Exploratory data analysis of the Cervical cancer (Risk factors) Data set

In [1]:

```
import numpy as np
import pandas as pd
import altair as alt
from sklearn.model_selection import train_test_split, StratifiedKFold

alt.data_transformers.enable('data_server')
alt.renderers.enable('mimetype')
```

Out[1]:

```
RendererRegistry.enable('mimetype')
```

Summary of the data set

The data set was collected at 'Hospital Universitario de Caracas' in Caracas, Venezuela. The data set comprises demographic information, habits, and historic medical records of 858 patients. Several patients decided not to answer some of the questions because of privacy concerns (missing values). This data set was sourced from the UCI Machine Learning Repository and can be found here.

The data set was used in Kelwin Fernandes, Jaime S. Cardoso, and Jessica Fernandes. 'Transfer Learning with Partial Observability Applied to Cervical Cancer Screening.' Iberian Conference on Pattern Recognition and Image Analysis. Springer International Publishing, 2017, available here.

The data set has 4 different target variables each having a value of 0(tested negative for that specific medical test) or 1(tested positive for that specific medical test). For the purpose of this project, these binary class variables will be combined into a single binary target variable which will be 1(True) if any medical test is positive and 0(False) if no test was positive.

In [2]:

```
# load dataset into pandas dataframe
cervical_raw = pd.read_csv('../data/risk_factors_cervical_cancer.csv')

# create target variable 'risk'
risk = []
for row in range(len(cervical_raw)):
    risk.append(
        cervical_raw.loc[cervical_raw.index[row], 'Hinselmann'] or
        cervical_raw.loc[cervical_raw.index[row], 'Schiller'] or
        cervical_raw.loc[cervical_raw.index[row], 'Citology'] or
        cervical_raw.loc[cervical_raw.index[row], 'Biopsy']
    )
cervical_modified = cervical_raw.copy()
cervical_modified['risk'] = risk

# drop the previous target variables
cervical_modified = cervical_modified.drop(columns=['Hinselmann', 'Schiller', 'Citology'

# create dataframe with counts of each class
class_counts = pd.DataFrame(cervical_modified['risk'].value_counts()).rename(index={0:'N
                                                                                  1:'R
                                                                         columns={'r
```

```python
# set caption for Table 1
class_counts.style.set_caption('Table 1. Counts of observation for each class')
```

Out[2]:

Table 1. Counts of observation for each class

	Target
No risk of cervical cancer	756
Risk of cervical cancer	102

Split data set into training and test splits

before splitting the dataset, we replace all occurences of '?' in the data with `np.nan` so that it is easier to work with the missing values. We also change the data types of columns to match the data stored in them.

In [3]:

```python
# replace the ? values with NaN
cervical_clean = cervical_modified.replace('?', np.nan)

# convert columns to relevant data types
for col_name in cervical_clean.columns:
    if cervical_clean[col_name].dtype == 'object':
        cervical_clean[col_name] = cervical_clean[col_name].astype(float)
```

We now split our data so that 80% of the examples are in the training set while 20% are in the test set.

In [4]:

```python
# split data into training and test sets
train_df, test_df = train_test_split(cervical_clean, test_size=0.2, random_state=123)
```

In [5]:

```python
# create dataframe with counts of each class and for both train and test set
train_class_counts = pd.DataFrame(train_df['risk'].value_counts())
test_class_counts = pd.DataFrame(test_df['risk'].value_counts())

train_test_class_counts = pd.concat([train_class_counts, test_class_counts], axis=1).ren
    index={0:'No risk of cervical cancer',
           1:'Risk of cervical cancer'}
)
train_test_class_counts.columns = ['Train', 'Test']

# set caption for Table 2
train_test_class_counts.style.set_caption('Table 2. Counts of observations for each clas
```

Out[5]:

Table 2. Counts of observations for each class and partition

	Train	Test
No risk of cervical cancer	608	148
Risk of cervical cancer	78	24

There is quite a bit of class imbalance in this dataset. We won't try and use under-sampling or over-sampling to remedy this since our data set is quite small. We will deal with this after the inital model building and tuning phase in the case that the model is performing poorly. We can evaluate whether class imbalance is a major issue based on the confusion matrix (if the False Negative rate is high).

Exploratory analysis on the training set

We plotted the distributions of each explanatory variable in the training data set to see whether or not it will be useful for predicting the target variable.

Most of the numeric features are extremely skewed. This can have a negative impact on the model as machine learning models generally perform better on normalized data. As such, we might experiment with some transformations (eg: log transformation) to try and normalize the data. A bunch of our feature variables have a either all or atleast a significant amount of missing values. These features will likely be omitted from the final model. Taking a look at correlations between certain columns, we can see that some features are almost colinear. This means they can be safely removed as they do not add to model performance. This should reduce complexity in the model as well.

In [6]:

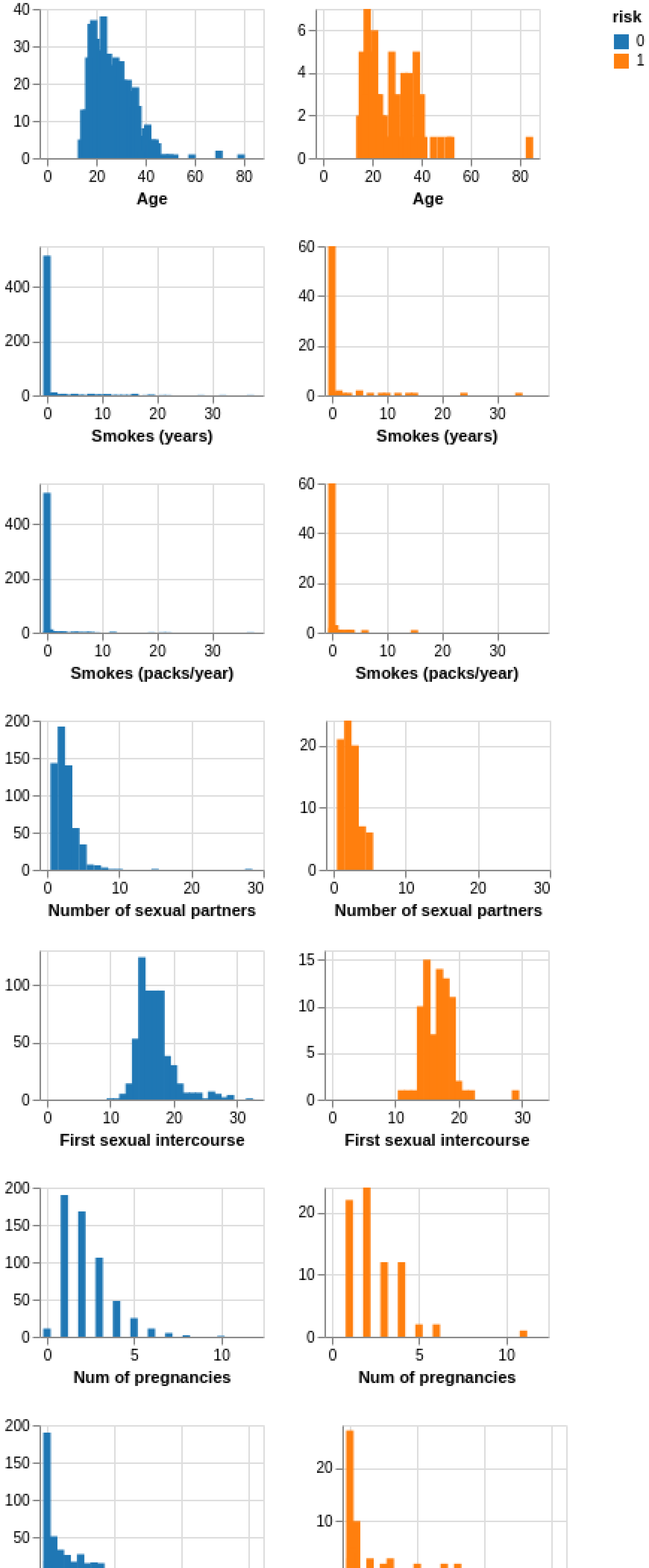
```
def hist( feat = None, feat_list = None, repeat = False):
    if repeat == False:
        chart = alt.Chart( train_df).mark_bar().encode(
            alt.X( 'Age', type='quantitative'),
            alt.Y( 'count()', stack=False, title=''),
            alt.Color( 'risk', type='ordinal', scale=alt.Scale(scheme='category10'))
        ).properties(
            height=100,
            width=150
        ).facet( 'risk', columns = 1)
        return chart
    if repeat == True:
        chart_list_0 = []
        chart_list_1 = []
        chart_list_concat = []
        for feat in feat_list:
            chart_tmp_0 = alt.Chart( train_df.query('risk==0')).mark_bar().encode(
                alt.X( feat, type='quantitative', scale = alt.Scale( domain = ( 0, train
                alt.Y( 'count()', stack=False, title=''),
                alt.Color( 'risk', type='ordinal', scale=alt.Scale(scheme='category10'))
            ).properties(
                height=100,
                width=150
            )
            chart_tmp_1 = alt.Chart( train_df.query('risk==1')).mark_bar().encode(
                alt.X( feat, type='quantitative', scale = alt.Scale( domain = ( 0, train
                alt.Y( 'count()', stack=False, title=''),
                alt.Color( 'risk', type='ordinal', scale=alt.Scale(scheme='category10'))
            ).properties(
                height=100,
                width=150
            )
            chart_list_0.append( chart_tmp_0)
            chart_list_1.append( chart_tmp_1)
            chart_concat = chart_tmp_0 | chart_tmp_1
            chart_list_concat.append( chart_concat)
        return alt.vconcat( *chart_list_concat)
```

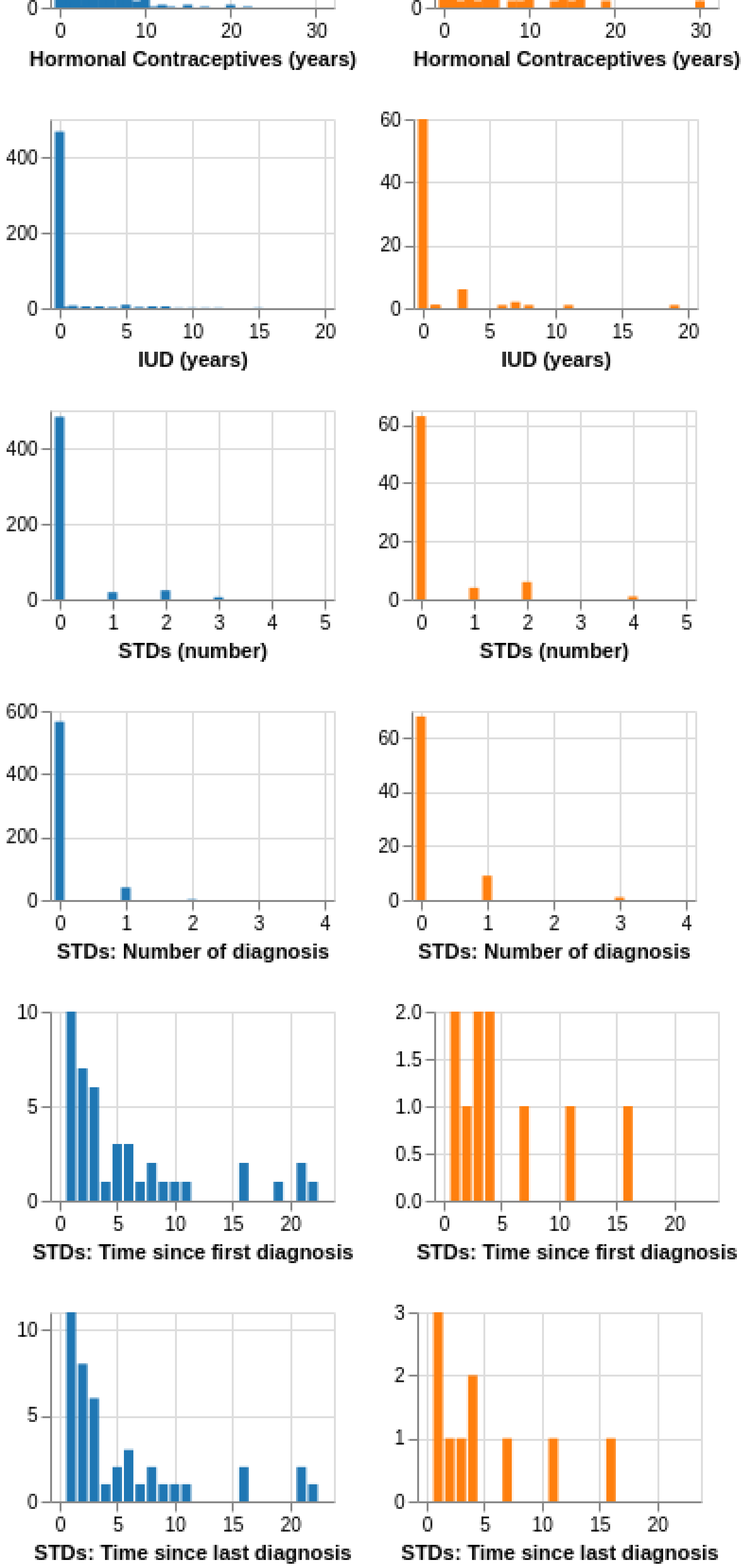
In [7]:

```
# create list of binary features
binary_features = ['Smokes', 'Hormonal Contraceptives', 'IUD', 'STDs', 'STDs:condylomato
                   'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
                   'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic i
                   'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AIDS', 'ST
                   'STDs:Hepatitis B', 'STDs:HPV', 'Dx:Cancer', 'Dx:CIN', 'Dx:HPV', 'Dx'

# create list of numeric features
numeric_features = ['Age', 'Smokes (years)', 'Smokes (packs/year)', 'Number of sexual pa
                    'Num of pregnancies', 'Hormonal Contraceptives (years)', 'IUD (years
                    'STDs (number)', 'STDs: Number of diagnosis', 'STDs: Time since firs
                    'STDs: Time since last diagnosis']

# create charts for binary features
binary_charts = alt.Chart(train_df).mark_bar().encode(
    alt.X(alt.repeat(), type='ordinal'),
    alt.Y('count()'),
    alt.Color('risk', type='ordinal', scale=alt.Scale(scheme='category10'))
).properties(
    height=150,
    width=75
).repeat(
    binary_features,
    columns=4
)
print("Figure 2: EDA for Numeric Features")
hist(feat_list=numeric_features, repeat=True)
```

```
Figure 2: EDA for Numeric Features
```

Out[7]:

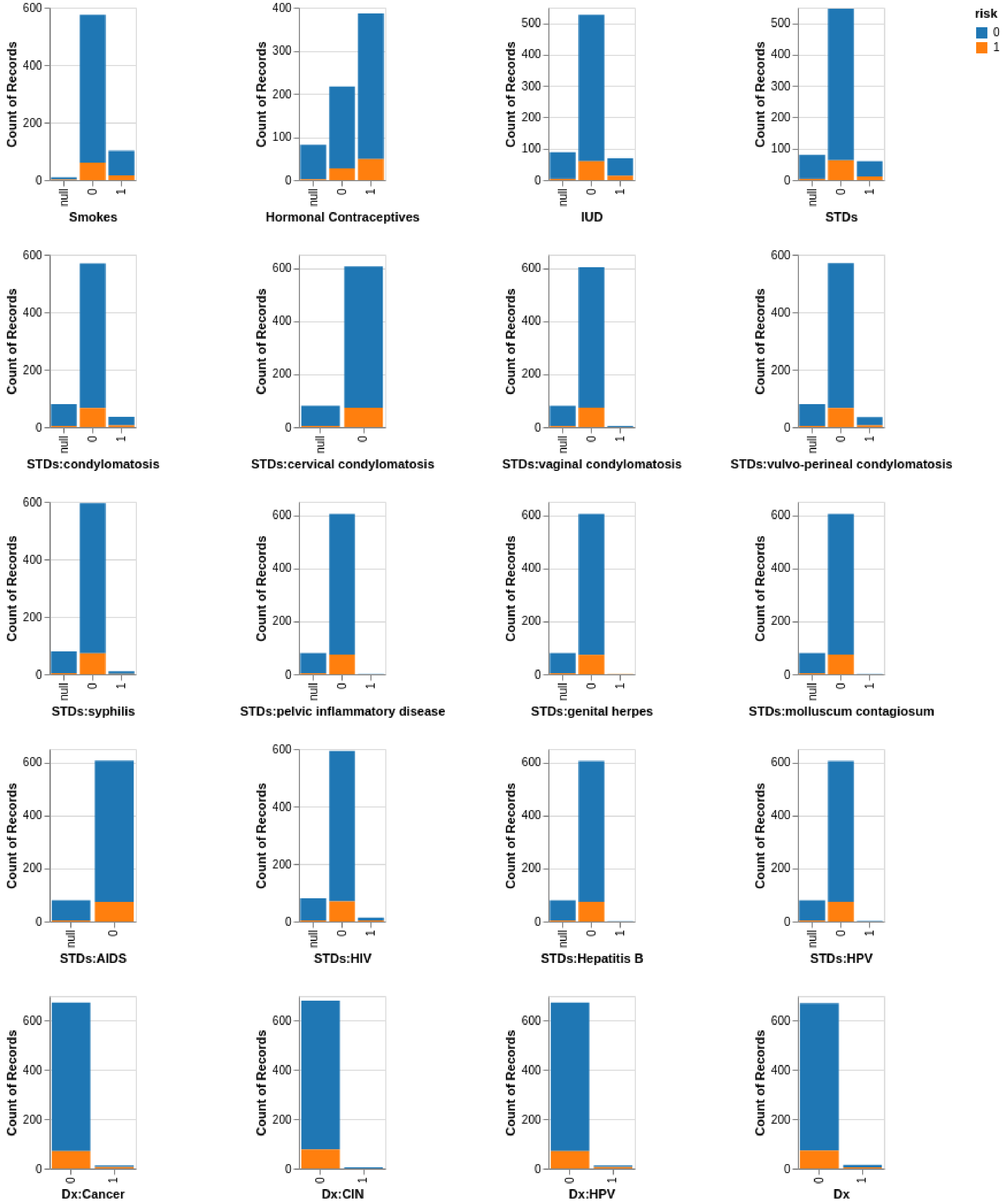

risk
0
1
40
30
20
10
0
0
20
40
60
80
Age
6
4
2
0
0
20
40
60
80
Age
400
200
0
0
10
20
30
Smokes (years)
60
40
20
0
0
10
20
30
Smokes (years)
400
200
0
0
10
20
30
Smokes (packs/year)
60
40
20
0
0
10
20
30
Smokes (packs/year)
200
150
100
50
0
0
10
20
30
Number of sexual partners
20
10
0
0
10
20
30
Number of sexual partners
100
50
0
0
10
20
30
First sexual intercourse
15
10
5
0
0
10
20
30
First sexual intercourse
200
150
100
50
0
0
5
10
Num of pregnancies
20
10
0
0
5
10
Num of pregnancies
200
150
100
50
20
10



```
In [8]: print("Figure 2: EDA for Binary/Categorical Features")
        binary_charts
```

```
Figure 2: EDA for Binary/Categorical Features
```

Out[8]:



In [9]:

```
train_df.loc[:,['Smokes', 'Smokes (years)', 'Smokes (packs/year)']].corr('spearman').sty
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/IPython/core/formatters.py:342,
 in BaseFormatter.__call__(self, obj)
    340     method = get_real_method(obj, self.print_method)
    341     if method is not None:
--> 342         return method()
    343     return None
    344 else:

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:272,
 in Styler._repr_html_(self)
    267 """
    268 Hooks into Jupyter notebook rich display system, which calls _repr_html_ by
    269 default if an object is returned at the end of a cell.
    270 """
    271 if get_option("styler.render.repr") == "html":
--> 272     return self.to_html()
    273 return None

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:1179
, in Styler.to_html(self, buf, table_uuid, table_attributes, sparse_index, sparse_column
s, bold_headers, caption, max_rows, max_columns, encoding, doctype_html, exclude_styles,
 **kwargs)
   1176     obj.set_caption(caption)
   1178 # Build HTML string..
-> 1179 html = obj._render_html(
   1180     sparse_index=sparse_index,
   1181     sparse_columns=sparse_columns,
   1182     max_rows=max_rows,
   1183     max_cols=max_columns,
   1184     exclude_styles=exclude_styles,
   1185     encoding=encoding or get_option("styler.render.encoding"),
   1186     doctype_html=doctype_html,
   1187     **kwargs,
   1188 )
   1190 return save_to_buffer(
   1191     html, buf=buf, encoding=(encoding if buf is not None else None)
   1192 )

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style_render.
py:162, in StylerRenderer._render_html(self, sparse_index, sparse_columns, max_rows, max
_cols, **kwargs)
    150 def _render_html(
    151     self,
    152     sparse_index: bool,
   (...)
    156     **kwargs,
    157 ) -> str:
    158     """
    159     Renders the ``Styler`` including all applied styles to HTML.
    160     Generates a dict with necessary kwargs passed to jinja2 template.
    161     """
--> 162     self._compute()
    163     # TODO: namespace all the pandas keys
    164     d = self._translate(sparse_index, sparse_columns, max_rows, max_cols)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style_render.
py:205, in StylerRenderer._compute(self)
    203 r = self
    204 for func, args, kwargs in self._todo:
```

```
--> 205     r = func(self)(*args, **kwargs)
    206 return r

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:1442
, in Styler._apply(self, func, axis, subset, **kwargs)
   1440 axis = self.data._get_axis_number(axis)
   1441 if axis == 0:
-> 1442     result = data.apply(func, axis=0, **kwargs)
   1443 else:
   1444     result = data.T.apply(func, axis=0, **kwargs).T  # see GH 42005

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/frame.py:8848, in D
ataFrame.apply(self, func, axis, raw, result_type, args, **kwargs)
   8837 from pandas.core.apply import frame_apply
   8839 op = frame_apply(
   8840     self,
   8841     func=func,
   (...)
   8846     kwargs=kwargs,
   8847 )
-> 8848 return op.apply().__finalize__(self, method="apply")

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:733, in Fr
ameApply.apply(self)
    730 elif self.raw:
    731     return self.apply_raw()
--> 733 return self.apply_standard()

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:857, in Fr
ameApply.apply_standard(self)
    856 def apply_standard(self):
--> 857     results, res_index = self.apply_series_generator()
    859     # wrap results
    860     return self.wrap_results(results, res_index)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:873, in Fr
ameApply.apply_series_generator(self)
    870 with option_context("mode.chained_assignment", None):
    871     for i, v in enumerate(series_gen):
    872         # ignore SettingWithCopy here in case the user mutates
--> 873         results[i] = self.f(v)
    874         if isinstance(results[i], ABCSeries):
    875             # If we have a view on v, we need to make a copy because
    876             #  series_generator will swap out the underlying data
    877             results[i] = results[i].copy(deep=False)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:138, in Ap
ply.__init__.<locals>.f(x)
    137 def f(x):
--> 138     return func(x, *args, **kwargs)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:3554
, in _background_gradient(data, cmap, low, high, text_color_threshold, vmin, vmax, gmap,
 text_only)
   3551 else:  # else validate gmap against the underlying data
   3552     gmap = _validate_apply_axis_arg(gmap, "gmap", float, data)
-> 3554 with _mpl(Styler.background_gradient) as (plt, mpl):
   3555     smin = np.nanmin(gmap) if vmin is None else vmin
   3556     smax = np.nanmax(gmap) if vmax is None else vmax

File ~/miniconda3/envs/rfcc/lib/python3.10/contextlib.py:135, in _GeneratorContextManage
r.__enter__(self)
```

```
    133 del self.args, self.kwds, self.func
    134 try:
--> 135     return next(self.gen)
    136 except StopIteration:
    137     raise RuntimeError("generator didn't yield") from None

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:82,
 in _mpl(func)
     80     yield plt, mpl
     81 else:
---> 82     raise ImportError(no_mpl_message.format(func.__name__))

ImportError: background_gradient requires matplotlib.
```

```
Out[9]: <pandas.io.formats.style.Styler at 0x7f1700b47820>
```

```
In [10]: stds = ['STDs:condylomatosis', 'STDs:vaginal condylomatosis',
                 'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic inflammatory
                 'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:HIV',
                 'STDs:Hepatitis B', 'STDs:HPV']
         train_df.loc[:, stds].corr('spearman').style.background_gradient()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/IPython/core/formatters.py:342,
 in BaseFormatter.__call__(self, obj)
    340     method = get_real_method(obj, self.print_method)
    341     if method is not None:
--> 342         return method()
    343     return None
    344 else:

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:272,
 in Styler._repr_html_(self)
    267 """
    268 Hooks into Jupyter notebook rich display system, which calls _repr_html_ by
    269 default if an object is returned at the end of a cell.
    270 """
    271 if get_option("styler.render.repr") == "html":
--> 272     return self.to_html()
    273 return None

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:1179
, in Styler.to_html(self, buf, table_uuid, table_attributes, sparse_index, sparse_column
s, bold_headers, caption, max_rows, max_columns, encoding, doctype_html, exclude_styles,
 **kwargs)
   1176     obj.set_caption(caption)
   1178 # Build HTML string..
-> 1179 html = obj._render_html(
   1180     sparse_index=sparse_index,
   1181     sparse_columns=sparse_columns,
   1182     max_rows=max_rows,
   1183     max_cols=max_columns,
   1184     exclude_styles=exclude_styles,
   1185     encoding=encoding or get_option("styler.render.encoding"),
   1186     doctype_html=doctype_html,
   1187     **kwargs,
   1188 )
   1190 return save_to_buffer(
   1191     html, buf=buf, encoding=(encoding if buf is not None else None)
   1192 )

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style_render.
py:162, in StylerRenderer._render_html(self, sparse_index, sparse_columns, max_rows, max
_cols, **kwargs)
    150 def _render_html(
    151     self,
    152     sparse_index: bool,
   (...)
    156     **kwargs,
    157 ) -> str:
    158     """
    159     Renders the ``Styler`` including all applied styles to HTML.
    160     Generates a dict with necessary kwargs passed to jinja2 template.
    161     """
--> 162     self._compute()
    163     # TODO: namespace all the pandas keys
    164     d = self._translate(sparse_index, sparse_columns, max_rows, max_cols)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style_render.
py:205, in StylerRenderer._compute(self)
    203 r = self
    204 for func, args, kwargs in self._todo:
```

```
--> 205     r = func(self)(*args, **kwargs)
    206 return r

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:1442, in Styler._apply(self, func, axis, subset, **kwargs)
   1440 axis = self.data._get_axis_number(axis)
   1441 if axis == 0:
-> 1442     result = data.apply(func, axis=0, **kwargs)
   1443 else:
   1444     result = data.T.apply(func, axis=0, **kwargs).T  # see GH 42005

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/frame.py:8848, in DataFrame.apply(self, func, axis, raw, result_type, args, **kwargs)
   8837 from pandas.core.apply import frame_apply
   8839 op = frame_apply(
   8840     self,
   8841     func=func,
   (...)
   8846     kwargs=kwargs,
   8847 )
-> 8848 return op.apply().__finalize__(self, method="apply")

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:733, in FrameApply.apply(self)
    730 elif self.raw:
    731     return self.apply_raw()
--> 733 return self.apply_standard()

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:857, in FrameApply.apply_standard(self)
    856 def apply_standard(self):
--> 857     results, res_index = self.apply_series_generator()
    859     # wrap results
    860     return self.wrap_results(results, res_index)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:873, in FrameApply.apply_series_generator(self)
    870 with option_context("mode.chained_assignment", None):
    871     for i, v in enumerate(series_gen):
    872         # ignore SettingWithCopy here in case the user mutates
--> 873         results[i] = self.f(v)
    874         if isinstance(results[i], ABCSeries):
    875             # If we have a view on v, we need to make a copy because
    876             #  series_generator will swap out the underlying data
    877             results[i] = results[i].copy(deep=False)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/core/apply.py:138, in Apply.__init__.<locals>.f(x)
    137 def f(x):
--> 138     return func(x, *args, **kwargs)

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:3554, in _background_gradient(data, cmap, low, high, text_color_threshold, vmin, vmax, gmap, text_only)
   3551 else:  # else validate gmap against the underlying data
   3552     gmap = _validate_apply_axis_arg(gmap, "gmap", float, data)
-> 3554 with _mpl(Styler.background_gradient) as (plt, mpl):
   3555     smin = np.nanmin(gmap) if vmin is None else vmin
   3556     smax = np.nanmax(gmap) if vmax is None else vmax

File ~/miniconda3/envs/rfcc/lib/python3.10/contextlib.py:135, in _GeneratorContextManager.__enter__(self)
```

```
    133 del self.args, self.kwds, self.func
    134 try:
--> 135     return next(self.gen)
    136 except StopIteration:
    137     raise RuntimeError("generator didn't yield") from None

File ~/miniconda3/envs/rfcc/lib/python3.10/site-packages/pandas/io/formats/style.py:82,
 in _mpl(func)
     80     yield plt, mpl
     81 else:
---> 82     raise ImportError(no_mpl_message.format(func.__name__))

ImportError: background_gradient requires matplotlib.
```

```
Out[10]: <pandas.io.formats.style.Styler at 0x7f1700b46740>
```

References

Dua, Dheeru, and Casey Graff. 2017. “UCI Machine Learning Repository.” University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml.

Fernandes, K., Cardoso, J.S., & Fernandes, J.C. (2017). Transfer Learning with Partial Observability Applied to Cervical Cancer Screening. Iberian Conference on Pattern Recognition and Image Analysis. https://www.semanticscholar.org/paper/Transfer-Learning-with-Partial-Observability-to-Fernandes-Cardoso/1c02438ba4dfa775399ba414508e9cd335b69012

Cervical cancer (Risk Factors) Data Set
https://archive.ics.uci.edu/ml/datasets/Cervical+cancer+%28Risk+Factors%29