Project Info

Team: 94

Member: Qinxi Wang

Email: qinxiw2@illinois.edu

Github: https://github.com/QinxiW/DLH_Cervical_Cancer_Local_Explanations

Video: https://drive.google.com/file/d/1Qp-ybEQICxGYxTfFf9jYx_HAM7XHZgNO/view?usp=sharing

Introduction

Background of the problem

Ayad et al discuss the importance of identifying and assessing factors that increase the risk of cervical cancer for early detection and treatment. They point out that the results from ML algorithms often are like blackbox decisions, and hard for the clinical practitioner to understand and decide if they want to follow through with the cervical cancer diagnosis in model prediction or not. This is a difficult problem because the lack of transparency not only hampers the interpretability of results, but also raises concerns about the reliability and safety of integrating machine learning into critical healthcare decision-making processe.

To address this, the authors examine various local explanation techniques aimed at elucidating a model's predictions for specific instances. The state of the art methods they used includes SHAP, TreeSHAP, LIME, DICE and other related interpretability methods.

Paper explanation

They thus propose a framework to evaluate the quality of various explanations regarding cervical cancer risk, which involves computing different metrics to identify the most suitable explanation for assessing cervical cancer risk. In their experiments specifically, they provide empirical study analyzing the performances of different methods for explaining cervical cancer risk factors, then for each method they contextualize how various formulations of these explanations could be suitable for different patient scenarios and when they might not be appropriate, and provide recommendations to practitioners for utilizing different types of explanations in assessing and determining key factors affecting cervical cancer risk.

The innovations lies in the proposed evaluation framework for the effectiveness of each

explanation techniques specifically for predicting cervical cancer risk, evaluated using RemOve And Retrain (ROAR) metrics where a number of works deem stability, consistency, compactness and faithfulness as important facets of interpretability for healthcare domains overall. The paper offers a critical analysis of existing local interpretability methods for explaining cervical cancer risk factors, aiming to assist clinicians in choosing appropriate explanations for different patient contexts. The approach also have application contribution, as it enables selecting the suitable explanations to reason about cervical cancer diagnosis and can be extended to other healthcare applications and areas where explanation is critical, and paved the future work for a user study with clinicians to assess how features weighted sums may lead to context-specific explanations.

Scope of Reproducibility:

The paper draws the following conclusions from the experiments. We wonder if we can first reproduce, and then test if the hypotheses are valid:

1. All explainers agree on HPV being the most important risk factor for cervical cancer.
2. SHAP(SHapley Additive exPlanations) explainers have exactly the same top 10 most important features for Patient 1.
3. The most unstable explainers are those that depend on creating local neighborhoods
4. No single explanation performs optimally across patients and metrics
5. The top 30% important features given by TreeSHAP have the most impact on model learning.
6. LIME(Local Interpretable Model-Agnostic Explanations) is the most stable, most robust to removal of features, and SHAP the most consistent in terms of feature and rank agreements.

We hope to test these hypotheses by repeat the process the authors have done in the paper, where we will first generate and compare the quality of each of the local explanation techniques, we will then examine the top features produced by each of the techniques, and assess the decrease in accuracy of models when successively removing a fraction of the top features for each explanation and see if the above hold. The paper came with various results, which we will also cover in the later section; for illustration purpose, the scope of reproducibility will be focused on Features and Methods -

```
import gdown
im1_url = 'https://drive.google.com/file/d/1zL-wxxPcY84t0ape_XX-yGVthttv7drQ/view
output1 = "features.png"
gdown.download(im1_url, output1, fuzzy=True)
```

```
im2_url = 'https://drive.google.com/file/d/1--32U-RBk9zLsqj1yLNA4zCCaQ76ntya/view
output2 = "methods.png"
gdown.download(im2_url, output2, fuzzy=True)
```

```
Downloading...
From: https://drive.google.com/uc?id=1zL-wxxPcY84t0ape_XX-yGVthttv7drQ
To: /content/features.png
100%|██████████| 293k/293k [00:00<00:00, 69.4MB/s]
Downloading...
From: https://drive.google.com/uc?id=1--32U-RBk9zLsqj1yLNA4zCCaQ76ntya
To: /content/methods.png
100%|██████████| 351k/351k [00:00<00:00, 66.9MB/s]
'methods.png'
```

```
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
img = cv2.imread(output1)
img2 = cv2.imread(output2)

cv2_imshow(img)
cv2_imshow(img2)
```

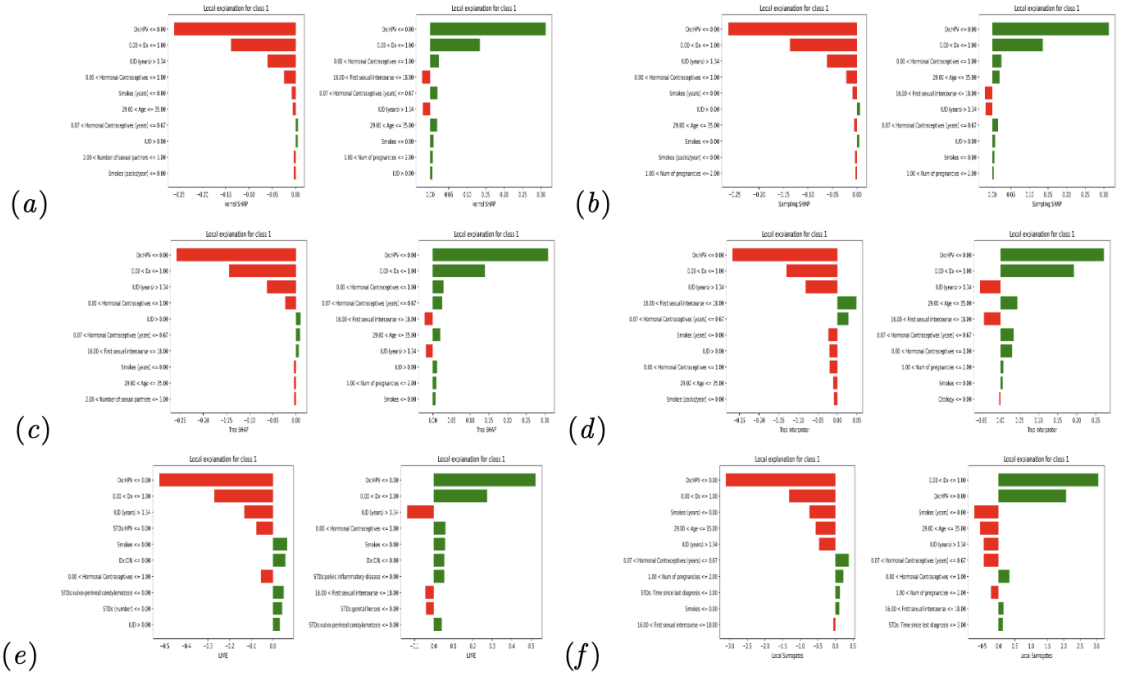


Figure 3: Comparison of feature importance for two similar patients (Patient 0 and Patient 1). On the left, Patient 0 diagnosed as not having cancer (Dx:Cancer=0) and on the right, Patient 1 diagnosed with cancer (Dx:Cancer=1). The key driving factors for this patient are similar to those of a patient with cancer.

Compacity of explanations: Compacity of explanations:

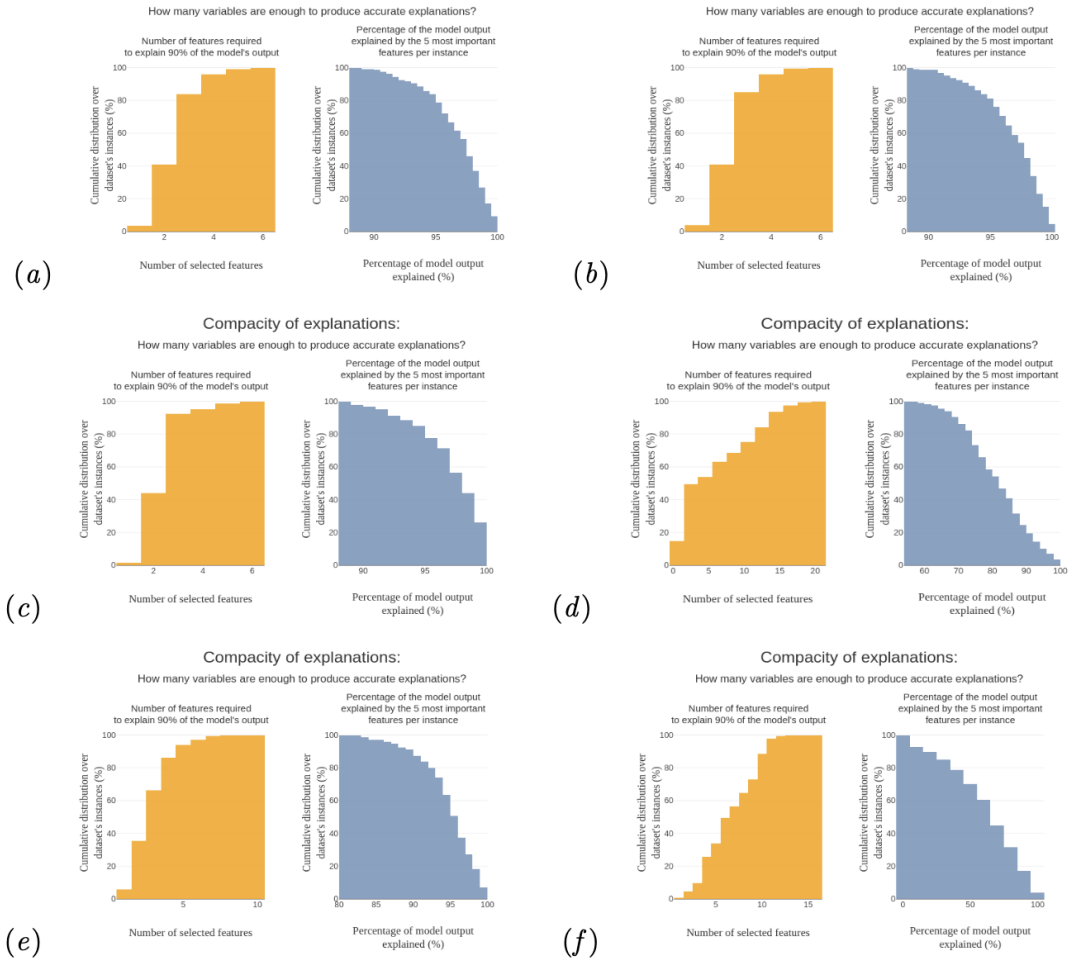


Figure 5: Compactness of the explanations generated by (a) Kernel SHAP, (b) Sampling SHAP, (c) Tree SHAP, (d) LIME,(e) Tree Interpreter, (f) Local surrogates.

Methodology

This is the core part of the project draft. Here we will go over four subsections **data**, **model**, **training** and **evaluation** in our experiment and analysis work so far.

```
# !pip3 freeze > requirements.txt

!python --version
```

```
Python 3.10.12
```

Environment

In this notebook we use Python version of 3.10.12.

Dependencies/packages needed are the following, you can uncomment and run via pip install. They are also freezed to requirement.txt at https://github.com/QinxiW/DLH_Cervical_Cancer_Local_Explanations/blob/main/requirements.txt.

```
# !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 spectralcluster
# !pip install -U kaleido
# !pip install gdown
# !pip install shapash

import pandas as pd
import seaborn as sns
import numpy as np
from numpy import arange
```

```
import itertools
import random

import matplotlib.pyplot as plt
# from matplotlib import pyplot as plt

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import plot, iplot, init_notebook_mode
from typing import List
from scipy.stats import spearmanr, bernoulli

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler,StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, cross_val_scor
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_recall_fscore_support, roc_auc_score, accur
from sklearn.inspection import permutation_importance, PartialDependenceDisplay,

from imblearn.over_sampling import SMOTE, ADASYN, RandomOverSampler

from interpret_community.mimic.mimic_explainer import MimicExplainer
from interpret_community.mimic.models import LinearExplainableModel
from interpret.blackbox import MorrisSensitivity

import shap
import lime
from lime import lime_tabular
from treeinterpreter import treeinterpreter as ti
import dice_ml

import warnings
warnings.filterwarnings('ignore')
```

Data

The dataset comes with 36 columns and 859 rows. Among them we have 858 patients and 35 features, including demographic information such as age and number of pregnancies, clinical tests such as Hinselmann and Citology, sexually transmitted diseases such as HPV and AIDS, and diagnosis taken by the patients such as HPV and CIN. All the data formats in the dataset are numbers or string, and all structured data, so from a data size perspective the memory and

are numbers or string, and all structured data, so from a data size perspective the memory and computation needed for working with is feasible.

- Source of the data:

  The dataset used in the paper is available from the UCI repository (Fernandes et al., 2017). It is open source, and also available on Kaggle which we could access and download directly at https://www.kaggle.com/datasets/loveall/cervical-cancer-risk-classification?resource=download

- Statistics:

  Here are the high-level subtopics we will cover in the code snippets below.

1. ovreview of unique values of each column
2. data distribution analysis, in the order of: top features correlated with cancer, Pregnancy Distribution by Age, Correlation of diagnoses,
3. more in-depth data analysis, including number of sex partner by age buckets created, Proportions of women who have Cervical Cancer / HPV, Hormonal Contraceptives and Cervical Cancer.

- Data process:

  Here are the high-level subtopics we will cover in the code snippets below.

1. convert all features to numeric column types if they are not already
2. bucket ages into 8 categories from "Child" to "70+"
3. aggregating various different std columns into a "total_std" new col.
4. dealing with imbalanced data
5. create train/test dataset for later training and evaluation

- Illustration:

  We will be printing results, plotting figures for illustration for each of the sub-Statistics and Data processing in order below.

Data download instruction

The source data is open source, and also available on Kaggle which we could access and download directly at https://www.kaggle.com/datasets/loveall/cervical-cancer-risk-classification?resource=download

you can also directly load the dataset into the notebook with gdown, and run the following code block.

```
import gdown
url = 'https://drive.google.com/file/d/1lfmBWeWp9L0RgKYqySnyWsm3ryfceLnN/view?usp
output = "kag_risk_factors_cervical_cancer.csv"
gdown.download(url, output, fuzzy=True)
```

```
Downloading...
From: https://drive.google.com/uc?id=1lfmBWeWp9L0RgKYqySnyWsm3ryfceLnN
To: /content/kag_risk_factors_cervical_cancer.csv
100%|██████████| 102k/102k [00:00<00:00, 54.2MB/s]
'kag_risk_factors_cervical_cancer.csv'
```

Statistics

```
# load dataframe
risk_factor_df = pd.read_csv(output, delimiter=',', encoding='utf-8', engine='pyt
risk_factor_df.head()
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)	Hor Contracep
0	18	4.0	15.0	1.0	0.0	0.0	0.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	
2	34	1.0	?	1.0	0.0	0.0	0.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	

5 rows × 36 columns

```
# calculate statistics
def calculate_stats(raw_data):
  # implement this function to calculate the statistics
  # it is encouraged to print out the results
  return None

# process raw data
def process_data(raw_data):
    # implement this function to process the data as you need
  return None

# processed_data = process_data(raw_data)

''' you can load the processed data directly
```

```
processed_data_dir = '/content/gdrive/My Drive/Colab Notebooks/<path-to-raw-data>
def load_processed_data(raw_data_dir):
  pass

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


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

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
 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
 '9.0']
dtype for Number of sexual partners is :object
------------------------------------------------------------------------------
Unique Values for First sexual intercourse:['15.0' '14.0' '?' '16.0' '21.0' '
 '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' '?'
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
 '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' '0.513
 '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' '
```

```
 '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'
 '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
--------------------------------------------------------------------------------
```

```
# 840 non HPV, 18 HPV

risk_factor_df[risk_factor_df['Dx:HPV'] == 1]
risk_factor_df['Dx:HPV'].value_counts()
```

```
Dx:HPV
0    840
1     18
Name: count, dtype: int64
```

```
# no nulls
risk_factor_df.isna().sum()
```

```
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
```

```
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
```

```
risk_factor_df.describe()
```

	Age	STDs: Number of diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	Dx	Hinselm
count	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.000
mean	26.820513	0.087413	0.020979	0.010490	0.020979	0.027972	0.040
std	8.497948	0.302545	0.143398	0.101939	0.143398	0.164989	0.197
min	13.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	20.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	25.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
max	84.000000	3.000000	1.000000	1.000000	1.000000	1.000000	1.000

Data Processing

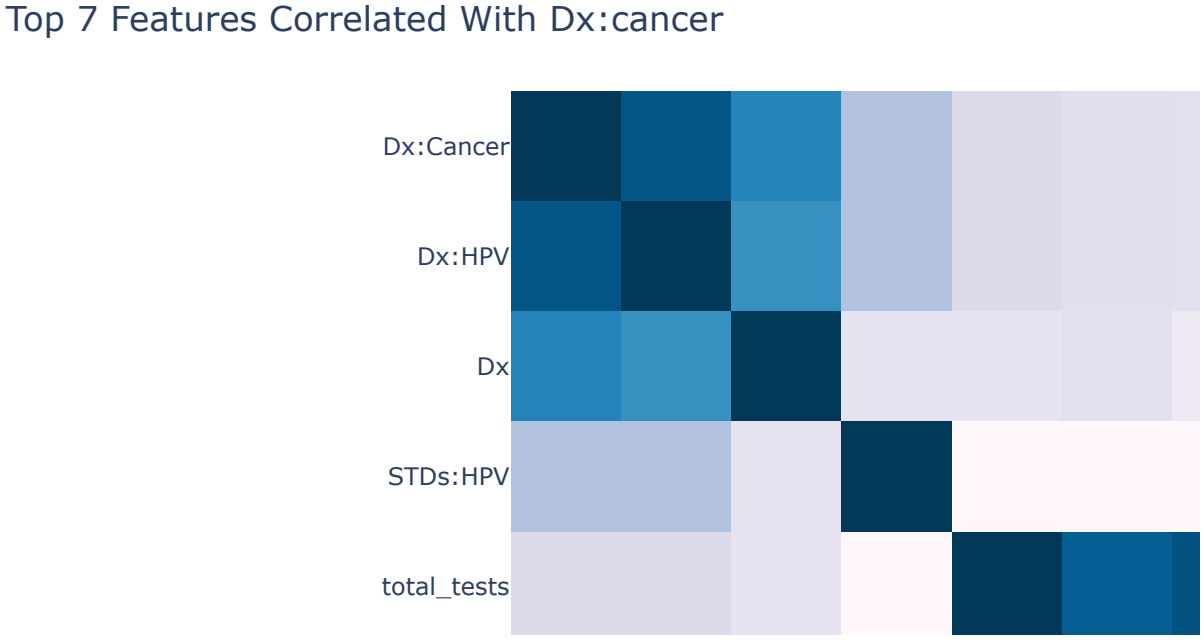
```
#these columns are not of type object, but are of type numeric
cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Num
                   'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contracepti
                   'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STDs
                   'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:v
                   'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:p
                   'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AID
                   '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_num
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))
```

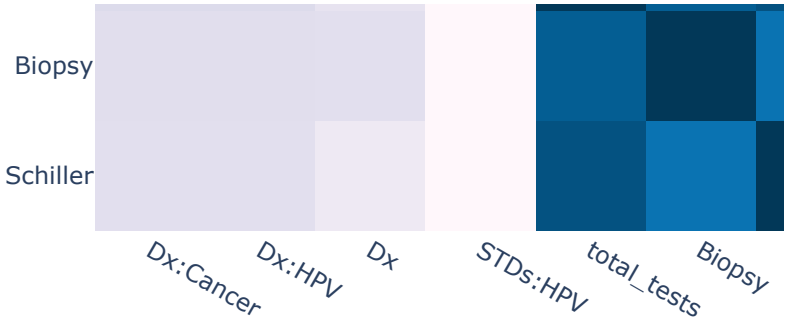
```
def age_cat(age):
    if age < 12:
        return "Child"
    elif age < 20:
        return "Teen"
    elif age < 30:
        return "20's"
    elif age < 40:
        return "30's"
    elif age < 50:
        return "40's"
    elif age < 60:
        return "50's"
    elif age < 70:
        return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

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()

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

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)
```

Data Analysis

```
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).ca
fig.show()
```



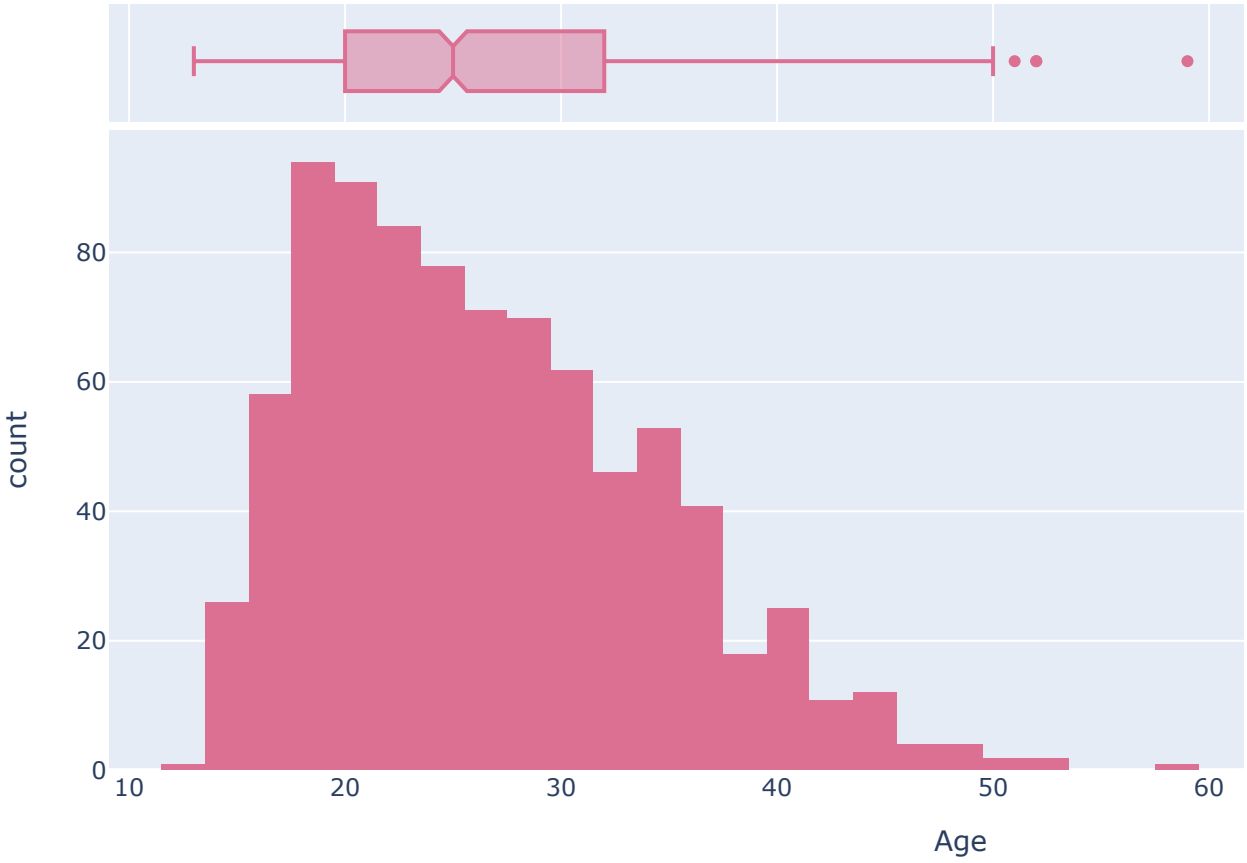



```
def stats(x):
    temp1=(df[[x,label]].value_counts(normalize=True).round(decimals=3)*100).rese
    Coloumn_To_Aggregate=[x,label]
    df6=pd.merge(df.groupby(Coloumn_To_Aggregate).size().reset_index(name='ind_si
                 df.groupby(Coloumn_To_Aggregate[:-1]).size().reset_index(name='T
    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)

df=risk_factor_df
label='age_cat'

stats('Dx:Cancer')
```

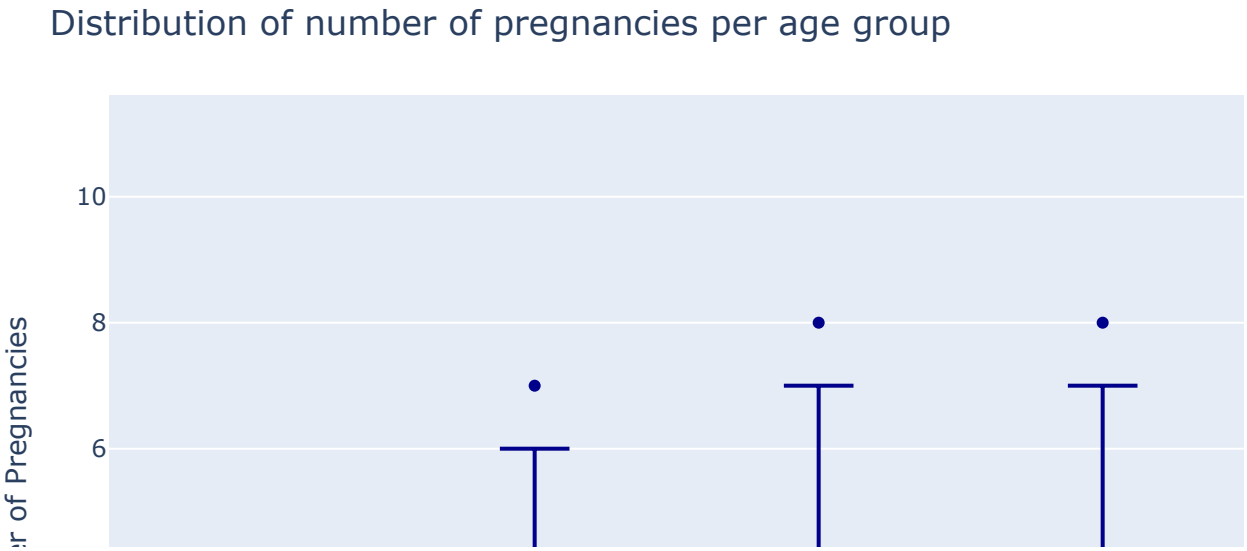
	proportion		Category_Percent	
Dx:Cancer	0	1	0	1
age_cat				
20's	45.3	0.6	46.31	27.78
30's	24.7	0.9	25.24	44.44
40's	6.2	0.3	6.31	16.67
50's	0.5	0.1	0.48	5.56
70+	0.5	NaN	0.48	NaN
Teen	20.7	0.1	21.19	5.56

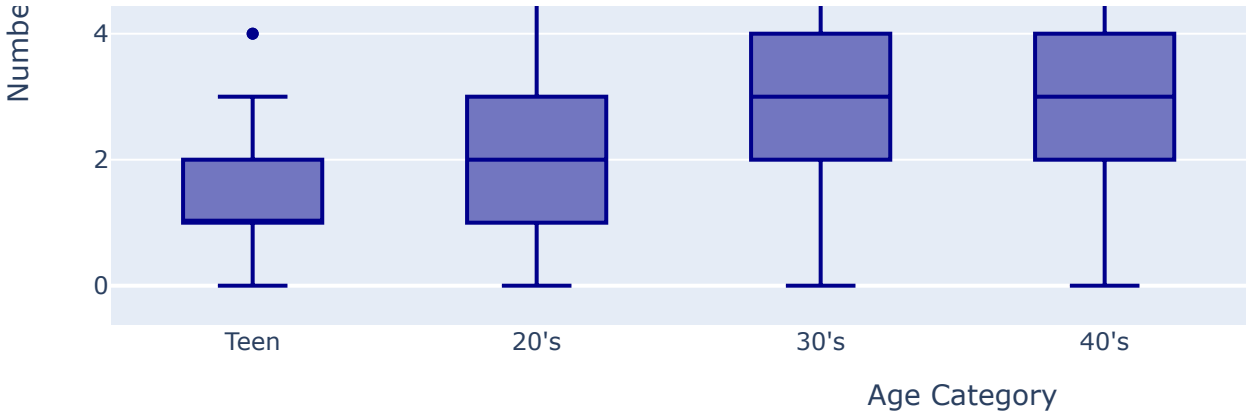
```
age_dist = px.histogram(risk_factor_df, x="Age", marginal="box", color_discrete_s
age_dist.update_layout(title="Age distribution")
age_dist.show()
```

Age distribution

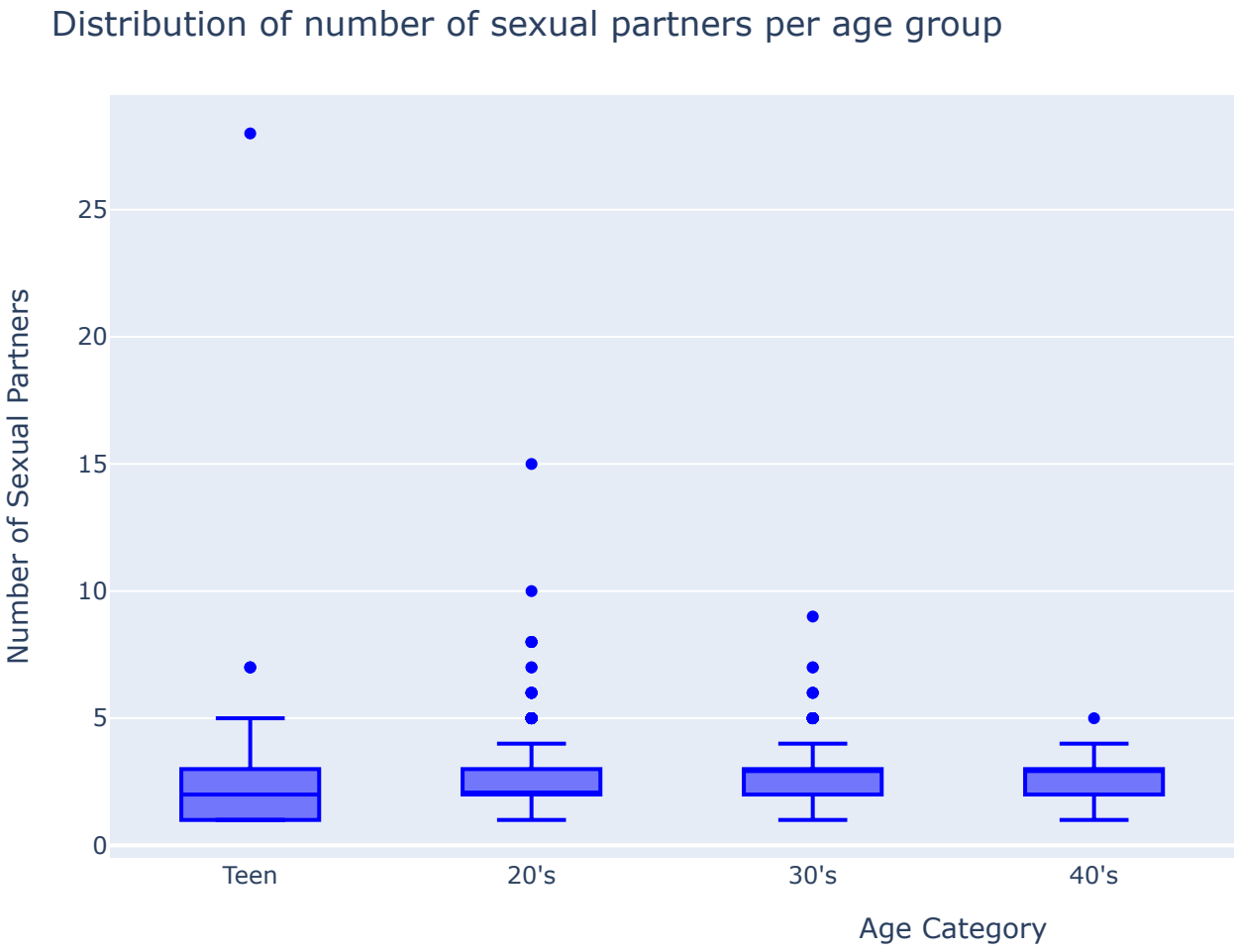



```
age_preg_bar = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="age
                      color_discrete_sequence=["darkblue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forti
                                       "Seventy and over"])
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 g
age_preg_bar.show()
```

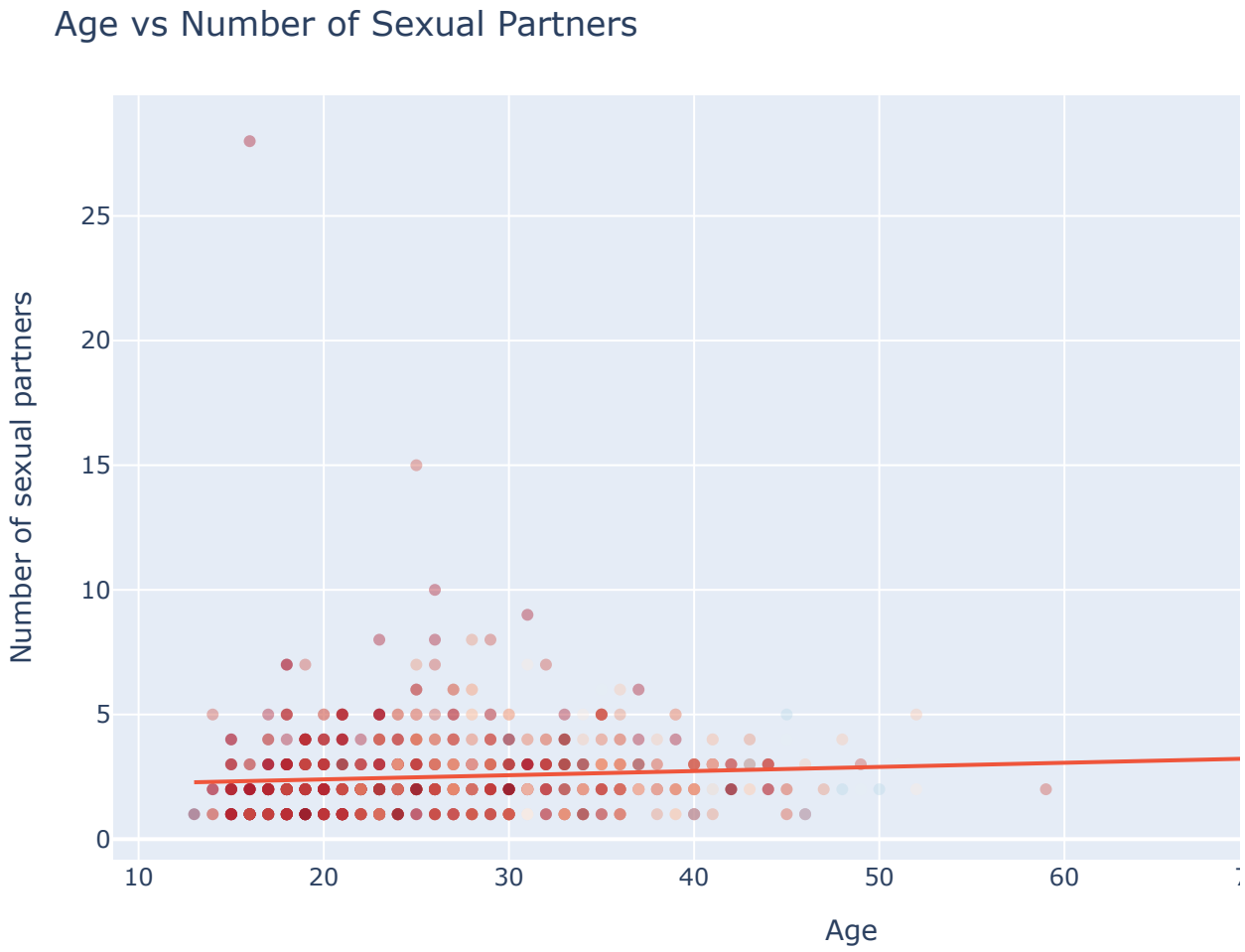




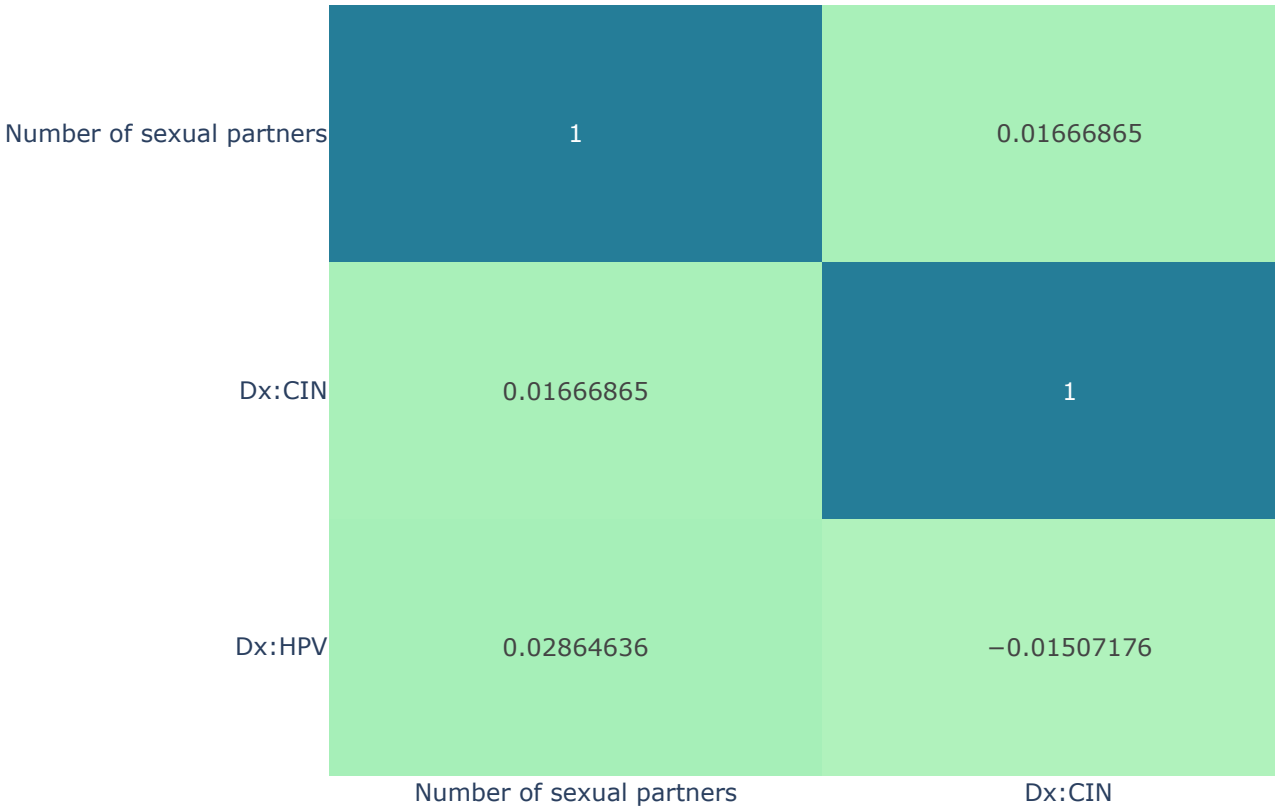

```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True)
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forti
                                       "Seventy and over"])
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 partne
age_num_sex_partners.show()
```



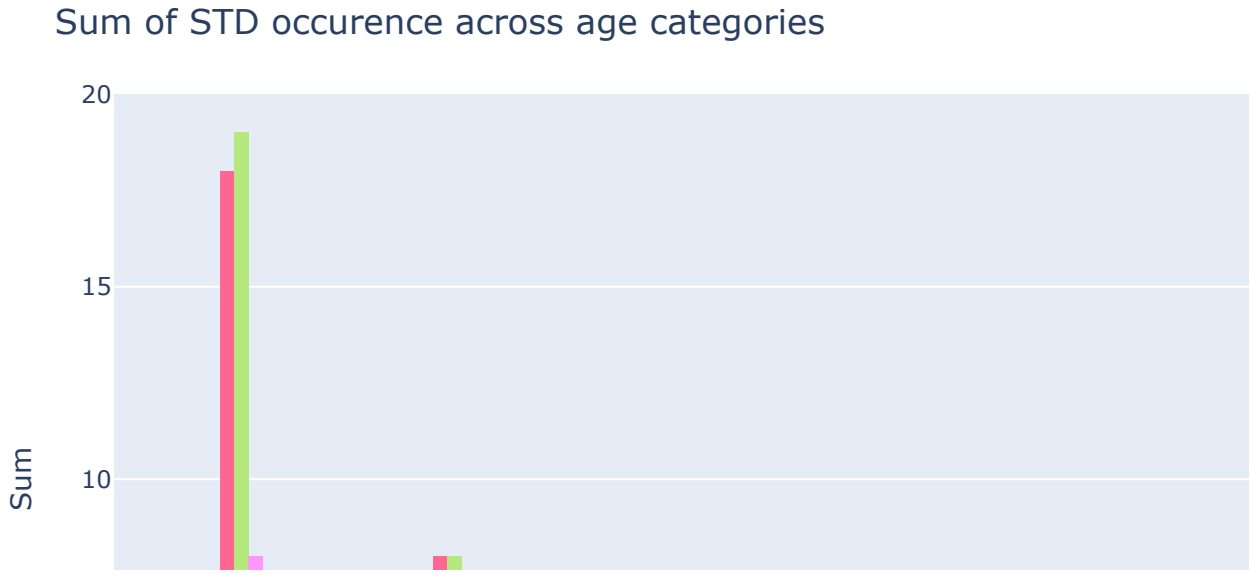
```
age_num_sex_partners = px.scatter(risk_factor_df, x="Age",
                                  y="Number of sexual partners",
                                  trendline="ols",
                                  opacity=0.4,
                                  color="Num of pregnancies",
            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

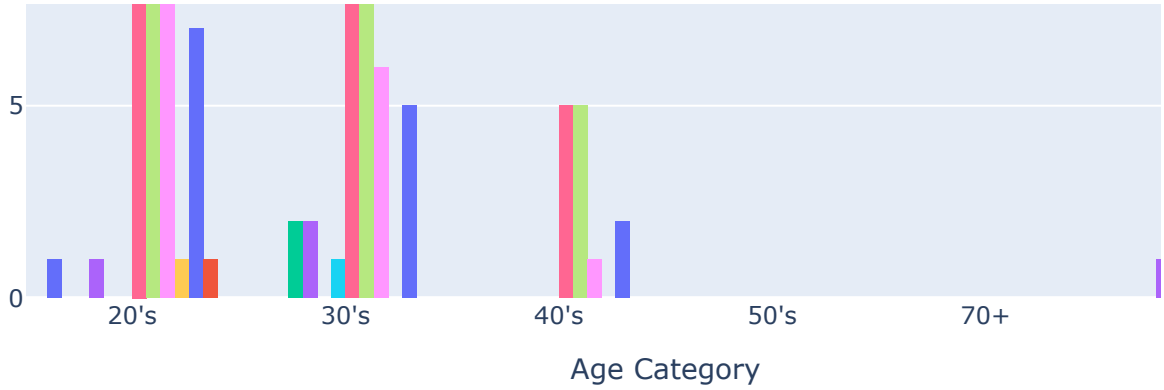



```
label = 'Number of sexual partners'
diagnoses_cols = [label,
                  'Dx:CIN',
                  'Dx:HPV']
diagnoses_corr_matrix = risk_factor_df[diagnoses_cols].corr()

diagnoses_heatmap = px.imshow(diagnoses_corr_matrix, aspect="auto", color_continu
diagnoses_heatmap.show()
```

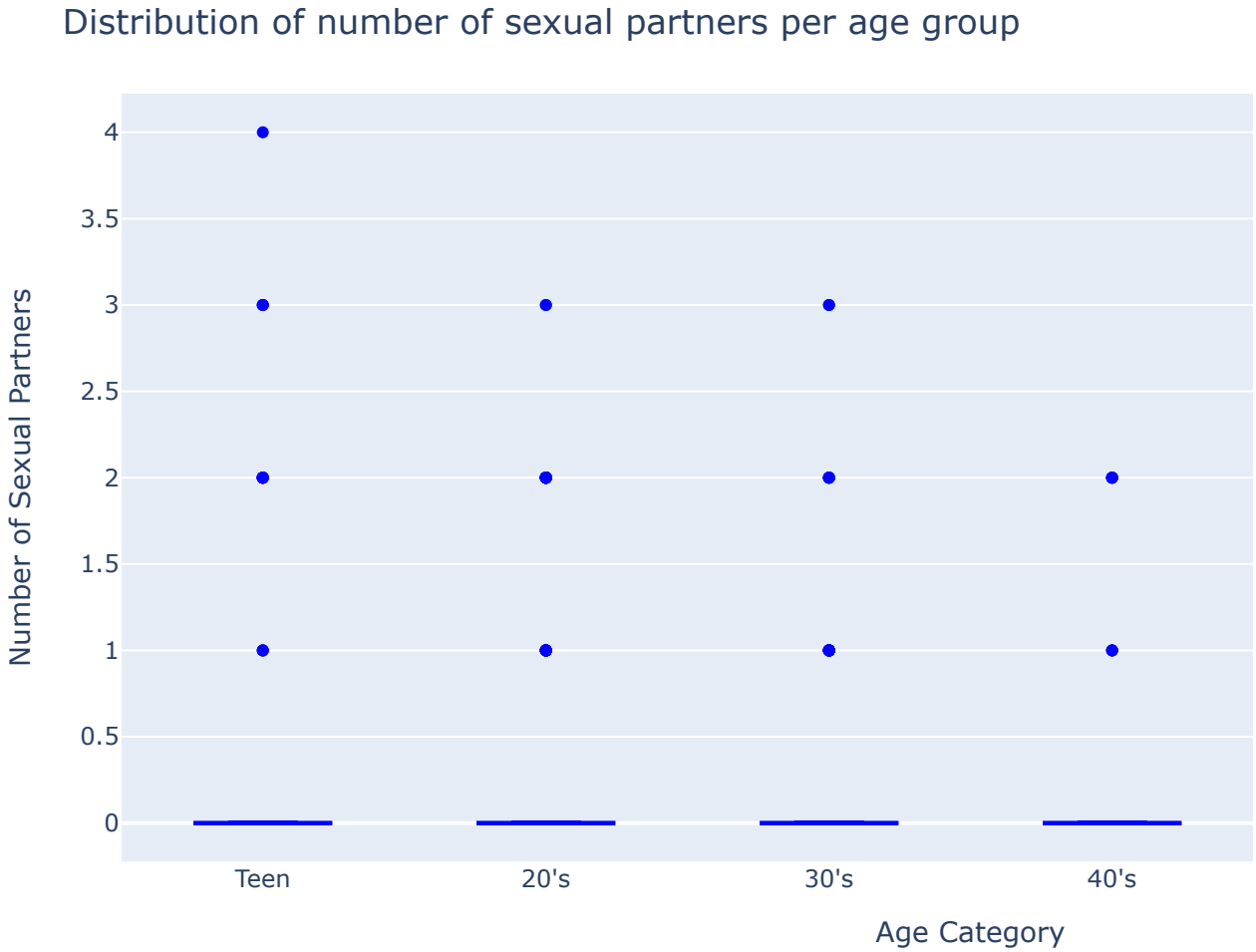



```
fig = px.histogram(std_agg, x="age_cat", y=list(std_cols), barmode="group", histf
fig.update_layout(title="Sum of STD occurence across age categories")
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Sum")
fig.show()
```



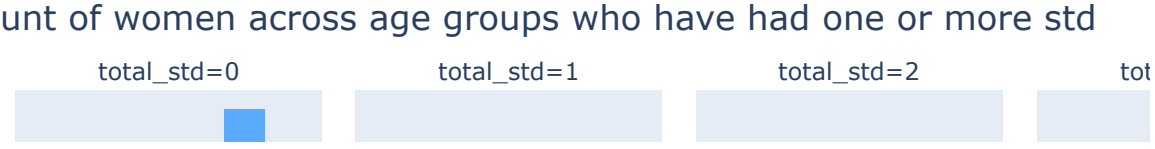


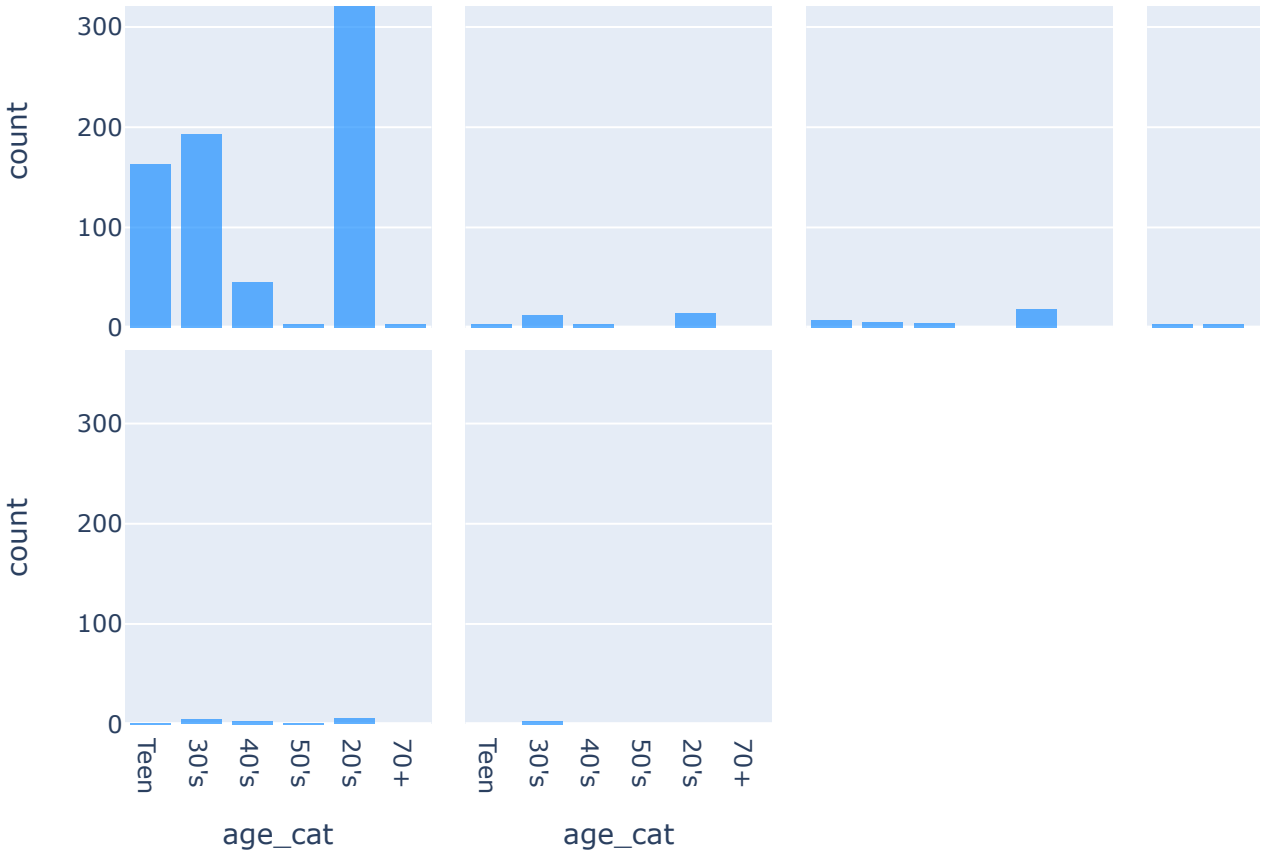

```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True)
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forti
age_num_sex_partners.update_xaxes(title="Age Category")
age_num_sex_partners.update_yaxes(title="Number of Sexual Partners")
age_num_sex_partners.update_layout(title="Distribution of number of sexual partne
age_num_sex_partners.show()
```



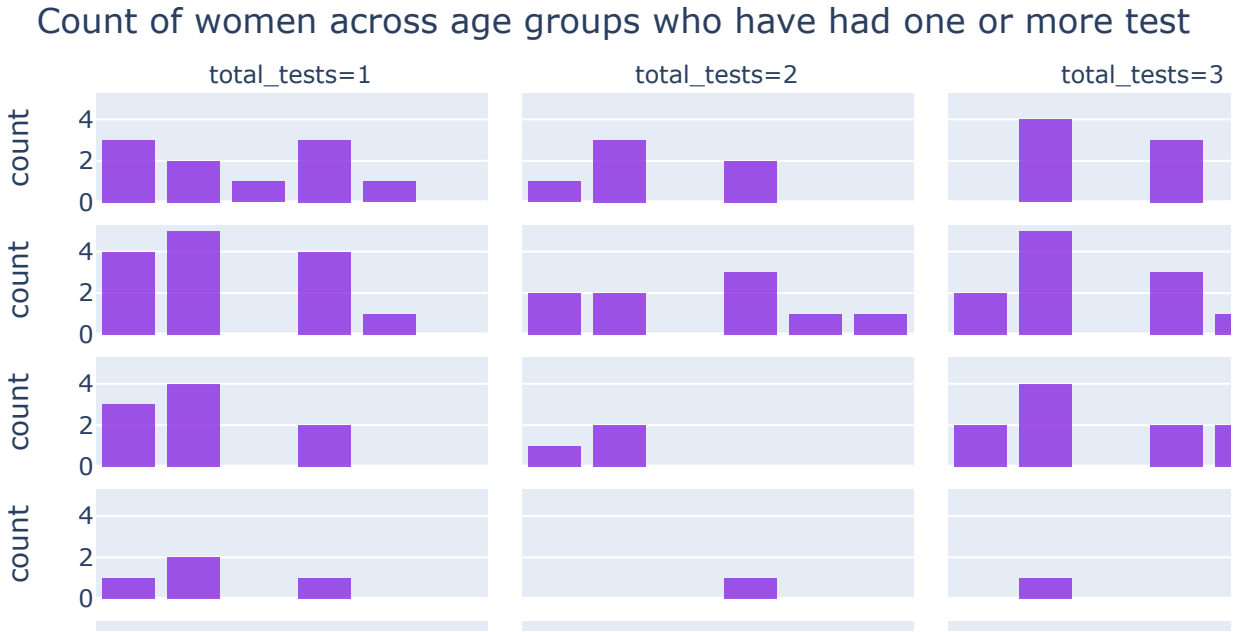
```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_st
                   x="age_cat",
                   facet_col="total_std",
                   facet_row=label,
                   color_discrete_sequence=["rebeccapurple"],
                   opacity=0.7)
fig.update_layout(title="Count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

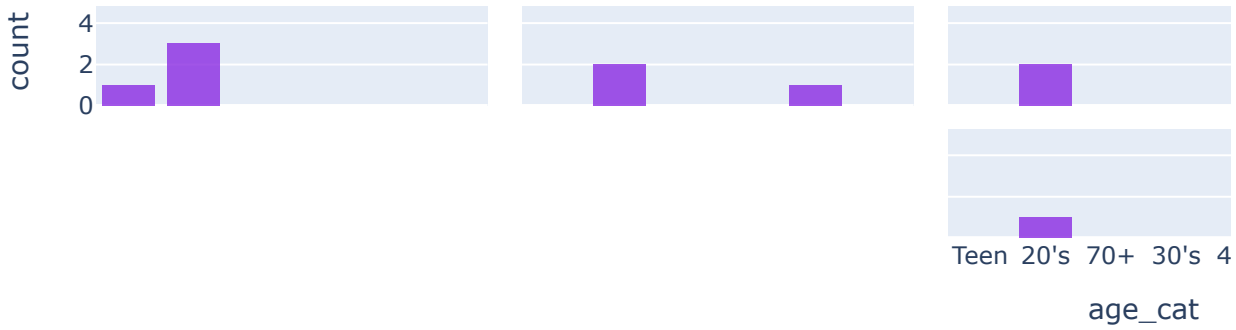

```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_st
                   x="age_cat",
                   facet_col="total_std",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["dodgerblue"],
                   opacity=0.7)
fig.update_layout(title="Count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



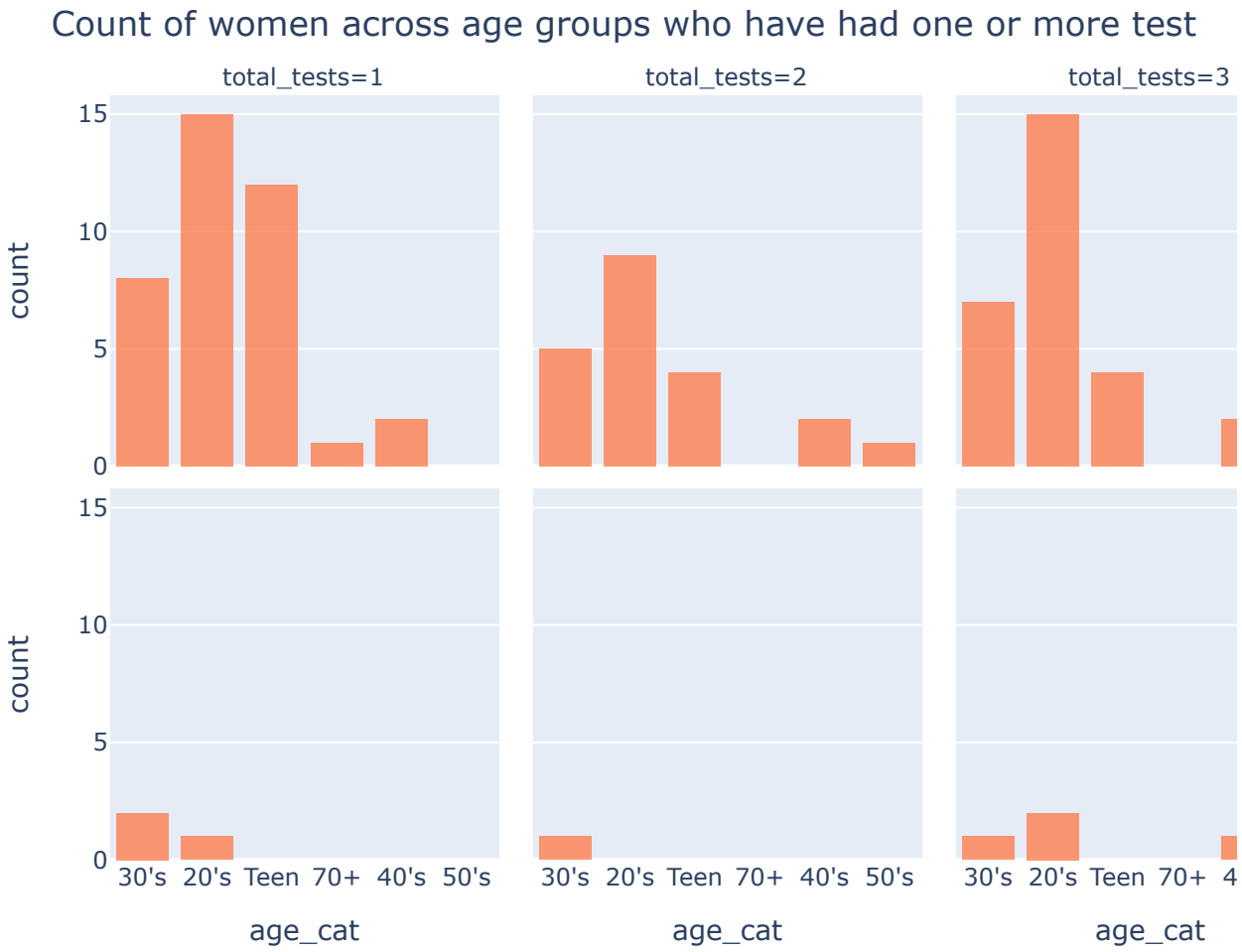



```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by="total_te
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row=label,
                   color_discrete_sequence=["blueviolet"],
                   opacity=0.8)
fig.update_layout(title="Count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

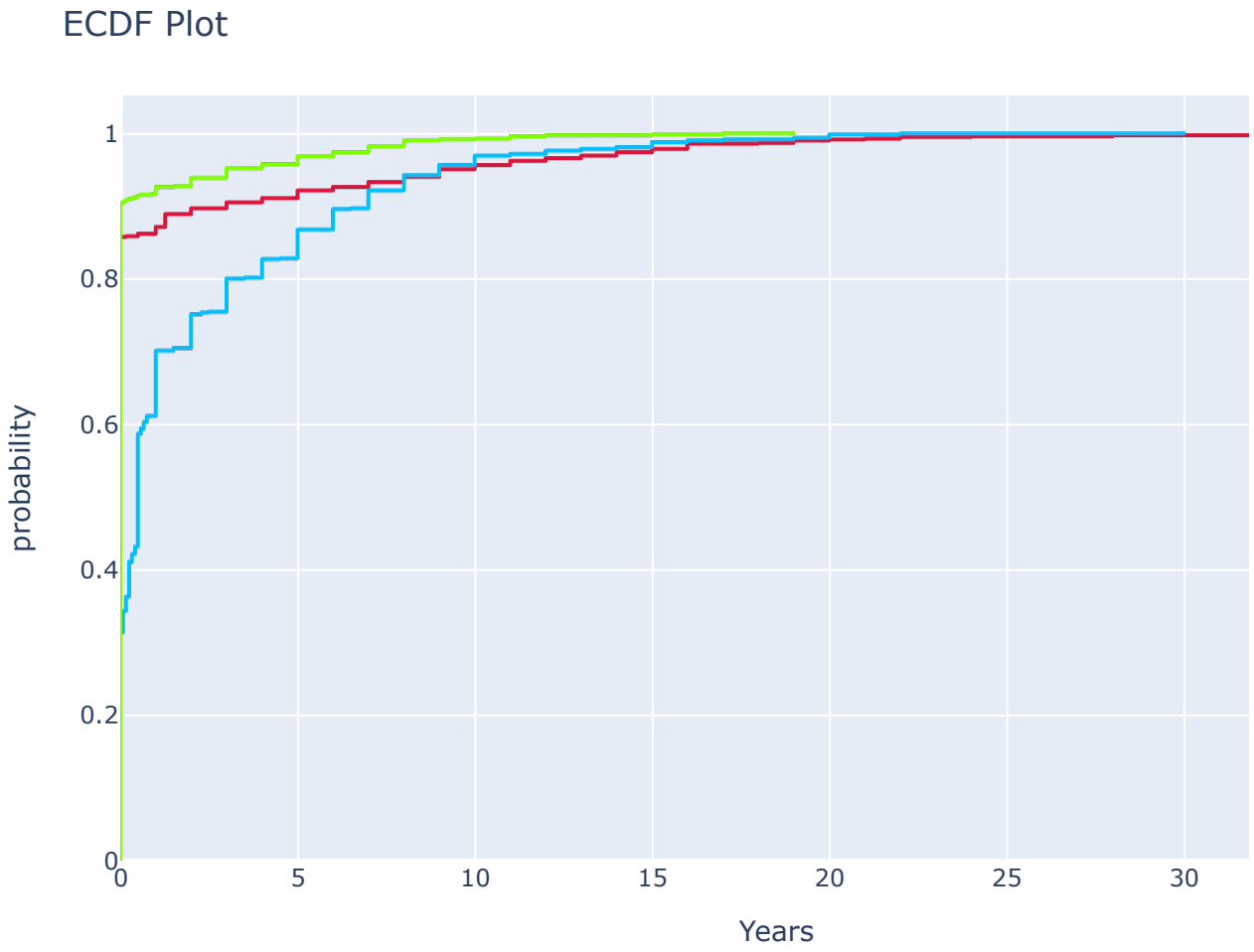





```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by=["total_te
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["coral"],
                   opacity=0.8)
fig.update_layout(title="Count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

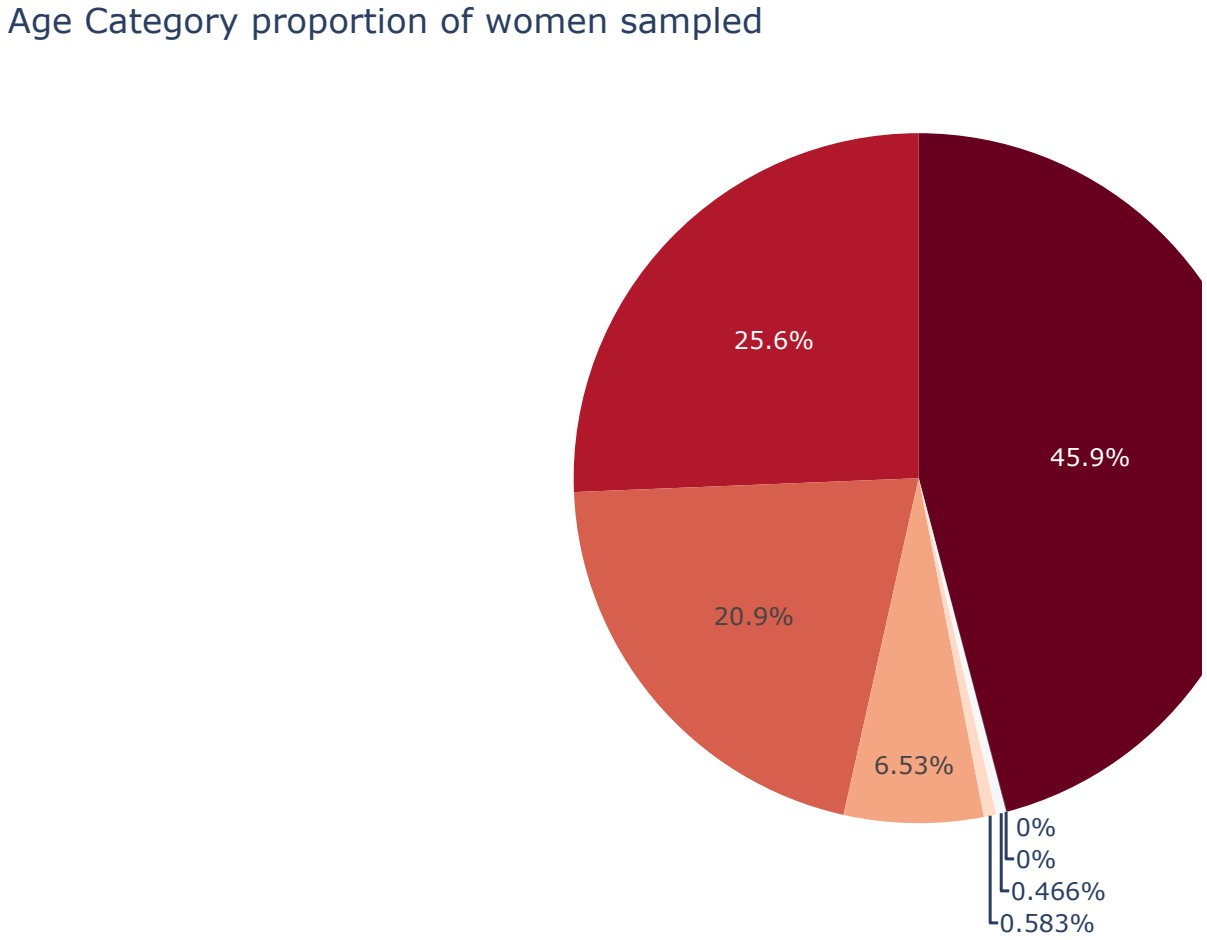


```
fig = px.ecdf(risk_factor_df, x=["Smokes (years)",
                                 "Hormonal Contraceptives (years)",
                                 "IUD (years)"],
              color_discrete_sequence=["crimson", "deepskyblue", "chartreuse"])
fig.update_xaxes(title="Years")
fig.update_layout(title="ECDF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




```
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"
for age_range, category in age_category_range.items():
    age_prop_dict[category] = risk_factor_df.query(age_range)[col].count() / len(
```

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

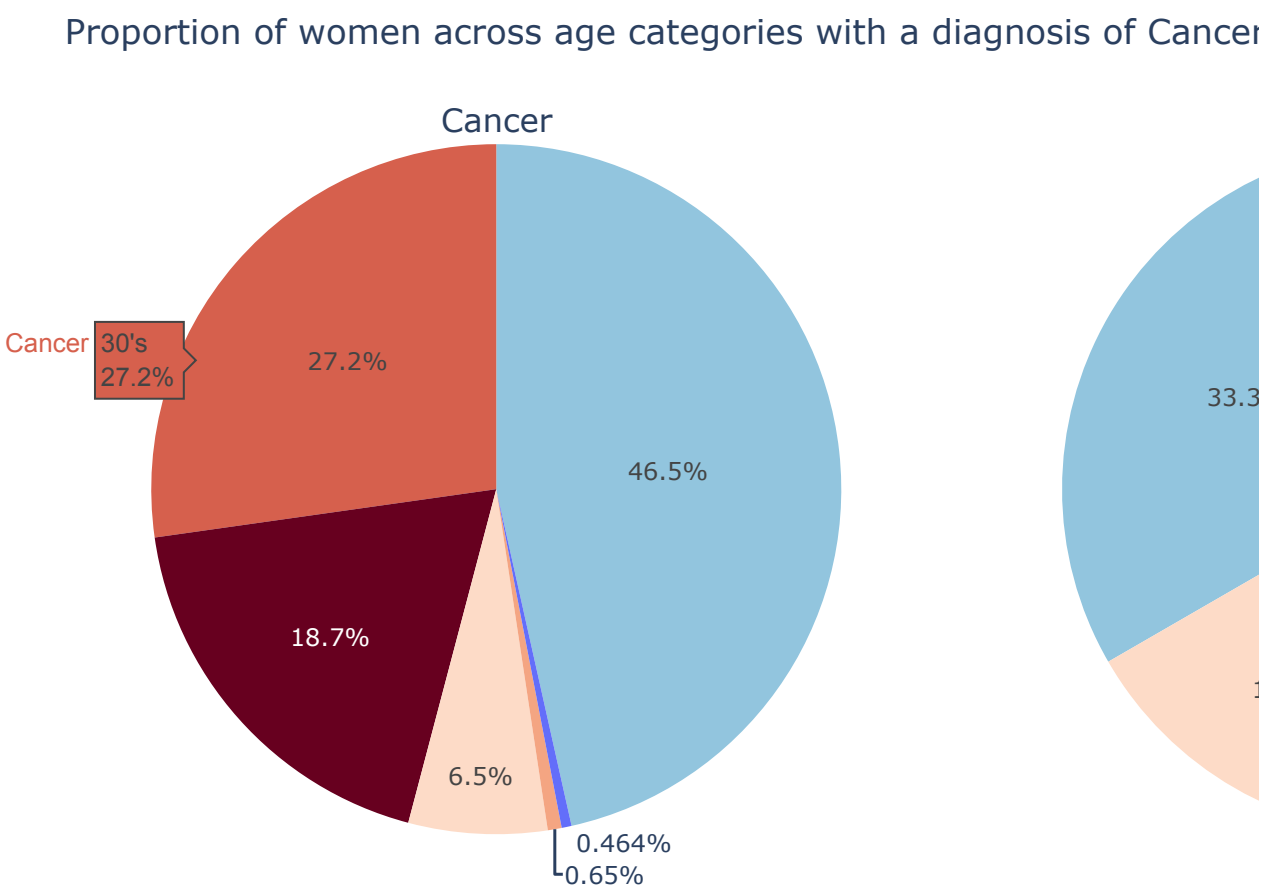


	Category	Sample Proportion
0	Child	0.000000
1	Teen	0.208625
2	20's	0.459207
3	30's	0.256410
4	40's	0.065268
5	50's	0.005828

6	60's	0.000000
7	70+	0.004662

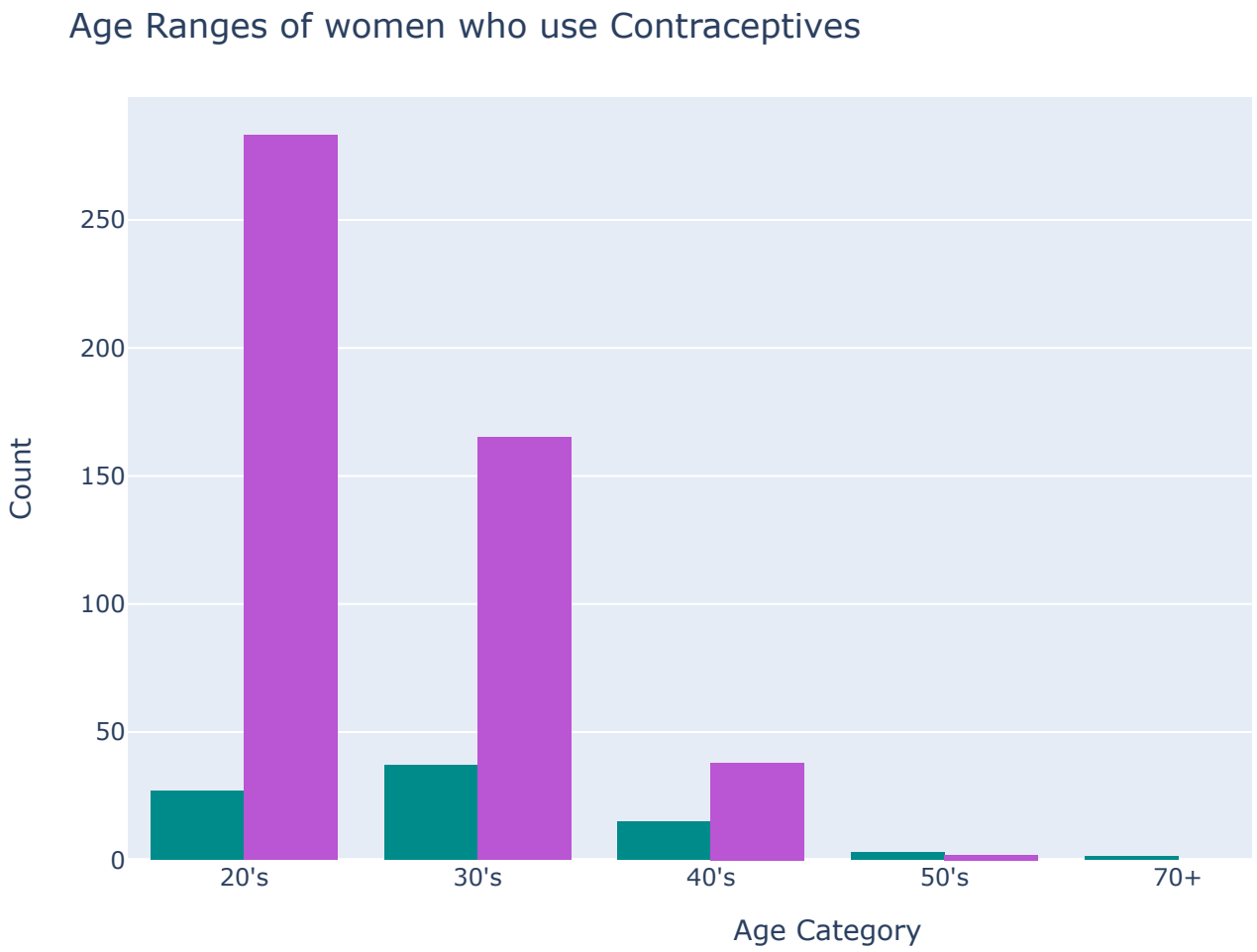
Next steps: View recommended plots

```
fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'domain'}, {'type': 'domain'
fig.add_trace(go.Pie(labels=risk_factor_df["age_cat"], values=risk_factor_df[labe
fig.add_trace(go.Pie(labels=risk_factor_df["age_cat"], values=risk_factor_df["Dx:

fig.update_traces(hole=.0, hoverinfo="label+percent+name")
fig.update_layout(title_text="Proportion of women across age categories with a di
fig.show()
```

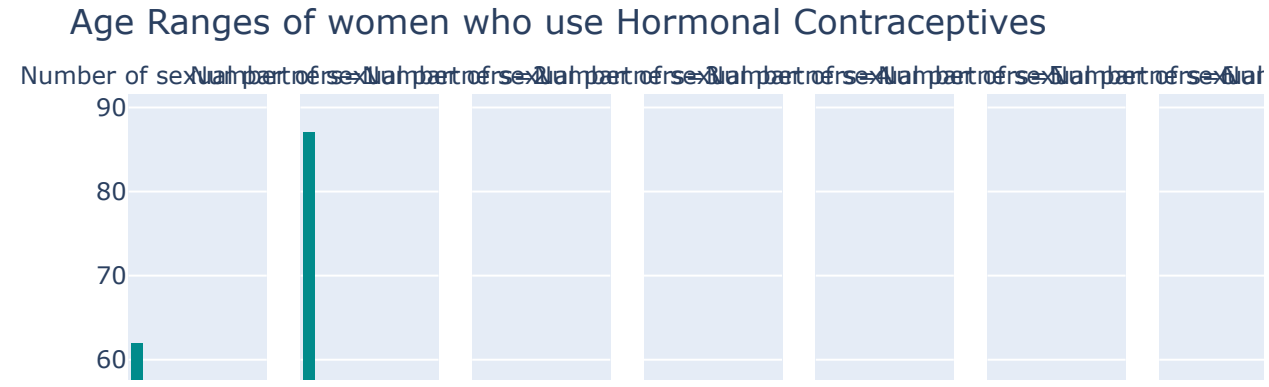


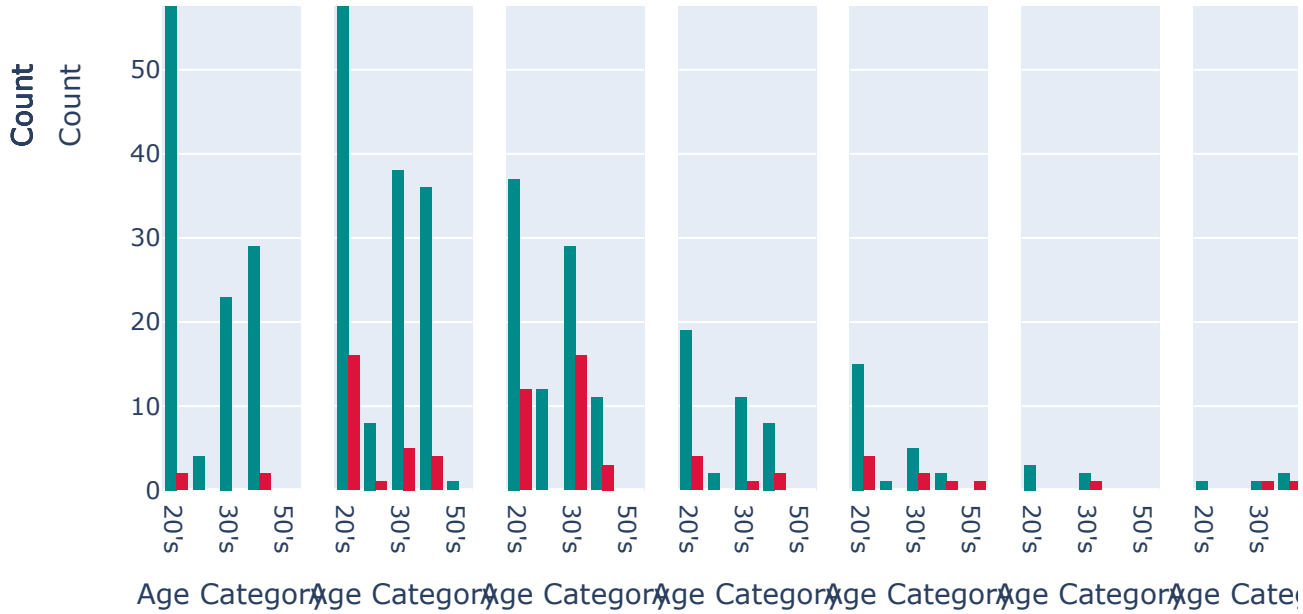

```
df_hormonal_compariosn = risk_factor_df.groupby(["age_cat"], as_index=False)[["IU
fig = px.histogram(df_hormonal_compariosn, x="age_cat", y=["IUD", "Hormonal Contr

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

```
fig.show()
```



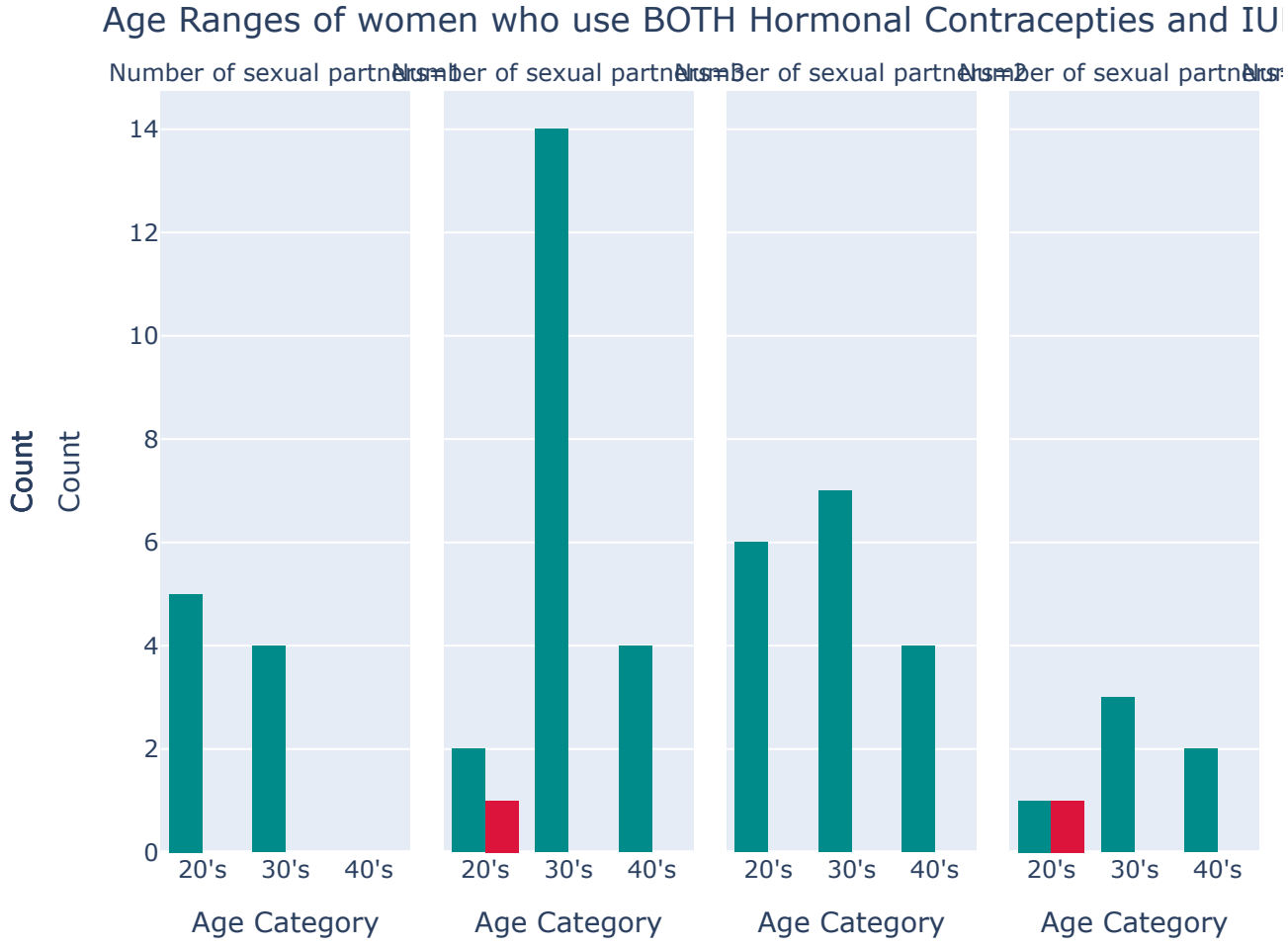

```
df_hormonal_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contracepti
df_hormonal_contraceptives = df_hormonal_contraceptives.sort_values(by=["Smokes",

fig = px.histogram(df_hormonal_contraceptives, x="age_cat", color="Smokes", barmo
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Count")
fig.update_layout(title="Age Ranges of women who use Hormonal Contraceptives")
fig.show()
```






```
df_IUD_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contraceptives"]
df_IUD_contraceptives = df_IUD_contraceptives.sort_values(by=["Smokes", label], a
fig = px.histogram(df_IUD_contraceptives, x="age_cat", color="Smokes", barmode="g

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()
```


```
df_both_contraceptives = risk_factor_df[(risk_factor_df["Hormonal Contraceptives"]
df_both_contraceptives = df_both_contraceptives.sort_values(by="Smokes")
fig = px.histogram(df_both_contraceptives, x="age_cat", color="Smokes", barmode="g
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Count")
fig.update_layout(title="Age Ranges of women who use BOTH Hormonal Contracepties a
fig.show()
```

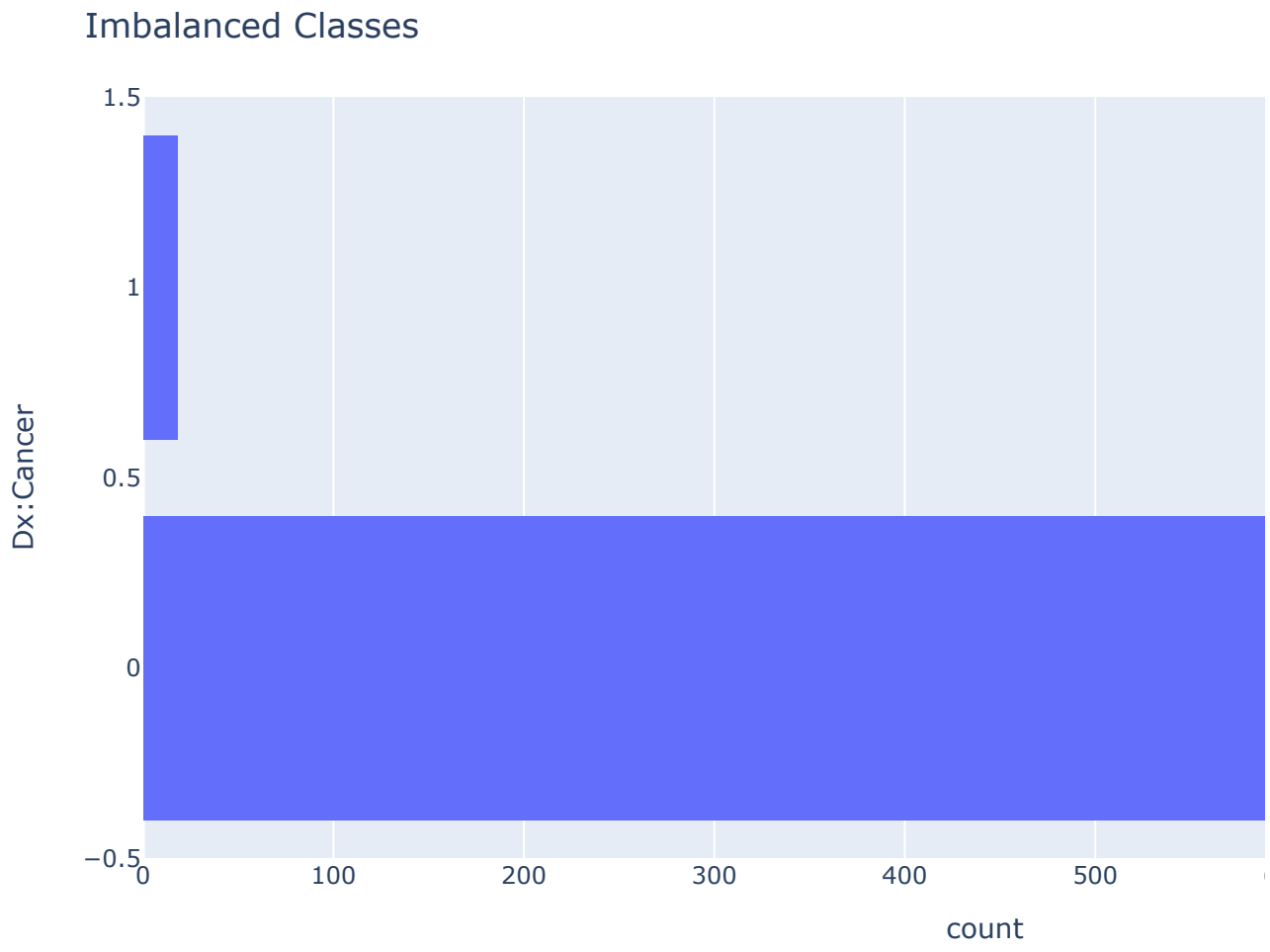


Test / Train Split

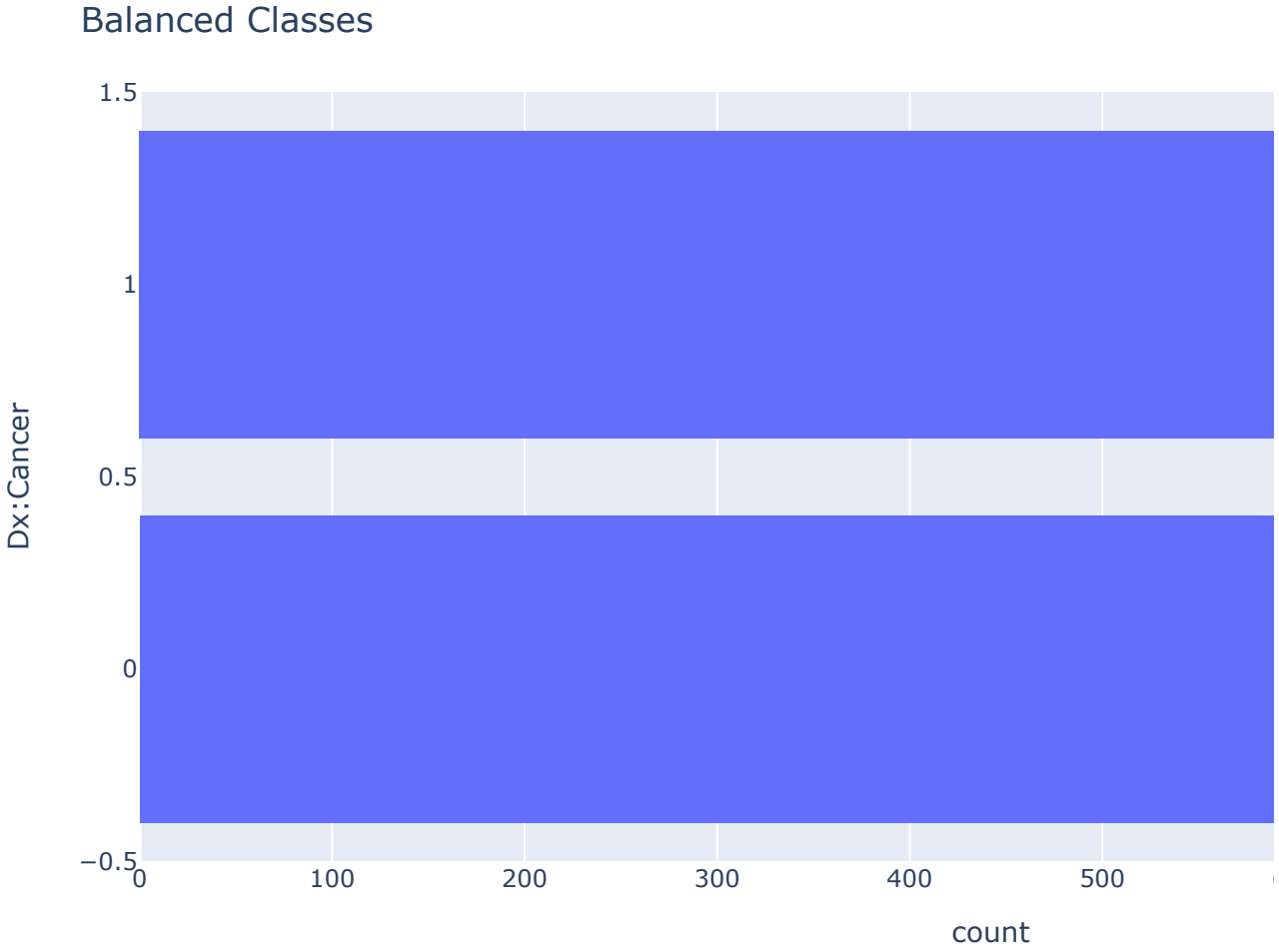
```
test = risk_factor_df[['Number of sexual partners', 'First sexual intercourse', '
test.round(2)
```

age_cat	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Dx	Hormonal Contraceptives	tota
20's	2.54	17.01	2.10	0.14	0.03	0.72	
30's	2.67	18.09	2.83	0.17	0.04	0.75	
40's	2.50	18.59	3.29	0.07	0.05	0.68	
50's	2.80	16.40	4.80	0.40	0.20	0.40	
70+	2.50	19.75	7.25	0.50	0.00	0.00	

```
label="Dx:Cancer"
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()

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



```
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)

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




```
train_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)

len(X_test.columns)
```

```
    35
```

```
# ## saving the data into csv for reuse
# # Without random var
# X_test.to_csv('/content/drive/My Drive/X_test.csv')
# y_test.to_csv('/content/drive/My Drive/y_test.csv')
# X_train.to_csv('/content/drive/My Drive/X_train.csv')
# y_train.to_csv('/content/drive/My Drive/y_train.csv')

# # With random var
# # Binary
# X_test.to_csv('/content/drive/My Drive/RX_test2.csv')
# y_test.to_csv('/content/drive/My Drive/Ry_test2.csv')
# X_train.to_csv('/content/drive/My Drive/RX_train2.csv')
# y_train.to_csv('/content/drive/My Drive/Ry_train2.csv')

# # Continuous
# X_test.to_csv('/content/drive/My Drive/RX_test.csv')
# y_test.to_csv('/content/drive/My Drive/Ry_test.csv')
# X_train.to_csv('/content/drive/My Drive/RX_train.csv')
# y_train.to_csv('/content/drive/My Drive/Ry_train.csv')

# adding a noise to the data
X_test['VARB']=bernoulli.rvs(.5, size=X_test.shape[0])
X_train['VARB']=bernoulli.rvs(.5, size=X_train.shape[0])
X_test['VARC']=bernoulli.rvs(.5, size=X_test.shape[0])
X_train['VARC']=bernoulli.rvs(.5, size=X_train.shape[0])

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

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

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

# Add a random variable
# Binary
# risk_factor_df['VAR_b']=bernoulli.rvs(.5, size=risk_factor_df.shape[0])

# Continuous
# risk_factor_df['VAR_c']=np.random.normal(loc=0, scale=1, size=risk_factor_df.sh
```

Model

Citation to the original paper

Ayad, C. W., Bonnier, T., Bosch, B., Read, J., & Parbhoo, S. (2023). Which Explanation Makes Sense? A Critical Evaluation of Local Explanations for Assessing Cervical Cancer Risk Factors.

Link to the original paper's repo (if applicable)

https://github.com/cwayad/Local-Explanations-for-Cervical-Cancer/blob/main/MLHC_cervical_cancer_classification.ipynb

Model descriptions

Ayad et al used 5 different model architectures that have widely been used in prior literature for cervical cancer risk assessment - namely Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), k-Nearest Neighbors (KNN), and Multilayer Perceptron (MLP).

These models often show high variance and lack interpretability, so Ayad et al generated local explanations for each of them, and compared the explanations generated with the set of bench evaluation metrics, and summarized the approach for assessing the quality of different explanations with their algorithm for assessing local feature contribution.

The model includes the model definitation which usually is a class, model training, and other necessary parts.

- Model architecture: layer number/size/type, activation function, etc
- Training objectives: loss function, optimizer, weight of each loss term, etc
- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc

validation, etc.

- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

Double-click (or enter) to edit

Logistic regression

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

Random forrest

```
rnd_clf = RandomForestClassifier()
```

KNN

```
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,
knn_clf_cv = KNeighborsClassifier(**knn_clf_cv.best_params_)
```

SVC

```
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)
```

MLP

```
nn_clf = MLPClassifier()
```

Training

This section we fit the five models mentioned in the Model section above. From the model architecture perspective, all the models that the paper uses are all regular supervised models

architecture perspective, all the models that the paper uses are all regular supervised models and can reasonably train and eval within hours on a laptop computation power, so also feasible on that front. So the dataset is workable for our applications without needing to downsampling.

Hyperparams

The paper trained the following models: Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), k-Nearest Neighbors (KNN), and Multilayer Perceptron (MLP). Some are trained with grid search for hyper parameter tuning, and the final models are trained with the following:

LR Hyperparameters: { C: 19306.977288832535 intercept_scaling: 1 max_iter: 100 penalty: l2 solver: lbfgs tol: 0.0001}

RF Hyperparameters: { ccp_alpha: 0.0 criterion: gini max_features: sqrt min_impurity_decrease: 0.0 min_samples_leaf: 1 min_samples_split: 2 min_weight_fraction_leaf: 0.0 n_estimators: 100 }

KNN Hyperparameters: { algorithm: auto leaf_size: 30 metric: minkowski metric_params: None n_jobs: None n_neighbors: 1 p: 2 weights: uniform }

SVC Hyperparameters: { C: 100.0 coef0: 0.0 decision_function_shape: ovr degree: 3 gamma: 0.001 kernel: rbf max_iter: -1 shrinking: True tol: 0.001 }

MLP Hyperparameters: { activation: relu alpha: 0.0001 batch_size: auto beta_1: 0.9 beta_2: 0.999 epsilon: 1e-08 hidden_layer_sizes: (100,) learning_rate: constant learning_rate_init: 0.001 max_fun: 15000 max_iter: 200 momentum: 0.9 n_iter_no_change: 10 nesterovs_momentum: True power_t: 0.5 shuffle: True solver: adam tol: 0.0001 validation_fraction: 0.1. }

Computational requirements

Given the dataset size and the model selection in the original paper, the model training does not have any hardware computation requirement. A Cpu runtime of the notebook on either a regular laptop or cloud setting such as Google collab can train the models in minutes. The same applies to the ablation studies we did below.

The model explanation generation is the more compute heavy steps, they are runnable on the same CPU settings, but some of the explaners would take hours and you can skip running the steps and load the pre-generated results.

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

est_name = []
est_acc = []
precision_score = []
```

```
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"],m
    roc.append(roc_auc_score(y_test, y_pred, average=None))
    print('roc',roc)
    est_name.append(estimators[i][0])
    est_acc.append(accuracy_score(y_test, y_pred))
    scores = precision_recall_fscore_support(y_test, y_pred, average="weighted")
    print('scores de '+str(clf_name), scores)
    precision_score.append(scores[0])
    recall_score.append(scores[1])
    f1score.append(scores[2])
    est_conf_matrix.append(confusion_matrix(y_test,y_pred))
```

```
roc [0.9970760233918129]
scores de LogisticRegression (0.997093023255814, 0.9941860465116279, 0.995146
roc [0.9970760233918129, 1.0]
scores de RandomForestClassifier  (1.0, 1.0, 1.0, None)
roc [0.9970760233918129, 1.0, 0.5]
scores de KNeighborsClassifier (0.9884058950784208, 0.9941860465116279, 0.991
roc [0.9970760233918129, 1.0, 0.5, 1.0]
scores de SupportVectorClassifier (1.0, 1.0, 1.0, None)
roc [0.9970760233918129, 1.0, 0.5, 1.0, 1.0]
scores de MLPClassifier (1.0, 1.0, 1.0, None)
```

```
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
```

Evaluation

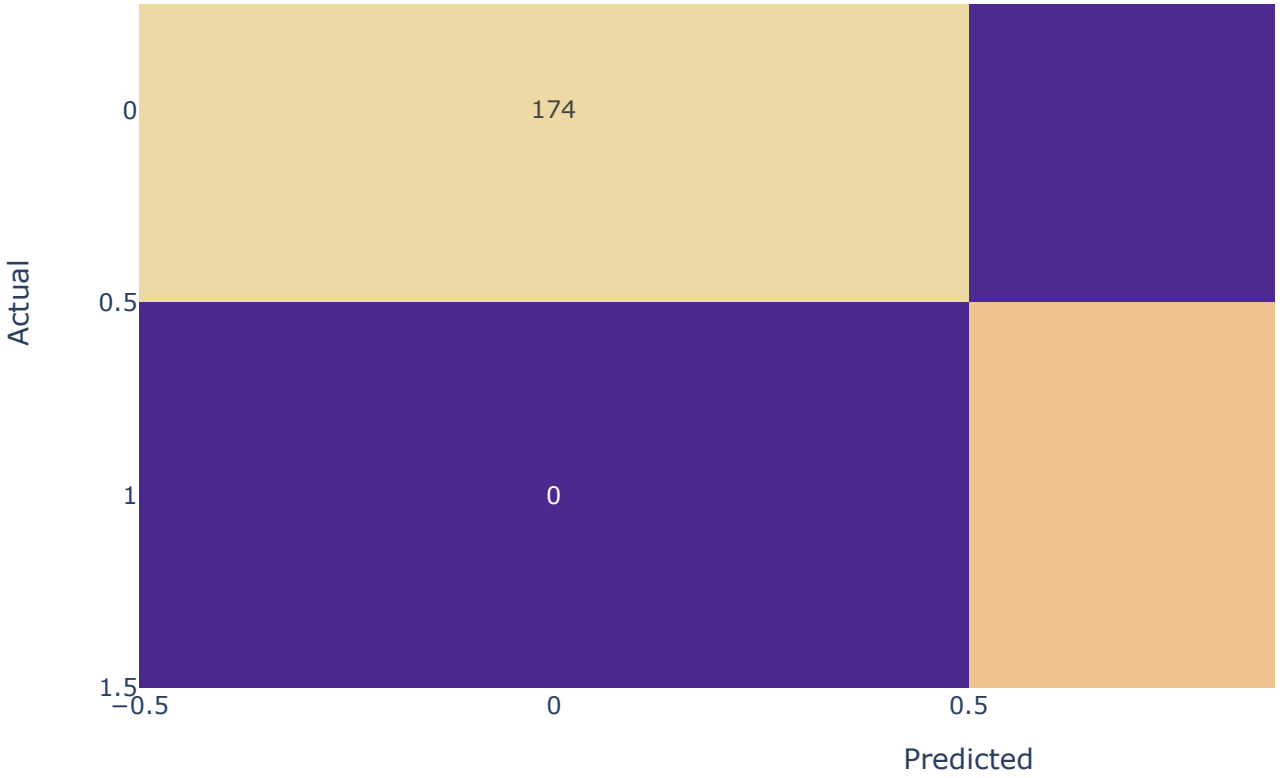
For evaluation, we will go over the base suites of metrics, for each model:

1. confusion matrix
    - the counts of true positives, true negatives, false positives, and false negatives.
2. accuracy
    - how many total predictions did the model get right
3. precision
    - true positives rate by mode out of all positive predictions
4. recall
    - true positives that were correctly identified by the model out of all actual positives.
5. F1
    - harmonic mean of precision and recall
6. AUC
    - the area under the Receiver Operating Characteristic (ROC) curve, which measures the model's ability to discriminate between positive and negative classes across different threshold values.

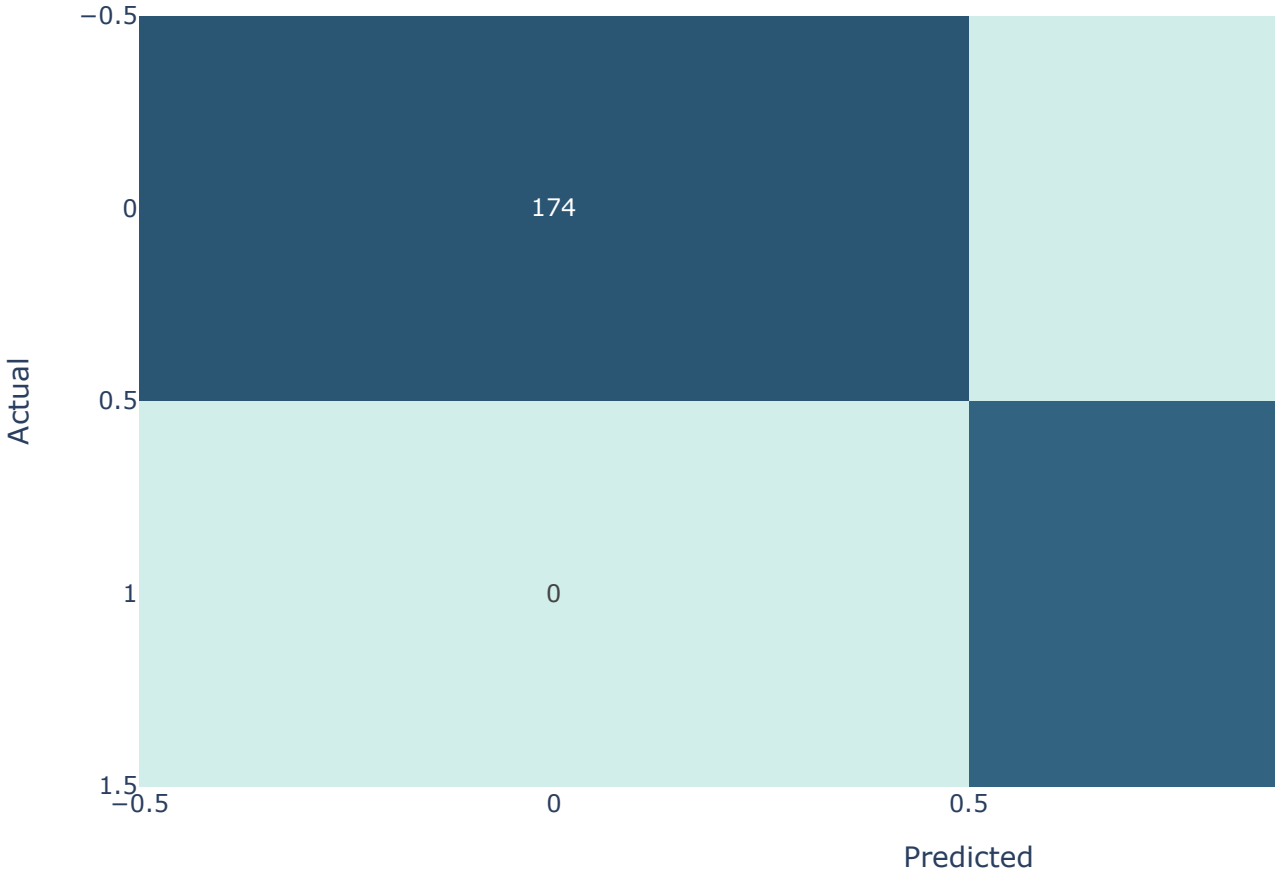
Later in the section we compare each model with others in visualization of the metrics aboves

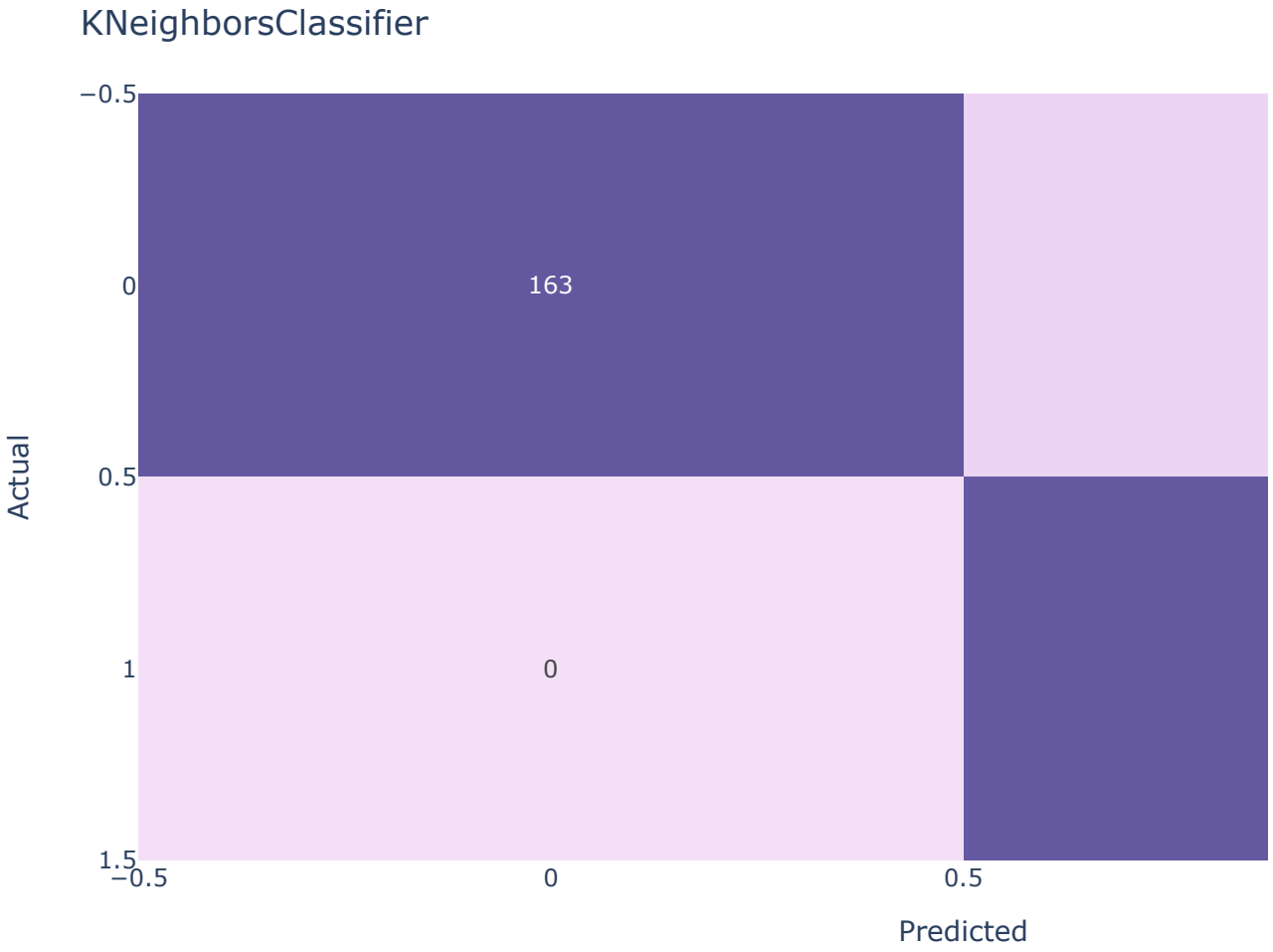
We also plan to use the standard precision, recall, and RemOve And Retrain (ROAR) (Hooker et al., 2018) mentioned in the paper for the faithfulness metric, where we will iteratively remove a subset of features from a dataset, and retrain the model on the reduced dataset to measure the changes in model accuracy or feature importance in each iteration, which will come later in the Results Analysis subsection.

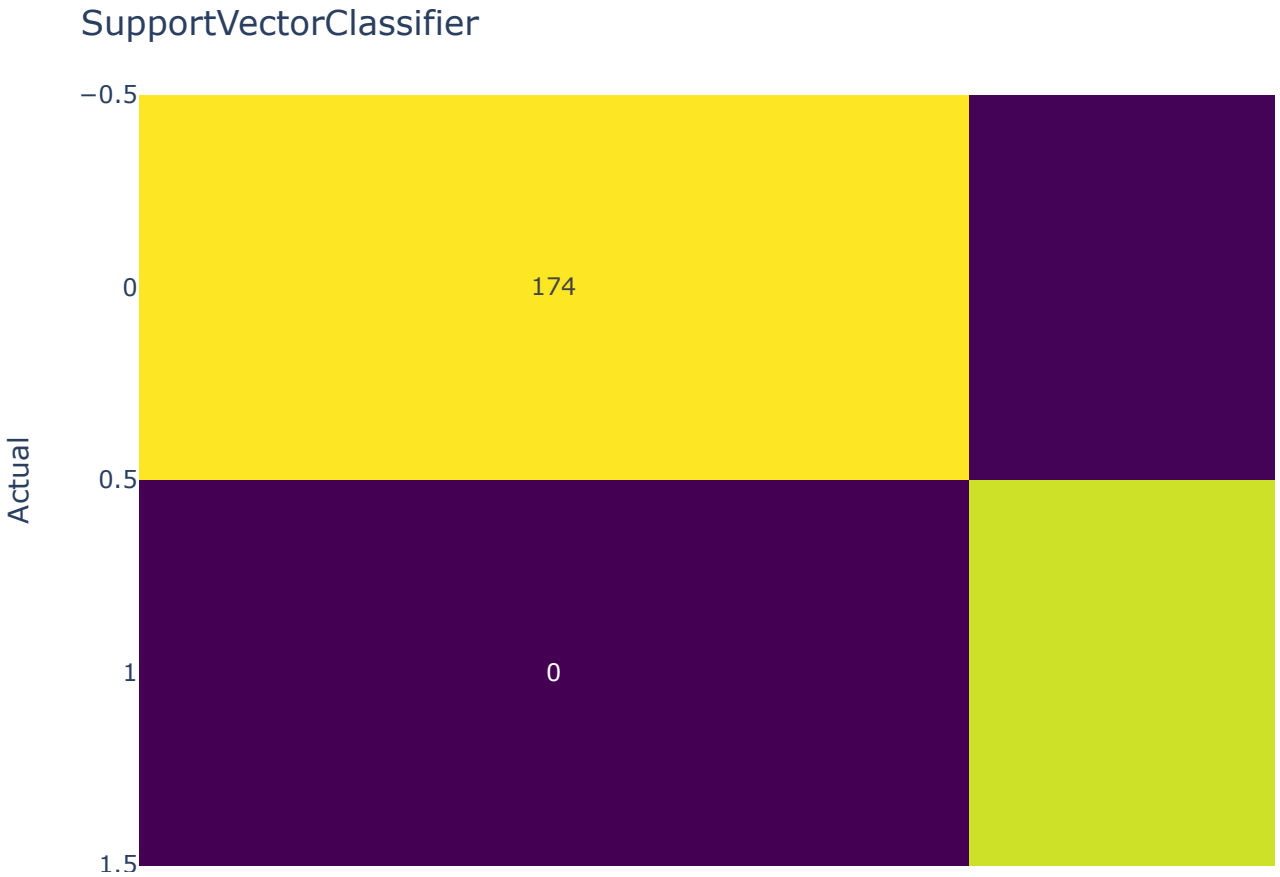
```
color_scales = ["agsunset","teal","purp","viridis","viridis"]
for i in range(0,len(est_conf_matrix)):
    heatmap = px.imshow(est_conf_matrix[i],aspect="auto",
                        text_auto=True,
                        color_continuous_scale=color_scales[i])
    heatmap.update_layout(title =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

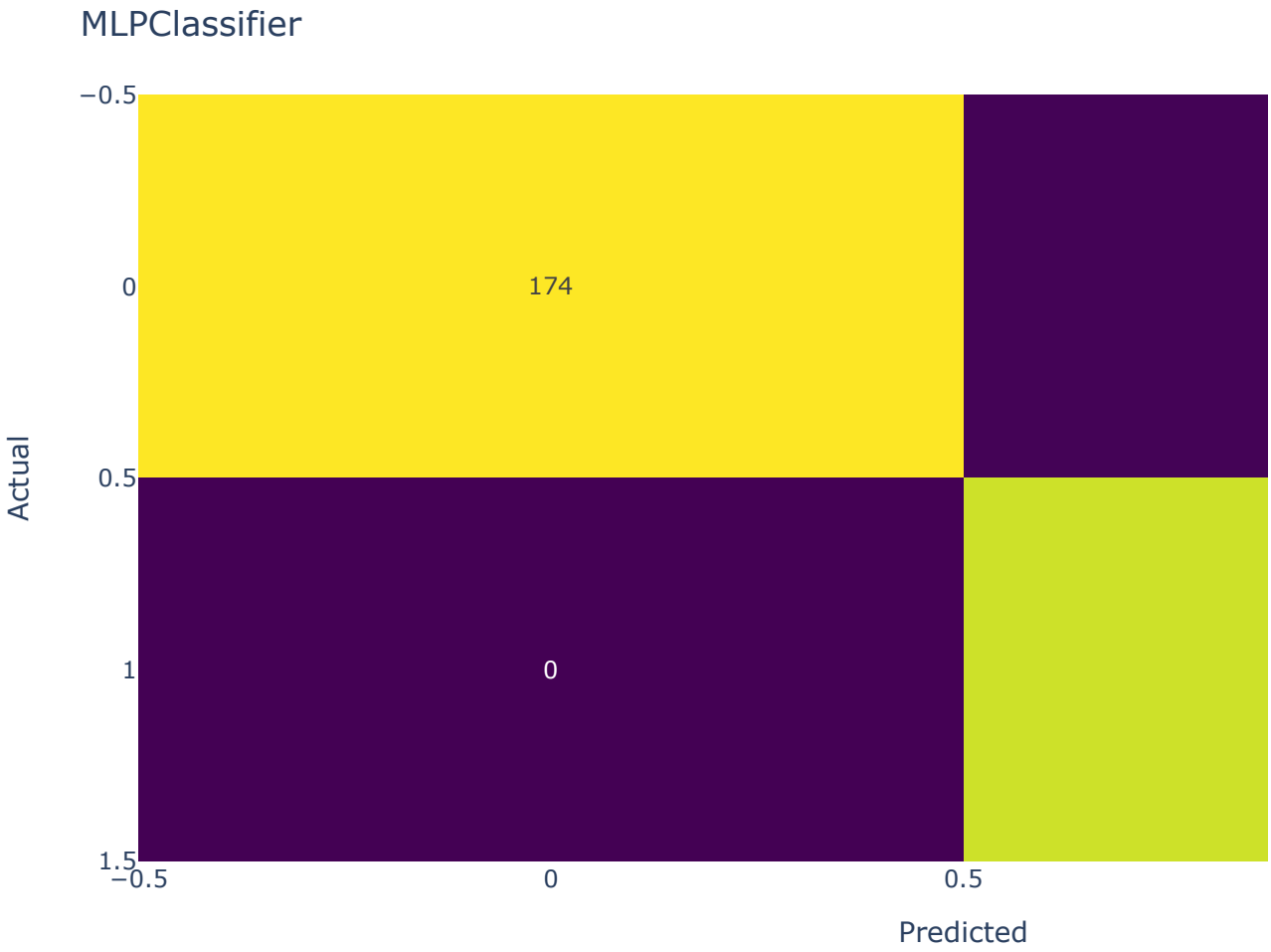


RandomForestClassifier




KNeighborsClassifier
−0.5
0
0.5
1
1.5
163
0
Actual
−0.5
0
0.5
Predicted


SupportVectorClassifier
−0.5
0
0.5
1
1.5
174
0
Actual



Results

Now that we finished training the model for Cervical Cancer risk factor prediction, we want to look into how locally explanables they each are to aid the healthcare process of diagnosis.

The models are evaluated to be pretty great, as noted right above, they score perfect in accuracy, and high in precision, recall, f1, and AUORC. Using MLP model as an example, the model achieved 0.997024 in accuracy, 0.997042 for precision, 0.997024 for recall, 0.997143 for AUC. Now how reliable are these models really in serving real-world cancer risk factor prediction?

In this section we will go over the Local Explainability results and analyses, including feature importances, explainability from each models, and ablation studies(this is the experiments we conducted beyond the original paper). We were able to reproduce each steps and components in the original paper, even though the precisions are not exactly the same we were able to test all

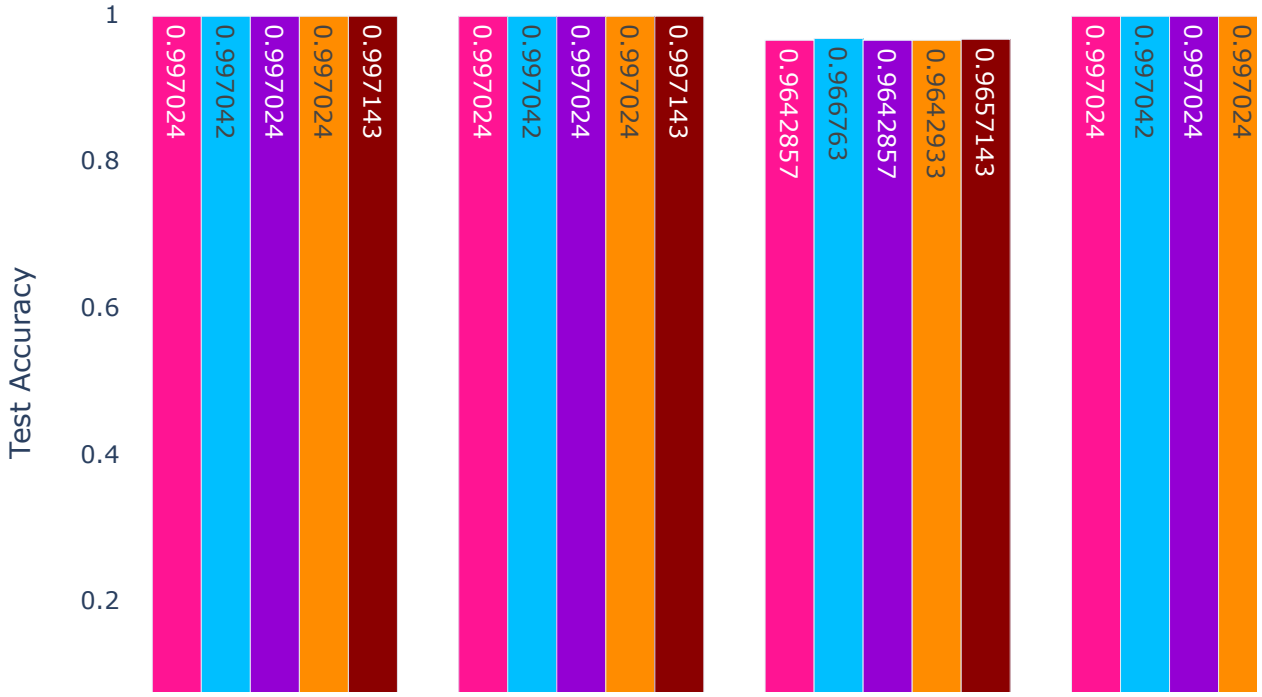
In the original paper, even though the precisions are not exactly the same we were able to test all the hypothesis and draw similar conclusions.

We focus our results work in the Analysis subsection here, where we show the original paper's claims with our experiment results on either one instance(patient record), and compare the hyposthesis testing with results from the original paper.

```
# results table
summary_df
```

	Classifier Name	Accuracy Score	Precision Score	Recall Score	F1 Score	AUROC
0	LogisticRegression	0.997024	0.997042	0.997024	0.997024	0.997143
1	RandomForestClassifier	0.997024	0.997042	0.997024	0.997024	0.997143
2	KNeighborsClassifier	0.964286	0.966763	0.964286	0.964293	0.965714
3	SupportVectorClassifier	0.997024	0.997042	0.997024	0.997024	0.997143
4	MLPClassifier	0.997024	0.997042	0.997024	0.997024	0.997143

```
acc_comparison = px.bar(summary_df, x="Classifier Name",
                        y=col_names[1:len(col_names)], labels={"value":"Test Accu
                        color_discrete_sequence=["deeppink", "deepskyblue", "dark
acc_comparison.update_layout({'plot_bgcolor': 'rgba(0, 0, 0, 0)', 'paper_bgcolor'
acc_comparison.show()
# acc_comparison.write_image('/content/drive/My Drive/modelsperf.png')
```



Explainability results

We follow the original paper's approach that they propose a multistage analysis pipeline where they first tested the performances of supervised learning models for predicting cervical cancer risk with binary label (0 or 1), then they used 5 different model architectures that have widely been used in prior literature for cervical cancer risk assessment - namely LR, RF, SVC, KNN, and MLP that we presented in Training and Evaluation sections above. These models often show high variance and lack interpretability, so Ayad et al generated local explanations for each of them, and compared the explanations generated with the set of bench evaluation metrics, and summarized the approach for assessing the quality of different explanations with their algorithm for assessing local feature contribution.

Feature importances

Here we compare the built-in feature importances from some models, before we run additional explainability libaries on them.

```
feature_names = X_test.columns

# Logistic Regression
logreg_coef = logreg_cv.coef_
logreg_feature_importance = np.abs(logreg_coef)
logreg_importance_sorted_idx = np.argsort(logreg_feature_importance)[0][::-1]

print("Logistic Regression Feature Importance:")
for idx in logreg_importance_sorted_idx:
    print(f"{feature_names[idx]}: {logreg_feature_importance[0][idx]}")
```

```
Logistic Regression Feature Importance:
Dx:HPV: 7.988561077579958
Dx: 7.001593472525832
Dx:CIN: 3.0026518048199398
Smokes: 2.547045607395325
STDs: Time since first diagnosis: 1.1325440927867418
Hormonal Contraceptives: 0.8486618828846253
STDs: Time since last diagnosis: 0.6486890454816002
Biopsy: 0.599871671959801
STDs:HIV: 0.5679250557576228
```

```
Num of pregnancies: 0.5676634681989842
STDs: Number of diagnosis: 0.5022981472709144
STDs: 0.49298176475962474
STDs:HPV: 0.4493780569188327
IUD: 0.44593443794174153
STDs:vulvo-perineal condylomatosis: 0.39122392360968333
IUD (years): 0.38154947734965966
STDs:Hepatitis B: 0.3666980963649427
STDs:cervical condylomatosis: 0.3412007120338539
VARC: 0.3164431156554607
VARB: 0.3027267867494243
STDs:pelvic inflammatory disease: 0.28268500048679923
STDs:condylomatosis: 0.2746989901746145
STDs (number): 0.20576098245320124
Citology: 0.18707848990416656
STDs:genital herpes: 0.14940638530138492
Hormonal Contraceptives (years): 0.1466204588616766
STDs:vaginal condylomatosis: 0.1460454320449084
Smokes (years): 0.10684698458895871
VAR: 0.10544118721645762
Hinselmann: 0.08116369389705501
Smokes (packs/year): 0.07924955424544078
Number of sexual partners: 0.0685012587798153
Age: 0.05320056667590457
STDs:molluscum contagiosum: 0.048529120550676666
STDs:AIDS: 0.03953678779951638
First sexual intercourse: 0.03511739741111344
STDs:syphilis: 0.03233248151498387
Schiller: 0.031610295673672656
```

```
# Random Forest
rnd_feature_importances = rnd_clf.feature_importances_
rnd_importance_sorted_idx = np.argsort(rnd_feature_importances)[::-1]

print("\nRandom Forest Feature Importance:")
for idx in rnd_importance_sorted_idx:
    print(f"{feature_names[idx]}: {rnd_feature_importances[idx]}")
```

```
Random Forest Feature Importance:
Dx:HPV: 0.34448057242464436
Dx: 0.25090646259995
IUD (years): 0.09308277134298404
Age: 0.07036118331593744
First sexual intercourse: 0.03989668419503147
Hormonal Contraceptives (years): 0.03365981017081919
IUD: 0.020995856071828467
Smokes: 0.013586396139185332
Hormonal Contraceptives: 0.012952843603373052
Num of pregnancies: 0.01271489065193197
Number of sexual partners: 0.009854216086801839
Smokes (packs/year): 0.009383178150438579
Smokes (years): 0.008833984468038132
STDs (number): 0.008252508014306938
```

```
STDs (number): 0.008252508014506938
STDs: Number of diagnosis: 0.00684193254445341
STDs: 0.00555201322775439
Citology: 0.005008409287009771
Dx:CIN: 0.0047931149369740645
Biopsy: 0.0046857867738465475
Schiller: 0.003766061330912571
STDs: Time since first diagnosis: 0.0037188527815323114
STDs: Time since last diagnosis: 0.0028645695546851368
STDs:condylomatosis: 0.0027660711488152368
VARB: 0.0027523723369704653
STDs:HPV: 0.00272038882176143
Hinselmann: 0.002558160043022986
STDs:vaginal condylomatosis: 0.002473383404008253
STDs:vulvo-perineal condylomatosis: 0.002303185979782125
STDs:pelvic inflammatory disease: 0.0022879996911746073
STDs:cervical condylomatosis: 0.0022823957260922705
STDs:HIV: 0.0021820854032376697
STDs:AIDS: 0.0021767027757759036
STDs:molluscum contagiosum: 0.002153723577278344
STDs:Hepatitis B: 0.001860442548385488
STDs:genital herpes: 0.0017847243336051432
VARC: 0.0017650453703947013
STDs:syphilis: 0.0015601660707233707
VAR: 0.00018105509653302336
```

SVC, K-Nearest Neighbors, and Multi-layer Perceptron don't have inherent feature importance attributes. We will skip checking the feature importance out of the trained model, and continue to check with explanability models in the next section.

```
# exclude_columns = ['VARB', 'VARC', 'VAR']
# features = X_test.columns[~X_test.columns.isin(exclude_columns)]
```

LR

```
features=X_test.columns
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.Data
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 = logreg_cv

print("-GLOSUR-")
# GloSur
explainer = MimicExplainer(model,
                           X_train[features],
                           LinearExplainableModel,
                           augment_data=False,
                           features=features,
                           model_task="classification")
```

```
global_explanation = explainer.explain_global(X_test[features])
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-
```

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

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

```
    WARNING:shap:Using 1341 background data samples could cause slower run times.
    -KSHAP-
    100%                                                  336/336 [1:06:11<00:00, 11.01s/it]
```

```
# print("-TSHAP-")  skip bc InvalidModelError: Model type not yet supported by Tr
# # TSHAP
# explainer = shap.TreeExplainer(model, X_train)
# shap_values = explainer.shap_values(X_test)

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

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


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

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

```
    -SSHAP-
```

```
    100%                                                  336/336 [18:27<00:00,  2.99s/it]
```

```
print("-LIME-")

# LIME
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classific

all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_featu
  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-
```

```
# 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)}


# print("-DICE-") UserConfigValidationException: continuous_features contains som
# import dice_ml
# df=risk_factor_df
# d = dice_ml.Data(dataframe=df, continuous_features=list(X_test.columns), outcom
# m = dice_ml.Model(model=model, backend="sklearn")

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

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

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

# logreg_cv
GloSur.to_csv("/content/drive/My Drive/glosur_logreg_cv.csv", index=False)
kernelSHAP.to_csv("/content/drive/My Drive/Kshap_logreg_cv.csv", index=False)
# treeSHAP.to_csv("/content/drive/My Drive/Tshap_logreg_cv.csv", index=False)
samplingSHAP.to_csv("/content/drive/My Drive/Sshap_logreg_cv.csv", index=False)
limecontrib.to_csv("/content/drive/My Drive/lime_logreg_cv.csv", index=False)
# ticontrib.to_csv("/content/drive/My Drive/ti_logreg_cv.csv", index=False)
# dicecontrib.to_csv("/content/drive/My Drive/dice_logreg_cv.csv", index=False)
```

RF

```
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.Data
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 = rnd_clf
features=X_test.columns

print("-GLOSUR-")
# GloSurhttps://colab.research.google.com/drive/1vxNVquzZYvVxl1VI3jCvxBWv1GxHxrhF
explainer = MimicExplainer(model,
                           X_train[features]
```

```
                            X_train[features],
                            LinearExplainableModel,
                            augment_data=False,
                            features=features,
                            model_task="classification")
global_explanation = explainer.explain_global(X_test[features])
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.to_csv("/content/drive/My Drive/glosur_rnd_clf.csv", index=False)
```

```
    -GLOSUR-
```

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

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

kernelSHAP.to_csv("/content/drive/My Drive/Kshap_rnd_clf.csv", index=False)
```

```
    WARNING:shap:Using 1341 background data samples could cause slower run times.
    -KSHAP-
    100%                                                  336/336 [3:03:38<00:00, 32.71s/it]
```

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

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

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

```
treeSHAP.to_csv("/content/drive/My Drive/Tshap_rnd_clf.csv", index=False)
```

```
    -TSHAP-
```

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

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

samplingSHAP.to_csv("/content/drive/My Drive/Sshap_rnd_clf.csv", index=False)
```

```
    -SSHAP-
    100%                                                  336/336 [21:59<00:00,  4.10s/it]
```

```
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)}

ticontrib.to_csv("/content/drive/My Drive/ti_rnd_clf.csv", index=False)
```

```
    -TI-
```

```
print("-LIME-")
# LIME
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classific
```

```
all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_featu
  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)}
limecontrib.to_csv("/content/drive/My Drive/lime_rnd_clf.csv", index=False)
```

-LIME-

```
# print("-DICE-") #continuous_features contains some feature names which are not
# df=risk_factor_df
# d = dice_ml.Data(dataframe=df, continuous_features=list(X_test.columns), outcom
# m = dice_ml.Model(model=model, backend="sklearn")

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

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

# res = dict()
# for j in list(dicecontrib.columns):
#   res[j]=np.mean(np.abs(dicecontrib[j]))
# fi_7={k: fi_7.get(k, 0) + res.get(k, 0) for k in set(fi_7)}
# dicecontrib.to_csv("/content/drive/My Drive/dice_rnd_clf.csv", index=False)
```

KNN

```
# KNN
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.DataF
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 = knn_clf_cv
features=X_test.columns


print("-GLOSUR-")
# GloSur
explainer = MimicExplainer(model,
                           X_train[features],
                           LinearExplainableModel,
                           augment_data=False,
                           features=features,
                           model_task="classification")
global_explanation = explainer.explain_global(X_test[features])
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.to_csv("/content/drive/My Drive/glosur_knn_clf.csv", index=False)
```

```
-GLOSUR-
```

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

res = dict()
for i in list(kernelSHAP.columns):
  res[i]=np.mean(np.abs(kernelSHAP[i]))
fi_2={k: fi_2.get(k, 0) + res.get(k, 0) for k in set(fi_2)}
kernelSHAP.to_csv("/content/drive/My Drive/Kshap_knn_clf.csv", index=False)
```

```
WARNING:shap:Using 1341 background data samples could cause slower run times.
-KSHAP-
```

100% 336/336 [3:43:16<00:00, 40.12s/it]

```
# print("-TSHAP-") #  Model type not yet supported by TreeExplainer: <class 'skle
# # TSHAP
# explainer = shap.TreeExplainer(model, X_train)
# shap_values = explainer.shap_values(X_test)

# temp=pd.DataFrame(shap_values[1], columns=features)
```

```
# treeSHAP=treeSHAP.add(temp, fill_value=0)

# res = dict()
# for i in list(treeSHAP.columns):
#   res[i]=np.mean(np.abs(treeSHAP[i]))
# fi_3={k: fi_3.get(k, 0) + res.get(k, 0) for k in set(fi_3)}
# treeSHAP.to_csv("/content/drive/My Drive/Tshap_knn_clf.csv", index=False)

print("-SSHAP-")
# SSHAP
explainer = shap.explainers.Sampling(model.predict_proba, X_train)
shap_values = explainer.shap_values(X_test)
temp=pd.DataFrame(shap_values[1], columns=features)
samplingSHAP=samplingSHAP.add(temp, fill_value=0)

res = dict()
for i in list(samplingSHAP.columns):
  res[i]=np.mean(np.abs(samplingSHAP[i]))
fi_4={k: fi_4.get(k, 0) + res.get(k, 0) for k in set(fi_4)}
samplingSHAP.to_csv("/content/drive/My Drive/Sshap_knn_clf.csv", index=False)
```

```
-SSHAP-
100%                                                  336/336 [23:28<00:00,  4.03s/it]
```

```
print("-LIME-")
# LIME
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classific

all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_featu
  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)}
```

```
limecontrib.to_csv("/content/drive/My Drive/lime_knn_clf.csv", index=False)
```

```
    -LIME-
```

```
# print("-TI-") # AttributeError: 'KNeighborsClassifier' object has no attribute
# # 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)}
# ticontrib.to_csv("/content/drive/My Drive/ti_knn_clf.csv", index=False)

print("-DICE-")
# DICE
df=risk_factor_df
d = dice_ml.Data(dataframe=df, continuous_features=list(X_test.columns), outcome_
m = dice_ml.Model(model=model, backend="sklearn")

exp = dice_ml.Dice(d, m, method="random")
query_instance = X_test
e1 = exp.generate_counterfactuals(query_instance, total_CFs=10, desired_range=Nor
                                  desired_class="opposite",
                                  permitted_range=None, features_to_vary="all")

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

res = dict()
for j in list(dicecontrib.columns):
  res[j]=np.mean(np.abs(dicecontrib[j]))
fi_7={k: fi_7.get(k, 0) + res.get(k, 0) for k in set(fi_7)}
dicecontrib.to_csv("/content/drive/My Drive/dice_knn_clf.csv", index=False)
```

SVC

```
# SVC
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.Data
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 = svm_clf_cv.best_estimator_
features=X_test.columns

# print("-GLOSUR-")
# # GloSur
# explainer = MimicExplainer(model,
#                            X_train[features],
#                            LinearExplainableModel,
#                            augment_data=False,
#                            features=features,
#                            model_task="classification")
# global_explanation = explainer.explain_global(X_test[features])
# temp=pd.DataFrame(global_explanation.local_importance_values[1], columns=featur
# 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.to_csv("/content/drive/My Drive/glosur_svc_clf.csv", index=False)


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

res = dict()
for i in list(kernelSHAP.columns):
  res[i]=np.mean(np.abs(kernelSHAP[i]))
fi_2={k: fi_2.get(k, 0) + res.get(k, 0) for k in set(fi_2)}
kernelSHAP.to_csv("/content/drive/My Drive/Kshap_svc_clf.csv", index=False)

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

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

res = dict()
for i in list(treeSHAP.columns):
```

```
  res[i]=np.mean(np.abs(treeSHAP[i]))
fi_3={k: fi_3.get(k, 0) + res.get(k, 0) for k in set(fi_3)}
treeSHAP.to_csv("/content/drive/My Drive/Tshap_svc_clf.csv", index=False)

print("-SSHAP-")
# SSHAP
explainer = shap.explainers.Sampling(model.predict_proba, X_train)
shap_values = explainer.shap_values(X_test)
temp=pd.DataFrame(shap_values[1], columns=features)
samplingSHAP=samplingSHAP.add(temp, fill_value=0)

res = dict()
for i in list(samplingSHAP.columns):
  res[i]=np.mean(np.abs(samplingSHAP[i]))
fi_4={k: fi_4.get(k, 0) + res.get(k, 0) for k in set(fi_4)}
samplingSHAP.to_csv("/content/drive/My Drive/Sshap_svc_clf.csv", index=False)


print("-LIME-")
# LIME
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classific

all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_featu
  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)}

limecontrib.to_csv("/content/drive/My Drive/lime_svc_clf.csv", index=False)


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)}
ticontrib.to_csv("/content/drive/My Drive/ti_svc_clf.csv", index=False)

print("-DICE-")
# DICE
df=risk_factor_df
d = dice_ml.Data(dataframe=df, continuous_features=list(X_test.columns), outcome_
m = dice_ml.Model(model=model, backend="sklearn")

exp = dice_ml.Dice(d, m, method="random")
query_instance = X_test
e1 = exp.generate_counterfactuals(query_instance, total_CFs=10, desired_range=Nor
                                  desired_class="opposite",
                                  permitted_range=None, features_to_vary="all")

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

res = dict()
for j in list(dicecontrib.columns):
  res[j]=np.mean(np.abs(dicecontrib[j]))
fi_7={k: fi_7.get(k, 0) + res.get(k, 0) for k in set(fi_7)}
dicecontrib.to_csv("/content/drive/My Drive/dice_svc_clf.csv", index=False)
```

MLP

```
# MLPClassifier
GloSur=kernelSHAP=treeSHAP=samplingSHAP=limecontrib=ticontrib=dicecontrib=pd.Data
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
features=X_test.columns

print("-GLOSUR-")
# GloSur
explainer = MimicExplainer(model,
```

```
explainer = XAIMLExplainer(model,
                           X_train[features],
                           LinearExplainableModel,
                           augment_data=False,
                           features=features,
                           model_task="classification")
global_explanation = explainer.explain_global(X_test[features])
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.to_csv("/content/drive/My Drive/glosur_nn_clf.csv", index=False)
```

```
-GLOSUR-
```

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

res = dict()
for i in list(kernelSHAP.columns):
  res[i]=np.mean(np.abs(kernelSHAP[i]))
fi_2={k: fi_2.get(k, 0) + res.get(k, 0) for k in set(fi_2)}
kernelSHAP.to_csv("/content/drive/My Drive/Kshap_nn_clf.csv", index=False)
```

```
WARNING:shap:Using 1341 background data samples could cause slower run times.
-KSHAP-
100%                                                  336/336 [54:15<00:00,  9.65s/it]
```

```
# print("-TSHAP-") # InvalidModelError: Model type not yet supported by TreeExpla
# # TSHAP
# explainer = shap.TreeExplainer(model, X_train)
# shap_values = explainer.shap_values(X_test)

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

# res = dict()
# for i in list(treeSHAP.columns):
#   res[i]=np.mean(np.abs(treeSHAP[i]))
# fi_3={k: fi_3.get(k, 0) + res.get(k, 0) for k in set(fi_3)}
# treeSHAP.to_csv("/content/drive/My Drive/Tshap_nn_clf.csv", index=False)
```

```
# treeSHAP.to_csv("/content/drive/My Drive/Tshap_nn_clf.csv", index=False)

print("-SSHAP-")
# SSHAP
explainer = shap.explainers.Sampling(model.predict_proba, X_train)
shap_values = explainer.shap_values(X_test)
temp=pd.DataFrame(shap_values[1], columns=features)
samplingSHAP=samplingSHAP.add(temp, fill_value=0)

res = dict()
for i in list(samplingSHAP.columns):
  res[i]=np.mean(np.abs(samplingSHAP[i]))
fi_4={k: fi_4.get(k, 0) + res.get(k, 0) for k in set(fi_4)}
samplingSHAP.to_csv("/content/drive/My Drive/Sshap_nn_clf.csv", index=False)
```

```
-SSHAP-
100%                                                  336/336 [18:48<00:00,  3.15s/it]
```

```
print("-LIME-")
# LIME
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classific

all=[]
for i in range (len(X_test)):
  exp = explainer.explain_instance(X_test.iloc[i], model.predict_proba, num_featu
  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)}

limecontrib.to_csv("/content/drive/My Drive/lime_nn_clf.csv", index=False)
```

```
-LIME-
```

```
# print("-TI-") # ValueError: Wrong model type. Base learner needs to be a Decisi
# # 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)}
# ticontrib.to_csv("/content/drive/My Drive/ti_nn_clf.csv", index=False)

# print("-DICE-"). #UserConfigValidationException: continuous_features contains s
# # DICE
# df=risk_factor_df
# d = dice_ml.Data(dataframe=df, continuous_features=list(X_test.columns), outcom
# m = dice_ml.Model(model=model, backend="sklearn")

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

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

# res = dict()
# for j in list(dicecontrib.columns):
#   res[j]=np.mean(np.abs(dicecontrib[j]))
# fi_7={k: fi_7.get(k, 0) + res.get(k, 0) for k in set(fi_7)}
# dicecontrib.to_csv("/content/drive/My Drive/dice_nn_clf.csv", index=False)
```

Ablation Study

The paper identified a set of most important features for assessing cervical cancer risk, we plan to rotate and systematically remove the top 30% most important features one by one and see if the model performance will have degradation, with the hope that it will help us understand the impact of specific features in determining cervical cancer, and analyze the five model's robustness or sensitivity to changes in the experiment.

```
estimators, features
```

```
[('LogisticRegression', LogisticRegression(C=31.622776601683793)),
 ('RandomForestClassifier ', RandomForestClassifier()),
 ('KNeighborsClassifier', KNeighborsClassifier(n_neighbors=1)),
 ('SupportVectorClassifier',
  GridSearchCV(cv=5, estimator=SVC(),
               param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00,
1.e+01, 1.e+02]),
                           'gamma': array([1.e-03, 1.e-02, 1.e-01, 1.e+00,
1.e+01, 1.e+02])})),
 ('MLPClassifier', MLPClassifier())]
```

```
# LogisticRegression
abla_model = LogisticRegression(C=19306.977288832535)

abla_model.fit(X_train, y_train)
y_pred = abla_model.predict(X_test)

# Evaluate baseline performance
baseline_accuracy = accuracy_score(y_test, y_pred)
print("Baseline Accuracy:", baseline_accuracy)

# Run ablation experiment
feature_performance = {}
for feature in X_test.columns:
    # Remove one feature
    print('remove ', feature)
    X_ablated = X_train.drop(columns=[feature])

    abla_model.fit(X_ablated, y_train)

    # Evaluate the model's performance after removing the feature
    ablated_accuracy = accuracy_score(y_test, abla_model.predict(X_test.drop(colu

    # Record the performance
    feature_performance[feature] = baseline_accuracy - ablated_accuracy

# Sort features by their impact
sorted_features = sorted(feature_performance.items(), key=lambda x: x[1], reverse

# Print feature importance
print("Feature Ablation Results:")
for feature, impact in sorted_features:
    print(f"{feature}: {impact}")
```

```
Baseline Accuracy: 0.9970238095238095
remove  Age
remove  Number of sexual partners
remove  First sexual intercourse
remove  Num of pregnancies
remove  Smokes
```

```
remove  Smokes (years)
remove  Smokes (packs/year)
remove  Hormonal Contraceptives
remove  Hormonal Contraceptives (years)
remove  IUD
remove  IUD (years)
remove  STDs
remove  STDs (number)
remove  STDs:condylomatosis
remove  STDs:cervical condylomatosis
remove  STDs:vaginal condylomatosis
remove  STDs:vulvo-perineal condylomatosis
remove  STDs:syphilis
remove  STDs:pelvic inflammatory disease
remove  STDs:genital herpes
remove  STDs:molluscum contagiosum
remove  STDs:AIDS
remove  STDs:HIV
remove  STDs:Hepatitis B
remove  STDs:HPV
remove  STDs: Number of diagnosis
remove  STDs: Time since first diagnosis
remove  STDs: Time since last diagnosis
remove  Dx:CIN
remove  Dx:HPV
remove  Dx
remove  Hinselmann
remove  Schiller
remove  Citology
remove  Biopsy
remove  VARB
remove  VARC
remove  VAR
Feature Ablation Results:
Dx:HPV: 0.0714285714285714
Dx: 0.0357142857142857
Dx:CIN: 0.008928571428571397
Age: 0.0029761904761904656
Hormonal Contraceptives: 0.0029761904761904656
STDs:condylomatosis: 0.0029761904761904656
STDs:pelvic inflammatory disease: 0.0029761904761904656
Number of sexual partners: 0.0
First sexual intercourse: 0.0
Smokes: 0.0
Smokes (years): 0.0
Smokes (packs/year): 0.0
Hormonal Contraceptives (years): 0.0
IUD: 0.0
IUD (years): 0.0
STDs: 0.0
STDs (number): 0.0
STDs:cervical condylomatosis: 0.0
```

```
# RandomForestClassifier
abla model = RandomForestClassifier()
```

```
abla_model.fit(X_train, y_train)
y_pred = abla_model.predict(X_test)

# Evaluate baseline performance
baseline_accuracy = accuracy_score(y_test, y_pred)
print("Baseline Accuracy:", baseline_accuracy)

# Run ablation experiment
feature_performance = {}
for feature in X_test.columns:
    # Remove one feature
    print('remove ', feature)
    X_ablated = X_train.drop(columns=[feature])

    abla_model.fit(X_ablated, y_train)

    # Evaluate the model's performance after removing the feature
    ablated_accuracy = accuracy_score(y_test, abla_model.predict(X_test.drop(colu

    # Record the performance
    feature_performance[feature] = baseline_accuracy - ablated_accuracy

# Sort features by their impact
sorted_features = sorted(feature_performance.items(), key=lambda x: x[1], reverse

# Print feature importance
print("Feature Ablation Results:")
for feature, impact in sorted_features:
    print(f"{feature}: {impact}")
```

```
Baseline Accuracy: 0.9970238095238095
remove  Age
remove  Number of sexual partners
remove  First sexual intercourse
remove  Num of pregnancies
remove  Smokes
remove  Smokes (years)
remove  Smokes (packs/year)
remove  Hormonal Contraceptives
remove  Hormonal Contraceptives (years)
remove  IUD
remove  IUD (years)
remove  STDs
remove  STDs (number)
remove  STDs:condylomatosis
remove  STDs:cervical condylomatosis
remove  STDs:vaginal condylomatosis
remove  STDs:vulvo-perineal condylomatosis
remove  STDs:syphilis
remove  STDs:pelvic inflammatory disease
remove  STDs:genital herpes
remove  STDs:molluscum contagiosum
```

```
remove  STDs:AIDS
remove  STDs:HIV
remove  STDs:Hepatitis B
remove  STDs:HPV
remove  STDs: Number of diagnosis
remove  STDs: Time since first diagnosis
remove  STDs: Time since last diagnosis
remove  Dx:CIN
remove  Dx:HPV
remove  Dx
remove  Hinselmann
remove  Schiller
remove  Citology
remove  Biopsy
remove  VARB
remove  VARC
remove  VAR
Feature Ablation Results:
Dx:HPV: 0.011904761904761862
Number of sexual partners: 0.0
First sexual intercourse: 0.0
Num of pregnancies: 0.0
Smokes (years): 0.0
Hormonal Contraceptives: 0.0
Hormonal Contraceptives (years): 0.0
IUD: 0.0
IUD (years): 0.0
STDs: 0.0
STDs:cervical condylomatosis: 0.0
STDs:vaginal condylomatosis: 0.0
STDs:vulvo-perineal condylomatosis: 0.0
STDs:syphilis: 0.0
STDs:pelvic inflammatory disease: 0.0
STDs:molluscum contagiosum: 0.0
STDs:HIV: 0.0
STDs:Hepatitis B: 0.0
```

```
# KNeighborsClassifier
abla_model = KNeighborsClassifier(n_neighbors=1)

abla_model.fit(X_train, y_train)
y_pred = abla_model.predict(X_test)

# Evaluate baseline performance
baseline_accuracy = accuracy_score(y_test, y_pred)
print("Baseline Accuracy:", baseline_accuracy)

# Run ablation experiment
feature_performance = {}
for feature in X_test.columns:
    # Remove one feature
    print('remove ', feature)
    X_ablated = X_train.drop(columns=[feature])
```

```
    abla_model.fit(X_ablated, y_train)

    # Evaluate the model's performance after removing the feature
    ablated_accuracy = accuracy_score(y_test, abla_model.predict(X_test.drop(colu

    # Record the performance
    feature_performance[feature] = baseline_accuracy - ablated_accuracy

# Sort features by their impact
sorted_features = sorted(feature_performance.items(), key=lambda x: x[1], reverse

# Print feature importance
print("Feature Ablation Results:")
for feature, impact in sorted_features:
    print(f"{feature}: {impact}")
```

```
Baseline Accuracy: 0.9583333333333334
remove  Age
remove  Number of sexual partners
remove  First sexual intercourse
remove  Num of pregnancies
remove  Smokes
remove  Smokes (years)
remove  Smokes (packs/year)
remove  Hormonal Contraceptives
remove  Hormonal Contraceptives (years)
remove  IUD
remove  IUD (years)
remove  STDs
remove  STDs (number)
remove  STDs:condylomatosis
remove  STDs:cervical condylomatosis
remove  STDs:vaginal condylomatosis
remove  STDs:vulvo-perineal condylomatosis
remove  STDs:syphilis
remove  STDs:pelvic inflammatory disease
remove  STDs:genital herpes
remove  STDs:molluscum contagiosum
remove  STDs:AIDS
remove  STDs:HIV
remove  STDs:Hepatitis B
remove  STDs:HPV
remove  STDs: Number of diagnosis
remove  STDs: Time since first diagnosis
remove  STDs: Time since last diagnosis
remove  Dx:CIN
remove  Dx:HPV
remove  Dx
remove  Hinselmann
remove  Schiller
remove  Citology
remove  Biopsy
remove  VARB
remove  VARC
```

```
remove  VARC
remove  VAR
Feature Ablation Results:
Dx:HPV: 0.023809523809523836
Dx: 0.011904761904761973
Smokes (years): 0.008928571428571508
First sexual intercourse: 0.005952380952381042
Num of pregnancies: 0.005952380952381042
Smokes (packs/year): 0.005952380952381042
IUD (years): 0.005952380952381042
VAR: 0.005952380952381042
Age: 0.0029761904761904656
STDs: Time since first diagnosis: 0.0029761904761904656
VARC: 0.0029761904761904656
Number of sexual partners: 0.0
Smokes: 0.0
Hormonal Contraceptives: 0.0
IUD: 0.0
STDs: 0.0
STDs (number): 0.0
STDs:condylomatosis: 0.0
```

```
# SVC
abla_model = SVC(C=100.0, gamma=0.001)

abla_model.fit(X_train, y_train)
y_pred = abla_model.predict(X_test)

# Evaluate baseline performance
baseline_accuracy = accuracy_score(y_test, y_pred)
print("Baseline Accuracy:", baseline_accuracy)

# Run ablation experiment
feature_performance = {}
for feature in X_test.columns:
    # Remove one feature
    # print('remove ', feature)
    X_ablated = X_train.drop(columns=[feature])

    abla_model.fit(X_ablated, y_train)

    # Evaluate the model's performance after removing the feature
    ablated_accuracy = accuracy_score(y_test, abla_model.predict(X_test.drop(colu

    # Record the performance
    feature_performance[feature] = baseline_accuracy - ablated_accuracy

# Sort features by their impact
sorted_features = sorted(feature_performance.items(), key=lambda x: x[1], reverse

# Print feature importance
print("Feature Ablation Results:")
for feature, impact in sorted_features:
```

```
for feature, impact in sorted_features:
    print(f"{feature}: {impact}")
```

```
Baseline Accuracy: 0.9940476190476191
Feature Ablation Results:
Dx:HPV: 0.0267857142857143
Dx: 0.008928571428571397
Number of sexual partners: 0.0
First sexual intercourse: 0.0
Smokes (packs/year): 0.0
Hormonal Contraceptives: 0.0
Hormonal Contraceptives (years): 0.0
IUD: 0.0
STDs: 0.0
STDs (number): 0.0
STDs:condylomatosis: 0.0
STDs:cervical condylomatosis: 0.0
STDs:vaginal condylomatosis: 0.0
STDs:vulvo-perineal condylomatosis: 0.0
STDs:syphilis: 0.0
STDs:pelvic inflammatory disease: 0.0
STDs:genital herpes: 0.0
STDs:molluscum contagiosum: 0.0
STDs:AIDS: 0.0
STDs:HIV: 0.0
STDs:Hepatitis B: 0.0
STDs:HPV: 0.0
STDs: Number of diagnosis: 0.0
STDs: Time since first diagnosis: 0.0
STDs: Time since last diagnosis: 0.0
Dx:CIN: 0.0
Hinselmann: 0.0
Schiller: 0.0
Citology: 0.0
Biopsy: 0.0
VARB: 0.0
VARC: 0.0
VAR: 0.0
Num of pregnancies: -0.0029761904761904656
Smokes: -0.0029761904761904656
Smokes (years): -0.0029761904761904656
IUD (years): -0.0029761904761904656
Age: -0.005952380952380931
```

```
# MLPClassifier
abla_model = MLPClassifier(learning_rate_init=0.001)

abla_model.fit(X_train, y_train)
y_pred = abla_model.predict(X_test)

# Evaluate baseline performance
baseline_accuracy = accuracy_score(y_test, y_pred)
print("Baseline Accuracy:", baseline_accuracy)
```

```
# Run ablation experiment
feature_performance = {}
for feature in X_test.columns:
    # Remove one feature
    print('remove ', feature)
    X_ablated = X_train.drop(columns=[feature])

    abla_model.fit(X_ablated, y_train)

    # Evaluate the model's performance after removing the feature
    ablated_accuracy = accuracy_score(y_test, abla_model.predict(X_test.drop(colu

    # Record the performance
    feature_performance[feature] = baseline_accuracy - ablated_accuracy

# Sort features by their impact
sorted_features = sorted(feature_performance.items(), key=lambda x: x[1], reverse

# Print feature importance
print("Feature Ablation Results:")
for feature, impact in sorted_features:
    print(f"{feature}: {impact}")
```

```
Baseline Accuracy: 0.9910714285714286
remove  Age
remove  Number of sexual partners
remove  First sexual intercourse
remove  Num of pregnancies
remove  Smokes
remove  Smokes (years)
remove  Smokes (packs/year)
remove  Hormonal Contraceptives
remove  Hormonal Contraceptives (years)
remove  IUD
remove  IUD (years)
remove  STDs
remove  STDs (number)
remove  STDs:condylomatosis
remove  STDs:cervical condylomatosis
remove  STDs:vaginal condylomatosis
remove  STDs:vulvo-perineal condylomatosis
remove  STDs:syphilis
remove  STDs:pelvic inflammatory disease
remove  STDs:genital herpes
remove  STDs:molluscum contagiosum
remove  STDs:AIDS
remove  STDs:HIV
remove  STDs:Hepatitis B
remove  STDs:HPV
remove  STDs: Number of diagnosis
remove  STDs: Time since first diagnosis
remove  STDs: Time since last diagnosis
remove  Dx:CIN
remove  Dx:HPV
```

```
remove  Dx
remove  Hinselmann
remove  Schiller
remove  Citology
remove  Biopsy
remove  VARB
remove  VARC
remove  VAR
Feature Ablation Results:
Dx:HPV: 0.008928571428571508
Dx: 0.005952380952380931
STDs (number): 0.0029761904761904656
STDs:AIDS: 0.0029761904761904656
Number of sexual partners: 0.0
First sexual intercourse: 0.0
Smokes (years): 0.0
Smokes (packs/year): 0.0
Hormonal Contraceptives: 0.0
Hormonal Contraceptives (years): 0.0
IUD: 0.0
STDs: 0.0
STDs:cervical condylomatosis: 0.0
STDs:vaginal condylomatosis: 0.0
STDs:vulvo-perineal condylomatosis: 0.0
STDs:syphilis: 0.0
STDs:pelvic inflammatory disease: 0.0
STDs:molluscum contagiosum: 0.0
```

Analyses

In the original paper's experiments, they first provide empirical study analyzing the performances of different methods for explaining cervical cancer risk factors, then for each method they contextualize how various formulations of these explanations could be suitable for different patient scenarios and when they might not be appropriate. Finally they provide recommendations to practitioners for utilizing different types of explanations in assessing and determining key factors affecting cervical cancer risk.

Here we aggregate all the local explanations we generated for each of the model and explanation methods. Each produced a dataframe of feature importance of predicting the risk outcome of cervical cancer. The paper identified a set of most important features for assessing cervical cancer risk, we plan to rotate and systematically remove the top 30% most important features one by one and see if the model performance will have degradation, with the hope that it will help us understand the impact of specific features in determining cervical cancer, and analyze the five model's robustness or sensitivity to changes in the experiment.

```
# preload the generated explaination files
url = 'https://drive.google.com/file/d/1m3x6GU2MIdPDqsxGTWABMFIKOReevDxh/view?usp
```

```
output = "glosur_nn_clf.csv"
gdown.download(url, output, fuzzy=True)

url = 'https://drive.google.com/file/d/1--AeLdxbnhplh_n8bPd7zHbPYe5t51m9/view?usp
output = "Kshap_nn_clf.csv"
gdown.download(url, output, fuzzy=True)

url = 'https://drive.google.com/file/d/1j3surG4Mf8CQ0Q9eXKVY0Sp8FGXeFvUl/view?usp
output = "Sshap_nn_clf.csv"
gdown.download(url, output, fuzzy=True)


url = 'https://drive.google.com/file/d/1--aR0nGebISITDmWhOPOXwSNIm-Wkpwv/view?usp
output = "lime_nn_clf.csv"
gdown.download(url, output, fuzzy=True)
```

```
Downloading...
From: https://drive.google.com/uc?id=1m3x6GU2MIdPDqsxGTWABMFIKOReevDxh
To: /content/glosur_nn_clf.csv
100%|██████████| 266k/266k [00:00<00:00, 59.8MB/s]
Downloading...
From: https://drive.google.com/uc?id=1--AeLdxbnhplh_n8bPd7zHbPYe5t51m9
To: /content/Kshap_nn_clf.csv
100%|██████████| 244k/244k [00:00<00:00, 69.8MB/s]
Downloading...
From: https://drive.google.com/uc?id=1j3surG4Mf8CQ0Q9eXKVY0Sp8FGXeFvUl
To: /content/Sshap_nn_clf.csv
100%|██████████| 279k/279k [00:00<00:00, 69.6MB/s]
Downloading...
From: https://drive.google.com/uc?id=1--aR0nGebISITDmWhOPOXwSNIm-Wkpwv
To: /content/lime_nn_clf.csv
100%|██████████| 270k/270k [00:00<00:00, 81.7MB/s]
'lime_nn_clf.csv'
```

```
GloSur = pd.read_csv('glosur_nn_clf.csv')
kernelSHAP = pd.read_csv('Kshap_nn_clf.csv')
samplingSHAP = pd.read_csv('Sshap_nn_clf.csv')
limecontrib = pd.read_csv('lime_nn_clf.csv')

fi_1=fi_2=fi_3=fi_4=fi_5=fi_6=fi_7={f'{x}':0.0 for x in X_test.columns}
res = dict()
# res = global_explanation.get_feature_importance_dict()
for i in list(GloSur.columns):
  res[i]=np.mean(np.abs(GloSur[i]))
fi_1={k: fi_1.get(k, 0) + res.get(k, 0) for k in set(fi_1)}

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)}
```

```
fi_2={k: fi_2.get(k, 0) + res.get(k, 0) for k in set(fi_2)}

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)}


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)}

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


dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
# dics.to_csv("/content/drive/My Drive/dics_logreg_cv_nn_binary.csv", index=False

dics
```

	STDs:condylomatosis	Biopsy	STDs:HIV	Age	Smokes (years)	STDs: Time since first	STDs:sy

						First diagnosis	
0	0.021312	0.015231	0.017400	0.644236	0.463626	0.019354	0
1	0.000647	0.006506	0.000624	0.000612	0.009944	0.014335	0
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
3	0.000473	0.006634	0.000621	0.000389	0.008879	0.013033	0
4	0.010433	0.030910	0.010787	0.010581	0.042087	0.087492	0
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0

7 rows × 39 columns

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


methods=all_fi['Method'].to_list()
weights=[GloSur, kernelSHAP, samplingSHAP, limecontrib]

GloSur
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)
0	-0.681038	-0.160173	0.020245	0.298994	0.199639	-0.445175	-0.024708
1	-0.135118	-0.022558	0.030602	0.363327	0.147175	-0.432408	-0.023608
2	-0.358211	0.043857	0.030980	0.278970	0.220969	-0.445083	-0.024268
3	-0.685670	0.028171	-0.028991	0.324395	0.088697	-0.425910	-0.023811
4	0.272490	-0.018936	0.152739	0.361490	-0.864179	0.060990	-0.001134
...	...	...	...	...	...	...	...
331	-0.547900	0.041318	0.081299	0.230699	0.084786	-0.431668	-0.023894
332	-0.411008	-0.034001	0.156476	-0.620803	-0.926274	0.512066	-0.016176
333	-1.456102	0.044514	0.224356	0.804806	0.125854	-0.434982	-0.023962
334	-0.696389	-0.086208	0.079997	0.354974	0.265924	-0.436939	-0.023800
335	-0.690366	-0.151153	0.032992	0.337837	0.058455	-0.430102	-0.023900

336 rows × 38 columns

kernelSHAP

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)	(
0	0.000000	-0.003916	0.000000	0.000000	0.010206	-0.007433	0.000000	
1	0.000000	0.000000	0.000000	0.001414	0.007191	-0.004398	0.000000	
2	0.000000	0.000000	0.000000	0.000000	0.004907	-0.006517	0.000000	
3	0.000000	0.003620	0.000000	0.000000	0.000000	-0.006526	0.000000	
4	0.000000	-0.001998	0.000000	0.000000	-0.035094	0.009538	0.000000	
...	...	...	...	...	...	...	...	
331	0.000000	0.003164	0.000000	0.000000	0.000000	0.000000	0.000000	
332	-0.003643	-0.004471	0.002166	-0.007080	-0.059064	0.034715	0.001108	
333	0.000000	0.000000	0.000000	0.005522	-0.001527	-0.003835	0.000000	
334	0.000000	0.000000	0.000000	0.000000	0.009515	0.000000	0.000000	
335	0.000000	0.000000	0.000000	0.000000	0.000000	-0.008220	0.000000	

336 rows × 38 columns

samplingSHAP

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smoke (packs yea
0	0.000112	-0.005716	0.000019	0.001403	0.013167	-0.007138	-9.379010e-(
1	-0.000079	-0.001429	-0.000025	0.002102	0.008160	-0.002617	1.941006e-(
2	-0.000090	0.001328	-0.000003	0.000709	0.002975	-0.002020	-3.934057e-(
3	0.000286	0.001171	-0.000022	0.000974	-0.001792	-0.005800	-9.746391e-(
4	-0.000646	-0.000521	0.000067	-0.000727	-0.032697	0.008814	5.647717e-(
...	...	...	...	...	...	...	
331	0.000041	0.000254	0.000022	0.000762	0.001795	-0.000207	-1.688425e-(

332	-0.000162	-0.001341	0.000052	-0.007720	-0.059720	0.035608	3.252351e-(
333	0.000620	0.001793	0.000152	0.001462	0.003190	-0.002439	-5.630290e-(
334	0.000065	-0.004084	0.000013	-0.000803	0.007786	-0.002270	-2.757952e-(
335	0.000196	-0.007025	-0.000064	0.005243	0.004245	-0.010188	-1.242327e-(

336 rows × 38 columns

limecontrib

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)
0	-0.000151	-0.018719	-0.011543	-0.000712	0.027028	-0.030282	-0.002615
1	-0.001696	-0.012987	-0.018210	-0.011062	0.009212	-0.047535	-0.005564
2	0.008282	0.006175	-0.002157	-0.026497	0.030211	-0.028917	0.012512
3	-0.005838	-0.006405	-0.004627	-0.003221	-0.012327	-0.064722	0.009363
4	-0.012229	0.001287	-0.017972	-0.010445	-0.060705	0.104723	0.002100
...	...	...	...	...	...	...	...
331	-0.011772	0.003212	0.024241	0.015557	0.016510	-0.010313	-0.001012
332	-0.008161	-0.016889	-0.003747	-0.037655	-0.062048	0.097309	0.017214
333	0.003671	0.007315	0.031340	0.030007	0.015854	-0.043143	0.007409
334	0.013551	-0.044924	-0.005621	0.002347	0.038675	-0.048181	-0.007858
335	0.005957	-0.005442	-0.012451	-0.002256	-0.052828	-0.033061	-0.011039

336 rows × 38 columns

dicecontrib

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)	H Contrac
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
331	0.0	0.0	0.0	0.0	0.0	0.0	0.0
332	0.0	0.0	0.0	0.0	0.0	0.0	0.0
333	0.0	0.0	0.0	0.0	0.0	0.0	0.0
334	0.0	0.0	0.0	0.0	0.0	0.0	0.0
335	0.0	0.0	0.0	0.0	0.0	0.0	0.0

336 rows × 38 columns

One instance

```
instance=291
var='W'
maxx=10
f=''

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/one_instance_nn_binary.csv')
# print('methods' in one_instance.columns)
one_instance
```

methods	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smok (pack yea
NaN	-0.860745	0.063822	0.189496	0.016497	0.061589	-0.516587	-0.048
NaN	0.004044	0.000000	0.022925	0.001258	-0.006141	-0.009461	0.003
NaN	0.001769	0.000179	0.035989	-0.008974	0.005230	-0.002986	0.004
NaN	-0.015140	0.005207	0.067269	-0.100143	0.000706	-0.047972	0.003

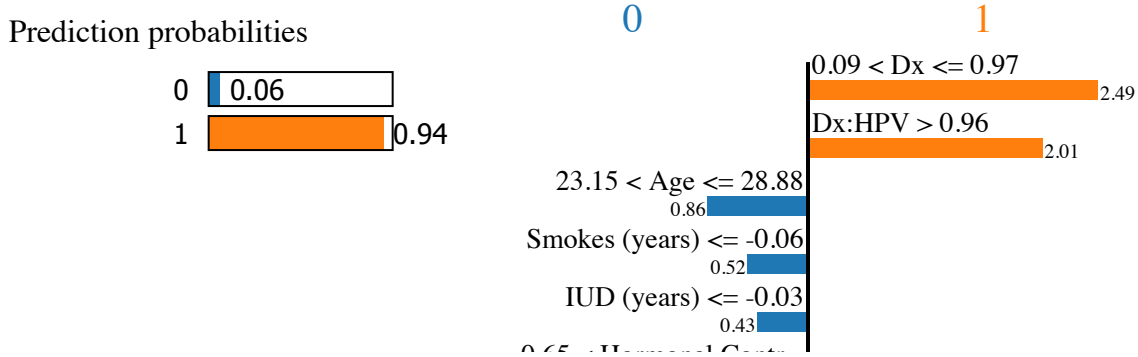
4 rows × 38 columns

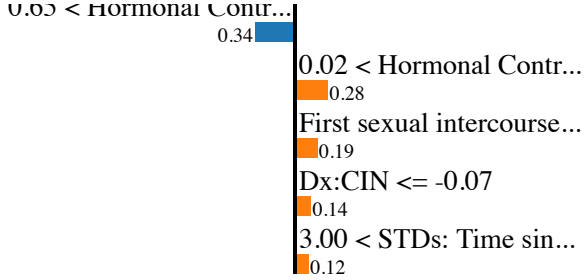
Eval of explainability

For evaluation of local explainabilities generated, we first exmaine the explaniable weights of key features contributed to the model classification. Then we use the RemOve And Retrain (ROAR) (Hooker et al., 2018) mentioned in the paper for the faithfulness metric, where we will iteratively remove a subset of features from a dataset, and retrain the model on the reduced dataset to measure the changes in model accuracy or feature importance in each iteration on the various local explainability we used on the multilayer perceptron model.

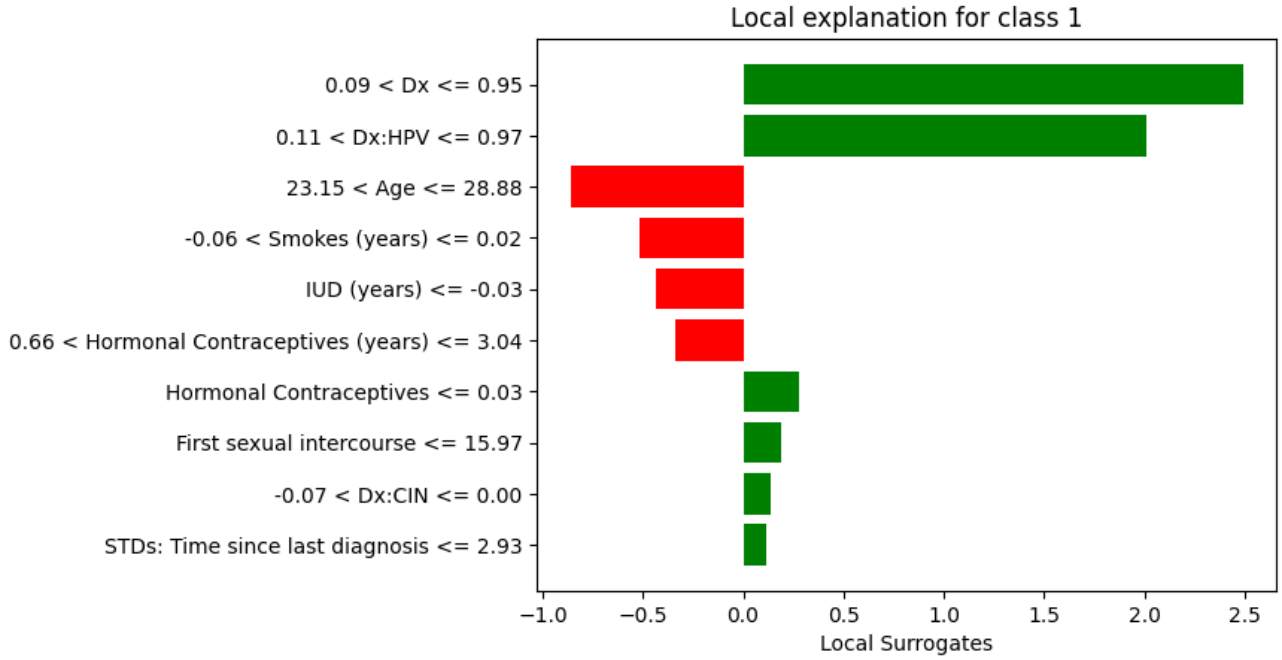
The original paper's aim is to evaluate the effectiveness of each explanation technique mentioned earlier specifically for predicting cervical cancer risk. A number of works deem stability, consistency, compactness, and faithfulness as important facets of interpretability for healthcare domains overall. We follow their specific approaches outlined, and explore the reproduced results and hypotheses validations.

```
model = nn_clf
model.predict(X_test)

explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,mode='classific
exp = explainer.explain_instance(X_test.iloc[instance], model.predict_proba, num_

gscontrib = GloSur
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)
```

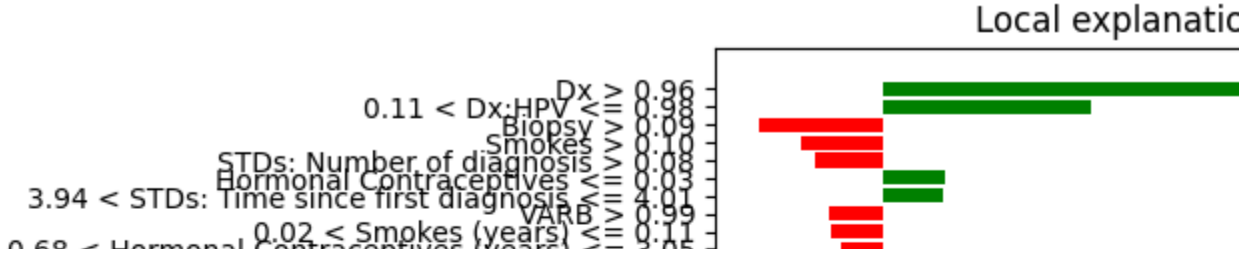


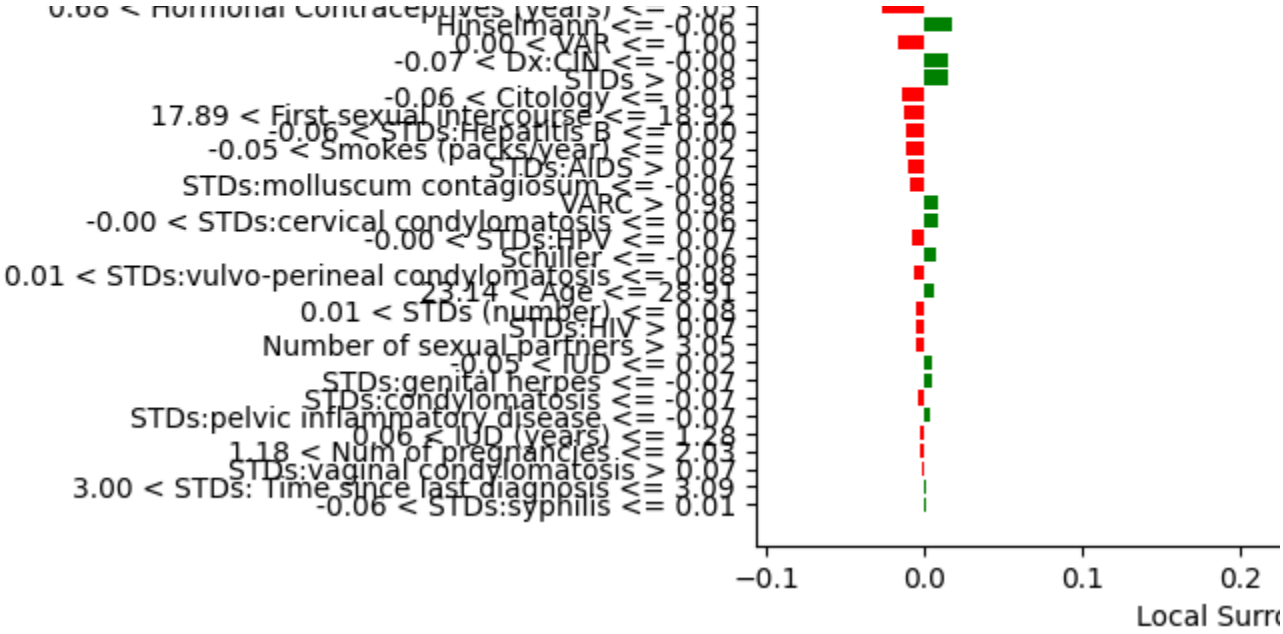



```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Local Surrogates")
# fig.savefig('/content/drive/My Drive/'+str(var)+'surrogate'+str(instance)+str(f
```

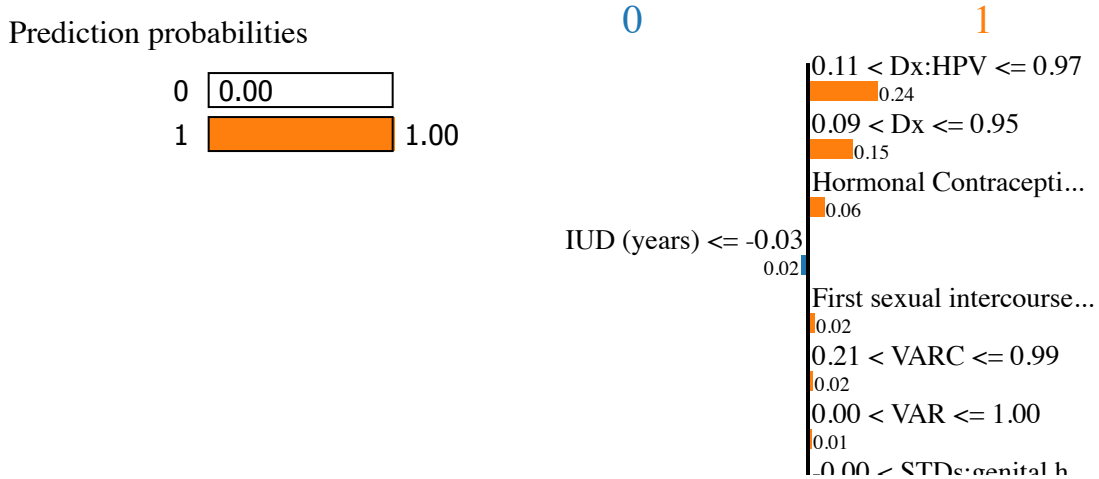


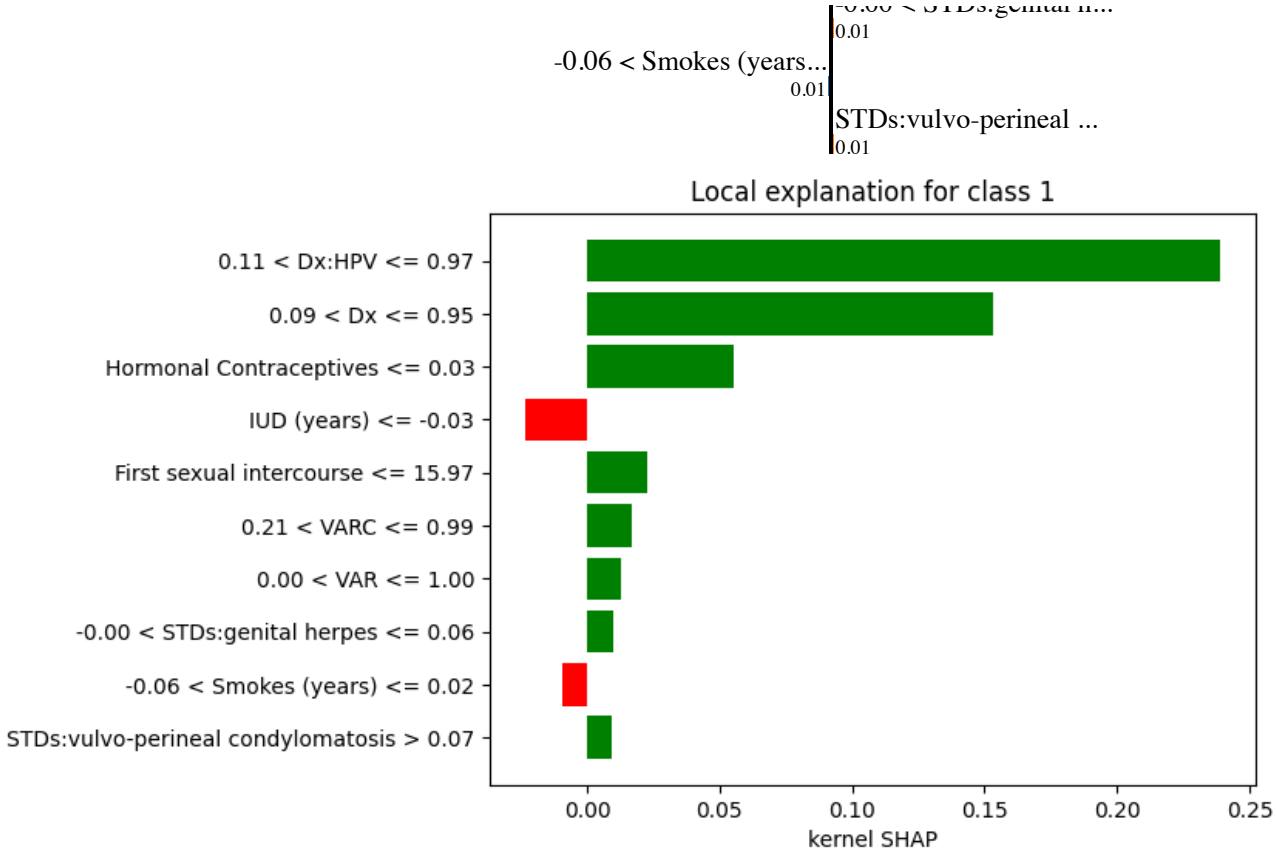

```
%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("Local Surrogates")
# fig.savefig('/content/drive/My Drive/'+str(var)+'surrogate'+str(instance)+str(f
```





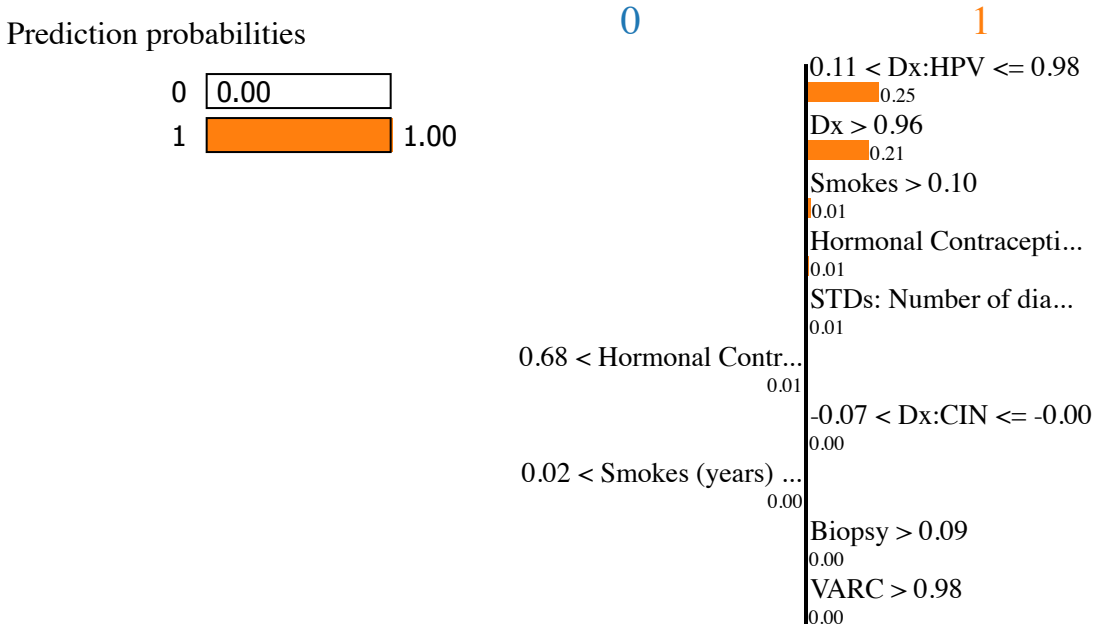

```
## kercontrib
kercontrib = kernelSHAP
items = kercontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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/'+str(var)+'kernelSHAP'+str(instance)+str(
```



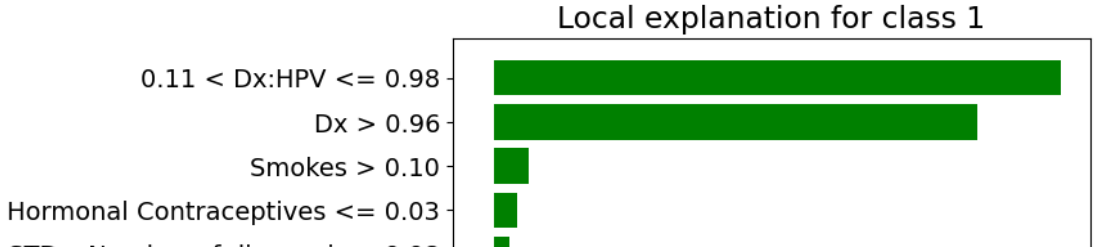


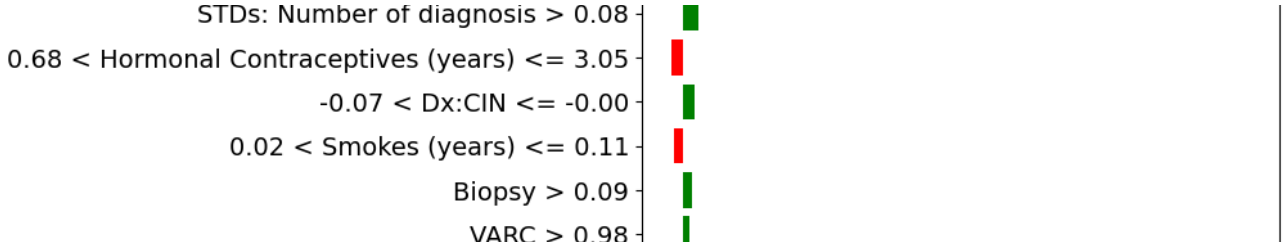

```
# ## trecontrib
# items = trecontrib.iloc[instance].to_dict()
# t = []
# count=0
# for i, item in enumerate(items):
#   if abs(items[item]) > 0.0 :
#     t.append((i, items[item]))

# 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)

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

### samcontrib
items = samplingSHAP.iloc[instance].to_dict()
t = []
```

```
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/'+str(var)+'samplingSHAP'+str(instance)+st
```

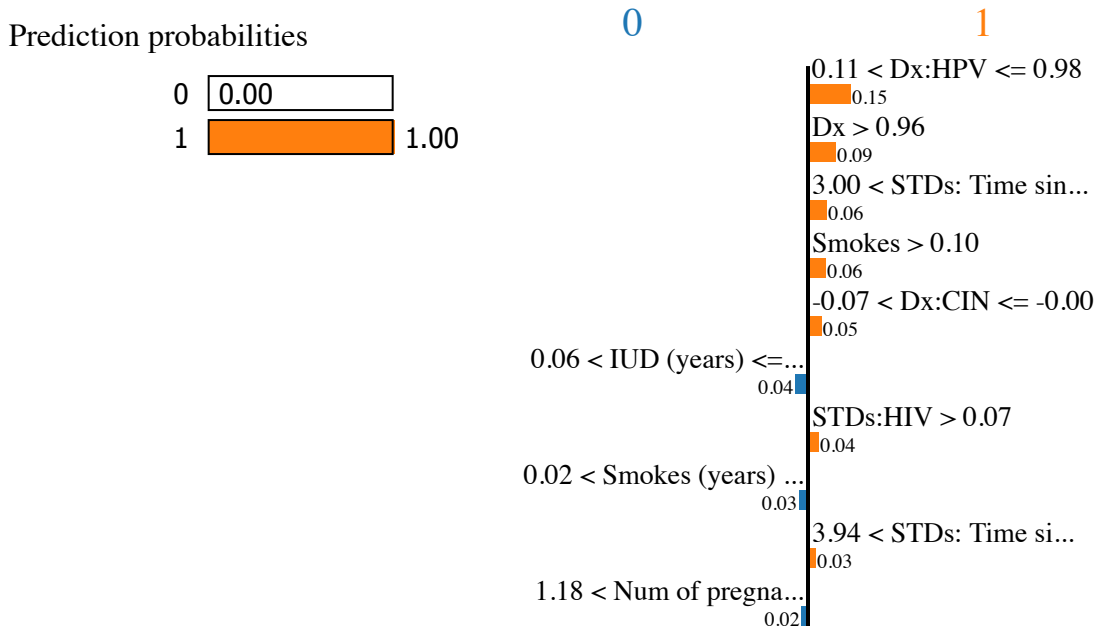


Feature	Value
Dx:HPV	0.93
Dx	0.98
Smokes	0.11
Hormonal Contraceptives	-0.11
STDs: Number of diagnosis	0.15
Hormonal Contraceptives (years)	2.33
Dx:CIN	-0.06
Smokes (years)	0.05
Biopsy	0.10
VARC	1.06



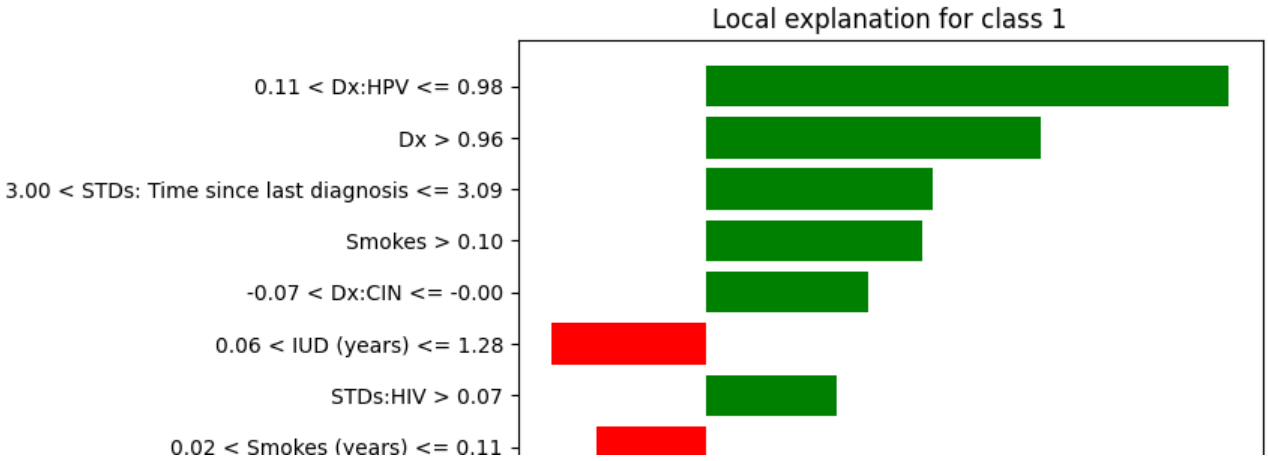



```
### lime
items = limecontrib.iloc[instance].to_dict()
t = []
count=0
for i, item in enumerate(items):
  if abs(items[item]) > 0.0 :
    t.append((i, items[item]))

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)

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

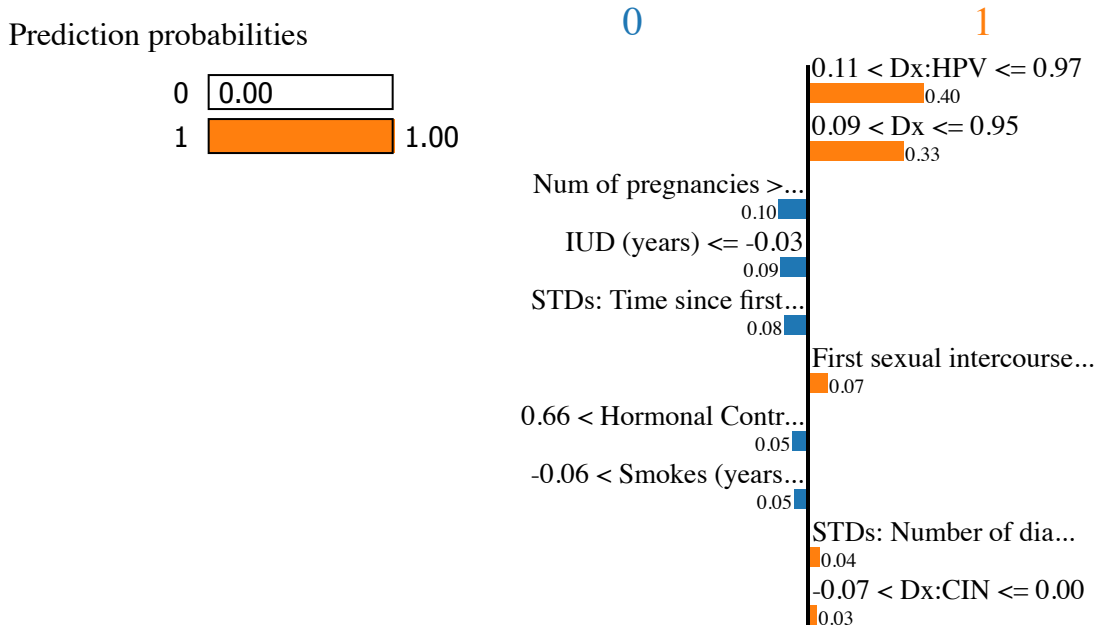


Feature	Value
Dx:HPV	0.93
Dx	0.98
STDs: Time since last diagnosis	3.07
Smokes	0.11
Dx:CIN	-0.06
IUD (years)	0.12
STDs:HIV	0.08

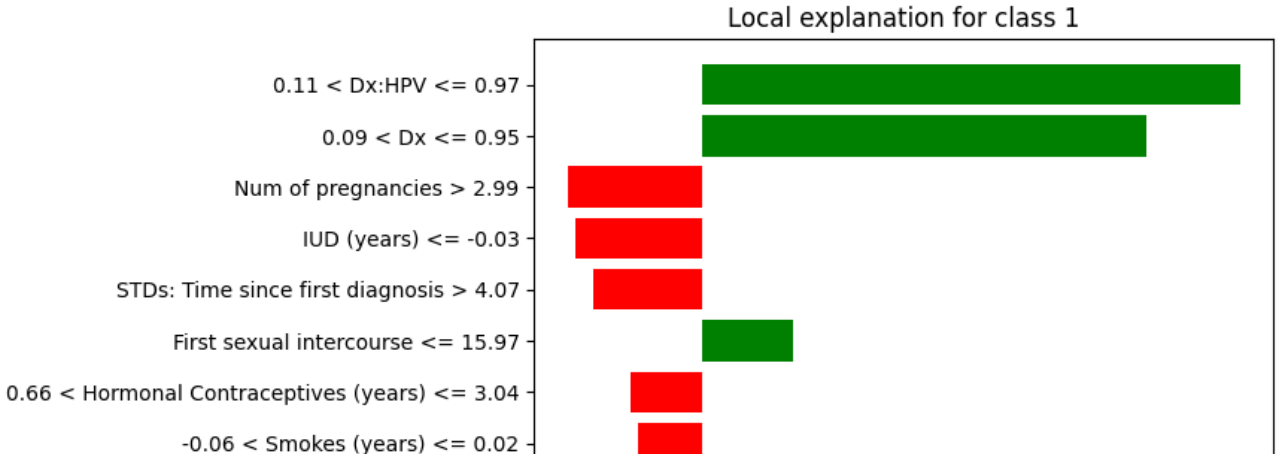
Smokes (years)	0.05
STDs: Time since first diagnosis	3.98
Num of pregnancies	1.98




```
## lime
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)

%matplotlib inline
fig = exp.as_pyplot_figure()
plt.xlabel("LIME")
# fig.savefig('/content/drive/My Drive/'+str(var)+'lime'+str(instance)+str(f)+'2.
```



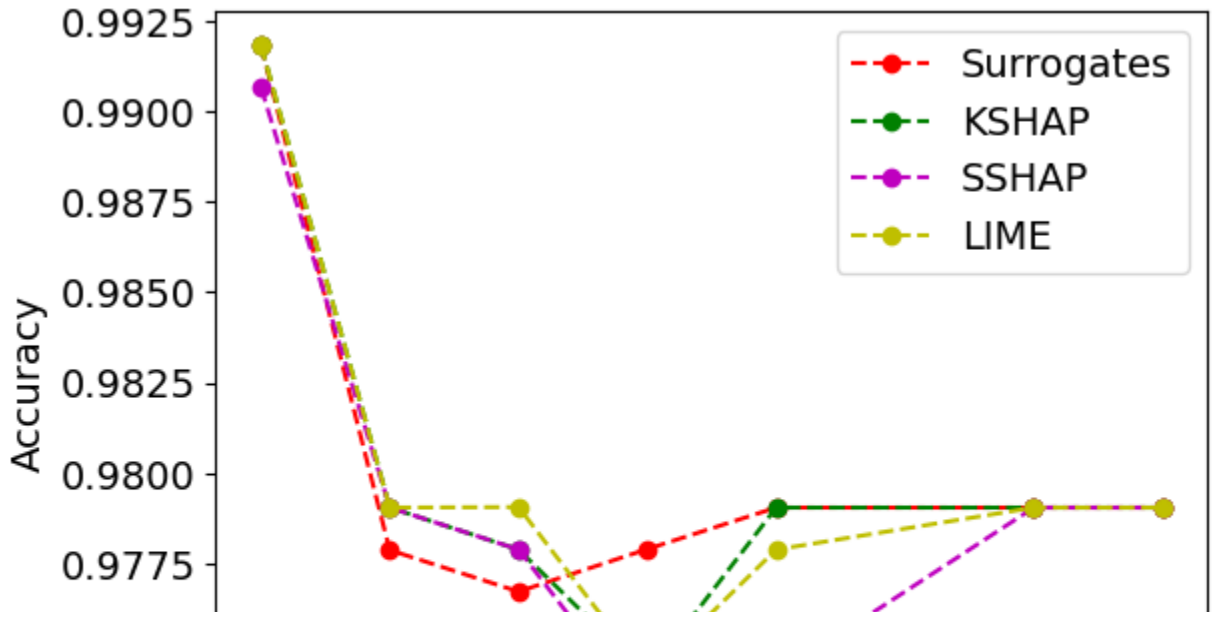
Feature	Value
Dx:HPV	0.87
Dx	0.80
Num of pregnancies	3.08
IUD (years)	-0.05
STDs: Time since first diagnosis	4.13
First sexual intercourse	13.96
Hormonal Contraceptives (years)	1.00
Smokes (years)	0.01
STDs: Number of diagnosis	-0.10
Dx:CIN	-0.03

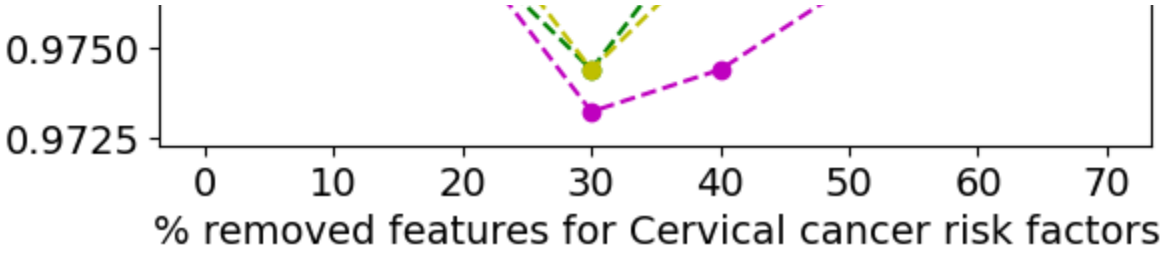


Faithfulness

```
### ROAR
def roar(featImp, feature_to_predict, datapath, savepath, dataname):
  a=['ro--', 'go--', 'mo--', 'yo--', 'co--', 'ko--', 'bo--']
  pourc=[0,10,20,30,40,60,70,90]
  font = {'size'   : 14}
  plt.rc('font', **font)

  for k in range(featImp.shape[0]):
    accuracies=[]
    new=[]
    for i in pourc:
      fi= featImp.iloc[k,:]
      fi.drop('Method', inplace=True)
      fi=fi.to_dict()
      fi=dict(sorted(fi.items(), key=lambda x:x[1], reverse=True))
      # fi.pop('Unnamed: 0')
      fi.pop('VARC')
      fi.pop('VARB')
      fi.pop('VAR')
      fii=list(fi.keys())

      df=datapath
```

```
        df=datapath
        # print(fi)
        top=int((len(df.columns)*i)/100)
        fii = fii[top:]
        new=fii
        new.append(feature_to_predict)
        df=df[new]

        # X=np.array(df[list(df.columns.drop(feature_to_predict))])
        # y=np.array(df[feature_to_predict])
        X = np.array(df[list(df.columns.drop([feature_to_predict]))])
        y = np.array(df[feature_to_predict].to_numpy().reshape(-1, 1))
        # print('X y', X.shape, y.shape)
        model = RandomForestClassifier(random_state = 42)
        if X.shape[1] == 0:
          continue
        model.fit(X, y)
        scores = cross_val_score(model, X, y, cv=10)
        accuracies.append(np.mean(scores))

      plt.plot(pourc[:len(accuracies)], 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+'roar2.png', bbox_inches='tight', dpi=300)
    plt.show()

datapath= risk_factor_df
savepath= '/content/drive/My Drive/'
dataname='Cervical cancer'

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





Contribution

```
from shapash.explainer.consistency import Consistency
from shapash import SmartExplainer
cns=Consistency()
```

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

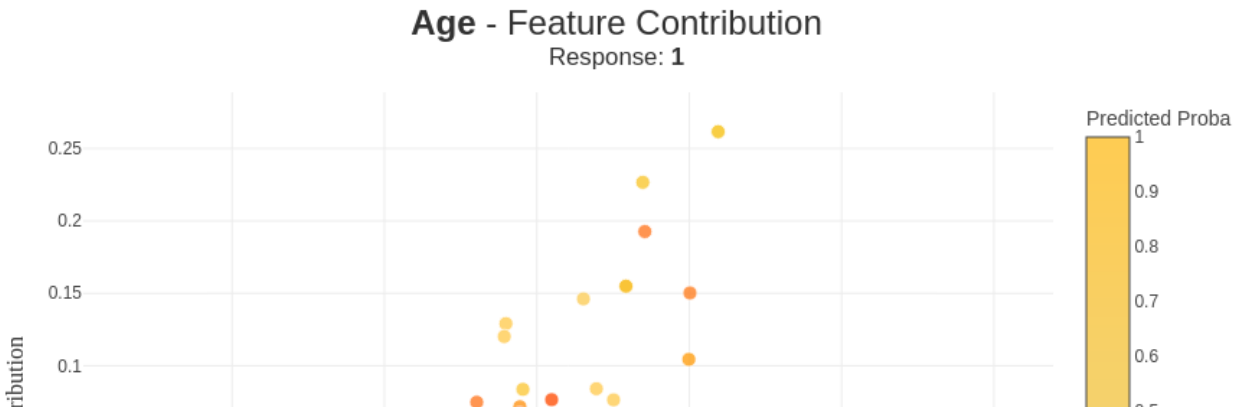
```
model = nn_clf
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test)
```

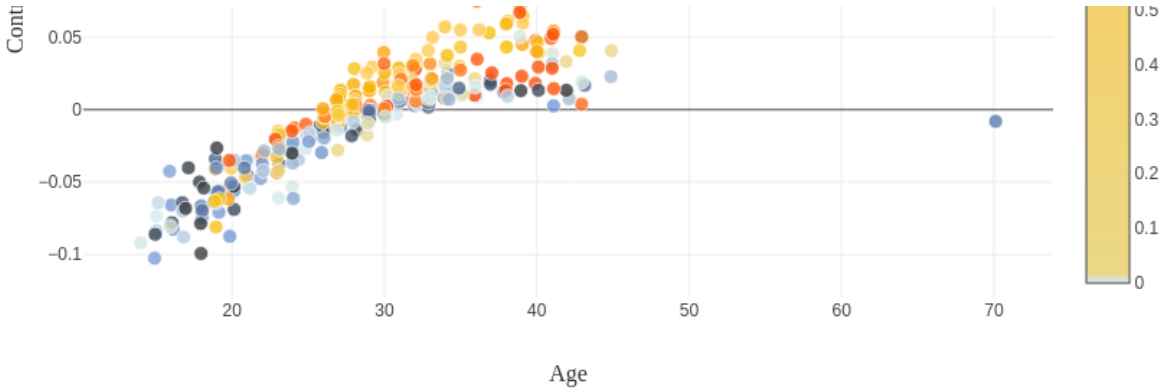
```
    INFO: Shap explainer type - shap.explainers.PermutationExplainer()
    PermutationExplainer explainer: 337it [00:39,  7.33it/s]
```

```
img=xpl.plot.contribution_plot(0)
# img.write_image('/content/drive/My Drive/contribage.png')
```

```
# contribage.png
im_url = 'https://drive.google.com/file/d/1-EKmCJlq6ecNwHU99fu1LWBDx7Ja78qW/view?
output = "contribage.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

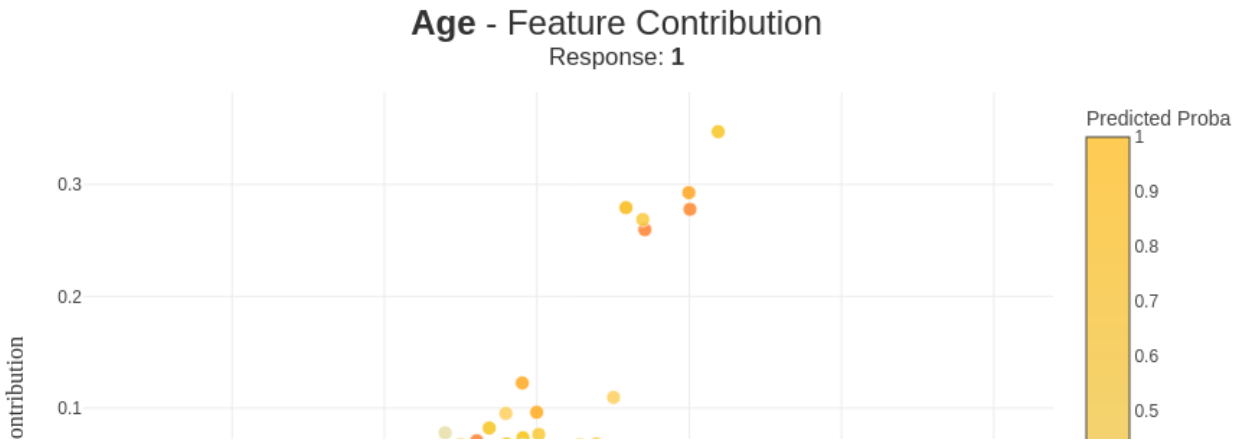
```
    Downloading...
    From: https://drive.google.com/uc?id=1-EKmCJlq6ecNwHU99fu1LWBDx7Ja78qW
    To: /content/contribage.png
    100%|██████████| 82.0k/82.0k [00:00<00:00, 55.3MB/s]
```

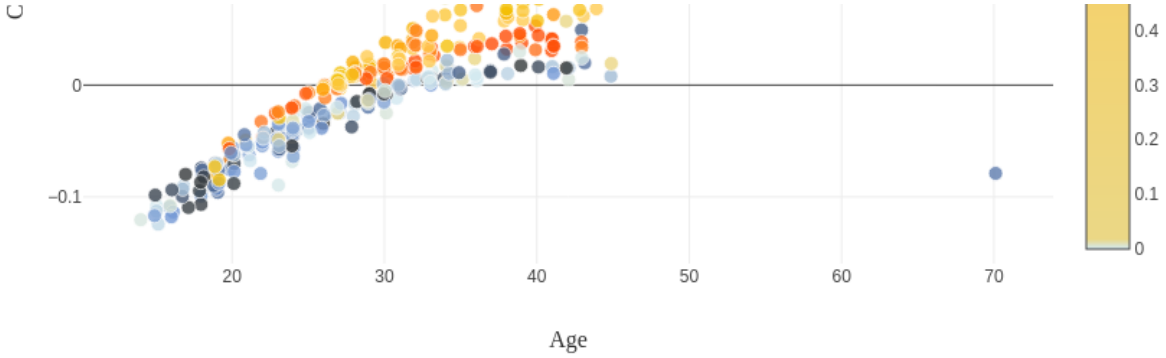





```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)

img=xpl.plot.contribution_plot(0)
# img.write_image('/content/drive/My Drive/contribagekernel.png')

# contribagekernel.png
im_url = 'https://drive.google.com/file/d/1-LiCVwiLG-roQUoQuJ7tPtSPPARgRSGm/view?
output = "contribagekernel.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

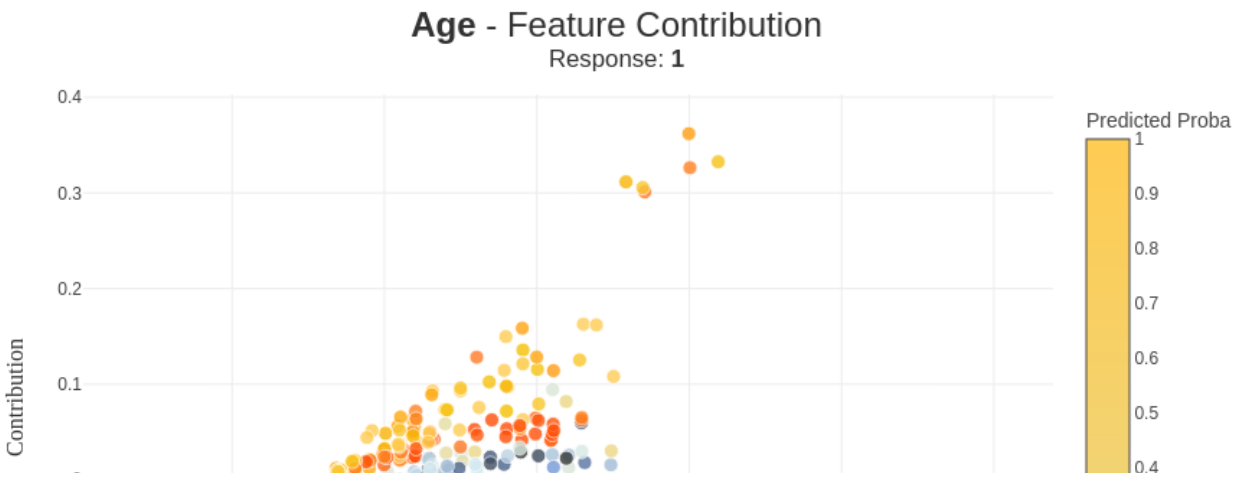
```
Downloading...
From: https://drive.google.com/uc?id=1-LiCVwiLG-roQUoQuJ7tPtSPPARgRSGm
To: /content/contribagekernel.png
100%|██████████| 78.2k/78.2k [00:00<00:00, 63.7MB/s]
```

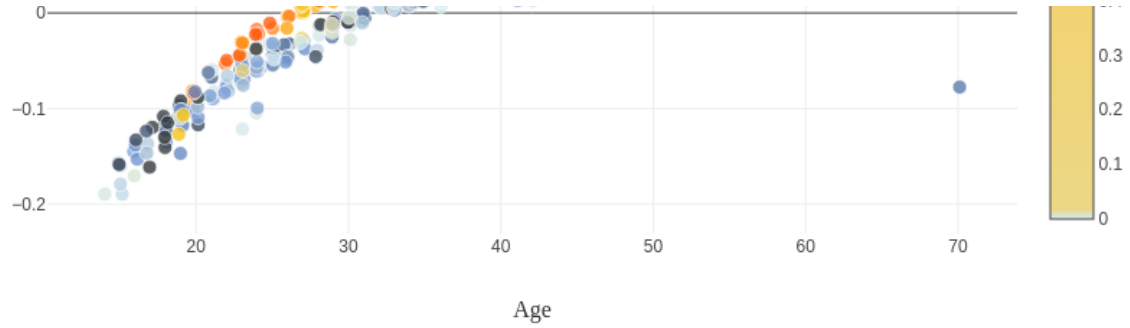





```
xpl = SmartExplainer(model=model)
samcontrib = samplingSHAP
xpl.compile(x=X_test,contributions=samcontrib)
img = xpl.plot.contribution_plot(0)
# img.write_image('/content/drive/My Drive/contribagesampling.png')

# contribagesampling.png
im_url = 'https://drive.google.com/file/d/1-MM3uvePYI_hl6_IMgjuRYhZiIZUx62p/view?
output = "contribagesamplingcontribagesampling.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

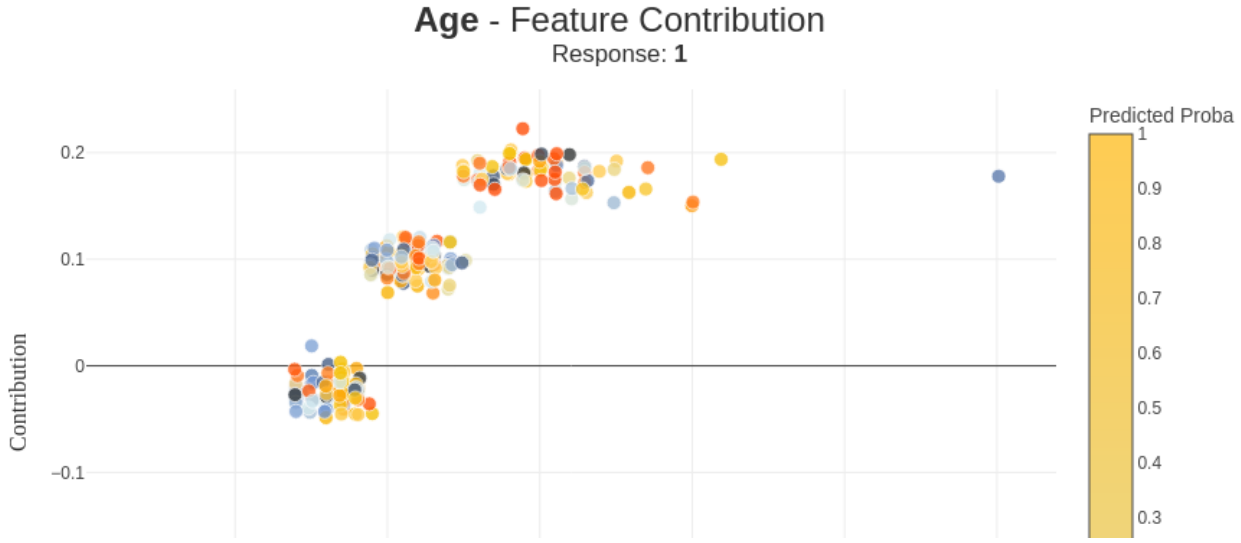
```
Downloading...
From: https://drive.google.com/uc?id=1-MM3uvePYI_hl6_IMgjuRYhZiIZUx62p
To: /content/contribagesamplingcontribagesampling.png
100%|██████████| 81.0k/81.0k [00:00<00:00, 63.6MB/s]
```

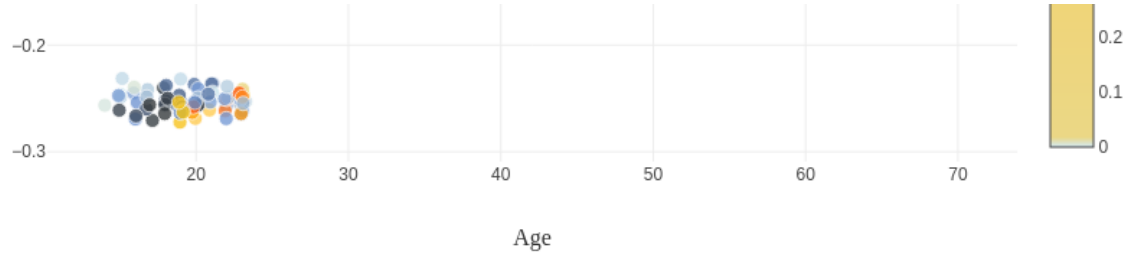




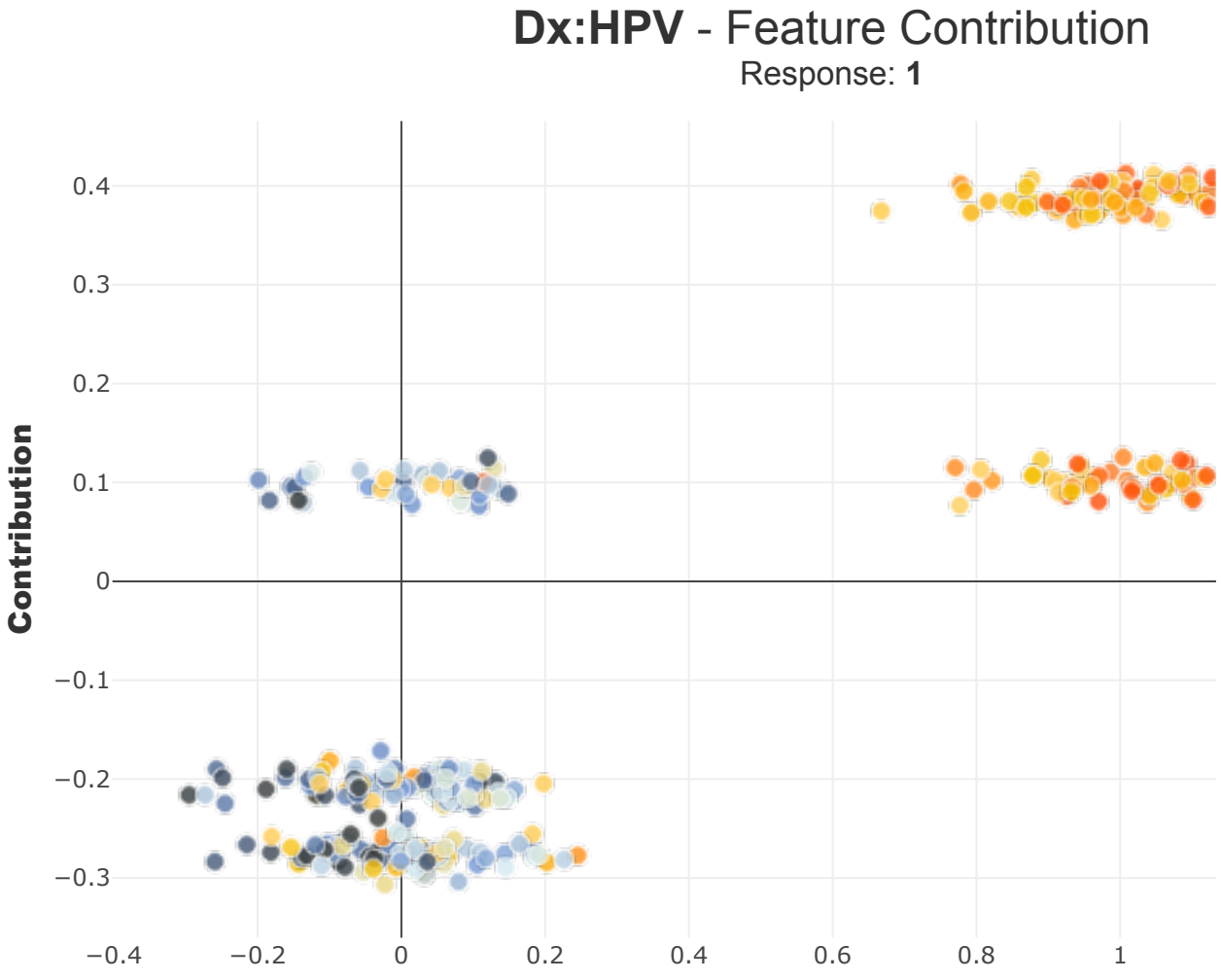

```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img = xpl.plot.contribution_plot(0)
# img.write_image('/content/drive/My Drive/contribagelime.png')

# contribagelime
im_url = 'https://drive.google.com/file/d/1-MqAa_CV22sdnA-zQFkygzHRnlqWQpIm/view?
output = "contribagelime.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

```
Downloading...
From: https://drive.google.com/uc?id=1-MqAa_CV22sdnA-zQFkygzHRnlqWQpIm
To: /content/contribagelime.png
100%|██████████| 76.3k/76.3k [00:00<00:00, 55.2MB/s]
```






```
# Contribution plot for features
fig_image = xpl.plot.contribution_plot(29) # change col name for features
fig_image.show()
```



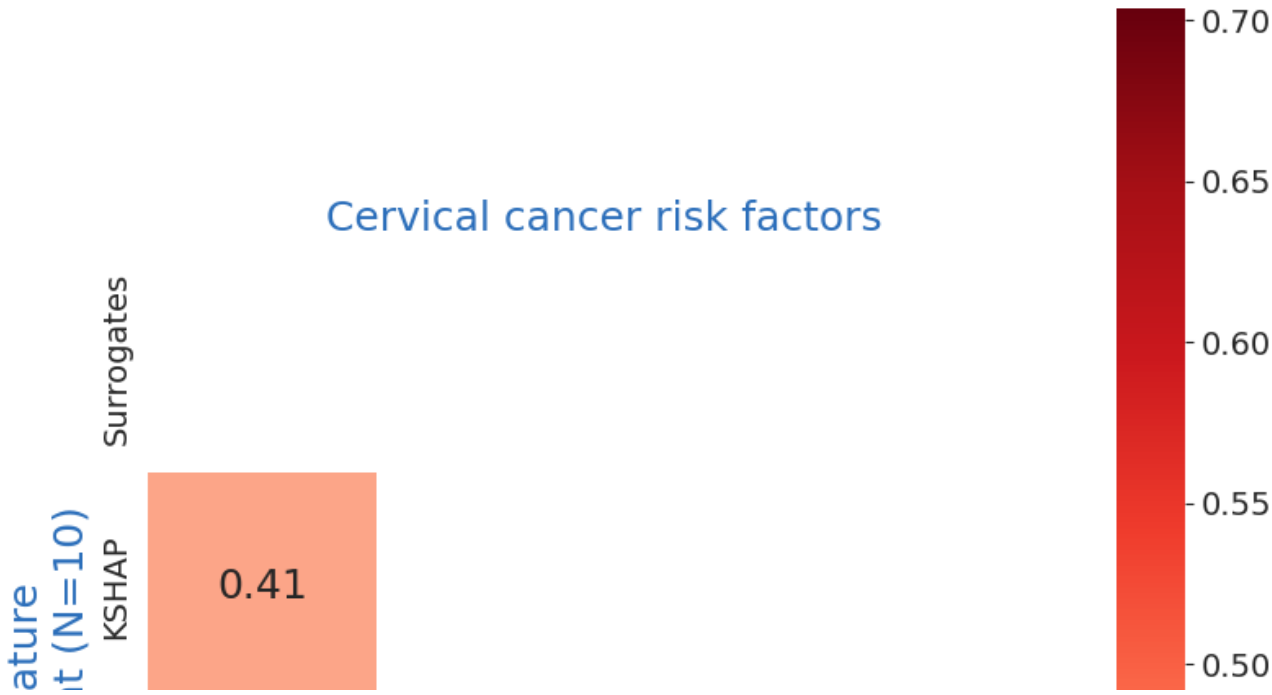
Dx:HPV

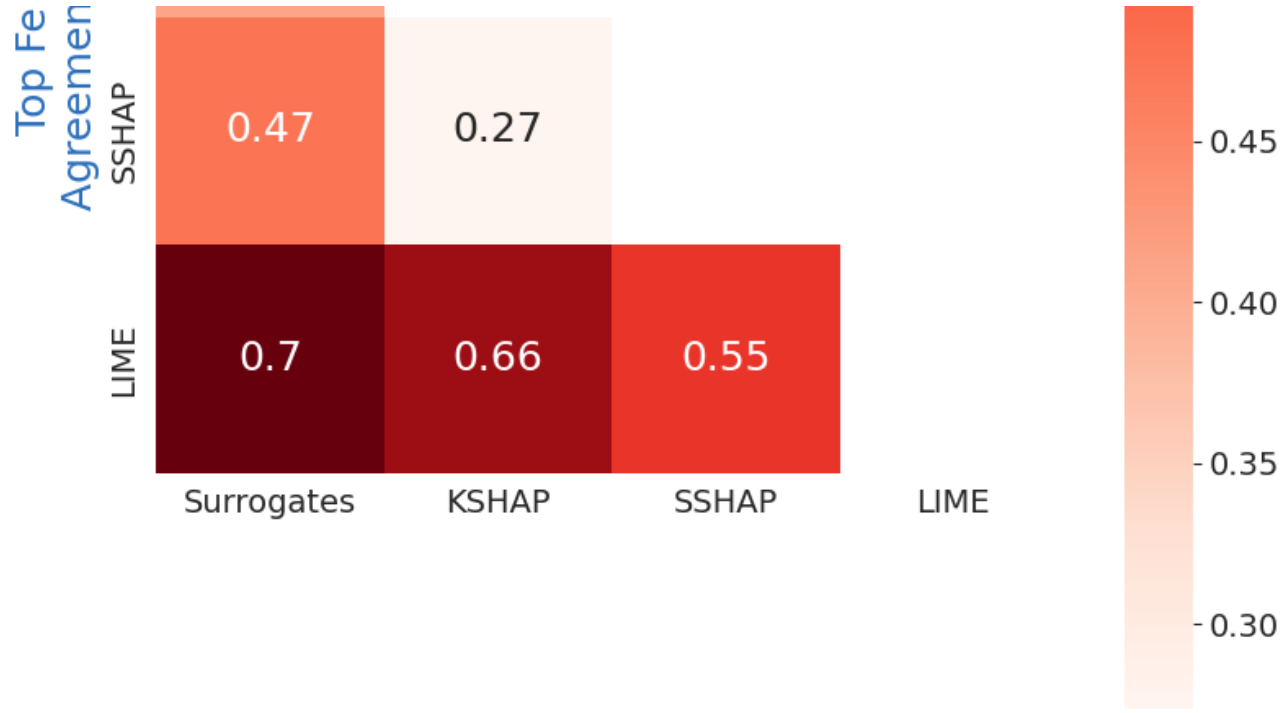
Consistency

```
# Consistency
pairwise_consistency = cns.calculate_all_distances(methods, weights)
test = pairwise_consistency[1].round(2)

test.style.background_gradient(cmap='Paired_r')
```

	Surrogates	KSHAP	SSHAP	LIME
Surrogates	0.000000	0.410000	0.470000	0.700000
KSHAP	0.410000	0.000000	0.270000	0.660000
SSHAP	0.470000	0.270000	0.000000	0.550000
LIME	0.700000	0.660000	0.550000	0.000000

```
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={'fontsiz
    ax.set_title("Cervical cancer risk factors", color='xkcd:medium blue', fontsi
    ax.set_ylabel('Top Feature\nAgreement (N=10)', color='xkcd:medium blue', font
    plt.show()
```






```
# fig.savefig('/content/drive/My Drive/consistency.png', bbox_inches='tight', dpi
```

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

```
Surrogates 0.4
KSHAP 0.34
SSHAP 0.32
LIME 0.48
```

Compactness

```
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 contri
    output_top_features = np.sum(top_features[:, :], axis=1)
```

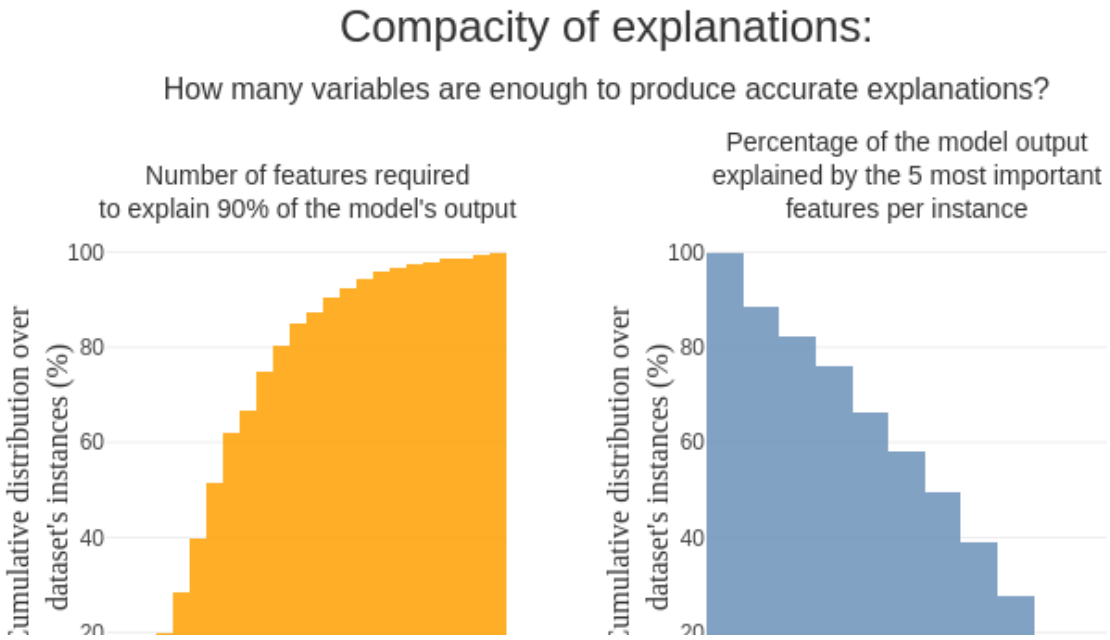
```
    output_all_features = np.sum(contributions[:, :], axis=1)

    if mode == "regression":
        distance = abs(output_top_features - output_all_features) / abs(output_al
    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 c
    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]
            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_featu
        features_needed = get_min_nb_features(selection, contributions, case, dis
        distance_reached = get_distance(selection, contributions, case, nb_featur
        # We clip large approximations to 100%
        distance_reached = np.clip(distance_reached, 0, 1)
        return {"features_needed": features_needed, "distance_reached": distance_

compacities=[]
for weight in weights:
  rr=compute_features_compacity(case="classification", contributions=weight, sele
  #rr=compute_features_compacity(case="classification", contributions=weight, sel
  compacities.append(pd.DataFrame.from_dict(rr))

maxx=[]
```

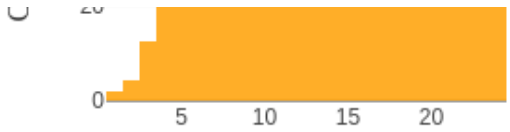
```
for c in compacities:
  maxx.append(c.iloc[c.distance_reached.idxmax()].tolist())
compacity=pd.DataFrame(data=maxx, columns=['features_needed', 'distance_reached']
compacity['Method']=methods
compacity.set_index('Method', drop=True).round(2)
```

Method	features_needed	distance_reached
Surrogates	6.0	1.00
KSHAP	1.0	0.20
SSHAP	1.0	0.13
LIME	1.0	0.48

```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=gscontrib)
img = xpl.plot.compacity_plot()
# img.write_image('/content/drive/My Drive/'+str(var)+'compactgs.png')

# Wcompactgs
im_url = 'https://drive.google.com/file/d/1-fyy2XGsH4x-NUhsag3PxH9Od7a_ddtC/view?
output = "Wcompactgs.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

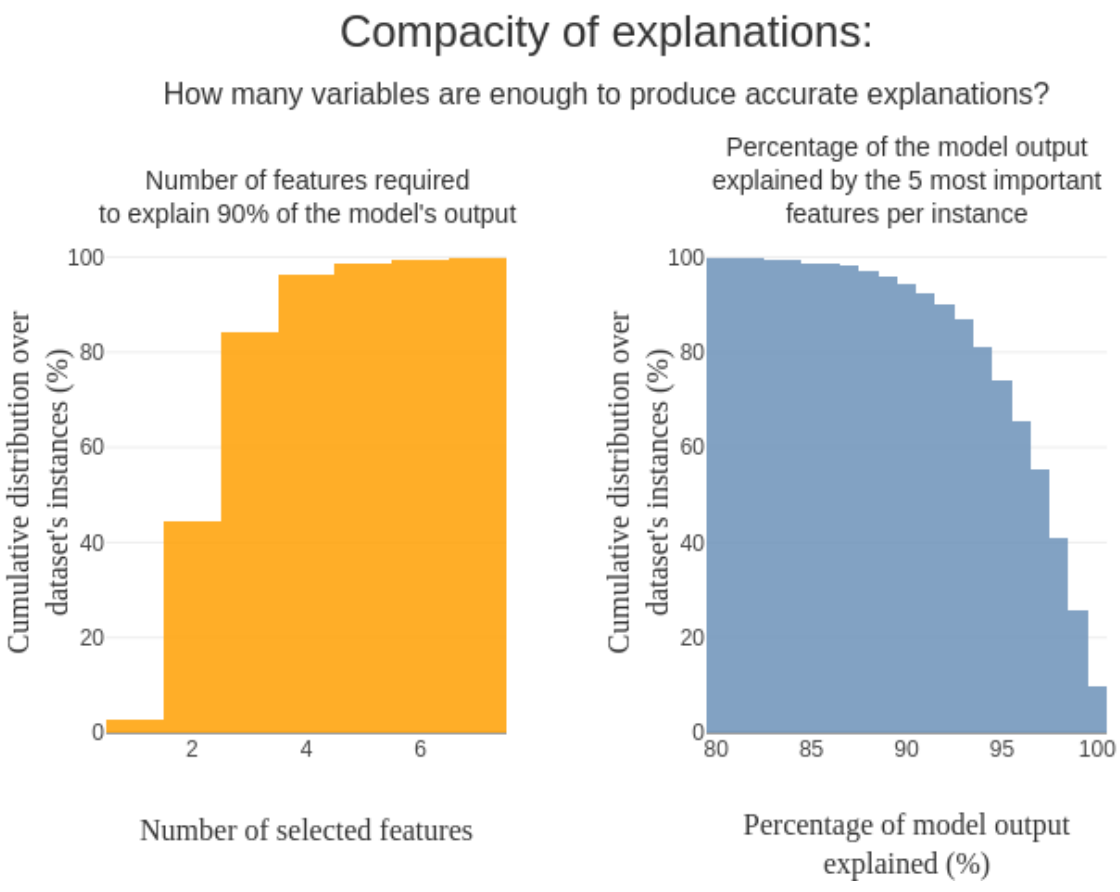
```
Downloading...
From: https://drive.google.com/uc?id=1-fyy2XGsH4x-NUhsag3PxH9Od7a_ddtC
To: /content/Wcompactgs.png
100%|██████████| 50.9k/50.9k [00:00<00:00, 58.0MB/s]
```





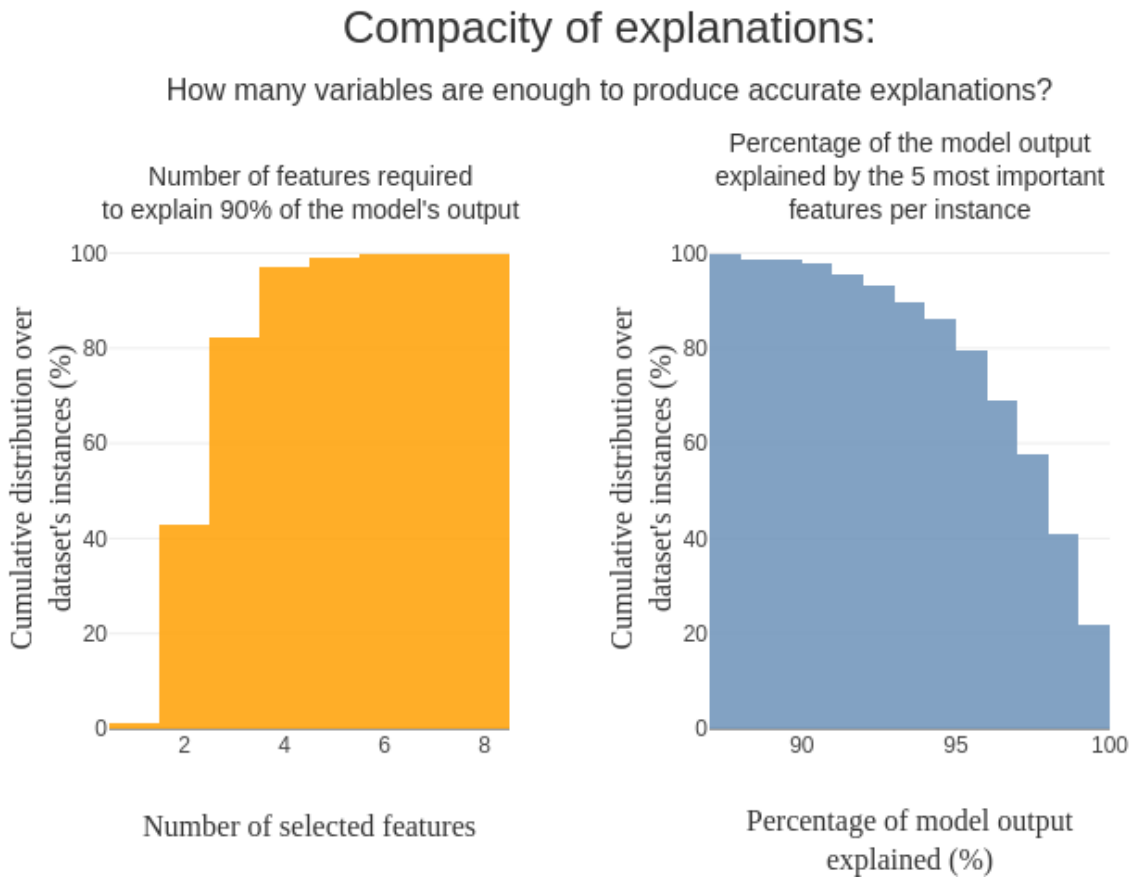

```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=kercontrib)
img = xpl.plot.compacity_plot()
# img.write_image('/content/drive/My Drive/'+str(var)+'compactkernel.png')

# Wcompactkernel
im_url = 'https://drive.google.com/file/d/1-mpL7zHmiLdKlvDzC4kJPM_CJVfHiyG6/view?
output = "Wcompactkernel.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

```
Downloading...
From: https://drive.google.com/uc?id=1-mpL7zHmiLdKlvDzC4kJPM_CJVfHiyG6
To: /content/Wcompactkernel.png
100%|██████████| 51.5k/51.5k [00:00<00:00, 34.1MB/s]
```



```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=samcontrib)
img = xpl.plot.compacity_plot()
# img.write_image('/content/drive/My Drive/'+str(var)+'compactsampling.png')

# Wcompactsampling
im_url = 'https://drive.google.com/file/d/1-xIiDFETg8H2OoIrHtLIlivYK046EJqQ/view?
output = "Wcompactsampling.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

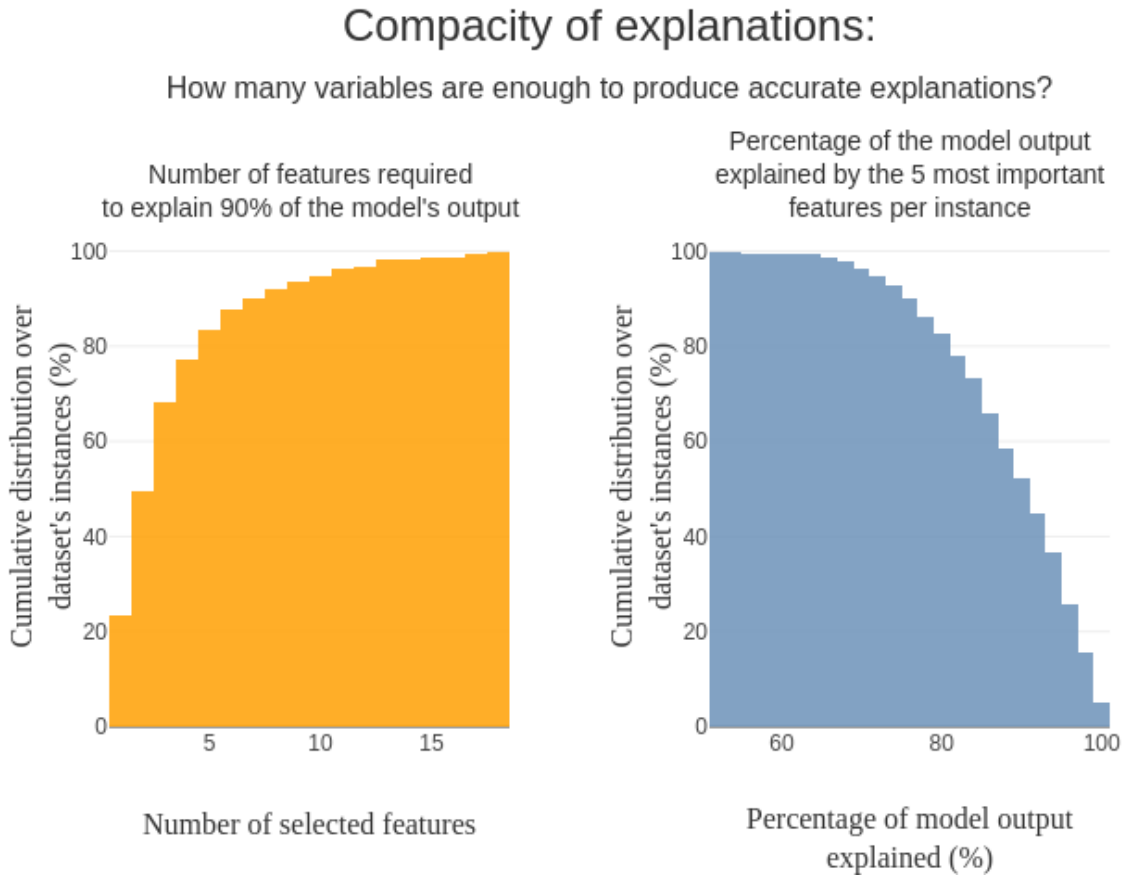
```
Downloading...
From: https://drive.google.com/uc?id=1-xIiDFETg8H2OoIrHtLIlivYK046EJqQ
To: /content/Wcompactsampling.png
100%|██████████| 51.1k/51.1k [00:00<00:00, 55.3MB/s]
```




```
xpl = SmartExplainer(model=model)
xpl.compile(x=X_test,contributions=limecontrib)
img= xpl.plot.compacity_plot()
# img.write_image('/content/drive/My Drive/'+str(var)+'compactlime.png')
```

```
# Wcompactlime
im_url = 'https://drive.google.com/file/d/1-yIOvEP9ZArOCs5KGa7VH2y2Qi2Uc7uK/view?
output = "Wcompactlime.png"
gdown.download(im_url, output, fuzzy=True)
img = cv2.imread(output)
cv2_imshow(img)
```

```
Downloading...
From: https://drive.google.com/uc?id=1-yIOvEP9ZArOCs5KGa7VH2y2Qi2Uc7uK
To: /content/Wcompactlime.png
100%|██████████| 50.8k/50.8k [00:00<00:00, 54.8MB/s]
```



Stability

```
def _compute_distance(x1, x2, mean_vector, epsilon=0.0000001):
    diff = np.sum(np.abs(x1 - x2) / (mean_vector + epsilon))
    return diff


def _compute_similarities(instance, dataset):
    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):
    # 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],
            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 (percer
    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 cc
    3. Filter neighbors whose distance is too big compared to a certain threshold
    """
    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
        all_neighbors = np.append(all_neighbors, neighbors, axis=0)

    # Calculate predictions for all instances and corresponding neighbors
```

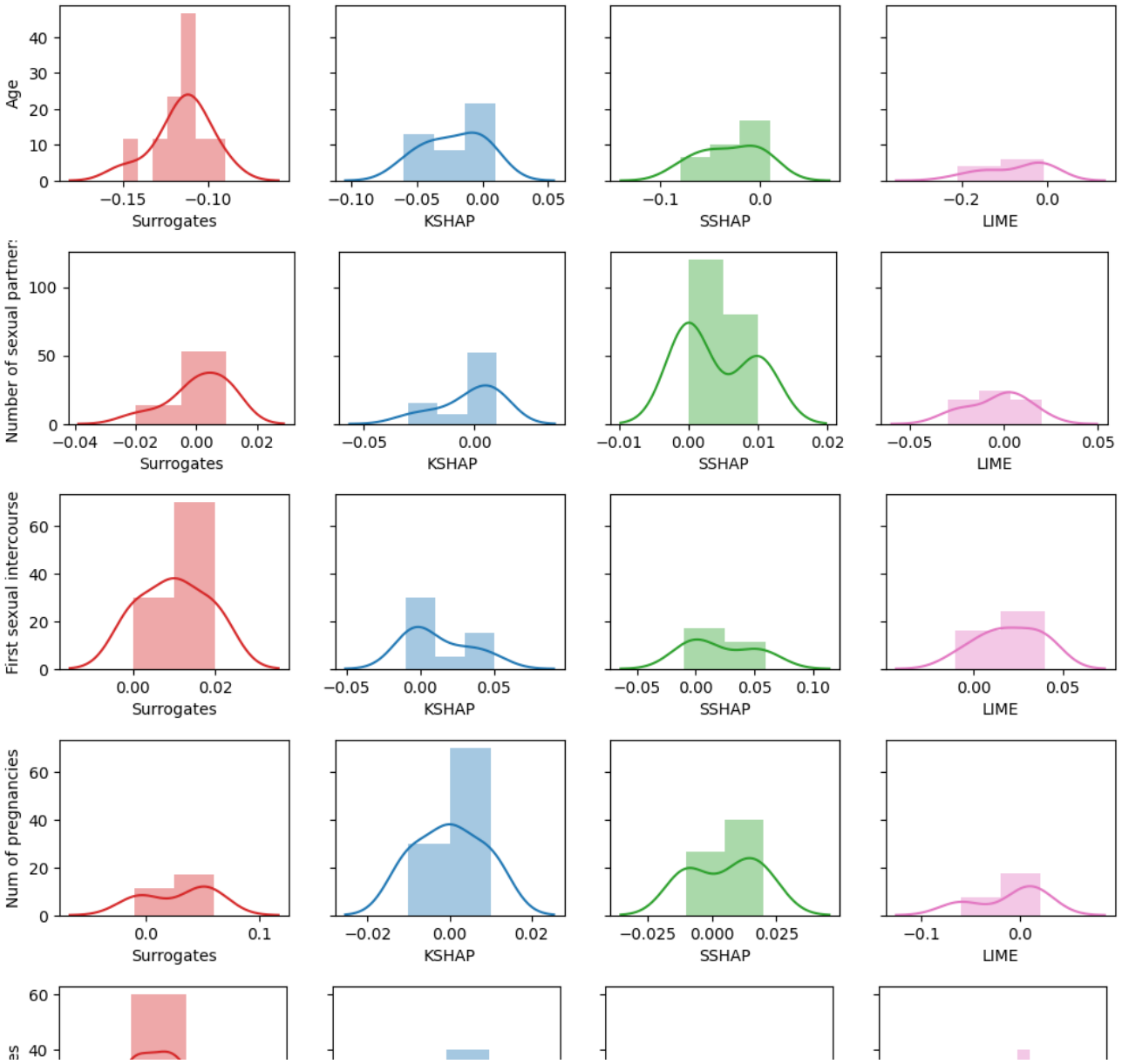
```
    # Calculate predictions for all instances and corresponding neighbors
    if mode == "regression":
        # For XGB it is necessary to add columns in df, otherwise columns mismatc
        predictions = model.predict(pd.DataFrame(all_neighbors[:, :-1], columns=c
    elif mode == "classification":
        predictions = model.predict_proba(pd.DataFrame(all_neighbors[:, :-1], col
    # Add prediction column
    all_neighbors = np.append(all_neighbors, predictions.reshape(all_neighbors.sh
    # 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])
    elif mode == "classification":
        for i, neighbors in enumerate(all_neighbors):
            all_neighbors[i] = neighbors[abs(neighbors[:, -1] - neighbors[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
    """
    # Extract SHAP values for instance and neighbors
    # :-2 indicates that two columns are disregarded : distance to instance and m
    ind = pd.merge(x_encoded.reset_index(), pd.DataFrame(instance[:, :-2], columr
        .set_index(x_encoded.index.name if x_encoded.index.name is not None else
    # 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
    average_diff = np.divide(norm_shap_values.std(axis=0), norm_abs_shap_values.m
                             out=np.zeros(norm_abs_shap_values.shape[1]),
```

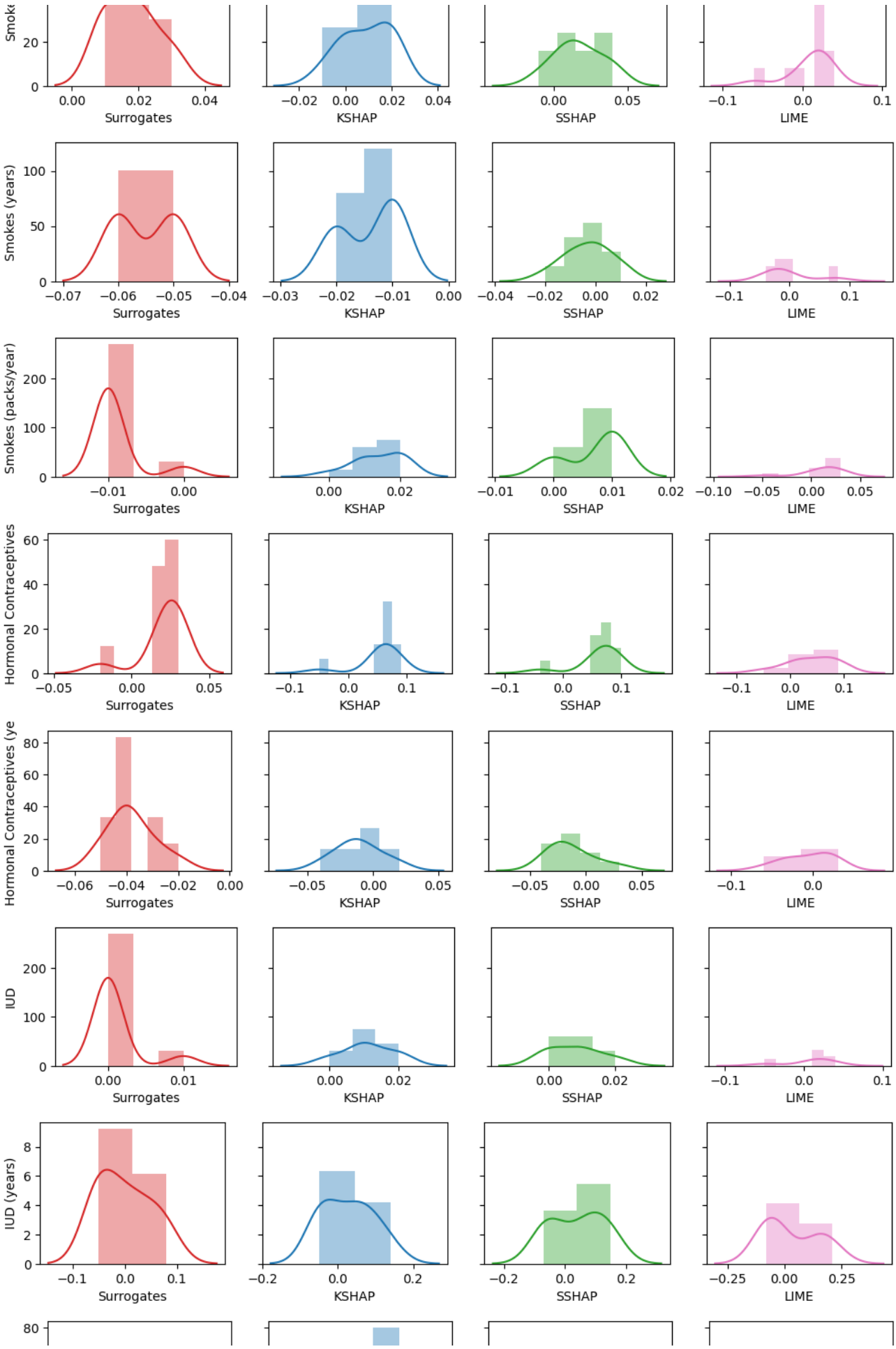
```
                                where=norm_abs_shap_values.mean(axis=0) != 0)

    return norm_shap_values, average_diff, norm_abs_shap_values[0, :]


def get_distance(selection, contributions, mode, nb_features):
    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 contri
    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_al
    elif mode == "classification":
        distance = abs(output_top_features - output_all_features)
    return distance


def compute_features_stability (case, x, selection, contributions):
      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, contribut
          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_neigh
          features_stability = {"variability": variability, "amplitude": amplitud
          return features_stability
```

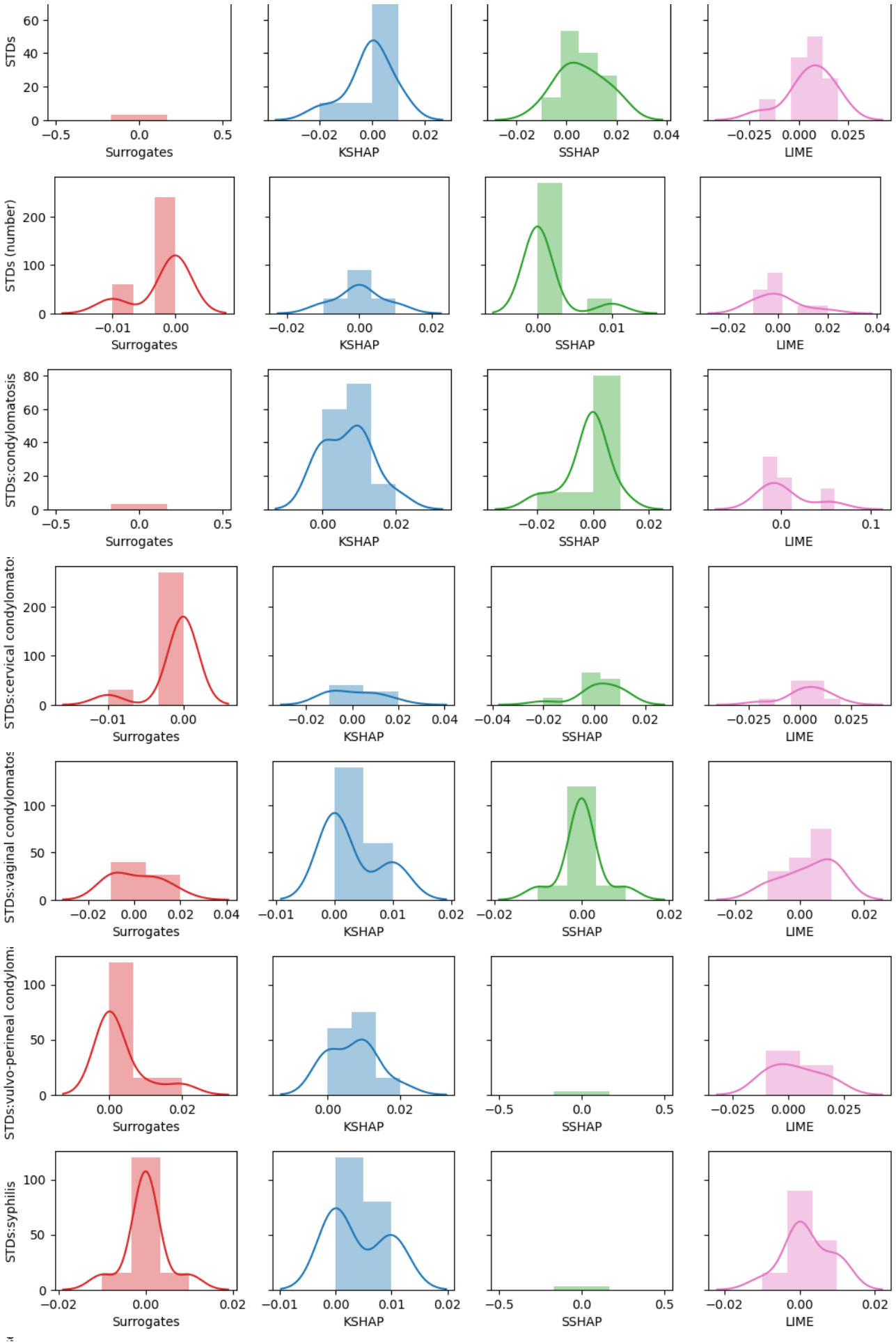
one instance

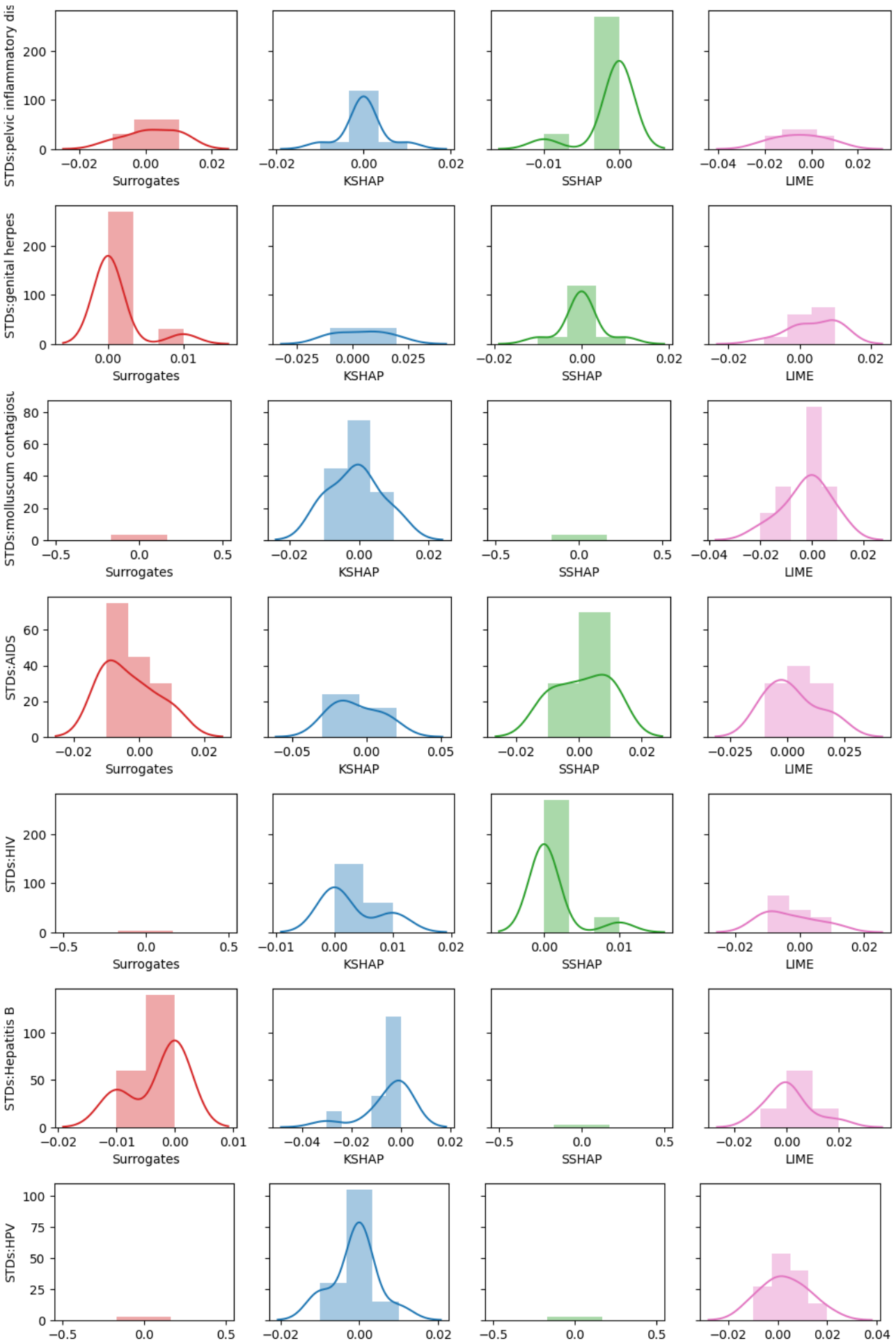
```
frames=[]
features=list(X_test.columns)
for weight in weights:
```

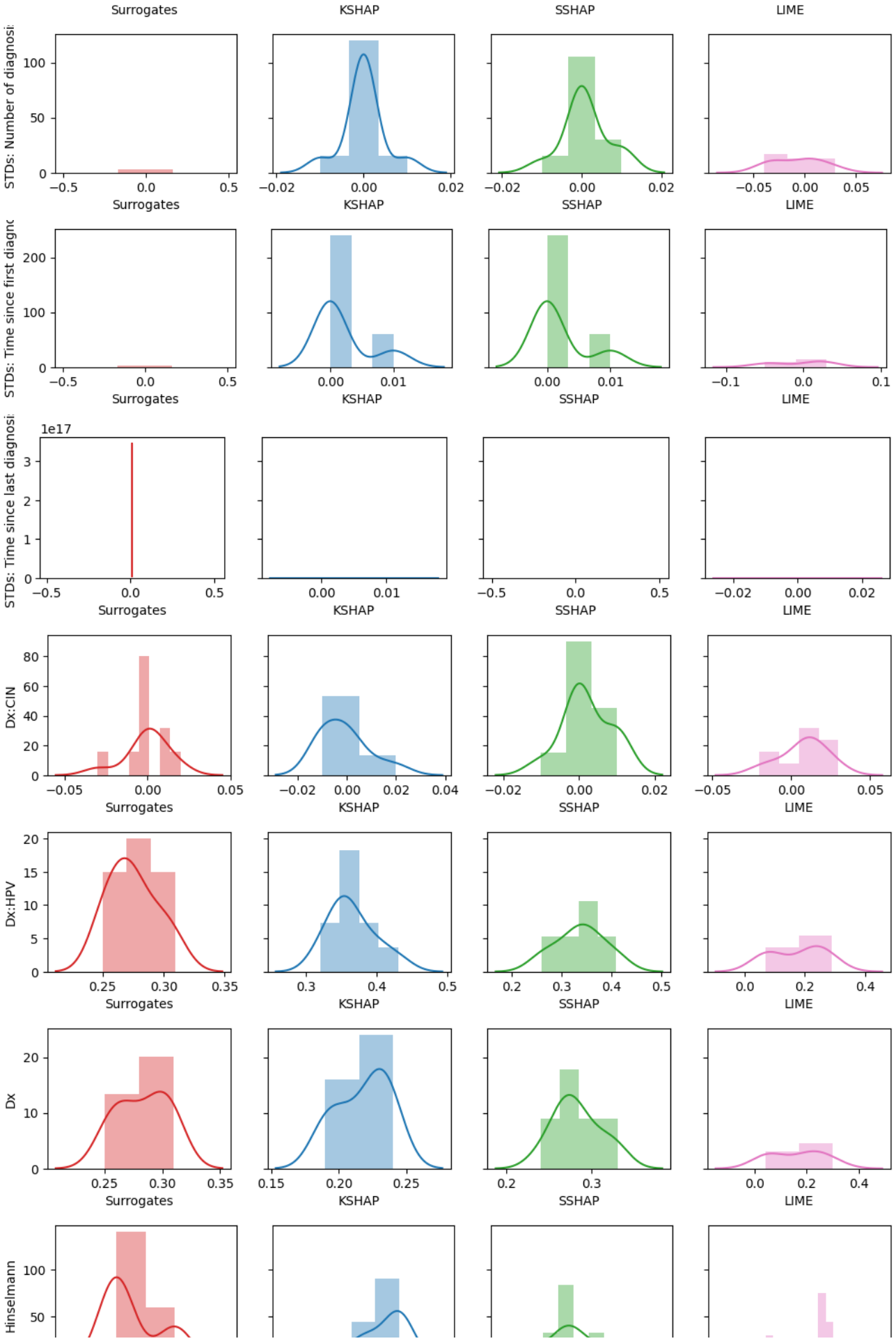
```
  #fs= compute_features_stability (case="classification", x=X_test, selection=lis
  fs= compute_features_stability (case="classification", x=X_test, selection=[ins
  frames.append(fs)

colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orar
for j in range(len(features)):
  fig, axes = plt.subplots(1, len(methods), figsize=(12, 2), sharey=True, dpi=100
  t=0

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

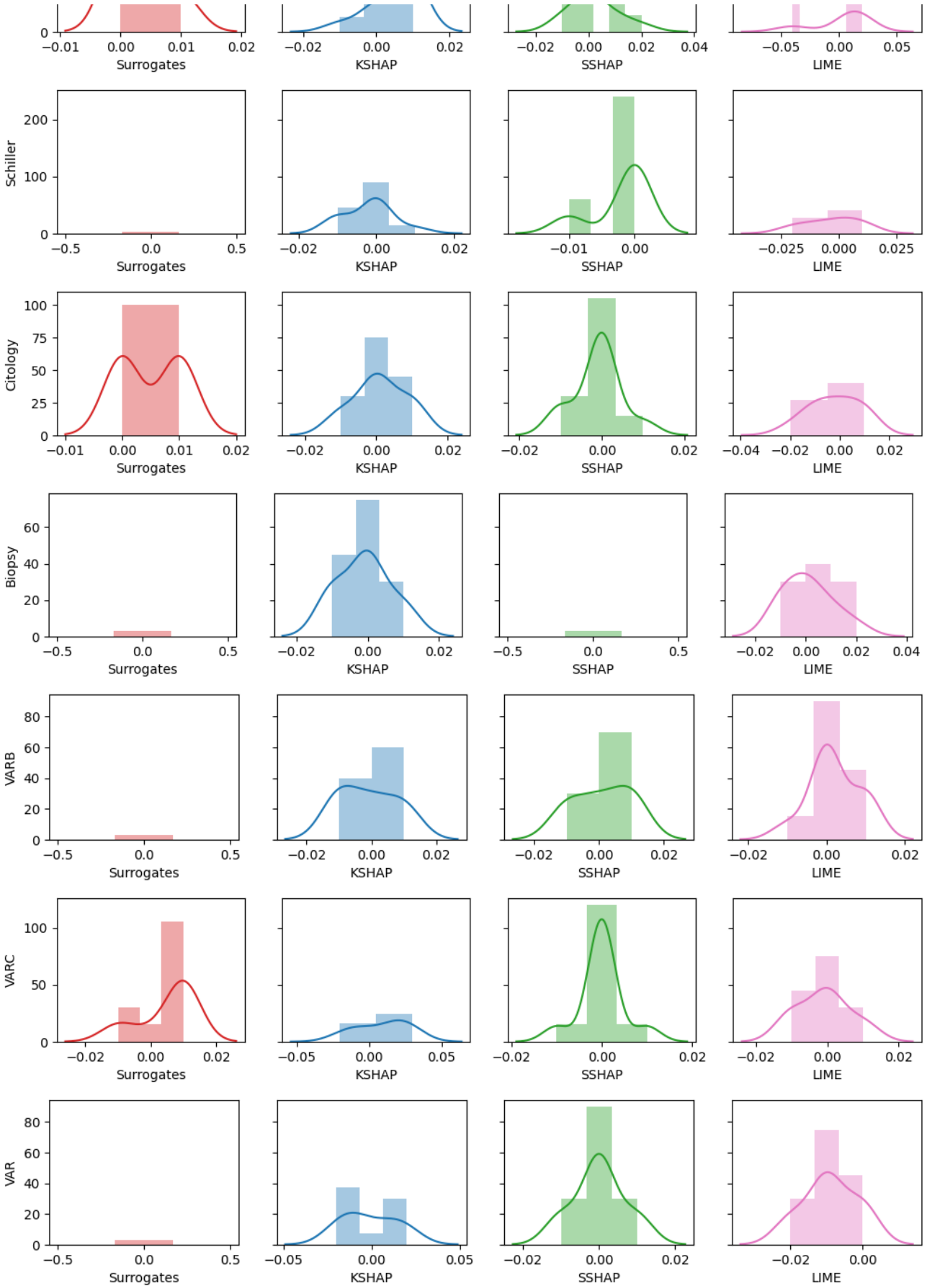



Smoke
Surrogates
KSHAP
SSHAP
LIME
Smokes (years)
Smokes (packs/year)
Hormonal Contraceptives
Hormonal Contraceptives (ye
IUD
IUD (years)


STDs
Surrogates
KSHAP
SSHAP
LIME
STDs (number)
STDs:condylomatosis
STDs:cervical condylomato
STDs:vaginal condylomato
STDs:vulvo-perineal condyloma
STDs:syphilis

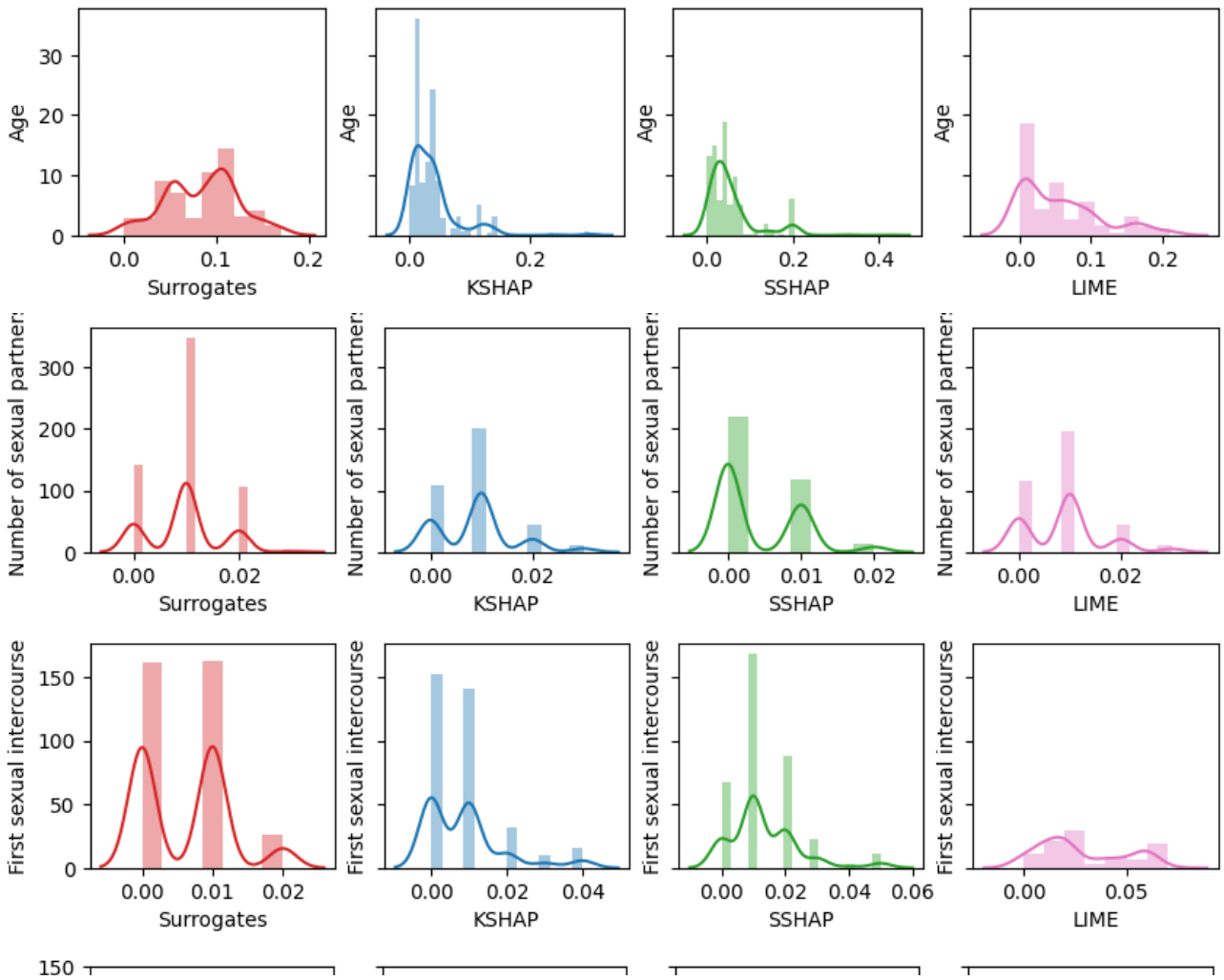

STDs:pelvic inflammatory dis
STDs:genital herpes
STDs:molluscum contagiosu
STDs:AIDS
STDs:HIV
STDs:Hepatitis B
STDs:HPV
Surrogates
KSHAP
SSHAP
LIME

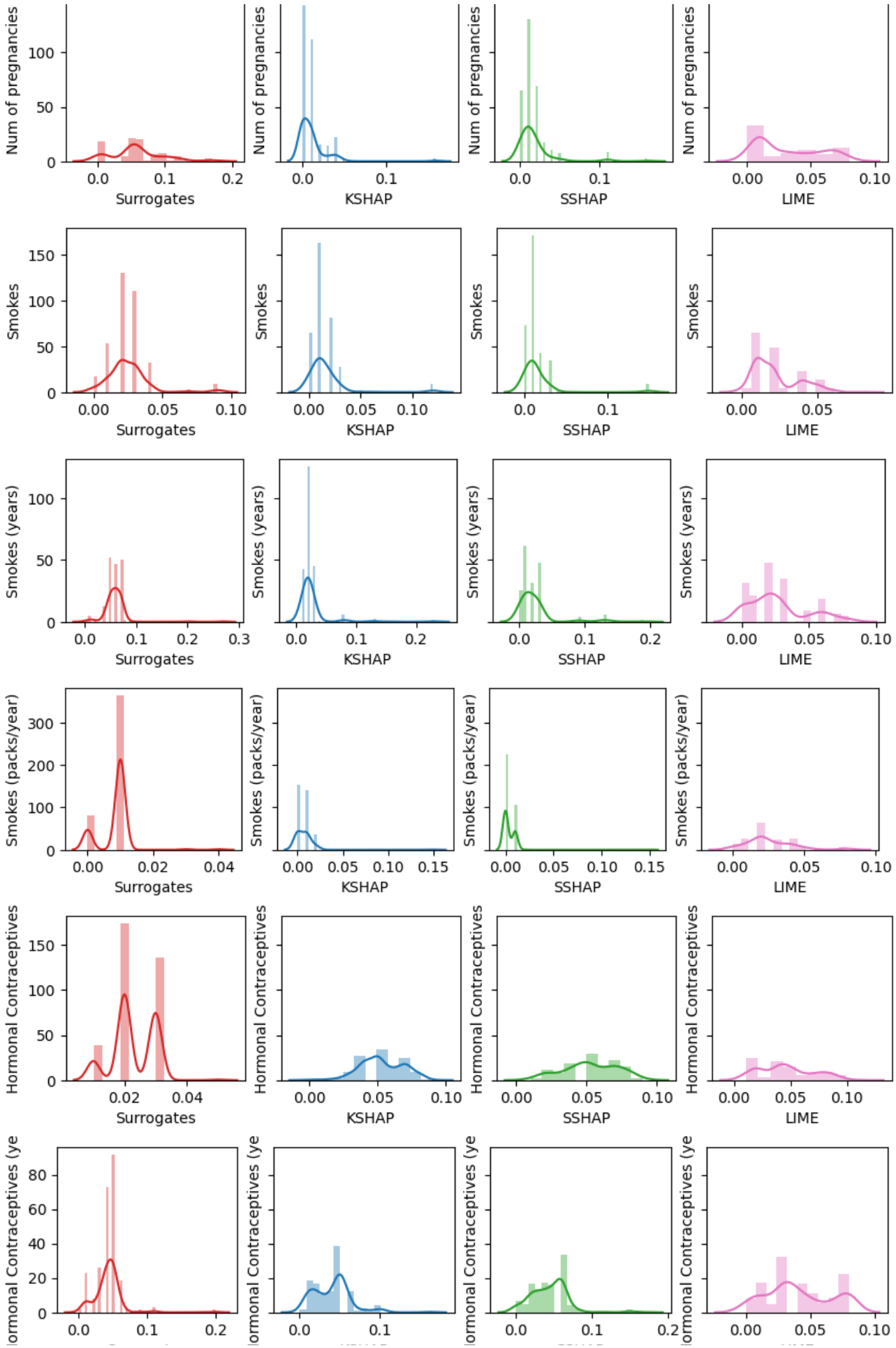

Surrogates
KSHAP
SSHAP
LIME
STDs: Number of diagnosi
STDs: Time since first diagnc
STDs: Time since last diagnosi
1e17
Dx:CIN
Dx:HPV
Dx
Hinselmann

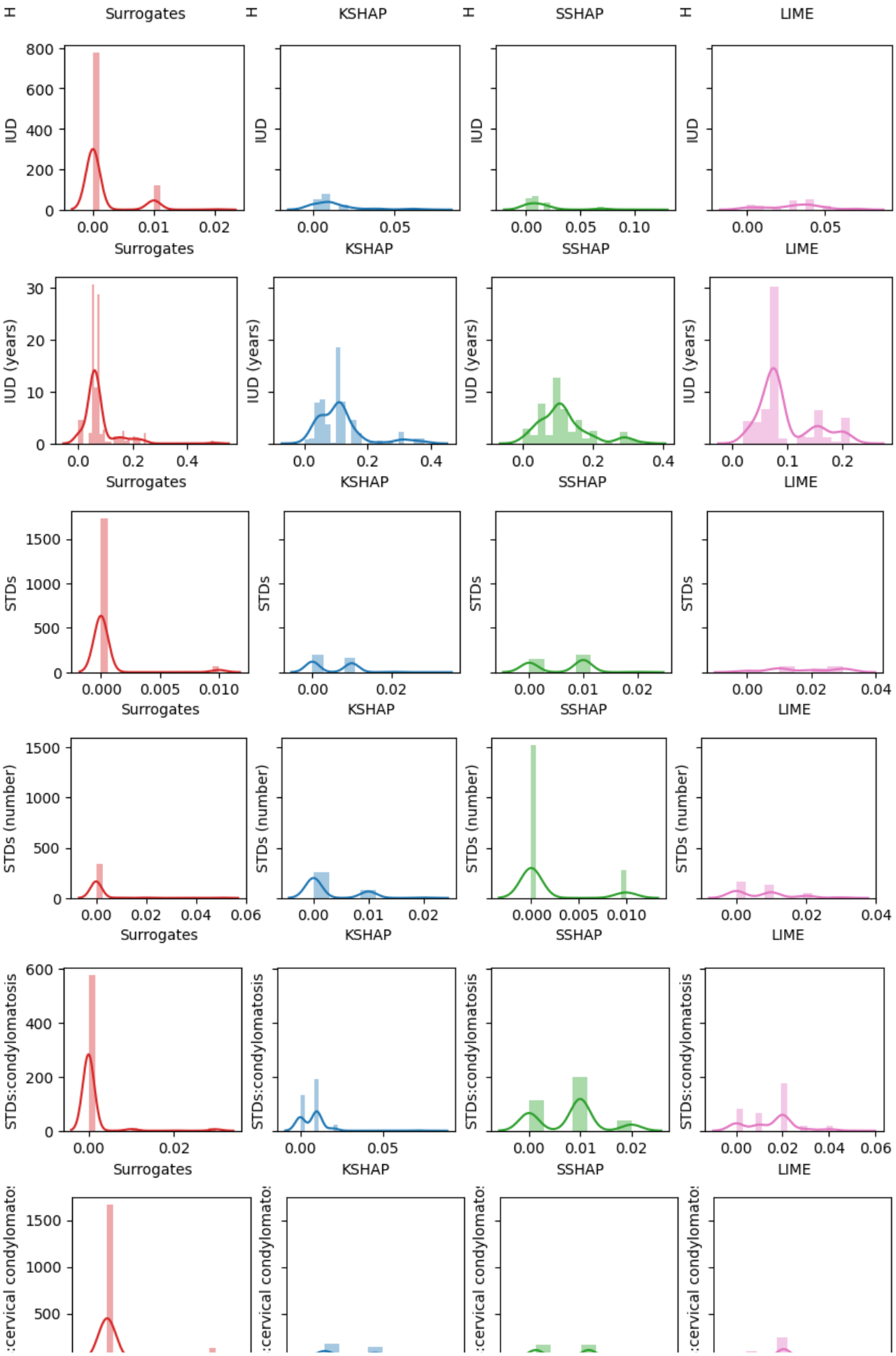

Surrogates
KSHAP
SSHAP
LIME
Schiller
Citology
Biopsy
VARB
VARC
VAR

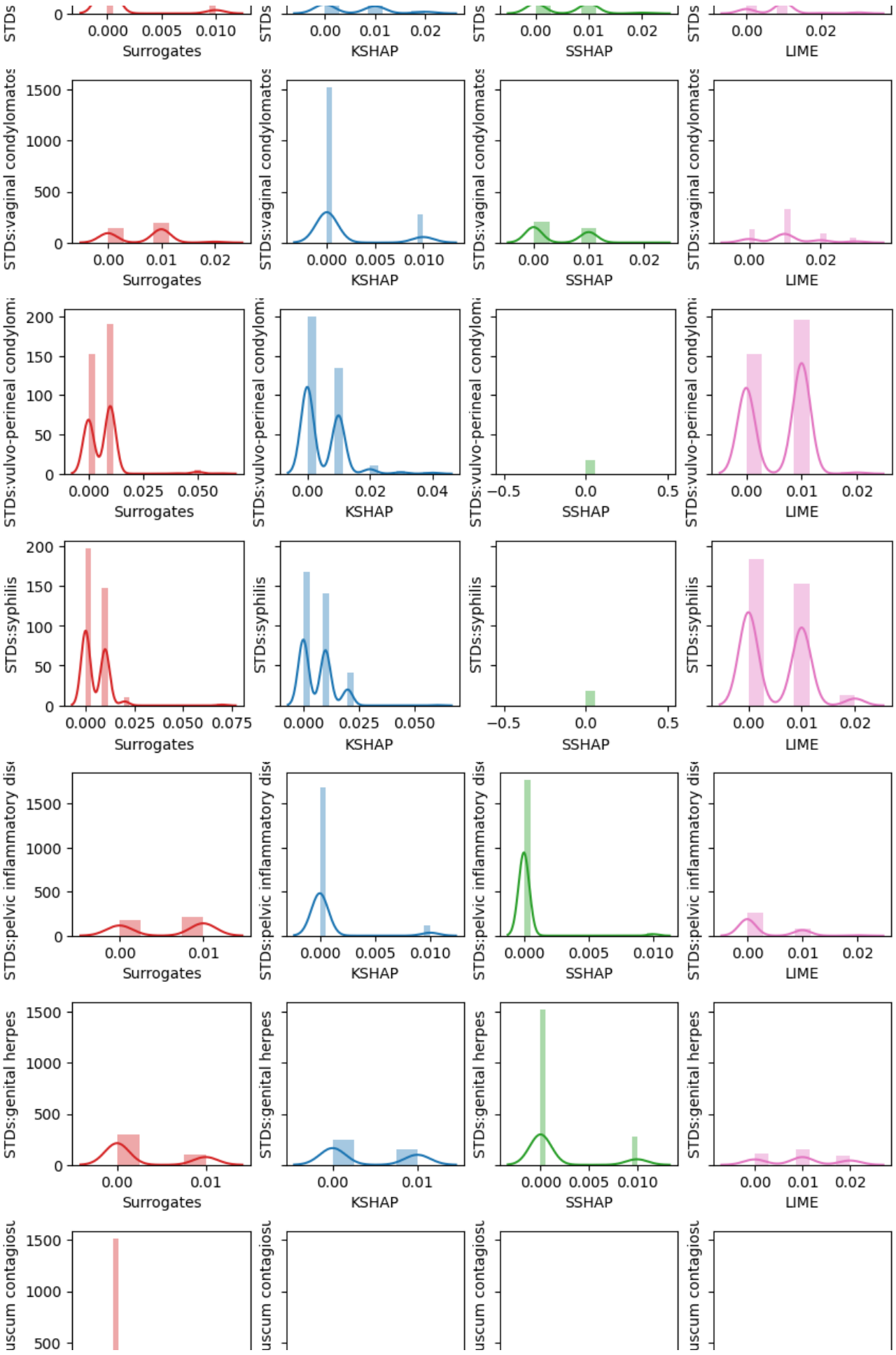
multiple instance

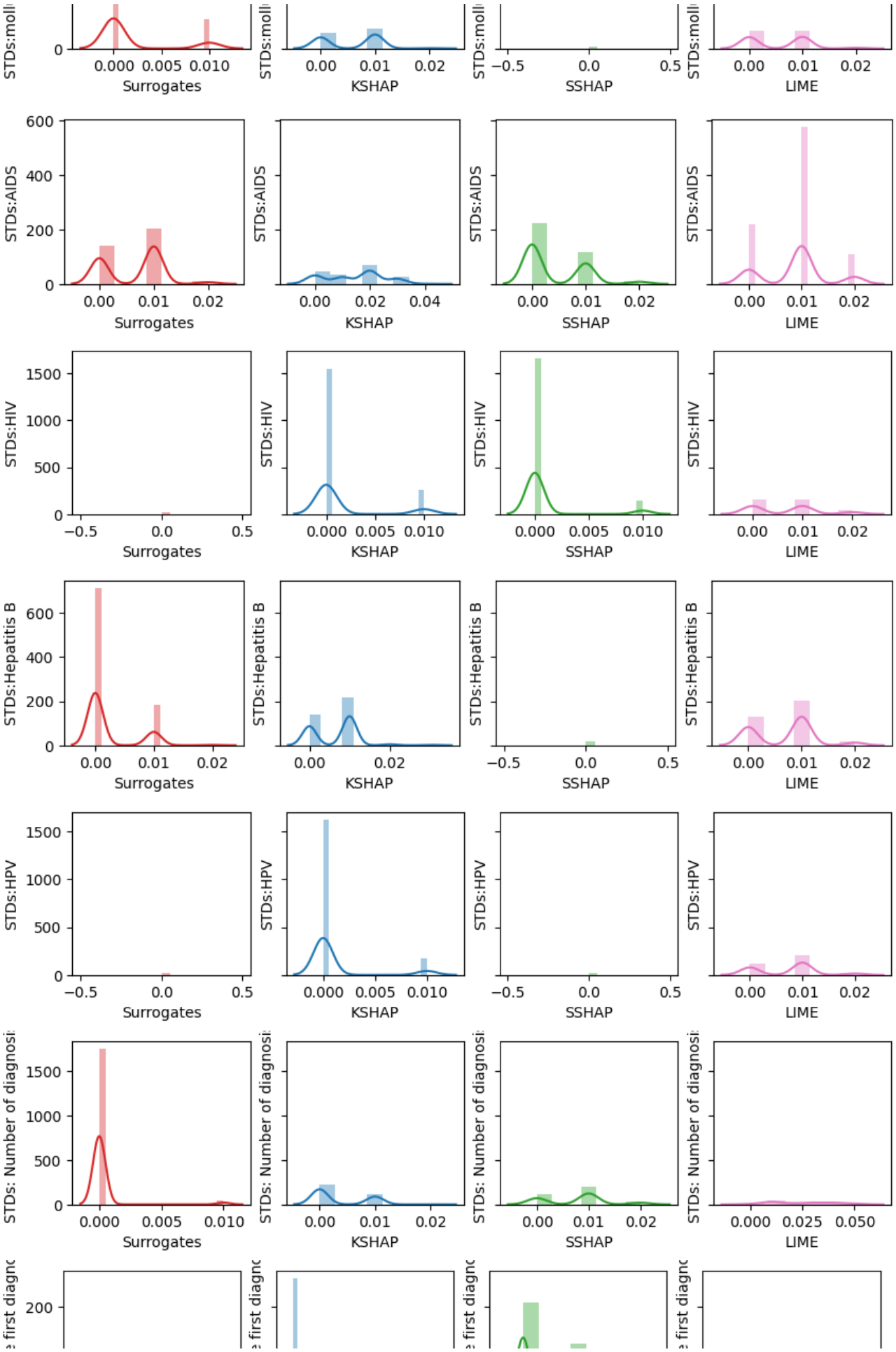
```
frames=[]
for weight in weights:
  fs= compute_features_stability (case="classification", x=X_test, selection=list
  #fs= compute_features_stability (case="classification", x=X_test, selection=[1,
  frames.append(fs)

colors = ['tab:red', 'tab:blue', 'tab:green', 'tab:pink', 'tab:olive',  'tab:orar
for j in range(len(features)):
  fig, axes = plt.subplots(1, len(methods), figsize=(10, 2), sharey=True, dpi=100
  t=0
  for fg, fs in enumerate(frames):
    vr=[]
    am=[]
    for i in range(len(fs['variability'])):
      vr.append(round(fs['variability'][i][j], 2)) # i INSCTANCE j Feature
      am.append(round(fs['amplitude'][i][j], 2))
    axes[fg].set_ylabel(features[j])
    sns.distplot(am, ax=axes[fg], color=colors[t], axlabel=methods[t])
    t=t+1
    #print('VR', vr)
  # plt.savefig('/content/drive/My Drive/'+str(var)+str(features[j])[:10]+'2.png'
  plt.show()
```

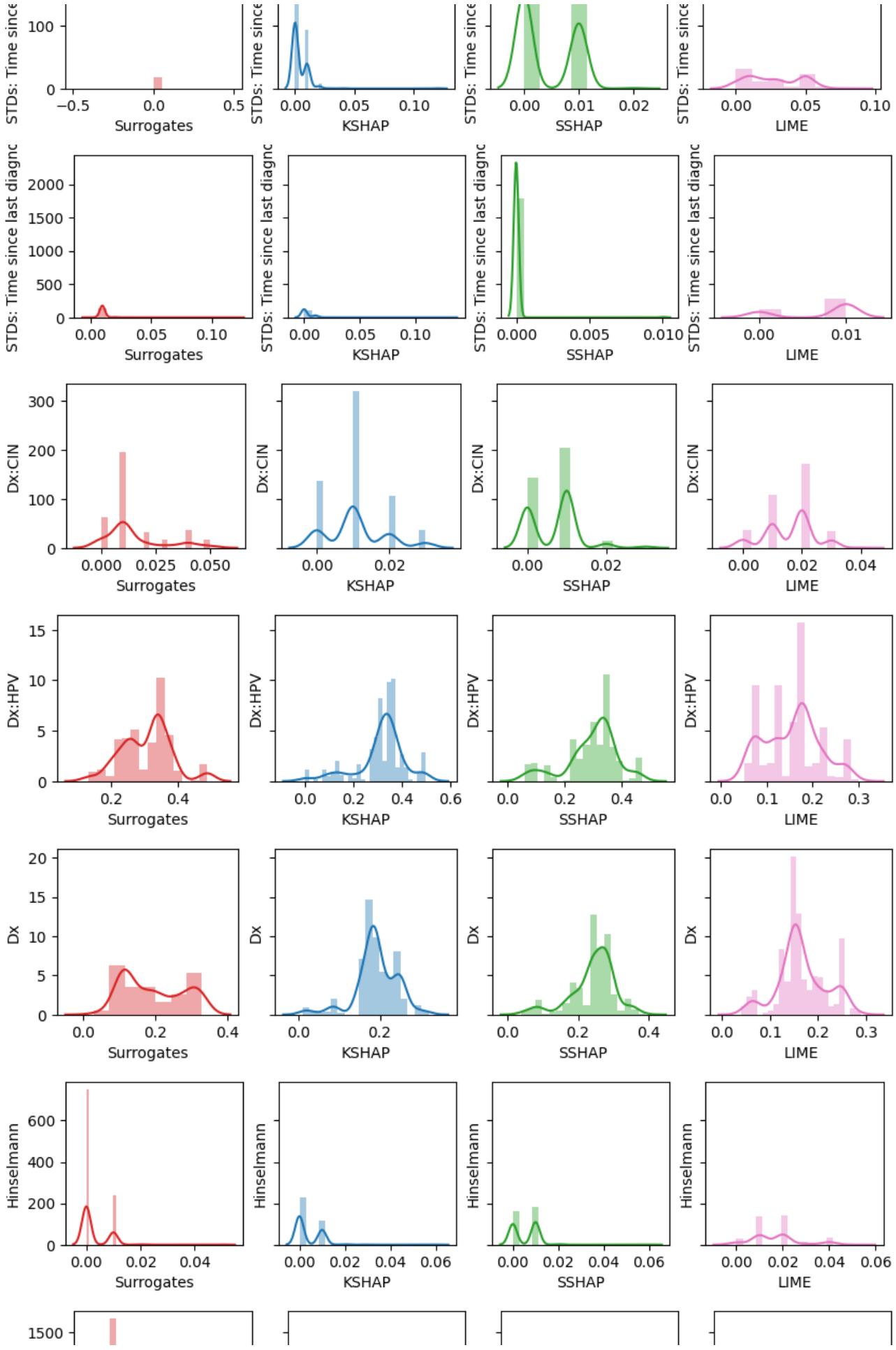


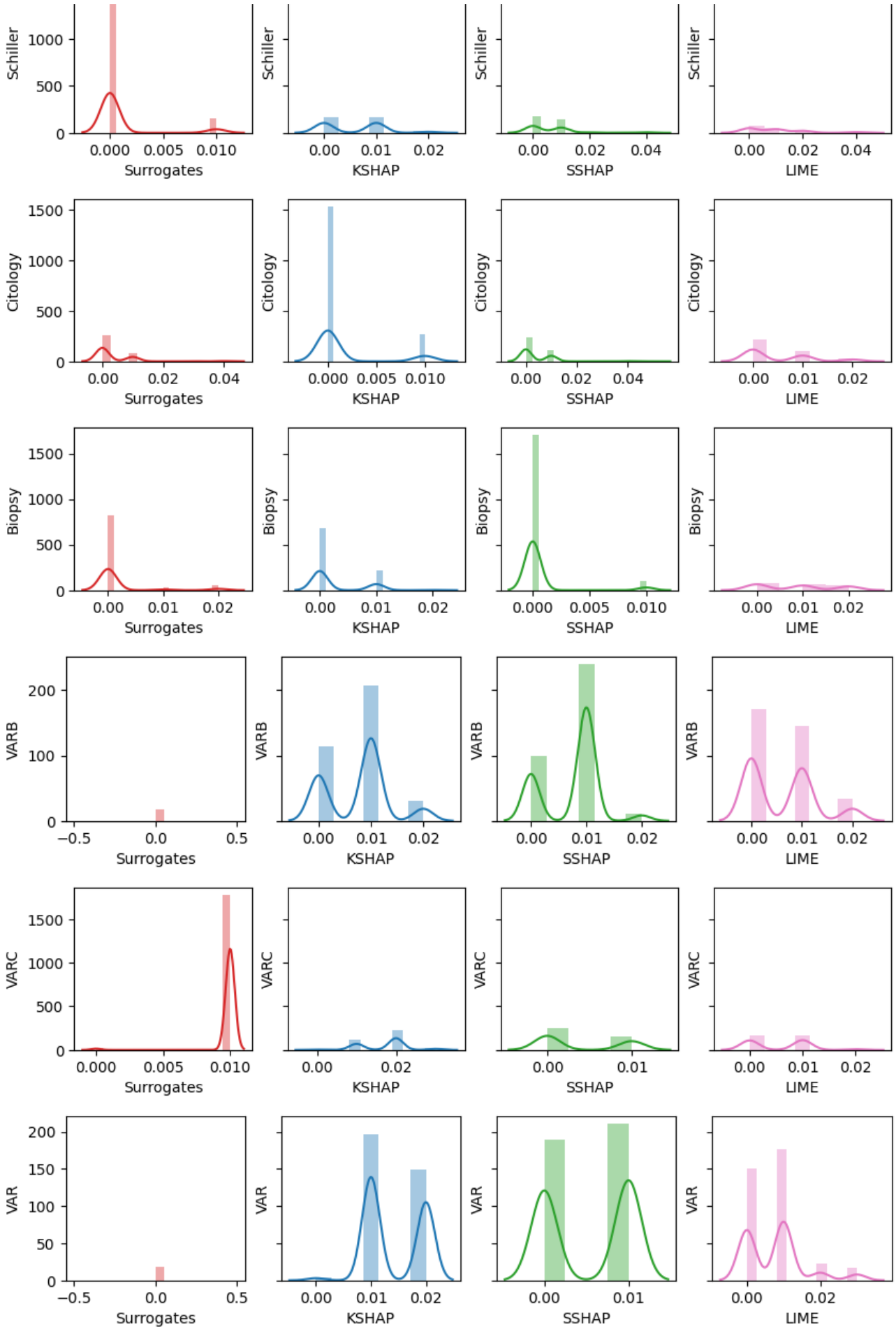

Num of pregnancies
Surrogates
KSHAP
SSHAP
LIME
Smokes
Smokes (years)
Smokes (packs/year)
Hormonal Contraceptives
Hormonal Contraceptives (ye


Surrogates
KSHAP
SSHAP
LIME
IUD
IUD (years)
STDs
STDs (number)
STDs:condylomatosis
:cervical condylomato:


STDs
STDs:vaginal condylomatos
STDs:vulvo-perineal condylomi
STDs:syphilis
STDs:pelvic inflammatory dise
STDs:genital herpes
uscum contagiosu
Surrogates
KSHAP
SSHAP
LIME


STDs:moll
Surrogates
KSHAP
SSHAP
LIME
STDs:AIDS
STDs:HIV
STDs:Hepatitis B
STDs:HPV
STDs: Number of diagnosi
STDs: first diagnc


STDs: Time since
Surrogates
KSHAP
SSHAP
LIME
STDs: Time since last diagnc
Dx:CIN
Dx:HPV
Dx
Hinselmann


Schiller
Surrogates
KSHAP
SSHAP
LIME
Citology
Biopsy
VARB
VARC
VAR

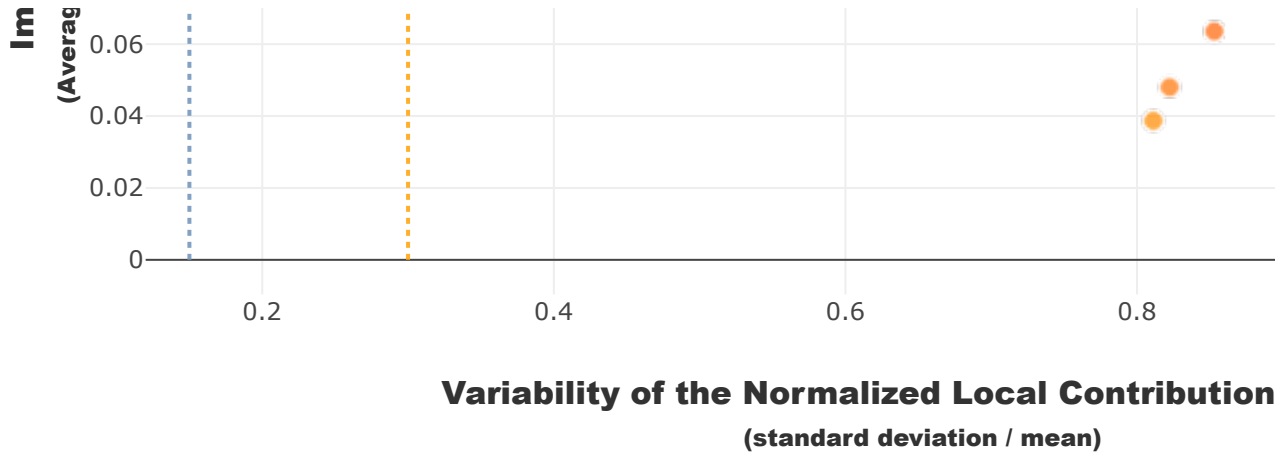
```
t=0
for fg, fs in enumerate(frames):
  vr=[]
  am=[]
  for j in range(len(features)):
    vr.append(np.mean(fs['variability'][j])) # i INSCTANCE j Feature
    am.append(np.std(fs['variability'][j]))
  print(methods[t], round(np.mean(vr),2))
  print(methods[t], round(np.std(am),2))
  t+=1
```

```
Surrogates 0.92
Surrogates 0.03
KSHAP 1.1
KSHAP 0.03
SSHAP 1.12
SSHAP 0.03
LIME 1.12
LIME 0.04
```

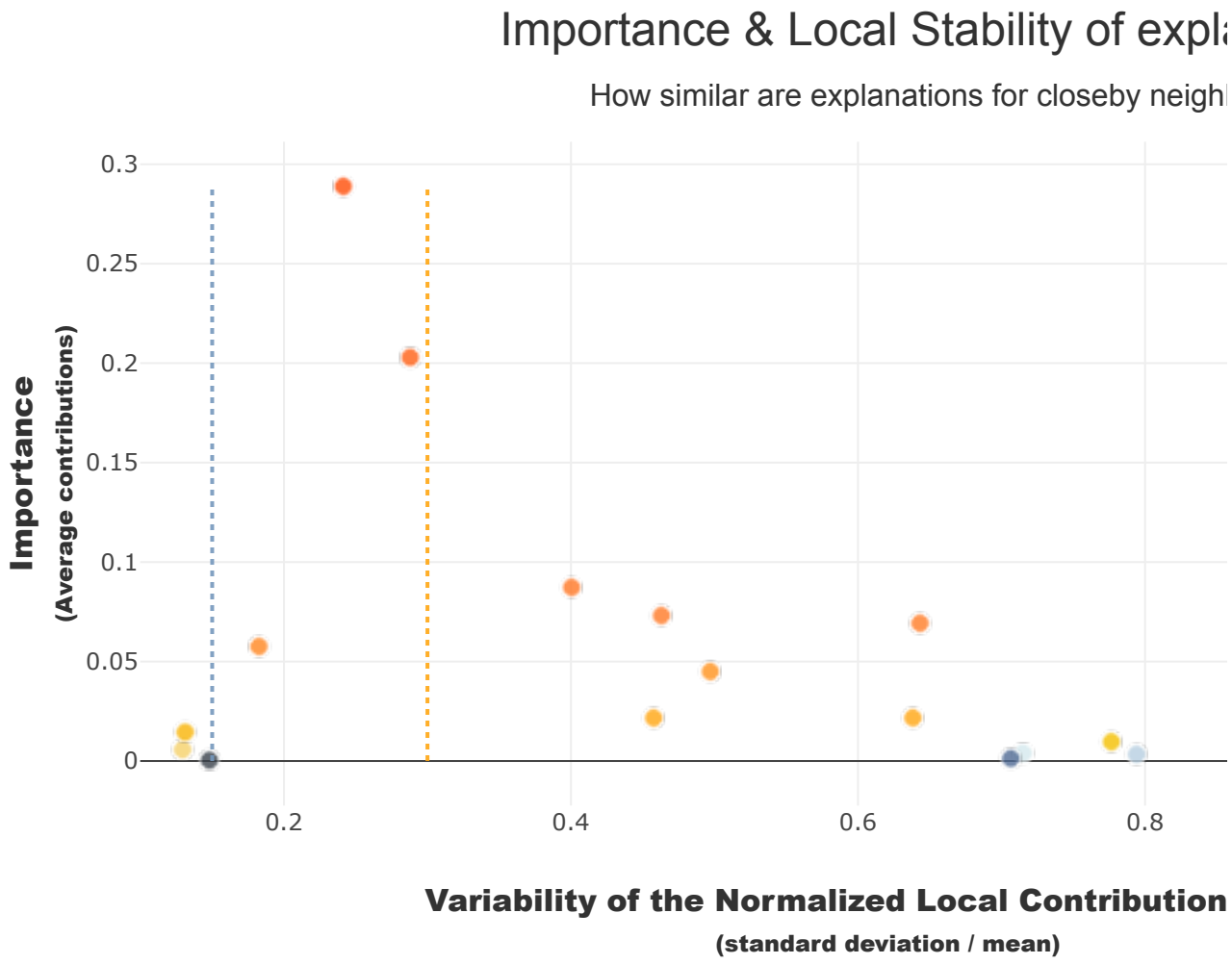
```
# xpl.plot.stability_plot(selection=[0, 1, 3])
fig_image=xpl.plot.stability_plot()
# plt.savefig('/content/drive/My Drive/stabplot.png')
```

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

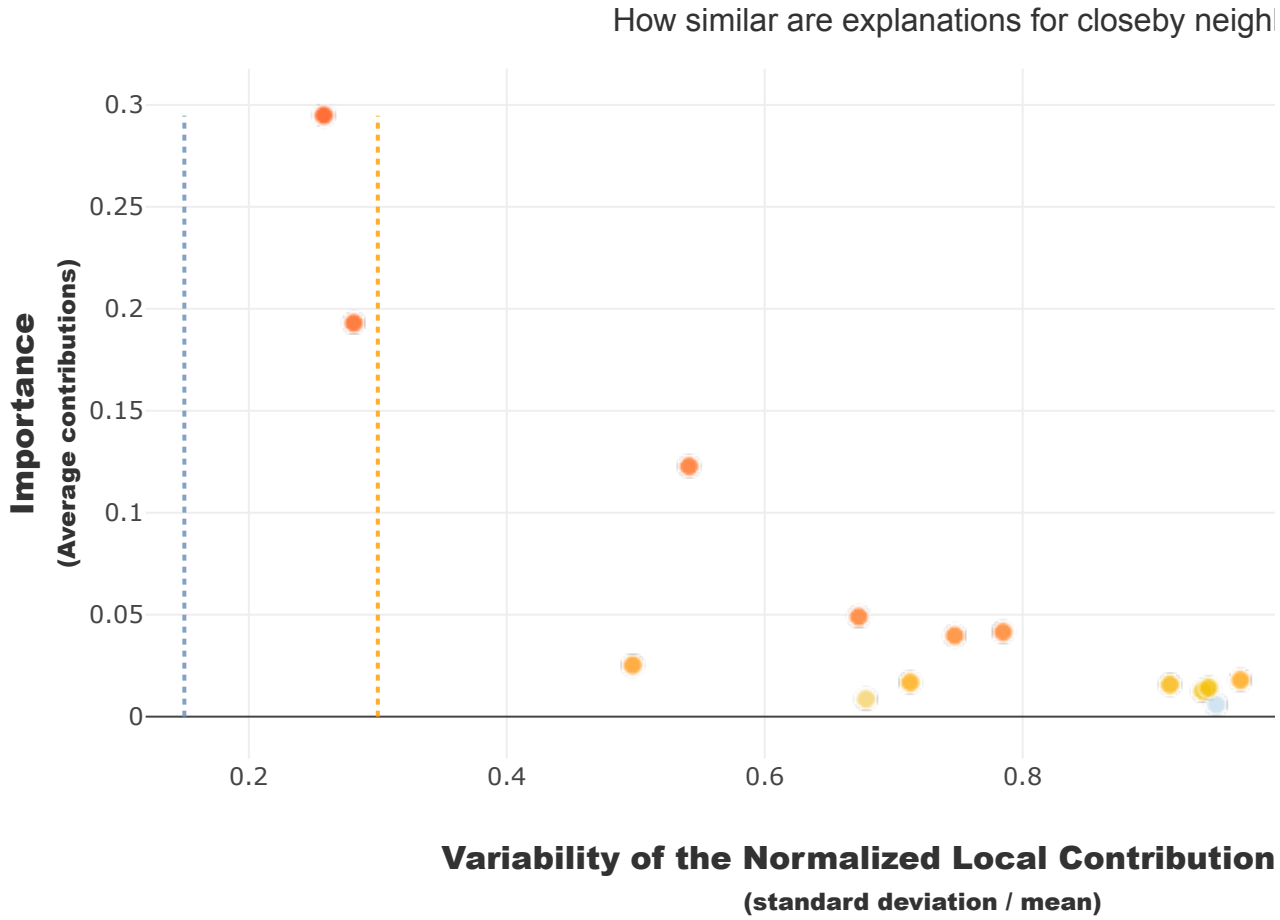
```
%matplotlib inline
fig_image.show()
```

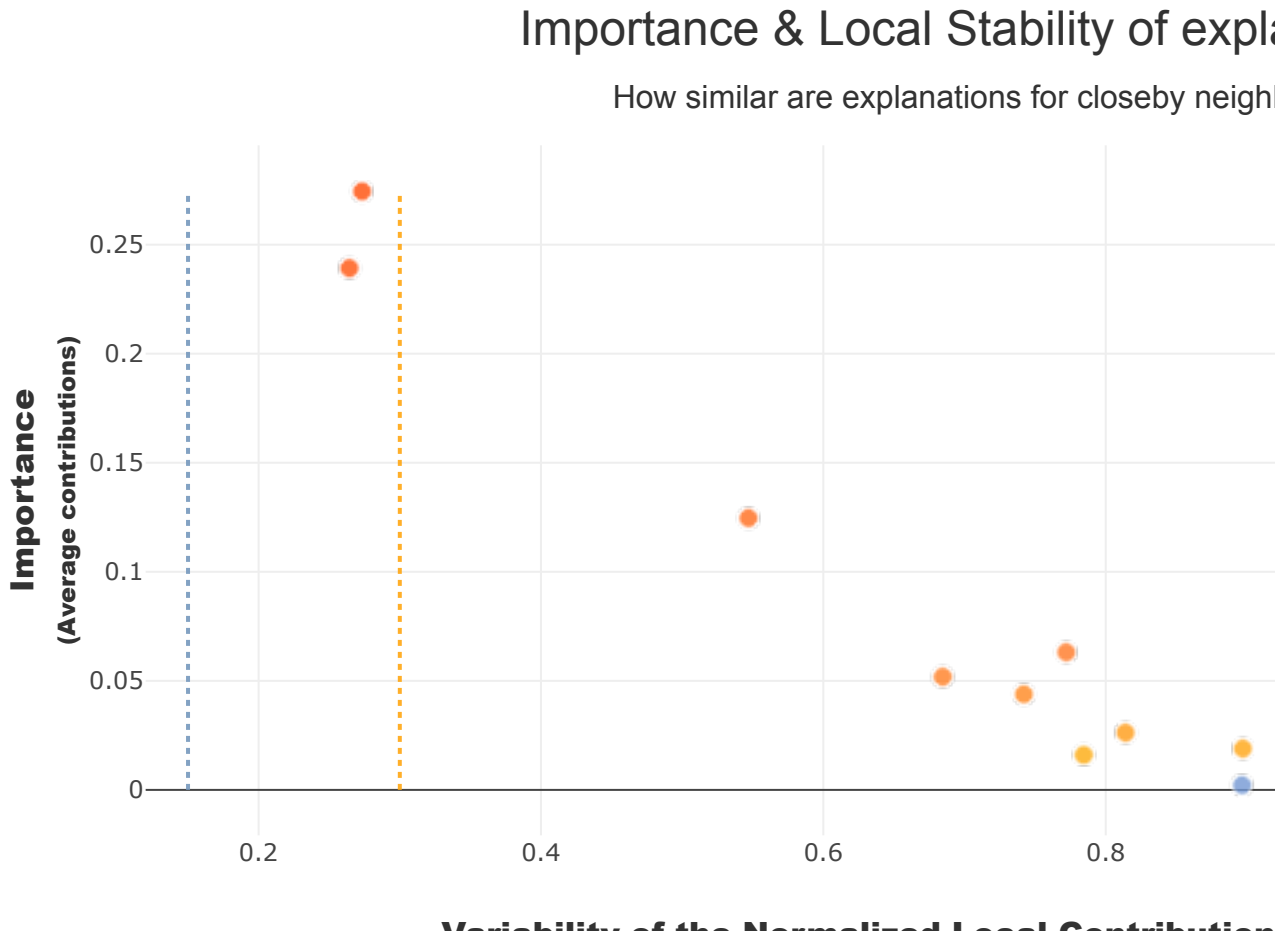


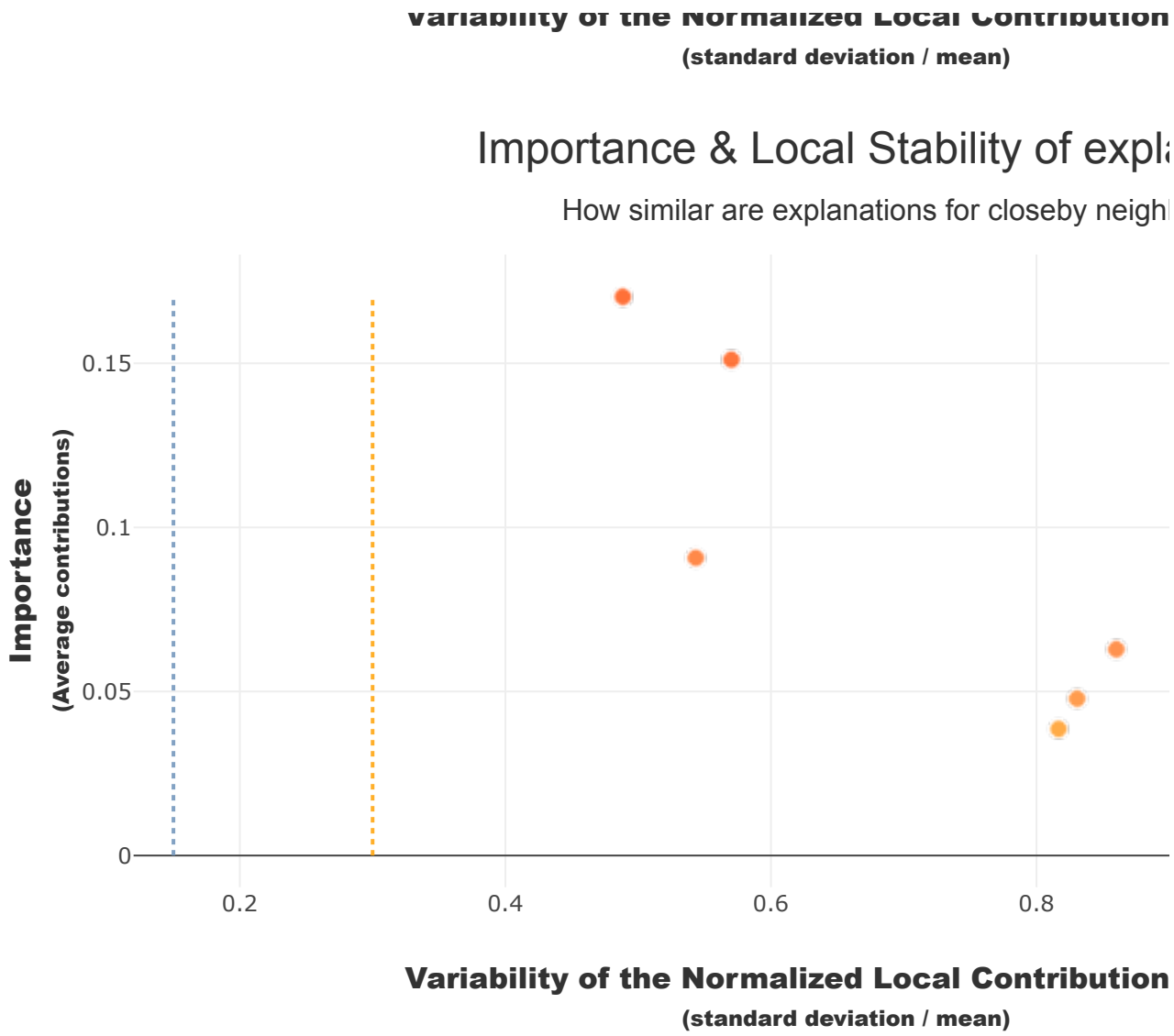

```
%matplotlib inline
for w in weights:
  xpl = SmartExplainer(model=model)
  xpl.compile(x=X_test,contributions=w)
  fig_image=xpl.plot.stability_plot()
  fig_image.show()
```



Importance & Local Stability of expl


How similar are explanations for closeby neigh
Importance
(Average contributions)
0.3
0.25
0.2
0.15
0.1
0.05
0
0.2
0.4
0.6
0.8
Variability of the Normalized Local Contribution
(standard deviation / mean)


Importance & Local Stability of expl
How similar are explanations for closeby neigh
Importance
(Average contributions)
0.25
0.2
0.15
0.1
0.05
0
0.2
0.4
0.6
0.8

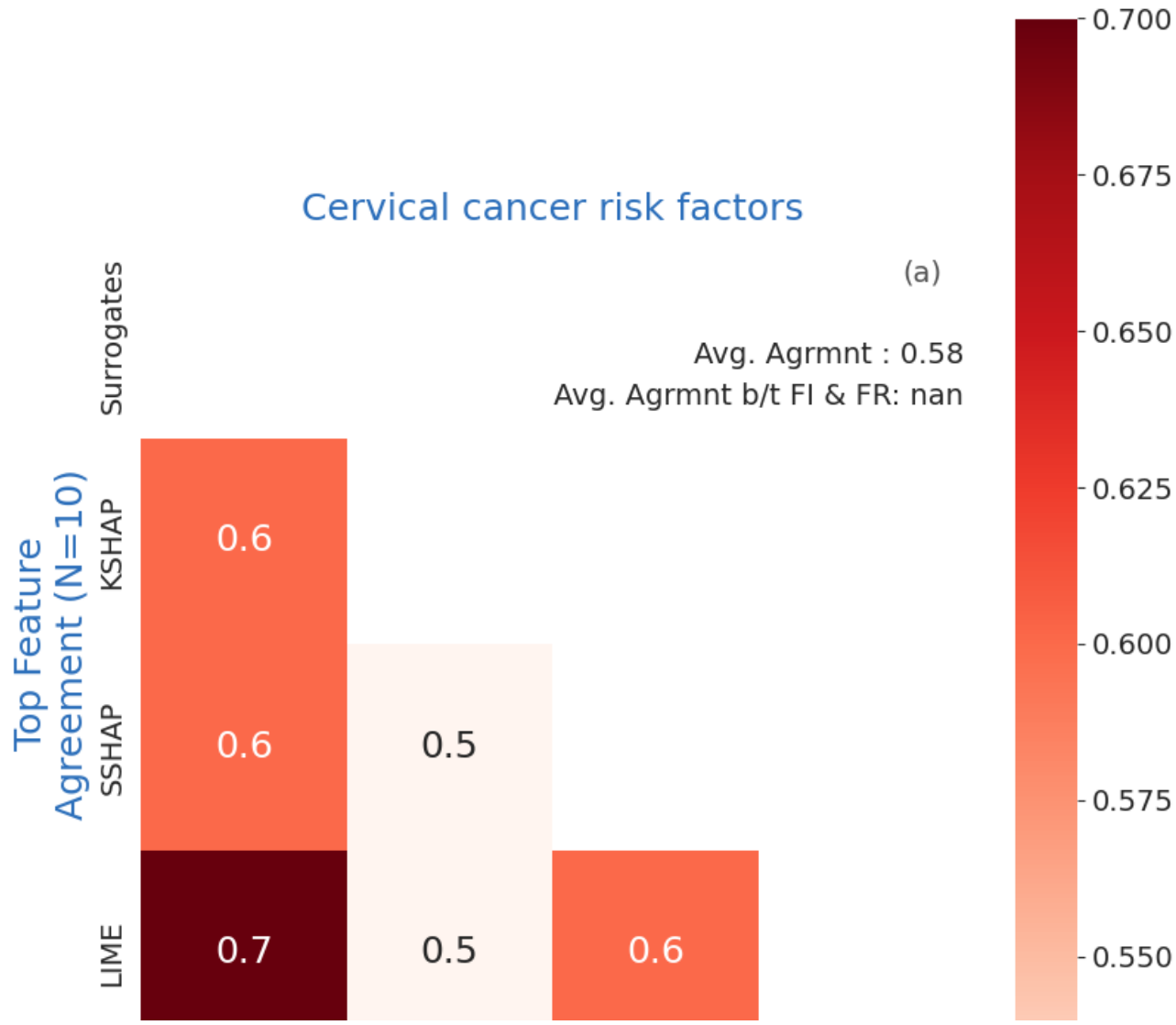


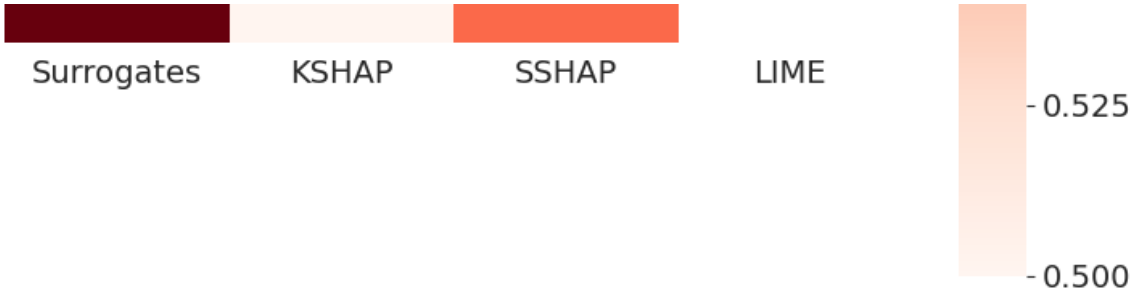
Feature and Rank disagreement

```
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):
    check_size(r1, r2)
    k = len(r1s)

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

def rank_agreement(r1, r2):
    check_size(r1, r2)
    k = len(r1)
```

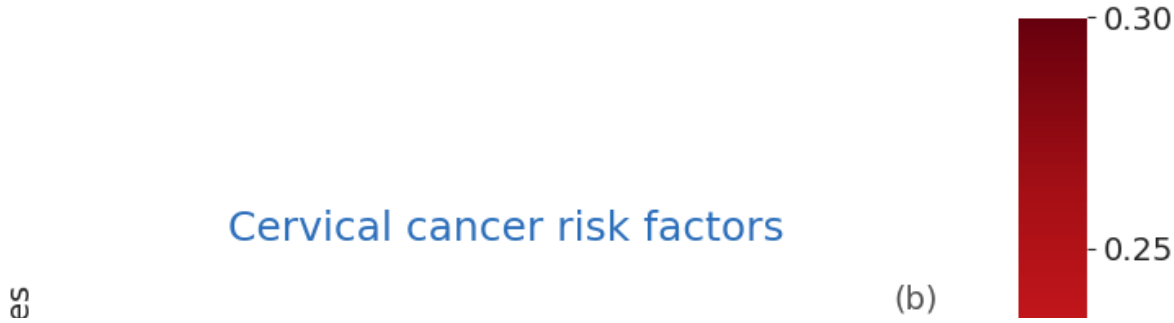
```
    return np.sum([True if x==y else False for x,y in zip(r1,r2)]) / k

def weak_rank_agreement(r1, r2):
    check_size(r1, r2)
    k = len(r1)
    window_size = 1 # Check if the rank is approximately close (within one rank).

    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):
    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

# compute feature agreement and rank agreement
feature_agree, rank_agree = compute_matrices(weights, instance)
```

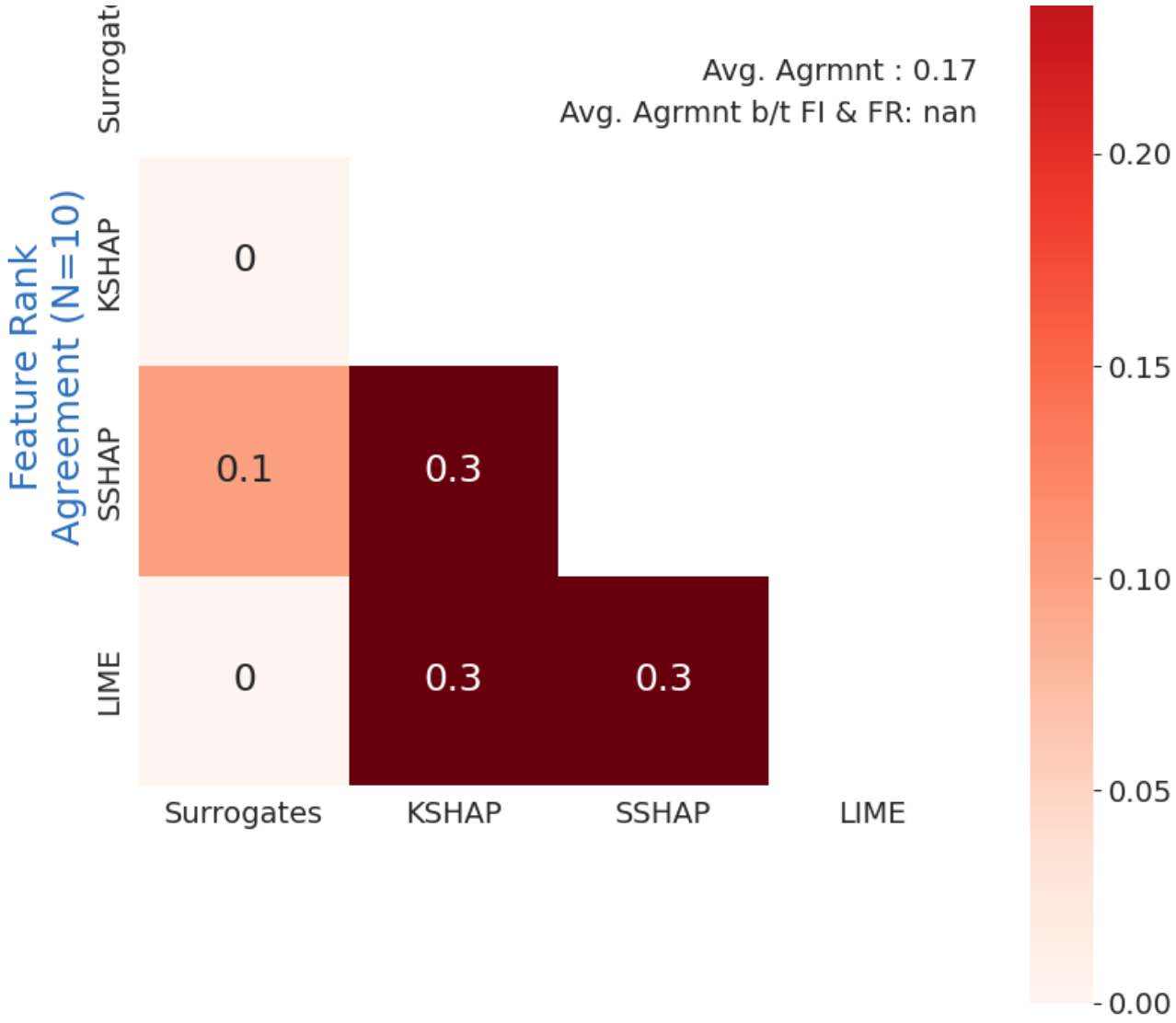
```
corr = feature_agree
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
fig = plt.figure(figsize=(9, 11))

with sns.axes_style("white"):
    ax = sns.heatmap(corr, mask=mask, square=True,annot=True, annot_kws={'fontsiz
    ax.set_title("Cervical cancer risk factors", color='xkcd:medium blue', fontsi
    ax.set_ylabel('Top Feature\nAgreement (N=10)', color='xkcd:medium blue', font
    ax.text(0.95, 0.95, f"(a)", fontsize=14, alpha=0.8, ha="center", va="center",

    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='rig

    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='rig
    plt.show()
```






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


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={'fontsiz
    ax.set_title("Cervical cancer risk factors", color='xkcd:medium blue', fontsi
    ax.set_ylabel('Feature Rank\nAgreement (N=10)', color='xkcd:medium blue',font
    ax.text(0.95, 0.95, f"(b)", fontsize=14, alpha=0.8, ha="center", va="center",

    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='rig

    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='rig
    plt.show()
```






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

Discussions

Implications of the experimental results

Most of the functionality in the original paper was reproducible. However, some of the specific explainability results we were unable to reproduce regardless which dataset preparation method

was used. We had difficulties on getting DICE on MLP, due to user config validation where there are feature names which are not part of columns in dataframe. TI and TSHAP methods are unable to run on the deep learning model because model type not yet supported by TreeExplainer / Base learner needs to be a DecisionTreeClassifier or DecisionTreeRegressor; yet these information was not documented in the original paper nor the code available.

“What was easy”

The parts around data processing and visualization were informative and easy to reproduce. Same with model training and evaluation. Among the local and global explainability that we were able to generate it was relatively straightforward to compare the methods and interpret the explainability results.

“What was difficult”

The most challenging part in producing the exact results based on the original paper's code provided. It was not documented what data produced which result, and what preparation work are used for the specific execution(with random noise, without noise, using continous or discret features, etc). It was also difficult to get all the local model explainations especially when it was not called out some of them only works for tree-based models.

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

There are three aspects that if the original authors have included in the code repo provided that would improve reproducibility. First is setting a seed on the notebook, and especially during the training period. They should also consider making the notebook clear on which block was the pre instead of commenting in and out code out a block without annotations on which preparation processing to use for which result, and which model to use for which explainability method. Lastly, they should freeze the libary versions used to aviod getting unexpected exceptions due to underlying version mismatch.

References

Main work:

Ayad, C. W., Bonnier, T., Bosch, B., Read, J., & Parbhoo, S. (2023). Which Explanation Makes Sense? A Critical Evaluation of Local Explanations for Assessing Cervical Cancer Risk Factors.

Local method reference:

Kelwin Fernandes, Jaime S Cardoso, and Jessica Fernandes. Transfer learning with partial

observability applied to cervical cancer screening. In Pattern Recognition and Image Analysis: 8th Iberian Conference, IbPRIA 2017, Faro, Portugal, June 20-23, 2017, Proceedings 8, pages 243–250. Springer, 2017.

Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A Benchmark for Interpretability Methods in Deep Neural Networks, June 2018