main

December 4, 2022

1 Title

1.1 Data Exploration

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('./heart_2020_cleaned.csv')
```

[ ]:
```
df.head()
```

[ ]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth
0	No	16.60	Yes	No	No	3.0
1	No	20.34	No	No	Yes	0.0
2	No	26.58	Yes	No	No	20.0
3	No	24.21	No	No	No	0.0
4	No	23.71	No	No	No	28.0

	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic
0	30.0	No	Female	55-59	White	Yes
1	0.0	No	Female	80 or older	White	No
2	30.0	No	Male	65-69	White	Yes
3	0.0	No	Female	75-79	White	No
4	0.0	Yes	Female	40-44	White	No

	PhysicalActivity	GenHealth	SleepTime	Asthma	KidneyDisease	SkinCancer
0	Yes	Very good	5.0	Yes	No	Yes
1	Yes	Very good	7.0	No	No	No
2	Yes	Fair	8.0	Yes	No	No
3	No	Good	6.0	No	No	Yes
4	Yes	Very good	8.0	No	No	No

Our dataset has 319,795 observations and 17 features and the target column `HeartDisease`.

[ ]:
```
df.shape
```

[ ]:
```
(319795, 18)
```

[ ]:
```
df.isna().sum()
```

[ ]:
```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

We can see there is no missing data or null values to fill in, so we will now encode our data.

The following code prints out the unique values for the categorical variables, to help us understand our data and better encode the data.

[ ]:
```
categoricals = df.select_dtypes(include=['object']).columns.tolist()
for cat in categoricals:
    print(f'{cat}: {sorted(df[cat].unique())}')
```

```
HeartDisease: ['No', 'Yes']
Smoking: ['No', 'Yes']
AlcoholDrinking: ['No', 'Yes']
Stroke: ['No', 'Yes']
DiffWalking: ['No', 'Yes']
Sex: ['Female', 'Male']
AgeCategory: ['18-24', '25-29', '30-34', '35-39', '40-44', '45-49', '50-54',
'55-59', '60-64', '65-69', '70-74', '75-79', '80 or older']
Race: ['American Indian/Alaskan Native', 'Asian', 'Black', 'Hispanic', 'Other',
'White']
Diabetic: ['No', 'No, borderline diabetes', 'Yes', 'Yes (during pregnancy)']
PhysicalActivity: ['No', 'Yes']
GenHealth: ['Excellent', 'Fair', 'Good', 'Poor', 'Very good']
Asthma: ['No', 'Yes']
KidneyDisease: ['No', 'Yes']
SkinCancer: ['No', 'Yes']
```

TODO: Describe this ^^^^^^ ## Categorical Variables #### HeartDisease * Whether survey

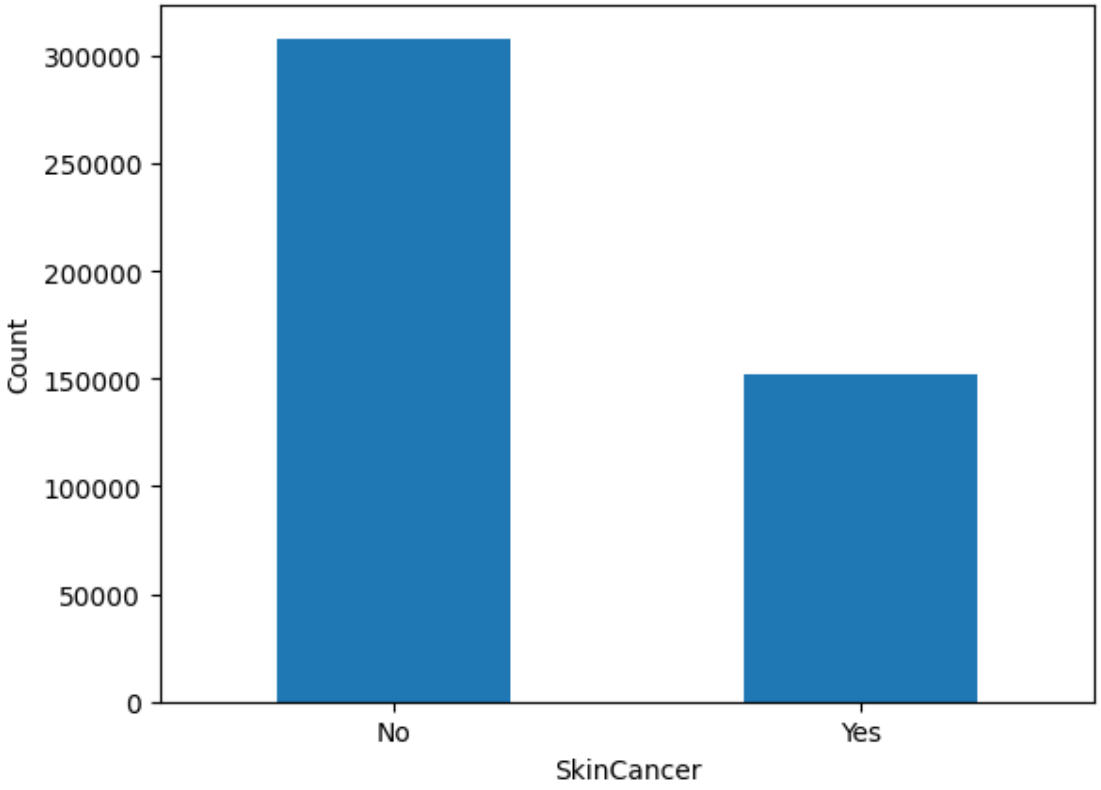
respondents reported having heart disease (either coronary heart disease or myocardial infarction) * Possible values: Yes, No #### Smoking * Whether survey participants have smoked greater than or equal to 100 cigarettes (5 packs) in their lifetime * Possible values: Yes, No #### Alcohol Drinking * Whether survey respondents are heavy drinkers (adult men who have over 14 drinks per week or adult women who have over 7 drink per week) * Possible values: Yes, No #### Stroke * Whether the respondents have ever had a stroke * Possible values: Yes, No #### DiffWalking * Whether respondents have serious difficulty climbing stairs or walking in general * Possible values: Yes, No #### Sex * Biological sex of respondent * Possible values: Male, Female #### AgeCategory * Which of 14 levels of age categories respondents fall into * Possible Values: 18-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59, 60-64, 65-69, 70-74, 75-79, 80 or older #### Race * Race category of respondent * Possible values: American Indian/Alaskan Native, Asian, Black, Hispanic, Other, White #### Diabetic * Whether the respondents have ever been told they have diabetes * Possible values: No; No, borderline diabetes; Yes; Yes (during pregnancy) #### Physical Activities * Whether respondents reported physical activity or exercise within the last 30 days outside their regular jobs * Possible values: Yes, No #### GenHealth * How respondents would describe their general health * Possible values: Excellent, Fair, Good, Poor, Very good #### Asthma * Whether respondents have ever been told they have asthma * Possible values: Yes, No #### KidneyDisease * Whether respondents have ever been told they have kidney disease (other than kidney stones, bladder infection, incontinence) * Possible values: Yes, No #### Skin Cancer * Whether respondents have ever been told they have skin cancer * Possible values: Yes, No

TODO: PLOT HOW MUCH OF EACH UNIQUE CATEGORY IN EACH CATEGORICAL VARIABLE GENERATE HISTOGRAM FOR CATEGORICAL DATA

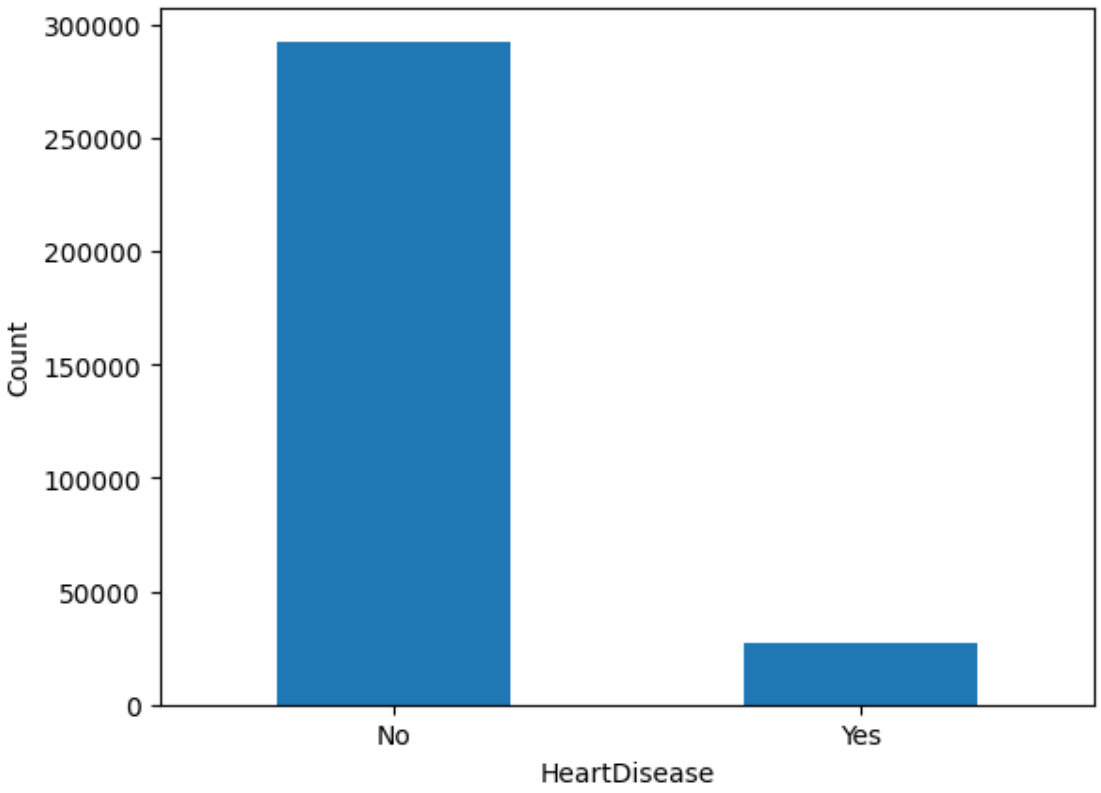
[ ]:
```
for c in categoricals:
    display(df[c].value_counts().plot(kind='bar', xlabel=c, ylabel='Count',␣
 ↪rot=0))
```

```
<AxesSubplot: xlabel='HeartDisease', ylabel='Count'>
<AxesSubplot: xlabel='Smoking', ylabel='Count'>
<AxesSubplot: xlabel='AlcoholDrinking', ylabel='Count'>
<AxesSubplot: xlabel='Stroke', ylabel='Count'>
<AxesSubplot: xlabel='DiffWalking', ylabel='Count'>
<AxesSubplot: xlabel='Sex', ylabel='Count'>
<AxesSubplot: xlabel='AgeCategory', ylabel='Count'>
<AxesSubplot: xlabel='Race', ylabel='Count'>
<AxesSubplot: xlabel='Diabetic', ylabel='Count'>
<AxesSubplot: xlabel='PhysicalActivity', ylabel='Count'>
<AxesSubplot: xlabel='GenHealth', ylabel='Count'>
<AxesSubplot: xlabel='Asthma', ylabel='Count'>
<AxesSubplot: xlabel='KidneyDisease', ylabel='Count'>
```

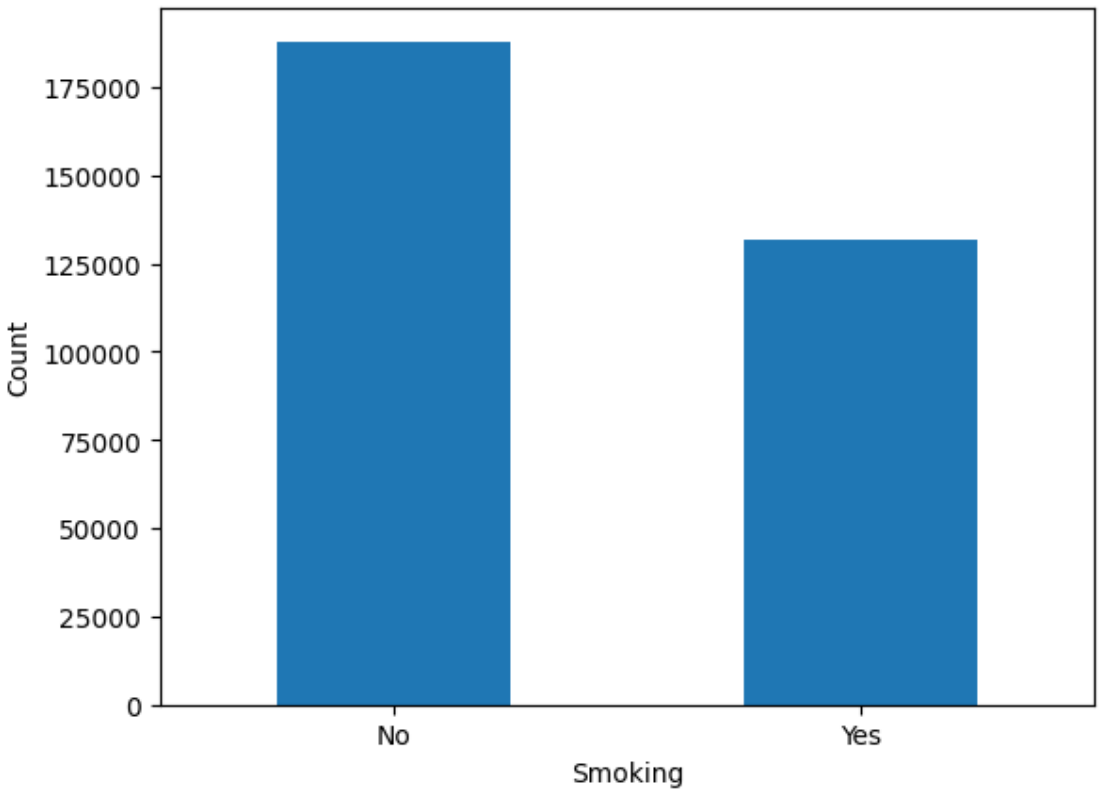
```
<AxesSubplot: xlabel='SkinCancer', ylabel='Count'>
```



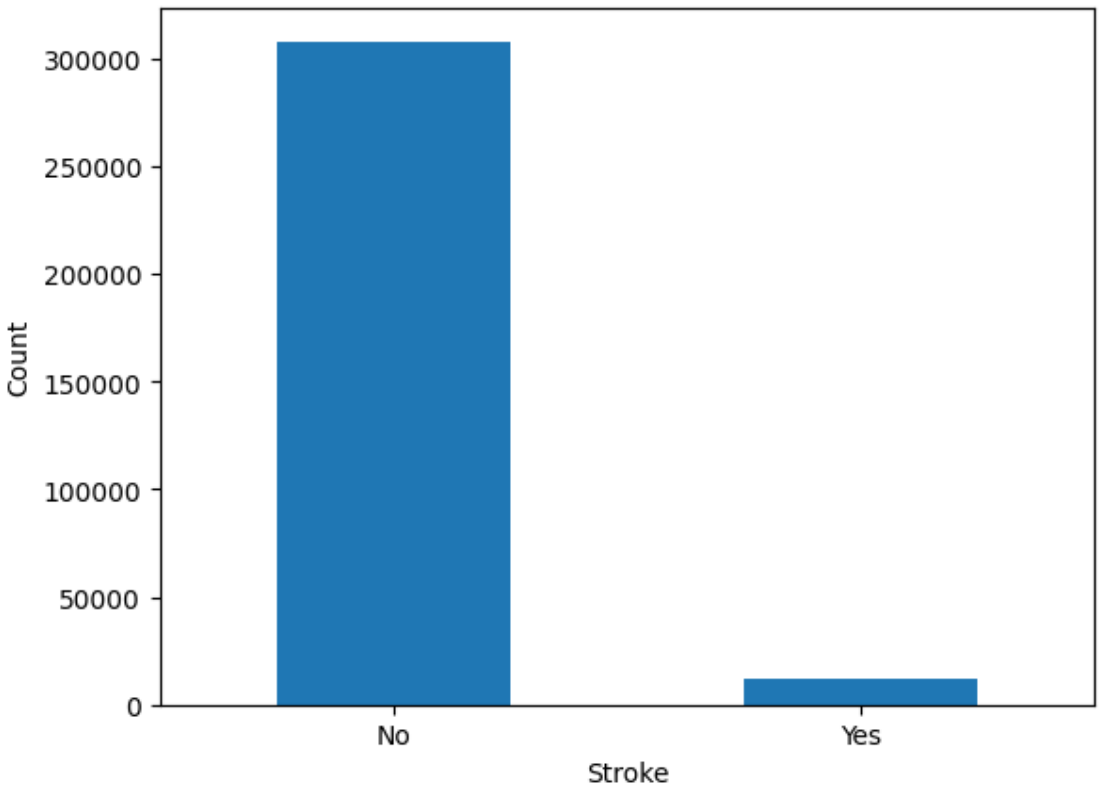
[ ]:
```
df['HeartDisease'].value_counts().plot(kind='bar', xlabel='HeartDisease',␣
↪ylabel='Count', rot=0);
```



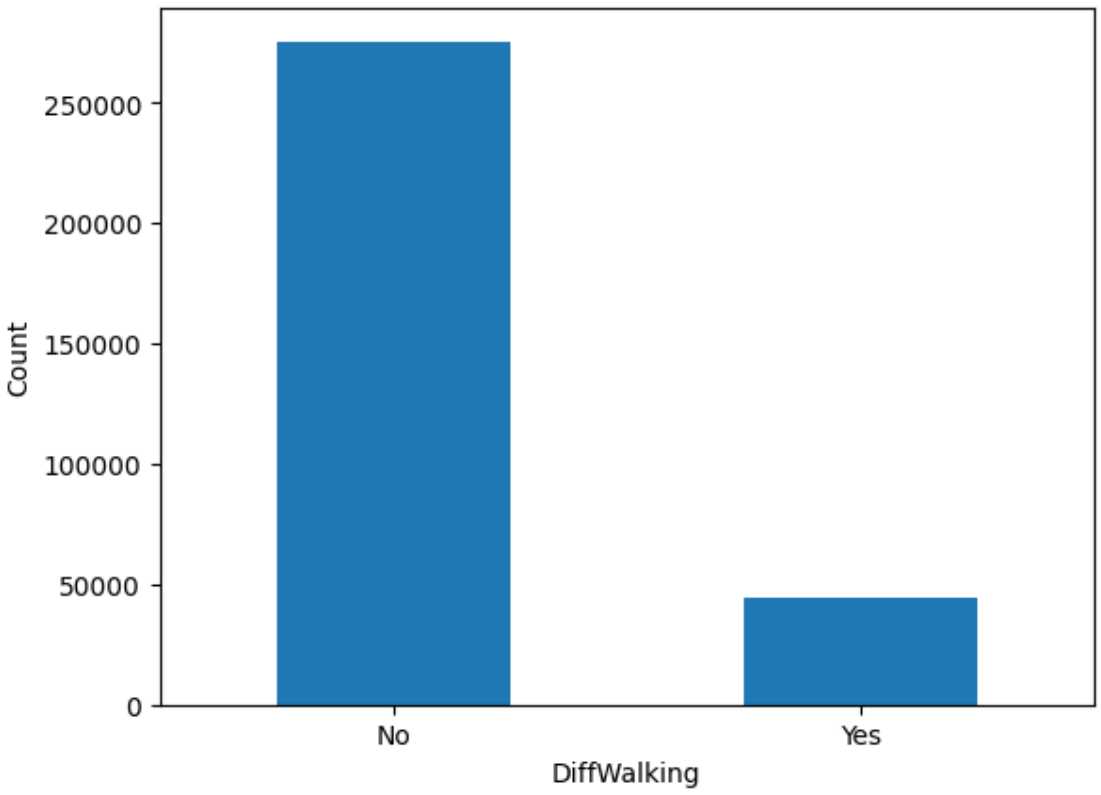
[ ]:
```
df['Smoking'].value_counts().plot(kind='bar', xlabel='Smoking', ylabel='Count',␣
↪rot=0);
```



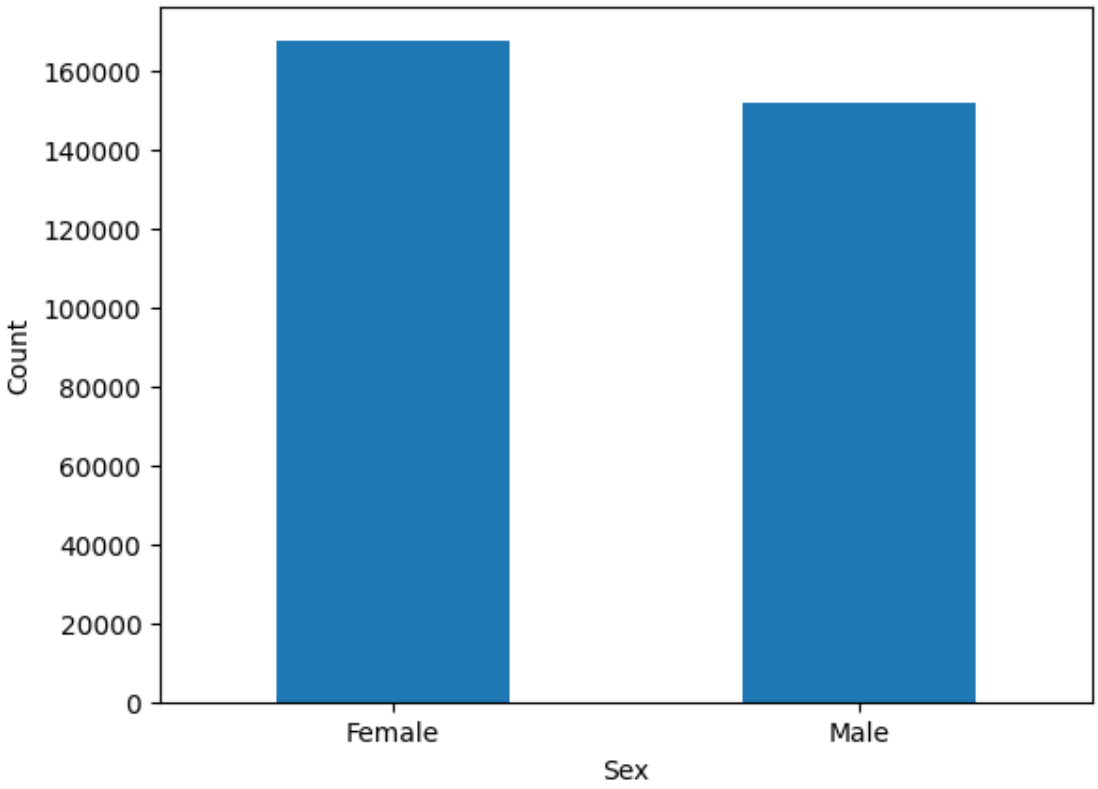
[ ]:
```
df['Stroke'].value_counts().plot(kind='bar', xlabel='Stroke', ylabel='Count',␣
↪rot=0);
```



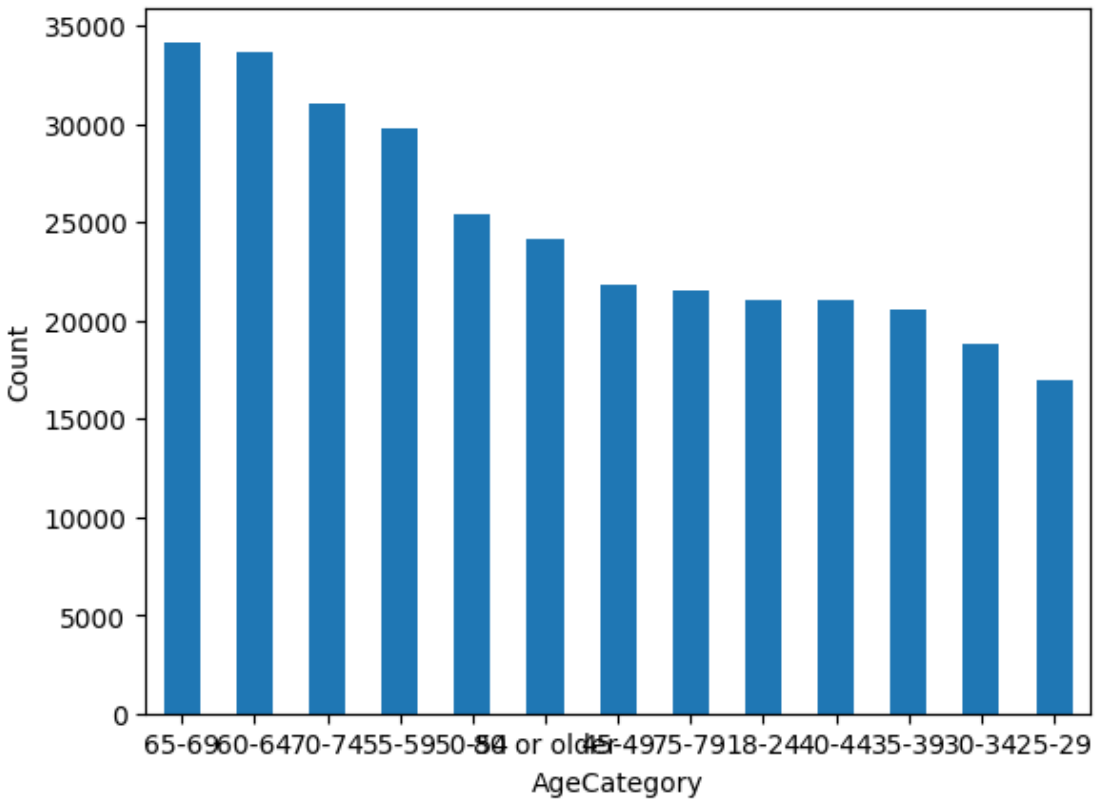
[ ]:
```
df['DiffWalking'].value_counts().plot(kind='bar', xlabel='DiffWalking',␣
↪ylabel='Count', rot=0);
```



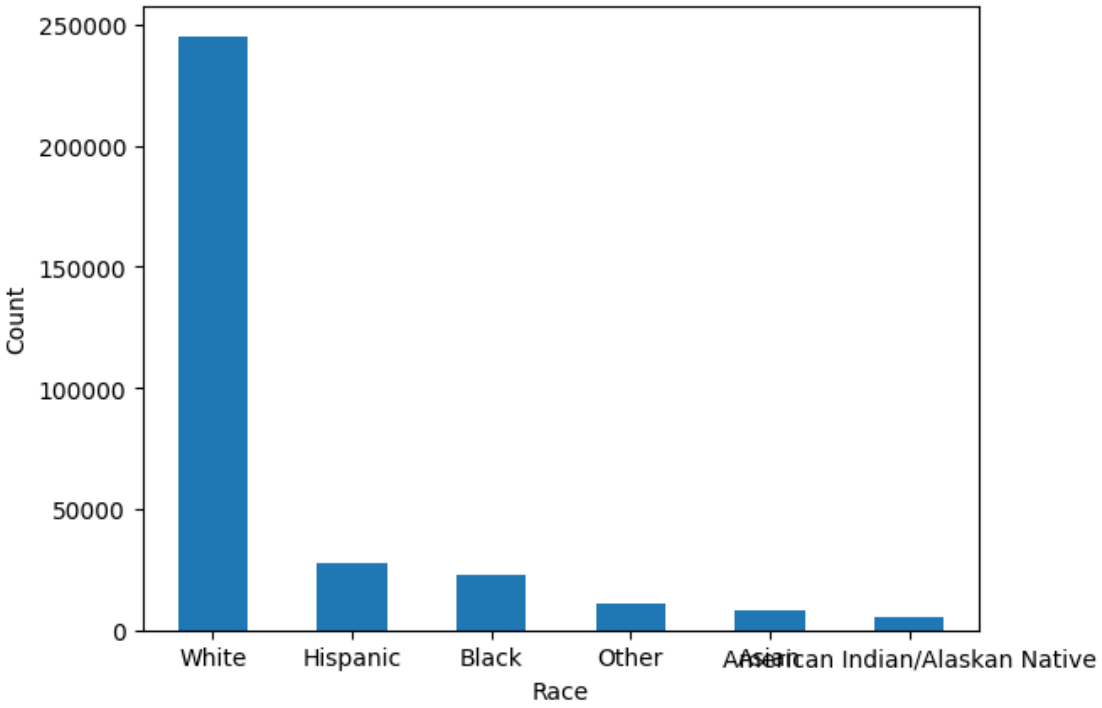
[ ]:
```
df['Sex'].value_counts().plot(kind='bar', xlabel='Sex', ylabel='Count', rot=0);
```



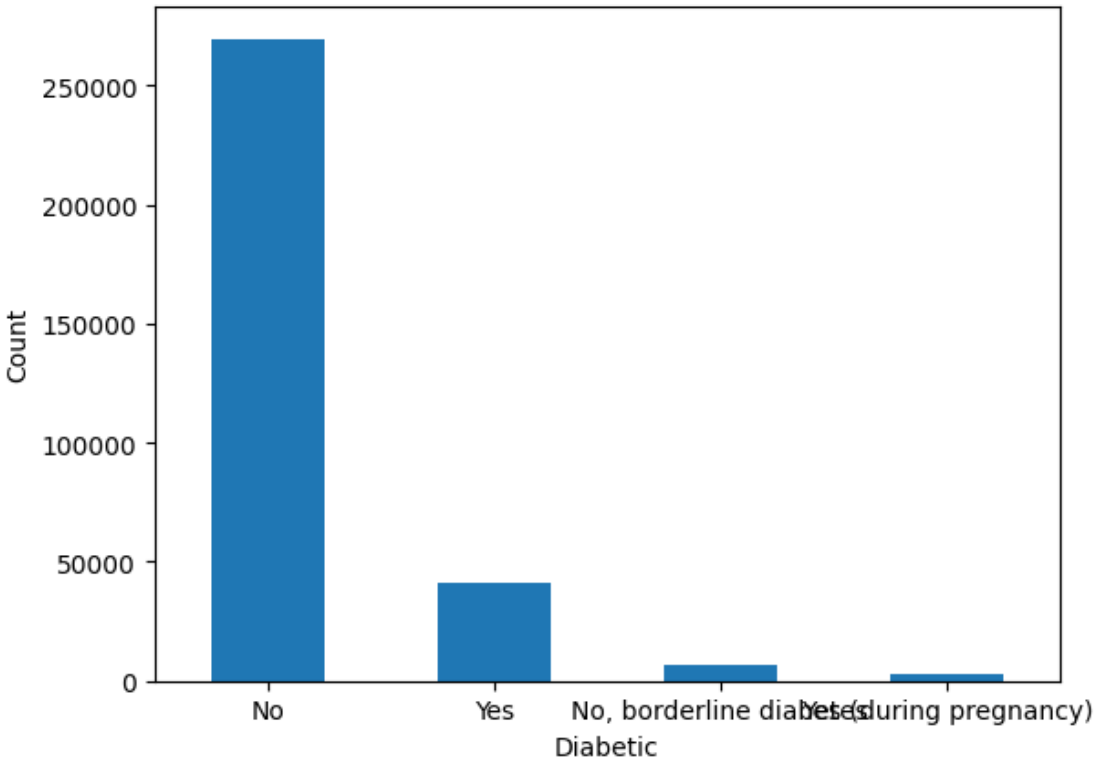
[ ]:
```
df['AgeCategory'].value_counts().plot(kind='bar', xlabel='AgeCategory',␣
↪ylabel='Count', rot=0);
```



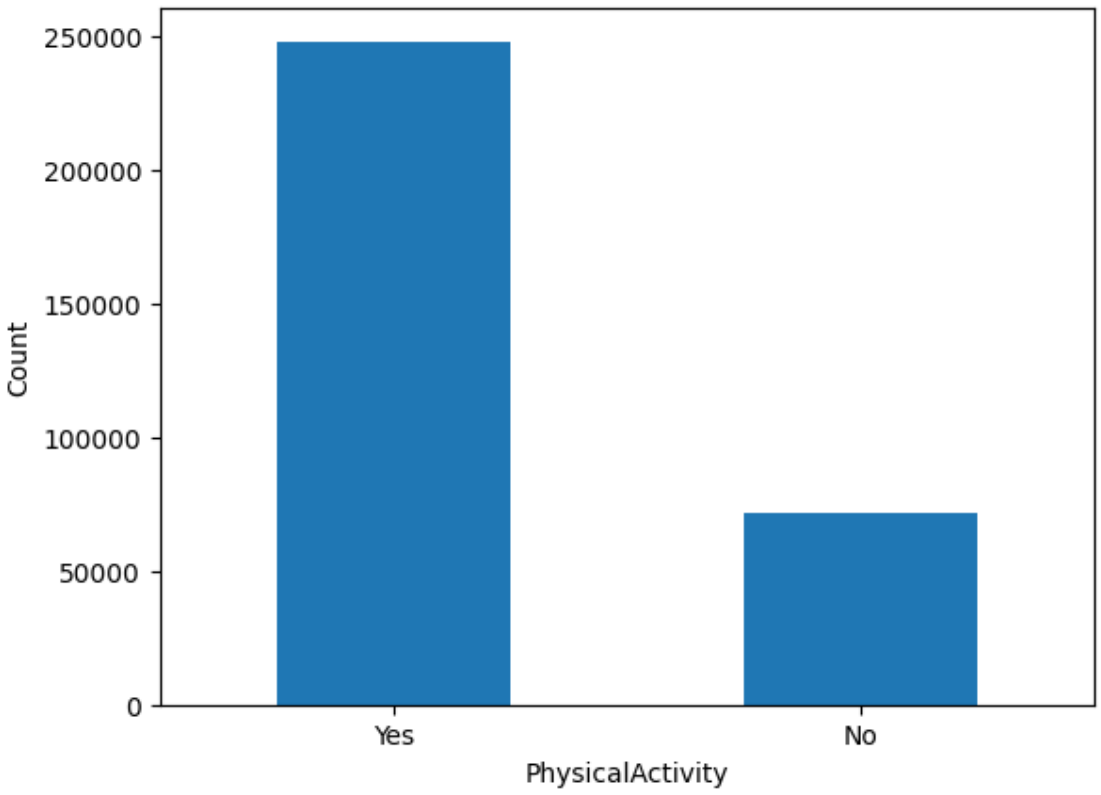
[ ]:
```
df['Race'].value_counts().plot(kind='bar', xlabel='Race', ylabel='Count',␣
↪rot=0);
```



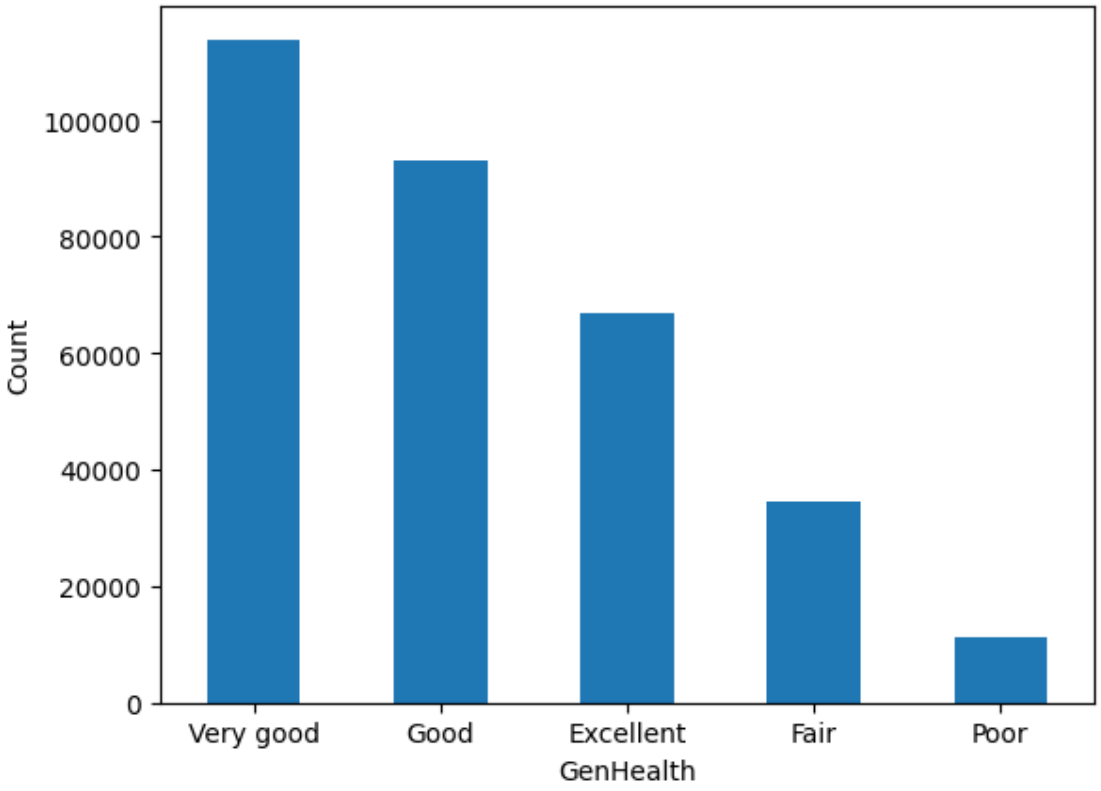
[ ]:
```
df['Diabetic'].value_counts().plot(kind='bar', xlabel='Diabetic',␣
↪ylabel='Count', rot=0);
```



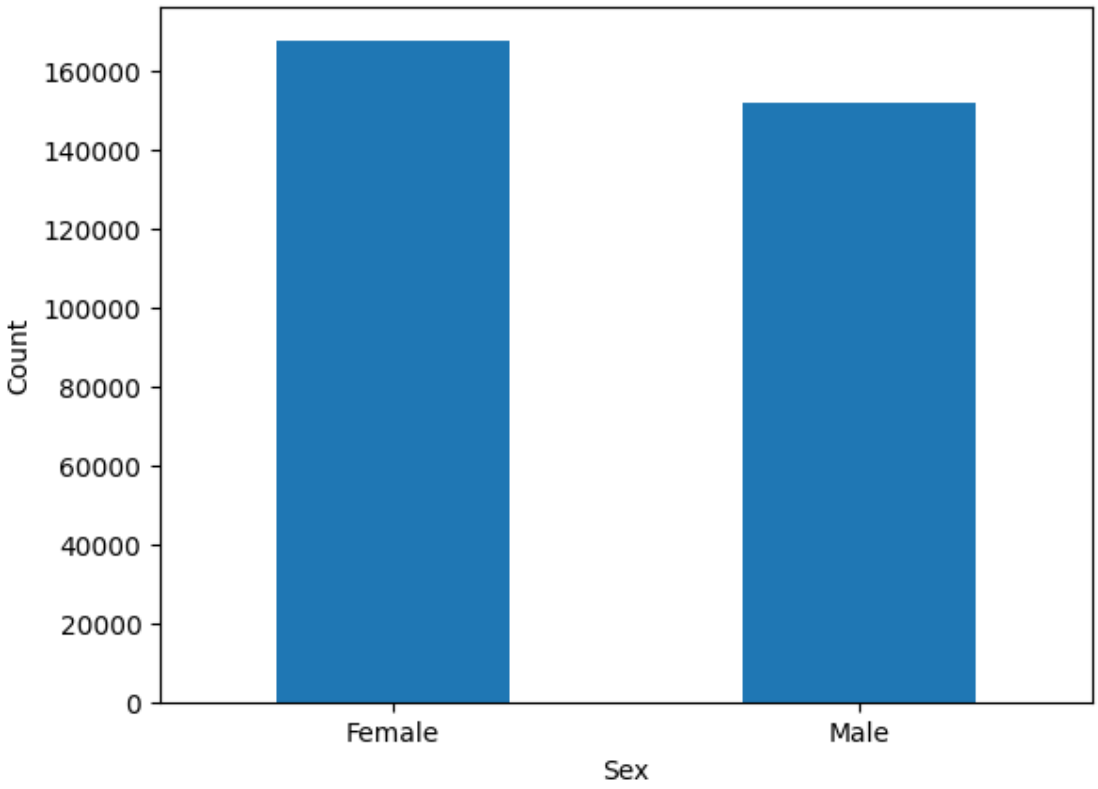
[ ]:
```
df['PhysicalActivity'].value_counts().plot(kind='bar',␣
↪xlabel='PhysicalActivity', ylabel='Count', rot=0);
```



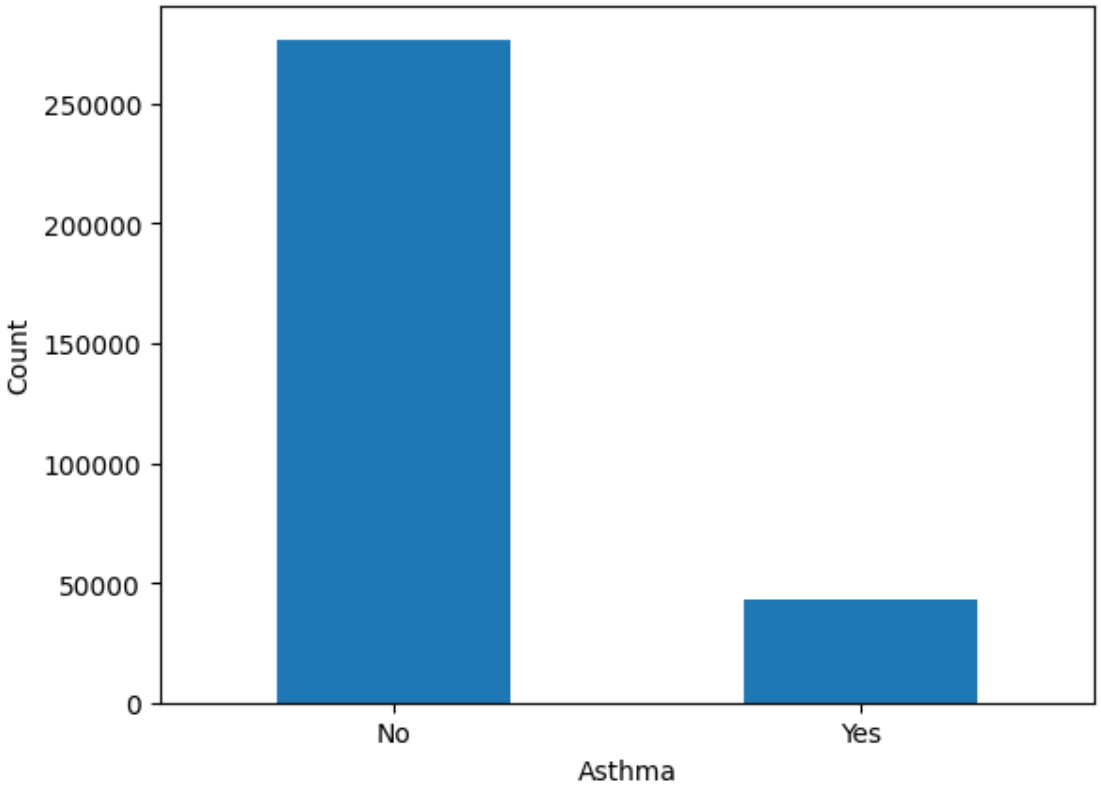
[ ]:
```
df['GenHealth'].value_counts().plot(kind='bar', xlabel='GenHealth',␣
↪ylabel='Count', rot=0);
```



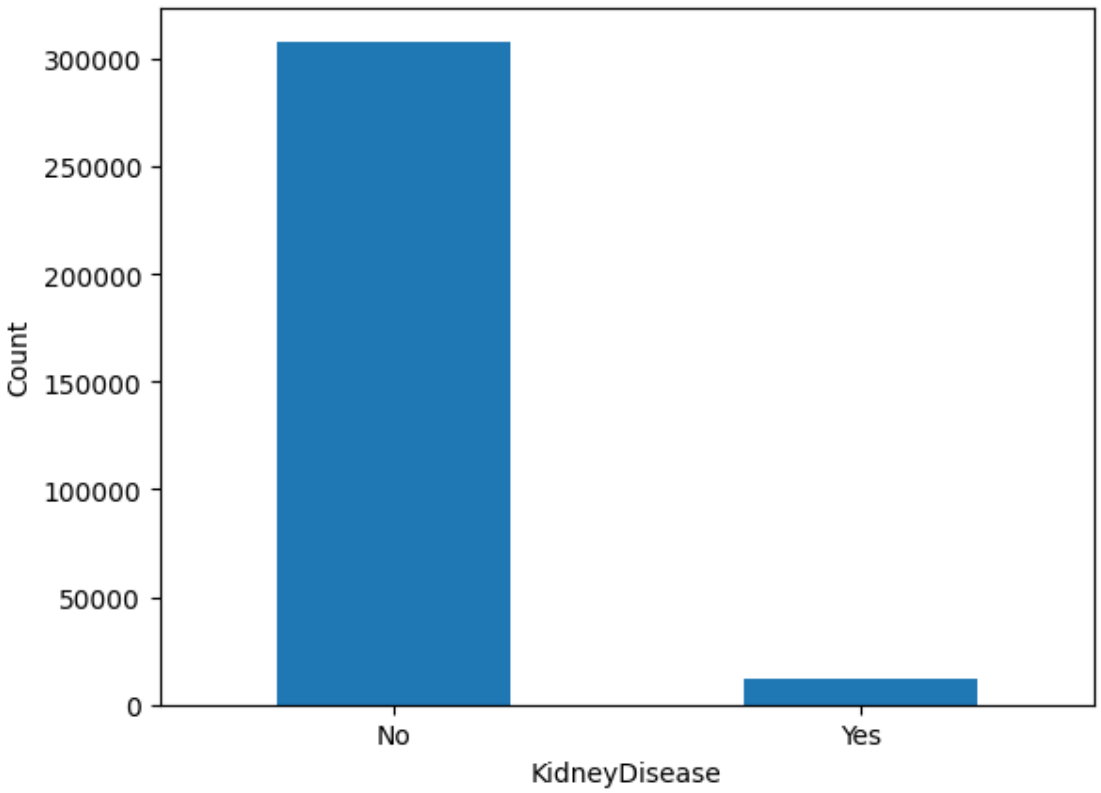
[ ]:
```
df['Sex'].value_counts().plot(kind='bar', xlabel='Sex', ylabel='Count', rot=0);
```



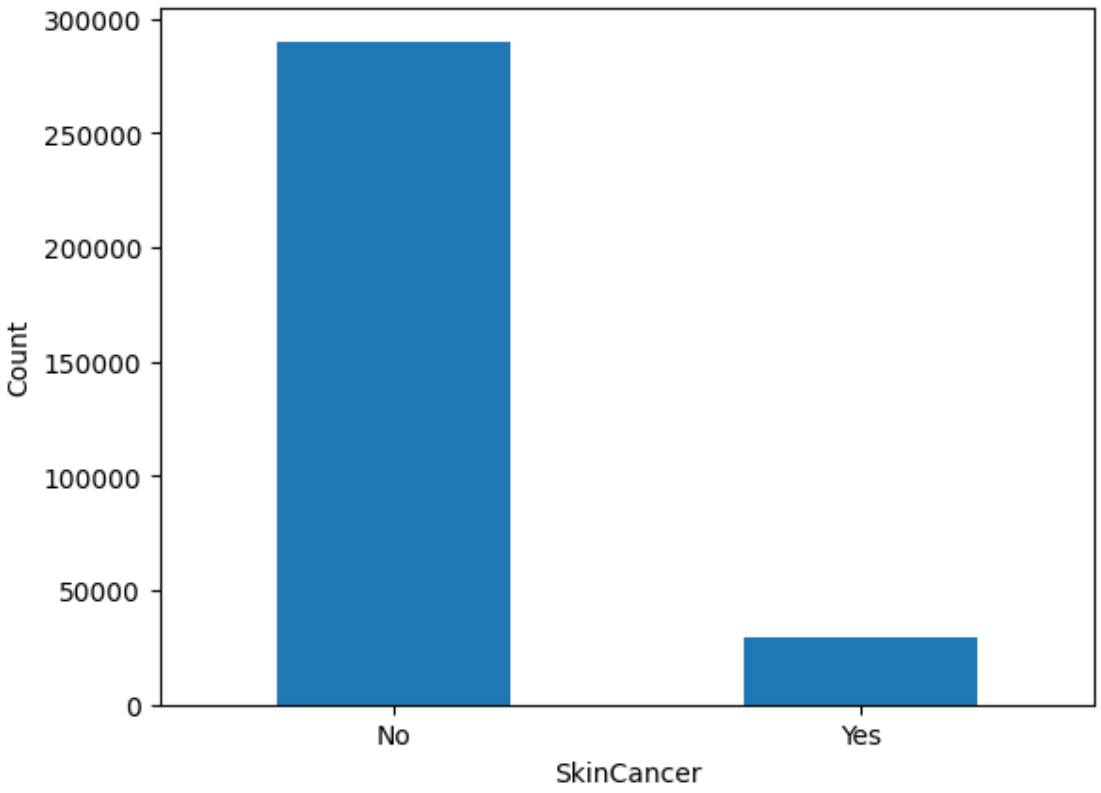
[ ]:
```
df['Asthma'].value_counts().plot(kind='bar', xlabel='Asthma', ylabel='Count',␣
↪rot=0);
```



[ ]:
```
df['KidneyDisease'].value_counts().plot(kind='bar', xlabel='KidneyDisease',␣
↪ylabel='Count', rot=0);
```

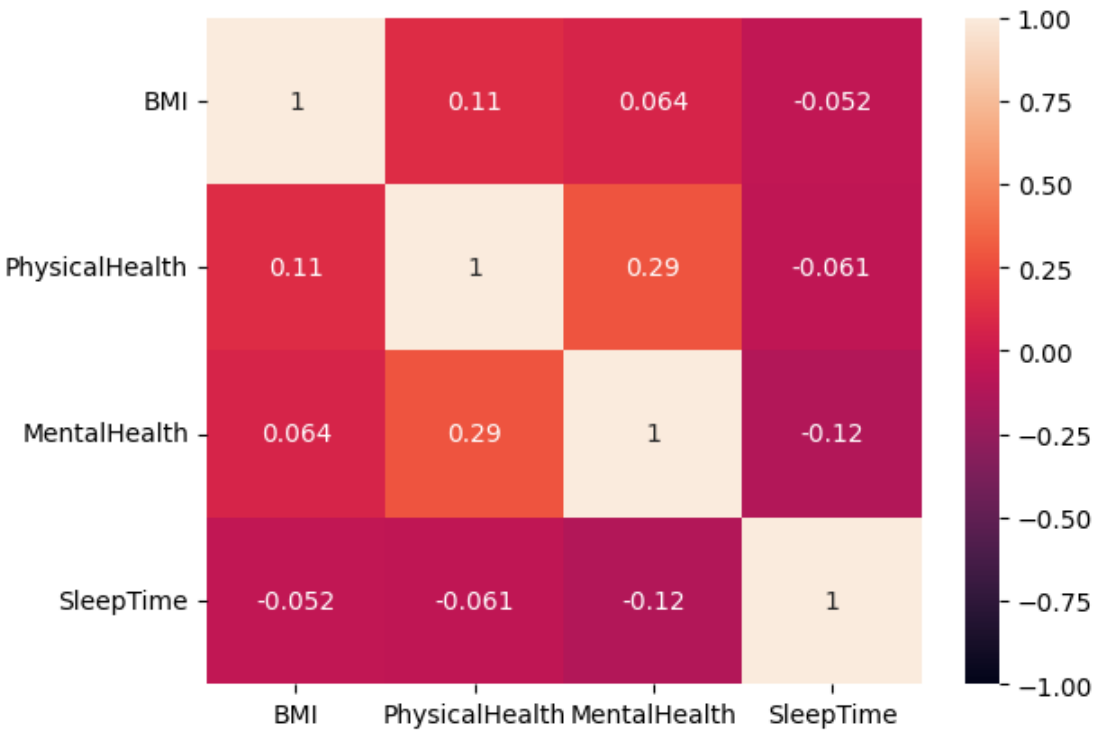


[ ]:
```
df['SkinCancer'].value_counts().plot(kind='bar', xlabel='SkinCancer',␣
↪ylabel='Count', rot=0);
```



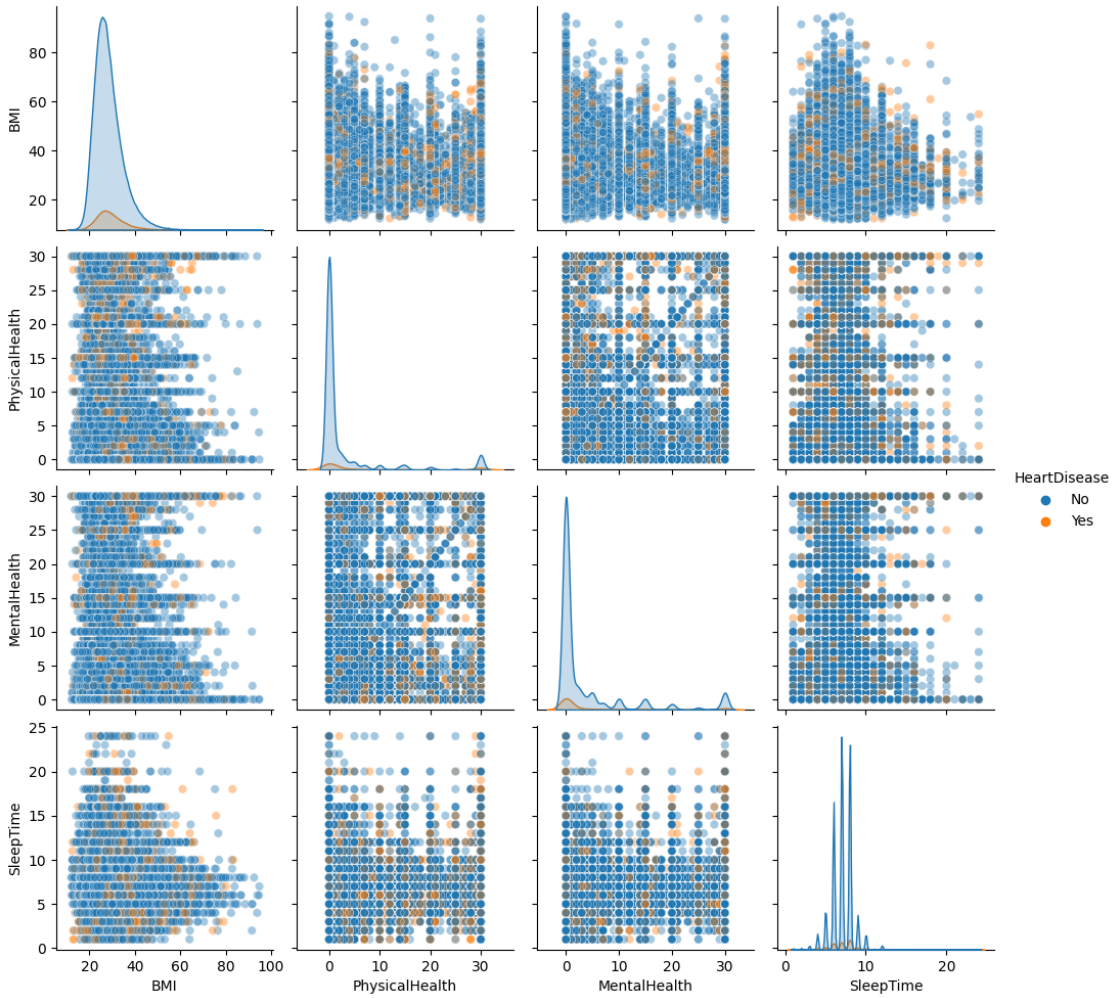
It doesn't make much sense to include our categorical variables in the correlation matrix below (even after encoding) as they mostly aren't consisting of ordered scales, instead they are almost all binary valued.

[ ]:
```
sns.heatmap(df.corr(numeric_only=True), vmin=-1, vmax=1, center=0, annot=True,␣
↪cmap='rocket');
```



Below we can see relations between numerical data and the distributions each type of data. For BMI we see a bit of a skewed normal distribution, but for the rest there appears to be some discrete aspect of separation to the data values, especially with sleep time. We also can't really see much of a clear linear relationship in the scatter plots which matches with the low correlation that we see in the correlation matrix.

[ ]:
```
sns.pairplot(df, hue='HeartDisease', plot_kws=dict(alpha=0.4));
```



See below for the discrete integer nature of the specified variables for confirmation of what we noticed above with the histograms

[ ]:
```
df['SleepTime'].unique() # see the discrete integer values
```

[ ]:
```
array([ 5.,  7.,  8.,  6., 12.,  4.,  9., 10., 15.,  3.,  2.,  1., 16.,
       18., 14., 20., 11., 13., 17., 24., 19., 21., 22., 23.])
```

[ ]:
```
df['MentalHealth'].unique()
```

[ ]:
```
array([30.,  0.,  2.,  5., 15.,  8.,  4.,  3., 10., 14., 20.,  1.,  7.,
       24.,  9., 28., 16., 12.,  6., 25., 17., 18., 21., 29., 22., 13.,
       23., 27., 26., 11., 19.])
```

[ ]:
```
df['PhysicalHealth'].unique()
```

[ ]:
```
array([ 3.,  0., 20., 28.,  6., 15.,  5., 30.,  7.,  1.,  2., 21.,  4.,
       10., 14., 18.,  8., 25., 16., 29., 27., 17., 24., 12., 23., 26.,
       22., 19.,  9., 13., 11.])
```

With a wide variety of features such as pre-existing health conditions, and general lifestyle information such as level of physical activity, or drinking alcohol, this data set provides us with a large amount of information to perform a binary classification on the presence of heart disease in an individual.

Some potential biases arise however when considering the source of the data, as this is data collected from a voluntary response phone poll, which means that there could be a significant difference between people who answer phone polls and people who don't. Additionally, the data collected from the poll could also have inaccuracies, as it is possible people lie or exaggerate about their lifestyle habits such as exercise or alcohol consumption.

Aside from any potential biases from the collection of the data, this data set provides a great opportunity to try and predict the likelihood of heart disease in an individual based on their pre-existing health conditions and their broad lifestyle habits.

1.2 Data Preprocessing

[ ]:
```
encode = lambda unique: dict( zip( sorted(unique), range(len(unique)) ) )

# encode( df[categoricals[0]].unique() )
encoding = {'GenHealth': {'Poor': 0, 'Fair': 1, 'Good': 2, 'Very good': 3,␣
 ↪'Excellent': 4}}

# perform genhealth encoding manually since no sorting function
df_encode = df.copy()
for c in categoricals:
  if c != 'GenHealth': encoding[c] = encode(df[c].unique()) # save to encoding
  df_encode[c] = df.apply( lambda row: encoding[c][row[c]], axis=1) # perform␣
 ↪encoding

encoding
```

[ ]:
```
{'GenHealth': {'Poor': 0,
  'Fair': 1,
  'Good': 2,
  'Very good': 3,
  'Excellent': 4},
 'HeartDisease': {'No': 0, 'Yes': 1},
 'Smoking': {'No': 0, 'Yes': 1},
 'AlcoholDrinking': {'No': 0, 'Yes': 1},
 'Stroke': {'No': 0, 'Yes': 1},
 'DiffWalking': {'No': 0, 'Yes': 1},
 'Sex': {'Female': 0, 'Male': 1},
 'AgeCategory': {'18-24': 0,
```

```
 '25-29': 1,
 '30-34': 2,
 '35-39': 3,
 '40-44': 4,
 '45-49': 5,
 '50-54': 6,
 '55-59': 7,
 '60-64': 8,
 '65-69': 9,
 '70-74': 10,
 '75-79': 11,
 '80 or older': 12},
'Race': {'American Indian/Alaskan Native': 0,
 'Asian': 1,
 'Black': 2,
 'Hispanic': 3,
 'Other': 4,
 'White': 5},
'Diabetic': {'No': 0,
 'No, borderline diabetes': 1,
 'Yes': 2,
 'Yes (during pregnancy)': 3},
'PhysicalActivity': {'No': 0, 'Yes': 1},
'Asthma': {'No': 0, 'Yes': 1},
'KidneyDisease': {'No': 0, 'Yes': 1},
'SkinCancer': {'No': 0, 'Yes': 1}}
```

We will be using one-hot encoding for race, as race is not a scale nor is it binary.

[ ]:
```
df_encode['American Indian/Alaskan Native'] = df_encode.apply( lambda x: int(x.
 ↪Race == 0), axis=1)
df_encode['Asian'] = df_encode.apply( lambda x: int(x.Race == 1), axis=1)
df_encode['Black'] = df_encode.apply( lambda x: int(x.Race == 2), axis=1)
df_encode['Hispanic'] = df_encode.apply( lambda x: int(x.Race == 3), axis=1)
df_encode['White'] = df_encode.apply( lambda x: int(x.Race == 5), axis=1)
df_encode['Other Race'] = df_encode.apply( lambda x: int(x.Race == 4), axis=1)
```

[ ]:
```
# uncomment to sanity check results
# df_encode_races = df_encode[['Race', 'Asian', 'Black', 'Hispanic', 'American␣
 ↪Indian/Alaskan Native', 'White', 'Other Race']]
df_encode = df_encode.drop(columns=['Race'])
```

Now that our races are encoded, we will check the distribution of races as percentages.

[ ]:
```
races = df_encode[['Asian', 'Black', 'Hispanic', 'American Indian/Alaskan␣
 ↪Native', 'White', 'Other Race']].sum()
print(races / races.sum() * 100)
```

```
Asian                              2.522866
Black                              7.173033
Hispanic                           8.582373
American Indian/Alaskan Native     1.626667
White                             76.677872
Other Race                         3.417189
dtype: float64
```

We can see that the most common race in our dataset is White at 76%, followed by Hispanic at 8.5%, and Black at 7.1%. This presents a possible source of bias in our model, as our model will likely be more effective at predicting heart disease in white people than other racial groups due to the sample size if race ends up being an important factor.

We encoded all categorical variables into numerical values. For race we did one hot encoding as race is not a scale nor is it binary, as otherwise a trained model could view White with a value of 5 as the polar opposite of American Indian/Alaskan Native with a value of 0. We did not adjust the scale of any of the features, as most features are binary encoded, with Age having the absolute largest value with a max value of 12. As such, huge values will likely not be too much of an issue with slowing down model training with larger values in back propogation, so as such, data will likely not need to be transformed, scaled, or normalized for a fast and effective model training.

[ ]:
```
df_num = df.select_dtypes(include=['float64'])

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_num_norm = scaler.fit(df_num).transform(df_num)
df_num_norm = pd.DataFrame(df_num_norm, columns=df_num.columns, index=df_num.
  ↪index)
df_num_norm.head()
```

[ ]:

	BMI	PhysicalHealth	MentalHealth	SleepTime
0	0.055294	0.100000	1.0	0.173913
1	0.100447	0.000000	0.0	0.260870
2	0.175782	0.666667	1.0	0.304348
3	0.147169	0.000000	0.0	0.217391
4	0.141132	0.933333	0.0	0.304348

Now we need to add this data back into our full data set.

[ ]:
```
numericals = df.select_dtypes(include=['float64']).columns
dft = df_encode.copy()
for colname in numericals:
    dft[colname] = df_num_norm[colname]
```

Now, `dft` contains our data with `Race` one hot encoded, other categorical variables label encoded, and the numerical features min max normalized. We can now begin building our model.

1.3 Model Building

Before we train a model, we first need to split our data into train and test data sets.

[ ]:
```
from sklearn.model_selection import train_test_split

X = dft.drop(columns=['HeartDisease'])
y = dft['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
dims = X_train.shape[1]
```

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense
```

[ ]:
```
model = Sequential()
model.add(Dense(units=4, activation='tanh', input_dim=dims,  ))
model.add(Dense(units=4, activation='relu', input_dim=dims,))
model.add(Dense(units=1, activation='sigmoid', input_dim=dims,  ))
```

[ ]:
```
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy')
history = model.fit(X_train.astype('float'), y_train, batch_size = 1, epochs =␣
↪10)
```

```
Epoch 1/10
255836/255836 [==============================] - 370s 1ms/step - loss: 0.3024
Epoch 2/10
255836/255836 [==============================] - 358s 1ms/step - loss: 0.3018
Epoch 3/10
255836/255836 [==============================] - 388s 2ms/step - loss: 0.3015
Epoch 4/10
255836/255836 [==============================] - 472s 2ms/step - loss: 0.3020
Epoch 5/10
255836/255836 [==============================] - 454s 2ms/step - loss: 0.3009
Epoch 6/10
255836/255836 [==============================] - 456s 2ms/step - loss: 0.3009
Epoch 7/10
255836/255836 [==============================] - 488s 2ms/step - loss: 0.3015
Epoch 8/10
255836/255836 [==============================] - 500s 2ms/step - loss: 0.3018
Epoch 9/10
255836/255836 [==============================] - 494s 2ms/step - loss: 0.3004
Epoch 10/10
255836/255836 [==============================] - 499s 2ms/step - loss: 0.3012
```

1.4 Model Evaluation

[ ]:
```
yhat_train = model.predict(X_train)
yhat_train = [ 1 if y>=0.5 else 0 for y in yhat_train ]
```

```
7995/7995 [==============================] - 10s 1ms/step
```

Here we threshold the predictions from the sigmoid percent likelihood of heart disease to a binary decision on presence of heart disease with a threshold of 0.5

[ ]:
```
yhat_test = model.predict(X_test)
yhat_test = [ 1 if y>=0.5 else 0 for y in yhat_test ]
```

```
1999/1999 [==============================] - 3s 1ms/step
```

[ ]:
```
from sklearn.metrics import classification_report
```

[ ]:
```
print(classification_report(y_train.values.tolist(), yhat_train))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.96    234055
           1       0.51      0.09      0.16     21781

    accuracy                           0.92    255836
   macro avg       0.71      0.54      0.56    255836
weighted avg       0.89      0.92      0.89    255836
```

The training data predictions has a ~92% accuracy

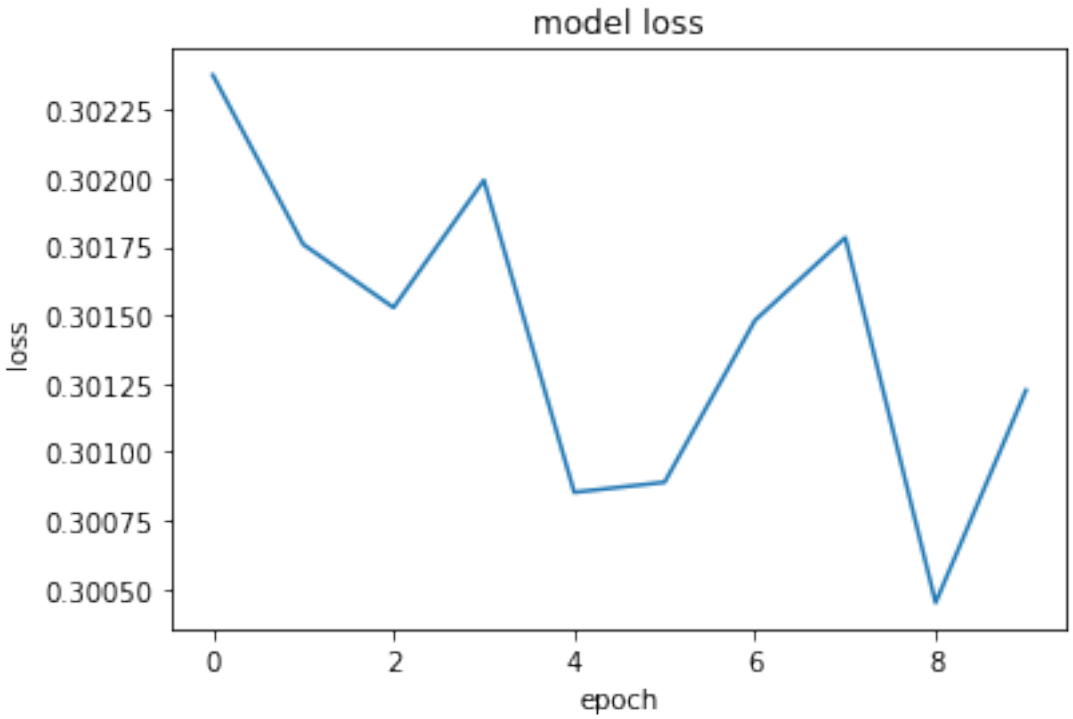
[ ]:
```
print(classification_report(y_test.values.tolist(), yhat_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     58367
           1       0.49      0.09      0.15      5592

    accuracy                           0.91     63959
   macro avg       0.71      0.54      0.55     63959
weighted avg       0.88      0.91      0.88     63959
```

The testing data predictions has a 91% accuracy

[ ]:
```
from matplotlib import pyplot as plt
plt.plot(history.history['loss'])
plt.title('model loss')
plt.xlabel('epoch')
plt.ylabel('loss')
```

```
plt.show()
```



The final iteration of the model is not the absolute minima in the fitting graph as seen here. It reached an absolute minima in epoch 8, and rose to a loss value ~0.30125 in epoch 10. This could be a sign of the model epochs correcting for overfitting, resulting in a less accurate model for training data but without overfitting. Or this rise in loss could be explained by overtraining the model.