main

Local-Explanations-for-Cervical-Cancer / MLHC_cervical_cancer_classification.ipynb

Go to file t

cwayad Add files via upload 8eb0b33 · 8 months ago History

Source: https://www.kaggle.com/code/renadope/cervical-cancer-classification-99-4-recall

In [ ]:
```
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_r
emount=True).
```

In [ ]:
```
!pip install psutil -U kaleido
import plotly.io as pio
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (5.9.5)
Requirement already satisfied: kaleido in /usr/local/lib/python3.10/dist-packages (0.2.1)
```

In [ ]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go

from sklearn.impute import SimpleImputer
from sklearn.model_selection import StratifiedShuffleSplit
from typing import List
from sklearn.preprocessing import RobustScaler,StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_recall_fscore_support

from imblearn.over_sampling import SMOTE, ADASYN
from imblearn.over_sampling import RandomOverSampler

from plotly.offline import plot, iplot, init_notebook_mode
#init_notebook_mode(connected=True)

import warnings

warnings.filterwarnings('ignore')
```

# Overview

In [ ]:
```
risk_factor_df = pd.read_csv('/content/drive/My Drive/risk_factors_cervical_cancer.csv', delimiter=',', encoding=
risk_factor_df.head()
```

Out[ ]:

| | Age | Number of sexual partners | First sexual intercourse | Num of pregnancies | Smokes | Smokes (years) | Smokes (packs/year) | Hormonal Contraceptives | Hormonal Contraceptives (years) | IUD | ... | STDs: Time since first diagnosis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 18 | 4.0 | 15.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | ? |
| **1** | 15 | 1.0 | 14.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | ? |
| **2** | 34 | 1.0 | ? | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | ? |
| **3** | 52 | 5.0 | 16.0 | 4.0 | 1.0 | 37.0 | 37.0 | 1.0 | 3.0 | 0.0 | ... | ? |
| **4** | 46 | 3.0 | 21.0 | 4.0 | 0.0 | 0.0 | 0.0 | 1.0 | 15.0 | 0.0 | ... | ? |

5 rows × 36 columns

In [ ]:
```
risk_factor_df[risk_factor_df['Dx:HPV']==1  ]
```

In [ ]:
```
risk_factor_df['Dx:HPV'].value_counts()
```

Out[ ]:
```
0    840
1     18
Name: Dx:HPV, dtype: int64
```

In [ ]:
```
risk_factor_df.isna().sum()
```

In [ ]:
```
risk_factor_df.info()
```

# Preprocessing

In [ ]:
```
def print_unique_values_df(df: pd.DataFrame):
```

```
def print_unique_values_df(df: pd.DataFrame):
    for col in list(df):
        print("Unique Values for "'{}'":{}".format(str(col), risk_factor_df[col].unique()))
        print("dtype for {} is :{}".format(str(col), risk_factor_df[col].dtypes))
        print("-" * 150)


def print_unique_values_for_col(df: pd.DataFrame, col_names: List[str] = None):
    for col in col_names:
        print("Unique Values for "'{}'":{}".format(str(col), risk_factor_df[col].unique()))
```

In [ ]:

```
print_unique_values_df(risk_factor_df)
```

```
Unique Values for Age:[18 15 34 52 46 42 51 26 45 44 27 43 40 41 39 37 38 36 35 33 31 32 30 23
 28 29 20 25 21 24 22 48 19 17 16 14 59 79 84 47 13 70 50 49]
dtype for Age is :int64
------------------------------------------------------------------------------------------------------------------------
Unique Values for Number of sexual partners:['4.0' '1.0' '5.0' '3.0' '2.0' '6.0' '?' '7.0' '15.0' '8.0' '10.0' '28.
0'
 '9.0']
dtype for Number of sexual partners is :object
------------------------------------------------------------------------------------------------------------------------
Unique Values for First sexual intercourse:['15.0' '14.0' '?' '16.0' '21.0' '23.0' '17.0' '26.0' '20.0' '25.0' '18.
0'
 '27.0' '19.0' '24.0' '32.0' '13.0' '29.0' '11.0' '12.0' '22.0' '28.0'
 '10.0']
dtype for First sexual intercourse is :object
------------------------------------------------------------------------------------------------------------------------
Unique Values for Num of pregnancies:['1.0' '4.0' '2.0' '6.0' '3.0' '5.0' '?' '8.0' '7.0' '0.0' '11.0' '10.0']
dtype for Num of pregnancies is :object
------------------------------------------------------------------------------------------------------------------------
Unique Values for Smokes:['0.0' '1.0' '?']
dtype for Smokes is :object
------------------------------------------------------------------------------------------------------------------------
Unique Values for Smokes (years):['0.0' '37.0' '34.0' '1.266972909' '3.0' '12.0' '?' '18.0' '7.0' '19.0'
 '21.0' '15.0' '13.0' '16.0' '8.0' '4.0' '10.0' '22.0' '14.0' '0.5' '11.0'
 '9.0' '2.0' '5.0' '6.0' '1.0' '32.0' '24.0' '28.0' '20.0' '0.16']
dtype for Smokes (years) is :object
------------------------------------------------------------------------------------------------------------------------
Unique Values for Smokes (packs/year):['0.0' '37.0' '3.4' '2.8' '0.04' '0.5132021277' '2.4' '6.0' '?' '9.0'
 '1.6' '19.0' '21.0' '0.32' '2.6' '0.8' '15.0' '2.0' '5.7' '1.0' '3.3'
 '3.5' '12.0' '0.025' '2.75' '0.2' '1.4' '5.0' '2.1' '0.7' '1.2' '7.5'
 '1.25' '3.0' '0.75' '0.1' '8.0' '2.25' '0.003' '7.0' '0.45' '0.15' '0.05'
```

```
 '0.25' '4.8' '4.5' '0.4' '0.37' '2.2' '0.16' '0.9' '22.0' '1.35' '0.5'
 '2.5' '4.0' '1.3' '1.65' '2.7' '0.001' '7.6' '5.5' '0.3']
dtype for Smokes (packs/year) is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for Hormonal Contraceptives:['0.0' '1.0' '?']
dtype for Hormonal Contraceptives is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for Hormonal Contraceptives (years):['0.0' '3.0' '15.0' '2.0' '8.0' '10.0' '5.0' '0.25' '7.0' '22.0'
'19.0'
 '0.5' '1.0' '0.58' '9.0' '13.0' '11.0' '4.0' '12.0' '16.0' '0.33' '?'
 '0.16' '14.0' '0.08' '2.282200521' '0.66' '6.0' '1.5' '0.42' '0.67'
 '0.75' '2.5' '4.5' '6.5' '0.17' '20.0' '3.5' '0.41' '30.0' '17.0']
dtype for Hormonal Contraceptives (years) is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for IUD:['0.0' '1.0' '?']
dtype for IUD is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for IUD (years):['0.0' '7.0' '?' '5.0' '8.0' '6.0' '1.0' '0.58' '2.0' '19.0' '0.5' '17.0'
 '0.08' '0.25' '10.0' '11.0' '3.0' '15.0' '12.0' '9.0' '1.5' '0.91' '4.0'
 '0.33' '0.41' '0.16' '0.17']
dtype for IUD (years) is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for STDs:['0.0' '1.0' '?']
dtype for STDs is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for STDs (number):['0.0' '2.0' '1.0' '?' '3.0' '4.0']
dtype for STDs (number) is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for STDs:condylomatosis:['0.0' '1.0' '?']
dtype for STDs:condylomatosis is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for STDs:cervical condylomatosis:['0.0' '?']
dtype for STDs:cervical condylomatosis is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for STDs:vaginal condylomatosis:['0.0' '?' '1.0']
dtype for STDs:vaginal condylomatosis is :object
--------------------------------------------------------------------------------------------------------------------
--------------------------------
Unique Values for STDs:vulvo-perineal condylomatosis:['0.0' '1.0' '?']
dtype for STDs:vulvo-perineal condylomatosis is :object
--------------------------------------------------------------------------------------------------------------------
```

```
------------------------------------
Unique Values for STDs:syphilis:['0.0' '1.0' '?']
dtype for STDs:syphilis is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:pelvic inflammatory disease:['0.0' '?' '1.0']
dtype for STDs:pelvic inflammatory disease is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:genital herpes:['0.0' '?' '1.0']
dtype for STDs:genital herpes is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:molluscum contagiosum:['0.0' '?' '1.0']
dtype for STDs:molluscum contagiosum is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:AIDS:['0.0' '?']
dtype for STDs:AIDS is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:HIV:['0.0' '1.0' '?']
dtype for STDs:HIV is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:Hepatitis B:['0.0' '?' '1.0']
dtype for STDs:Hepatitis B is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs:HPV:['0.0' '?' '1.0']
dtype for STDs:HPV is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs: Number of diagnosis:[0 1 3 2]
dtype for STDs: Number of diagnosis is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs: Time since first diagnosis:['?' '21.0' '2.0' '15.0' '19.0' '3.0' '12.0' '1.0' '11.0' '9.0'
'7.0'
 '8.0' '16.0' '6.0' '5.0' '10.0' '4.0' '22.0' '18.0']
dtype for STDs: Time since first diagnosis is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for STDs: Time since last diagnosis:['?' '21.0' '2.0' '15.0' '19.0' '3.0' '12.0' '1.0' '11.0' '9.0'
'7.0'
 '8.0' '16.0' '6.0' '5.0' '10.0' '4.0' '22.0' '18.0']
dtype for STDs: Time since last diagnosis is :object
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Dx:Cancer:[0 1]
dtype for Dx:Cancer is :int64
```

```
dtype for Dx:Cancer is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Dx:CIN:[0 1]
dtype for Dx:CIN is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Dx:HPV:[0 1]
dtype for Dx:HPV is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Dx:[0 1]
dtype for Dx is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Hinselmann:[0 1]
dtype for Hinselmann is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Schiller:[0 1]
dtype for Schiller is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Citology:[0 1]
dtype for Citology is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
Unique Values for Biopsy:[0 1]
dtype for Biopsy is :int64
------------------------------------------------------------------------------------------------------------------------
------------------------------------
```

In [ ]:

```
#these columns are not of type object, but are of type numeric
cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Num of pregnancies', 'Smokes',
                   'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contraceptives',
                   'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STDs', 'STDs (number)',
                   'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
                   'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic inflammatory disease',
                   'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AIDS', 'STDs:HIV', 'STDs:Hepatitis B
                   'STDs:HPV', 'STDs: Time since first diagnosis',
                   'STDs: Time since last diagnosis']
# for i in range(0,len(cols_to_convert)):
#     print("{}={}".format(i,cols_to_convert[i]))
risk_factor_df[cols_to_convert] = risk_factor_df[cols_to_convert].apply(pd.to_numeric, errors="coerce")
risk_factor_df[cols_to_convert].fillna(np.nan, inplace=True)
imp = SimpleImputer(strategy="median")
X = imp.fit_transform(risk_factor_df)
risk_factor_df = pd.DataFrame(X, columns=list(risk_factor_df.columns))
```

In [ ]:
```
def age_cat(age):
    if age < 12:
        return "Child"
    elif age < 20:
        return "Teen"
    elif age < 30:
        return "20's"
    elif age < 40:
        return "30's"
    elif age < 50:
        return "40's"
    elif age < 60:
        return "50's"
    elif age < 70:
        return "60's"
    else:
        return "70+"


risk_factor_df["Age"] = risk_factor_df["Age"].astype(int)
risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)
```

In [ ]:
```
std_cols = {'STDs:condylomatosis',
            'STDs:cervical condylomatosis',
            'STDs:vaginal condylomatosis',
            'STDs:vulvo-perineal condylomatosis',
            'STDs:syphilis',
            'STDs:pelvic inflammatory disease',
            'STDs:genital herpes',
            'STDs:molluscum contagiosum',
            'STDs:AIDS',
            'STDs:HIV',
            'STDs:Hepatitis B',
            'STDs:HPV'}

risk_factor_df["total_std"] = risk_factor_df[list(std_cols)].sum(axis=1)
std_agg = risk_factor_df.groupby("age_cat", as_index=False)[list(std_cols)].sum()
```

In [ ]:
```
test_cols = ["Hinselmann", "Schiller", "Citology", "Biopsy"]
risk_factor_df["total_tests"] = risk_factor_df[test_cols].sum(axis = 1)
```

In [ ]:
```
to_int_and_beyond = {"total_tests",
                     "total_std",
                     "Smokes",
                     "Biopsy",
                     "Dx:Cancer",
```

```
                      "Num of pregnancies",
                      "Number of sexual partners",
                      "First sexual intercourse",
                      "Hormonal Contraceptives",
                      "IUD",
                      "STDs",
                      "STDs (number)",
                      "STDs: Number of diagnosis",
                      "Dx:CIN",
                      "Dx:HPV",
                      "Dx",
                      "Hinselmann",
                      "Schiller",
                      "Biopsy",
                      "Citology"}

to_int_and_beyond = to_int_and_beyond.union(std_cols)

for col in to_int_and_beyond:
    risk_factor_df[col] = risk_factor_df[col].astype(int)
```

In [ ]:

```
risk_factor_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 858 entries, 0 to 857
Data columns (total 39 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Age                                 858 non-null    int64
 1   Number of sexual partners           858 non-null    int64
 2   First sexual intercourse            858 non-null    int64
 3   Num of pregnancies                  858 non-null    int64
 4   Smokes                              858 non-null    int64
 5   Smokes (years)                      858 non-null    float64
 6   Smokes (packs/year)                 858 non-null    float64
 7   Hormonal Contraceptives             858 non-null    int64
 8   Hormonal Contraceptives (years)     858 non-null    float64
 9   IUD                                 858 non-null    int64
 10  IUD (years)                         858 non-null    float64
 11  STDs                                858 non-null    int64
 12  STDs (number)                       858 non-null    int64
 13  STDs:condylomatosis                 858 non-null    int64
 14  STDs:cervical condylomatosis        858 non-null    int64
 15  STDs:vaginal condylomatosis         858 non-null    int64
 16  STDs:vulvo-perineal condylomatosis  858 non-null    int64
 17  STDs:syphilis                       858 non-null    int64
 18  STDs:pelvic inflammatory disease    858 non-null    int64
 19  STDs:genital herpes                 858 non-null    int64
 20  STDs:molluscum contagiosum          858 non-null    int64
```

```
 21  STDs:AIDS                           858 non-null    int64
 22  STDs:HIV                            858 non-null    int64
 23  STDs:Hepatitis B                    858 non-null    int64
 24  STDs:HPV                            858 non-null    int64
 25  STDs: Number of diagnosis           858 non-null    int64
 26  STDs: Time since first diagnosis    858 non-null    float64
 27  STDs: Time since last diagnosis     858 non-null    float64
 28  Dx:Cancer                           858 non-null    int64
 29  Dx:CIN                              858 non-null    int64
 30  Dx:HPV                              858 non-null    int64
 31  Dx                                  858 non-null    int64
 32  Hinselmann                          858 non-null    int64
 33  Schiller                            858 non-null    int64
 34  Citology                            858 non-null    int64
 35  Biopsy                              858 non-null    int64
 36  age_cat                             858 non-null    object
 37  total_std                           858 non-null    int64
 38  total_tests                         858 non-null    int64
dtypes: float64(6), int64(32), object(1)
memory usage: 261.5+ KB
```

In [ ]:

```
# corr_matrix = risk_factor_df.corr()
# corr_matrix.fillna(0,inplace=True)
# corr_graph = px.imshow(corr_matrix, aspect="auto")
# corr_graph.show()
```

In [ ]:

```
n = 7
target = label = "Dx:Cancer"
corr = risk_factor_df.select_dtypes(include=np.number).corr()

x = corr.nlargest(n,target).index
corr_df =  risk_factor_df[list(x)]
corr = corr_df.corr()
fig = px.imshow(corr,color_continuous_scale = "PuBu")
fig.update_layout(title="Top "+str(n)+" Features Correlated With "+str(target).capitalize())
fig.show()
```

In [ ]:

```
def stats(x):
    temp1=(df[[x,label]].value_counts(normalize=True).round(decimals=3)*100).reset_index().rename(columns={0:'Ove
    Coloumn_To_Aggregate=[x,label]
    df6=pd.merge(df.groupby(Coloumn_To_Aggregate).size().reset_index(name='ind_siz'),
                 df.groupby(Coloumn_To_Aggregate[:-1]).size().reset_index(name='Total'), on =Coloumn_To_Aggregate
    df6['Category_Percent']=round((df6['ind_siz']/df6['Total'])*100 ,2)
    temp2=df6[[x,label,'Category_Percent']]
    temp3=temp1.merge(temp2,on=[x,label])
    return temp3.pivot(columns=x,index=label)
```

In [ ]:
```
df=risk_factor_df
label='age_cat'
```

In [ ]:
```
stats('Dx:Cancer')
```

Out[ ]:

| | Overall_Percent | | Category_Percent | |
|---|---|---|---|---|
| **Dx:Cancer** | **0** | **1** | **0** | **1** |
| **age_cat** | | | | |
| **20's** | 45.3 | 0.6 | 46.31 | 27.78 |
| **30's** | 24.7 | 0.9 | 25.24 | 44.44 |
| **40's** | 6.2 | 0.3 | 6.31 | 16.67 |
| **50's** | 0.5 | 0.1 | 0.48 | 5.56 |
| **70+** | 0.5 | NaN | 0.48 | NaN |
| **Teen** | 20.7 | 0.1 | 21.19 | 5.56 |

# Visualization

In [ ]:
```
age_dist = px.histogram(risk_factor_df, x="Age", marginal="box", color_discrete_sequence=["palevioletred"])
age_dist.update_layout(title="Age distribution")
age_dist.show()
```

## Pregnancy Distribution by Age

In [ ]:
```
tempp=risk_factor_df.sort_values(by="Age",ascending=True)
```

In [ ]:
```
risk_factor_df.age_cat
```

Out[ ]:
```
0      Teen
1      Teen
2      30's
```

```
3      50's
4      40's
       ...
853    30's
854    30's
855    20's
856    30's
857    20's
Name: age_cat, Length: 858, dtype: object
```

In [ ]:

```
age_preg_bar = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="age_cat", y="Num of pregnancies",
                      color_discrete_sequence=["darkblue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forties", "Fifties","Sixties",
                                       "Seventy and over"])
age_preg_bar.update_xaxes(title="Age Category")
age_preg_bar.update_yaxes(title="Number of Pregnancies")
age_preg_bar.update_layout(title="Distribution of number of pregnancies per age group")
age_preg_bar.show()
```

## Risk factors for cervical cancer include:

### From the mayo clinic:

- Many sexual partners. The greater your number of sexual partners — and the greater your partner's number of sexual partners — the greater your chance of acquiring HPV.
- Early sexual activity. Having sex at an early age increases your risk of HPV.
- Other sexually transmitted infections (STIs). Having other STIs — such as chlamydia, gonorrhea, syphilis and HIV/AIDS — increases your risk of HPV.
- A weakened immune system. You may be more likely to develop cervical cancer if your immune system is weakened by another health condition and you have HPV.
- Smoking. Smoking is associated with squamous cell cervical cancer.
- Exposure to miscarriage prevention drug. If your mother took a drug called diethylstilbestrol (DES) while pregnant in the 1950s, you may have an increased risk of a certain type of cervical cancer called clear cell adenocarcinoma.

In [ ]:

```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="age_cat", y="Number of sexua
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forties", "Fifties",
                                       "Seventy and over"])
age_num_sex_partners.update_xaxes(title="Age Category")
age_num_sex_partners.update_yaxes(title="Number of Sexual Partners")
age_num_sex_partners.update_layout(title="Distribution of number of sexual partners per age group")
age_num_sex_partners.show()
```

From the scatterplot, it is seen that the number of sexual partners have remained consistent throught out differing age ranges.

In [ ]:
```
age_num_sex_partners = px.scatter(risk_factor_df, x="Age",
                                  y="Number of sexual partners",
                                  trendline="ols",
                                  opacity=0.4,
                                  color="Num of pregnancies",
                                  color_continuous_scale="rdbu",)
age_num_sex_partners.update_layout(title="Age vs Number of Sexual Partners")
age_num_sex_partners.show()
```

From the heatmap, we can see that there a correlation coefficent very close to 0, this indicates that, from the data, the number of sexual partners does not have any linear relationship with any of the respective diagnoses. However, we also visually knew that the number of sexual partners remained fairly consistent across age ranges and therefore there are more likely causes of HPV and Cervical Cancer than number of sexual partners with respect to the data.

In [ ]:
```
diagnoses_num_partner_compare_cols = [label,
                                      'Dx:HPV',
                                      "Number of sexual partners",]
corr_matrix = risk_factor_df[diagnoses_num_partner_compare_cols].corr()
print(corr_matrix)
diagnoses_num_partner_heatmap = px.imshow(corr_matrix,
                                aspect="auto",
                                color_continuous_scale="gnbu",
                                text_auto=True)
diagnoses_num_partner_heatmap.show()
```

```
                          Dx:HPV  Number of sexual partners
Dx:HPV                  1.000000                   0.028646
Number of sexual partners  0.028646                   1.000000
```

## Correlation of diagnoses

Comparing the diagnoses, to see if there is any correlation among them. It's seen that a HPV diagnosis and Cervical Cancer Diagnosis have a correlation of approximately +0.89, this is indicative of a strong positive correlation. In some regard, it can be interprted as a diagnosis of HPV is likely to lead to a diagnosis of Cervical Cancer.

In [ ]:
```
diagnoses_cols = [label,
                  'Dx:CIN',
```

```
                    'Dx:HPV']
diagnoses_corr_matrix = risk_factor_df[diagnoses_cols].corr()
# print(diagnoses_corr_matrix)
diagnoses_heatmap = px.imshow(diagnoses_corr_matrix, aspect="auto", color_continuous_scale="tealgrn", text_auto=Tr
diagnoses_heatmap.show()
```

# STD's Definitions

## Syphilis

Syphillis is a bacterial infection usually spread by sexual contact. The disease starts as a painless sore — typically on the genitals, rectum or mouth. Syphilis spreads from person to person via skin or mucous membrane contact with these sores.After the initial infection, the syphilis bacteria can remain inactive in the body for decades before becoming active again. Early syphilis can be cured, sometimes with a single shot (injection) of penicillin.Without treatment, syphilis can severely damage the heart, brain or other organs, and can be life-threatening. Syphilis can also be passed from mothers to unborn children. Source

## HIV/AIDS

HIV (human immunodeficiency virus) is a virus that attacks cells that help the body fight infection, making a person more vulnerable to other infections and diseases. It is spread by contact with certain bodily fluids of a person with HIV, most commonly during unprotected sex (sex without a condom or HIV medicine to prevent or treat HIV), or through sharing injection drug equipment. *If left untreated, HIV can lead to the disease AIDS (acquired immunodeficiency syndrome* Source

## Cervical / Vaginal Condylomatosis

Condyloma or genital warts affect the tissues of the genital area due to infections induced by Human papillomavirus. Source

## Vulvo-perineal condylomatosis

It is a benign epithelial proliferative viral lesion that can affect any area of the vulvo-perineal district supported by human papilloma virus (HPV). Source.)

## Genital Herpes

Genital herpes is a common sexually transmitted infection caused by the herpes simplex virus (HSV). Sexual contact is the primary way that the virus spreads. After the initial infection, the virus lies dormant in your body and can reactivate several times a year. Genital herpes can cause pain, itching and sores in your genital area. But you may have no signs or symptoms of genital herpes. If infected, you can be contagious even if you have no visible sores.There's no cure for genital herpes, but medications can ease symptoms and reduce

the risk of infecting others. Condoms also can help prevent the spread of a genital herpes infection. Source

## HPV

HPV infection is a viral infection that commonly causes skin or mucous membrane growths (warts). There are more than 100 varieties of human papillomavirus (HPV). Some types of HPV infection cause warts, and some can cause different types of cancer. Most HPV infections don't lead to cancer. But some types of genital HPV can cause cancer of the lower part of the uterus that connects to the vagina (cervix). Other types of cancers, including cancers of the anus, penis, vagina, vulva and back of the throat (oropharyngeal), have been linked to HPV infection. These infections are often transmitted sexually or through other skin-to-skin contact. Vaccines can help protect against the strains of HPV most likely to cause genital warts or cervical cancer. Source

## Molluscum Contagisum

Molluscum contagiosum is an infection caused by a poxvirus (molluscum contagiosum virus). The result of the infection is usually a benign, mild skin disease characterized by lesions (growths) that may appear anywhere on the body. Within 6-12 months, Molluscum contagiosum typically resolves without scarring but may take as long as 4 years. The lesions, known as Mollusca, are small, raised, and usually white, pink, or flesh-colored with a dimple or pit in the center. They often have a pearly appearance. They're usually smooth and firm. In most people, the lesions range from about the size of a pinhead to as large as a pencil eraser (2 to 5 millimeters in diameter). They may become itchy, sore, red, and/or swollen. Mollusca may occur anywhere on the body including the face, neck, arms, legs, abdomen, and genital area, alone or in groups. The lesions are rarely found on the palms of the hands or the soles of the feet. Source

The virus that causes molluscum spreads from direct person-to-person physical contact and through contaminated fomites. Fomites are inanimate objects that can become contaminated with virus; in the instance of molluscum contagiosum this can include linens such as clothing and towels, bathing sponges, pool equipment, and toys Source

Someone with molluscum can spread it to other parts of their body by touching or scratching a lesion and then touching their body somewhere else. This is called autoinoculation. Shaving and electrolysis can also spread mollusca to other parts of the body. *Molluscum can spread from one person to another by sexual contact. Many, but not all, cases of molluscum in adults are caused by sexual contact.* Source

## Hepatitis B

Hepatitis B is a vaccine-preventable liver infection caused by the hepatitis B virus (HBV). Hepatitis B is spread when blood, semen, or other body fluids from a person infected with the virus enters the body of someone who is not infected. This can happen through sexual contact; sharing needles, syringes, or other drug-injection equipment; or from mother to baby at birth. **Source**

In [ ]:

```
fig = px.histogram(std_agg, x="age_cat", y=list(std_cols), barmode="group", histfunc="sum")
fig.update_layout(title="Sum of STD occurence across age categories")
fig.update_xaxes(title="Age Category")
```

```
fig.update_yaxes(title="Sum")
fig.show()
```

In [ ]:
```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="age_cat", y="total_std",
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forties", "Fifties",
                                       "Seventy and over"])
age_num_sex_partners.update_xaxes(title="Age Category")
age_num_sex_partners.update_yaxes(title="Number of Sexual Partners")
age_num_sex_partners.update_layout(title="Distribution of number of sexual partners per age group")
age_num_sex_partners.show()
```

We see that the most amount of STD's garnered by any patient, is a total of 4. As from before, we also see that the majority of patients do not have any STD's and aren't diagnosed with cancer and/or HPV. However, there is a small amount of patients who have no STD and have Cervical Cancer and/or HPV. *It should be noted that HPV infections can be sexually transmitted or non-sexually acquired.*

In [ ]:
```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_std", label], ascending=True),
                   x="age_cat",
                   facet_col="total_std",
                   facet_row=label,
                   color_discrete_sequence=["rebeccapurple"],
                   opacity=0.7)
fig.update_layout(title="Count of women across age groups who have had one or more std")

fig.show()
```

In [ ]:
```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_std","Dx:HPV"], ascending=True),
                   x="age_cat",
                   facet_col="total_std",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["dodgerblue"],
                   opacity=0.7)
fig.update_layout(title="Count of women across age groups who have had one or more std")

fig.show()
```

## Tests used

Here we observe the number of tests done by patients to determine if they have Cerivcal Cancer / HPV.

The tests used were:

## Hinselmann

A colposcopy is a type of cervical cancer test. It lets your doctor or nurse get a close-up look at your cervix — the opening to your uterus. It's used to find abnormal cells in your cervix. Source

## Citology

Cytology is the exam of a single cell type, as often found in fluid specimens. It's mainly used to diagnose or screen for cancer. It's also used to screen for fetal abnormalities, for pap smears, to diagnose infectious organisms, and in other screening and diagnostic areas. Source

## Biopsy

A cervical biopsy is a procedure to remove tissue from the cervix to test for abnormal or precancerous conditions, or cervical cancer. Source

## Schiller

A test in which iodine is applied to the cervix. The iodine colors healthy cells brown; abnormal cells remain unstained, usually appearing white or yellow. Source

In [ ]:
```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by="total_tests", ascending=True),
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row=label,
                   color_discrete_sequence=["blueviolet"],
                   opacity=0.8)
fig.update_layout(title="Count of women across age groups who have had one or more test")

fig.show()
```

In [ ]:
```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by=["total_tests","Dx:HPV"], ascending=True),
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["coral"],
                   opacity=0.8)
fig.update_layout(title="Count of women across age groups who have had one or more test")
```

```
fig.show()
```

We see from the ECDF plot, that:

- There is roughly a 95% probability that patients have smoked for less than 10 years
- There is roughly a 99% probability that patients have used IUD's for less than 10 years
- There is roughly a 99% probabilty that patients have used Hormonal Contraceptives for less than 10 years

In [ ]:
```
fig = px.ecdf(risk_factor_df, x=["Smokes (years)",
                                 "Hormonal Contraceptives (years)",
                                 "IUD (years)"],
              color_discrete_sequence=["crimson", "deepskyblue", "chartreuse"])
fig.update_xaxes(title="Years")
fig.update_layout(title="ECDF Plot")
fig.show()
```

## Proportions of women who have Cervical Cancer / HPV

This represents the proportion of women by age category who were diagnosed with Cervical Cancer/ HPV. It is seen that women in their 30's have the most prevalance of Cervical Cancer and HPV, followed by women in their 20's.

It is also seen that of all the samples taken, approximately 26% are of women in their 30's. With respect to the women who have cervical cancer, approximately 44% of cases are women in their 30's, also, out of the women who have HPV, approximately 39% of women are in their 30's. This is contrasted with 45% of all samples being women in their 20's and only 28% of the women have cancer are in their 20's, HPV is more comparable at 33%.

In [ ]:
```
age_category_range = {
    "Age<12": "Child",
    "Age>=12 & Age<20": "Teen",
    "Age>=20 & Age<30": "20's",
    "Age>=30 & Age<40": "30's",
    "Age>=40 & Age<50": "40's",
    "Age>=50 & Age<60": "50's",
    "Age>=60 & Age<70": "60's",
    "Age>=70": "70+"}
age_prop_dict = {}
col = "Age"  # Just to get the count
for age_range, category in age_category_range.items():
    age_prop_dict[category] = risk_factor_df.query(age_range)[col].count() / len(risk_factor_df)
```

```
proportion_samples_df = pd.DataFrame.from_dict(age_prop_dict, orient="index",
                                               columns=[ "Sample Proportion"])
proportion_samples_df = proportion_samples_df.reset_index()
proportion_samples_df.columns = proportion_samples_df.columns.str.replace("index","Category")
fig = px.pie(proportion_samples_df,
             values='Sample Proportion',
             names="Category",
             title='Age Category proportion of women sampled',color_discrete_sequence=px.colors.sequential.RdBu)
fig.show()
proportion_samples_df
```

Out[ ]:

| | Category | Sample Proportion |
|---|---|---|
| **0** | Child | 0.000000 |
| **1** | Teen | 0.208625 |
| **2** | 20's | 0.459207 |
| **3** | 30's | 0.256410 |
| **4** | 40's | 0.065268 |
| **5** | 50's | 0.005828 |
| **6** | 60's | 0.000000 |
| **7** | 70+ | 0.004662 |

In [ ]:

```
fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'domain'}, {'type': 'domain'}]],
                    subplot_titles=["Cancer", "HPV"])
fig.add_trace(go.Pie(labels=risk_factor_df["age_cat"],
                     values=risk_factor_df[label],
                     name="Cancer", marker_colors=px.colors.sequential.RdBu),
              1, 1)
fig.add_trace(go.Pie(labels=risk_factor_df["age_cat"],
                     values=risk_factor_df["Dx:HPV"],
                     name="HPV", marker_colors=px.colors.sequential.RdBu),
              1, 2)

fig.update_traces(hole=.0, hoverinfo="label+percent+name")

fig.update_layout(
    title_text="Proportion of women across age categories with a diagnosis of Cancer, HPV",
)
fig.show()
```

# Contraceptive Overview

## IUD

IUD stands for Intrauterine Device (basically: a device inside your uterus). It's a small piece of flexible plastic shaped like a T. Sometimes it’s called an IUC — intrauterine contraception. Can cost up to $1,300.00 USD

IUDs are divided into 2 types:

- Hormonal IUDs
- Copper IUDs

Both copper IUDs and hormonal IUDs prevent pregnancy by changing the way sperm cells move so they can't get to an egg. If sperm can’t make it to an egg, pregnancy can’t happen. Source

## Hormonal Contraceptive

- The birth control pill works by stopping sperm from joining with an egg. When sperm joins with an egg it’s called fertilization.
- The hormones in the pill safely stop ovulation. No ovulation means there’s no egg for sperm to fertilize, so pregnancy can’t happen.
- The pill’s hormones also thicken the mucus on the cervix. This thicker cervical mucus blocks sperm so it can’t swim to an egg — kind of like a sticky security guard.
- Can cost up to $50.00 USD. Source

# Hormonal Contraceptives and Cervical Cancer

Women who have used oral contraceptives for 5 or more years have a higher risk of cervical cancer than women who have never used oral contraceptives. The longer a woman uses oral contraceptives, the greater the increase in her risk of cervical cancer. One study found a 10% increased risk for less than 5 years of use, a 60% increased risk with 5–9 years of use, and a doubling of the risk with 10 or more years of use. However, the risk of cervical cancer has been found to decline over time after women stop using oral contraceptives. Source

The usage of hormonal contraceptives is significantly higher than the usage of IUD's, this can most likely be attributed to it's low cost and easy accessibility

In [ ]:

In [ ]:
```
df_hormonal_compariosn = risk_factor_df.groupby(["age_cat"], as_index=False)[["IUD", "Hormonal Contraceptives"]].s
fig = px.histogram(df_hormonal_compariosn, x="age_cat", y=["IUD", "Hormonal Contraceptives"], barmode="group"
                   , color_discrete_sequence=["darkcyan", "mediumorchid"])

fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Count")
fig.update_layout(title="Age Ranges of women who use Contraceptives")

fig.show()
```

In [ ]:
```
df_hormonal_contraceptives = risk_factor_df[
    (risk_factor_df["Hormonal Contraceptives"] == 1) & (risk_factor_df["IUD"] == 0)]
df_hormonal_contraceptives = df_hormonal_contraceptives.sort_values(by=["Smokes", label])
fig = px.histogram(df_hormonal_contraceptives, x="age_cat", color="Smokes", barmode="group", facet_col=label,
                   color_discrete_sequence=["darkcyan", "crimson"])
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Count")
fig.update_layout(title="Age Ranges of women who use Hormonal Contraceptives")
# fig.for_each_annotation(lambda a: a.update(text=a.text.split(":")[-1]))
fig.show()
```

In [ ]:
```
df_IUD_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contraceptives"] == 0) & (risk_factor_df["IUD"] =
df_IUD_contraceptives = df_IUD_contraceptives.sort_values(by=["Smokes", label], ascending=True)
fig = px.histogram(df_IUD_contraceptives, x="age_cat", color="Smokes", barmode="group", facet_col=label,
                   color_discrete_sequence=["darkcyan", "crimson"])
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Sum of IUD Usage across age category")
fig.update_layout(title="Age Ranges of women who use IUD's")
fig.show()
```

In [ ]:
```
df_both_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contraceptives"] == 1) & (risk_factor_df["IUD"]
df_both_contraceptives = df_both_contraceptives.sort_values(by="Smokes")
fig = px.histogram(df_both_contraceptives, x="age_cat", color="Smokes", barmode="group", facet_col=label,
                   color_discrete_sequence=["darkcyan", "crimson"])
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Count")
fig.update_layout(title="Age Ranges of women who use BOTH Hormonal Contracepties and IUD's")
fig.show()
```

In [ ]:
```
y_train
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-41-f56205002e60> in <cell line: 1>()
----> 1 y_train

NameError: name 'y_train' is not defined
```

# Imbalanced Class

The "Dx:Cancer" class is an imbalanced class with just 18 classified as cancer and 840 as not cancer. This roughly translates to 2.1% classified as cancer and 97.9 % classified as not cancer.

In [ ]:
```
test=risk_factor_df[['Number of sexual partners',      'First sexual intercourse',     'Num of pregnancies',   'S
```

In [ ]:
```
with open('summary.tex','w') as tf:
    tf.write(test.round(2).to_latex())
```

In [ ]:
```
risk_factor_df.columns
```

Out[ ]:
```
Index(['Age', 'Number of sexual partners', 'First sexual intercourse',
       'Num of pregnancies', 'Smokes', 'Smokes (years)', 'Smokes (packs/year)',
       'Hormonal Contraceptives', 'Hormonal Contraceptives (years)', 'IUD',
       'IUD (years)', 'STDs', 'STDs (number)', 'STDs:condylomatosis',
       'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
       'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis',
       'STDs:pelvic inflammatory disease', 'STDs:genital herpes',
       'STDs:molluscum contagiosum', 'STDs:AIDS', 'STDs:HIV',
       'STDs:Hepatitis B', 'STDs:HPV', 'STDs: Number of diagnosis',
       'STDs: Time since first diagnosis', 'STDs: Time since last diagnosis',
       'Dx:Cancer', 'Dx:CIN', 'Dx:HPV', 'Dx', 'Hinselmann', 'Schiller',
       'Citology', 'Biopsy', 'age_cat', 'total_std', 'total_tests'],
      dtype='object')
```

In [ ]:
```
label="Dx:Cancer"
```

In [ ]:
```
dx_cancer = px.histogram(risk_factor_df, y=label)
dx_cancer.update_layout(bargap=0.2)
dx_cancer.update_layout(title = "Imbalanced Classes")
dx_cancer.show()
```

In [ ]:
```
X = risk_factor_df.drop([label, "age_cat"], axis=1)
y = risk_factor_df[label].copy()
```

In [ ]:
```
# smote = SMOTE(random_state=42)
# x_smote, y_smote = smote.fit_resample(X, y)
# risk_factor_df = x_smote.join(y_smote)
# risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)
```

In [ ]:
```
adasyn = ADASYN(random_state=42)
x_adasyn,y_adasyn = adasyn.fit_resample(X,y)
risk_factor_df = x_adasyn.join(y_adasyn)
```

In [ ]:
```
# ros = RandomOverSampler(random_state=42)
# x_ros, y_ros = ros.fit_resample(X, y)
# risk_factor_df = x_ros.join(y_ros)
```

In [ ]:
```
risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)
```

In [ ]:
```
dx_cancer = px.histogram(risk_factor_df, y=label)
dx_cancer.update_layout(bargap=0.2)
dx_cancer.update_layout(title = "Balanced Classes")
dx_cancer.show()
```

# Train-Test Split

Data split was stratified on **Age Category**

In [ ]:
```
train_set = None
test_set = None
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in split.split(risk_factor_df, risk_factor_df["age_cat"]):
    train_set = risk_factor_df.loc[train_idx]
    test_set = risk_factor_df.loc[test_idx]
cols_to_drop = ["age_cat","total_std","total_tests"]
for set_ in (train_set, test_set):
    for col in cols_to_drop:
```

```
        set_.drop(col, axis=1, inplace=True)
```

In [ ]:
```
X_train = train_set.drop(label, axis=1)
y_train = train_set[label].copy()

X_test = test_set.drop(label, axis=1)
y_test = test_set[label].copy()

X_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
```

In [ ]:
```
len(X_test.columns)
```

Out[ ]: 35

## Without random var

In [ ]:
```
X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/X_test.csv')
y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/y_test.csv')
X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/X_train.csv')
y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/y_train.csv')
```

## With random var

### Binary

In [ ]:
```
X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/RX_test2.csv')
y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/Ry_test2.csv')
X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/RX_train2.csv')
y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/Ry_train2.csv')
```

### Continuous

In [ ]:
```
X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/RX_test.csv')
y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/Ry_test.csv')
X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/RX_train.csv')
y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/Ry_train.csv')
```

# Comparing different models: RF, SVM, LR, KNN, MLP

In [ ]:
```
from sklearn.metrics import roc_auc_score
```

In [ ]:
```
param_grid = {'C': np.logspace(-5, 8, 15)}
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg, param_grid, cv=10,refit=True).fit(X_train,y_train)
logreg_cv = LogisticRegression(**logreg_cv.best_params_)
```

In [ ]:
```
rnd_clf = RandomForestClassifier()
#rnd_clf.fit(X_train, y_train)
```

In [ ]:
```
knn_clf = KNeighborsClassifier()
knn_param_grid = {"n_neighbors": list(np.arange(1, 100, 2))}
knn_clf_cv = GridSearchCV(knn_clf, knn_param_grid, cv=10,refit=True).fit(X_train,y_train)
knn_clf_cv = KNeighborsClassifier(**knn_clf_cv.best_params_)
```

In [ ]:
```
svm_clf = SVC()
svc_param_grid = {'C': np.logspace(-3, 2, 6), 'gamma': np.logspace(-3, 2, 6), }
svm_clf_cv = GridSearchCV(svm_clf, svc_param_grid, cv=5)
```

In [ ]:
```
from sklearn.neural_network import MLPClassifier

nn_clf = MLPClassifier()
#nn_clf.fit(X_train, y_train)
```

In [ ]:
```
col_names = ["Classifier Name", "Accuracy Score", "Precision Score",
             "Recall Score", "F1 Score", "AUROC"]
summary_df = pd.DataFrame(columns=col_names)

est_name = []
est_acc = []
precision_score = []
recall_score = []
f1score = []
est_conf_matrix = []
roc=[]

estimators = [
```

```
estimators = [
    ("LogisticRegression", logreg_cv),
    ("RandomForestClassifier ", rnd_clf),
    ("KNeighborsClassifier", knn_clf_cv),
    ("SupportVectorClassifier", svm_clf_cv),
    ("MLPClassifier", nn_clf)]

for i in range(0, len(estimators)):
    clf_name = estimators[i][0]
    clf = estimators[i][1]
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    #print(pd.crosstab(y_test,y_pred,rownames=["Actual"],colnames=["predicted"],margins=True))
    roc.append(roc_auc_score(y_test, y_pred, average=None))
    print('roc',roc)
    est_name.append(estimators[i][0])
    est_acc.append(accuracy_score(y_test, y_pred))
    scores = precision_recall_fscore_support(y_test, y_pred, average="weighted")
    print('scores de '+str(clf_name), scores)
    precision_score.append(scores[0])
    recall_score.append(scores[1])
    f1score.append(scores[2])
    est_conf_matrix.append(confusion_matrix(y_test,y_pred))


summary_df[col_names[0]] = est_name
summary_df[col_names[1]] = est_acc
summary_df[col_names[2]] = precision_score
summary_df[col_names[3]] = recall_score
summary_df[col_names[4]] = f1score
summary_df[col_names[5]] = roc
```

```
roc [1.0]
scores de LogisticRegression (1.0, 1.0, 1.0, None)
roc [1.0, 1.0]
scores de RandomForestClassifier  (1.0, 1.0, 1.0, None)
roc [1.0, 1.0, 0.9628571428571429]
scores de KNeighborsClassifier (0.9642001915708812, 0.9613095238095238, 0.961313979066094, None)
roc [1.0, 1.0, 0.9628571428571429, 0.9971428571428572]
scores de SupportVectorClassifier (0.9970421810699589, 0.9970238095238095, 0.997024152746606, None)
roc [1.0, 1.0, 0.9628571428571429, 0.9971428571428572, 1.0]
scores de MLPClassifier (1.0, 1.0, 1.0, None)
```

In [ ]:

```
estimators
```

## Summary

In [ ]:

```
color_scales = ["agsunset","teal","purp","viridis","viridis"]
for i in range(0,len(est_conf_matrix)):
    heatmap = px.imshow(est_conf_matrix[i],aspect="auto",
                        text_auto=True,
                        color_continuous_scale=color_scales[i])
    heatmap.update_layout(title = est_name[i])
    heatmap.update_xaxes(title="Predicted")
    heatmap.update_yaxes(title="Actual")
    heatmap.show()
```

In [ ]:
```
summary_df
```

In [ ]:
```
px.colors.sequential.RdBu
```

In [ ]:
```
#https://plotly.com/python/error-bars/
#https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.07-Error-Bars/
acc_comparison = px.bar(summary_df, x="Classifier Name",
                        y=col_names[1:len(col_names)], labels={"value":"Test Accuracy", "variable":"Metrics"}, tex
                        color_discrete_sequence=["deeppink",
                                                 "deepskyblue",
                                                 "darkviolet",
                                                 "darkorange",
                                                 "darkred"],
                        barmode="group"
                        #,error_y=[dict(type='data', array=[0.5, 1, 2],visible=True), dict(type='data', array=[0.5
                        #,error_y_minus = [dict(type='data', array=[0.5, 1, 2, 2, 1],visible=True), dict(type='dat
                        )
acc_comparison.update_layout({'plot_bgcolor': 'rgba(0, 0, 0, 0)',
'paper_bgcolor': 'rgba(0, 0, 0, 0)'
})
acc_comparison.show()
```

In [ ]:
```
acc_comparison.write_image('/content/drive/My Drive/dataXAI/cancer/modelsperf.png')
```

## Interpretation of the results

- TP: True Positive, these are the values that are positive and were predicted positive
- FP: False Positive, The values which are negative but were wrongly predicted as positive
- TN: True Negative, these are the values that are negative and were predicted negative
- FN: False Negative, The values which are positive but were wrongly predicted as negative

- FN: False Negative, The values which are positive but were wrongly predicted as negative

### Precision

$$precision = \frac{TP}{TP + FP}$$

This metric measures the actual positive outcomes out of the total predicted positive outcomes. It attempts to identify the proportion of positive identifications that were correct. The Logistic Regression model and Support Vector Classifer model performed equally well with a precision score of 99.41%.

In the context of diagnoising cervical cancer, this is metric would not be the most ideal to measure performance, as a negative case being labelled as a positive case is easily solved with confirmatory tests. However, one has to also consider the emotional and mental issues brought upon by being diagnosed with cervical cancer, as this can have a lingering effect even after having confirmatory tests. These tests should be done as soon as possible, as there may be another underlying illness that brought them to see a healthcare professional in the first place.

### Recall

$$recall = \frac{TP}{TP + FN}$$

This metirc measures the correctly positive predicted outocmes of the total number of positive outcomes. It answers the question of what proportions of actual positives were identified correctly. The Logistic Regression model and Support Vector Classifer model performed equally well with a recall score of 99.4%. In terms of measuring performance of the model, this is the metric that should be highly considered.

In the context of diagnosing cervical cancer, we want to reduce the number of false negatives (Actual positive cases labelled as negative cases) as much possible. If an actual positive case is labelled as negative, this has serious consequecnes as the patient would go about their life without actually receiving potentially life saving treatment.

There are many reasons why a cancer can go misdiagnosed, these include:

- The symptoms,especially in the early stages being mistaken for some other type of less serious illness.
- The actual test adminstered by a healthcare professional may give the wrong diagnosis

The 5-year survival rate tells you what percent of people live at least 5 years after the cancer is found. Percent means how many out of 100. The 5-year survival rate for all people with cervical cancer is 66%. Source

Survival rates also depend on the stage of cervical cancer that is diagnosed. When detected at an early stage, the 5-year survival rate for people with invasive cervical cancer is 92%. About 44% of people with cervical cancer are diagnosed at an early stage. If cervical cancer has spread to surrounding tissues or organs and/or the regional lymph nodes, the 5-year survival rate is 58%. If the cancer has

spread to a distant part of the body, the 5-year survival rate is 18%. Source

**It is clearly important and evident that a correct diagnosis and early treatment is the best possible way to ensure that a patient has a high chance of surviving.**

### F1 Score

$$F1Score = \frac{TP}{TP + \frac{FN+FP}{2}}$$

The F1 score is defined as the harmonic mean of precision and recall. Therefore, a high F1 score means both a high precision and recall, same for low and a medium score if one score is high and the other is low.

The Logistic Regression model and Support Vector Classifer model performed equally well with an accuracy score of 99.4%

### Accuracy

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The Logistic Regression model and Support Vector Classifer model performed equally well with an accuracy score of 99.4%

# XAI

## Add a random variable

### Binary

In [ ]:
```
from scipy.stats import bernoulli
```

In [ ]:
```
risk_factor_df['VAR']=bernoulli.rvs(.5, size=risk_factor_df.shape[0])
```

### Continuous

In [ ]:
```
risk_factor_df['VAR']=np.random.normal(loc=0, scale=1, size=risk_factor_df.shape[0])
```

In [ ]:
```
risk_factor_df.columns
```

Out[ ]:
```
Index(['Age', 'Number of sexual partners', 'First sexual intercourse',
       'Num of pregnancies', 'Smokes', 'Smokes (years)', 'Smokes (packs/year)',
       'Hormonal Contraceptives', 'Hormonal Contraceptives (years)', 'IUD',
       'IUD (years)', 'STDs', 'STDs (number)', 'STDs:condylomatosis',
       'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
       'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis',
       'STDs:pelvic inflammatory disease', 'STDs:genital herpes',
       'STDs:molluscum contagiosum', 'STDs:AIDS', 'STDs:HIV',
       'STDs:Hepatitis B', 'STDs:HPV', 'STDs: Number of diagnosis',
       'STDs: Time since first diagnosis', 'STDs: Time since last diagnosis',
       'Dx:Cancer', 'Dx:CIN', 'Dx:HPV', 'Dx', 'Hinselmann', 'Schiller',
       'Citology', 'Biopsy', 'age_cat', 'total_std', 'total_tests', 'VAR'],
      dtype='object')
```

# get data and model

## Without rand

In [ ]:
```
#risk_factor_df.drop(cols_to_drop, axis=1, inplace=True)
#risk_factor_df.to_csv('/content/drive/My Drive/dataXAI/cancer/Rcancer2.csv', index=False)
risk_factor_df=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/cancer.csv')
```

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

## With Rand

### Adding noise to the features

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
```

```
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [ ]:
```
from scipy.stats import bernoulli

X_test['VARB']=bernoulli.rvs(.5, size=X_test.shape[0])
X_train['VARB']=bernoulli.rvs(.5, size=X_train.shape[0])
X_test['VARC']=bernoulli.rvs(.5, size=X_test.shape[0])
X_train['VARC']=bernoulli.rvs(.5, size=X_train.shape[0])
```

In [ ]:
```
for col in X_test.columns:
  X_test[col]+=np.random.normal(loc=0, scale=.1, size=X_test.shape[0])
  X_train[col]+=np.random.normal(loc=0, scale=.1, size=X_train.shape[0])
```

In [ ]:
```
X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/AX_test.csv')
y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/Ay_test.csv')
X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/AX_train.csv')
y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/Ay_train.csv')
```

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/AX_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Ay_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/AX_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Ay_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

## binary

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [ ]:
```
from scipy.stats import bernoulli
```

```
X_test['VAR']=bernoulli.rvs(.5, size=X_test.shape[0])
X_train['VAR']=bernoulli.rvs(.5, size=X_train.shape[0])
```

In [ ]:
```
X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/BX_test.csv')
y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/By_test.csv')
X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/BX_train.csv')
y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/By_train.csv')
```

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/BX_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/By_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/BX_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/By_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [ ]:
```
X_train
```

## Continuous

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [ ]:
```
X_test['VAR']=np.random.normal(loc=0, scale=1, size=X_test.shape[0])
X_train['VAR']=np.random.normal(loc=0, scale=1, size=X_train.shape[0])
```

In [ ]:
```
X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/CX_test.csv')
y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/Cy_test.csv')
X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/CX_train.csv')
y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/Cy_train.csv')
```

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/CX_test.csv')
```

```
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Cy_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/CX_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Cy_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

## Model

In [ ]:
```
rnd_clf = RandomForestClassifier()
rnd_clf.fit(X_train, y_train)
nn_clf.score(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier()
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [ ]:
```
nn_clf = MLPClassifier()
nn_clf.fit(X_train, y_train)
nn_clf.score(X_train, y_train)
```

Out[ ]:
```
0.9992542878448919
```

In [ ]:
```
nn_clf.score(X_test, y_test)
```

Out[ ]:
```
0.9970238095238095
```

In [ ]:
```
model=nn_clf
```

# Local methods

## Generate local FI

In [ ]:
```
!pip install shap
!pip install lime
!pip install interpret-community
!pip install alibi
!pip install treeinterpreter
```

```
!pip install SALib
!pip install dice-ml
!pip install pip install spectralcluster
!pip install -U kaleido
```

In [ ]:

```
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import PartialDependenceDisplay, partial_dependence
from interpret_community.mimic.mimic_explainer import MimicExplainer
from interpret_community.mimic.models import LinearExplainableModel
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from interpret.blackbox import MorrisSensitivity
import shap
import lime
from lime import lime_tabular
from treeinterpreter import treeinterpreter as ti

import pandas as pd
import numpy as np
from numpy import arange
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go

import random
```

data with random variable

Tools

In [ ]:

```
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.DataFrame([[0.0]*X_test.shape[1]]*X_t
fi_1=fi_2=fi_3=fi_4=fi_5=fi_6=fi_7={f'{x}':0.0 for x in X_test.columns}

model = nn_clf

res = dict()
features=X_test.columns
```

In [ ]:

```
print("-GLOSUR-")
# GloSur
explainer = MimicExplainer(model,
                           X_train,
                           LinearExplainableModel,
                           augment_data=False,
```

```
                            features=features,
                            model_task="classification")
global_explanation = explainer.explain_global(X_test)
temp=pd.DataFrame(global_explanation.local_importance_values[1], columns=features)
GloSur=GloSur.add(temp, fill_value=0)


res = dict()
res = global_explanation.get_feature_importance_dict()
fi_1={k: fi_1.get(k, 0) + res.get(k, 0) for k in set(fi_1)}
```

```
-GLOSUR-
```

In [ ]:

```
print("-KSHAP-")
# KSHAP
explainer = shap.KernelExplainer(model.predict_proba, X_train)
shap_values = explainer.shap_values(X_test)
temp=pd.DataFrame(shap_values[1], columns=features)
kernelSHAP=kernelSHAP.add(temp, fill_value=0)

res = dict()
for i in list(kernelSHAP.columns):
  res[i]=np.mean(np.abs(kernelSHAP[i]))
fi_2={k: fi_2.get(k, 0) + res.get(k, 0) for k in set(fi_2)}
```

```
WARNING:shap:Using 1341 background data samples could cause slower run times. Consider using shap.sample(data, K) o
r shap.kmeans(data, K) to summarize the background as K samples.
-KSHAP-
  0%|          | 0/336 [00:00<?, ?it/s]
```

In [ ]:

```
# print("-TSHAP-")
# # TSHAP
# explainer = shap.TreeExplainer(model,X_train)
# shap_values = explainer.shap_values(X_test)

# temp=pd.DataFrame(shap_values[1], columns=features)
# treeSHAP=treeSHAP.add(temp, fill_value=0)

# res = dict()
# for i in list(treeSHAP.columns):
#   res[i]=np.mean(np.abs(treeSHAP[i]))
# fi_3={k: fi_3.get(k, 0) + res.get(k, 0) for k in set(fi_3)}
```

```
-TSHAP-
```

In [ ]:

```
print("-SSHAP-")
# SSHAP
explainer = shap.explainers.Sampling(model.predict_proba, X_train)
```

```
shap_values = explainer.shap_values(X_test)
temp=pd.DataFrame(shap_values[1], columns=features)
samplingSHAP=samplingSHAP.add(temp, fill_value=0)

res = dict()
for i in list(samplingSHAP.columns):
  res[i]=np.mean(np.abs(samplingSHAP[i]))
fi_4={k: fi_4.get(k, 0) + res.get(k, 0) for k in set(fi_4)}
```

```
-SSHAP-
  0%|          | 0/336 [00:00<?, ?it/s]
```

In [ ]:

```
print("-LIME-")

# LIME
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classification',feature_names=X_test.colur

all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_features=X_test.shape[1])
  all.append(sorted(exp.as_map()[1]))


all_res=[]
for i in range(len(all)):
  res = dict()
  for j in range(len(all[0])):
    res[features[j]] = all[i][j][1]
  all_res.append(res)

temp=pd.DataFrame(all_res, columns=features)
limecontrib=limecontrib.add(temp, fill_value=0)

res = dict()
for j in list(limecontrib.columns):
  res[j]=np.mean(np.abs(limecontrib[j]))
fi_5={k: fi_5.get(k, 0) + res.get(k, 0) for k in set(fi_5)}
```

```
-LIME-
```

In [ ]:

```
# print("-TI-")
# # TI
# prediction, bias, contributions = ti.predict(model, X_test)

# all_res=[]
# for i in range(len(contributions)):
#   res = dict()
#   for j in range(len(features)):
#     res[features[j]] = contributions[j][j][1]
```

```
#   all_res.append(res)

# temp=pd.DataFrame(all_res, columns=features)
# ticontrib=ticontrib.add(temp, fill_value=0)

# res = dict()
# for j in list(ticontrib.columns):
#   res[j]=np.mean(np.abs(ticontrib[j]))
# fi_6={k: fi_6.get(k, 0) + res.get(k, 0) for k in set(fi_6)}
```

-TI-

In [ ]:
```
#df.select_dtypes(exclude=int)
```

In [ ]:
```
label = "Dx:Cancer"
```

In [ ]:
```
# temp1=X_train
# temp1['Dx:Cancer']=y_train
# temp2=X_test
# temp2['Dx:Cancer']=y_test

# temp3=pd.concat([temp1,temp2])

# risk_factor_df=temp3
```

In [ ]:
```
print("-DICE-")
import dice_ml
df=risk_factor_df
d = dice_ml.Data(dataframe=df, continuous_features=list(X_test.columns), outcome_name=label)
m = dice_ml.Model(model=model, backend="sklearn")

exp = dice_ml.Dice(d, m, method="random")
query_instance = X_test
e1 = exp.generate_counterfactuals(query_instance, total_CFs=10, desired_range=None,
                                  desired_class="opposite",
                                  permitted_range=None, features_to_vary="all")

imp = exp.local_feature_importance(query_instance, posthoc_sparsity_param=None)
dicecontrib=pd.DataFrame.from_dict(imp.local_importance)

res = dict()
for j in list(dicecontrib.columns):
  res[j]=np.mean(np.abs(dicecontrib[j]))
fi_7={k: fi_7.get(k, 0) + res.get(k, 0) for k in set(fi_7)}
```

```
-DICE-
100%|██████████| 336/336 [02:51<00:00,  1.96it/s]
100%|██████████| 336/336 [01:23<00:00,  4.01it/s]
```

### Without random var

DL

In [ ]:
```
GloSur.to_csv("/content/drive/My Drive/dataXAI/cancer/glosur.csv", index=False)
kernelSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/Kshap.csv", index=False)
#treeSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/Tshap.csv", index=False)
samplingSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/Sshap.csv", index=False)
limecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/lime.csv", index=False)
#ticontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/ti.csv", index=False)
dicecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/dice.csv", index=False)
```

In [ ]:
```
dics = []

fi_1['Method'] = 'Surrogates'
dics.append(fi_1)
fi_2['Method'] = 'KSHAP'
dics.append(fi_2)
# fi_3['Method'] = 'TSHAP'
# dics.append(fi_3)
fi_4['Method'] = 'SSHAP'
dics.append(fi_4)
fi_5['Method'] = 'LIME'
dics.append(fi_5)
# fi_6['Method'] = 'TI'
# dics.append(fi_6)
fi_7['Method'] = 'DICE'
dics.append(fi_7)


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
dics.to_csv("/content/drive/My Drive/dataXAI/cancer/toutfi.csv", index=False)
```

RF

In [ ]:
```
GloSur.to_csv("/content/drive/My Drive/dataXAI/cancer/glosur.csv", index=False)
kernelSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/Kshap.csv", index=False)
treeSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/Tshap.csv", index=False)
samplingSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/Sshap.csv", index=False)
limecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/lime.csv", index=False)
```

```
limecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/lime.csv", index=False)
ticontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/ti.csv", index=False)
dicecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/dice.csv", index=False)
```

In [ ]:

```
dics = []

fi_1['Method'] = 'Surrogates'
dics.append(fi_1)
fi_2['Method'] = 'KSHAP'
dics.append(fi_2)
fi_3['Method'] = 'TSHAP'
dics.append(fi_3)
fi_4['Method'] = 'SSHAP'
dics.append(fi_4)
fi_5['Method'] = 'LIME'
dics.append(fi_5)
fi_6['Method'] = 'TI'
dics.append(fi_6)
fi_7['Method'] = 'DICE'
dics.append(fi_7)


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
dics.to_csv("/content/drive/My Drive/dataXAI/cancer/toutfi.csv", index=False)
```

### With a random variable

All variable

In [ ]:

```
GloSur.to_csv("/content/drive/My Drive/dataXAI/cancer/Aglosur.csv", index=False)
kernelSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/AKshap.csv", index=False)
treeSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/ATshap.csv", index=False)
samplingSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/ASshap.csv", index=False)
limecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Alime.csv", index=False)
ticontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Ati.csv", index=False)
dicecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Adice.csv", index=False)
```

In [ ]:

```
dics = []

fi_1['Method'] = 'Surrogates'
dics.append(fi_1)
fi_2['Method'] = 'KSHAP'
dics.append(fi_2)
fi_3['Method'] = 'TSHAP'
```

```
dics.append(fi_3)
fi_4['Method'] = 'SSHAP'
dics.append(fi_4)
fi_5['Method'] = 'LIME'
dics.append(fi_5)
fi_6['Method'] = 'TI'
dics.append(fi_6)
fi_7['Method'] = 'DICE'
dics.append(fi_7)


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
dics.to_csv("/content/drive/My Drive/dataXAI/cancer/Atoutfi.csv", index=False)
```

binary

In [ ]:

```
GloSur.to_csv("/content/drive/My Drive/dataXAI/cancer/Bglosur.csv", index=False)
kernelSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/BKshap.csv", index=False)
treeSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/BTshap.csv", index=False)
samplingSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/BSshap.csv", index=False)
limecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Blime.csv", index=False)
ticontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Bti.csv", index=False)
dicecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Bdice.csv", index=False)
```

In [ ]:

```
dics = []

fi_1['Method'] = 'Surrogates'
dics.append(fi_1)
fi_2['Method'] = 'KSHAP'
dics.append(fi_2)
fi_3['Method'] = 'TSHAP'
dics.append(fi_3)
fi_4['Method'] = 'SSHAP'
dics.append(fi_4)
fi_5['Method'] = 'LIME'
dics.append(fi_5)
fi_6['Method'] = 'TI'
dics.append(fi_6)
fi_7['Method'] = 'DICE'
dics.append(fi_7)


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
```

```
dics.to_csv("/content/drive/My Drive/dataXAI/cancer/Btoutfi.csv", index=False)
```

In [ ]:
```
dics
```

continue

In [ ]:
```
GloSur.to_csv("/content/drive/My Drive/dataXAI/cancer/Cglosur.csv", index=False)
kernelSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/CKshap.csv", index=False)
treeSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/CTshap.csv", index=False)
samplingSHAP.to_csv("/content/drive/My Drive/dataXAI/cancer/CSshap.csv", index=False)
limecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Clime.csv", index=False)
ticontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Cti.csv", index=False)
dicecontrib.to_csv("/content/drive/My Drive/dataXAI/cancer/Cdice.csv", index=False)
```

In [ ]:
```
dics = []

fi_1['Method'] = 'Surrogates'
dics.append(fi_1)
fi_2['Method'] = 'KSHAP'
dics.append(fi_2)
fi_3['Method'] = 'TSHAP'
dics.append(fi_3)
fi_4['Method'] = 'SSHAP'
dics.append(fi_4)
fi_5['Method'] = 'LIME'
dics.append(fi_5)
fi_6['Method'] = 'TI'
dics.append(fi_6)
fi_7['Method'] = 'DICE'
dics.append(fi_7)


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
dics.to_csv("/content/drive/My Drive/dataXAI/cancer/Ctoutfi.csv", index=False)
```

# Evaluation

## Get explanations

In [ ]:
```
instance=291
```

## Without random var

### DL

In [ ]:
```
gscontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/glosur.csv')
kercontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Kshap.csv')
samcontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Sshap.csv')
#trecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Tshap.csv')
limecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/lime.csv')
#ticontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/ti.csv')
dicecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/dice.csv')
all_fi=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/toutfi.csv')
```

### RF

In [ ]:
```
gscontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/glosur.csv')
kercontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Kshap.csv')
samcontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Sshap.csv')
trecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Tshap.csv')
limecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/lime.csv')
ticontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/ti.csv')
dicecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/dice.csv')
all_fi=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/toutfi.csv')
```

## With random var

### ALL var

In [ ]:
```
gscontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Aglosur.csv')
kercontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/AKshap.csv')
samcontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/ASshap.csv')
trecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/ATshap.csv')
limecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Alime.csv')
ticontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Ati.csv')
dicecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Adice.csv')
all_fi=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Atoutfi.csv')
```

### VAR Continue

In [ ]:
```
gscontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Cglosur.csv')
kercontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/CKshap.csv')
```

```
samcontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/CSshap.csv')
trecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/CTshap.csv')
limecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Clime.csv')
ticontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Cti.csv')
dicecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Cdice.csv')
all_fi=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Ctoutfi.csv')
```

VAR binary

In [ ]:
```
gscontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Bglosur.csv')
kercontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/BKshap.csv')
samcontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/BSshap.csv')
trecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/BTshap.csv')
limecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Blime.csv')
ticontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Bti.csv')
dicecontrib=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Bdice.csv')
all_fi=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/Btoutfi.csv')
```

In [ ]:
```
# res = dict()
# for i in list(kercontrib.columns):
#   res[i]=np.mean(np.abs(kercontrib[i]))
# fi_2={k: fi_2.get(k, 0) + res.get(k, 0) for k in set(fi_2)}
# fi_2['Method'] = 'KSHAP'
```

In [ ]:
```
# all_fi.loc[1]=fi_2
```

In [ ]:
```
# all_fi
```

In [ ]:
```
# all_fi.to_csv("/content/drive/My Drive/dataXAI/cancer/Btoutfi.csv", index=False)
```

TOOLS

In [ ]:
```
all_fi.fillna(0, inplace=True)
all_fi.iloc[:,:-1]=np.abs(all_fi.iloc[:,:-1])
all_fi.reset_index(drop=True, inplace=True)
```

In [ ]:
```
label="Dx:Cancer"
```

In [ ]:
```
methods_all_fi['Method'].to_list()
```

```
methods=all_fi['Method'].to_list()
weights=[gscontrib, kercontrib, samcontrib, limecontrib, dicecontrib]
```

In [ ]:
```
methods=all_fi['Method'].to_list()
weights=[gscontrib, kercontrib, trecontrib, samcontrib, limecontrib, ticontrib, dicecontrib]
```

## Normalize ?

In [ ]:
```
gscontrib_norm=gscontrib.div(gscontrib.sum(axis=1), axis=0)
kercontrib_norm=kercontrib.div(kercontrib.sum(axis=1), axis=0)
samcontrib_norm=samcontrib.div(samcontrib.sum(axis=1), axis=0)
trecontrib_norm=trecontrib.div(trecontrib.sum(axis=1), axis=0)
limecontrib_norm=limecontrib.div(limecontrib.sum(axis=1), axis=0)
ticontrib_norm=ticontrib.div(ticontrib.sum(axis=1), axis=0)
dicecontrib_norm=dicecontrib.div(dicecontrib.sum(axis=1), axis=0)
```

## One instance

In [ ]:
```
risk_factor_df.describe().iloc[1]
```

In [ ]:
```
xx=risk_factor_df.describe().iloc[1]
```

In [ ]:
```
instance=3
```

In [ ]:
```
xx=X_test.iloc[instance]
```

In [ ]:
```
idx=list(xx.to_numpy().nonzero()[0])
```

In [ ]:
```
xx=xx.to_frame()
xxx=xx.T.columns
new=pd.DataFrame()
for i in range(len(xxx)):
  if i in idx:
    new[xxx[i]]=xx.T[xxx[i]]
```

In [ ]:
```
new.T.round(2)
```

Out[ ]:

| | 291 |
|---|---|
| **Age** | 27.00 |
| **Number of sexual partners** | 2.00 |
| **First sexual intercourse** | 14.00 |
| **Num of pregnancies** | 3.00 |
| **Hormonal Contraceptives (years)** | 0.86 |
| **STDs: Time since first diagnosis** | 4.00 |
| **STDs: Time since last diagnosis** | 3.00 |
| **Dx:HPV** | 1.00 |
| **Dx** | 1.00 |

In [ ]:

```
with open('instance.tex','w') as tf:
    tf.write(new.T.round(2).to_latex())
```

## Instance datafame

In [ ]:

```
one_instance=[]

for i in range(len(methods)):
  one_instance.append(weights[i].iloc[instance])

one_instance=pd.DataFrame(one_instance, columns=X_test.columns)
one_instance['methods']=methods
one_instance.set_index('methods', inplace=True)

one_instance.to_csv('/content/drive/My Drive/dataXAI/cancer/sonali/one_instance.csv')
```

In [ ]:

```
'methods' in one_instance.columns
```

In [ ]:

```
one_instance
```

Out[ ]:

| | Age | Number of sexual partners | First sexual intercourse | Num of pregnancies | Smokes | Smokes (years) | Smokes (packs/year) | Hormonal Contraceptives | Hormonal Contraceptives (years) |
|---|---|---|---|---|---|---|---|---|---|

| methods | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Surrogates** | -0.558553 | 0.018907 | 0.155247 | -0.225771 | 0.113706 | -0.733447 | 5.014138e-02 | 0.331956 | -0.442469 |
| **KSHAP** | 0.018433 | -0.002457 | -0.021277 | 0.006551 | 0.008620 | -0.001977 | 0.000000e+00 | 0.024006 | 0.020089 |
| **TSHAP** | 0.019730 | -0.003178 | -0.021415 | 0.008250 | 0.006676 | -0.000964 | 5.158781e-07 | 0.027791 | 0.024452 |
| **SSHAP** | 0.019459 | 0.001181 | -0.019259 | 0.003429 | 0.005861 | -0.001945 | 5.598938e-04 | 0.024208 | 0.015767 |
| **LIME** | -0.004105 | -0.001152 | -0.043255 | -0.000816 | 0.058835 | -0.024221 | -4.774516e-03 | 0.059702 | 0.025091 |
| **TI** | 0.044565 | 0.001765 | -0.042974 | 0.007374 | 0.005567 | -0.000827 | -2.342180e-03 | 0.030810 | 0.035062 |
| **DICE** | 0.200000 | 0.000000 | 0.000000 | 0.100000 | 0.000000 | 0.100000 | 0.000000e+00 | 0.000000 | 0.100000 |

7 rows × 35 columns

## Plot FI for one instance

In [ ]:
```
X_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('/content/drive/My Drive/dataXAI/cancer/y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [ ]:
```
instance=291
var='W'
maxx=10
f=''
#vale=0
```

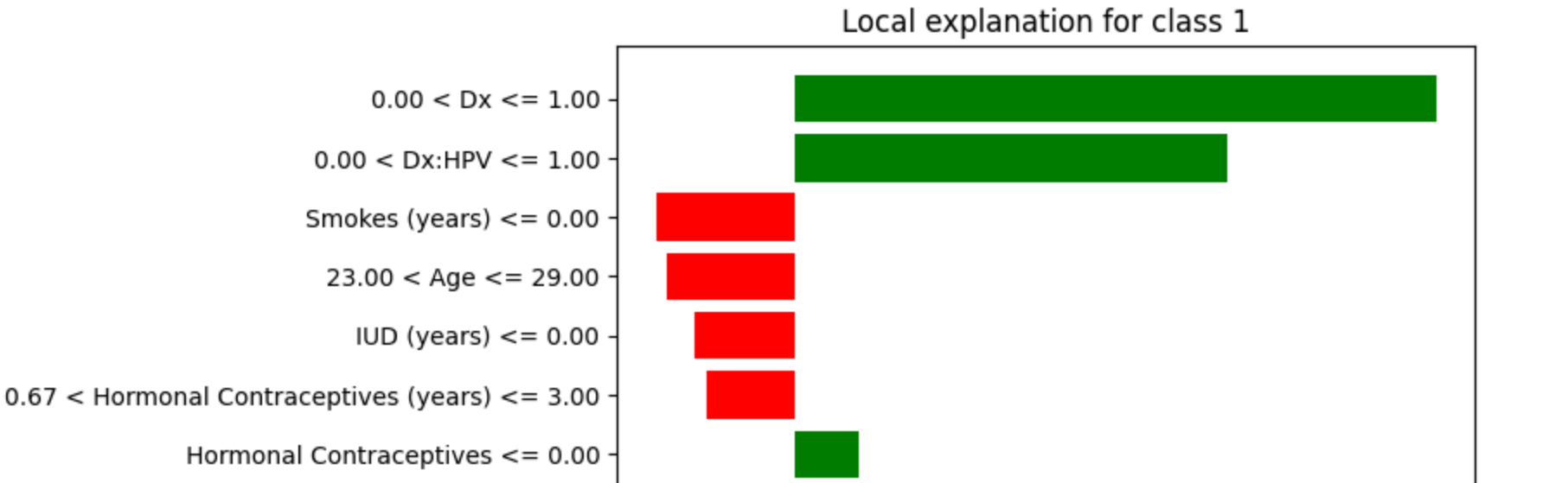
In [ ]:
```
# tempt=X_test.iloc[291]
# tempt[f]=vale
# #tempt["Age"]=1
# #tempt["Num of pregnancies"]=120
# #tempt["Smokes"]=200
# tempt
# X_test.iloc[291]=tempt
# X_test.iloc[291]
```

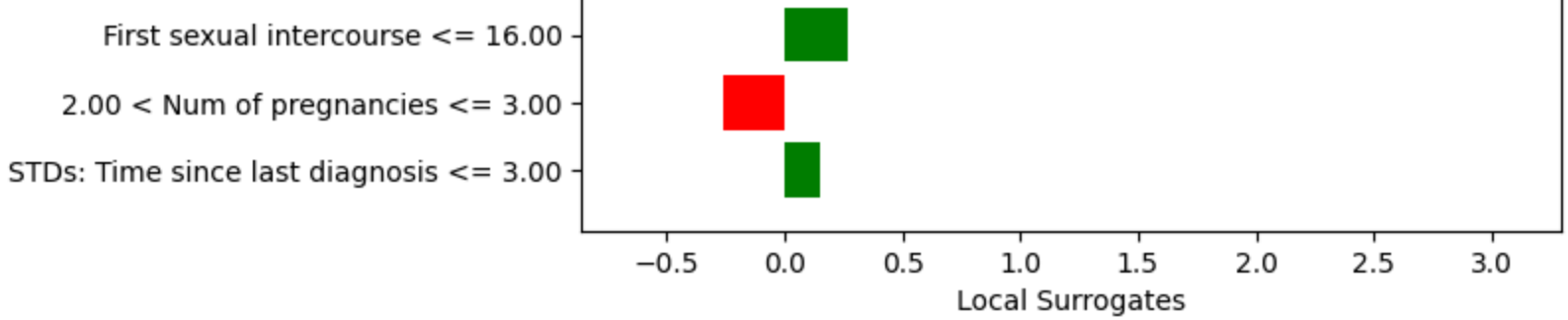
In [ ]:
```
model.predict(X_test)
```

In [ ]:
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classification',feature_names=X_test.colur
exp = explainer.explain_instance(X_test.iloc[instance], model.predict_proba, num_features=X_test.shape[1])
```

In [ ]:
```
items = gscontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=False,show_predicted_value=False)
```

In [ ]:
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Local Surrogates")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'surrogate'+str(instance)+str(f)+'.png', bbox_inche
```



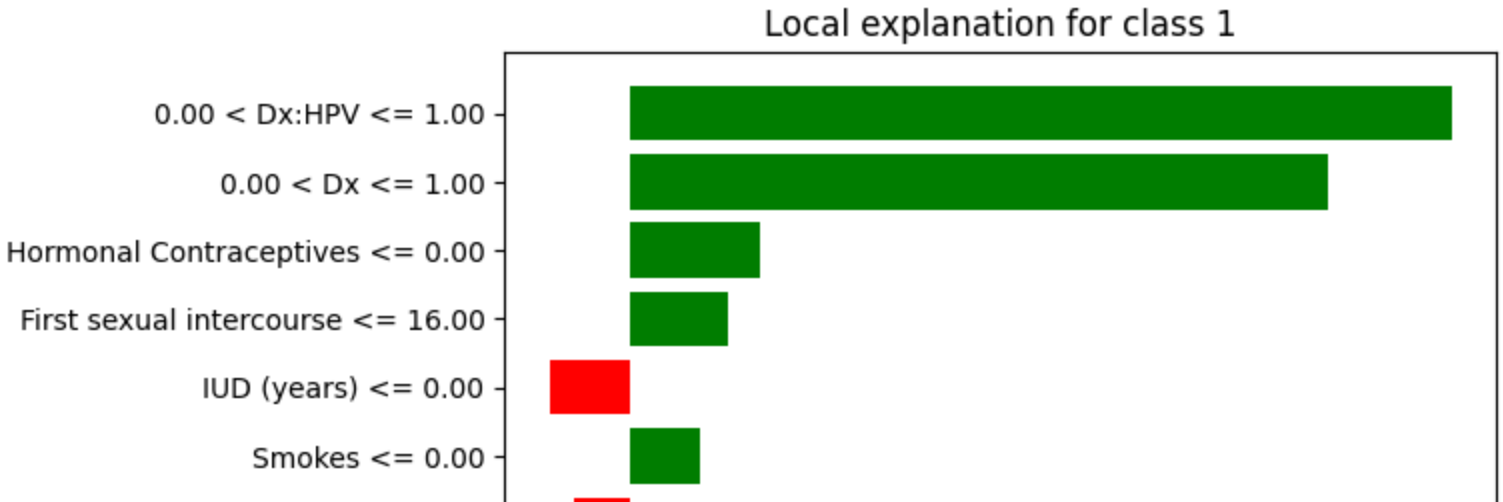


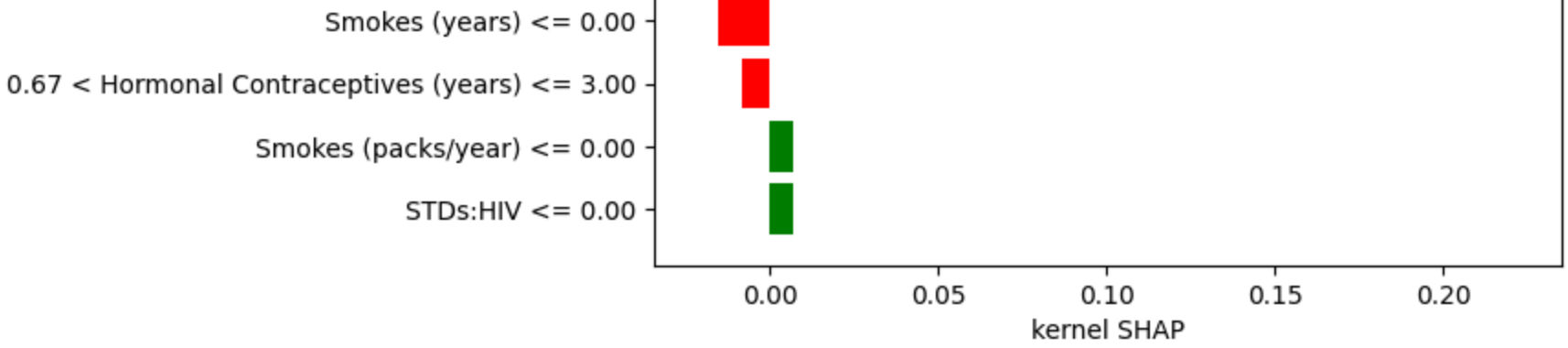

In [ ]:

```
items = kercontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=False,show_predicted_value=False)
```

In [ ]:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("kernel SHAP")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'kernelSHAP'+str(instance)+str(f)+'.png', bbox_inch
```



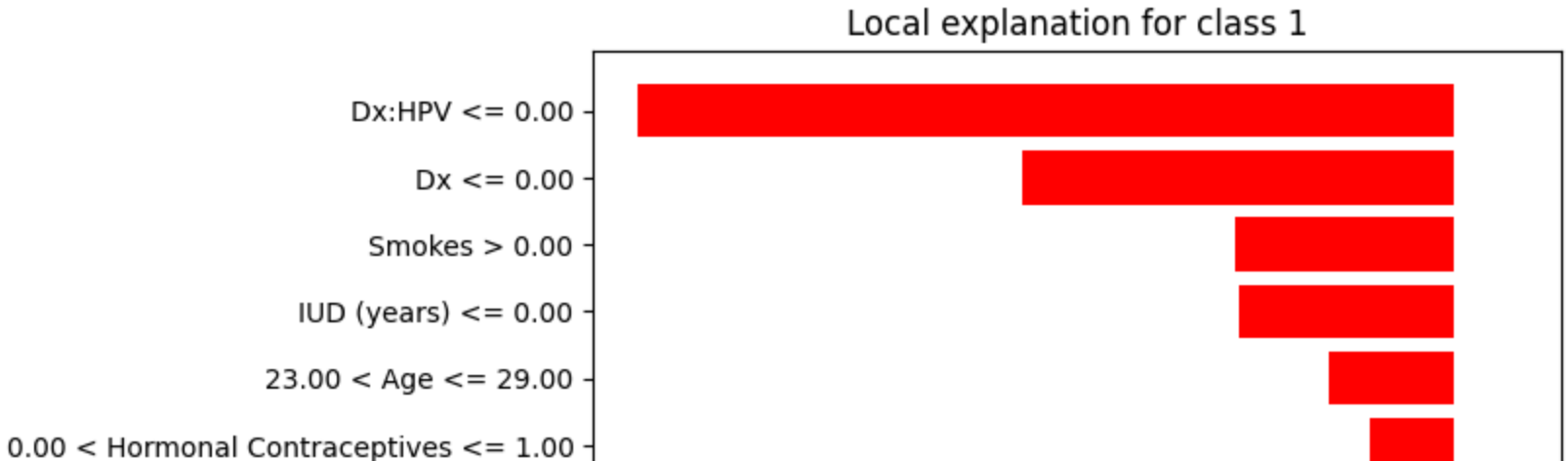


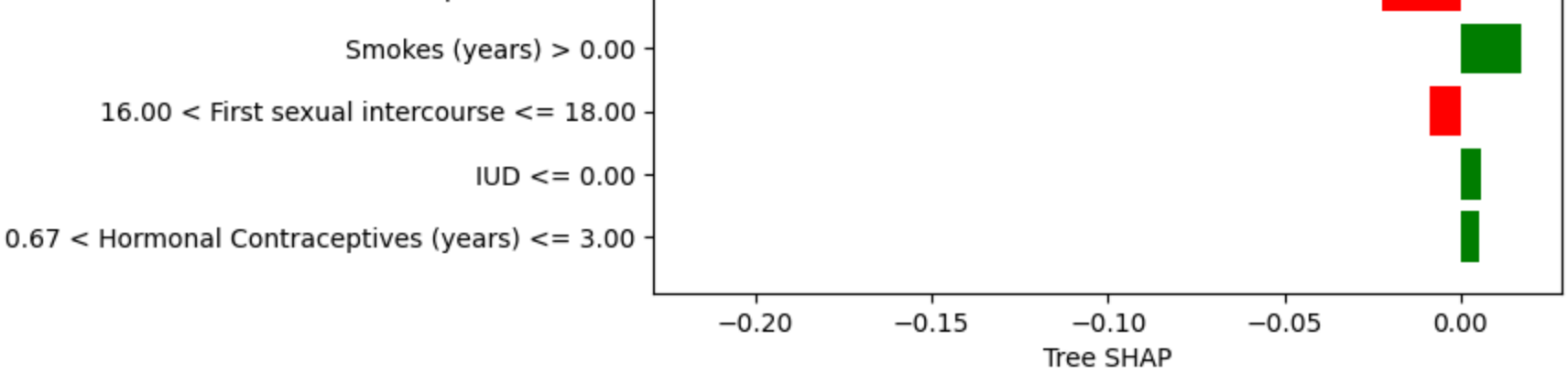

In [ ]:

```
items = trecontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=True,show_predicted_value=False)
```

In [ ]:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Tree SHAP")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'treeSHAP'+str(instance)+str(f)+'.png', bbox_inches
```



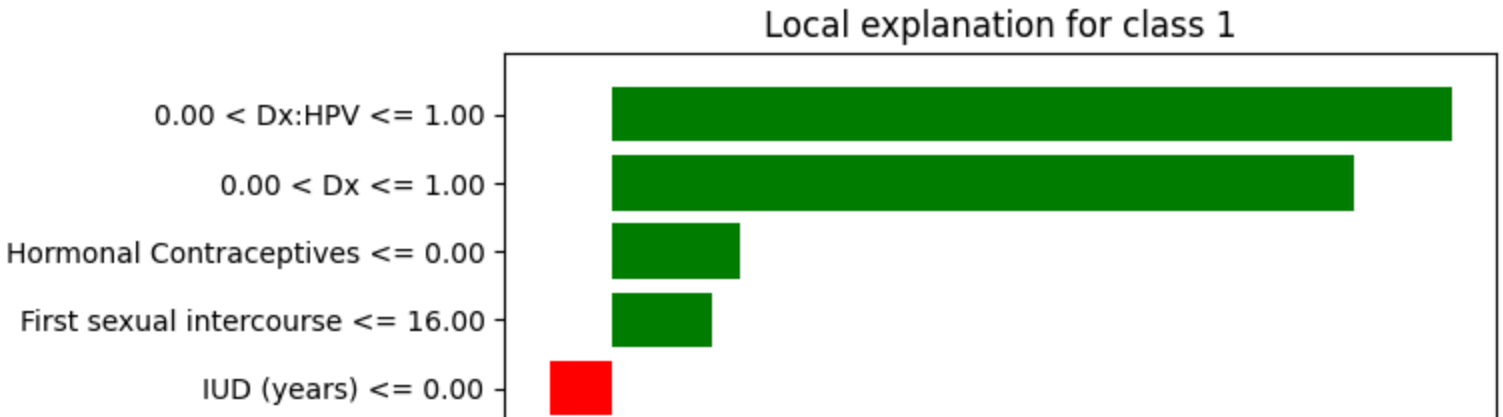


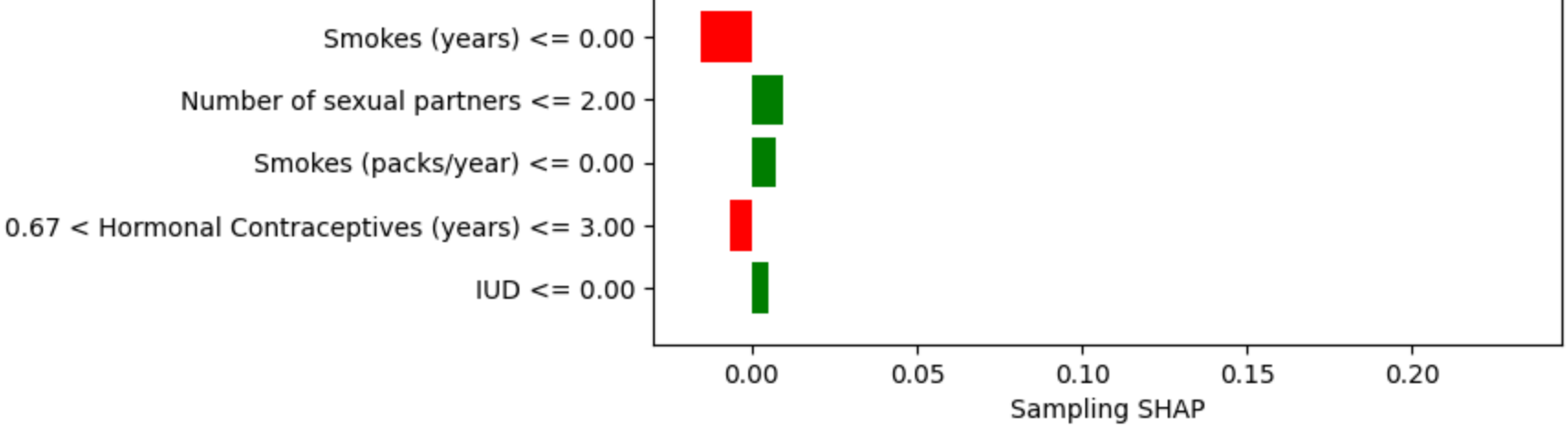

In [ ]:

```
items = samcontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=True,show_predicted_value=False)
```

In [ ]:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Sampling SHAP")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'samplingSHAP'+str(instance)+str(f)+'.png', bbox_i
```



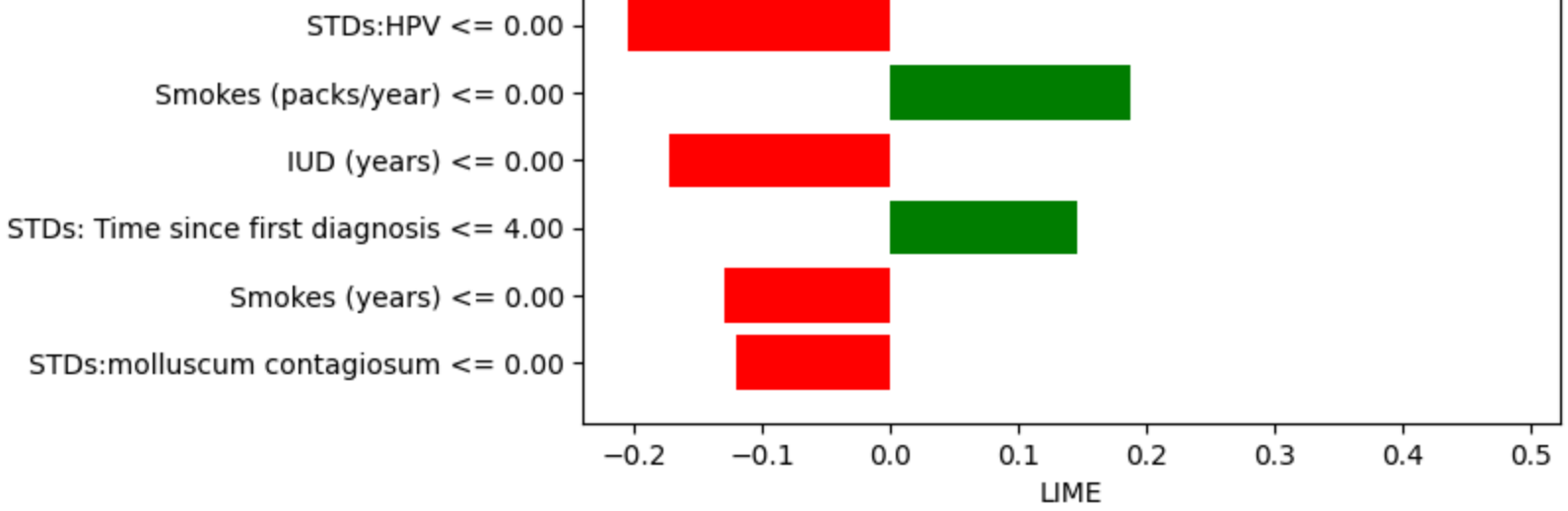



In [ ]:

```
items = limecontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=True,show_predicted_value=False)
```

In [ ]:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("LIME")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'lime'+str(instance)+str(f)+'.png', bbox_inches='t
```

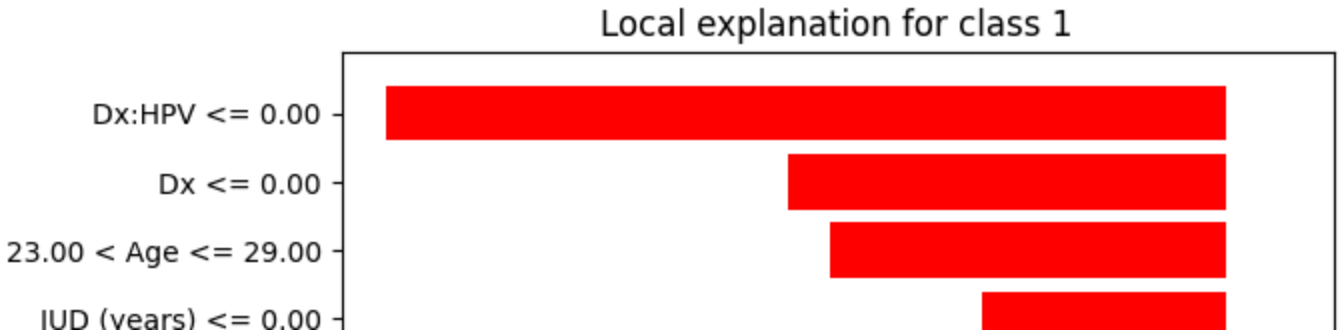


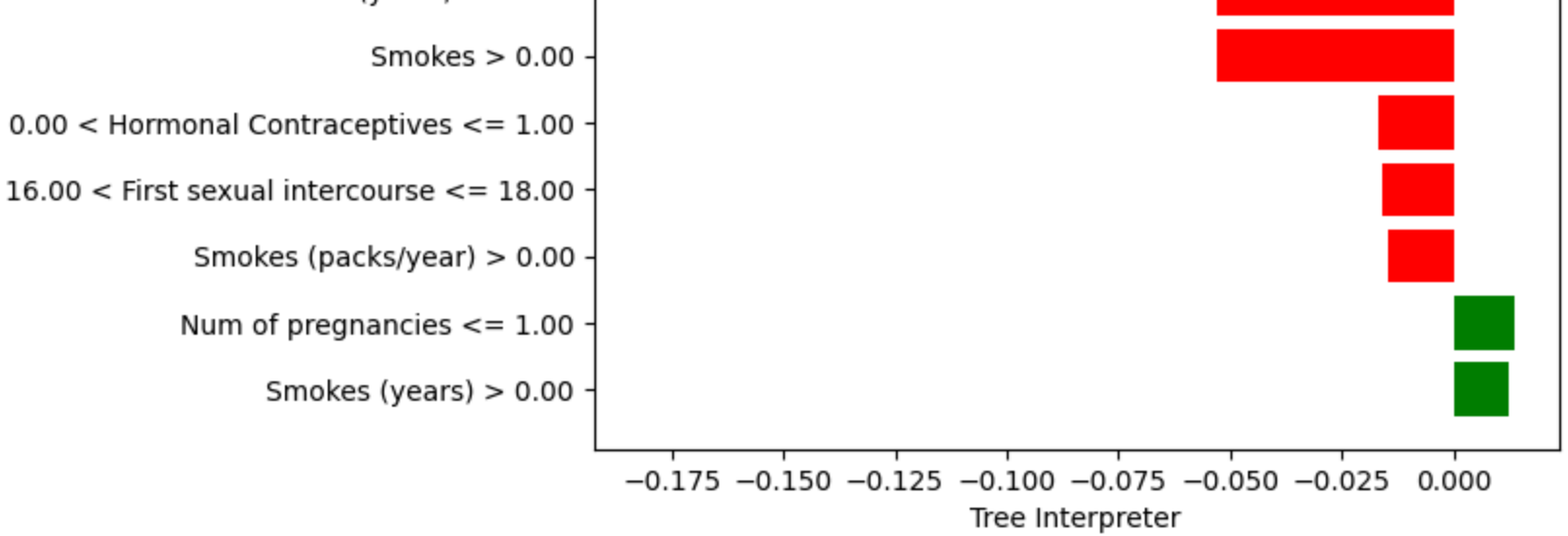

In [ ]:

```
items = ticontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=False,show_predicted_value=False)
```

In [ ]:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Tree Interpreter")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'ti'+str(instance)+str(f)+'.png', bbox_inches='tigh
```



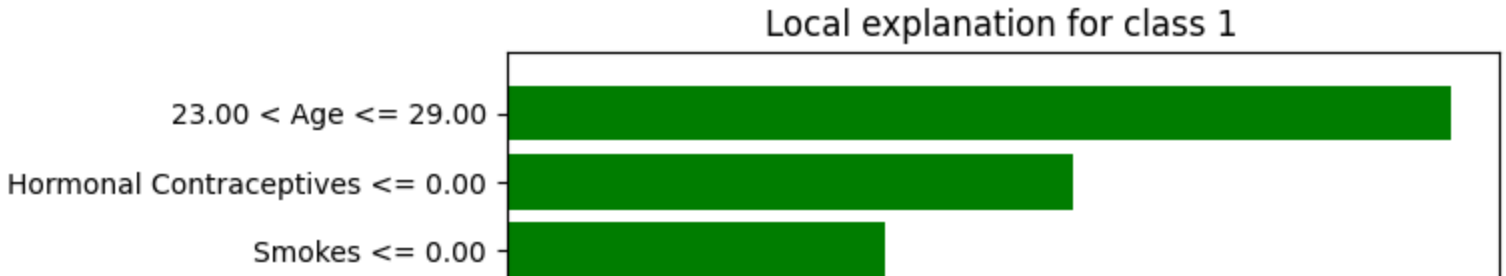


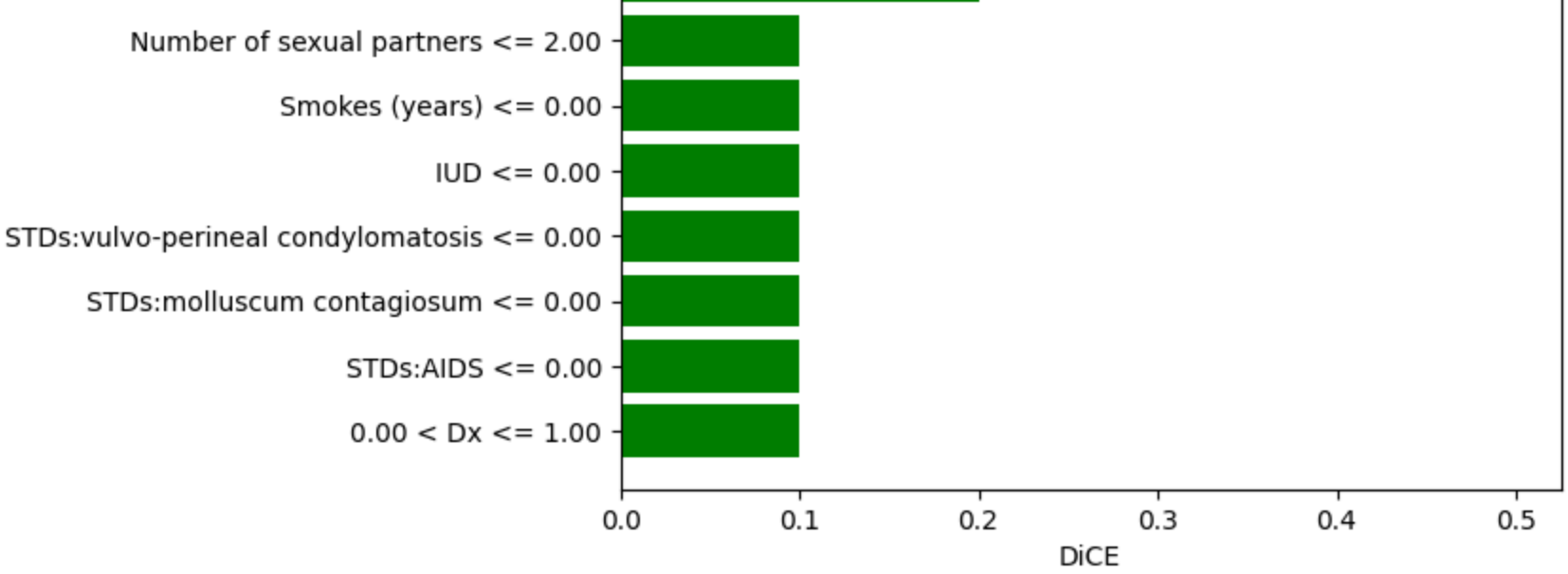

In [ ]:

```
items = dicecontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

t = sorted(t, key=lambda tup: abs(tup[1]),  reverse=True)

exp_test = {1: t[0:maxx]}


exp.local_exp = exp_test
exp.show_in_notebook(show_table=False,show_predicted_value=False)
```

In [ ]:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("DiCE")
fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'dice'+str(instance)+str(f)+'.png', bbox_inches='t
```






## ROAR

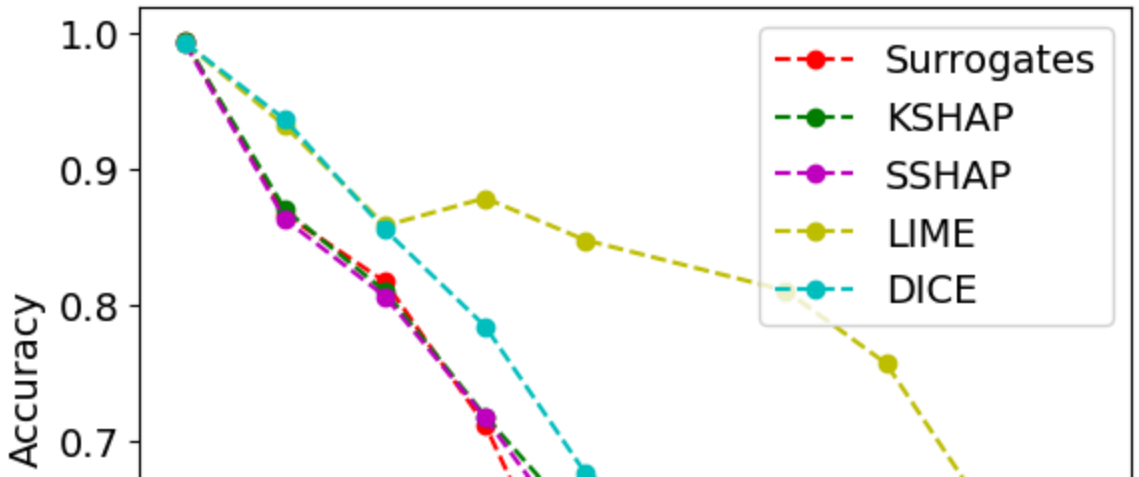
In [ ]:

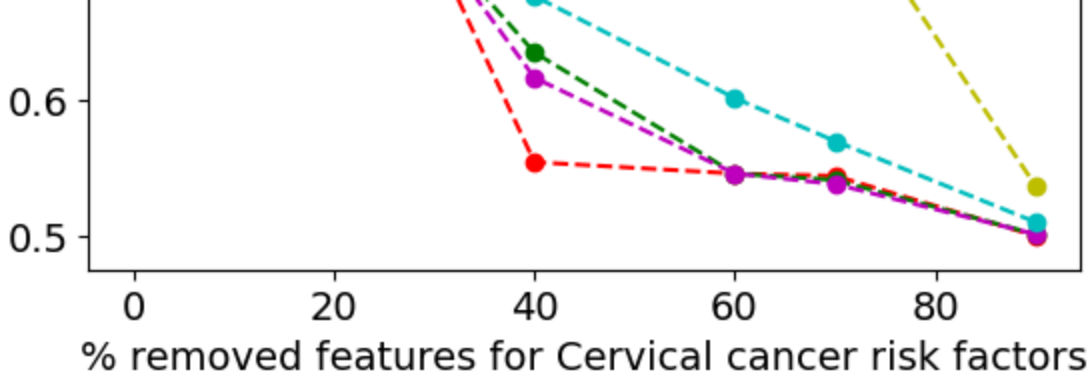
```
from matplotlib import pyplot as plt
from sklearn.model_selection import StratifiedKFold, cross_val_score

def roar(featImp, feature_to_predict, datapath,savepath,dataname):

  a=['ro--', 'go--', 'mo--', 'yo--', 'co--', 'ko--', 'bo--']
  pourc=[0,10,20,30,40,60,70,90]
  font = {'size'   : 14}

  plt.rc('font', **font)


  for k in range(featImp.shape[0]):
    accuracies=[]
    new=[]
    for i in pourc:

      fi= featImp.iloc[k,:]
      fi.drop('Method', inplace=True)
      fi=fi.to_dict()
      fi=dict(sorted(fi.items(), key=lambda x:x[1], reverse=True))

      fii=list(fi.keys())

      df= pd.read_csv(datapath)
```

```
        top=int((len(df.columns)*i)/100)
        fii = fii[top:]

        #df.drop(fii[0:top], axis=1, inplace=True)
        new=fii
        new.append(feature_to_predict)
        df=df[new]

        X=np.array(df[list(df.columns.drop(feature_to_predict))])
        y=np.array(df[feature_to_predict])

        model = RandomForestClassifier(random_state = 42)


        scores = cross_val_score(model, X, y, cv=10)
        accuracies.append(np.mean(scores))

      plt.plot(pourc, accuracies, a[k], label=featImp['Method'][k])


    plt.xlabel('% removed features for Cervical cancer risk factors')


    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.savefig(savepath+'roar.png', bbox_inches='tight', dpi=300)
    plt.show()
```

In [ ]:

```
datapath='/content/drive/My Drive/dataXAI/cancer/cancer.csv'
savepath= '/content/drive/My Drive/dataXAI/cancer/'
dataname='Cervical cancer'

roar(all_fi, label, datapath, savepath, dataname)
```






In [ ]:
```
all_fi
```

## SHAPASH

https://towardsdatascience.com/building-confidence-on-explainability-methods-66b9ee575514

In [ ]:
```
!pip install shapash
```

In [ ]:
```
from shapash.explainer.consistency import Consistency
from shapash import SmartExplainer
```

In [ ]:
```
from shapash.explainer.consistency import Consistency
from shapash import SmartExplainer
cns=Consistency()
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test)
```

### Contribution plots

In [ ]:
```
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribage.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
```

In [ ]:
```
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagekernel.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagesampling.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=trecontrib)
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagetree.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagelime.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=ticontrib)
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribageti.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
img=xpl.plot.contribution_plot(0)
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagegs.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=dicecontrib)
xpl.plot.contribution_plot(0)
```

### Contributions plot

In [ ]:
```
fig_image=xpl.plot.contribution_plot(0)
plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/age.png')
```

In [ ]:
```
fig_image=xpl.plot.contribution_plot(0)
plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/age.png')
```

```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
fig_image=xpl.plot.contribution_plot(29)
plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/dxhpv.png')
```

```
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
fig_image=xpl.plot.contribution_plot(30)
plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/dxhpv.png')
```

```
<Figure size 640x480 with 0 Axes>
```

### Consistency

In [ ]:
```
pairwise_consistency=cns.calculate_all_distances(methods, weights)
```

In [ ]:
```
test=pairwise_consistency[1].round(2)
```

In [ ]:
```
test.style.background_gradient(cmap='Paired_r')
```

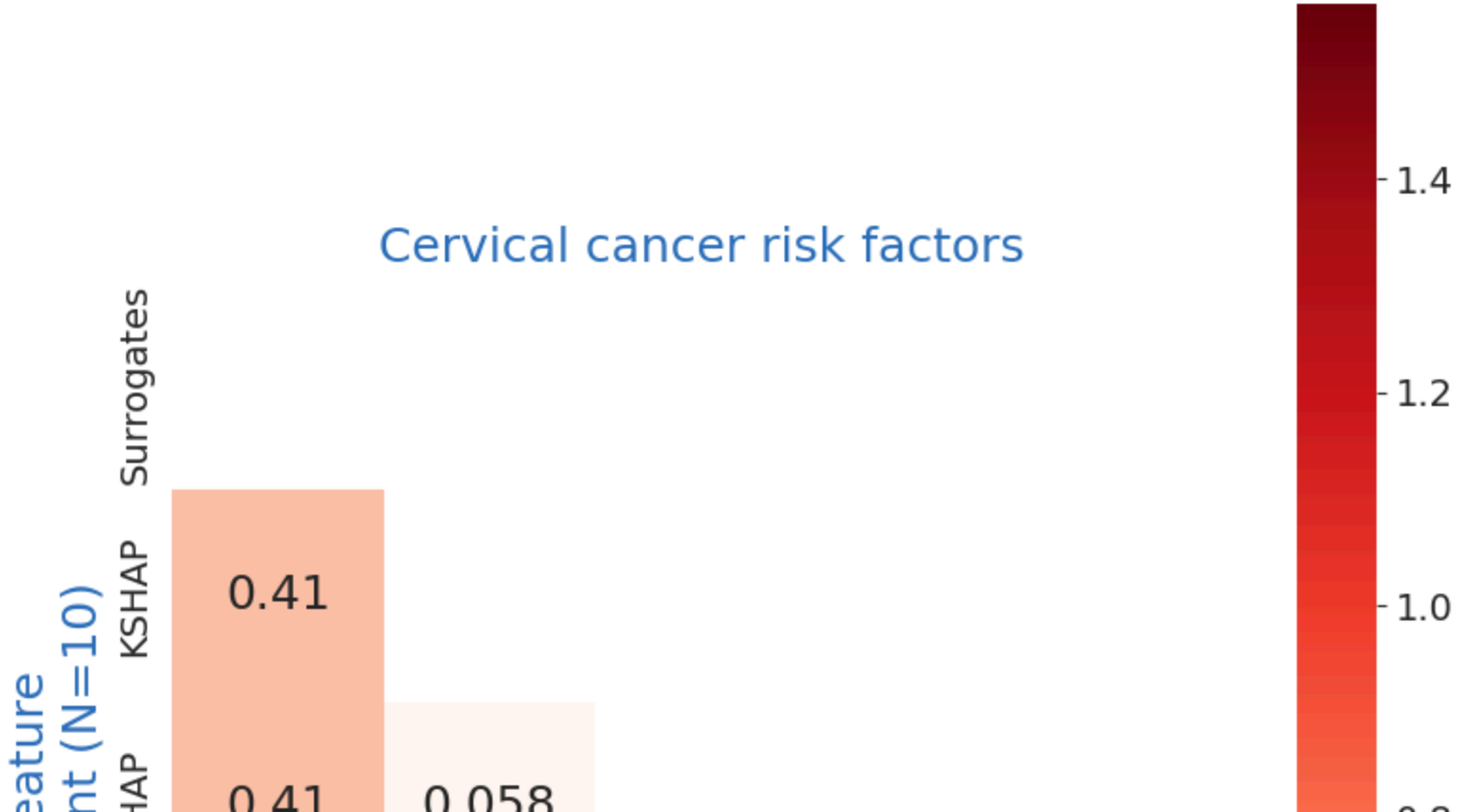
Out[ ]:

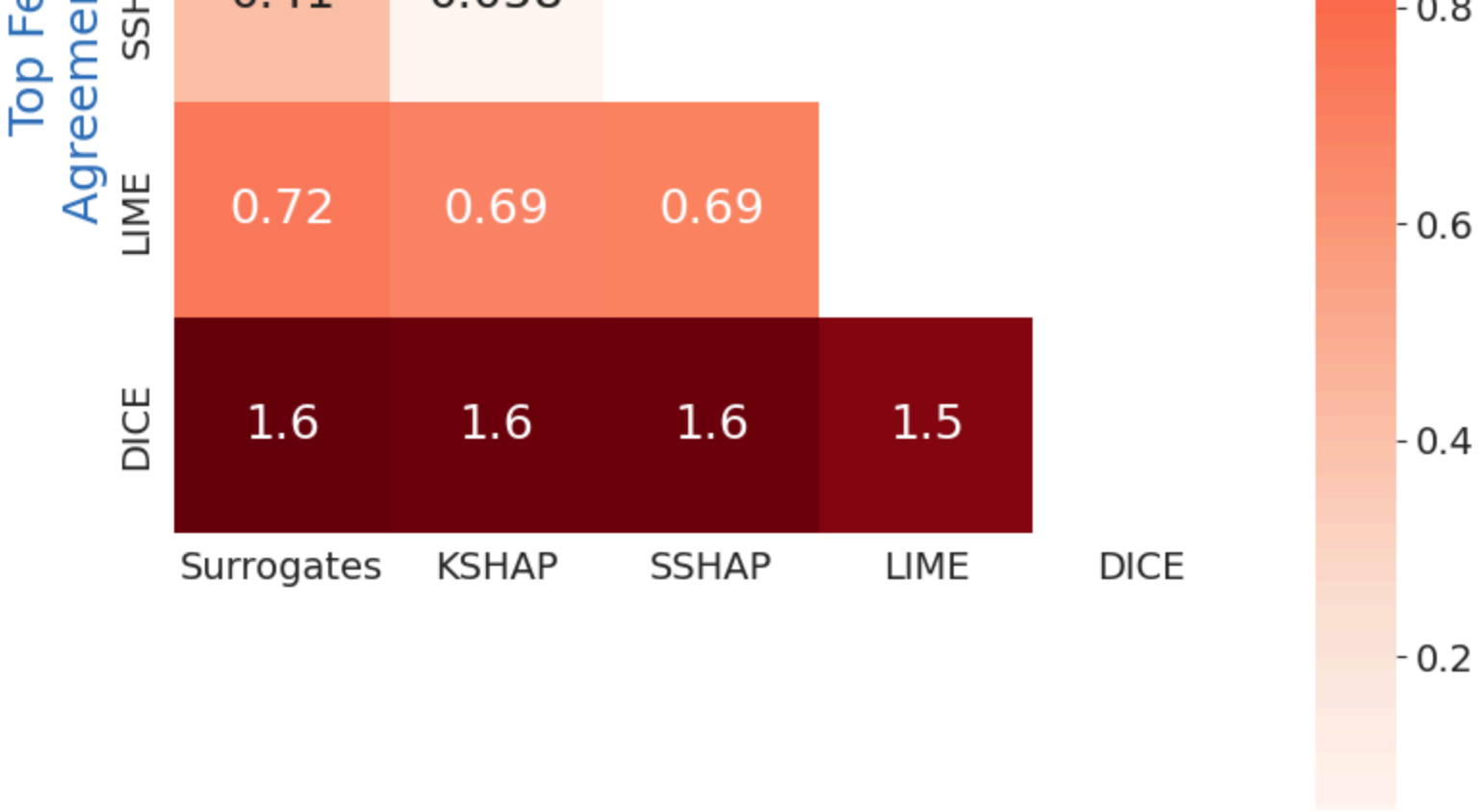
| | Surrogates | KSHAP | SSHAP | LIME | DICE |
|---|---|---|---|---|---|
| **Surrogates** | 0.000000 | 0.410000 | 0.410000 | 0.720000 | 1.560000 |
| **KSHAP** | 0.410000 | 0.000000 | 0.060000 | 0.690000 | 1.560000 |
| **SSHAP** | 0.410000 | 0.060000 | 0.000000 | 0.690000 | 1.560000 |
| **LIME** | 0.720000 | 0.690000 | 0.690000 | 0.000000 | 1.470000 |

| | | | | | |
|---|---|---|---|---|---|
| **DICE** | 1.560000 | 1.560000 | 1.560000 | 1.470000 | 0.000000 |

In [ ]:
```
with open('/content/drive/My Drive/dataXAI/cancer/consistency.tex','w') as tf:
    tf.write(test.to_latex())
```

In [ ]:
```
corr = pairwise_consistency[1]
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
fig = plt.figure(figsize=(9, 11))
with sns.axes_style("white"):

    ax = sns.heatmap(corr, mask=mask, square=True,annot=True, annot_kws={'fontsize': 18},
                xticklabels=methods, yticklabels=methods, cmap="Reds", cbar=True)
    ax.set_title("Cervical cancer risk factors", color='xkcd:medium blue', fontsize=18)
    ax.set_ylabel('Top Feature\nAgreement (N=10)', color='xkcd:medium blue', fontsize=18)



    plt.show()
```






In [ ]:
```
fig.savefig('/content/drive/My Drive/dataXAI/cancer/consistency.png', bbox_inches='tight', dpi=300)
```

Mean Consistency

In [ ]:
```
for i in pairwise_consistency[1].columns:
  print(i, round(np.mean(pairwise_consistency[1][i]),2))
```

```
Surrogates 0.62
KSHAP 0.54
SSHAP 0.54
LIME 0.72
DICE 1.23
```

## Compactness

In [ ]:
```
def get_distance(selection, contributions, mode, nb_features):

    if mode == "classification" and len(contributions) == 2:
        contributions = contributions[1]
```

```
    assert nb_features <= contributions.shape[1]

    contributions = contributions.loc[selection].values
    top_features = np.array([sorted(row, key=abs, reverse=True) for row in contributions])[:, :nb_features]
    output_top_features = np.sum(top_features[:, :], axis=1)
    output_all_features = np.sum(contributions[:, :], axis=1)

    if mode == "regression":
        distance = abs(output_top_features - output_all_features) / abs(output_all_features)
    elif mode == "classification":
        distance = abs(output_top_features - output_all_features)
    return distance


def get_min_nb_features(selection, contributions, mode, distance):

    assert 0 <= distance <= 1

    if mode == "classification" and len(contributions) == 2:
        contributions = contributions[1]
    contributions = contributions.loc[selection].values
    features_needed = []
    # For each instance, add features one by one (ordered by SHAP) until we get close enough
    for i in range(contributions.shape[0]):
        ids = np.flip(np.argsort(np.abs(contributions[i, :])))
        output_value = np.sum(contributions[i, :])

        score = 0
        for j, idx in enumerate(ids):
            # j : number of features needed
            # idx : positions of the j top shap values
            score += contributions[i, idx]
            # CLOSE_ENOUGH
            if mode == "regression":
                if abs(score - output_value) < distance * abs(output_value):
                    break
            elif mode == "classification":
                if abs(score - output_value) < distance:
                    break
        features_needed.append(j + 1)
    return features_needed



def compute_features_compacity(case, contributions, selection, distance, nb_features):
        #if (case == "classification") and (len(classes) > 2):
        #    raise AssertionError("Multi-class classification is not supported")

        features_needed = get_min_nb_features(selection, contributions, case, distance)
        distance_reached = get_distance(selection, contributions, case, nb_features)
```

```
        # We clip large approximations to 100%
        distance_reached = np.clip(distance_reached, 0, 1)

        return {"features_needed": features_needed, "distance_reached": distance_reached}
```

In [ ]:
```
compacities=[]

for weight in weights:
  rr=compute_features_compacity(case="classification", contributions=weight, selection=list(range(0, len(X_test))))
  #rr=compute_features_compacity(case="classification", contributions=weight, selection=[instance], distance=0.9,
  compacities.append(pd.DataFrame.from_dict(rr))
```

In [ ]:
```
maxx=[]
for c in compacities:
  maxx.append(c.iloc[c.distance_reached.idxmax()].tolist())
compacity=pd.DataFrame(data=maxx, columns=['features_needed', 'distance_reached'])
compacity['Method']=methods
compacity.set_index('Method', drop=True).round(2)
```

Out[ ]:

| Method | features_needed | distance_reached |
|---|---|---|
| **Surrogates** | 2.0 | 1.00 |
| **KSHAP** | 1.0 | 0.18 |
| **SSHAP** | 1.0 | 0.17 |
| **LIME** | 1.0 | 1.00 |
| **DICE** | 8.0 | 1.00 |

In [ ]:
```
with open('/content/drive/My Drive/dataXAI/cancer/compactness'+str(instance)+'.tex','w') as tf:
    tf.write(compacity.to_latex())
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
xpl.plot.compacity_plot
```

Plots

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
```

```
img=xpl.plot.compacity_plot()
```

In [ ]:
```
img
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactgs.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
img=xpl.plot.compacity_plot()
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactkernel.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=trecontrib)
img=xpl.plot.compacity_plot()
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compacttree.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
img=xpl.plot.compacity_plot()
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactsampling.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img=xpl.plot.compacity_plot()
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactlime.png')
```

In [ ]:
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=ticontrib)
img=xpl.plot.compacity_plot()
```

```
xpl.plot.compacity_plot()
```

## Stability

tool

In [ ]:

```
from sklearn.preprocessing import normalize


def _compute_distance(x1, x2, mean_vector, epsilon=0.0000001):
    """
    Compute distances between data points by using L1 on normalized data : sum(abs(x1-x2)/(mean_vector+epsilon))
    Parameters
    ----------
    x1 : array
        First vector
    x2 : array
        Second vector
    mean_vector : array
        Each value of this vector is the std.dev for each feature in dataset
    Returns
    -------
    diff : float
        Returns :math:`\\sum(\\frac{|x1-x2|}{mean\_vector+epsilon})`
    """
    diff = np.sum(np.abs(x1 - x2) / (mean_vector + epsilon))
    return diff


def _compute_similarities(instance, dataset):
    """
    Compute pairwise distances between an instance and all other data points
    Parameters
    ----------
    instance : 1D array
        Reference data point
    dataset : 2D array
        Entire dataset used to identify neighbors
    Returns
    -------
    similarity_distance : array
        V[i] == distance between actual instance and instance i
```

```
    """
    mean_vector = np.array(dataset, dtype=np.float32).std(axis=0)
    similarity_distance = np.zeros(dataset.shape[0])

    for j in range(0, dataset.shape[0]):
        # Calculate distance between point and instance j
        dist = _compute_distance(instance, dataset[j], mean_vector)
        similarity_distance[j] = dist

    return similarity_distance


def _get_radius(dataset, n_neighbors, sample_size=50, percentile=95):
    """
    Calculate the maximum allowed distance between points to be considered as neighbors
    Parameters
    ----------
    dataset : DataFrame
        Pool to sample from and calculate a radius
    n_neighbors : int
        Maximum number of neighbors considered per instance
    sample_size : int, optional
        Number of data points to sample from dataset, by default 500
    percentile : int, optional
        Percentile used to calculate the distance threshold, by default 95
    Returns
    -------
    radius : float
        Distance threshold
    """
    # Select 500 points max to sample
    size = min([dataset.shape[0], sample_size])
    # Randomly sample points from dataset
    sampled_instances = dataset[np.random.randint(0, dataset.shape[0], size), :]
    # Define normalization vector
    mean_vector = np.array(dataset, dtype=np.float32).std(axis=0)
    # Initialize the similarity matrix
    similarity_distance = np.zeros((size, size))
    # Calculate pairwise distance between instances
    for i in range(size):
        for j in range(i, size):
            dist = _compute_distance(sampled_instances[i], sampled_instances[j], mean_vector)
            similarity_distance[i, j] = dist
            similarity_distance[j, i] = dist
    # Select top n_neighbors
    ordered_X = np.sort(similarity_distance)[:, 1: n_neighbors + 1]
    # Select the value of the distance that captures XX% of all distances (percentile)
    return np.percentile(ordered_X.flatten(), percentile)
```

```python
def find_neighbors(selection, dataset, model, mode, n_neighbors=30):
    """
    For each instance, select neighbors based on 3 criteria:
    1. First pick top N closest neighbors (L1 Norm + st. dev normalization)
    2. Filter neighbors whose model output is too different from instance (see condition below)
    3. Filter neighbors whose distance is too big compared to a certain threshold
    Parameters
    ----------
    selection : list
        Indices of rows to be displayed on the stability plot
    dataset : DataFrame
        Entire dataset used to identify neighbors
    model : model object
        ML model
    mode : str
        "classification" or "regression"
    n_neighbors : int, optional
        Top N neighbors initially allowed, by default 10
    Returns
    -------
    all_neighbors : list of 2D arrays
        Wrap all instances with corresponding neighbors in a list with length (#instances).
        Each array has shape (#neighbors, #features) where #neighbors includes the instance itself.
    """
    instances = dataset.loc[selection].values

    all_neighbors = np.empty((0, instances.shape[1] + 1), float)
    """Filter 1 : Pick top N closest neighbors"""
    for instance in instances:
        c = _compute_similarities(instance, dataset.values)
        # Pick indices of the closest neighbors (and include instance itself)
        neighbors_indices = np.argsort(c)[: n_neighbors + 1]
        # Return instance with its neighbors
        neighbors = dataset.values[neighbors_indices]
        # Add distance column
        neighbors = np.append(neighbors, c[neighbors_indices].reshape(n_neighbors + 1, 1), axis=1)
        all_neighbors = np.append(all_neighbors, neighbors, axis=0)

    # Calculate predictions for all instances and corresponding neighbors
    if mode == "regression":
        # For XGB it is necessary to add columns in df, otherwise columns mismatch
        predictions = model.predict(pd.DataFrame(all_neighbors[:, :-1], columns=dataset.columns))
    elif mode == "classification":
        predictions = model.predict_proba(pd.DataFrame(all_neighbors[:, :-1], columns=dataset.columns))[:, 1]

    # Add prediction column
    all_neighbors = np.append(all_neighbors, predictions.reshape(all_neighbors.shape[0], 1), axis=1)
    # Split back into original chunks (1 chunck = instance + neighbors)
    all_neighbors = np.split(all_neighbors, instances.shape[0])
```

```
    """Filter 2 : neighbors with similar blackbox output"""
    # Remove points if prediction is far away from instance prediction
    if mode == "regression":
        # Trick : use enumerate to allow the modifcation directly on the iterator
        for i, neighbors in enumerate(all_neighbors):
            all_neighbors[i] = neighbors[abs(neighbors[:, -1] - neighbors[0, -1]) < 0.1 * abs(neighbors[0, -1])]
    elif mode == "classification":
        for i, neighbors in enumerate(all_neighbors):
            all_neighbors[i] = neighbors[abs(neighbors[:, -1] - neighbors[0, -1]) < 0.1]

    """Filter 3 : neighbors below a distance threshold"""
    # Remove points if distance is bigger than radius
    radius = _get_radius(dataset.values, n_neighbors)

    for i, neighbors in enumerate(all_neighbors):
        # -2 indicates the distance column
        all_neighbors[i] = neighbors[neighbors[:, -2] < radius]
    return all_neighbors

def shap_neighbors(instance, x_encoded, contributions, mode):
    """
    For an instance and corresponding neighbors, calculate various
    metrics (described below) that are useful to evaluate local stability
    Parameters
    ----------
    instance : 2D array
        Instance + neighbours with corresponding features
    x_encoded : DataFrame
        Entire dataset used to identify neighbors
    contributions : DataFrame
        Calculated contribution values for the dataset
    Returns
    -------
    norm_shap_values : array
        Normalized SHAP values (with corresponding sign) of instance and its neighbors
    average_diff : array
        Variability (stddev / mean) of normalized SHAP values (using L1) across neighbors for each feature
    norm_abs_shap_values[0, :] : array
        Normalized absolute SHAP value of the instance
    """
    # Extract SHAP values for instance and neighbors
    # :-2 indicates that two columns are disregarded : distance to instance and model output
    ind = pd.merge(x_encoded.reset_index(), pd.DataFrame(instance[:, :-2], columns=x_encoded.columns), how='inner
        .set_index(x_encoded.index.name if x_encoded.index.name is not None else 'index').index
    # If classification, select contrbutions of one class only
    if mode == "classification" and len(contributions) == 2:
        contributions = contributions[1]
    shap_values = contributions.loc[ind]
    # For neighbors comparison, the sign of SHAP values is taken into account
    norm_shap_values = normalize(shap_values, axis=1, norm="l1")
```

```
    # But not for the average impact of the features across the dataset
    norm_abs_shap_values = normalize(np.abs(shap_values), axis=1, norm="l1")
    # Compute the average difference between the instance and its neighbors
    # And replace NaN with 0
    average_diff = np.divide(norm_shap_values.std(axis=0), norm_abs_shap_values.mean(axis=0),
                             out=np.zeros(norm_abs_shap_values.shape[1]),
                             where=norm_abs_shap_values.mean(axis=0) != 0)

    return norm_shap_values, average_diff, norm_abs_shap_values[0, :]


def get_distance(selection, contributions, mode, nb_features):
    """
    Determine how close we get to the output with all features by using only a subset of them
    Parameters
    ----------
    selection : list
        Indices of rows to be displayed on the stability plot
    contributions : DataFrame
        Calculated contribution values for the dataset
    mode : str
        "classification" or "regression"
    nb_features : int, optional
        Number of features used, by default 5
    Returns
    -------
    distance : array
        List of distances for each instance by using top selected features (ex: np.array([0.12, 0.16...])).

        * For regression:

            * normalized distance between the output of current model and output of full model
        * For classifciation:
            * distance between probability outputs (absolute value)
    """
    if mode == "classification" and len(contributions) == 2:
        contributions = contributions[1]
    assert nb_features <= contributions.shape[1]

    contributions = contributions.loc[selection].values
    top_features = np.array([sorted(row, key=abs, reverse=True) for row in contributions])[:, :nb_features]
    output_top_features = np.sum(top_features[:, :], axis=1)
    output_all_features = np.sum(contributions[:, :], axis=1)

    if mode == "regression":
        distance = abs(output_top_features - output_all_features) / abs(output_all_features)
    elif mode == "classification":
        distance = abs(output_top_features - output_all_features)
    return distance
```

```
def compute_features_stability (case, x, selection, contributions):
        """
        For a selection of instances, compute features stability metrics used in
        methods `local_neighbors_plot` and `local_stability_plot`.
        - If selection is a single instance, the method returns the (normalized) contribution values
        of instance and corresponding neighbors.
        - If selection represents multiple instances, the method returns the average (normalized) contribution val
        of instances and neighbors (=amplitude), as well as the variability of those values in the neighborhood (=
        Parameters
        ----------
        selection: list
            Indices of rows to be displayed on the stability plot
        Returns
        -------
        Dictionary
            Values that will be displayed on the graph. Keys are "amplitude", "variability" and "norm_shap"
        """
        #if (case == "classification") and (len(self._classes) > 2):
        #    raise AssertionError("Multi-class classification is not supported")
        x_encoded=x
        x_init=x
        all_neighbors = find_neighbors(selection, x_encoded, model, case)

        # Check if entry is a single instance or not
        if len(selection) == 1:
            # Compute explanations for instance and neighbors
            norm_shap, _, _ = shap_neighbors(all_neighbors[0], x_encoded, contributions, case)
            local_neighbors = {"norm_shap": norm_shap}
            return local_neighbors
        else:
            numb_expl = len(selection)
            amplitude = np.zeros((numb_expl, x_init.shape[1]))
            variability = np.zeros((numb_expl, x_init.shape[1]))
            # For each instance (+ neighbors), compute explanation
            for i in range(numb_expl):
                (_, variability[i, :], amplitude[i, :],) = shap_neighbors(all_neighbors[i], x_encoded, contributio
            features_stability = {"variability": variability, "amplitude": amplitude}
            return features_stability
```

In [ ]:

```
from matplotlib import pyplot as plt
```

One instance

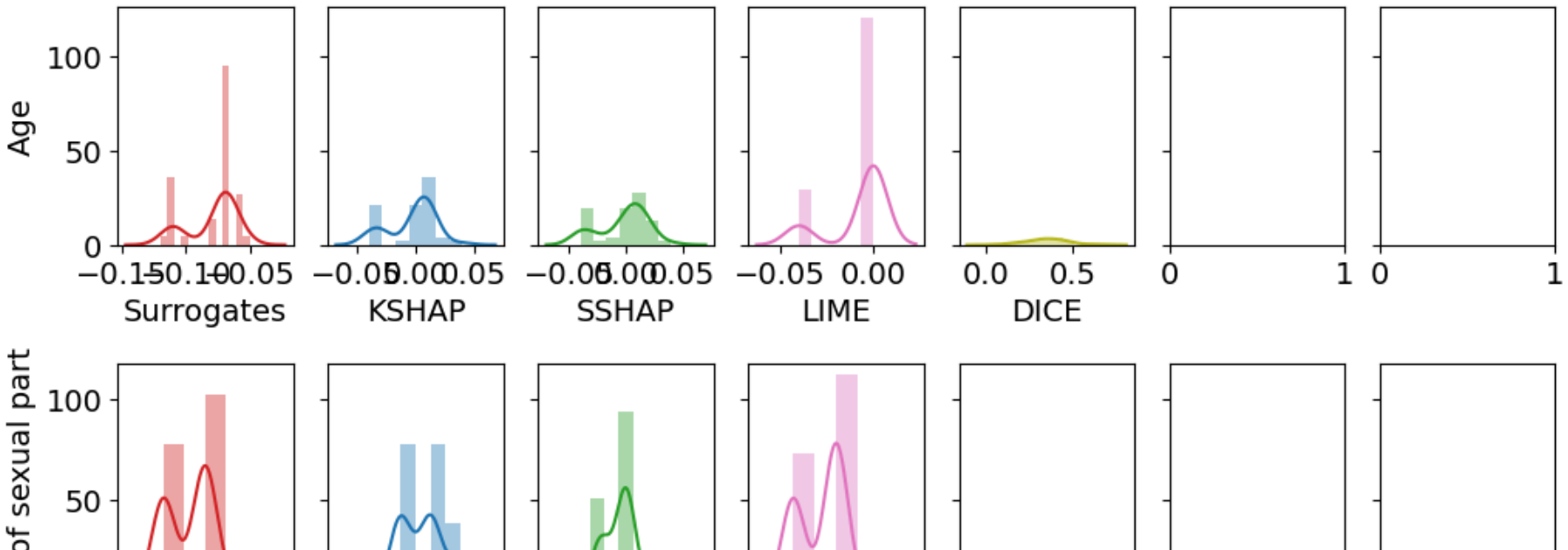
In [ ]:

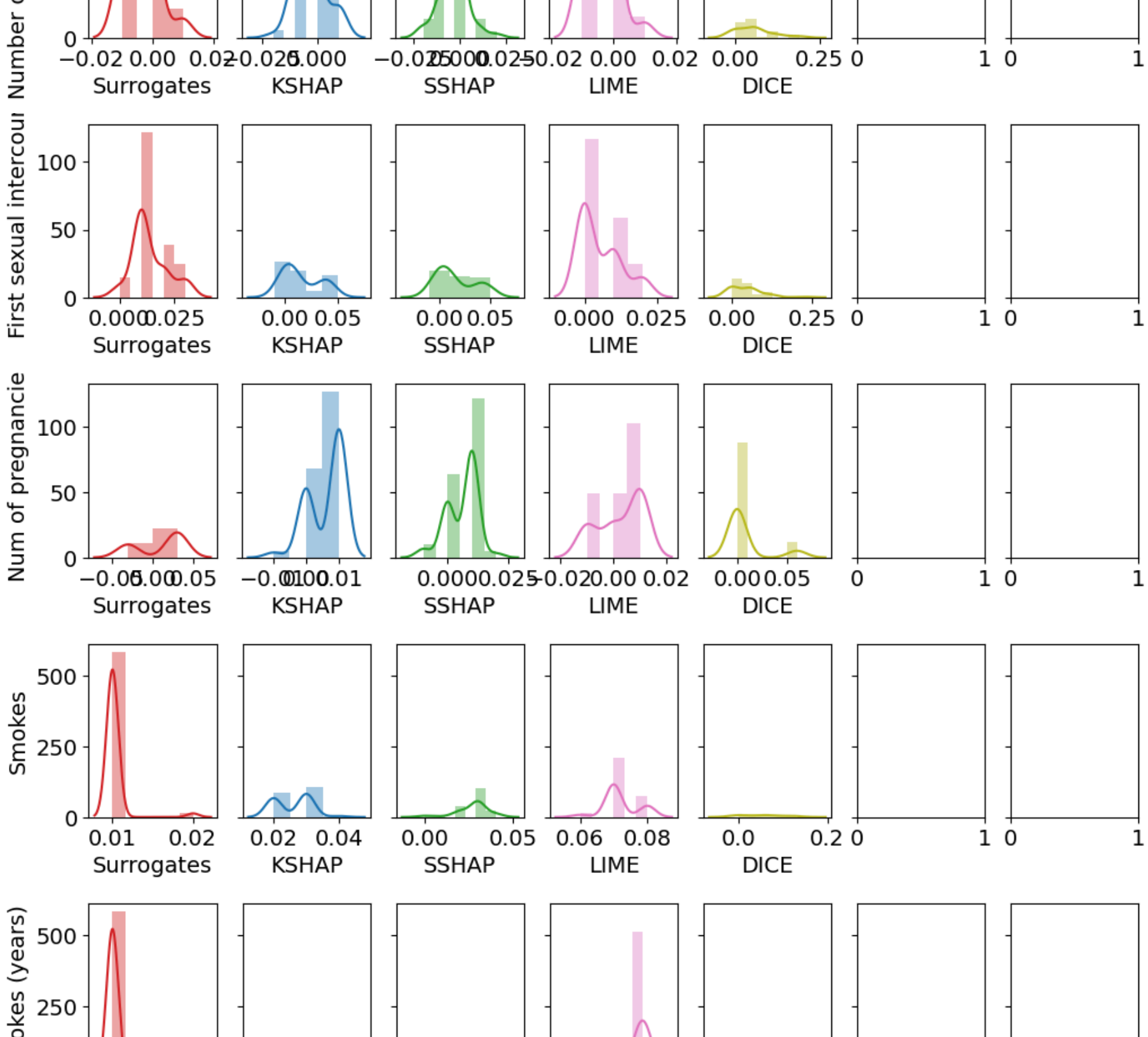
```
features=list(X_test.columns)
```

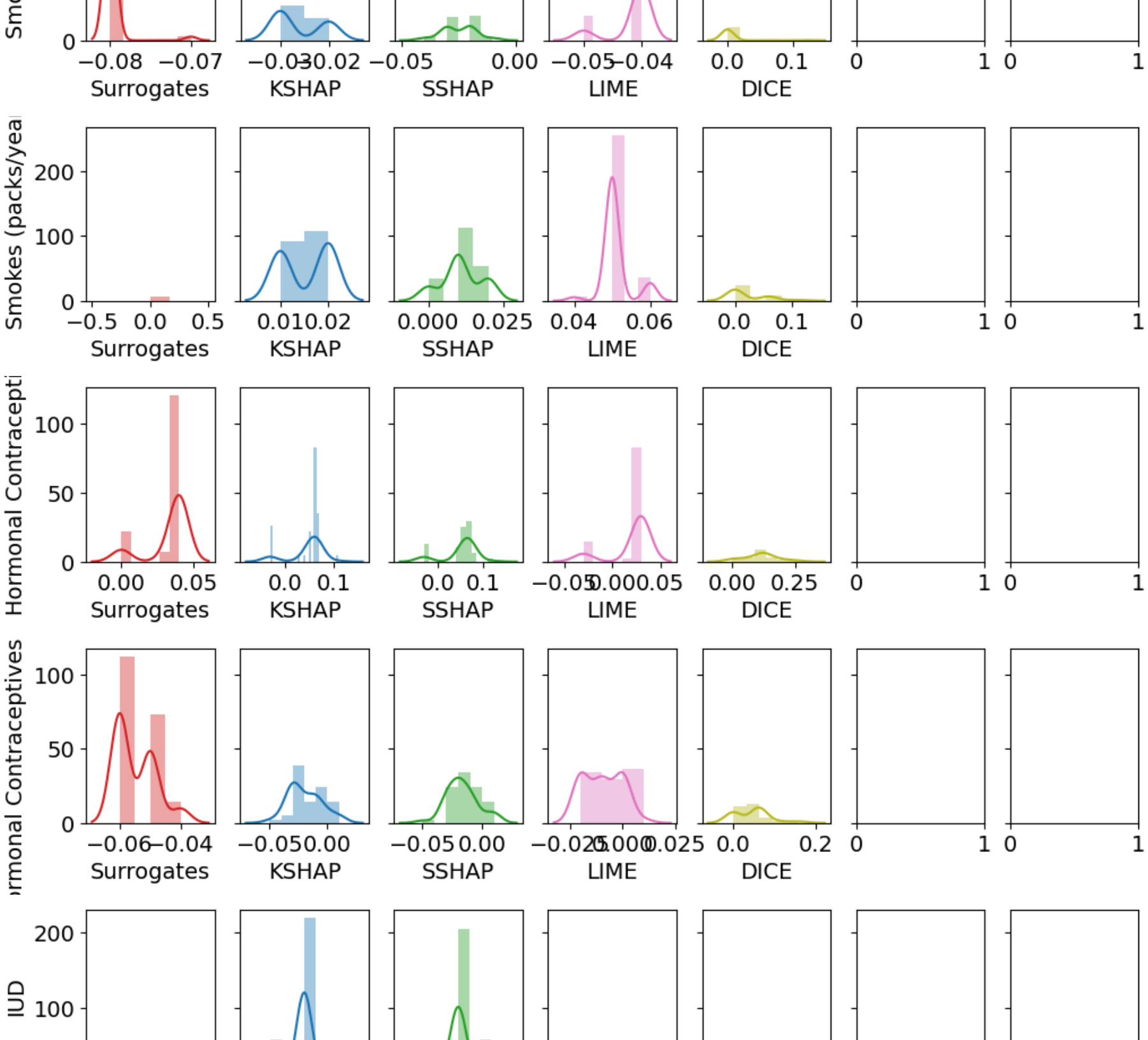
In [ ]:
```
#X_test.reset_index(inplace=True)
#X_test.drop('index', inplace=True, axis=1)
```

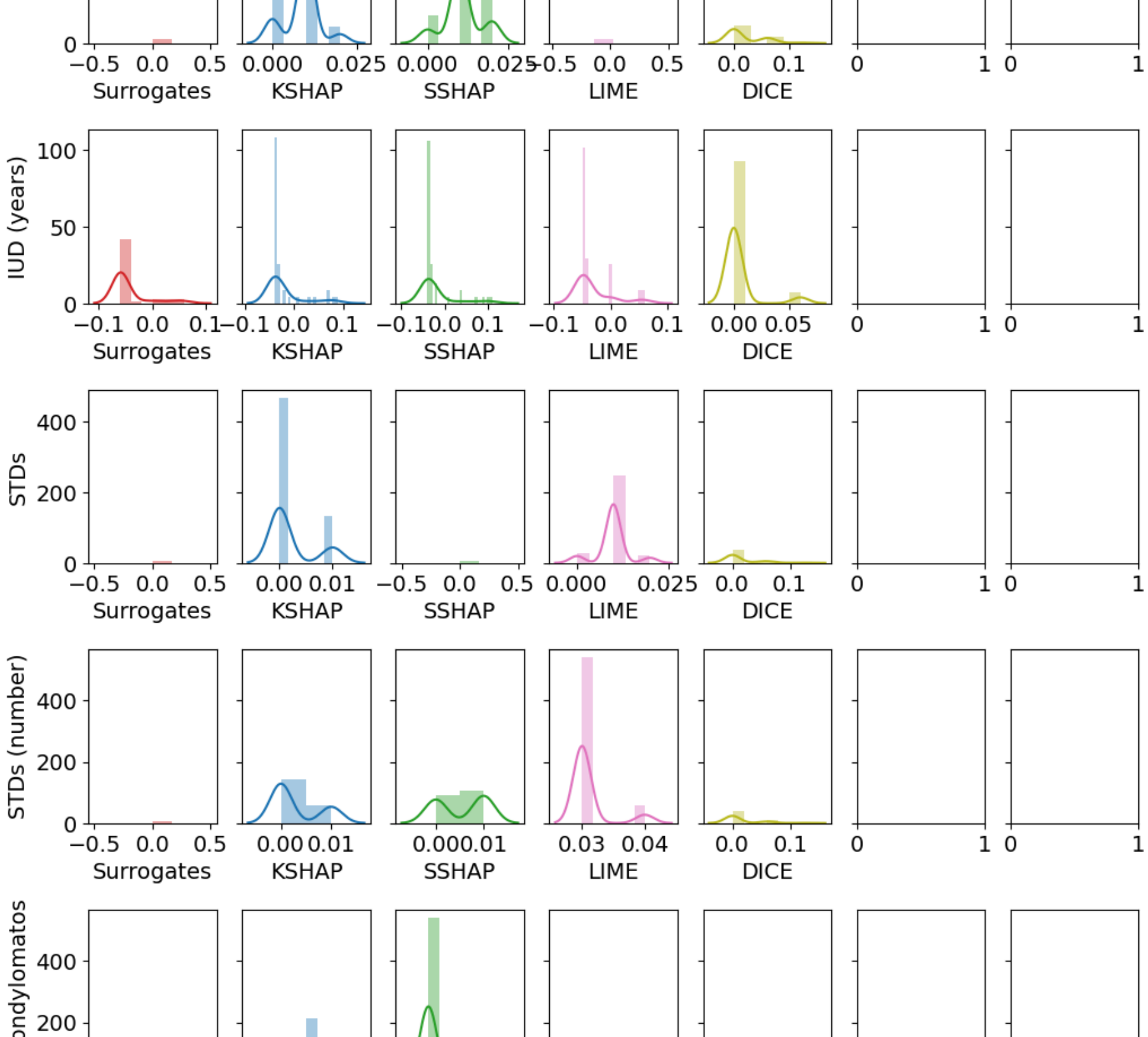
In [ ]:
```
#compute_features_compacity(case="classification", contributions=weight, selection=list(range(0, len(x_test))), di
frames=[]
for weight in weights:
  #fs= compute_features_stability (case="classification", x=X_test, selection=list(range(0, len(X_test))), contrib
  fs= compute_features_stability (case="classification", x=X_test, selection=[instance], contributions=weight)
  frames.append(fs)
```

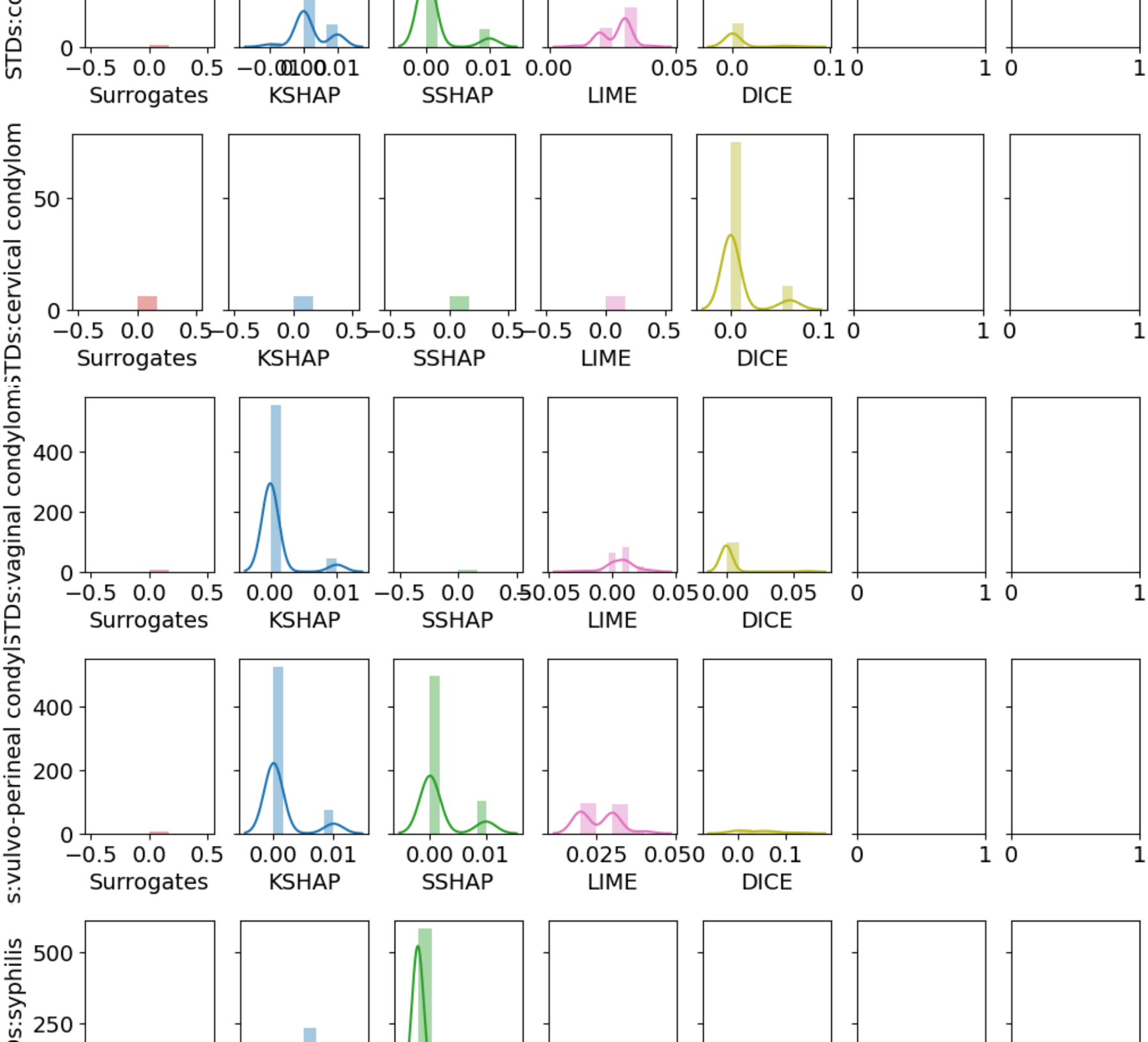
In [ ]:
```
colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orange',  'tab:gray']

for j in range(len(features)):
  fig, axes = plt.subplots(1, 7, figsize=(12, 2), sharey=True, dpi=100)
  t=0

  for fg, fs in enumerate(frames):
    am=[]

    for i in range(len(fs['norm_shap'])):
      am.append(round(fs['norm_shap'][i][j], 2)) # i INSCTANCE j Feature
    sns.distplot(am, ax=axes[fg], color=colors[t], axlabel=methods[t])
    t=t+1
    axes[fg].set_ylabel(features[j])
  plt.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+str(instance)+str(features[j])[:10]+'.png', bbox_
  plt.show()
```

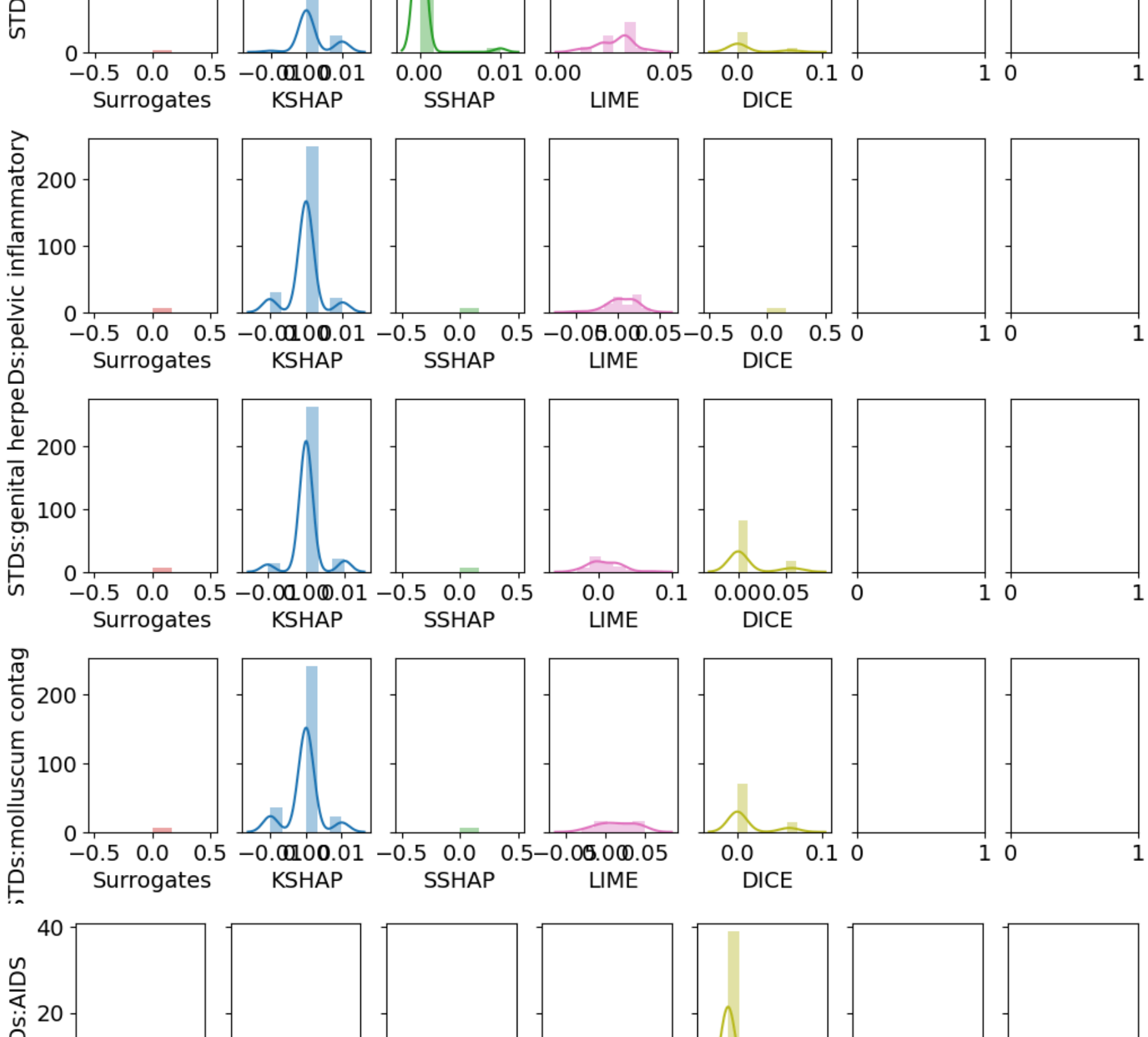


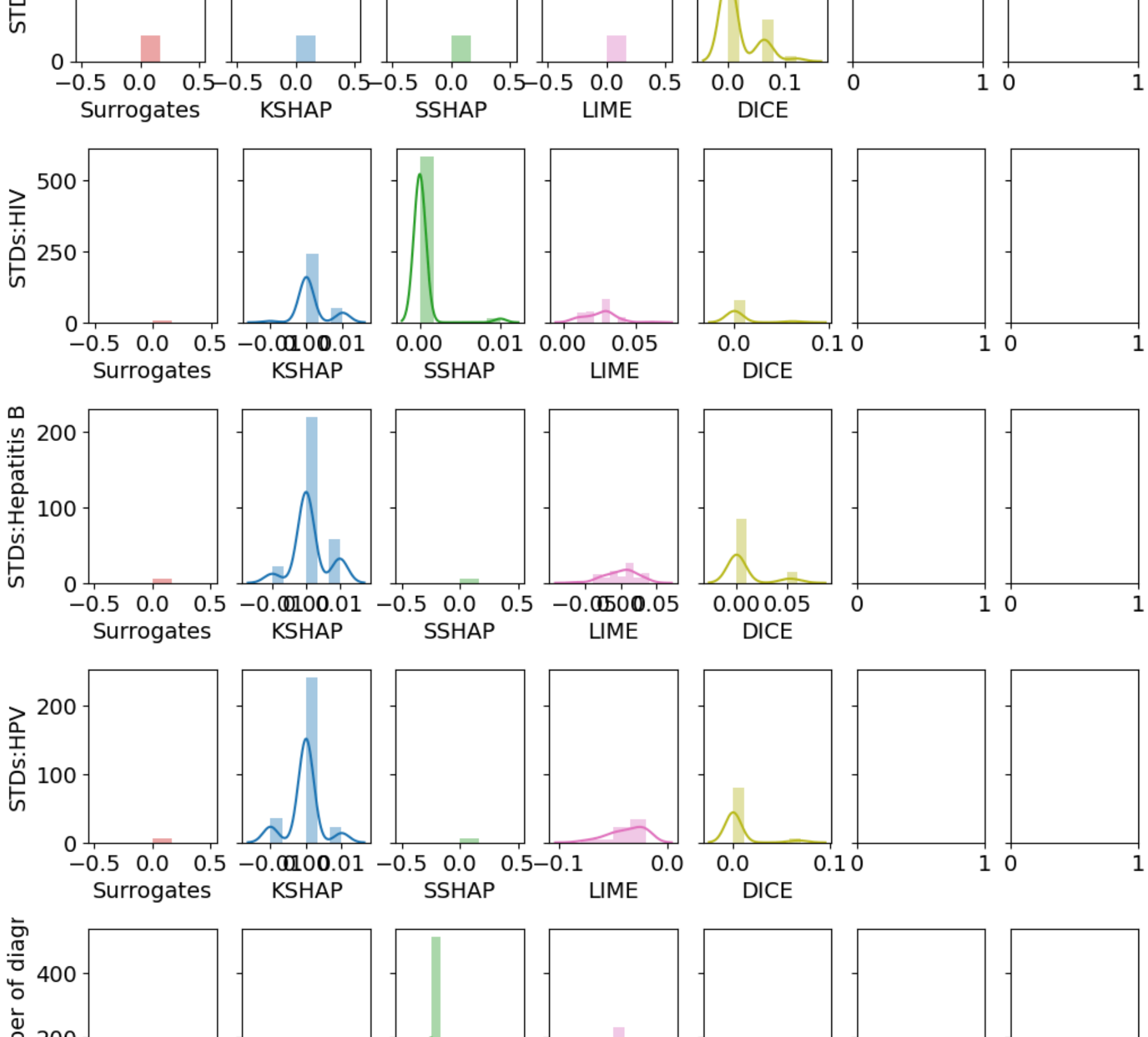

Number
First sexual intercou
Num of pregnancie
Smokes
okes (years)
Surrogates
KSHAP
SSHAP
LIME
DICE

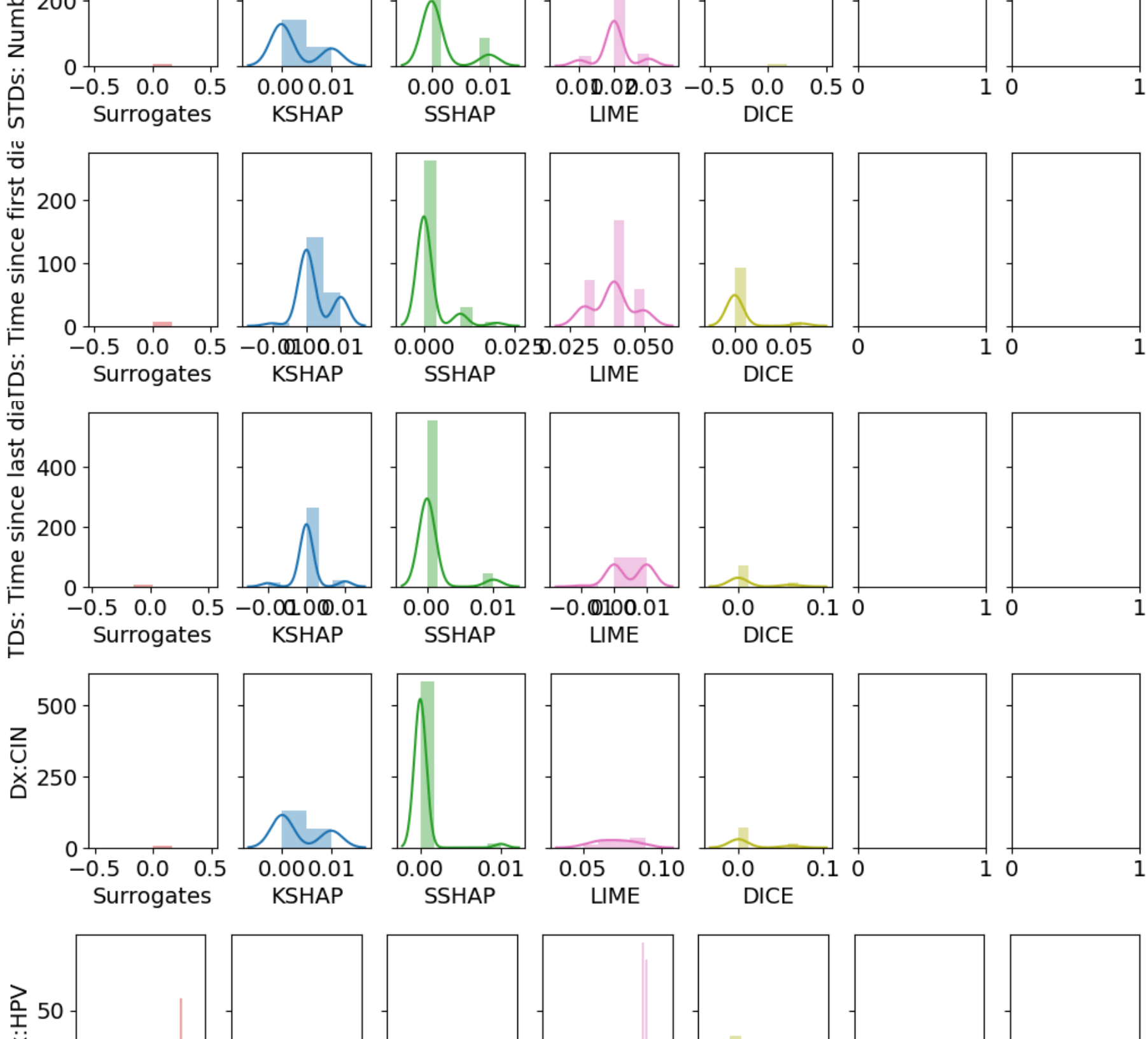

Surrogates
KSHAP
SSHAP
LIME
DICE
Smokes (packs/yea
Hormonal Contracepti
rmonal Contraceptives
IUD

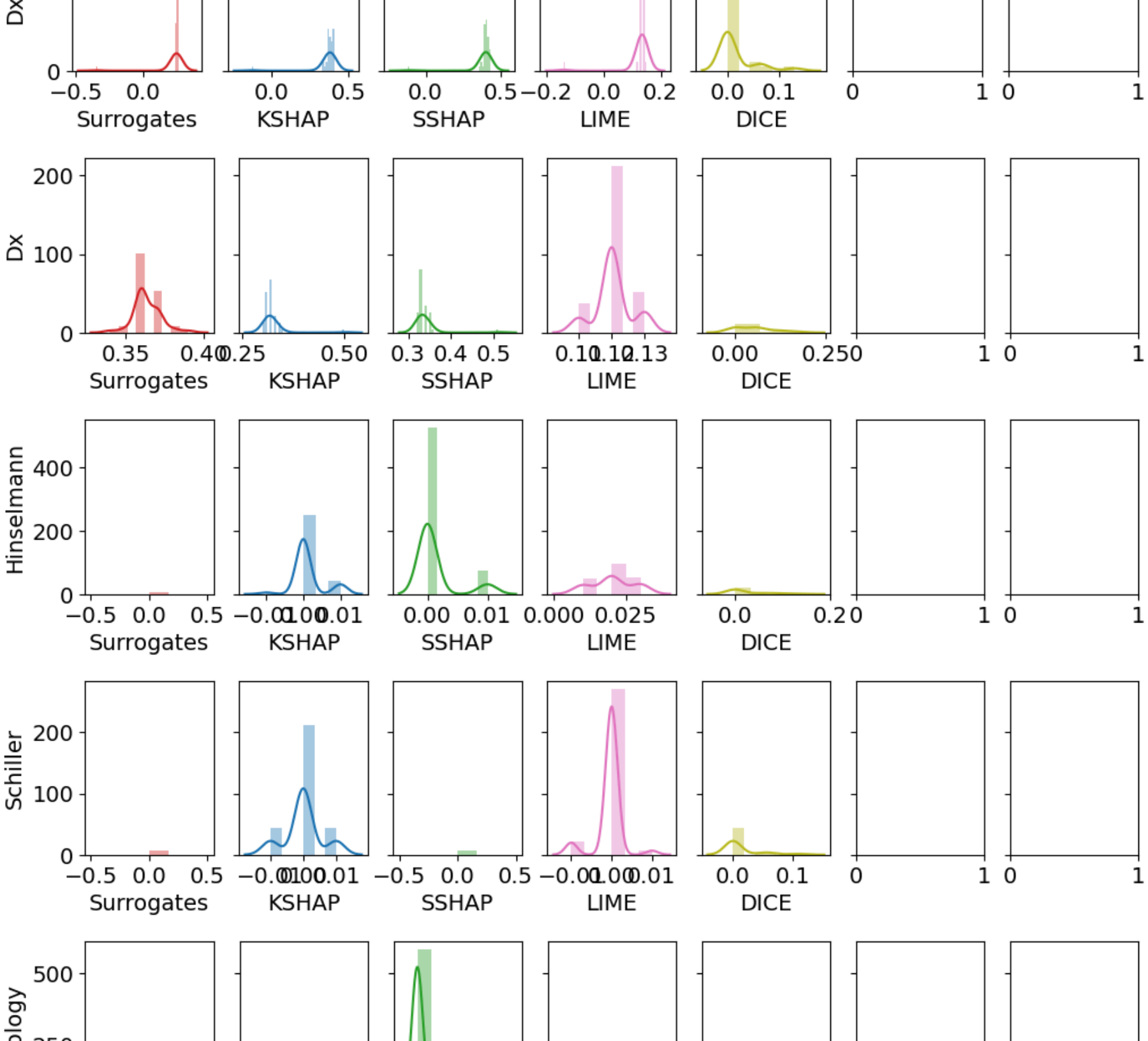

IUD (years)
STDs
STDs (number)
Condylomatos
Surrogates
KSHAP
SSHAP
LIME
DICE

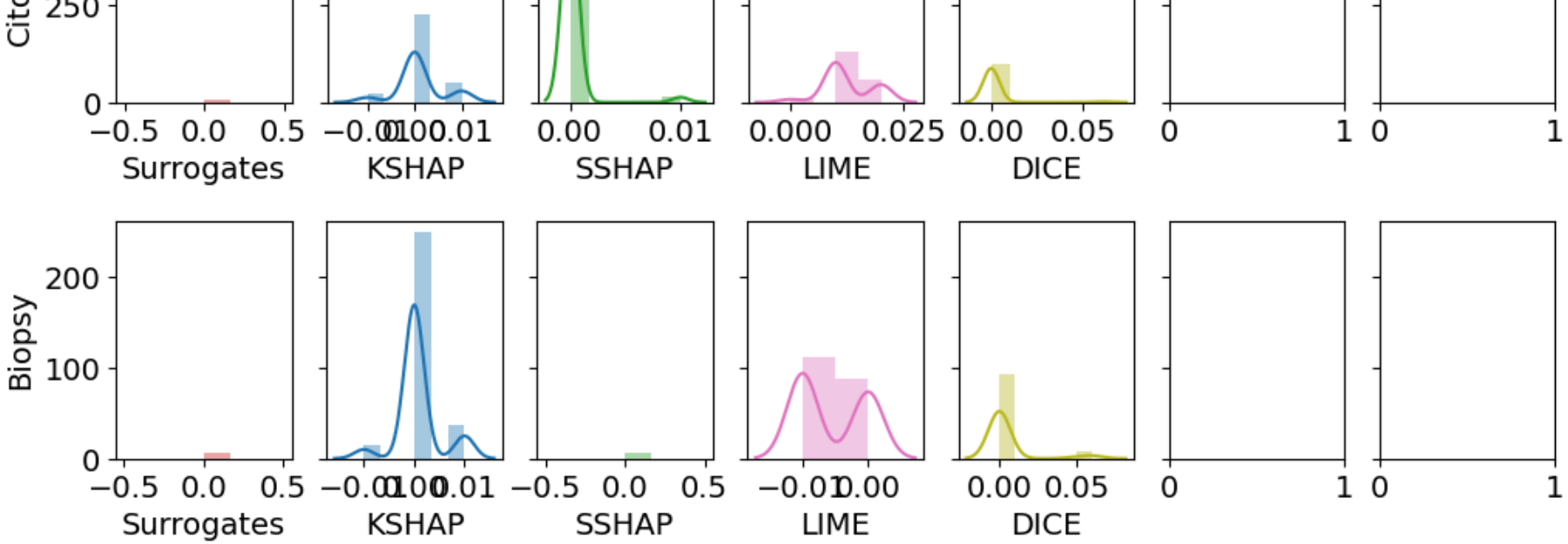

STDs:cervical condylom
STDs:vaginal condylom
s:vulvo-perineal condyl
s:syphilis
Surrogates
KSHAP
SSHAP
LIME
DICE


STDs:pelvic inflammatory
STDs:genital herpe
STDs:molluscum contag
STDs:AIDS
Surrogates
KSHAP
SSHAP
LIME
DICE


STDs:HIV
STDs:Hepatitis B
STDs:HPV
Surrogates
KSHAP
SSHAP
LIME
DICE


STDs: Numb
STDs: Time since first dia
TDs: Time since last dia
Dx:CIN
:HPV
Surrogates
KSHAP
SSHAP
LIME
DICE


Dx
Hinselmann
Schiller
Surrogates
KSHAP
SSHAP
LIME
DICE




Multiple instances

In [ ]:

```
features=X_test.columns
```

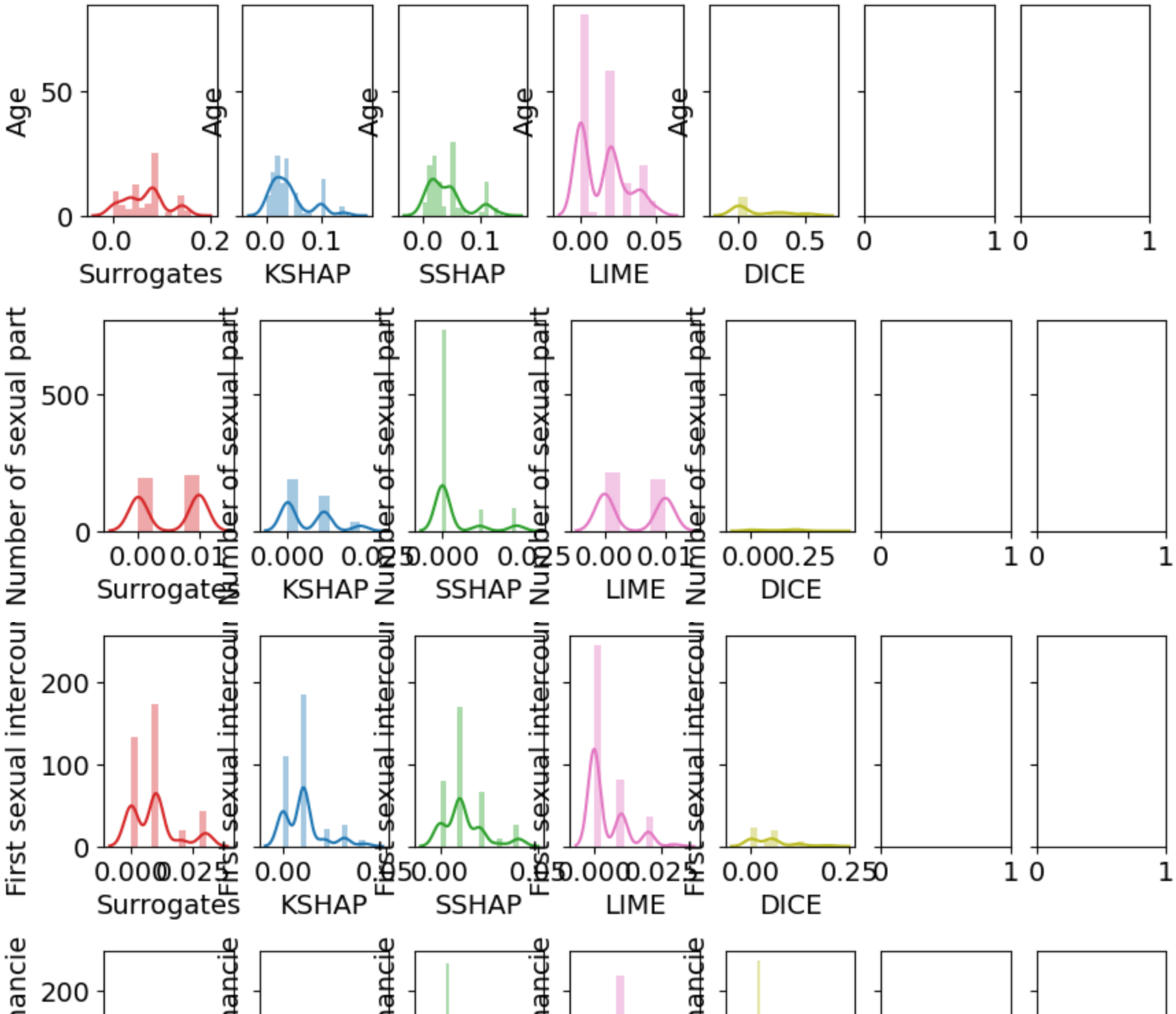
In [ ]:

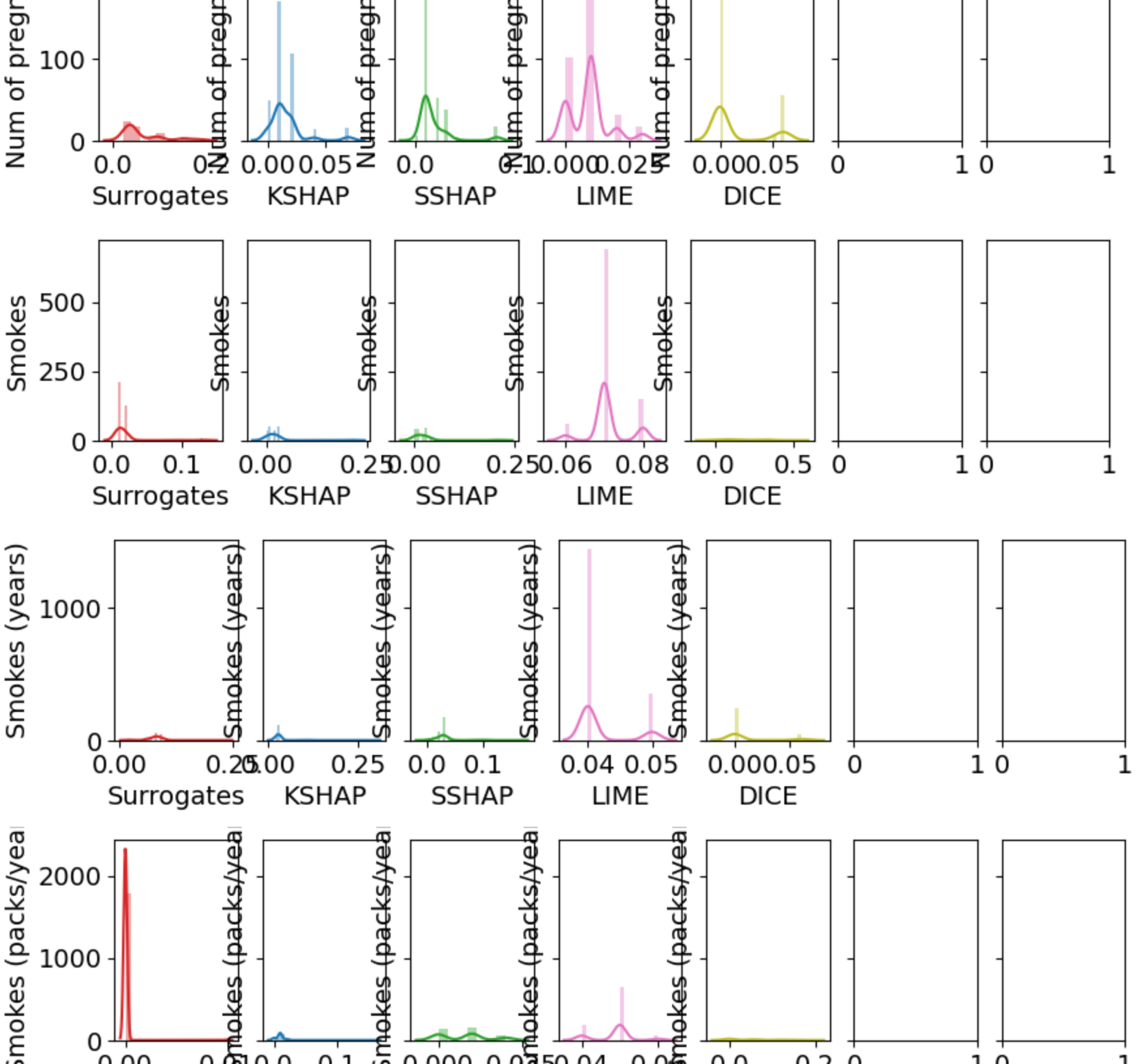
```
#compute_features_compacity(case="classification", contributions=weight, selection=list(range(0, len(x_test))), di
frames=[]
for weight in weights:
  fs= compute_features_stability (case="classification", x=X_test, selection=list(range(0, len(X_test))), contribu
  #fs= compute_features_stability (case="classification", x=X_test, selection=[1,4], contributions=weight)
  frames.append(fs)
```

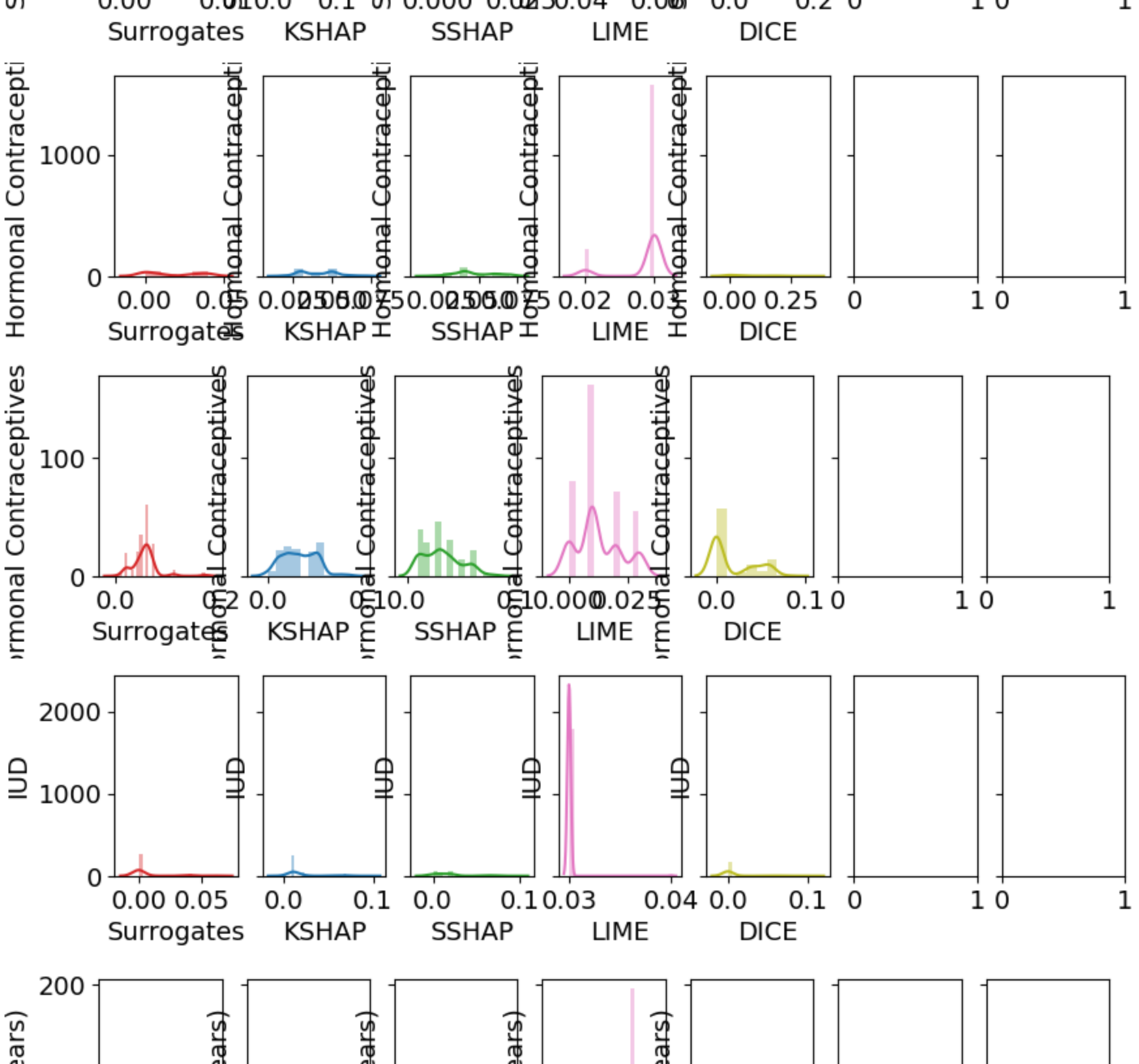
In [ ]:

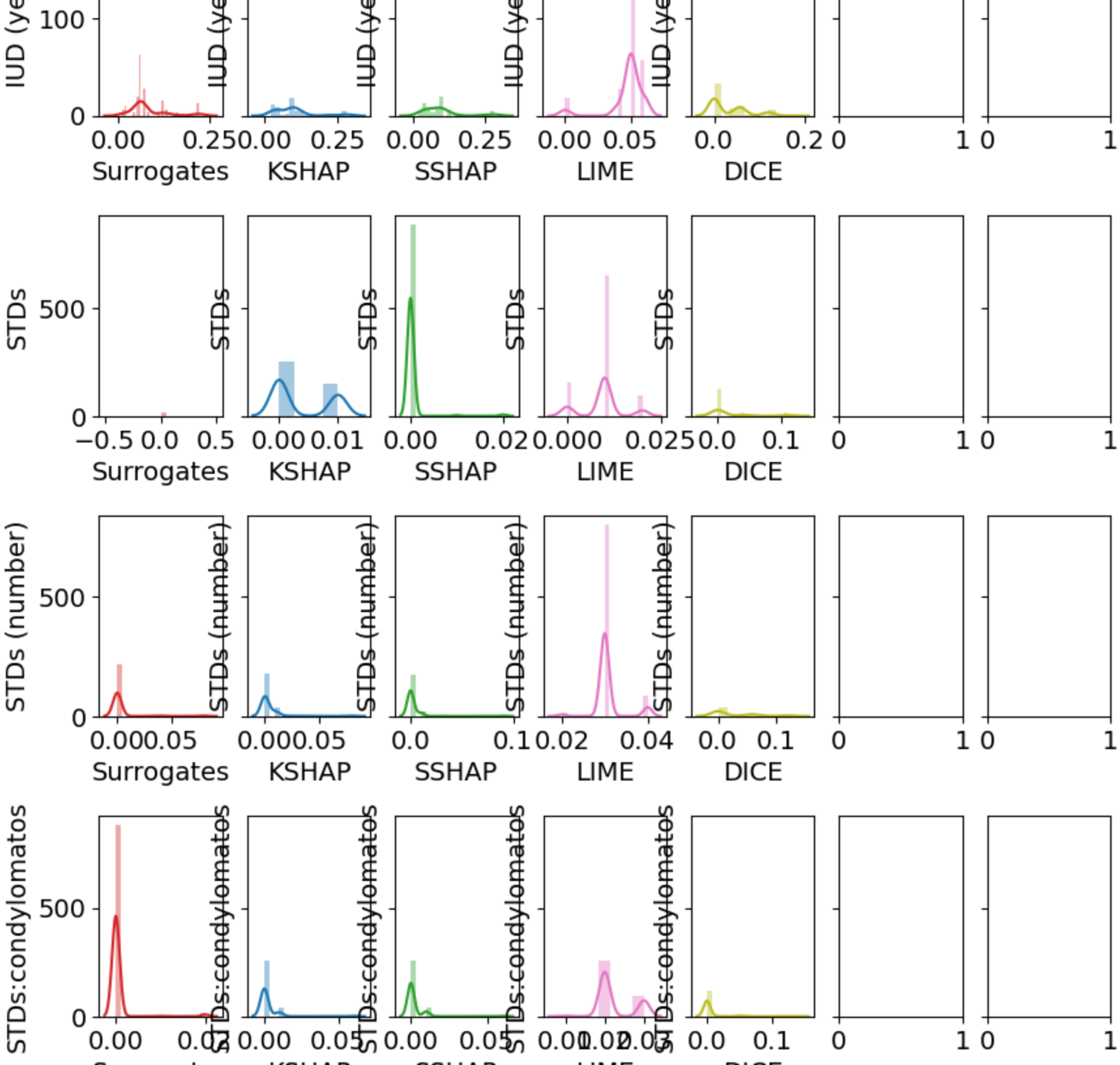
```
colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orange',  'tab:gray']

for j in range(len(features)):
  fig, axes = plt.subplots(1, 7, figsize=(10, 2), sharey=True, dpi=100)
  t=0
  for fg, fs in enumerate(frames):
    vr=[]
    am=[]
    for i in range(len(fs['variability'])):
      vr.append(round(fs['variability'][i][j], 2)) # i INSCTANCE j Feature
      am.append(round(fs['amplitude'][i][j], 2))
    axes[fg].set_ylabel(features[j])
    sns.distplot(am, ax=axes[fg], color=colors[t], axlabel=methods[t])
    t=t+1
    #print('VR', vr)
```

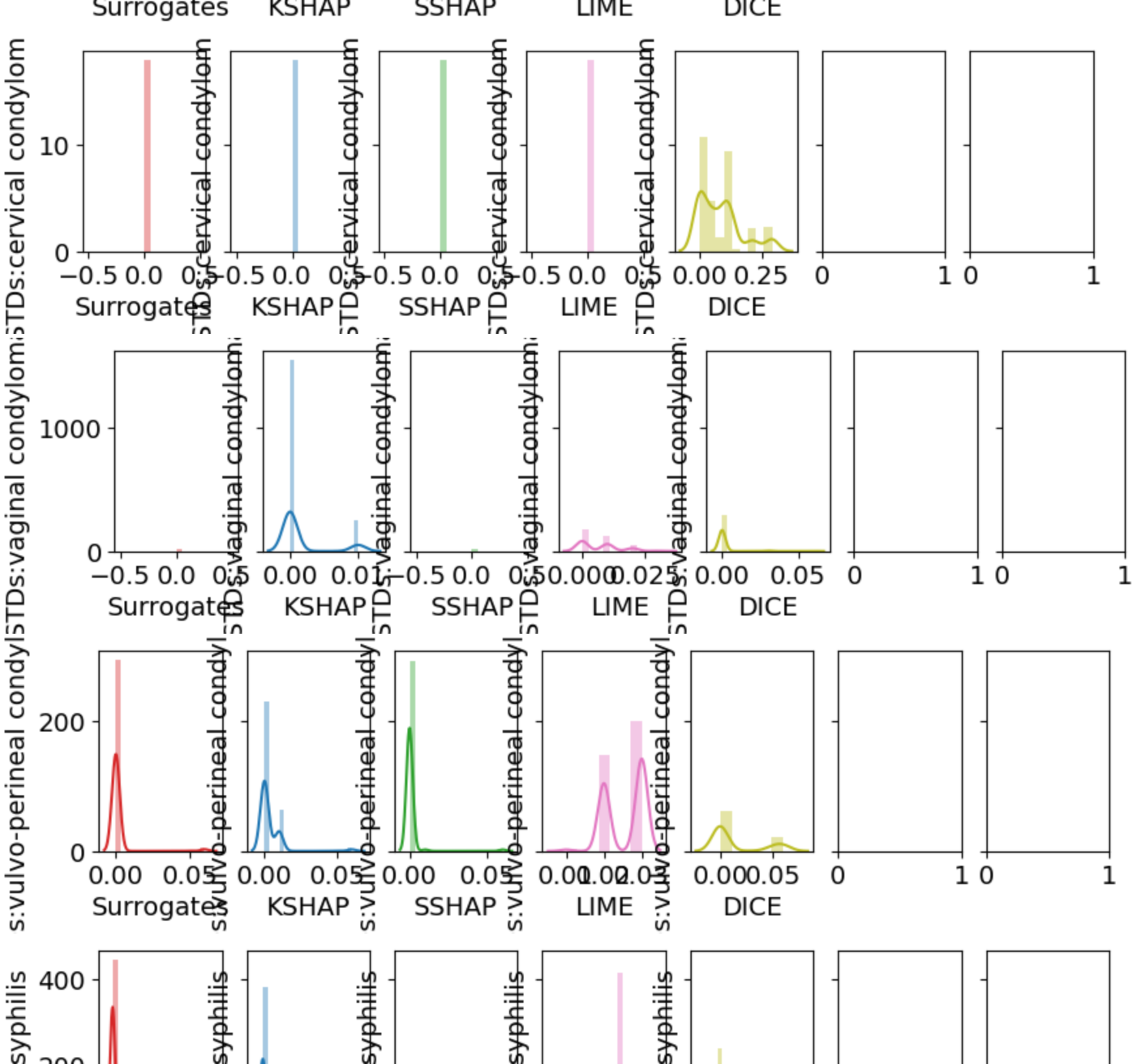
```
plt.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+str(features[j])[:10]+'.png')
plt.show()
```

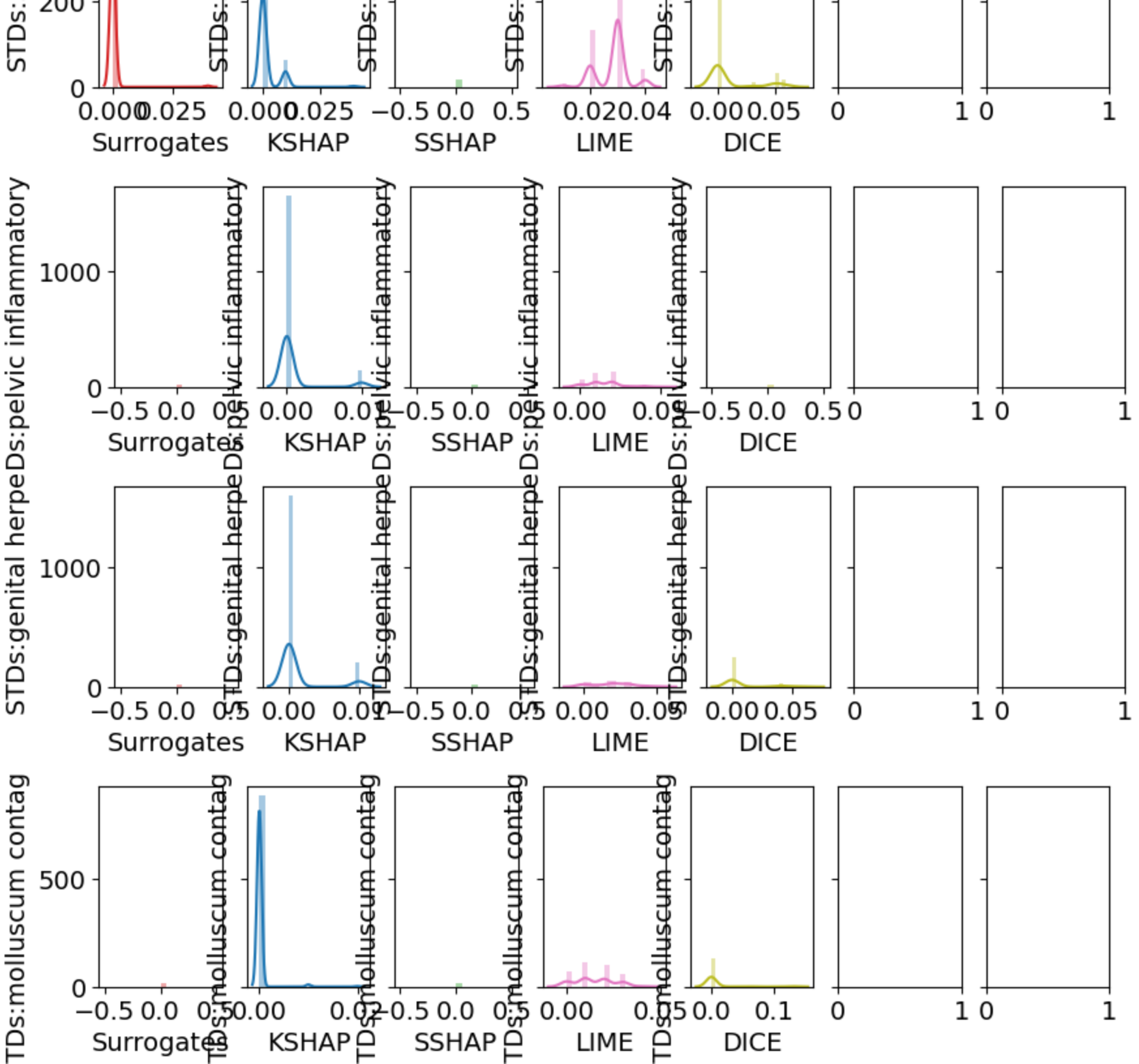


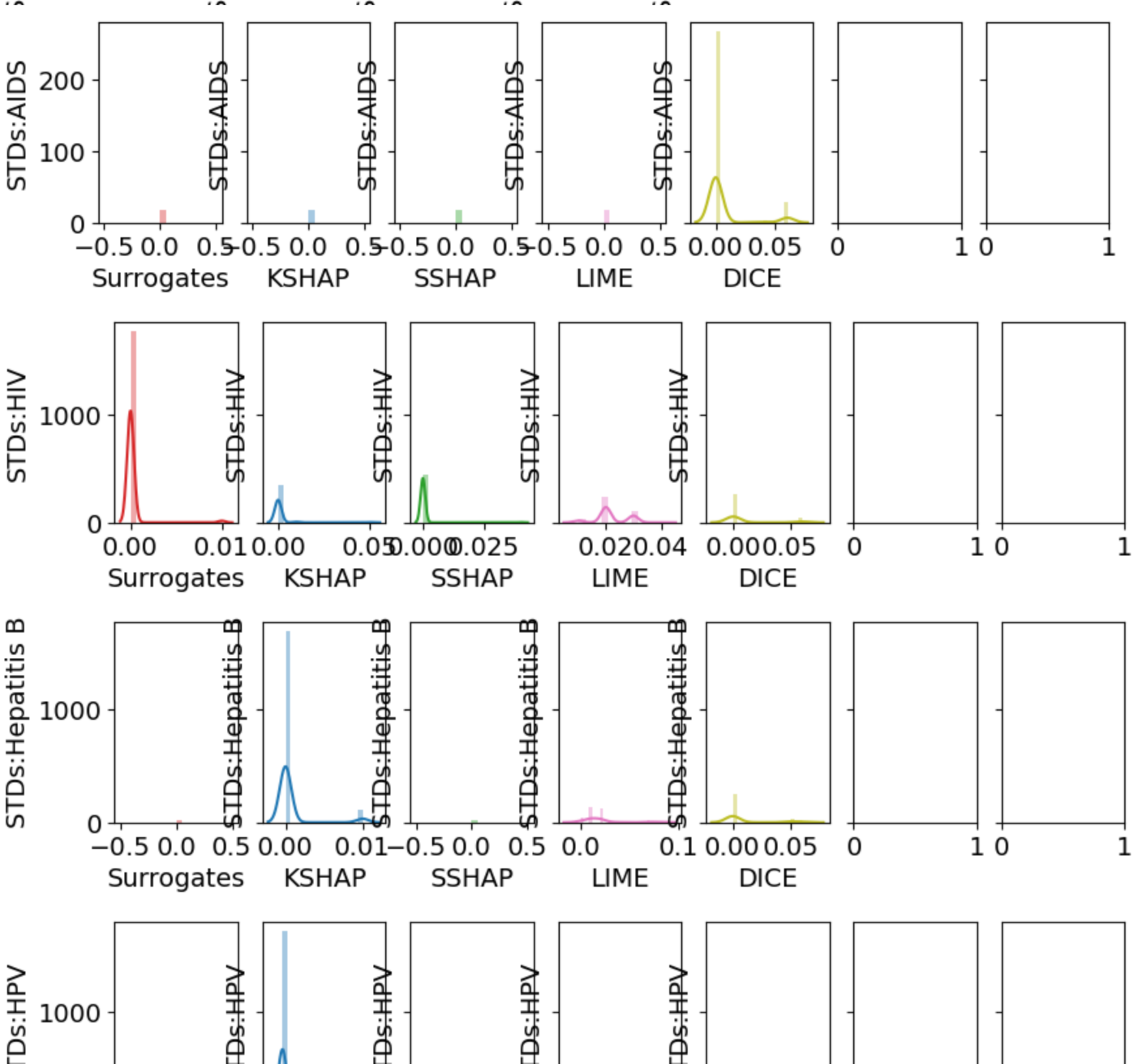

Num of pregn
Smokes
Smokes (years)
Smokes (packs/yea
Surrogates
KSHAP
SSHAP
LIME
DICE

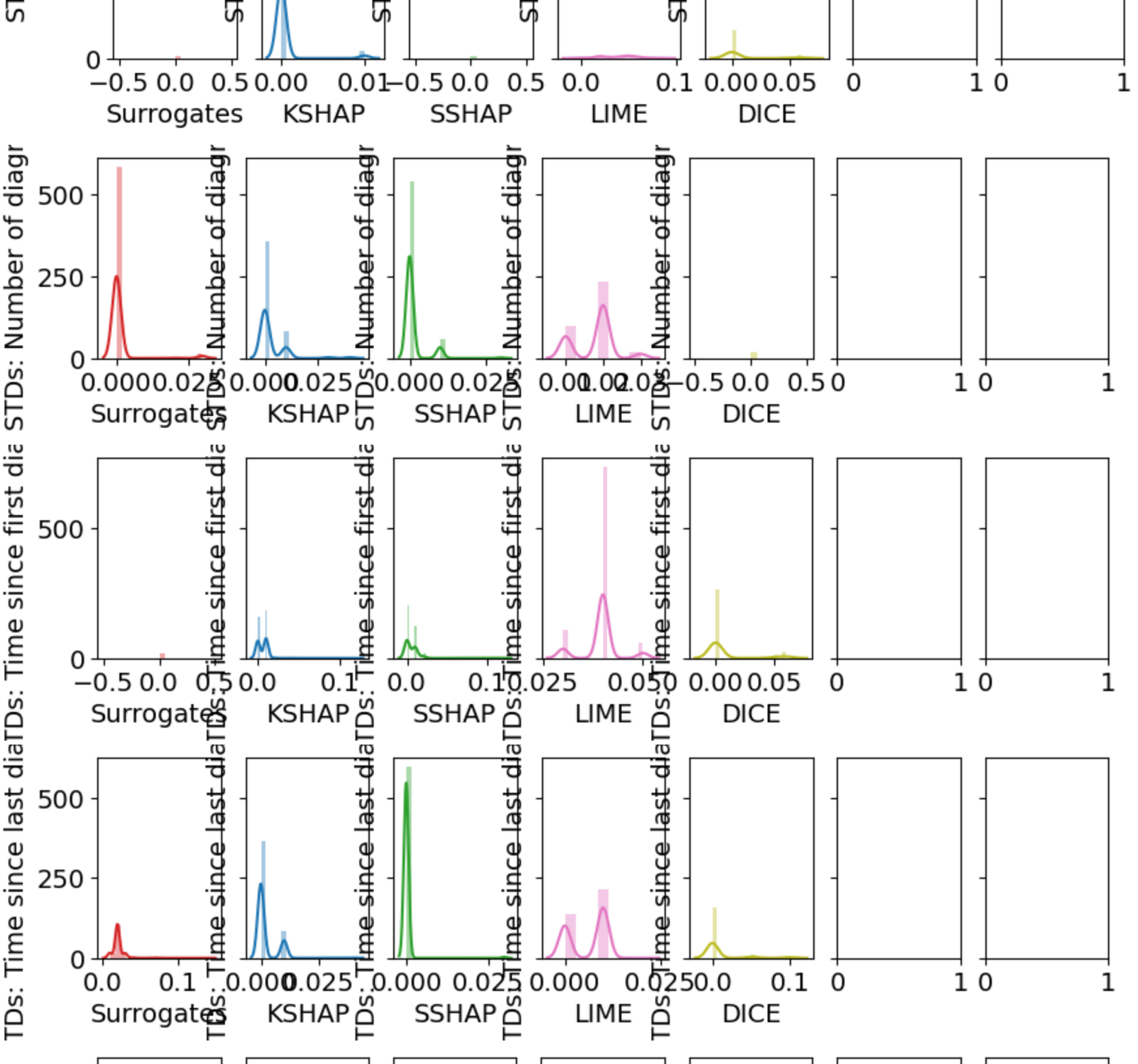

Surrogates
KSHAP
SSHAP
LIME
DICE
Hormonal Contraceptives
IUD
Years)

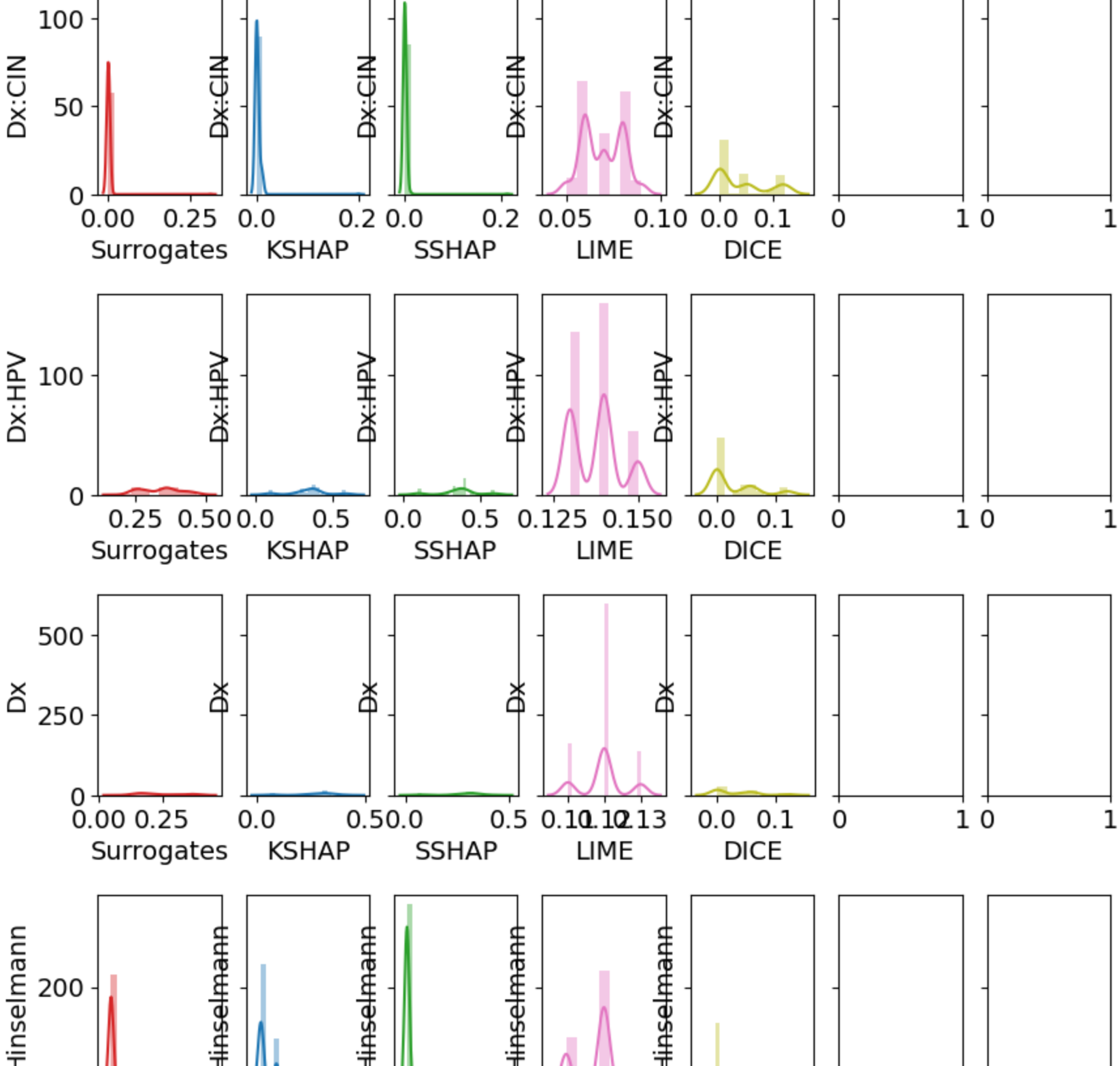

IUD (ye
100
0
0.00 0.25
Surrogates
KSHAP
SSHAP
0.00 0.05
LIME
0.0 0.2
DICE
0 1
STDs
500
−0.5 0.0 0.5
0.00 0.01
0.00 0.02
0.000 0.025
0.0 0.1
STDs (number)
0.000.05
0.0 0.1
0.02 0.04
STDs:condylomatos

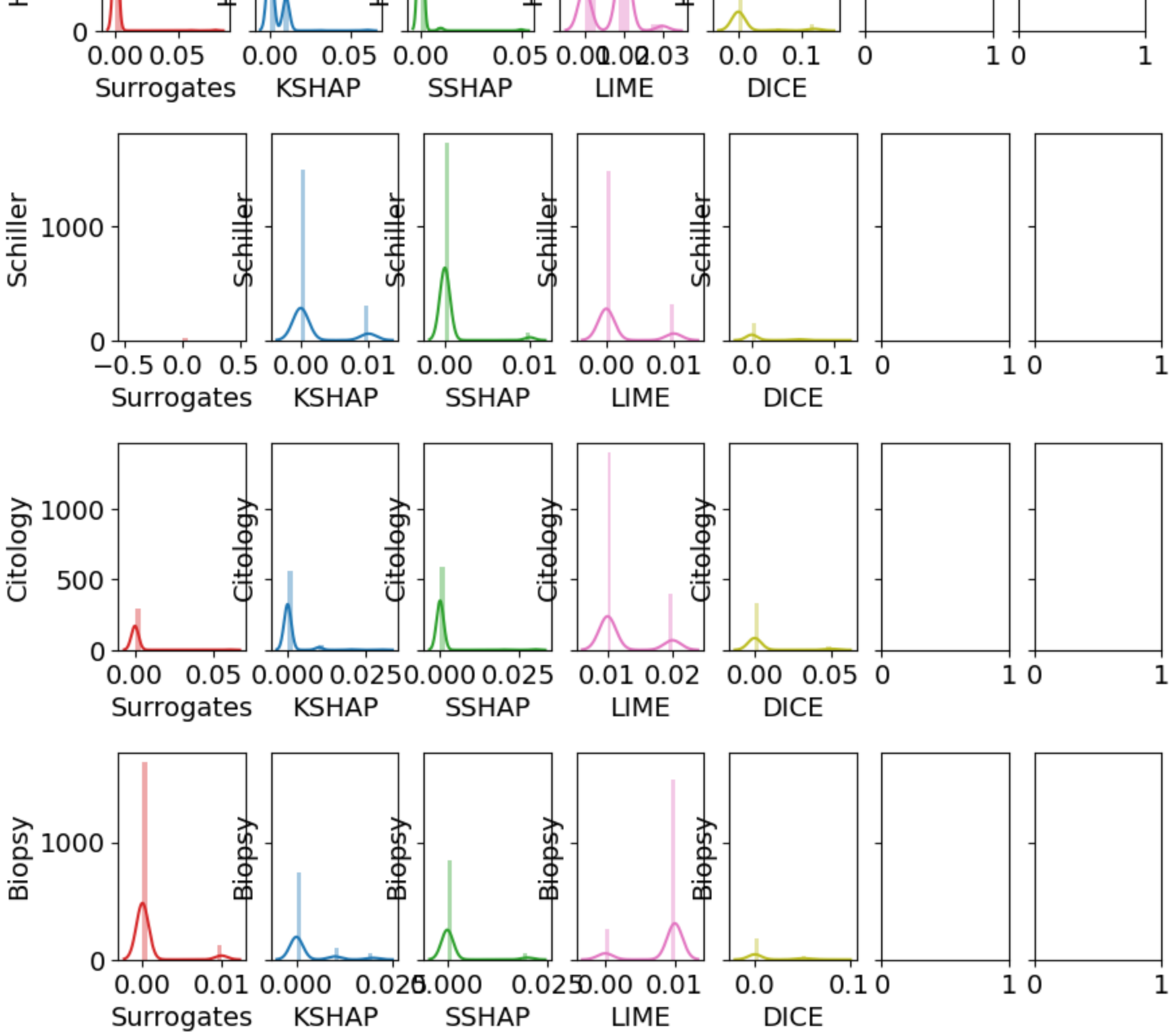

Surrogates
KSHAP
SSHAP
LIME
DICE
STDs:cervical condylom
STDs:vaginal condylom
STDs:vulvo-perineal condyl
STDs:syphilis


STDs:
STDs:pelvic inflammatory
STDs:genital herpe
STDs:molluscum contag
Surrogates
KSHAP
SSHAP
LIME
DICE


STDs:AIDS
STDs:HIV
STDs:Hepatitis B
STDs:HPV
Surrogates
KSHAP
SSHAP
LIME
DICE


Surrogates
KSHAP
SSHAP
LIME
DICE
STDs: Number of diagr
STDs: Time since first dia
STDs: Time since last dia


Dx:CIN
Dx:HPV
Dx
Hinselmann
Surrogates
KSHAP
SSHAP
LIME
DICE


Surrogates
KSHAP
SSHAP
LIME
DICE
Schiller
Citology
Biopsy
0
1000
500

In [ ]:

```
t=0
```

```
for fg, fs in enumerate(frames):
  vr=[]
  am=[]
  for j in range(len(features)):
    vr.append(np.mean(fs['variability'][j])) # i INSCTANCE j Feature
    am.append(np.std(fs['variability'][j]))
  print(methods[t], round(np.mean(vr),2))
  print(methods[t], round(np.std(am),2))
  t+=1
```

```
Surrogates 0.22
Surrogates 0.22
KSHAP 1.05
KSHAP 0.09
SSHAP 1.66
SSHAP 0.2
LIME 0.52
LIME 0.03
DICE 2.04
DICE 0.25
```

Plots

In [ ]:
```
xpl.plot.stability_plot(selection=[0, 1, 3])
```

In [ ]:
```
img.write_image('/content/drive/My Drive/dataXAI/cancer/compactgs.png')
```

In [ ]:
```
fig_image=xpl.plot.stability_plot()
#plt.xlabel("Local Surrogates")
plt.savefig('/content/drive/My Drive/dataXAI/cancer/stabplot.png')
```

```
Computed values from previous call are used
<Figure size 640x480 with 0 Axes>
```

In [ ]:
```
for w in weights:
  xpl = SmartExplainer(model=model)
  xpl.compile(x=X_test,contributions=w)
  xpl.plot.stability_plot()
```

## Feature and Rank disagreement

Tool

In [ ]:

```python
from scipy.stats import spearmanr
import numpy as np
import itertools

def intersection(r1, r2):
    return list(set(r1) & set(r2))

def check_size(r1, r2):
    assert len(r1) == len(r2), 'Both rankings should be the same size'

def feature_agreement(r1, r2):
    """
    Measures the fraction of common features between the
    sets of top-k features of the two rankings.

    From Krishna et al. (2022), The Disagreement Problem in
    Explainable Machine Learning: A Practitioner's Perspective

    Parameters
    ----------------
    r1, r2 : list
        Two feature rankings of identical shape
    """
    check_size(r1, r2)
    k = len(r1)

    return len(intersection(r1, r2)) / k

def rank_agreement(r1, r2):
    """
    Stricter than feature agreement, rank agreement checks
    that the feature order is comparable between the two rankings.

    From Krishna et al. (2022), The Disagreement Problem in
    Explainable Machine Learning: A Practitioner's Perspective

    Parameters
    ----------------
    r1, r2 : list
        Two feature rankings of identical shape
    """
    check_size(r1, r2)
    k = len(r1)

    return np.sum([True if x==y else False for x,y in zip(r1,r2)]) / k

def weak_rank_agreement(r1, r2):
    """
    Check if the rank is approximately close (within one rank).
```

```
    """
    check_size(r1, r2)
    k = len(r1)
    window_size=1

    rank_agree=[]
    for i, v in enumerate(r1):
        if i == 0:
            if v in r2[i:i+window_size+1]:
                rank_agree.append(True)
            else:
                rank_agree.append(False)
        else:
            if v in r2[i-window_size:i+window_size+1]:
                rank_agree.append(True)
            else:
                rank_agree.append(False)

    return np.sum(rank_agree)/k


def rank_correlation(r1, r2):
    return spearmanr(r1, r2)

# def to_rankings(df, instance=1):
#     """
#     Convert feature attributions to a list of top features.
#     """
#     columns = df.columns
#     contrib_features = [c for c in columns ]

#     vals = df[contrib_features].values[instance,:]
#     inds = np.argsort(np.absolute(vals))[::-1]


#     features = [c.replace('_contrib', '') for c in contrib_features]
#     rankings = list(np.array(features)[inds])

#     return inds

def to_rankings(df, instance):
    """
    Convert feature attributions to a list of top features.
    """
    contrib_features = df.columns


    vals = df[contrib_features].values[instance,:]
    rankings = np.argsort(np.absolute(vals))[::-1]
    features = vals[rankings]
```

```
    return rankings


def compute_matrices(weights, instance):
    n_rankings = len(methods)

    feature_agree = np.zeros((n_rankings, n_rankings))
    rank_agree = np.zeros((n_rankings, n_rankings))
    corr = np.zeros((n_rankings, n_rankings))


    for i, j in itertools.product(range(n_rankings), range(n_rankings)):
        r1 = to_rankings(weights[i], instance)[:10]
        r2 = to_rankings(weights[j], instance)[:10]
        feature_agree[i,j] = feature_agreement(r1, r2)
        rank_agree[i,j] = rank_agreement(r1, r2)

    return feature_agree, rank_agree
```

## Plots

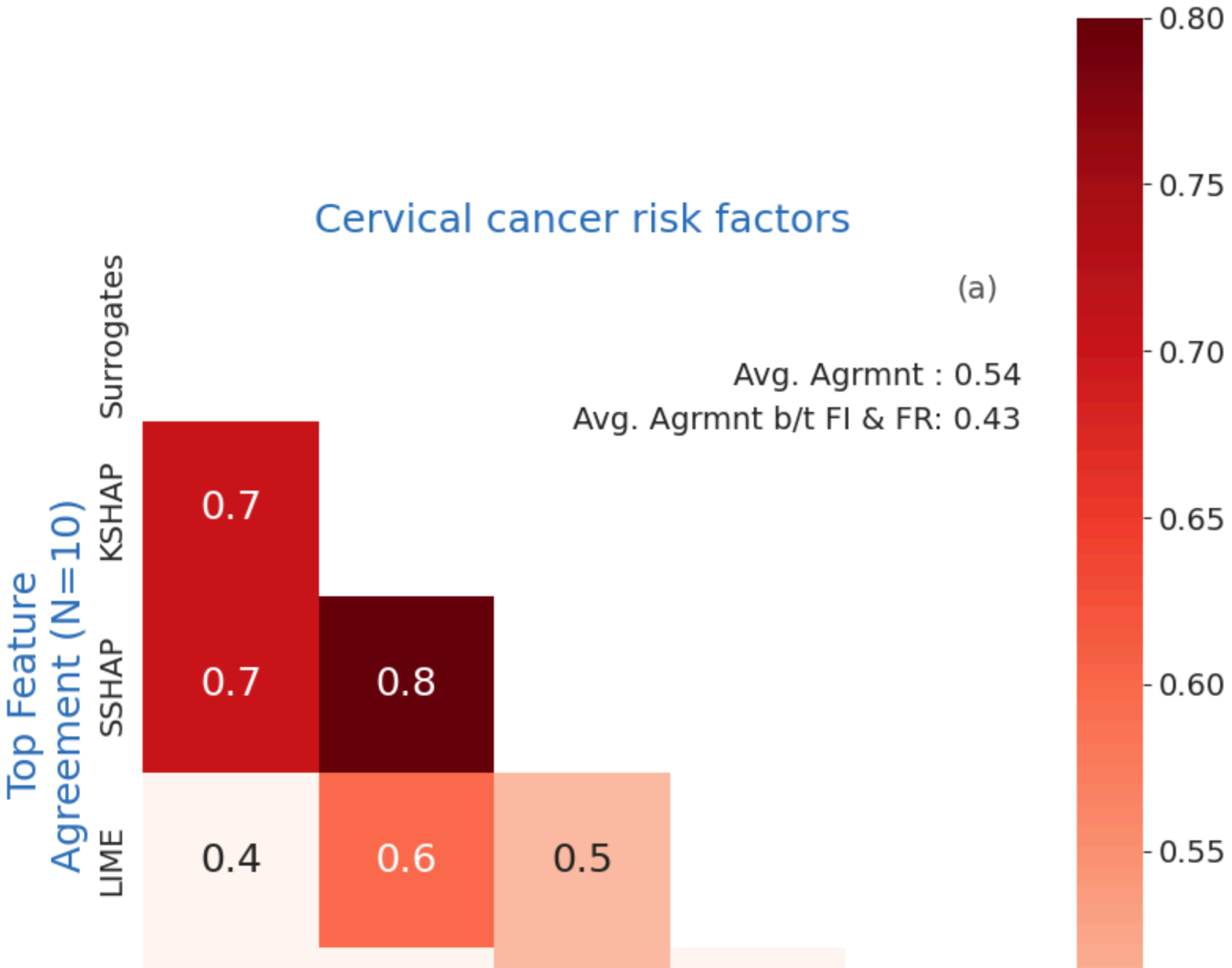
In [ ]:

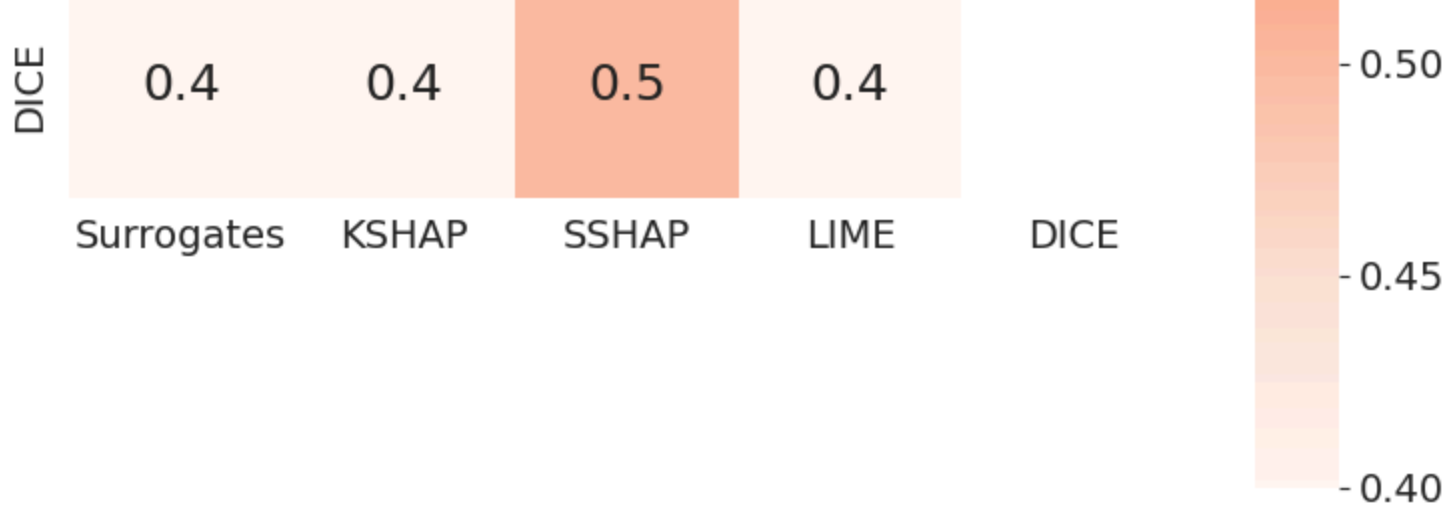
```
feature_agree, rank_agree = compute_matrices(weights, instance)
```

In [ ]:

```
corr = feature_agree
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
fig = plt.figure(figsize=(9, 11))
with sns.axes_style("white"):

    ax = sns.heatmap(corr, mask=mask, square=True,annot=True, annot_kws={'fontsize': 18},
                xticklabels=methods, yticklabels=methods, cmap="Reds", cbar=True)
    ax.set_title("Cervical cancer risk factors", color='xkcd:medium blue', fontsize=18)
    ax.set_ylabel('Top Feature\nAgreement (N=10)', color='xkcd:medium blue', fontsize=18)
    ax.text(0.95,
                0.95,
                f"(a)",
                fontsize=14,
                alpha=0.8,
                ha="center",
                va="center",
                transform=ax.transAxes,
            )
    data=corr
    avg = np.mean(data[mask==0])
    text = f'Avg. Agrmnt : {avg:.2f}'
```

```
ax.annotate(text, (1.0, 0.84), xycoords='axes fraction', fontsize=14, ha='right')

avg = np.mean(data[4:, :4])
text = f'Avg. Agrmnt b/t FI & FR: {avg:.2f}'
ax.annotate(text, (1.0, 0.79), xycoords='axes fraction', fontsize=14, ha='right')



plt.show()
```

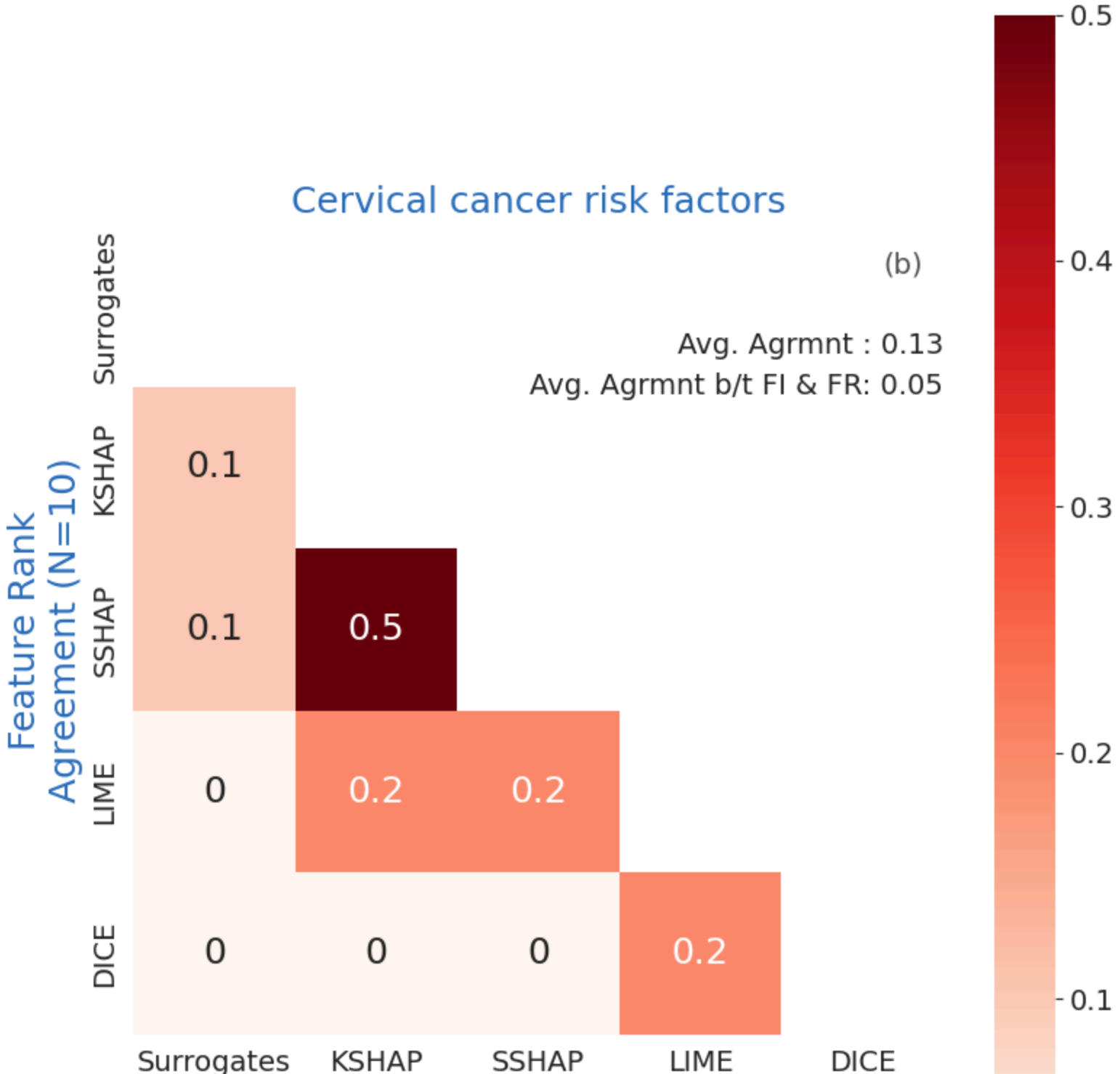


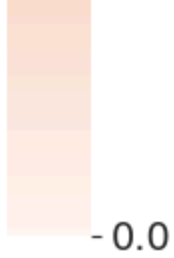



In [ ]:
```
fig.savefig('/content/drive/My Drive/dataXAI/cancer/featagrem'+str(instance)+'.png', bbox_inches='tight', dpi=300)
```

In [ ]:
```
corr = rank_agree
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
fig = plt.figure(figsize=(9, 11))
with sns.axes_style("white"):

    ax = sns.heatmap(corr, mask=mask, square=True,annot=True, annot_kws={'fontsize': 18},
                xticklabels=methods, yticklabels=methods, cmap="Reds")
    ax.set_title("Cervical cancer risk factors", color='xkcd:medium blue', fontsize=18)
    ax.set_ylabel('Feature Rank\nAgreement (N=10)', color='xkcd:medium blue',fontsize=18 )

    ax.text(0.95,
                0.95,
                f"(b)",
                fontsize=14,
                alpha=0.8,
                ha="center",
                va="center",
                transform=ax.transAxes,
            )
    data=corr
    avg = np.mean(data[mask==0])
    text = f'Avg. Agrmnt : {avg:.2f}'
    ax.annotate(text, (1.0, 0.84), xycoords='axes fraction', fontsize=14, ha='right')

    avg = np.mean(data[4:, :4])
    text = f'Avg. Agrmnt b/t FI & FR: {avg:.2f}'
    ax.annotate(text, (1.0, 0.79), xycoords='axes fraction', fontsize=14, ha='right')
```

```
plt.show()
```


Cervical cancer risk factors
(b)
Avg. Agrmnt : 0.13
Avg. Agrmnt b/t FI & FR: 0.05
Feature Rank Agreement (N=10)
Surrogates
KSHAP
SSHAP
LIME
DICE
0.1
0.1
0.5
0
0.2
0.2
0
0
0
0.2
0.5
0.4
0.3
0.2
0.1




In [ ]:

In [ ]:
```
fig.savefig('/content/drive/My Drive/dataXAI/cancer/rankagrem'+str(instance)+'.png', bbox_inches='tight', dpi=300)
```