TA Instructions

Hello! Welcome to Team 11's report. The checkpoints are labeled as such in the headers to make it easier for you to run, but everything has been commented out that is not necessary. Github repo: https://github.com/LoganB99/DL4H-SP24-Local-Explanations-For-Cervical-Cancer/

Mount Notebook to Google Drive

To clear up clutter, the original FAQ and Attentions are in https://colab.research.google.com/drive/1MGxB_J2TvhAANcQG8VNMvQp1QdQrcxWb?authuser=1

```
# import pandas as pd
# from google.colab import drive
# drive.mount('/content/drive', force_remount=True)


# #Testing
# !pip install gspread google-auth
# from google.colab import auth
# auth.authenticate_user()

# import gspread
# from google.auth import default
# creds, _ = default()

# gc = gspread.authorize(creds)
```

Introduction

- Background of the problem

  Cervical Cancer prediction is a prevalent and necessary problem within today's healthcare system. Specifcally, researchers are working towards identifying risk factors and potential causes for cervical cancer. Current models, however, cannot explain why a decision is made. Local explainability techniques have been developed aiming to the causes and effects of changes within a model and explain the decision-making process at an individual prediction level, but may not be applicable in every scenario, and the explanations are not always consistent or faithful. Consequently, clinicians cannot explain why a model selected a patient as low or high risk through black-box magic models.

- Paper explanation

  The paper aims to combat the current problem of ambiguity in explainability across scenarios and analyze the existing local interpretability methods to propose methods to help clinicians determine which type of explanation models to use in a given context. Specifically, the researchers tested 5 differenet ML algorithms to identify the model with the highest disease predition accuracy, and then applied explainability methods to the model to identify the best practice for disease prediction. The paper then uses the predictions of the best model to generate explanations for each model across popular interpretability methods. These methods include LIME, SHAP, Diverse Counterfactual Explanations, Tree Interpreter, and Local Surrogates. Finally, each of the interpretability methods are evaluated on a set of metrics. The model found that LIME is the most robust explainability method, but no single explaination performance optimally across all metrics. Therefore, the researches suggest that clinicians should choose a method based on the setting or choose a weighted sum of metrics. This approach helps satisfy the desired explainability properties when determining patient risk in cervical cancer. As a result, the paper also recommends and influences future models to consider interpetability methods in their analysis to improve trust in the new world of generative and predictive models.

Scope of Reproducibility:

We aim to reproduce the results of the model using the UCI dataset and suggested models. We will use ADASYN to balance the dataset and remove and retrain features as one of our ablations. As another ablation, we want to test the model on the unbalanced dataset.

1. Hypothesis 1: Random Forest is the most performing model in predicting cervical cancer in terms of AUC.
2. Hypothesis 2: LIME is the most robust explainability model shown by the ROAR (remove and retrain) faithfulness metric.

Methodology

Checkpoint: Dependencies

Note: because of Colab's default setup, you may have to restart the notebook and rerun after installing packages.

```
# External package installation
!pip install psutil kaleido
!pip install gdown

# download files and models
import gdown
import pickle

# Basic data handling and scientific computing
import numpy as np  # Numerical computing library
import pandas as pd  # Data manipulation and analysis

# Visualization libraries
import seaborn as sns  # Statistical data visualization
import matplotlib.pyplot as plt  # Basic plotting library
import plotly.express as px  # Interactive plotting library
```

```
import plotly.graph_objects as go  # For creating custom plots with Plotly
from plotly.subplots import make_subplots  # For creating subplots with Plotly

# Data preprocessing and model evaluation tools
from sklearn.impute import SimpleImputer  # For handling missing data
from sklearn.model_selection import StratifiedShuffleSplit  # For creating strati
from sklearn.preprocessing import RobustScaler, StandardScaler  # Data scaling me
from sklearn.decomposition import PCA  # Principal Component Analysis
from sklearn.pipeline import Pipeline  # For creating modeling pipelines
from sklearn.metrics import accuracy_score, confusion_matrix  # Model evaluation
from sklearn.metrics import precision_recall_fscore_support  # Precision, recall,
from sklearn.metrics import roc_auc_score # AUC score

# Machine learning models
from sklearn.linear_model import LogisticRegression  # Logistic regression model
from sklearn.ensemble import RandomForestClassifier, VotingClassifier  # Ensemble
from sklearn.neighbors import KNeighborsClassifier  # k-Nearest Neighbors model
from sklearn.svm import SVC  # Support Vector Machine model
from sklearn.neural_network import MLPClassifier

# Model selection and hyperparameter tuning
from sklearn.model_selection import GridSearchCV  # For hyperparameter tuning

# Data balancing techniques
from imblearn.over_sampling import SMOTE, ADASYN  # Over-sampling techniques
from imblearn.over_sampling import RandomOverSampler  # Random over-sampling

# Additional utilities
from typing import List  # For type hints
from google.colab import drive  # Google Colab drive integration (if using Google
import warnings  # For controlling warning messages


warnings.filterwarnings('ignore')  # Suppress warning messages for cleaner output

# Plotly setup for notebooks
from plotly.offline import plot, iplot, init_notebook_mode
# init_notebook_mode(connected=True)
```

```
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: kaleido in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/d
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/di
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/pytho
```

Data

The raw dataset comes from UC Irvine Machine Learning Repository. https://archive.ics.uci.edu/dataset/383/cervical+cancer+risk+factors

Fernandes,Kelwin, Cardoso,Jaime, and Fernandes,Jessica. (2017). Cervical Cancer (Risk Factors). UCI Machine Learning Repository. https://doi.org/10.24432/C5Z310.

The dataset was collected at 'Hospital Universitario de Caracas' in Caracas, Venezuela. The dataset contains demographic information, habits, and historic medical records of 858 patients. Several patients decided not to answer some of the questions because of privacy concerns (missing values represented as '?').

The missing values certainly make some of the columns less accurate. Time since STD Diagnosis is largely unknown, followed by presence of an IUD. So there is not a perfect correlation to be determined for every feature.

We load the raw data and display statistics, showing how many patients are without cancer and with cancer. We display the number of unknowns before we clean our data.

To clean and process the data, we convert numerical 'object' columns to integers. We replace '?'s with the median of that column. We rerun the overall statistics to confirm there are no "unknown" values and the number of cancer patients remained the same. We create a new age_category column that stratifies the ages.

We run statistics on the age category to gain context. For example, we can see things like just over 2% of the dataset is diagnosed with cancer, but 44% of those diagnosed with cancer are in their 30's, and 20% of patients who are in their 50's have cancer. (We must remember the dataset is small, which is why we sample using ADASYN). There are only 4 patients 70 or older, and they all happen to not have cancer. This does not mean age is not important.

Processing and Statistics

Checkpoint: Load Data Function

```
def load_data(data_dir):
  # implement this function to load raw data to dataframe/numpy array/tensor
    return pd.read_csv(data_dir, delimiter=',', encoding='utf-8')
```

Checkpoint: Load Raw Data

```
root = '/content/drive/My Drive/DL4H_Sp24_Final_Project/'
# dir and function to load raw data
have_access = True


try:
  data_dir = '/content/drive/My Drive/DL4H_Sp24_Final_Project/risk_factors_cervic
  raw_risk_factor_df = load_data(data_dir)
  have_access = True
except:
  have_access = False
  data_dir = 'risk_factors_cervical_cancer.csv'
  gdown.download('https://drive.google.com/file/d/13Co6aIxBU4KXMNH56TDyQd70pkk_b4
  raw_risk_factor_df = load_data(data_dir)
  print("No access, Used Gdown!")
# https://drive.google.com/drive/folders/1AUr8BgW16UU-7XjFf8O77XAgjA27gISV?usp=sh
raw_risk_factor_df
```

```
No access, Used Gdown!
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Co
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	
...	...	...	...	...	...	...	...	
853	34	3.0	18.0	0.0	0.0	0.0	0.0	
854	32	2.0	19.0	1.0	0.0	0.0	0.0	
855	25	2.0	17.0	0.0	0.0	0.0	0.0	
856	33	2.0	24.0	2.0	0.0	0.0	0.0	
857	29	2.0	20.0	1.0	0.0	0.0	0.0	

858 rows × 36 columns

Checkpoint: Define Statistics methods

```
# calculate dataset statistics
def calculate_dataset_stats(df):
  print(len(df), " total patients")
  print(df['Dx:Cancer'].value_counts()[0], " patients without Cancer")
  print(df['Dx:Cancer'].value_counts()[1], " patients with Cancer")
  print(df.applymap(lambda x: x == "?").sum().sum(), " unknown values")
  specified_value = '?'  # Replace with the value you're interested in
  max_count = -1
  column_with_most = None

  for column in df.columns:
    if column == "STDs: Time since last diagnosis" or column == "STDs: Time since
      continue
    value_counts = df[column].value_counts()
    if specified_value in value_counts:
      if value_counts[specified_value] > max_count:
          max_count = value_counts[specified_value]
          column_with_most = column

  print(f"Column with the most '{specified_value}': {column_with_most} (Count: {m
  dataset_size = df.shape[0] * df.shape[1]
  print(100 * df.applymap(lambda x: x == "?").sum().sum()/dataset_size, " percent

  # Formatting to match the style: dtypes: float64(2), int64(2), object(1)
  formatted_summary = ", ".join([f"{k}: {v}" for k, v in df.dtypes.value_counts()
  print("dtypes:", formatted_summary)


# NOTE: this is a deviation from the source code. In the source code, category_pe
# summed up to 100 across the diagnosis, we also wanted to show the percent for e

def col_stats(df, diagnosis_column, category_column):
    """
    Calculates statistics for diagnosis distributions across categories.

    Parameters:
    - df (DataFrame): The input data frame containing the relevant data.
    - diagnosis_column (str): The name of the column containing diagnosis informa
    - category_column (str): The name of the column containing category labels (e

    Returns:
    - DataFrame: A pivot table presenting the calculated statistics.
    """
```

```
# Calculate the overall percentage of each diagnosis-category combination.
overall_percentages = df[[diagnosis_column, category_column]] \
    .value_counts(normalize=True) \
    .mul(100) \
    .round(decimals=4) \
    .reset_index(name='Overall_Percent')

# Count the occurrences within each category for a diagnosis.
diagnosis_by_category_counts = df.groupby([diagnosis_column, category_column]
    .size() \
    .reset_index(name='Count_in_Category')

# Count the total occurrences within each category.
total_in_category = df.groupby(category_column) \
    .size() \
    .reset_index(name='Total_in_Category')

# Calculate the percentage of each diagnosis within specific categories.
category_percentages = pd.merge(diagnosis_by_category_counts, total_in_catego
category_percentages['Category_Percent'] = category_percentages['Count_in_Cat
    .div(category_percentages['Total_in_Category']) \
    .mul(100) \
    .round(decimals=4)

# Count occurrences of each diagnosis and calculate the percentage within the
diagnosis_counts_and_percentages = pd.merge(
    df.groupby([diagnosis_column, category_column]).size().reset_index(name='
    df.groupby(diagnosis_column).size().reset_index(name='Total_in_Diagnosis'
    on=diagnosis_column
)
diagnosis_counts_and_percentages['Diagnosis_Percent'] = diagnosis_counts_and_
    .div(diagnosis_counts_and_percentages['Total_in_Diagnosis']) \
    .mul(100) \
    .round(4)

# Merge the overall percentages with category-specific percentages.
temp_merged = pd.merge(
    overall_percentages,
    category_percentages[[diagnosis_column, category_column, 'Category_Percen
    on=[diagnosis_column, category_column]
)

# Merge with the diagnosis percentage data.
final_merged = pd.merge(
    temp_merged,
```

```
        diagnosis_counts_and_percentages[[diagnosis_column, category_column, 'Dia
        on=[diagnosis_column, category_column]
    )

    # Create a pivot table for better presentation.
    final_pivot_table = final_merged.pivot(index=category_column, columns=diagnos
    final_pivot_table.fillna(0.000, inplace=True)
    return final_pivot_table

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

Checkpoint: Define Processing Method

```
# process raw data
def process_data(raw_data):
    # implement this function to process the data as you need
  #these columns are not of type object, but are of type numeric
  cols_to_convert = ['Number of sexual partners', 'First sexual intercourse', 'Nu
                     'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contracept
                     'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STD
                     'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:
                     'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:
                     'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AI
                     'STDs:HPV', 'STDs: Time since first diagnosis',
                     'STDs: Time since last diagnosis']

  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'}
```

```
    test_cols = ["Hinselmann", "Schiller", "Citology", "Biopsy"]

    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)

    # convert object columns to numeric and replace with nan
    raw_data[cols_to_convert] = raw_data[cols_to_convert].apply(pd.to_numeric, erro
    raw_data[cols_to_convert].fillna(np.nan, inplace=True)
    # replace nan values with the median of the column
    imp = SimpleImputer(strategy="median")
    X = imp.fit_transform(raw_data)
    risk_factor_df = pd.DataFrame(X, columns=list(raw_data.columns))

    # make new columns
    risk_factor_df["Age"] = risk_factor_df["Age"].astype(int)
    risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)
    risk_factor_df["total_std"] = risk_factor_df[list(std_cols)].sum(axis=1)
    risk_factor_df["total_tests"] = risk_factor_df[test_cols].sum(axis = 1)
    for col in to_int_and_beyond:
      risk_factor_df[col] = risk_factor_df[col].astype(int)

    # Aggregate the STD counts by age categories
    std_agg = risk_factor_df.groupby("age_cat", as_index=False)[list(std_cols)].sum
    return risk_factor_df, std_agg
```

```
# categorize the age ranges
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+"

def save_data(df, path):
  if have_access:
    print("Save checkpoint granted")
    df.to_csv(path, index=False)
  else:
    print("You have no access to save data, skipping save checkpoint")
```

Checkpoint: Calculate Raw Data Stats

```
print('RAW DATA STATS')
calculate_dataset_stats(raw_risk_factor_df)
```

```
RAW DATA STATS
858  total patients
840  patients without Cancer
18  patients with Cancer
3622  unknown values
Column with the most '?': IUD (Count: 117)
11.726236726236726  percent of the dataset is unknown
dtypes: object: 26, int64: 10
```

(BEFORE ADASYN) Process Data, Calculate Processed Stats, and Save Data

```
# # process data set
# risk_factor_df, std_agg = process_data(raw_risk_factor_df)
# save_data(risk_factor_df, '/content/drive/My Drive/DL4H_Sp24_Final_Project/proc
# save_data(std_agg, '/content/drive/My Drive/DL4H_Sp24_Final_Project/processed_s
```

Checkpoint: Load Processed Data

```
try:
  data_dir = '/content/drive/My Drive/DL4H_Sp24_Final_Project/processed_risk_fact
  risk_factor_df = load_data(data_dir)
  unbalanced_risk_factor_df = risk_factor_df
  data_dir = '/content/drive/My Drive/DL4H_Sp24_Final_Project/processed_std_agg.c
  std_agg = load_data(data_dir)
  have_access = True
except:
  have_access = False
  data_dir = 'processed_factors_cervical_cancer.csv'
  gdown.download('https://drive.google.com/file/d/1n-II-zTyOpjToeWmetQFrbZ4YKFjfZ
  risk_factor_df = load_data(data_dir)
  unbalanced_risk_factor_df = risk_factor_df
  data_dir = 'processed_std_agg.csv'
  gdown.download('https://drive.google.com/file/d/11OiEdMjfYzcMtZNh1jIWQJlIOw4iEb
  std_agg = load_data(data_dir)
  print("No access, Used Gdown!")
print("-" * 150)
print('PROCESSED DATA STATS')
# print overall stats, might need to edit
calculate_dataset_stats(risk_factor_df)
print("-" * 150)
# print column specific stats
print('Dx:Cancer by age category')
dxCancerByAge = col_stats(risk_factor_df, 'Dx:Cancer', 'age_cat')
print(dxCancerByAge)
```

```
No access, Used Gdown!
--------------------------------------------------------------------------------
PROCESSED DATA STATS
858  total patients
840  patients without Cancer
18  patients with Cancer
0  unknown values
Column with the most '?': None (Count: -1)
0.0  percent of the dataset is unknown
dtypes: int64: 32, float64: 6, object: 1
--------------------------------------------------------------------------------
Dx:Cancer by age category
          age_cat Overall_Percent         Category_Percent           \
Dx:Cancer                       0       1                0        1
0            20's         45.3380  0.5828          98.7310   1.2690
1            30's         24.7086  0.9324          96.3636   3.6364
2            40's          6.1772  0.3497          94.6429   5.3571
3            50's          0.4662  0.1166          80.0000  20.0000
4             70+          0.4662  0.0000         100.0000   0.0000
5           Teen          20.7459  0.1166          99.4413   0.5587

          Diagnosis_Percent
Dx:Cancer                 0        1
0                   46.3095  27.7778
1                   25.2381  44.4444
2                    6.3095  16.6667
3                    0.4762   5.5556
4                    0.4762   0.0000
5                   21.1905   5.5556
```

Checkpoint: Visualizations

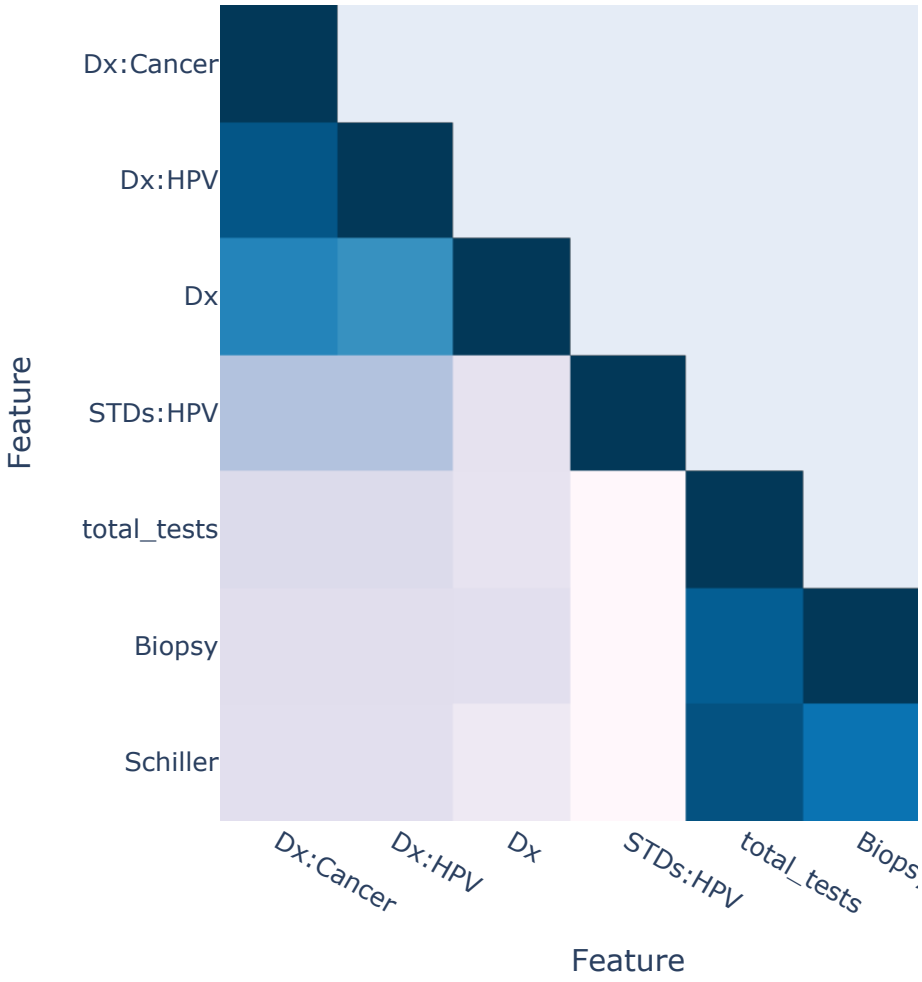
Top features that correlate with cancer Dx

```
# Features that correlate with a cancer diagnosis
n = 7
target = label = "Dx:Cancer"
# correlate the numerical columns of the df
corr = risk_factor_df.select_dtypes(include=np.number).corr()
# find the top 7 correlations with Dx:Cancer
x = corr.nlargest(n,target).index
print(x)
# make a corr_df with only the top 7 columns
corr_df =  risk_factor_df[list(x)]
```

```
# recalculate the correlation
corr = corr_df.corr()
# Creating a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool), k=1)

# Use the mask to replace the upper triangle with np.nan
corr_masked = corr.where(~mask)

# Plot using Plotly Express
fig = px.imshow(corr_masked,
                color_continuous_scale="PuBu",
                labels=dict(x="Feature", y="Feature", color="Correlation"),
                x=corr.columns,  # Adding column names here
                y=corr.index)    # Adding row names here

# Update layout with title
fig.update_layout(title="Top "+str(n)+" Features Correlated With "+str(target).ca
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
# Show plot
fig.show()
```

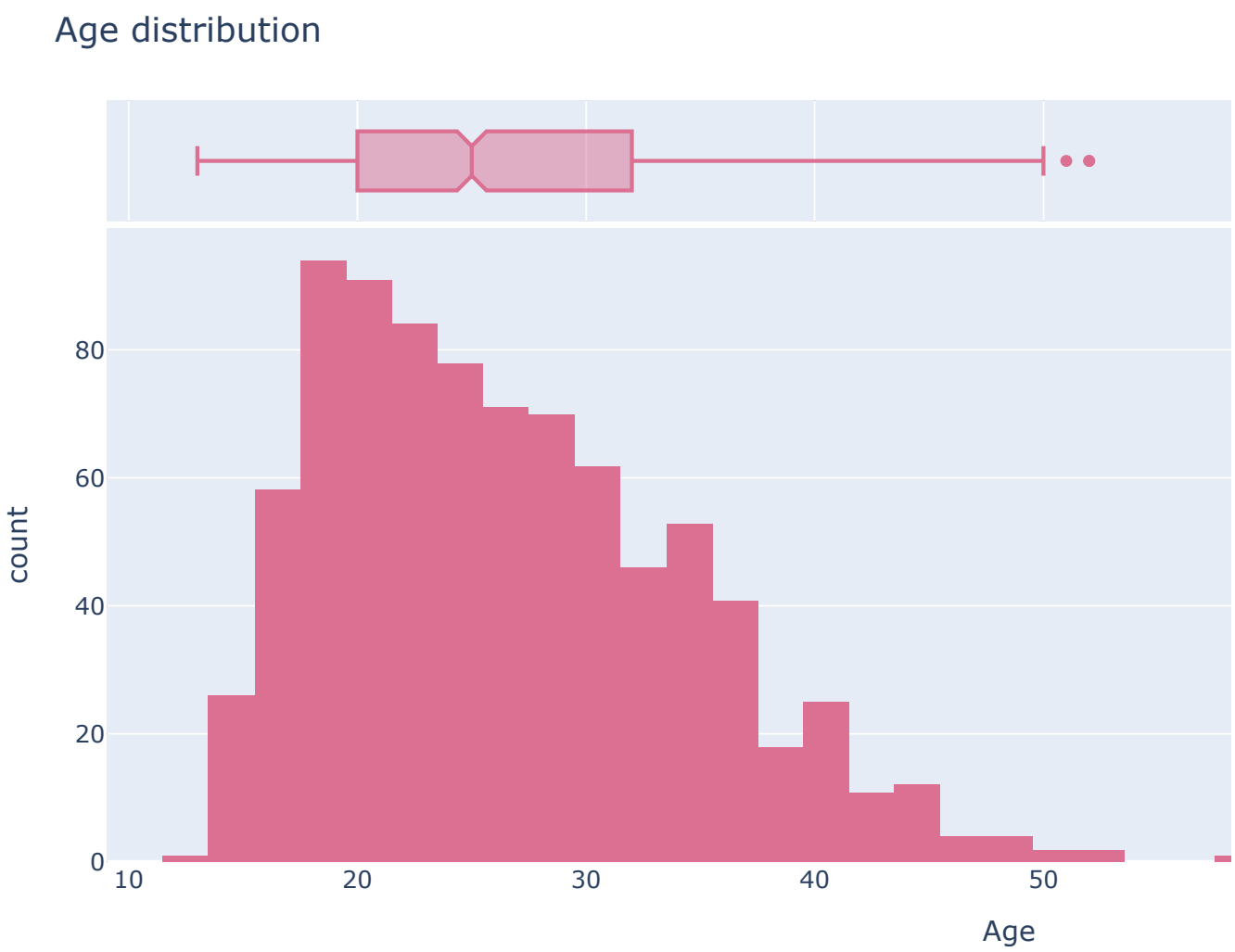
```
Index(['Dx:Cancer', 'Dx:HPV', 'Dx', 'STDs:HPV', 'total_tests', 'Biopsy',
       'Schiller'],
      dtype='object')
```

Top 7 Features Correlated With Dx:cancer



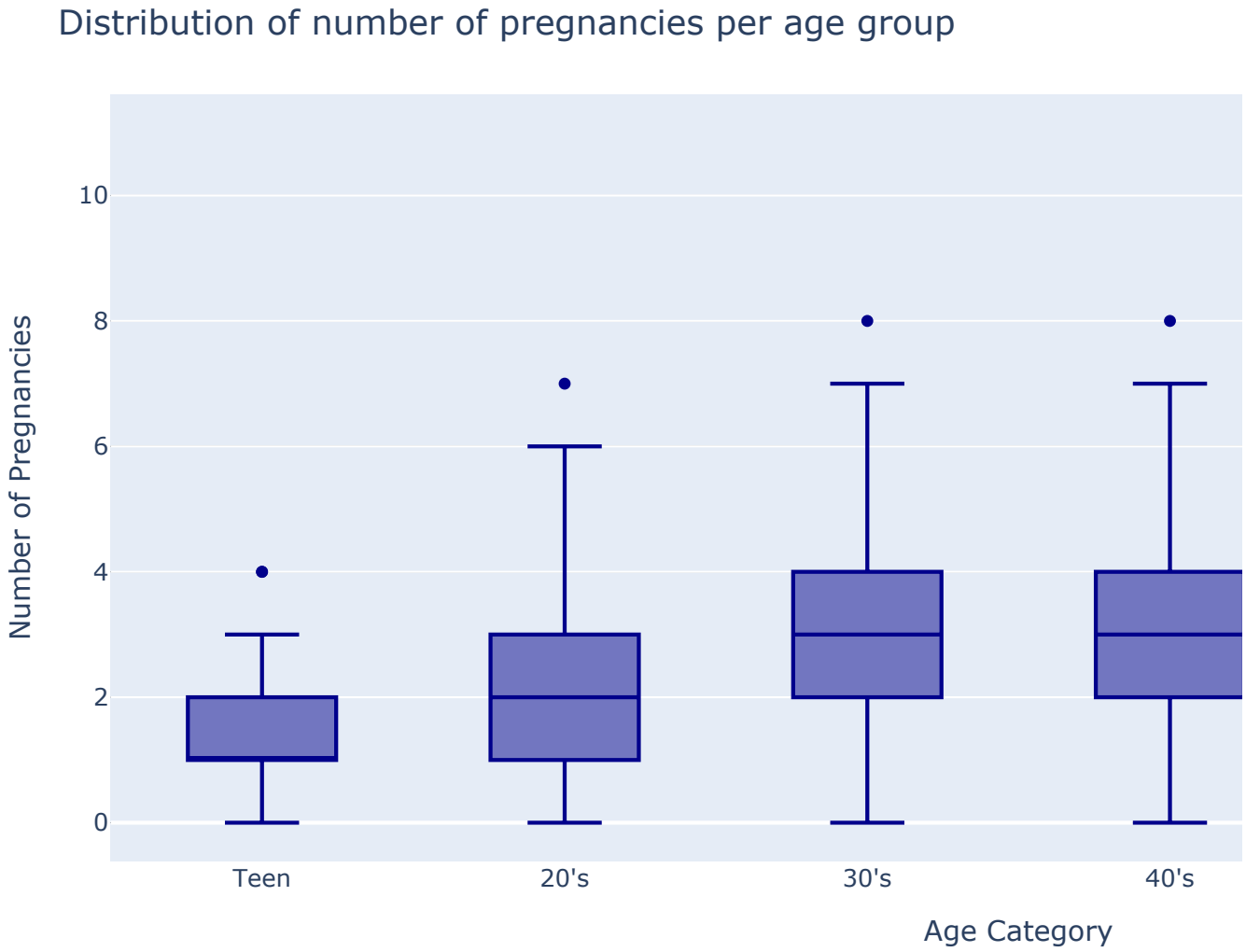
Distribution of age

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



Pregnancy Distribution by Age

```
#Pregancy Distribution by Age
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()
```



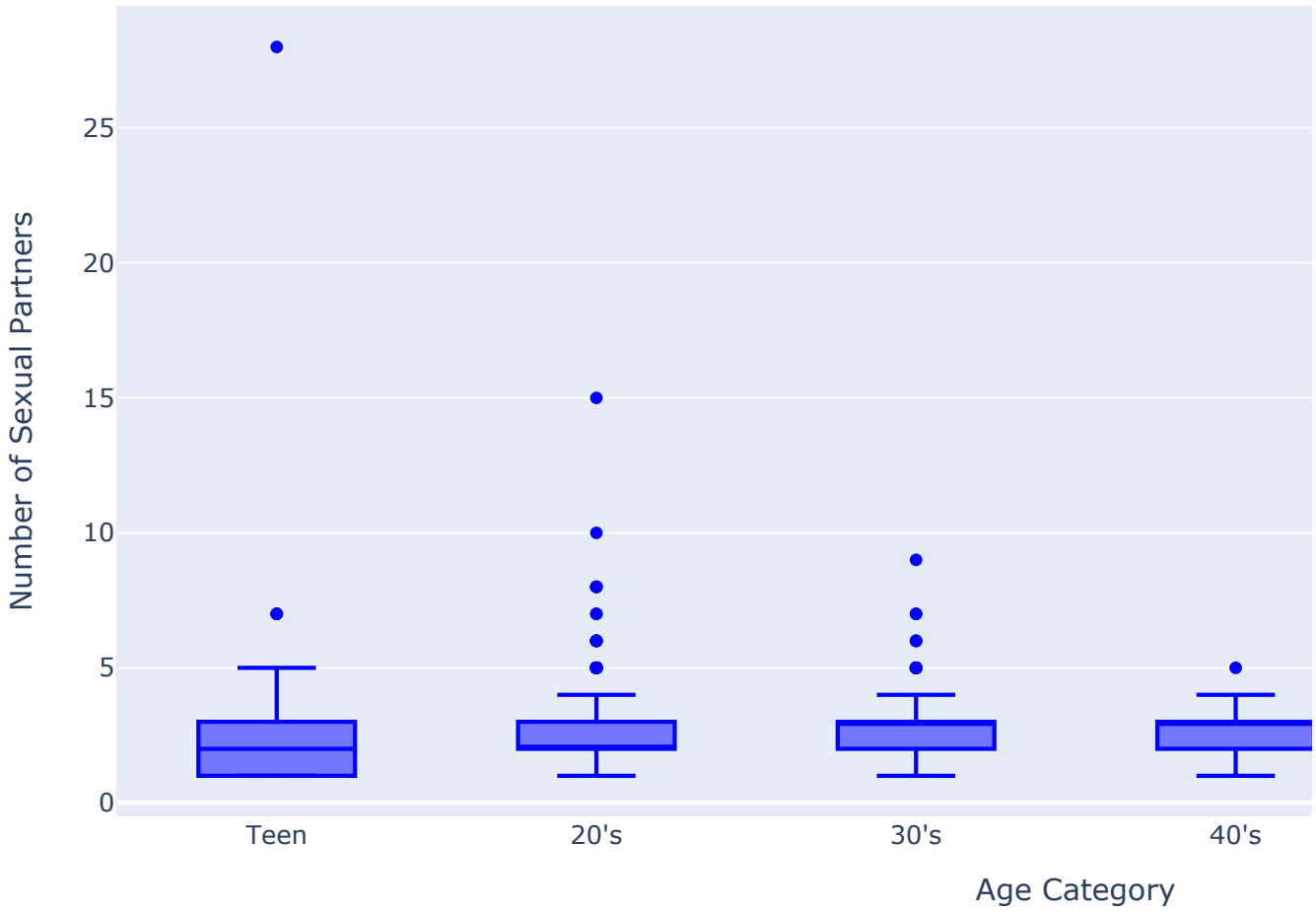
Mayo Risk Factors (May split further)

Mayo Clinic provides many risk factors for cervical cancer, including many sexual partners, earlier sexual activity, STIs, a wekeaned immune system, smoking, and the exposure to miscarriage preventian drug DES.

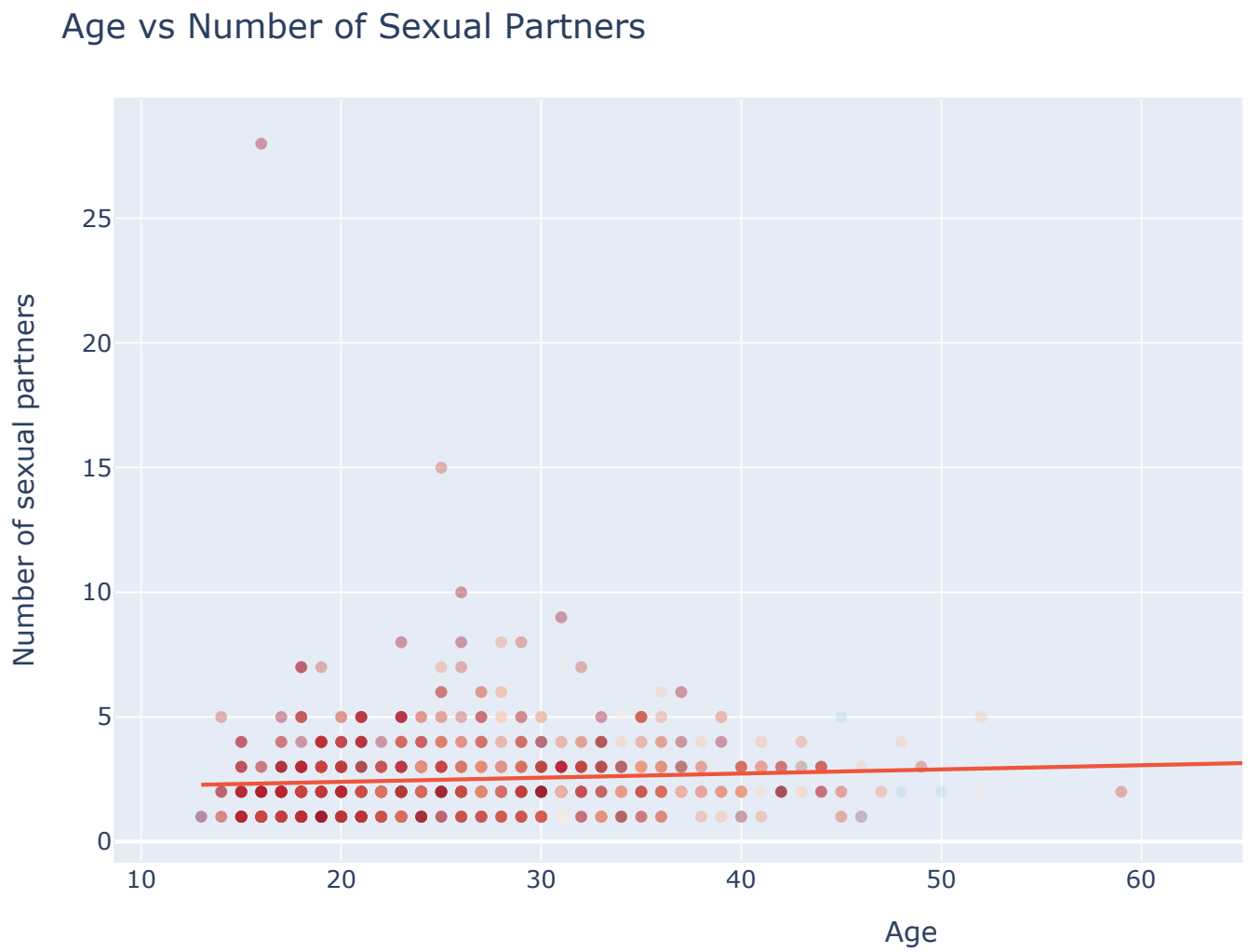
We can see from the following visualizations that number of sexual partners remain fairly consistent across age ranges. We can also see a very low correlation between number of sexual partners and any relevant diagnoses.

We see a very high correlation between HPV and Cancer, but a low correlation between CIN and HPV. This could be due to the incompletenss of this dataset, that we will try to balance alter.

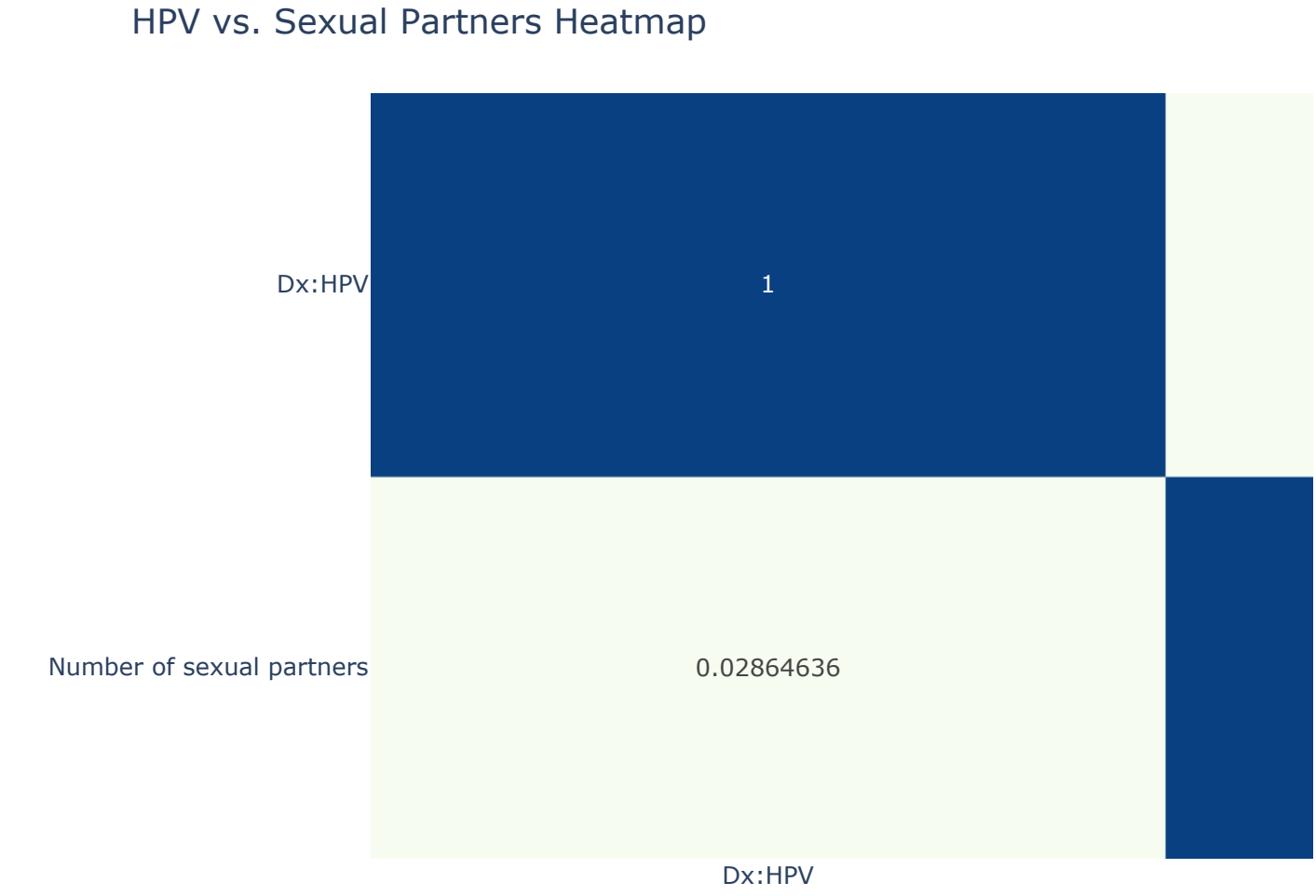
```
# Mayo Risk factors - sexual partners, sexual activity, STIs, immune system, smok

label = 'age_cat'
# Plotting a box plot to visualize the distribution of the number of sexual partn
# The data is sorted by age and plotted with outliers, using a blue color for the
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"])
# Updating axis titles for better readability and clarity.
age_num_sex_partners.update_xaxes(title="Age Category")
age_num_sex_partners.update_yaxes(title="Number of Sexual Partners")
# Updating the layout to add a title to the plot.
age_num_sex_partners.update_layout(title="Distribution of number of sexual partne
# Displaying the plot.
age_num_sex_partners.show()
```


Distribution of number of sexual partners per age group
Number of Sexual Partners
0
5
10
15
20
25
Teen
20's
30's
40's
Age Category

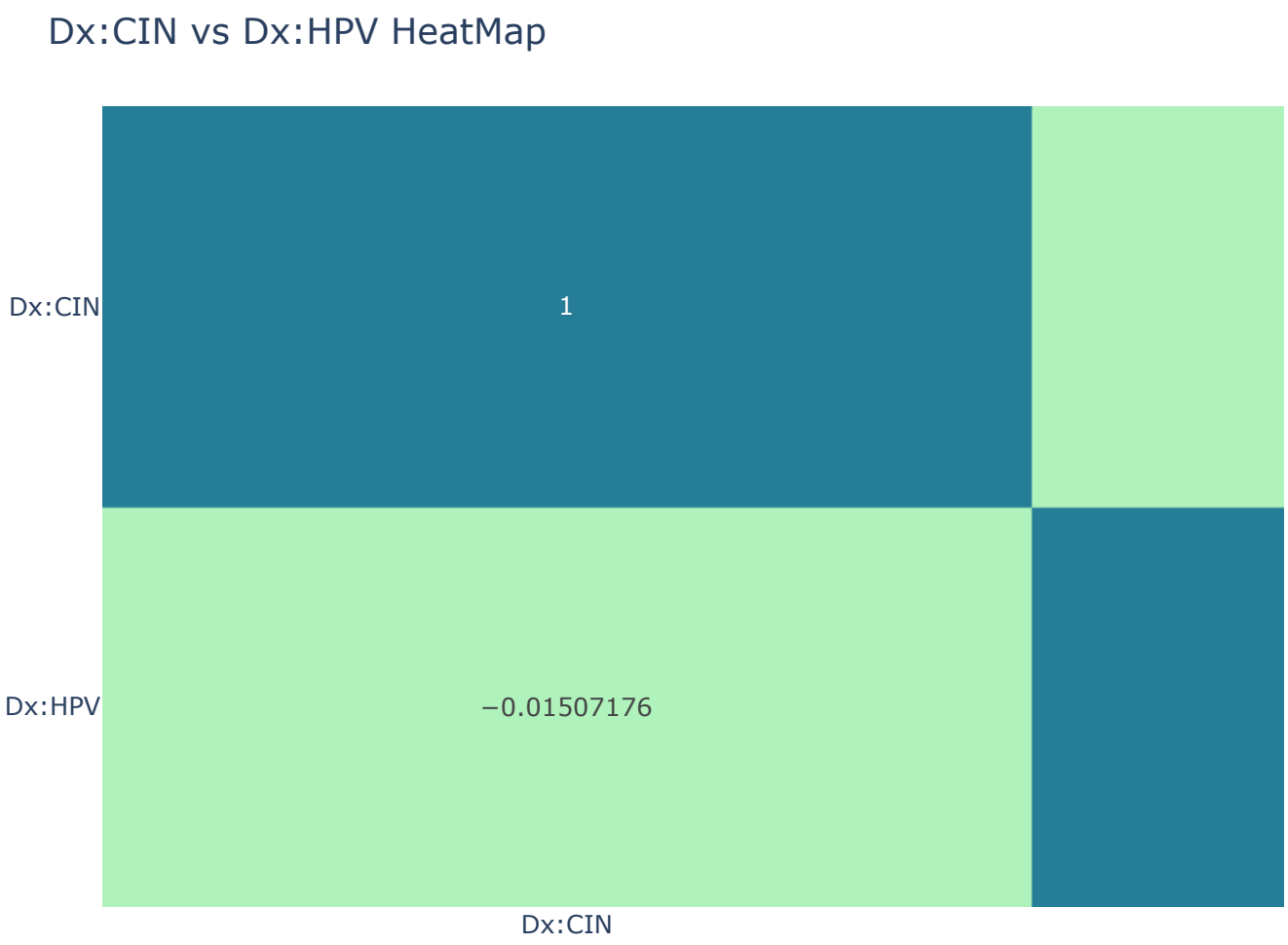
```
# Creating a scatter plot to visualize the relationship between age and number of
# The plot includes a trend line (ordinary least squares - OLS) to indicate the g
# Opacity is set to 0.4 to handle overplotting, and color represents the number o
age_num_sex_partners = px.scatter(risk_factor_df, x="Age",
                                  y="Number of sexual partners",
                                  trendline="ols",
                                  opacity=0.4,
                                  color="Num of pregnancies",
                                  color_continuous_scale="rdbu")
# Updating the layout to add a title to the plot.
age_num_sex_partners.update_layout(title="Age vs Number of Sexual Partners")
# Displaying the plot.
age_num_sex_partners.show()
```



```
# Selecting columns related to diagnoses and number of sexual partners for correl
diagnoses_num_partner_compare_cols = [label, 'Dx:HPV', "Number of sexual partners
# Calculating the correlation matrix for the selected columns.
corr_matrix = risk_factor_df[diagnoses_num_partner_compare_cols].corr(numeric_onl
# Printing the correlation matrix.
# print(corr_matrix)
# Visualizing the correlation matrix using a heatmap with text annotations for co
diagnoses_num_partner_heatmap = px.imshow(corr_matrix,
                                          aspect="auto",
                                          color_continuous_scale="gnbu",
                                          text_auto=True)
diagnoses_num_partner_heatmap.update_layout(title='HPV vs. Sexual Partners Heatma
# Displaying the heatmap.
diagnoses_num_partner_heatmap.show()
```



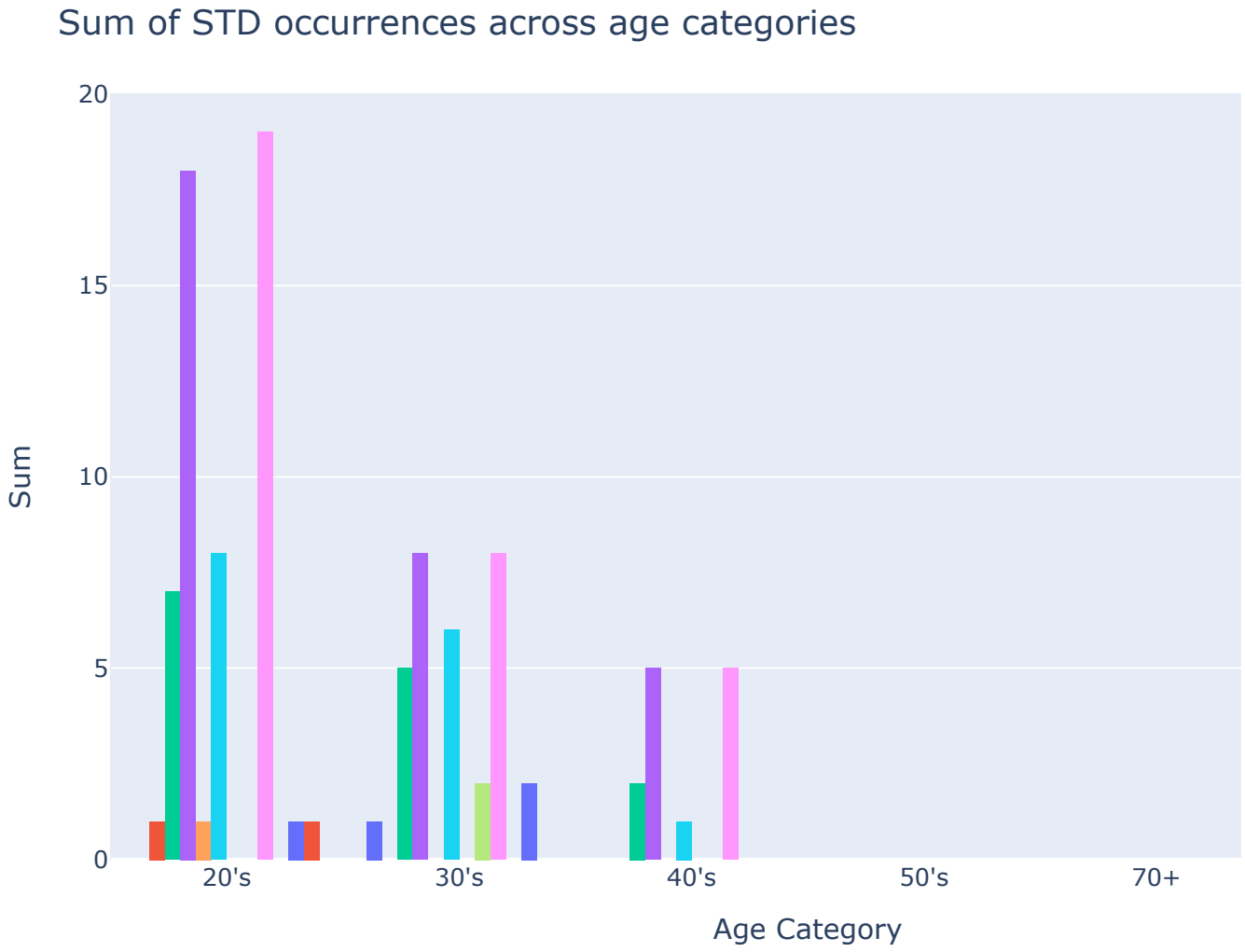
```
# Selecting columns related only to diagnoses for correlation analysis.
diagnoses_cols = [label, 'Dx:CIN', 'Dx:HPV']
# Calculating the correlation matrix for the selected diagnoses columns.
diagnoses_corr_matrix = risk_factor_df[diagnoses_cols].corr(numeric_only=True)
# Visualizing the correlation matrix using a heatmap with teal-green color scale
diagnoses_heatmap = px.imshow(diagnoses_corr_matrix, aspect="auto", color_continu
# Displaying the heatmap.
diagnoses_heatmap.update_layout(title="Dx:CIN vs Dx:HPV HeatMap")
diagnoses_heatmap.show()
```



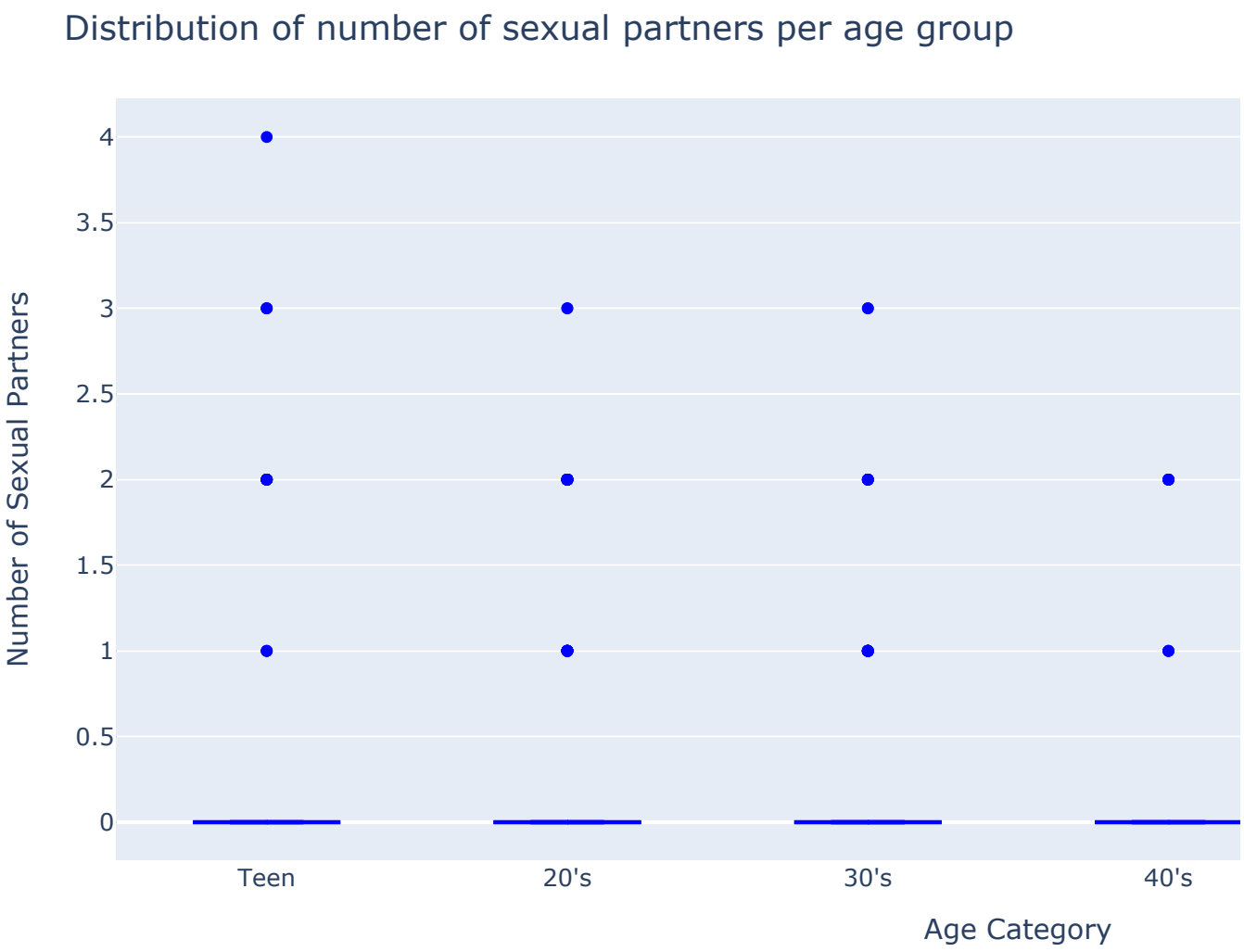
STDs

```
#data processing - to provide access of std_cols in STD graphs - do we want to ad
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'}
```

```
#create historgram to understand Sum of STD occurences across age
fig = px.histogram(std_agg, x = "age_cat",
                   y = list(std_cols),
                   barmode = "group",
                   histfunc = "sum")
fig.update_layout(title = "Sum of STD occurrences across age categories")
fig.update_xaxes(title = "Age Category")
fig.update_yaxes(title = "Sum")
#show plot
fig.show()
#some discrepencies exist
```

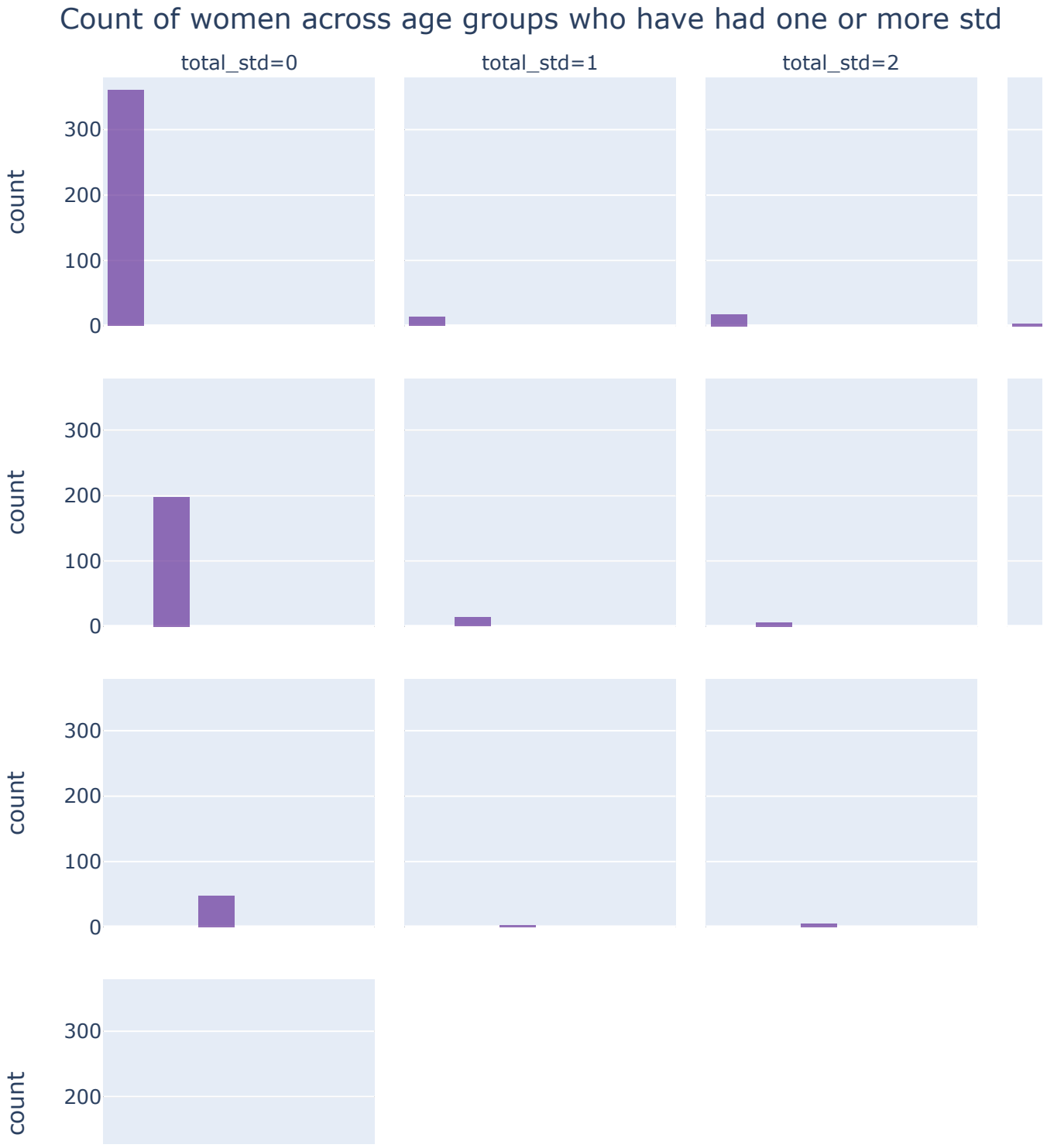


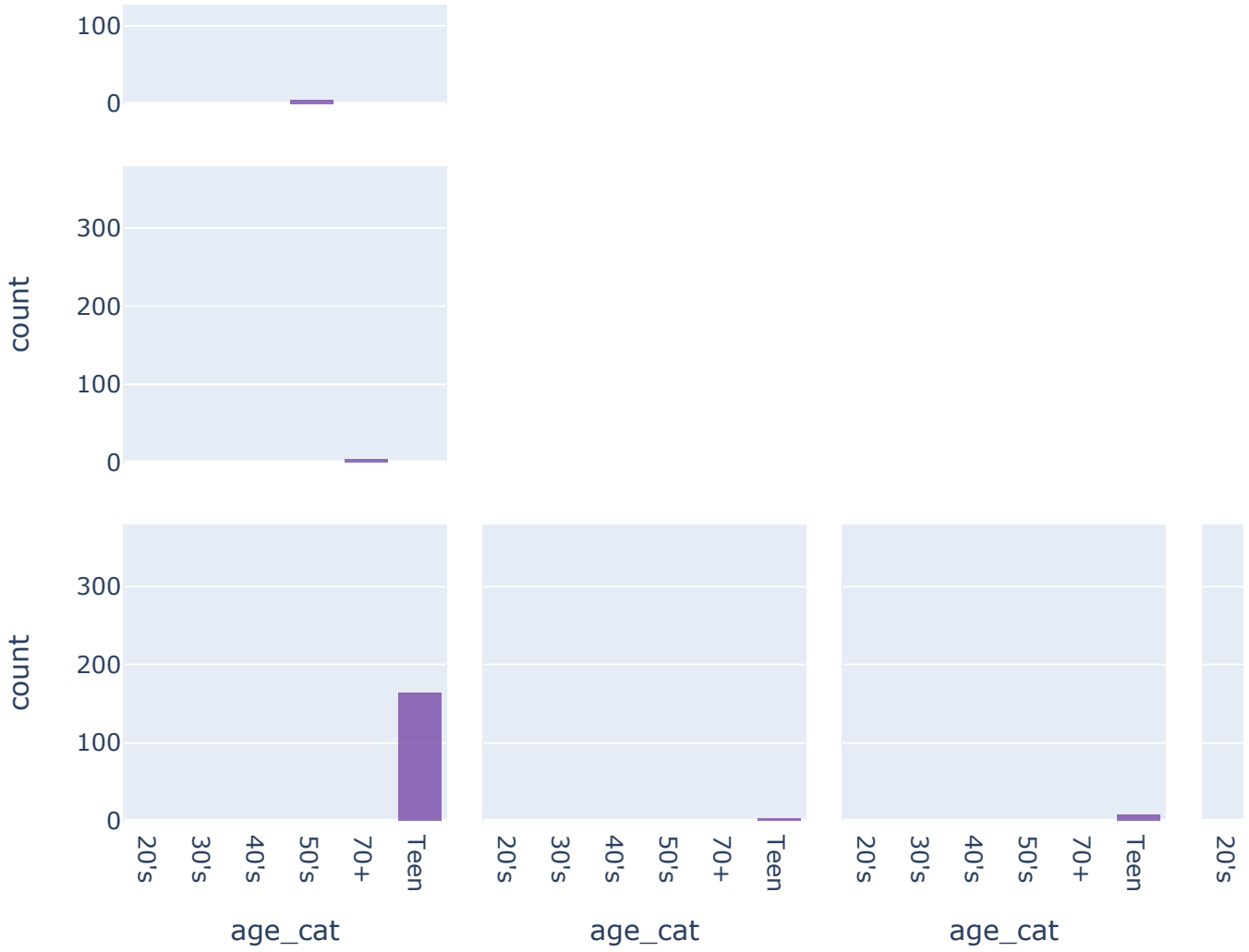
```
#create boxplot to understand Distribution of number of sexual partners per age g
age_num_sex_partners = px.box(risk_factor_df.sort_values(by="Age",ascending=True)
                      color_discrete_sequence=["blue"], points="outliers",
                      category_orders=["Teenager", "Twenties", "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
#show plot
age_num_sex_partners.show()
```



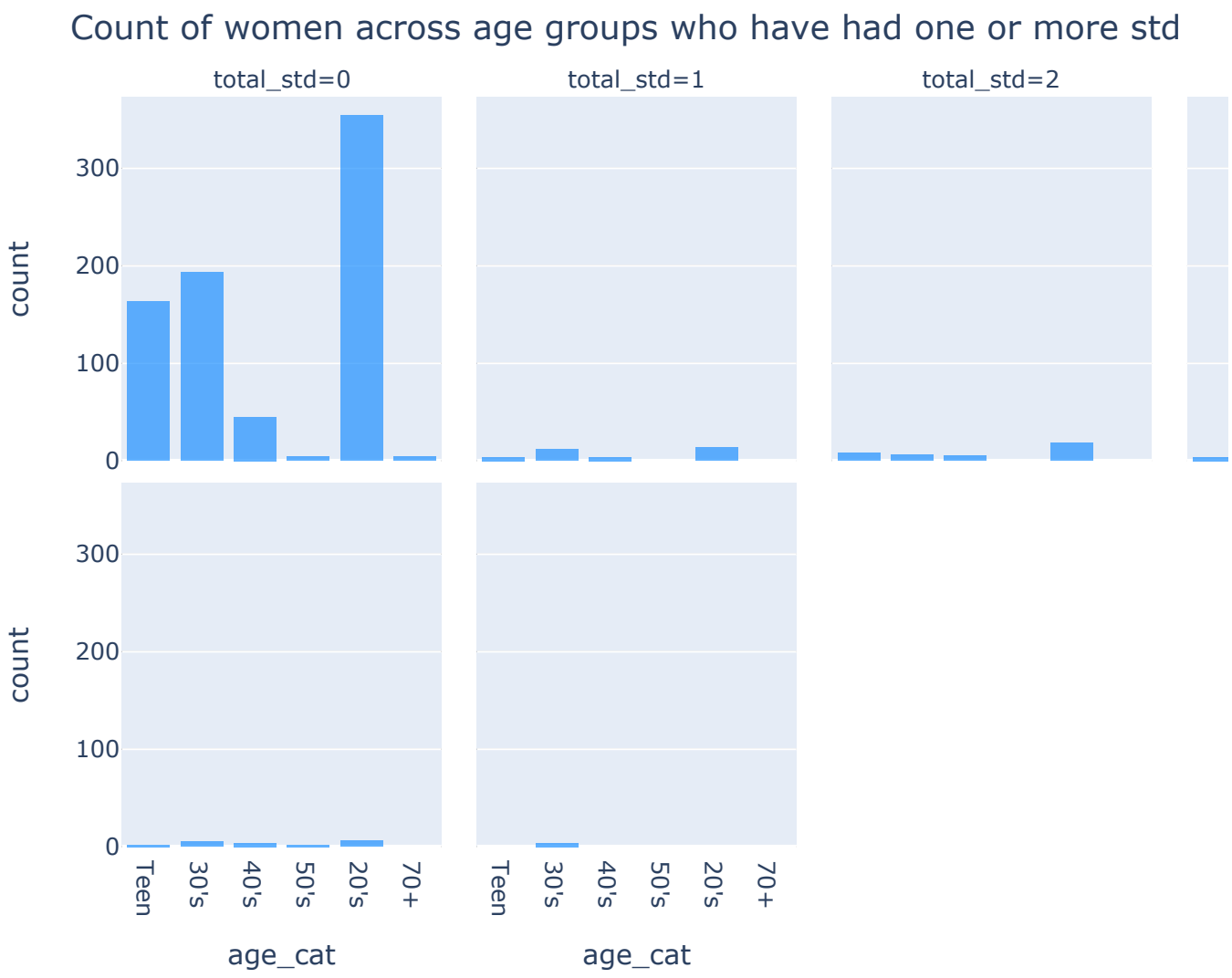

```
#create histogram to understand Count of women across age groups who have had one
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_st
                   x="age_cat",
                   facet_col="total_std",
```

```
                    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.update_layout(height=1200)
#show plot
fig.show()
```




100
0
300
200
100
0
count
300
200
100
0
count
20's
30's
40's
50's
70+
Teen
age_cat
20's
30's
40's
50's
70+
Teen
age_cat
20's
30's
40's
50's
70+
Teen
20's
age_cat

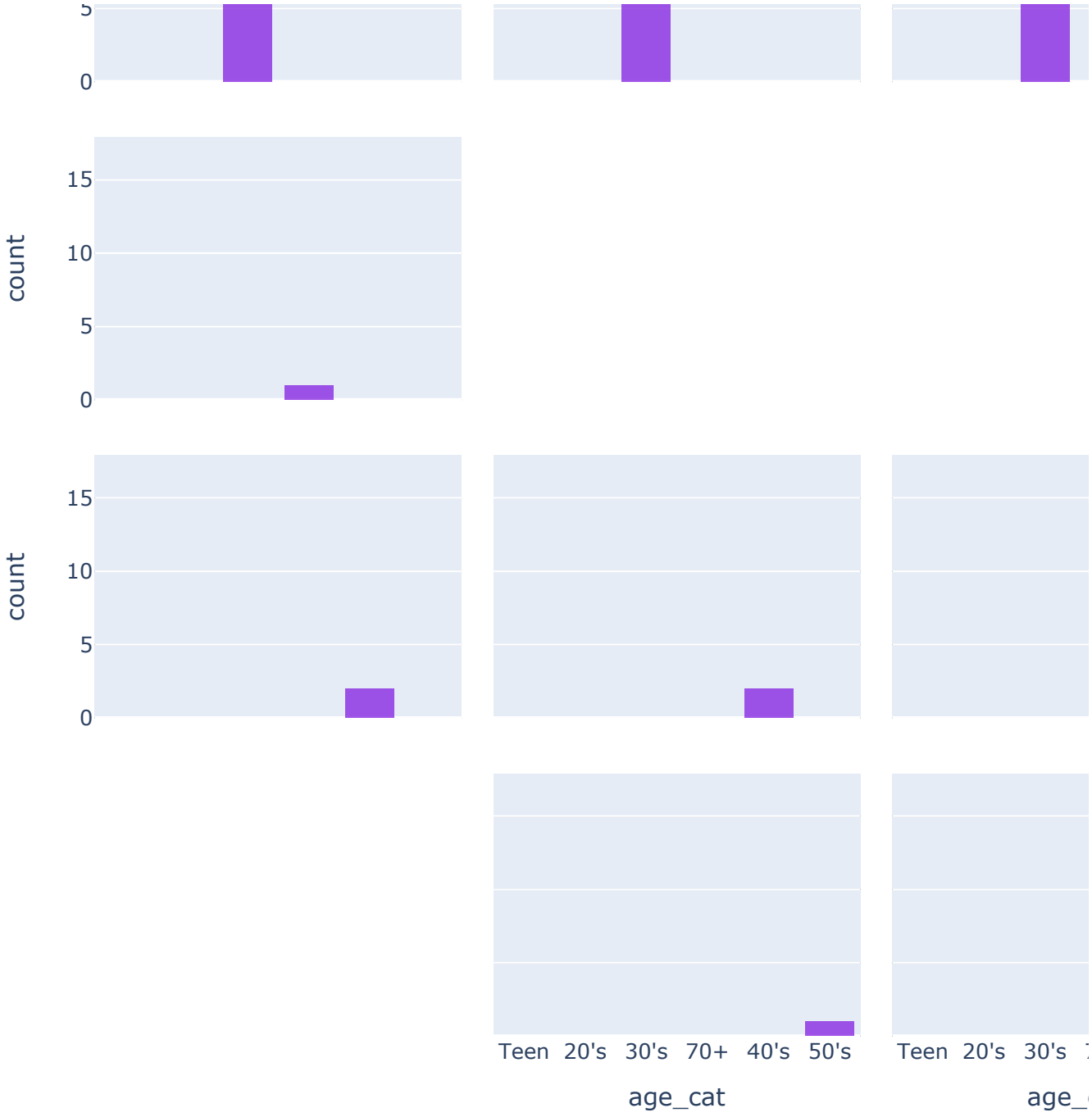
```
#create histogram to understand Count of women across age groups who have had one
fig = px.histogram(risk_factor_df.query("total_std>=0").sort_values(by=["total_st
                   x="age_cat",
                   facet_col="total_std",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["dodgerblue"],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

fig.show()
```



Tests used

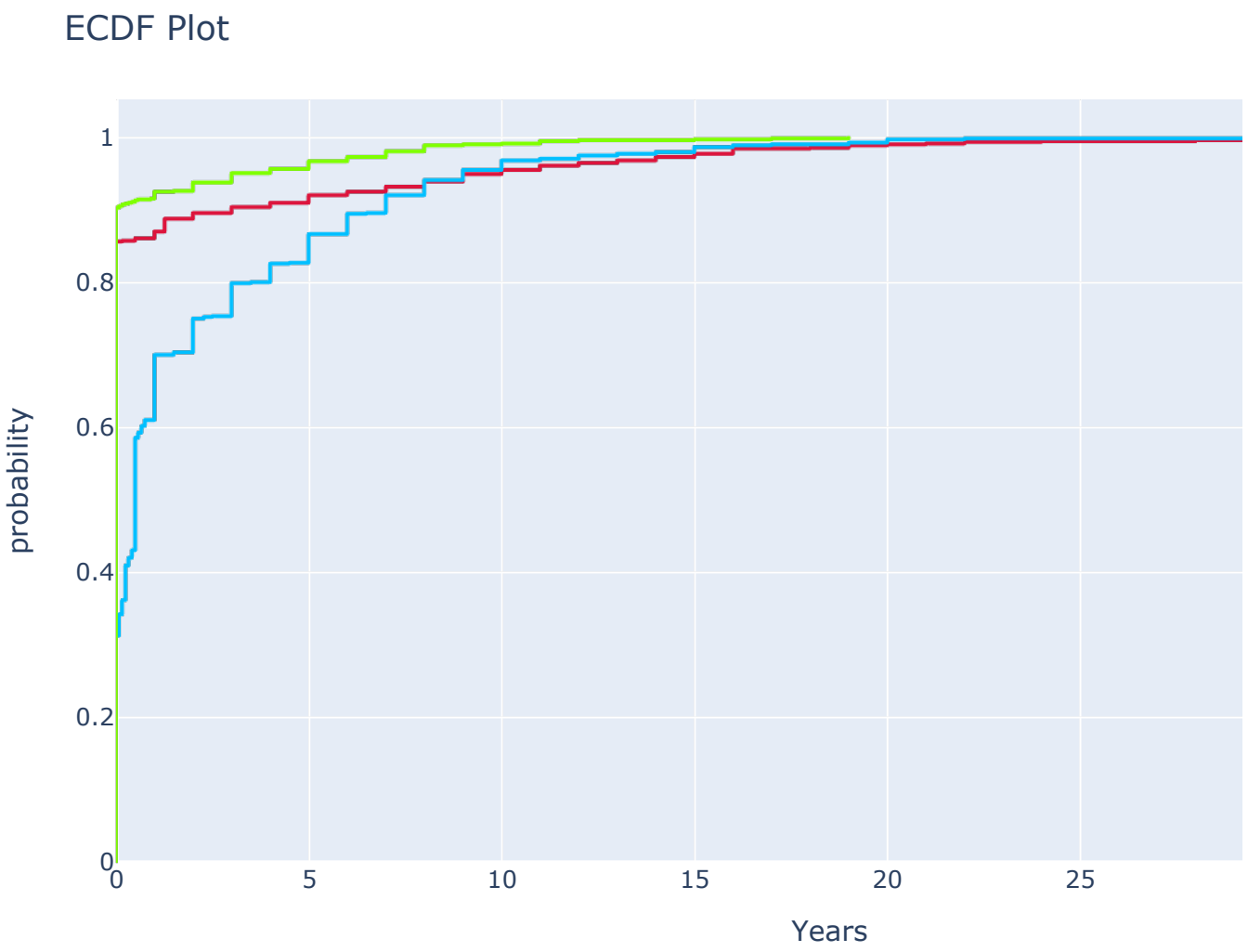
Here we observe the number of tests done by patients to determine if they have Cerivcal Cancer / HPV.

```
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.update_layout(height=1200)
fig.show()
```


5
0
15
10
5
0
count
15
10
5
0
count
Teen
20's
30's
70+
40's
50's
age_cat
Teen
20's
30's
age_

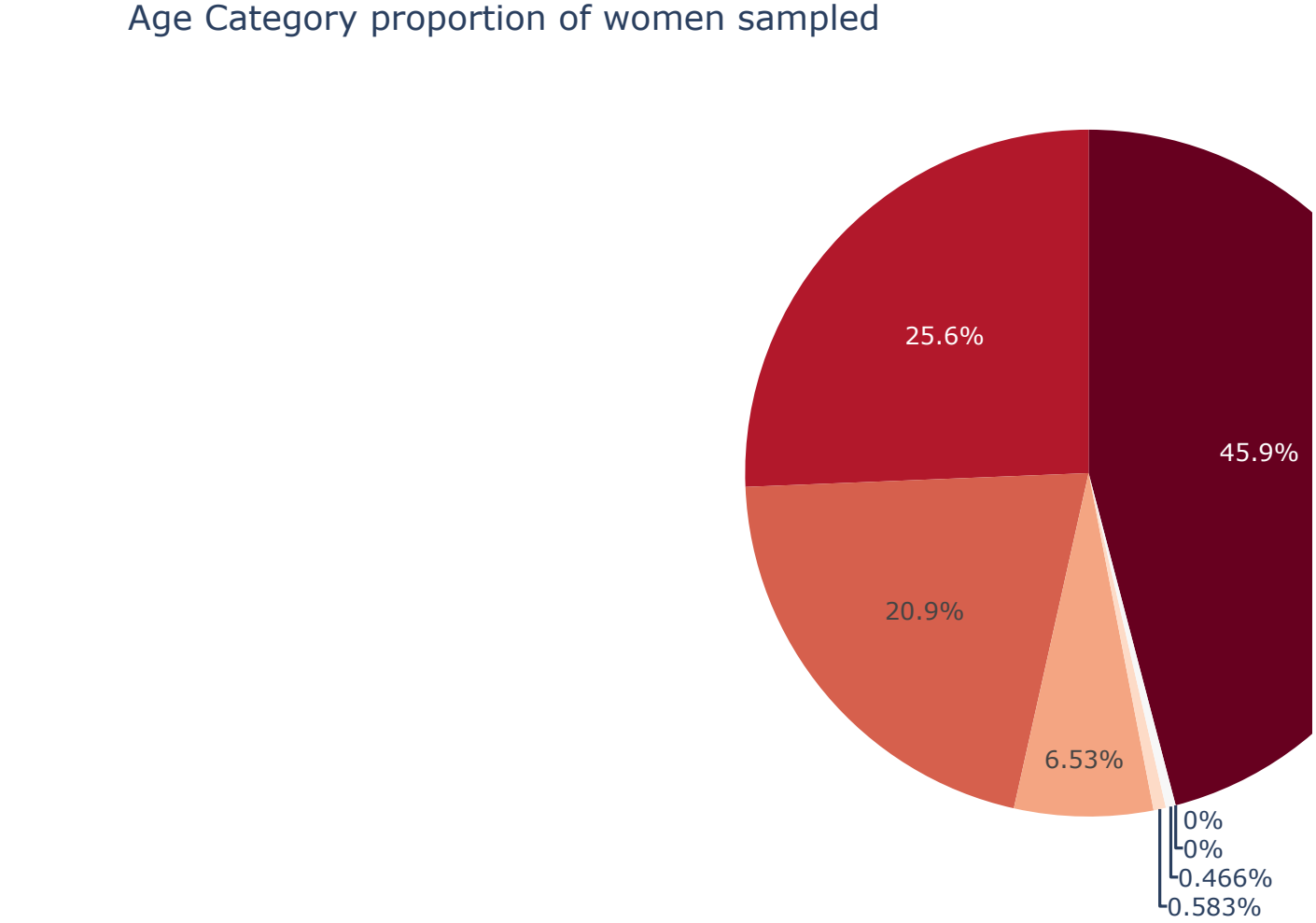
```
fig = px.histogram(risk_factor_df.query("total_tests>0").sort_values(by=["total_t
                   x="age_cat",
                   facet_col="total_tests",
                   facet_row="Dx:HPV",
                   color_discrete_sequence=["coral"],
                   opacity=0.8)
fig.update_layout(title="Count 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()
```




```
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(

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

```
fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'domain'}, {'type': 'domain'
```

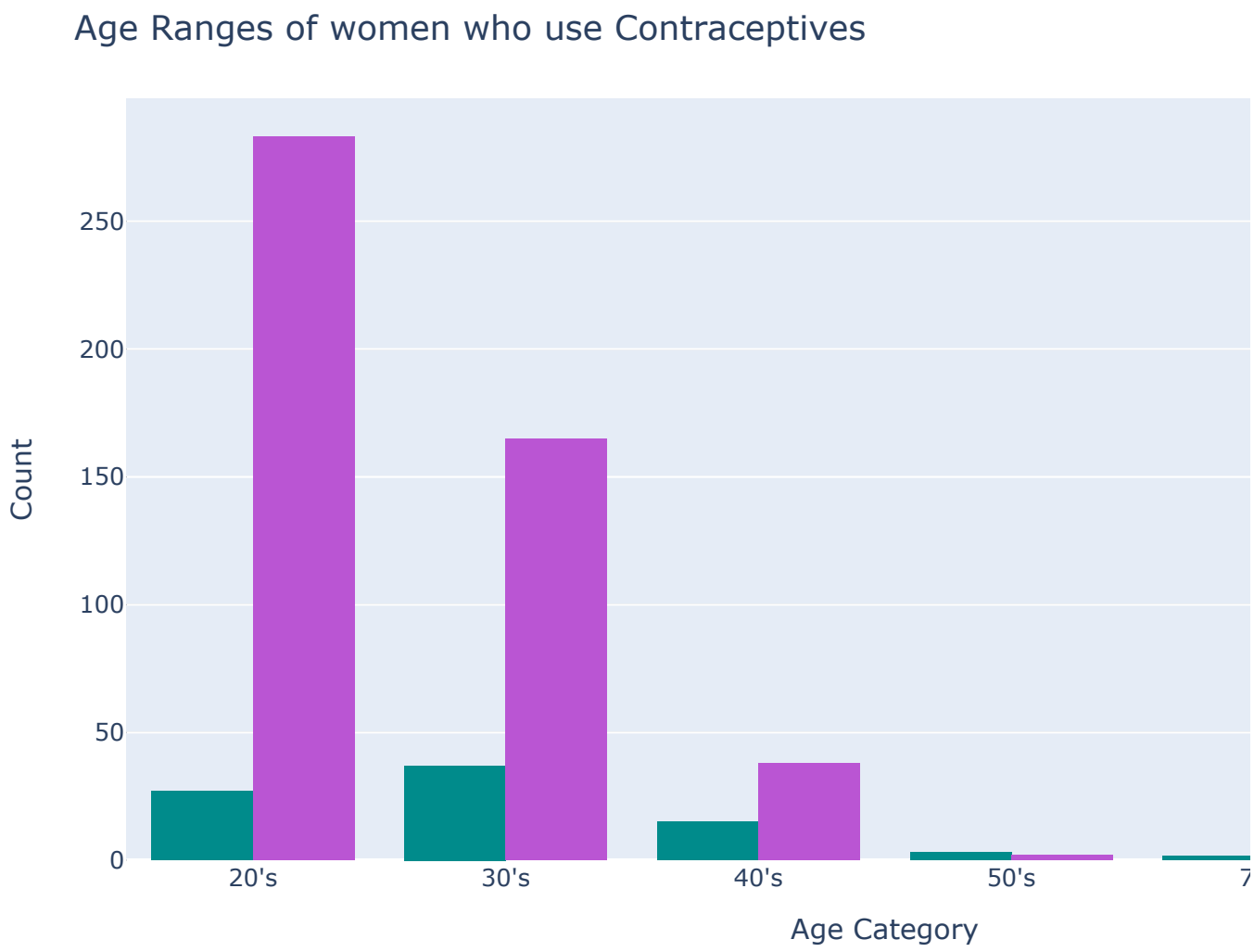
```
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'},{'type':'domain'
                    subplot_titles=["Cancer", "HPV"])
fig.add_trace(go.Pie(labels=risk_factor_df["age_cat"],
                     values=risk_factor_df[label],
                     name="Cancer", marker_colors=px.colors.sequential.RdBu),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

Proportion of women across age categories with a diagnosis of Cancer

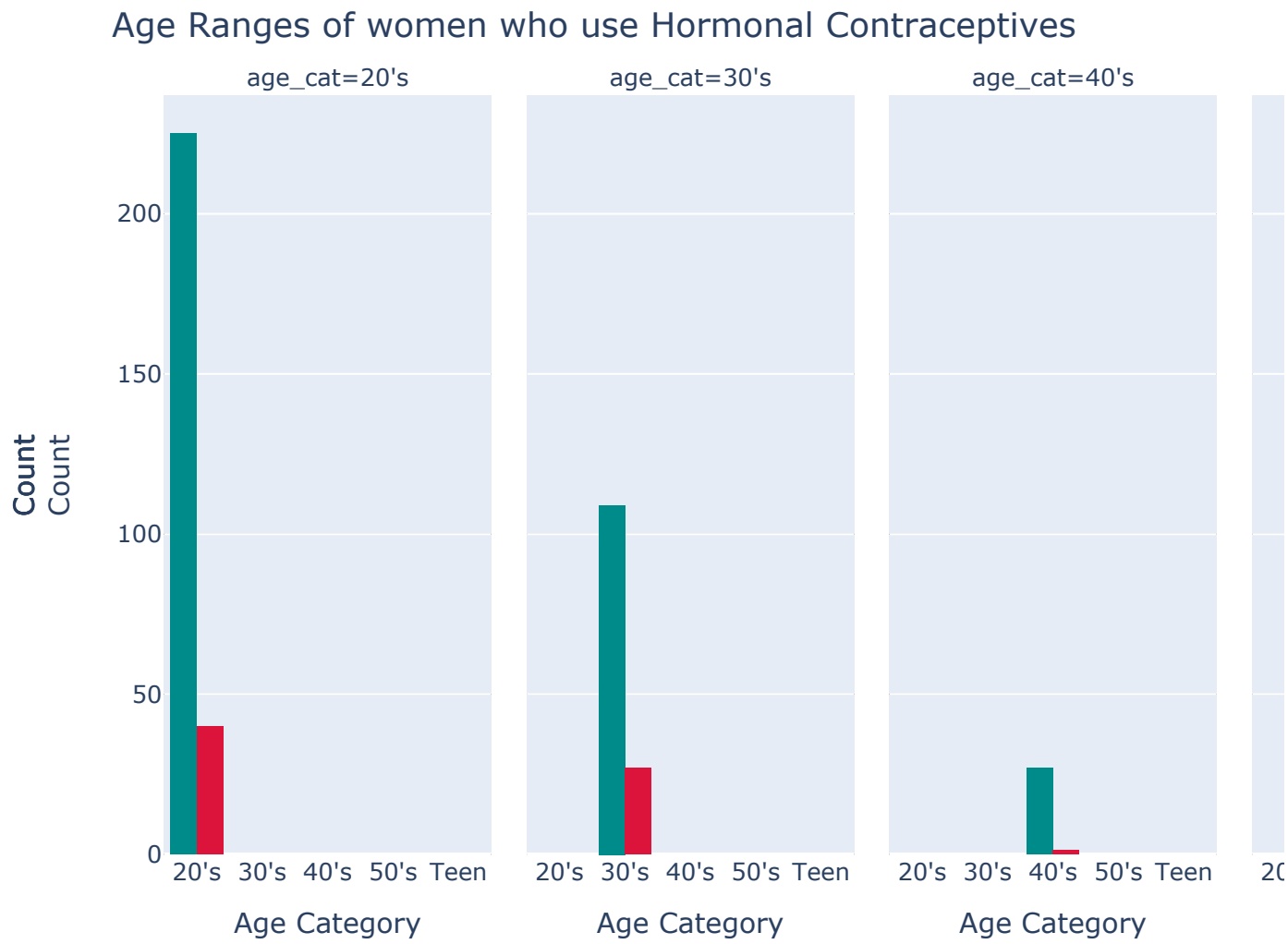
Cancer

Contraceptive

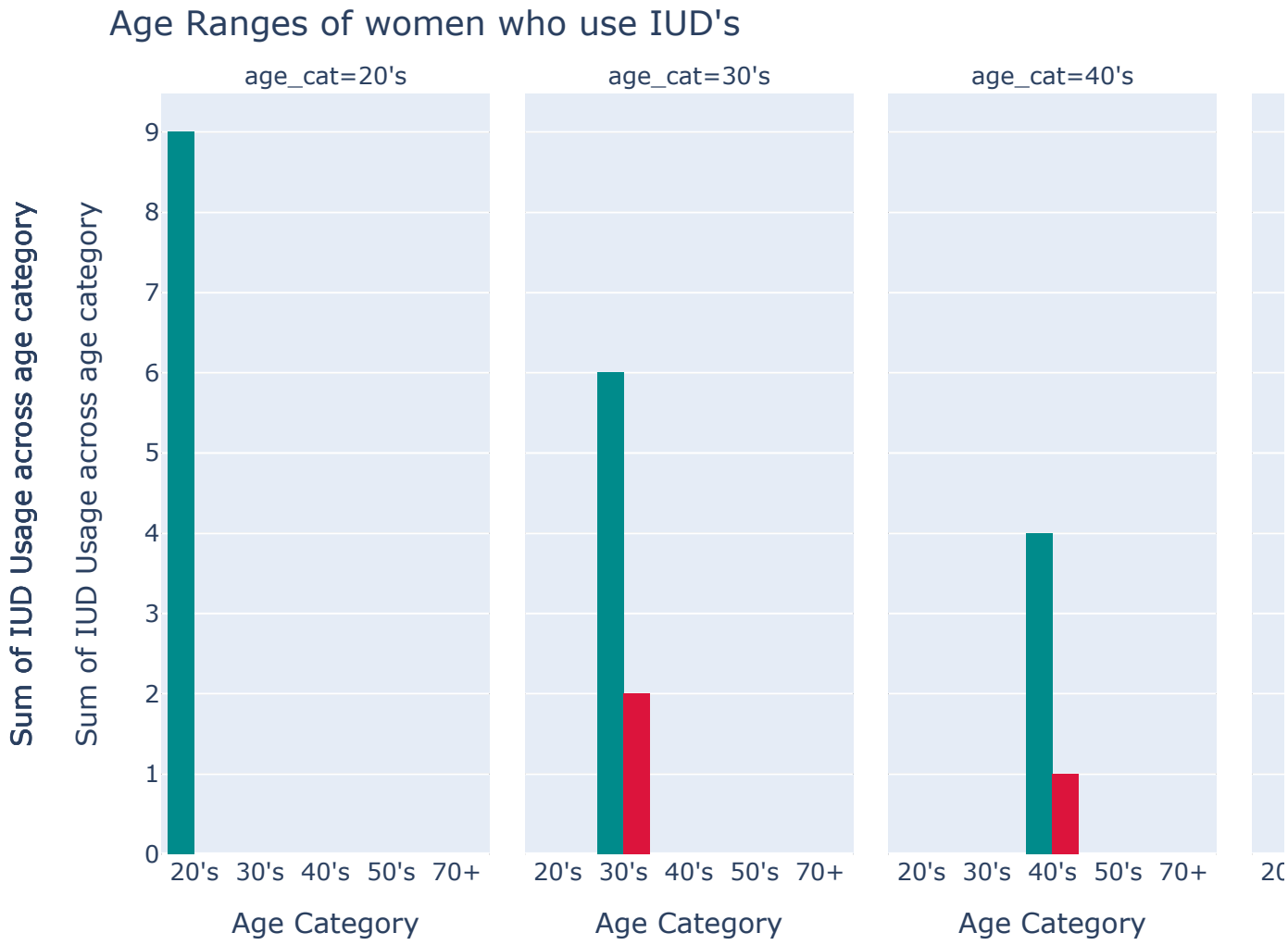
```
df_hormonal_compariosn = risk_factor_df.groupby(["age_cat"], as_index=False)[["IU
fig = px.histogram(df_hormonal_compariosn, x="age_cat", y=["IUD", "Hormonal Contr
                   , color_discrete_sequence=["darkcyan", "mediumorchid"])

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

fig.show()
```



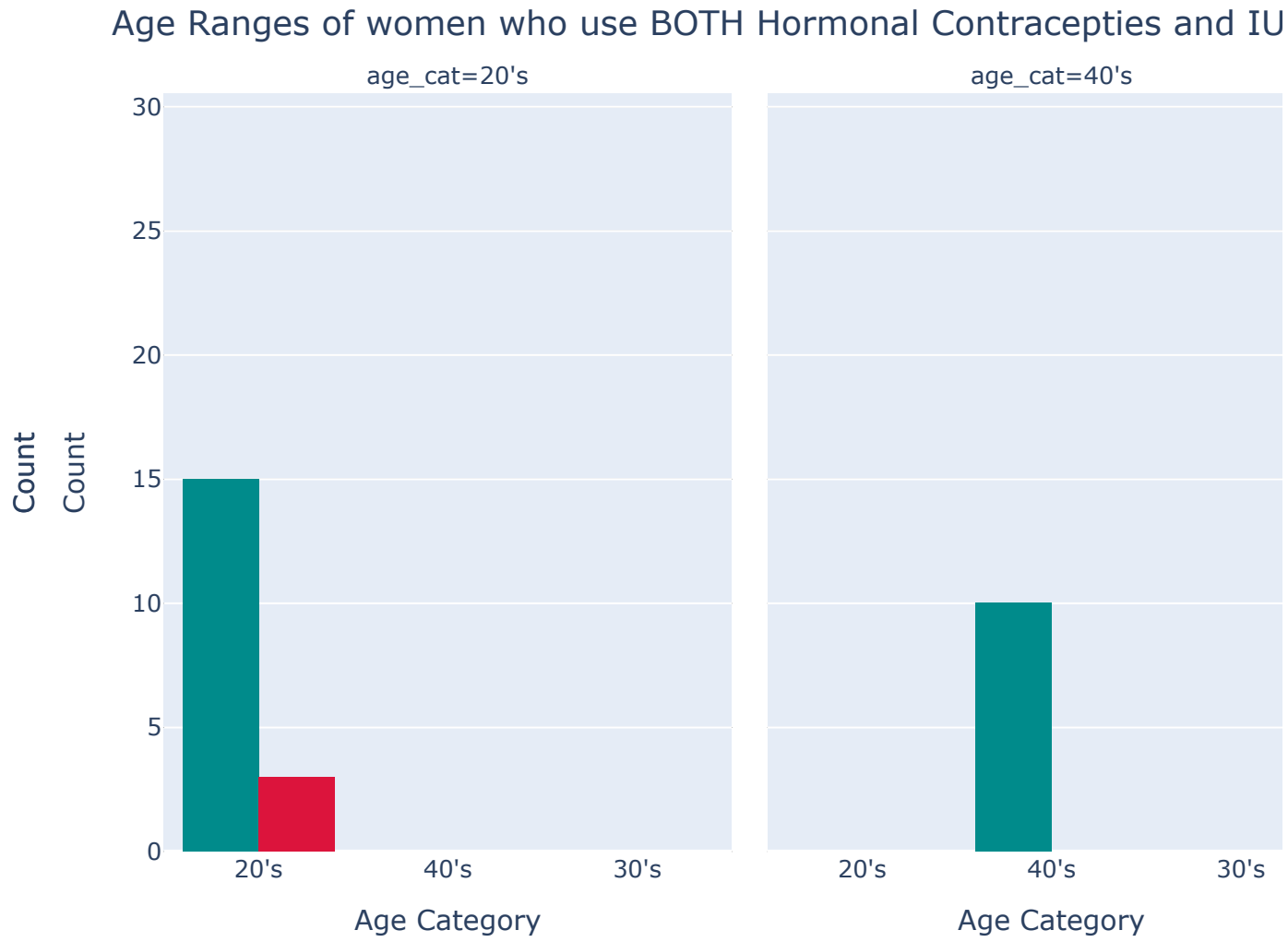
```
df_hormonal_contraceptives = risk_factor_df[
    (risk_factor_df["Hormonal Contraceptives"] == 1) & (risk_factor_df["IUD"] ==
df_hormonal_contraceptives = df_hormonal_contraceptives.sort_values(by=["Smokes",
fig = px.histogram(df_hormonal_contraceptives, x="age_cat", color="Smokes", barmo
                   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()
```



```
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()
```

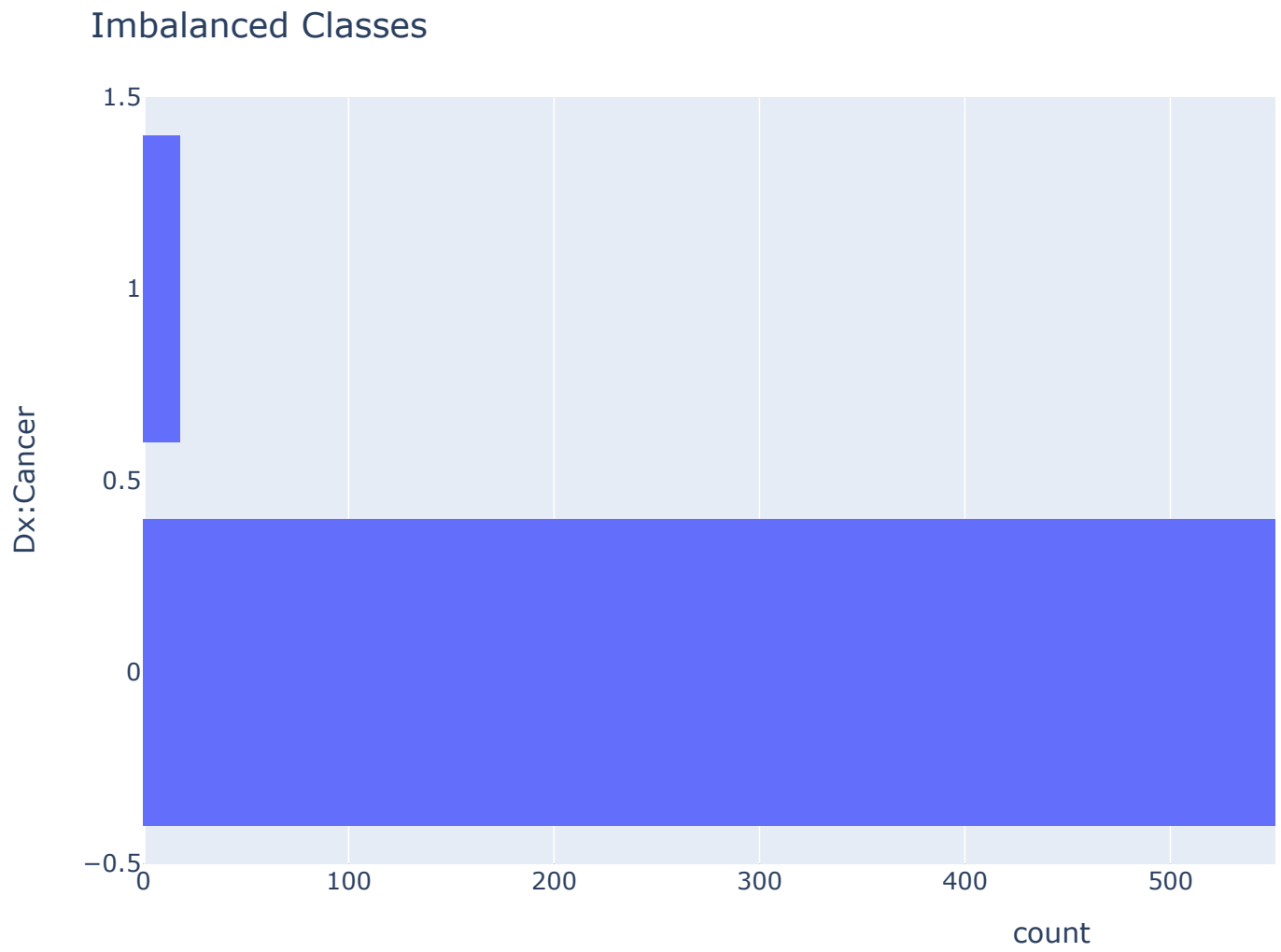


ADASYN

```
# 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
```

Checkpoint: Display Imbalanced Classes

```
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()
```



Here we use ADASYN to balance the dataset

```
# unbalanced_risk_factor_df = risk_factor_df
# 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)
```

Save ADASYN dataset

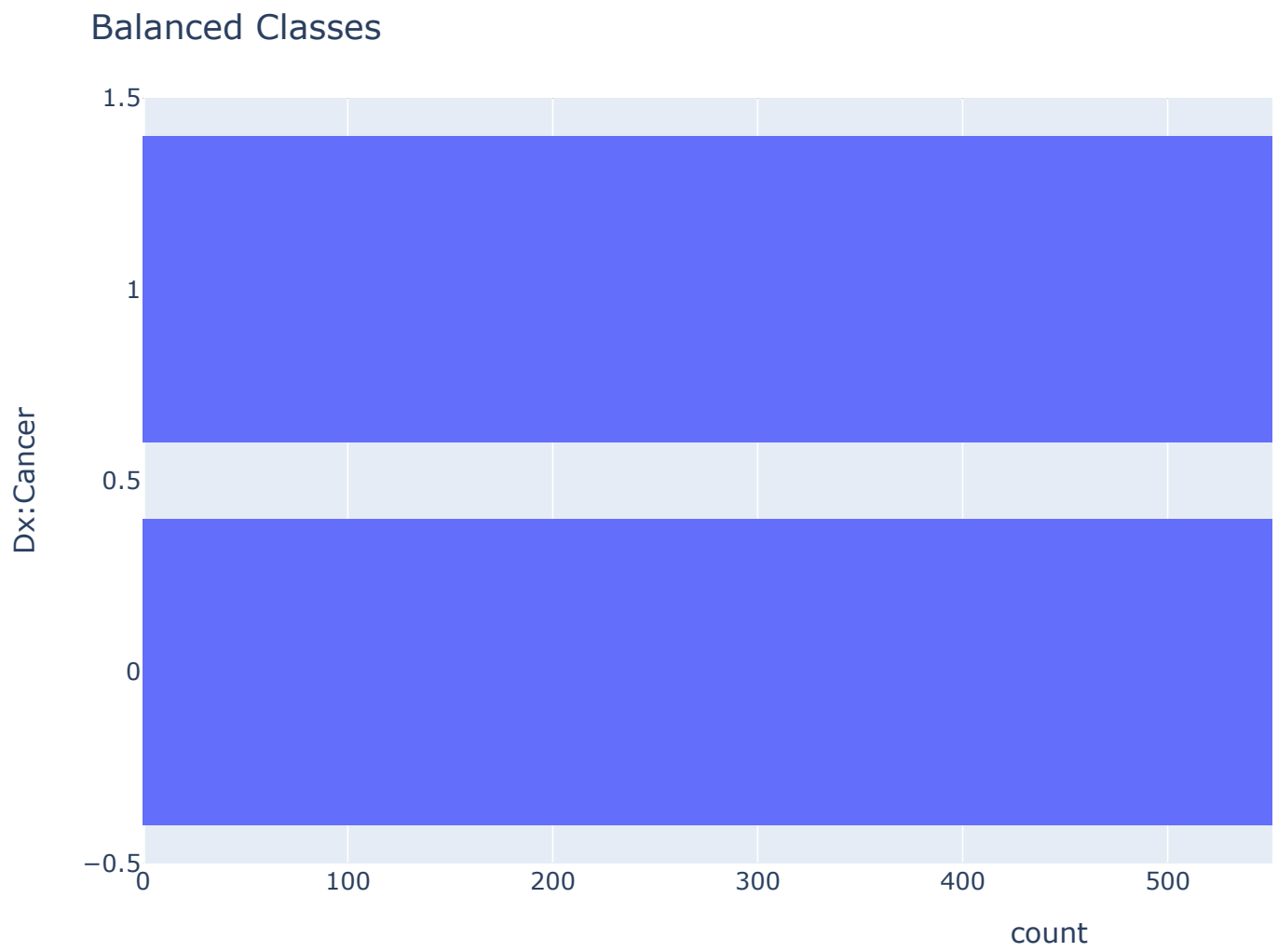
```
# risk_factor_df["age_cat"] = risk_factor_df["Age"].apply(age_cat)
# save_data(risk_factor_df, '/content/drive/My Drive/DL4H_Sp24_Final_Project/bala
```

Checkpoint: Load Balanced data

```
try:
  data_dir = '/content/drive/My Drive/DL4H_Sp24_Final_Project/balanced_risk_factc
  risk_factor_df = load_data(data_dir)
  have_access = True
except:
  have_access = False
  data_dir = 'balanced_factors_cervical_cancer.csv'
  gdown.download('https://drive.google.com/file/d/1-4EiqdYBbae16azaiZry7Gwhpq7Xhw
  risk_factor_df = load_data(data_dir)
  print("No access, Used Gdown!")
risk_factor_df
label="Dx:Cancer"
```

```
No access, Used Gdown!
```

```
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()
```



Checkpoint: Train Test Split

Stratifying the data on Age Category

Unbalanced

```
unbalanced_train_set = None
unbalanced_test_set = None
#Stratify the data
# 20% in test and 80% in train
unbalanced_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state
for train_idx, test_idx in unbalanced_split.split(unbalanced_risk_factor_df, unba
    unbalanced_train_set = unbalanced_risk_factor_df.loc[train_idx]
    unbalanced_test_set = unbalanced_risk_factor_df.loc[test_idx]
unbalanced_cols_to_drop = ["age_cat","total_std","total_tests"]
for set_ in (unbalanced_train_set, unbalanced_test_set):
    for col in unbalanced_cols_to_drop:
        set_.drop(col, axis=1, inplace=True)


unbalanced_X_train = unbalanced_train_set.drop(label, axis=1)
unbalanced_y_train = unbalanced_train_set[label].copy()

unbalanced_X_test = unbalanced_test_set.drop(label, axis=1)
unbalanced_y_test = unbalanced_test_set[label].copy()

unbalanced_X_test.reset_index(drop=True, inplace=True)
unbalanced_y_test.reset_index(drop=True, inplace=True)
unbalanced_X_train.reset_index(drop=True, inplace=True)
unbalanced_y_train.reset_index(drop=True, inplace=True)


print("unbalanced_X_test length: ", len(unbalanced_X_test))
print("unbalanced_X_train length: ", len(unbalanced_X_train))
print("unbalanced_Y_test length: ", len(unbalanced_y_test))
print("unbalanced_Y_train length: ", len(unbalanced_y_train))
```

```
unbalanced_X_test length:  172
unbalanced_X_train length:  686
unbalanced_Y_test length:  172
unbalanced_Y_train length:  686
```

Balanced

```
train_set = None
test_set = None
#Stratify the data
# 20% in test and 80% in train
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in split.split(risk_factor_df, risk_factor_df["age_cat"])
    train_set = risk_factor_df.loc[train_idx]
    test_set = risk_factor_df.loc[test_idx]
cols_to_drop = ["age_cat","total_std","total_tests"]
for set_ in (train_set, test_set):
    for col in cols_to_drop:
        set_.drop(col, axis=1, inplace=True)


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

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

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


print("X_test length: ", len(X_test))
print("X_train length: ", len(X_train))
print("Y_test length: ", len(y_test))
print("Y_train length: ", len(y_train))
```

```
X_test length:  336
X_train length:  1341
Y_test length:  336
Y_train length:  1341
```

Unclear where this comes into the code at the moment

```
# #without random var
# X_test.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/X_test.csv')


# #have not ran the rest of this block yet yet
# y_test.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/y_test.csv')
# X_train.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/X_train.csv')
# y_train.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/y_train.csv')

# #with random var
# #binary
# X_test.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/RX_test2.csv')
# y_test.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/Ry_test2.csv')
# X_train.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/RX_train2.csv')
# y_train.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/Ry_train2.csv')
# #continous
# X_test.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/RX_test.csv')
# y_test.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/Ry_test.csv')
# X_train.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/RX_train.csv')
# y_train.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/Ry_train.csv')
```

Checkpoint: Model Setup

We will be comparing different models: RF, SVM, LR, KNN, MLP

Model 1: Logistic Regression.

This is a simple linear model and does not have layers.

The sigmoid function is used as its activation.

lbfgs is used as the solver with L2 regularization

The default max_iter for convergence is 100.

```
param_grid = {'C': np.logspace(-5, 8, 15)}

logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg, param_grid, cv=10,refit=True)
unbalanced_logreg_cv = GridSearchCV(logreg, param_grid, cv=10,refit=True)
```

Model 2: RandomForestClassifier

Ensemble machine learning model using groups of decision trees to reduce overfitting

The default n_estimators is 100

Default criterion = gini

Convergence ends when all leaves are pure or until each leaf is equal to min_samples_leaf (by default set to 2)

```
rnd_clf = RandomForestClassifier()
unbalanced_rnd_clf = RandomForestClassifier()
```

Model 3: KNeighborsClassifier

A machine learning iterative classification method that uses a distance function to group similar data

n_neighbors (number of surrounding data points to consider) is 5 by default

```
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)
unbalanced_knn_clf_cv = GridSearchCV(knn_clf, knn_param_grid, cv=10,refit=True)
```

MODEL 4: SupportVectorClassifier

Support vector models are classfication algorithm that uses hyperplances to classify data points using a maximum margin between decision boundaries and the closest data points.

The regularization parameter C is set to 1 by default.

The kernel type is set to 'rbf' (radial basis function) by default

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

MODEL 5: MLPClassifier

MLPs are neural networks used for pattern detection.

They consist of 3 layers of nodes, an input layer, a hidden layer, and an output layer.

The default max_iter is 200.

The optimizer is 'Adam'

Each node uses a nonlinear activation function. By default this is ReLu for the hidden layer. For the output layer, softmax is used by default for multi-class classification problems and logistic for binary classification problems.

Backpropagation and gradient descent is used to to train and minimize loss.

```
nn_clf = MLPClassifier()
unbalanced_nn_clf = MLPClassifier()
```

Set up training metrics

```
# metrics to evaluate my model
# Define column names for the summary DataFrame
col_names = ["Classifier Name", "Accuracy Score", "Precision Score", "Recall Scor
```

```
# Initialize the summary DataFrame with predefined column names
unbalanced_summary_df = pd.DataFrame(columns=col_names)

# Lists to store the metrics for each estimator
unbalanced_est_name = []
unbalanced_est_acc = []
unbalanced_precision_score = []
unbalanced_recall_score = []
unbalanced_f1score = []
unbalanced_est_conf_matrix = []
unbalanced_roc=[]

# List of tuples containing the classifiers to evaluate and their respective vari
unbalanced_estimators = [
    ("UnbalancedLogisticRegression", unbalanced_logreg_cv),
    ("UnbalancedRandomForestClassifier", unbalanced_rnd_clf),
    ("UnbalancedKNeighborsClassifier", unbalanced_knn_clf_cv),
    ("UnbalancedSupportVectorClassifier", unbalanced_svm_clf_cv),
    ("UnbalancedMLPClassifier", unbalanced_nn_clf)
    ]

unbalanced_models = ['UnbalancedLogisticRegression.pkl',
                     'UnbalancedRandomForestClassifier.pkl',
                     'UnbalancedKNeighborsClassifier.pkl',
                     'UnbalancedSupportVectorClassifier.pkl',
                     'UnbalancedMLPClassifier.pkl']

summary_df = pd.DataFrame(columns=col_names)


# Lists to store the metrics for each estimator
est_name = []
est_acc = []
precision_score = []
recall_score = []
f1score = []
est_conf_matrix = []
roc=[]

# List of tuples containing the classifiers to evaluate and their respective vari
estimators = [
    ("LogisticRegression", logreg_cv),
    ("RandomForestClassifier", rnd_clf),
    ("KNeighborsClassifier", knn_clf_cv),
    ("SupportVectorClassifier", svm_clf_cv),
```

```
    ("MLPClassifier", nn_clf)
    ]
balanced_models = ['LogisticRegression.pkl',
                   'RandomForestClassifier.pkl',
                   'KNeighborsClassifier.pkl',
                   'SupportVectorClassifier.pkl',
                   'MLPClassifier.pkl']
```

Model Training

Dataset is small enough to compute on an 8 GB 2133 MHz LPDDR3 2.3 GHz Dual-Core Intel Core i5.

Average run time for training is < 2 minutes.

Unbalanced Training (Ablation)

Average runtime: < 2 minutes

```
# # Iterate over the estimators to train
# for i in range(len(unbalanced_estimators)):
#     unbalanced_clf_name, unbalanced_clf = unbalanced_estimators[i]  # Unpack th
#     print("Training ", unbalanced_clf_name)
#     unbalanced_clf.fit(unbalanced_X_train, unbalanced_y_train)  # Train the cla
#     gd_model_name = root + unbalanced_clf_name + '.pkl'
#     with open(gd_model_name, 'wb') as file:
#       pickle.dump(unbalanced_clf, file)
# unbalanced_clf_name, unbalanced_clf = '',''
```

Balanced Training

Average runtime: < 2 minutes

```
# for i in range(len(estimators)):
#     clf_name, clf = estimators[i]  # Unpack the classifier name and the classifi
#     print("Training ", clf_name)
#     clf.fit(X_train, y_train)  # Train the classifier
#     gd_model_name = root + clf_name + '.pkl'
#     with open(gd_model_name, 'wb') as file:
#       pickle.dump(clf, file)
# clf_name, clf = '',''
```

Summary

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

```
# # Our Models:
# # Logistic Regression
# # Random Forest
# # Support Vector Machines (SVM)
# # K-Nearest Neighbors (KNN)
# # Multilayer Perceptron (MLP)

# # Use a negative log-loss function with a regularization constant for training.
# # Select the best model based on the minimum loss across all five models.

# import pandas as pd
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LogisticRegression
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.svm import SVC
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.neural_network import MLPClassifier
# from sklearn.metrics import log_loss

# # Define a mapping from age categories to numeric values
# age_cat_mapping = {
#     'Teen': 10,
#     "20's": 20,
#     "30's": 30,
#     "40's": 40,
#     "50's": 50,
#     '70+': 70
```

```
# }

# # Apply the mapping to the 'age_cat' column
# risk_factor_df['age_cat'] = risk_factor_df['age_cat'].map(age_cat_mapping)

# # Define X (features) and y (target)
# # Drop the target column to create the feature matrix
# X = risk_factor_df.drop(columns=['Dx:Cancer'])
# # Target variable
# y = risk_factor_df['Dx:Cancer']

# # Split the data into training and test sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# # Define the models
# models = {
#     'Logistic Regression': LogisticRegression(solver='liblinear', random_state=4
#     'Random Forest': RandomForestClassifier(random_state=42),
#     'SVM': SVC(probability=True, random_state=42),
#     'KNN': KNeighborsClassifier(),
#     'MLP': MLPClassifier(random_state=42)
# }

# # Train the models and calculate the negative log loss
# log_losses = {}
# for name, model in models.items():
#     model.fit(X_train, y_train)
#     if hasattr(model, "predict_proba"):
#         y_pred_proba = model.predict_proba(X_test)
#         log_losses[name] = log_loss(y_test, y_pred_proba)
#     else:
#         print(f"Model {name} does not support probability estimates, skipping lo

# # Find the best model based on the minimum log loss
# if log_losses:
#     best_model_name = min(log_losses, key=log_losses.get)
#     best_model = models[best_model_name]
#     print(f"The best model is {best_model_name} with a log loss of {log_losses[b
```

Checkpoint: Results

In this section, you should finish training your model training or loading your trained model. That is a great experiment! You should share the results with others with necessary metrics and figures.

Please test and report results for all experiments that you run with:

- specific numbers (accuracy, AUC, RMSE, etc)
- figures (loss shrinkage, outputs from GAN, annotation or label of sample pictures, etc)

Download pickle models with gdown

```
# LogisticRegression
gdown.download('https://drive.google.com/file/d/1-4D7VkQ4qewUnyrzPw1izMk9WjMvQ13D
gdown.download('https://drive.google.com/file/d/1-94qCsT0VdS-CvO237Z0TeUWCouKDeyl
# RandomForestClassifier
gdown.download('https://drive.google.com/file/d/1-PBTxxalNfXQNKpX8Bg9ojFRtPGEsvbw
gdown.download('https://drive.google.com/file/d/1-Qdk5tk2RCLhbfVW400S2qBI6ikStRBh
# KNeighborsClassifier
gdown.download('https://drive.google.com/file/d/1-Rg4cRA6NdtPeHPFnablplZ0T7e323Qg
gdown.download('https://drive.google.com/file/d/1-TKog0oufiDcfCJtv_gQdYTiw1rK6VkN
# SupportVectorClassifier
gdown.download('https://drive.google.com/file/d/1_g0BZHdpuPP9xlj0jKWJ-8fVZe4sTPkA
gdown.download('https://drive.google.com/file/d/1-3sNvATHD55evt_9nSzf3nBP6XWLknjL
# MLPClassifier
gdown.download('https://drive.google.com/file/d/1-3P6L-WKDrfMNgwp372f3vgY9Px0SEIm
gdown.download('https://drive.google.com/file/d/1-DmdcIFHJFKUtAhhVW1XbVlwbfjOpWS2
```

```
'MLPClassifier.pkl'
```

Model Execution

Unbalanced evaluation (ablation)

```
# Iterate over the trained models to evaluate each one
```

```
for i in range(len(unbalanced_estimators)):
    with open(unbalanced_models[i], 'rb') as file:
      downloaded_model = pickle.load(file)
    unbalanced_clf_name, _ = unbalanced_estimators[i]
    unbalanced_y_pred = downloaded_model.predict(unbalanced_X_test)  # Predict th
    # Calculate the AUROC score and append it to the roc list
    unbalanced_roc.append(roc_auc_score(unbalanced_y_test, unbalanced_y_pred, ave

    # Append classifier name to the est_name list
    unbalanced_est_name.append(unbalanced_clf_name)

    # Calculate and append accuracy to the est_acc list
    unbalanced_est_acc.append(accuracy_score(unbalanced_y_test, unbalanced_y_pred

    # Calculate precision, recall, and F1 scores and append them to their respect
    unbalanced_scores = precision_recall_fscore_support(unbalanced_y_test, unbala
    unbalanced_precision_score.append(unbalanced_scores[0])
    unbalanced_recall_score.append(unbalanced_scores[1])
    unbalanced_f1score.append(unbalanced_scores[2])

    # Append the confusion matrix for each classifier to the est_conf_matrix list
    unbalanced_est_conf_matrix.append(confusion_matrix(unbalanced_y_test, unbalan

# Populate the summary DataFrame with the collected metrics for each classifier
unbalanced_summary_df[col_names[0]] = unbalanced_est_name
unbalanced_summary_df[col_names[1]] = unbalanced_est_acc
unbalanced_summary_df[col_names[2]] = unbalanced_precision_score
unbalanced_summary_df[col_names[3]] = unbalanced_recall_score
unbalanced_summary_df[col_names[4]] = unbalanced_f1score
unbalanced_summary_df[col_names[5]] = unbalanced_roc

# plot figures to better show the results

# it is better to save the numbers and figures for your presentation.
unbalanced_summary_df
```

	Classifier Name	Accuracy Score	Precision Score	Recall Score	F1 Score	AUROC
0	UnbalancedLogisticRegression	1.000000	1.000000	1.000000	1.000000	1.0
1	UnbalancedRandomForestClassifier	1.000000	1.000000	1.000000	1.000000	1.0
2	UnbalancedKNeighborsClassifier	0.994186	0.988406	0.994186	0.991288	0.5
3	UnbalancedSupportVectorClassifier	1.000000	1.000000	1.000000	1.000000	1.0
4	UnbalancedMLPClassifier	1.000000	1.000000	1.000000	1.000000	1.0

Balanced evaluation

```
# Iterate over the trained models to evaluate each one
for i in range(len(estimators)):
    with open(balanced_models[i], 'rb') as file:
      downloaded_model = pickle.load(file)
    clf_name, _ = estimators[i]
    y_pred = downloaded_model.predict(X_test)  # Predict the test set outcomes

    # Calculate the AUROC score and append it to the roc list
    roc.append(roc_auc_score(y_test, y_pred, average=None))

    # Append classifier name to the est_name list
    est_name.append(clf_name)

    # Calculate and append accuracy to the est_acc list
    est_acc.append(accuracy_score(y_test, y_pred))

    # Calculate precision, recall, and F1 scores and append them to their respect
    scores = precision_recall_fscore_support(y_test, y_pred, average="weighted")
    print(scores)
    precision_score.append(scores[0])
    recall_score.append(scores[1])
    f1score.append(scores[2])

    # Append the confusion matrix for each classifier to the est_conf_matrix list
    est_conf_matrix.append(confusion_matrix(y_test, y_pred))

# Populate the summary DataFrame with the collected metrics for each classifier
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

# plot figures to better show the results

# it is better to save the numbers and figures for your presentation.
summary_df
```

```
(1.0, 1.0, 1.0, None)
(1.0, 1.0, 1.0, None)
(0.9642001915708812, 0.9613095238095238, 0.961313979066094, None)
(0.9970421810699589, 0.9970238095238095, 0.997024152746606, None)
(1.0, 1.0, 1.0, None)
```

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

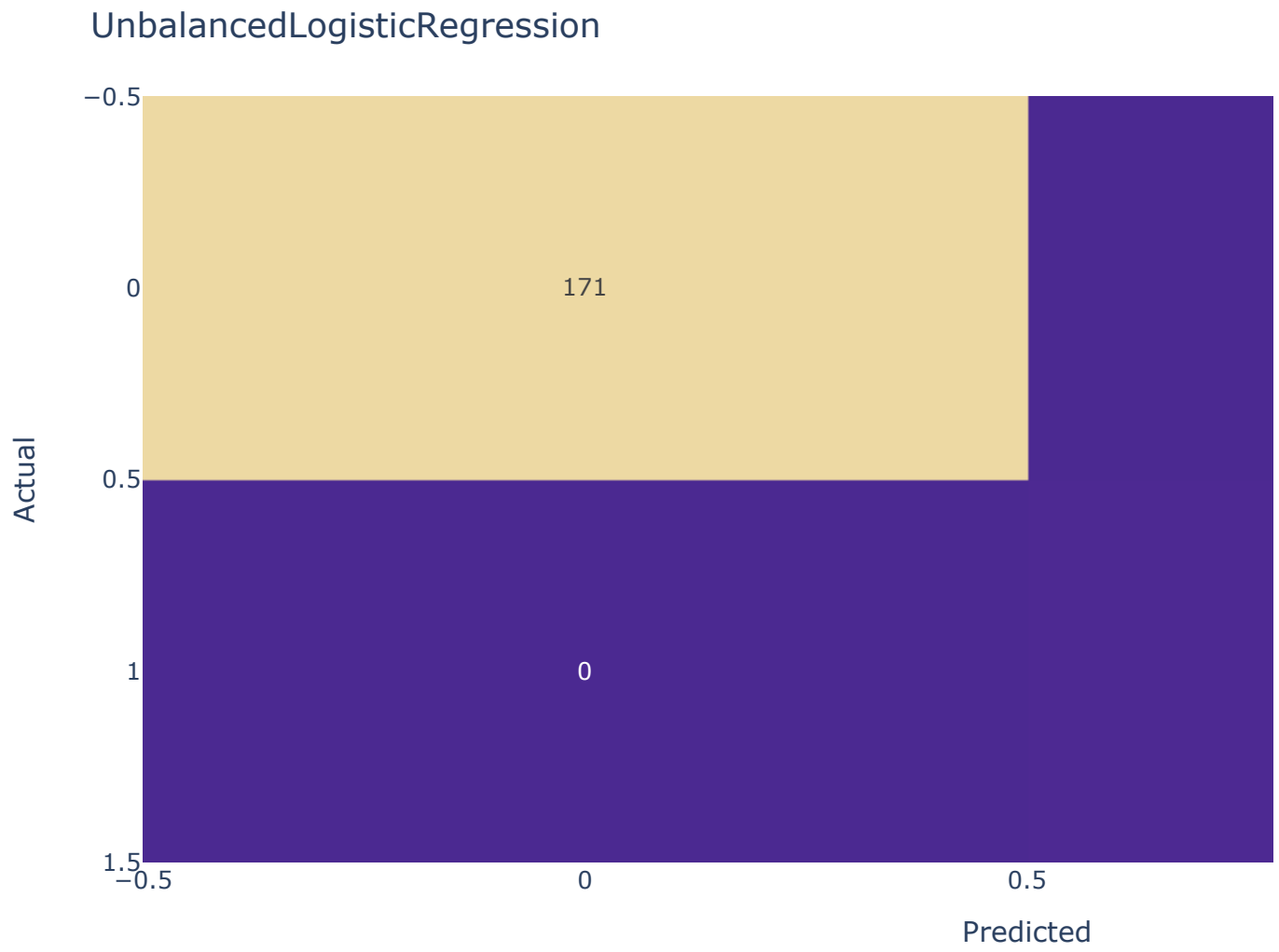
Model comparison

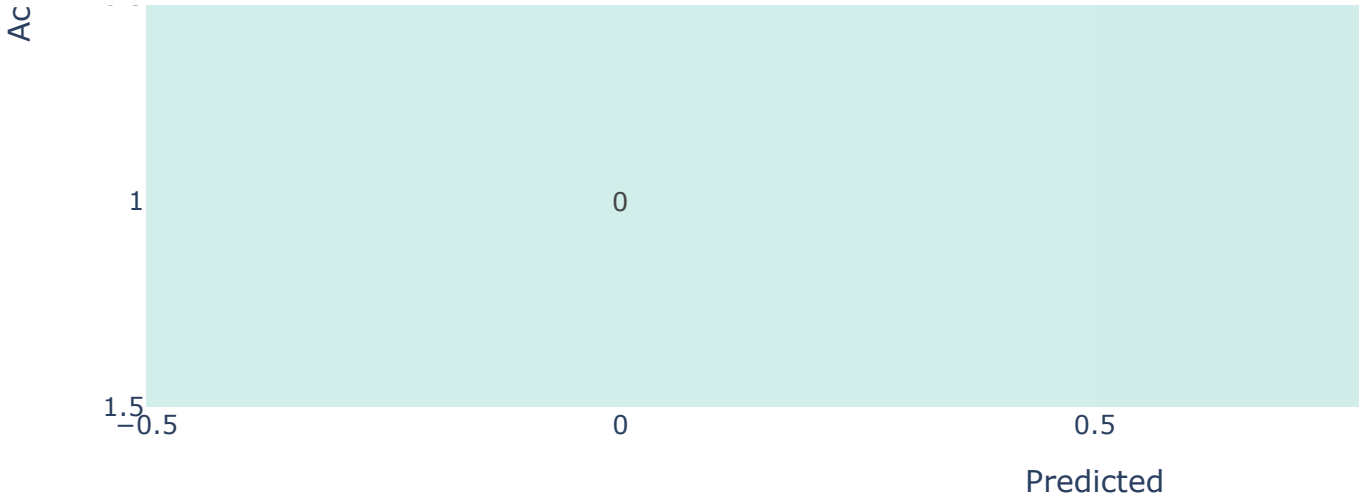
```
# compare you model with others
# you don't need to re-run all other experiments, instead, you can directly refer
```

Unbalanced comparison

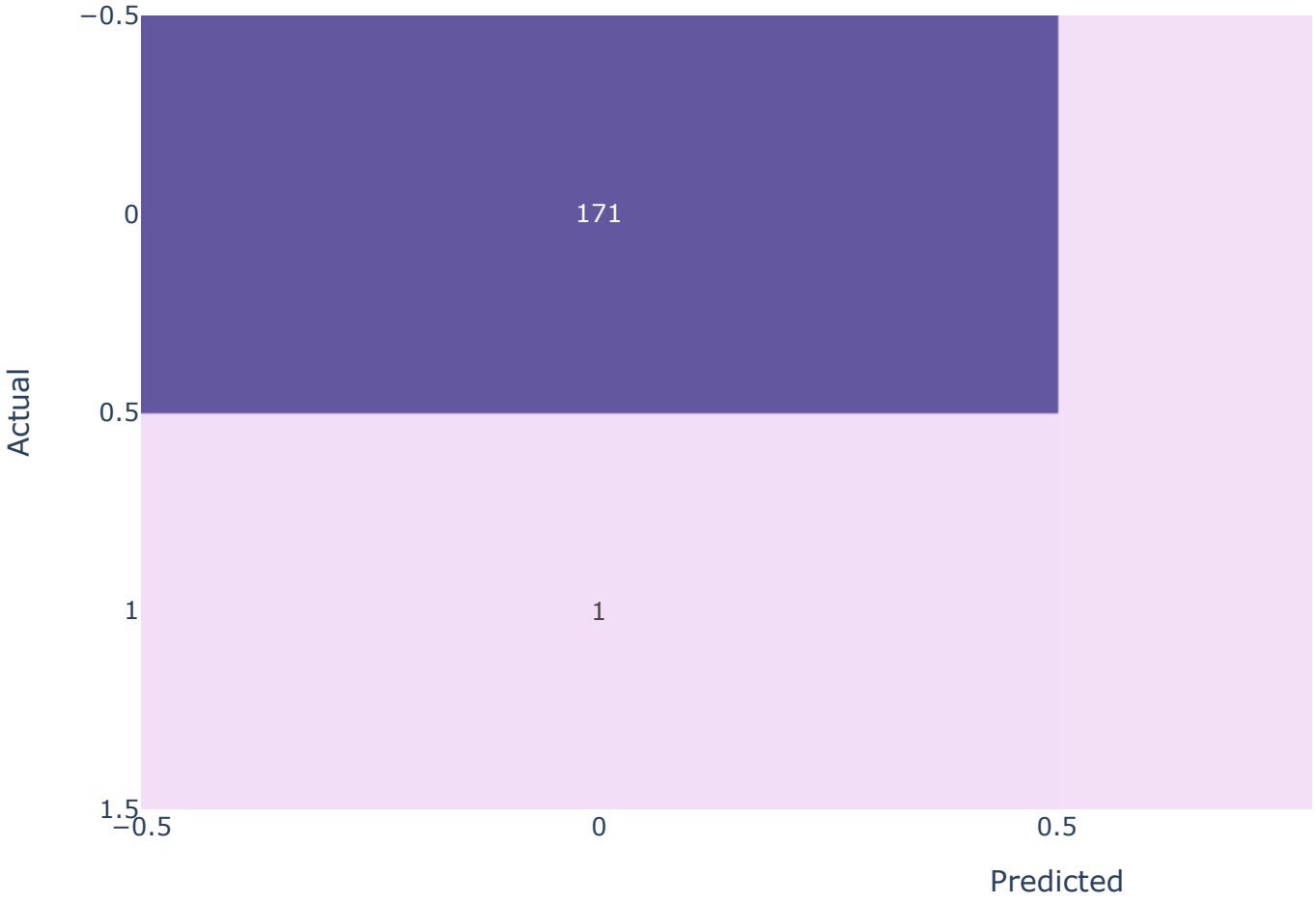
```
color_scales = ["agsunset","teal","purp","viridis","viridis"]
for i in range(0,len(unbalanced_est_conf_matrix)):
    unbalanced_heatmap = px.imshow(unbalanced_est_conf_matrix[i],aspect="auto",
                        text_auto=True,
                        color_continuous_scale=color_scales[i])
    unbalanced_heatmap.update_layout(title = unbalanced_est_name[i])
```

```
unbalanced_heatmap.update_xaxes(title="Predicted")
unbalanced_heatmap.update_yaxes(title="Actual")
unbalanced_heatmap.show()
```

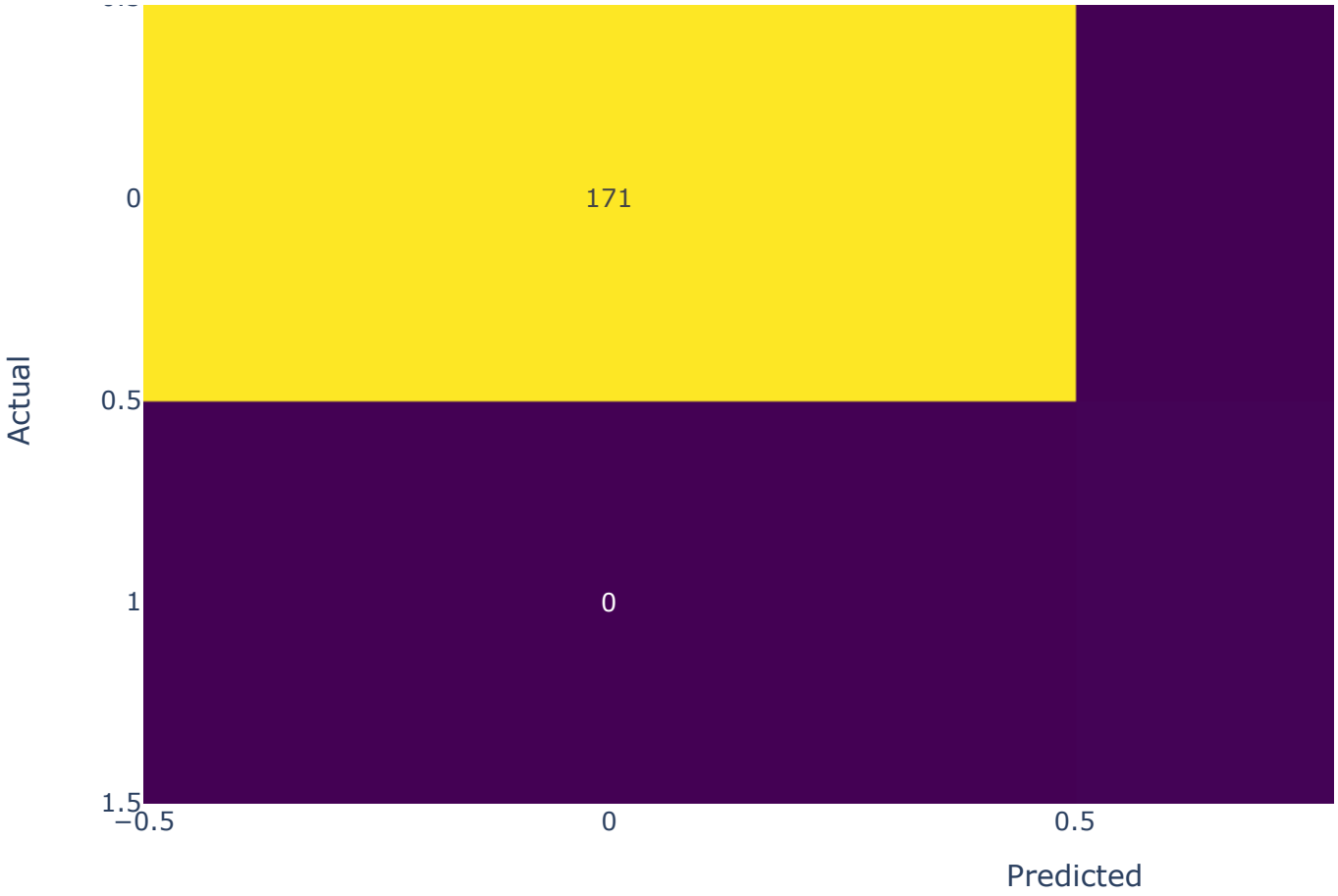




UnbalancedKNeighborsClassifier

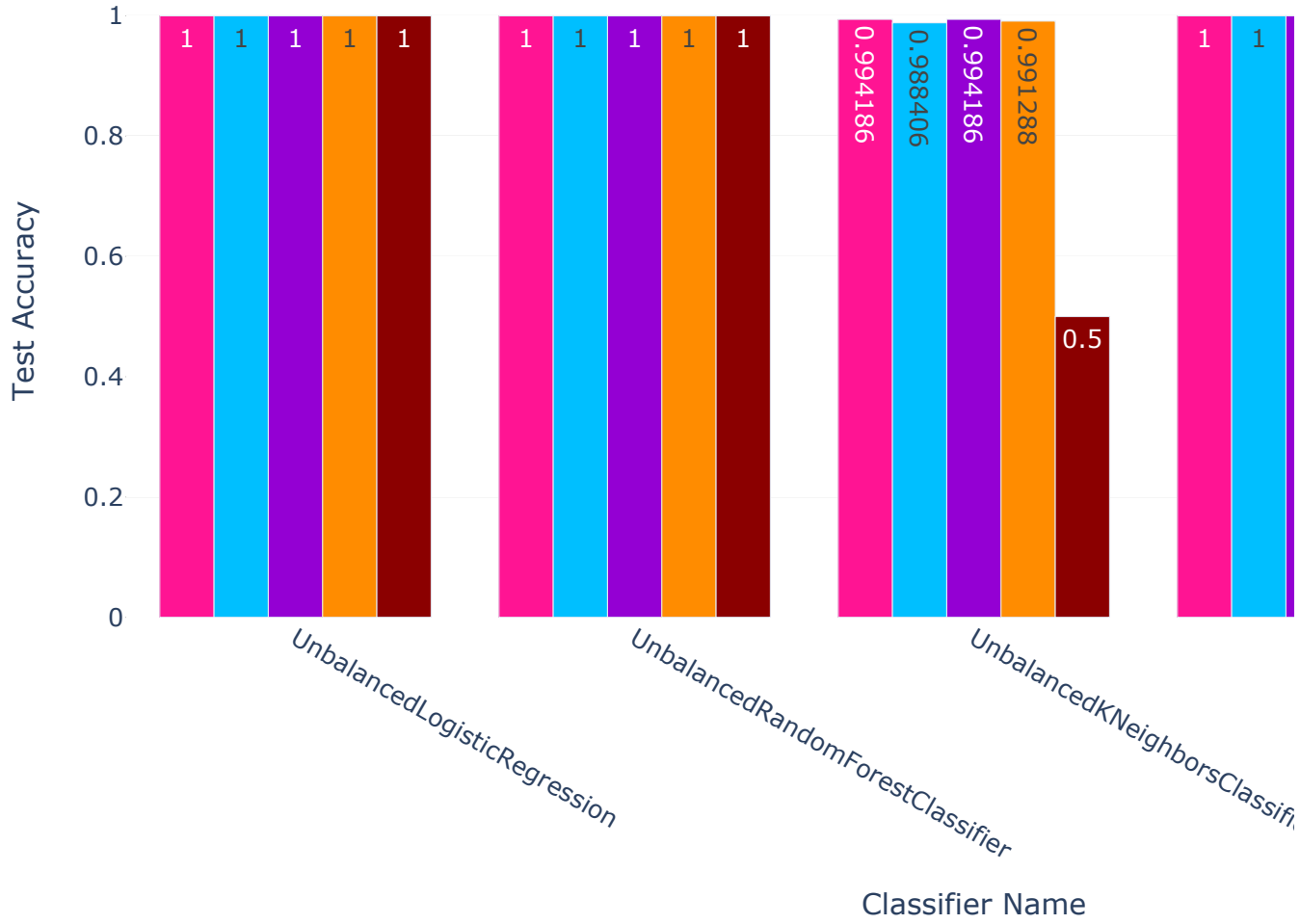


UnbalancedSupportVectorClassifier



UnbalancedMLPClassifier


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

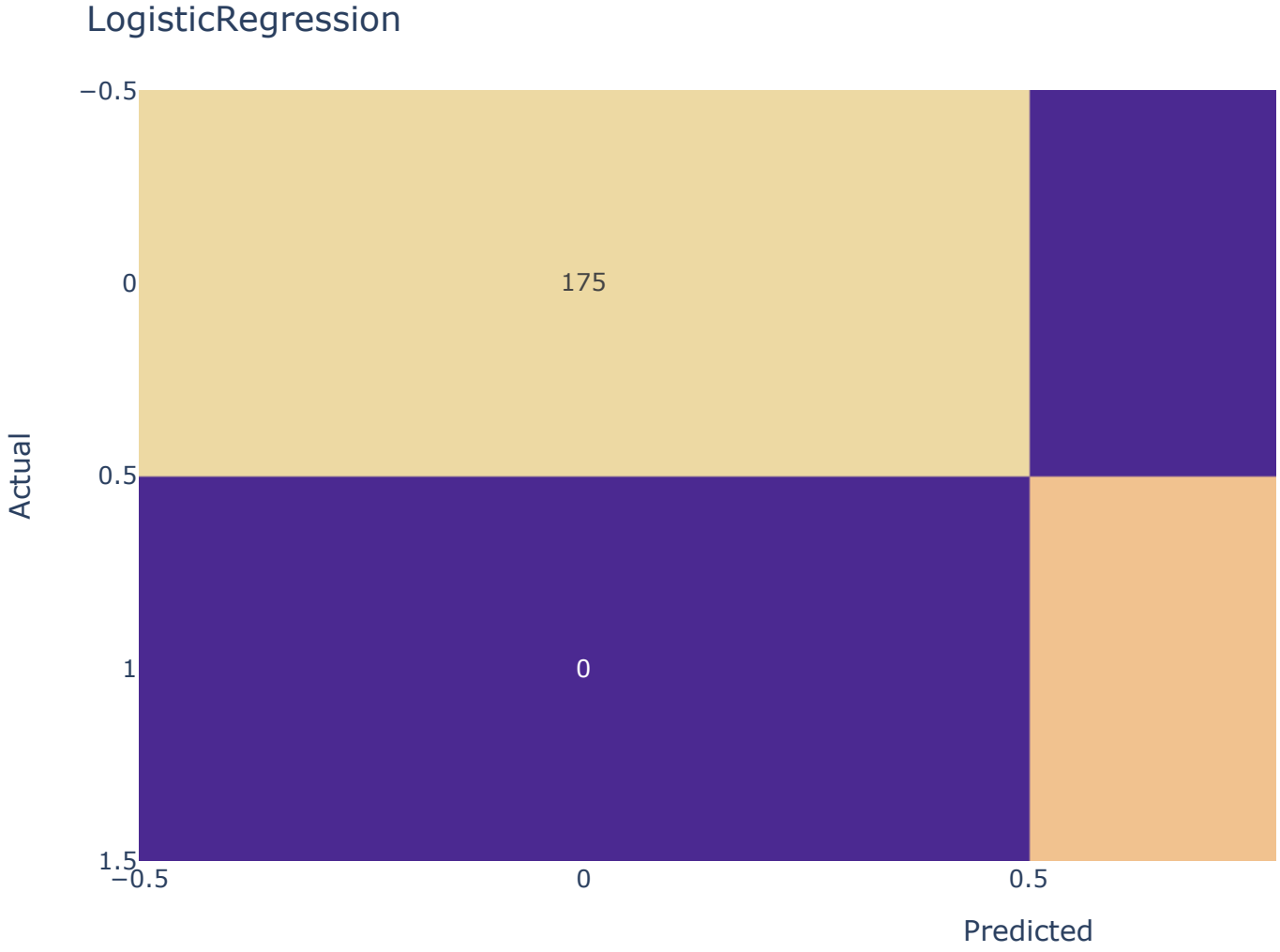


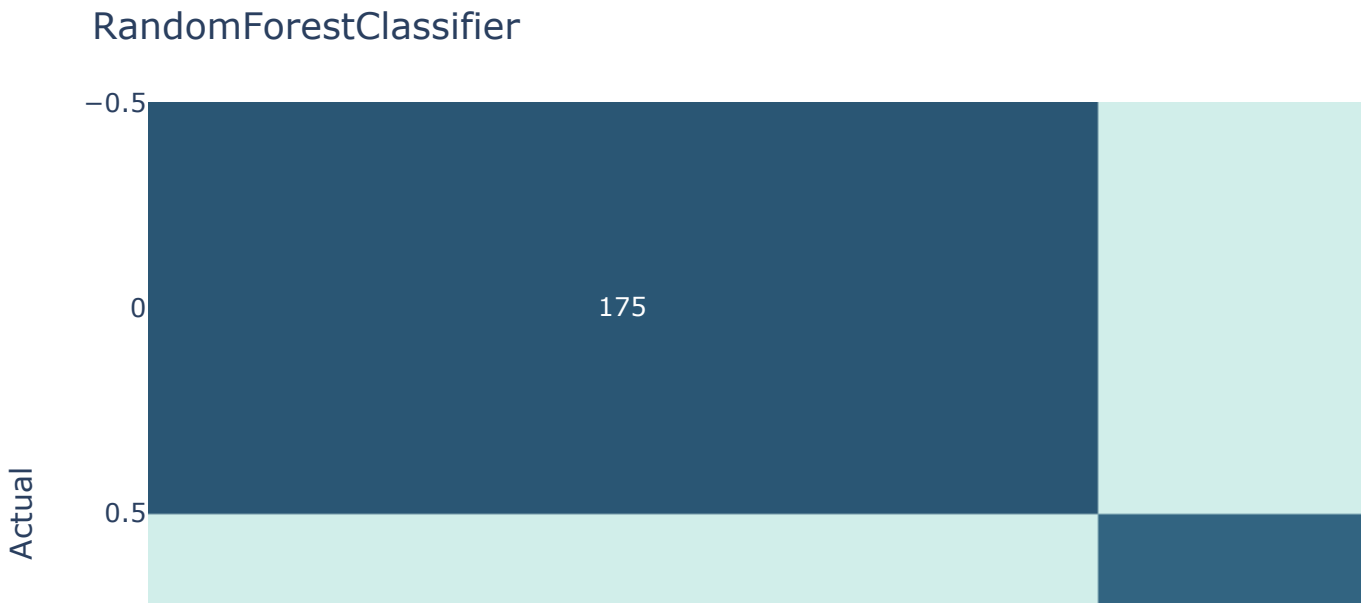
Balanced comparison

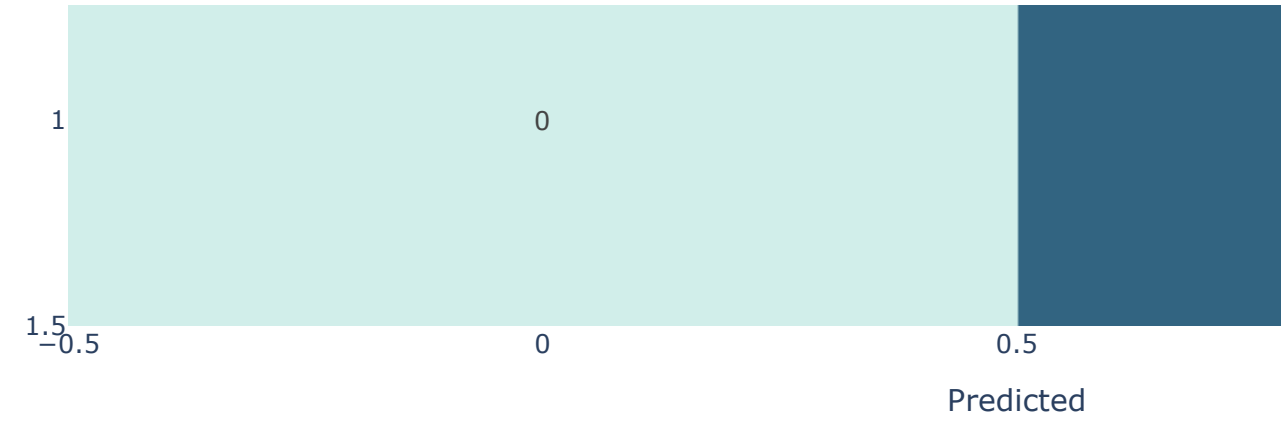
Summary with heat map

```
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")
```

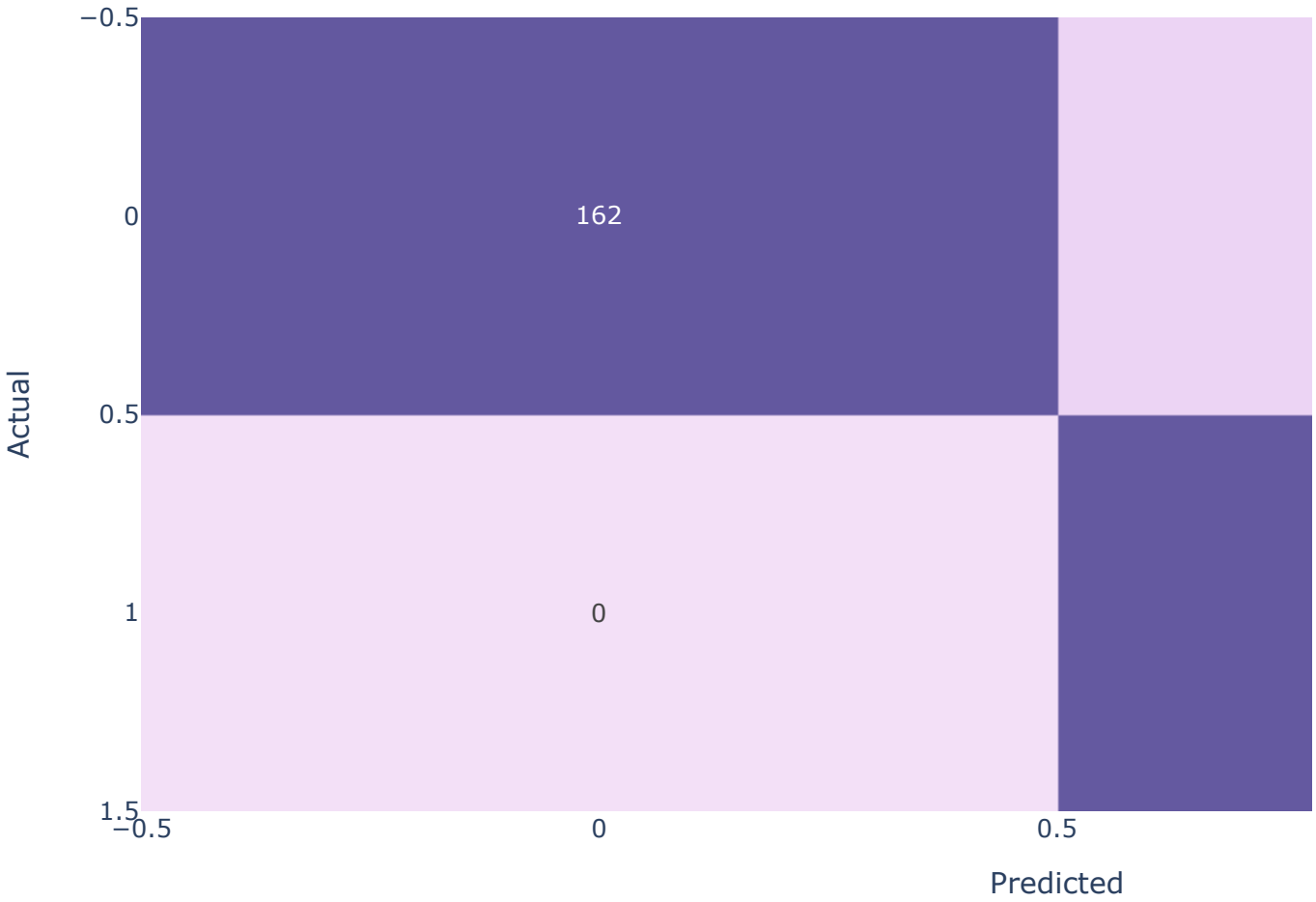
```
heatmap.show()
```



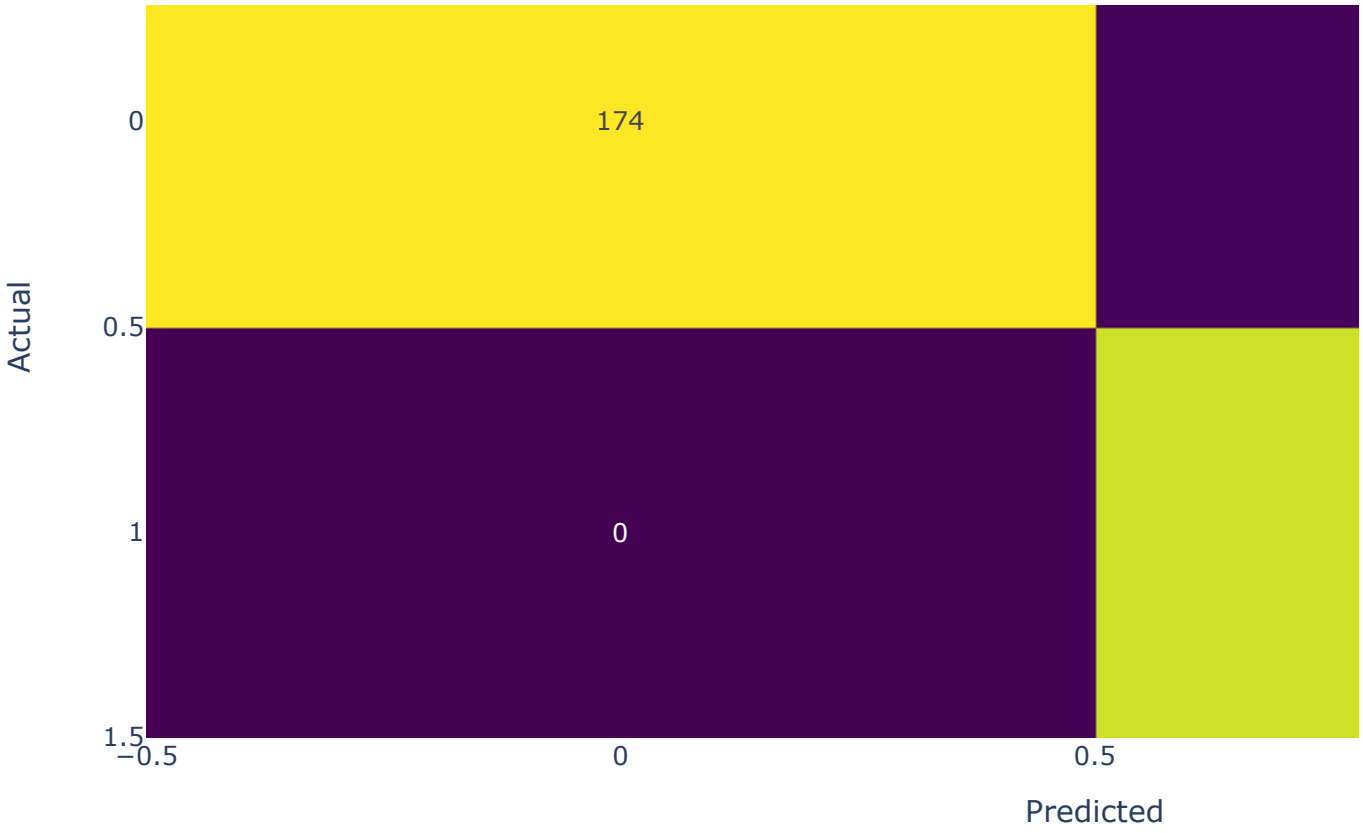




KNeighborsClassifier

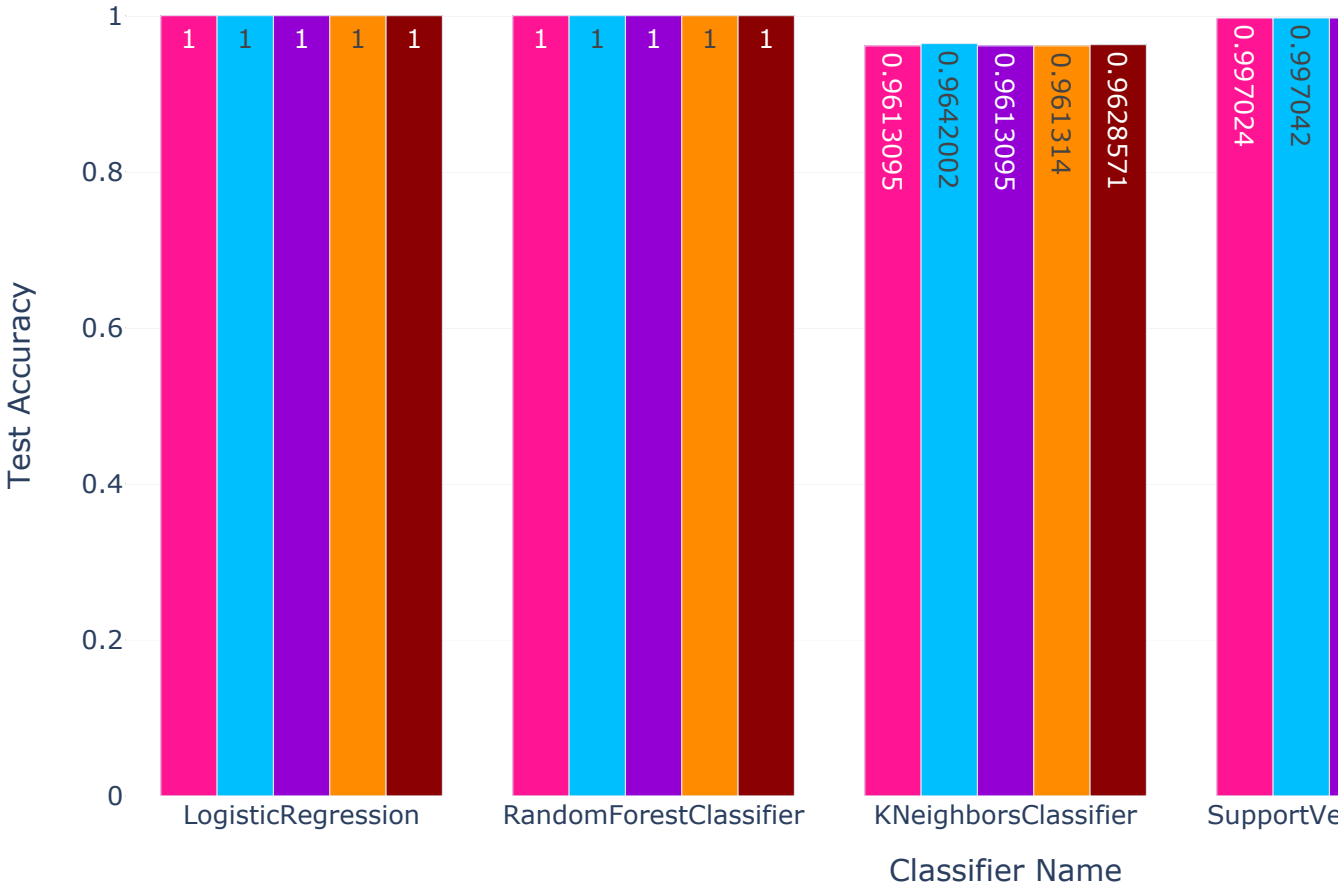


SupportVectorClassifier



MLPClassifier


```
#https://plotly.com/python/error-bars/
#https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.07-Error-Bar
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",
                                                 "darkviolet",
                                                 "darkorange",
                                                 "darkred"],
                        barmode="group"
                        #,error_y=[dict(type='data', array=[0.5, 1, 2],visible=Tr
                        #,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()
```



** Description of Metrics**

TP: True Positive, these are the values that are positive and were predicted positive

FP: False Positive, The values which are negative but were wrongly predicted as positive

TN: True Negative, these are the values that are negative and were predicted negative

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

Precision = TP / (TP+FP)

This metric measures the actual positive outcomes out of the total predicted positive outcomes. It attempts to identify the proportion of positive identifications that were correct. KNeighbors

and SVC gave the worst precision score

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

Recall

Recall = TP / (TP+FN)

This metirc measures the correctly positive predicted outocmes of the total number of positive outcomes. It answers the question of what proportions of actual positives were identified correctly. KNeighbors and SVC gave the worst precision score

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

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

The symptoms,especially in the early stages being mistaken for some other type of less serious illness. The actual test adminstered by a healthcare professional may give the wrong diagnosis The 5-year survival rate tells you what percent of people live at least 5 years after the cancer is found. Percent means how many out of 100. The 5-year survival rate for all people with cervical cancer is 66%. Source

Survival rates also depend on the stage of cervical cancer that is diagnosed. When detected at an early stage, the 5-year survival rate for people with invasive cervical cancer is 92%. About 44% of people with cervical cancer are diagnosed at an early stage. If cervical cancer has spread to surrounding tissues or organs and/or the regional lymph nodes, the 5-year survival rate is 58%. If the cancer has spread to a distant part of the body, the 5-year survival rate is 18%.

F1 Score

F1 Score =TP / (TP + ((FN+FP)/2))

The F1 score is defined as the harmonic mean of precision and recall. Therefore, a high F1 score

means both a high precision and recall, same for low and a medium score if one score is high and the other is low.

Accuracy Accuracy = (TP + TN) / (TP + FP + TN + FN)

XAI TODO

Add a random var

```
# from scipy.stats import bernoulli
# risk_factor_df['VAR']=bernoulli.rvs(.5, size=risk_factor_df.shape[0])


# #continous
# risk_factor_df['VAR']=np.random.normal(loc=0, scale=1, size=risk_factor_df.shap
# risk_factor_df.columns
```

Get data and model (unsure about this section)

```
#without RAND

# risk_factor_df.drop(cols_to_drop, axis=1, inplace=True)
# risk_factor_df.to_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/Rcancer2
# risk_factor_df=pd.read_csv('/content/drive/My Drive/DL4H_Sp24_Final_Project/can

#need cancer.csv
```

```
#with randn

#adding noise

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




# 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])
```

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


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


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


# X_train
```

```
#continous

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

# 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])


# X_test.to_csv('/content/drive/My Drive/dataXAI/cancer/CX_test.csv')
# y_test.to_csv('/content/drive/My Drive/dataXAI/cancer/Cy_test.csv')
# X_train.to_csv('/content/drive/My Drive/dataXAI/cancer/CX_train.csv')
# y_train.to_csv('/content/drive/My Drive/dataXAI/cancer/Cy_train.csv')


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

Discussion

The paper we chose has shown to be reproducible. We were able to acquire, process, and visual the data, and run the 5 models and ablations against the data with common Python libraries and minimal error or confusion. During the reproduction, moving the data to adapt to our local enviroments in a shareable format was initially difficult, but ultimateley possible. Processsing data was also hard, as the data set needed reformatting to work correctly with some models. Transferring the code and adding ablations was successful because of the heavily commented and organized code that could be split into many sections. In order to make this paper more easy to reproduce, we would suggest adding checkpoints to the program as well as providing more documentation regarding data cleaning. In this draft we began the evaluation phase, and accuratley began to reproduced the model and set up our ablations for evalutation and results. We began conducting results and comparing our ablations against the original model.

Remaining work

In the next phase, we finish calculating results using the metrics we have defined, we will also apply the interpretability models (DICE, LIME, etc.) to the results from our model evaluation, and analyze the interpretability metrics included in the paper to understand to success of our ablations and complete the reproduction. In order to better understand the results, we will use plotting and visualization to understand the strengths and weaknesses of various interpretability models. We will also conduct the ROAR phase of our model testing.

References

[1] Ayad, 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. Ecole polytechnique, 1-50.

[2] Fernandes,Kelwin, Cardoso,Jaime, and Fernandes,Jessica. (2017). Cervical cancer (Risk Factors). UCI Machine Learning Repository. https://doi.org/10.24432/C5Z310. https://github.com/cwayad/Local-Explanations-for-Cervical-Cancer