```
#### Project Info

Team:   94

Member: Qinxi Wang

Email:  qinxiw2@illinois.edu

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

Project Info

Team: 94

Member: Qinxi Wang

Email: qinxiw2@illinois.edu

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

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

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 -

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

img_dir = '/content/drive/MyDrive/features.png'
img_dir2 = '/content/drive/MyDrive/methods.png'

import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
img = cv2.imread(img_dir)
img2 = cv2.imread(img_dir2)

cv2_imshow(img)
cv2_imshow(img2)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

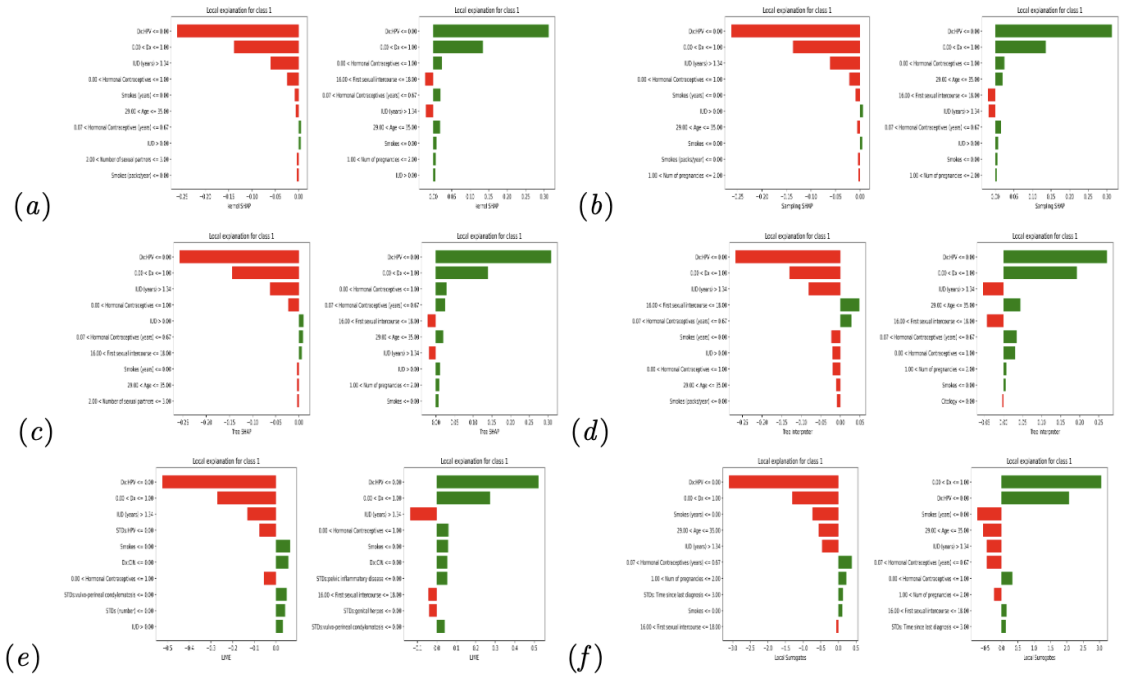


Figure 3: Comparison of feature importance for two similar patients (Patient 0 and Patient 1). On the left, Patient 0 diagnosed as not having cancer (Dx:Cancer=0) and on the right, Patient 1 diagnosed with cancer (Dx:Cancer=1). The key driving factors for this patient are similar to those of a patient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

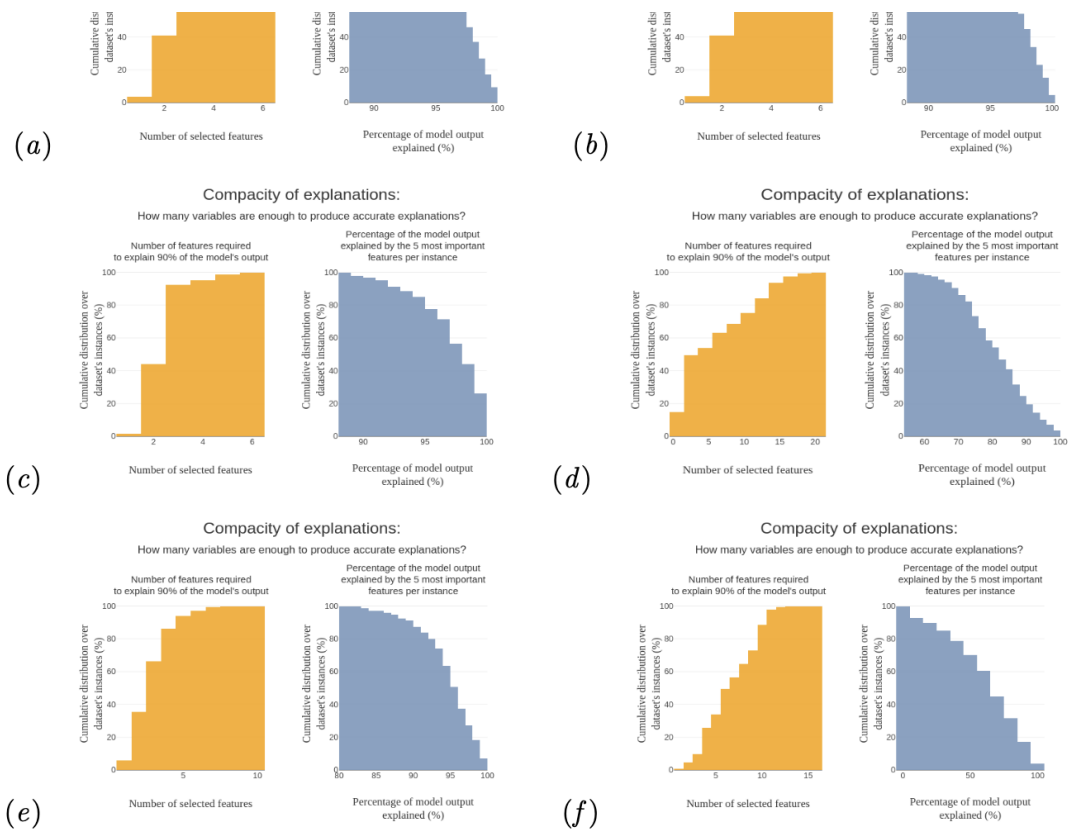


Figure 5: Compactness of the explanations generated by (a) Kernel SHAP, (b) Sampling SHAP, (c) Tree SHAP, (d) LIME,(e) Tree Interpreter, (f) Local surrogates.

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.

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

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

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

from plotly.offline import plot, iplot, init_notebook_mode

import warnings
warnings.filterwarnings('ignore')


# !pip install psutil -U kaleido
# import plotly.io as pio
```

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 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.

Statistics

Statistics

```
# Data notes
# dir and function to load raw data
# raw_data_dir = '/content/gdrive/My Drive/Colab Notebooks/<path-to-raw-data>'
risk_factor_df = pd.read_csv('/content/drive/My Drive/kag_risk_factors_cervical_c
risk_factor_df.head()

# def load_raw_data(raw_data_dir):
#   # implement this function to load raw data to dataframe/numpy array/tensor
#   return None

# raw_data = load_raw_data(raw_data_dir)

# 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
--------------------------------------------------------------------------------
```

```
# code comment is used as inline annotations for your coding
```

```
# code comment is used as inline annotations for your coding
# risk_factor_df = pd.read_csv('/content/drive/My Drive/kag_risk_factors_cervical
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

```
# 840 non HPV, 18 HPV

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

```
0    840
1     18
Name: Dx:HPV, 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: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 
```

```
 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

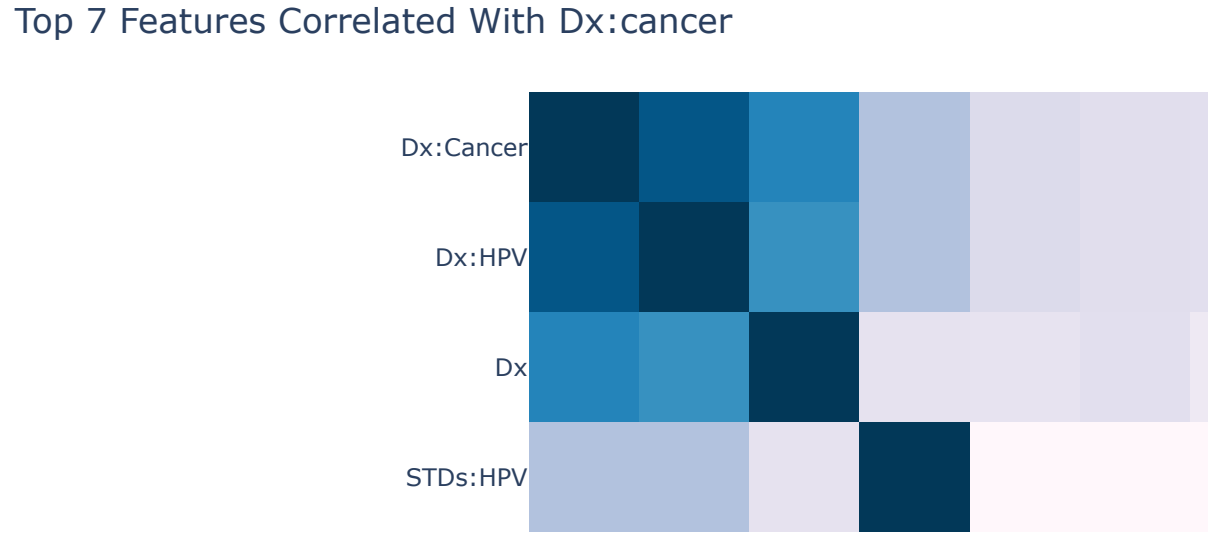
```
#these columns are not of type object, but are of type numeric
cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Num
                   'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contracepti
                   'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', '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']
# for i in range(0,len(cols_to_convert)):
#     print("{}={}".format(i,cols_to_convert[i]))
risk_factor_df[cols_to_convert] = risk_factor_df[cols_to_convert].apply(pd.to_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))
```

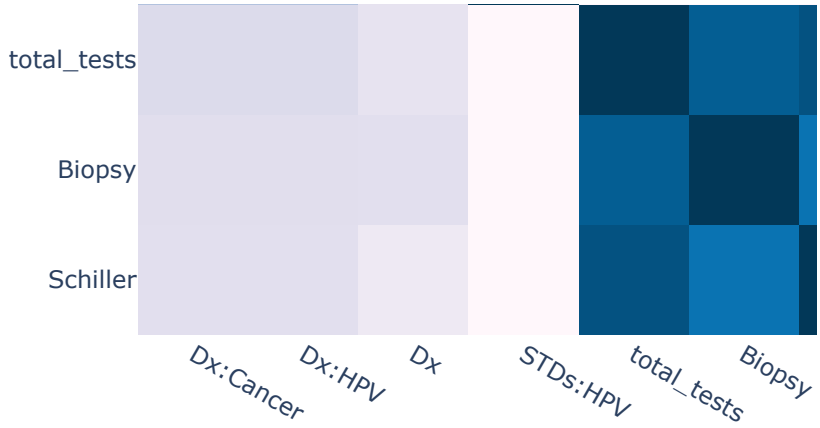
```
def age_cat(age):
    if age < 12:
        return "Child"
    elif age < 20:
        return "Teen"
    elif age < 30:
        return "20's"
    elif age < 40:
        return "30's"
    elif age < 50:
        return "40's"
    elif age < 60:
        return "50's"
    elif age < 70:
        return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

```
                        "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)

# risk_factor_df.info()
```

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()
```

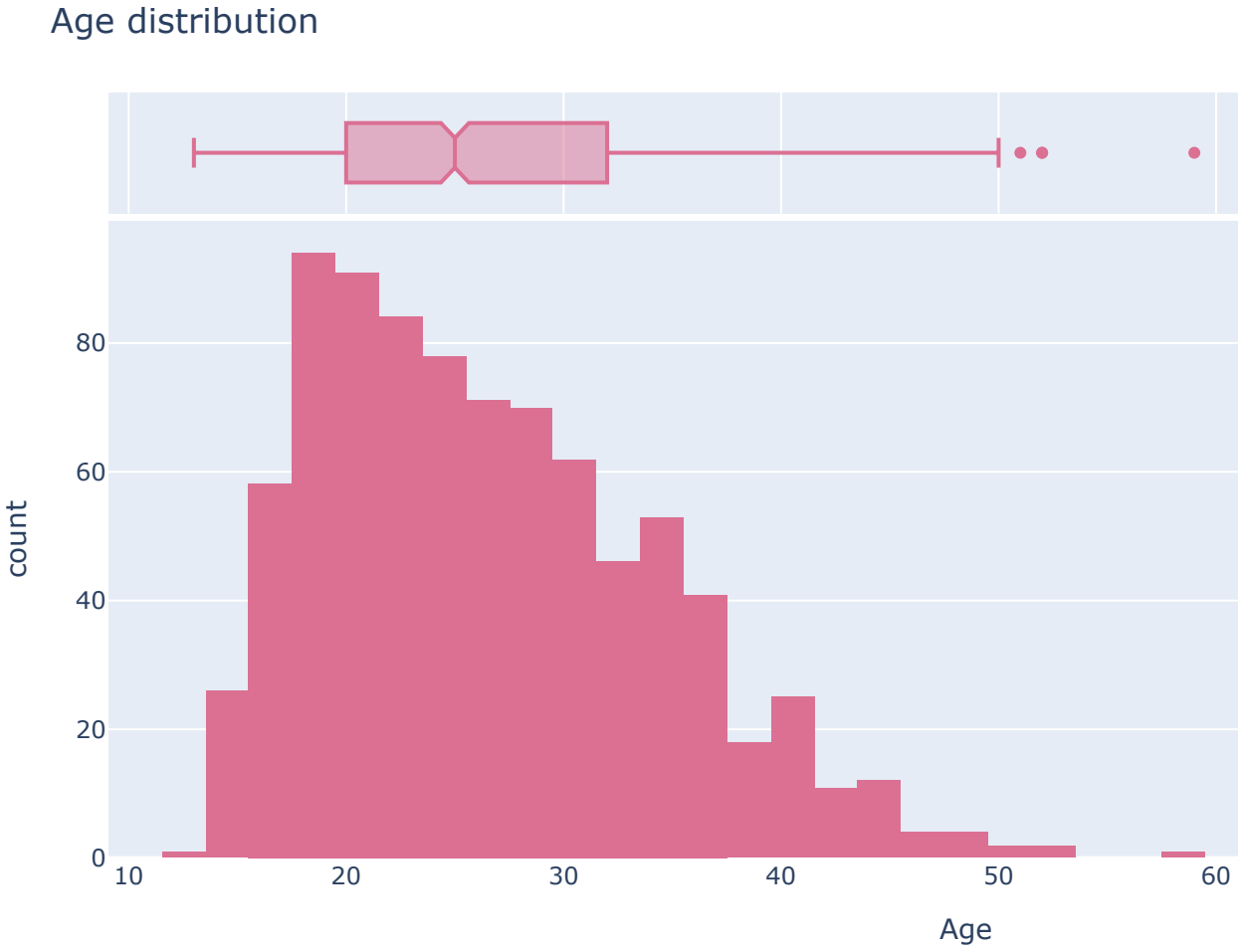




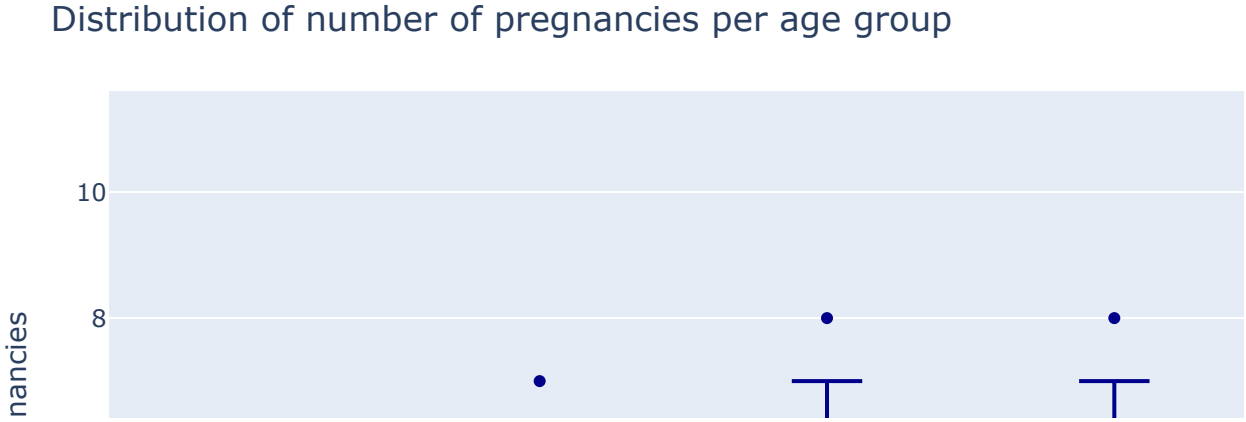

```
def stats(x):
    temp1=(df[[x,label]].value_counts(normalize=True).round(decimals=3)*100).reset_
    Coloumn_To_Aggregate=[x,label]
    df6=pd.merge(df.groupby(Coloumn_To_Aggregate).size().reset_index(name='ind_siz
                 df.groupby(Coloumn_To_Aggregate[:-1]).size().reset_index(name='To
    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')
```

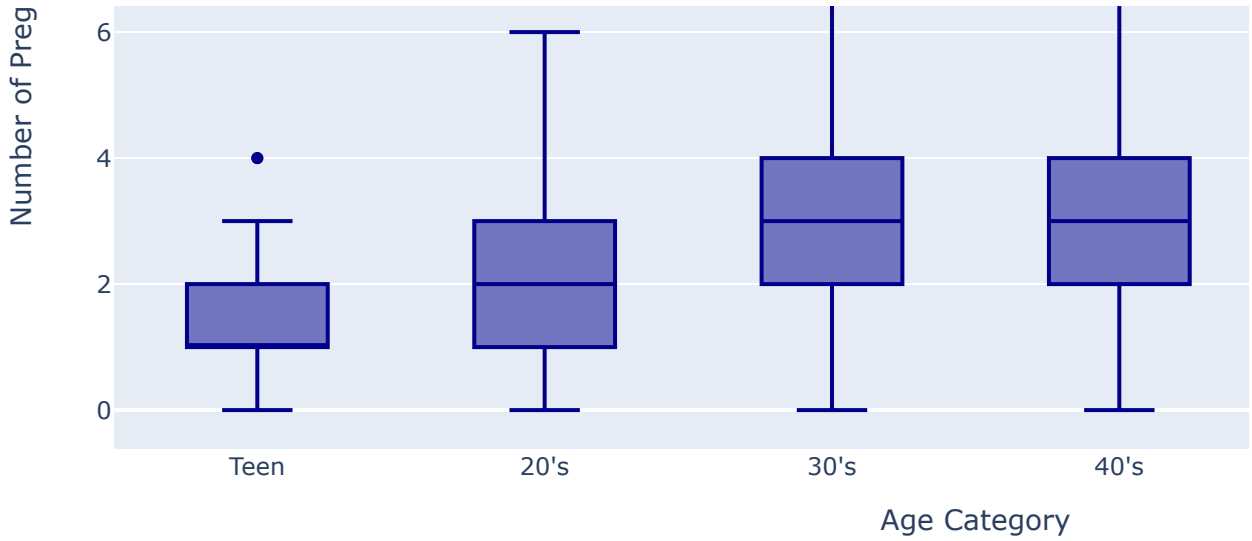
	Overall_Percent		Category_Percent	
Dx:Cancer	**0**	**1**	**0**	**1**
age_cat				
20's	45.3	0.6	46.31	27.78
30's	24.7	0.9	25.24	44.44
40's	6.2	0.3	6.31	16.67
50's	0.5	0.1	0.48	5.56
70+	0.5	NaN	0.48	NaN
Teen	20.7	0.1	21.19	5.56

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

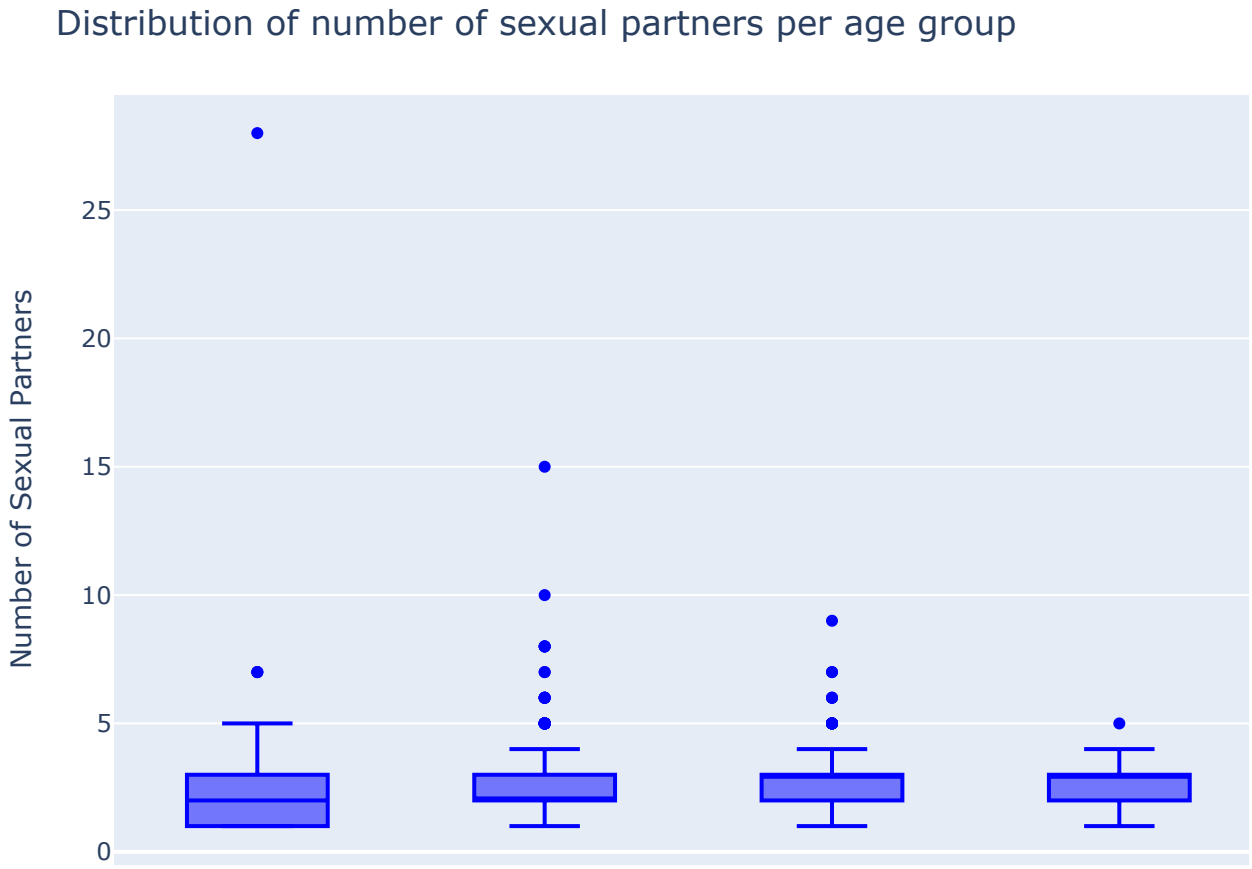



```
age_preg_bar = px.box(risk_factor_df.sort_values(by="Age",ascending=True), x="age
                      color_discrete_sequence=["darkblue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forti
                                       "Seventy and over"])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

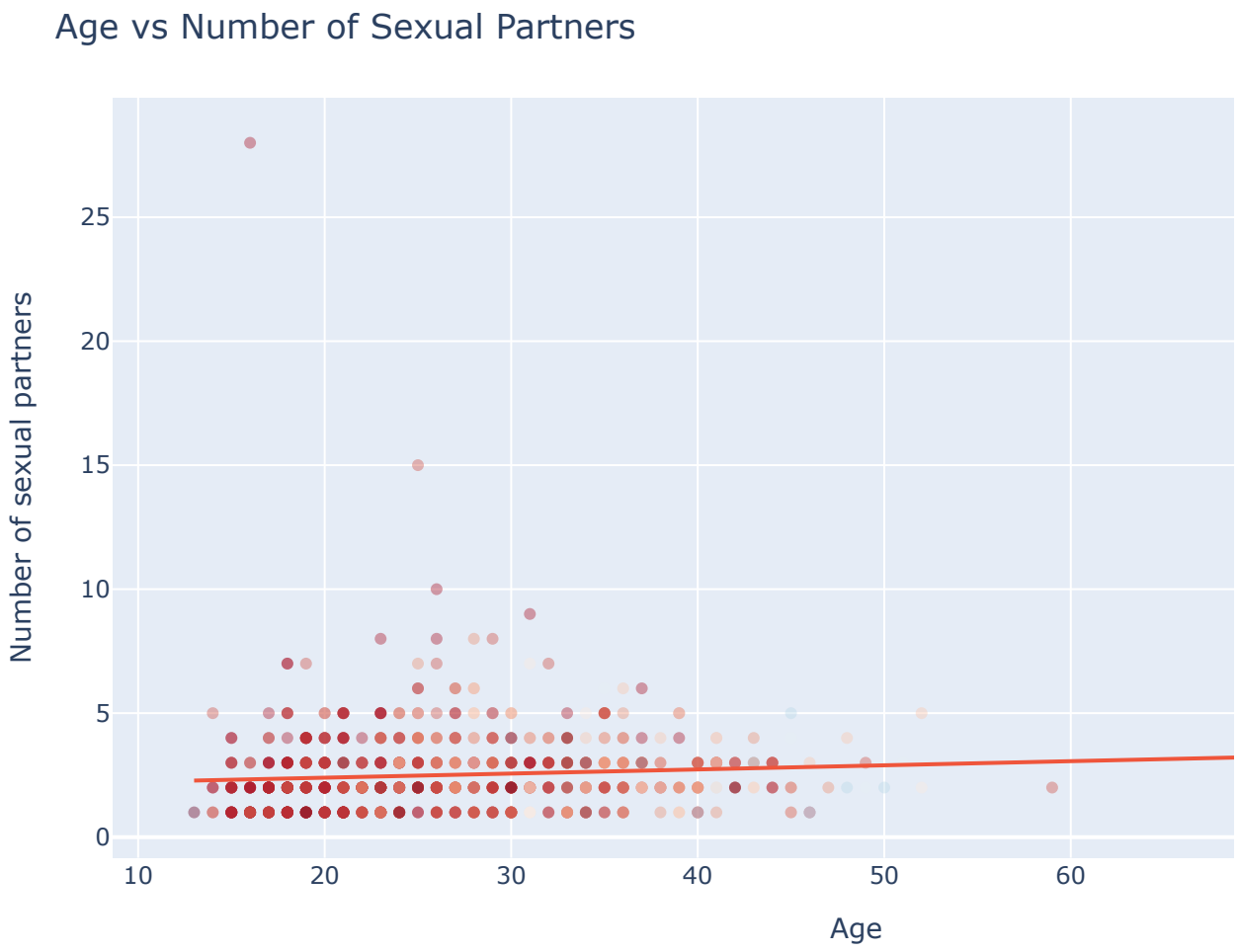





```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True)
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forti
                                       "Seventy and over"])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

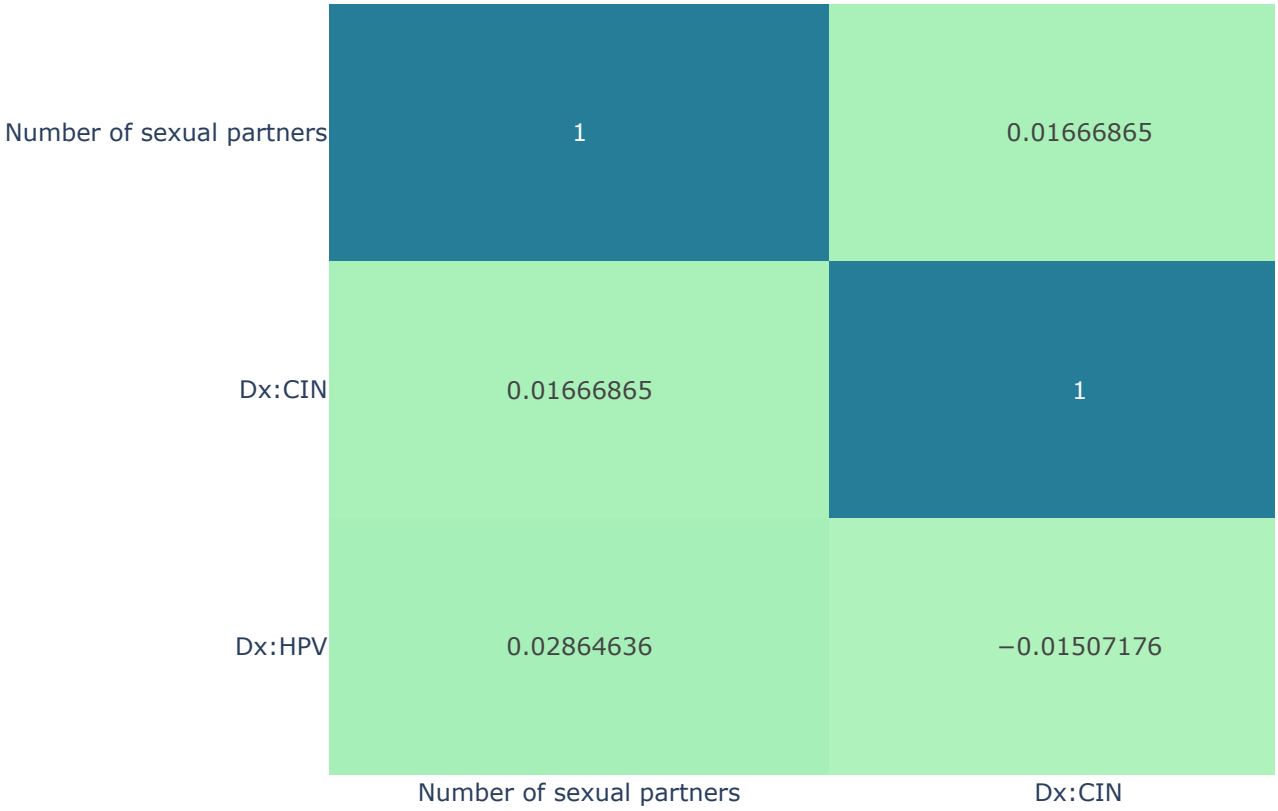



```
age_num_sex_partners = px.scatter(risk_factor_df, x="Age",
                                  y="Number of sexual partners",
                                  trendline="ols",
                                  opacity=0.4,
                                  color="Num of pregnancies",
            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

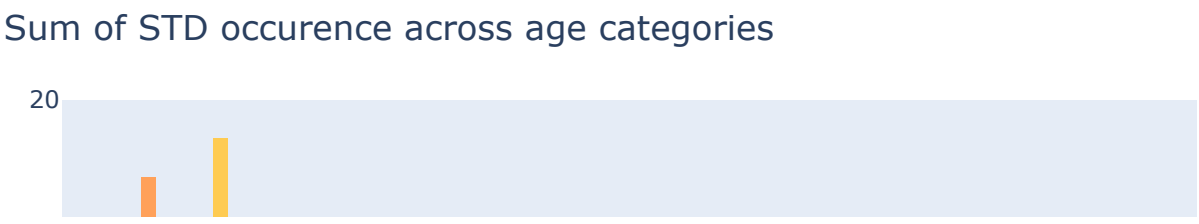


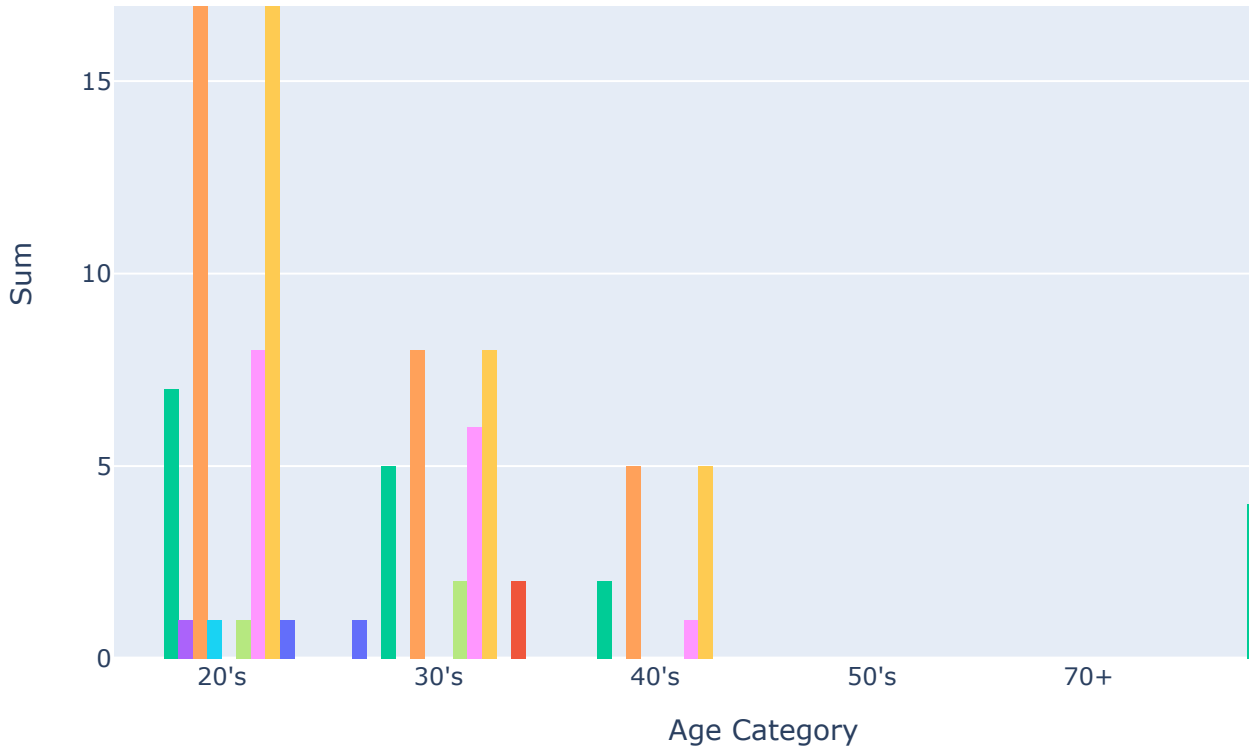

```
# risk_factor_df.columns
# exclude_columns = ['age_cat', 'total_std']

# # Select columns excluding those in exclude_columns
# selected_columns = risk_factor_df.columns[~risk_factor_df.columns.isin(exclude_
```

```
label = 'Number of sexual partners'
diagnoses_cols = [label,
                  'Dx:CIN',
                  'Dx:HPV']
diagnoses_corr_matrix = risk_factor_df[diagnoses_cols].corr()
# print(diagnoses_corr_matrix)
diagnoses_heatmap = px.imshow(diagnoses_corr_matrix, aspect="auto", color_continu
diagnoses_heatmap.show()
```

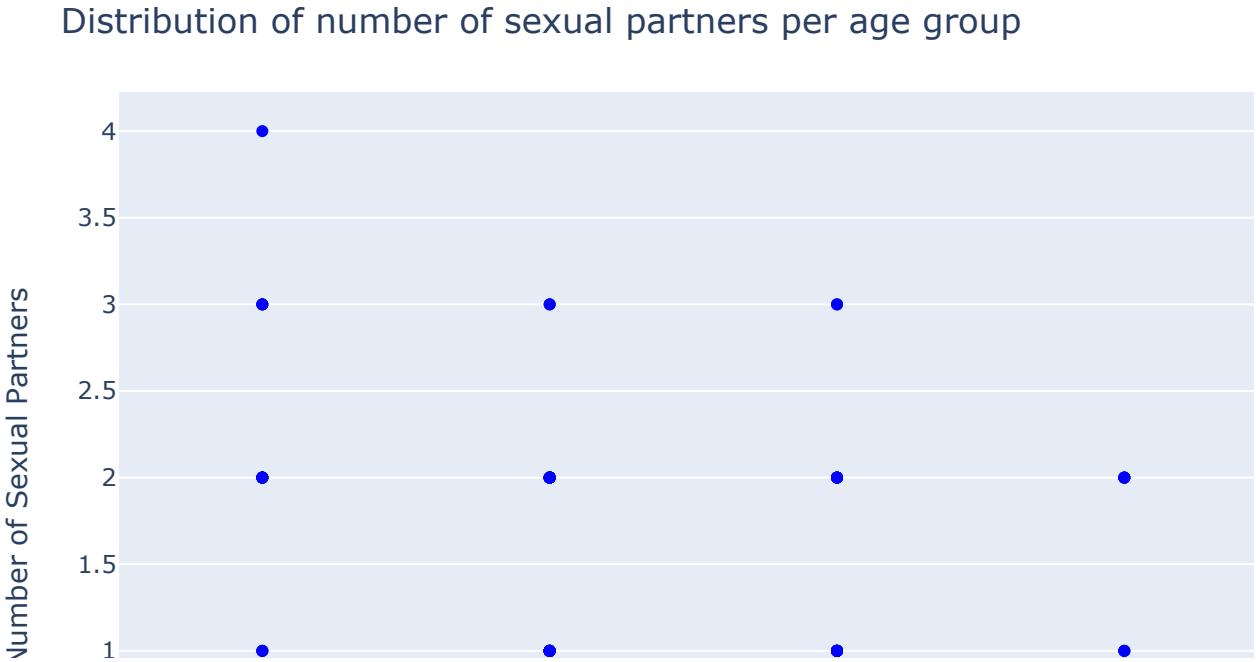


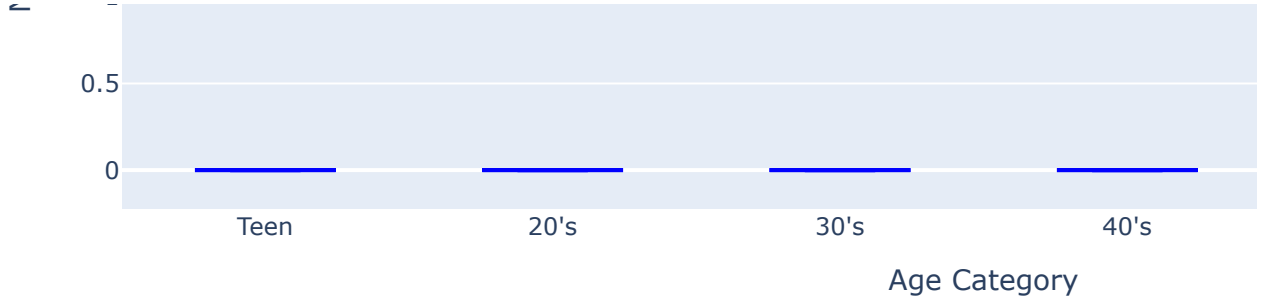

```
fig = px.histogram(std_agg, x="age_cat", y=list(std_cols), barmode="group", histf
fig.update_layout(title="Sum of STD occurence across age categories")
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Sum")
fig.show()
```





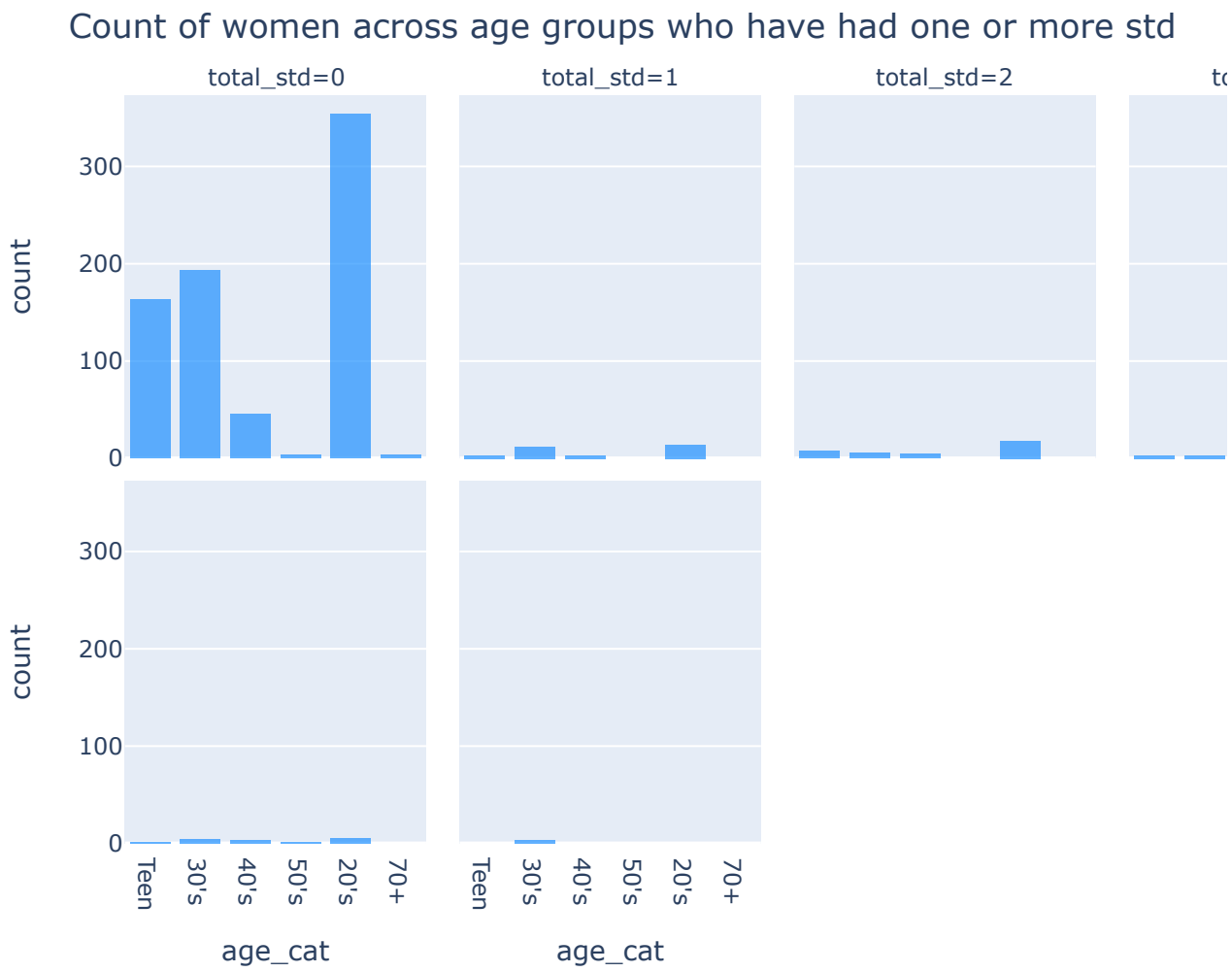

```
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True)
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "Thirties", "Forti
                                       "Seventy and over"])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



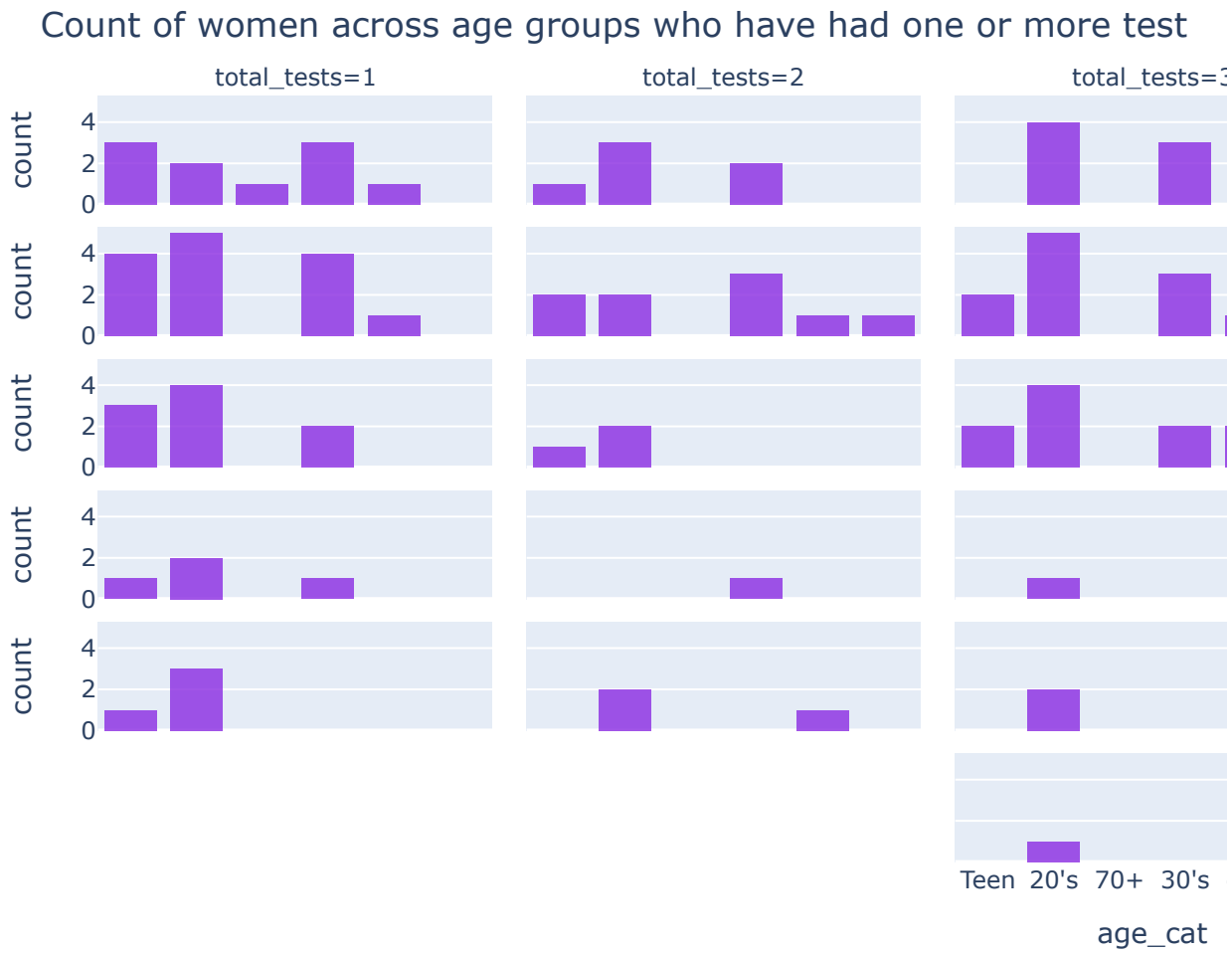



```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_st
                   x="age_cat",
                   facet_col="total_std",
                   facet_row=label,
                   color_discrete_sequence=["rebeccapurple"],
                   opacity=0.7)
fig.update_layout(title="Count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

fig.show()
```

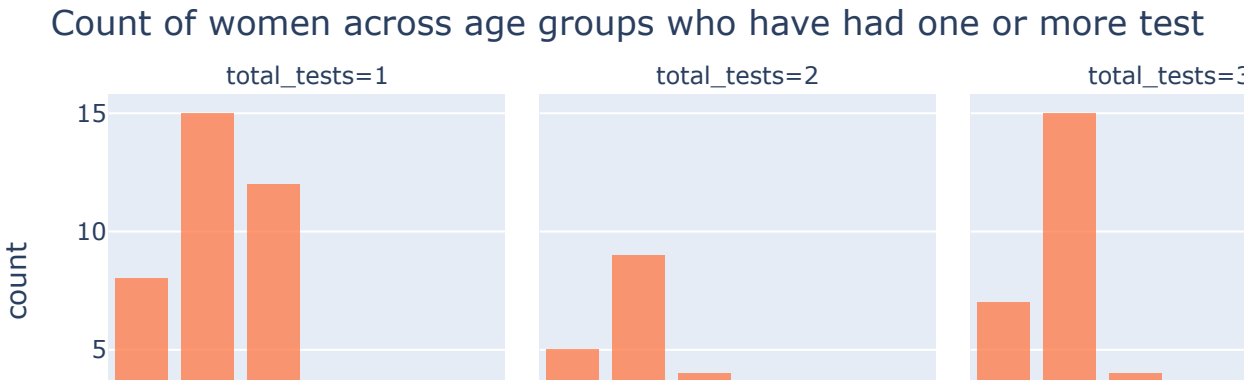
```
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_st
                   x="age_cat",
                   facet_col="total_std",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["dodgerblue"],
                   opacity=0.7)
fig.update_layout(title="Count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

fig.show()
```

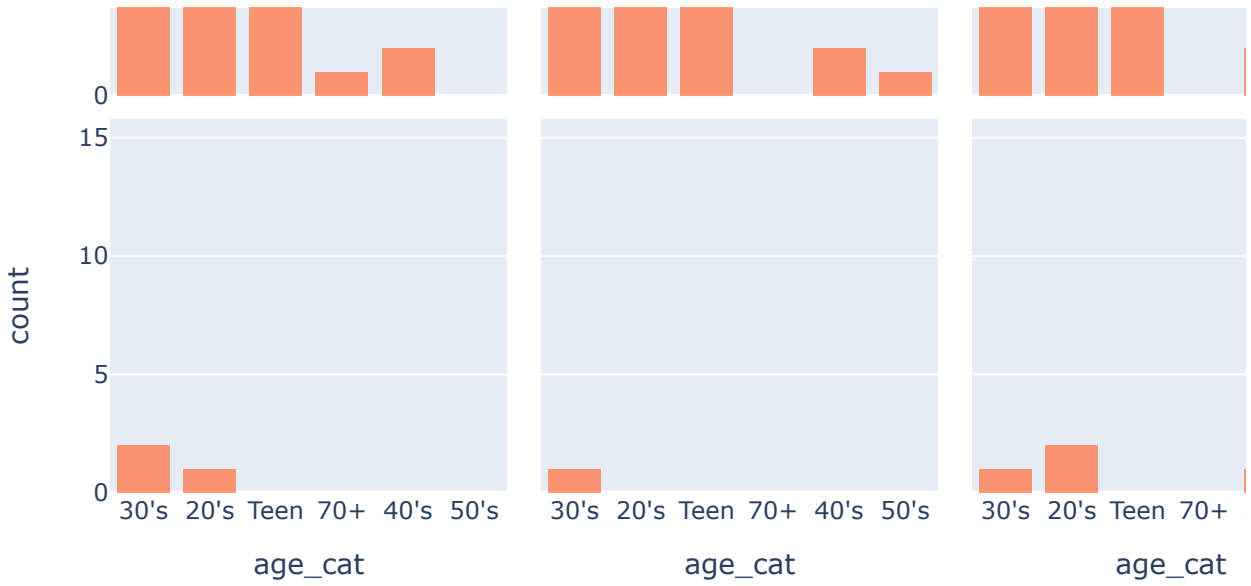



```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by="total_te
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row=label,
                   color_discrete_sequence=["blueviolet"],
                   opacity=0.8)
fig.update_layout(title="Count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

fig.show()
```

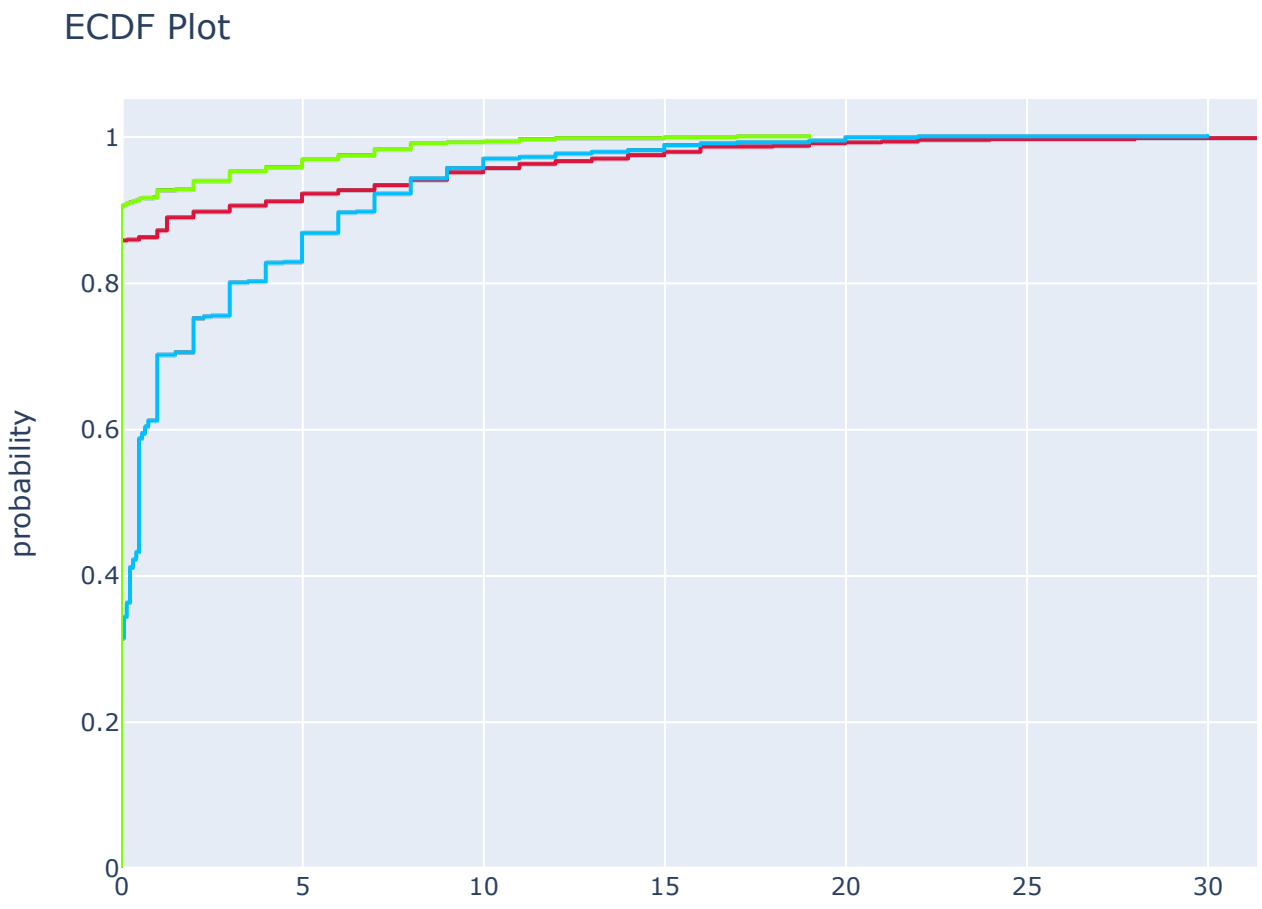



```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by=["total_t
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["coral"],
                   opacity=0.8)
fig.update_layout(title="Count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

fig.show()
```





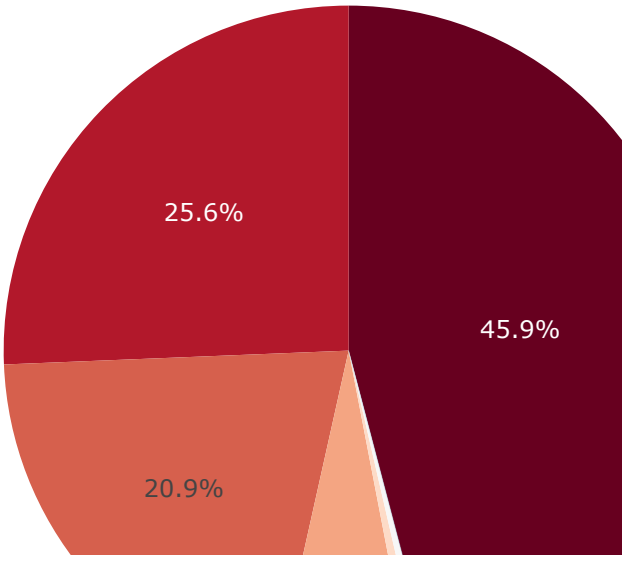

```
fig = px.ecdf(risk_factor_df, x=["Smokes (years)",
                                 "Hormonal Contraceptives (years)",
                                 "IUD (years)"],
              color_discrete_sequence=["crimson", "deepskyblue", "chartreuse"])
fig.update_xaxes(title="Years")
fig.update_layout(title="ECDF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

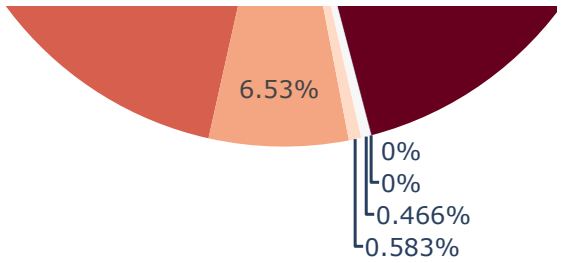


Years

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

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_seque
fig.show()
proportion_samples_df
```

Age Category proportion of women sampled

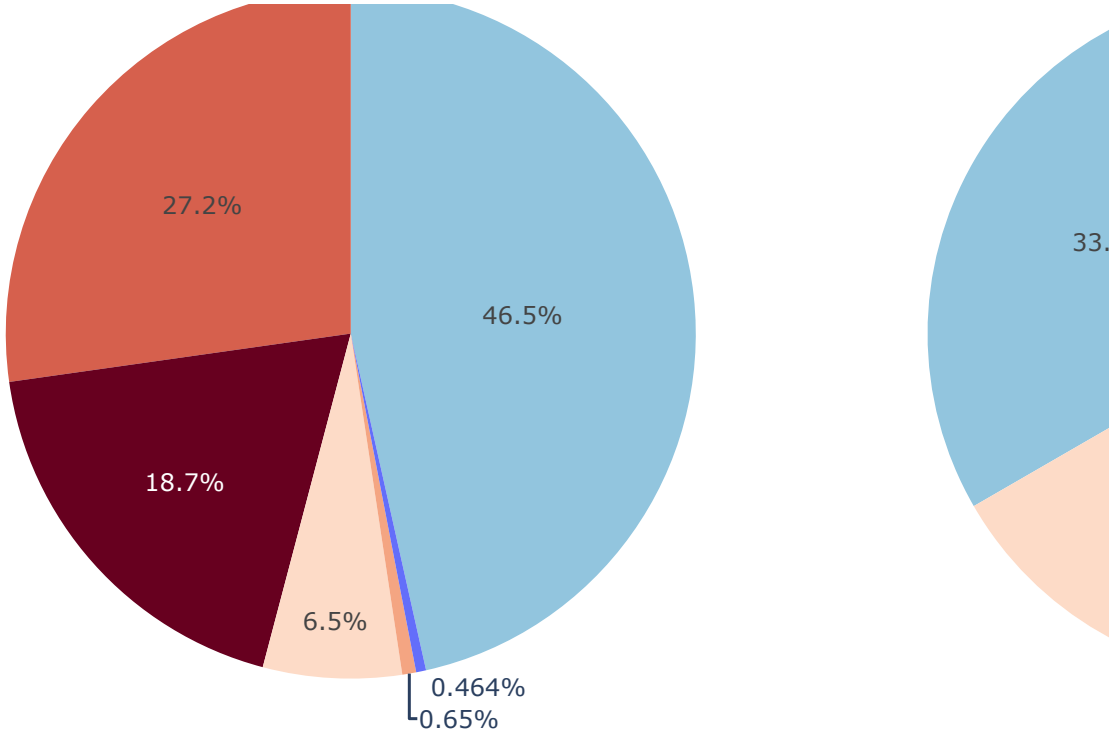




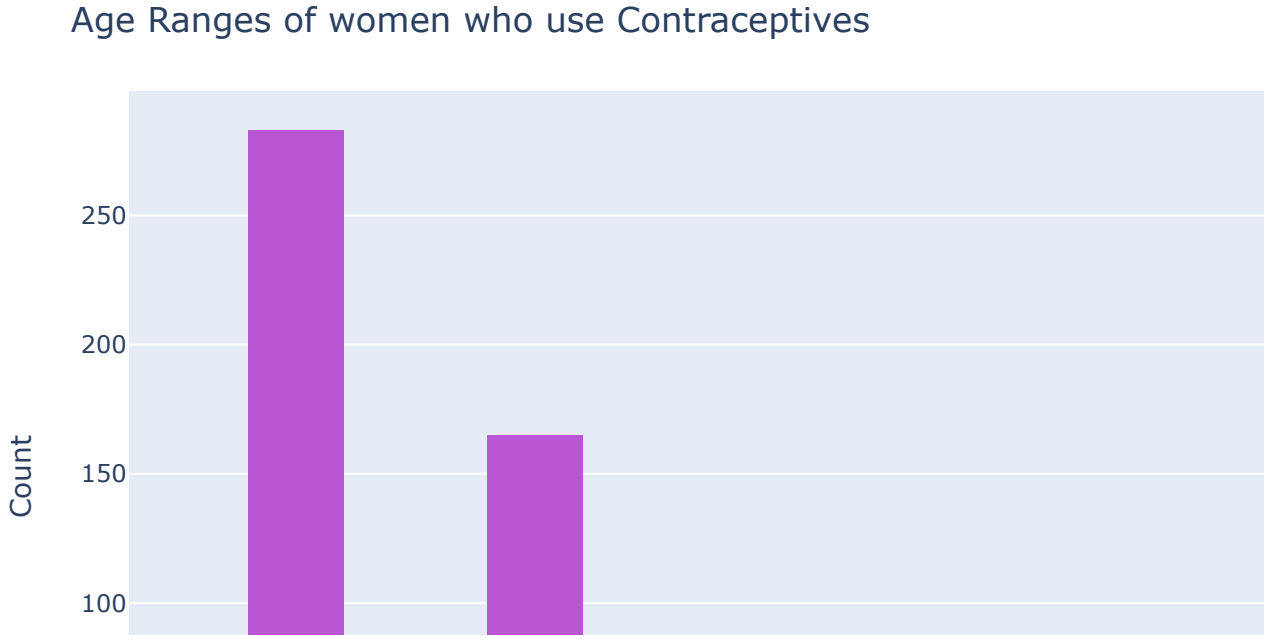
	Category	Sample Proportion
0	Child	0.000000
1	Teen	0.208625
2	20's	0.459207
3	30's	0.256410
4	40's	0.065268
5	50's	0.005828
6	60's	0.000000
7	70+	0.004662

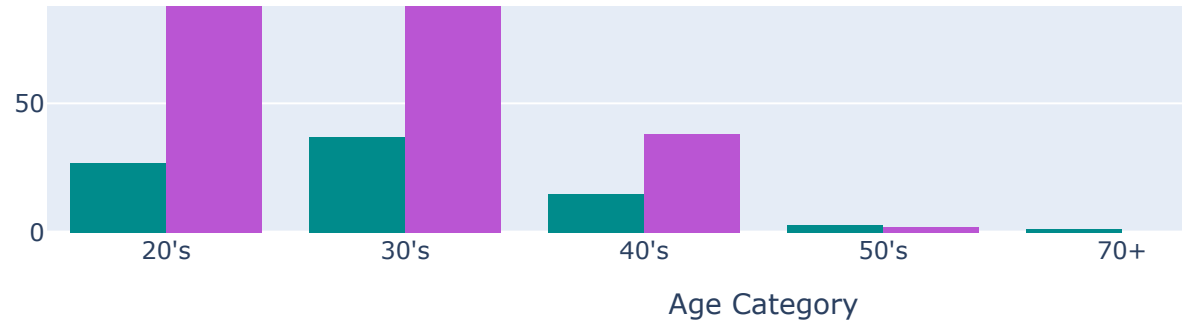
Next steps: View recommended plots

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

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

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

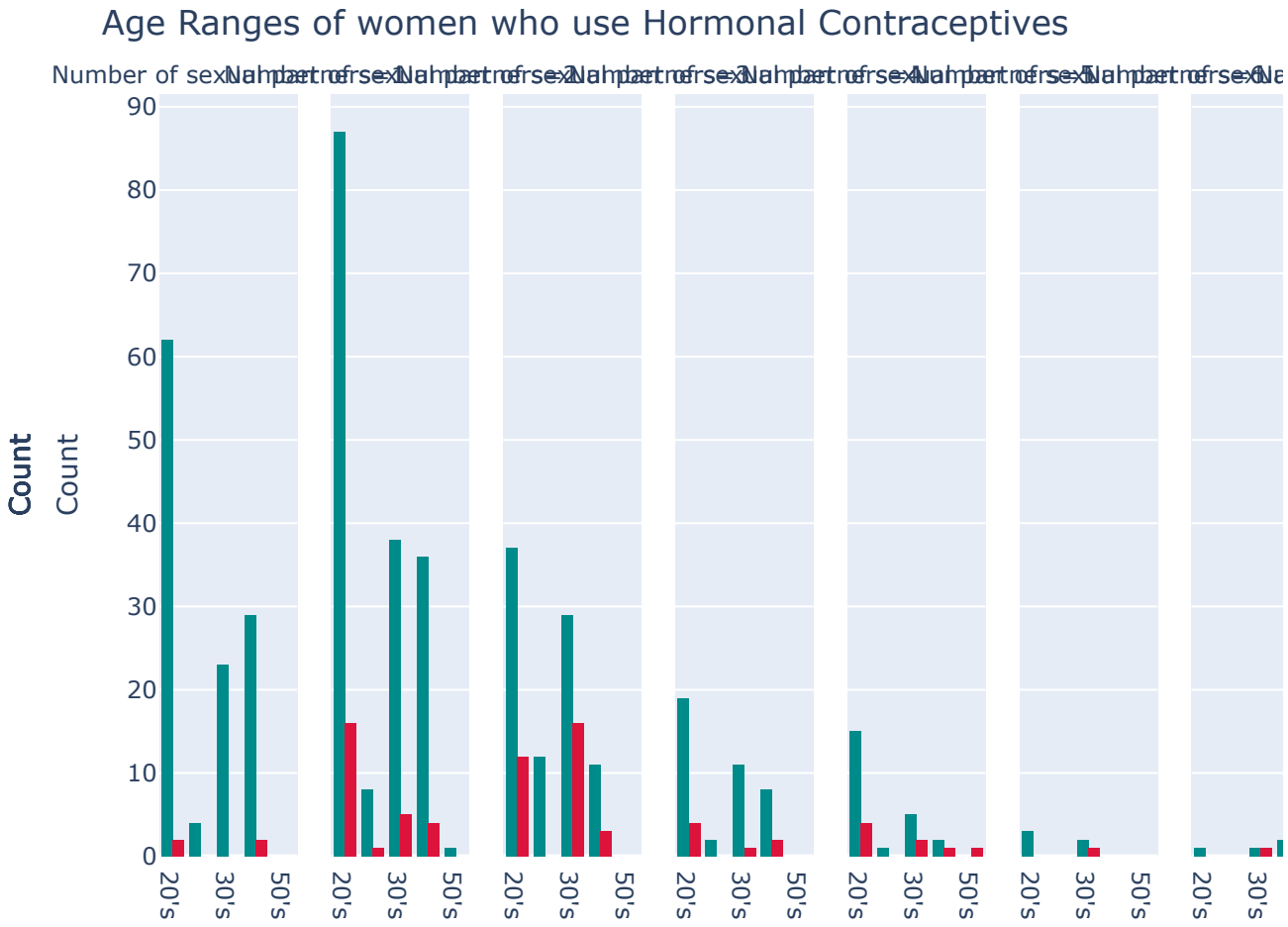



```
df_hormonal_compariosn = risk_factor_df.groupby(["age_cat"], as_index=False)[["Il
fig = px.histogram(df_hormonal_compariosn, x="age_cat", y=["IUD", "Hormonal Contr
                   , color_discrete_sequence=["darkcyan", "mediumorchid"])

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

fig.show()
```



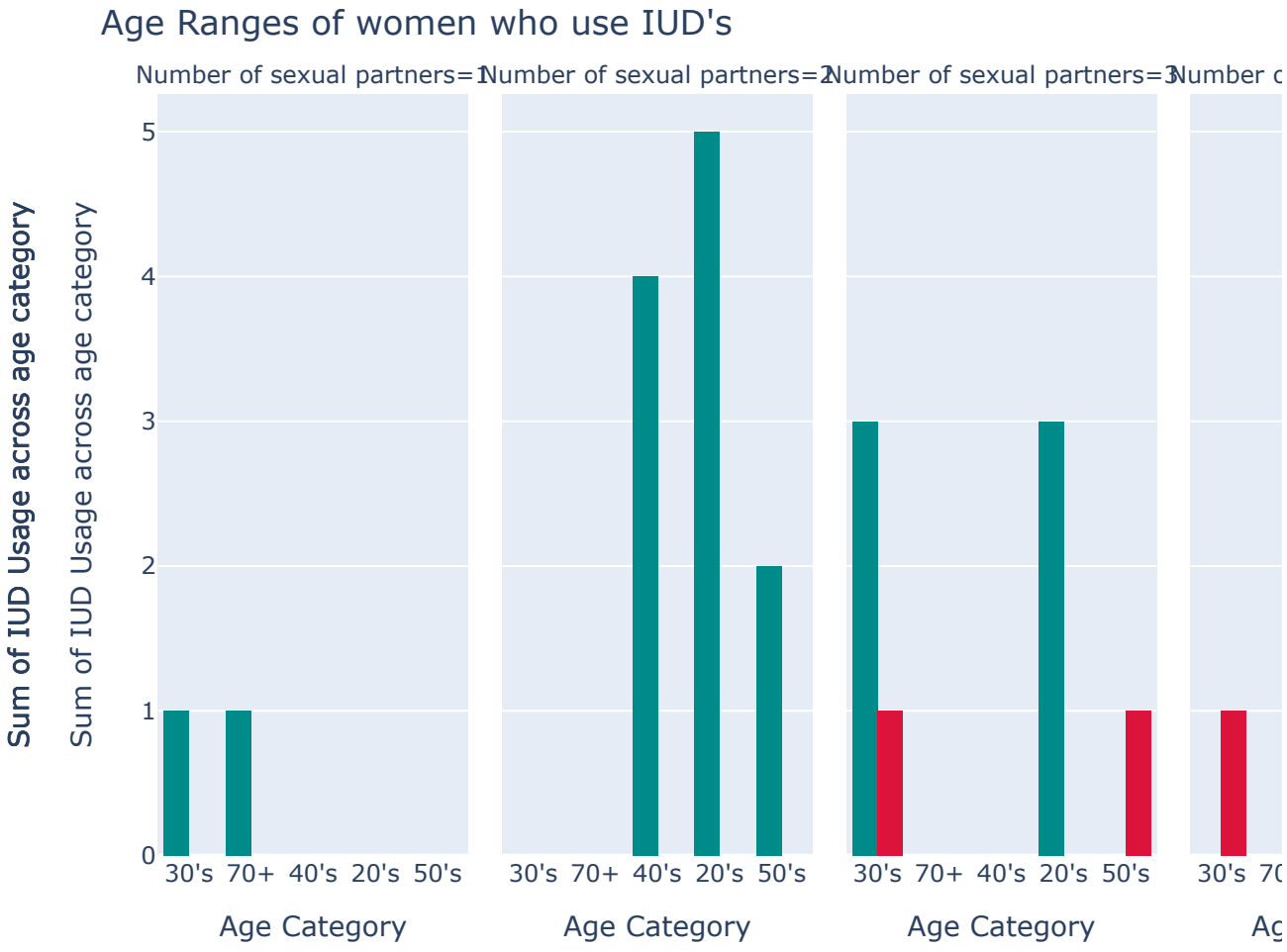



```
df_hormonal_contraceptives = risk_factor_df[
    (risk_factor_df["Hormonal Contraceptives"] == 1) & (risk_factor_df["IUD"] == 0
df_hormonal_contraceptives = df_hormonal_contraceptives.sort_values(by=["Smokes",
fig = px.histogram(df_hormonal_contraceptives, x="age_cat", color="Smokes", barmod
   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



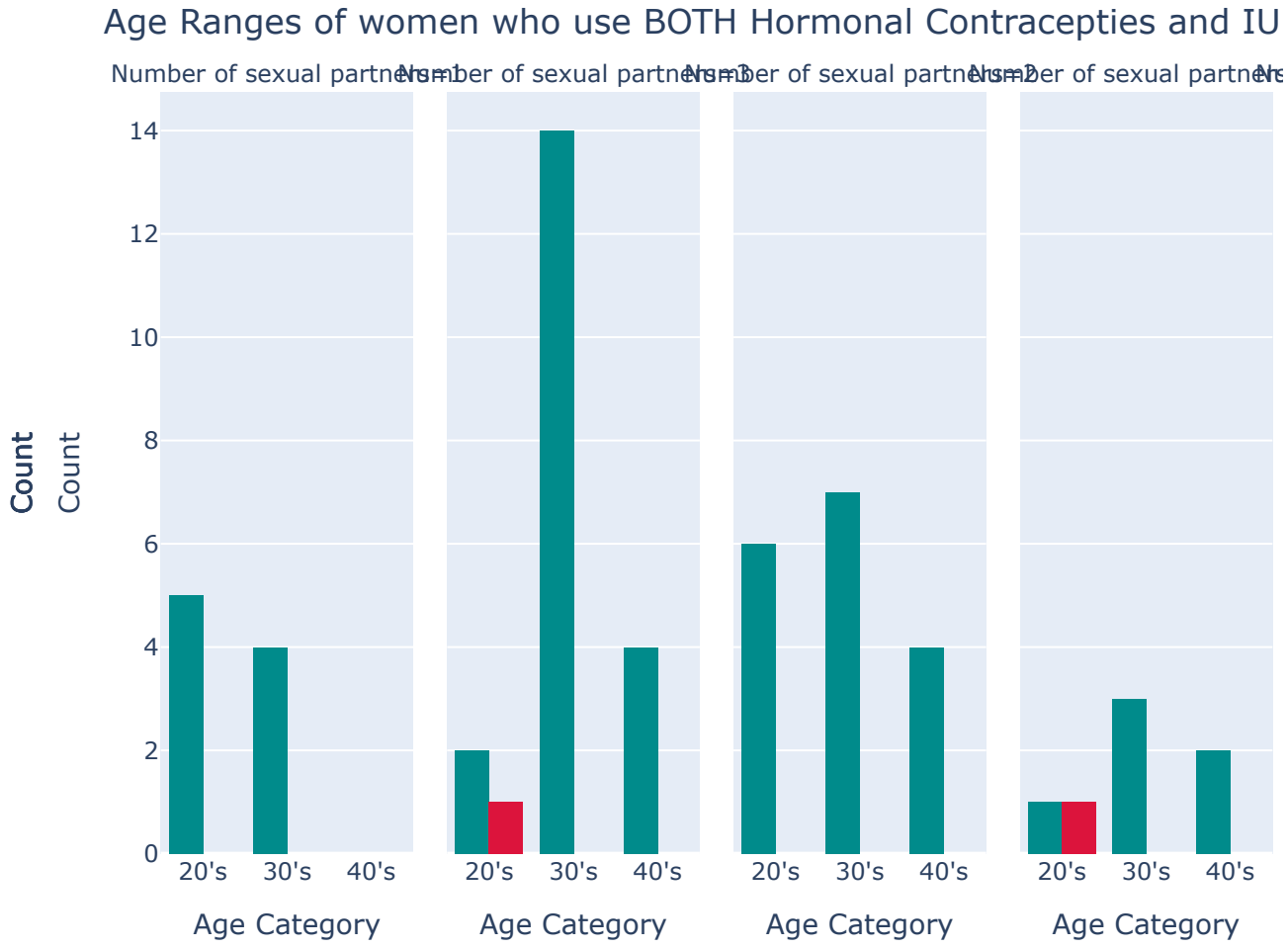
Age CategoryAge CategoryAge CategoryAge CategoryAge CategoryAge CategoryAge Cate

```
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
                   color_discrete_sequence=["darkcyan", "crimson"])
fig.update_xaxes(title="Age Category")
fig.update_yaxes(title="Sum of IUD Usage across age category")
fig.update_layout(title="Age Ranges of women who use IUD's")
fig.show()
```




```
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="
                   color_discrete_sequence=["darkcyan", "crimson"])
fig.update_xaxes(title="Age Category")
```

```
fig.update_yaxes(title="Count")
fig.update_layout(title="Age Ranges of women who use BOTH Hormonal Contracepties
fig.show()
```



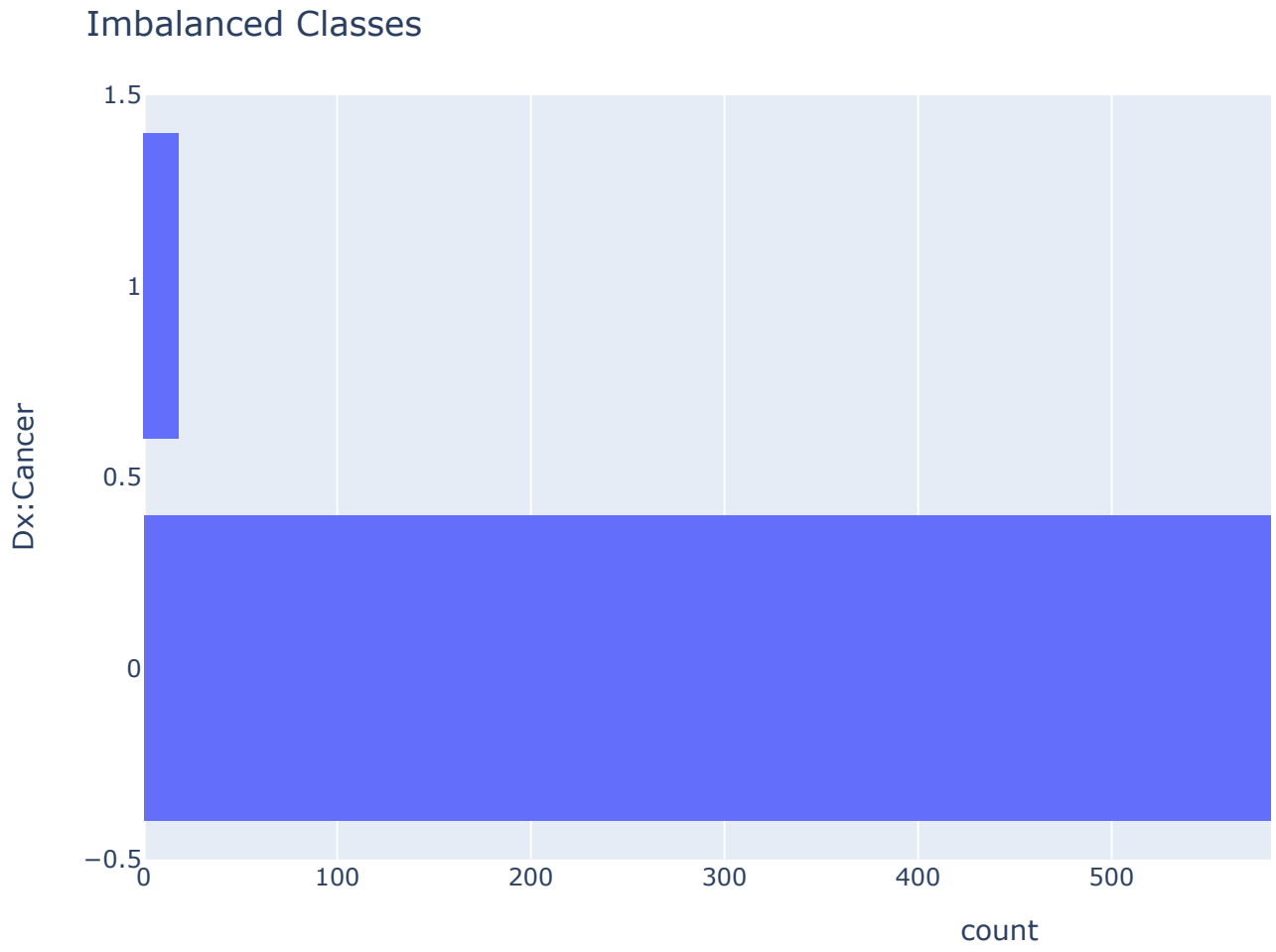
Test / Train Split

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

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

risk_factor_df.columns
```

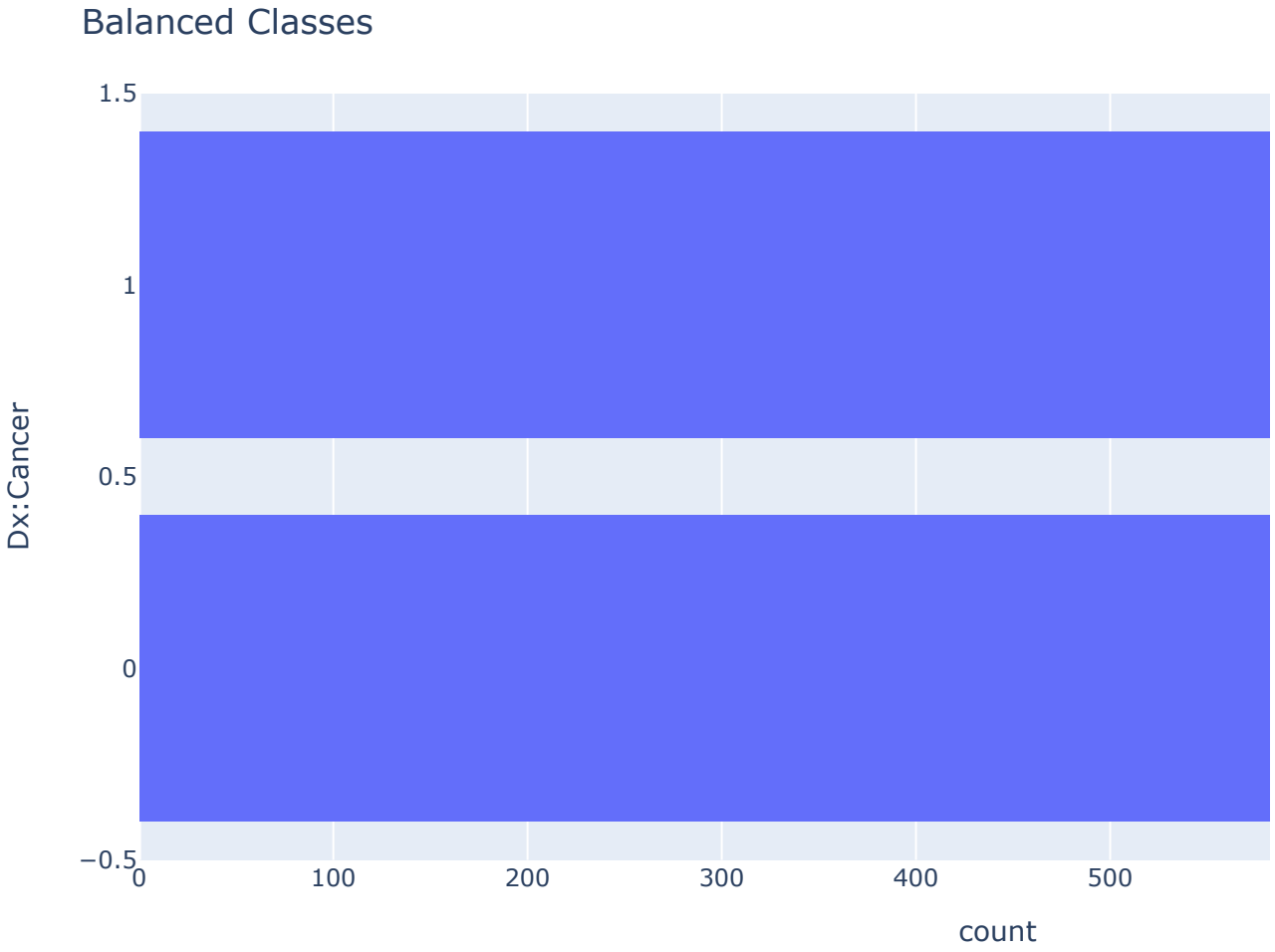
```
Index(['Age', 'Number of sexual partners', 'First sexual intercourse',
       'Num of pregnancies', 'Smokes', 'Smokes (years)', 'Smokes (packs/
year)',
       'Hormonal Contraceptives', 'Hormonal Contraceptives (years)', 'IUD',
       'IUD (years)', 'STDs', 'STDs (number)', 'STDs:condylomatosis',
       'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis'
```

```
       'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
       'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis',
       'STDs:pelvic inflammatory disease', 'STDs:genital herpes',
       'STDs:molluscum contagiosum', 'STDs:AIDS', 'STDs:HIV',
       'STDs:Hepatitis B', 'STDs:HPV', 'STDs: Number of diagnosis',
       'STDs: Time since first diagnosis', 'STDs: Time since last
diagnosis',
       'Dx:Cancer', 'Dx:CIN', 'Dx:HPV', 'Dx', 'Hinselmann', 'Schiller',
       'Citology', 'Biopsy', 'age_cat', 'total_std', 'total_tests'],
      dtype='object')
```

```
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)

# ros = RandomOverSampler(random_state=42)
# x_ros, y_ros = ros.fit_resample(X, y)
# risk_factor_df = x_ros.join(y_ros)

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()
```




```
rain_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')
```

Model

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.
- 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.

```
# class my_model():
#   # use this class to define your model
#   pass

# model = my_model()
# loss_func = None
# optimizer = None

# def train_model_one_iter(model, loss_func, optimizer):
#   pass

# num_epoch = 10
# # model training loop: it is better to print the training/validation losses dur
# for i in range(num_epoch):
#   train_model_one_iter(model, loss_func, optimizer)
#   train_loss, valid_loss = None, None
#   print("Train Loss: %.2f, Validation Loss: %.2f" % (train_loss, valid_loss))
```

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,y
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

```
from sklearn.neural_network import MLPClassifier

nn_clf = MLPClassifier()



# # Add a random variable
# # Binary

# from scipy.stats import bernoulli

# 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.sha

# risk_factor_df.columns
```

```
# from scipy.stats import bernoulli

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


# 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])


# from scipy.stats import bernoulli

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



# continous
# 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])
```

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 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.

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

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

est_name = []
est_acc = []
precision_score = []
recall_score = []
f1score = []
```

```
est_conf_matrix = []
roc=[]

estimators = [
    ("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 [1.0]
scores de LogisticRegression (1.0, 1.0, 1.0, None)
roc [1.0, 1.0]
scores de RandomForestClassifier  (1.0, 1.0, 1.0, None)
roc [1.0, 1.0, 0.9628571428571429]
scores de KNeighborsClassifier (0.9642001915708812, 0.9613095238095238, 0.961
roc [1.0, 1.0, 0.9628571428571429, 0.9971428571428572]
scores de SupportVectorClassifier (0.9970421810699589, 0.9970238095238095, 0.
roc [1.0, 1.0, 0.9628571428571429, 0.9971428571428572, 0.9971428571428572]
scores de MLPClassifier (0.9970421810699589, 0.9970238095238095, 0.9970241527
```

```
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
```

```
estimators
```

```
l('LogisticRegression', LogisticRegression(C=1389495.494373136)),
 ('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())]
```

Evaluation

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

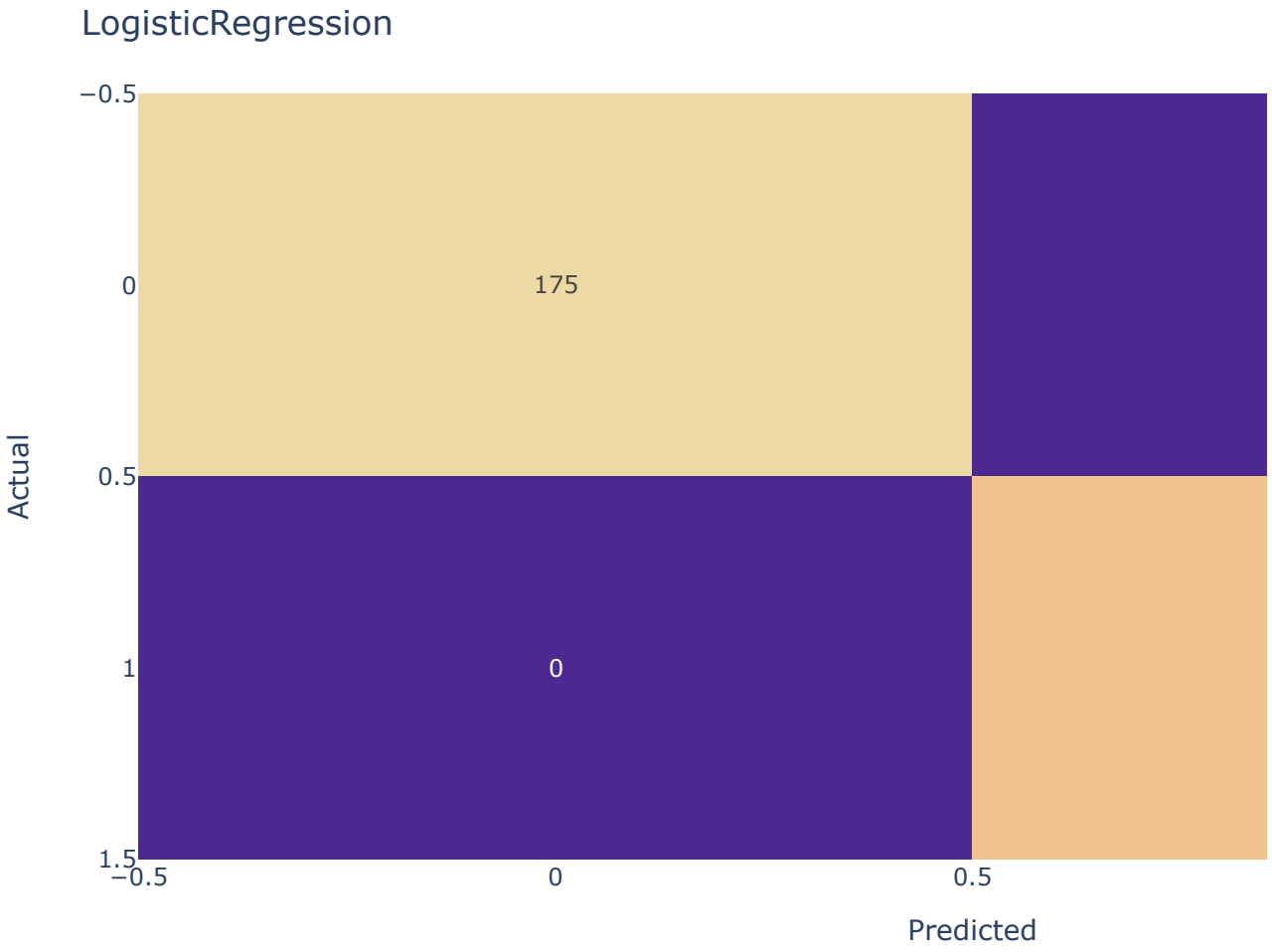
1. confusion matrix
    - the counts of true positives, true negatives, false positives, and false negatives.
2. accuracy
    - how many total predictions did the model get right
3. precision
    - true positives rate by mode out of all positive predictions
4. recall
    - true positives that were correctly 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.

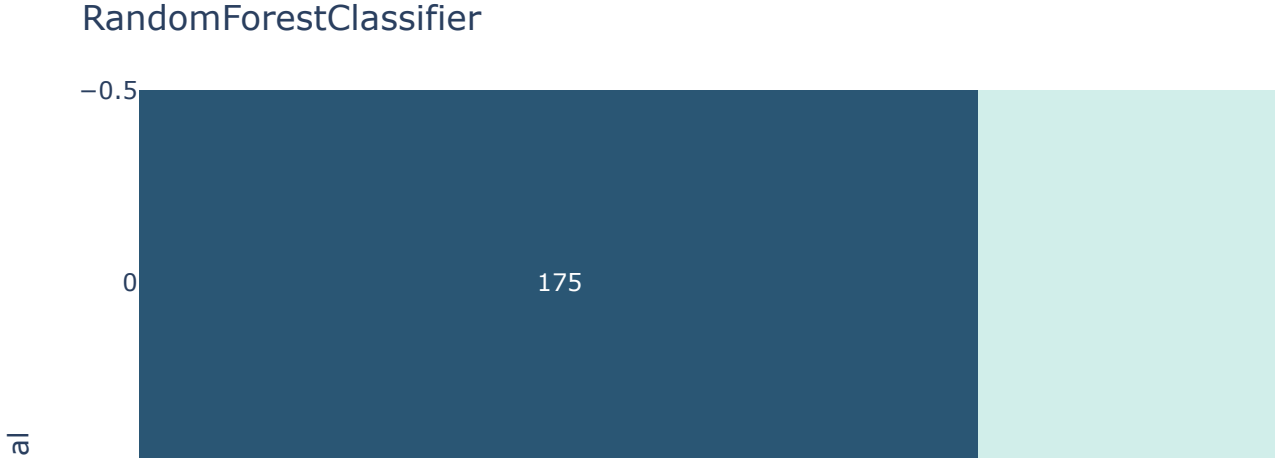
Later in the section we compare each model with others in visualization of the metrics aboves

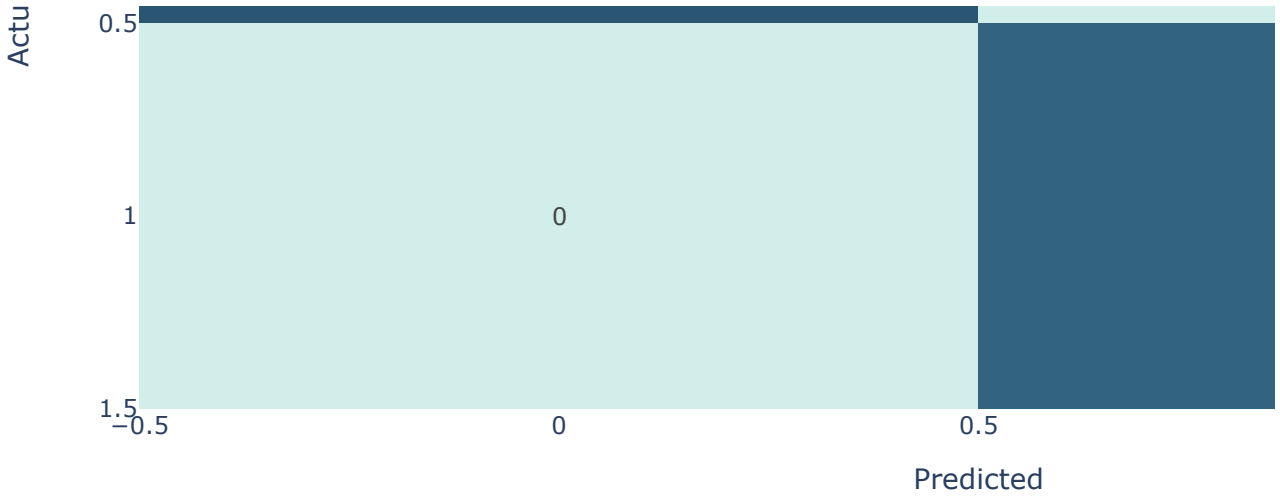
We also plan to use the standard precision, recall, and RemOve And Retrain (ROAR) (Hooker et al., 2018) mentioned in the paper for the faithfulness metric, where we will iteratively remove a subset of features from a dataset, and retrain the model on the reduced dataset to measure the changes in model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

```
color_scales = ["agsunset","teal","purp","viridis","viridis"]
```

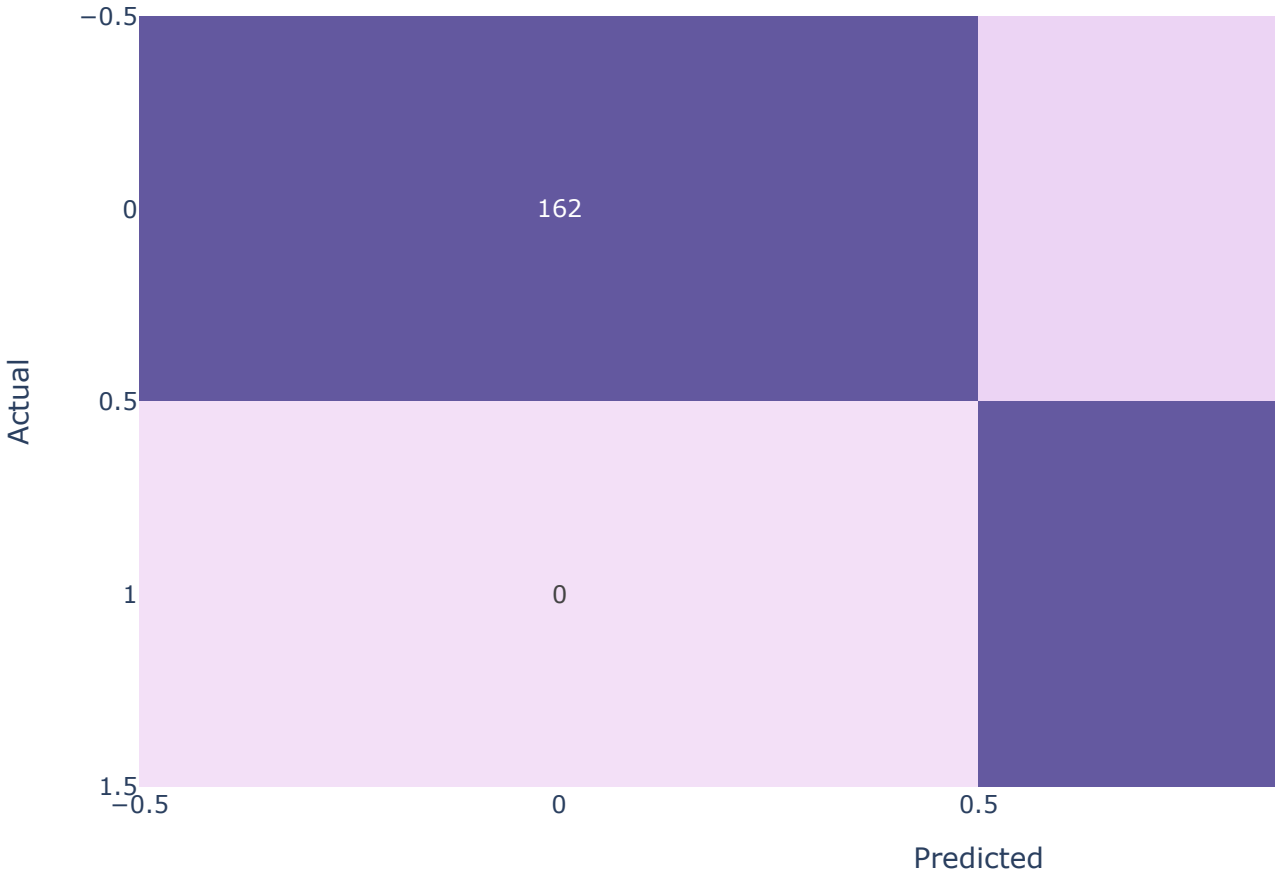
```
for i in range(0,len(est_conf_matrix)):
    heatmap = px.imshow(est_conf_matrix[i],aspect="auto",
                        text_auto=True,
                        color_continuous_scale=color_scales[i])
    heatmap.update_layout(title = est_name[i])
    heatmap.update_xaxes(title="Predicted")
   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



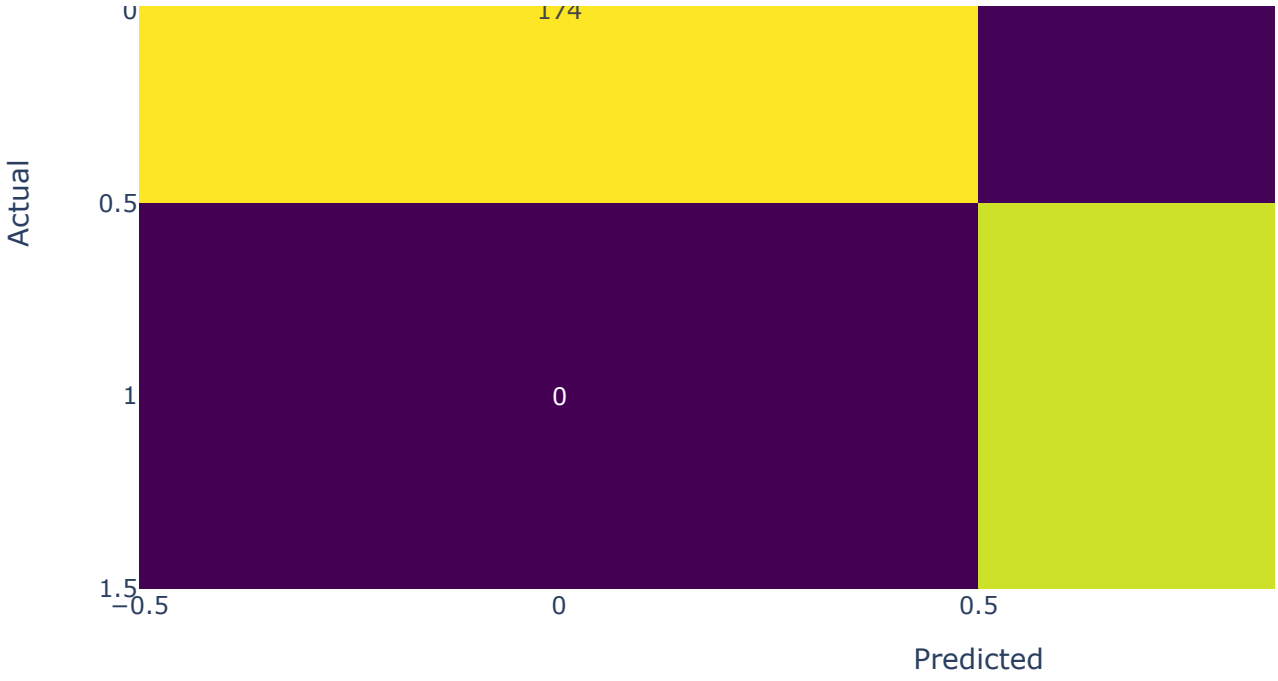




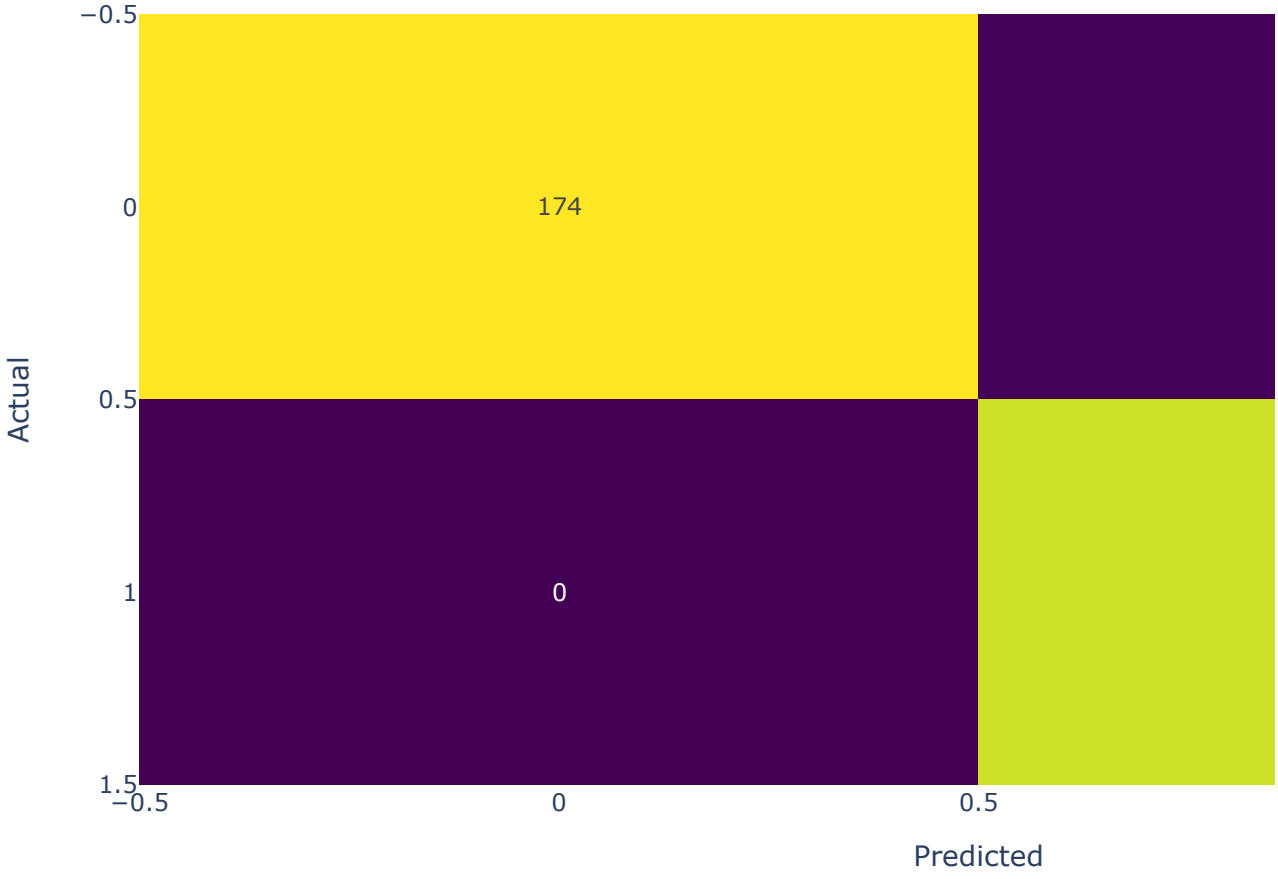
KNeighborsClassifier



SupportVectorClassifier



MLPClassifier

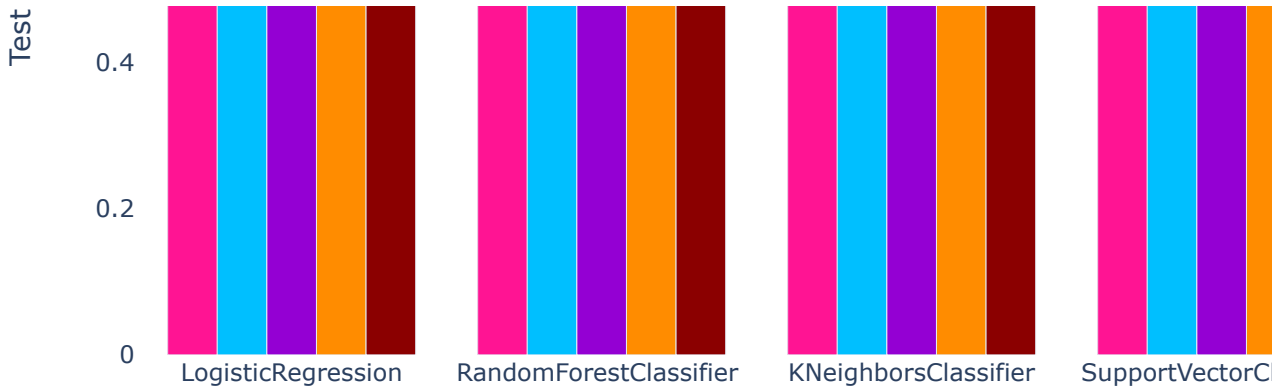



```
summary_df
```

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

Next steps: View recommended plots

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

# acc_comparison.write_image('/content/drive/My Drive/modelsperf.png')
```



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 Explanability results and analyses, and briefly go over our plan for the rest of the project time that we will continue to work on post this draft submission.

Explainability results

```
# Local methods

# Generate local FI

# !pip install shap
# !pip install lime
# !pip install interpret-community
# !pip install alibi
# !pip install treeinterpreter
# !pip install SALib
# !pip install dice-ml
# !pip install pip install spectralcluster
# !pip install -U kaleido
```

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

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

import random

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

res = dict()
features=X_test.columns

exclude_columns = ['VARB', 'VARC', 'VAR']

# Select columns excluding those in exclude_columns
features = X_test.columns[~X_test.columns.isin(exclude_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-
```

```
# 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)}
# really slow to run

# 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)}
```

```
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 [19:31<00:00,  2.76s/it]
```

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

```
# 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("-DICE-")
import dice_ml
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)}
```

```
    -DICE-
      0%|          | 0/336 [00:00<?, ?it/s]
    ---------------------------------------------------------------------------
    KeyError                                  Traceback (most recent call last)
    /usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                                 8 frames

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'total_std'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, check indexing error
```

res

```
{'Age': 0.05478741581076661,
 'Number of sexual partners': 0.020373274215737223,
 'First sexual intercourse': 0.027395499708499062,
 'Num of pregnancies': 0.025703293084869093,
 'Smokes': 0.2682359087067908,
 'Smokes (years)': 0.1888831409462688,
 'Smokes (packs/year)': 0.17768843073881263,
 'Hormonal Contraceptives': 0.10349758437167045,
 'Hormonal Contraceptives (years)': 0.04206428462594364,
 'IUD': 0.07064987998543601,
 'IUD (years)': 0.15508139298275814,
 'STDs': 0.02577736303723636,
 'STDs (number)': 0.06749160942678899,
 'STDs:condylomatosis': 0.11466246856142194,
 'STDs:cervical condylomatosis': 0.0,
 'STDs:vaginal condylomatosis': 0.052943594187400876,
 'STDs:vulvo-perineal condylomatosis': 0.12140552171841196,
 'STDs:syphilis': 0.12575384267839157,
 'STDs:pelvic inflammatory disease': 0.07021053835007887,
 'STDs:genital herpes': 0.07396533873349494,
 'STDs:molluscum contagiosum': 0.0791510627351735,
 'STDs:AIDS': 0.0,
 'STDs:HIV': 0.10917638771414036,
 'STDs:Hepatitis B': 0.06723006421744263,
 'STDs:HPV': 0.16675271206523373,
 'STDs: Number of diagnosis': 0.027763451020485597,
```

```
      'STDs: Time since first diagnosis': 0.03752274842324762,
      'STDs: Time since last diagnosis': 0.1340259341724159,
      'Dx:CIN': 0.26923880683727835,
      'Dx:HPV': 0.4952454472486191,
      'Dx': 0.4392184136697498,
      'Hinselmann': 0.062271076232955365,
      'Schiller': 0.007886414985872194,
      'Citology': 0.052673188695461365,
      'Biopsy': 0.0133621182309997 43}
```

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

Analyses

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.

```
dics = []

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

```
dics = pd.DataFrame(dics)
methods=dics['Method']
dics['Method']=methods
dics.to_csv("/content/drive/My Drive/dics_1.csv", index=False)

dics
```

	Smokes (years)	STDs (number)	STDs:pelvic inflammatory disease	Age	Biopsy	Hormonal Contraceptives (years)	STDs:cer condyloma
0	0.691009	0.023605	0.000000	0.570522	0.003037	0.436398	
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
3	0.033802	0.001697	0.000024	0.024584	0.000926	0.017731	
4	0.188883	0.067492	0.070211	0.054787	0.013362	0.042064	
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

7 rows × 36 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, dicecontrib]

GloSur
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)	(
0	-0.554033	-0.112369	-0.026319	0.248043	0.119635	-0.688855	0.008948	
1	0.014206	-0.014657	-0.026319	0.248043	0.119635	-0.688855	0.008948	
2	-0.198884	0.034199	-0.026319	0.248043	0.119635	-0.688855	0.008948	

3	-0.554033	0.034199	-0.092116	0.248043	0.119635	-0.688855	0.008948
4	0.440385	-0.014657	0.105275	0.248043	-1.076716	-0.141089	0.004311
...	...	...	...	...	...	...	...
331	-0.411973	0.034199	0.039478	0.248043	0.119635	-0.688855	0.008948
332	-0.269914	-0.014657	0.105275	-0.744128	-1.076716	0.406676	0.007093
333	-1.335362	0.034199	0.171072	0.744128	0.119635	-0.688855	0.008948
334	-0.554033	-0.063513	0.039478	0.248043	0.119635	-0.688855	0.008948
335	-0.554033	-0.112369	-0.026319	0.248043	0.119635	-0.688855	0.008948

336 rows × 35 columns

kernelSHAP

	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	
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 × 35 columns

samplingSHAP

	Number of	First	Num of	Smokes	Smokes

	Age	sexual partners	sexual intercourse	pregnancies	Smokes	(years)	(packs/ year)
0	0.003530	-0.007144	-0.008059	0.000582	0.016175	-0.020187	0.005484
1	0.034117	-0.000340	-0.002150	0.011293	0.020569	-0.009904	-0.002616
2	0.005105	0.000866	-0.000717	0.000257	0.000025	-0.016398	0.003184
3	-0.015931	0.002320	-0.010264	0.002579	0.007484	-0.021510	0.003640
4	0.013928	-0.003248	0.003791	0.008231	-0.172198	0.095506	-0.026514
...	...	...	...	...	...	...	...
331	-0.008771	0.002575	0.002041	0.000695	0.008223	-0.020169	-0.000167
332	0.008955	-0.002582	0.000060	-0.046124	-0.172417	0.110410	-0.006795
333	-0.064155	0.001011	0.019818	0.011782	0.001460	-0.019123	-0.000168
334	0.005135	-0.001690	0.005338	0.006634	0.016874	-0.017238	0.007723
335	0.008570	-0.004711	-0.001903	0.007978	0.022626	-0.027418	0.000354

336 rows × 35 columns

`limecontrib`

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/ year)
0	0.000328	-0.017003	0.008783	0.020998	0.256379	-0.192050	0.188822
1	0.043672	0.001982	0.009414	0.014600	0.259647	-0.183518	0.184216
2	0.042447	0.024446	0.005563	0.026736	0.266055	-0.195469	0.167712
3	0.002944	0.017486	-0.014570	0.010403	0.274756	-0.190716	0.181888
4	0.069138	0.000993	0.033841	0.009556	-0.249597	0.200148	-0.182206
...	...	...	...	...	...	...	...
331	-0.002873	0.024156	0.008391	0.019495	0.255597	-0.191625	0.161903
332	0.053471	-0.008048	0.037956	-0.067385	-0.271954	0.179016	-0.180678
333	-0.111065	0.011109	0.043487	0.040939	0.257762	-0.179450	0.166274
334	-0.016064	-0.052732	0.007145	0.016374	0.269565	-0.182994	0.178543
335	-0.003862	-0.036915	0.002820	0.025680	0.285565	-0.188553	0.180705

336 rows × 35 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	
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 × 35 columns

```
methods # todo fix me
weights=[GloSur, dicecontrib, dicecontrib, samplingSHAP, limecontrib, dicecontrib
```

```
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.csv')

print('methods' in one_instance.columns)

one_instance
```

```
False
```

methods	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smo (pac ye
Surrogates	-0.554033	0.034199	0.236869	-0.248043	0.119635	-0.688855	0.008
DICE	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
DICE	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
SSHAP	0.005352	0.002953	0.025960	0.003540	0.019518	-0.015715	0.009
LIME	-0.002815	0.023132	0.048514	-0.008781	0.267751	-0.201849	0.169
DICE	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
DICE	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000

7 rows × 35 columns

```
model.predict(X_test)
```

```
array([1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1])
```

```
## gscontrib

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

```
exp = explainer.explain_instance(X_test.iloc[instance], model.predict_proba, num_

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)
```

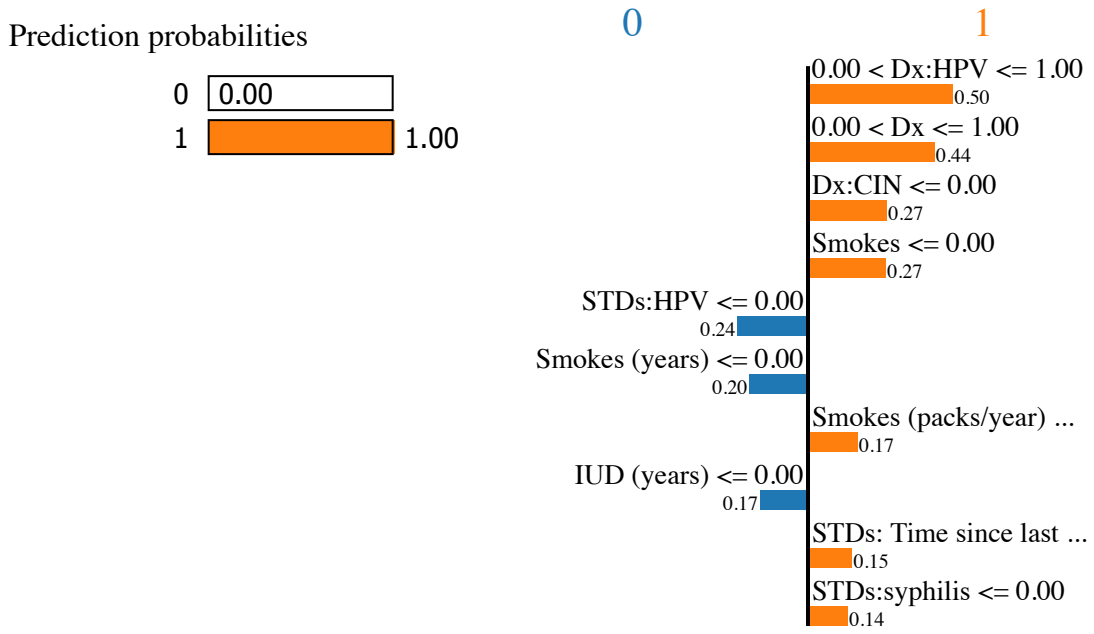
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-86-531e5749fd30> in <cell line: 6>()
      4 exp = explainer.explain_instance(X_test.iloc[instance],
model.predict_proba, num_features=X_test.shape[1])
      5
----> 6 items = gscontrib.iloc[instance].to_dict()
      7 t = []
      8 count=0

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

```
%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
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.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(f)



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

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

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


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

%matplotlib inline
```

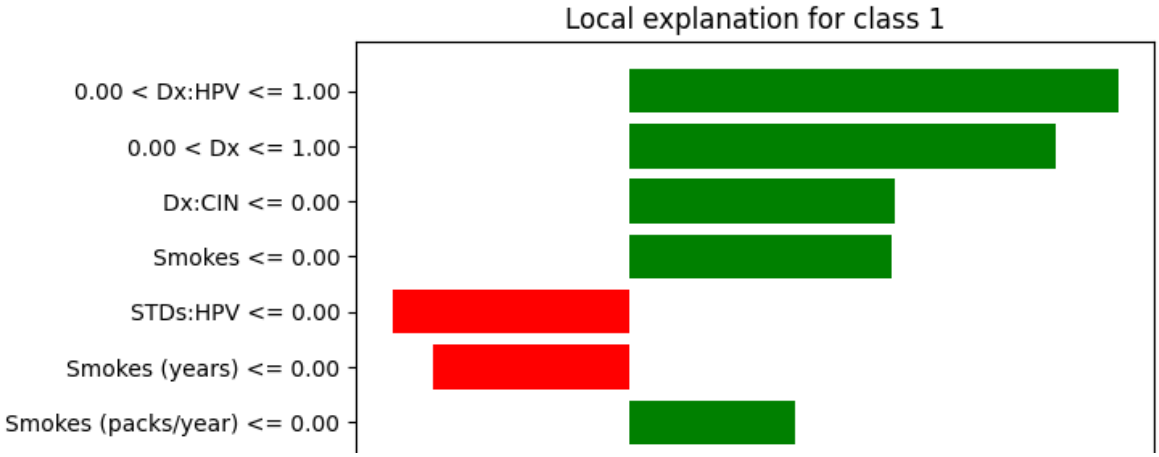
```
fig = exp.as_pyplot_figure()
plt.xlabel("Sampling SHAP")
fig.savefig('/content/drive/My Drive/'+str(var)+'samplingSHAP'+str(instance)+str(
```

```
### 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)+'.png
```

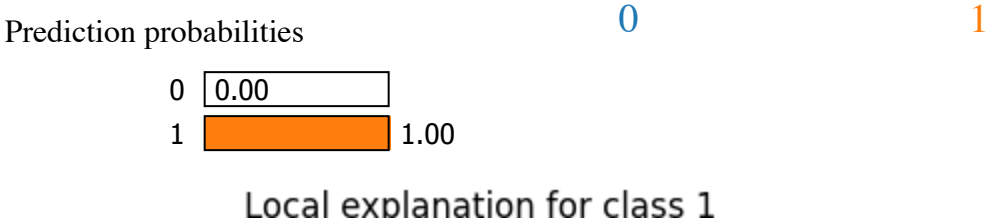


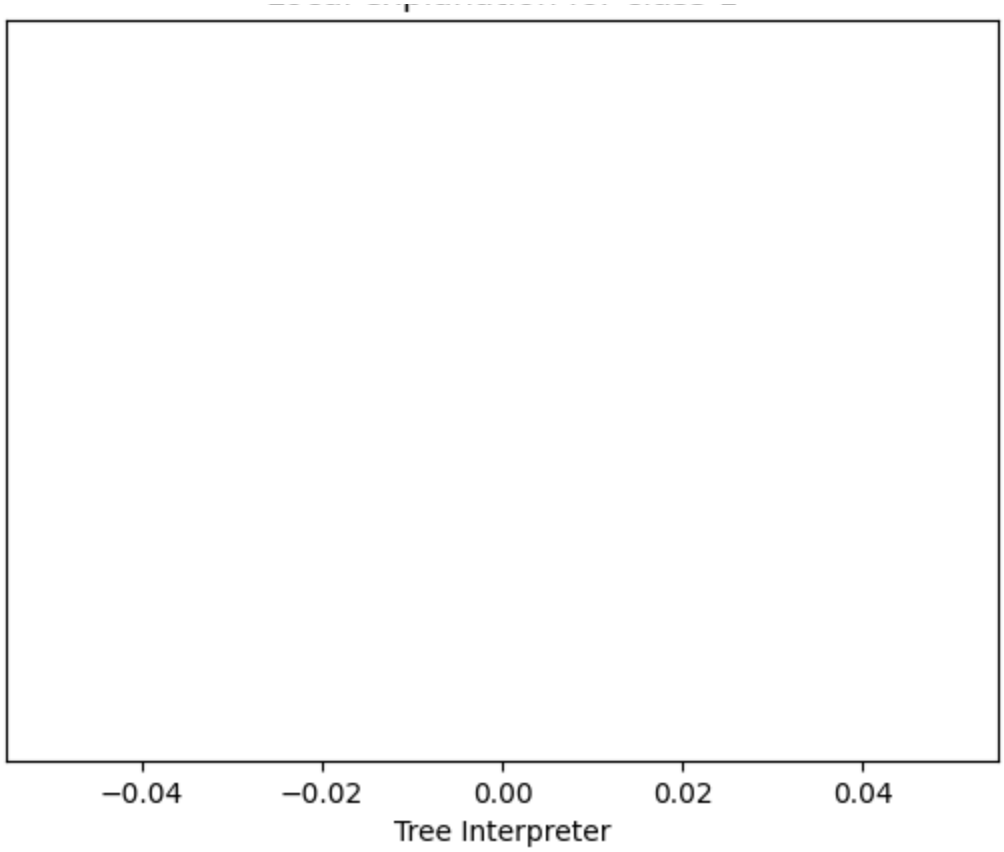
Feature	Value
Dx:HPV	1.00
Dx	1.00
Dx:CIN	0.00

Smokes	0.00
STDs:HPV	0.00
Smokes (years)	0.00
Smokes (packs/year)	0.00
IUD (years)	0.00
STDs: Time since last diagnosis	3.00
STDs:syphilis	0.00

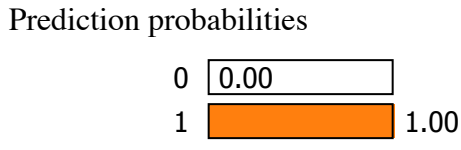


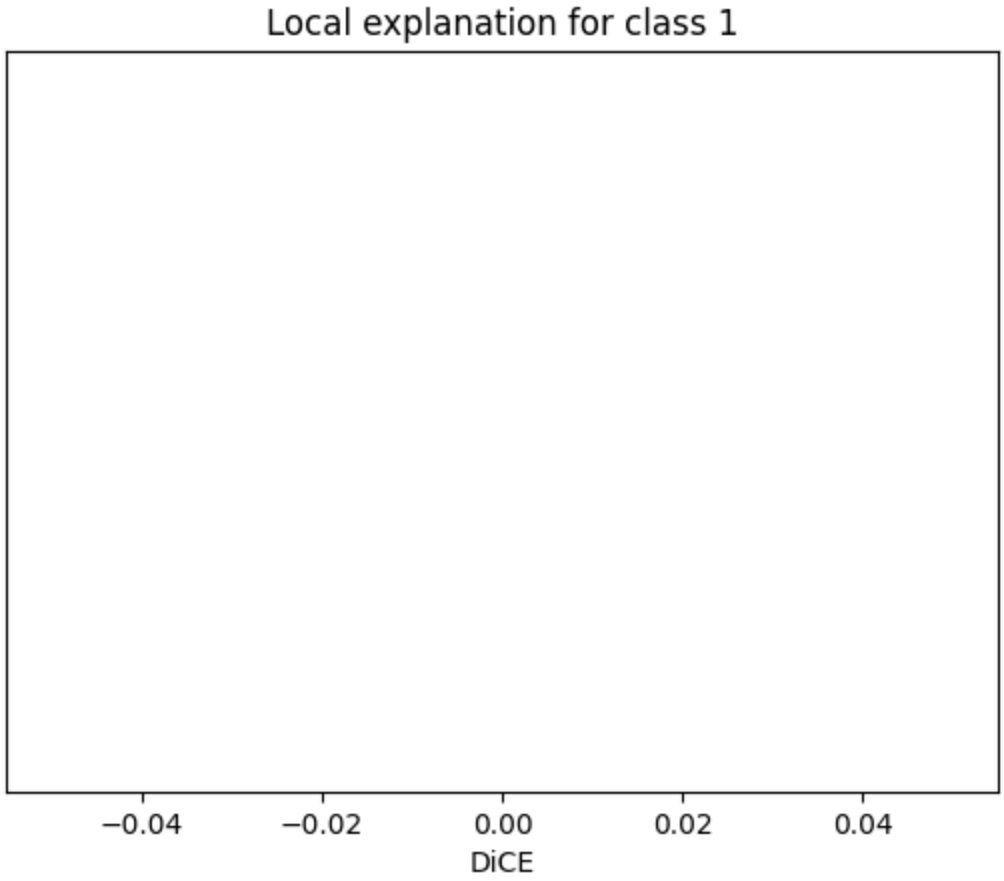

```
### ticontrib

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

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

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


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

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






```
### dice

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

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

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


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

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





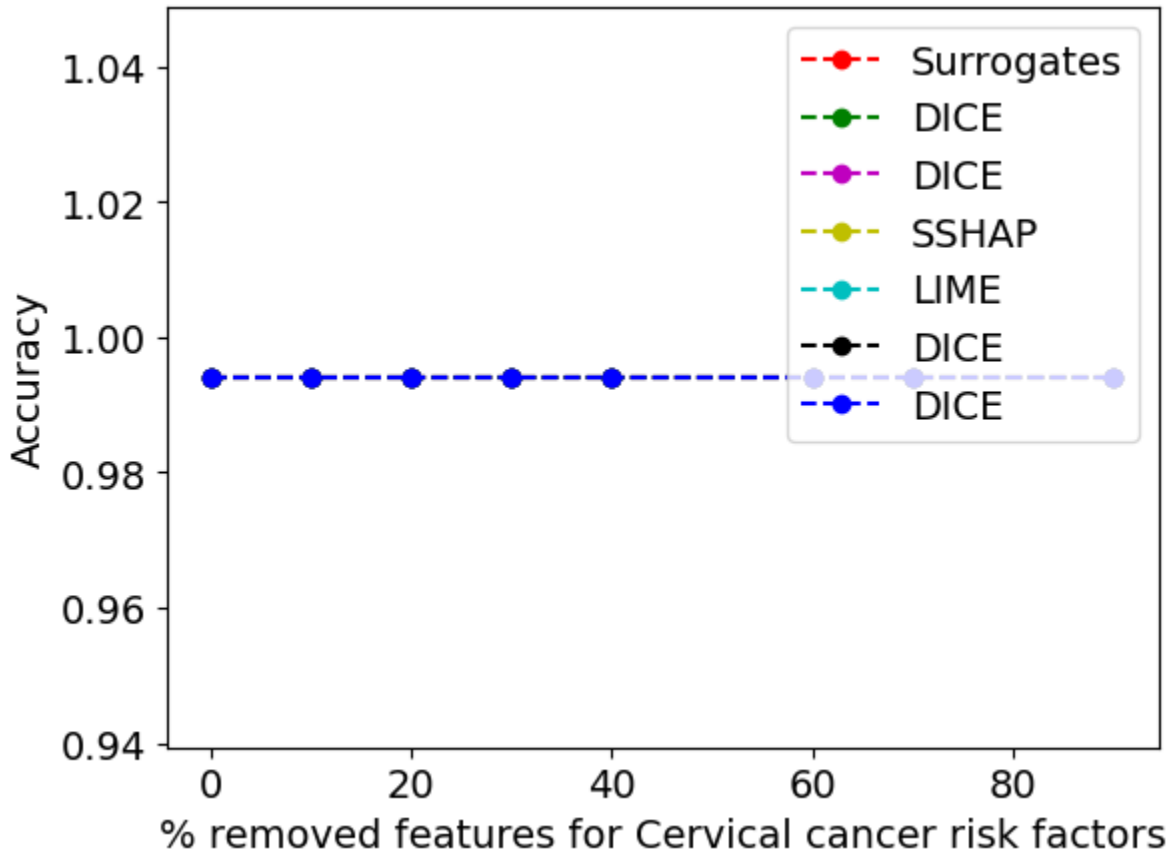

```
### ROAR

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

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

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

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

  df = datapath
  X = df[list(df.columns.drop([feature_to_predict, 'age_cat']))]
  y = df[feature_to_predict]

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

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

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

      fii=list(fi.keys())

      # df= pd.read_csv(datapath)


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

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


      model = RandomForestClassifier(random_state = 42)
      # model.fit(X, y)
      scores = cross_val_score(model, X, y, cv=10)
      accuracies.append(np.mean(scores))

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


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


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


# risk_factor_df.head()
# risk_factor_df[label]
# (risk_factor_df[list(risk_factor_df.columns.drop(label))])
X=risk_factor_df[list(risk_factor_df.columns.drop([label, 'age_cat']))]
y=np.array(risk_factor_df[label])

model = RandomForestClassifier(random_state = 42)
model.fit(X, y)

scores = cross_val_score(model, X, y, cv=10)

scores
```

```
    array([0.95238095, 0.99404762, 1.        , 1.        , 0.99404762,
           1.        , 1.        , 1.        , 1.        , 1.        ])
```

```
datapath = risk_factor_df #'/content/drive/My Drive/kag_risk_factors_cervical_cancer
```

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

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



Plan

Our plan for the remaining work include the following two main areas, post the draft work submission:

1. Feature contributions plot, computing, measuring, and benchmarking Consistency, Compactness, and Stability of the local explanations generated with each method.
2. Validate Feature and Rank disagreement shown in the paper, and see if we reproduce the same feature and rank agreement distribution among the key explanation methods like visualized in the original paper.

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