Installing Libraries

In [2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Importing the Dataset

In [3]:
```
from google.colab import drive
drive.mount("/content/gdrive")
data = pd.read_csv('/content/gdrive/My Drive/Final Year Project/cervicalcancer.cs
print(data.shape)
data
```

```
Mounted at /content/gdrive
(835, 36)
```

Out[3]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Con
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	
2	34	1.0	NaN	1.0	0.0	0.0	0.0	0.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	...
830	34	3.0	18.0	0.0	0.0	0.0	0.0	0.0	
831	32	2.0	19.0	1.0	0.0	0.0	0.0	1.0	
832	25	2.0	17.0	0.0	0.0	0.0	0.0	1.0	
833	33	2.0	24.0	2.0	0.0	0.0	0.0	1.0	
834	29	2.0	20.0	1.0	0.0	0.0	0.0	1.0	

835 rows × 36 columns

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 835 entries, 0 to 834
Data columns (total 36 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Age                                 835 non-null    int64  
 1   Number of sexual partners           810 non-null    float64
 2   First sexual intercourse            828 non-null    float64
 3   Num of pregnancies                  779 non-null    float64
 4   Smokes                              822 non-null    float64
 5   Smokes (years)                      822 non-null    float64
 6   Smokes (packs/year)                 822 non-null    float64
 7   Hormonal Contraceptives             732 non-null    float64
 8   Hormonal Contraceptives (years)     732 non-null    float64
 9   IUD                                 723 non-null    float64
 10  IUD (years)                         723 non-null    float64
 11  STDs                                735 non-null    float64
 12  STDs (number)                       735 non-null    float64
 13  STDs:condylomatosis                 735 non-null    float64
 14  STDs:cervical condylomatosis        735 non-null    float64
 15  STDs:vaginal condylomatosis         735 non-null    float64
 16  STDs:vulvo-perineal condylomatosis  735 non-null    float64
 17  STDs:syphilis                       735 non-null    float64
 18  STDs:pelvic inflammatory disease    735 non-null    float64
 19  STDs:genital herpes                 735 non-null    float64
 20  STDs:molluscum contagiosum          735 non-null    float64
 21  STDs:AIDS                           735 non-null    float64
 22  STDs:HIV                            735 non-null    float64
 23  STDs:Hepatitis B                    735 non-null    float64
 24  STDs:HPV                            735 non-null    float64
 25  STDs: Number of diagnosis           835 non-null    int64  
 26  STDs: Time since first diagnosis    71 non-null     float64
 27  STDs: Time since last diagnosis     71 non-null     float64
 28  Dx:Cancer                           835 non-null    int64  
 29  Dx:CIN                              835 non-null    int64  
 30  Dx:HPV                              835 non-null    int64  
 31  Dx                                  835 non-null    int64  
 32  Hinselmann                          835 non-null    int64  
 33  Schiller                            835 non-null    int64  
 34  Citology                            835 non-null    int64  
 35  Biopsy                              835 non-null    int64  
dtypes: float64(26), int64(10)
memory usage: 235.0 KB
```

In [5]:
```
data.describe()
```

Out[5]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Cor
count	835.000000	810.000000	828.000000	779.000000	822.000000	822.000000	822.000000	
mean	27.023952	2.551852	17.020531	2.304236	0.149635	1.253850	0.465823	
std	8.482986	1.676686	2.817000	1.455817	0.356930	4.140727	2.256273	
min	13.000000	1.000000	10.000000	0.000000	0.000000	0.000000	0.000000	
25%	21.000000	2.000000	15.000000	1.000000	0.000000	0.000000	0.000000	
50%	26.000000	2.000000	17.000000	2.000000	0.000000	0.000000	0.000000	
75%	32.000000	3.000000	18.000000	3.000000	0.000000	0.000000	0.000000	
max	84.000000	28.000000	32.000000	11.000000	1.000000	37.000000	37.000000	

8 rows × 36 columns

In [6]:
```
# Determining the null values in each column
data.isnull().sum()
```

Out[6]:
```
Age                                   0
Number of sexual partners            25
First sexual intercourse              7
Num of pregnancies                   56
Smokes                               13
Smokes (years)                       13
Smokes (packs/year)                  13
Hormonal Contraceptives             103
Hormonal Contraceptives (years)     103
IUD                                 112
IUD (years)                         112
STDs                                100
STDs (number)                       100
STDs:condylomatosis                 100
STDs:cervical condylomatosis        100
STDs:vaginal condylomatosis         100
STDs:vulvo-perineal condylomatosis   100
STDs:syphilis                       100
STDs:pelvic inflammatory disease    100
STDs:genital herpes                 100
STDs:molluscum contagiosum          100
STDs:AIDS                           100
STDs:HIV                            100
STDs:Hepatitis B                    100
STDs:HPV                            100
STDs: Number of diagnosis             0
STDs: Time since first diagnosis    764
STDs: Time since last diagnosis     764
Dx:Cancer                             0
Dx:CIN                                0
Dx:HPV                                0
Dx                                    0
Hinselmann                            0
Schiller                              0
Citology                              0
Biopsy                                0
dtype: int64
```

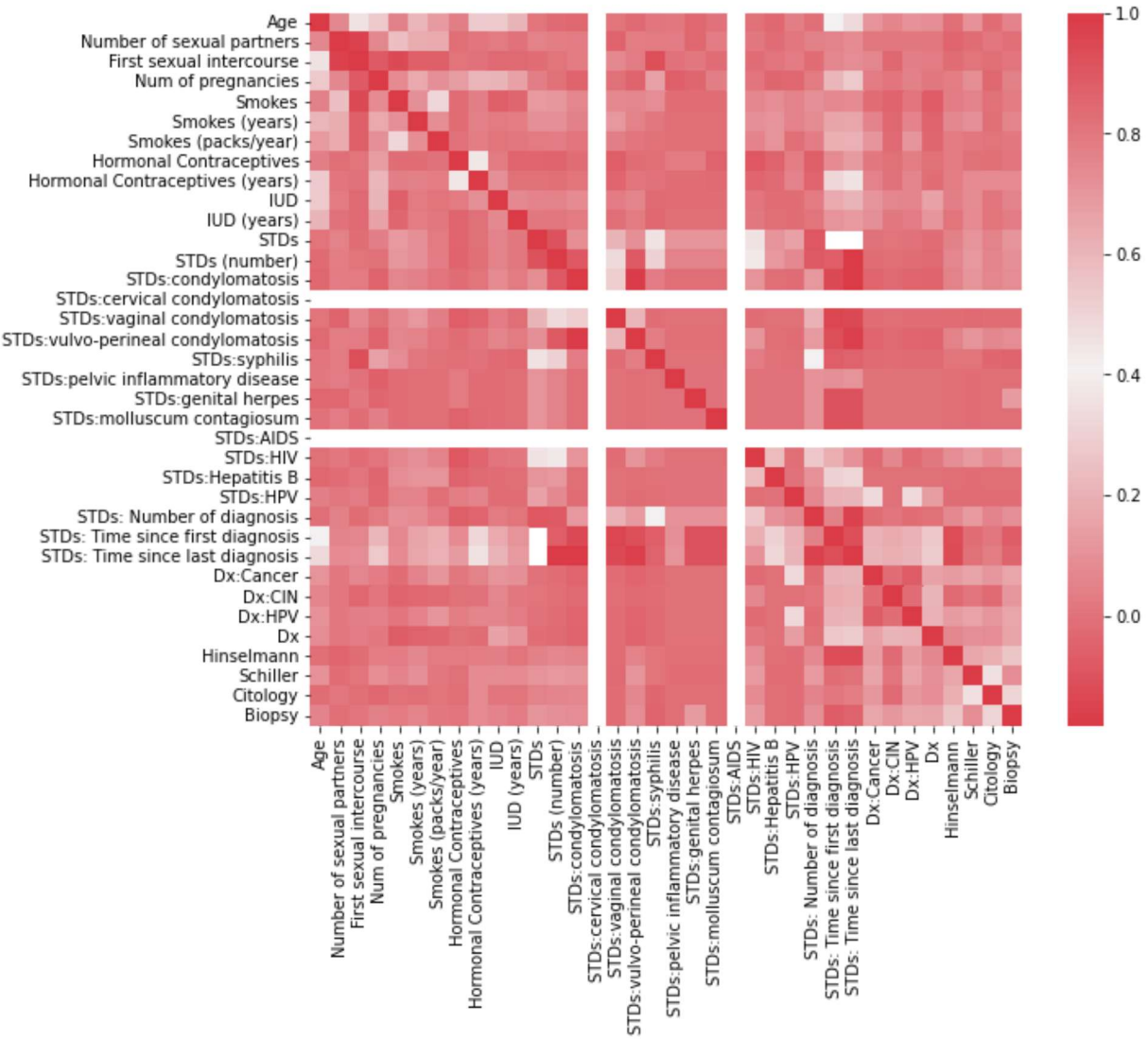
In [7]:
```
data.columns
```

Out[7]:
```
Index(['Age', 'Number of sexual partners', 'First sexual intercourse',
       'Num of pregnancies', 'Smokes', 'Smokes (years)', 'Smokes (packs/year)',
       'Hormonal Contraceptives', 'Hormonal Contraceptives (years)', 'IUD',
       'IUD (years)', 'STDs', 'STDs (number)', 'STDs:condylomatosis',
       'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
       'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis',
       'STDs:pelvic inflammatory disease', 'STDs:genital herpes',
       'STDs:molluscum contagiosum', 'STDs:AIDS', 'STDs:HIV',
       'STDs:Hepatitis B', 'STDs:HPV', 'STDs: Number of diagnosis',
       'STDs: Time since first diagnosis', 'STDs: Time since last diagnosis',
       'Dx:Cancer', 'Dx:CIN', 'Dx:HPV', 'Dx', 'Hinselmann', 'Schiller',
       'Citology', 'Biopsy'],
      dtype='object')
```

Correlation Plot

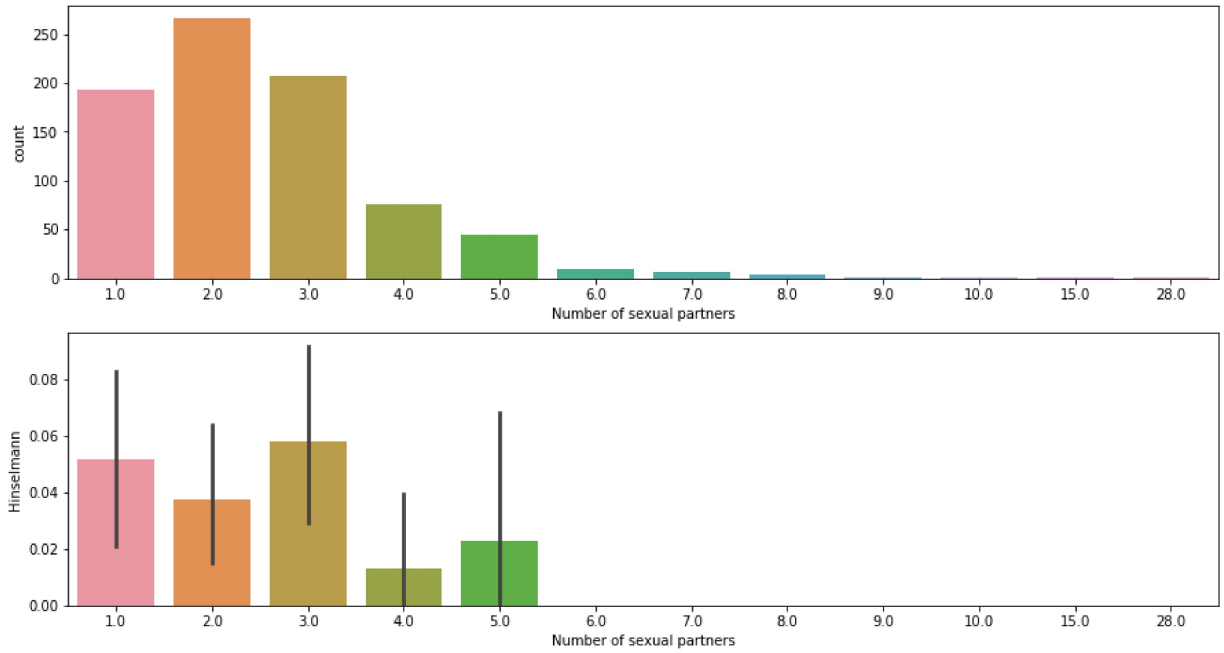
In [8]:
```
# correlation plot

f, ax = plt.subplots(figsize = (10, 8))

corr = data.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool),
            cmap = sns.diverging_palette(10, 10, as_cmap = True), square = True,
```

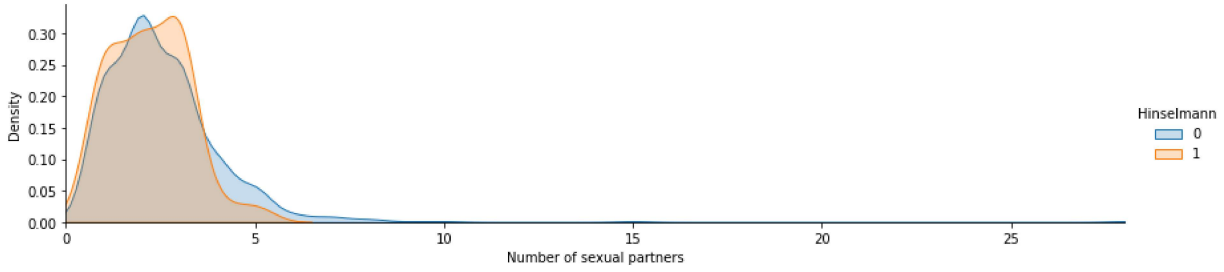
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f084f545b90>



In [9]:
```
# Hinselmann vs no. of sexual partners

#categorical to categorical
fig, (ax1,ax2) = plt.subplots(2, 1, figsize = (15, 8))
sns.countplot(x = 'Number of sexual partners', data = data, ax=ax1)
sns.barplot(x = 'Number of sexual partners', y = 'Hinselmann', data = data, ax=ax

#continuous to categorical
facet = sns.FacetGrid(data, hue='Hinselmann',aspect=4)
facet.map(sns.kdeplot,'Number of sexual partners',shade= True)
facet.set(xlim=(0, data['Number of sexual partners'].max()))
facet.add_legend()
```

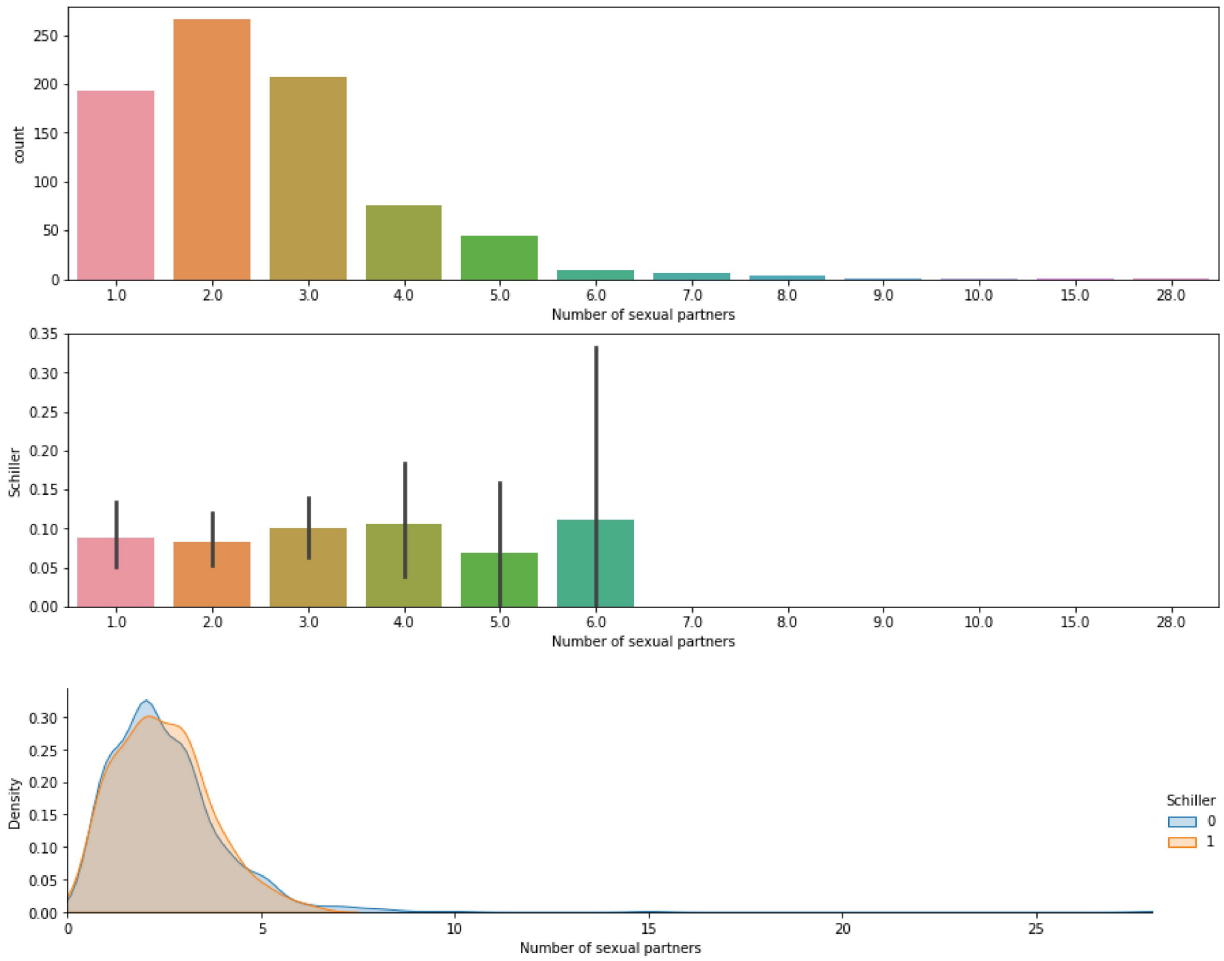
Out[9]: <seaborn.axisgrid.FacetGrid at 0x7f084ca1a790>





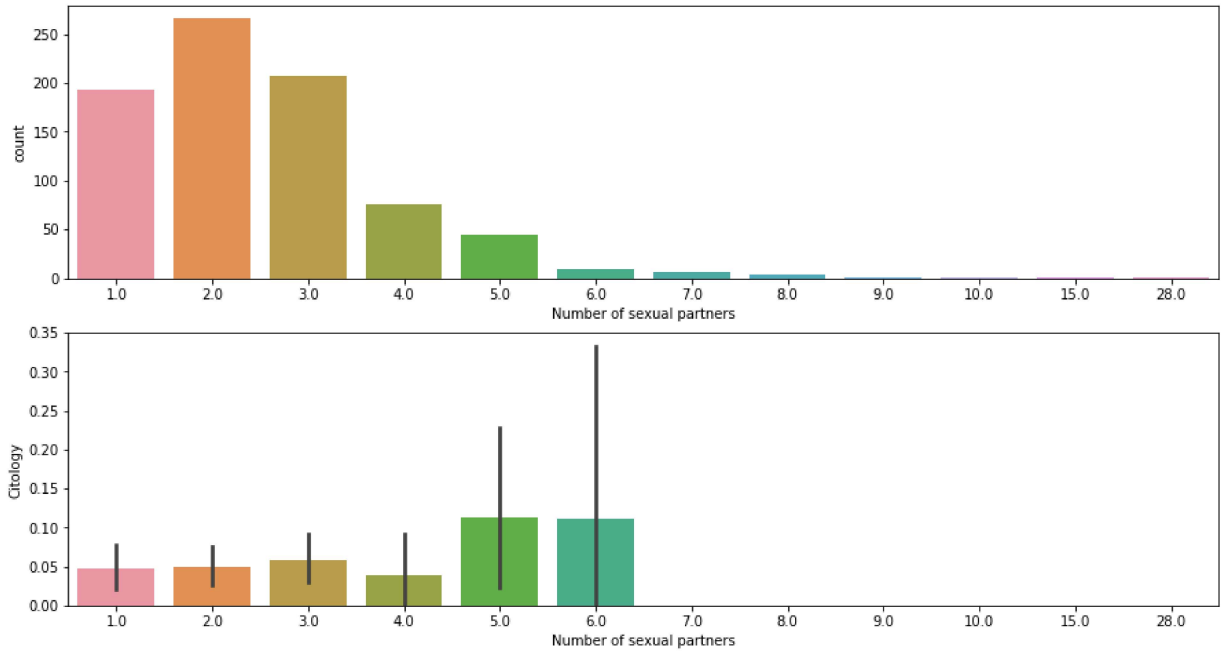
In [10]:
```
# Schiller vs no. of sexual partners

#categorical to categorical
fig, (ax1,ax2) = plt.subplots(2, 1, figsize = (15, 8))
sns.countplot(x = 'Number of sexual partners', data = data, ax=ax1)
sns.barplot(x = 'Number of sexual partners', y = 'Schiller', data = data, ax=ax2)

#continuous to categorical
facet = sns.FacetGrid(data, hue='Schiller',aspect=4)
facet.map(sns.kdeplot,'Number of sexual partners',shade= True)
facet.set(xlim=(0, data['Number of sexual partners'].max()))
facet.add_legend()
```

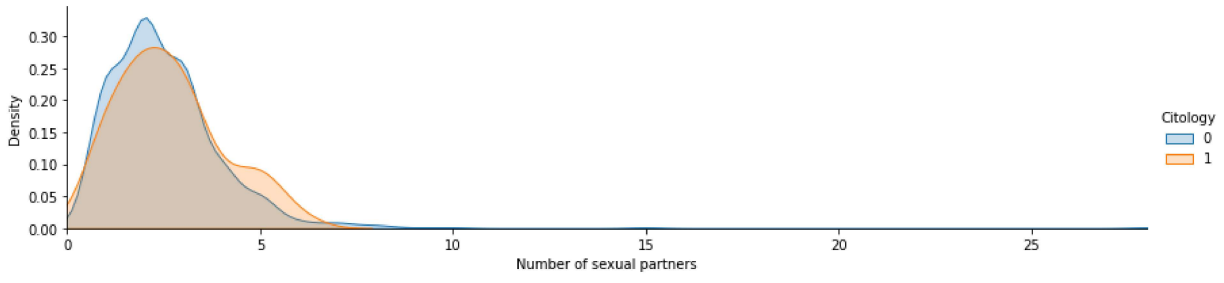
Out[10]: <seaborn.axisgrid.FacetGrid at 0x7f084ab67c10>



In [11]:
```
# Citology vs no. of sexual partners

#categorical to categorical
fig, (ax1,ax2) = plt.subplots(2, 1, figsize = (15, 8))
sns.countplot(x = 'Number of sexual partners', data = data, ax=ax1)
sns.barplot(x = 'Number of sexual partners', y = 'Citology', data = data, ax=ax2)

#continuous to categorical
facet = sns.FacetGrid(data, hue='Citology',aspect=4)
facet.map(sns.kdeplot,'Number of sexual partners',shade= True)
facet.set(xlim=(0, data['Number of sexual partners'].max()))
facet.add_legend()
```

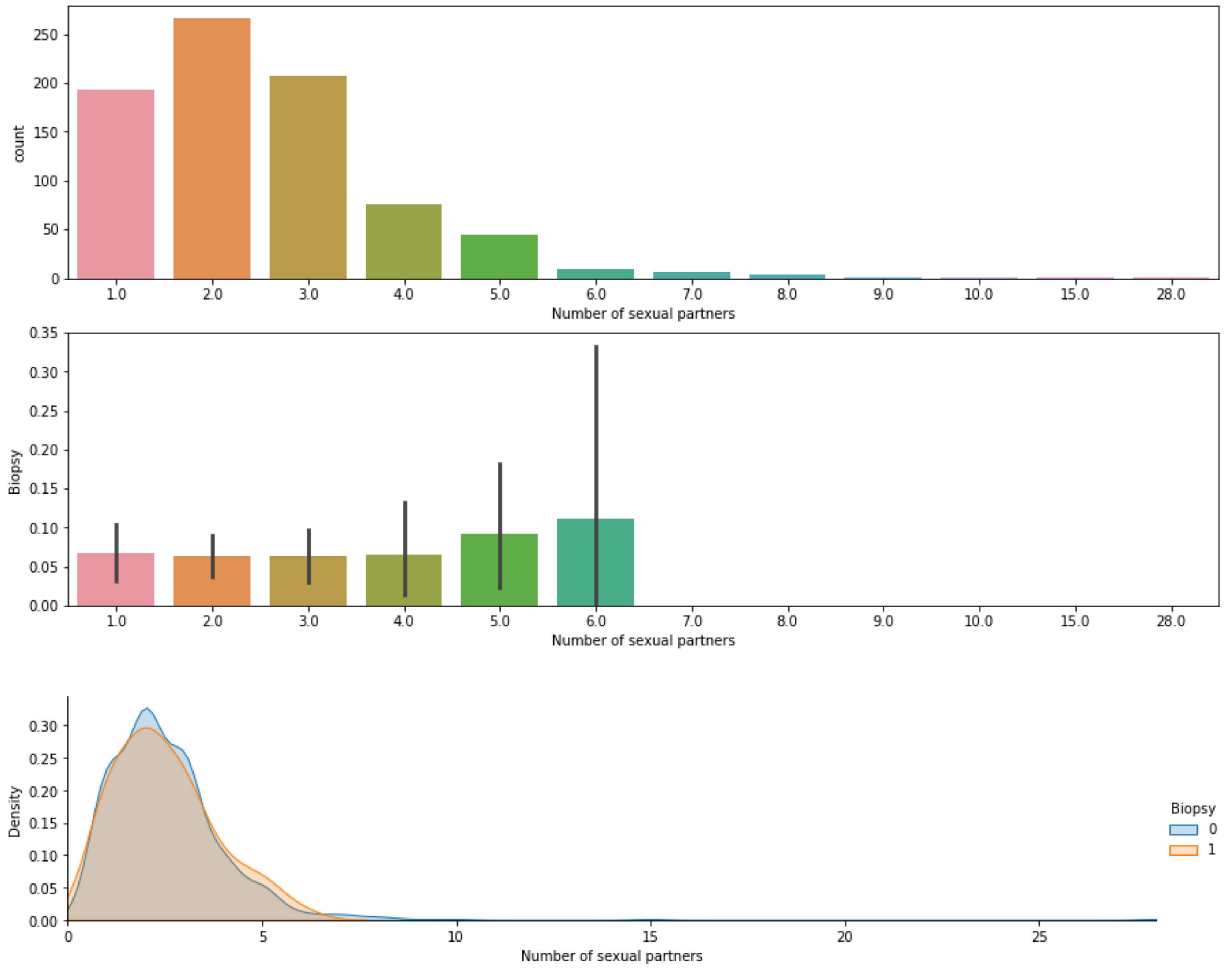
Out[11]: <seaborn.axisgrid.FacetGrid at 0x7f084ca1a450>





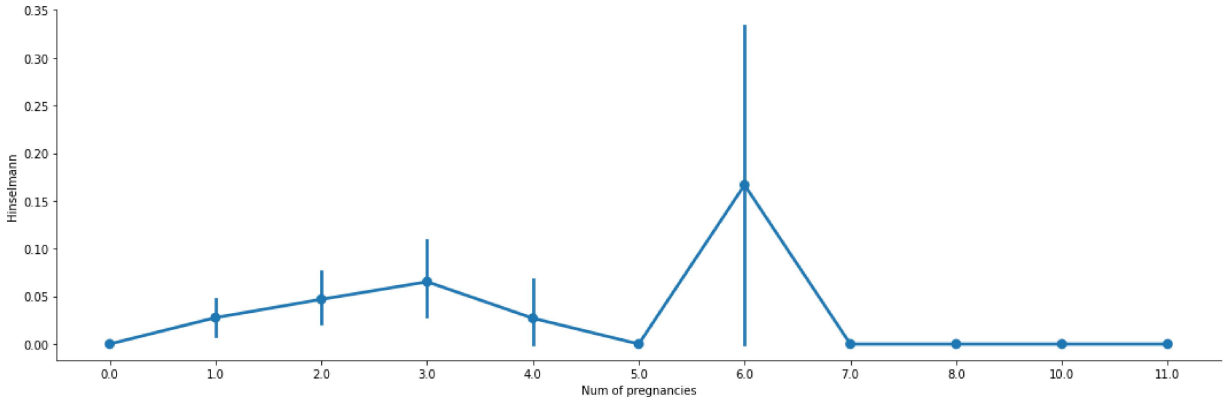
In [12]:
```
# Biopsy vs no. of sexual partners

#categorical to categorical
fig, (ax1,ax2) = plt.subplots(2, 1, figsize = (15, 8))
sns.countplot(x = 'Number of sexual partners', data = data, ax=ax1)
sns.barplot(x = 'Number of sexual partners', y = 'Biopsy', data = data, ax=ax2)

#continuous to categorical
facet = sns.FacetGrid(data, hue='Biopsy',aspect=4)
facet.map(sns.kdeplot,'Number of sexual partners',shade= True)
facet.set(xlim=(0, data['Number of sexual partners'].max()))
facet.add_legend()
```

Out[12]: <seaborn.axisgrid.FacetGrid at 0x7f084ca22110>



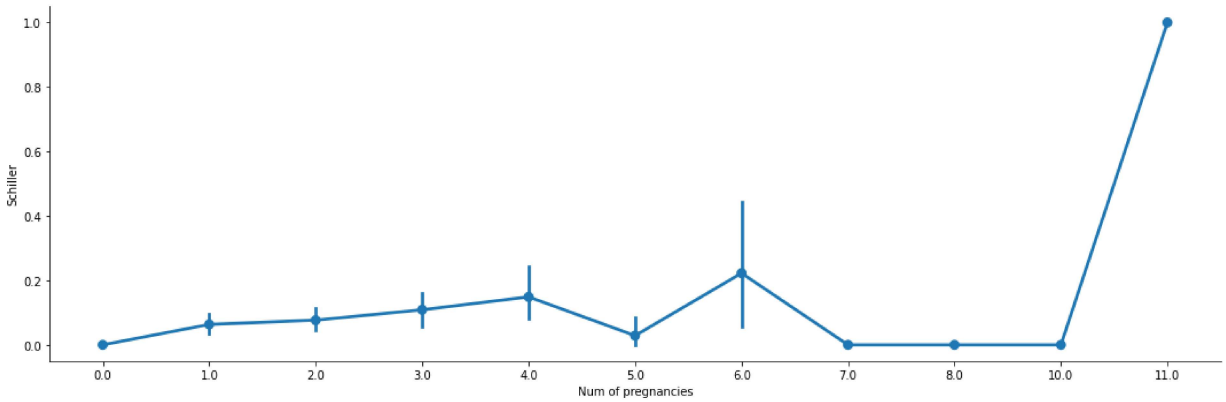
In [13]:
```
# Hinselmann vs no. of pregnancies

sns.factorplot('Num of pregnancies','Hinselmann',data = data, size=5, aspect=3)
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0x7f084ab2ba90>



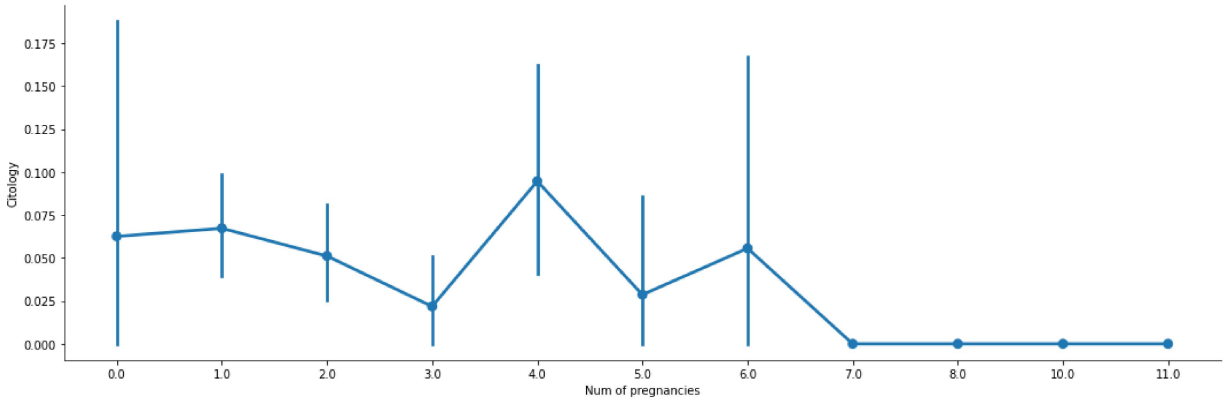
In [14]:
```
# Schiller vs no. of pregnancies

sns.factorplot('Num of pregnancies','Schiller',data = data, size=5, aspect=3)
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x7f084acf7b10>



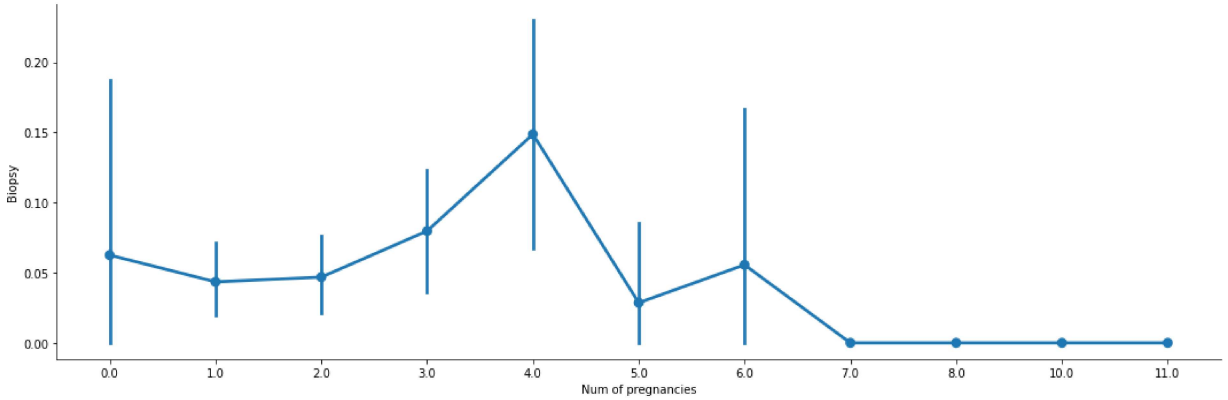
In [15]:
```
# Citology vs no. of pregnancies

sns.factorplot('Num of pregnancies','Citology',data = data, size=5, aspect=3)
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x7f084a887a10>



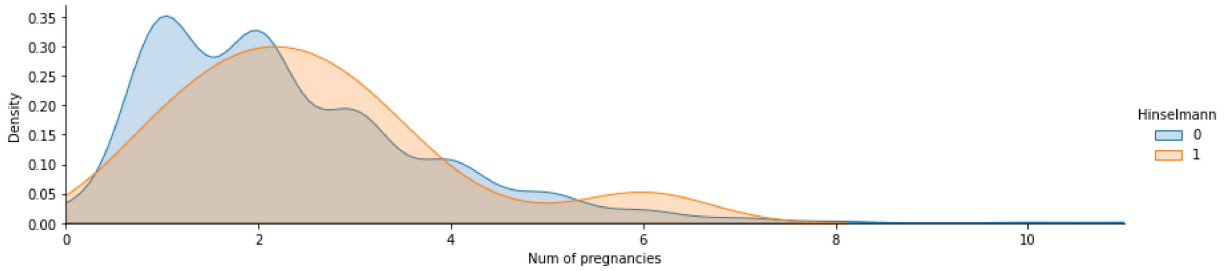
In [16]:
```
# Biopsy vs no. of pregnancies

sns.factorplot('Num of pregnancies','Biopsy',data = data, size=5, aspect=3)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x7f084ab4d310>



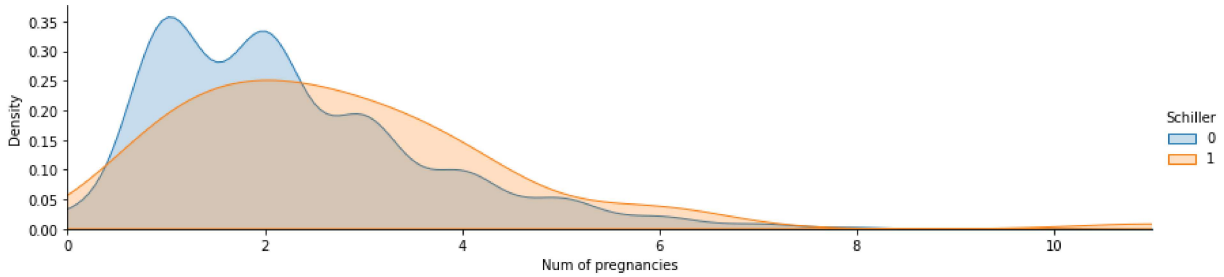
In [17]:
```
#continuous to categorical
facet = sns.FacetGrid(data, hue='Hinselmann', aspect=4)
facet.map(sns.kdeplot,'Num of pregnancies', shade= True)
facet.set(xlim=(0, data['Num of pregnancies'].max()))
facet.add_legend()
```

Out[17]: <seaborn.axisgrid.FacetGrid at 0x7f084a67d650>



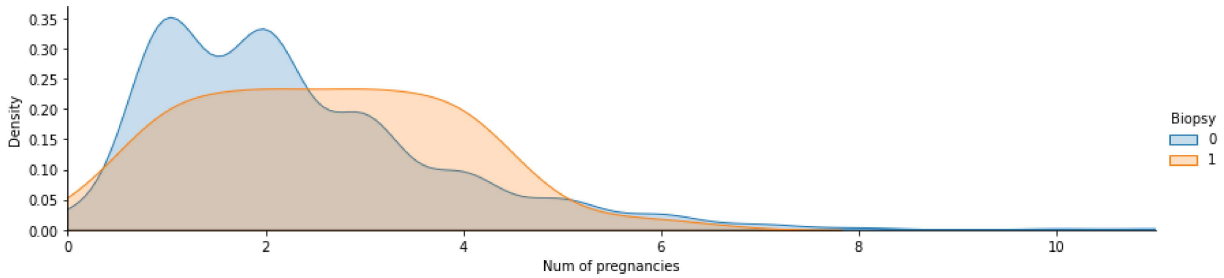
In [18]:
```
#continuous to categorical
facet = sns.FacetGrid(data, hue='Schiller', aspect=4)
facet.map(sns.kdeplot,'Num of pregnancies', shade= True)
facet.set(xlim=(0, data['Num of pregnancies'].max()))
facet.add_legend()
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x7f084a67d0d0>



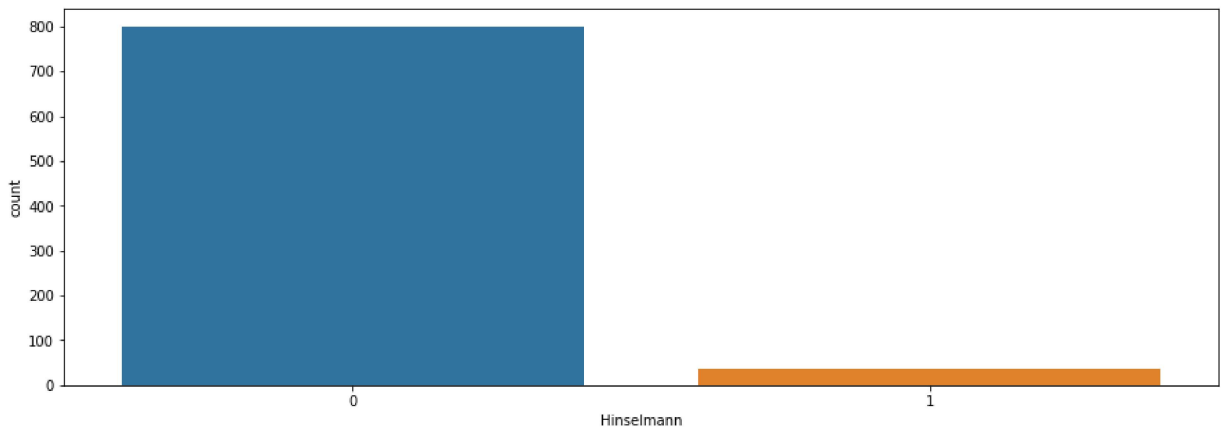
In [19]:
```
#continuous to categorical
facet = sns.FacetGrid(data, hue='Biopsy', aspect=4)
facet.map(sns.kdeplot,'Num of pregnancies', shade= True)
facet.set(xlim=(0, data['Num of pregnancies'].max()))
facet.add_legend()
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x7f084a51bad0>



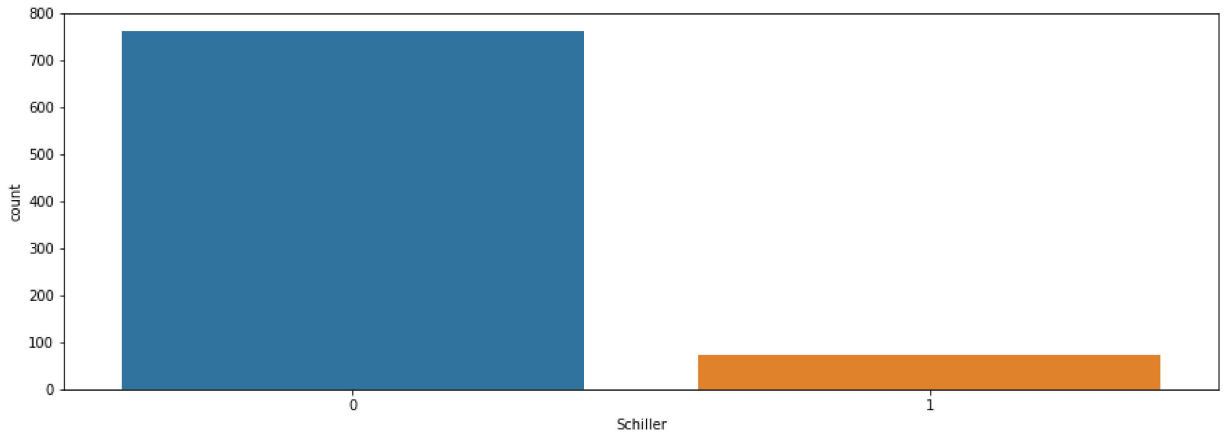
In [20]:
```
fig, (axis1) = plt.subplots(1, 1, figsize = (15, 5))
sns.countplot(x = 'Hinselmann', data=data, ax = axis1)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f086479b990>



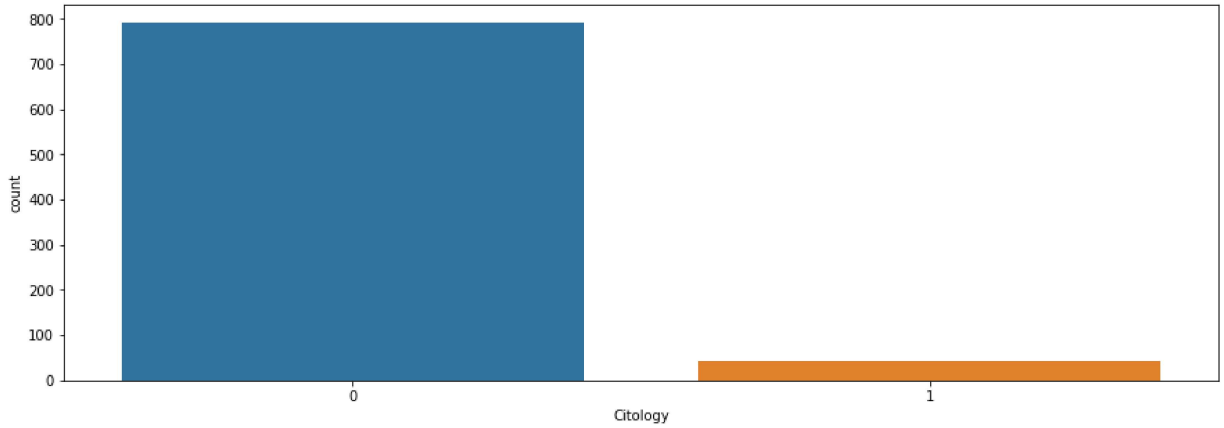
In [21]:
```
fig, (axis1) = plt.subplots(1, 1, figsize = (15, 5))
sns.countplot(x = 'Schiller', data=data, ax = axis1)
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f084a4c39d0>



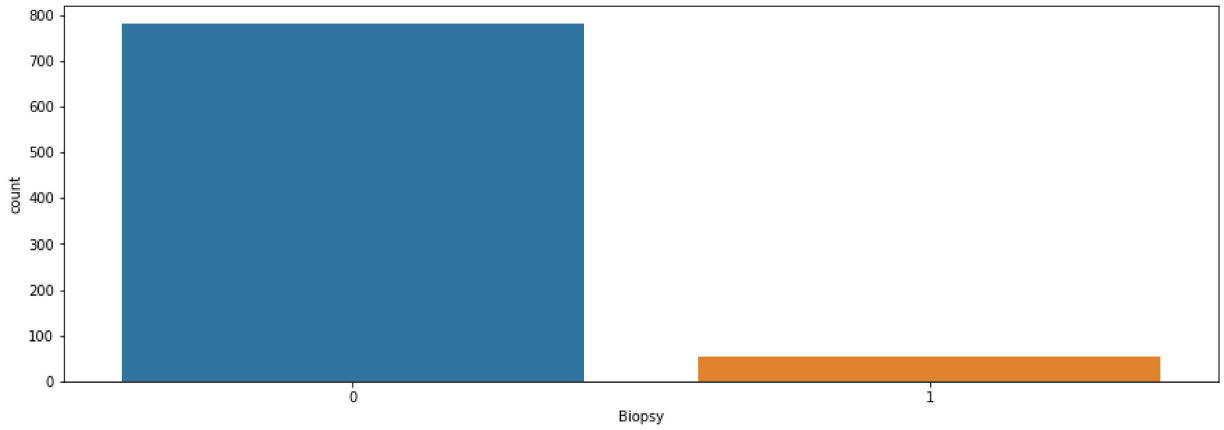
In [22]:
```
fig, (axis1) = plt.subplots(1, 1, figsize = (15, 5))
sns.countplot(x = 'Citology', data=data, ax = axis1)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f084a3b8710>

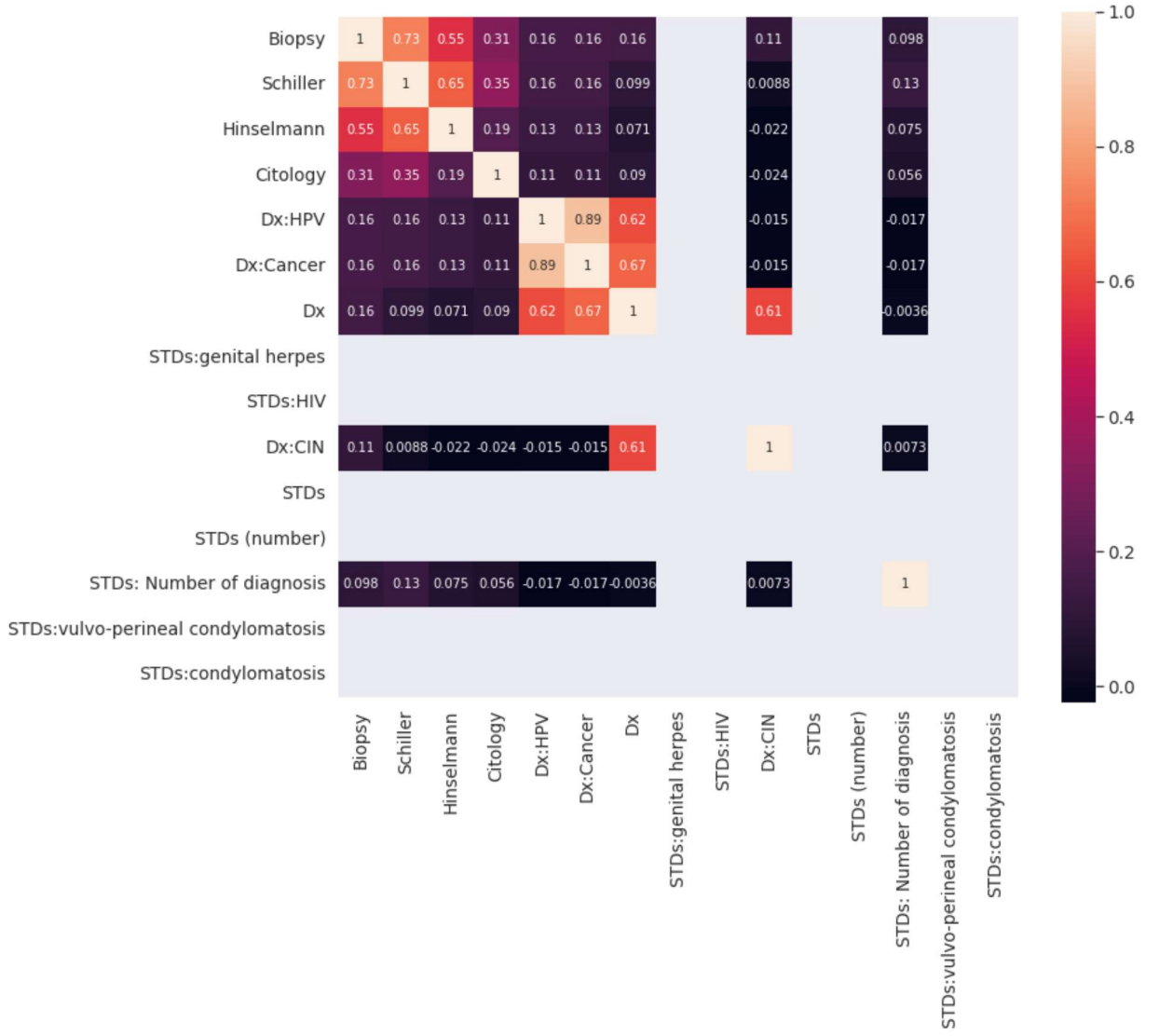


In [23]:
```
fig, (axis1) = plt.subplots(1, 1, figsize = (15, 5))
sns.countplot(x = 'Biopsy', data=data, ax = axis1)
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f084a3181d0>
```



In [24]:
```
# list the heatmap of top correlation

corr = data.corr()

# number of variables for heatmap
k = 15

cols = corr.nlargest(k, 'Biopsy')['Biopsy'].index
cm = np.corrcoef(data[cols].values.T)

plt.figure(figsize=(12, 10))

sns.set(font_scale=1.25)
sns.heatmap(cm, cbar = True, annot = True, square = True, annot_kws = {'size': 10
                 yticklabels = cols.values, xticklabels = cols.values)
plt.show()
```



Data Preprocessing

In [25]:
```
# Inputing the missing values from the given dataset
# we will impute the categorical variables with 0 or 1 and continuous variables w

data['Number of sexual partners'] = data['Number of sexual partners'].fillna(data
data['Number of sexual partners'].isnull().any()

# data['Number of sexual partners'].value_counts()
```

Out[25]: False

In [26]:
```
# Inputing the missing values from First sexual intercourse

data['First sexual intercourse'] = data['First sexual intercourse'].fillna(data['
data['First sexual intercourse'].isnull().any()

# data['First sexual intercourse'].value_counts()
```

Out[26]: False

In [27]:
```
# Inputing the missing values from Num of pregnancies

data['Num of pregnancies'] = data['Num of pregnancies'].fillna(data['Num of pregn
data['Num of pregnancies'].isnull().any()

# data['Num of pregnancies'].value_counts()
```

Out[27]: False

In [28]:
```
# Inputing the missing values from Smokes

data['Smokes'] = data['Smokes'].fillna(data['Smokes'].median())
data['Smokes'].isnull().any()

# data['Smokes'].value_counts()
```

Out[28]: False

In [29]:
```
# Inputing the missing values from Smokes (years)

data['Smokes (years)'] = data['Smokes (years)'].fillna(1)
data['Smokes (years)'].isnull().any()

# data['Smokes (years)'].value_counts()
```

Out[29]: False

In [30]:
```
# Inputing the missing values from Smokes (packs/year)

data['Smokes (packs/year)'] = data['Smokes (packs/year)'].fillna(data['Smokes (pa
data['Smokes (packs/year)'].isnull().any()

# data['Smokes (packs/year)'].value_counts()
```

Out[30]: False

In [31]:
```
# Inputing the missing values from Hormonal Contraceptives

data['Hormonal Contraceptives'] = data['Hormonal Contraceptives'].fillna(data['Ho
data['Hormonal Contraceptives'].isnull().any()

# data['Hormonal Contraceptives'].value_counts()
```

Out[31]: False

In [32]:
```
# Inputing the missing values from Hormonal Contraceptives (years)

data['Hormonal Contraceptives (years)'] = data['Hormonal Contraceptives (years)']
data['Hormonal Contraceptives (years)'].isnull().any()

# data['Hormonal Contraceptives (years)'].value_counts()
```

Out[32]: False

In [33]:
```
# Inputing the missing values from IUD

data['IUD'] = data['IUD'].fillna(0)
data['IUD'].isnull().any()

# data['IUD'].value_counts()
```

Out[33]: False

In [34]:
```
# Inputing the missing values from IUD (years)

data['IUD (years)'] = data['IUD (years)'].fillna(0)
data['IUD (years)'].isnull().any()

# data['IUD (years)'].value_counts()
```

Out[34]: False

In [35]:
```
# Inputing the missing values from STDs

data['STDs'] = data['STDs'].fillna(1)
data['STDs'].isnull().any()

# data['STDs'].value_counts()
```

Out[35]: False

In [36]:
```
# Inputing the missing values from STDs (number)

data['STDs (number)'] = data['STDs (number)'].fillna(data['STDs (number)'].median
data['STDs (number)'].isnull().any()

# data['STDs (number)'].value_counts()
```

Out[36]: False

In [37]:
```
# Inputing the missing values from STDs:condylomatosis

data['STDs:condylomatosis'] = data['STDs:condylomatosis'].fillna(data['STDs:condy
data['STDs:condylomatosis'].isnull().any()

# data['STDs:condylomatosis'].value_counts()
```

Out[37]: False

In [38]:
```
# Inputing the missing values from STDs:cervical condylomatosis

data['STDs:cervical condylomatosis'] = data['STDs:cervical condylomatosis'].fillr
data['STDs:cervical condylomatosis'].isnull().any()

# data['STDs:cervical condylomatosis'].value_counts()
```

Out[38]: False

In [39]:
```
# Inputing the missing values from STDs:vaginal condylomatosis

data['STDs:vaginal condylomatosis'] = data['STDs:vaginal condylomatosis'].fillna(
data['STDs:vaginal condylomatosis'].isnull().any()

# data['STDs:vaginal condylomatosis'].value_counts()
```

Out[39]: False

In [40]:
```
# Inputing the missing values from STDs:vulvo-perineal condylomatosis

data['STDs:vulvo-perineal condylomatosis'] = data['STDs:vulvo-perineal condylomat
data['STDs:vulvo-perineal condylomatosis'].isnull().any()

# data['STDs:vulvo-perineal condylomatosis'].value_counts()
```

Out[40]: False

In [41]:
```
# Inputing the missing values from STDs:syphilis

data['STDs:syphilis'] = data['STDs:syphilis'].fillna(data['STDs:syphilis'].median
data['STDs:syphilis'].isnull().any()

# data['STDs:syphilis'].value_counts()
```

Out[41]: False

In [42]:
```
# Inputing the missing values from STDs:pelvic inflammatory diseases

data['STDs:pelvic inflammatory disease'] = data['STDs:pelvic inflammatory disease
data['STDs:pelvic inflammatory disease'].isnull().any()

# data['STDs:pelvic inflammatory disease'].value_counts()
```

Out[42]: False

In [43]:
```
# Inputing the missing values from STDs:genital herpes

data['STDs:genital herpes'] = data['STDs:genital herpes'].fillna(data['STDs:genit
data['STDs:genital herpes'].isnull().any()

# data['STDs:genital herpes'].value_counts()
```

Out[43]: False

In [44]:
```
# Inputing the missing values from STDs:molluscum contagiosum

data['STDs:molluscum contagiosum'] = data['STDs:molluscum contagiosum'].fillna(da
data['STDs:molluscum contagiosum'].isnull().any()

# data['STDs:molluscum contagiosum'].value_counts()
```

Out[44]: False

In [45]:
```
# Inputing the missing values from STDs:AIDS

data['STDs:AIDS'] = data['STDs:AIDS'].fillna(data['STDs:AIDS'].median())
data['STDs:AIDS'].isnull().any()

# data['STDs:AIDS'].value_counts()
```

Out[45]: False

In [46]:
```
# Inputing the missing values from STDs:HIV

data['STDs:HIV'] = data['STDs:HIV'].fillna(data['STDs:HIV'].median())
data['STDs:HIV'].isnull().any()

# data['STDs:HIV'].value_counts()
```

Out[46]: False

In [47]:
```
# Inputing the missing values from STDs:Hepatitis B

data['STDs:Hepatitis B'] = data['STDs:Hepatitis B'].fillna(data['STDs:Hepatitis B
data['STDs:Hepatitis B'].isnull().any()

# data['STDs:Hepatitis B'].value_counts()
```

Out[47]: False

In [48]:
```
# Inputing the missing values from STDs:HPV

data['STDs:HPV'] = data['STDs:HPV'].fillna(data['STDs:HPV'].median())
data['STDs:HPV'].isnull().any()

# data['STDs:HPV'].value_counts()
```

Out[48]: False

In [49]:
```
# Inputing the missing values from STDs: Time since first diagnosis

data['STDs: Time since first diagnosis'] = data['STDs: Time since first diagnosis
data['STDs: Time since first diagnosis'].isnull().any()

# data['STDs: Time since first diagnosis'].value_counts()
```

Out[49]: False

In [50]:
```
# Inputing the missing values from STDs: Time since last diagnosis

data['STDs: Time since last diagnosis'] = data['STDs: Time since last diagnosis']
data['STDs: Time since last diagnosis'].isnull().any()

# data['STDs: Time since last diagnosis'].value_counts()
```

Out[50]: False

In [51]:
```
data.describe()
```

Out[51]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Cor
count	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	835.000000	
mean	27.023952	2.535329	17.020359	2.283832	0.147305	1.249898	0.458571	
std	8.482986	1.654044	2.805154	1.408152	0.354623	4.108449	2.239363	
min	13.000000	1.000000	10.000000	0.000000	0.000000	0.000000	0.000000	
25%	21.000000	2.000000	15.000000	1.000000	0.000000	0.000000	0.000000	
50%	26.000000	2.000000	17.000000	2.000000	0.000000	0.000000	0.000000	
75%	32.000000	3.000000	18.000000	3.000000	0.000000	0.000000	0.000000	
max	84.000000	28.000000	32.000000	11.000000	1.000000	37.000000	37.000000	

8 rows × 36 columns

Data Visualization

In [52]:
```
fig, (ax1,ax2,ax3,ax4,ax5,ax6,ax7) = plt.subplots(7, 1, figsize = (20,40))
sns.countplot(x='Age', data=data, ax=ax1)
sns.countplot(x='Number of sexual partners', data=data, ax=ax2)
sns.countplot(x='Num of pregnancies', data=data, ax=ax3)
sns.countplot(x='Smokes (years)', data=data, ax=ax4)
sns.countplot(x='Hormonal Contraceptives (years)', data=data, ax=ax5)
sns.countplot(x='IUD (years)', data=data, ax=ax6)
sns.countplot(x='STDs (number)', data=data, ax=ax7)
```

Out[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f084a1ce0d0>
```

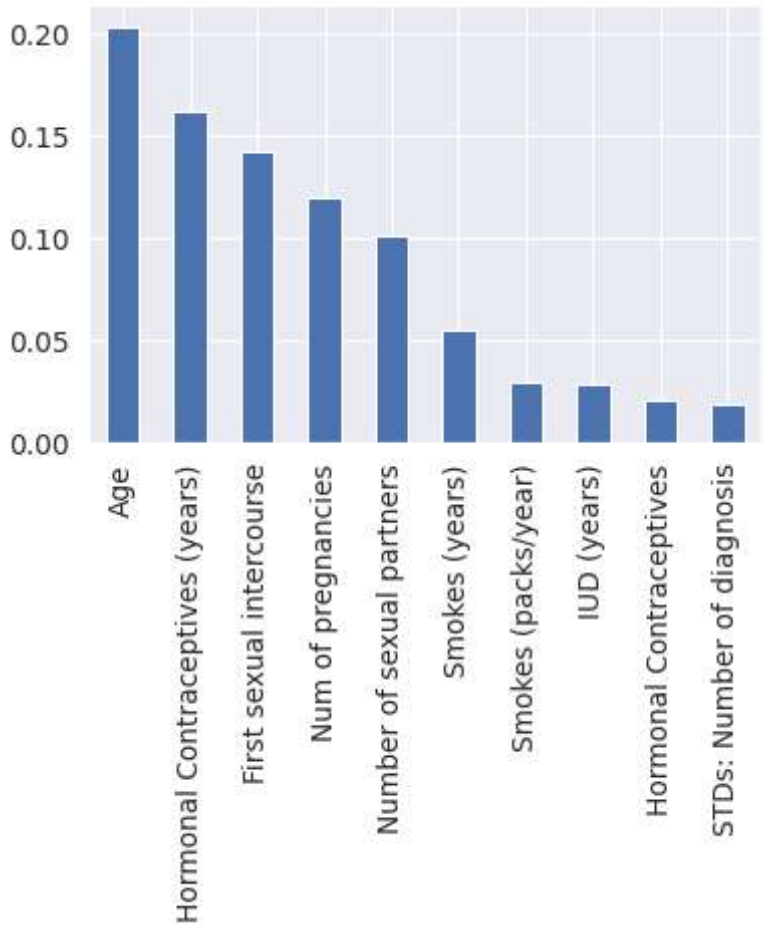
In [53]:
```
x=data[['Age', 'Number of sexual partners', 'First sexual intercourse','Num of pr
yh=data[['Hinselmann']]
ys=data[['Schiller']]
yc=data[['Citology']]
yb=data[['Biopsy']]
```

In [54]:
```
print(x.shape)
print(yh.shape)
print(ys.shape)
print(yc.shape)
print(yb.shape)
```

```
(835, 32)
(835, 1)
(835, 1)
(835, 1)
(835, 1)
```

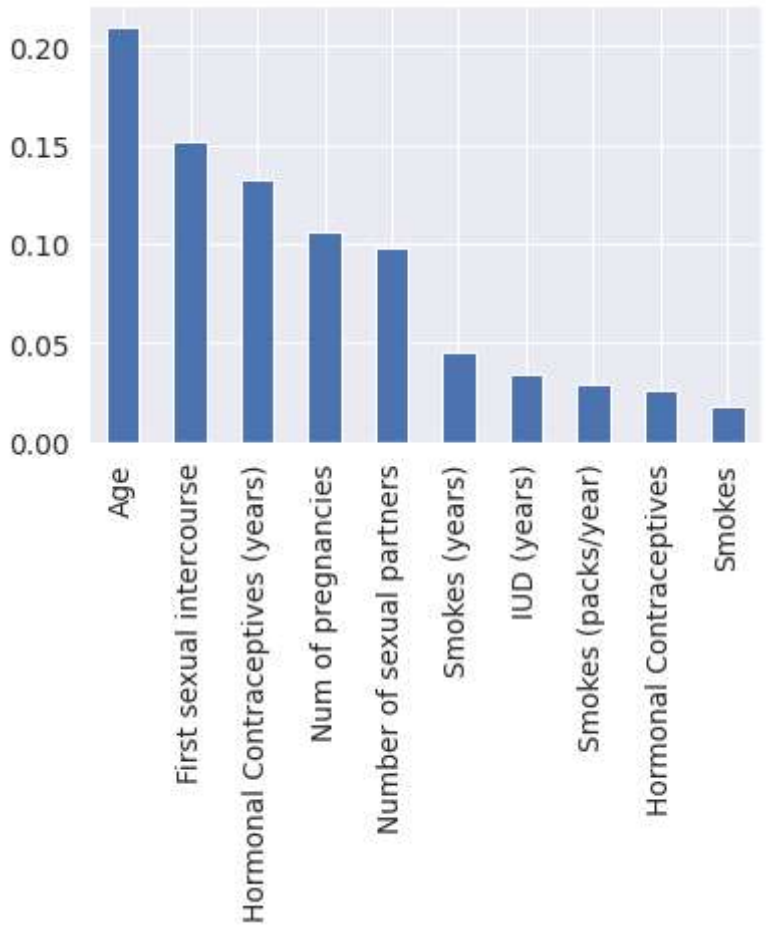
In [55]:
```
from sklearn.ensemble import ExtraTreesClassifier
# Building the model
model = ExtraTreesClassifier()

# Training the model
model.fit(x, yh)
col=x.columns
imp=pd.Series(model.feature_importances_,index=col)
imp
imp.nlargest(10).plot(kind='bar')
```

Out[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0849f70450>
```



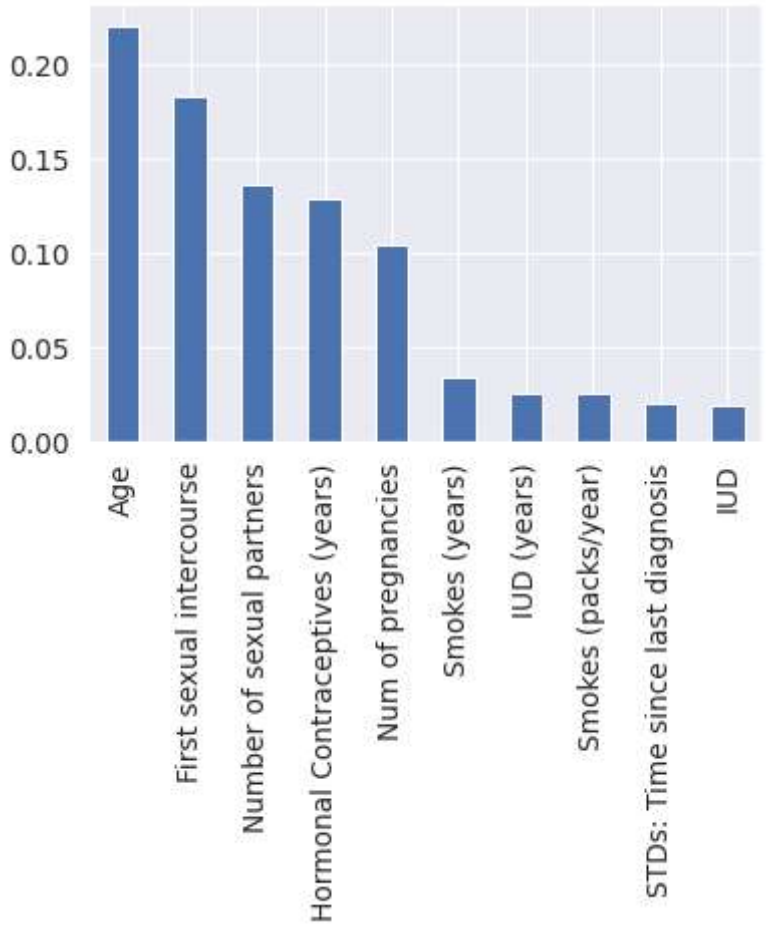
In [56]:
```
from sklearn.ensemble import ExtraTreesClassifier
# Building the model
model = ExtraTreesClassifier()

# Training the model
model.fit(x, ys)
col=x.columns
imp=pd.Series(model.feature_importances_,index=col)
imp
imp.nlargest(10).plot(kind='bar')
```

Out[56]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08493da910>
```



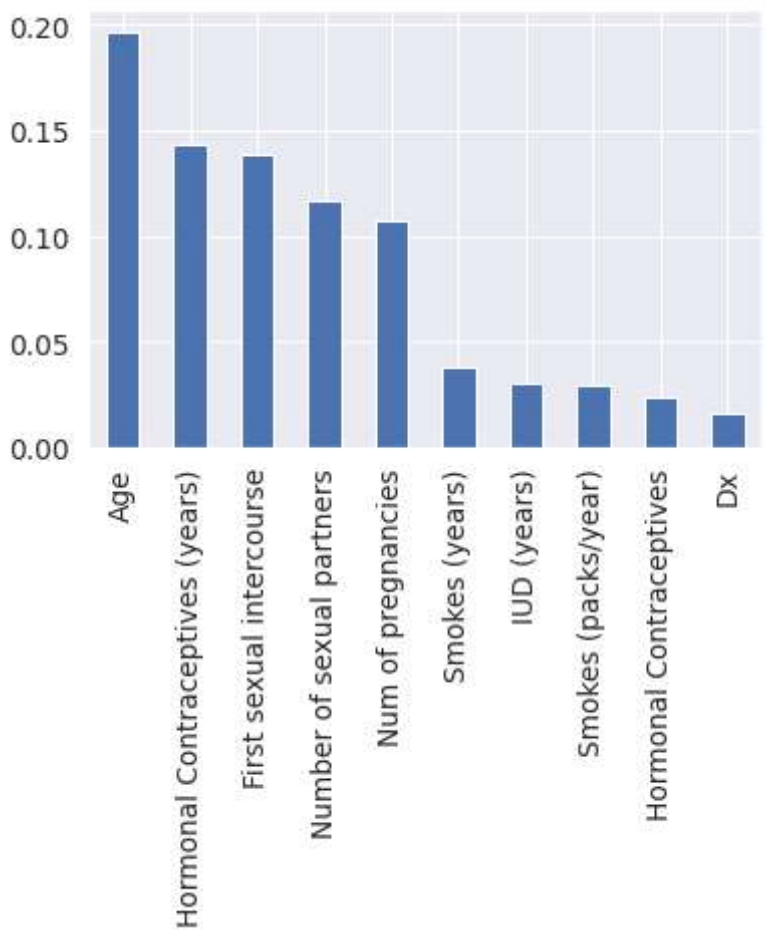
In [57]:
```
from sklearn.ensemble import ExtraTreesClassifier
# Building the model
model = ExtraTreesClassifier()

# Training the model
model.fit(x, yc)
col=x.columns
imp=pd.Series(model.feature_importances_,index=col)
imp
imp.nlargest(10).plot(kind='bar')
```

Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0849039590>



In [58]:
```
from sklearn.ensemble import ExtraTreesClassifier
# Building the model
model = ExtraTreesClassifier()

# Training the model
model.fit(x, yb)
col=x.columns
imp=pd.Series(model.feature_importances_,index=col)
imp
imp.nlargest(10).plot(kind='bar')
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x7f08490a0d90>



In [59]:
```
#Only considering the most important features for prediction

xnew=data[['Age','Number of sexual partners','First sexual intercourse','Num of p
```

For Target Variable HINSELMANN

In [60]:
```
# splitting the dataset into  training and test set for Hinselmann

from sklearn.model_selection import train_test_split

xh_train, xh_test, yh_train, yh_test = train_test_split(xnew, yh, test_size = 0.4

print(xh_train.shape)
print(yh_train.shape)
print(xh_test.shape)
print(yh_test.shape)
```

```
(501, 5)
(501, 1)
(334, 5)
(334, 1)
```

In [61]:
```
# MinMaxScaling

from sklearn.preprocessing import MinMaxScaler

# creating a minmax scaler
mm = MinMaxScaler()

# feeding the independent data into the scaler
xh_train = mm.fit_transform(xh_train)
xh_test = mm.fit_transform(xh_test)
```

For Target Variable SCHILLER

In [62]:
```
# splitting the dataset into  training and test set for Schiller

from sklearn.model_selection import train_test_split

xs_train, xs_test, ys_train, ys_test = train_test_split(xnew, ys, test_size = 0.4

print(xs_train.shape)
print(ys_train.shape)
print(xs_test.shape)
print(ys_test.shape)
```

```
(501, 5)
(501, 1)
(334, 5)
(334, 1)
```

In [63]:
```
# MinMaxScaling

from sklearn.preprocessing import MinMaxScaler

# creating a minmax scaler
mm = MinMaxScaler()

# feeding the independent data into the scaler
xs_train = mm.fit_transform(xs_train)
xs_test = mm.fit_transform(xs_test)
```

For Target Variable CITOLOGY

In [64]:
```
# splitting the dataset into  training and test set for Citology

from sklearn.model_selection import train_test_split

xc_train, xc_test, yc_train, yc_test = train_test_split(xnew, yc, test_size = 0.4

print(xc_train.shape)
print(yc_train.shape)
print(xc_test.shape)
print(yc_test.shape)
```

```
(501, 5)
(501, 1)
(334, 5)
(334, 1)
```

In [65]:
```
# MinMaxScaling

from sklearn.preprocessing import MinMaxScaler

# creating a minmax scaler
mm = MinMaxScaler()

# feeding the independent data into the scaler
xc_train = mm.fit_transform(xc_train)
xc_test = mm.fit_transform(xc_test)
```

For Target Variable BIOPSY

In [66]:
```
# splitting the dataset into  training and test set for Biopsy

from sklearn.model_selection import train_test_split

xb_train, xb_test, yb_train, yb_test = train_test_split(xnew, yb, test_size = 0.4

print(xb_train.shape)
print(yb_train.shape)
print(xb_test.shape)
print(yb_test.shape)
```

```
(501, 5)
(501, 1)
(334, 5)
(334, 1)
```

In [67]:
```
# MinMaxScaling

from sklearn.preprocessing import MinMaxScaler

# creating a minmax scaler
mm = MinMaxScaler()

# feeding the independent data into the scaler
xb_train = mm.fit_transform(xb_train)
xb_test = mm.fit_transform(xb_test)
```

Modelling (for Hinselmann)

Logistic Regression

In [68]:
```
from sklearn.linear_model import LogisticRegression

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
yh_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9481037924151696
Testing accuracy : 0.9730538922155688
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       325
           1       0.00      0.00      0.00         9

    accuracy                           0.97       334
   macro avg       0.49      0.50      0.49       334
weighted avg       0.95      0.97      0.96       334

[[325   0]
 [  9   0]]
```

Support Vector Machine

In [69]:
```
from sklearn.svm import SVC

# creating the model
model = SVC()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
ys_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9481037924151696
Testing accuracy : 0.9730538922155688
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       325
           1       0.00      0.00      0.00         9

    accuracy                           0.97       334
   macro avg       0.49      0.50      0.49       334
weighted avg       0.95      0.97      0.96       334

[[325   0]
 [  9   0]]
```

Decision Tree

In [70]:
```
from sklearn.tree import DecisionTreeClassifier

# creating the model
model = DecisionTreeClassifier()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
yh_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9960079840319361
Testing accuracy : 0.9101796407185628
              precision    recall  f1-score   support

           0       0.97      0.94      0.95       325
           1       0.00      0.00      0.00         9

    accuracy                           0.91       334
   macro avg       0.49      0.47      0.48       334
weighted avg       0.95      0.91      0.93       334

[[304  21]
 [  9   0]]
```

Random Forest

In [71]:
```
from sklearn.ensemble import RandomForestClassifier

# creating the model
model = RandomForestClassifier()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
yh_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9960079840319361
Testing accuracy : 0.9700598802395209
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       325
           1       0.00      0.00      0.00         9

    accuracy                           0.97       334
   macro avg       0.49      0.50      0.49       334
weighted avg       0.95      0.97      0.96       334

[[324   1]
 [  9   0]]
```

Modelling(for Schiller)

Logistic Regression

In [72]:
```
from sklearn.linear_model import LogisticRegression

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xs_train, ys_train)

# predicting the test set results
ys_pred = model.predict(xs_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xs_train, ys_train))
print("Testing accuracy :", model.score(xs_test, ys_test))

# classification report
print(classification_report(ys_test, ys_pred))

# confusion matrix
print(confusion_matrix(ys_test, ys_pred))
```

```
Training accuracy : 0.908183632734531
Testing accuracy : 0.9191616766467066
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       307
           1       0.00      0.00      0.00        27

    accuracy                           0.92       334
   macro avg       0.46      0.50      0.48       334
weighted avg       0.84      0.92      0.88       334

[[307   0]
 [ 27   0]]
```

Support Vector Machine

In [73]:
```
from sklearn.svm import SVC

# creating the model
model = SVC()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
yh_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9481037924151696
Testing accuracy : 0.9730538922155688
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       325
           1       0.00      0.00      0.00         9

    accuracy                           0.97       334
   macro avg       0.49      0.50      0.49       334
weighted avg       0.95      0.97      0.96       334

[[325   0]
 [  9   0]]
```

Decision Tree

In [74]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# creating the model
model = DecisionTreeClassifier()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
yh_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9960079840319361
Testing accuracy : 0.9191616766467066
              precision    recall  f1-score   support

           0       0.97      0.94      0.96       325
           1       0.00      0.00      0.00         9

    accuracy                           0.92       334
   macro avg       0.49      0.47      0.48       334
weighted avg       0.95      0.92      0.93       334

[[307  18]
 [  9   0]]
```

Random Forest

In [75]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xh_train, yh_train)

# predicting the test set results
yh_pred = model.predict(xh_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xh_train, yh_train))
print("Testing accuracy :", model.score(xh_test, yh_test))

# classification report
print(classification_report(yh_test, yh_pred))

# confusion matrix
print(confusion_matrix(yh_test, yh_pred))
```

```
Training accuracy : 0.9481037924151696
Testing accuracy : 0.9730538922155688
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       325
           1       0.00      0.00      0.00         9

    accuracy                           0.97       334
   macro avg       0.49      0.50      0.49       334
weighted avg       0.95      0.97      0.96       334

[[325   0]
 [  9   0]]
```

Modelling (for Citilogy)

Logistic Regression

In [76]:
```
from sklearn.linear_model import LogisticRegression

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xc_train, yc_train)

# predicting the test set results
yc_pred = model.predict(xc_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xc_train, yc_train))
print("Testing accuracy :", model.score(xc_test, yc_test))

# classification report
print(classification_report(yc_test, yc_pred))

# confusion matrix
print(confusion_matrix(yc_test, yc_pred))
```

```
Training accuracy : 0.9461077844311377
Testing accuracy : 0.9520958083832335
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       318
           1       0.00      0.00      0.00        16

    accuracy                           0.95       334
   macro avg       0.48      0.50      0.49       334
weighted avg       0.91      0.95      0.93       334

[[318   0]
 [ 16   0]]
```

Support Vector Machine

In [77]:
```
from sklearn.svm import SVC

# creating the model
model = SVC()

# feeding the training data into the model
model.fit(xc_train, yc_train)

# predicting the test set results
yc_pred = model.predict(xc_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xc_train, yc_train))
print("Testing accuracy :", model.score(xc_test, yc_test))

# classification report
print(classification_report(yc_test, yc_pred))

# confusion matrix
print(confusion_matrix(yc_test, yc_pred))
```

```
Training accuracy : 0.9481037924151696
Testing accuracy : 0.9520958083832335
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       318
           1       0.00      0.00      0.00        16

    accuracy                           0.95       334
   macro avg       0.48      0.50      0.49       334
weighted avg       0.91      0.95      0.93       334

[[318   0]
 [ 16   0]]
```

Decision Tree

In [78]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# creating the model
model = DecisionTreeClassifier()

# feeding the training data into the model
model.fit(xc_train, yc_train)

# predicting the test set results
yc_pred = model.predict(xc_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xc_train, yc_train))
print("Testing accuracy :", model.score(xc_test, yc_test))

# classification report
print(classification_report(yc_test, yc_pred))

# confusion matrix
print(confusion_matrix(yc_test, yc_pred))
```

```
Training accuracy : 0.998003992015968
Testing accuracy : 0.8023952095808383
              precision    recall  f1-score   support

           0       0.95      0.83      0.89       318
           1       0.05      0.19      0.08        16

    accuracy                           0.80       334
   macro avg       0.50      0.51      0.49       334
weighted avg       0.91      0.80      0.85       334

[[265  53]
 [ 13   3]]
```

Random Forest

In [79]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xc_train, yc_train)

# predicting the test set results
yc_pred = model.predict(xc_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xc_train, yc_train))
print("Testing accuracy :", model.score(xc_test, yc_test))

# classification report
print(classification_report(yc_test, yc_pred))

# confusion matrix
print(confusion_matrix(yc_test, yc_pred))
```

```
Training accuracy : 0.9461077844311377
Testing accuracy : 0.9520958083832335
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       318
           1       0.00      0.00      0.00        16

    accuracy                           0.95       334
   macro avg       0.48      0.50      0.49       334
weighted avg       0.91      0.95      0.93       334

[[318   0]
 [ 16   0]]
```

Modelling (for Biopsy)

Logistic Regression

In [80]:
```
from sklearn.linear_model import LogisticRegression

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xb_train, yb_train)

# predicting the test set results
yb_pred = model.predict(xb_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xb_train, yb_train))
print("Testing accuracy :", model.score(xb_test, yb_test))

# classification report
print(classification_report(yb_test, yb_pred))

# confusion matrix
print(confusion_matrix(yb_test, yb_pred))
```

```
Training accuracy : 0.9341317365269461
Testing accuracy : 0.937125748502994
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       313
           1       0.00      0.00      0.00        21

    accuracy                           0.94       334
   macro avg       0.47      0.50      0.48       334
weighted avg       0.88      0.94      0.91       334

[[313   0]
 [ 21   0]]
```

Support Vector Machine

In [81]:
```
from sklearn.svm import SVC

# creating the model
model = SVC()

# feeding the training data into the model
model.fit(xb_train, yb_train)

# predicting the test set results
yb_pred = model.predict(xb_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xb_train, yb_train))
print("Testing accuracy :", model.score(xb_test, yb_test))

# classification report
print(classification_report(yb_test, yb_pred))

# confusion matrix
print(confusion_matrix(yb_test, yb_pred))
```

```
Training accuracy : 0.9341317365269461
Testing accuracy : 0.937125748502994
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       313
           1       0.00      0.00      0.00        21

    accuracy                           0.94       334
   macro avg       0.47      0.50      0.48       334
weighted avg       0.88      0.94      0.91       334

[[313   0]
 [ 21   0]]
```

Decision Tree

In [82]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# creating the model
model = DecisionTreeClassifier()

# feeding the training data into the model
model.fit(xb_train, yb_train)

# predicting the test set results
yb_pred = model.predict(xb_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xb_train, yb_train))
print("Testing accuracy :", model.score(xb_test, yb_test))

# classification report
print(classification_report(yb_test, yb_pred))

# confusion matrix
print(confusion_matrix(yb_test, yb_pred))
```

```
Training accuracy : 0.998003992015968
Testing accuracy : 0.8053892215568862
              precision    recall  f1-score   support

           0       0.95      0.84      0.89       313
           1       0.11      0.29      0.16        21

    accuracy                           0.81       334
   macro avg       0.53      0.56      0.52       334
weighted avg       0.89      0.81      0.84       334

[[263  50]
 [ 15   6]]
```

Random Forest

In [83]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# creating the model
model = LogisticRegression()

# feeding the training data into the model
model.fit(xb_train, yb_train)

# predicting the test set results
yb_pred = model.predict(xb_test)

# Calculating the accuracies
print("Training accuracy :", model.score(xb_train, yb_train))
print("Testing accuracy :", model.score(xb_test, yb_test))

# classification report
print(classification_report(yb_test, yb_pred))

# confusion matrix
print(confusion_matrix(yb_test, yb_pred))
```

```
Training accuracy : 0.9341317365269461
Testing accuracy : 0.937125748502994
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       313
           1       0.00      0.00      0.00        21

    accuracy                           0.94       334
   macro avg       0.47      0.50      0.48       334
weighted avg       0.88      0.94      0.91       334

[[313   0]
 [ 21   0]]
```