Which Explanation Makes Sense? A Critical Evaluation of Local Explanations for Assessing Cervical Cancer Risk Factors

Submitted for UIUC CS-598: Deep Learning For Healthcare, SP24

Instructors: Jimeng Sun Siddhartha Laghuvarapu

Prepared by:

Himangshu Das	hdas4@illinois.edu
Jeremy Samuel	sjeremy3@illinois.edu
Mahesh Matta	maheshm3@illinois.edu

Team: 84

Project Github link

Project Video link

Introduction

Cervical cancer happens when cells in the cervix, the lower part of the uterus that connects to the vagina, start to become abnormal. While it isn't perfectly clear what sparks the cervical cells to change their DNA, it is certain that human papilloma virus, or HPV, plays a role. The early stages of cervical cancer generally show no signs or symptoms. Hence it is very important to be able to detect it sooner. The paper offers a thorough examination of various local interpretability techniques used to elucidate the risk factors linked with cervical cancer. Its aim is to support healthcare professionals utilizing AI by providing insights into the most suitable explanation methods for specific situations. The framework proposed in the paper will be assessed to gauge the effectiveness of different explanations in assessing cervical cancer risk, and to determine how various explanations can be tailored to different patient scenarios. Additionally, the paper will introduce an empirical approach for calculating the faithfulness metric of different machine learning interpretability methods. This approach involves analyzing feature and rank agreement, and ensuring that removing the top N features, as predicted by local explanations, does not significantly impact model performance.

Scope of Reproducibility

The original paper focuses on evaluating different explanation methods for assessing cervical cancer risk. It aims to provide insights into the quality and suitability of these explanations in a healthcare context. The contribution lies in offering a systematic framework for interpreting model predictions and assessing the performance of various explanation techniques. The scope of reproducibility in the project draft involves replicating the evaluation of explanation methods conducted in the original paper. This includes implementing the systematic approach for interpreting model predictions and comparing the performances of different explanation methods in the context of cervical cancer risk assessment.

References:

1. Ayad, Wafa & Bonnier, Thomas & Bosch, Benjamin & Read, Jesse & Parbhoo, Sonali. (2023). Which Explanation Makes Sense? A Critical Evaluation of Local Explanations for Assessing Cervical Cancer Risk Factors Ecole polytechnique. 1-50.
2. Data Source: https://archive.ics.uci.edu/dataset/383/cervical+cancer+risk+factors
3. Original github: https://github.com/cwayad/Local-Explanations-for-Cervical-Cancer
4. Kaggle: https://www.kaggle.com/code/renadope/cervical-cancer-classification-99-4-recall

Methodology

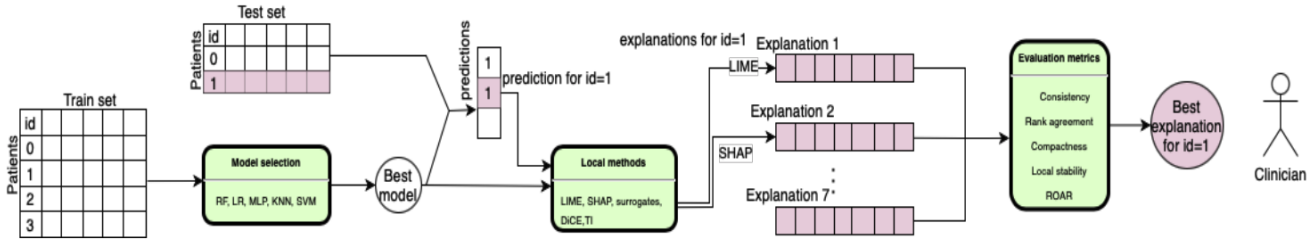
The methodology involves training multiple machine learning models (LR, RF, SVM, KNN, MLP) on preprocessed data to predict cervical cancer risk. The Random Forest model is selected as the best-performing model based on AUC performance. Local explainability techniques are then applied to interpret the predictions of the Random Forest model, including LIME and SHAP, to assess the quality of different explanations for cervical cancer risk assessment. Various metrics are computed to determine which explanation method makes the most sense for assessing cervical cancer risk.

The paper proposes a systematic approach for interpreting the predictions of a black-box model using multiple interpretability methods, and compare the explanations based on desired criteria.

The approach consists of three key phases:

1. first we train a series of models for cervical cancer risk assessment and choose the best among these models;
2. next, we interpret the models from 1) using a series of local explainability techniques;
3. we compute a series of metrics to assess the plausibility and coherency of each of the explainability methods considered.

It is as described in the figure below -



Environment

Tested the following code on Python version 3.9.12.

Dependencies required are listed below but following are the main packages: shap, lime, treeinterpreter, dice-ml, shapash, sklearn, numpy, pandas, matplotlib, seaborn, gdown

In [ ]:
```
!pip install psutil -U kaleido
!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 shapash
#!pip install -U kaleido
```

In [ ]:
```
!pip install seaborn
!pip install imblearn
```

In [818…
```
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
import plotly.io as pio
from numpy import arange

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
from scipy.stats import spearmanr
import itertools
from sklearn.metrics import roc_auc_score
from sklearn.neural_network import MLPClassifier
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import PartialDependenceDisplay, partial_dependence
from sklearn.model_selection import StratifiedKFold, cross_val_score
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
import dice_ml
from lime import lime_tabular
from treeinterpreter import treeinterpreter as ti
from sklearn.preprocessing import normalize
#from matplotlib import pyplot as plt

import random
import pickle

from shapash.explainer.consistency import Consistency
```

```
from shapash import SmartExplainer

import sys
import os


warnings.filterwarnings('ignore')
```

In [819…
```
print(sys.version)
```

```
3.9.19 (main, May  6 2024, 14:39:30)
[Clang 14.0.6 ]
```

Data

Data description:

The original paper uses e Cervical cancer risk factors dataset X from the UCI repository, the data is available publicly here. This data contains 858 female patients characterized by D = 35 features including demographic information such as age and number of pregnancies, clinical tests such as Hinselmann, Schiller and Citology, many Sexually Transmitted Diseases such as HPV and AIDS, and diagnosis taken by the patients such as HPV and CIN. For the scope of the project we will be reusing the sample dataset from the original paper.

Implementation code

The .csv is stored in GDrive and made available on runtime using gdown library. Majority of time was spent on importing the code from the original paper setting up the Google colab environment.

The dataset comprises demographic information, habits, and historic medical records of 858 patients. Several patients decided not to answer some of the questions because of privacy concerns (missing values). The code uses the ADASYN (AdaptiveSynthetic Sampling) technique in order to balance balanced the dataset by oversampling the minority class. After preprocessing with ADASYN, the new dataset contains 1677 patients. The dataset is split into 80% for training and 20% for testing.

In [820…
```
#from google.colab import drive
#drive.mount('/content/drive')
```

In [ ]:
```
# Downloads data from Gdrive hosted in hdas4@illinois.edu
import gdown

url = "https://drive.google.com/file/d/1jH2A93bp2z86Avm08ev7Fz6khDDQia63/view?usp=sharin
output = "risk_factors_cervical_cancer.csv"
gdown.download(url=url, output=output, fuzzy=True)

#url = 'https://drive.google.com/drive/folders/1mT-A-HIQ8w6t260PiDHDndA_nwhp-KmL'
#gdown.download_folder(url, quiet=True, remaining_ok=True, use_cookies=False)
```

Overview - Exploratory Data Analysis

In [822…
```
risk_factor_df = pd.read_csv('risk_factors_cervical_cancer.csv', delimiter=',', encoding
risk_factor_df.head()
```

Out[822]:

	Age	Number of	First sexual	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives	II

		sexual partners	intercourse						(years)	
0	18	4.0	15.0	1.0	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
2	34	1.0	?	1.0	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
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	15.0	0

5 rows × 36 columns

In [823…
```
risk_factor_df[risk_factor_df['Dx:HPV']==1  ]
```

Out[823]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Horm Contracep (ye
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	
8	45	1.0	20.0	5.0	0.0	0.0	0.0	0.0	
23	40	1.0	20.0	2.0	0.0	0.0	0.0	1.0	
64	38	2.0	15.0	4.0	0.0	0.0	0.0	1.0	
109	32	2.0	17.0	1.0	0.0	0.0	0.0	?	
188	27	5.0	19.0	2.0	0.0	0.0	0.0	1.0	
335	29	2.0	18.0	4.0	0.0	0.0	0.0	0.0	
372	21	5.0	13.0	3.0	1.0	1.266972909	0.5132021277	1.0	
578	19	1.0	18.0	1.0	0.0	0.0	0.0	1.0	
610	21	2.0	18.0	3.0	0.0	0.0	0.0	0.0	
669	38	3.0	22.0	2.0	?	?	?	1.0	
727	31	2.0	19.0	2.0	0.0	0.0	0.0	1.0	
738	27	6.0	17.0	2.0	0.0	0.0	0.0	0.0	
763	41	3.0	18.0	5.0	0.0	0.0	0.0	1.0	
775	27	2.0	14.0	3.0	0.0	0.0	0.0	1.0	
797	33	3.0	19.0	3.0	0.0	0.0	0.0	1.0	
822	36	3.0	20.0	2.0	0.0	0.0	0.0	1.0	
849	32	3.0	18.0	1.0	1.0	11.0	0.16	1.0	

18 rows × 36 columns

In [824…
```
risk_factor_df['Dx:HPV'].value_counts()
```

Out[824]:
```
Dx:HPV
0    840
1     18
Name: count, dtype: int64
```

In [825…
```
risk_factor_df.isna().sum()
```

Out[825]:
```
Age                                 0
Number of sexual partners           0
First sexual intercourse            0
```

```
Num of pregnancies                    0
Smokes                                0
Smokes (years)                        0
Smokes (packs/year)                   0
Hormonal Contraceptives               0
Hormonal Contraceptives (years)       0
IUD                                   0
IUD (years)                           0
STDs                                  0
STDs (number)                         0
STDs:condylomatosis                   0
STDs:cervical condylomatosis          0
STDs:vaginal condylomatosis           0
STDs:vulvo-perineal condylomatosis    0
STDs:syphilis                         0
STDs:pelvic inflammatory disease      0
STDs:genital herpes                   0
STDs:molluscum contagiosum            0
STDs:AIDS                             0
STDs:HIV                              0
STDs:Hepatitis B                      0
STDs:HPV                              0
STDs: Number of diagnosis             0
STDs: Time since first diagnosis      0
STDs: Time since last diagnosis       0
Dx:Cancer                             0
Dx:CIN                                0
Dx:HPV                                0
Dx                                    0
Hinselmann                            0
Schiller                              0
Citology                              0
Biopsy                                0
dtype: int64
```

In [826…

```
risk_factor_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 858 entries, 0 to 857
Data columns (total 36 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Age                                 858 non-null    int64
 1   Number of sexual partners           858 non-null    object
 2   First sexual intercourse            858 non-null    object
 3   Num of pregnancies                  858 non-null    object
 4   Smokes                              858 non-null    object
 5   Smokes (years)                      858 non-null    object
 6   Smokes (packs/year)                 858 non-null    object
 7   Hormonal Contraceptives             858 non-null    object
 8   Hormonal Contraceptives (years)     858 non-null    object
 9   IUD                                 858 non-null    object
 10  IUD (years)                         858 non-null    object
 11  STDs                                858 non-null    object
 12  STDs (number)                       858 non-null    object
 13  STDs:condylomatosis                 858 non-null    object
 14  STDs:cervical condylomatosis        858 non-null    object
 15  STDs:vaginal condylomatosis         858 non-null    object
 16  STDs:vulvo-perineal condylomatosis  858 non-null    object
 17  STDs:syphilis                       858 non-null    object
 18  STDs:pelvic inflammatory disease    858 non-null    object
 19  STDs:genital herpes                 858 non-null    object
 20  STDs:molluscum contagiosum          858 non-null    object
 21  STDs:AIDS                           858 non-null    object
 22  STDs:HIV                            858 non-null    object
 23  STDs:Hepatitis B                    858 non-null    object
```

```
 24  STDs:HPV                            858 non-null    object
 25  STDs: Number of diagnosis           858 non-null    int64
 26  STDs: Time since first diagnosis    858 non-null    object
 27  STDs: Time since last diagnosis     858 non-null    object
 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
dtypes: int64(10), object(26)
memory usage: 241.4+ KB
```

Preprocessing

In [827…

```
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)
```

In [828…

```
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' '?' '1
8.0' '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
--------------------------------------------------------------------------------
--------------------------------------------------------------
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 [829…
```
#these columns are not of type object, but are of type numeric
cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Num of preg
                   'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contraceptives',
                   'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STDs', 'STD
                   'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:vaginal
                   'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic i
                   'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AIDS', 'ST
                   'STDs:HPV', 'STDs: Time since first diagnosis',
                   'STDs: Time since last diagnosis']

risk_factor_df[cols_to_convert] = risk_factor_df[cols_to_convert].apply(pd.to_numeric, e
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))
```

Categorizing data based on age

In [830…
```
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 [831…
```
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 [832…
```
test_cols = ["Hinselmann", "Schiller", "Citology", "Biopsy"]
risk_factor_df["total_tests"] = risk_factor_df[test_cols].sum(axis = 1)
```

In [833…

```
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 [834…

```
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 [836…
```
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).capitaliz
fig.show()
```

In [837…
```
def stats(x):
    temp1=(df[[x,label]].value_counts(normalize=True).round(decimals=3)*100).reset_index
    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'),
    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 [838…
```
df=risk_factor_df
label='age_cat'
```

In [839…
```
stats('Dx:Cancer')
```

Out[839]:

	proportion		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 [840…
```
age_dist = px.histogram(risk_factor_df, x="Age", marginal="box", color_discrete_sequence
```

```
age_dist.update_layout(title="Age distribution")
age_dist.show()
```

Pregnancy Distribution by Age

In [841…
```
tempp=risk_factor_df.sort_values(by="Age",ascending=True)
```

In [842…
```
risk_factor_df.age_cat
```

Out[842]:
```
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 [843…
```
age_preg_bar = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="age_cat",
                      color_discrete_sequence=["darkblue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forties", "F
                                       "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 [844…
```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="ag
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forties", "F
                                       "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_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 [845…
```
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()
```

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:Cancer',
                  '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_sca
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 [848…
```
fig = px.histogram(std_agg, x="age_cat", y=list(std_cols), barmode="group", histfunc="su
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 [849…
```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="ag
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forties", "F
                                       "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_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 [850…
```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_std", lab
                   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 [851…
```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_std","Dx:
                   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 [852…
```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by="total_tests", a
                   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 [853…
```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by=["total_tests","
                   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 [854…

```
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 [855…

```
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_fa

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","Categ
fig = px.pie(proportion_samples_df,
             values='Sample Proportion',
             names="Category",
             title='Age Category proportion of women sampled',color_discrete_sequence=px
fig.show()
proportion_samples_df
```

Out[855]:

	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 [856…

```
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, HP
)
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 [857…
```
df_hormonal_compariosn = risk_factor_df.groupby(["age_cat"], as_index=False)[["IUD", "Ho
fig = px.histogram(df_hormonal_compariosn, x="age_cat", y=["IUD", "Hormonal Contraceptiv
                   , 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 [858…
```
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="gro
                   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 [859…
```
df_IUD_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contraceptives"] == 0)
df_IUD_contraceptives = df_IUD_contraceptives.sort_values(by=["Smokes", label], ascendin
fig = px.histogram(df_IUD_contraceptives, x="age_cat", color="Smokes", barmode="group",
                   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 [860…
```
df_both_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contraceptives"] == 1)
df_both_contraceptives = df_both_contraceptives.sort_values(by="Smokes")
fig = px.histogram(df_both_contraceptives, x="age_cat", color="Smokes", barmode="group",
                   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
fig.show()
```

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 [861…
```
test=risk_factor_df[['Number of sexual partners',       'First sexual intercourse',
```

In [862…
```
with open('summary.tex','w') as tf:
    tf.write(test.round(2).to_latex())
```

In [863…
```
risk_factor_df.columns
```

Out[863]:
```
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 [864…
```
label="Dx:Cancer"
```

In [865…
```
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 [866…
```
X = risk_factor_df.drop([label, "age_cat"], axis=1)
y = risk_factor_df[label].copy()
```

Train-Test Split

Data split was stratified on **Age Category**

Ablations

1. To try running the ML models on imbalanced classes
2. Try different sampling techniques other than ADASYN and test model accuracies

In [867…
```
ros = RandomOverSampler(random_state=42)
x_ros, y_ros = ros.fit_resample(X, y)
```

```
risk_factor_df_ros = x_ros.join(y_ros)
risk_factor_df_ros["age_cat"] = risk_factor_df_ros["Age"].apply(age_cat)
```

In [868…
```
smote = SMOTE(random_state=42)
x_smote, y_smote = smote.fit_resample(X, y)
risk_factor_df_smote = x_smote.join(y_smote)
risk_factor_df_smote["age_cat"] = risk_factor_df_smote["Age"].apply(age_cat)
```

In [869…
```
adasyn = ADASYN(random_state=42)
x_adasyn,y_adasyn = adasyn.fit_resample(X,y)
risk_factor_df = x_adasyn.join(y_adasyn)
risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)
```

In [870…
```
risk_factor_df_org = X.join(y)
risk_factor_df_org["age_cat"] = risk_factor_df["Age"].apply(age_cat)
```

In [871…
```
def classes_details(risk_factor_df,type):
    dx_cancer = px.histogram(risk_factor_df, y=label)
    dx_cancer.update_layout(bargap=0.2)
    dx_cancer.update_layout(title = "Balanced Classes for "+type)
    dx_cancer.show()
classes_details(risk_factor_df_ros,"Random Over Sampler")
classes_details(risk_factor_df_smote, "SMOTE")
classes_details(risk_factor_df_org,"Original")
classes_details(risk_factor_df,"ADASYN")
```

Model

Original Paper: Ayad, Wafa & Bonnier, Thomas & Bosch, Benjamin & Read, Jesse & Parbhoo, Sonali. (2023). Which Explanation Makes Sense? A Critical Evaluation of Local Explanations for Assessing Cervical Cancer Risk Factors Ecole polytechnique. 1-50. https://www.researchgate.net/profile/Wafa-Ayad/publication/374061335_Which_Explanation_Makes_Sense_A_Critical_Evaluation_of_Local_Explanations Explanation-Makes-Sense-A-Critical-Evaluation-of-Local-Explanations-for-Assessing-Cervical-Cancer-Risk-Factors-Ecole-polytechnique.pdf

Original paper repo: https://github.com/cwayad/Local-Explanations-for-Cervical-Cancer

Model descriptions

The project employs a variety of machine learning models and local explanation techniques to interpret predictions and assess the risk factors associated with cervical cancer. The models utilized include:

1. **Logistic Regression (LR)**: A linear model that estimates probabilities using a logistic function, making it suitable for binary classification tasks like predicting cervical cancer risk.
2. **Random Forest (RF)**: A popular ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random forests are known for their robustness and ability to handle high-dimensional data.
3. **Multi-Layer Perceptron (MLP)**: A type of feedforward artificial neural network composed of multiple layers of nodes, each layer fully connected to the next one. MLPs are capable of learning complex patterns in data and are commonly used for classification tasks.

4. **Support Vector Machine (SVM)**: A supervised learning algorithm that constructs a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification, regression, or other tasks. SVMs are effective in high-dimensional spaces and are particularly well-suited for cases where the number of dimensions exceeds the number of samples.
5. **K-Nearest Neighbors (KNN)**: A non-parametric, lazy learning algorithm used for classification and regression tasks. KNN works by finding the k-nearest neighbors of a given data point and making predictions based on their class labels or feature values.

For interpreting the predictions of these models, various local explanation techniques are employed, including:

1. **LIME (Local Interpretable Model-agnostic Explanations)**: A method for explaining the predictions of any classifier by approximating it locally with an interpretable model.
2. **SHAP (SHapley Additive exPlanations)**: A game-theoretic approach to explain the output of any machine learning model. SHAP values provide a unified measure of feature importance.
3. **DiCE (Diverse Counterfactual Explanations)**: A framework for generating diverse counterfactual explanations for individual predictions of machine learning models.
4. **Tree Interpreter**: A technique specifically designed for interpreting decision trees by calculating feature importances and contributions to predictions.
5. **Local Surrogates**: A method for creating a simpler, interpretable model that approximates the behavior of a complex black-box model locally.

Implementation code

Currently implemented all the basic ML models: RF, SVM, LR, KNN, MLP on the given dataset and compared the metrics for each of the model based on F1 score, AUROC, Accuracy, Precision and Recall. Please find below for more details. The local explanation techniques are being currently worked on.

Training

Computational requirements

The computational requirements for the project include:

1. **Hardware Specifications**:
    - CPU: A multi-core processor with sufficient computational power for training machine learning models. A CPU with at least 4 cores and a clock speed of 2.5 GHz or higher is recommended.
    - RAM: Adequate RAM to handle the dataset size and model training. A minimum of 8 GB RAM is recommended for handling the preprocessing and training tasks efficiently.
2. **Software Tools**:
    - Python Environment: Python 3.x environment for running the project code. Anaconda distribution is recommended for managing Python dependencies.
    - Libraries: Required Python libraries include scikit-learn, TensorFlow or PyTorch (for deep learning models), SHAP, Pandas, NumPy, and gdown for downloading datasets from Google Drive.

- IDE or Text Editor: A code editor such as Jupyter Notebook or Google Colab for running Python scripts and executing code cells interactively.

3. **Storage Requirements**:

   - Dataset Storage: Sufficient storage space to store the dataset file (.csv format) and any additional files generated during preprocessing and training. Approximately 1 GB of free storage space may be required
   - Model Checkpoints: Storage space to save model checkpoints during training to resume training from a specific point in case of interruptions.

4. **Network Connectivity**:

   - Stable Internet Connection: A stable internet connection is necessary for downloading the dataset from the UCI repository and accessing files stored on Google Drive. It is also required for accessing external resources and libraries.

5. **Training Time**:

   - Model training time varies depending on the selected machine learning algorithms and the complexity of the dataset. Training multiple machine learning models (LR, RF, SVM, KNN, MLP) and then evaluation the local explainations may require several hours to complete. We are working on trying to reduce the computation models by saving the pretrained models and then using it in the Jupyter notebook.

   - Utilizing hardware accelerators such as GPUs can significantly reduce training time for deep learning models. If available, a GPU with CUDA support is recommended for faster training.

6. **High-Performance Computing (HPC)**:

   - For future experimentation involving computationally intensive tasks, access to high-performance computing (HPC) clusters or cloud-based computing resources may be beneficial. These resources can accelerate model training and experimentation by distributing tasks across multiple nodes or GPUs.

   Overall, the project requires standard computational resources including a CPU with sufficient processing power, an adequate amount of RAM, storage space for datasets and model checkpoints, stable internet connectivity, and potentially access to high-performance computing resources for faster training.

Implementation code

Currently the different models: RF, SVM, LR, KNN, MLP have been trained and evaluated below along with a few local explanation techniques. The local explanation techniques like SHAP (KSHAP, SSHAP) require around 1-2 hours of CPU runtime - we have pretrained these models and stored the generated shap values to reduce time for execution of the jupyter notebook.

Some hyperparams used are : Cross validation is 10 for KNN and 5 for SVM. We use GridSearchCV to calculate the best params for Logistic regression, KNN and SVM. Some of the observation from below ablations are mentioned below but for the sake of reproducibility of the paper we use the result with ADASYN sampled data which is:

LogisticRegression(C=268.2695795279727), KNeighborsClassifier(n_neighbors=1), SVC(C=100.0, gamma=0.001)

The computation for KSHAP for example takes around 336 iterations each one taking around 10s/iteration (for MLP model)

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

In [872…

```
col_names = ["Classifier Name", "Accuracy Score", "Precision Score",
                "Recall Score", "F1 Score", "AUROC"]

def comparing_models_on_different_sampled_data(risk_factor_df,type):
    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)
    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)
    print(f"Sampled Data: {type}")
    #len(X_test.columns)

    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)
    print(logreg_cv.best_estimator_)
    logreg_cv = LogisticRegression(**logreg_cv.best_params_)


    rnd_clf = RandomForestClassifier()
    #rnd_clf.fit(X_train, y_train)

    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_t
    print(knn_clf_cv.best_estimator_)
    knn_clf_cv = KNeighborsClassifier(**knn_clf_cv.best_params_)

    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,refit=True).fit(X_train,y_tr
    print(svm_clf_cv.best_estimator_)
    svm_clf_cv = SVC(**svm_clf_cv.best_params_)

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

    summary_df = pd.DataFrame(columns=col_names)

    est_name = []
```

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

    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"],marg
        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 of '+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

    #print(estimators)

    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()

    print(summary_df)
    return X_train, X_test, y_train, y_test, summary_df
```

In [873…

```
comparing_models_on_different_sampled_data(risk_factor_df_ros,"Random Over Sampler")
comparing_models_on_different_sampled_data(risk_factor_df_smote, "SMOTE")
comparing_models_on_different_sampled_data(risk_factor_df_org,"Original")
X_train, X_test, y_train, y_test, summary_df = comparing_models_on_different_sampled_dat
```

```
Sampled Data: Random Over Sampler
LogisticRegression(C=31.622776601683793)
KNeighborsClassifier(n_neighbors=1)
SVC(C=0.1, gamma=1.0)
```

```
           Classifier Name  Accuracy Score  Precision Score  Recall Score  \
0       LogisticRegression        0.994048         0.994117      0.994048
1  RandomForestClassifier         0.997024         0.997041      0.997024
2     KNeighborsClassifier        0.991071         0.991226      0.991071
3  SupportVectorClassifier        1.000000         1.000000      1.000000
4            MLPClassifier        0.997024         0.997041      0.997024

   F1 Score     AUROC
0  0.994047  0.993976
1  0.997024  0.996988
2  0.991070  0.990964
3  1.000000  1.000000
4  0.997024  0.996988
Sampled Data: SMOTE
LogisticRegression(C=19306.977288832535)
KNeighborsClassifier(n_neighbors=1)
SVC(C=100.0, gamma=0.001)
```

```
           Classifier Name  Accuracy Score  Precision Score  Recall Score  \
0       LogisticRegression        0.997024         0.997041      0.997024
1  RandomForestClassifier         1.000000         1.000000      1.000000
2     KNeighborsClassifier        0.955357         0.958270      0.955357
3  SupportVectorClassifier        1.000000         1.000000      1.000000
4            MLPClassifier        1.000000         1.000000      1.000000

   F1 Score     AUROC
0  0.997023  0.996855
1  1.000000  1.000000
2  0.955383  0.957307
3  1.000000  1.000000
4  1.000000  1.000000
Sampled Data: Original
LogisticRegression(C=3.727593720314938)
KNeighborsClassifier(n_neighbors=11)
SVC(C=100.0, gamma=0.001)
```

```
           Classifier Name  Accuracy Score  Precision Score  Recall Score  \
0       LogisticRegression        1.000000         1.000000      1.000000
1  RandomForestClassifier         1.000000         1.000000      1.000000
2     KNeighborsClassifier        0.994186         0.988406      0.994186
3  SupportVectorClassifier        1.000000         1.000000      1.000000
4            MLPClassifier        1.000000         1.000000      1.000000

   F1 Score  AUROC
0  1.000000    1.0
1  1.000000    1.0
2  0.991288    0.5
3  1.000000    1.0
4  1.000000    1.0
Sampled Data: ADASYN
LogisticRegression(C=268.2695795279727)
KNeighborsClassifier(n_neighbors=1)
SVC(C=100.0, gamma=0.001)
```

```
           Classifier Name  Accuracy Score  Precision Score  Recall Score  \
0       LogisticRegression        1.000000         1.000000      1.000000
```

```
1  RandomForestClassifier         1.000000         1.000000      1.000000
2     KNeighborsClassifier        0.961310         0.964200      0.961310
3  SupportVectorClassifier        0.997024         0.997042      0.997024
4            MLPClassifier        1.000000         1.000000      1.000000

   F1 Score     AUROC
0  1.000000  1.000000
1  1.000000  1.000000
2  0.961314  0.962857
3  0.997024  0.997143
4  1.000000  1.000000
```

In [874…
```
X_test.to_csv('X_test.csv')
y_test.to_csv('y_test.csv')
X_train.to_csv('X_train.csv')
y_train.to_csv('y_train.csv')
```

Summary

In [875…
```
summary_df
```

Out[875]:

	Classifier Name	Accuracy Score	Precision Score	Recall Score	F1 Score	AUROC
0	LogisticRegression	1.000000	1.000000	1.000000	1.000000	1.000000
1	RandomForestClassifier	1.000000	1.000000	1.000000	1.000000	1.000000
2	KNeighborsClassifier	0.961310	0.964200	0.961310	0.961314	0.962857
3	SupportVectorClassifier	0.997024	0.997042	0.997024	0.997024	0.997143
4	MLPClassifier	1.000000	1.000000	1.000000	1.000000	1.000000

In [876…
```
px.colors.sequential.RdBu
```

Out[876]:
```
['rgb(103,0,31)',
 'rgb(178,24,43)',
 'rgb(214,96,77)',
 'rgb(244,165,130)',
 'rgb(253,219,199)',
 'rgb(247,247,247)',
 'rgb(209,229,240)',
 'rgb(146,197,222)',
 'rgb(67,147,195)',
 'rgb(33,102,172)',
 'rgb(5,48,97)']
```

In [877…
```
acc_comparison = px.bar(summary_df, x="Classifier Name",
                        y=col_names[1:len(col_names)], labels={"value":"Test Accuracy",
                        color_discrete_sequence=["deeppink",
                                                 "deepskyblue",
                                                 "darkviolet",
                                                 "darkorange",
                                                 "darkred"],
                        barmode="group"
                        #,error_y=[dict(type='data', array=[0.5, 1, 2],visible=True), di
                        #,error_y_minus = [dict(type='data', array=[0.5, 1, 2, 2, 1],vis
                        )
acc_comparison.update_layout({'plot_bgcolor': 'rgba(0, 0, 0, 0)',
'paper_bgcolor': 'rgba(0, 0, 0, 0)'
})
acc_comparison.show()
```

In [878…
```
#acc_comparison.write_image('/content/drive/MyDrive/CS598/CS598:DLH-Project/project_outp
#acc_comparison.write_image('modelsperf.png')
```

Evaluation

Metrics descriptions

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

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%

AUROC

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease. Source

Consistency Metric: Compares explanations attributed to the same instance by different feature importance-based methods, enhancing user confidence in model predictions. This is done through computing an L2 distance between pairs of explanations. If two different methods attribute similar feature importance to the same instance or set of instances, the user's confidence and trust in the model's predictions increases.

Stability Metric: Compares explanations for instances with similar feature values and predictions, ensuring consistent attributions. In order to perform the stability analysis this uses the local Lipschitz metric for explanation stability -

$$\hat{L}(x_i) = \underset{x_j \in B_\epsilon(x_i)}{\operatorname{argmax}} \frac{\|\phi(x_i) - \phi(x_j)\|_2}{\|x_i - x_j\|_2} \tag{4}$$

where x_i refers to an instance, $B_\epsilon(x_i)$ is the ϵ-sphere centered at x_i, and $\phi(x_i)$ and $\phi(x_j)$ are the explanation parameters for x_i and x_j. Lower values indicate more stable explanations.

Compactness Metric: Measures the number of features needed to explain a certain percentage of the prediction, aiding in understanding model decisions. This can be obtained by fixing the percentage we want to reach and compute the number of features needed to explain that fixed percentage of the prediction.

Faithfulness Metric (ROAR): Involves iteratively removing features, retraining the model, and evaluating changes in accuracy or feature importance, assessing the impact of feature removal on model performance.

Implementation code

The evaluation metrics are implemented using Python libraries such as scikit-learn and SHAP.

For the Consistency Metric, L2 distance computation between pairs of explanations can be achieved using numpy.

The Stability Metric can be implemented by comparing explanations for instances with similar feature values and predictions using scikit-learn.

The Compactness Metric can be calculated by fixing a percentage of the prediction and computing the number of features needed to explain that percentage using pandas and numpy.

The Faithfulness Metric (ROAR) can be implemented by iteratively removing features, retraining the model, and evaluating changes in accuracy or feature importance using scikit-learn and pandas.

Explainable AI (XAI)

In [879…
```
risk_factor_df.columns
```

Out[879]:
```
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:CIN', 'Dx:HPV', 'Dx', 'Hinselmann', 'Schiller', 'Citology',
       'Biopsy', 'total_std', 'total_tests', 'Dx:Cancer', 'age_cat'],
      dtype='object')
```

In [880…
```
risk_factor_df=pd.read_csv('risk_factors_cervical_cancer.csv')
```

In [881…
```
X_test=pd.read_csv('X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
```

```
y_train=pd.read_csv('y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

Model

In [882…
```
rnd_clf = RandomForestClassifier()
rnd_clf.fit(X_train, y_train)
rnd_clf.score(X_train, y_train)
```

Out[882]:
```
1.0
```

In [883…
```
nn_clf = MLPClassifier()
nn_clf.fit(X_train, y_train)
nn_clf.score(X_train, y_train)
```

Out[883]:
```
0.9992542878448919
```

In [884…
```
nn_clf.score(X_test, y_test)
```

Out[884]:
```
0.9970238095238095
```

In [885…
```
model=rnd_clf
```

Local methods

Generate local FI

Tools

In [886…
```
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.DataFrame([
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
model = rnd_clf
res = dict()
features=X_test.columns
```

In [887…
```
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 [888…
```
print("-KSHAP-")
# KSHAP
```

```
K=10
explainer = shap.KernelExplainer(model.predict_proba, X_train)
#shap_values = explainer.shap_values(X_test)
#shap_values = explainer.shap_values(shap.sample(X_test, K))
#with open("kshap-rf", "wb") as fp:
#  pickle.dump(shap_values, fp)

with open("kshap-rf", "rb") as fp:
  shap_values = pickle.load(fp)
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 u
sing shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K sampl
es.
```

```
-KSHAP-
```

In [889…

```
print("-TSHAP-")
# # TSHAP
K=10
explainer = shap.TreeExplainer(model,X_train)
#shap_values = explainer.shap_values(X_test)
#with open("tshap-rf", "wb") as fp:
#  pickle.dump(shap_values, fp)

with open("tshap-rf", "rb") as fp:
  shap_values = pickle.load(fp)
#shap_values = explainer.shap_values(shap.sample(X_test, K))

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 [890…

```
print("-SSHAP-")
# SSHAP
K=10
explainer = shap.explainers.Sampling(model.predict_proba, X_train)
#shap_values = explainer.shap_values(X_test)
#with open("sshap-rf", "wb") as fp:
#  pickle.dump(shap_values, fp)

with open("sshap-rf", "rb") as fp:
  shap_values = pickle.load(fp)
#shap_values = explainer.shap_values(shap.sample(X_test, K))
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-
```

In [891…

```
print("-LIME-")
```

```
# LIME
K=10
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classification',

all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_features=X_t
  #exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_features=K,
  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 [892…

```
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[i][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 [893…

```
#df.select_dtypes(exclude=int)
```

In [894…

```
label = "Dx:Cancer"
```

In [895…

```
temp1=X_train
temp1['Dx:Cancer']=y_train
temp2=X_test
temp2['Dx:Cancer']=y_test
temp3=pd.concat([temp1,temp2])

risk_factor_df_test=temp3
```

In [896…

```
#these columns are not of type object, but are of type numeric
cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Num of preg
                   'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contraceptives',
```

```
                   'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STDs', 'STD
                   'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:vaginal
                   'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic i
                   'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AIDS', 'ST
                   '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, e
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 [897…

```
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 [898…

```
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 [899…

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

In [900…

```
print("-DICE-")

to_int_and_beyond = to_int_and_beyond.union(std_cols)
#print(to_int_and_beyond,risk_factor_df.columns)
for col in to_int_and_beyond:
    risk_factor_df[col] = risk_factor_df[col].astype(int)
df=risk_factor_df.drop(['total_std', 'age_cat', 'total_tests'],axis=1)

#print(list(X_test.columns),list(df.columns),X_test.head(),df.head())
```

```
#feature_names = [            name for name in df.columns.tolist() if name != label]

#number_of_features = len(feature_names)
#print(len(set(list(X_test.columns)))-number_of_features)

#print(list(X_test.columns),feature_names)
cont_feat = list(X_test.columns)
cont_feat.remove(label)

#print(cont_feat)

d = dice_ml.Data(dataframe=df, continuous_features=cont_feat, outcome_name=label)
#d = dice_ml.Data(dataframe=df, continuous_features=['Age','smokes'], outcome_name=label
m = dice_ml.Model(model=model, backend="sklearn")

exp = dice_ml.Dice(d, m, method="random")
query_instance = X_test.drop(label,axis=1)
#e1 = exp.generate_counterfactuals(query_instance, total_CFs=10, desired_range=None,
#                                  desired_class="opposite",
#                                  permitted_range=None, features_to_vary="all")
#with open("dice-e1-rf", "wb") as fp:
#  pickle.dump(e1, fp)

with open("dice-e1-rf", "rb") as fp:
  e1 = pickle.load(fp)
#imp = exp.local_feature_importance(query_instance, posthoc_sparsity_param=None)

#with open("dice-imp-rf", "wb") as fp:
#  pickle.dump(imp, fp)

with open("dice-imp-rf", "rb") as fp:
  imp = pickle.load(fp)

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-

In [901…

```
GloSur.to_csv("glosur-rf.csv", index=False)
kernelSHAP.to_csv("Kshap-rf.csv", index=False)
treeSHAP.to_csv("Tshap-rf.csv", index=False)
samplingSHAP.to_csv("Sshap-rf.csv", index=False)
limecontrib.to_csv("lime-rf.csv", index=False)
ticontrib.to_csv("ti-rf.csv", index=False)
dicecontrib.to_csv("dice-rf.csv", index=False)
```

In [902…

```
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("toutfi-rf.csv", index=False)
```

Get explanations

In [903…
```
instance=291
```

In [904…
```
gscontrib=pd.read_csv('glosur-rf.csv')
kercontrib=pd.read_csv('Kshap-rf.csv')
samcontrib=pd.read_csv('Sshap-rf.csv')
trecontrib=pd.read_csv('Tshap-rf.csv')
limecontrib=pd.read_csv('lime-rf.csv')
ticontrib=pd.read_csv('ti-rf.csv')
dicecontrib=pd.read_csv('dice-rf.csv')
all_fi=pd.read_csv('toutfi-rf.csv')
```

TOOLS

In [905…
```
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 [906…
```
label="Dx:Cancer"
```

In [907…
```
methods=all_fi['Method'].to_list()
weights=[gscontrib, kercontrib, samcontrib, limecontrib, dicecontrib]
```

In [908…
```
methods=all_fi['Method'].to_list()
weights=[gscontrib, kercontrib, trecontrib, samcontrib, limecontrib, ticontrib, dicecont
```

Normalize ?

In [909…
```
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 [910…
```
risk_factor_df.describe().iloc[1]
```

Out[910]:
```
Age                                 26.820513
Number of sexual partners            2.511655
First sexual intercourse            16.995338
Num of pregnancies                   2.257576
Smokes                               0.143357
Smokes (years)                       1.201241
Smokes (packs/year)                  0.446278
Hormonal Contraceptives              0.686480
Hormonal Contraceptives (years)      2.035331
IUD                                  0.096737
```

```
IUD (years)                              0.444604
STDs                                     0.092075
STDs (number)                            0.155012
STDs:condylomatosis                      0.051282
STDs:cervical condylomatosis             0.000000
STDs:vaginal condylomatosis              0.004662
STDs:vulvo-perineal condylomatosis       0.050117
STDs:syphilis                            0.020979
STDs:pelvic inflammatory disease         0.001166
STDs:genital herpes                      0.001166
STDs:molluscum contagiosum               0.001166
STDs:AIDS                                0.000000
STDs:HIV                                 0.020979
STDs:Hepatitis B                         0.001166
STDs:HPV                                 0.002331
STDs: Number of diagnosis                0.087413
STDs: Time since first diagnosis         4.177156
STDs: Time since last diagnosis          3.233100
Dx:Cancer                                0.020979
Dx:CIN                                   0.010490
Dx:HPV                                   0.020979
Dx                                       0.027972
Hinselmann                               0.040793
Schiller                                 0.086247
Citology                                 0.051282
Biopsy                                   0.064103
total_std                                0.155012
total_tests                              0.242424
Name: mean, dtype: float64
```

In [911…
```
xx=risk_factor_df.describe().iloc[1]
```

In [912…
```
#instance=3
instance=291
```

In [913…
```
xx=X_test.iloc[instance]
```

In [914…
```
idx=list(xx.to_numpy().nonzero()[0])
```

In [915…
```
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 [916…
```
new.T.round(2)
```

Out[916]:

	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

Dx:Cancer	1.00

In [917…
```
with open('instance.tex','w') as tf:
    tf.write(new.T.round(2).to_latex())
```

Instance datafame

In [918…
```
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')
one_instance.to_csv('one_instance.csv')
```

In [919…
```
'methods' in one_instance.columns
```

Out[919]:
```
False
```

In [920…
```
one_instance
```

Out[920]:

methods	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives
Surrogates	-0.830778	0.016780	0.201892	-0.138221	0.213723	-0.495039	0.000063	0.284140
KSHAP	0.034349	0.000000	-0.018645	0.014517	0.007108	0.000000	0.000311	0.034987
TSHAP	0.029736	-0.000792	-0.020874	0.012146	0.007838	-0.000912	-0.000256	0.038204
SSHAP	0.033871	0.001465	-0.021065	0.009400	0.004610	-0.001949	0.001623	0.038279
LIME	-0.005027	0.000909	-0.030621	0.005678	0.073551	-0.011215	-0.024087	0.046161
TI	0.070551	0.001145	-0.044610	0.009832	0.007511	-0.001752	-0.002998	0.023319
DICE	0.100000	0.000000	0.100000	0.000000	0.000000	0.100000	0.100000	0.300000

7 rows × 36 columns

Plot FI for one instance

In [921…
```
X_test=pd.read_csv('X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [922…
```
instance=291
var='W'
maxx=10
```

```
f=''
#vale=0
```

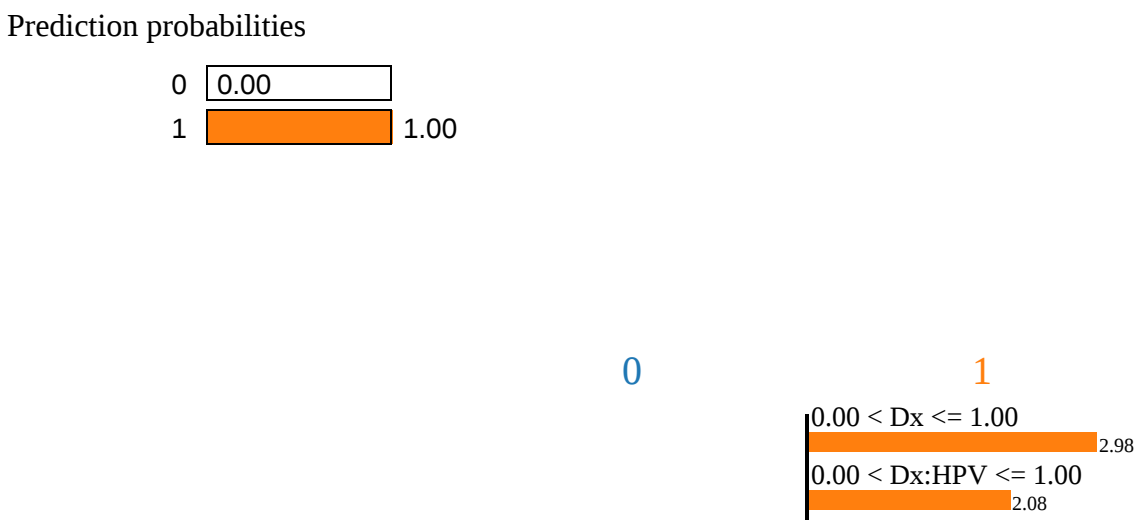
In [923…
```
# tempt=X_test.iloc[291]
# tempt[f]=vale
# #tempt["Age"]=1
# #tempt["Num of pregnancies"]=120
# #tempt["Smokes"]=200
# tempt
# X_test.iloc[291]=tempt
# X_test.iloc[291]
```

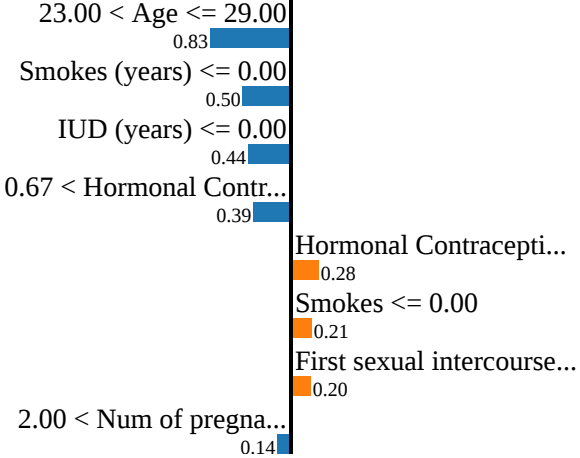
In [924…
```
model.predict(X_test)
```

Out[924]:
```
array([1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1])
```

In [925…
```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classification',
exp = explainer.explain_instance(X_test.iloc[instance], model.predict_proba, num_feature
```

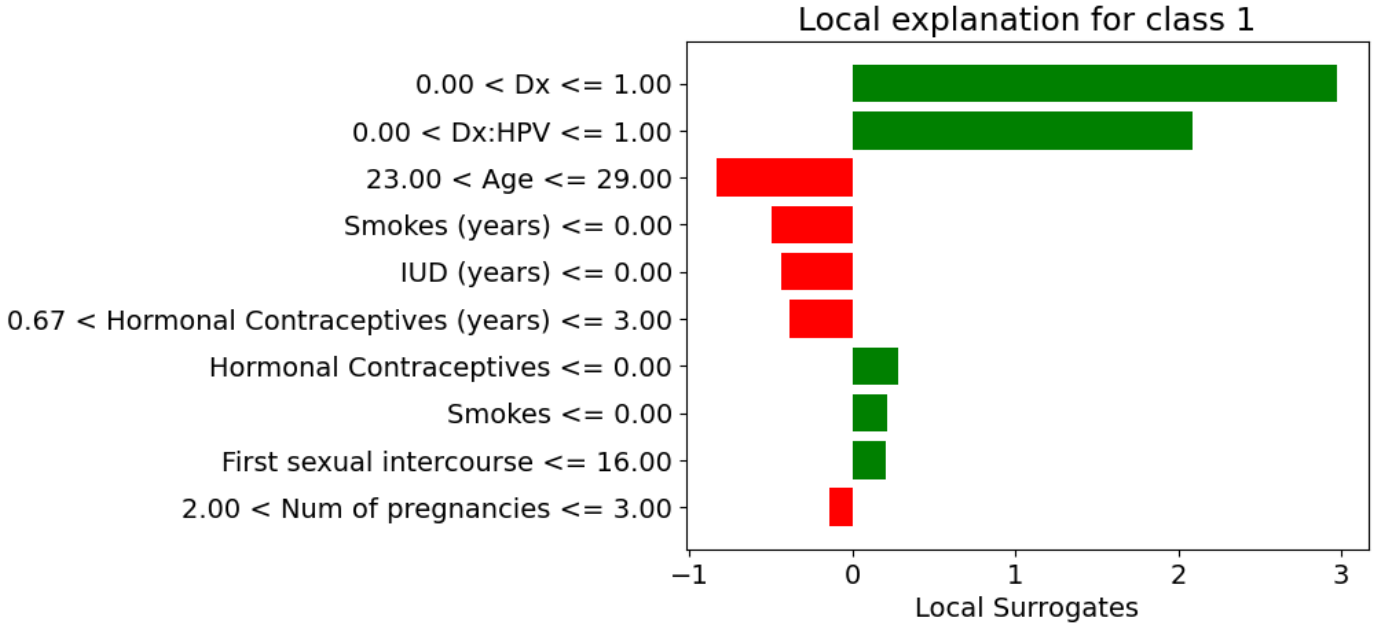
In [926…
```
items = gscontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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)
```





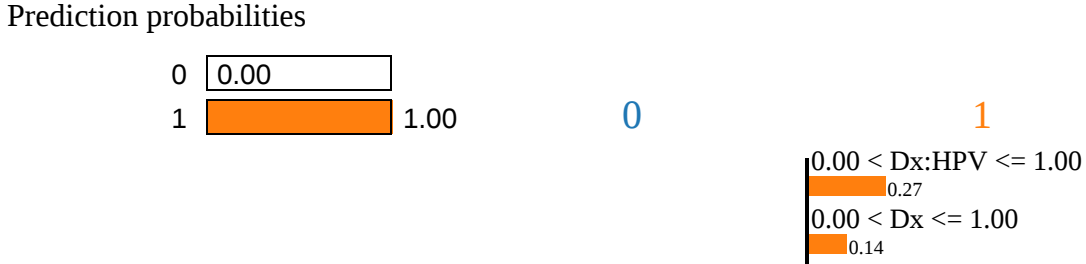
In [927…

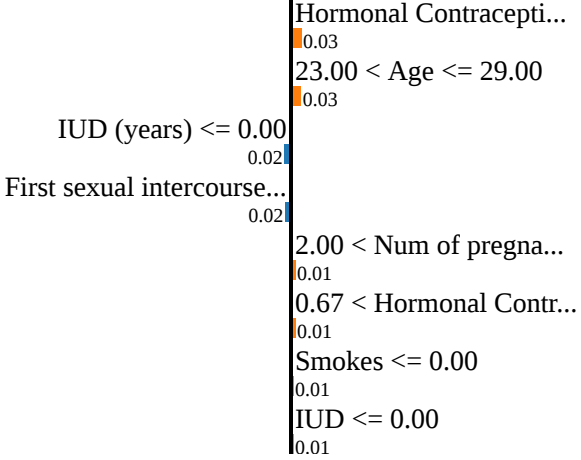
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Local Surrogates")
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'surrogate'+str(instance
fig.savefig(''+str(var)+'surrogate'+str(instance)+str(f)+'.png', bbox_inches='tight')
```



In [928…

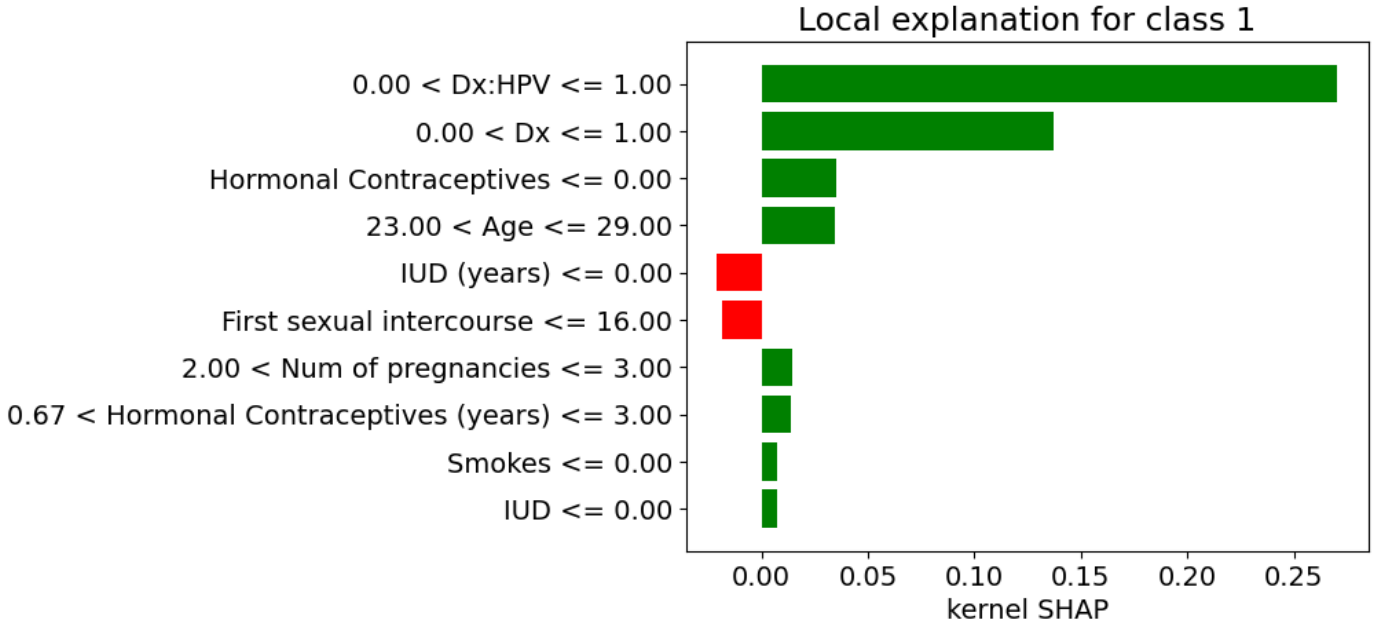
```
items = kercontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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)
```





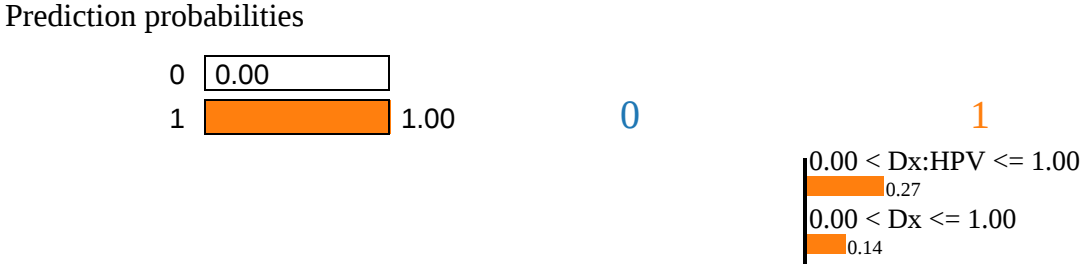
In [929…

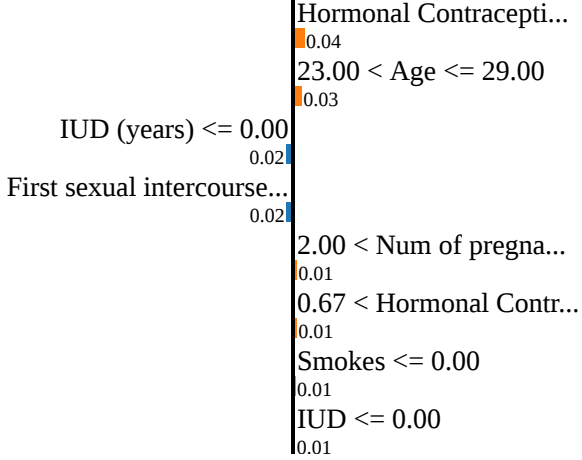
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("kernel SHAP")
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'kernelSHAP'+str(instanc
fig.savefig(''+str(var)+'kernelSHAP'+str(instance)+str(f)+'.png', bbox_inches='tight')
```



In [930…

```
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)
```

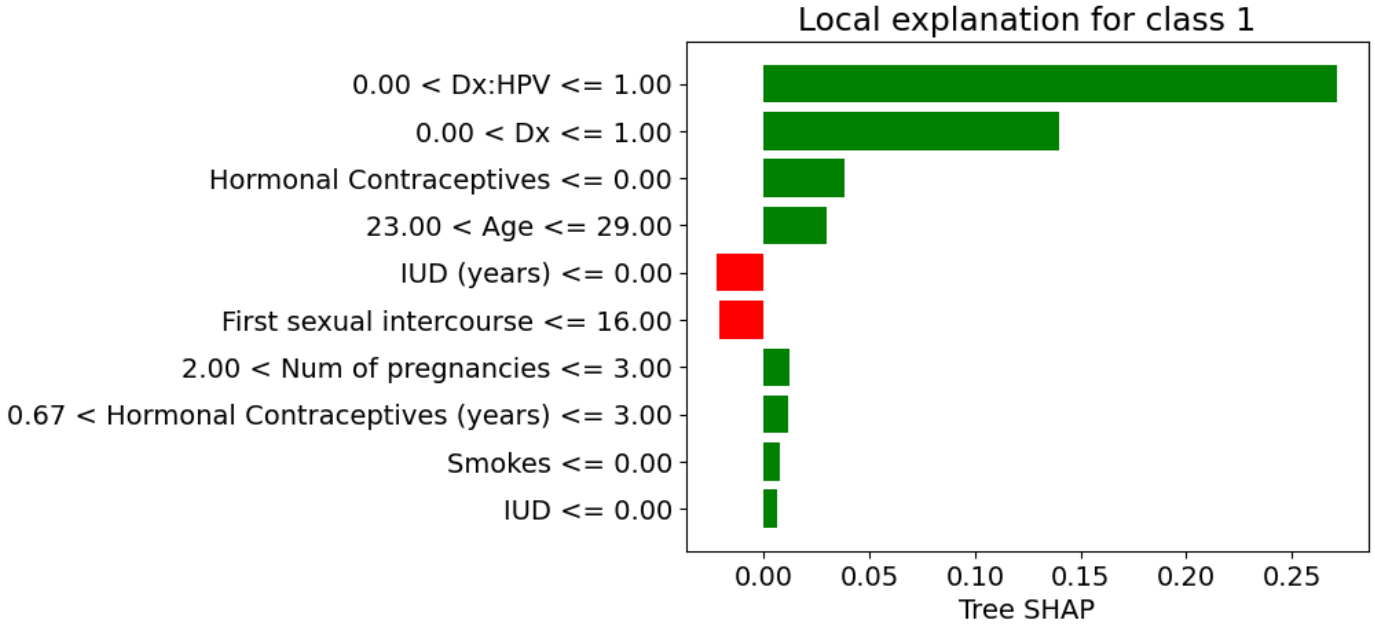




Feature	Value
Dx:HPV	1.00
Dx	1.00
Hormonal Contraceptives	0.00
Age	27.00
IUD (years)	0.00
First sexual intercourse	14.00
Num of pregnancies	3.00
Hormonal Contraceptives (years)	0.86
Smokes	0.00
IUD	0.00

In [931…

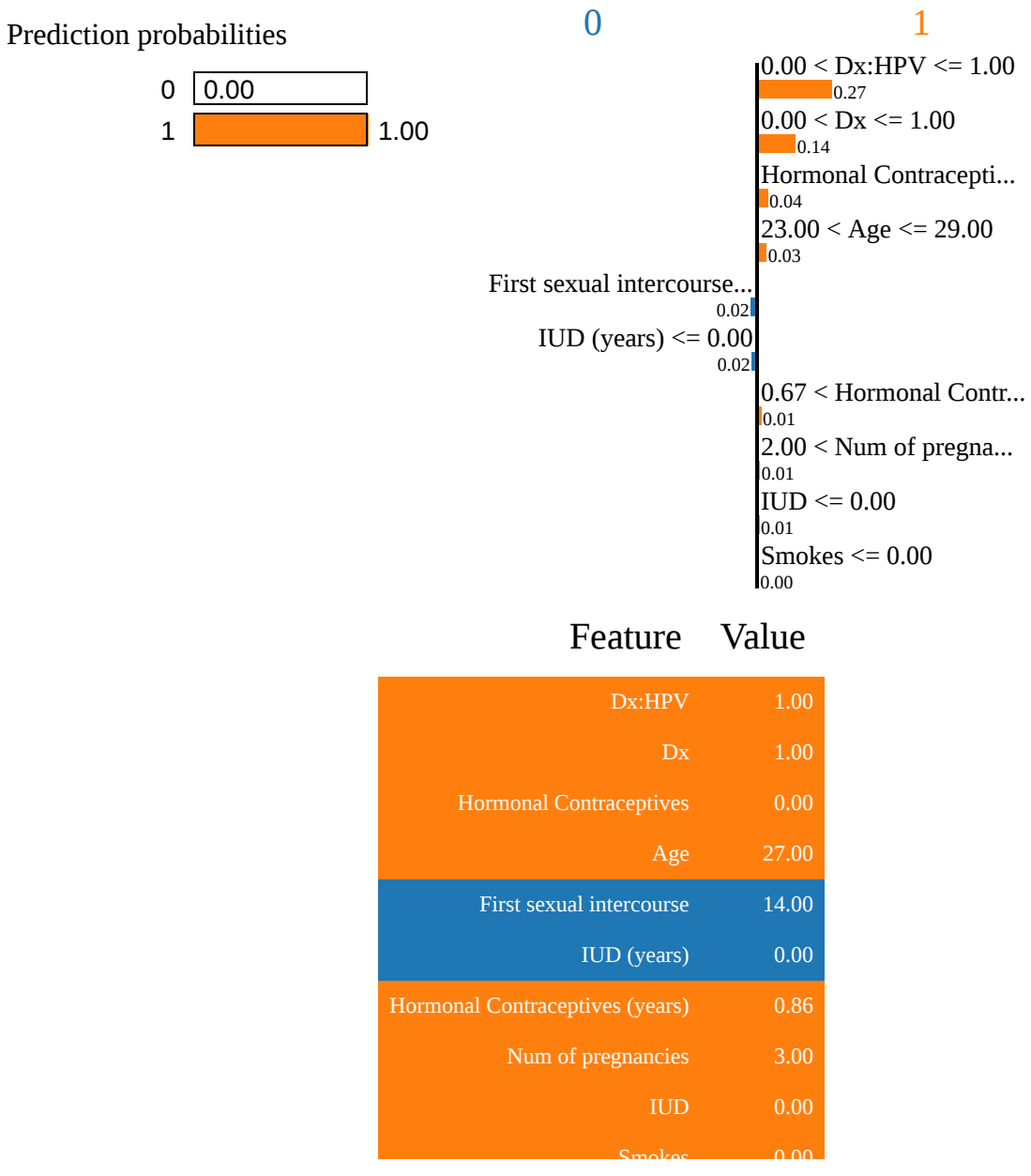
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Tree SHAP")
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'treeSHAP'+str(instance)
fig.savefig(''+str(var)+'treeSHAP'+str(instance)+str(f)+'.png', bbox_inches='tight')
```



In [932…

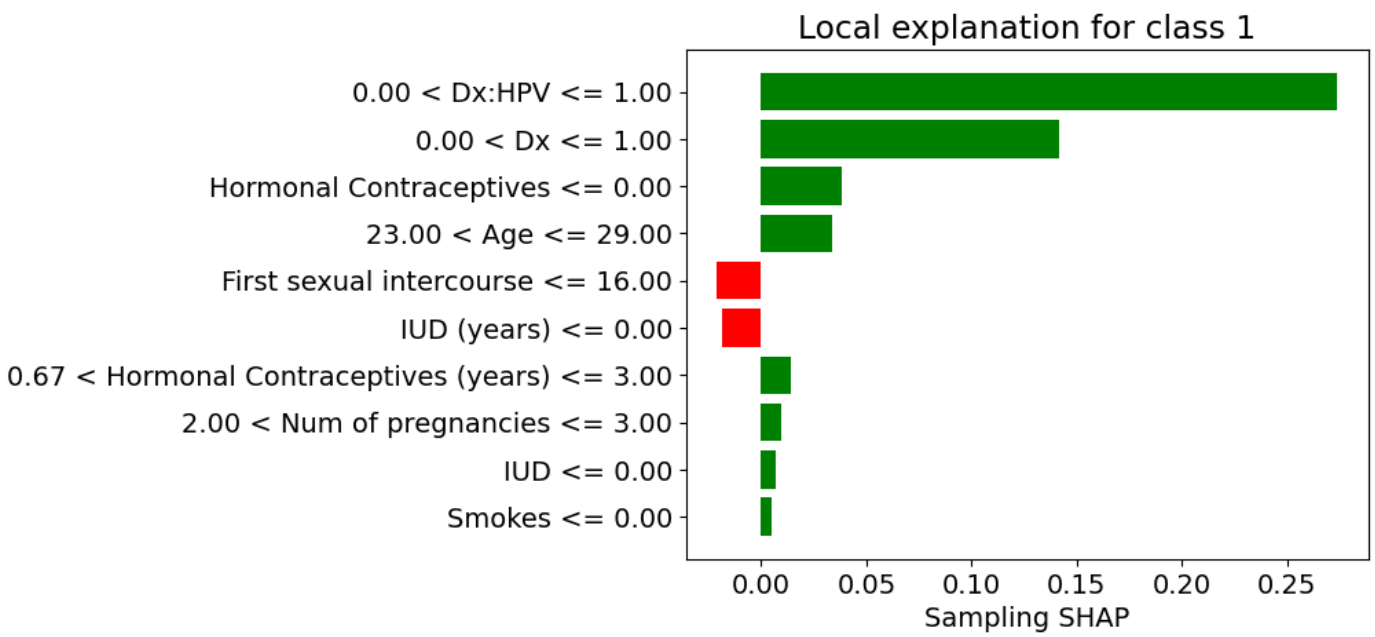
```
items = samcontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
```

```
  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)
```



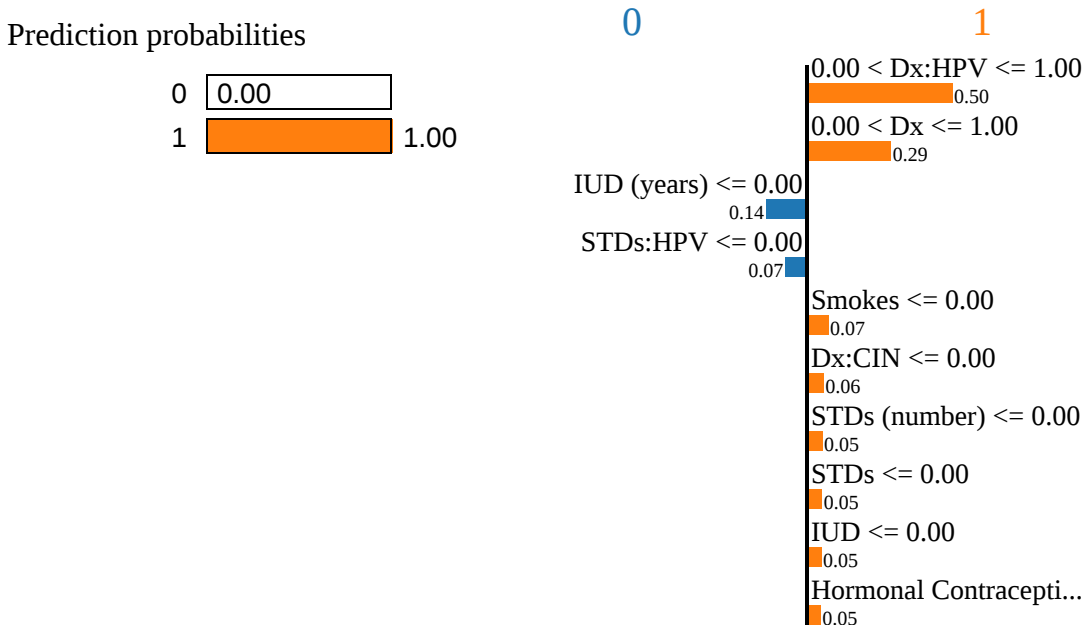
In [933…

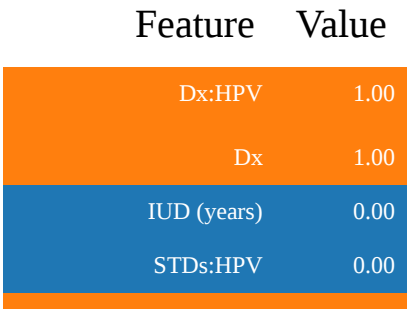
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Sampling SHAP")
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'samplingSHAP'+str(insta
fig.savefig(''+str(var)+'samplingSHAP'+str(instance)+str(f)+'.png', bbox_inches='tight')
```



In [934…

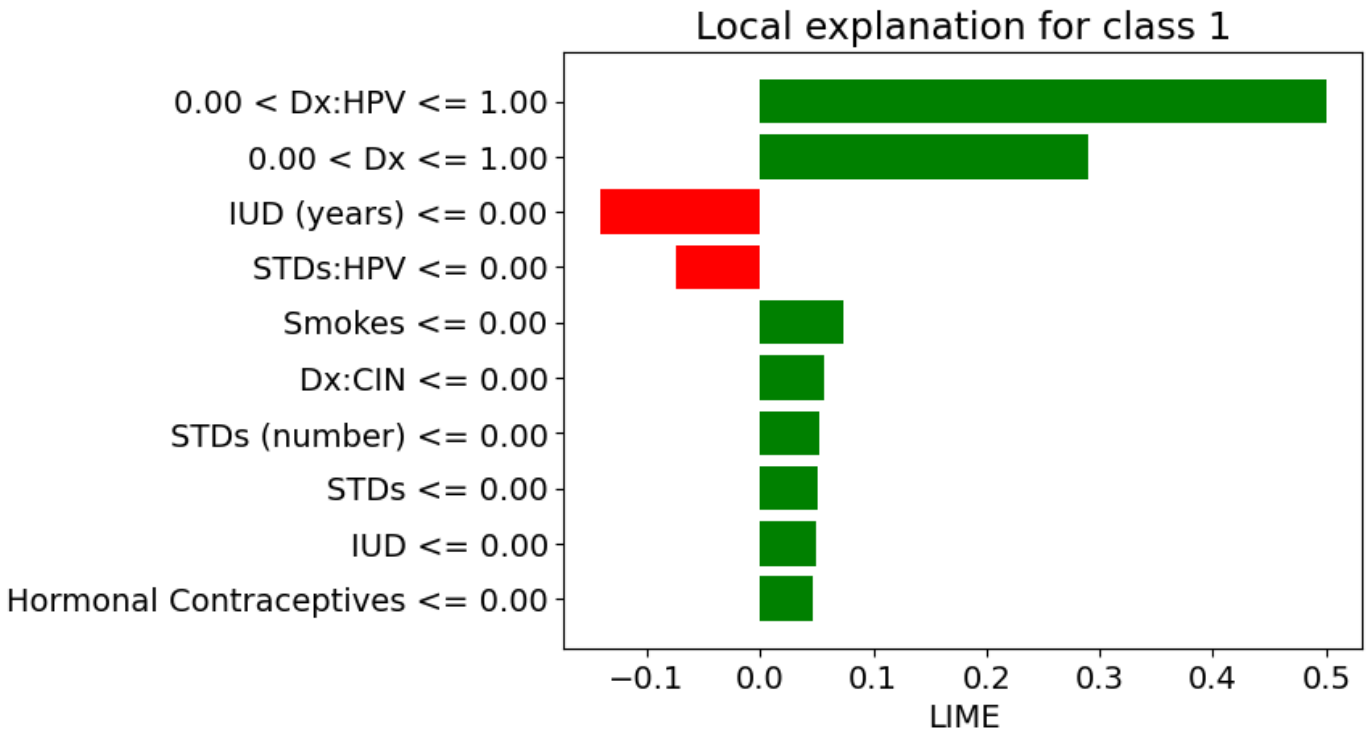
```
items = limecontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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)
```





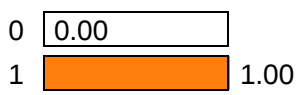
Smokes	0.00
Dx:CIN	0.00
STDs (number)	0.00
STDs	0.00
IUD	0.00

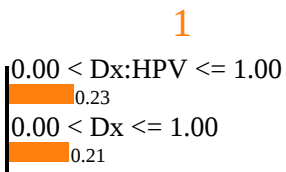
In [935…
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("LIME")
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'lime'+str(instance)+str
fig.savefig(''+str(var)+'lime'+str(instance)+str(f)+'.png', bbox_inches='tight')
```

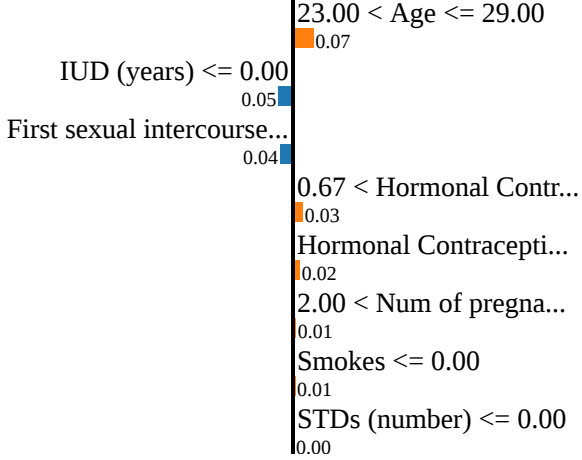


In [936…
```
items = ticontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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)
```

Prediction probabilities

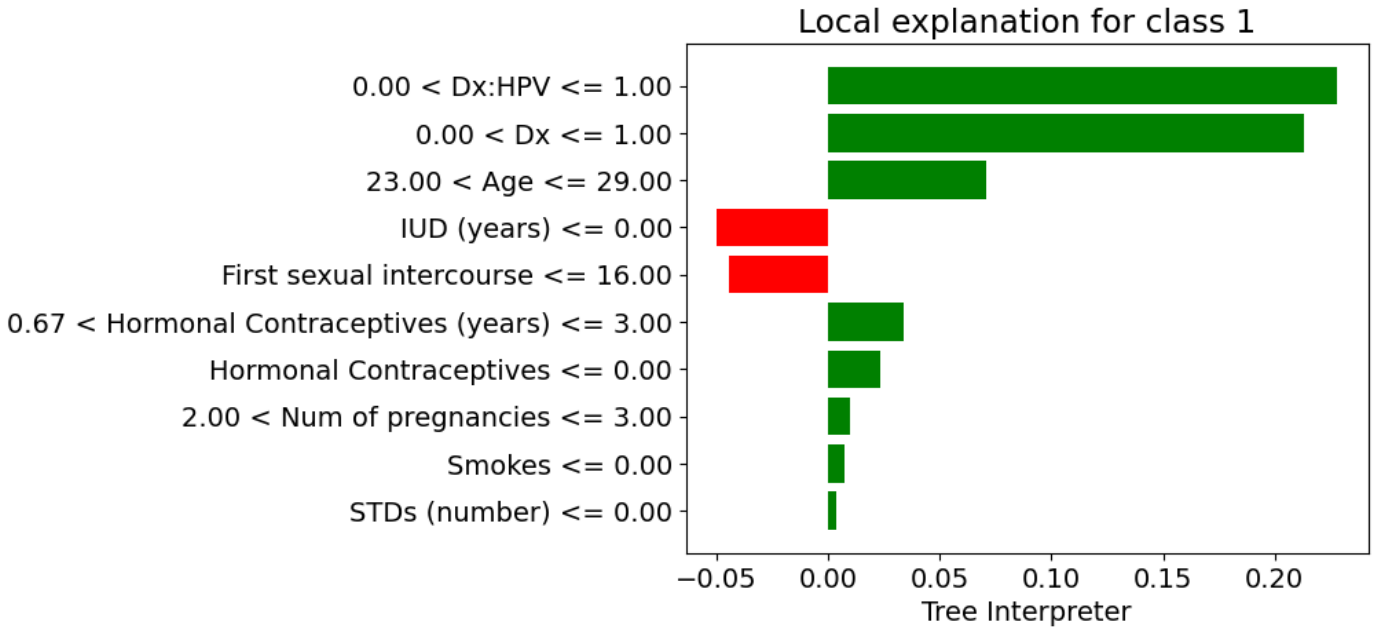






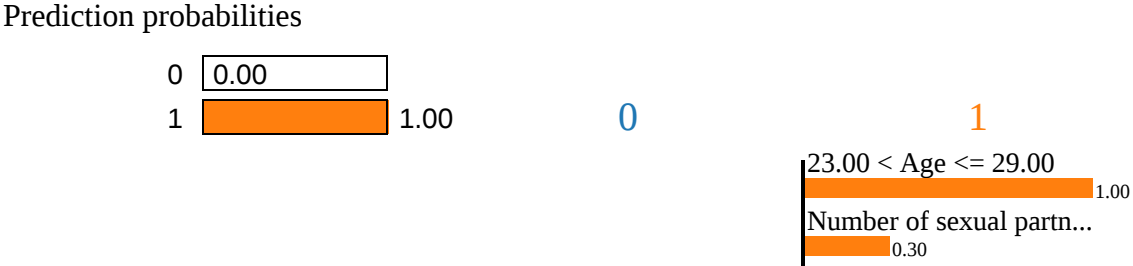
In [937…

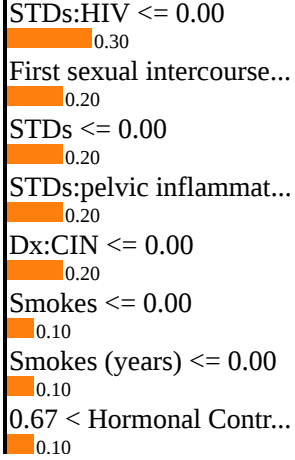
```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Tree Interpreter")
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'ti'+str(instance)+str(f
fig.savefig(''+str(var)+'ti'+str(instance)+str(f)+'.png', bbox_inches='tight')
```



In [938…

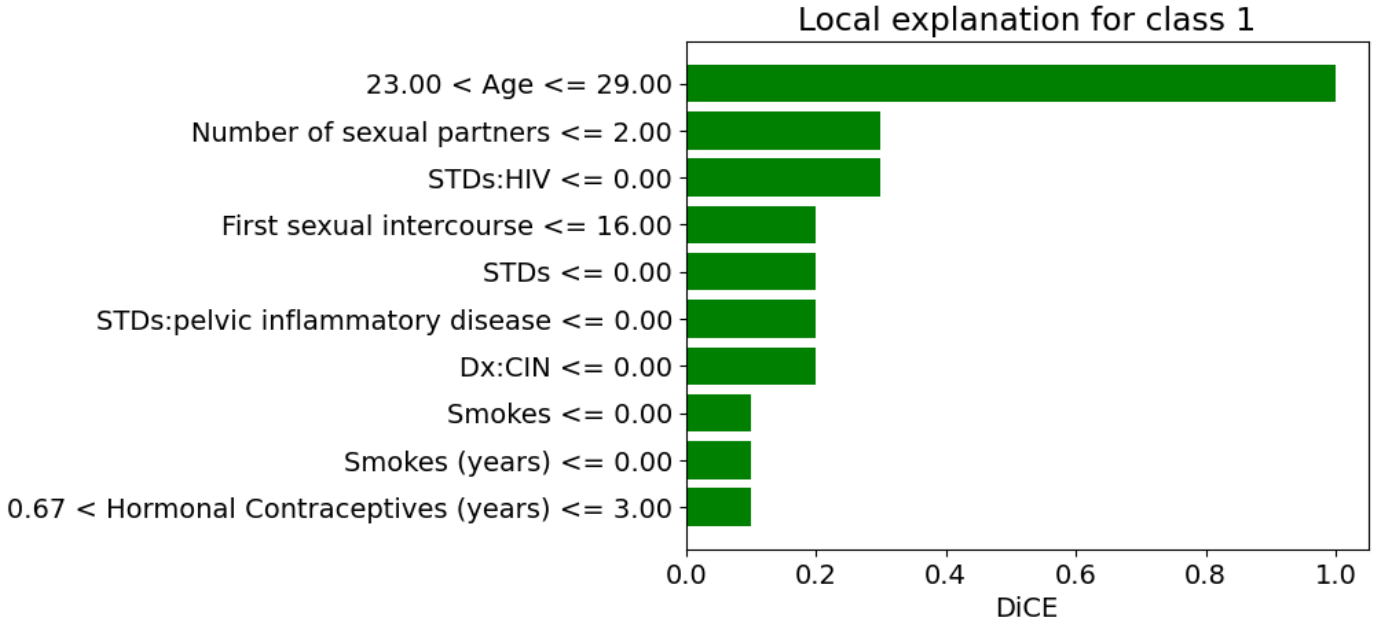
```
items = dicecontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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 [939…

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



ROAR

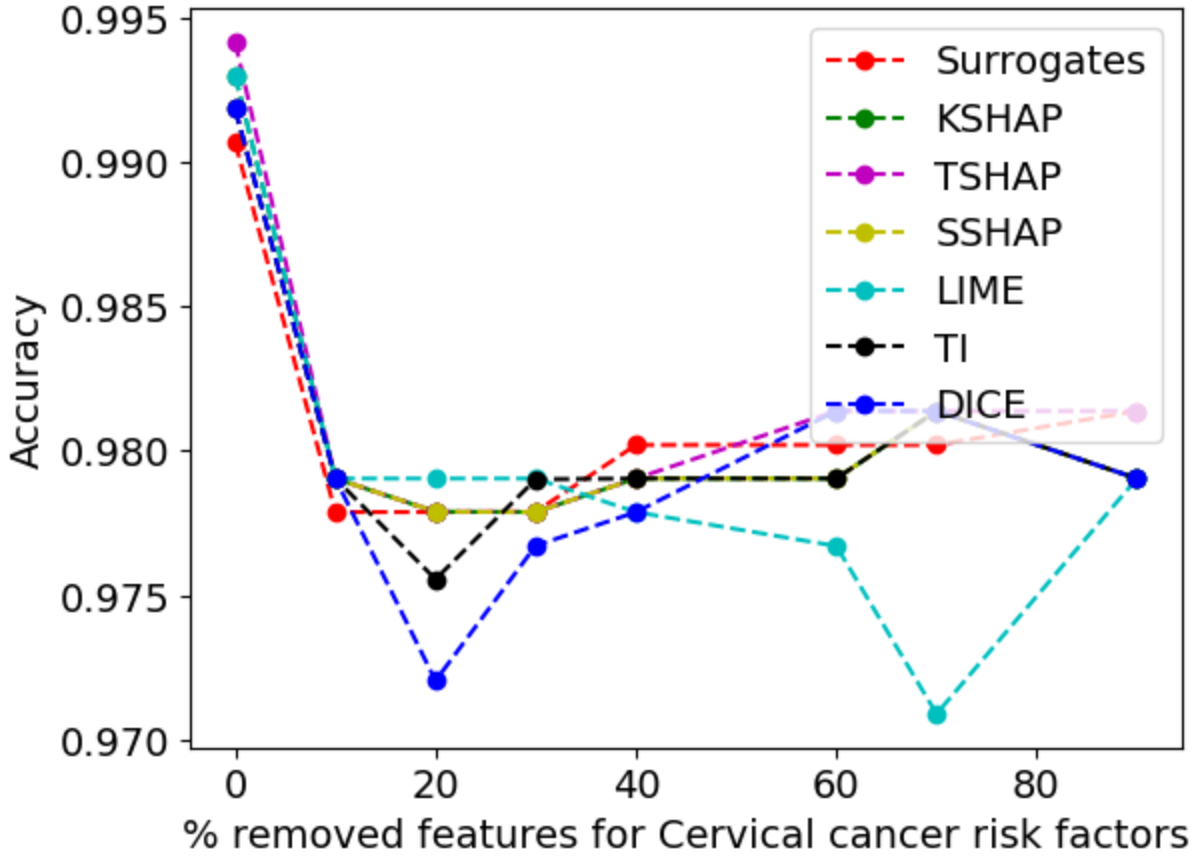
In [940…

```
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)

  #print(featImp, feature_to_predict, datapath,savepath,dataname)


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

      fi= featImp.iloc[k,:]
      #print(i,k,fi)
      fi.drop('Method', inplace=True)
      fi=fi.to_dict()
```

```
      fi=dict(sorted(fi.items(), key=lambda x:x[1], reverse=True))
      #print(fi)

      fii=list(fi.keys())
      #print(fii)

      df= pd.read_csv(datapath)

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

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

      df[df.columns] = df[df.columns].apply(pd.to_numeric, errors="coerce")
      df[df.columns].fillna(np.nan, inplace=True)
      imp = SimpleImputer(strategy="median")
      temp = imp.fit_transform(df)
      df = pd.DataFrame(temp, columns=list(df.columns))
      X=np.array(df[list(df.columns.drop(feature_to_predict))])
      #print("X",X.shape)
      #print("2",X)
      y=np.array(df[feature_to_predict])
      #print("Y",y.shape)

      model = RandomForestClassifier(random_state = 42)

      #print(X.shape,X,y.shape,y)
      scores = cross_val_score(model, X, y, cv=10)
      #print("Scores",scores)
      accuracies.append(np.mean(scores))
      #print("Acc",accuracies)

    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 [941…

```
#datapath='/content/drive/My Drive/dataXAI/cancer/cancer.csv'
#savepath= '/content/drive/My Drive/dataXAI/cancer/'
datapath='risk_factors_cervical_cancer.csv'
savepath= ''
dataname='Dx:Cancer'

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



In [942…
```
all_fi
```

Out[942]:

	Number of sexual partners	Hormonal Contraceptives (years)	Hormonal Contraceptives	Num of pregnancies	STDs:HIV	STDs:Hepatitis B	Citology	Smokes	ST cond
0	0.027750	0.418397	0.192809	0.509277	0.006710	0.000000	0.020115	0.261429	
1	0.006702	0.022344	0.026817	0.006680	0.000232	0.000151	0.001087	0.009873	
2	0.007266	0.023401	0.024398	0.006831	0.000071	0.000000	0.001121	0.010138	
3	0.006961	0.022381	0.026880	0.006739	0.000145	0.000000	0.001204	0.009667	
4	0.006638	0.015181	0.045952	0.007093	0.013710	0.028813	0.006688	0.077708	
5	0.009140	0.032471	0.012949	0.008784	0.000121	0.000000	0.003350	0.012953	
6	0.058631	0.081250	0.037500	0.063988	0.025893	0.025893	0.027083	0.054167	

7 rows × 36 columns

SHAPASH

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

In [943…
```
cns=Consistency()
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test)
```

```
INFO: Shap explainer type - <shap.explainers._tree.TreeExplainer object at 0x3482e3430>
```

Contribution plots

In [944…
```
img=xpl.plot.contribution_plot(0)
```

```
img
```

In [945…
```
#img.write_image('contribage.png')
```

In [946…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
```

In [947…
```
img=xpl.plot.contribution_plot(0)
img
```

In [948…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagekernel.png')
```

In [949…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [950…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagesampling.png')
```

In [951…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=trecontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [952…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagetree.png')
```

In [953…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [954…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagelime.png')
```

In [955…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=ticontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [956…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/contribageti.png')
```

In [957…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [958…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/contribagegs.png')
```

In [959…
```
#xpl = SmartExplainer(model=model)
#xpl.compile(x=X_test,contributions=dicecontrib)
#xpl.plot.contribution_plot(0)
#img
```

Contributions plot

In [960…
```
#fig_image=xpl.plot.contribution_plot(0)
#plt.savefig('age.png')
```

In [961…
```
#fig_image=xpl.plot.contribution_plot(0)
#plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/age.png')
```

In [962…
```
#fig_image=xpl.plot.contribution_plot(29)
#plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/dxhpv.png')
```

In [963…
```
#fig_image=xpl.plot.contribution_plot(30)
#plt.savefig('/content/drive/My Drive/dataXAI/cancer/sonali/dxhpv.png')
```

Consistency

In [964…
```
pairwise_consistency=cns.calculate_all_distances(methods, weights)
```

In [965…
```
test=pairwise_consistency[1].round(2)
```

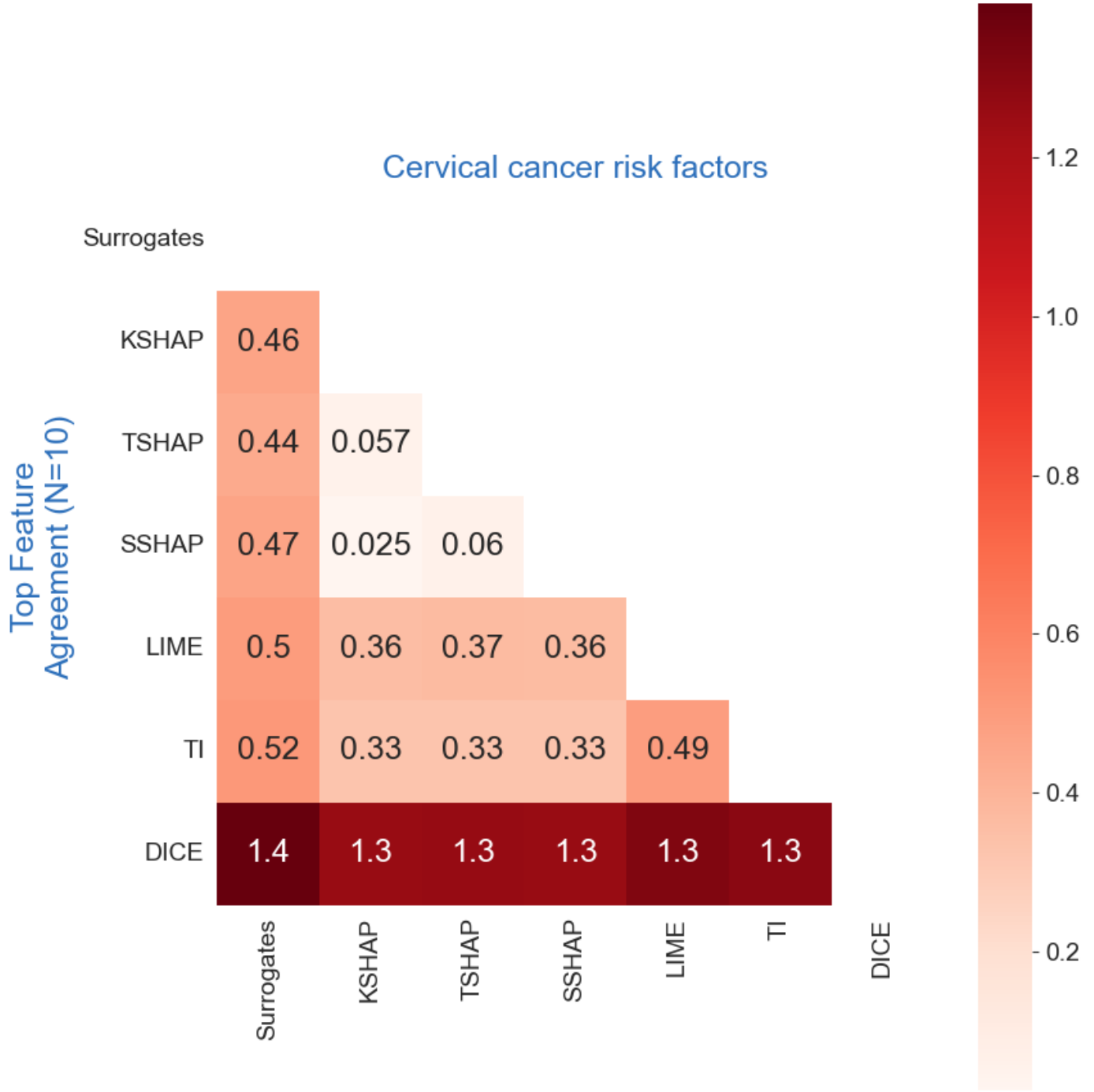
In [966…
```
test.style.background_gradient(cmap='Paired_r')
```

Out[966]:

	Surrogates	KSHAP	TSHAP	SSHAP	LIME	TI	DICE
Surrogates	0.000000	0.460000	0.440000	0.470000	0.500000	0.520000	1.390000
KSHAP	0.460000	0.000000	0.060000	0.020000	0.360000	0.330000	1.260000
TSHAP	0.440000	0.060000	0.000000	0.060000	0.370000	0.330000	1.270000
SSHAP	0.470000	0.020000	0.060000	0.000000	0.360000	0.330000	1.260000
LIME	0.500000	0.360000	0.370000	0.360000	0.000000	0.490000	1.320000
TI	0.520000	0.330000	0.330000	0.330000	0.490000	0.000000	1.290000
DICE	1.390000	1.260000	1.270000	1.260000	1.320000	1.290000	0.000000

In [967…
```
with open('consistency.tex','w') as tf:
    tf.write(test.to_latex())
```

In [968…
```
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 [969…
```
fig.savefig('consistency.png', bbox_inches='tight', dpi=300)
```

Mean Consistency

In [970…
```
for i in pairwise_consistency[1].columns:
  print(i, round(np.mean(pairwise_consistency[1][i]),2))
```

```
Surrogates 0.54
KSHAP 0.36
TSHAP 0.36
SSHAP 0.36
LIME 0.49
TI 0.47
DICE 1.11
```

Compactness

In [971…
```
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
    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_featu
    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 en
    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 [972…

```
compacities=[]

for weight in weights:
  rr=compute_features_compacity(case="classification", contributions=weight, selection=l
  #rr=compute_features_compacity(case="classification", contributions=weight, selection=
  compacities.append(pd.DataFrame.from_dict(rr))
```

In [973…

```
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[973]:

Method	features_needed	distance_reached
Surrogates	2.0	1.00
KSHAP	1.0	0.16
TSHAP	1.0	0.17
SSHAP	1.0	0.16
LIME	1.0	0.53
TI	1.0	0.25
DICE	11.0	1.00

In [974…
```
with open('compactness'+str(instance)+'.tex','w') as tf:
    tf.write(compacity.to_latex())
```

In [975…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
xpl.plot.compacity_plot
```

Out[975]:
```
<bound method SmartPlotter.compacity_plot of <shapash.explainer.smart_plotter.SmartPlot
ter object at 0x33dc9bc40>>
```

Plots

In [976…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
img=xpl.plot.compacity_plot()
```

In [977…
```
img
```

In [978…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactgs.png')
```

In [979…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
img=xpl.plot.compacity_plot()
img
```

In [980…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactkernel.png')
```

In [981…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=trecontrib)
img=xpl.plot.compacity_plot()
img
```

In [982…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compacttree.png')
```

In [983…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
```

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

In [984…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactsampling.png
```

In [985…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img=xpl.plot.compacity_plot()
img
```

In [986…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactlime.png')
```

In [987…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=ticontrib)
img=xpl.plot.compacity_plot()
img
```

In [988…
```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/'+str(var)+'compactti.png')
```

In [989…
```
#xpl = SmartExplainer(model=model)
#xpl.compile(x=X_test,contributions=dicecontrib)
#xpl.plot.compacity_plot()
```

Stability

tool

In [990…
```
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
    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[j] == distance between actual instance and instance j
    """
    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_ve
            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)


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
    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 (#instance
        Each array has shape (#neighbors, #features) where #neighbors includes the insta
    """
    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
        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.
    elif mode == "classification":
        predictions = model.predict_proba(pd.DataFrame(all_neighbors[:, :-1], columns=da

    # Add prediction column
    all_neighbors = np.append(all_neighbors, predictions.reshape(all_neighbors.shape[0],
    # 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
    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 neighbor
    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 ou
    ind = pd.merge(x_encoded.reset_index(), pd.DataFrame(instance[:, :-2], columns=x_enc
        .set_index(x_encoded.index.name if x_encoded.index.name is not None else '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(axi
                             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
    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

        * For regression:

            * normalized distance between the output of current model and output of full
        * 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
    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_featu
```

```
    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) contrib
        of instance and corresponding neighbors.
        - If selection represents multiple instances, the method returns the average (no
        of instances and neighbors (=amplitude), as well as the variability of those val
        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", "variabili
        """
        #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,
            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[
            features_stability = {"variability": variability, "amplitude": amplitude}
            return features_stability
```

One instance

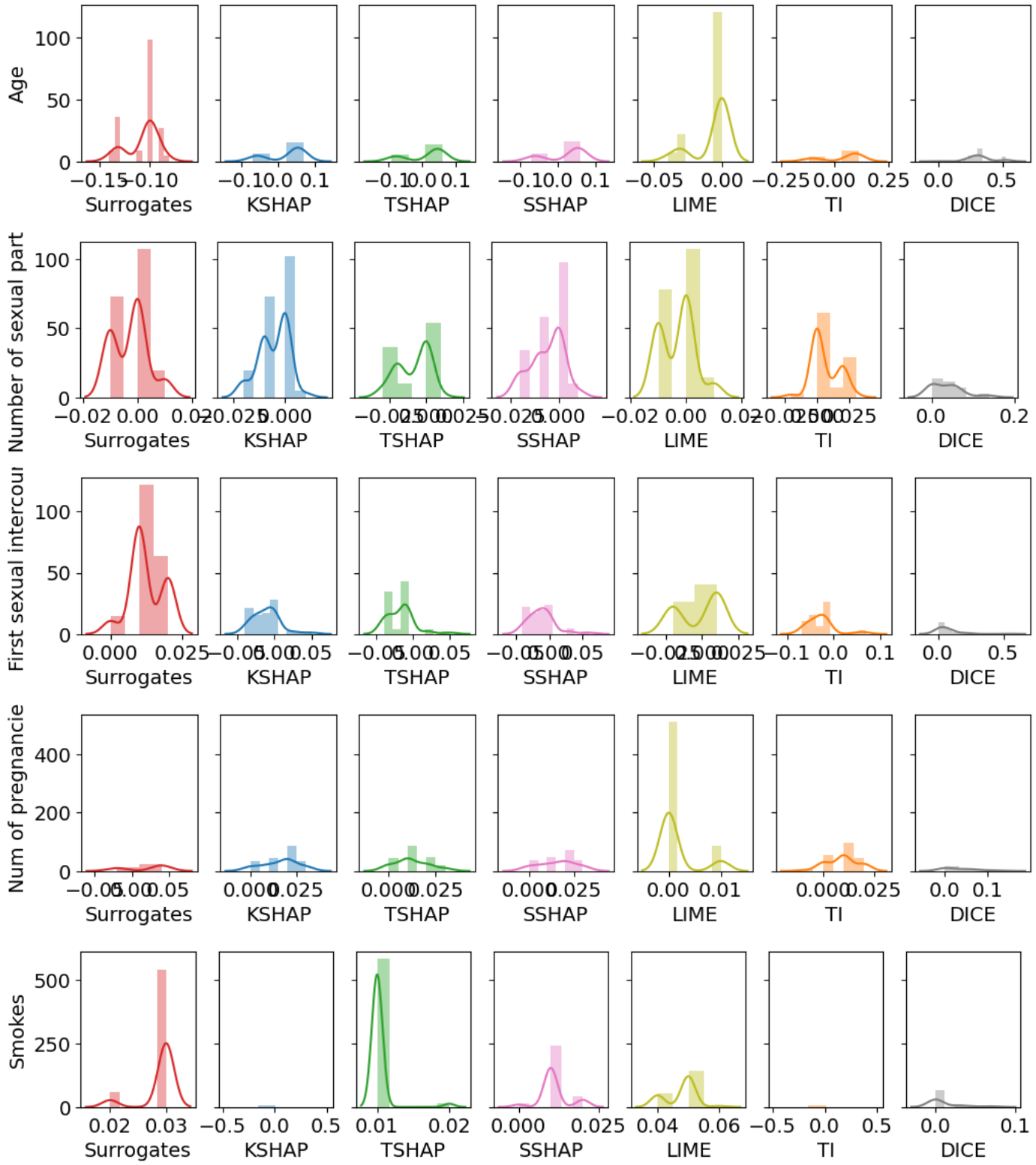
In [991…
```
features=list(X_test.columns)
```

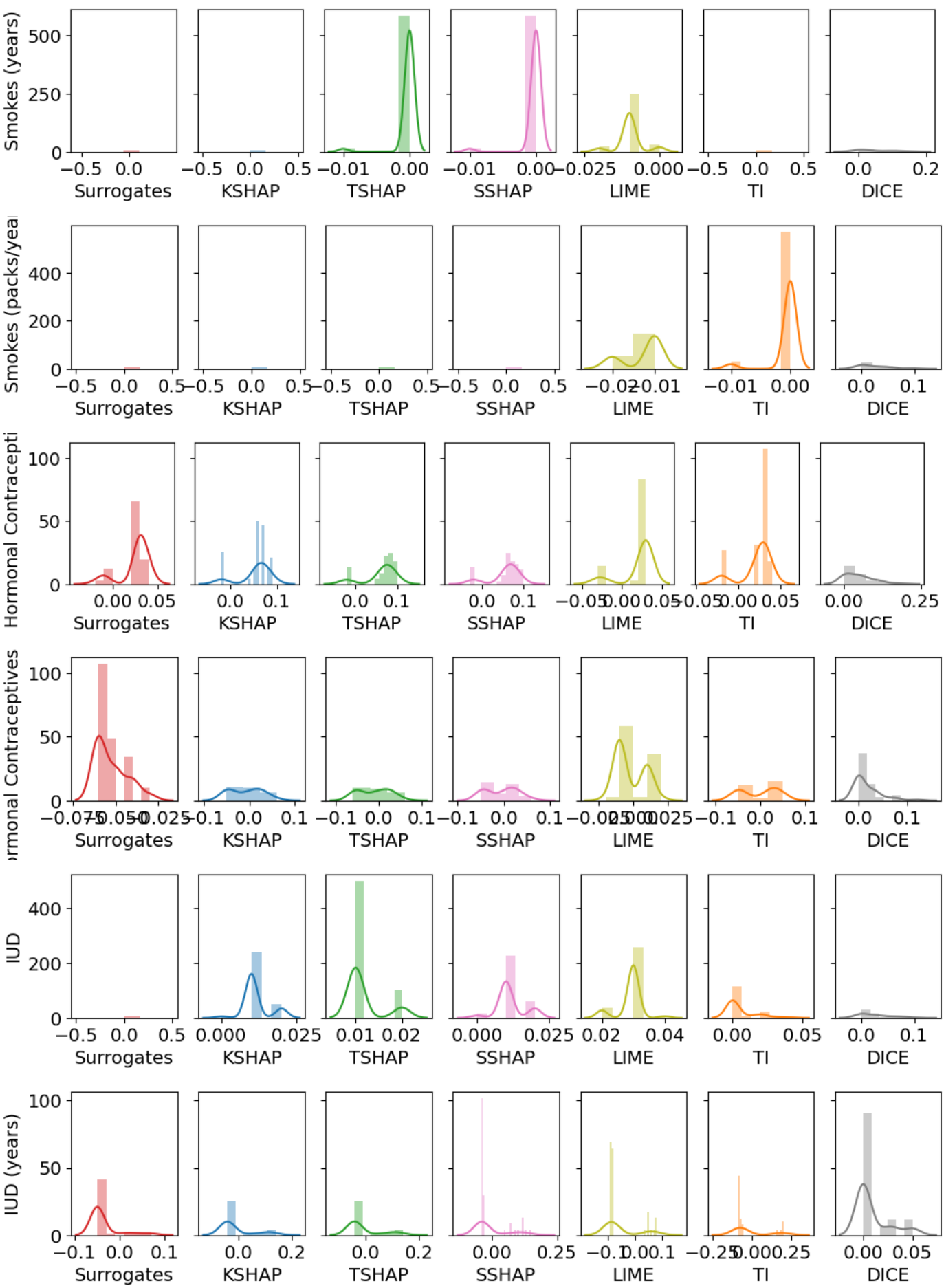
In [992…
```
#X_test.reset_index(inplace=True)
#X_test.drop('index', inplace=True, axis=1)
```

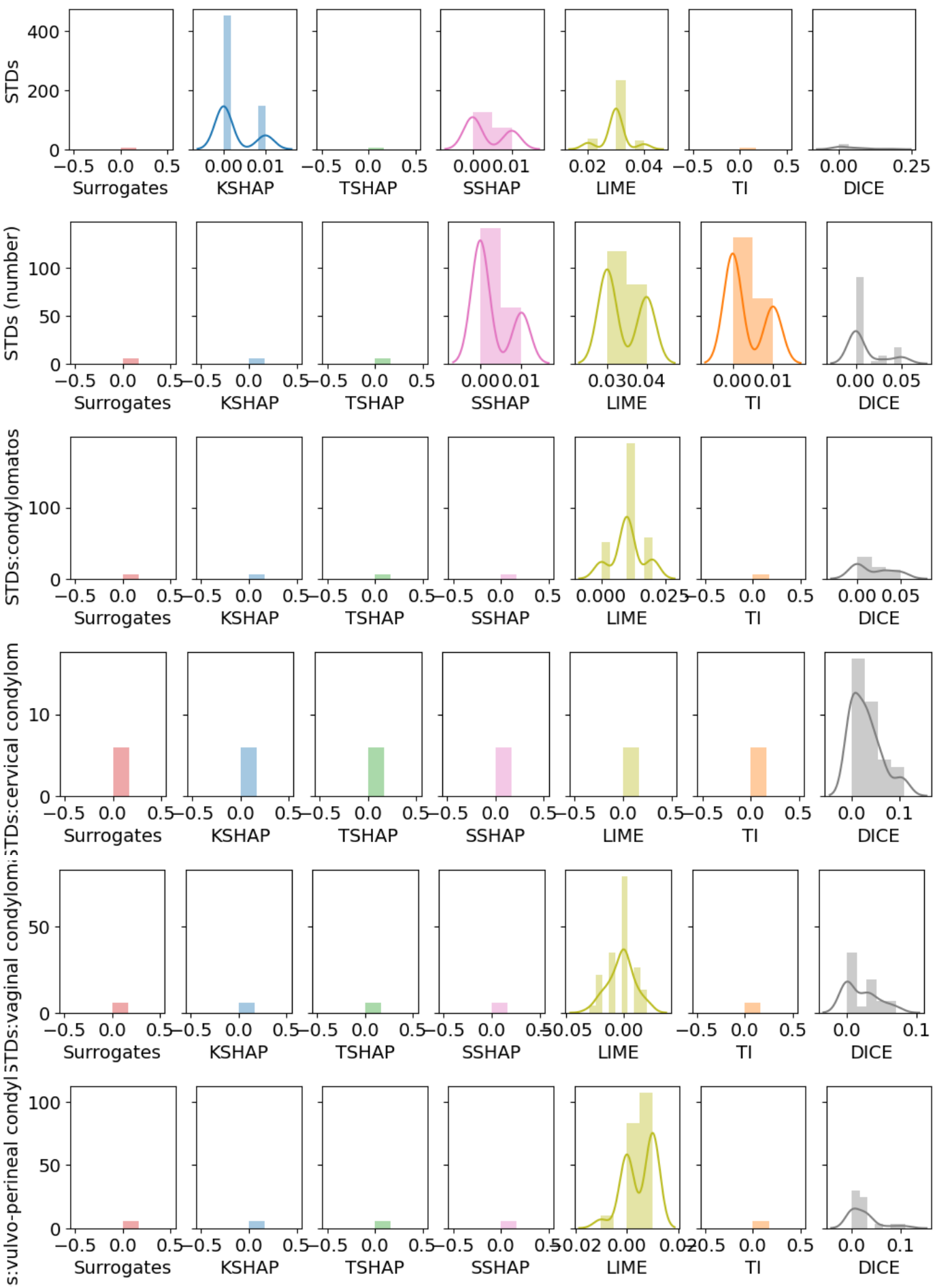
In [993…
```
#compute_features_compacity(case="classification", contributions=weight, selection=list(
frames=[]
for weight in weights:
  #fs= compute_features_stability (case="classification", x=X_test, selection=list(range
  fs= compute_features_stability (case="classification", x=X_test, selection=[instance],
  frames.append(fs)
```

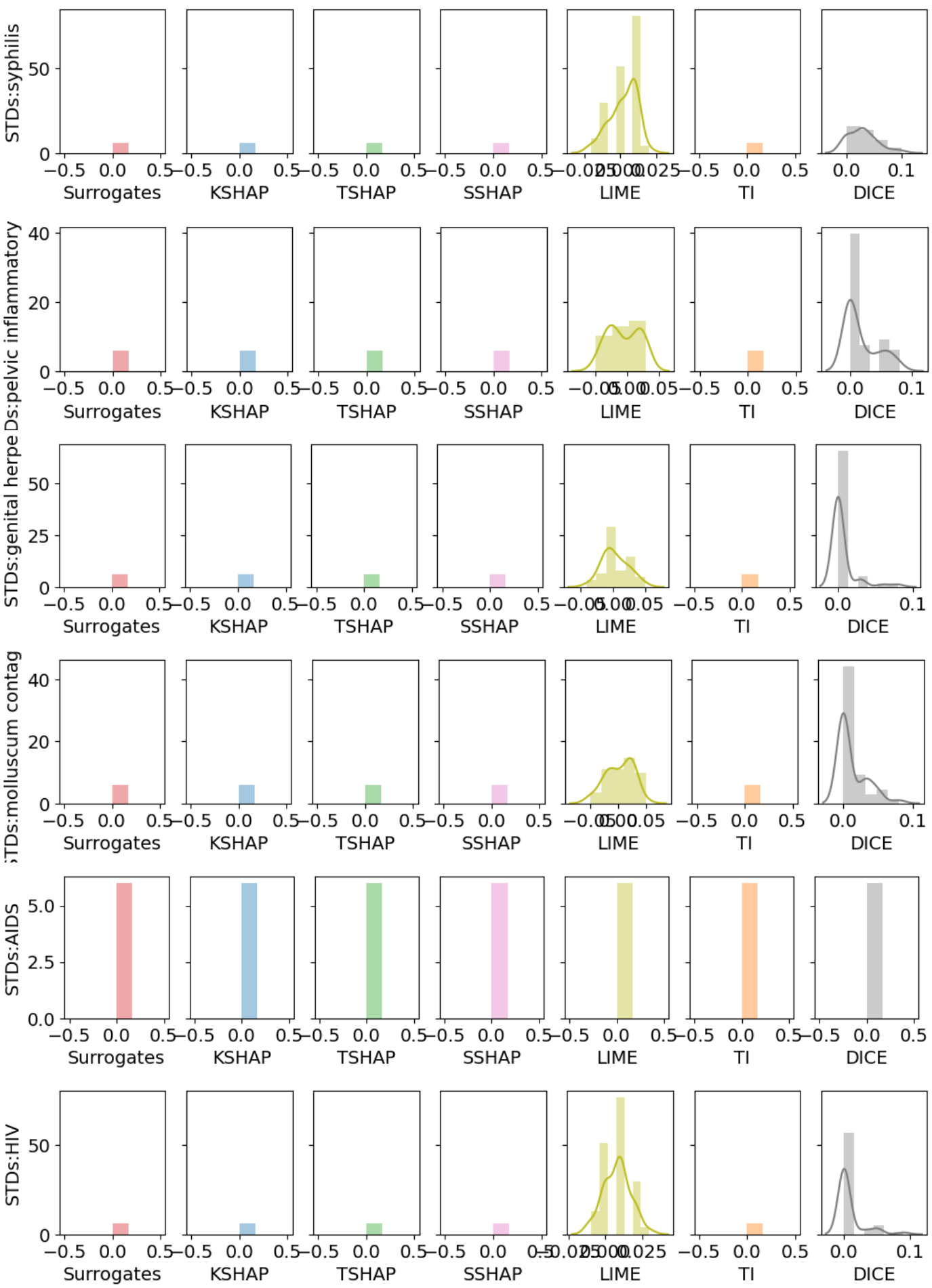
In [994…
```
colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orange',  '

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

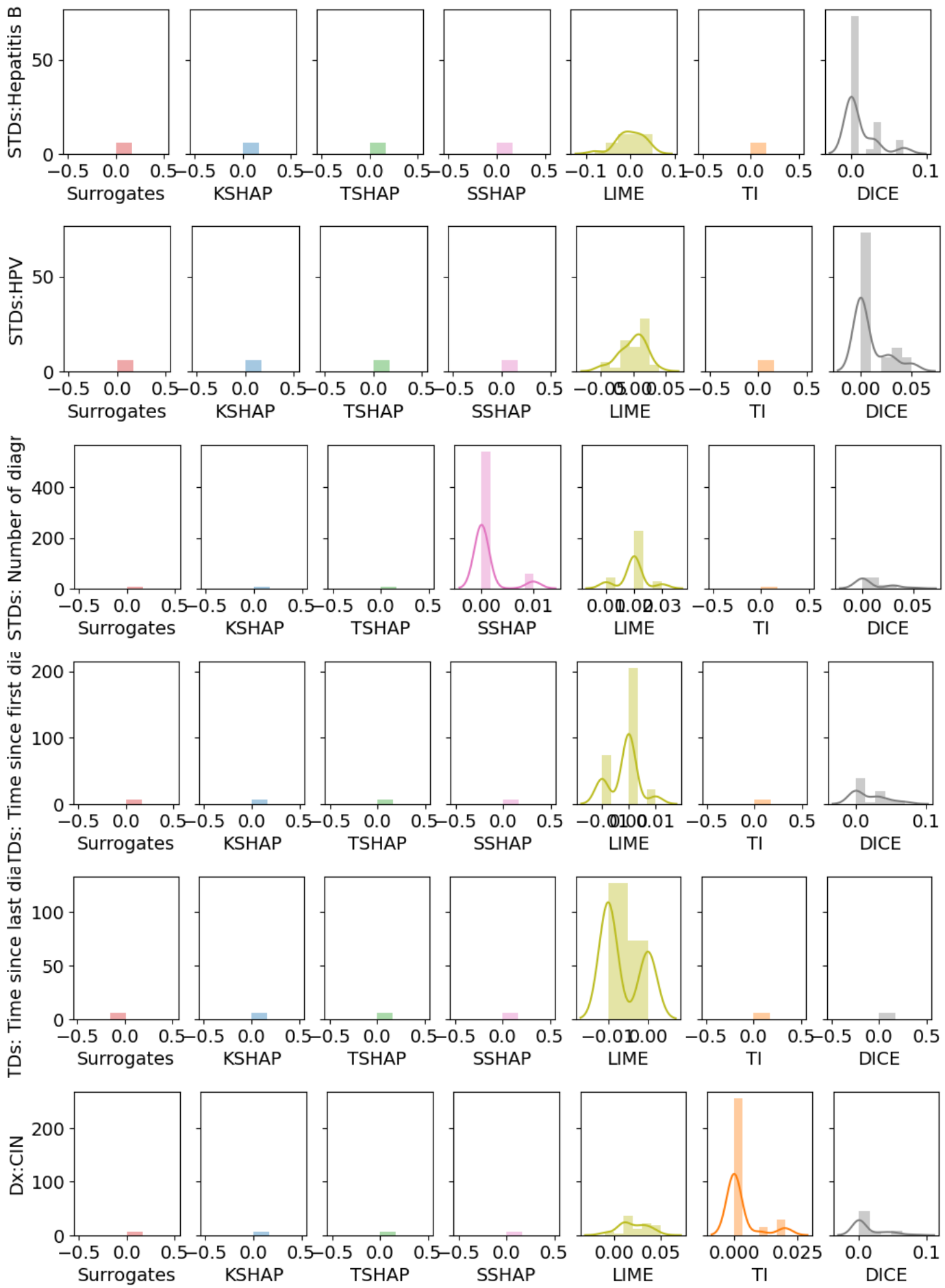
```
t=0

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(''+str(var)+str(instance)+str(features[j])[:10]+'.png', bbox_inches='tight
plt.show()
```

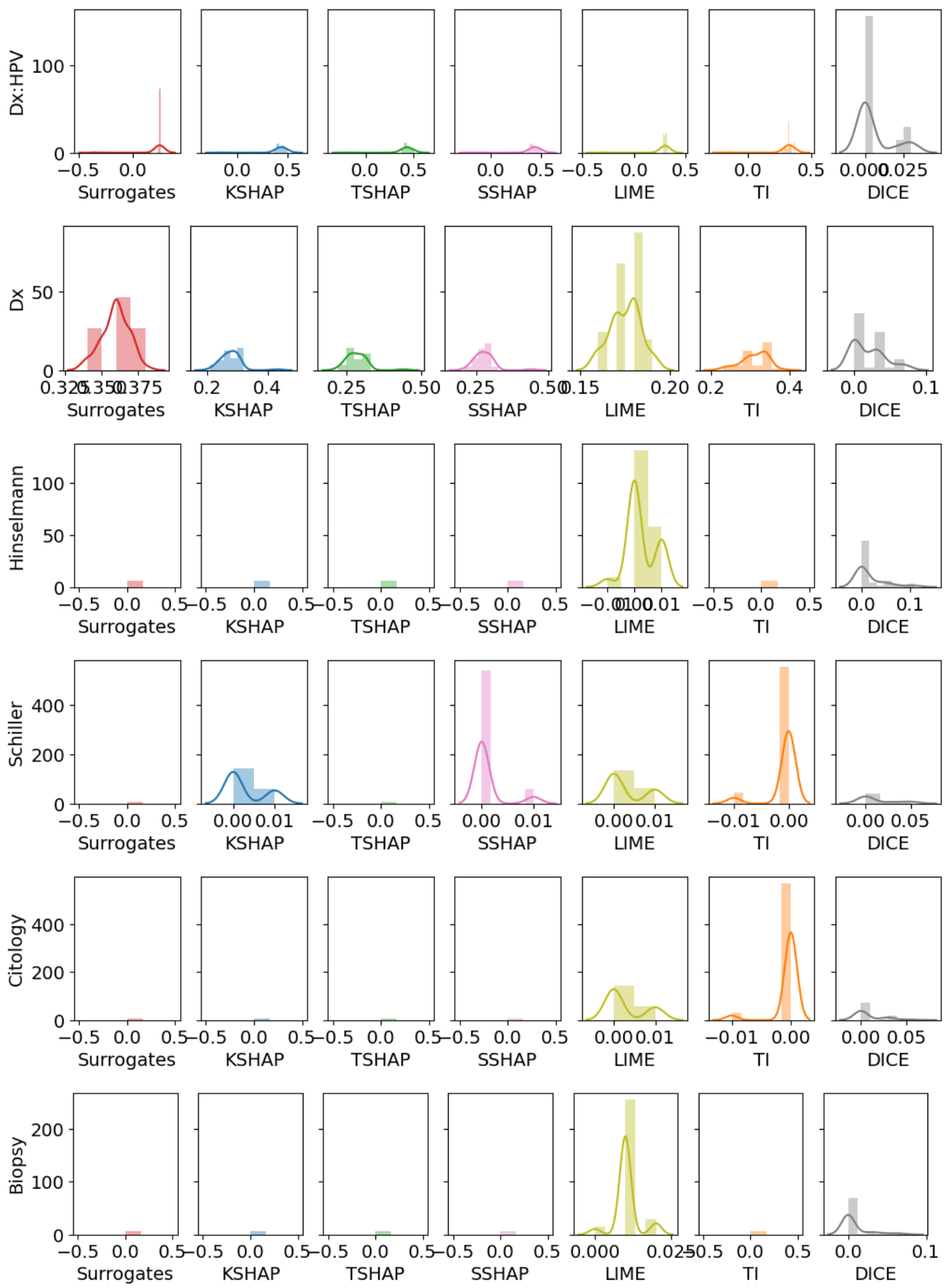



Smokes (years)
Smokes (packs/yea
Hormonal Contracepti
rmonal Contraceptives
IUD
IUD (years)
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE


STDs
STDs (number)
STDs:condylomatos
STDs:cervical condylom
STDs:vaginal condylom
s:vulvo-perineal condyl
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE


STDs:syphilis
STDs:pelvic inflammatory
STDs:genital herpe
STDs:molluscum contag
STDs:AIDS
STDs:HIV
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE


STDs:Hepatitis B
STDs:HPV
STDs: Number of diagr
TDs: Time since first dia
TDs: Time since last dia
Dx:CIN
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE



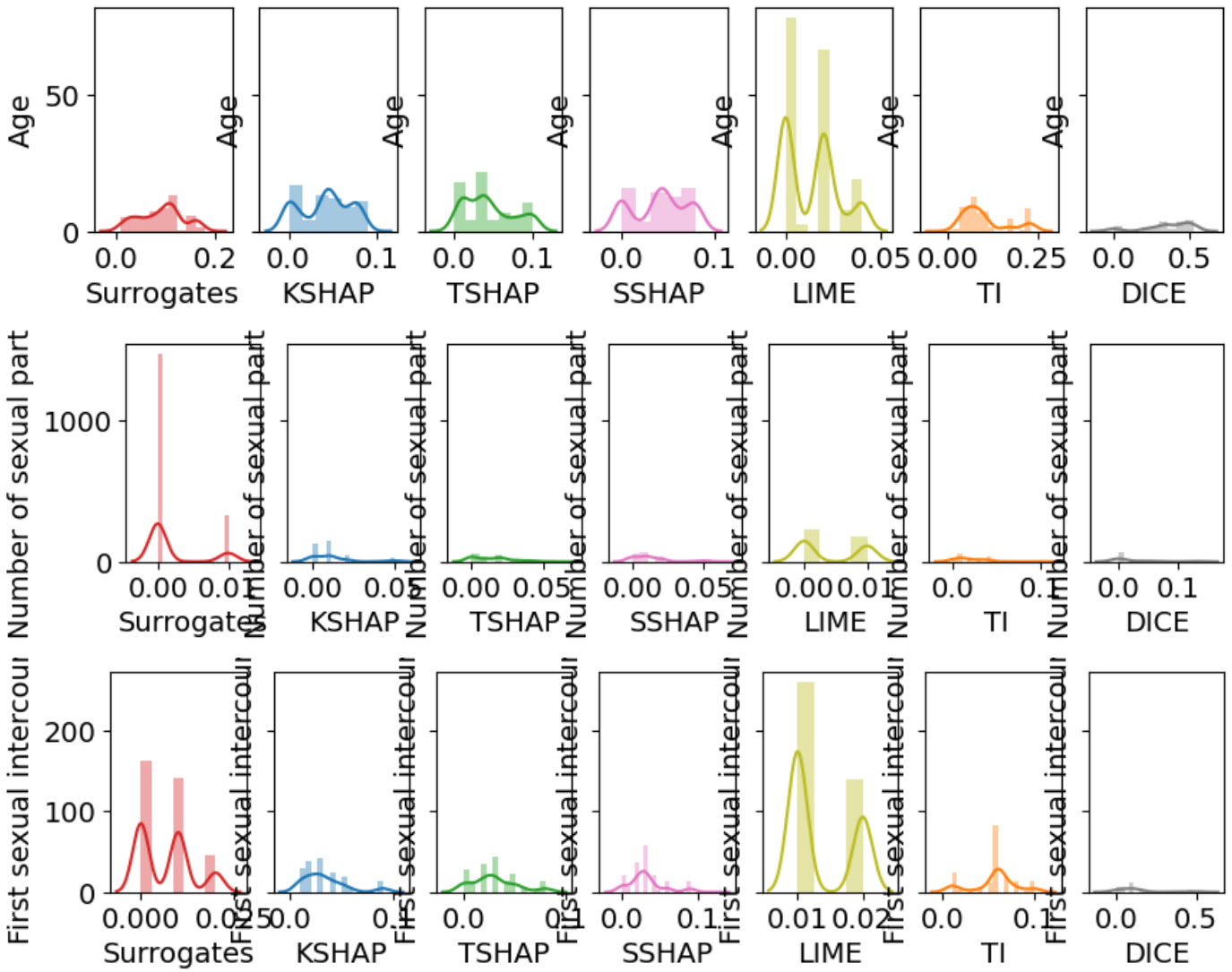
Multiple instances

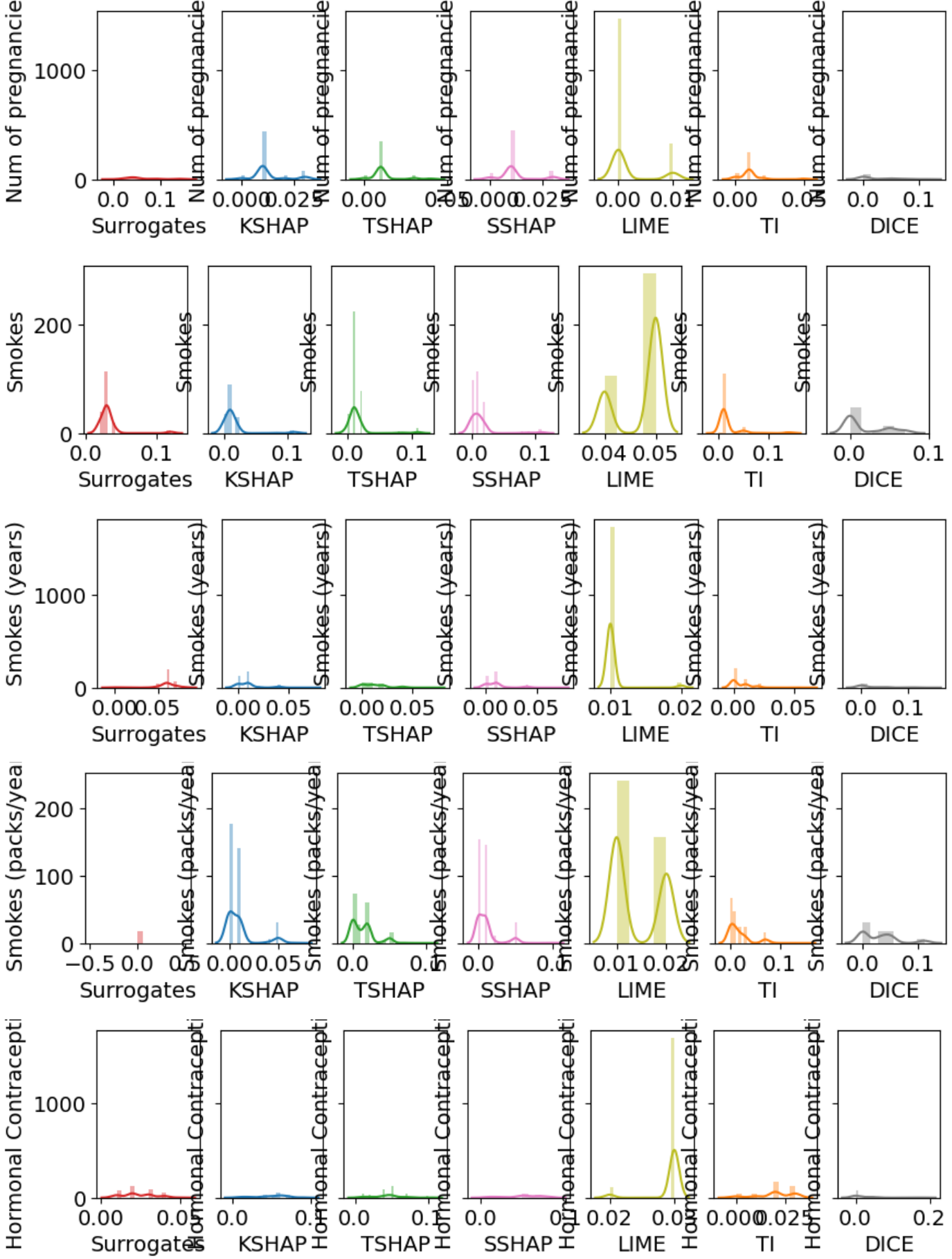
In [995…

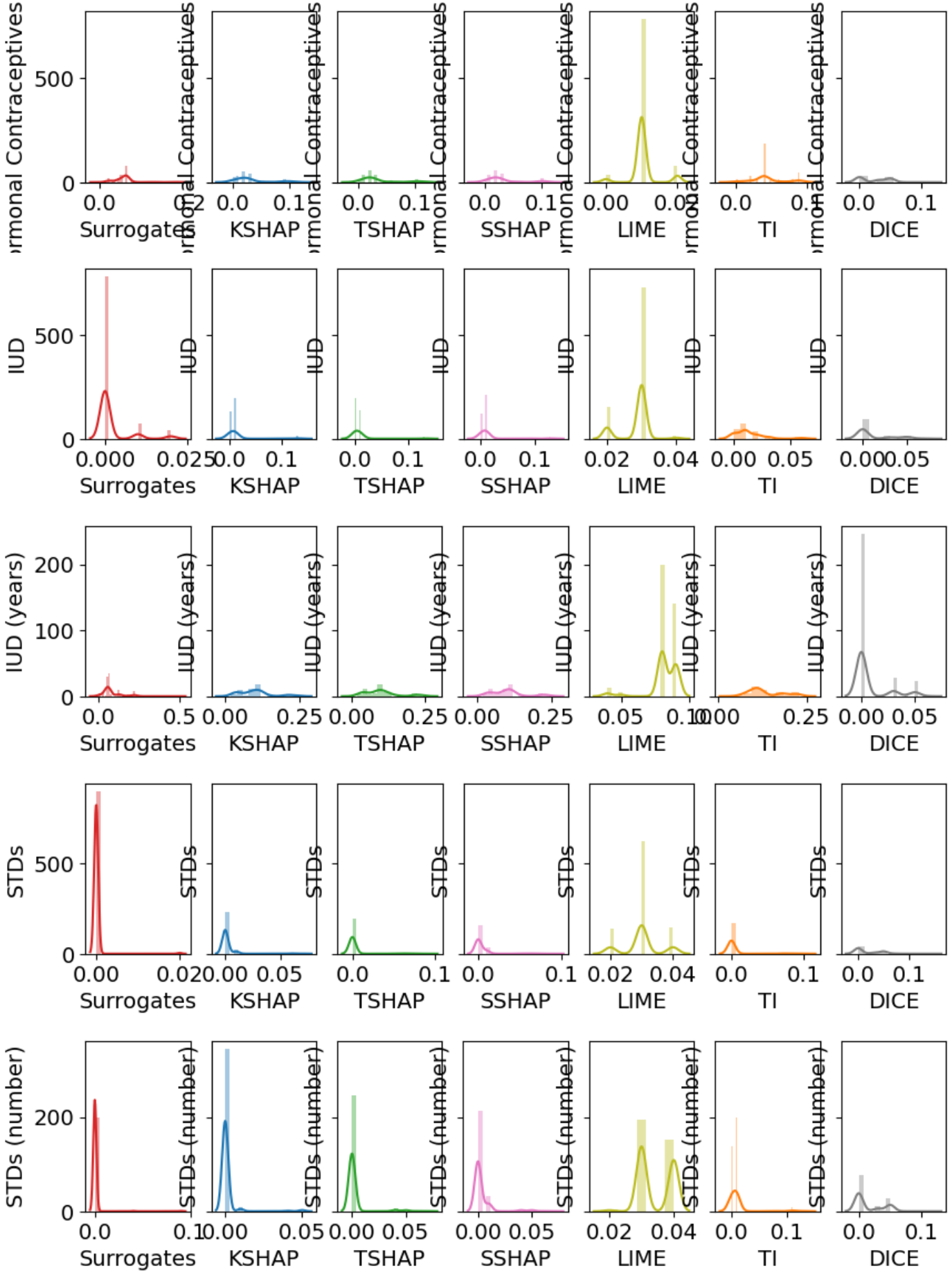
```
features=X_test.columns
```

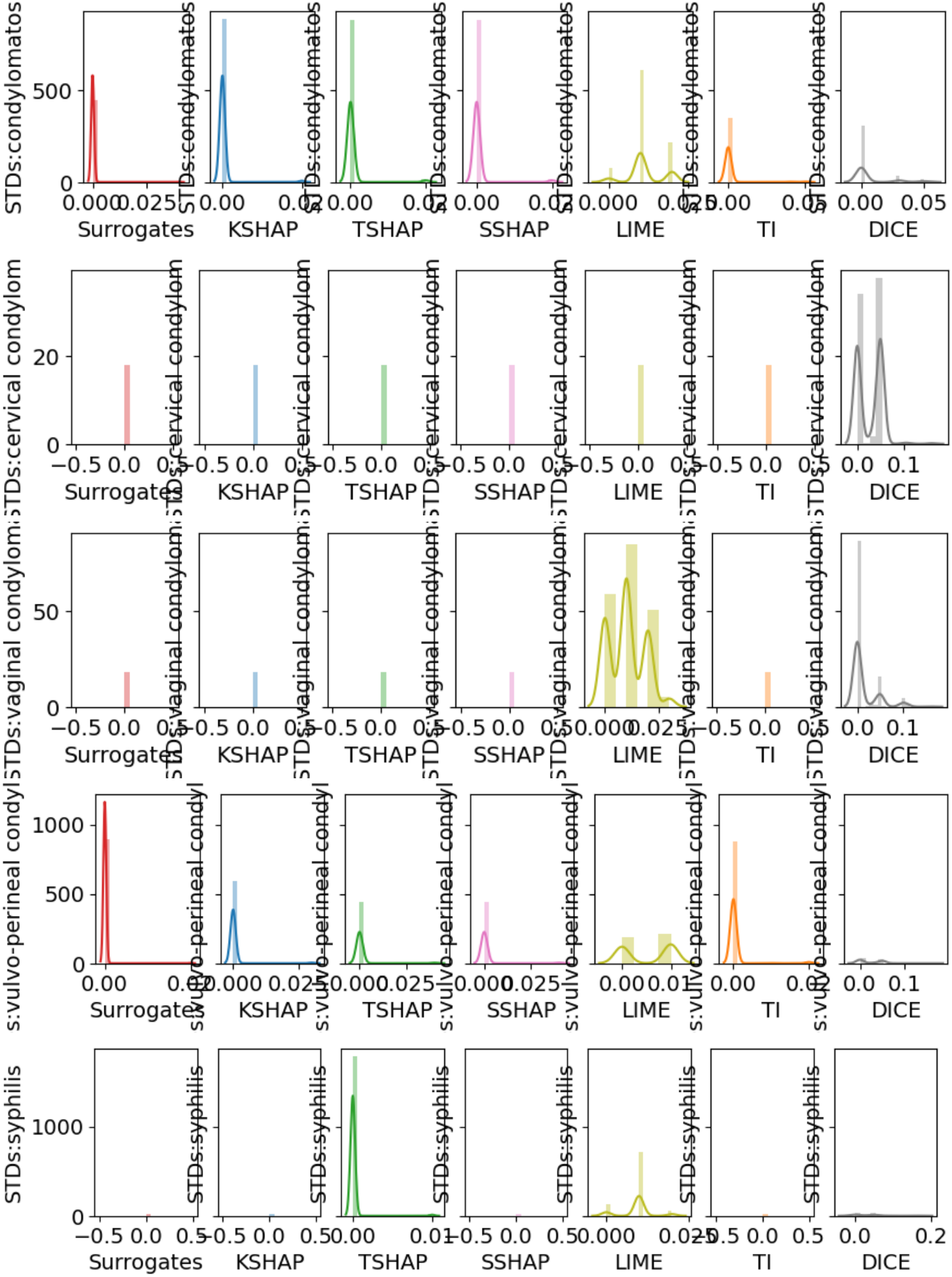
In [996…
```
#compute_features_compacity(case="classification", contributions=weight, selection=list(
frames=[]
for weight in weights:
  fs= compute_features_stability (case="classification", x=X_test, selection=list(range(
  #fs= compute_features_stability (case="classification", x=X_test, selection=[1,4], con
  frames.append(fs)
```

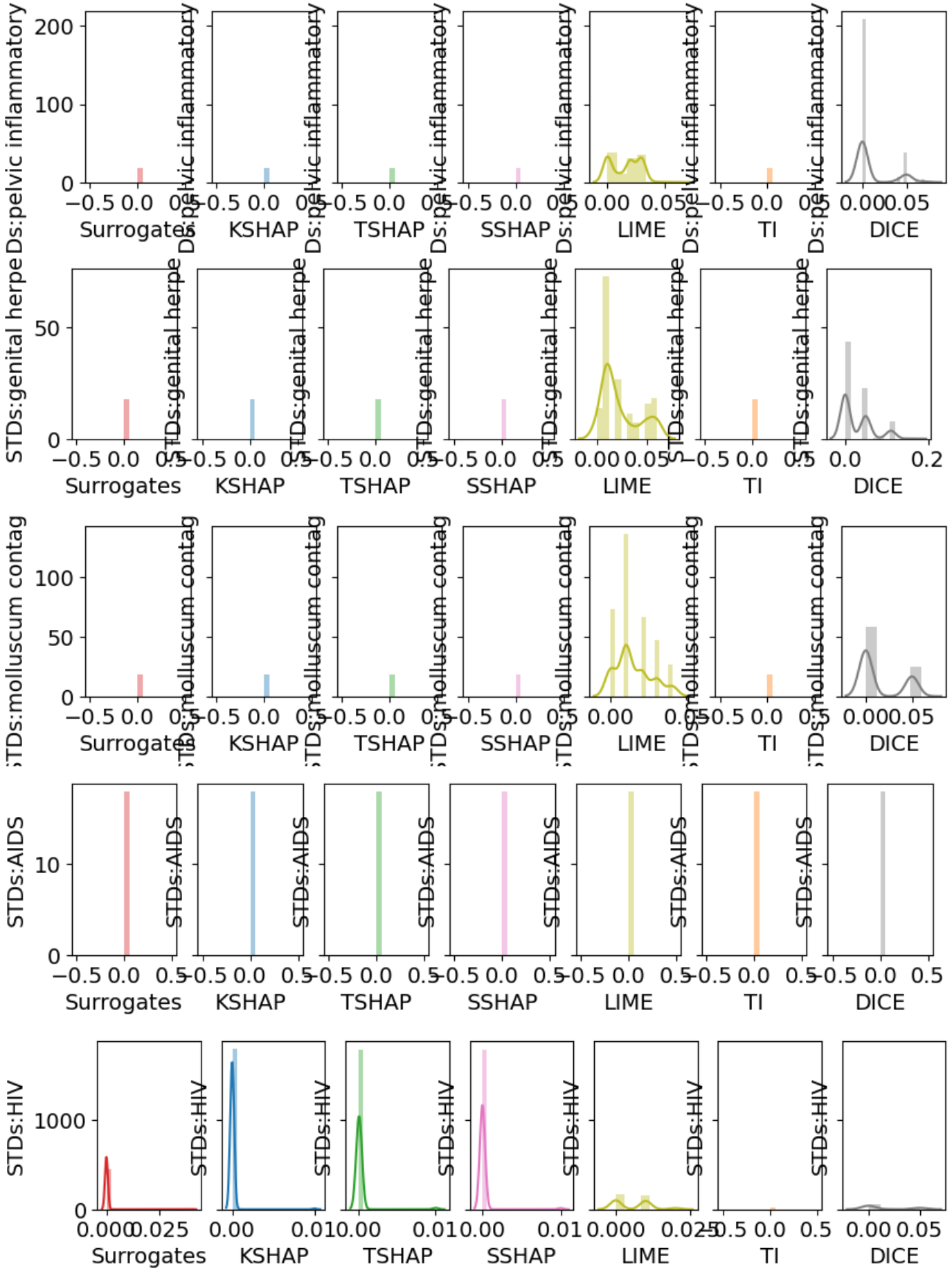
In [997…
```
colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orange',  '

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(''+str(var)+str(features[j])[:10]+'.png')
  plt.show()
```

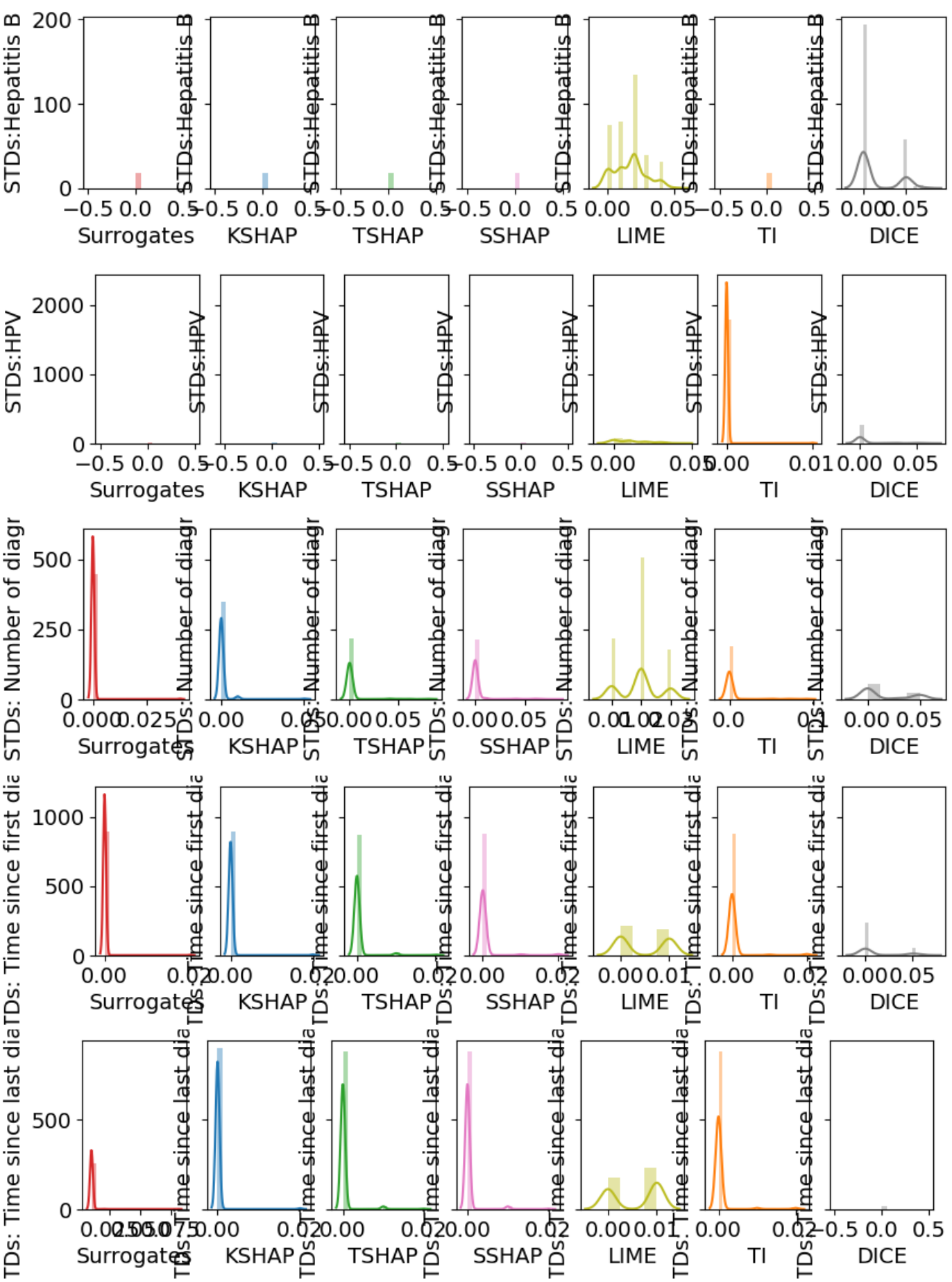


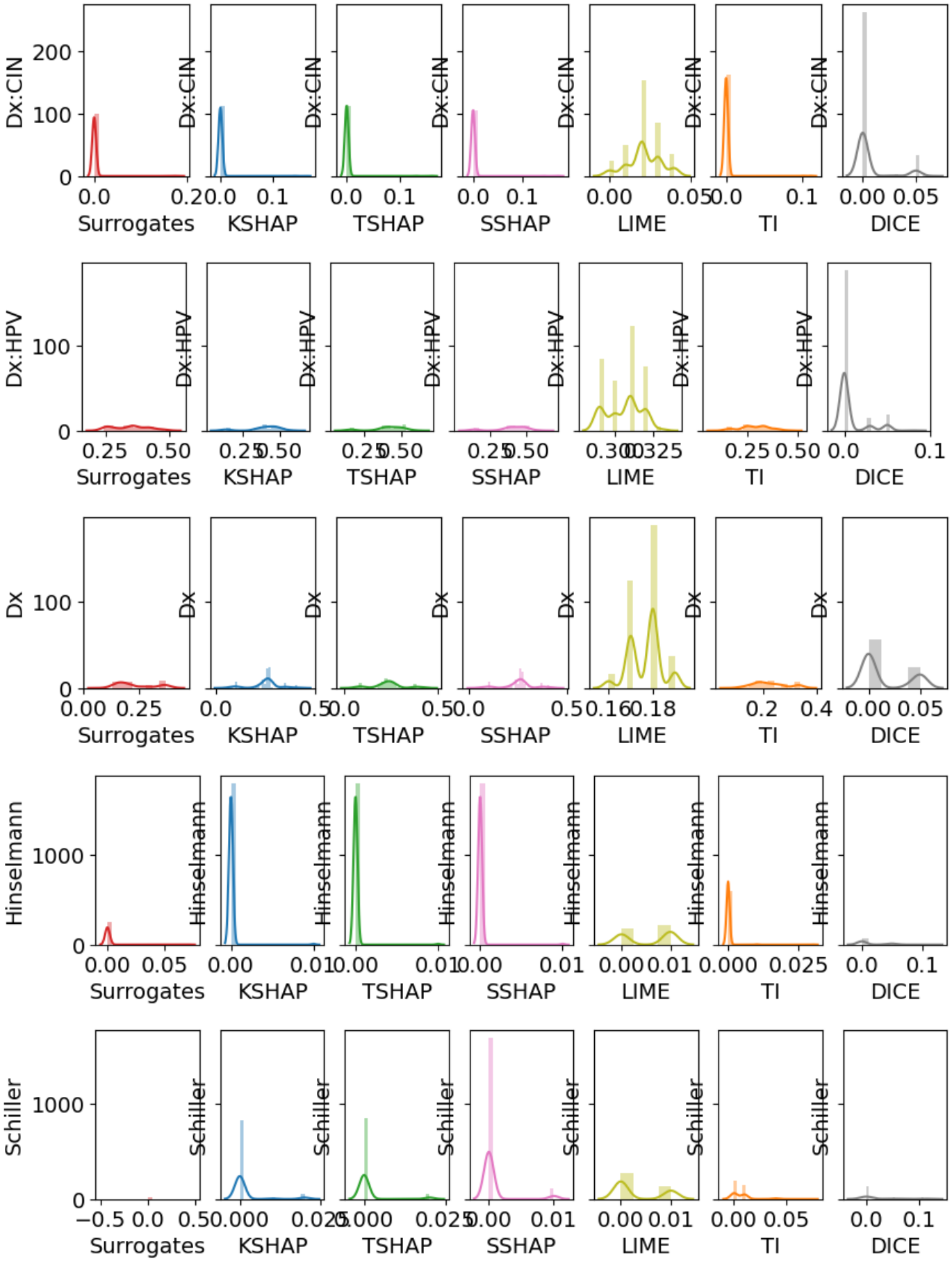

Num of pregnancies
Smokes
Smokes (years)
Smokes (packs/yea
Hormonal Contracepti
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE

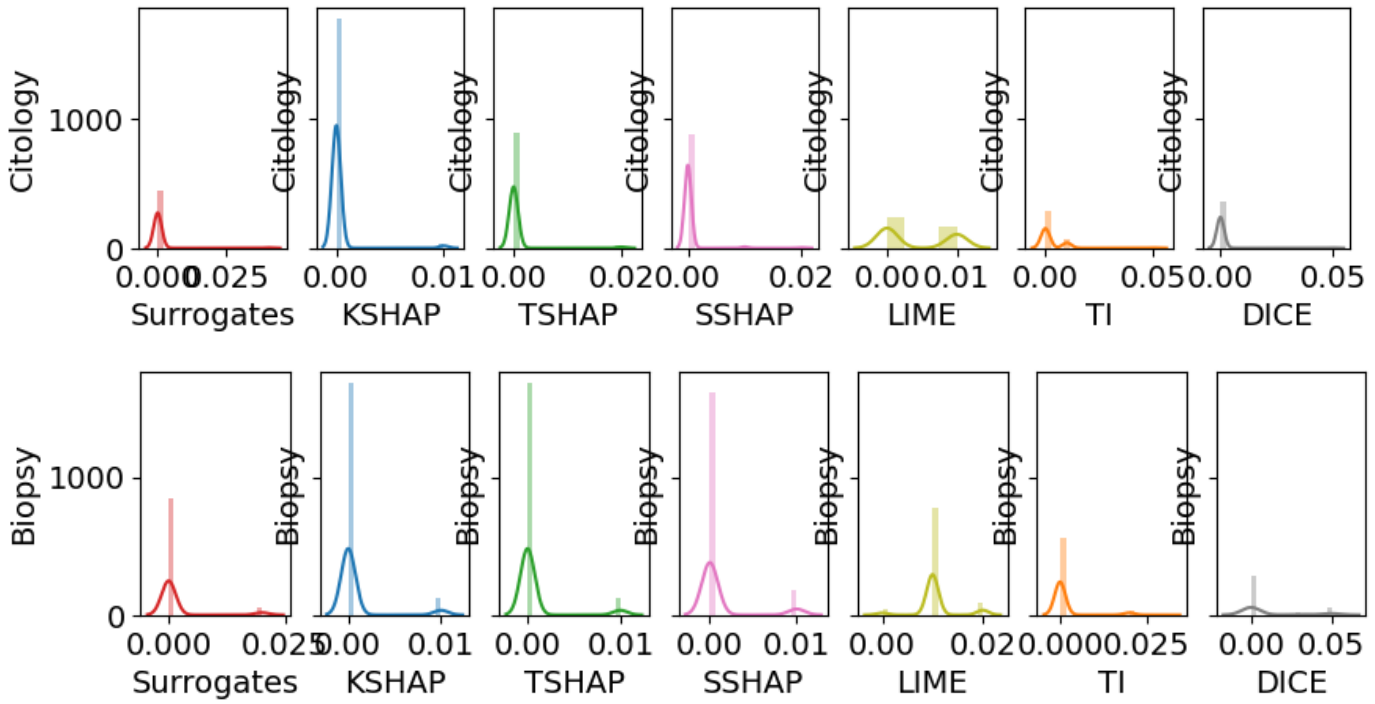

Hormonal Contraceptives
IUD
IUD (years)
STDs
STDs (number)
0
500
100
200
0.0
0.5
0.000
0.025
0.00
0.02
0.05
0.1
0.2
0.25
0.04
0.10
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE


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


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


STDs:Hepatitis B
STDs:HPV
STDs: Number of diagr
STDs: Time since first dia
STDs: Time since last dia
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE


Dx:CIN
Dx:HPV
Dx
Hinselmann
Schiller
Surrogates
KSHAP
TSHAP
SSHAP
LIME
TI
DICE



In [998…

```
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 0.23
Surrogates 0.28
KSHAP 1.4
KSHAP 0.25
TSHAP 0.4
TSHAP 0.07
SSHAP 0.93
SSHAP 0.18
LIME 0.67
LIME 0.02
TI 0.67
TI 0.26
DICE 1.73
DICE 0.21
```

Plots

In [999…

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

In [100…

```
#img.write_image('/content/drive/My Drive/dataXAI/cancer/compactgs.png')
```

In [100…

```
fig_image=xpl.plot.stability_plot()
#plt.xlabel("Local Surrogates")
plt.savefig('stabplot.png')
```

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

In [100…

```
#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 [100…

```
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
```

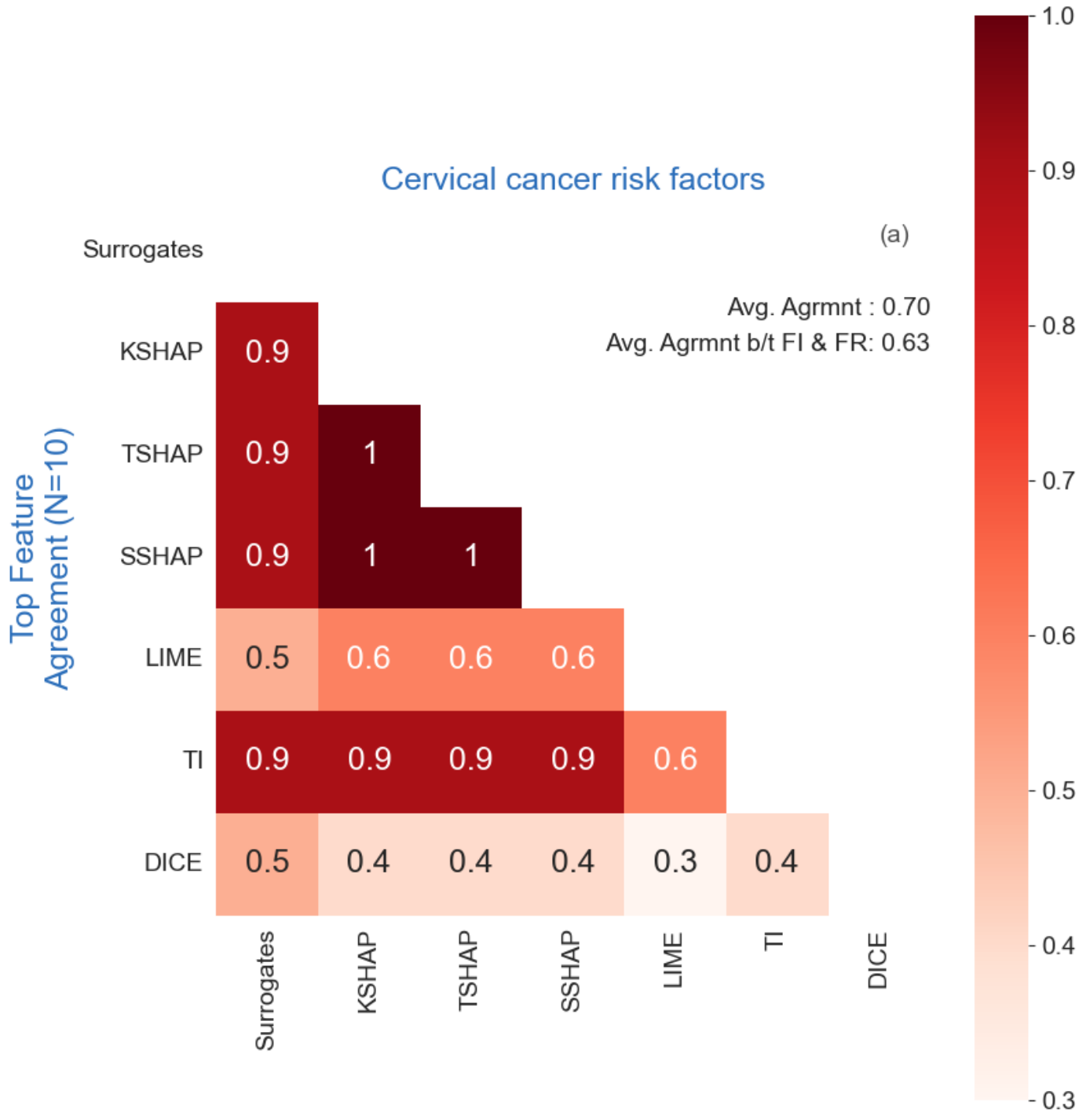
Plots

In [100…

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

In [100…

```
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()
```



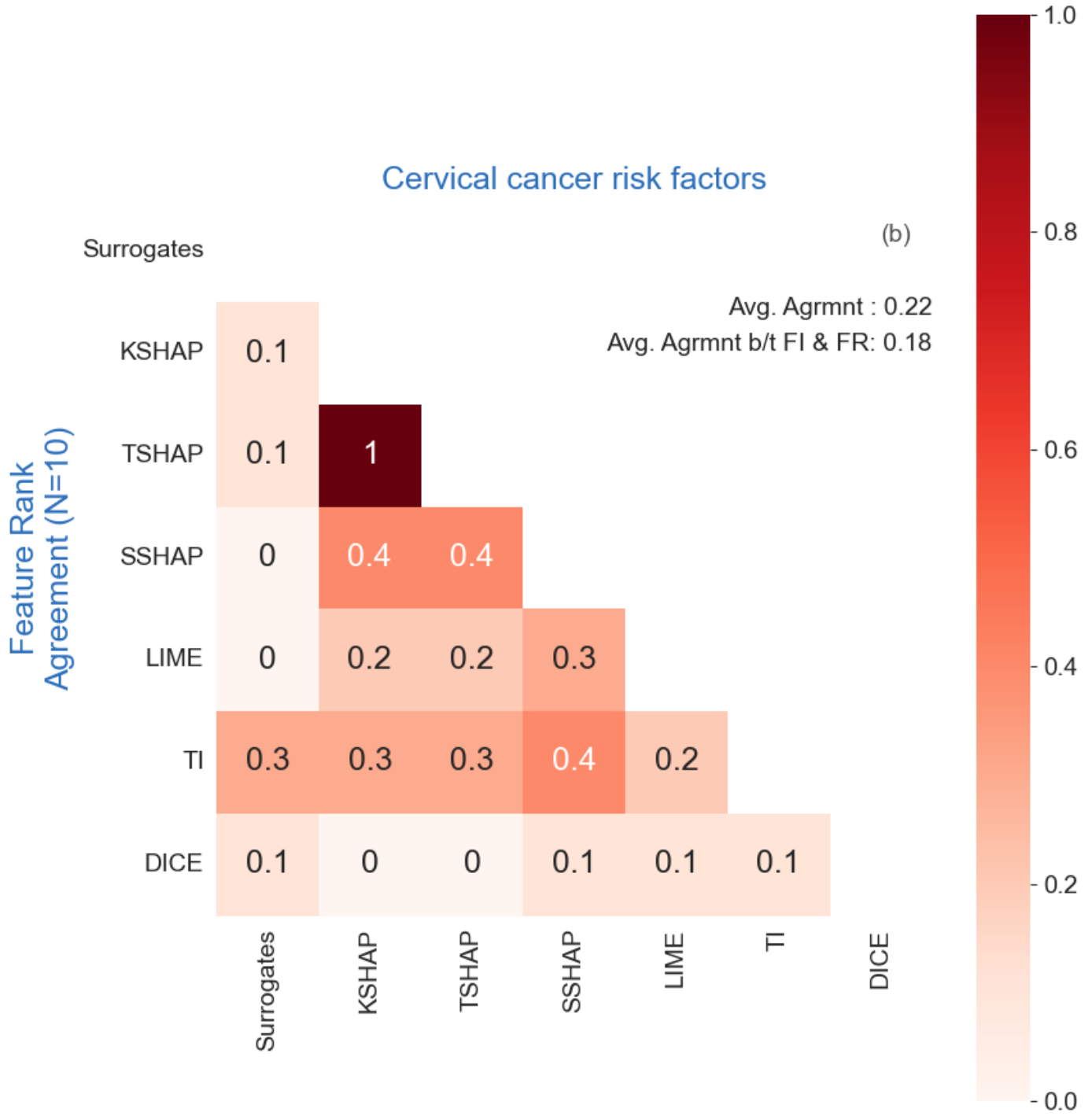
In [100…

```
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/featagrem'+str(instance)+'.png', bb
```

In [100…

```
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()
```



In [100…

```
#fig.savefig('/content/drive/My Drive/dataXAI/cancer/rankagrem'+str(instance)+'.png', bb
```

Ablations

Using MLP instead of RF

In [100…
```
risk_factor_df=pd.read_csv('risk_factors_cervical_cancer.csv')
X_test=pd.read_csv('X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)
```

In [101…
```
nn_clf = MLPClassifier()
nn_clf.fit(X_train, y_train)
nn_clf.score(X_train, y_train)
nn_clf.score(X_test, y_test)
model=nn_clf
```

In [101…
```
GloSur=kernelSHAP=samplingSHAP=limecontrib=dicecontrib=pd.DataFrame([[0.0]*X_test.shape[
fi_1=fi_2=fi_3=fi_4=fi_5={f'{x}':0.0 for x in X_test.columns}

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

In [101…
```
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 [101…
```
print("-KSHAP-")
# KSHAP

explainer = shap.KernelExplainer(model.predict_proba, X_train)
if not os.path.isfile("kshap-ml"):
    shap_values = explainer.shap_values(X_test)
    with open("kshap-ml", "wb") as fp:
        pickle.dump(shap_values, fp)

with open("kshap-ml", "rb") as fp:
    shap_values = pickle.load(fp)
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 u
sing shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K sampl
es.
```

```
-KSHAP-
```

In [101…

```
print("-SSHAP-")
# SSHAP

explainer = shap.explainers.Sampling(model.predict_proba, X_train)
if not os.path.isfile("sshap-ml"):
    shap_values = explainer.shap_values(X_test)
    with open("sshap-ml", "wb") as fp:
        pickle.dump(shap_values, fp)

with open("sshap-ml", "rb") as fp:
    shap_values = pickle.load(fp)

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_3={k: fi_3.get(k, 0) + res.get(k, 0) for k in set(fi_3)}
```

```
-SSHAP-
```

In [101…

```
print("-LIME-")

# LIME

explainer = lime_tabular.LimeTabularExplainer(X_train.values,mode='classification',featu

all=[]
if not os.path.isfile("lime-ml"):
    for i in range (len(X_test)):
        exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_featur
        all.append(sorted(exp.as_map()[1]))
    with open("lime-ml", "wb") as fp:
        pickle.dump(all, fp)

with open("lime-ml", "rb") as fp:
    all = pickle.load(fp)

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_4={k: fi_4.get(k, 0) + res.get(k, 0) for k in set(fi_4)}
```

```
-LIME-
```

In [101…

```
label = "Dx:Cancer"
#these columns are not of type object, but are of type numeric
```

```
cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Num of preg
                   'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contraceptives',
                   'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STDs', 'STD
                   'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:vaginal
                   'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic i
                   'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AIDS', 'ST
                   '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, e
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))
risk_factor_df["Age"] = risk_factor_df["Age"].astype(int)
risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)

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()
risk_factor_df["total_tests"] = risk_factor_df[test_cols].sum(axis = 1)
```

In [101…

```
print("-DICE-")

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)
df=risk_factor_df.drop(['total_std', 'age_cat', 'total_tests'],axis=1)


cont_feat = list(X_test.columns)
#cont_feat.remove(label)

d = dice_ml.Data(dataframe=df, continuous_features=cont_feat, outcome_name=label)
m = dice_ml.Model(model=model, backend="sklearn")

exp = dice_ml.Dice(d, m, method="random")
#query_instance = X_test.drop(label,axis=1)
if not os.path.isfile("dice-e1-ml"):
    e1 = exp.generate_counterfactuals(query_instance, total_CFs=10, desired_range=None,
                                      desired_class="opposite",
                                      permitted_range=None, features_to_vary="all")
    with open("dice-e1-ml", "wb") as fp:
        pickle.dump(e1, fp)

with open("dice-e1-ml", "rb") as fp:
    e1 = pickle.load(fp)

if not os.path.isfile("dice-imp-ml"):
    imp = exp.local_feature_importance(query_instance, posthoc_sparsity_param=None)

    with open("dice-imp-ml", "wb") as fp:
        pickle.dump(imp, fp)

with open("dice-imp-ml", "rb") as fp:
    imp = pickle.load(fp)

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_5={k: fi_5.get(k, 0) + res.get(k, 0) for k in set(fi_5)}
```

-DICE-

In [101…
```
GloSur.to_csv("glosur-ml.csv", index=False)
kernelSHAP.to_csv("Kshap-ml.csv", index=False)
samplingSHAP.to_csv("Sshap-ml.csv", index=False)
limecontrib.to_csv("lime-ml.csv", index=False)
dicecontrib.to_csv("dice-ml.csv", index=False)
```

In [101…
```
dics = []

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


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
dics.to_csv("toutfi-ml.csv", index=False)
```

In [102…
```
instance=291
gscontrib=pd.read_csv('glosur-ml.csv')
kercontrib=pd.read_csv('Kshap-ml.csv')
samcontrib=pd.read_csv('Sshap-ml.csv')
limecontrib=pd.read_csv('lime-ml.csv')
dicecontrib=pd.read_csv('dice-ml.csv')
all_fi=pd.read_csv('toutfi-ml.csv')
all_fi.fillna(0, inplace=True)
all_fi.iloc[:,:-1]=np.abs(all_fi.iloc[:,:-1])
all_fi.reset_index(drop=True, inplace=True)
label="Dx:Cancer"
methods=all_fi['Method'].to_list()
weights=[gscontrib, kercontrib, samcontrib, limecontrib, dicecontrib]
```

In [102…
```
risk_factor_df.describe().iloc[1]
xx=risk_factor_df.describe().iloc[1]
instance=291
xx=X_test.iloc[instance]
idx=list(xx.to_numpy().nonzero()[0])
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]]

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

```
                                  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 [102…

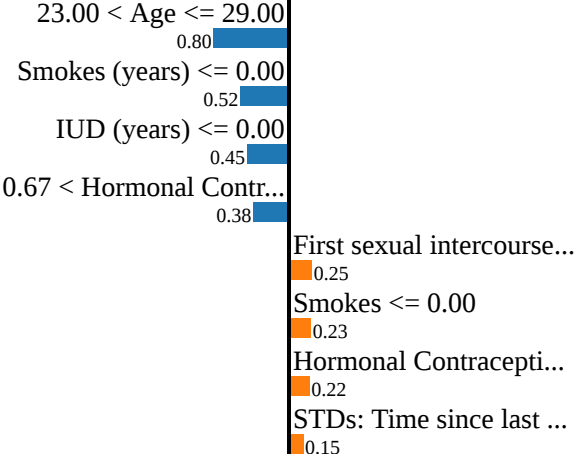
```
one_instance=[]

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')
one_instance.to_csv('one_instance-ml.csv')
print('methods' in one_instance.columns)
print(one_instance)

X_test=pd.read_csv('X_test.csv')
X_test.drop('Unnamed: 0', inplace=True, axis=1)
y_test=pd.read_csv('y_test.csv')
y_test.drop('Unnamed: 0', inplace=True, axis=1)
X_train=pd.read_csv('X_train.csv')
X_train.drop('Unnamed: 0', inplace=True, axis=1)
y_train=pd.read_csv('y_train.csv')
y_train.drop('Unnamed: 0', inplace=True, axis=1)

instance=291
var='W'
maxx=10
f=''
#vale=0

print(model.predict(X_test))
explainer = lime_tabular.LimeTabularExplainer(X_train.values,mode='classification',featu
exp = explainer.explain_instance(X_test.iloc[instance], model.predict_proba, num_feature
```

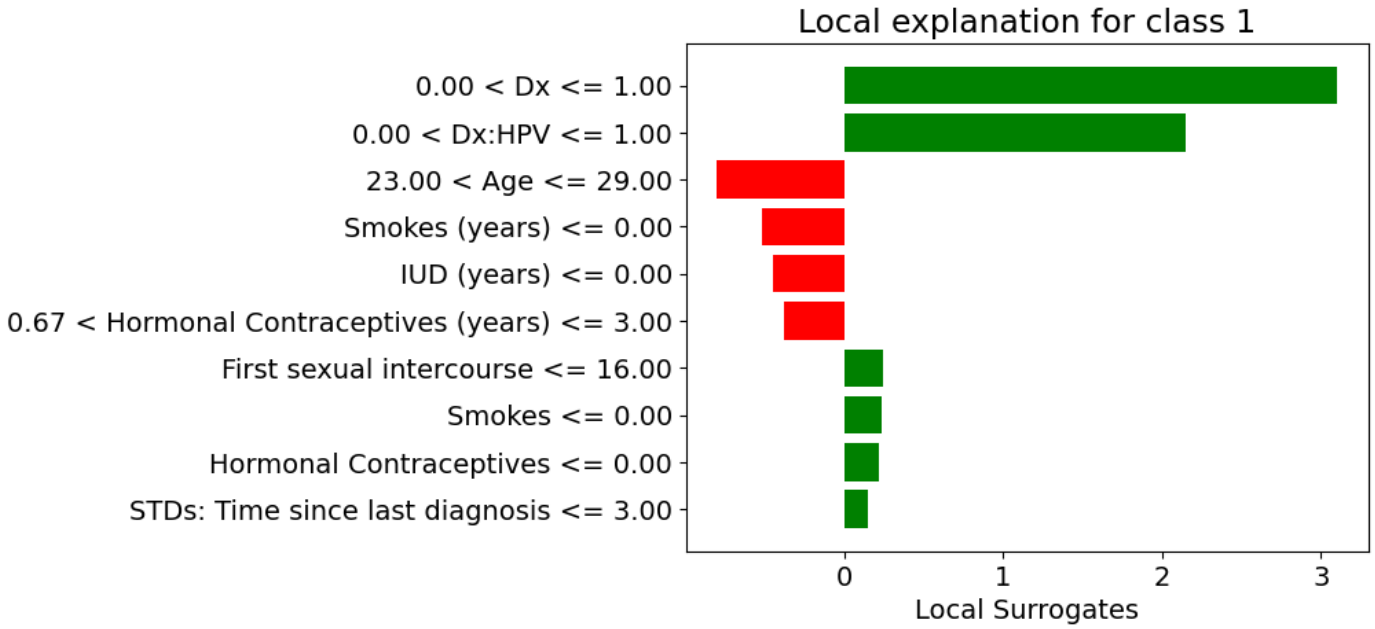
```
False
                Age  Number of sexual partners  First sexual intercourse  \
methods
Surrogates -0.803202                    0.030480                  0.245144
KSHAP       0.006638                   -0.005688                  0.024383
SSHAP       0.004082                    0.000243                  0.027317
LIME       -0.005662                    0.016251                  0.026677
DICE        0.000000                    0.600000                  0.400000

            Num of pregnancies    Smokes  Smokes (years)  Smokes (packs/year)  \
methods
Surrogates           -0.149748  0.232401       -0.518659             0.006198
KSHAP                 0.007011  0.021332       -0.015251             0.002536
SSHAP                 0.002212  0.018869       -0.003958            -0.001884
LIME                 -0.000358  0.251304       -0.209455             0.105680
DICE                  0.000000  0.100000        0.000000             0.000000

            Hormonal Contraceptives  Hormonal Contraceptives (years)  \
methods
Surrogates                 0.220756                        -0.378412
KSHAP                      0.035206                        -0.009220
SSHAP                      0.034036                        -0.009946
LIME                       0.128293                         0.015767
DICE                       0.000000                         0.100000

                IUD  ...  STDs: Number of diagnosis  \
methods              ...
Surrogates  0.000000  ...                  0.000000
KSHAP       0.000962  ...                  0.000000
SSHAP       0.003086  ...                  0.002197
```

```
LIME        0.062129  ...                    0.083427
DICE        0.000000  ...                    0.000000

            STDs: Time since first diagnosis  STDs: Time since last diagnosis  \
methods
Surrogates                         -0.002299                         0.151134
KSHAP                               0.007133                         0.004451
SSHAP                               0.001352                         0.000091
LIME                                0.170646                         0.060994
DICE                                0.100000                         0.100000

              Dx:CIN    Dx:HPV        Dx  Hinselmann  Schiller  Citology  \
methods
Surrogates  0.000000  2.146876  3.103454    0.000000  0.000000  0.000000
KSHAP       0.000000  0.226624  0.188086    0.000000  0.000000  0.004352
SSHAP       0.000000  0.226332  0.190298    0.000005 -0.001582  0.000439
LIME        0.226351  0.508559  0.466250    0.037361  0.004997  0.026442
DICE        0.100000  0.000000  0.100000    0.000000  0.000000  0.000000

              Biopsy
methods
Surrogates  0.000000
KSHAP       0.000000
SSHAP      -0.000036
LIME       -0.015850
DICE        0.000000

[5 rows x 35 columns]
[1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1
 1 0 0 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 1 0
 1 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1 1
 1 1 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1
 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1
 1 0 0 1 0 0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1
 1 0 1 1 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 1
 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1 0 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 0 1 0 1 1 0
 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0
 0 1 1]
```

In [102…

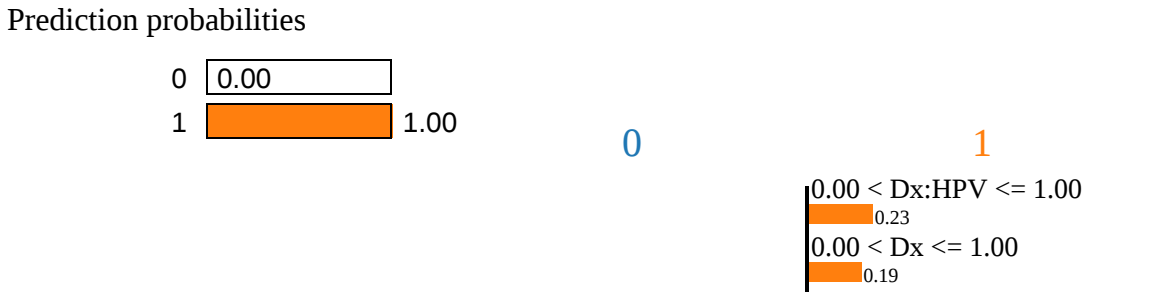
```
items = gscontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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)

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

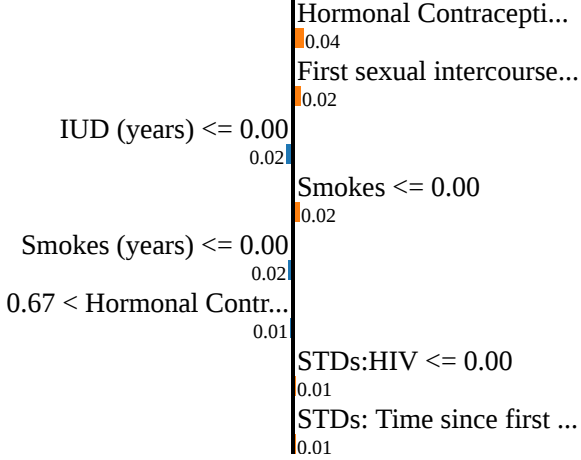


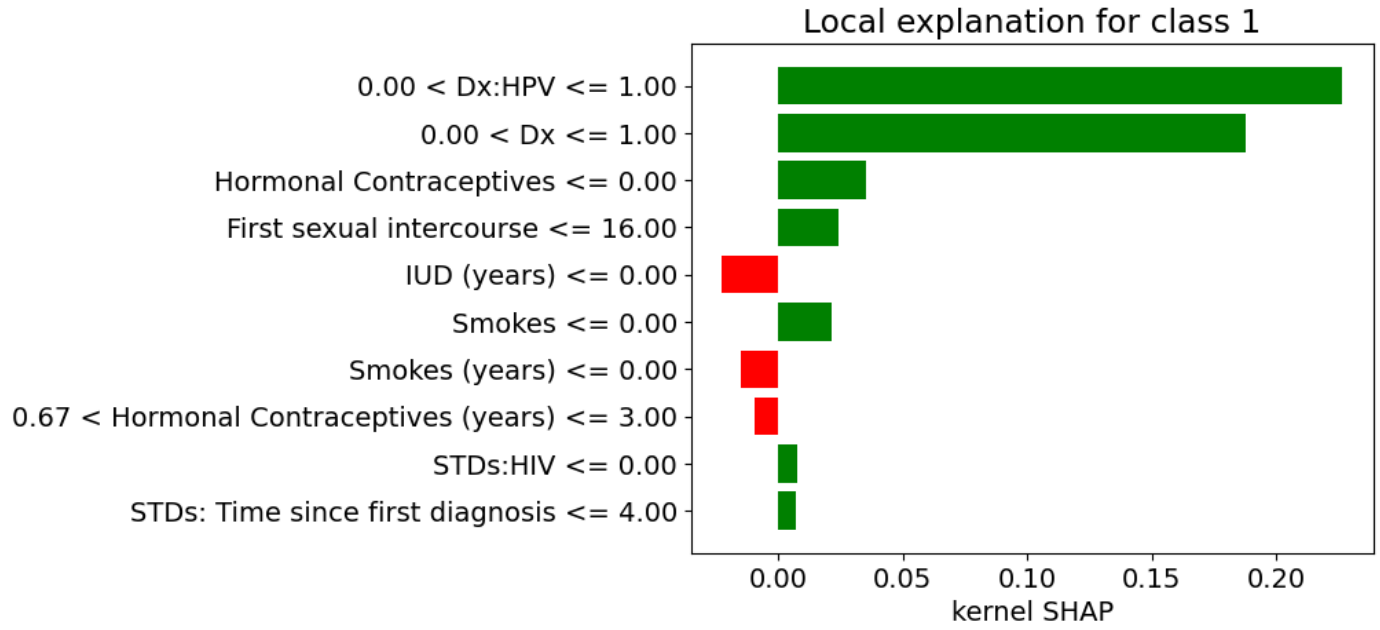


In [102…

```
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)

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

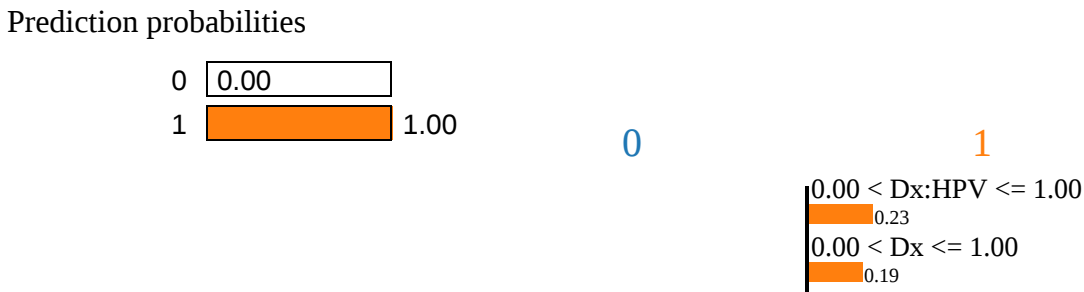


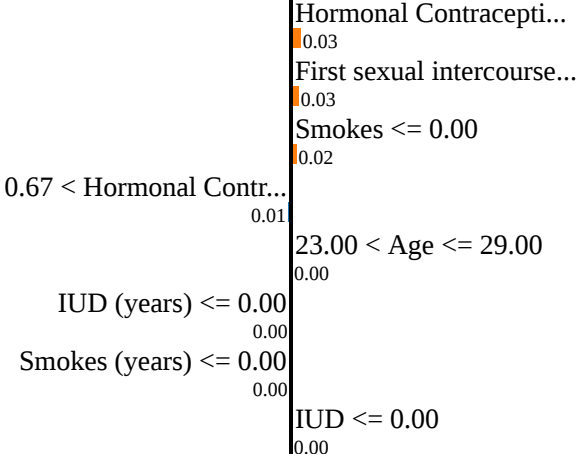


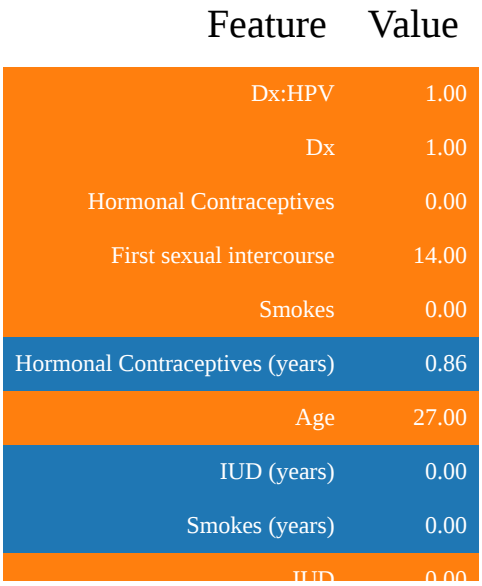


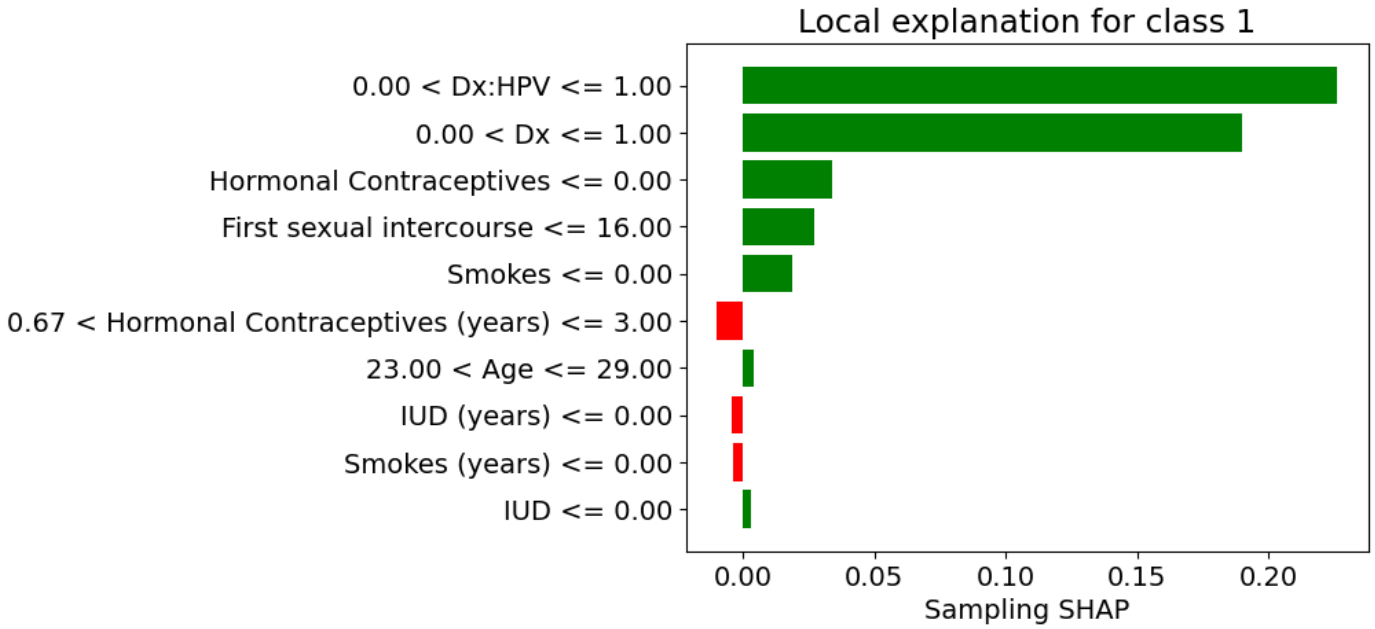
In [102…

```
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)

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





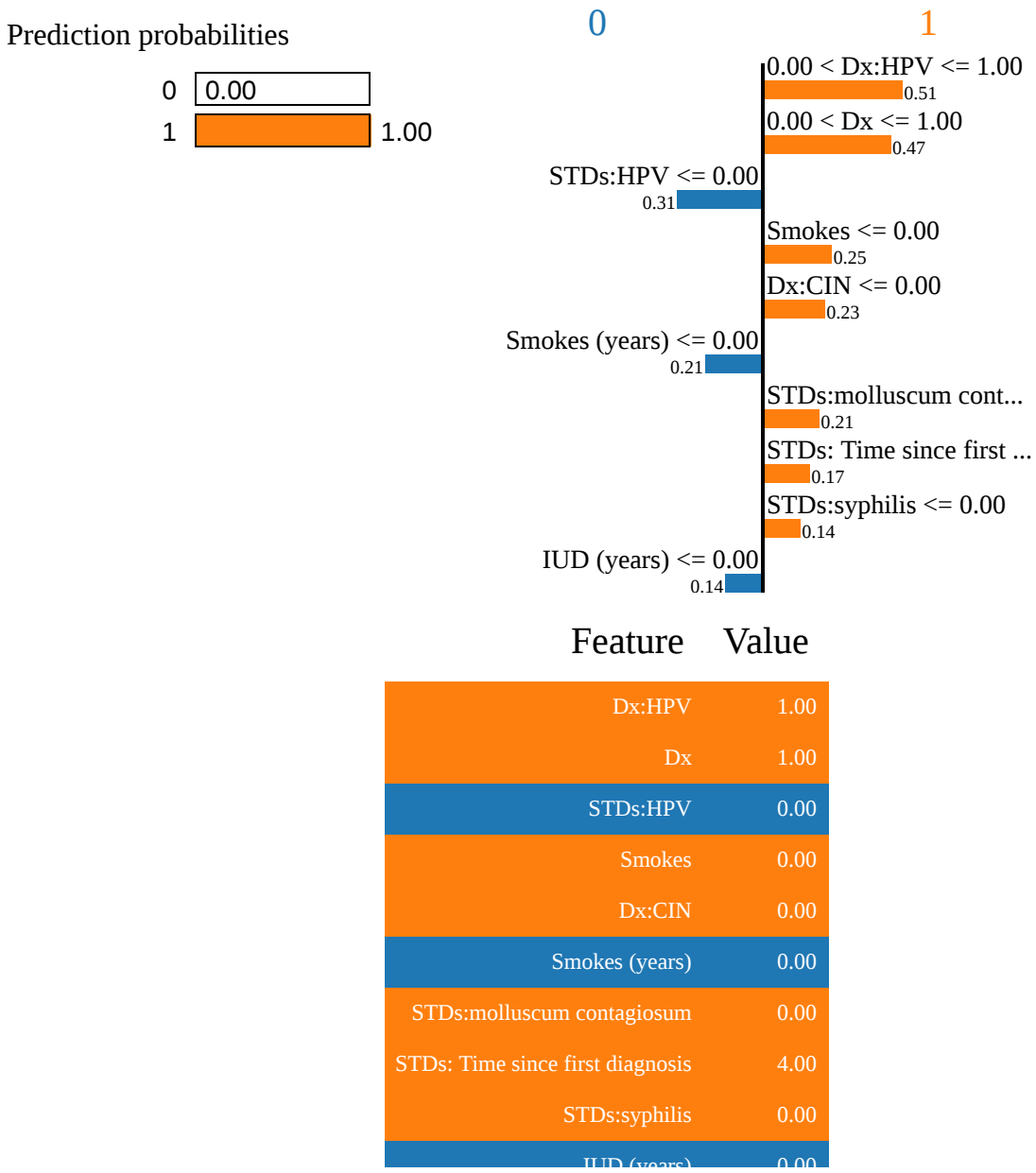


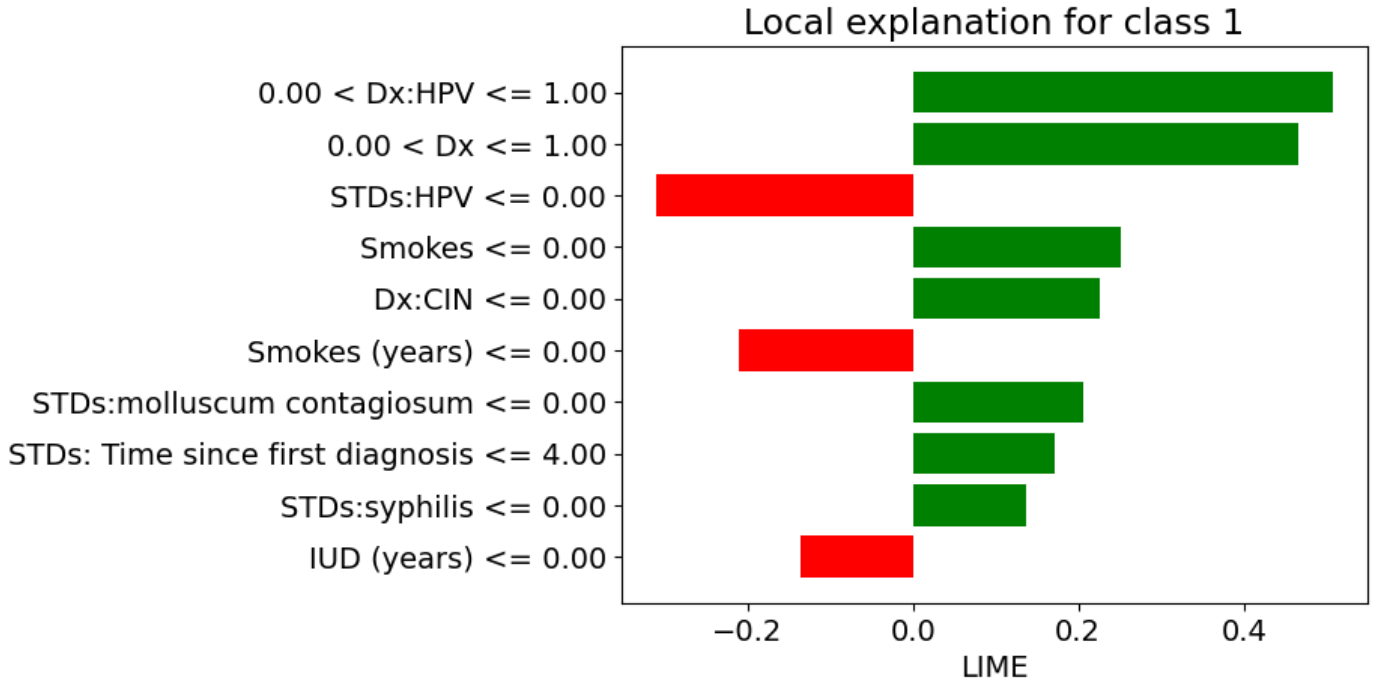


In [102…

```
items = limecontrib.iloc[instance].to_dict()
#print(limecontrib)
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]}
```

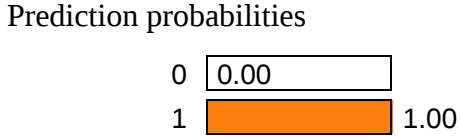
```
exp.local_exp = exp_test
exp.show_in_notebook(show_table=True,show_predicted_value=False)

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

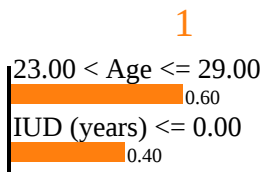




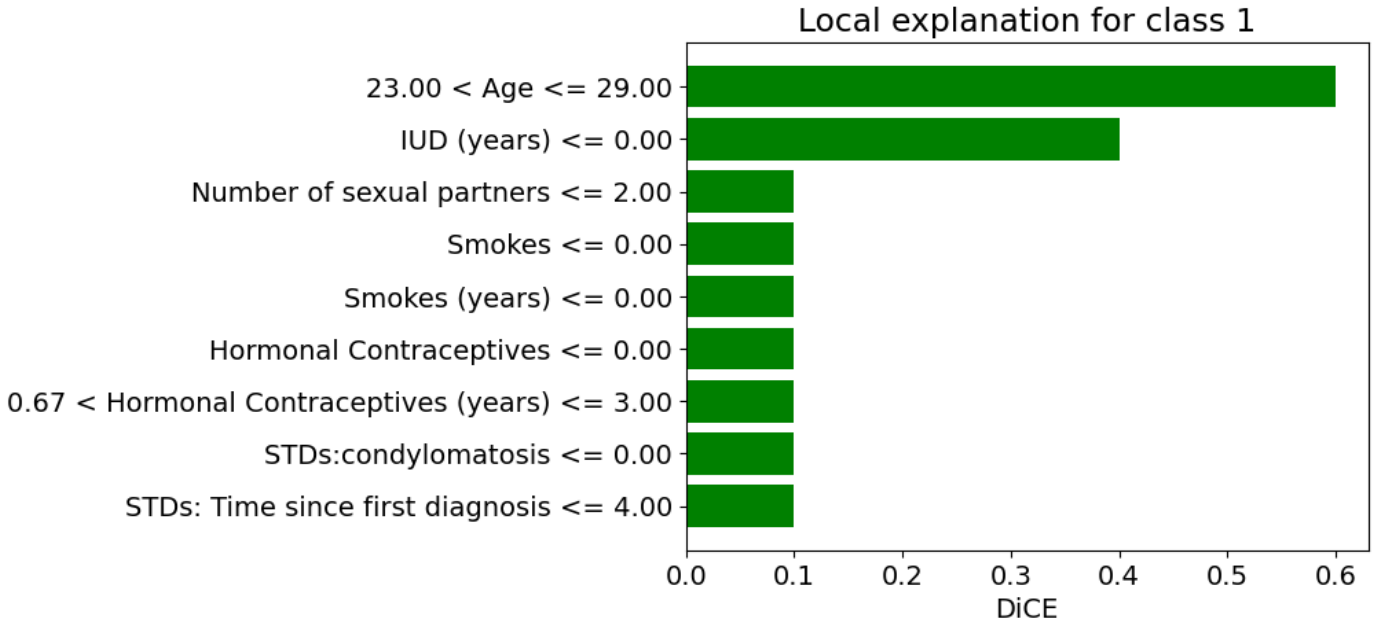
In [102…

```
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)

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



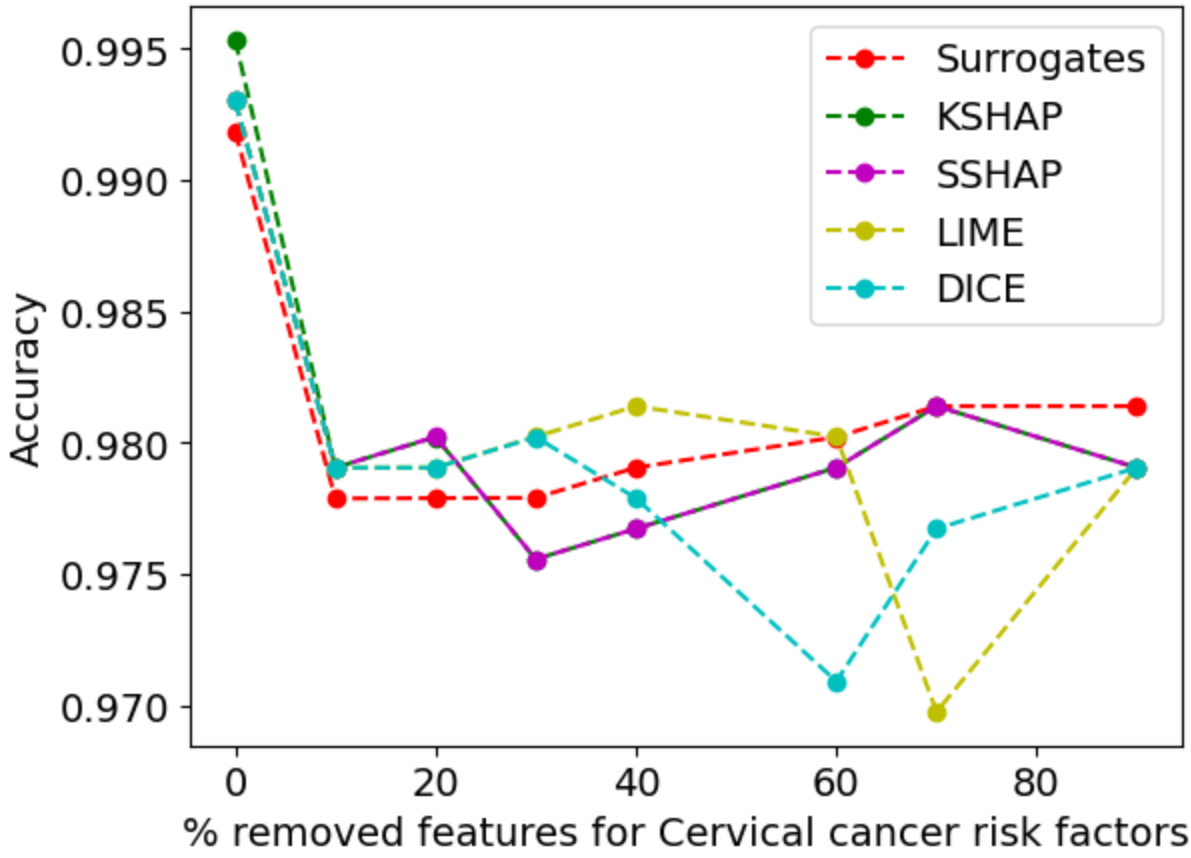


Number of sexual partn...
0.10
Smokes <= 0.00
0.10
Smokes (years) <= 0.00
0.10
Hormonal Contracepti...
0.10
0.67 < Hormonal Contr...
0.10
STDs:condylomatosis ...
0.10
STDs: Time since first ...
0.10



In [102…

```
#datapath='/content/drive/My Drive/dataXAI/cancer/cancer.csv'
#savepath= '/content/drive/My Drive/dataXAI/cancer/'
datapath='risk_factors_cervical_cancer.csv'
savepath= ''
dataname='Dx:Cancer'

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



	Number of sexual partners	Hormonal Contraceptives (years)	\
0	0.050407	0.410812	
1	0.002722	0.019811	
2	0.002379	0.019023	
3	0.010125	0.060588	
4	0.176190	0.059821	

	Hormonal Contraceptives	Num of pregnancies	STDs:HIV	STDs:Hepatitis B	\
0	0.149799	0.551748	0.001243	0.000000	
1	0.025779	0.006801	0.002208	0.001591	
2	0.025346	0.006747	0.000812	0.000001	
3	0.124763	0.018418	0.071859	0.063086	
4	0.036012	0.042262	0.017857	0.015179	

	Citology	Smokes	STDs:vaginal condylomatosis	Dx:HPV	...	\
0	0.036573	0.284276	0.000000	2.792217	...	
1	0.002792	0.025779	0.001585	0.224105	...	
2	0.002050	0.024895	0.000059	0.224763	...	
3	0.044492	0.257555	0.047563	0.519051	...	
4	0.015179	0.064286	0.016964	0.304167	...	

	STDs:pelvic inflammatory disease	STDs: Number of diagnosis	Biopsy	\
0	0.000000	0.011281	0.005950	
1	0.001560	0.003408	0.003008	
2	0.000024	0.002561	0.002218	
3	0.076828	0.080227	0.010627	
4	0.013690	0.041369	0.020833	

	STDs:vulvo-perineal condylomatosis	STDs:molluscum contagiosum	Hinselmann	\
0	0.012385	0.000000	0.022743	
1	0.002169	0.001459	0.002903	
2	0.001314	0.000006	0.001954	
3	0.104659	0.080921	0.043362	
4	0.014583	0.014286	0.012798	

	Smokes (years)	STDs:HPV	STDs:AIDS	Method
0	0.547120	0.000000	0.0	Surrogates
1	0.034241	0.001720	0.0	KSHAP

```
2        0.032812  0.000101        0.0       SSHAP
3        0.214862  0.162151        0.0        LIME
4        0.145833  0.051488        0.0        DICE

[5 rows x 36 columns]
```

In [102…
```
cns=Consistency()
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test)

img=xpl.plot.contribution_plot(0)
img
```

```
INFO: Shap explainer type - shap.explainers.PermutationExplainer()
```

In [103…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [103…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
img=xpl.plot.contribution_plot(0)
img
```

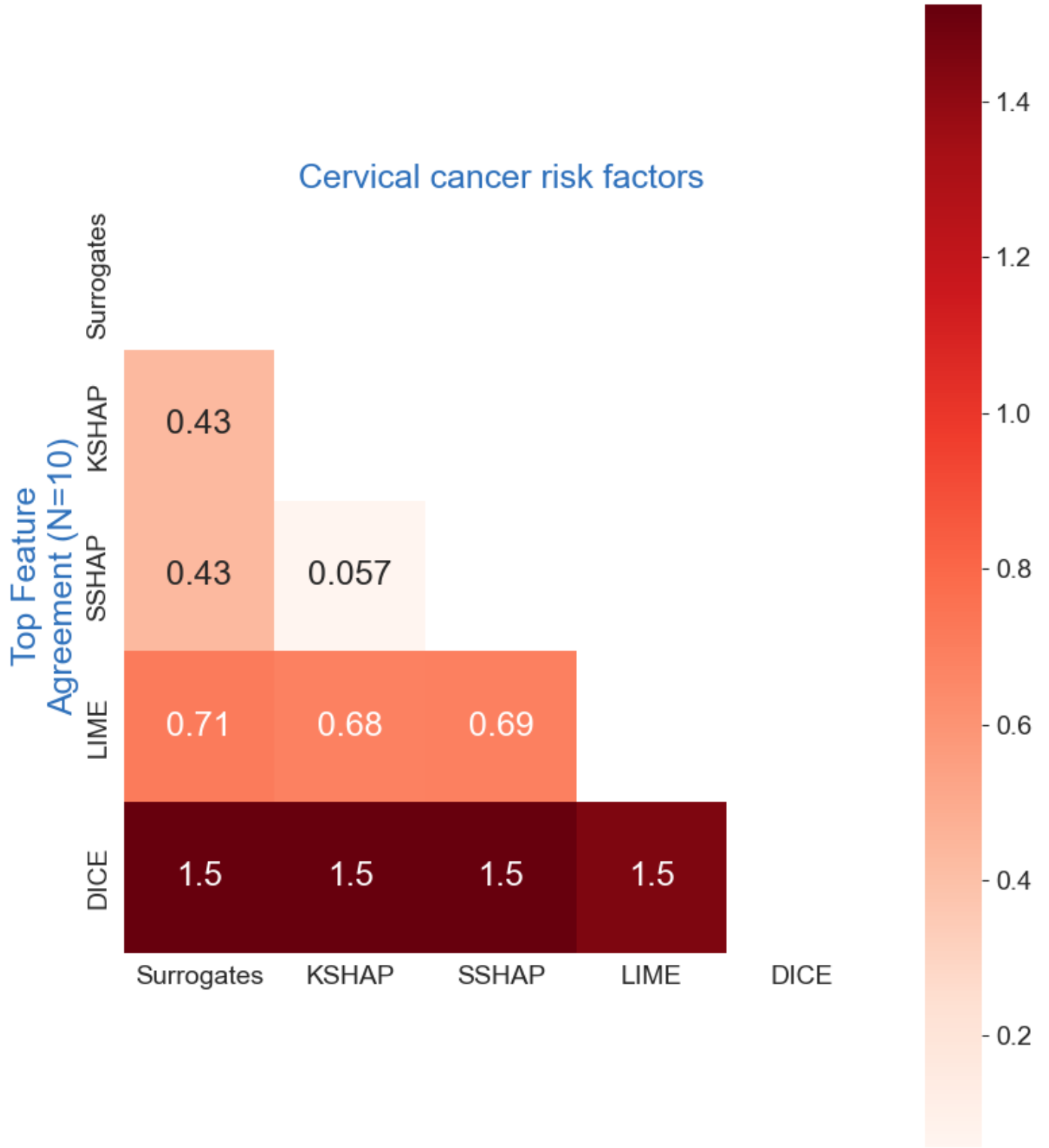
In [103…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img=xpl.plot.contribution_plot(0)
img
```

In [103…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
img=xpl.plot.contribution_plot(0)
img
```

Consistency

In [103…
```
pairwise_consistency=cns.calculate_all_distances(methods, weights)
test=pairwise_consistency[1].round(2)
test.style.background_gradient(cmap='Paired_r')
with open('consistency-ml.tex','w') as tf:
    tf.write(test.to_latex())
```

In [103…
```
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()
fig.savefig('consistency-ml.png', bbox_inches='tight', dpi=300)
```



In [103…

```
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 0.71
DICE 1.2
```

In [103…

```
compacities=[]

for weight in weights:
  rr=compute_features_compacity(case="classification", contributions=weight, selection=l
  #rr=compute_features_compacity(case="classification", contributions=weight, selection=
  compacities.append(pd.DataFrame.from_dict(rr))
```

In [103…
```
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[1038]:

Method	features_needed	distance_reached
Surrogates	2.0	1.00
KSHAP	1.0	0.18
SSHAP	1.0	0.18
LIME	4.0	1.00
DICE	6.0	1.00

In [103…
```
with open('compactness-ml-'+str(instance)+'.tex','w') as tf:
    tf.write(compacity.to_latex())
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
xpl.plot.compacity_plot
```

Out[1039]:
```
<bound method SmartPlotter.compacity_plot of <shapash.explainer.smart_plotter.SmartPlo
tter object at 0x33f340670>>
```

In [104…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
img=xpl.plot.compacity_plot()
```

In [104…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
img=xpl.plot.compacity_plot()
img
```

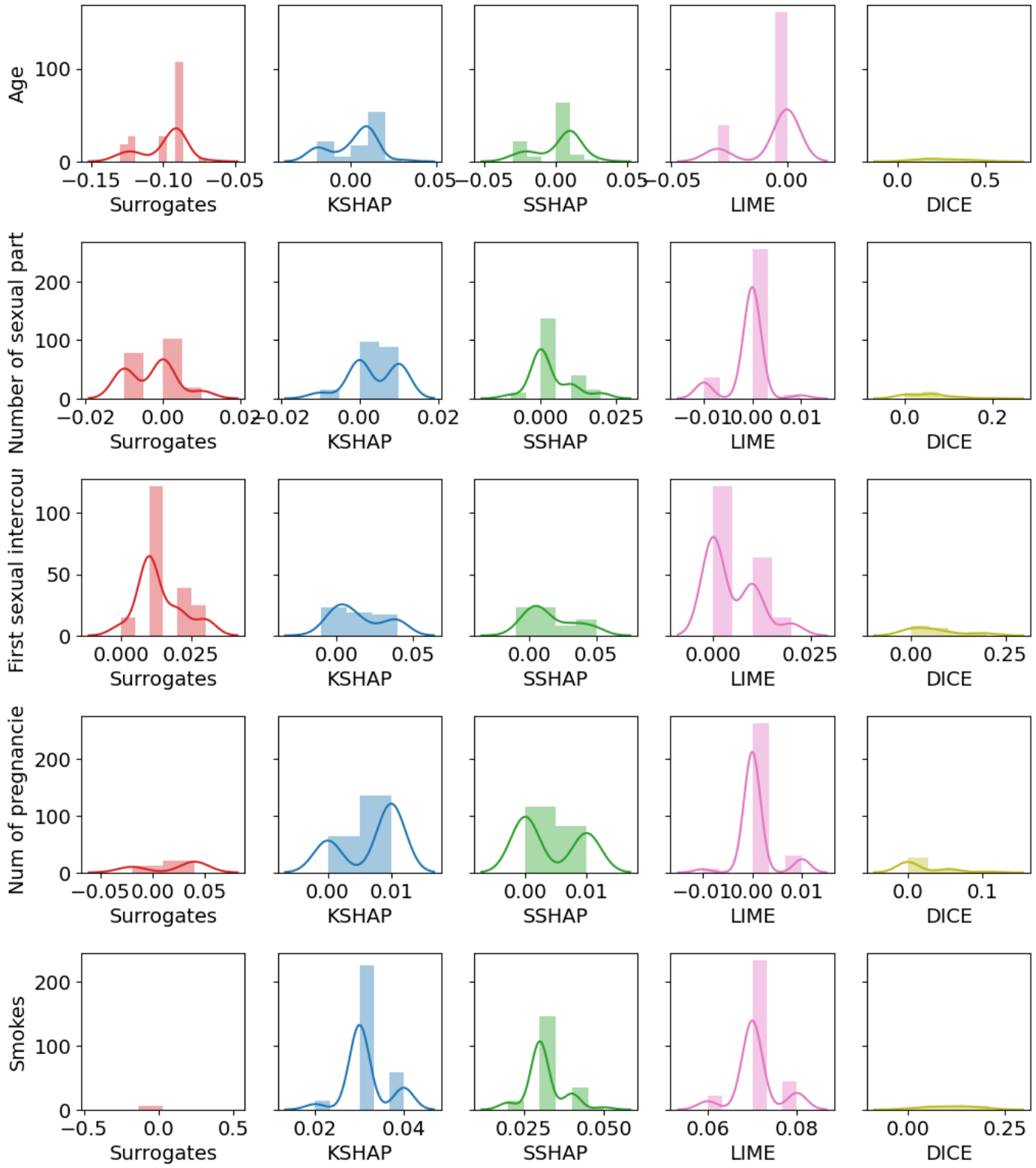
In [104…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
img=xpl.plot.compacity_plot()
img
```

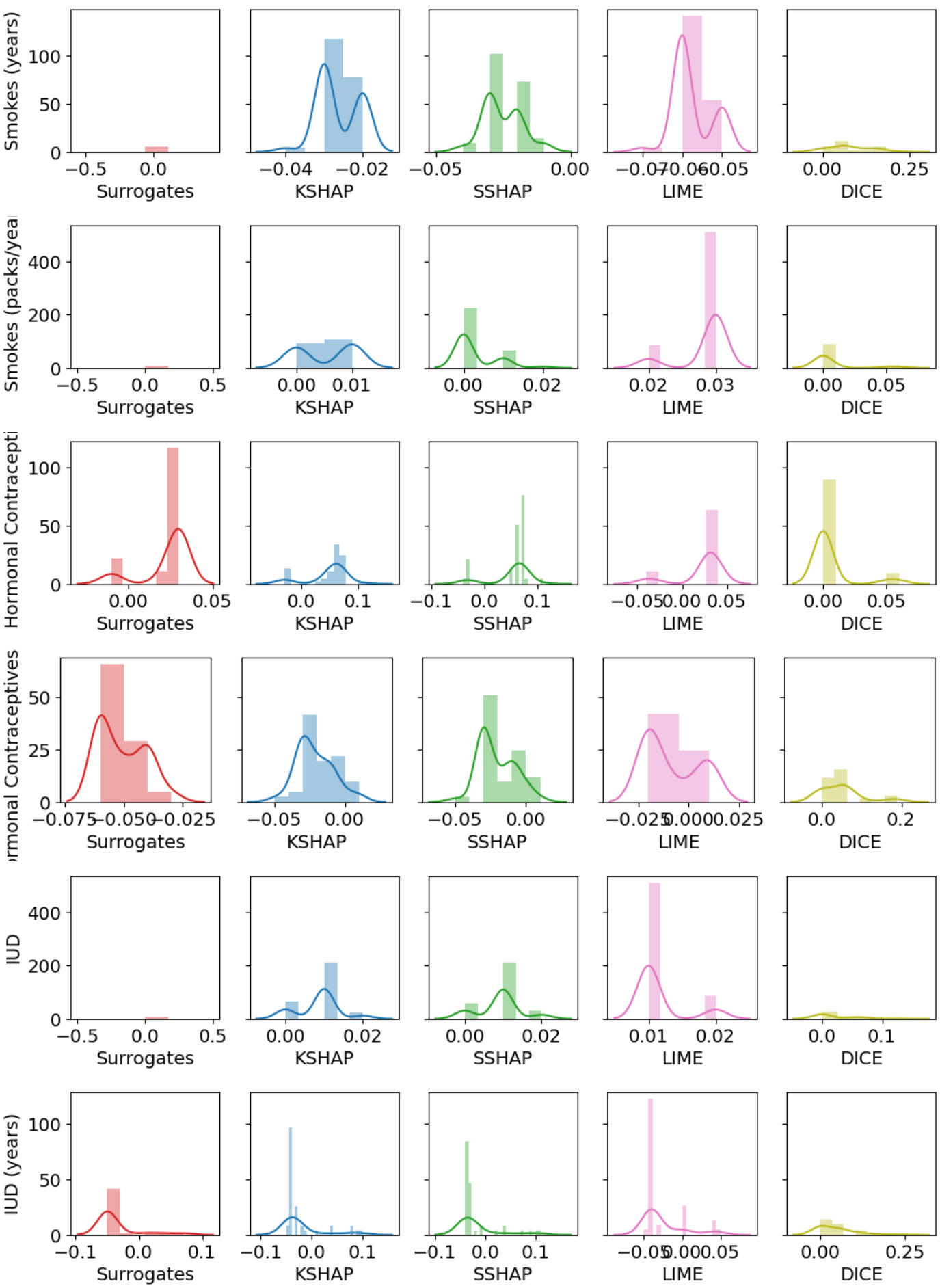
In [104…
```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img=xpl.plot.compacity_plot()
img
```

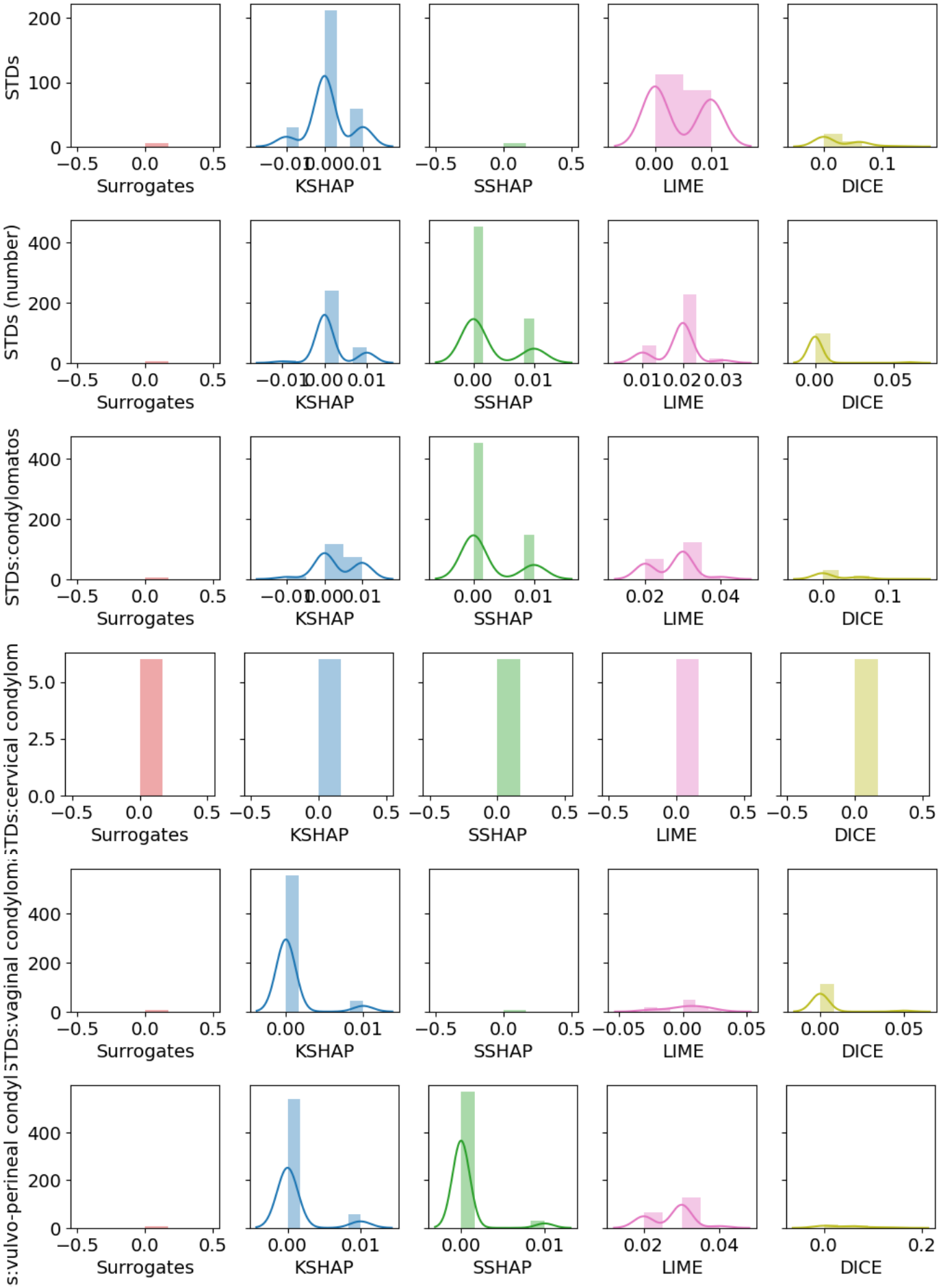
One instance

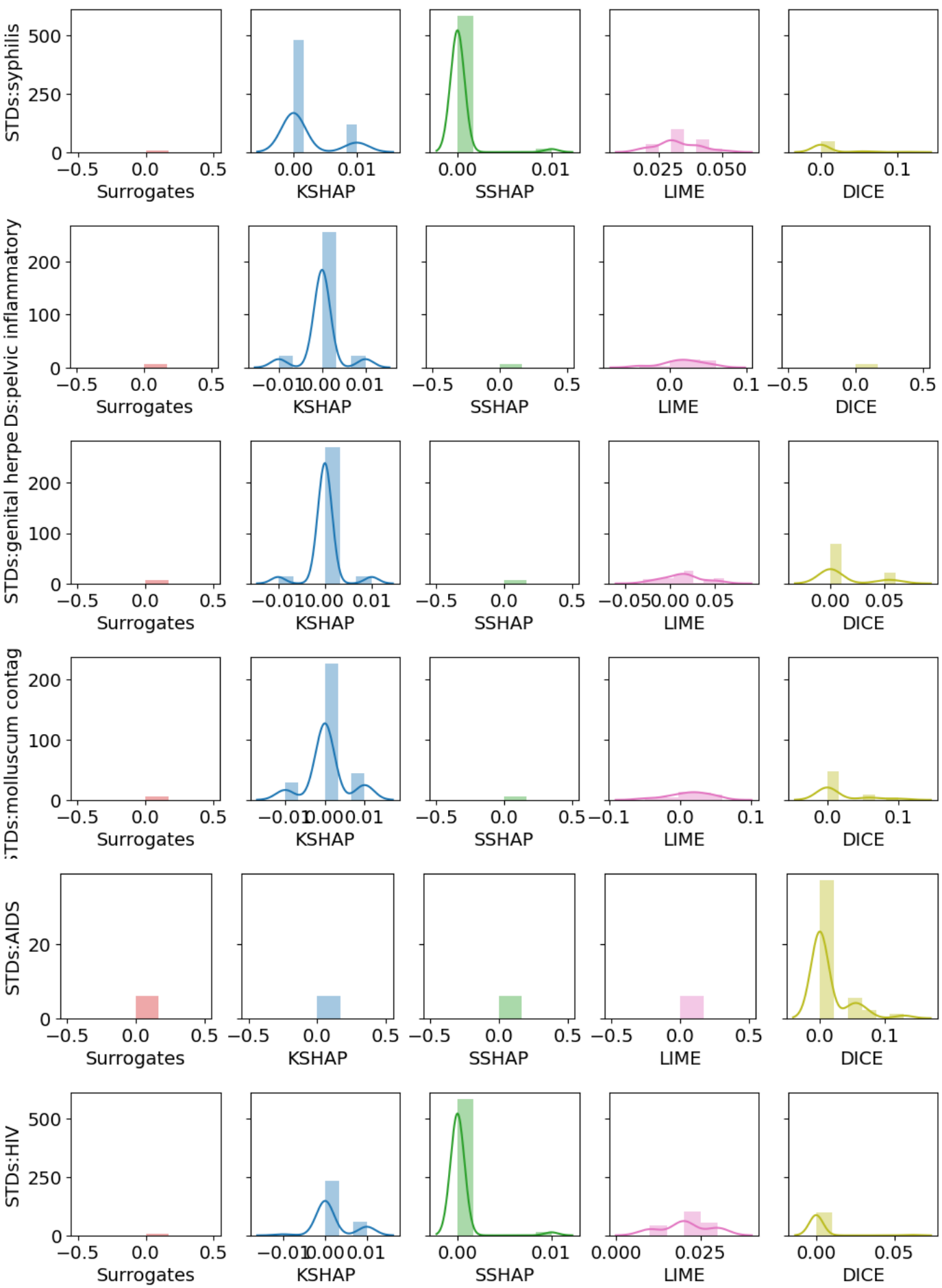
In [104…
```
features=list(X_test.columns)
#compute_features_compacity(case="classification", contributions=weight, selection=list(
frames=[]
for weight in weights:
  #fs= compute_features_stability (case="classification", x=X_test, selection=list(range
  fs= compute_features_stability (case="classification", x=X_test, selection=[instance],
  frames.append(fs)
colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orange',  '

for j in range(len(features)):
```

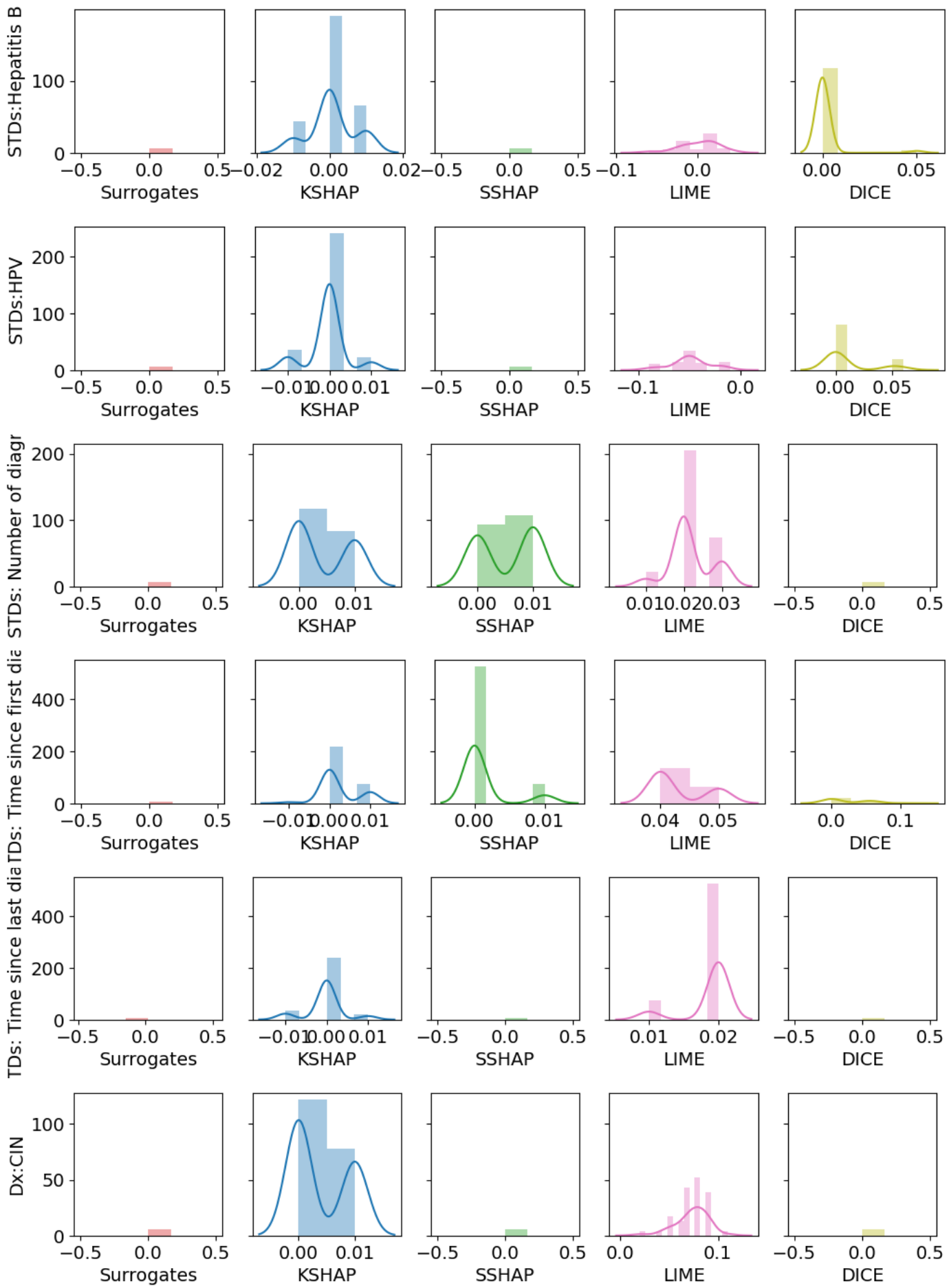
```
fig, axes = plt.subplots(1, 5, figsize=(12, 2), sharey=True, dpi=100)
t=0

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(''+str(var)+str(instance)+str(features[j])[:10]+'.png', bbox_inches='tight
plt.show()
```

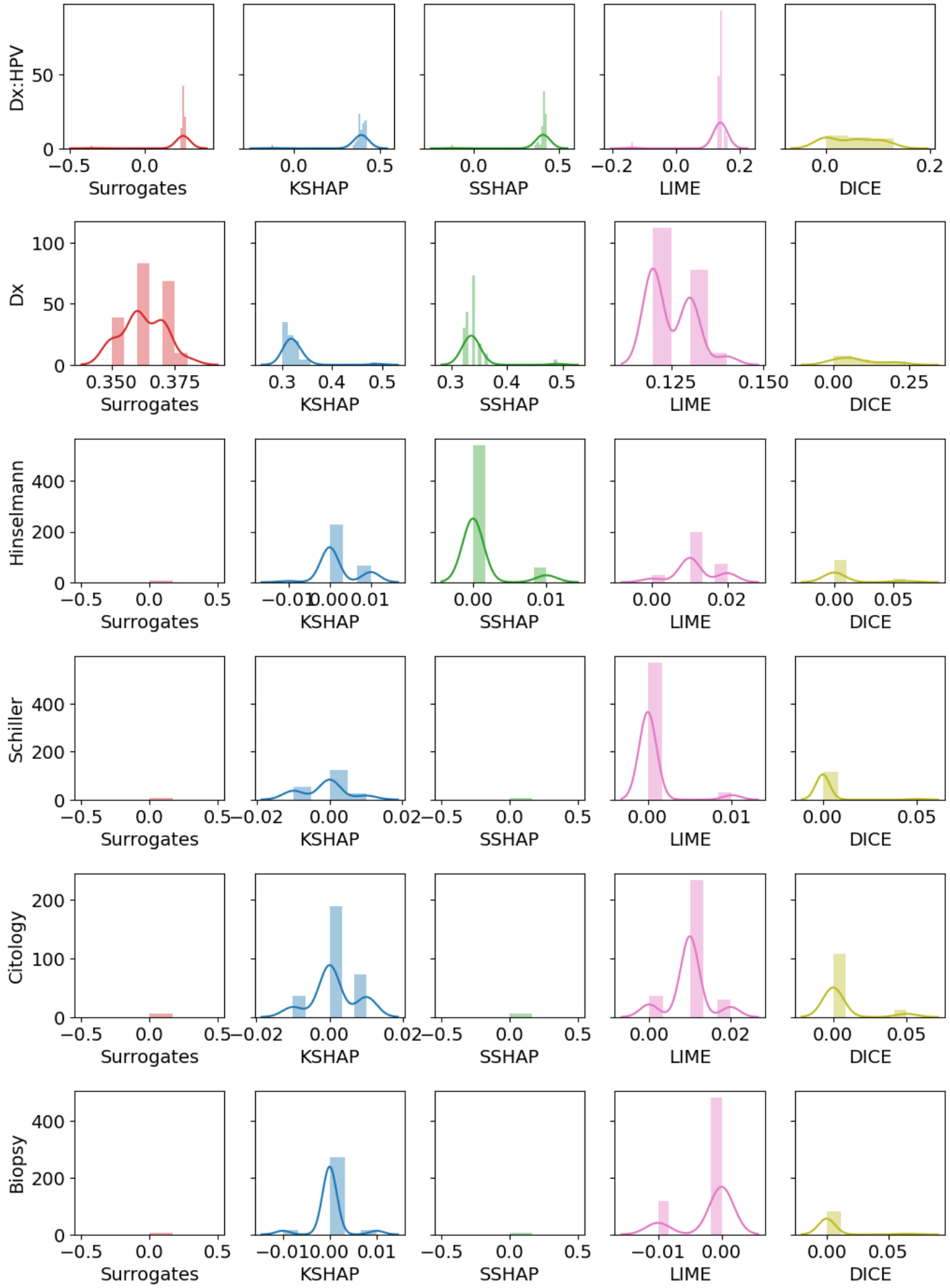



Smokes (years)
Surrogates
KSHAP
SSHAP
LIME
DICE
Smokes (packs/yea
Hormonal Contracepti
rmonal Contraceptives
IUD
IUD (years)


STDs
STDs (number)
STDs:condylomatos
STDs:cervical condylom
STDs:vaginal condylom
s:vulvo-perineal condyl
Surrogates
KSHAP
SSHAP
LIME
DICE


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


STDs:Hepatitis B
STDs:HPV
STDs: Number of diagr
TDs: Time since first dia
TDs: Time since last dia
Dx:CIN
Surrogates
KSHAP
SSHAP
LIME
DICE

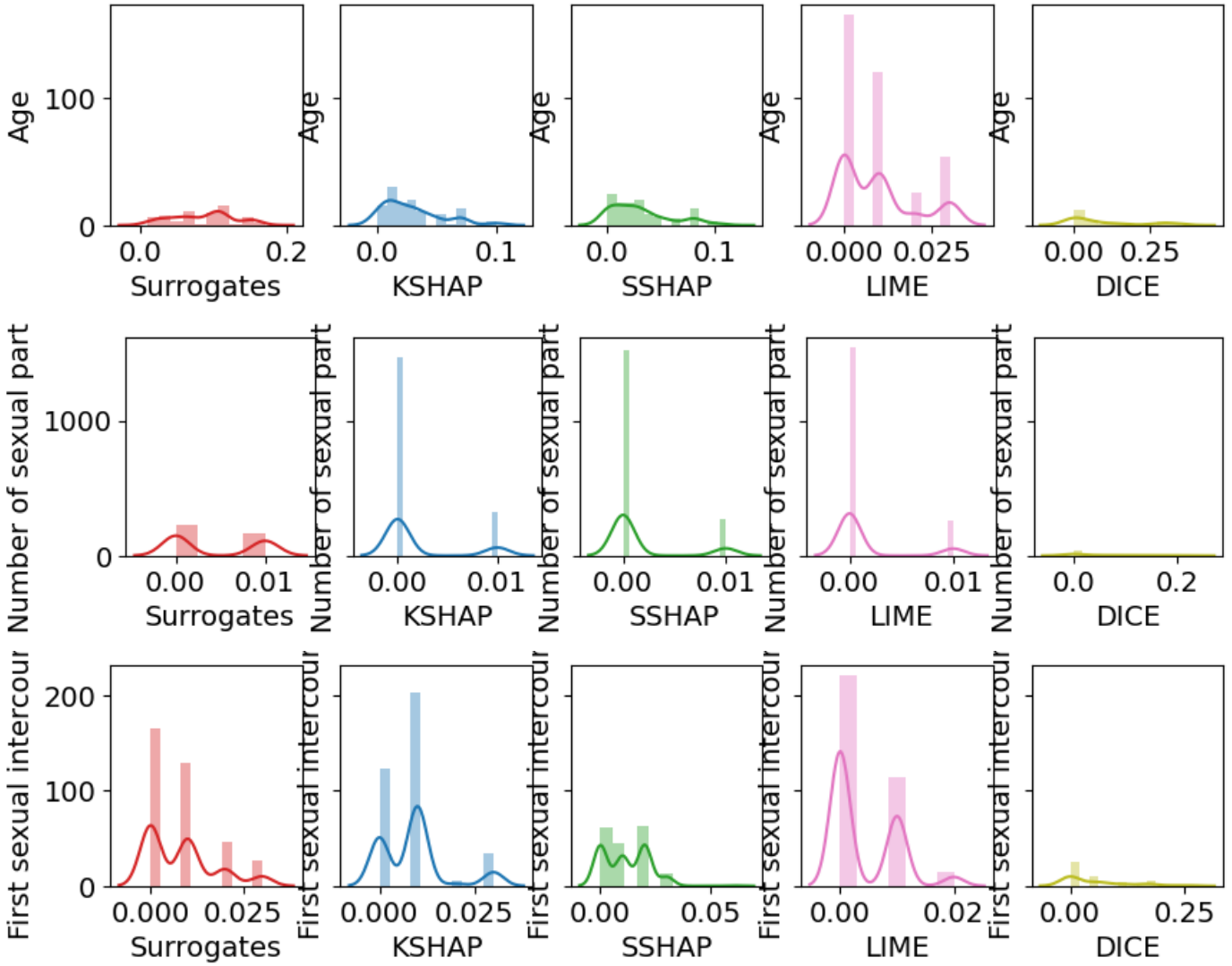


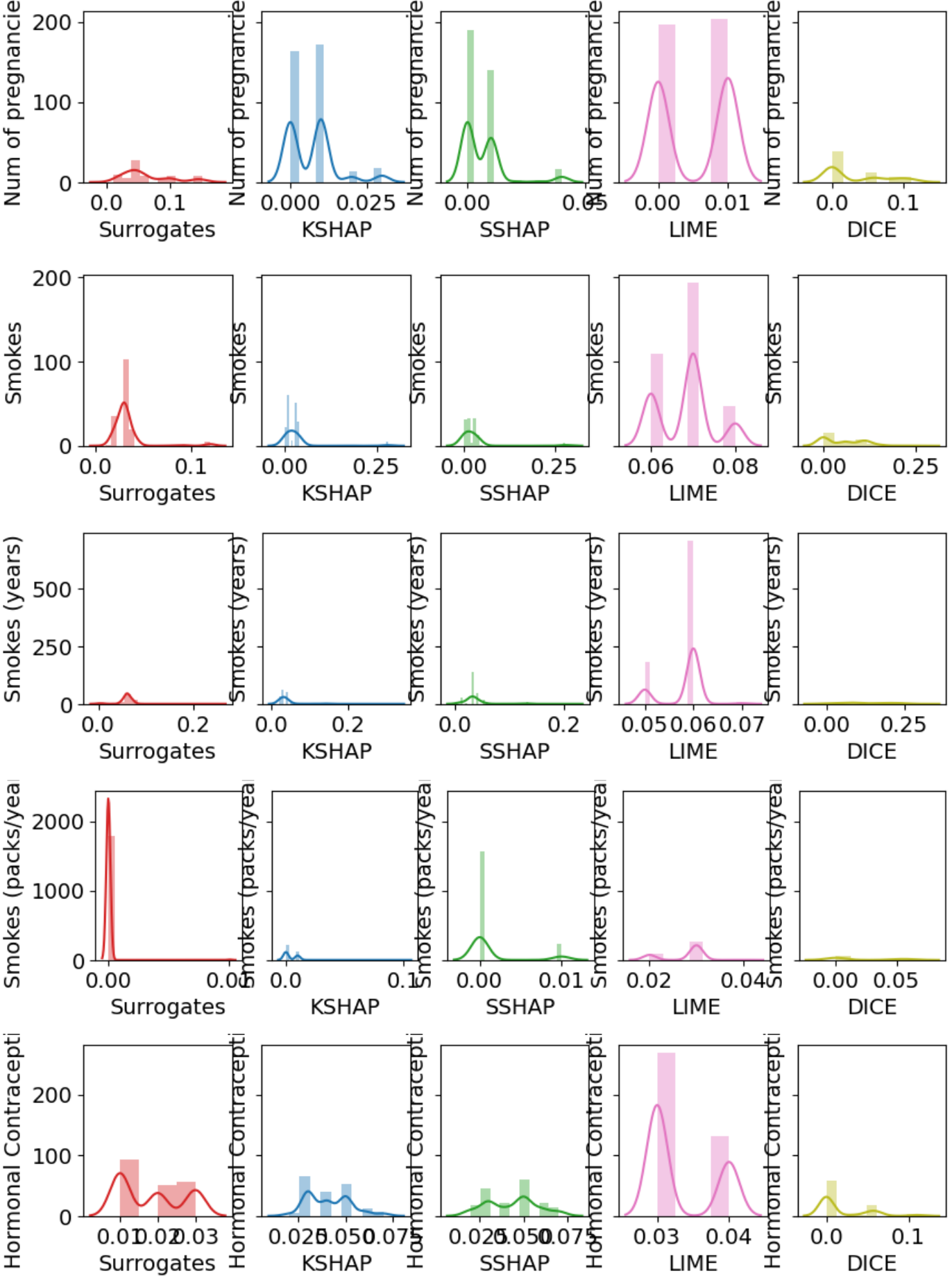
Multiple instances

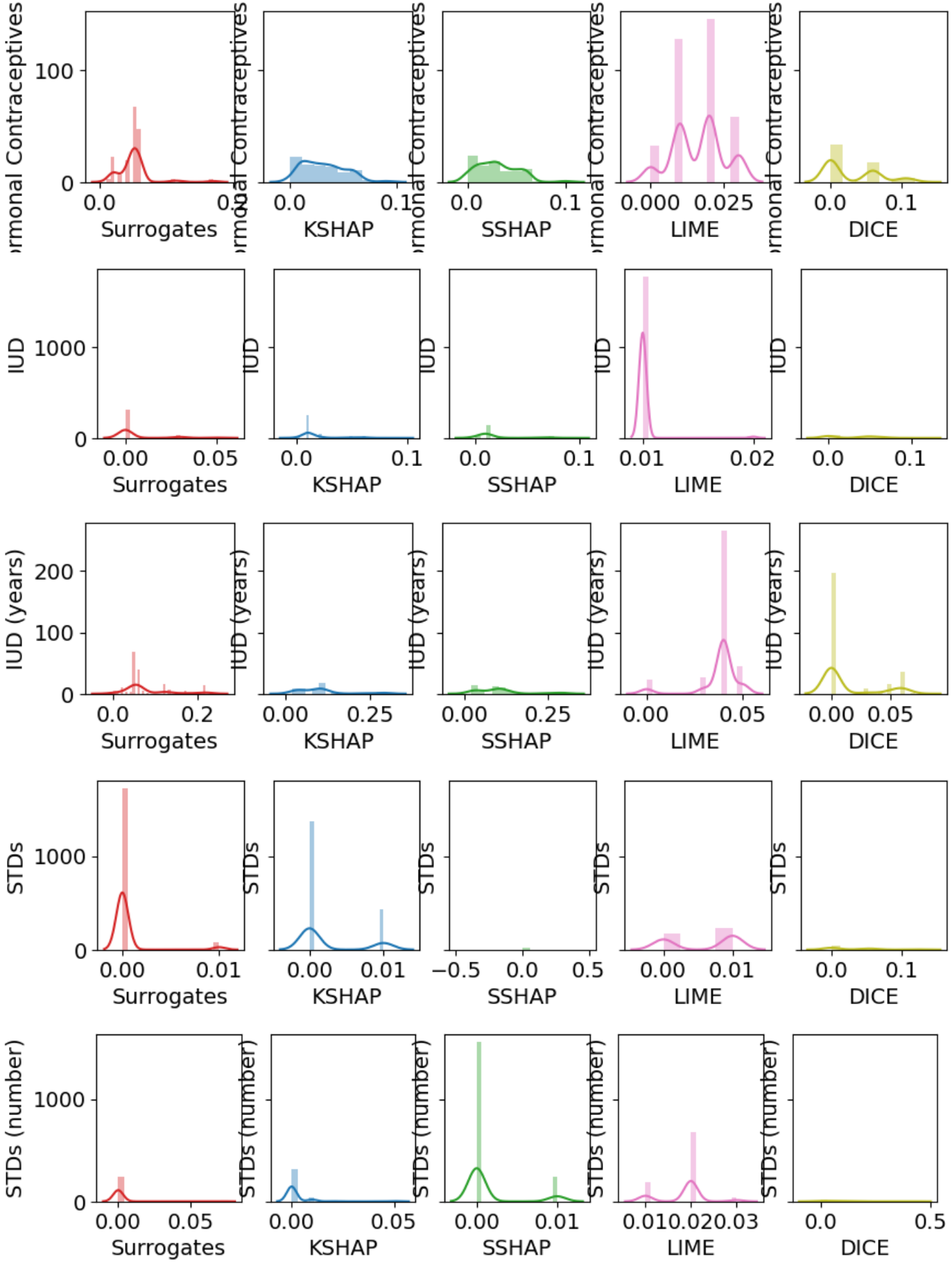
In [104…

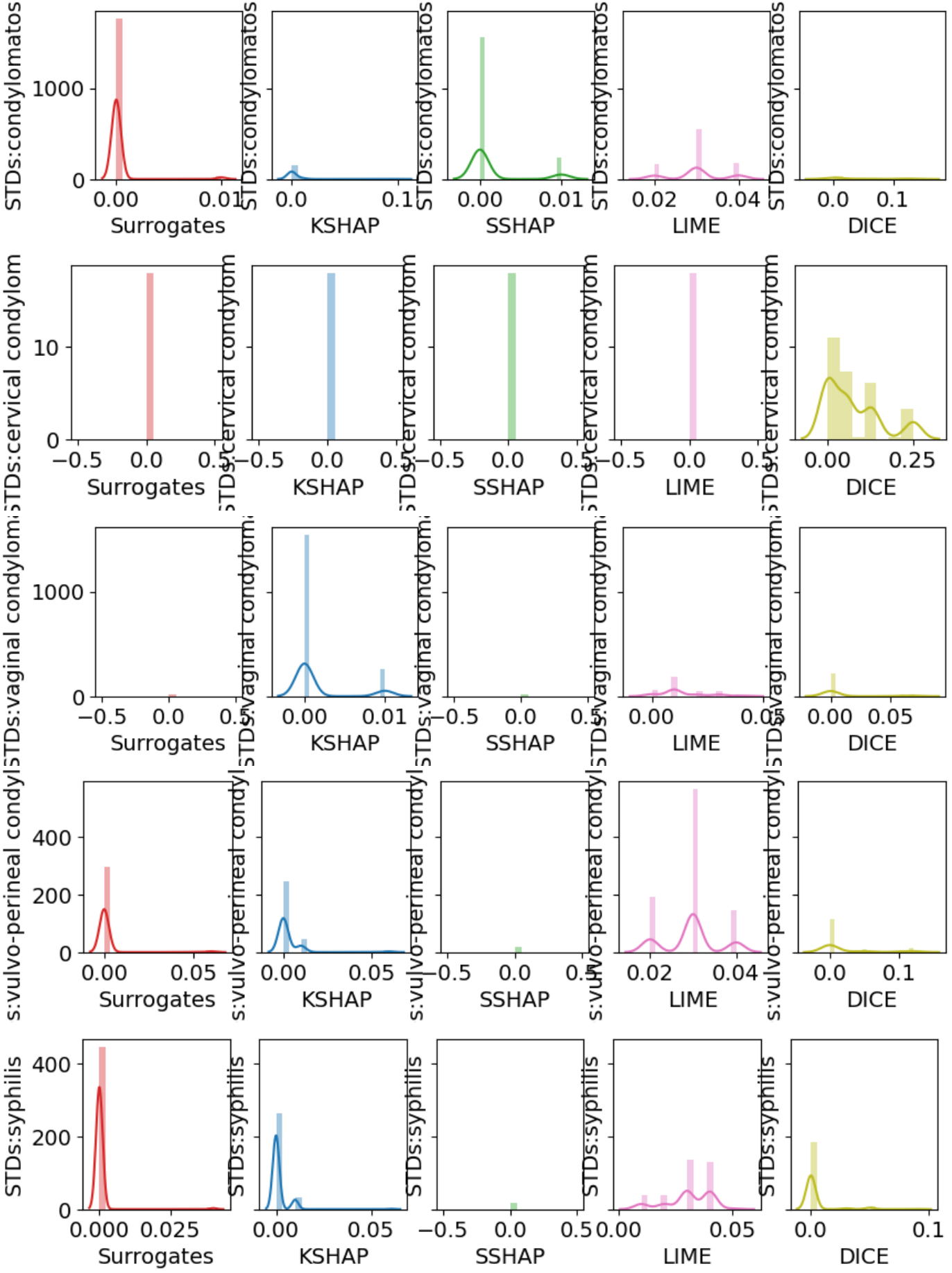
```
features=X_test.columns
#compute_features_compacity(case="classification", contributions=weight, selection=list(
frames=[]
```

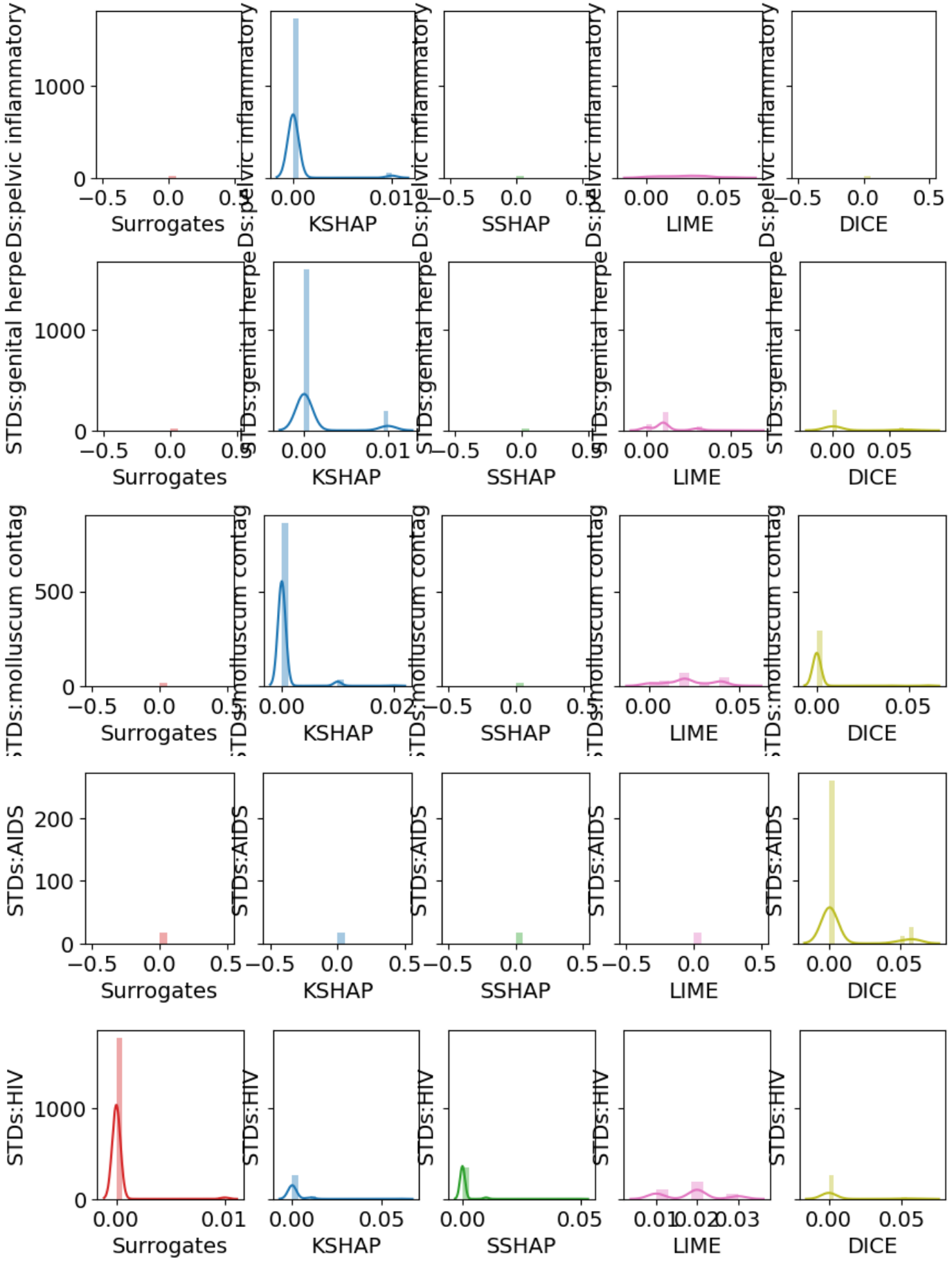
```
for weight in weights:
  fs= compute_features_stability (case="classification", x=X_test, selection=list(range(
  #fs= compute_features_stability (case="classification", x=X_test, selection=[1,4], con
  frames.append(fs)
colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orange',  '

for j in range(len(features)):
  fig, axes = plt.subplots(1, 5, 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(''+str(var)+str(features[j])[:10]+'.png')
  plt.show()
```

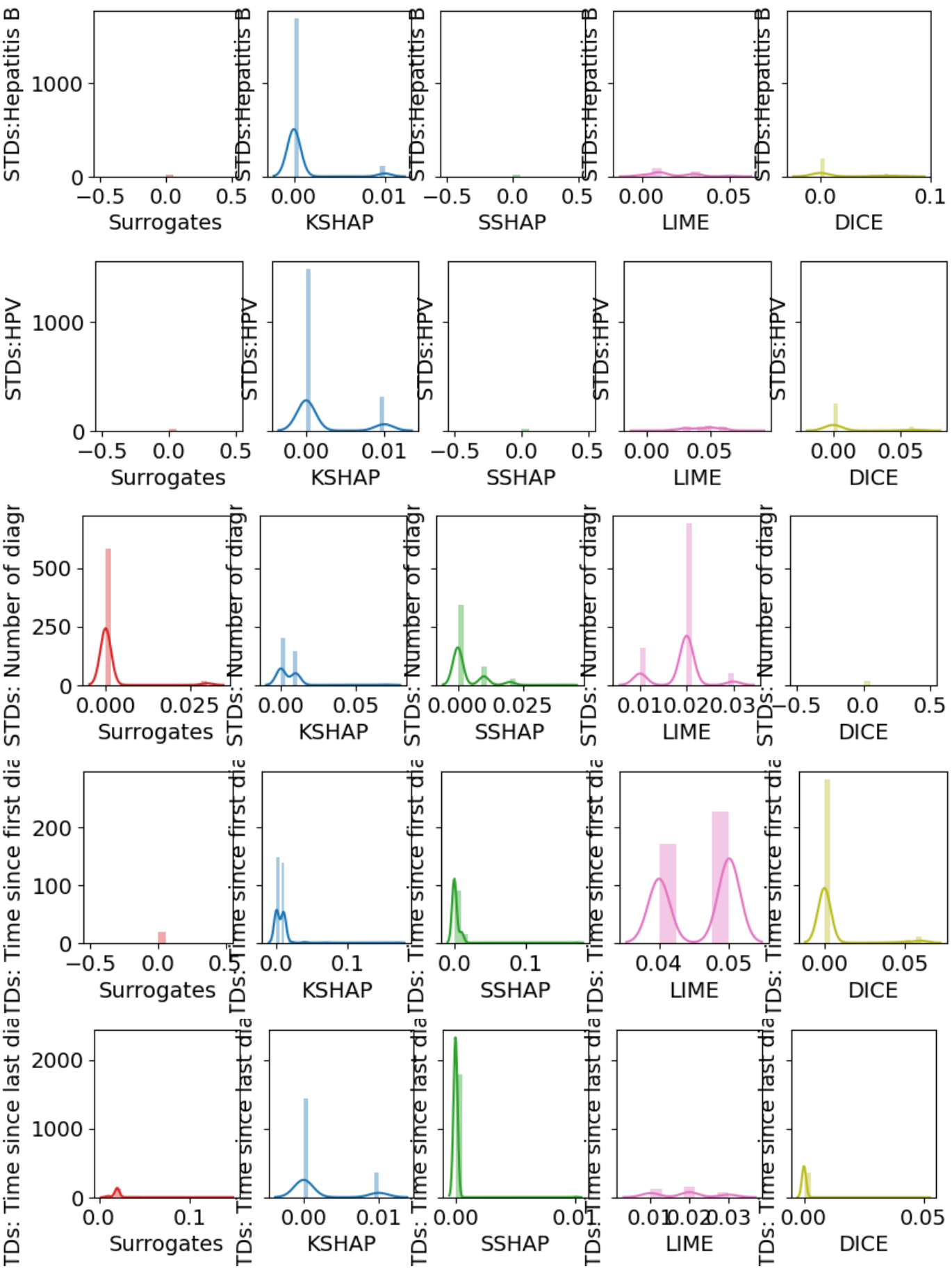


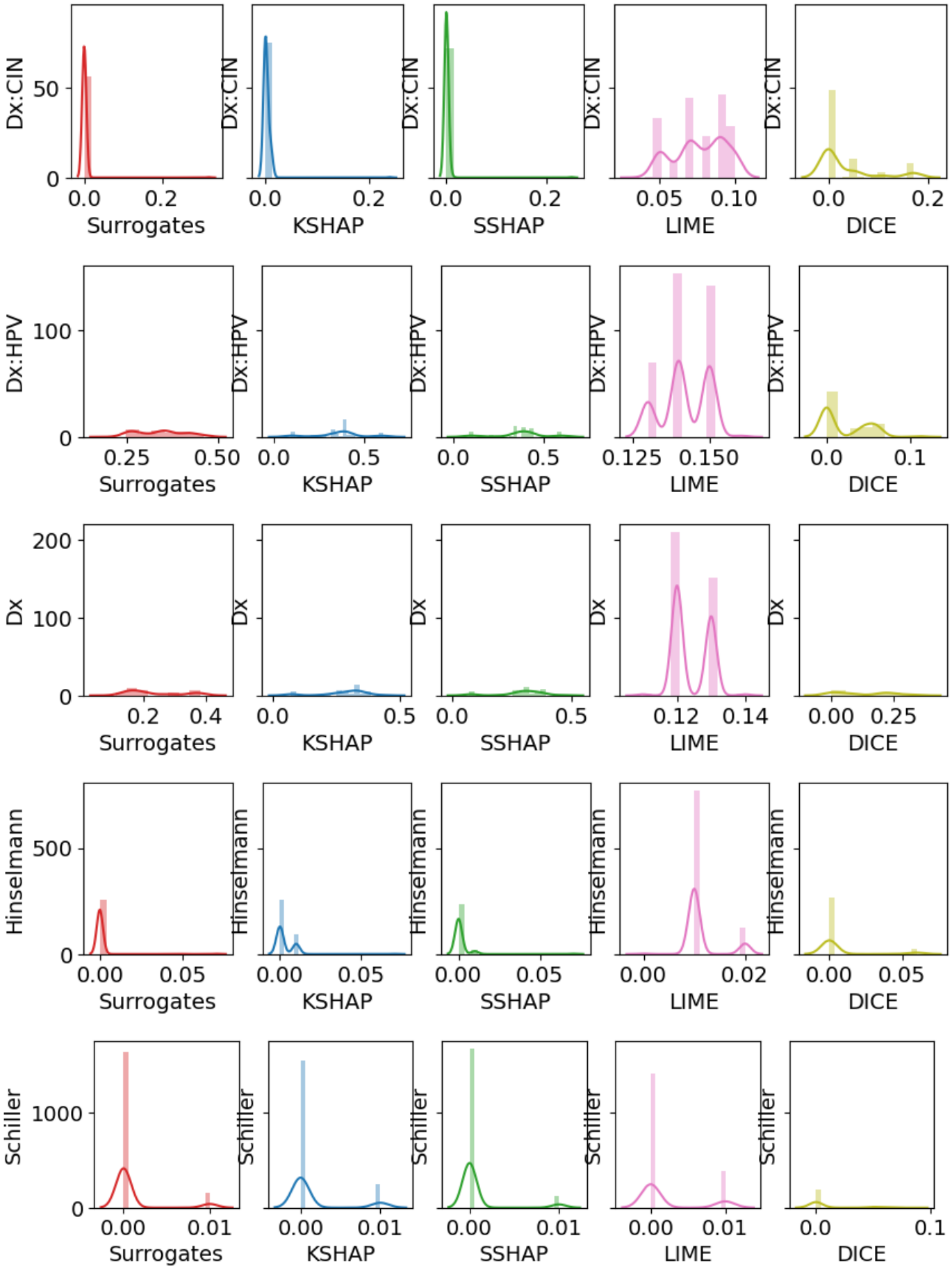

Num of pregnancies
Surrogates
KSHAP
SSHAP
LIME
DICE
Smokes
Smokes (years)
Smokes (packs/yea
Hormonal Contracepti

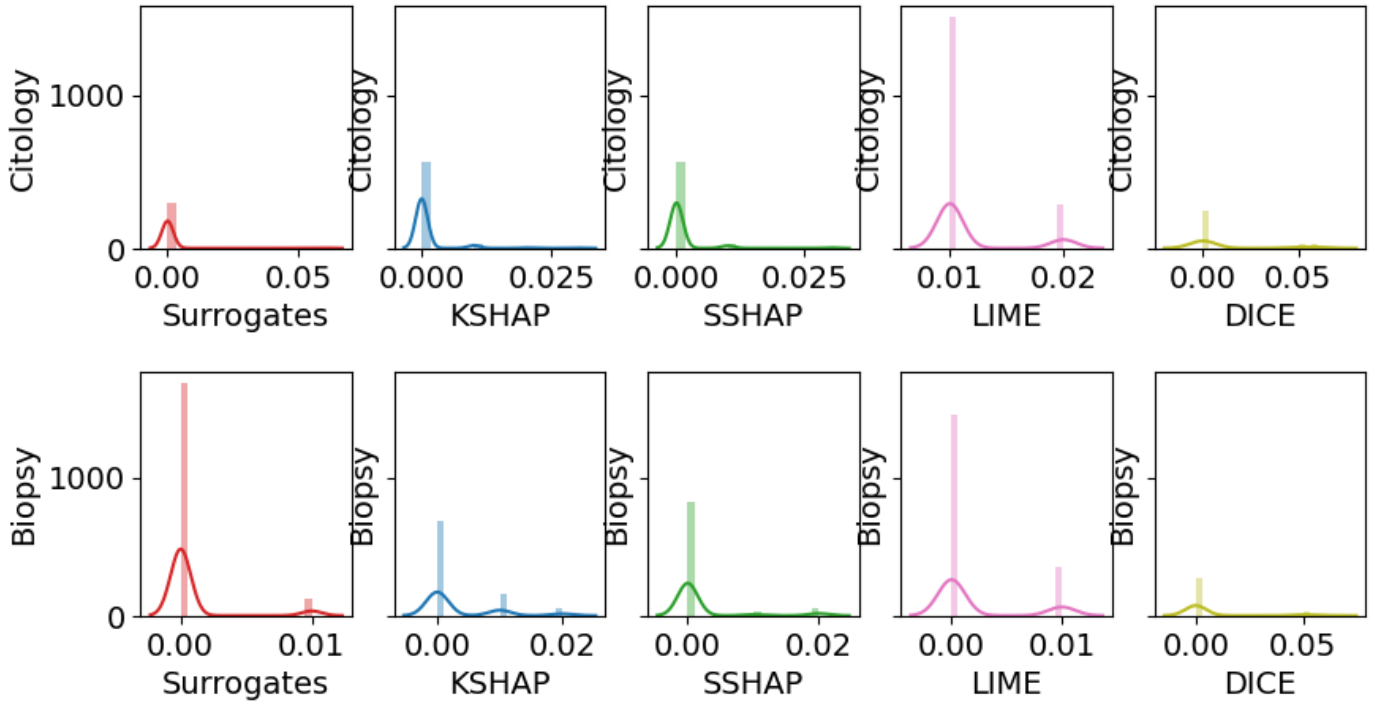

rmonal Contraceptives
100
0
0.0
0.0
Surrogates
0.0
0.1
KSHAP
0.0
0.1
SSHAP
0.000
0.025
LIME
0.0
0.1
DICE
IUD
1000
0
0.00
0.05
Surrogates
0.0
0.1
KSHAP
0.0
0.1
SSHAP
0.01
0.02
LIME
0.0
0.1
DICE
IUD (years)
200
100
0
0.0
0.2
Surrogates
0.00
0.25
KSHAP
0.00
0.25
SSHAP
0.00
0.05
LIME
0.00
0.05
DICE
STDs
1000
0
0.00
0.01
Surrogates
0.00
0.01
KSHAP
−0.5
0.0
0.5
SSHAP
0.00
0.01
LIME
0.0
0.1
DICE
STDs (number)
1000
0
0.00
0.05
Surrogates
0.00
0.05
KSHAP
0.00
0.01
SSHAP
0.01
0.02
0.03
LIME
0.0
0.5
DICE


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


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


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


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



In [104…
```
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 0.21
Surrogates 0.22
KSHAP 1.06
KSHAP 0.11
SSHAP 1.69
SSHAP 0.17
LIME 0.53
LIME 0.04
DICE 2.04
DICE 0.29
```

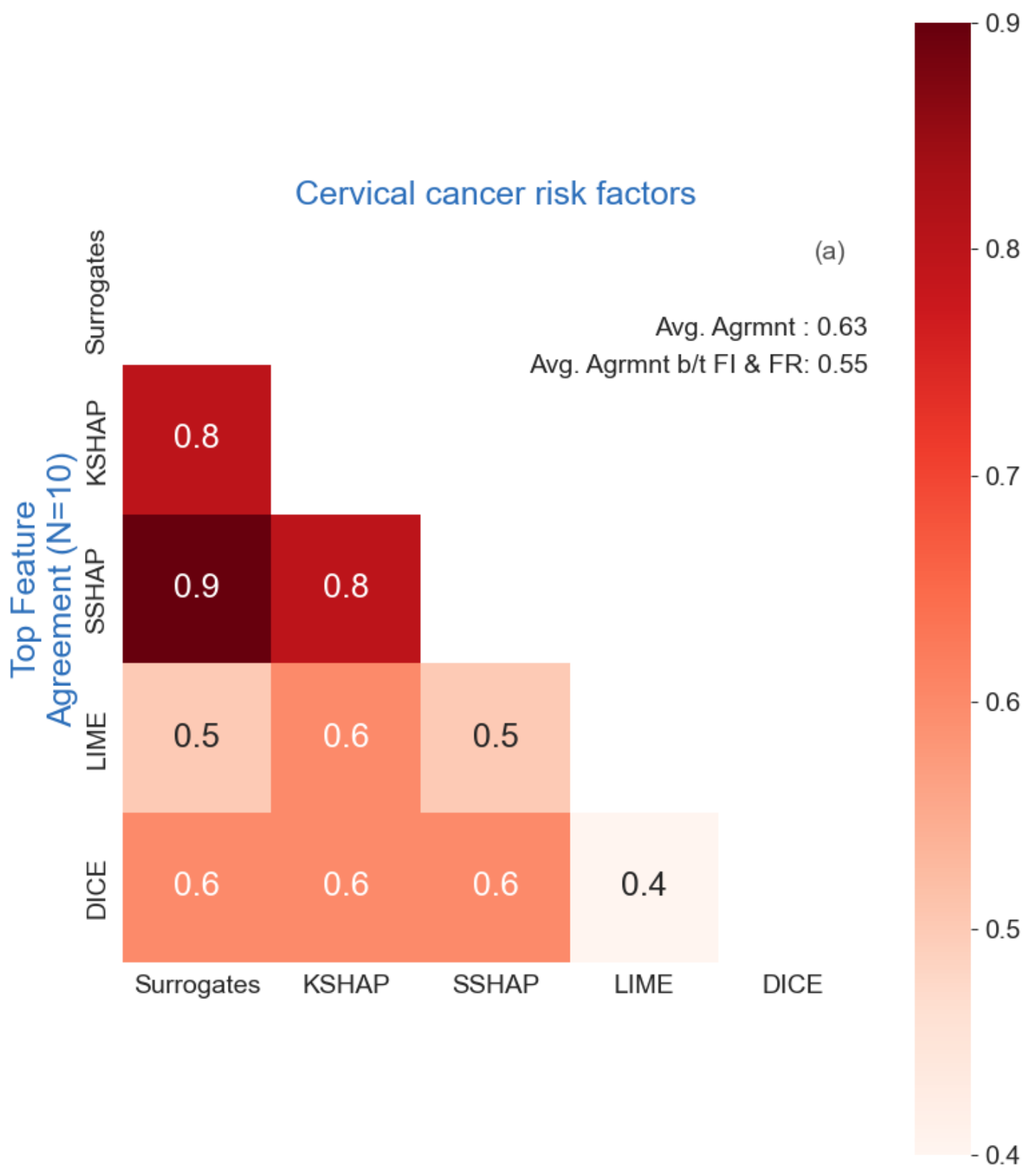
In [104…
```
xpl.plot.stability_plot(selection=[0, 1, 3])
fig_image=xpl.plot.stability_plot()
#plt.xlabel("Local Surrogates")
plt.savefig('stabplot.png')
```

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

Feature and Rank Disagreement

In [104…
```
feature_agree, rank_agree = compute_matrices(weights, instance)
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
```

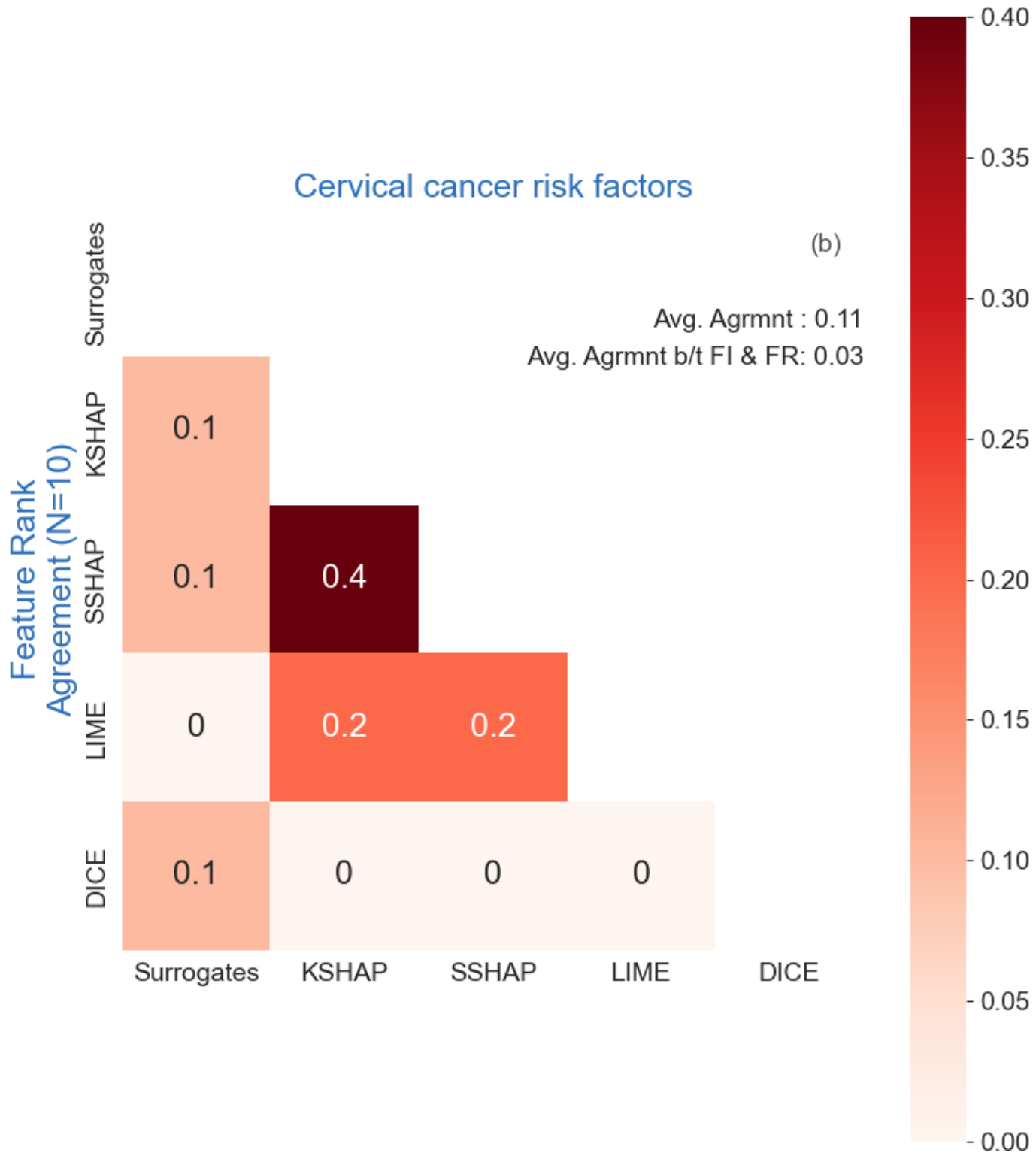
```
    ax.text(0.95,
                0.95,
                f"(a)",
                fontsize=14,
                alpha=0.8,
                ha="center",
                va="center",
                transform=ax.transAxes,
            )
    data=corr
    avg = np.mean(data[mask==0])
    text = f'Avg. 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 [104…

```
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()
```



Results

Table of results

Experiments	Original Paper Expectations (Hypothesis and Results)	Results	Discussions
HPV is the most important risk factor for cervical cancer	All SHAP models primarily rely on 1 feature and that is HPV presence to predict cervical cancer	We too see a similar result where SHAP models heavily depend on 1 feature but Surrogates and DICE use 4 to 10 different features to make their predictions	Similar results as the original paper
Faithfulness - removal of	LIME is seen as the model with most faithful	We see LIME perform well upto 40% of feature while the accuracy of	The results were not identical primarily becuase the original

features doesn't affect model's accuracy	explanations while TreeSHAP has the lowest faithfulness score	DICE model falls drastically till removal of 30% of features	solution was using a csv to which we didnt have access to validate and use the data present in it.
Trained five different models - LR, RF, MLP, SVM and KNN with CV=10	The original paper mentioned RF as the best performing ML model for cervical cancer	RF, LR and MLP produced the best result with a score of 1 on F1, Accuracy, Precision, Recall and AUROC	
Consistency between pairs of explainable AI models	Tree SHAP and Sampling SHAP is the most consistent pair, while Local Surrogate and DiCE is the least consistent pair.	We see that KSHAP and SSHAP having the least score of 0.025 and Local Surrogate and DiCE is the least consistent pair with a score of 1.4	All SHAP models have similar scores and are quite consistent
Stability of explanations	LIME, KSHAP and local surrogates are found to be least stable with a lot of variability	LIME, KSHAP and local surrogates are found to be least stable with a lot of variability in our results as well	Similar result as the original paper

Ablation Study

1. Carried out different sampling techniques like Random over sampler, SMOTE, using original data and using ADASYN which is mentioned in the paper. We see that random forest, SVM and MLP generally perform well while KNN performs the worst on the given sampled data.
2. Used MLP instead of RF as mentioned in paper and we were able to justify similar outcomes when generating the compactness, stability, faithfulness metrics when using RF model

Discussion

1. Implications of the experimental results, whether the original paper was reproducible, and if it wasn't, what factors made it irreproducible

   Most of the results mentioned in the original paper were reproducible but some of them didnt exactly match the values mentioned in the paper. This was probably because there were no clear steps mentioned on how they had reproduced the results. Also some of the steps had links to person drive and csv files to which we couldnt gain access to making it difficult to verify the data in those sources resulting in additional preprocessing steps and assumptions to be made.

1. What was easy?

   The dataset and preprocessing steps were easy- because the dataset used was not huge (contained around 850 records and was around 65KB in size) and was easy to load and process as compared to some other datasets which could span over 1GB of size.

1. What was difficult?

   The explainable AI (XAI) steps were not clearly structured in an ordered manner due to which some of the steps failed and additional preprocessing steps needed to be added to make sure that the data was compatible for processing. Also for some of the XAI models there were mentions of some csv files which seemed to point to personal drive links and were not accessible to verify and utilize the same for reproduction of the results. This made it slightly more difficult to reproduce exact results and some assumptions had to be made (like the csv mentioned was the same as the dataset from UCI)

1. Recommendations to the original authors or others who work in this area for improving reproducibility

   The refactoring and some of the ablations carried out in this project will definitely help the original authors and anyone else working on this project to have a structured way of executing the steps. The explainable code needs to be refactored in the original solution to achieve desirable results. Also personal drive links need to be replaced by public ones and libraries like gdown should be used to make available any extra datasets or csvs being used which are not mentioned in the original paper.

References:

1. Ayad, Wafa & Bonnier, Thomas & Bosch, Benjamin & Read, Jesse & Parbhoo, Sonali. (2023). Which Explanation Makes Sense? A Critical Evaluation of Local Explanations for Assessing Cervical Cancer Risk Factors Ecole polytechnique. 1-50. https://www.researchgate.net/profile/Wafa-Ayad/publication/374061335_Which_Explanation_Makes_Sense_A_Critical_Evaluation_of_Local_Explana Explanation-Makes-Sense-A-Critical-Evaluation-of-Local-Explanations-for-Assessing-Cervical-Cancer-Risk-Factors-Ecole-polytechnique.pdf
2. Data Source: https://archive.ics.uci.edu/dataset/383/cervical+cancer+risk+factors
3. https://www.kaggle.com/code/renadope/cervical-cancer-classification-99-4-recall
4. https://towardsdatascience.com/building-confidence-on-explainability-methods-66b9ee575514
5. https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.07-Error-Bars/