Respiratory

January 25, 2024

1 Machine Learning Supervised - Classification

1.1 Load dataset

[3]:
```
# import dataset from pydataset
from pydataset import data
```

[4]:
```
# Use the respiratory dataset
# Respiratory Illness Data
df = data('respiratory')
df
```

[4]:
```
     center  id treat sex  age  baseline  visit  outcome
1         1   1     P   M   46         0      1        0
2         1   1     P   M   46         0      2        0
3         1   1     P   M   46         0      3        0
4         1   1     P   M   46         0      4        0
5         1   2     P   M   28         0      1        0
..      …  ..   …  ..  …       …    …      …
440       2  54     A   F   63         1      4        1
441       2  55     A   M   31         1      1        1
442       2  55     A   M   31         1      2        1
443       2  55     A   M   31         1      3        1
444       2  55     A   M   31         1      4        1

[444 rows x 8 columns]
```

[5]:
```
df.shape
```

[5]: (444, 8)

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 444 entries, 1 to 444
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
```

```
 0   center    444 non-null    int64
 1   id        444 non-null    int64
 2   treat     444 non-null    object
 3   sex       444 non-null    object
 4   age       444 non-null    int64
 5   baseline  444 non-null    int64
 6   visit     444 non-null    int64
 7   outcome   444 non-null    int64
dtypes: int64(6), object(2)
memory usage: 31.2+ KB
```

[7]:
```
df.describe().transpose()
```

[7]:

	count	mean	std	min	25%	50%	75%	max
center	444.0	1.495495	0.500544	1.0	1.00	1.0	2.00	2.0
id	444.0	28.252252	16.040844	1.0	14.00	28.0	42.00	56.0
age	444.0	33.279279	13.607309	11.0	23.00	31.0	43.00	68.0
baseline	444.0	0.450450	0.498100	0.0	0.00	0.0	1.00	1.0
visit	444.0	2.500000	1.119295	1.0	1.75	2.5	3.25	4.0
outcome	444.0	0.558559	0.497119	0.0	0.00	1.0	1.00	1.0

1.2 Import libraries

[9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from mlxtend.plotting import plot_decision_regions
import missingno as msno
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import VarianceThreshold

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix
from sklearn import metrics
#from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

1.3 Exploratory Data Analysis

[11]:
```
# Null values
df.isnull().sum()
```

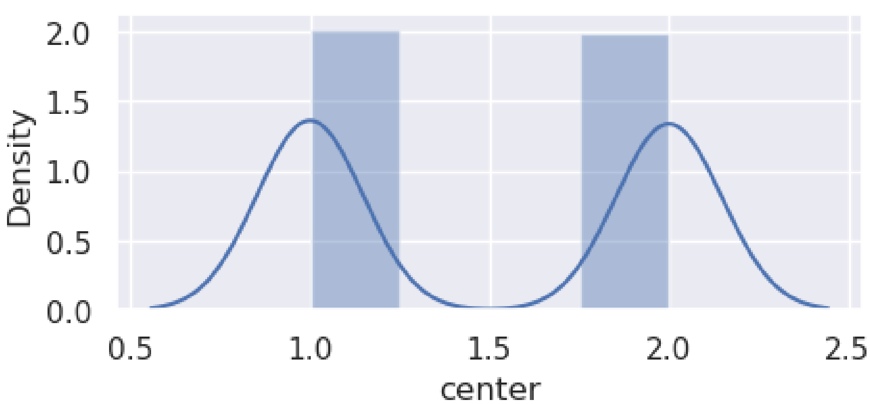
[11]:
```
center      0
id          0
treat       0
sex         0
age         0
baseline    0
visit       0
outcome     0
dtype: int64
```

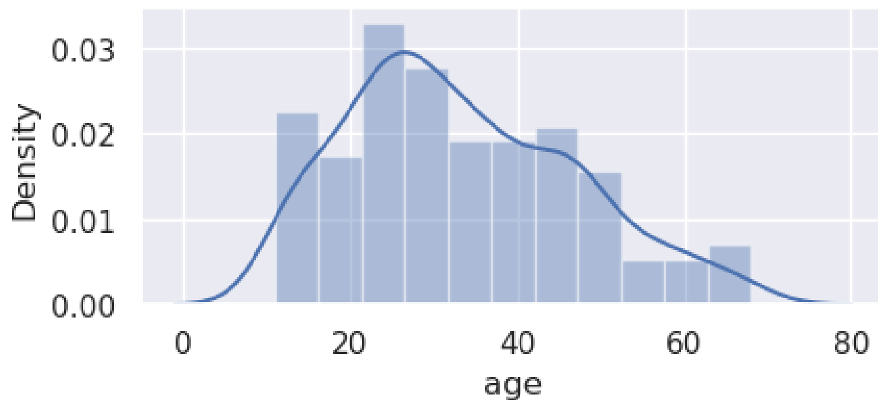
[12]:
```
# Drop id
df=df.drop(['id'], axis=1)
df.head()
```

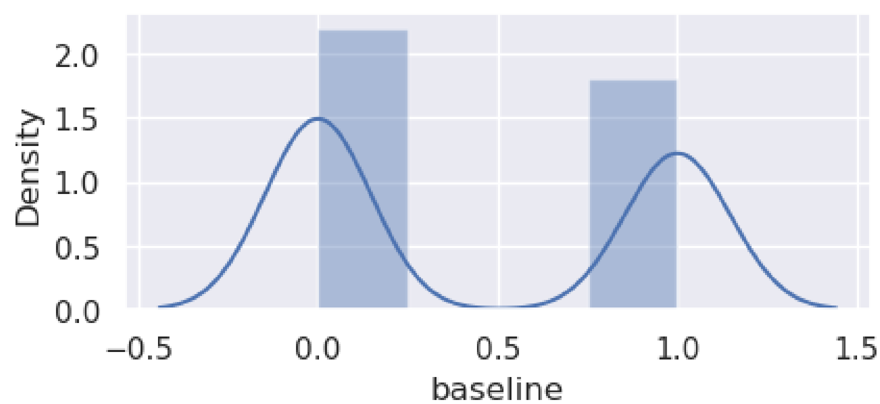
[12]:

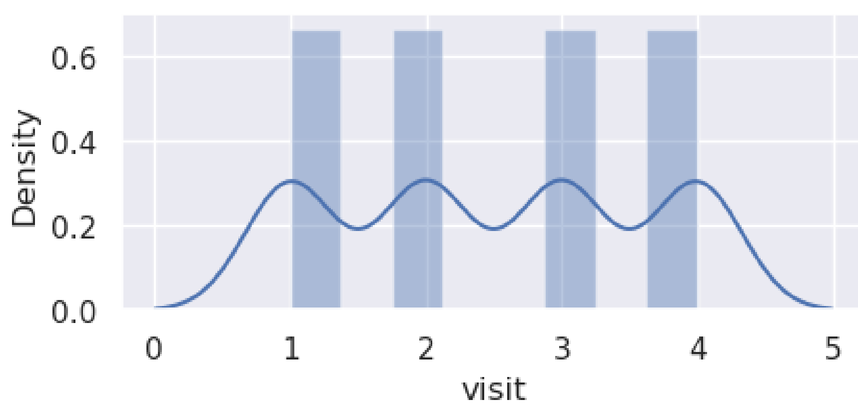
	center	treat	sex	age	baseline	visit	outcome
1	1	P	M	46	0	1	0
2	1	P	M	46	0	2	0
3	1	P	M	46	0	3	0
4	1	P	M	46	0	4	0
5	1	P	M	28	0	1	0

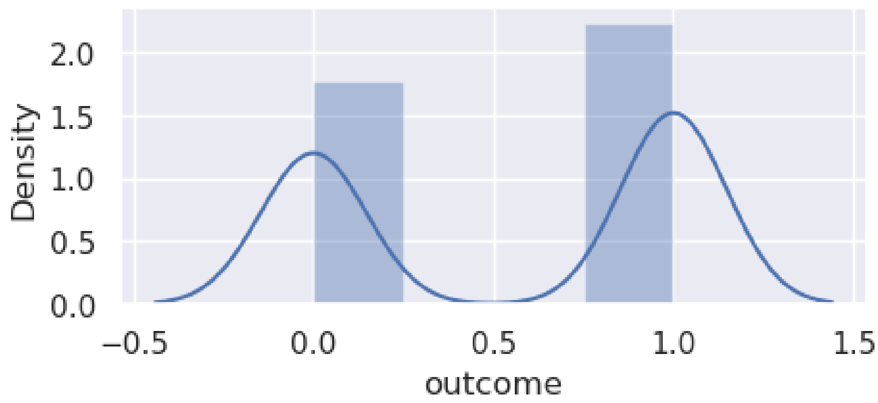
[13]:
```
#Histogram
columns = ['center', 'age', 'baseline', 'visit', 'outcome']
for i in columns:
    plt.figure(figsize=(5,2))
    sns.distplot(df[i])
```




Density
0.00
0.01
0.02
0.03
0
20
40
60
80
age


Density
0.0
0.5
1.0
1.5
2.0
−0.5
0.0
0.5
1.0
1.5
baseline

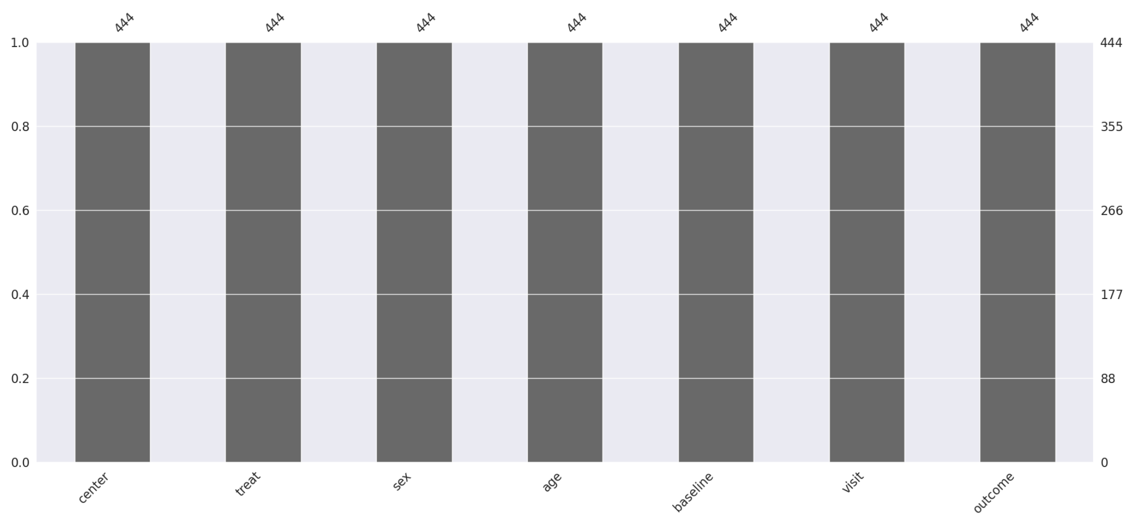

Density
0.0
0.2
0.4
0.6
0
1
2
3
4
5
visit



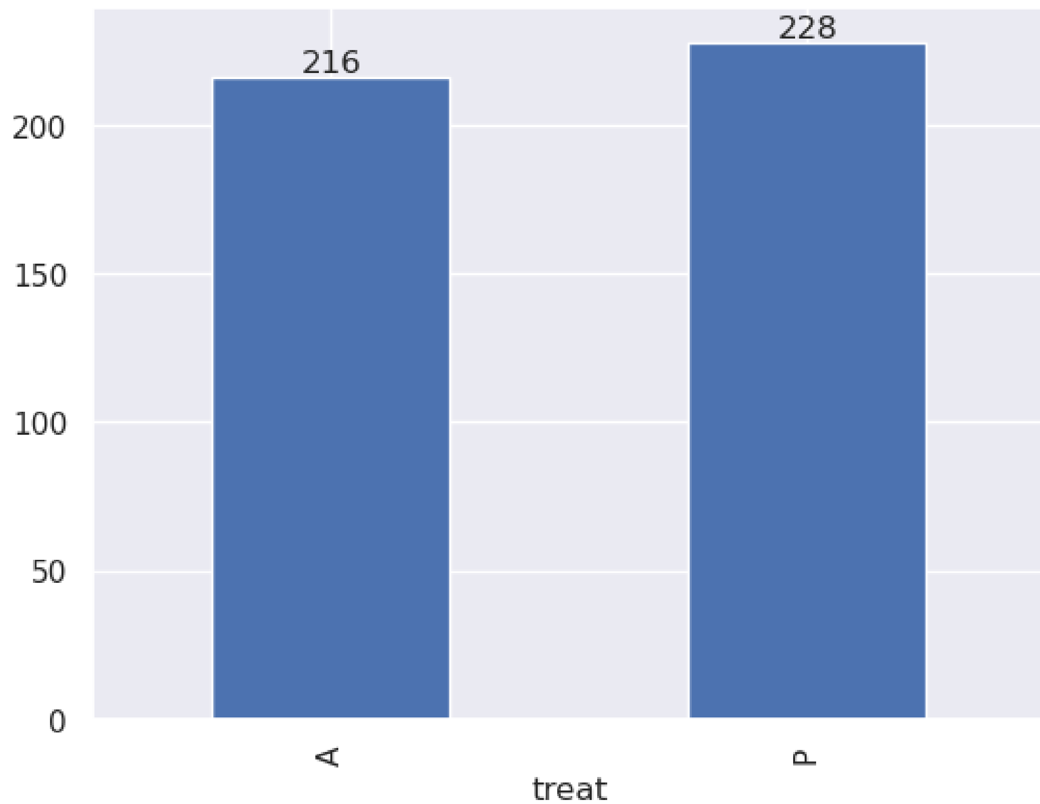
[14]:
```
#pip install mlxtend
```

[15]:
```
#pip install missingno
```

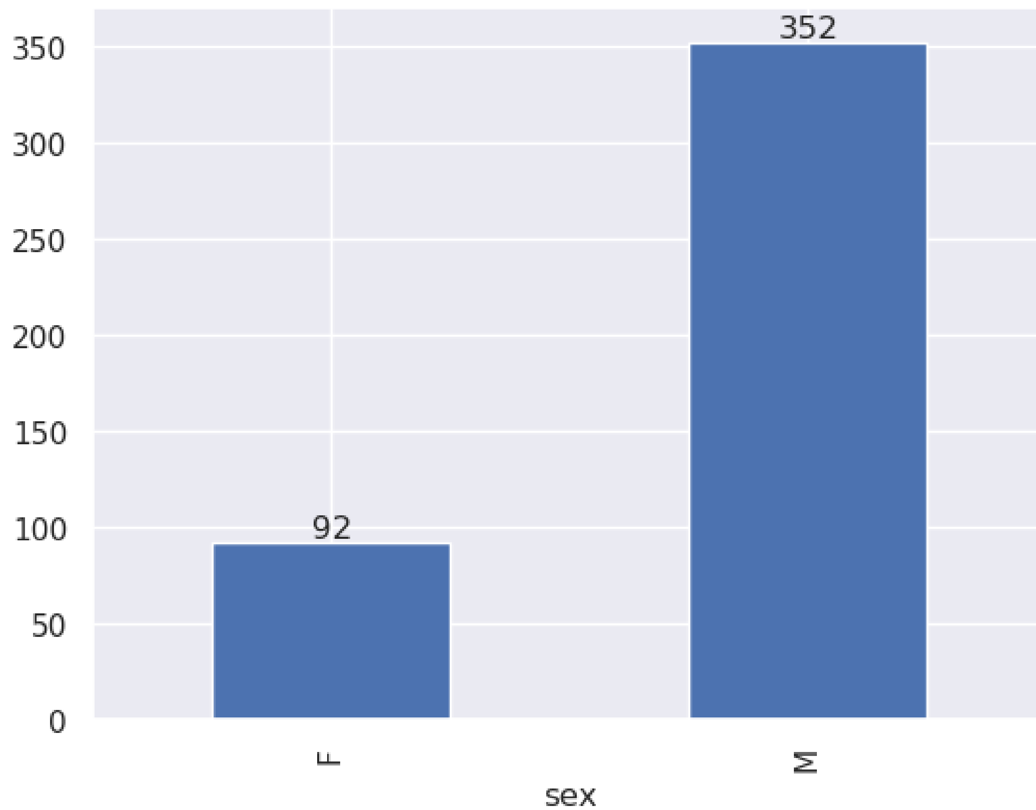
[16]:
```
#plotting null count analysis plot
msno.bar(df)
plt.show()
```



[17]:
```
# Treat
ax=df.treat.value_counts().sort_values(ascending=True).plot(kind="bar")
ax.bar_label(ax.containers[0])
plt.show()
```

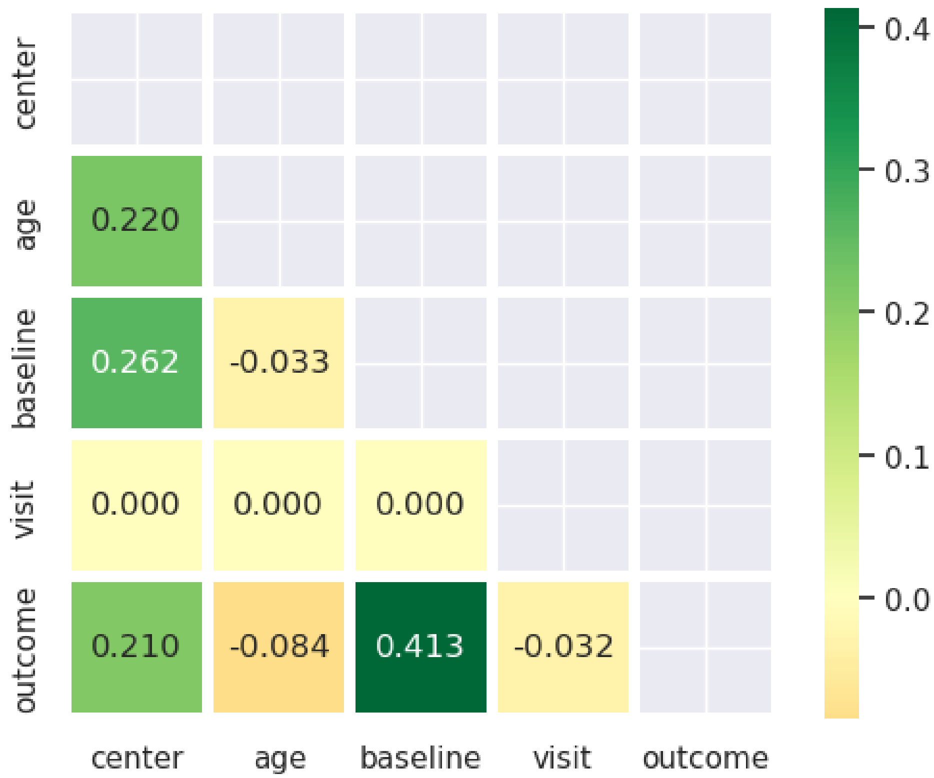


[18]:
```
# Sex
ax=df.sex.value_counts().sort_values(ascending=True).plot(kind="bar")
ax.bar_label(ax.containers[0])
plt.show()
```



[19]:
```
#correlation between numeric features

mask = np.zeros_like(df.corr(numeric_only=True))
mask[np.triu_indices_from(mask)]=True
sns.heatmap(df.corr(numeric_only=True), annot=True,center=0,fmt='.3f',␣
↪square=True, linewidth=3, mask=mask, cmap='RdYlGn')
```

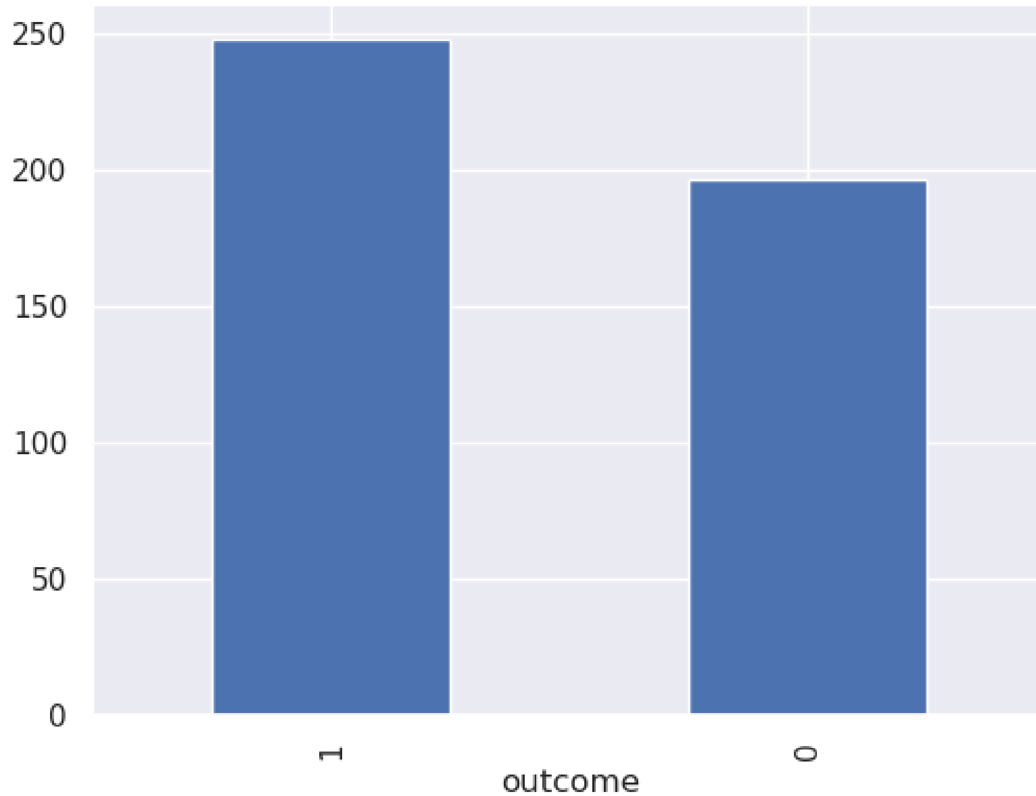
[19]: <Axes: >



1.4 Data Preparation

[21]:
```
#check how well outcome is balanced
print(df.outcome.value_counts())
p=df.outcome.value_counts().plot(kind="bar")
```

```
outcome
1    248
0    196
Name: count, dtype: int64
```



[22]:
```
df
```

[22]:

	center	treat	sex	age	baseline	visit	outcome
1	1	P	M	46	0	1	0
2	1	P	M	46	0	2	0
3	1	P	M	46	0	3	0
4	1	P	M	46	0	4	0
5	1	P	M	28	0	1	0
..	…	…	..	…	…	…	…
440	2	A	F	63	1	4	1
441	2	A	M	31	1	1	1
442	2	A	M	31	1	2	1
443	2	A	M	31	1	3	1
444	2	A	M	31	1	4	1

[444 rows x 7 columns]

1.4.1 Convert Categorical features to numeric

[24]:
```python
# Convert categorical features treat and sex to numeric values

from sklearn.preprocessing import LabelEncoder

df[['treat','sex']] = df[['treat', 'sex']].apply(LabelEncoder().fit_transform)
df
```

[24]:
```
     center  treat  sex  age  baseline  visit  outcome
1         1      1    1   46         0      1        0
2         1      1    1   46         0      2        0
3         1      1    1   46         0      3        0
4         1      1    1   46         0      4        0
5         1      1    1   28         0      1        0
..      …      …    …   …        …      …        …
440       2      0    0   63         1      4        1
441       2      0    1   31         1      1        1
442       2      0    1   31         1      2        1
443       2      0    1   31         1      3        1
444       2      0    1   31         1      4        1

[444 rows x 7 columns]
```

1.4.2 Separate Features from Target

[26]:
```python
# Separate the features for the target

#Features
X = df.drop('outcome', axis=1)
X.head()
```

[26]:
```
   center  treat  sex  age  baseline  visit
1       1      1    1   46         0      1
2       1      1    1   46         0      2
3       1      1    1   46         0      3
4       1      1    1   46         0      4
5       1      1    1   28         0      1
```

[27]:
```python
# Target
y = df['outcome']
y.head()
```

[27]:
```
1    0
2    0
3    0
4    0
```

```
5    0
Name: outcome, dtype: int64
```

1.4.3 Address imbalance for outcome values

[29]:
```
# Address imbalance between outcome values

from imblearn.over_sampling import RandomOverSampler

#Oversampling & fit
ros = RandomOverSampler()
X_res,y_res = ros.fit_resample(X,y)

#Before and after oversampling counts
from collections import Counter
print('Original dataset shape {}'. format(Counter(y)))
print('Resampled dataset shape {}'. format(Counter(y_res)))
```

```
Original dataset shape Counter({1: 248, 0: 196})
Resampled dataset shape Counter({0: 248, 1: 248})
```

1.5 Model Building

Model selection:

Decision Tree

Logistic Regression

1.5.1 Split the data into training and testing data using the train_test_split function

[32]:
```
# Split the data into training and test sets
# set random_state so that train data will be constant For every run
# test_size = 0.2.  20% of data will be used for testing, 80% for training

X_train, X_test, y_train, y_test = train_test_split(X_res,y_res,test_size = 0.
 ↪33, random_state = 42)
```

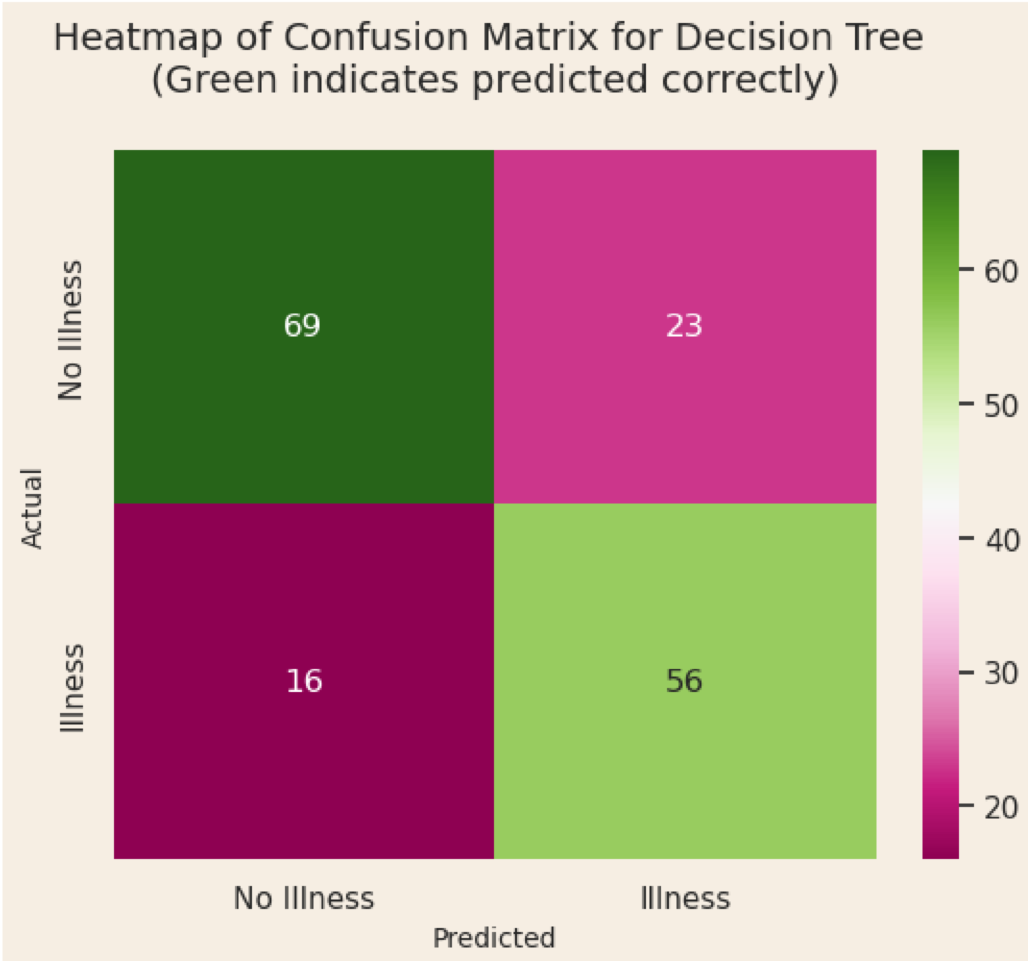
1.5.2 Decision Tree

Build model using Decision Tree

[113]:
```
from sklearn.tree import DecisionTreeClassifier
#model
dtree_model = Pipeline([('scaler', StandardScaler()), ('selector',␣
 ↪VarianceThreshold()) ,('Decision_Tree', DecisionTreeClassifier(random_state␣
 ↪= 42))])
```

```python
#fit
dtree_model.fit(X_train, y_train)

# predict
dtree_pred = dtree_model.predict(X_test)

# Check accuracy
#precision:  out of all the YES predications how many were correct?
#recall:  how good was the model at predicting all YES events
#accuracy: out of the predictions made by the model, what percentage is correct?
#f1 score:   F1 score incorporates both precision and recall into a single␣
↪metric, and a high F1 score is a sign of a well-performing model

from sklearn.metrics import classification_report

print("Classification Report for Decision Tree")
print(classification_report(y_test,dtree_pred))

# Define the classes of the outcomes
classes = ['No Illness', 'Illness']
sns.set(rc={'figure.facecolor':'#F6EEE3'})
sns.heatmap(confusion_matrix(y_test,dtree_pred), annot=True,␣
↪fmt="d",cmap="PiYG",xticklabels=classes, yticklabels=classes)


plt.title('Heatmap of Confusion Matrix for Decision Tree \n (Green indicates␣
↪predicted correctly) \n', fontsize = 14) # title with fontsize 20
plt.xlabel('Predicted', fontsize = 10) # x-axis label with fontsize 15
plt.ylabel('Actual', fontsize = 10) # y-axis label with fontsize 15

plt.show()
```

```
Classification Report for Decision Tree
              precision    recall  f1-score   support

           0       0.81      0.75      0.78        92
           1       0.71      0.78      0.74        72

    accuracy                           0.76       164
   macro avg       0.76      0.76      0.76       164
weighted avg       0.77      0.76      0.76       164
```



The Decision Tree model has an accuracy of 0.76. The model predicted 125 (69+56) of 164 correctly.

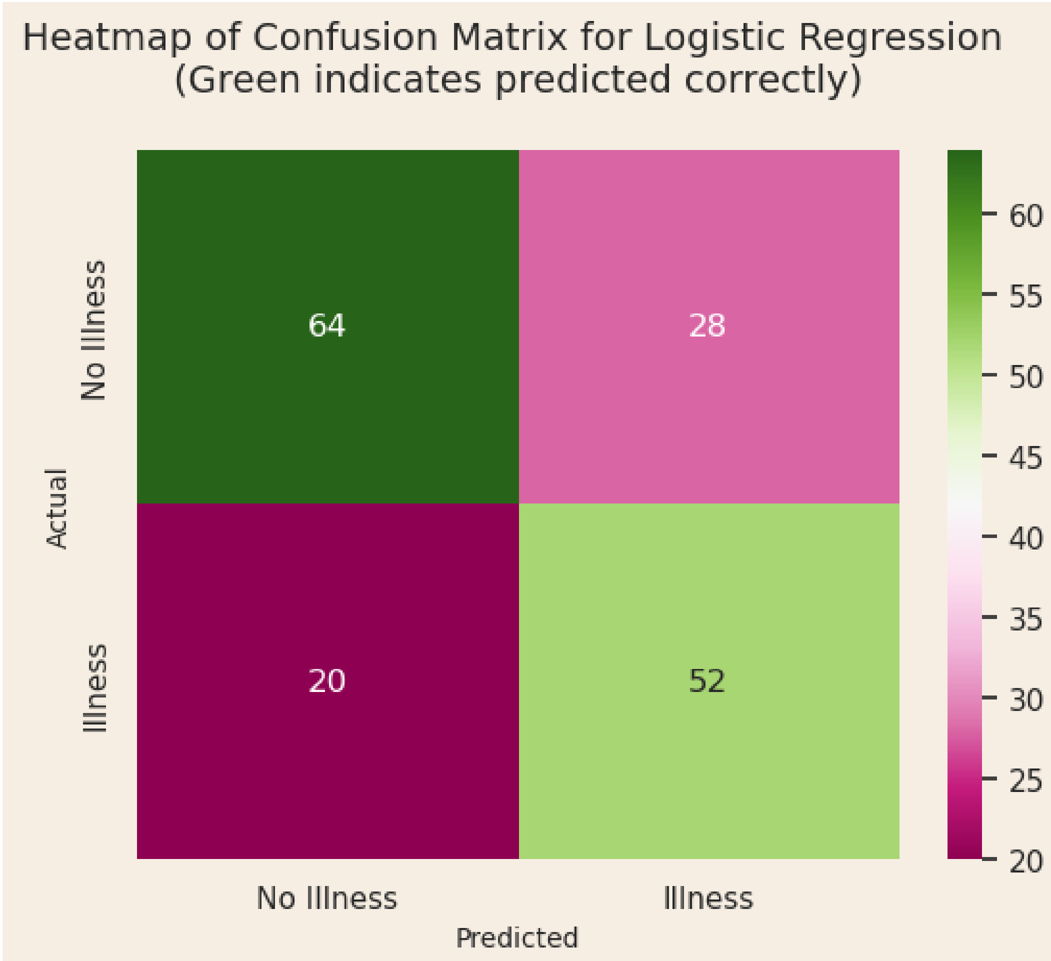
1.5.3 Logistic Regression

Build model using Logistic Regression

[45]:
```
from sklearn.linear_model import LogisticRegression

# model
lg_model = Pipeline([('scaler', StandardScaler()), ('selector',␣
↪VarianceThreshold()),('lr', LogisticRegression(random_state = 42))])

#fit
lg_model.fit(X_train, y_train)

#predict
lg_pred = lg_model.predict(X_test)
```

```python
# Check accuracy
#precision:  out of all the YES predications how many were correct?
#recall:  how good was the model at predicting all YES events
#accuracy: out of the predictions made by the model, what percentage is correct?
#f1 score:   F1 score incorporates both precision and recall into a single␣
↪metric, and a high F1 score is a sign of a well-performing model

#from sklearn.metrics import classification_report

print("Classification Report for Logistic Regression")
print(classification_report(y_test,lg_pred))

# Define the classes of the outcomes
classes = ['No Illness', 'Illness']
sns.set(rc={'figure.facecolor':'#F6EEE3'})
sns.heatmap(confusion_matrix(y_test,lg_pred), annot=True,␣
↪fmt="d",cmap="PiYG",xticklabels=classes, yticklabels=classes)

plt.title('Heatmap of Confusion Matrix for Logistic Regression \n (Green␣
↪indicates predicted correctly) \n', fontsize = 14) # title with fontsize 20
plt.xlabel('Predicted', fontsize = 10) # x-axis label with fontsize 15
plt.ylabel('Actual', fontsize = 10) # y-axis label with fontsize 15
plt.show()
```

```
Classification Report for Logistic Regression
              precision    recall  f1-score   support

           0       0.76      0.70      0.73        92
           1       0.65      0.72      0.68        72

    accuracy                           0.71       164
   macro avg       0.71      0.71      0.71       164
weighted avg       0.71      0.71      0.71       164
```



The Logistic Regression model has an accuracy of 0.71. The model predicted 116 (64+52) of 164 correctly.

1.6 Summarize Model Results

[47]:
```
Accuracy_Summary = pd.DataFrame({"Accuracy":
                              [metrics.accuracy_score(y_test,dtree_pred),
                              metrics.accuracy_score(y_test,lg_pred)]},
                   index = ["Decision Tree", "Logistic Regression"])
Accuracy_Summary
```

[47]:

	Accuracy
Decision Tree	0.762195
Logistic Regression	0.707317

1.6.1 Conclusion from Model Building

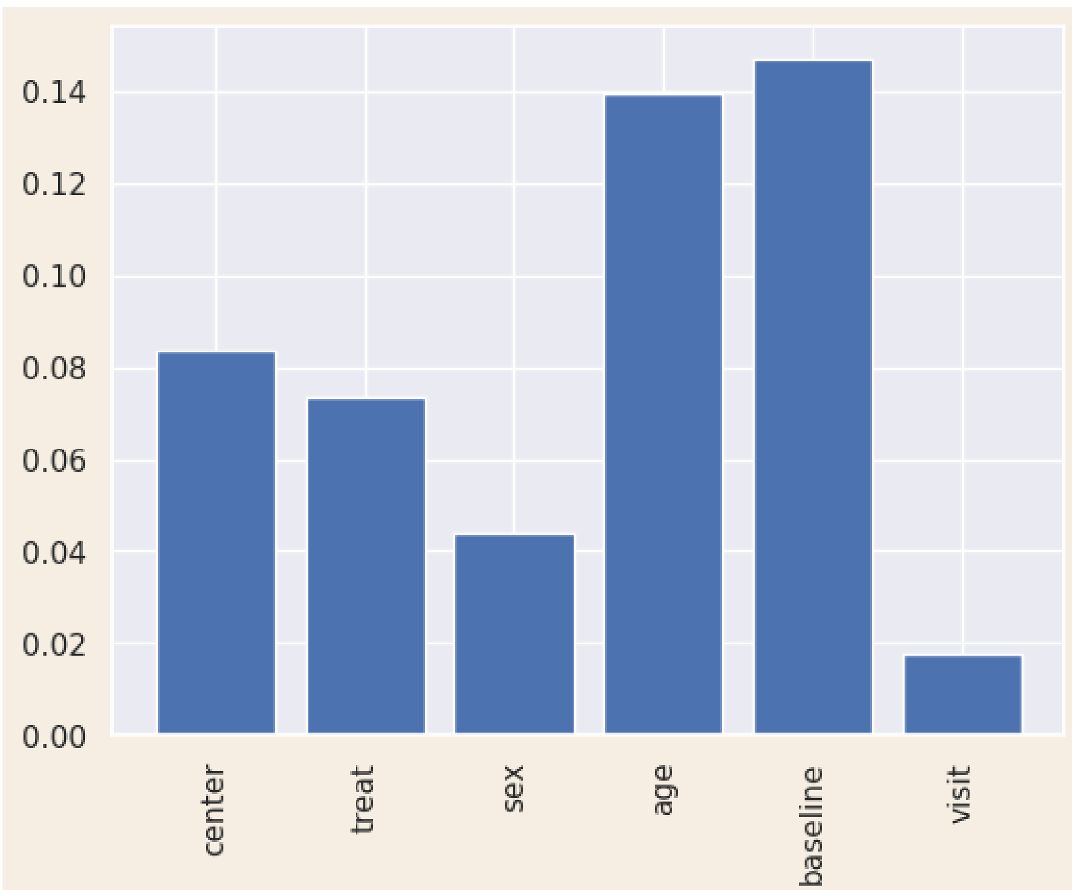
Decision Tree has the best results at 0.76.

1.6.2 Get Feature Importance - Decision Tree Model

[49]:
```
from sklearn.inspection import permutation_importance
feature_importances = permutation_importance(
    dtree_model, X_test, y_test, n_repeats=10, random_state=42
)
```

1.6.3 Plot Feature Importance - Decision Tree Model

[51]:
```
import matplotlib.pyplot as plt

features = X_train.columns
importances = feature_importances.importances_mean

plt.bar(features, importances)
plt.xticks(rotation=90)
plt.show()
```



The above graph shows age and baseline are the most important features for the Decision Tree model.