Klasifikasi Model Dengan Logistic Regression Dan KNN pada Dataset Iris TASK 1

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Machine Learning Week 1 Task 1 - Azmi Taqiuddin Syah - 1103213078

Import libraries Yang Dibutuhkan

Library yang di butuhkan Sebagai Berikut

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn import model_selection, preprocessing, feature_selection, ensemble, linear_model, metrics, decomposition
```

```
column_names = ['tahun'] + [f'x{i}' for i in range(1, 91)]
```

```
dataset = "/content/sample_data/RegresiUTSTelkom.csv"
data = pd.read_csv(dataset,names=column_names)
```

```
data.head()
```

	tahun	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x81	x82	x83
0	2001	49.94357	21.47114	73.07750	8.74861	-17.40628	-13.09905	-25.01202	-12.23257	7.83089	...	13.01620	-54.40548	58.99367
1	2001	48.73215	18.42930	70.32679	12.94636	-10.32437	-24.83777	8.76630	-0.92019	18.76548	...	5.66812	-19.68073	33.04964
2	2001	50.95714	31.85602	55.81851	13.41693	-6.57898	-18.54940	-3.27872	-2.35035	16.07017	...	3.03800	26.05866	-50.92779
3	2001	48.24750	-1.89837	36.29772	2.58776	0.97170	-26.21683	5.05097	-10.34124	3.55005	...	34.57337	-171.70734	-16.96705
4	2001	50.97020	42.20998	67.09964	8.46791	-15.85279	-16.81409	-12.48207	-9.37636	12.63699	...	9.92661	-55.95724	64.92712

5 rows × 91 columns

```
data.describe()
```

	tahun	x1	x2	x3	x4	x5	x6	x7	
count	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.000000	515345.0
mean	1998.397082	43.387126	1.289554	8.658347	1.164124	-6.553601	-9.521975	-2.391089	-1.7
std	10.931046	6.067558	51.580351	35.268585	16.322790	22.860785	12.857751	14.571873	7.9
min	1922.000000	1.749000	-337.092500	-301.005060	-154.183580	-181.953370	-81.794290	-188.214000	-72.5
25%	1994.000000	39.954690	-26.059520	-11.462710	-8.487500	-20.666450	-18.440990	-10.780600	-6.4
50%	2002.000000	44.258500	8.417850	10.476320	-0.652840	-6.007770	-11.188390	-2.046670	-1.7
75%	2006.000000	47.833890	36.124010	29.764820	8.787540	7.741870	-2.388960	6.508580	2.9
max	2011.000000	61.970140	384.065730	322.851430	335.771820	262.068870	166.236890	172.402680	126.7

8 rows × 91 columns

```
data.tail()
```

	tahun	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x81	x82	
515340	2006	51.28467	45.88068	22.19582	-5.53319	-3.61835	-16.36914	2.12652	5.18160	-8.66890	...	4.81440	-3.75991	-3
515341	2006	49.87870	37.93125	18.65987	-3.63581	-27.75665	-18.52988	7.76108	3.56109	-2.50351	...	32.38589	-32.75535	-6
515342	2006	45.12852	12.65758	-38.72018	8.80882	-29.29985	-2.28706	-18.40424	-22.28726	-4.52429	...	-18.73598	-71.15954	-12
515343	2006	44.16614	32.38368	-3.34971	-2.49165	-19.59278	-18.67098	8.78428	4.02039	-12.01230	...	67.16763	282.77624	-
515344	2005	51.85726	59.11655	26.39436	-5.46030	-20.69012	-19.95528	-6.72771	2.29590	10.31018	...	-11.50511	-69.18291	6

5 rows × 91 columns

```
# View NaN values
print("Rows with NaN values:")
print(data[data.isna().any(axis=1)])

# View duplicates
print("\nDuplicate rows:")
print(data[data.duplicated()])

# Drop duplicates
data = data.drop_duplicates()
```

```
[0 rows x 91 columns]

Duplicate rows:
        tahun        x1        x2        x3        x4        x5        x6  \
5551     1973  41.22353   3.20571  43.66712   7.81090 -27.93823  -2.67931
9941     1999  46.98706  38.24010  22.51761   7.24891  -3.88296  -5.25372
9942     1998  43.28314  25.37917   6.80249  16.41132 -14.76744   3.84884
10071    2006  45.88913   3.50835 -16.79630   2.95203 -12.71814 -10.46804
17330    2008  31.59176 -43.16626 -53.11768  -7.08228  36.23470 -25.61305
...       ...       ...       ...       ...       ...       ...       ...
500568   1988  38.97271  35.64567 -57.35630 -12.21976 -46.33510  -2.40727
505982   2002  41.71236 -74.26705  -1.79009   2.35874 -41.19781   4.17658
507253   2005  45.80278  31.58951  -7.59075 -14.01903 -18.81706  -7.66548
507479   2008  43.46480 -10.83611  31.90017 -13.44218 -11.36997   2.02308
508200   1992  39.18044 -23.14181   6.16137   2.35418  -9.26563  -6.34051

              x7        x8        x9  ...       x81        x82        x83  \
5551   -18.24996 -10.16836  10.21118  ...  20.80983  -46.42137  -33.63520
9941    12.18594   0.18605  12.75414  ...  10.74268  -50.30651   -7.37361
9942    11.74961   2.87696   5.98809  ...   8.24654  -76.93471 -130.44354
10071    0.08869  16.29474  -0.49840  ...  12.71373 -163.83204 -156.44085
17330  -31.75173 -11.16459  22.02261  ...  27.60406 -256.30083  -11.62091
...          ...       ...       ...  ...       ...        ...        ...
500568 -13.88561  -1.84301   0.86373  ...  11.23003 -129.69128  -66.70936
505982 -13.86373   2.79708  -1.92353  ...  -1.52568 -341.90923   77.86735
507253  -9.95065  -2.58846   4.83188  ...  63.15197   10.50641  247.87142
507479 -27.88310  -4.38305  20.88566  ...  52.10270   74.33104  128.10386
508200  -3.67249 -24.84162  -2.77967  ...  20.63775  -40.43440  -16.27189

              x84       x85        x86        x87       x88        x89  \
5551    -14.65970   1.63648   52.43969  -17.95543  -2.48364 -104.09383
9941     95.50208   5.34527  -22.54009   95.45694   1.57117 -121.46786
9942     49.00444   5.17171  -22.77865  196.49517   9.08196  -81.65460
10071   165.20790   1.67669  -64.35191   94.78478  -5.55255   23.30911
17330   119.28601  17.95582   51.36605 -116.69768   4.80931  114.35652
...           ...       ...        ...        ...       ...        ...
500568   13.65987  -5.99570  139.55599 -227.12107  11.44736  -82.32164
505982  208.76815 -19.98203   64.68677 -147.48197  14.85343  -76.69082
507253   66.71739  12.37958    6.93733  104.76817  22.78140 -151.14191
507479   35.25644  14.86020  164.93570   25.08485   9.54500 -111.64292
508200   48.69949  10.15451  111.65179  -47.06901   1.20738  129.55055

             x90
5551     6.01573
9941    -8.07735
9942    -2.64752
10071   -6.75154
17330  -11.11994
...          ...
500568  -5.94241
505982 -16.05562
507253  19.05184
507479  16.17438
508200   3.48504

[214 rows x 91 columns]
```

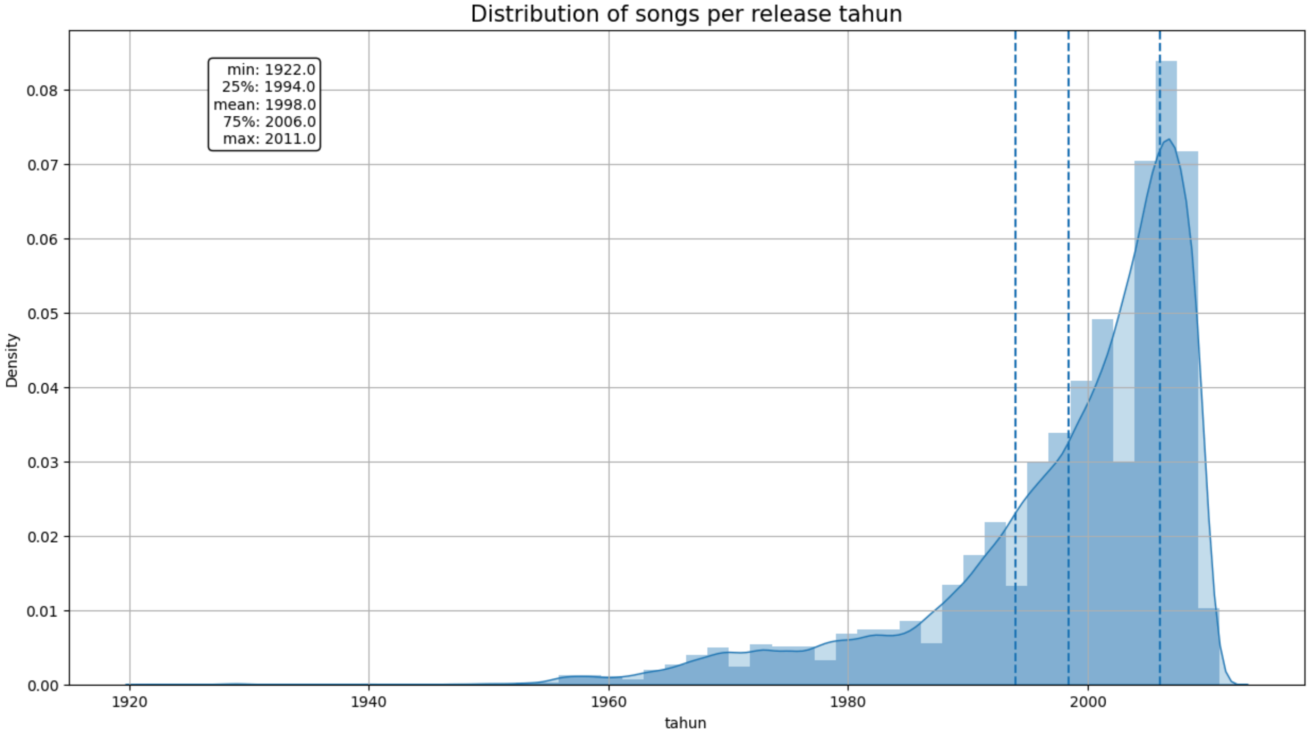
```
print("\nData after dropping duplicates:")
print(data.tail())
```

```
Data after dropping duplicates:
        tahun        x1        x2        x3       x4        x5        x6  \
515340   2006  51.28467  45.88068  22.19582 -5.53319  -3.61835 -16.36914
515341   2006  49.87870  37.93125  18.65987 -3.63581 -27.75665 -18.52988
515342   2006  45.12852  12.65758 -38.72018  8.80882 -29.29985  -2.28706
515343   2006  44.16614  32.38368  -3.34971 -2.49165 -19.59278 -18.67098
515344   2005  51.85726  59.11655  26.39436 -5.46030 -20.69012 -19.95528

              x7        x8        x9  ...       x81        x82        x83  \
515340   2.12652   5.18160  -8.66890  ...   4.81440   -3.75991  -30.92584
515341   7.76108   3.56109  -2.50351  ...  32.38589  -32.75535  -61.05473
515342 -18.40424 -22.28726  -4.52429  ... -18.73598  -71.15954 -123.98443
515343   8.78428   4.02039 -12.01230  ...  67.16763  282.77624   -4.63677
515344  -6.72771   2.29590  10.31018  ... -11.50511  -69.18291   60.58456

              x84       x85       x86        x87       x88       x89       x90
515340   26.33968  -5.03390  21.86037 -142.29410   3.42901 -41.14721 -15.46052
515341   56.65182  15.29965  95.88193  -10.63242  12.96552  92.11633  10.88815
515342  121.26989  10.89629  34.62409 -248.61020  -6.07171  53.96319  -8.09364
515343  144.00125  21.62652 -29.72432   71.47198  20.32240  14.83107  39.74909
515344   28.64599  -4.39620 -64.56491  -45.61012  -5.51512  32.35602  12.17352

[5 rows x 91 columns]
```

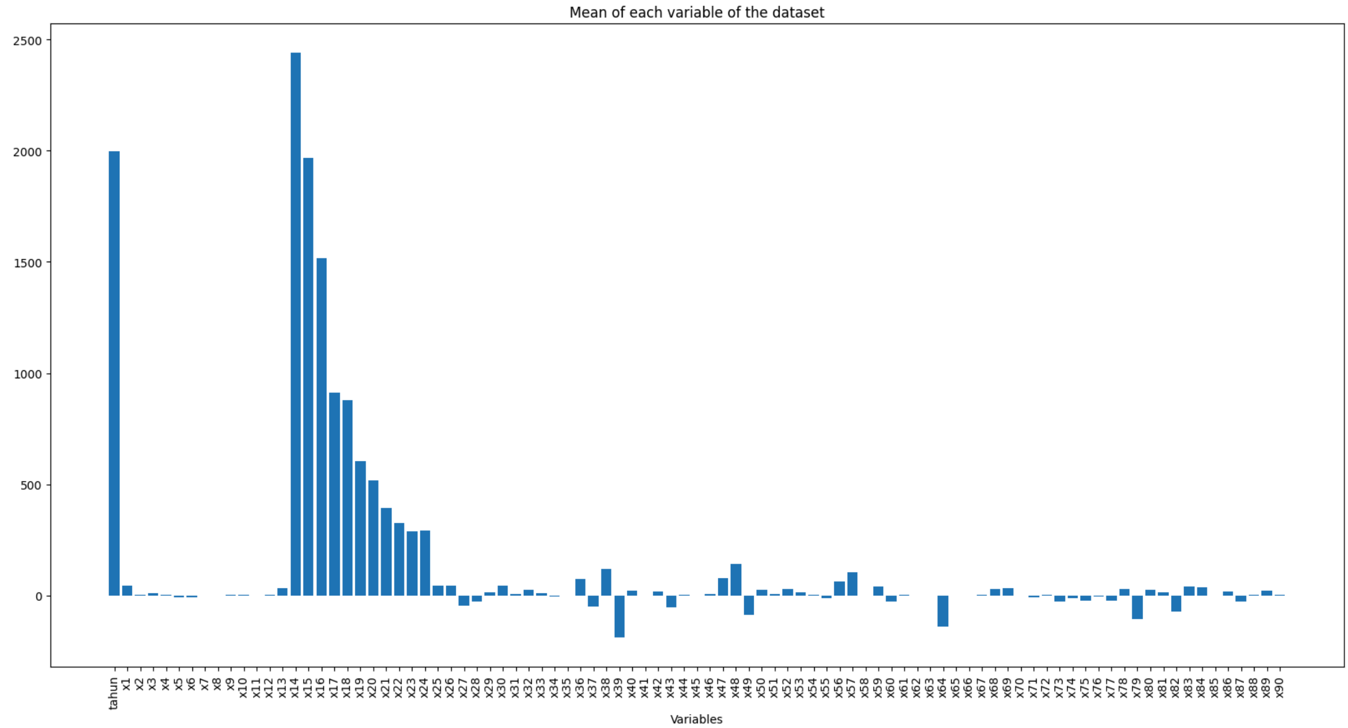
EDA

```
fig, ax = plt.subplots(figsize=(15,8))

ax.set_title('Distribution of songs per release tahun', fontsize=15)
variable = data['tahun']

sns.distplot(variable, hist=True, kde=True, kde_kws={"shade": True}, ax=ax)
des = data['tahun'].describe()
ax.axvline(des["25%"], ls='--')
ax.axvline(des["mean"], ls='--')
ax.axvline(des["75%"], ls='--')
ax.grid(True)

des = round(des).apply(lambda x: str(x))
box = '\n'.join(("min: "+des["min"], "25%: "+des["25%"], "mean: "+des["mean"], "75%: "+des["75%"], "max: "+des["max"]))
ax.text(0.20, 0.95, box, transform=ax.transAxes, fontsize=10, va='top', ha="right", bbox=dict(boxstyle='round', facecolor='white', alpha
```

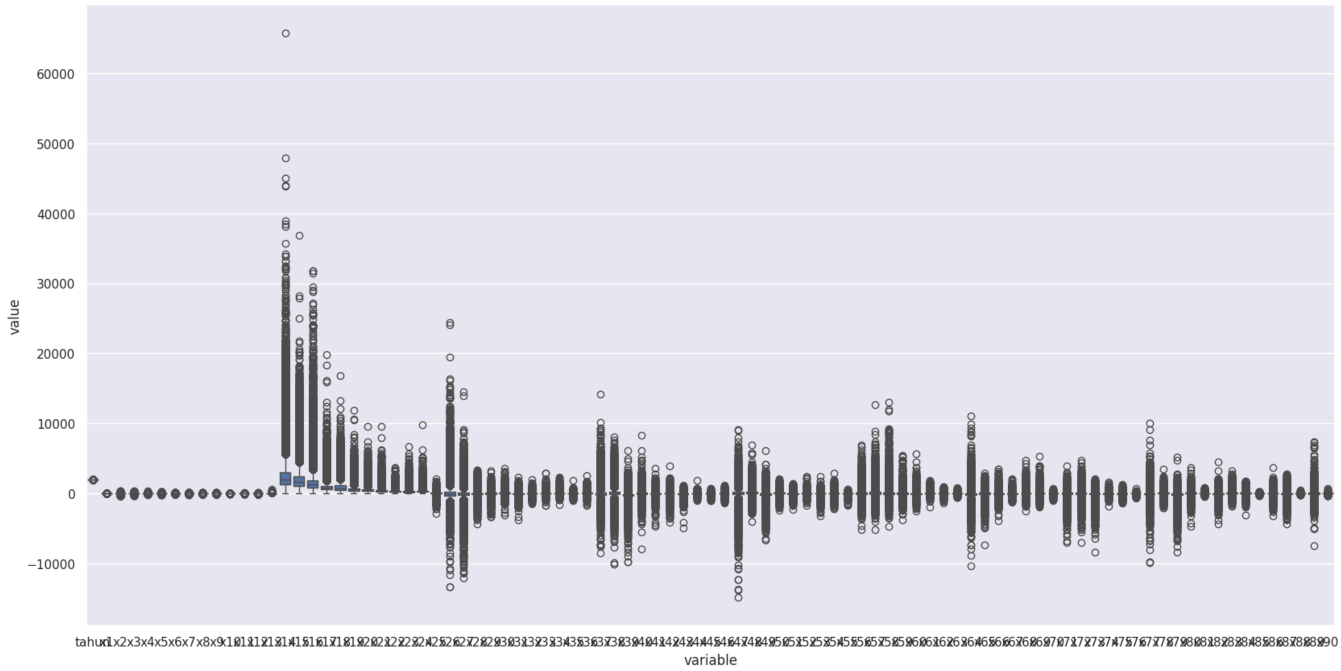
```
<ipython-input-10-05c37db356bd>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(variable, hist=True, kde=True, kde_kws={"shade": True}, ax=ax)
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2496: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  kdeplot(**{axis: a}, ax=ax, color=kde_color, **kde_kws)
```



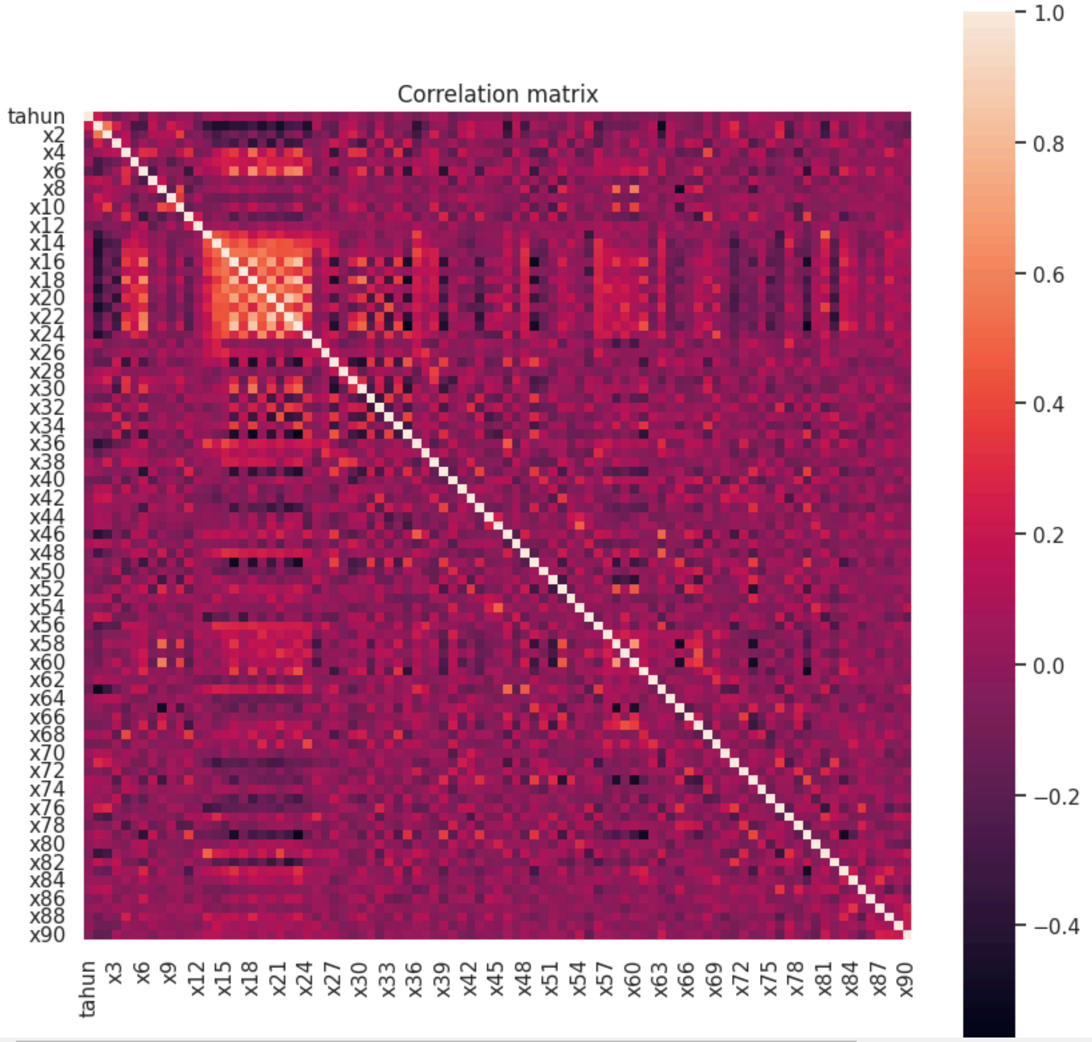

```
fig, ax = plt.subplots(figsize=(20,10))
ax.bar(data.columns.map(str), data.mean().values)
ax.set_xlabel('Variables')
plt.xticks(rotation = 90)
plt.title('Mean of each variable of the dataset');
```




```
data_melted = pd.melt(data)

sns.set(rc={"figure.figsize":(20, 10)}) #width=3, #height=4
sns.boxplot(x='variable', y='value', data=data_melted);
```

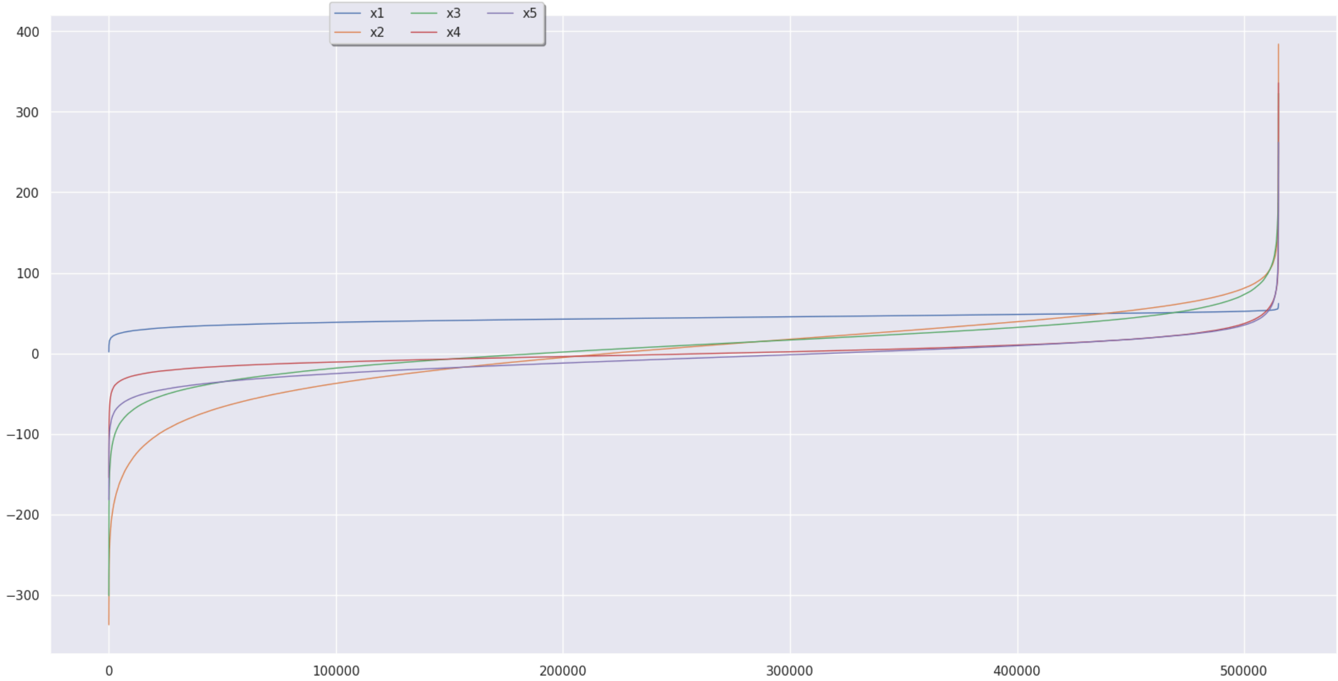



```
# Correlation between the release tahun and features
corr = data.corr()
fig, ax = plt.subplots(figsize=(10,10))
plt.title("Correlation matrix")
sns.heatmap(corr, square=True);
```



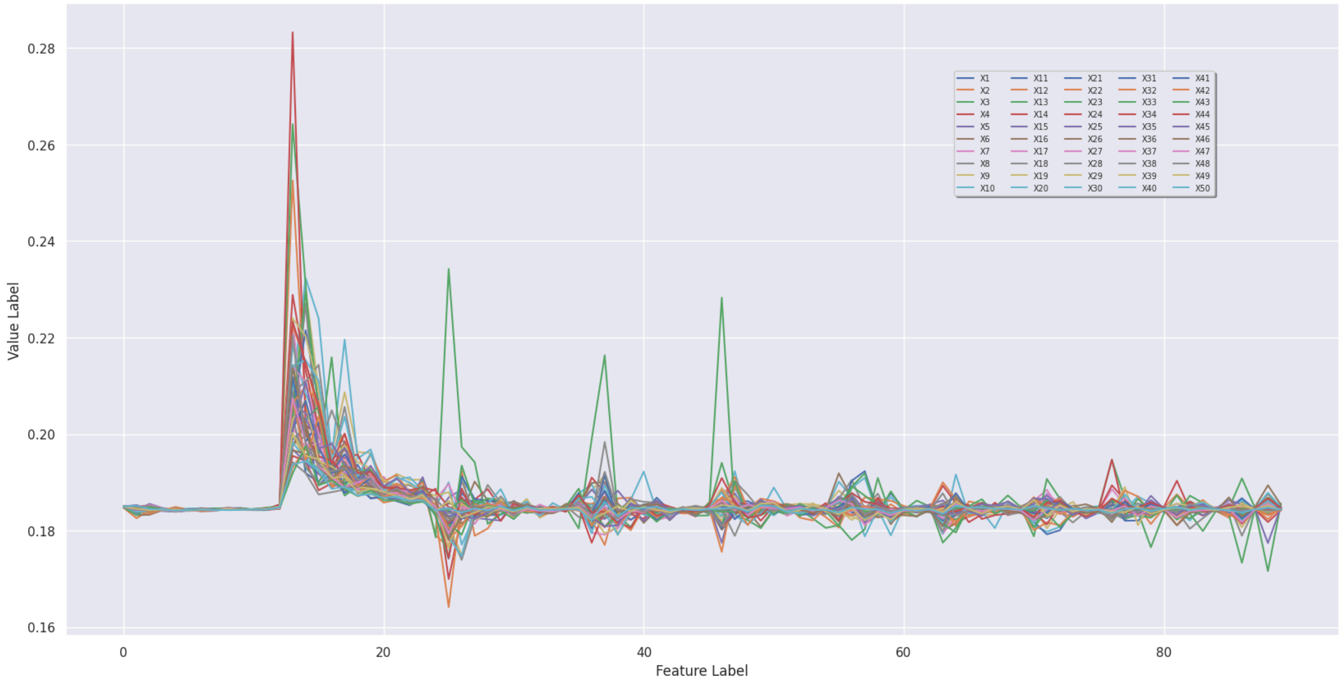

```
for t in column_names[1:6]:
    y = data[t].to_numpy()
    plt.plot(sorted(y), label=t, linewidth=1)

plt.legend(loc='upper center', bbox_to_anchor=(0.3, 1.03), ncol=3, fancybox=True, shadow=True)
```

```
<matplotlib.legend.Legend at 0x7cd6ffd70f10>
```

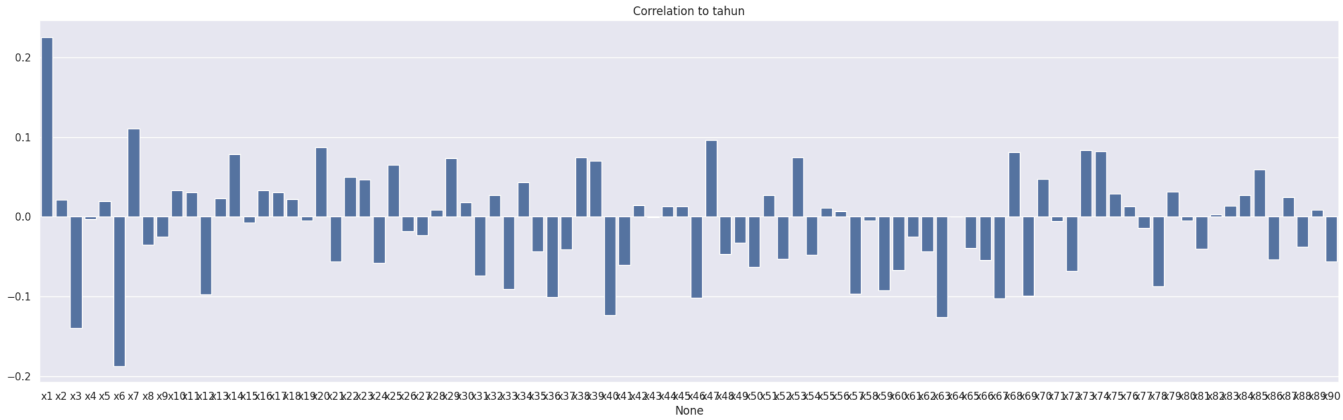



```
X = data.iloc[:, 1:].to_numpy()
X = (X - X.min()) / (X.max() - X.min())


for i in range(1, 51):
    plt.plot(X[i], label='X' + str(i))

plt.xlabel("Feature Label")
plt.ylabel("Value Label")
plt.legend(loc='upper center', bbox_to_anchor=(0.8, 0.9), ncol=5, fancybox=True, shadow=True, fontsize=7)
```

```
<matplotlib.legend.Legend at 0x7cd700d62f80>
```




```
fig, ax = plt.subplots(figsize=(25,7))
sns.barplot(x=corr['tahun'][1:].index, y=corr['tahun'][1:].values)
plt.title('Correlation to tahun');
```



Splitting the data set

We split the data set into a training and a testing data set, before applying any pre-processing of the data,as it would otherwise put information from the testing set into the training set.

We follow the instruction given on the data set page on the UCI Machine Learning Repository and split the data set this way :

train: first 463,715 examples

test: last 51,630 examples

Which according to the website "avoids the 'producer effect' by making sure no song from a given artist ends up in both the train and test set."

```
data_train=data.iloc[:463715,:]
print(data_train.shape)
data_test=data.iloc[463715:,:]
print(data_test.shape)
```

```
(463715, 91)
(51416, 91)
```

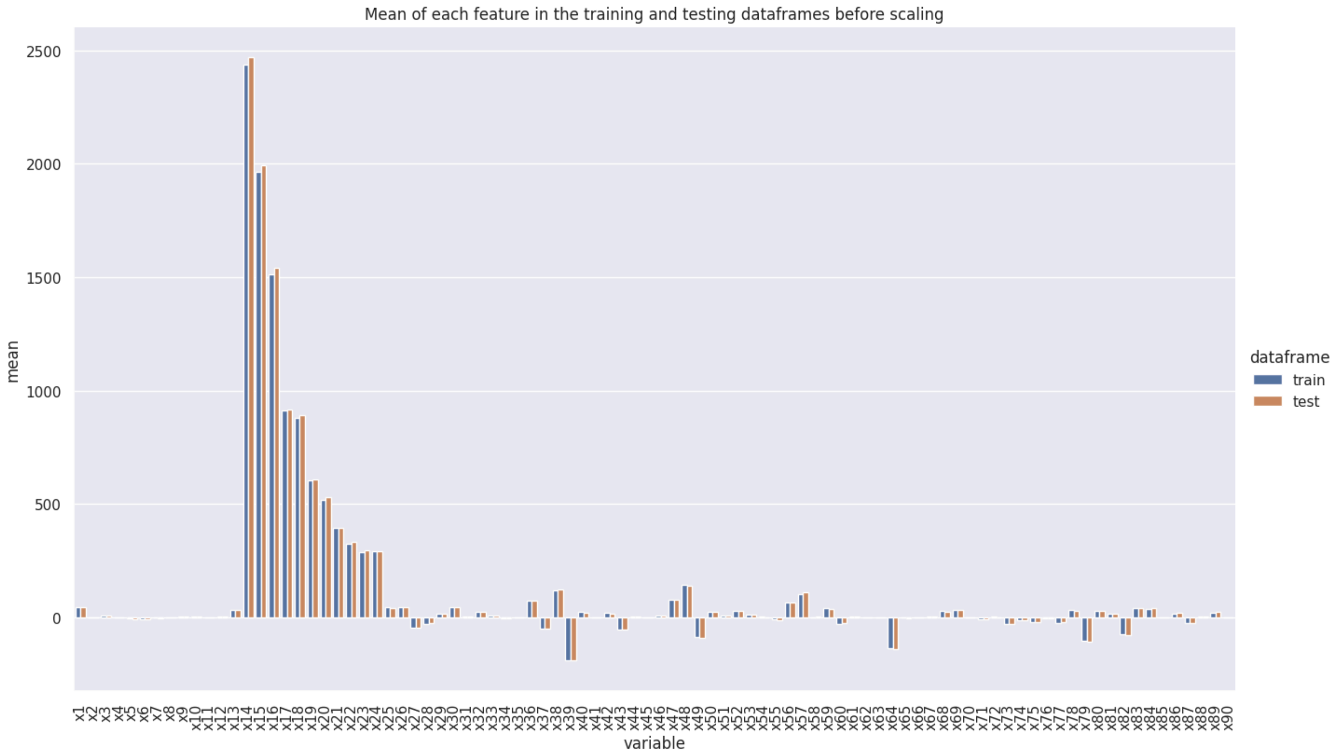
```
data_train.describe()
```

	tahun	x1	x2	x3	x4	x5	x6	x7	
count	463715.000000	463715.000000	463715.000000	463715.000000	463715.000000	463715.000000	463715.000000	463715.000000	463715.0
mean	1998.386492	43.385407	1.253786	8.651491	1.130590	-6.513477	-9.566442	-2.383797	-1.7
std	10.940319	6.079760	51.612880	35.264853	16.334058	22.854770	12.836177	14.580237	7.9
min	1922.000000	1.749000	-337.092500	-301.005060	-154.183580	-181.953370	-81.794290	-188.214000	-72.5
25%	1994.000000	39.957510	-26.161450	-11.441550	-8.514270	-20.635820	-18.469000	-10.774800	-6.4
50%	2002.000000	44.262120	8.361490	10.472720	-0.691060	-5.993610	-11.209400	-2.047330	-1.7
75%	2006.000000	47.833710	36.136950	29.744940	8.756665	7.745720	-2.423955	6.516025	2.9
max	2011.000000	61.970140	384.065730	322.851430	289.527430	262.068870	119.815590	172.402680	105.2

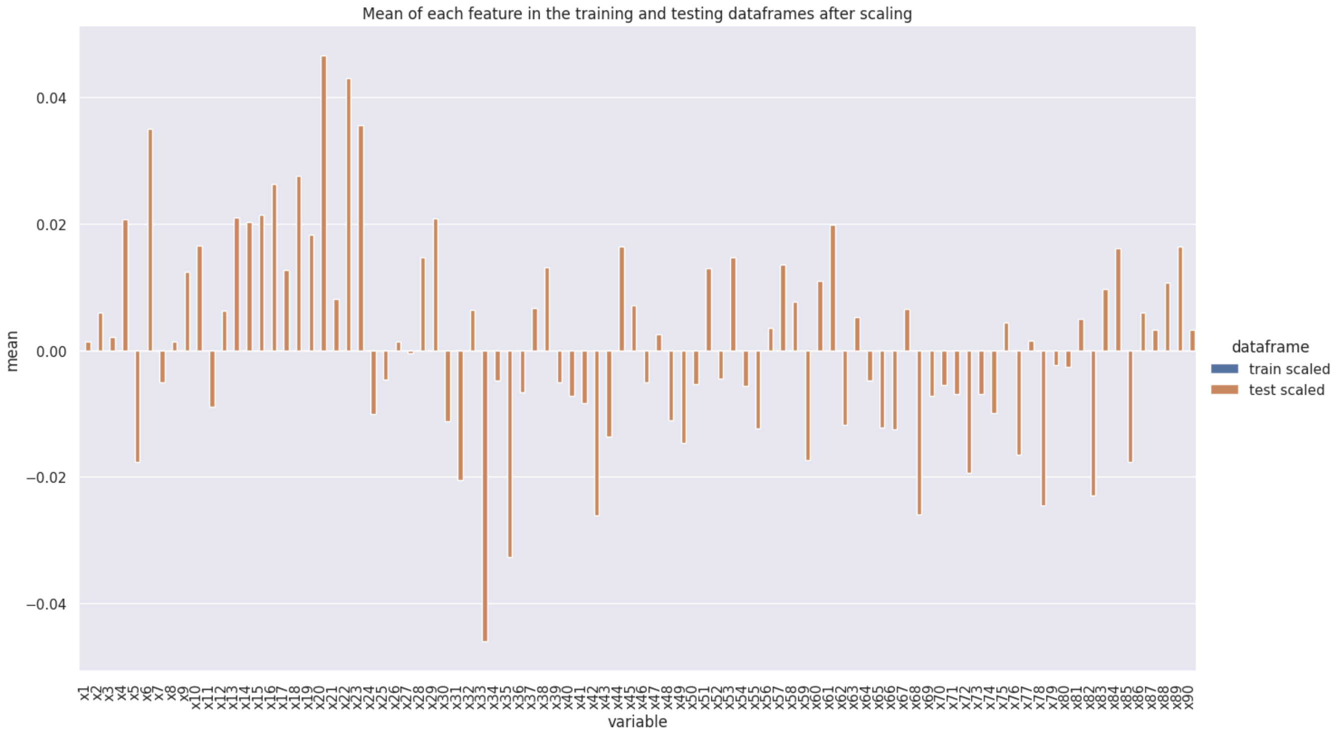
8 rows × 91 columns

```
# create the scaler
ss = preprocessing.StandardScaler()

# create new dataframes to keep the non scaled ones
data_train_s=data_train.copy()
data_test_s=data_test.copy()

# apply the scaler to the dataframe subset
data_train_s.iloc[:,1:] = ss.fit_transform(data_train_s.iloc[:,1:])
data_test_s.iloc[:,1:] = ss.transform(data_test_s.iloc[:,1:])
```

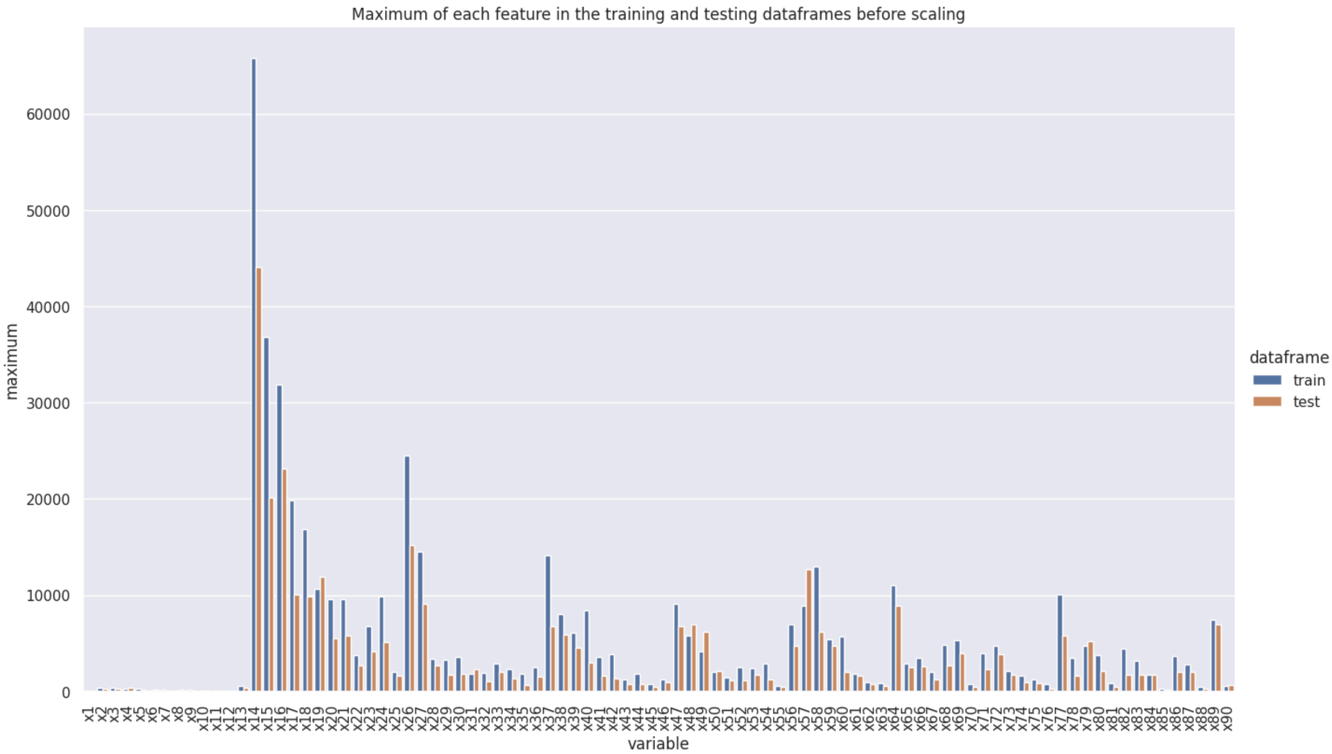
```
means = pd.DataFrame(list(zip(data_train.columns, data_train.mean(), data_test.mean())),
                   columns=['variable', 'train', 'test'])
means.drop(0, inplace=True)

means = pd.melt(means, id_vars="variable", var_name="dataframe", value_name="mean")

sns.catplot(x='variable', y='mean', hue='dataframe', data=means, kind='bar', height=8, aspect=1.7)
plt.xticks(rotation = 90)
plt.title('Mean of each feature in the training and testing dataframes before scaling');
```



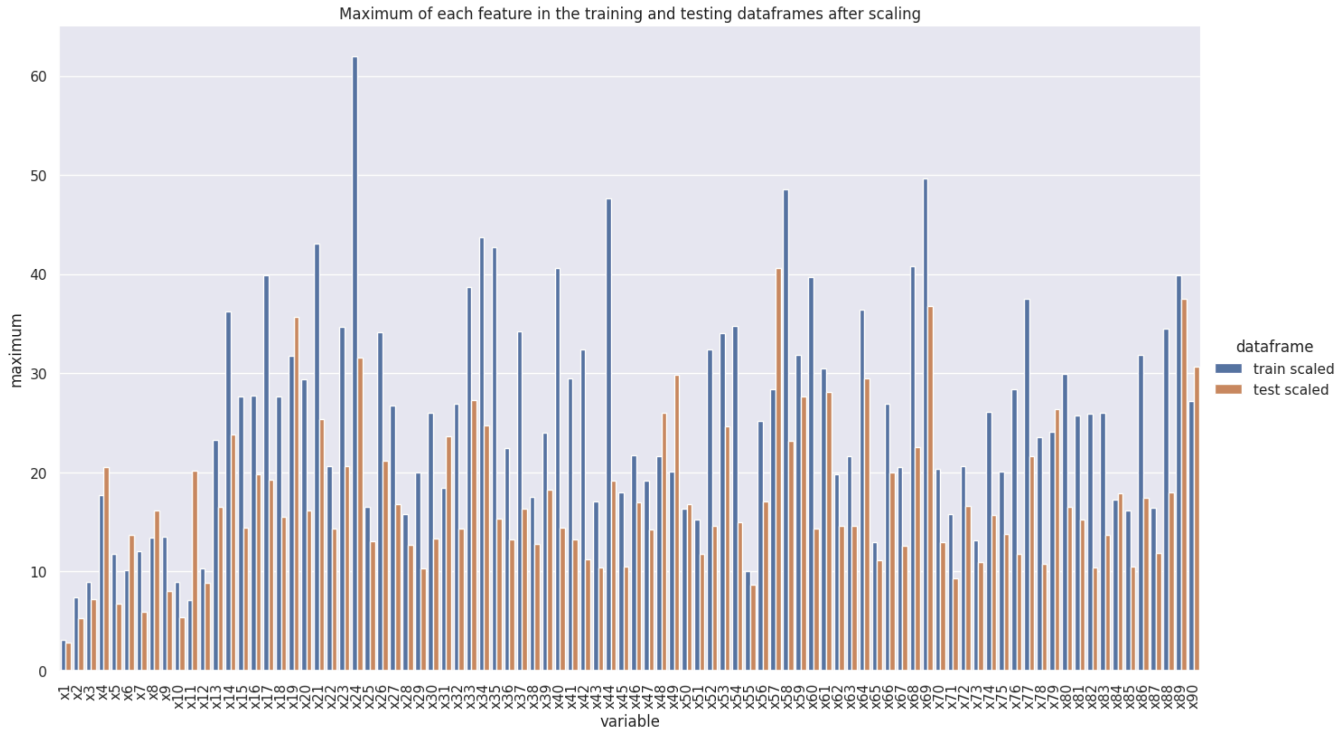

```
means = pd.DataFrame(list(zip(data_train.columns, data_train_s.mean(), data_test_s.mean())),
                 columns=['variable', 'train scaled', 'test scaled'])
means.drop(0, inplace=True)

means = pd.melt(means, id_vars="variable", var_name="dataframe", value_name="mean")

sns.catplot(x='variable', y='mean', hue='dataframe', data=means, kind='bar', height=8, aspect=1.7)
plt.xticks(rotation = 90)
plt.title('Mean of each feature in the training and testing dataframes after scaling');
```



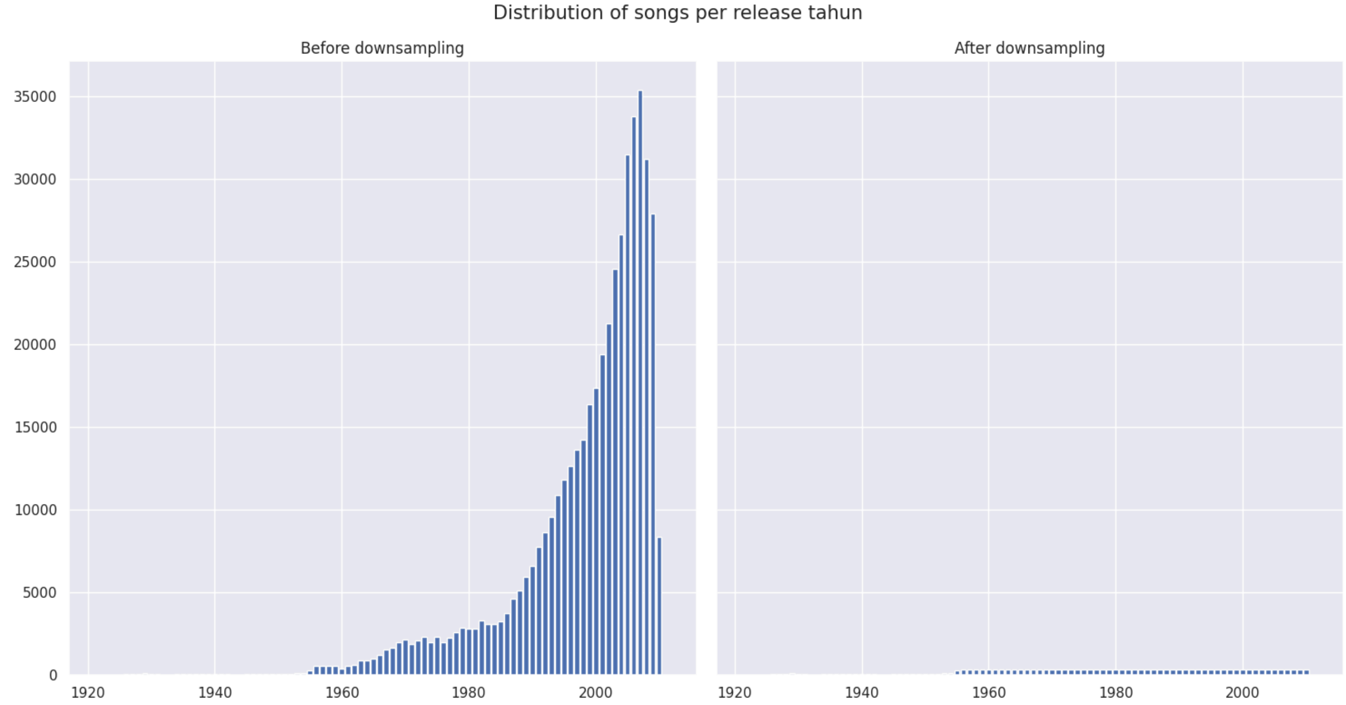

```
maxi = pd.DataFrame(list(zip(data_train.columns, data_train.max(), data_test.max())),
                   columns=['variable', 'train', 'test'])
maxi.drop(0, inplace=True)

maxi = pd.melt(maxi, id_vars="variable", var_name="dataframe", value_name="maximum")

sns.catplot(x='variable', y='maximum', hue='dataframe', data=maxi, kind='bar', height=8, aspect=1.7)
plt.xticks(rotation = 90)
plt.title('Maximum of each feature in the training and testing dataframes before scaling');
```




```
maxi = pd.DataFrame(list(zip(data_train.columns, data_train_s.max(), data_test_s.max())),
                   columns=['variable', 'train scaled', 'test scaled'])
maxi.drop(0, inplace=True)

maxi = pd.melt(maxi, id_vars="variable", var_name="dataframe", value_name="maximum")

sns.catplot(x='variable', y='maximum', hue='dataframe', data=maxi, kind='bar', height=8, aspect=1.7)
plt.xticks(rotation = 90)
plt.title('Maximum of each feature in the training and testing dataframes after scaling');
```




```
# Downsampling by 'tahun'
min_samples = 1000
tahuns = data_train_s.tahun.unique()
sampled_dfs = []  # List to store each sampled DataFrame

for tahun in tahuns:
    if data_train_s[data_train_s.tahun == tahun].shape[0] > min_samples:
        sampled_dfs.append(data_train_s[data_train_s.tahun == tahun].sample(min_samples))
    else:
        sampled_dfs.append(data_train_s[data_train_s.tahun == tahun])

# Concatenate all sampled DataFrames at once
data_train_sampled = pd.concat(sampled_dfs, ignore_index=True)



fig, ax = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(15,8))
fig.suptitle('Distribution of songs per release tahun', fontsize=15)

ax[0].bar(data_train_s.tahun.value_counts().index, data_train_s.tahun.value_counts())
ax[0].set_title('Before downsampling')

ax[1].bar(data_train_s.tahun.value_counts().index, data_train_sampled.tahun.value_counts())
ax[1].set_title('After downsampling')

plt.tight_layout()
```




```
del data
```

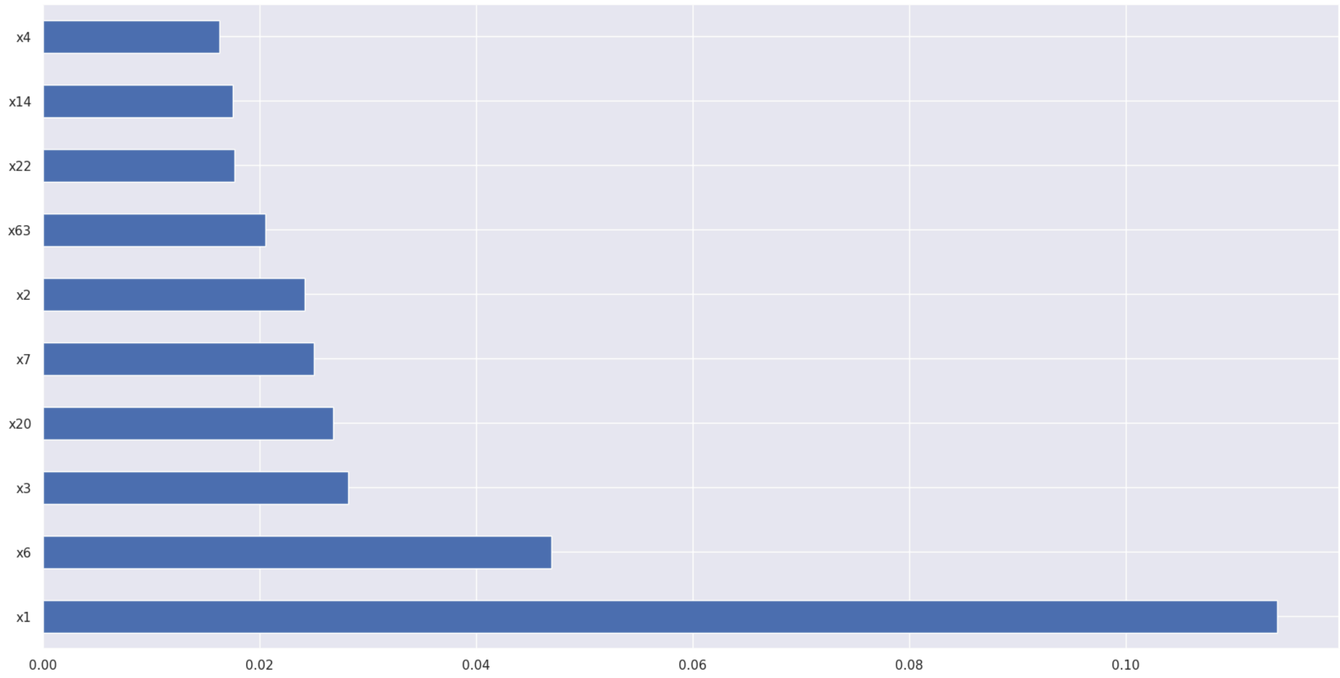
```
data_train_s.shape
data_train_sampled.shape
```

```
(17909, 91)
```

```
from sklearn.ensemble import ExtraTreesClassifier

# we separate the target from the features
X_train = data_train_s.iloc[:,1:]
y_train = data_train_s.iloc[:,0]

model = ExtraTreesClassifier(n_estimators=10, max_depth=10, warm_start=True)
for i in range(10, 51, 10):  # Increasing trees gradually
    model.n_estimators = i
    model.fit(X_train, y_train)
```

```
# graph of the 10 most important features
feat_importances = pd.Series(model.feature_importances_, index=X_train.columns)
feat_importances.nlargest(10).plot(kind='barh')
```

<Axes: >




```
# graph of the 20 most important features
feat_importances = pd.Series(model.feature_importances_, index=X_train.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

```
<Axes: >
```

```
names10=['tahun']
names10.extend(list(feat_importances.nlargest(10).index.sort_values()))
names10
```

```
['tahun', 'x1', 'x14', 'x2', 'x20', 'x22', 'x3', 'x4', 'x6', 'x63', 'x7']
```

```
names20=['tahun']
names20.extend(list(feat_importances.nlargest(20).index.sort_values()))
names20
```

```
['tahun',
 'x1',
 'x14'.
```