemble)
acc_test_ensemble, prec_test_ensemble, rec_test_ensemble, f1_test_ensemble =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_ensemble)

print('Voting Ensemble Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_ensemble)
print('Precision: ', prec_train_ensemble)
print('Recall: ', rec_train_ensemble)
print('F1-Score: ', f1_train_ensemble)
print()

print('Voting Ensemble Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_ensemble)
print('Precision: ', prec_val_ensemble)
print('Recall: ', rec_val_ensemble)
print('F1-Score: ', f1_val_ensemble)
print()

print('Voting Ensemble Model Evaluation - Test Set')
print('-----------------------------------------------')
print('Accuracy: ', acc_test_ensemble)
print('Precision: ', prec_test_ensemble)
print('Recall: ', rec_test_ensemble)
print('F1-Score: ', f1_test_ensemble)
print()
```

```
Voting Ensemble Model Evaluation - Training Set
-----------------------------------------------------
Accuracy:  0.7901234567901234
Precision:  0.7931944700719403
Recall:  0.7886045327221799
F1-Score:  0.7896235848249162

Voting Ensemble Evaluation - Validation Set
-----------------------------------------------------
Accuracy:  0.8148148148148148
Precision:  0.8175748328974134
```

```
Recall:  0.81342383107089
F1-Score:  0.814663348840564

Voting Ensemble Model Evaluation - Test Set
-------------------------------------------------
Accuracy:  0.7962962962962963
Precision:  0.8124338624338625
Recall:  0.8112984128385037
F1-Score:  0.7975212002609263
```

[62]:
```
from sklearn.ensemble import StackingClassifier

stack = StackingClassifier(estimators=ensemble_estimators,
                           final_estimator=LogisticRegression())

stack.fit(X_train, y_train)

y_pred_train_stack = stack.predict(X_train)
y_pred_val_stack = stack.predict(X_val)
y_pred_test_stack = stack.predict(X_test)

acc_train_stack, prec_train_stack, rec_train_stack, f1_train_stack =␣
 ↪compute_evaluation_metrics(y_train, y_pred_train_stack)
acc_val_stack, prec_val_stack, rec_val_stack, f1_val_stack =␣
 ↪compute_evaluation_metrics(y_val, y_pred_val_stack)
acc_test_stack, prec_test_stack, rec_test_stack, f1_test_stack =␣
 ↪compute_evaluation_metrics(y_test, y_pred_test_stack)

print('Stacking Ensemble Model Evaluation - Training Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_train_stack)
print('Precision: ', prec_train_stack)
print('Recall: ', rec_train_stack)
print('F1-Score: ', f1_train_stack)
print()

print('Stacking Ensemble Model Evaluation - Validation Set')
print('-----------------------------------------------------')
print('Accuracy: ', acc_val_stack)
print('Precision: ', prec_val_stack)
print('Recall: ', rec_val_stack)
print('F1-Score: ', f1_val_stack)
print()

print('Stacking Ensemble Model Evaluation - Test Set')
print('-----------------------------------------------')
```

```
print('Accuracy: ', acc_test_stack)
print('Precision: ', prec_test_stack)
print('Recall: ', rec_test_stack)
print('F1-Score: ', f1_test_stack)
print()
```

```
Stacking Ensemble Model Evaluation - Training Set
--------------------------------------------------
Accuracy:  0.7993827160493827
Precision:  0.7993589743589743
Recall:  0.7995968084203379
F1-Score:  0.799430145061213

Stacking Ensemble Model Evaluation - Validation Set
--------------------------------------------------
Accuracy:  0.8055555555555556
Precision:  0.8094338240679705
Recall:  0.8036199095022626
F1-Score:  0.8050925925925926

Stacking Ensemble Model Evaluation - Test Set
--------------------------------------------------
Accuracy:  0.8333333333333334
Precision:  0.836248012718601
Recall:  0.8423061647764882
F1-Score:  0.832841971615791
```

Findings:

1. Ensemble vs. Individual Classifiers: The ensemble methods, both Voting and Stacking is better at generalization. >- Training Set: The Voting Ensemble model achieved an accuracy of 79.01% on the training set, with a balanced precision, recall, and F1-Score around 79%. >- Validation Set: The ensemble performs well on the validation set with an accuracy of 81.48% and balanced precision, recall, and F1-Score around 81%. >- Test Set: The performance on the test set is consistent, with an accuracy of 79.63% and similar balanced precision, recall, and F1-Score around 80%.

   Random Forest, while the best individual classifier, does not generalize as well as the ensemble methods. Ensembles can capture the strengths of different base classifiers and mitigate their weaknesses.

2. Voting vs. Stacking: The Stacking Ensemble demonstrates a slight performance advantage over the Voting Ensemble on the test set. This indicates that the stacking approach of training a meta-model on top of base models provides additional benefits in terms of generalization.

3. Balanced Performance: Notably, both ensemble methods exhibit balanced performance in terms of precision, recall, and F1-Score, indicating that they can make accurate predictions without bias towards false positives or false negatives.

4. Generalization: The Voting Classifier and Stacking Ensemble demonstrates better generalization to the test set, as seen in the higher accuracy and F1-Score. This suggests that the ensemble approach effectively combines the strengths of the base models for improved predictive performance on new data.

5. Role of Individual Classifiers: While individual classifiers (Logistic Regression, SVM, and Random Forest) may have their limitations, the ensemble methods can compensate for these limitations by combining their outputs. The ensemble methods harness the diversity of individual models to achieve improved predictive accuracy.