In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sqlm.autobook import sqbm
import numpy as np
import matplotlib.pyplot as plt

import numpy as np
import matplotlib.pyplot as plt



import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd



import numpy as np



import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pyod
```

In [2]:
```
def BARPLOT(__data , __targit) :
    plt.figure(figsize=(12, 6))
    __data[__targit].value_counts()
    sns.barplot(data=__data, x=__targit, y= __data[__targit].map(count)  ,  order=count.index)
    plt.xticks(rotation=45)
    plt.xlabel(__targit)
    plt.ylabel("Count")
    plt.title("Count of Each "+__targit+"")
    plt.show()

def refill_table(__table): :
    def fill_table(__row) :
        try :
            str = __row["Exercice"] - __row["Date Survenance"]
            return(table[2][__row['Date Survenance']+1] == __row['Reglement']
        except :
            print(nbf)
            print(__row['Exercice'])
            print(__row['Date Survenance'])
            print(__row['Reglement'] )
            print()
        return fill_table
```

In [3]:
```
df = pd.read_csv("Base-de-Données-MATGHARCE.csv");
df["Date Survenance"] = pd.to_datetime(df["Date Survenance"], format="%d/%m/%Y").dt.year
df["Reglement"] = df["Reglement"].str.replace(",", ".", regex=True).astype(float)
df["Exercice"] = df["Exercice "].astype(int)
df.info()
df
```

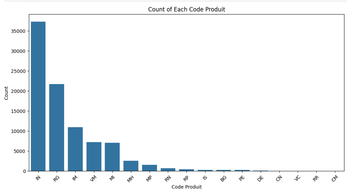
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90223 entries, 0 to 90228
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Exercice              90223 non-null  int64
 1   Branche               90223 non-null  object
 2   Code Produit          90223 non-null  object
 3   Désignation Produit   90223 non-null  object
 4   Sous-Branche          90223 non-null  object
 5   Date Survenance       90223 non-null  int32
 6   Reglement             90223 non-null  float64
 7   Exercice              90223 non-null  int64
dtypes: float64(1), int32(1), int64(2), object(4)
memory usage: 5.2+ MB
```

Out[3]:

	Exercice	Branche	Code Produit	Désignation Produit	Sous-Branche	Date Survenance	Reglement	Exercice
0	2021	Risques Industriels	IN	Incendie Risques Annexes	Incendie	2019	3.056632e+09	2021
1	2018	Risques Industriels	IN	Incendie Risques Annexes	Incendie	2017	2.972783e+09	2018
2	2019	Risques Industriels	IN	Incendie Risques Annexes	Incendie	2017	2.611488e+09	2019
3	2016	Risques Industriels	IN	Incendie Risques Annexes	Incendie	2015	2.547476e+09	2016
4	2022	Risques Industriels	IN	Incendie Risques Annexes	Incendie	2021	2.057632e+09	2022
...	...	...	...	...	...	...	...	...
90224	2014	Risques Industriels	RP	RC Professionnelle	Responsabilité Civile	2013	1.185982e+03	2014
90225	2014	Risques Industriels	RP	RC Professionnelle	Responsabilité Civile	2013	1.114240e+03	2014
90226	2014	Risques Industriels	MH	Multirisques Habitation	Risque simple	2013	7.000000e+02	2014
90227	2019	Risques Industriels	MH	Multirisques Habitation	Risque simple	2017	5.577150e+02	2019
90228	2015	Risques Industriels	MH	Multirisques Habitation	Risque simple	2013	4.000000e+02	2015

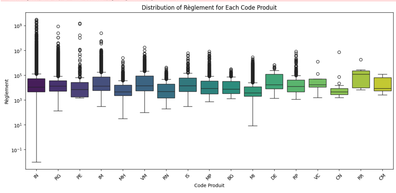
90229 rows × 8 columns

In [4]:
```
BARPLOT(df, "Code Produit")
```



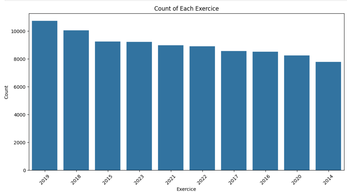
In [5]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 6))

sns.boxplot(data=df, x="Code Produit", y="Reglement", palette="viridis")

plt.xticks(rotation=45)
plt.xlabel("Code Produit")
plt.ylabel("Reglement")
plt.title("Distribution of Reglement for Each Code Produit")

plt.yscale("log")
plt.show()
```

```
/tmp/ipykernel_577325/2570742108.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x="Code Produit", y="Reglement", palette="viridis")
```

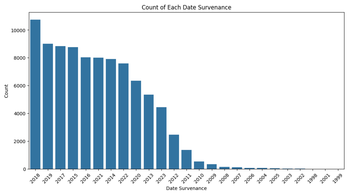


In [6]:
```

```

In [7]:
```
BARPLOT(df, "Exercice")
```



In [8]:
```
BARPLOT(df, "Date Survenance")
```

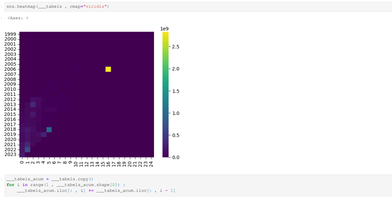


In [9]:
```
# df = df[df["Code Produit"] == "RD" ]
df = df[df["Code Produit"] != "CM" ]
```

In [10]:
```
__table1 = pd.DataFrame()
__min = df["Date Survenance"].min()
__max = df["Date Survenance"].max()
print("min : " , __min)
print("max : " , __max)
for i in range(__max - __min):
    __table1["{}".__min  + i ] ="{} for i in range(__max - __min)}
__table1 = __table1
df.apply(refill_table(__table1) , axis=1)
```

In [11]:
```
sns.heatmap(__table1 , cmap="viridis")
```

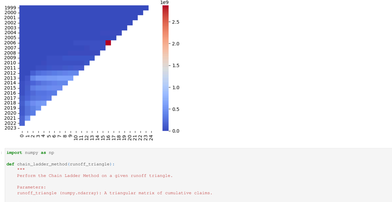
Out[11]:
```
<Axes: >
```



In [12]:
```
__table1_arum = __table1.copy()
for i in range(1 , __table1_arum.shape[0]) :
    __table1_arum.iloc[: , i] += __table1_arum.iloc[: , i - 1]

__table1_arum_test = __table1_arum.copy()
for i in range(__table1_arum.shape[0]) : __table1_arum.iloc[i,-i+1:] = np.nan
```

In [13]:
```
sns.heatmap(__table1_arum , cmap="coolwarm" )
```

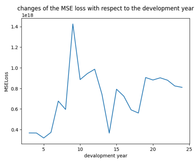
Out[13]:
```
<Axes: >
```



In [14]:
```
import numpy as np

def chain_ladder_method(runoff_triangle):
    """
    Perform the Chain Ladder Method on a given runoff triangle.

    Parameters:
    runoff_triangle (numpy.ndarray): A triangular matrix of cumulative claims.

    Returns:
    completed_triangle (numpy.ndarray): The completed runoff triangle.
    ave_loss (float): The Mean Squared Error (MSE) loss.
    """
    n = runoff_triangle.shape[0]  # Number of accident years
    completed_triangle = runoff_triangle.copy()

    # Step 1: Calculate Development Factors (Link Ratios)
    development_factors = []
    for j in range(n - 1):
        numerator = np.sum(completed_triangle[:n - j - 1, j + 1])
        denominator = np.sum(completed_triangle[:n - j - 1, j])
        development_factors.append(numerator / denominator)

    # Step 2: Project Ultimate Claims
    for i in range(1, n):
        for j in range(n - i, n):
            completed_triangle[i, j] = completed_triangle[i, j - 1] * development_factors[j - 1]

    # Step 3: Calculate MSE Loss
    actual_values = []
    predicted_values = []
    for i in range(1, n):
        for j in range(n - i, n):
            actual_values.append(runoff_triangle[i, j])
            predicted_values.append(completed_triangle[i, j])


    return completed_triangle

def get_other_diagonal(matrix):
    return np.flip(np.fliplr(matrix).diagonal())

result = {}

for __index in range(1 , __table1_arum.shape[0]) :
    tmp = __table1_arum.values[:__index - 1 , -__index:-1]
    result[__index] = {
        "preds" : chain_ladder_method(tmp),
        "loss" : np.nan
    }
```

In [15]:
```
def MSE(y , y_hat) :
    return np.mean(np.pow(y - y_hat , 2))
y = get_other_diagonal(__table1_arum_test.values)
for __index in  range(1 , __table1_arum.shape[0]) :
    if __index == 1 : continue
    y_hat = get_other_diagonal(result[__index]["preds"])
    result[__index]["loss"] = MSE(y_hat , y[:len(y_hat)])
result = pd.DataFrame(result).T
```

In [16]:
```
sns.lineplot(data=result , y="loss" , x=result.index )
plt.xlabel("Development year")
plt.ylabel("MSELoss ")
plt.title("Changes of the MSE loss with respect to the development year")
plt.show()
```


changes of the MSE loss with respect to the development year
1e18
1.4
1.2
1.0
0.8
0.6
0.4
MSELoss
5
10
15
20
25
development year

In [ ]: