Programming Machine Learning Lab

Exercise 2

General Instructions:

1. You need to submit the PDF as well as the filled notebook file.
2. Name your submissions by prefixing your matriculation number to the filename. Example, if your MR is 12345 then rename the files as **"12345_Exercise_2.xxx"**
3. Complete all your tasks and then do a clean run before generating the final pdf. (*Clear All Ouputs* and *Run All* commands in Jupyter notebook)

Exercise Specific instructions::

1. You are allowed to use only NumPy and Pandas (unless stated otherwise). You can use any library for visualizations.

Data Handling

For this exercise we will use the file **"train.csv"** and **"test.csv"**. The files includes a subset of dataset from **Categorical Feature Encoding Challenge** from Kaggle (link).

The dataset does not contain any numerical variables, and we will explore how to deal with non-numerical data variables.

Data Exploration

1. Read the dataset. There are multiple types of variable present in the dataset, some common types are

   - **Binary data** : A variable that has only 2 values.
   - **Categorical data** : A variable that can only take a limited number of values.
   - **Nominal data** : A variable that has no numerical importance. Like name of a person.
   - **Time Series data** : A variable that has some temporal value attached to it.

2. Explore the train dataset (both statistically and visually).

3. Identify which columns belong to which type of data.

4. Assume the columns with more than 50 unique values to be of nominal data type and remove them from both datasets.

In [ ]:
```python
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```python
### Write your code here
df = pd.read_csv('train.csv', index_col=None, header=0)


print(df.describe())
print(df.info())
df = df.drop(labels=[col for col in df.columns if df[col].nunique() > 50], axis=1)
df.head()
```

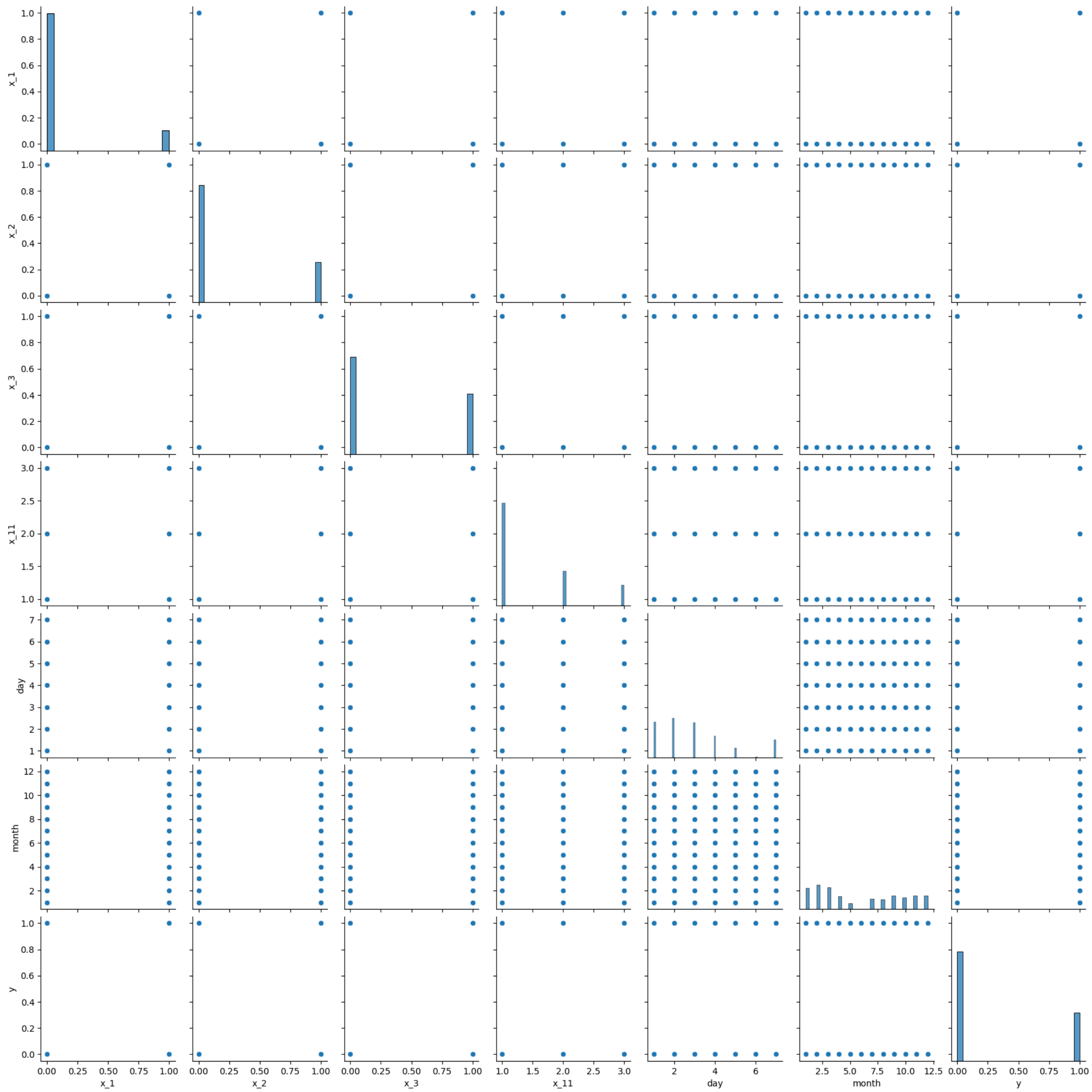
```
                x_1          x_2          x_3         x_11          day  \
count  80000.000000  80000.000000  80000.000000  80000.000000  80000.000000
mean       0.127075      0.255075      0.383088      1.479475      3.007750
std        0.333059      0.435906      0.486142      0.713765      1.819404
min        0.000000      0.000000      0.000000      1.000000      1.000000
25%        0.000000      0.000000      0.000000      1.000000      2.000000
50%        0.000000      0.000000      0.000000      1.000000      3.000000
75%        0.000000      1.000000      1.000000      2.000000      4.000000
max        1.000000      1.000000      1.000000      3.000000      7.000000

              month            y
count  80000.000000  80000.000000
mean       5.784775      0.305887
std        3.848748      0.460785
min        1.000000      0.000000
25%        2.000000      0.000000
50%        4.000000      0.000000
75%        9.000000      1.000000
max       12.000000      1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80000 entries, 0 to 79999
Data columns (total 18 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   x_1     80000 non-null  int64
 1   x_2     80000 non-null  int64
 2   x_3     80000 non-null  int64
 3   x_4     80000 non-null  object
 4   x_5     80000 non-null  object
 5   x_6     80000 non-null  object
 6   x_7     80000 non-null  object
 7   x_8     80000 non-null  object
 8   x_9     80000 non-null  object
 9   x_10    80000 non-null  object
 10  x_11    80000 non-null  int64
 11  x_12    80000 non-null  object
 12  x_13    80000 non-null  object
 13  x_14    80000 non-null  object
 14  x_15    80000 non-null  object
 15  day     80000 non-null  int64
 16  month   80000 non-null  int64
 17  y       80000 non-null  int64
dtypes: int64(7), object(11)
memory usage: 11.0+ MB
None
```

Out[ ]:

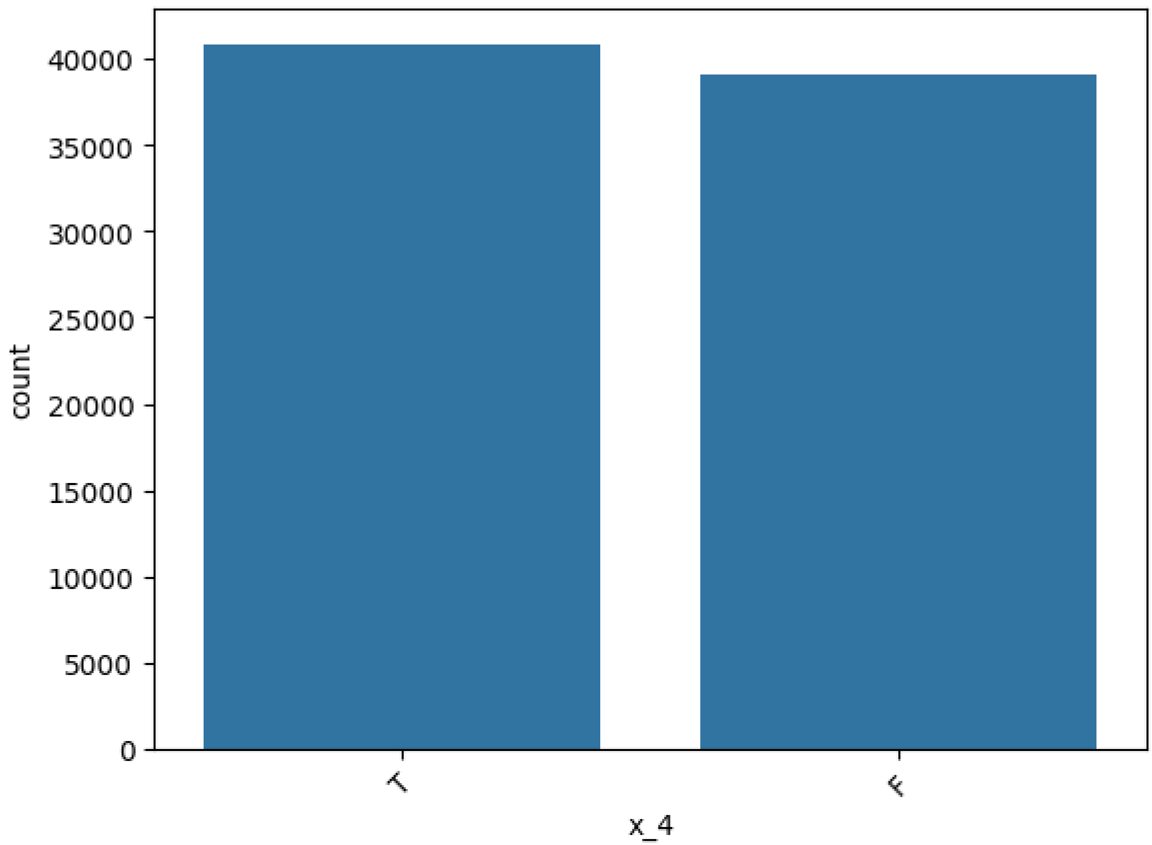
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_10	x_11	x_12
0	0	1	1	T	N	Blue	Star	Hamster	Canada	Piano	2	Grandmaster
1	0	0	1	F	Y	Red	Polygon	Lion	Russia	Oboe	2	Novice
2	0	0	1	F	Y	Green	Trapezoid	Cat	Russia	Bassoon	1	Novice
3	0	0	0	F	Y	Blue	Star	Cat	China	Bassoon	3	Grandmaster
4	0	0	1	T	Y	Green	Triangle	Lion	Costa Rica	Oboe	2	Master

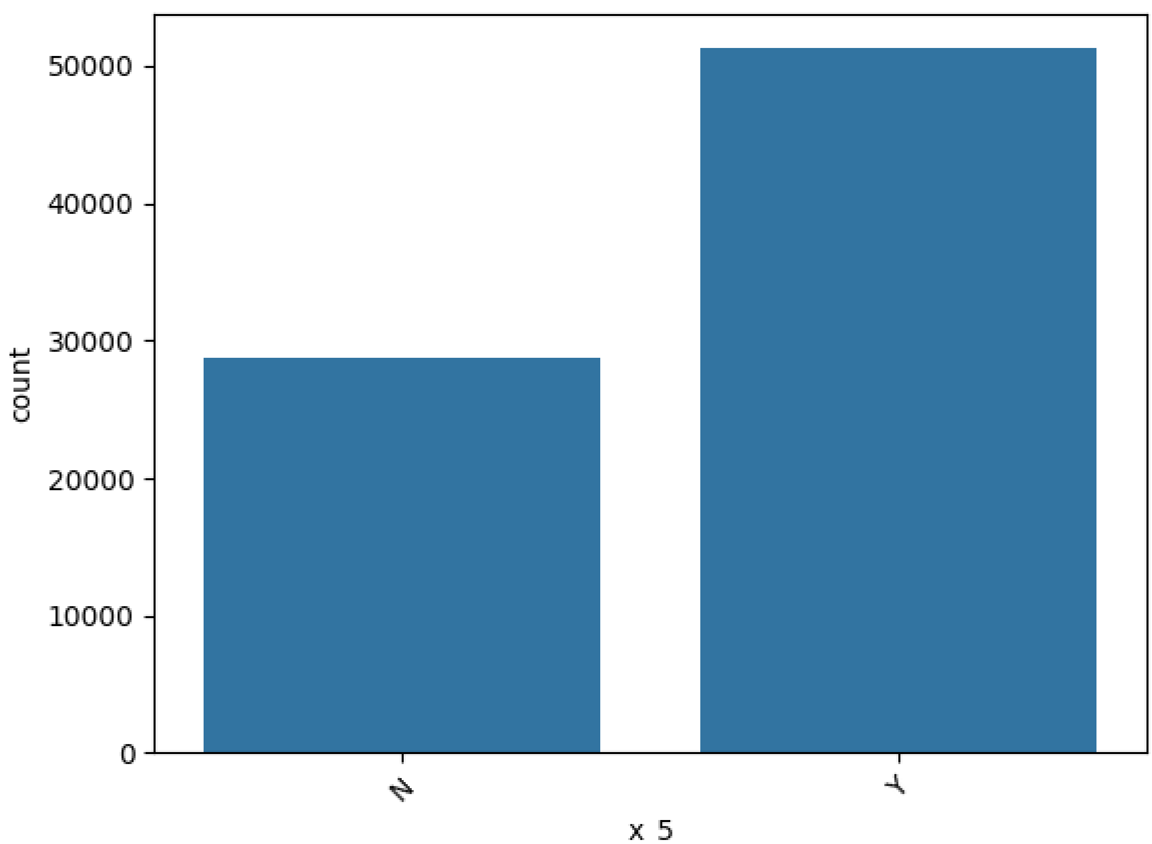
In [ ]:

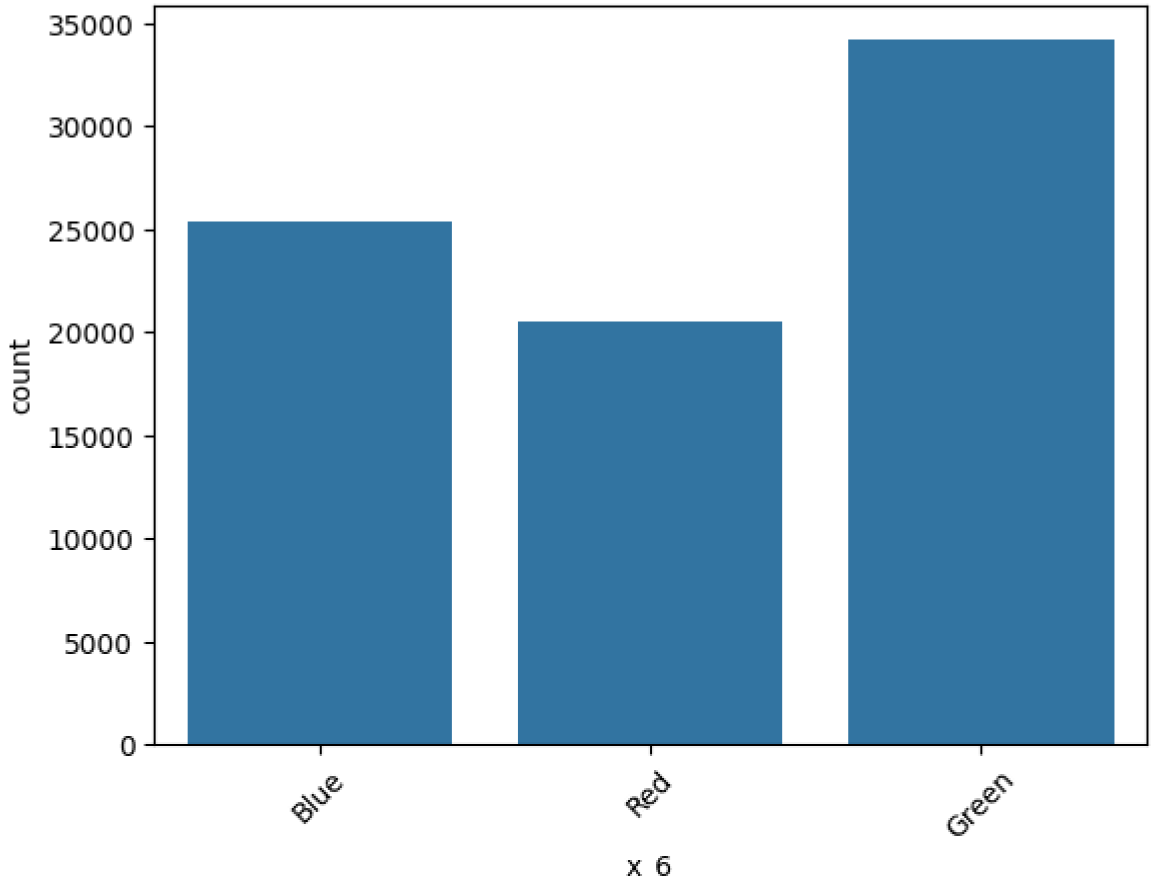
```python
# Pairplot for numerical columns
numerical_columns = df.select_dtypes(include=['int64'])
sns.pairplot(numerical_columns)
plt.show()
categorical_columns = df.select_dtypes(exclude=[
    'int64'
])
for categorical_column in categorical_columns:
    sns.countplot(data=df, x=categorical_column)
    plt.xticks(rotation=45)
    plt.show()
```

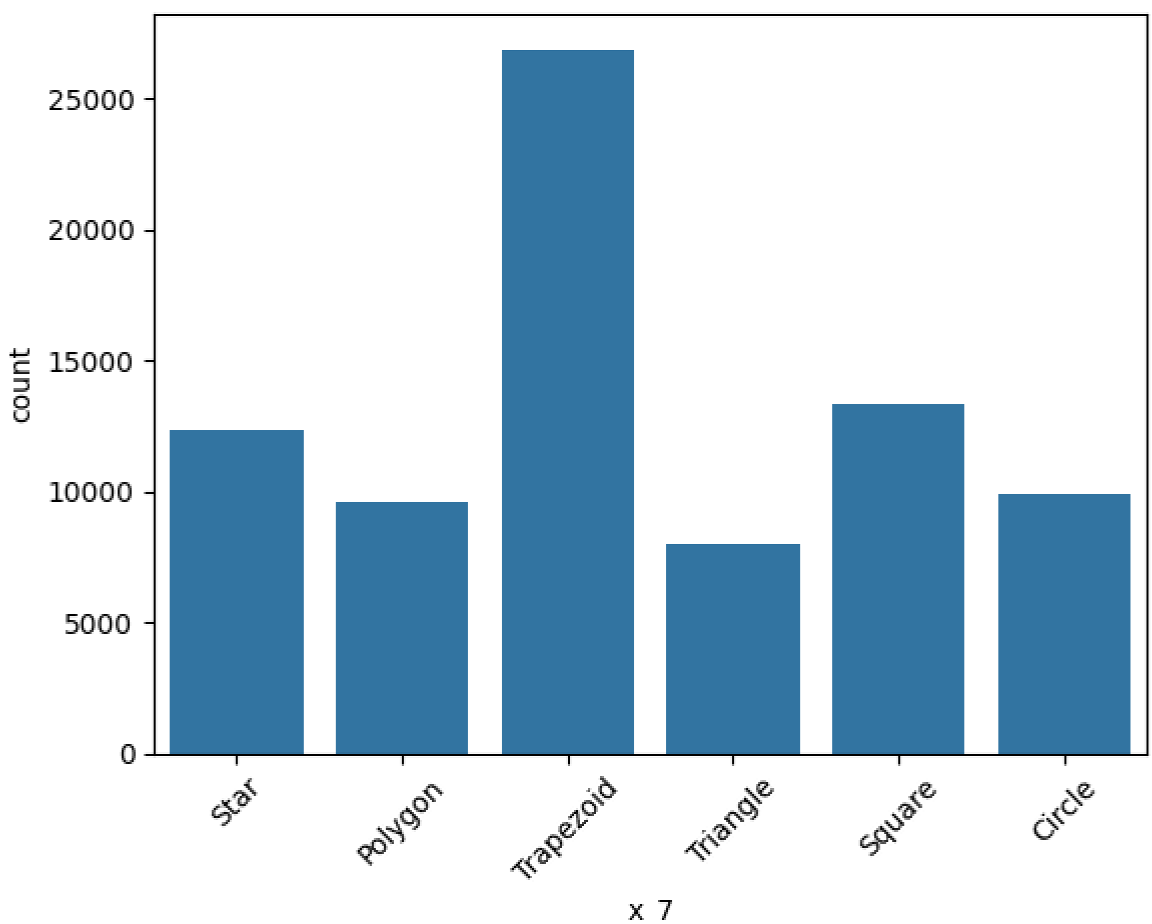


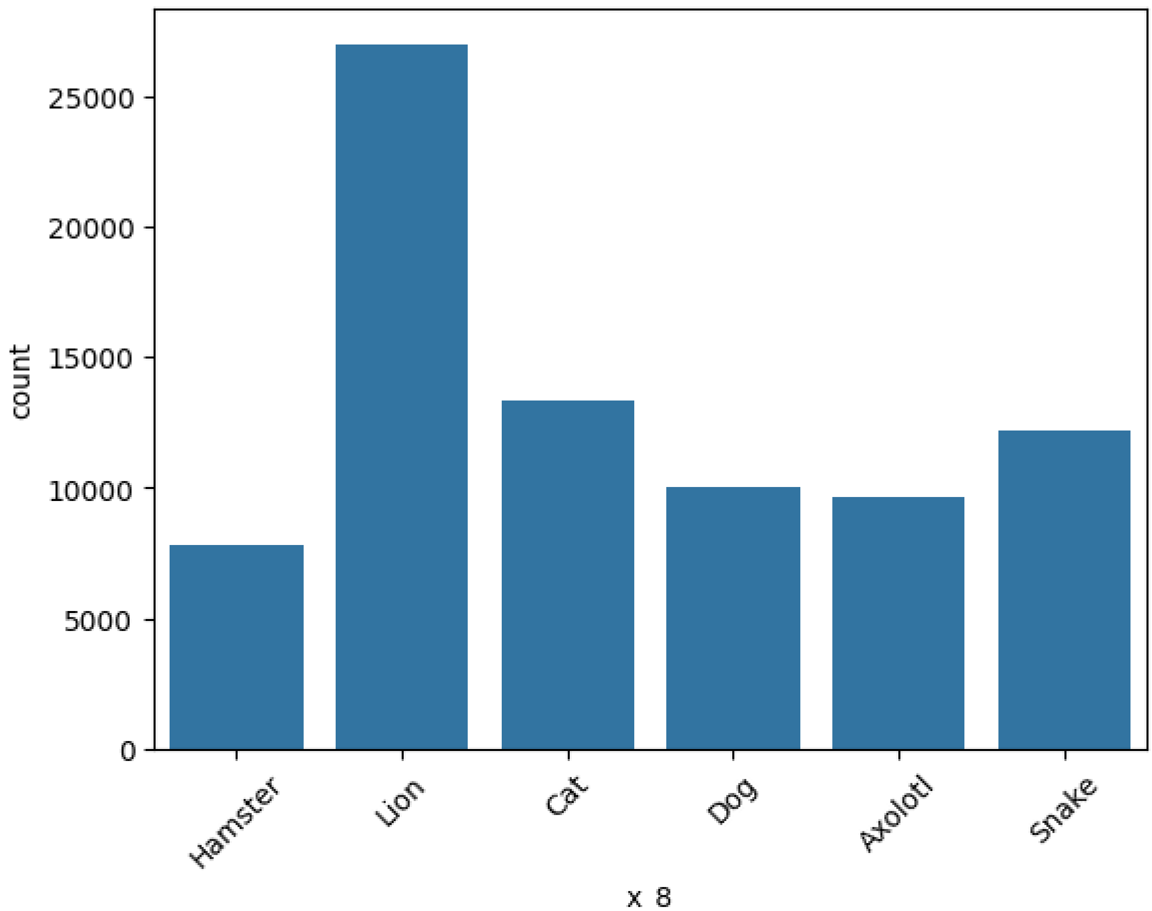
x_1
x_2
x_3
x_11
day
month
y


40000
35000
30000
25000
20000
15000
10000
5000
0
count
T
F
x_4

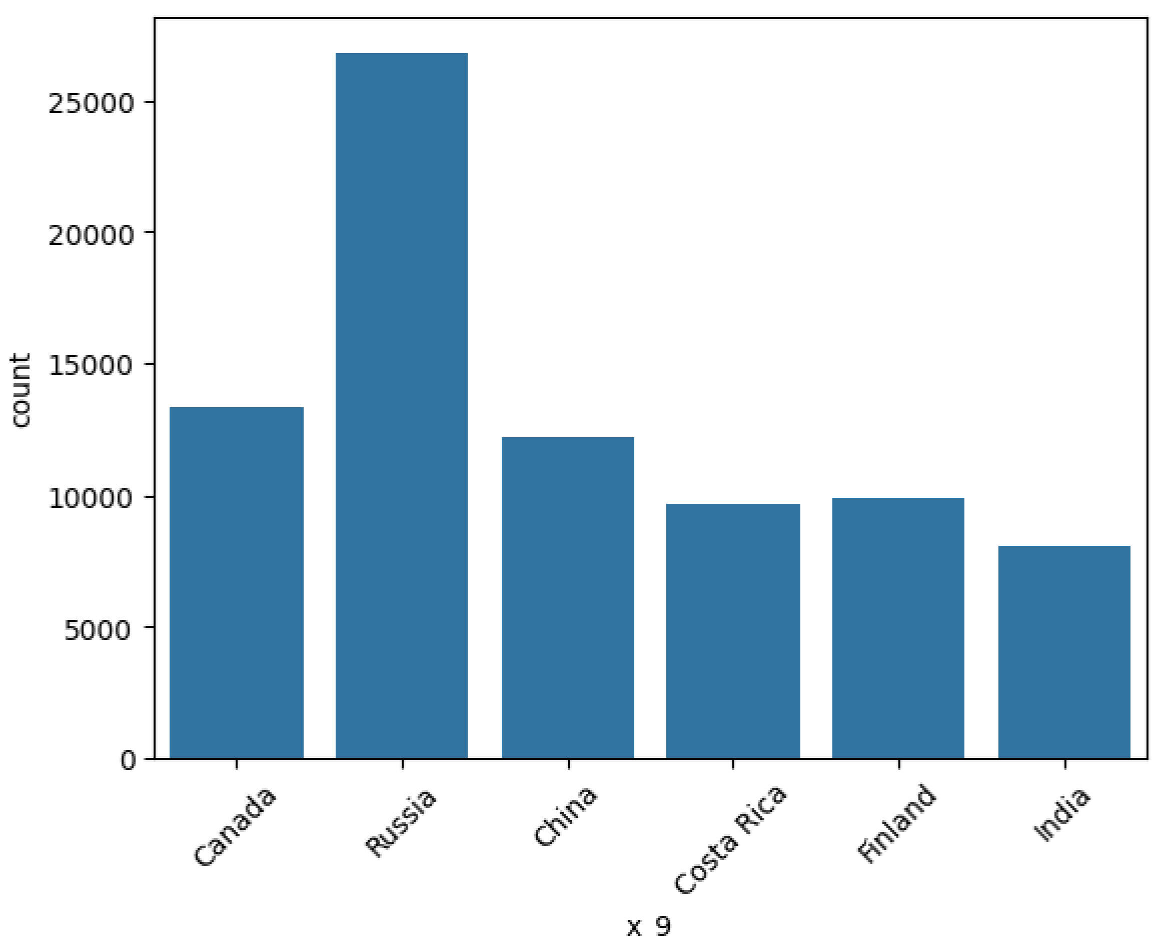

50000
40000
30000
20000
10000
0
count
N
Y
x_5


35000
30000
25000
20000
15000
10000
5000
0
count
Blue
Red
Green
x_6

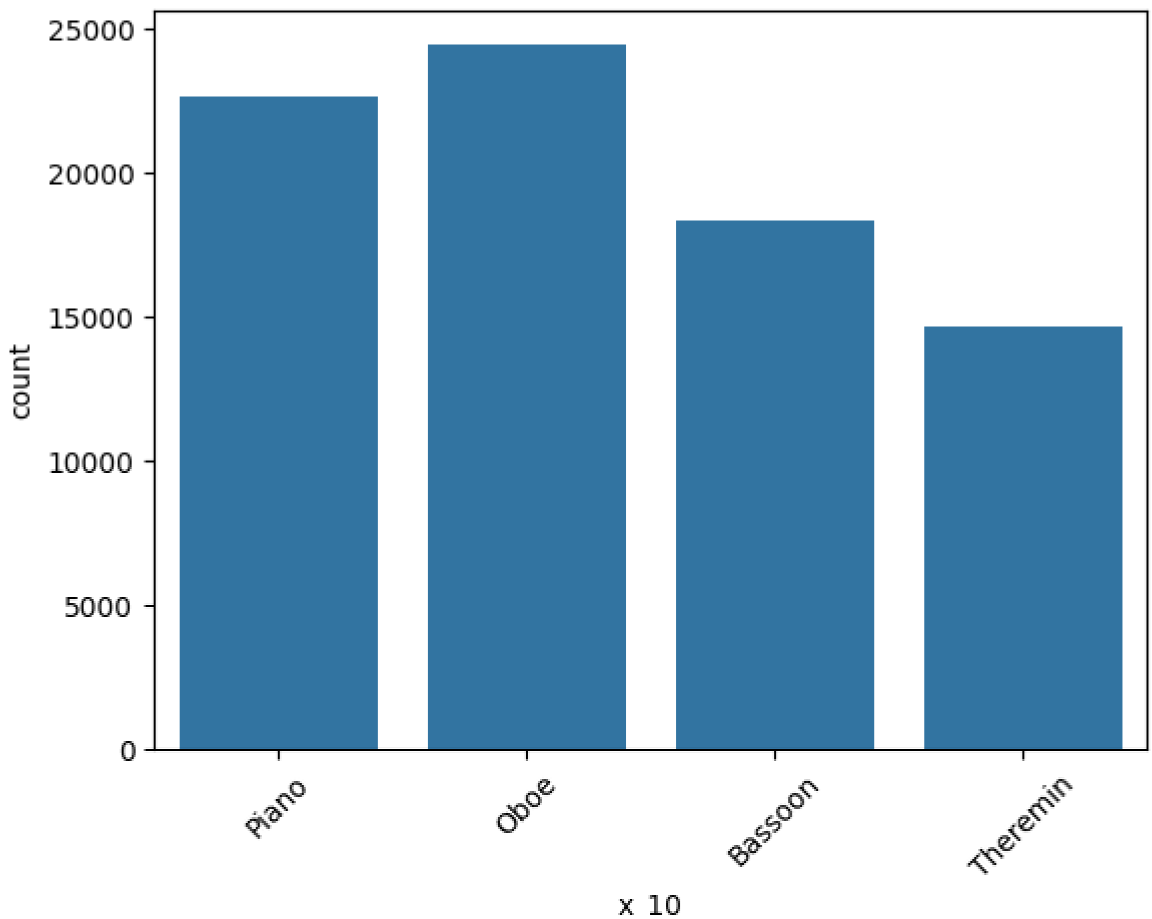

25000
20000
15000
10000
5000
0
count
Star
Polygon
Trapezoid
Triangle
Square
Circle
x_7

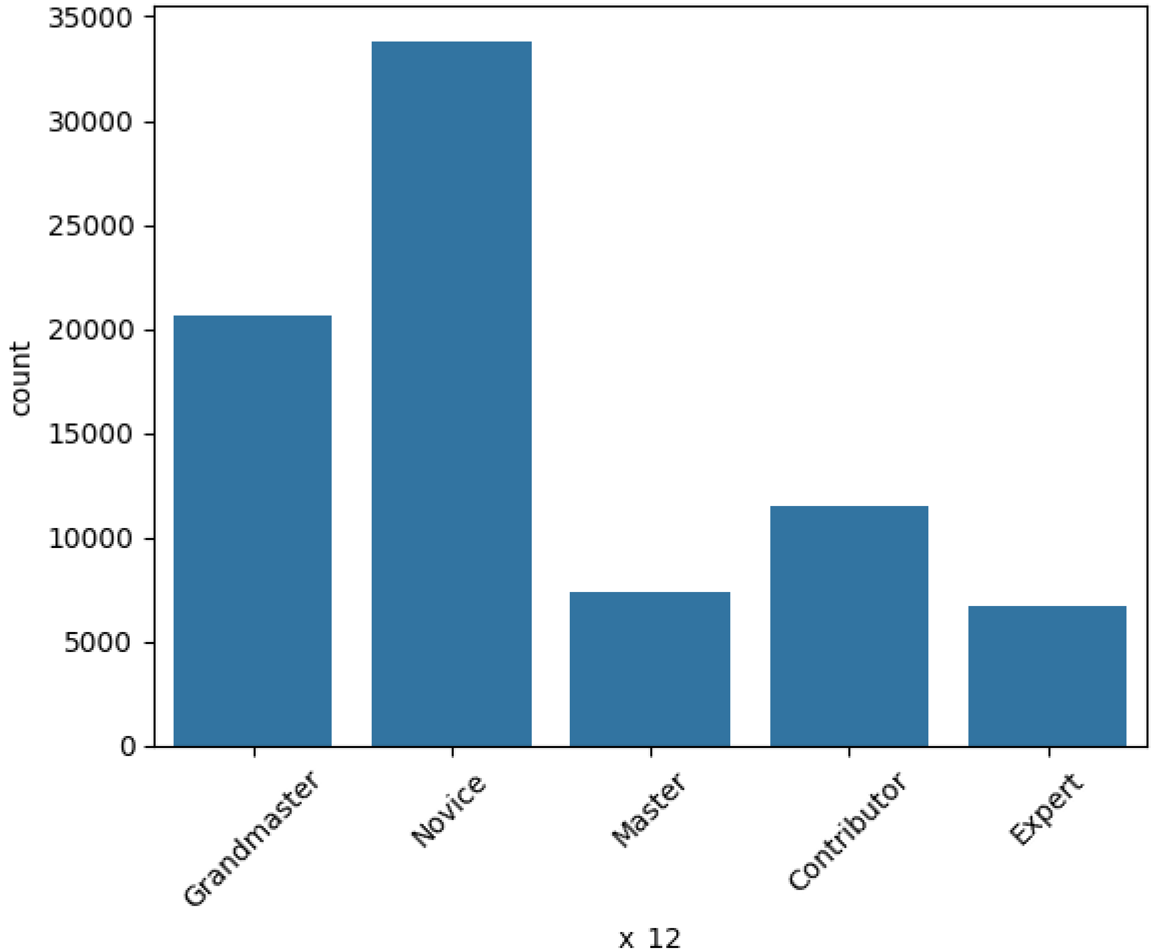


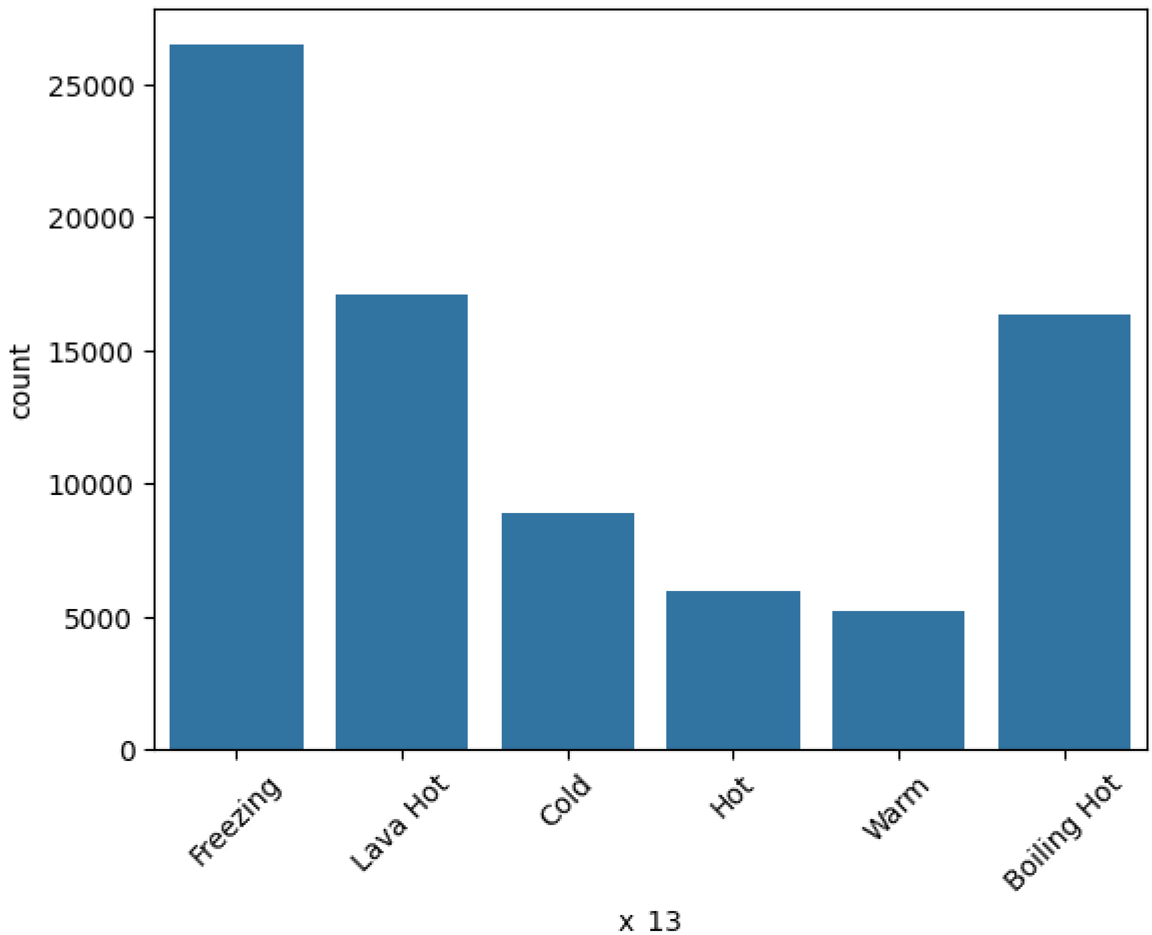
count
25000
20000
15000
10000
5000
0
Hamster
Lion
Cat
Dog
Axolotl
Snake
x_8

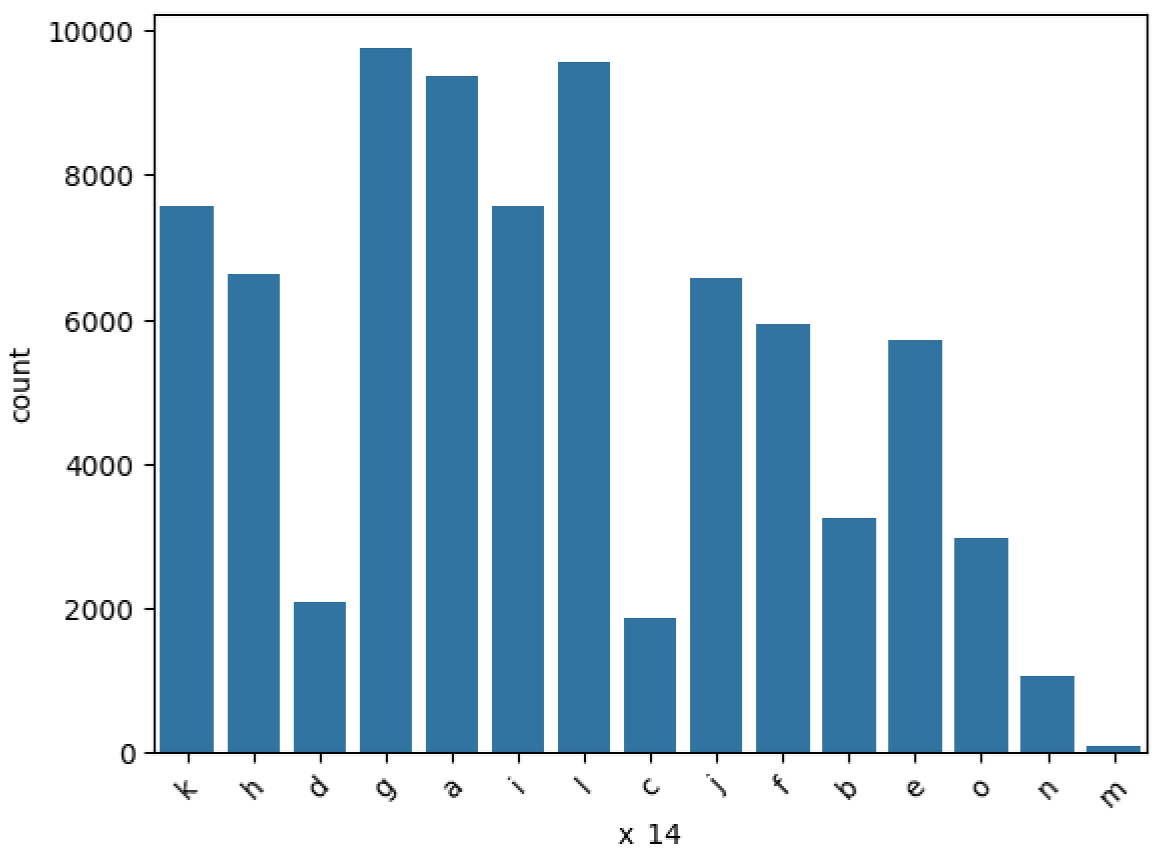


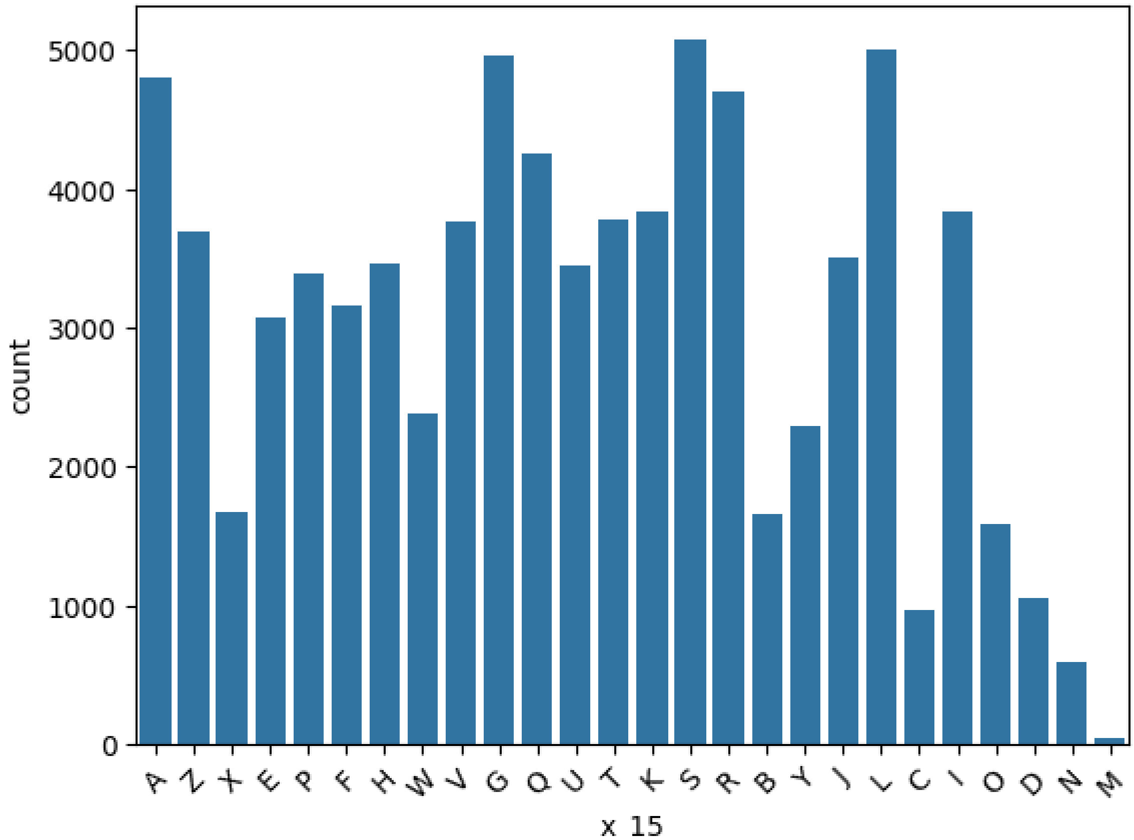
count
25000
20000
15000
10000
5000
0
Canada
Russia
China
Costa Rica
Finland
India
x_9


25000
20000
15000
10000
5000
0
count
Piano
Oboe
Bassoon
Theremin
x_10


35000
30000
25000
20000
15000
10000
5000
0
count
Grandmaster
Novice
Master
Contributor
Expert
x_12


25000
20000
15000
10000
5000
0
count
Freezing
Lava Hot
Cold
Hot
Warm
Boiling Hot
x_13


10000
8000
6000
4000
2000
0
count
k
h
d
g
a
i
l
c
j
f
b
e
o
n
m
x_14



Plot the target ratio of all categorical column. The target ratio is the average of the target for a specific value. For example in the given table,

Country	Target
Germany	1
France	0
Germany	0
France	1
Germany	1

Target Ratio for Germany = [Number of true targets under the label Germany/ Total Number of targets under the label Germany] which is 2/3 = 0.66.

Hence, the target ratios for the values would then be:

Values	Target Ratios
Germany	0.66
France	0.50

In [ ]:

```python
### Write your code here
for cat_col in categorical_columns:
    print(df.value_counts(subset=cat_col, normalize=True))
```

```
x_4
T    0.510713
F    0.489287
Name: proportion, dtype: float64
x_5
Y    0.640125
N    0.359875
Name: proportion, dtype: float64
x_6
Green    0.427237
Blue     0.317113
Red      0.255650
Name: proportion, dtype: float64
x_7
Trapezoid    0.336112
Square       0.166463
Star         0.154537
Circle       0.123450
Polygon      0.119487
Triangle     0.099950
Name: proportion, dtype: float64
x_8
Lion       0.337350
Cat        0.166475
Snake      0.152075
Dog        0.125300
Axolotl    0.120862
Hamster    0.097937
Name: proportion, dtype: float64
x_9
Russia        0.335500
Canada        0.166875
China         0.152537
Finland       0.123950
Costa Rica    0.120362
India         0.100775
Name: proportion, dtype: float64
x_10
Oboe        0.305063
Piano       0.283125
Bassoon     0.229137
Theremin    0.182675
Name: proportion, dtype: float64
x_12
Novice         0.422550
Grandmaster    0.257525
Contributor    0.143775
Master         0.092600
Expert         0.083550
Name: proportion, dtype: float64
x_13
Freezing       0.331175
Lava Hot       0.213812
Boiling Hot    0.204462
Cold           0.111412
Hot            0.074088
```

```
Warm           0.065050
Name: proportion, dtype: float64
x_14
g    0.121887
l    0.119650
a    0.117062
k    0.094575
i    0.094488
h    0.083000
j    0.082125
f    0.074150
e    0.071575
b    0.040525
o    0.037213
d    0.026187
c    0.023175
n    0.013312
m    0.001075
Name: proportion, dtype: float64
x_15
S    0.063450
L    0.062600
G    0.062075
A    0.059987
R    0.058763
Q    0.053112
I    0.048025
K    0.047962
T    0.047250
V    0.047012
Z    0.046187
J    0.043737
H    0.043338
U    0.043150
P    0.042400
F    0.039462
E    0.038375
W    0.029750
Y    0.028625
X    0.020913
B    0.020712
O    0.019912
D    0.013250
C    0.012050
N    0.007375
M    0.000525
Name: proportion, dtype: float64
```

Ordinal Data

There is special type of categorical variable which is called **Ordinal**. The ordinal variable has some order associated with it. Check which of the categorical columns are ordinal in nature. An example of ordinal values would (baby, child, teenager, adult, elder). From the plots generated earlier, determine which of the categorical columns are ordinal in nature.

Hint: Sorting the "Values" or "Target Ratios" in alphabetic order may reveal their ordinal nature.

In [ ]:
```
### Write your code here
print(f'Ordinals are : x_13 , x_12')
```

```
Ordinals are : x_13 , x_12
```

Data Encoding

Since the ML models can only deal with numerical data, we need to encode our dataset accordingly. Read up on how to encode the different types of variables and then perform the encoding.

- Encode binary labels as 0/1, if needed.
- To encode Categorical Variables, implement one-hot encoding from scratch.
- For Ordinal Variable, map the variables to numeric values. In case of the example given above, the mapping would be {baby:0, child:1, teenager:2, adult:3, elder:4}.
- Treat the time-series data (day/month) as cyclical features and encode them into two-dimensional sin-cos features. (Read on cyclical encoding of time).

Once the dataset is encoded, create a correlation heatmap using binary, ordinal and time-series variables (i.e. all variable except the one-hot encoded categorical variables).

In [ ]:
```
### Write your code here
df.head()
```

Out[ ]:

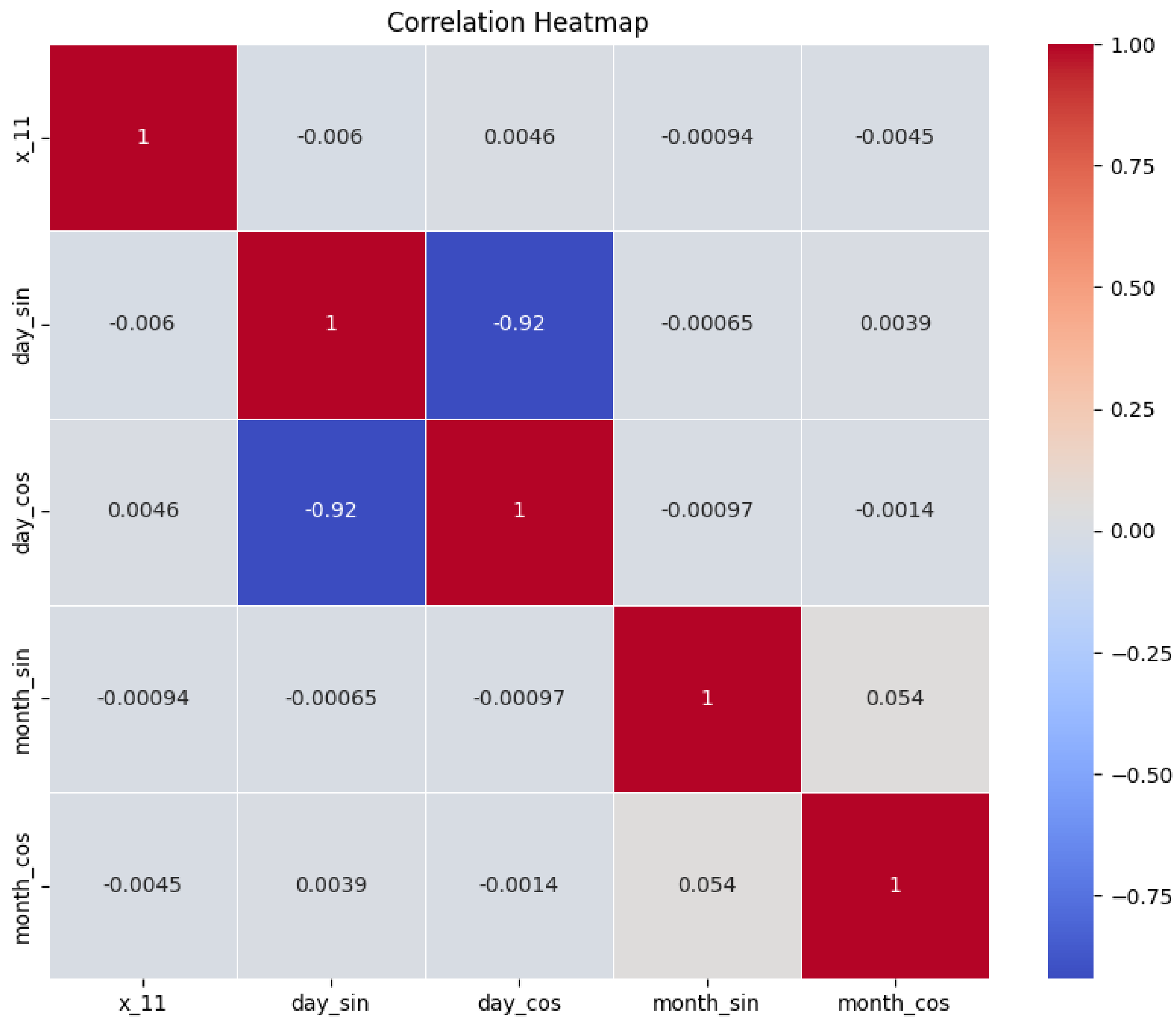
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_10	x_11	x_12
0	0	1	1	T	N	Blue	Star	Hamster	Canada	Piano	2	Grandmaster
1	0	0	1	F	Y	Red	Polygon	Lion	Russia	Oboe	2	Novice
2	0	0	1	F	Y	Green	Trapezoid	Cat	Russia	Bassoon	1	Novice
3	0	0	0	F	Y	Blue	Star	Cat	China	Bassoon	3	Grandmaster
4	0	0	1	T	Y	Green	Triangle	Lion	Costa Rica	Oboe	2	Master

In [ ]:
```
binary_cols = [
    "x_1", "x_2", "x_3"
]
for b_col in binary_cols:
    df[b_col] = df[b_col].map({'F': 0, 'T': 1})

ordinal_columns = [ "x_13" , "x_12" ]
x_13_mapping = {
    "Freezing": 0,
```

```
    "Cold": 1,
    "Warm": 2,
    "Hot": 3,
    "Boiling Hot": 4,
    "Lava Hot": 5
}
df["x_13"] = df["x_13"].map(x_13_mapping)
x_12_mapping = {
    "Novice": 0,
    "Contributor": 2,
    "expert": 3,
    "master": 4,
    "Grandmaster": 5
}
df["x_12"] = df["x_12"].map(x_12_mapping)

categorical_columns_except_ordinal = [
    col for col in categorical_column if col not in ordinal_columns
]

# This line is for one hot encoding - but it is automated
# df = pd.get_dummies(df, columns=categorical_columns_except_ordinal)

df['day_sin'] = np.sin(2 * np.pi * df['day'] / 31)
df['day_cos'] = np.cos(2 * np.pi * df['day'] / 31)
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)
# Drop the original 'day' and 'month' columns
df = df.drop(['day', 'month'], axis=1)
```

In [ ]:

```
selected_columns = [ 'x_11', 'day_sin', 'day_cos', 'month_sin', 'month_cos']

correlation_matrix = df[selected_columns].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



Binary Encoding

Another way of encoding categorical data is called Binary encoding. Read up on binary encoding and implement a function that takes in a column of categorical values and returns the encoded values. Also comment on the pros and/or cons of using Binary Encoding vs One-hot encoding.

In [ ]:
```
def binary_encode(column):
    # Step 1: Integer Encoding
    category_to_code = {category: code for code, category in enumerate(column.uniqu
    column_encoded = column.map(category_to_code)

    # Step 2: Binary Representation
    binary_representation = column_encoded.apply(lambda x: format(x, 'b'))

    # Step 3: Split binary representation into individual columns
    binary_columns = binary_representation.str.extractall(r'(\d)').unstack().fillna
    binary_columns.columns = ['Bit_' + str(col) for col in binary_columns.columns]

    return binary_columns
```

In [ ]:

```
# example of endoing to a two col from x_11
encoded_col = binary_encode(df["x_11"])
print(encoded_col)
```

```
      Bit_(0, 0) Bit_(0, 1)
0              0          0
1              0          0
2              1          0
3              1          0
4              0          0
...          ...        ...
79995          1          0
79996          1          0
79997          1          0
79998          0          0
79999          0          0

[80000 rows x 2 columns]
```

In [ ]:

```
### Write your code here
'''
This explaination is with the help of chatGPT
'''

'''
Binary Encoding Pros:

Memory Efficiency: Binary encoding uses fewer columns compared to one-hot encoding,
Maintains Information: Unlike one-hot encoding, binary encoding retains some ordina

Binary Encoding Cons:

Limited to Numeric Features: Binary encoding is primarily suited for nominal catego
Interpretability: The binary representation can make the resulting features less in

One-Hot Encoding Pros:

No Assumptions About Ordinality: One-hot encoding is suitable for nominal and ordin
Interpretability: Each column explicitly represents a category, making the resultin

One-Hot Encoding Cons:

Memory Inefficiency: One-hot encoding can significantly increase the dimensionality
Potential for Collinearity: One-hot encoding can introduce multicollinearity among
'''
```

Out[ ]:

```
'\nBinary Encoding Pros:\n\nMemory Efficiency: Binary encoding uses fewer columns 
compared to one-hot encoding, which can be especially beneficial when dealing with
a large number of categories.\nMaintains Information: Unlike one-hot encoding, bin
ary encoding retains some ordinal information in the encoded values because each b
it position represents a different power of 2.\n\nBinary Encoding Cons:\n\nLimited
to Numeric Features: Binary encoding is primarily suited for nominal categorical f
eatures, where there is no inherent order among categories. It may not be appropri
ate for ordinal categorical data.\nInterpretability: The binary representation can
make the resulting features less interpretable compared to one-hot encoding, which
explicitly identifies the category in each column.\n\nOne-Hot Encoding Pros:\n\nNo
Assumptions About Ordinality: One-hot encoding is suitable for nominal and ordinal
categorical features as it does not assume any inherent order among categories.\nI
nterpretability: Each column explicitly represents a category, making the resultin
g features highly interpretable.\n\nOne-Hot Encoding Cons:\n\nMemory Inefficiency:
One-hot encoding can significantly increase the dimensionality of the data, leadin
g to higher memory usage when there are many categories.\nPotential for Collineari
ty: One-hot encoding can introduce multicollinearity among the binary columns, whi
ch may not be ideal for certain machine learning models.\n'
```