In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
df = pd.read_csv ("glass.csv")
df.head()
```

Out[4]:

	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.00.1	1.1
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.00	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.00	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.00	1
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.00	1
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	1

In [11]:
```
df.shape
# there are 11 columns and 213 rows are present in the data set
```

Out[11]:
```
(213, 11)
```

In [12]:
```
df.columns
# columns names are not present in the dataset.
#  but given in the project, so first we have to assign coulmn names into the data set
```

Out[12]:
```
Index(['1', '1.52101', '13.64', '4.49', '1.10', '71.78', '0.06', '8.75',
       '0.00', '0.00.1', '1.1'],
      dtype='object')
```

In [13]:
```
column = ["id number", "refractive index","sodium","magnesium", "aluminium","silicon","potassium","calcium","barium","iron'
column
```

Out[13]:
```
['id number',
 'refractive index',
 'sodium',
 'magnesium',
 'aluminium',
 'silicon',
 'potassium',
 'calcium',
 'barium',
 'iron',
 'type of glass']
```

In [14]:
```
# so here above we are having all eleven coumn names and we have to assign them into the data set.
```

In [14]:
```
df.columns=column
```

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

Out[15]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

In [16]:
```
df.head(2)
```

Out[16]:

	id number	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron	type of glass
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1

In [18]:
```
df.columns.unique()
# here as we assign you can se that the same result is occured, that means there there is not repetation of any column in t
```

Out[18]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

In [19]:
```
df.columns.nunique()
# the total no. of cloumns are same as we can check earlier in df.shape
```

Out[19]:
```
11
```

In [20]:
```
df.dtypes
# here we can see that all the columns present are in - [ int64, float64 ] only.
```

Out[20]:
```
id number             int64
refractive index    float64
sodium              float64
magnesium           float64
aluminium           float64
silicon             float64
potassium           float64
calcium             float64
barium              float64
iron                float64
type of glass         int64
dtype: object
```

In [21]:
```
df.info()
#  here we can see that
# 1) total number for columns present : 11
# 2) total number of rows presnet : 213
# 3) total "data types present in data set" : 2 (i.e "int64 & float64")
#  out of which   9 columns of - float64
#                 2 column of - int64
# 4)NO NULL VALUES are present in our dataset.
# 5) No integer or float columns are in object data type, so we can say that there is no whitespaces in our dataset as null
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213 entries, 0 to 212
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id number         213 non-null    int64
 1   refractive index  213 non-null    float64
 2   sodium            213 non-null    float64
 3   magnesium         213 non-null    float64
 4   aluminium         213 non-null    float64
 5   silicon           213 non-null    float64
 6   potassium         213 non-null    float64
 7   calcium           213 non-null    float64
 8   barium            213 non-null    float64
 9   iron              213 non-null    float64
 10  type of glass     213 non-null    int64
dtypes: float64(9), int64(2)
memory usage: 18.4 KB
```

In [ ]:

CHECKING NULL VALUES

==

In [24]:
```
df.isnull().sum()
# Here also it is conformed that there are no NULL VALUES are present in our dataset.
```

Out[24]:
```
id number           0
refractive index    0
sodium              0
magnesium           0
aluminium           0
silicon             0
potassium           0
calcium             0
barium              0
iron                0
type of glass       0
dtype: int64
```

In [ ]:

CHECKING UNIQUE VALUES PRENSENT IN DATASET & UNIVARIATE ANALYSIS

==

In [26]:
```
df.head(3)
```

Out[26]:

	id number	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron	type of glass
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1

In [ ]:

In [27]:
```
df['id number'].nunique()
# there is no repetation in 'id-number' colummn
```

Out[27]:
```
213
```

In [29]:
```
df['refractive index'].nunique()
# there are some of the values are repeteted in this columns.
```

Out[29]:
```
177
```

In [30]:
```
df['sodium'].nunique()
# here also we can see that few of the values are repeating.
```

Out[30]:
```
142
```

In [32]:
```
df['magnesium'].nunique()
# out of 213 only 93 values are unique.
```

Out[32]:
```
93
```

In [34]:
```
df['aluminium'].nunique()
# here also some values are repeating.
```

Out[34]:
```
117
```

In [35]:
```
df['silicon'].nunique()
# same in silicon.
```

Out[35]:
```
132
```

In [36]:
```
df['potassium'].nunique()
# out of 213 values only 65 values are unique.
```

Out[36]:
```
65
```

In [46]:
```
df['potassium'].value_counts().head(10)
# highest entries (30) on 'potassium value' = 0.00
# then after most of the entries are lying in between 'potassim value' = 0.56 - 0.60
```

Out[46]:
```
0.00    30
0.57    12
0.56    11
0.60    11
0.58    10
0.64     8
0.61     8
0.59     7
0.55     6
0.54     6
Name: potassium, dtype: int64
```

In [ ]:

In [37]:
```
df['calcium'].nunique()
```

Out[37]:
```
143
```

In [38]:
```
df['barium'].nunique()
# out of 213 only 34 vlaues are unique.
```

Out[38]:
```
34
```

In [45]:
```
df['barium'].value_counts().head(10)
# here in the barium column, out of 213 etnries , 175 entries are having 'barium value'= 0.00
```

Out[45]:
```
0.00    175
0.64      2
1.57      2
0.09      2
1.59      2
0.11      2
3.15      1
0.81      1
1.64      1
1.06      1
Name: barium, dtype: int64
```

In [ ]:

In [ ]:

In [39]:
```
df['iron'].nunique()
# the most repeating values are present in iron column.
```

Out[39]:
```
32
```

In [43]:
```
df['iron'].value_counts().head(10)
# here in iron column we can find that out of 213 entries, 143 entries are having 'iron value'=0.00
```

Out[43]:
```
0.00    143
0.24      7
0.17      7
0.09      6
0.10      5
0.11      4
0.16      3
0.28      3
0.12      3
0.22      3
Name: iron, dtype: int64
```

In [ ]:

In [ ]:

In [28]:
```
df['type of glass'].unique()
# the types of glass containing six categories from 1-7
# that means it is a categorical column.
# this is the only categorical column present in the dataset.
```

Out[28]:
```
array([1, 2, 3, 5, 6, 7], dtype=int64)
```

In [40]:
```
df['type of glass'].nunique()
```

Out[40]:
```
6
```

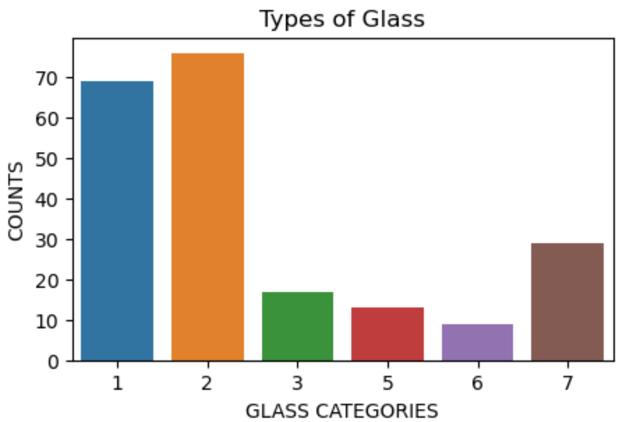
In [41]:
```
df['type of glass'].value_counts()
# here we can find that most of the values are lying in 1,2, & 7 types .
```

Out[41]:
```
2    76
1    69
7    29
3    17
5    13
6     9
Name: type of glass, dtype: int64
```

In [57]:
```
plt.figure (figsize = (5,3), facecolor = "white")
plt.title('Types of Glass')
sns.countplot(x='type of glass', data = df)
plt.xlabel('GLASS CATEGORIES', fontsize=10)
# plt.xticks(rotation=30, ha = 'right')
plt.ylabel('COUNTS')
# plt.yticks(rotation=30, ha = 'right')

# Here we can see that the most of counts are present in category 1 ,2 & 7.
# as above also we find the same in numerical form.
# here we know that from category 1-4 are WINDOW GLASS, & and 5-7 are NON WINDOW GLASS
# so here we find that most of the values are in 'WINDOW GLASS CATEGORY'
```

Out[57]:
```
Text(0, 0.5, 'COUNTS')
```



In [ ]:

BIVARIATE ANALYSIS
==

In [58]:
```
df.columns
```

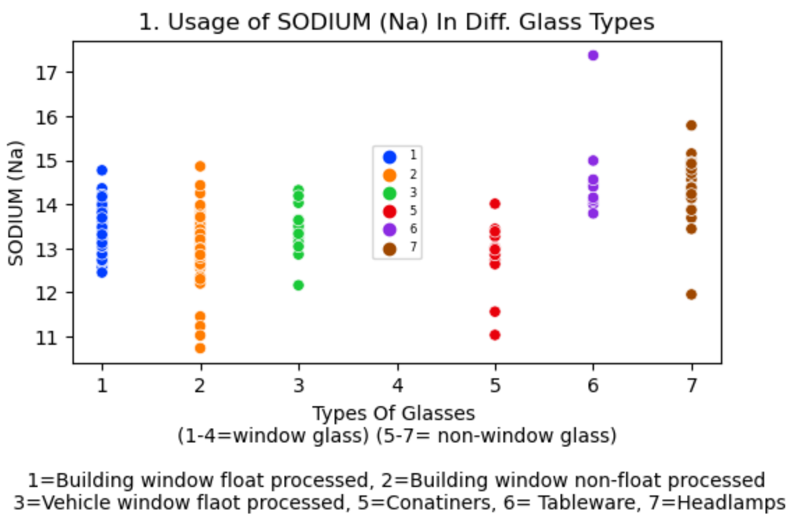
Out[58]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

1) ANALYSING SODIUM (Na) FOR DIFFERENT CATEGORIES

In [116]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('1. Usage of SODIUM (Na) In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'sodium', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('SODIUM (Na)')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'SODIUM (Na) INGREDIANT QUANTITY' for different types of glasses.
# so form the below graph we can say that 'the quantity of SODIUM (Na)' used for 'window glass' & 'non winow glass'. is in
# here we can see that in category-2 and 5 the lowest level of SODIUM (Na) is upto 10
# and it is Highest in that category 6 (i.e above 17)

#  CONCLUSION ==>  Quantity of mixing 'SODIUM (Na)' for both 'window & non -window glasses' is in between 12-15 range
#  but we can also say that that usage of SODIUM (Na) is higher as compared to 'window glasses' categories.
```

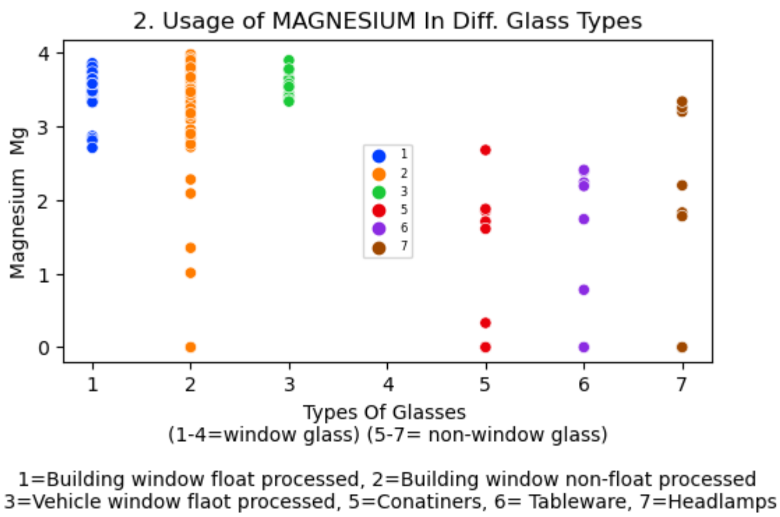


2) ANALYSING MAGNESIUM (Mg) FOR DIFFERENT CATEGORIES

In [117]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('2. Usage of MAGNESIUM In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'magnesium', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Magnesium  Mg')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'MAGNESIUM INGREDIANT QUANTITY' for different types of glasses.
# so form the below graph we can say that 'the quantity of Magnesium' used for 'window glass' is higher as compared to
#                                                                                                    'non winow glass'.

#  CONCLUSION ==>  Quantity of mixing 'MAGNESIUM' for 'window glass' is higher as compared to 'non winow glass'.
```

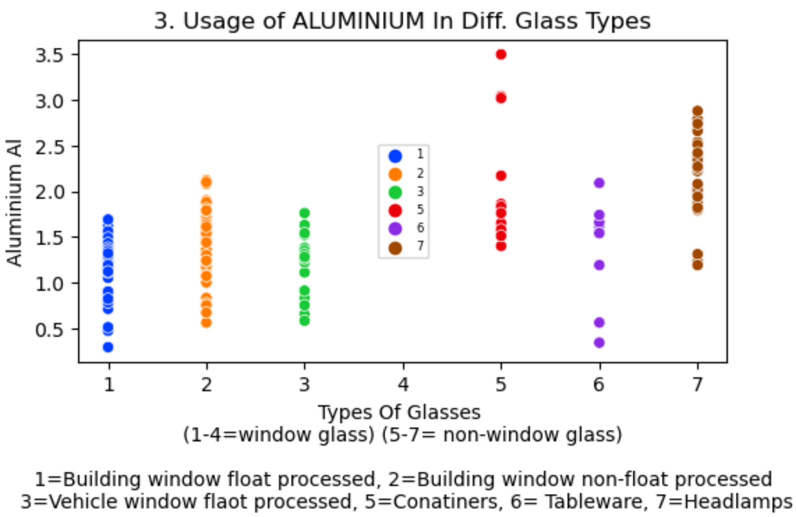


In [ ]:

3) ANALYSING ALUMINIUM (Al) FOR DIFFERENT CATEGORIES

In [118]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('3. Usage of ALUMINIUM In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'aluminium', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Aluminium Al')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'ALUMINIUM INGREDIANT QUANTITY' for different types of glasses.
# so from the below graph we can say that 'the quantity of aluminium' in making of 'window glass' is higher as LOWERED to
#                                                                                                   'non winow glass'.
# here we also find that 'the aluminium quantity' is higehr in Category 5 & 7
#  CONCLUSION ==>  Quantity of mixing 'ALUMINUM' for 'window glass' is LOWERED as compared to 'non winow glass'.
```

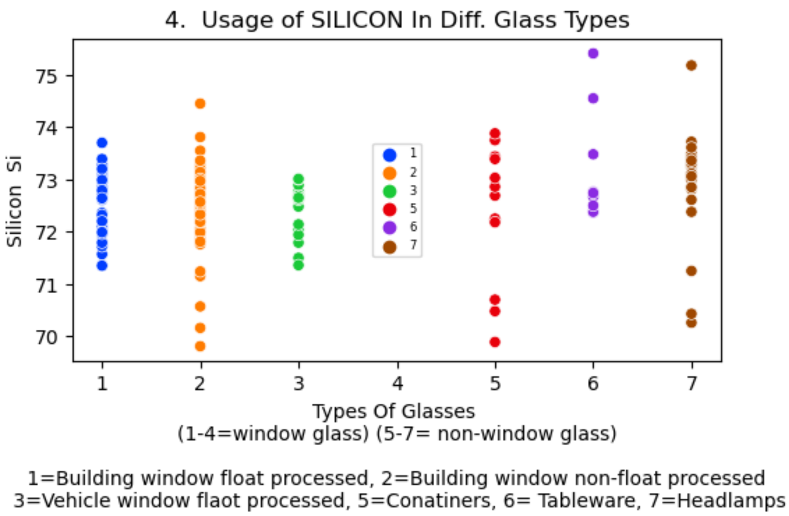


In [ ]:

4) ANALYSING SILICON (Si) FOR DIFFERENT CATEGORIES

In [119]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('4.  Usage of SILICON In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'silicon', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Silicon  Si')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'SILICON INGREDIANT QUANTITY' for different types of glasses.
# so from the below graph we can say that 'the quantity of SILICON' in making of 'window glass' & 'non winow glass'
#  is HIGHER.
#                                                                                          .
# here we also find that 'the SILICON quantity' is higehr in Category 2,6 & 7
#  CONCLUSION ==>  Quantity of mixing 'SILICON' for 'window glass' & 'non-window glass' both is HIGHER.
#                                                              (Specifically HIGHER in category 2,6 & 7)
```



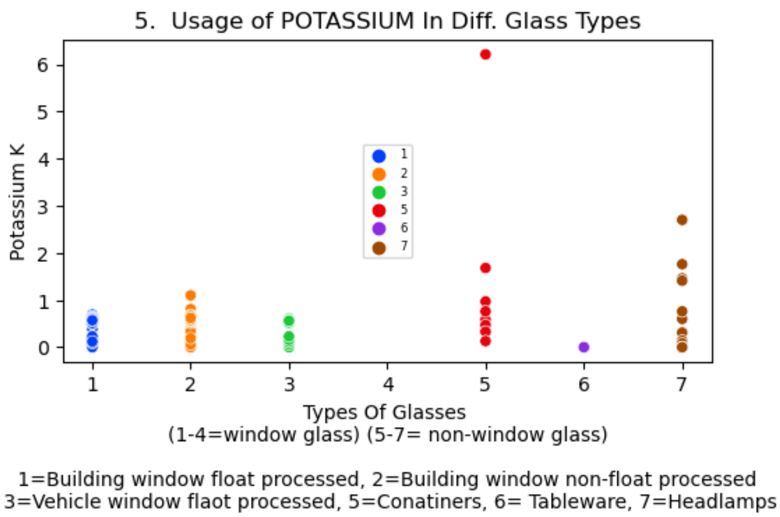
In [ ]:

5) ANALYSING POTASSIUM (K) FOR DIFFERENT CATEGORIES

In [120]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('5.  Usage of POTASSIUM In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'potassium', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Potassium K')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'POTASSIUM (K) INGREDIANT QUANTITY' for different types of glasses.
# so from the below graph we can say that 'the quantity of POTASSIUM' in making of both 'window glass' & 'non winow glass'
#  LOW . there are only few of the outliers of POTASSIUM n category 5 & 7.

#  CONCLUSION ==>  Quantity of mixing 'POTASSIUM (K)' for both 'window & non glass' is LOW.
```



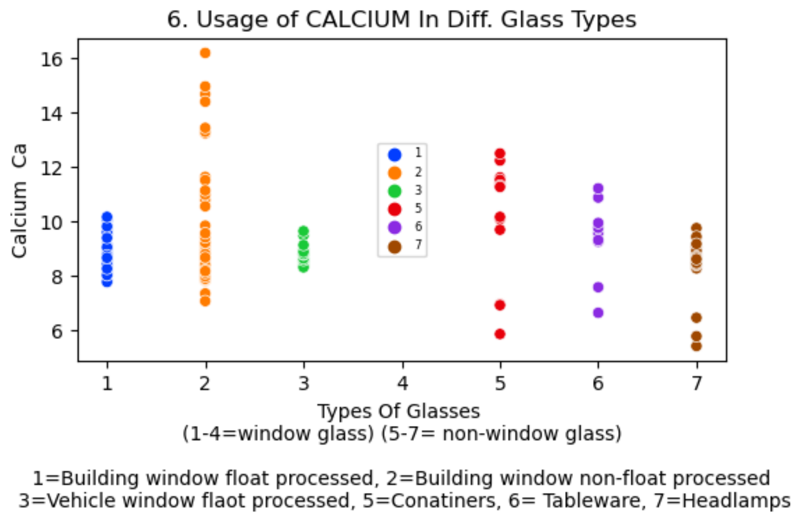
In [ ]:

6) ANALYSING CALCIUM (Ca) FOR DIFFERENT CATEGORIES

In [121]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('6. Usage of CALCIUM In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'calcium', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildi
plt.ylabel('Calcium  Ca')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'CALCIUM (Ca) INGREDIANT QUANTITY' for different types of glasses.
# so from the below graph we can say that 'the quantity of CALCIUM (Ca)' in making of 'window glass' is Average range of (7
#                                                                     but we can also see the huge hike in Category-2 (i.e upto
# & for 'non winow glass' the average quantity is upto (7-12) but in all 5,6 & 7 we can also find the lowest quantity upto
#  is HIGHER.
#                                                                                                         .

#  CONCLUSION ==>  Average Quantity of mixing 'CALCIUM (Ca)' for 'window glass' & 'non-window glass' both is in between (8-
#                                                                 (Specifically HIGHER in category 2 & LOWER in category 5, 6 &
```



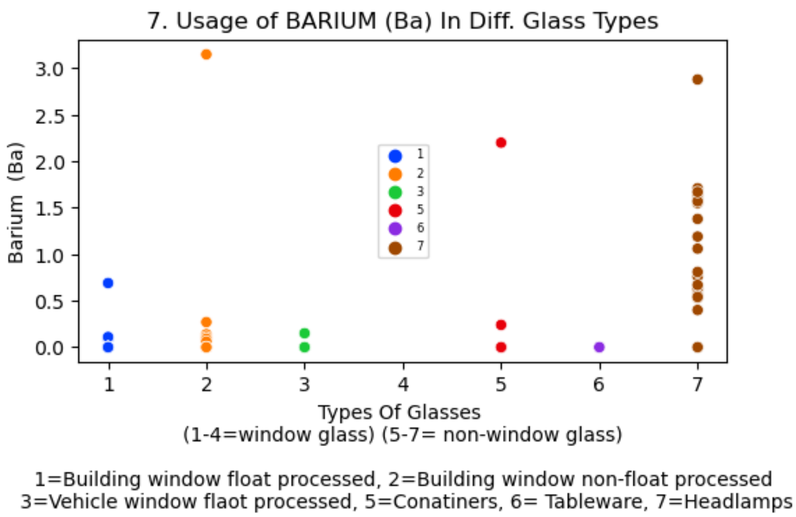
In [ ]:

7) ANALYSING BARIUM (Ba) FOR DIFFERENT CATEGORIES

In [122]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('7. Usage of BARIUM (Ba) In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'barium', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Barium  (Ba)')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'BARIUM (Ba) INGREDIANT QUANTITY' for different types of glasses.
# so from the below graph we can say that 'the quantity of BARIUM (Ba)' in making of both 'window glass' & 'non winow glass
# very LESS near about (0.0 - 0.2) . there are only few of the outliers of BARIUM (Ba) in category 2,5 & 7.

#  CONCLUSION ==>  Quantity of mixing 'BARIUM (Ba)' for both 'window & non glass' is LESS or NEGLIGIBLE.
```



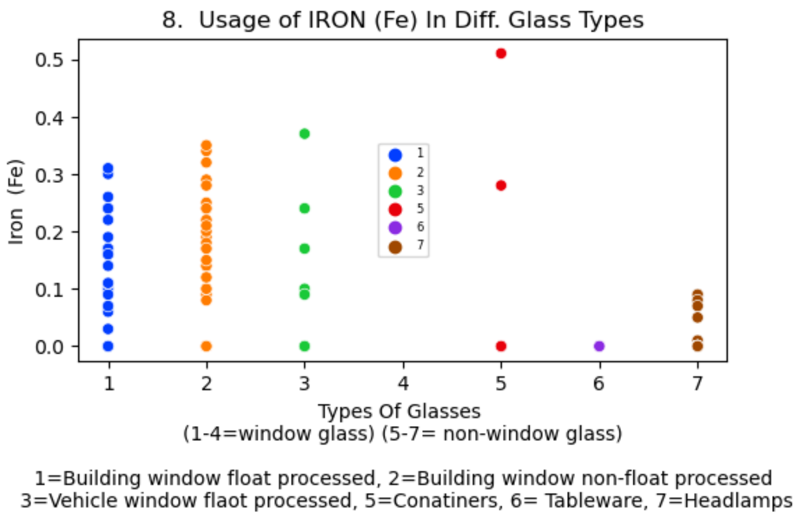
In [ ]:

8.) ANALYSING IRON (Fe) FOR DIFFERENT CATEGORIES

In [123]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('8.  Usage of IRON (Fe) In Diff. Glass Types')
sns.scatterplot (x= 'type of glass', y = 'iron', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Iron  (Fe)')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'IRON (Fe) INGREDIANT QUANTITY' for different types of glasses.
# From the below graph we can say that 'the quantity of IRON (Fe)' in making of 'window glass' verymuch HIGHER as compared
# to 'non winow glass'.
# The usage of IRON (Fe) in making of window glass (specially in category 1 & 2 is verymuch highered as compared to others)
#  we can also find only few of the outliers in the category 5 only upto 0.5

#  CONCLUSION ==>  Quantity of mixing 'IRON (Fe)' for 'window glass' is HIGHER as compared to non window glass.
#  or we can also say's that for "Building_window_float & non_float processed " usage of IRON (Fe) is very HIGHER.
```

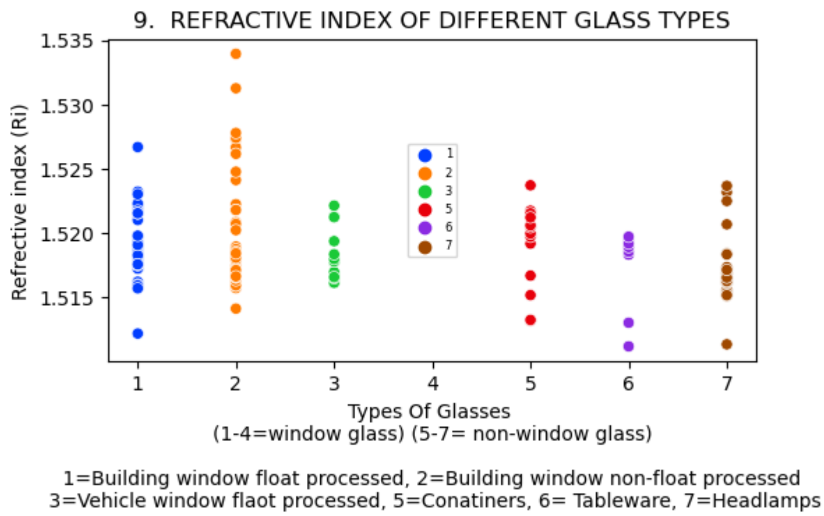


In [ ]:

9) ANALYSING REFRECTIVE INDEX (RI) FOR DIFFERENT CATEGORIES

In [18]:
```
plt.figure (figsize = (6,3), facecolor = "white")
plt.title('9.  REFRACTIVE INDEX OF DIFFERENT GLASS TYPES')
sns.scatterplot (x= 'type of glass', y = 'refractive index', hue = 'type of glass', data= df, palette = "bright")
# plt.xticks(rotation=30, ha = 'right')
plt.xlabel('Types Of Glasses \n(1-4=window glass) (5-7= non-window glass)\n \n 1=Building window float processed, 2=Buildir
plt.ylabel('Refrective index (Ri)')
plt.legend(loc= 'center', fontsize=6)
plt.show()

# Here from the below graph we find the 'Refrective index (Ri)' for different types of glasses.
# From the below graph we can say that 'the Average Refrective index (Ri)' of 'window glass'& 'non winow glass'is like sim
#  but Refrective index (Ri) of "building_window_non_float_processed_glass" (category-2) is very much HIGHER as compared to
# The Refrective index (Ri) of category 1,5,6,7 is touches the lowe lavels also.
#  CONCLUSION ==> From the below graph we can say that the AVERAGE Refrective index (Ri) of all categories liyes in betweer
#   but offcourse category-2 having HIGHER Refrective index (Ri) as compared to others.
```



In [ ]:

PLOTTING ALL COLUMNS TOGETHER
==

In [19]:
```
df.columns
```

Out[19]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

In [23]:
```
sns.pairplot ( hue = 'type of glass', data= df, palette = "bright")
plt.show()
```

In [ ]:

In [ ]:

CHECKING FOR OUTLIERS
==

In [130]:
```
df.describe()
```

Out[130]:

	id number	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron	type of glass
count	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000
mean	108.000000	1.518353	13.406761	2.676056	1.446526	72.655023	0.499108	8.957934	0.175869	0.057277	2.788732
std	61.631972	0.003039	0.818371	1.440453	0.499882	0.774052	0.653035	1.426435	0.498245	0.097589	2.105130
min	2.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000	1.000000
25%	55.000000	1.516520	12.900000	2.090000	1.190000	72.280000	0.130000	8.240000	0.000000	0.000000	1.000000
50%	108.000000	1.517680	13.300000	3.480000	1.360000	72.790000	0.560000	8.600000	0.000000	0.000000	2.000000
75%	161.000000	1.519150	13.830000	3.600000	1.630000	73.090000	0.610000	9.180000	0.000000	0.100000	3.000000
max	214.000000	1.533930	17.380000	3.980000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000	7.000000

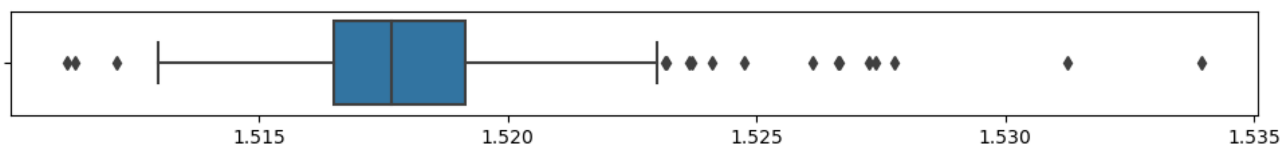
In [ ]:
```
# here as we can see in the above table, we see a huge difference between 75% & Max of some columns,"POTASSIUM","CALCIUM",
#  due to which we can assume that there may presence of outliers, so we have to check this with "BOXPLOT METHOD"

# Here we can also observed that there is huge difference between 75 Prcentile & MAX in above mentioned columns,
#  but if outliers are present then STNDARD DEVIATION should also be HIGH, but we can see that the standard deviation is
#  not high (except in CALCIUM COLUMN, little bit higher)
#  so we can't sure about the presence of outliers in the above mentioned columns also.
```

In [25]:
```
df.columns
```

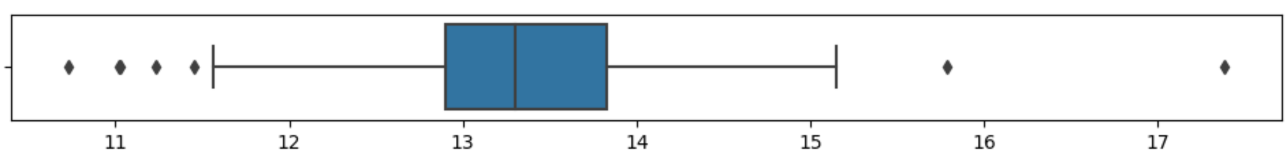
Out[25]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

In [37]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='refractive index',data=df)
plt.xlabel('\n 1. Checking Outliers in Refractive index')
plt.show()
# here we can see the presence of Outliers, but their range is very near to the maximum.
```



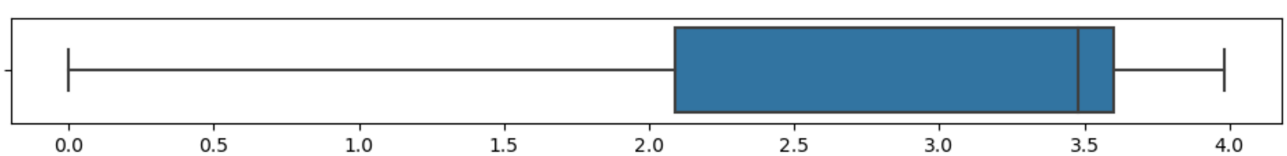
1. Checking Outliers in Refractive index

In [39]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='sodium',data=df)
plt.xlabel('\n 2. Checking Outliers in sodium')
plt.show()
# here we can see the presence of Outliers.
```



2. Checking Outliers in sodium

In [40]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='magnesium',data=df)
plt.xlabel('\n 3. Checking Outliers in Magnesium')
plt.show()
# here we can see the no presence of Outliers.
```

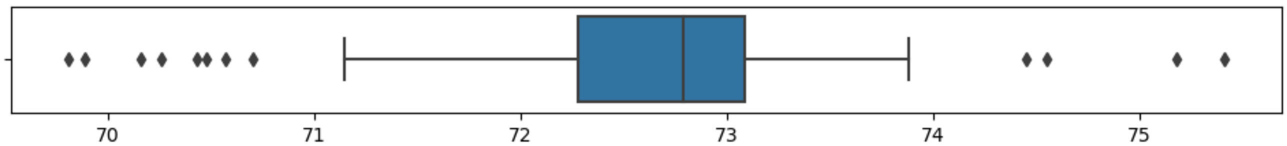


3. Checking Outliers in Magnesium

In [41]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='aluminium',data=df)
plt.xlabel('\n 4. Checking Outliers in Aluminium')
plt.show()
# here we can see the presence of Outliers.
```

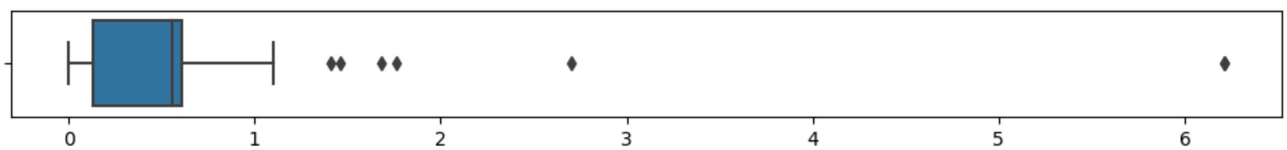
4. Checking Outliers in Aluminium

In [42]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='silicon',data=df)
plt.xlabel('\n 5. Checking Outliers in Silicon')
plt.show()
# here we can see the presence of Outliers.
```



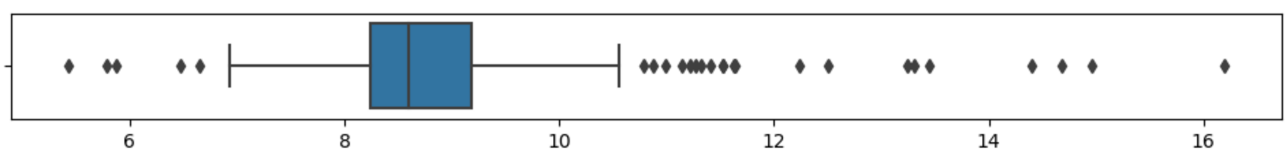
5. Checking Outliers in Silicon

In [43]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='potassium',data=df)
plt.xlabel('\n 6. Checking Outliers in Potassium')
plt.show()
# here we can see the presence of Outliers.
```



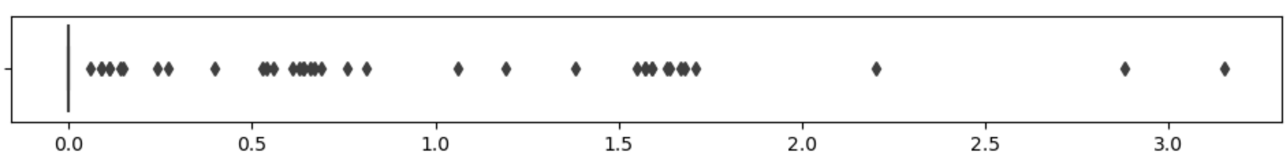
6. Checking Outliers in Potassium

In [44]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='calcium',data=df)
plt.xlabel('\n 7. Checking Outliers in Calcium')
plt.show()
# here we can see the presence of Outliers.
```



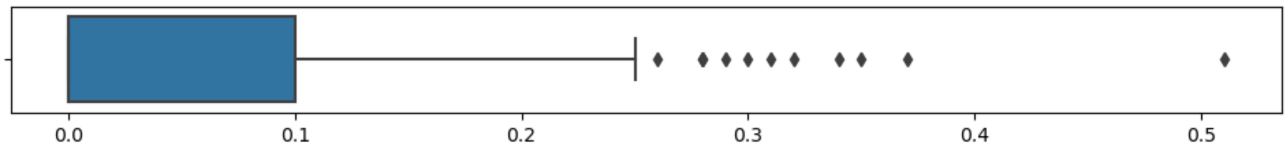
7. Checking Outliers in Calcium

In [45]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='barium',data=df)
plt.xlabel('\n 8. Checking Outliers in Barium')
plt.show()
# here we can see the presence of Outliers.
```



8. Checking Outliers in Barium

In [46]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='iron',data=df)
plt.xlabel('\n 9. Checking Outliers in Iron')
plt.show()
# here we can see the presence of Outliers.
```



In [ ]:

Removing Of OutLiers by applyin Z-Score Method
==

In [53]:
```
# We can't remove OUTLIERS from our TARGET COLUMN
```

In [54]:
```
from scipy.stats import zscore
```

In [55]:
```
z = np.abs(zscore(df))
z.head(5)

# by applying 'abs' (absolute method), we are getting
```

Out[55]:

	id number	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron	type of glass
0	1.723938	0.245101	0.591880	0.642937	0.173500	0.097091	0.029329	0.792599	0.353808	0.588301	0.851703
1	1.707675	0.716826	0.150946	0.608144	0.187433	0.433777	0.167472	0.827734	0.353808	0.588301	0.851703
2	1.691411	0.228607	0.240996	0.705564	0.313863	0.058303	0.108813	0.518546	0.353808	0.588301	0.851703
3	1.675147	0.307777	0.167507	0.656854	0.414122	0.550322	0.078115	0.623951	0.353808	0.588301	0.851703
4	1.658884	0.789399	0.755419	0.649895	0.347848	0.407878	0.216258	0.623951	0.353808	2.082200	0.851703

In [56]:
```
threshold = 3
print(np.where(z>3))
```

```
(array([104, 105, 105, 105, 105, 105, 106, 106, 106, 109, 110, 111, 111,
       130, 144, 161, 162, 162, 162, 170, 170, 171, 171, 173, 183, 183,
       187, 188, 200, 200, 202, 206, 212], dtype=int64), array([7, 1, 2, 5, 7, 8, 1, 5, 7, 7, 7, 1, 7, 7, 9, 9, 4, 5, 8,
4, 6, 4,
       6, 9, 2, 5, 5, 8, 5, 6, 8, 8, 8], dtype=int64))
```

In [ ]:
```
# here above we found 33 those values whose z-score is more then > 3
# i.e means we are having 33 outlier still present in our dataset, and we have to remove those outliers
```

In [57]:
```
df_new = df[(z<3).all(axis=1)]
df_new.shape
df_new
```

Out[57]:

	id number	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron	type of glass
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.00	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.00	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.00	1
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.00	1
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.00	0.26	1
...	...	...	...	...	...	...	...	...	...	...	...
207	209	1.51640	14.37	0.00	2.74	72.85	0.00	9.45	0.54	0.00	7
208	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.00	7
209	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.00	7
210	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.00	7
211	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.00	7

193 rows × 11 columns

In [58]:
```
df_new.shape
```
Out[58]:
```
(193, 11)
```

In [59]:
```
df.shape
```
Out[59]:
```
(213, 11)
```

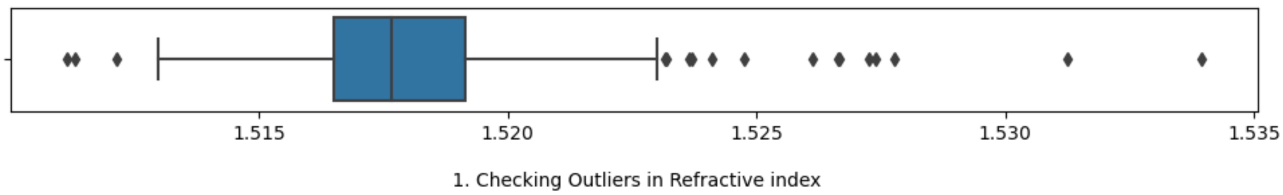
In [60]:
```
213-193
# here you can see our rows are reduced from 213 to 193, that means 20 Outliers are removed from our dataset.
```
Out[60]:
```
20
```

In [ ]:

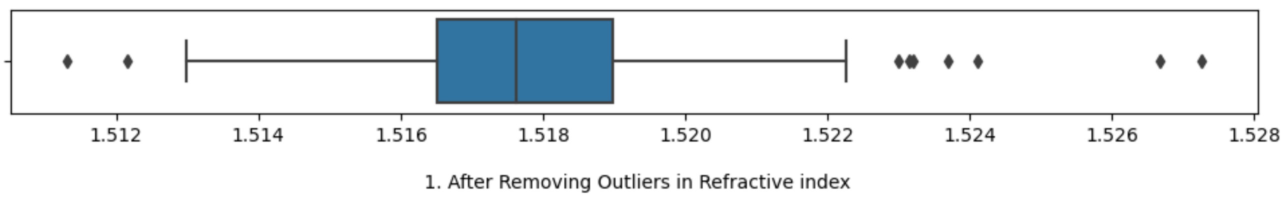
CHECKING REMOVAL OF OUTLIERS BY BOXPLOT (COMPARING 'df' & 'df_new')

===

In [62]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='refractive index',data=df)
plt.xlabel('\n 1. Checking Outliers in Refractive index')
plt.show()
# here we can see the presence of Outliers, but their range is very near to the maximum.
```



In [65]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='refractive index',data=df_new)
plt.xlabel('\n 1. After Removing Outliers in Refractive index')
plt.show()
# here we can see the removal of OUTLIERS takesplace succesfully
```

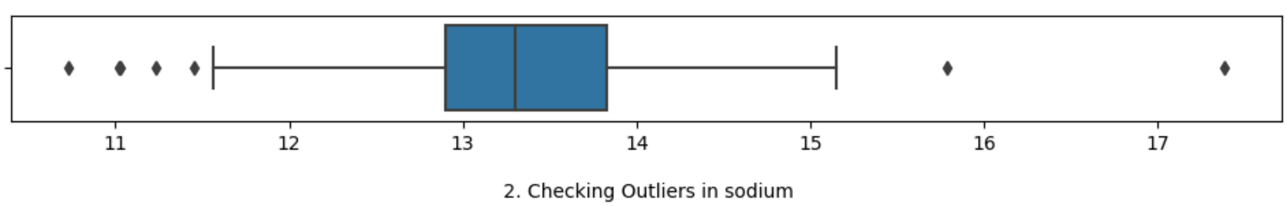


In [ ]:

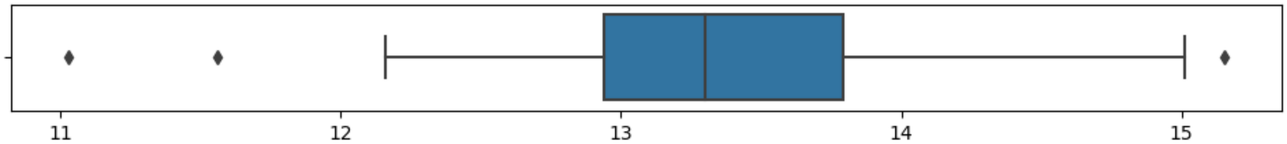
In [ ]:

In [ ]:

In [66]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='sodium',data=df)
plt.xlabel('\n 2. Checking Outliers in sodium')
plt.show()
# here we can see the presence of Outliers.
```



In [67]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='sodium',data=df_new)
plt.xlabel('\n 2. After Removing Outliers in sodium')
plt.show()
# here we can see the removal of outliers in new dataset.
```



2. After Removing Outliers in sodium

In [ ]:

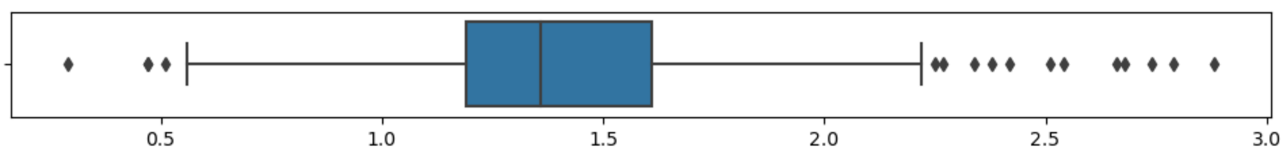
In [ ]:

In [ ]:

In [69]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='aluminium',data=df)
plt.xlabel('\n 4. Checking Outliers in Aluminium')
plt.show()
# here we can see the presence of Outliers.
```

4. Checking Outliers in Aluminium

In [70]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='aluminium',data=df_new)
plt.xlabel('\n 4. After Removingof  Outliers in Aluminium')
plt.show()
# here we can see the outliers at 3.5 are successfully removed.
```



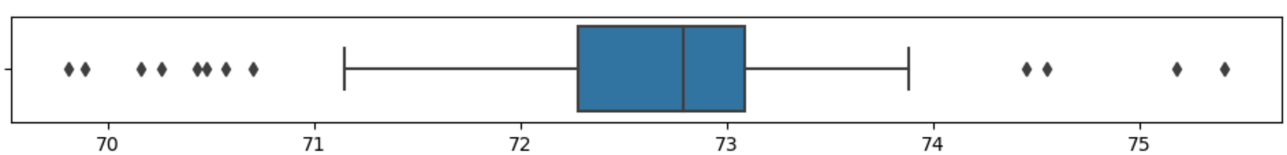
4. After Removingof Outliers in Aluminium

In [ ]:

In [ ]:

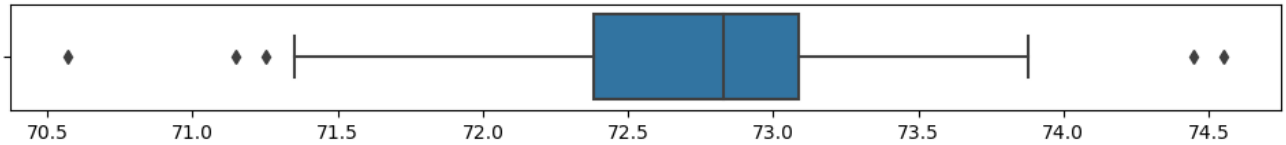
In [ ]:

In [71]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='silicon',data=df)
plt.xlabel('\n 5. Checking Outliers in Silicon')
plt.show()
# here we can see the presence of Outliers.
```



5. Checking Outliers in Silicon

In [72]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='silicon',data=df_new)
plt.xlabel('\n 5. After Removing Outliers in Silicon')
plt.show()
# here we can see the Outliers of both sides are removed successfully.
```



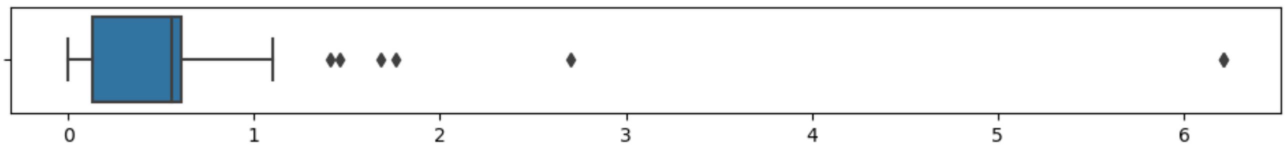
5. After Removing Outliers in Silicon

In [ ]:

In [ ]:

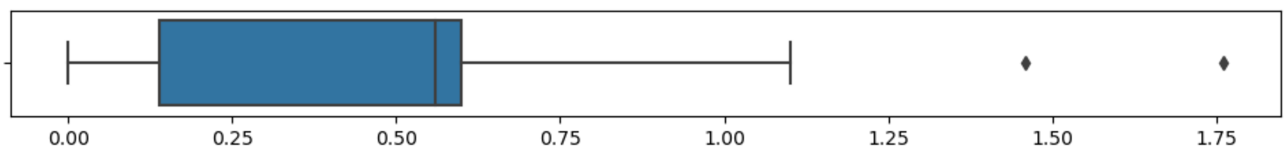
In [ ]:

In [73]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='potassium',data=df)
plt.xlabel('\n 6. Checking Outliers in Potassium')
plt.show()
# here we can see the presence of Outliers.
```



6. Checking Outliers in Potassium

In [74]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='potassium',data=df_new)
plt.xlabel('\n 6. After Removing Outliers in Potassium')
plt.show()
# here we can see the outliers are removed succesfully.
```



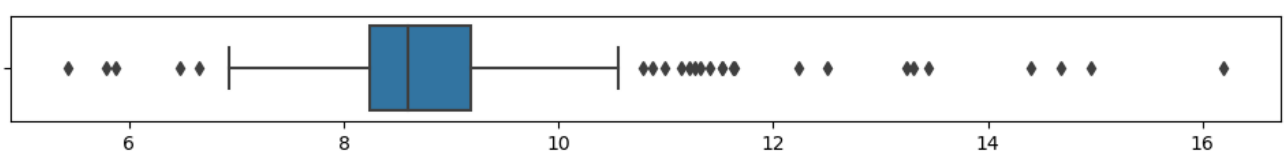
6. After Removing Outliers in Potassium

In [ ]:

In [ ]:

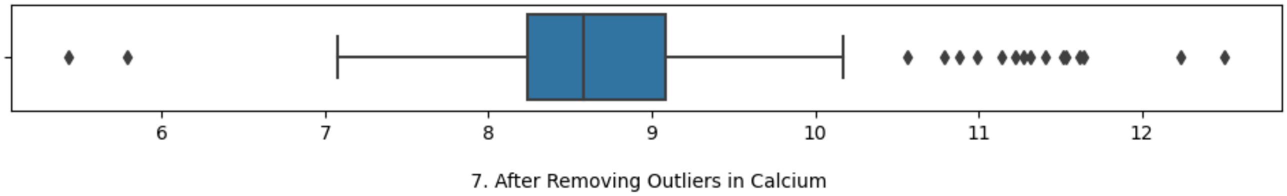
In [ ]:

In [75]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='calcium',data=df)
plt.xlabel('\n 7. Checking Outliers in Calcium')
plt.show()
# here we can see the presence of Outliers.
```



7. Checking Outliers in Calcium

In [76]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='calcium',data=df_new)
plt.xlabel('\n 7. After Removing Outliers in Calcium')
plt.show()
# here we can see the Outliers are removed successfully..
```

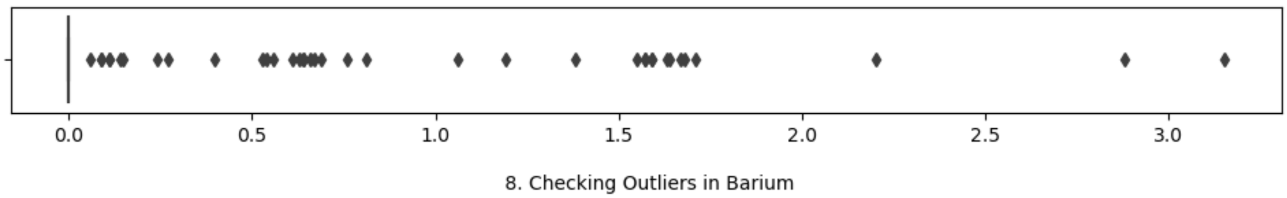


In [ ]:

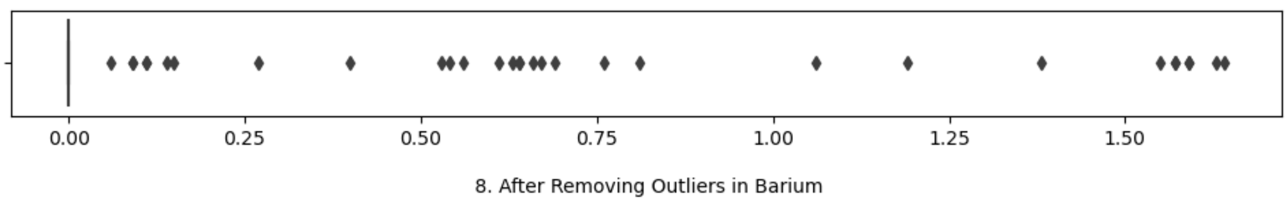
In [ ]:

In [ ]:

In [77]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='barium',data=df)
plt.xlabel('\n 8. Checking Outliers in Barium')
plt.show()
# here we can see the presence of Outliers.
```



In [78]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='barium',data=df_new)
plt.xlabel('\n 8. After Removing Outliers in Barium')
plt.show()
# here als we can see the clear difference after removing outlier from the column.
```

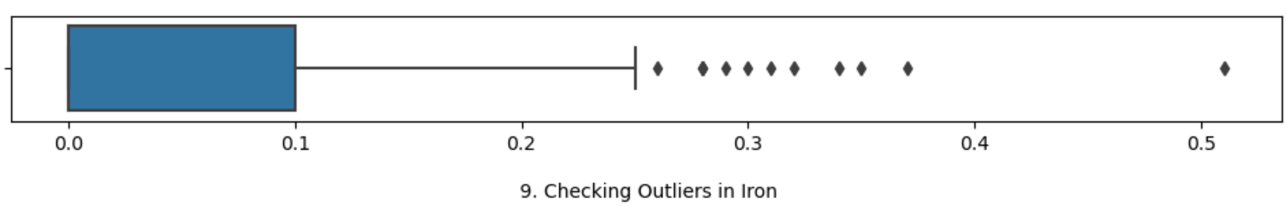


In [ ]:

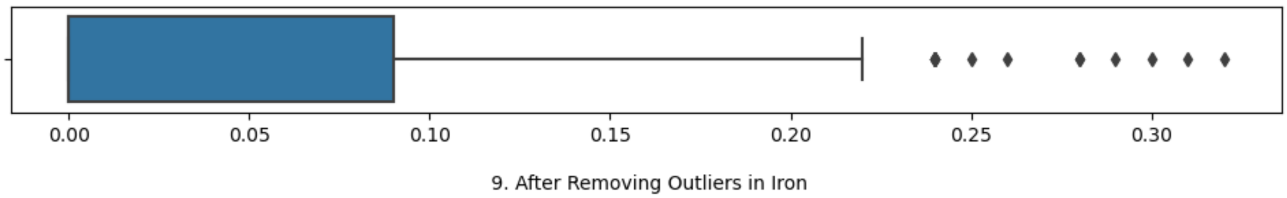
In [ ]:

In [ ]:

In [79]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='iron',data=df)
plt.xlabel('\n 9. Checking Outliers in Iron')
plt.show()
# here we can see the presence of Outliers.
```



In [80]:
```
plt.figure (figsize = (12,1), facecolor = "white")
sns.boxplot(x='iron',data=df_new)
plt.xlabel('\n 9. After Removing Outliers in Iron')
plt.show()
# outliers are successfully removed from the column.
```



In [ ]:
```
======================== successfully removed outliers from the dataset ================================================
```

In [ ]:

CHECKING SKEWNESS
==>>>

In [52]:
```
df.skew()
# skewness before removing outliers
```

Out[52]:
```
id number           0.000000
refractive index    1.639658
sodium              0.457318
magnesium          -1.154323
aluminium           0.900017
silicon            -0.744546
potassium           6.549276
calcium             2.040591
barium              3.406749
iron                1.747173
type of glass       1.108861
dtype: float64
```

In [81]:
```
df_new.skew()
# skewness after removing outliers. WE CAN CLEARLY SEE THE DIFFERENCE BETWEEEN BOTH.
```

Out[81]:
```
id number           0.107364
refractive index    0.971729
sodium              0.375857
magnesium          -1.533664
aluminium           0.649917
silicon            -0.436288
potassium           0.297900
calcium             1.074092
barium              3.178256
iron                1.478611
type of glass       1.277279
dtype: float64
```

In [ ]:
```
# the skewness shows the distribution of data, if the data is widely skewed that means it is not good for our model.
#  ideal range of skewness is ( -0.5 to +0.5)
# We can't remove skewness from our Target Column
# here we can see the skewness is present in 'MAGNESIUM', 'CALCIUM','BARIUM',& 'IRON'.
#  so we need to remove skewness from those mentioned columns.
```

In [ ]:
```
# so we have to remove skewness from those columns by using 'cuberoot' method.
```

In [84]:
```
df_new['magnesium'] = np.cbrt(df_new['magnesium'])
```

In [85]:
```
df_new['calcium'] = np.cbrt(df_new['calcium'])
```

In [86]:
```
df_new['barium'] = np.cbrt(df_new['barium'])
```

In [87]:
```
df_new['iron'] = np.cbrt(df_new['iron'])
```

In [88]:
```
df_new.skew()
# here we can see that skewness of most of the treated columns are removed , but still there is little skewness present in
#    cloumn 'barium'
```

Out[88]:
```
id number           0.107364
refractive index    0.971729
sodium              0.375857
magnesium          -1.892297
aluminium           0.649917
silicon            -0.436288
potassium           0.297900
calcium             0.609725
barium              2.198820
iron                0.875093
type of glass       1.277279
dtype: float64
```

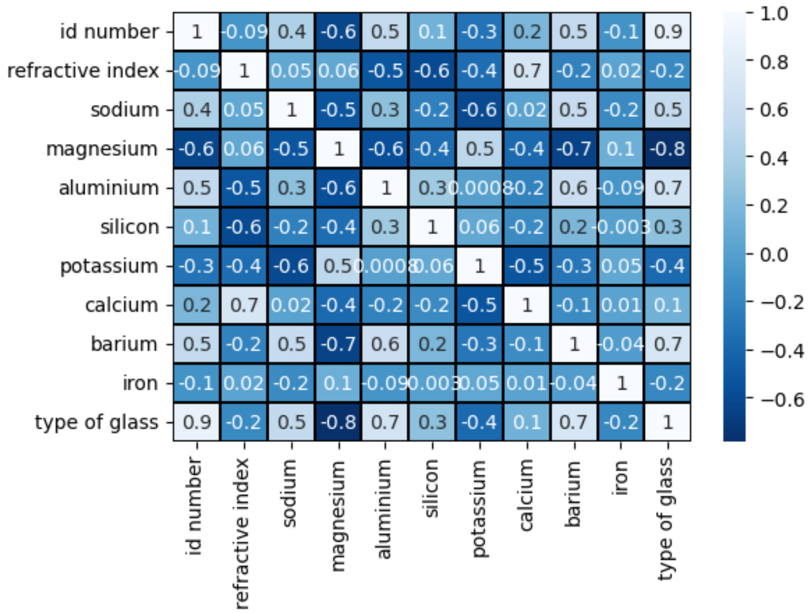
In [ ]:

FINDING CORRELATION (GRAPHICALLY)
===

In [90]:
```
cor = df_new.corr()
```

In [91]:
```
plt.figure (figsize = (6,4), facecolor = "white")
sns.heatmap(df_new.corr(),linewidth=0.1,fmt="0.1g",linecolor="black",annot=True,cmap="Blues_r")
plt.yticks(rotation=0);
plt.show()

# here we can say there is good correlation between 'type of glass' & 'aluminum' = 0.7
#                           good correlation between 'type of glass' & 'barium' = 0.7
#                           good correlation between 'type of glass' & 'sodium' = 0.5
#                           neagtive correlation between 'type of glass' & 'Magnesium' = -0.8
#                          good correlation between 'calcium' & 'refractive index' = 0.7
#                          good correlation between 'type of potassium' & 'magnesium' = 0.5
```



In [92]:
```
cor['type of glass'].sort_values(ascending=False)
# here we can see in the correltion of all independent vaules with Target Column = 'type of glass'
#  there no such any huge correation with target column.
```

Out[92]:
```
type of glass       1.000000
id number           0.873287
barium              0.700780
aluminium           0.659242
sodium              0.535440
silicon             0.257640
calcium             0.136329
refractive index   -0.162981
iron               -0.185437
potassium          -0.391378
magnesium          -0.781817
Name: type of glass, dtype: float64
```

In [ ]:

DIVIDING DATA INTO INDEPENDENT & TARGET VARIABLE
==

In [95]:
```
df_new.head(5)
```
Out[95]:

	id number	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron	type of glass
0	2	1.51761	13.89	1.532619	1.36	72.73	0.48	1.985732	0.0	0.00000	1
1	3	1.51618	13.53	1.525490	1.54	72.99	0.39	1.981496	0.0	0.00000	1
2	4	1.51766	13.21	1.545286	1.29	72.61	0.57	2.018168	0.0	0.00000	1
3	5	1.51742	13.27	1.535452	1.24	73.08	0.55	2.005816	0.0	0.00000	1
4	6	1.51596	12.79	1.534037	1.62	72.97	0.64	2.005816	0.0	0.63825	1

In [96]:
```
df_new.columns
```
Out[96]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

In [111]:
```
x = df_new[['refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron']]

# here we are droping the column 'id number', because we think there is no relevance of 'id number' in the model.
```

In [112]:
```
y = df_new[['type of glass']]

# here we are taking 'type of glass' as oue TARGET COLUMN.
```

In [113]:
```
x.shape
```
Out[113]:
```
(193, 9)
```

In [114]:
```
y.shape
```
Out[114]:
```
(193, 1)
```

In [ ]:

APPLYING SCALING TECHNIQUES
==>:

In [105]:
```
# here we need to apply scaling techniques on our dataset, because in few of the columns like- sodium,silicon their
#   values are too high as compared to others, therefor by scaling techniques we normalise the values.
#  we can't apply SCALING TECHNIQUES on TARGET VARIABLE
#  to aplly scaling techinuque we need to import some libraries first.
```

In [115]:
```
from sklearn.preprocessing import StandardScaler
```

In [116]:
```
st = StandardScaler()
```

In [117]:
```
x = st.fit_transform(x)
x
```
Out[117]:
```
array([[-0.19147536,  0.70449765,  0.48418773, ..., -1.03570349,
        -0.41656857, -0.68059545],
       [-0.81909158,  0.16965831,  0.47087741, ..., -1.09193464,
        -0.41656857, -0.68059545],
       [-0.16953074, -0.30575444,  0.50783905, ..., -0.60510737,
        -0.41656857, -0.68059545],
       ...,
       [-0.52503363,  2.23473244, -2.37749167, ..., -0.41095801,
         3.13848105, -0.68059545],
       [ 1.14275771,  1.40276013, -2.37749167, ..., -0.36819118,
         3.17536179, -0.68059545],
       [-0.67425707,  1.43247343, -2.37749167, ..., -0.32555926,
         3.12351224, -0.68059545]])
```

In [118]:
```
xf = pd.DataFrame(data=x)
print(xf)

# here we get our dataset (xf) after applying SCALING TECHING (STANDARD SCALER)
```

```
            0         1         2         3         4         5         6  \
0   -0.191475  0.704498  0.484188 -0.145324  0.007760  0.165576 -1.035703
1   -0.819092  0.169658  0.470877  0.260836  0.455342 -0.154697 -1.091935
2   -0.169531 -0.305754  0.507839 -0.303276 -0.198816  0.485849 -0.605107
3   -0.274865 -0.216615  0.489477 -0.416098  0.610275  0.414677 -0.769075
4   -0.915648 -0.929734  0.486835  0.441351  0.420913  0.734950 -0.769075
..        ...       ...       ...       ...       ...       ...       ...
188 -0.722535  1.417617 -2.377492  2.968571  0.214336 -1.542547  0.669606
189 -0.797147  1.075914 -2.377492  3.284473 -0.198816 -1.257860  0.399720
190 -0.525034  2.234732 -2.377492  1.276236  0.575845 -1.542547 -0.410958
191  1.142758  1.402760 -2.377492  1.343930  1.195574 -1.542547 -0.368191
192 -0.674257  1.432473 -2.377492  1.163414  1.522654 -1.542547 -0.325559

            7         8
0   -0.416569 -0.680595
1   -0.416569 -0.680595
2   -0.416569 -0.680595
3   -0.416569 -0.680595
4   -0.416569  1.848061
..        ...       ...
188  2.063771 -0.680595
189  2.689053 -0.680595
190  3.138481 -0.680595
191  3.175362 -0.680595
192  3.123512 -0.680595

[193 rows x 9 columns]
```

In [122]:
```
xf.columns
```

Out[122]:
```
RangeIndex(start=0, stop=9, step=1)
```

In [123]:
```
df_new.columns
```

Out[123]:
```
Index(['id number', 'refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron', 'type of glass'],
      dtype='object')
```

In [124]:
```
column = ['refractive index', 'sodium', 'magnesium', 'aluminium',
       'silicon', 'potassium', 'calcium', 'barium', 'iron']
```

In [125]:
```
xf.columns=column
```

In [126]:
```
xf.head(5)
```

Out[126]:

	refractive index	sodium	magnesium	aluminium	silicon	potassium	calcium	barium	iron
0	-0.191475	0.704498	0.484188	-0.145324	0.007760	0.165576	-1.035703	-0.416569	-0.680595
1	-0.819092	0.169658	0.470877	0.260836	0.455342	-0.154697	-1.091935	-0.416569	-0.680595
2	-0.169531	-0.305754	0.507839	-0.303276	-0.198816	0.485849	-0.605107	-0.416569	-0.680595
3	-0.274865	-0.216615	0.489477	-0.416098	0.610275	0.414677	-0.769075	-0.416569	-0.680595
4	-0.915648	-0.929734	0.486835	0.441351	0.420913	0.734950	-0.769075	-0.416569	1.848061

In [ ]:
```
# similarly for target column.
```

In [127]:
```
yf=y
```

In [128]:
```
yf.head(2)
```

Out[128]:

	type of glass
0	1
1	1

In [ ]:

```
FINDING MULTICOLINEARITY
=====================================================================================================
```

In [130]:
```
# We have to find the multicollinearity between the features and to remove it we can use VIF (VARIANCE INFLATION FACTOR)
# we can not apply VIF on the TARGET COLUMN
# for apllyin VIF we have to import some libraries as follows
```

In [131]:
```
import statsmodels.api as sm
from scipy import stats
from statsmodels .stats.outliers_influence import variance_inflation_factor
```

In [132]:
```
# here we are making "def function" for calculating VIF
def calc_vif(xf):
    vif = pd.DataFrame()
    vif["FETURES"] = xf.columns
    vif["VIF FACTOR"] = [variance_inflation_factor(xf.values,i) for i in range (xf.shape[1])]
    return (vif)
```

In [133]:
```
xf.shape
```

Out[133]:
```
(193, 9)
```

In [134]:
```
yf.shape
```

Out[134]:
```
(193, 1)
```

In [135]:
```
calc_vif(xf)
# here we didn't find MULTICOLINEARITY between the independent Columns.
```

Out[135]:

	FETURES	VIF FACTOR
0	refractive index	5.126622
1	sodium	7.815622
2	magnesium	14.959042
3	aluminium	3.517437
4	silicon	5.603866
5	potassium	4.266499
6	calcium	10.433140
7	barium	3.634047
8	iron	1.054808

In [ ]:
```
# here we can see that the highest VIF values are 14.95 & 10.43 for 'magnesium' & 'calcium'
#  we can drop 'magnesium' & 'calcium' column
#  but before droping those column, we need to chek the correlation of the column with the "TARGET COLUMN"
```

In [136]:
```
cor['type of glass'].sort_values(ascending=False)
```

Out[136]:
```
type of glass       1.000000
id number           0.873287
barium              0.700780
aluminium           0.659242
sodium              0.535440
silicon             0.257640
calcium             0.136329
refractive index   -0.162981
iron               -0.185437
potassium          -0.391378
magnesium          -0.781817
Name: type of glass, dtype: float64
```

In [ ]:
```
# here we can see that 'MAGNESIUM' is highly NEAGTIVE CORRELATED  with the TARGET COLUMN.
# so i think we should not drop the 'MAGENSIUM COLUMN'.
#  But we can see that 'calcium' relation with the TARGET COLUMN is only = 0.13.
# so i think we can drop 'calcium columns' and then after droping , we should again chek VIF of MAGNESIUM.
```

In [143]:
```
xf.drop(['calcium'],axis=1,inplace=True)
```

In [145]:
```
xf.shape
```

Out[145]:
```
(193, 8)
```

In [147]:
```
calc_vif(xf)
# here we are again checking VIF for the remaining columns
# here we can clearly seen the difference between the VIF values of earlier and now.
#  the VIF value of ' MAGNESIUM' is reduced from 14.95 to 4.88, and other VIF values are also reduced.
```

Out[147]:

	FETURES	VIF FACTOR
0	refractive index	4.464314
1	sodium	4.093225
2	magnesium	4.884832
3	aluminium	3.177844
4	silicon	4.295527
5	potassium	2.655712
6	barium	2.364805
7	iron	1.053033

In [ ]:

In [148]:
```
xf.shape
```

Out[148]:
```
(193, 8)
```

In [149]:
```
yf.shape
```

Out[149]:
```
(193, 1)
```

In [ ]:

RESAMPLING (APPLYING SMOTE)
===

In [ ]:
```
# Here we know that our Target Column is a Categorical column. which is having values from 1-6.
# so we have to chek the distribution of values are equal or not, offcourse i would be not, so we have to make them equally
#  'equally balanced distributed' for better results.

# SOLVING CLASS IMMBALANCE PROBLEM BY SMOTE TECHNIQUE.
```

In [150]:
```
yf.value_counts()
# here we can see that the CLASS IMMBALANCE PROBLEM
# every category is having different values.
```

Out[150]:
```
type of glass
1                69
2                68
7                23
3                16
5                 9
6                 8
dtype: int64
```

In [ ]:
```
# To solve this prolem we need import SMOTE LIBRARY from the IMBLEARN.
```

In [152]:
```
from imblearn.over_sampling import SMOTE
```

In [153]:
```
smt = SMOTE()
```

In [154]:
```
trainx, trainy = smt.fit_resample(xf,yf)
```

In [155]:
```
trainy.value_counts()
# here as you can see below the immbalancenes is cleared now.
```

Out[155]:
```
type of glass
1                69
2                69
3                69
5                69
6                69
7                69
dtype: int64
```

In [156]:
```
trainx.shape
```
Out[156]:
```
(414, 8)
```

In [157]:
```
trainy.shape
```
Out[157]:
```
(414, 1)
```

In [158]:
```
# Now here our both INDEPENDENT VALUES & DEPENDENT VALUES are BALANCED.
```

================ UPTO HERE EDA AND OTHER TECHINIQUES ARE COMPLETED ==================================

=============================== NOW WE NEED TO APPLY ML MODELS
===

In [180]:
```
from sklearn.model_selection import train_test_split
```

In [181]:
```
 x_train,x_test,y_train,y_test = train_test_split(trainx,trainy,test_size=0.20,random_state=42)
```

In [ ]:

In [161]:
```
import sklearn
```

In [170]:
```
from sklearn.linear_model import LogisticRegression
```

In [163]:
```
from sklearn.naive_bayes import GaussianNB
```

In [164]:
```
from sklearn.svm import SVC
```

In [166]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [167]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [168]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
```

In [171]:
```
lg = LogisticRegression()
```

In [172]:
```
gnb = GaussianNB()
```

In [173]:
```
svc = SVC()
```

In [174]:
```
dtc = DecisionTreeClassifier()
```

In [177]:
```
knn = KNeighborsClassifier()
```

In [178]:
```
model = [lg,gnb,svc,dtc,knn]
```

In [182]:
```
for i in model:
    i.fit(x_train,y_train)
    i.score(x_train,y_train)
    ipred = i.predict(x_train)

    print('Accuracy Score of ', i,'is:')
    print (accuracy_score(y_train,ipred))

    print(confusion_matrix(y_train,ipred))
    print(classification_report(y_train,ipred))
    print('\n')
```

```
Accuracy Score of  LogisticRegression() is:
0.8398791540785498
[[34  8 11  0  0  0]
 [12 26 11  0  1  0]
 [ 3  6 50  0  0  0]
 [ 0  0  0 57  0  0]
 [ 0  0  0  0 56  0]
 [ 1  0  0  0  0 55]]
              precision    recall  f1-score   support

           1       0.68      0.64      0.66        53
           2       0.65      0.52      0.58        50
           3       0.69      0.85      0.76        59
           5       1.00      1.00      1.00        57
           6       0.98      1.00      0.99        56
           7       1.00      0.98      0.99        56

    accuracy                           0.84       331
   macro avg       0.83      0.83      0.83       331
```

In [ ]:
```
# here from the above result we find the following accuracy :-
# 1) LOGISTIC REGRESSION = 84 %
# 2) GUASSIAN NB         = 70 %
# 3) SVC                 = 85 %
# 4) DTC                 = 100 %
#  5) KNN                = 90 %

#  SO FROM THE ABOVE WE CONCLUDE THAT OUR ALL MODELS ARE WORKING VERY GOOD,
#  SPECIALLY ( DTC, KNN, SVC & LG)
```

In [183]:
```
# SO FOR FINAL MODEL WE ARE USING DECISION TREE CLASSIFIER :
```

FINDING BEST PERAMETERS WITH GRIDSEARCH CV FOR DTC MODEL
==

In [185]:
```
from sklearn.model_selection import GridSearchCV
```

In [186]:
```
grid_param = {'criterion':['gini','entropy']}
```

In [187]:
```
gd_sr = GridSearchCV (estimator=dtc, param_grid= grid_param, scoring="accuracy",cv=5)
```

In [188]:
```
gd_sr.fit(x_train,y_train)
```

Out[188]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy']}, scoring='accuracy')
```

In [189]:
```
best_perameter = gd_sr.best_params_
print(best_perameter)
```

```
{'criterion': 'entropy'}
```

In [190]:
```
# here we can find the best perameter for the model is "entropy"
```

In [191]:
```
best_result = gd_sr.best_score_
print(best_result)
```

```
0.8819990954319312
```

In [192]:
```
print(round(best_result,2))
```

```
0.88
```

In [ ]:
```
#  the best score is .88%
```

In [193]:
```
final_model = DecisionTreeClassifier (criterion="entropy")
```

In [194]:
```
final_model.fit(x_train,y_train)
final_model.score(x_train,y_train)
final_model_pred = final_model.predict(x_test)
print(accuracy_score(y_test,final_model_pred))
print(confusion_matrix(y_test,final_model_pred))
print(classification_report(y_test,final_model_pred))
```

```
0.8554216867469879
[[10  6  0  0  0  0]
 [ 2 14  2  0  1  0]
 [ 1  0  9  0  0  0]
 [ 0  0  0 12  0  0]
 [ 0  0  0  0 13  0]
 [ 0  0  0  0  0 13]]
              precision    recall  f1-score   support

           1       0.77      0.62      0.69        16
           2       0.70      0.74      0.72        19
           3       0.82      0.90      0.86        10
           5       1.00      1.00      1.00        12
           6       0.93      1.00      0.96        13
           7       1.00      1.00      1.00        13

    accuracy                           0.86        83
   macro avg       0.87      0.88      0.87        83
weighted avg       0.85      0.86      0.85        83
```

In [ ]:
```
# HERE ABOVE WE CAN FIND THE ACCURACY OF OUR MODEL ID = 86 %
```

CREATING FUNCTION TO PREDICT
===

In [206]:
```
def pred_func(g):
    g= g.reshape(1,8)
    gt = final_model.predict(g)
    print(gt)

    if gt == 1:
        print("Building Windows - Float Processed")
    elif (gt == 2):
        print ("Building Windows - Non Float Processed")
    elif (gt == 3):
        print ("Vehicle Windows - Float Processed")
    elif (gt == 4):
        print ("Vehicle Windows - Non Float Processed")
    elif (gt == 5):
        print ("Container")
    elif (gt == 6):
        print ("Tableware")
    elif (gt == 7):
        print("Head Lamps")
    else:
        print('Not Found')
```

In [207]:
```
g= np.array([-0.191475,0.704498,0.484188,-0.145324,0.007760,0.165576,-0.416569,-0.680595])
pred_func(g)
```

```
[1]
Building Windows - Float Processed
```

In [212]:
```
g= np.array([-1.135084,0.977792,-2.377492,2.625567,0.778999,-1.186657,2.336964,-0.168386])
pred_func(g)
```

```
[7]
Head Lamps
```

In [ ]:

SAVING MODEL
===

In [214]:
```
import pickle
```

In [215]:
```
file_name = 'glass_identification_prediction.pkl'
pickle.dump(final_model,open(file_name,'wb'))
```

======================================= FINISHED ===

In [ ]: