Contents

1 Executive Summary

Decision tree learning is one of the predictive modelling approaches used in machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (branches) to conclusions about the item's target value (leaves). We usually deal with two types of tree models:

- Tree models where the target variable can take a discrete set of values are called classification trees
- Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Our focus here is on the classification trees.

2 Introduction

I plan to build a Decision tree classifier in python to estimate the vulnerability of patients in having a stroke. By identifying relevent features which may lead to stroke, medical professional can provide meedical advice to their patients. In order to improve the accuracy of the model, I will utilize and compare different algorithms here. I will also address how having an imbalanced dataset may lead to having a wrong prediction for the minorty class in a dataset. I also discuss several techniques to address this issue. In the last step of my analysis I will focus on Dalex for interpreting the performance of the classification models.

3 Loading and Exploring Data

3.1 Loading libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

The columns' titles were located at the first row of the raw data so I seperated them.

3.2 Loading Data

In [3]:
```
# Making a list of missing value types
missing_values = ["n/a", "na", "--"]
data1=pd.read_csv('Downloads/train_2v.csv', header=0, na_values = missing_values)
```

In [4]:
```
data1.head()
```

Out[4]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	30669	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	
1	30468	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	
2	16523	Female	8.0	0	0	No	Private	Urban	110.89	17.6	NaN	
3	56543	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	
4	46136	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	NaN	

In [5]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43400 entries, 0 to 43399
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 43400 non-null  int64
 1   gender             43400 non-null  object
 2   age                43400 non-null  float64
 3   hypertension       43400 non-null  int64
 4   heart_disease      43400 non-null  int64
 5   ever_married       43400 non-null  object
 6   work_type          43400 non-null  object
 7   Residence_type     43400 non-null  object
 8   avg_glucose_level  43400 non-null  float64
 9   bmi                41938 non-null  float64
 10  smoking_status     30108 non-null  object
 11  stroke             43400 non-null  int64
dtypes: float64(3), int64(4), object(5)
memory usage: 4.0+ MB
```

3.3 Data size and structure

In [7]:
```
data1.shape
```

Out[7]: (43400, 12)

In [8]:
```
data1.describe()
```

Out[8]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	43400.000000	43400.000000	43400.000000	43400.000000	43400.000000	41938.000000	43400.000000
mean	36326.142350	42.217894	0.093571	0.047512	104.482750	28.605038	0.018041
std	21072.134879	22.519649	0.291235	0.212733	43.111751	7.770020	0.133103
min	1.000000	0.080000	0.000000	0.000000	55.000000	10.100000	0.000000
25%	18038.500000	24.000000	0.000000	0.000000	77.540000	23.200000	0.000000
50%	36351.500000	44.000000	0.000000	0.000000	91.580000	27.700000	0.000000
75%	54514.250000	60.000000	0.000000	0.000000	112.070000	32.900000	0.000000
max	72943.000000	82.000000	1.000000	1.000000	291.050000	97.600000	1.000000

It seems we have some missing values at BMI.

In [9]:
```
data1['stroke'] = pd.to_numeric(data1['stroke'],errors='coerce')
```

In [69]:
```
data1.std()
```

Out[69]:
```
id                   21072.134879
age                     22.519649
hypertension             0.291235
heart_disease            0.212733
avg_glucose_level       43.111751
bmi                      7.770020
stroke                   0.133103
dtype: float64
```

In [71]:
```
data1.nunique()
```

Out[71]:
```
id                   43400
gender                   3
age                    104
hypertension             2
heart_disease            2
ever_married             2
work_type                5
Residence_type           2
avg_glucose_level    12543
bmi                    555
smoking_status           3
stroke                   2
dtype: int64
```

In [73]:
```
data1.isna().sum()
```

Out[73]:
```
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                 1462
smoking_status     13292
stroke                 0
dtype: int64
```

In [7]:
```
13292/43400*100
```

Out[7]: 30.62672811059908

for BMI missing data acounts for 3.3 % of all rows so we try to do imputation for smoking_status missing data acounts for 30.6 % of all rows so we try to do imputation

In [10]:
```
data1 = data1.dropna()
```

In [9]:
```
data1.gender.unique()
```

Out[9]: array(['Male', 'Female', 'Other'], dtype=object)

In [10]:
```
data1.work_type.unique()
```

Out[10]:
```
array(['Private', 'Self-employed', 'Govt_job', 'children', 'Never_worked'],
      dtype=object)
```

In [13]:
```
data1.smoking_status.unique()
```

Out[13]: array(['never smoked', 'formerly smoked', 'smokes'], dtype=object)

In [14]:
```
data1.Residence_type.unique()
```

Out[14]: array(['Urban', 'Rural'], dtype=object)

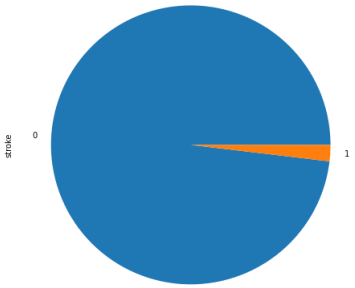
In [11]:
```
data1.drop(['id'], axis=1, inplace=True)
```

'ID' as a categorical variable will add a hige dimension to our data set so we will delete it but I will keep it for now.

4 Imbalanced data

In [12]:
```
data1['stroke'].value_counts().plot(kind='pie', figsize=(8,8))
data1['stroke'].value_counts()
```

Out[12]:
```
0    28524
1      548
Name: stroke, dtype: int64
```



our data set is imbalanced and thus we should not use traditional models like logistic regression.

4.1 Up-sample Minority Class

Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce its signal.

In [27]:
```
from sklearn.utils import resample
not_stroke = data1[data1.stroke==0]
stroke = data1[data1.stroke==1]
```

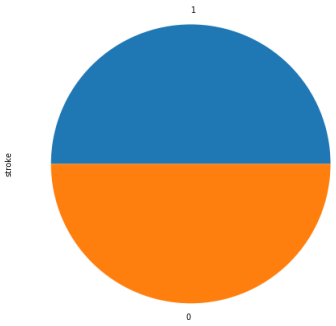
In [28]:
```
from sklearn.utils import resample
stroke_upsampled = resample(stroke,
                            replace=True, # sample with replacement
                            n_samples=len(not_stroke), # match number in majority class
                            random_state=27) # reproducible results
```

In [267]:
```
upsampled = pd.concat([not_stroke, stroke_upsampled])
# check new class counts
upsampled.stroke.value_counts()
```

Out[267]:
```
0    29072
Name: stroke, dtype: int64
```

In [10]:
```
upsampled['stroke'].value_counts().plot(kind='pie', figsize=(8,8))
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24b4ce7ad48>
```



4.1.1 Creating Dummy variables

In [268]:
```
cat_list= list(data1.select_dtypes(['object']).columns)
cat_list
dummy_ranks=pd.get_dummies(upsampled[cat_list])
dummy_ranks.head(1)
upsampled = pd.get_dummies(upsampled, columns = cat_list, drop_first=True)
```

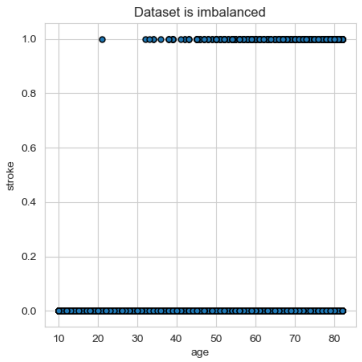
The above mentioned techniques leads to duplication of records multiple times in the resulting dataset! We want to use the general principle of bootstrap to sample with replacement from our minority class, but we want to adjust each re-sampled value to avoid exact duplicates of our original data. This is where the Synthetic Minority Oversampling TEchnique (SMOTE) algorithm comes in.

4.2 SMOTE

The SMOTE algorithm can be broken down into four steps:

Randomly pick a point from the minority class. Compute the k-nearest neighbors (for some pre-specified k) for this point. Add k new points somewhere between the chosen point and each of its neighbors. For example, let k = 5. Then we randomly pick a point from the minority class. Next, we compute its 5-nearest neighbors from points that are also in the minority class. Finally for each neighbor, we compute the line segment connecting the chosen point to its neighbor and add a new point somewhere along that line.

In [275]:
```
width_in_inches = 5
height_in_inches = 5
dots_per_inch = 100
plt.figure(
    figsize=(width_in_inches, height_in_inches),
    dpi=dots_per_inch)
plt.title('Dataset is imbalanced')
plt.xlabel('age')
plt.ylabel('stroke')
plt.scatter(data1[0:]['age'], data1['stroke'], marker='o',
            s=25, edgecolor='k', cmap=plt.cm.coolwarm)
plt.show()
```



4.2.1 Creating Dummy variables

In [243]:
```
cat_list= list(data1.select_dtypes(['object']).columns)
cat_list
dummy_ranks=pd.get_dummies(data1[cat_list])
dummy_ranks.head(1)
SmoteSample = pd.get_dummies(data1, columns = cat_list, drop_first=True)
SmoteSample
```

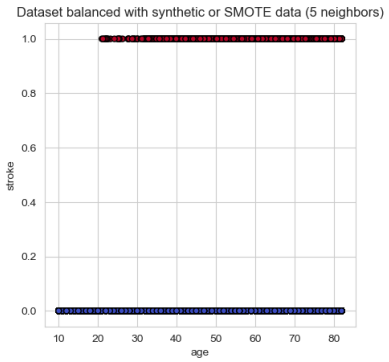
Out[243]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Male	gender_Other	ever_married_Yes	work_type_Never_worked
1	58.0	1	0	87.96	39.2	0	1	0	1	0
3	70.0	0	0	69.04	35.9	0	0	0	1	0
6	52.0	0	0	77.59	17.7	0	0	0	1	0
7	75.0	0	1	243.53	27.0	0	0	0	1	0
8	32.0	0	0	77.67	32.3	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...
43395	10.0	0	0	58.64	20.4	0	0	0	0	0
43396	56.0	0	0	213.61	55.4	0	0	0	1	0
43397	82.0	1	0	91.94	28.9	0	0	0	1	0
43398	40.0	0	0	99.16	33.2	0	1	0	1	0
43399	82.0	0	0	79.48	20.6	0	0	0	1	0

29072 rows × 16 columns

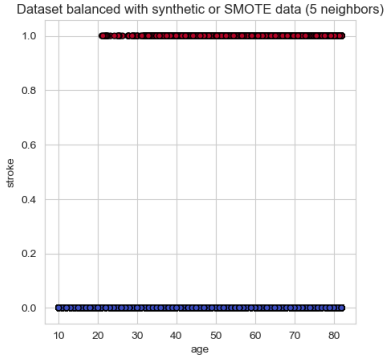
In [245]:
```
from imblearn.over_sampling import SMOTE
# for reproducibility purposes
seed = 100
# SMOTE number of neighbors
k = 1
#df = pd.read_csv('df_imbalanced.csv', encoding='utf-8', engine='python')
# make a new df made of all the columns, except the target class
X = SmoteSample.loc[:, SmoteSample.columns != 'stroke']
y = data1['stroke']
sm = SMOTE(sampling_strategy='auto', k_neighbors=k, random_state=seed)
X_res, y_res = sm.fit_resample(X, y)
```

In [276]:
```
width_in_inches = 5
height_in_inches = 5
dots_per_inch = 100
plt.figure(
    figsize=(width_in_inches, height_in_inches),
    dpi=dots_per_inch)
plt.title('Dataset balanced with synthetic or SMOTE data ({} neighbors)'.format(k))
plt.xlabel('age')
plt.ylabel('stroke')
plt.scatter(X_res[0:]['age'], y_res[0:], marker='o', c=y_res,
            s=25, edgecolor='k', cmap=plt.cm.coolwarm)
plt.show()
```



In [259]:
```
from imblearn.over_sampling import SMOTE
# for reproducibility purposes
seed = 100
# SMOTE number of neighbors
k = 5
#df = pd.read_csv('df_imbalanced.csv', encoding='utf-8', engine='python')
# make a new df made of all the columns, except the target class
X = SmoteSample.loc[:, SmoteSample.columns != 'stroke']
y = data1['stroke']
sm = SMOTE(sampling_strategy='auto', k_neighbors=k, random_state=seed)
X_res, y_res = sm.fit_resample(X, y)
```

In [277]:
```
width_in_inches = 5
height_in_inches = 5
dots_per_inch = 100
plt.figure(
    figsize=(width_in_inches, height_in_inches),
    dpi=dots_per_inch)
plt.title('Dataset balanced with synthetic or SMOTE data ({} neighbors)'.format(k))
plt.xlabel('age')
plt.ylabel('stroke')
plt.scatter(X_res[0:]['age'], y_res[0:], marker='o', c=y_res,
            s=25, edgecolor='k', cmap=plt.cm.coolwarm)
plt.show()
```



4.3 Down-sample Majority Class

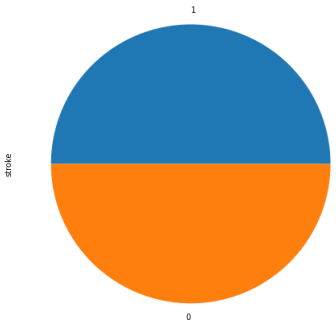
Down-sampling involves randomly removing observations from the majority class to prevent its signal from dominating the learning algorithm.

In [13]:
```
from sklearn.utils import resample
not_stroke = data1[data1.stroke==0]
stroke = data1[data1.stroke==1]

# Downsample majority class
stroke_upsampled = resample(not_stroke,
                                 replace=False,    # sample without replacement
                                 n_samples=548,     # to match minority class
                                 random_state=27) # reproducible results

# Combine minority class with downsampled majority class
downsampled = pd.concat([stroke_upsampled, stroke])

# Display new class counts
downsampled.stroke.value_counts()
```

Out[13]:
```
1    548
0    548
Name: stroke, dtype: int64
```

In [14]:
```
downsampled['stroke'].value_counts().plot(kind='pie', figsize=(8,8))
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x268c9264cc8>



4.3.1 Creating Dummy variables

In [15]:
```
cat_list= list(data1.select_dtypes(['object']).columns)
dummy_ranks=pd.get_dummies(downsampled[cat_list])
dummy_ranks.head(1)
downsampled = pd.get_dummies(downsampled, columns = cat_list, drop_first=True)
downsampled
```

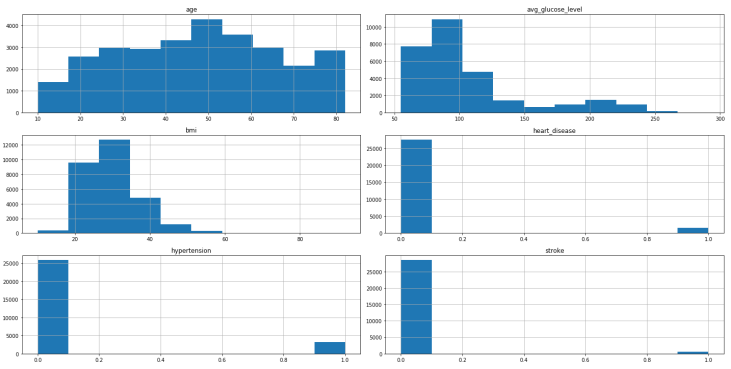
Out[15]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Male	ever_married_Yes	work_type_Never_worked	work_type_Pr
1551	12.0	0	0	81.66	23.5	0	0	0	0	
27730	38.0	0	0	112.69	23.4	0	0	1	0	
37555	75.0	1	0	91.50	26.5	0	0	1	0	
7796	78.0	0	0	97.70	27.7	0	0	1	0	
22185	62.0	1	0	213.45	32.2	0	1	1	0	
...	...	...	...	...	...	...	...	...	...	
43076	79.0	0	1	88.29	36.0	1	1	1	0	
43119	76.0	0	0	93.38	26.7	1	1	1	0	
43148	56.0	0	0	83.27	32.9	1	0	1	0	
43304	80.0	0	0	75.91	26.7	1	0	1	0	
43318	62.0	1	1	77.97	31.5	1	1	1	0	

1096 rows × 15 columns

5 Distributions of observations within numerical variables

In [88]:
```
data1.hist(figsize=(20, 10))
plt.tight_layout()
```



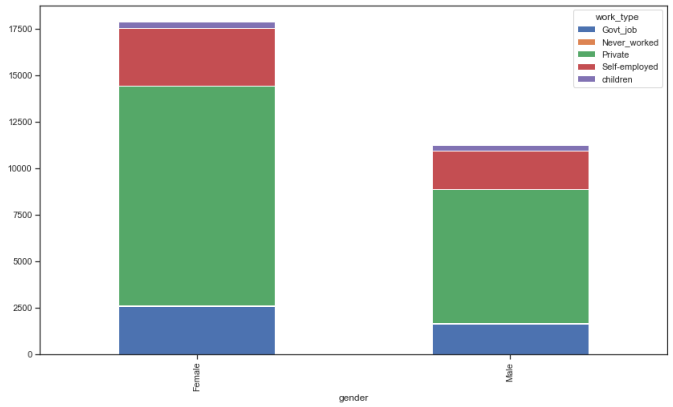
we may need some binning with respect to avg_glucose_level or BMI

6 Distributions of observations within categories

In [5]:
```
data1.stroke.unique()
```

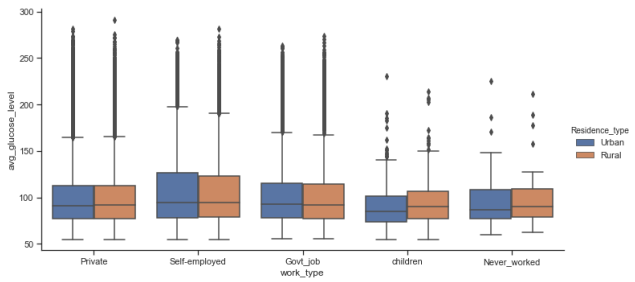
Out[5]: array([0, 1], dtype=int64)

In [6]:
```
sns.set(style="ticks")
filtered_data = data1[(data1['gender'].isin(['Male', 'Female'])) & (data1['work_type'].isin(['Private', 'Self-emp
df_plot = filtered_data.groupby(['gender', 'work_type']).size().reset_index().pivot(columns='gender', index='work
g = df_plot.set_index('work_type').T.plot(kind='bar', stacked=True, color=sns.color_palette())
g.figure.set_size_inches(14,8)
plt.show()
```



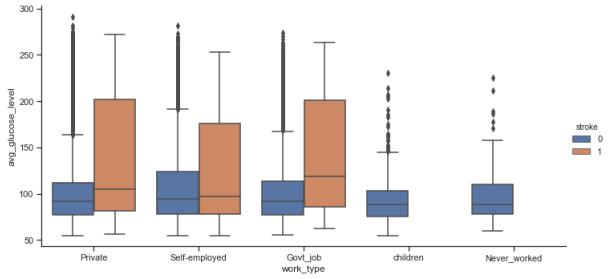
In [19]:
```
sns.catplot(x="work_type", y="avg_glucose_level", data=data1, kind="box", hue="Residence_type", height=5, aspect=
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x1a3f33ed488>



In [20]:
```
sns.catplot(x="work_type", y="avg_glucose_level", data=data1, kind="box", hue="stroke", height=5, aspect=2)
```

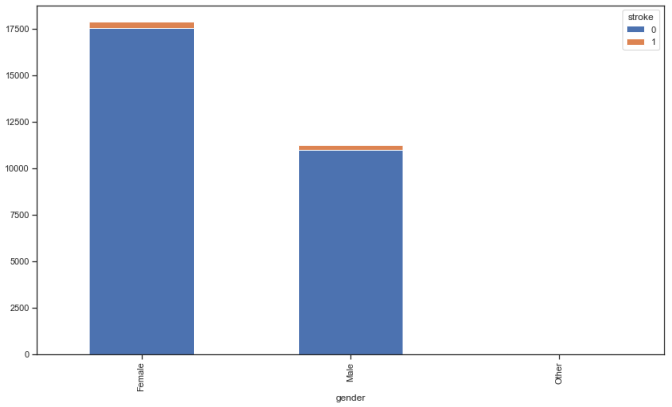
Out[20]: <seaborn.axisgrid.FacetGrid at 0x1a3f4421748>



In [21]:
```
data1.columns
```

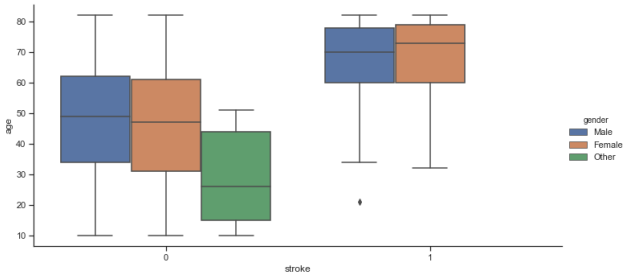
Out[21]:
```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

In [22]:
```
sns.set(style="ticks")
filtered_data = data1[(data1['gender'].isin(['Female', 'Male', 'Other'])) & (data1['stroke'].isin([0,1])) ]
df_plot = filtered_data.groupby(['gender', 'stroke']).size().reset_index().pivot(columns='gender', index='stroke'
g = df_plot.set_index('stroke').T.plot(kind='bar', stacked=True, color=sns.color_palette())
g.figure.set_size_inches(14,8)
plt.show()
```



In [23]:
```
sns.catplot(x='stroke',  y="age", data=data1, kind="box", hue="gender", height=5, aspect=2)
```

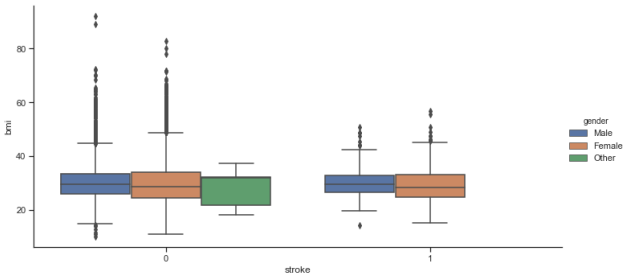
Out[23]: <seaborn.axisgrid.FacetGrid at 0x1a3f3689d88>



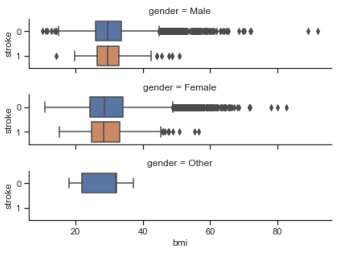
g = sns.catplot(x="smoking_status", y="bmi", data=data1, kind="violin", inner=None, height=5, aspect=2, hue="stroke")
sns.swarmplot(x="smoking_status", y="bmi", data=data1, color="k", size=3, ax=g.ax)

In [24]:
```
sns.catplot(x='stroke',  y="bmi", data=data1, kind="box", hue="gender", height=5, aspect=2)
```

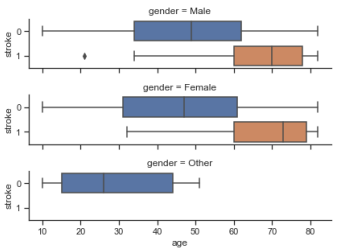
Out[24]: <seaborn.axisgrid.FacetGrid at 0x1a3f36c9488>



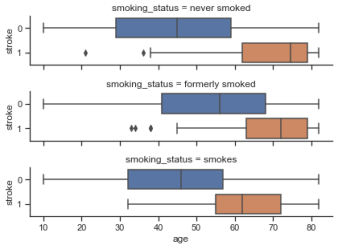
In [25]:
```
g = sns.catplot(x="bmi", y="stroke", row="gender",kind="box", orient="h", height=1.5, aspect=4, data=data1.query(
```



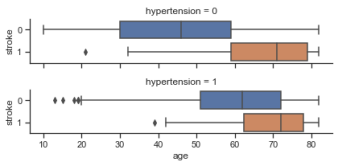
In [26]:
```
g = sns.catplot(x="age", y="stroke", row="gender",kind="box", orient="h", height=1.5, aspect=4, data=data1.query(
```



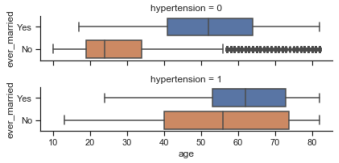
In [27]:
```
g = sns.catplot(x="age", y="stroke", row="smoking_status",kind="box", orient="h", height=1.5, aspect=4, data=data
```



In [28]:
```
g = sns.catplot(x="age", y="stroke", row="hypertension",kind="box", orient="h", height=1.5, aspect=4, data=data1)
```

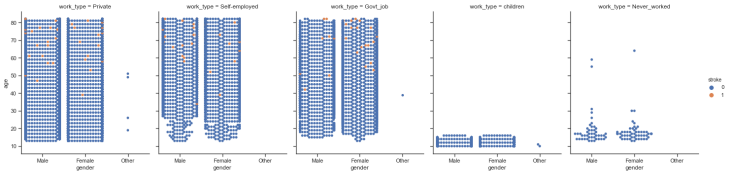


In [29]:
```
g = sns.catplot(x="age", y="ever_married", row="hypertension",kind="box", orient="h", height=1.5, aspect=4, data=
```



In [100]:
```
sns.catplot(x="gender", y="age", data=data1, hue="stroke",col="work_type", aspect=.8,kind="swarm")
```

Out[100]:
```
<seaborn.axisgrid.FacetGrid at 0x20de01da0c8>
```



We have less churn in long term contracts. I guess that is the idea of customer locking :-) also higher frequency of churns is visible in short term contracts with payments through Electronic checks.

7 Statistics, Missing data, label encoding

In [16]:
```
cat_list= list(data1.select_dtypes(['object']).columns)
cat_list
```

Out[16]:
```
['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
```

I performed the label encoding on relevant features.

In [17]:
```
dummy_ranks=pd.get_dummies(data1[cat_list])
dummy_ranks.head(1)
```

Out[17]:

	gender_Female	gender_Male	gender_Other	ever_married_No	ever_married_Yes	work_type_Govt_job	work_type_Never_worked	work_type_Priva
1	0	1	0	0	1	0	0	

In [17]:
```

data1 = pd.get_dummies(data1, columns = cat_list)
```

In [6]:
```
data1.columns
```

Out[6]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke', 'gender_Female', 'gender_Male', 'gender_Other',
       'ever_married_No', 'ever_married_Yes', 'work_type_Govt_job',
       'work_type_Never_worked', 'work_type_Private',
       'work_type_Self-employed', 'work_type_children', 'Residence_type_Rural',
       'Residence_type_Urban', 'smoking_status_formerly smoked',
       'smoking_status_never smoked', 'smoking_status_smokes'],
      dtype='object')
```

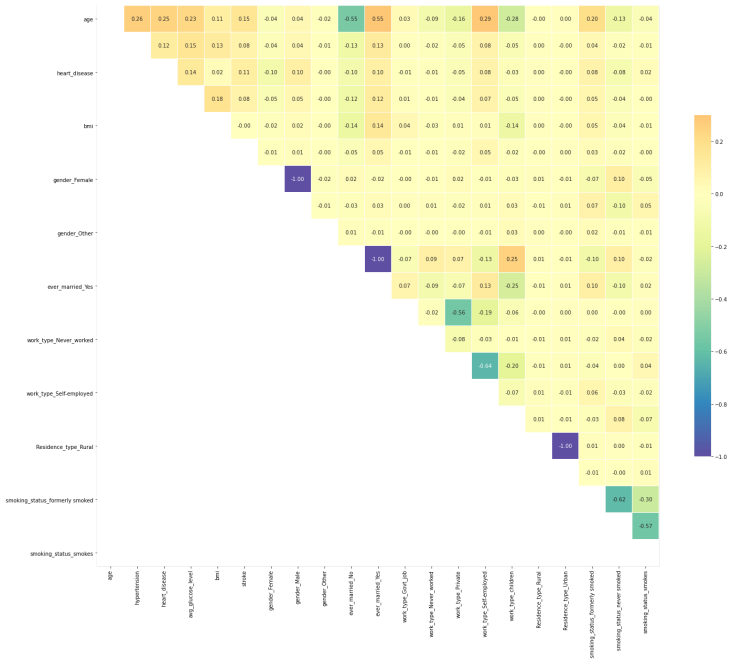
In [18]:
```
data1 = data1.dropna()
data1.isna().sum()
```

Out[18]:
```
age                              0
hypertension                     0
heart_disease                    0
avg_glucose_level                0
bmi                              0
stroke                           0
gender_Female                    0
gender_Male                      0
gender_Other                     0
ever_married_No                  0
ever_married_Yes                 0
work_type_Govt_job               0
work_type_Never_worked           0
work_type_Private                0
work_type_Self-employed          0
work_type_children               0
Residence_type_Rural             0
Residence_type_Urban             0
smoking_status_formerly smoked   0
smoking_status_never smoked      0
smoking_status_smokes            0
dtype: int64
```

7.1 The Correlations

7.1.1 Heatmap of the Variables

In [19]:
```
corr = data1.corr()
mask = np.tril(corr)
g = sns.heatmap(corr,  vmax=.3, center=0, square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True, fmt='.
g.figure.set_size_inches(24,24)
plt.show()
```



8 Statistical estimation

8.1 Classification using Logistic Regression

In [19]:
```
from sklearn.metrics import accuracy_score, mean_squared_error as mse
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression as LR
```

8.1.1 Prepare data for model training and testing input

In [20]:
```
y = downsampled.stroke.values
X = downsampled.drop(["stroke"],axis=1)
```

In [22]:
```
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size = 0.3, random_state=101)
```

8.1.2 Model training and testing input

In [23]:
```
logreg = LR(C = 1)
logreg = logreg.fit(X_train,y_train)
prediction = logreg.predict(X_test)
print("Mean-squared error using Logistic Regression:", mse(y_test, prediction))
print("Accuracy with Logistic Regression:",accuracy_score(y_test, prediction))
```

```
Mean-squared error using Logistic Regression: 0.20364741641337386
Accuracy with Logistic Regression: 0.7963525835866262
```

In [24]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, prediction)
print(confusion_matrix)
```

```
[[132  40]
 [ 27 130]]
```

In [25]:
```
logreg.coef_
```

Out[25]:
```
array([[ 0.0790865 ,  0.54665015,  0.78175534,  0.00283867, -0.00349646,
         0.1418785 ,  0.3577944 , -0.05543579,  0.3554652 ,  0.38241569,
        -0.51143456,  0.30151514,  0.39554308,  0.59250453]])
```

8.1.3 Looking at the p values

In [27]:
```
downsampled.columns = downsampled.columns.str.replace(' ', '_')
downsampled.columns = downsampled.columns.str.replace('(', '')
downsampled.columns = downsampled.columns.str.replace(')', '')
downsampled.columns = downsampled.columns.str.replace('-', '_')
downsampled.columns
```

Out[27]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke', 'gender_Male', 'ever_married_Yes', 'work_type_Never_worked',
       'work_type_Private', 'work_type_Self_employed', 'work_type_children',
       'Residence_type_Urban', 'smoking_status_never_smoked',
       'smoking_status_smokes'],
      dtype='object')
```

In [28]:
```
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression
import statsmodels.api as smf
from scipy import stats
X2 = smf.add_constant(X)
est = smf.OLS(y, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.381
Model:                            OLS   Adj. R-squared:                  0.373
Method:                 Least Squares   F-statistic:                     47.55
Date:                Thu, 24 Sep 2020   Prob (F-statistic):          2.59e-102
Time:                        21:32:42   Log-Likelihood:                -532.51
No. Observations:                1096   AIC:                             1095.
Df Residuals:                    1081   BIC:                             1170.
Df Model:                          14
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           -0.4778      0.083     -5.772      0.000      -0.640      -0.315
age                              0.0143      0.001     18.074      0.000       0.013       0.016
hypertension                     0.1010      0.032      3.205      0.001       0.039       0.163
heart_disease                    0.1431      0.038      3.736      0.000       0.068       0.218
avg_glucose_level                0.0006      0.000      2.388      0.017    9.98e-05       0.001
bmi                             -0.0006      0.002     -0.313      0.755      -0.004       0.003
gender_Male                      0.0112      0.025      0.445      0.656      -0.038       0.061
ever_married_Yes                -0.0198      0.035     -0.574      0.566      -0.088       0.048
work_type_Never_worked           0.1490      0.399      0.373      0.709      -0.634       0.932
work_type_Private                0.0544      0.035      1.544      0.123      -0.015       0.124
work_type_Self-employed          0.0788      0.041      1.929      0.054      -0.001       0.159
work_type_children               0.2536      0.112      2.268      0.024       0.034       0.473
Residence_type_Urban             0.0222      0.024      0.920      0.358      -0.025       0.069
smoking_status_never smoked      0.0003      0.029      0.010      0.992      -0.056       0.057
smoking_status_smokes            0.0718      0.035      2.037      0.042       0.003       0.141
==============================================================================
Omnibus:                       34.413   Durbin-Watson:                   0.775
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               25.158
Skew:                          -0.268   Prob(JB):                     3.44e-06
Kurtosis:                       2.487   Cond. No.                     4.84e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.84e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [29]:
```
est2.params
```

Out[29]:
```
const                          -0.477822
age                             0.014348
hypertension                    0.101042
heart_disease                   0.143118
avg_glucose_level               0.000560
bmi                            -0.000579
gender_Male                     0.011236
ever_married_Yes               -0.019808
work_type_Never_worked          0.149027
work_type_Private               0.054406
work_type_Self-employed         0.078777
work_type_children              0.253588
Residence_type_Urban            0.022153
smoking_status_never smoked     0.000292
smoking_status_smokes           0.071832
dtype: float64
```

8.1.4 Converting the coefficients of the logistic regression model into odd ratios

In [30]:
```
model_odds = pd.DataFrame(np.exp(est2.params), columns= ['OR'])
model_odds['z-value']= est2.pvalues
model_odds[['2.5%', '97.5%']] = np.exp(est2.conf_int())
model_odds
```

Out[30]:

	OR	z-value	2.5%	97.5%
const	0.620132	1.020463e-08	0.527163	0.729497
age	1.014451	5.204230e-64	1.012873	1.016033
hypertension	1.106323	1.388992e-03	1.039965	1.176916
heart_disease	1.153866	1.968899e-04	1.070310	1.243946
avg_glucose_level	1.000560	1.712321e-02	1.000100	1.001021
bmi	0.999421	7.545502e-01	0.995793	1.003061
gender_Male	1.011300	6.560558e-01	0.962469	1.062608
ever_married_Yes	0.980387	5.662553e-01	0.916175	1.049100
work_type_Never_worked	1.160704	7.089090e-01	0.530442	2.539830
work_type_Private	1.055913	1.228787e-01	0.985374	1.131502
work_type_Self-employed	1.081963	5.398119e-02	0.998650	1.172226
work_type_children	1.288641	2.354286e-02	1.034760	1.604812
Residence_type_Urban	1.022400	3.578935e-01	0.975207	1.071877
smoking_status_never smoked	1.000292	9.918987e-01	0.945447	1.058318
smoking_status_smokes	1.074475	4.192672e-02	1.002631	1.151466

8.1.5 Example of Interpretations

When making an initial check of a model it is usually most useful to look at the column called z, which shows the z-statistics. The way we read this is that the further a value is from 0, the stronger its role as a predictor. The negative sign tells us that as the value of the variable increases, the probability of churning decreases. We can also see that the weakest predictors have z scores close to 0.
P-values less than the threshold which is 0.05 are considered non-significant.So, we can remove them but we will check whether removing them increases the chi2(chi squared) score and decreases the deviance. Because chi2 measure how good the model fits to the data and deviance measures badness of fit(higher the number ,worst the fit.)

8.2 Classification using Decision Tree

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from IPython.display import Image
from sklearn import tree
```

In [32]:
```
downsampled.columns = downsampled.columns.str.replace(' ', '_')
downsampled.columns = downsampled.columns.str.replace('(', '')
downsampled.columns = downsampled.columns.str.replace(')', '')
downsampled.columns = downsampled.columns.str.replace('-', '_')
downsampled.columns
```

Out[32]:
```
Index(['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
       'stroke', 'gender_Male', 'ever_married_Yes', 'work_type_Never_worked',
       'work_type_Private', 'work_type_Self_employed', 'work_type_children',
       'Residence_type_Urban', 'smoking_status_never_smoked',
       'smoking_status_smokes'],
      dtype='object')
```

In [33]:
```
y = downsampled.stroke.values
X = downsampled.drop(["stroke"],axis=1)
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [35]:
```
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

Out[35]:
```
DecisionTreeClassifier()
```

In [36]:
```
y_pred = dt.predict(X_test)
```

In [37]:
```
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[98 46]
 [42 88]]
```

In [38]:
```
print('Training accuracy:', dt.score(X_train, y_train))
print('Test accuracy:', dt.score(X_test, y_test))
print('')
```

```
Training accuracy: 1.0
Test accuracy: 0.6788321167883211
```

In [ ]:
```
fn=['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi', 'gender_Female', 'gender_Male', 'gender_O
       'ever_married_No', 'ever_married_Yes', 'work_type_Govt_job',
       'work_type_Never_worked', 'work_type_Private',
       'work_type_Self-employed', 'work_type_children', 'Residence_type_Rural',
       'Residence_type_Urban', 'smoking_status_formerly_smoked',
       'smoking_status_never_smoked', 'smoking_status_smokes']
cn=['true' , 'false']

width_in_inches = 50
height_in_inches = 50
dots_per_inch = 300
plt.figure(
    figsize=(width_in_inches, height_in_inches),dpi=dots_per_inch)
fig, axes = plt.subplots(nrows = 1,ncols = 1)
tree.plot_tree(dt, feature_names = fn, class_names=cn, filled = True, fontsize=10, rounded=True)
fig = plt.gcf()
#fig.set_size_inches(25, 25)
#tight_layout()
```

In [18]:
```
fig.savefig('first_dt.png')
```

8.2.1 Hyper parameter tuning

In [40]:
```
from sklearn.model_selection import GridSearchCV
#tree_param = [{'criterion': ['entropy', 'gini'], 'max_depth':np.arange(3, 15)},
              #{'min_samples_leaf': min_samples_leaf_range}]
tree_param = {
    'criterion': ['entropy', 'gini'],
    'max_depth': [8, 9],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [100, 200]
}
dt_pt = GridSearchCV(estimator = dt, param_grid= tree_param, cv=5, n_jobs = -1, verbose = 200)
```

In [ ]:
```
dt_pt.fit(X_train, y_train)
```

In [43]:
```
dt_pt.best_params_
```

Out[43]:
```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 3,
 'min_samples_leaf': 3,
 'min_samples_split': 100}
```

In [44]:
```
dt = DecisionTreeClassifier(criterion='gini',max_depth=8,max_features=3, min_samples_leaf=3,min_samples_split= 10
dt.fit(X_train, y_train)
```

Out[44]:
```
DecisionTreeClassifier(max_depth=8, max_features=3, min_samples_leaf=3,
                       min_samples_split=100)
```

In [45]:
```
y_preddt = dt.predict(X_test)
```

In [46]:
```
print('Training accuracy:', dt.score(X_train, y_train))
print('Test accuracy:', dt.score(X_test, y_test))
print('')
```

```
Training accuracy: 0.7396593673965937
Test accuracy: 0.7299270072992701
```

In [ ]:
```
fn=['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi', 'gender_Female', 'gender_Male', 'gender_O
       'ever_married_No', 'ever_married_Yes', 'work_type_Govt_job',
       'work_type_Never_worked', 'work_type_Private',
       'work_type_Self-employed', 'work_type_children', 'Residence_type_Rural',
       'Residence_type_Urban', 'smoking_status_formerly_smoked',
       'smoking_status_never_smoked', 'smoking_status_smokes']
cn=['true' , 'false']
width_in_inches = 50
height_in_inches = 50
dots_per_inch = 300
plt.figure(
    figsize=(width_in_inches, height_in_inches),
    dpi=dots_per_inch)
fig, axes = plt.subplots(nrows = 1,ncols = 1)
#plt.rcParams['figure.figsize'] = [10, 10]
plt.show
tree.plot_tree(dt, feature_names = fn, class_names=cn, filled = True, fontsize=10, rounded=True)
fig = plt.gcf()
#fig.set_size_inches(25, 25)
#plt.tight_layout()
```

In [23]:
```
fig.savefig('second_dt.png')
```

8.3 Gradient Boosting With LightGBM

In [80]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
import lightgbm as lgb
from lightgbm import LGBMClassifier
from sklearn.model_selection import GridSearchCV
```

In [81]:
```
y = downsampled.stroke.values
X = downsampled.drop(["stroke"],axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [83]:
```
def algorithm_pipeline(X_train_data, X_test_data, y_train_data, y_test_data,
                       model, param_grid, cv=10, scoring_fit='neg_mean_squared_error',
                       do_probabilities = True):
    gs = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=cv,
        n_jobs=-1,
        scoring=scoring_fit,
        verbose=2
    )
    fitted_model = gs.fit(X_train_data, y_train_data)

    if do_probabilities:
        pred = fitted_model.predict_proba(X_test_data)
    else:
        pred = fitted_model.predict(X_test_data)

    return fitted_model, pred
```

In [84]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [85]:
```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

In [86]:
```
lgbmodel = lgb.LGBMClassifier()
param_grid = {
    'n_estimators': [100],
    'colsample_bytree': [0.7, 0.8],
    'max_depth': [9],
    'num_leaves': [50],
    'reg_alpha': [1.1],
    'reg_lambda': [1.1],
    'min_split_gain': [0.3, 0.4],
    'subsample': [0.7],
    'subsample_freq': [20]
}

lgb_model, pred = algorithm_pipeline(X_train, X_test, y_train, y_test, lgbmodel,
                                     param_grid, cv=cv, scoring_fit='accuracy')

print(lgb_model.best_score_)
print(lgb_model.best_params_)
```

```
Fitting 30 folds for each of 4 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  58 tasks      | elapsed:    1.7s

0.7518219218336762
{'colsample_bytree': 0.8, 'max_depth': 9, 'min_split_gain': 0.4, 'n_estimators': 100, 'num_leaves': 50, 'reg_alpha':
1.1, 'reg_lambda': 1.1, 'subsample': 0.7, 'subsample_freq': 20}

[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:    3.1s finished
```

In [87]:
```
lgbmodel = lgb.LGBMClassifier(colsample_bytree= 0.8, max_depth= 9, min_split_gain= 0.4, n_estimators= 100, num_le
```

In [88]:
```
lgbmodel.fit(X_train, y_train)
```

Out[88]:
```
LGBMClassifier(colsample_bytree=0.8, max_depth=9, min_split_gain=0.4,
               num_leaves=50, reg_alpha=1.1, reg_lambda=1.1, subsample=0.7,
               subsample_freq=20)
```

In [89]:
```
y_predlgb = lgbmodel.predict(X_test)
```

In [90]:
```
print('Training accuracy:', lgbmodel.score(X_train, y_train))
print('Test accuracy:', lgbmodel.score(X_test, y_test))
print('')
```

```
Training accuracy: 0.8321167883211679
Test accuracy: 0.7737226277372263
```

Light GBM builds trees one at a time, where each new tree helps to correct errors made by previously trained tree. It performs the optimization in function space (rather than in parameter space) which makes the use of custom loss functions much easier. It is also faster in speed and accuracy as compared to bagging and adaptive boosting. It is capable of performing equally good with large data sets with a significant reduction in training time as compared to XGBOOST. But along with these advantages there is most disadvantageous feature of LGBM i.e. parameter tuning in LightGBM. It should be done carefully. Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have large number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored.

8.4 Gradient Boosting With Catboost

In [56]:
```
from catboost import CatBoostClassifier
from sklearn.model_selection import cross_val_score
y = downsampled.stroke.values
X = downsampled.drop(["stroke"],axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [57]:
```
import pickle
Cat_model = CatBoostClassifier(verbose=0, n_estimators=10)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
Cat_model.fit(X_train, y_train)
# make a single prediction
```

Out[57]:
```
<catboost.core.CatBoostClassifier at 0x268cd828148>
```

In [58]:
```
y_pred = Cat_model.predict(X_test)
```

In [59]:
```
print('Training accuracy:', Cat_model.score(X_train, y_train))
print('Test accuracy:', Cat_model.score(X_test, y_test))
print('')
```

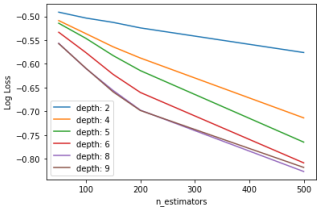
```
Training accuracy: 0.8102189781021898
Test accuracy: 0.7846715328467153
```

8.5 Gradient Boosting With XGBoost Decision Tree

In [66]:
```
import os
import matplotlib.pyplot as plt
os.environ["PATH"] += os.pathsep + 'C:/Myprogram/Library/bin/graphviz/'
from sklearn.tree import export_graphviz
from sklearn.model_selection import StratifiedKFold
import numpy as np
from numpy import loadtxt
from xgboost import XGBClassifier
from xgboost import plot_tree
import matplotlib.pyplot as plt
xgb_model = XGBClassifier()
n_estimators = [50, 100, 150, 200, 500]
max_depth = [2, 4, 5, 6, 8, 9]
#learning_rate= [0.1, 0.05]
param_grid = dict(max_depth=max_depth, n_estimators=n_estimators)
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=101)
grid_search = GridSearchCV(xgb_model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold, verbose=1)
grid_result = grid_search.fit(X_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
scores = np.array(means).reshape(len(max_depth), len(n_estimators))
for i, value in enumerate(max_depth):
    plt.plot(n_estimators, scores[i], label='depth: ' + str(value))
plt.legend()
plt.xlabel('n_estimators')
plt.ylabel('Log Loss')

#xgb_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  76 tasks      | elapsed:    5.5s
[Parallel(n_jobs=-1)]: Done 143 out of 150 | elapsed:   13.9s remaining:    0.6s

Best: -0.490995 using {'max_depth': 2, 'n_estimators': 50}
-0.490995 (0.045497) with: {'max_depth': 2, 'n_estimators': 50}
-0.503376 (0.051511) with: {'max_depth': 2, 'n_estimators': 100}
-0.512521 (0.052728) with: {'max_depth': 2, 'n_estimators': 150}
-0.524140 (0.054534) with: {'max_depth': 2, 'n_estimators': 200}
-0.576055 (0.062971) with: {'max_depth': 2, 'n_estimators': 500}
-0.508703 (0.042495) with: {'max_depth': 4, 'n_estimators': 50}
-0.536284 (0.048833) with: {'max_depth': 4, 'n_estimators': 100}
-0.564011 (0.057288) with: {'max_depth': 4, 'n_estimators': 150}
-0.587614 (0.067131) with: {'max_depth': 4, 'n_estimators': 200}
-0.713800 (0.102298) with: {'max_depth': 4, 'n_estimators': 500}
-0.514297 (0.037655) with: {'max_depth': 5, 'n_estimators': 50}
-0.546199 (0.053765) with: {'max_depth': 5, 'n_estimators': 100}
-0.582492 (0.060390) with: {'max_depth': 5, 'n_estimators': 150}
-0.614194 (0.068897) with: {'max_depth': 5, 'n_estimators': 200}
-0.764912 (0.102642) with: {'max_depth': 5, 'n_estimators': 500}
-0.533291 (0.043456) with: {'max_depth': 6, 'n_estimators': 50}
-0.576234 (0.053919) with: {'max_depth': 6, 'n_estimators': 100}
-0.622163 (0.061561) with: {'max_depth': 6, 'n_estimators': 150}
-0.660228 (0.065516) with: {'max_depth': 6, 'n_estimators': 200}
-0.808615 (0.099469) with: {'max_depth': 6, 'n_estimators': 500}
-0.556614 (0.039274) with: {'max_depth': 8, 'n_estimators': 50}
-0.609719 (0.055653) with: {'max_depth': 8, 'n_estimators': 100}
-0.656091 (0.059258) with: {'max_depth': 8, 'n_estimators': 150}
-0.697518 (0.069323) with: {'max_depth': 8, 'n_estimators': 200}
-0.826837 (0.094939) with: {'max_depth': 8, 'n_estimators': 500}
-0.557239 (0.033941) with: {'max_depth': 9, 'n_estimators': 50}
-0.609237 (0.045997) with: {'max_depth': 9, 'n_estimators': 100}
-0.659038 (0.063184) with: {'max_depth': 9, 'n_estimators': 150}
-0.698234 (0.067299) with: {'max_depth': 9, 'n_estimators': 200}
-0.818248 (0.084329) with: {'max_depth': 9, 'n_estimators': 500}

[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:   15.8s finished
```

Out[66]: Text(0, 0.5, 'Log Loss')



In [67]:
```
xgb_model = XGBClassifier(max_depth= 2, n_estimators= 50, learning_rate=0.05)
xgb_model.fit(X_train, y_train)
```

Out[67]: XGBClassifier(learning_rate=0.05, max_depth=2, n_estimators=50)

In [68]:
```
ypredxgb=xgb_model.predict(X_test)
```

In [69]:
```
print('Training accuracy:', xgb_model.score(X_train, y_train))
print('Test accuracy:', xgb_model.score(X_test, y_test))
print('')
```

```
Training accuracy: 0.7834549878345499
Test accuracy: 0.781021897810219
```

In [ ]:
```
fig = plt.figure(dpi=300)
plot_tree(xgb_model, rankdir='LR')
fig = plt.gcf()
fig.set_size_inches(10, 10)
plt.tight_layout()
plt.show()
```

8.6 Random Forest Classifier

In [71]:
```
from nested_cv import NestedCV
```

In [72]:
```
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
```

In [73]:
```
RFmodel=RandomForestClassifier()
param_grid = {
    'max_depth': [3, 6, 9],
    'n_estimators': [10, 20 , 100],
   'max_features' : [3, 5, 10]
}

NCV = NestedCV(model=RFmodel, params_grid=param_grid,
               outer_kfolds=5, inner_kfolds=5,
               cv_options={'metric':roc_auc_score,
                           'metric_score_indicator_lower':False,
                           'randomized_search_iter':30,
                           'predict_proba':True})
NCV.fit(X=X_train, y=y_train)

NCV.outer_scores

```

Out[73]:
```
[0.8224423418095802,
 0.848295621510432,
 0.8464631422189128,
 0.8036458333333334,
 0.8246996996996997]
```

In [74]:
```
model_param_grid = NCV.best_params
model_param_grid
```

Out[74]:
```
{'max_depth': [6, 3],
 'max_features': [5, 3, 10],
 'n_estimators': [20, 100, 10]}
```

In [75]:
```
RFmodel=RandomForestClassifier(max_depth= 6 , n_estimators= 20 , max_features=5 )
```

In [76]:
```
RFmodel.fit(X=X_train, y=y_train)
```

Out[76]:
```
RandomForestClassifier(max_depth=6, max_features=5, n_estimators=20)
```

In [77]:
```
RFypred=RFmodel.predict(X_test)
```

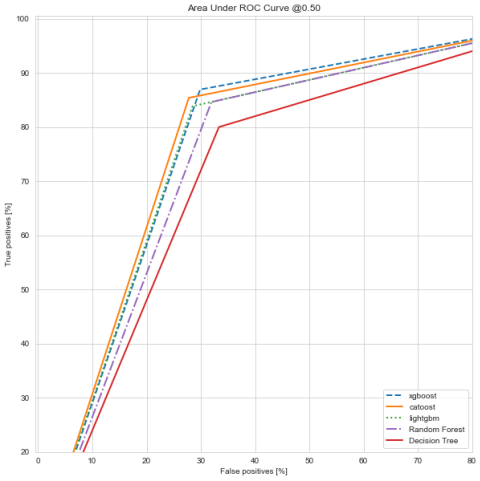
In [78]:
```
print('Training accuracy:', RFmodel.score(X_train, y_train))
print('Test accuracy:', RFmodel.score(X_test, y_test))
print('')
```

```
Training accuracy: 0.8272506082725061
Test accuracy: 0.7591240875912408
```

Nested Cross-Validation with Grid Search(useful for running a GridSearchCV that is unbiased). In Nested Cross-Validation you get the optimal bias-variance trade-off and, by the theory, as unbiased of a score as possible. https://mlfromscratch.com/nested-cross-validation-python-code/#/ (https://mlfromscratch.com/nested-cross-validation-python-code/#/)

In [91]:
```
plt.rcParams['figure.figsize'] = (12, 10)
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
from sklearn.metrics import roc_curve
def plot_roc(name, labels, predictions, p=0.5, **kwargs):
  fp, tp, _ = roc_curve(labels, predictions)
  plt.plot(100*fp, 100*tp, label=name, linewidth=2, **kwargs)
  plt.xlabel('False positives [%]')
  plt.ylabel('True positives [%]')
  plt.xlim([-0.5,80])
  plt.title('Area Under ROC Curve @{:.2f}'.format(p))
  plt.ylim([20,100.5])
  plt.grid(True)
  ax = plt.gca()
  ax.set_aspect('equal')
sns.set_style("whitegrid")
plot_roc("xgboost", y_test, ypredxgb, color=colors[0],linestyle='--')
plot_roc("catoost", y_test, y_pred, color=colors[1],linestyle='-')
plot_roc("lightgbm", y_test, y_predlgb, color=colors[2],linestyle=':')
plot_roc("Random Forest", y_test, RFypred, color=colors[4],linestyle='-.')
plot_roc("Decision Tree", y_test, y_preddt, color=colors[3],linestyle='-')
plt.legend(loc='lower right')
```

Out[91]:
```
<matplotlib.legend.Legend at 0x268ce4ec2c8>
```



Some models have better skills

9 Interpretable discussions using Dalex

In [92]:
```
import dalex as dx
```

In [93]:
```
exp = dx.Explainer(xgb_model, data=X_train, y=y_train, label='XGBOOST')
exp2 = dx.Explainer(Cat_model, data=X_train, y=y_train, label='CatBOOST')
exp3 = dx.Explainer(lgb_model, data=X_train, y=y_train, label='LightGBM')
exp4 = dx.Explainer(dt, data=X_train, y=y_train, label='Decision Tree')
exp5 = dx.Explainer(RFmodel, data=X_train, y=y_train, label='Random Forest')

```

```
Preparation of a new explainer is initiated

  -> data              : 822 rows 14 cols
  -> target variable   : 822 values
  -> model_class       : xgboost.sklearn.XGBClassifier (default)
  -> label             : XGBOOST
  -> predict function  : <function yhat_proba_default at 0x00000268CE564CA8> will be used (default)
  -> predicted values  : min = 0.0743, mean = 0.508, max = 0.88
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.877, mean = 0.000346, max = 0.907
  -> model_info        : package xgboost

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 822 rows 14 cols
  -> target variable   : 822 values
  -> model_class       : catboost.core.CatBoostClassifier (default)
  -> label             : CatBOOST
  -> predict function  : <function yhat_proba_default at 0x00000268CE564CA8> will be used (default)
  -> predicted values  : min = 0.0348, mean = 0.506, max = 0.93
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.874, mean = 0.00207, max = 0.934
  -> model_info        : package catboost

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 822 rows 14 cols
  -> target variable   : 822 values
  -> model_class       : sklearn.model_selection._search.GridSearchCV (default)
  -> label             : LightGBM
  -> predict function  : <function yhat_proba_default at 0x00000268CE564CA8> will be used (default)
  -> predicted values  : min = 0.0194, mean = 0.505, max = 0.957
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.931, mean = 0.0033, max = 0.96
  -> model_info        : package sklearn

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 822 rows 14 cols
  -> target variable   : 822 values
  -> model_class       : sklearn.tree._classes.DecisionTreeClassifier (default)
  -> label             : Decision Tree
  -> predict function  : <function yhat_proba_default at 0x00000268CE564CA8> will be used (default)
  -> predicted values  : min = 0.0145, mean = 0.509, max = 0.893
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.893, mean = -1.4e-17, max = 0.986
  -> model_info        : package sklearn

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 822 rows 14 cols
  -> target variable   : 822 values
  -> model_class       : sklearn.ensemble._forest.RandomForestClassifier (default)
  -> label             : Random Forest
  -> predict function  : <function yhat_proba_default at 0x00000268CE564CA8> will be used (default)
  -> predicted values  : min = 0.0156, mean = 0.514, max = 0.991
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.829, mean = -0.00529, max = 0.855
  -> model_info        : package sklearn

A new explainer has been created!
```

In [94]:
```
exp.dump(open('explainer_xgboost.pickle', 'wb'))
exp2.dump(open('explainer_Catboost.pickle', 'wb'))
exp3.dump(open('explainer_Lightgbm.pickle', 'wb'))
exp4.dump(open('explainer_DecisionTree.pickle', 'wb'))
exp5.dump(open('explainer_RandomForest.pickle', 'wb'))
```

```
  -> Residual function is local, thus has to be dropped.
  -> Residual function is local, thus has to be dropped.
Finished loading model, total used 73 iterations
  -> Residual function is local, thus has to be dropped.
  -> Residual function is local, thus has to be dropped.
  -> Residual function is local, thus has to be dropped.
```

9.1 Analysis on the Dataset Level

9.1.1 model_performance

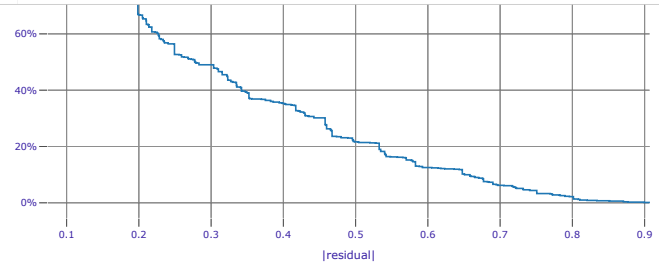
'classification' 'regression' This function calculates various Model Performance measures: classification: F1, accuracy, recall, precision and AUC regression: mean squared error, R squared, median absolute deviation

In [95]:
```
results= pd.DataFrame(columns=['Method', 'recall', 'precision', 'f1', 'accuracy','auc'])
mp= exp.model_performance(model_type = 'classification')
mp2= exp2.model_performance(model_type = 'classification')
mp3= exp3.model_performance(model_type = 'classification')
mp4= exp4.model_performance(model_type = 'classification')
mp5= exp5.model_performance(model_type = 'classification')
results.loc[0]= ["XGBOOST", mp.result['recall'][0],mp.result['precision'][0],mp.result['f1'][0],mp.result['accura
results.loc[1]= ["CatBOOST", mp2.result['recall'][0],mp2.result['precision'][0],mp2.result['f1'][0],mp2.result['a
results.loc[2]= ["LightGBM", mp3.result['recall'][0],mp3.result['precision'][0],mp3.result['f1'][0],mp3.result['a
results.loc[3]= ["Decision Tree", mp4.result['recall'][0],mp4.result['precision'][0],mp4.result['f1'][0],mp4.resu
results.loc[4]= ["Random Forest", mp5.result['recall'][0],mp5.result['precision'][0],mp5.result['f1'][0],mp5.resu
results
```

Out[95]:

	Method	recall	precision	f1	accuracy	auc
0	XGBOOST	0.858852	0.751046	0.801339	0.783455	0.862061
1	CatBOOST	0.854067	0.789823	0.820690	0.810219	0.890829
2	LightGBM	0.863636	0.816742	0.839535	0.832117	0.898491
3	Decision Tree	0.820574	0.711618	0.762222	0.739659	0.808079
4	Random Forest	0.863636	0.809417	0.835648	0.827251	0.928366

In [97]:
```
# sample plot for Xgboost
mp.plot()
```



9.1.2 model_parts

'variable_importance' 'ratio' 'difference'

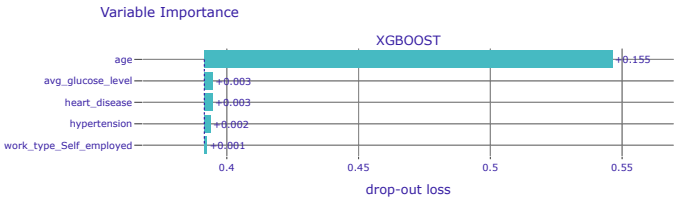
This function calculates Variable Importance.

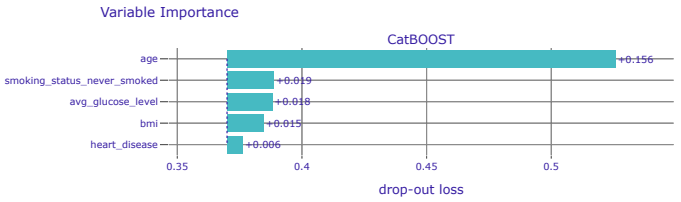
In [98]:
```
vi = exp.model_parts()
vi2 = exp2.model_parts()
vi3 = exp3.model_parts()
vi4 = exp4.model_parts()
vi5 = exp5.model_parts()
vi4.result
```

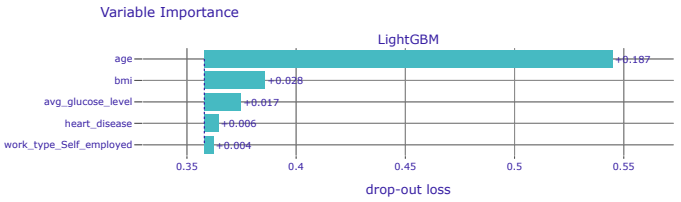
Out[98]:

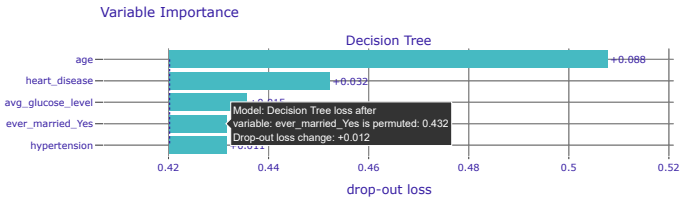
	variable	dropout_loss	label
0	_full_model_	0.420222	Decision Tree
1	smoking_status_smokes	0.420222	Decision Tree
2	work_type_Never_worked	0.420222	Decision Tree
3	work_type_children	0.420222	Decision Tree
4	smoking_status_never_smoked	0.420769	Decision Tree
5	gender_Male	0.420778	Decision Tree
6	Residence_type_Urban	0.421135	Decision Tree
7	work_type_Private	0.425686	Decision Tree
8	work_type_Self_employed	0.427283	Decision Tree
9	bmi	0.430504	Decision Tree
10	hypertension	0.431706	Decision Tree
11	ever_married_Yes	0.431746	Decision Tree
12	avg_glucose_level	0.435715	Decision Tree
13	heart_disease	0.452299	Decision Tree
14	age	0.507802	Decision Tree
15	_baseline_	0.566203	Decision Tree

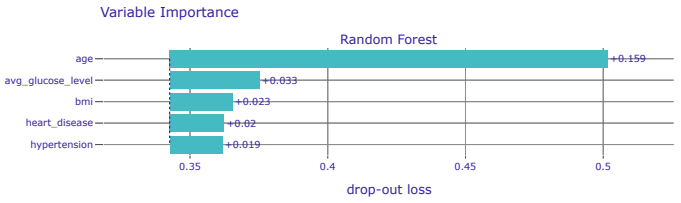
In [99]:
```
vi.plot(max_vars=5)
vi2.plot(max_vars=5)
vi3.plot(max_vars=5)
vi4.plot(max_vars=5)
vi5.plot(max_vars=5)
```











9.1.3 model_profile

This function calculates explanations that explore model response as a function of selected variables.The explanations can be calulated as Partial Dependence Profile or Accumulated Local Dependence Profile.

In [100]:
```
pdp_num = exp.model_profile(type = 'partial')
pdp_num.result["_label_"] = 'pdp'
```

```
Calculating ceteris paribus!: 100%|██████████████████████████████████████| 14/14 [00:00<00:00, 36.29it/s]
```

Ceteris-paribus (CP) profiles show how a model's prediction would change if the value of a single exploratory variable changed. In essence, a CP profile shows the dependence of the conditional expectation of the dependent variable (response) on the values of the particular explanatory variable.

In [101]:
```
ale_num = exp.model_profile(type = 'accumulated')
ale_num.result["_label_"] = 'ale'
```

```
Calculating ceteris paribus!: 100%|| 14/14 [00:00<00:00, 31.91it/s]
Calculating accumulated dependency!: 100%|| 14/14 [00:06<00:00,  2.24it/s]
```

In [ ]:
```
#pdp_num.plot(ale_num)
```

In [88]:
```
pdp_cat = exp.model_profile(type = 'partial', variable_type='numerical', variables = ["gender_Male","Residence_ty
pdp_cat.result['_label_'] = 'pdp'
ale_cat = exp.model_profile(type = 'accumulated', variable_type='numerical', variables = ["gender_Male","Residenc
ale_cat.result['_label_'] = 'ale'
```

```
Calculating ceteris paribus!: 100%|| 14/14 [00:00<00:00, 37.66it/s]
Calculating ceteris paribus!: 100%|| 14/14 [00:00<00:00, 39.02it/s]
Calculating accumulated dependency!: 100%|| 2/2 [00:00<00:00,  3.75it/s]
```

In [ ]:
```
#ale_cat.plot(pdp_cat)
```

9.2 Analysis on the Individual Level

9.2.1 Prediction

In [103]:
```
#"John"
john=X[1:2]
johnp=exp.predict(X)[1:2]
johnp
```

Out[103]: array([0.12539852], dtype=float32)

In [104]:
```
#"Merry"
mary=X[2:3]
maryp=exp.predict(X)[2:3]
maryp
```

Out[104]: array([0.7507369], dtype=float32)

9.2.2 predict_parts

'break_down' 'break_down_interactions' 'shap'

This function calculates Variable Attributions as Break Down, iBreakDown or Shapley Values explanations. Model prediction is decomposed into parts that are attributed for particular variables.

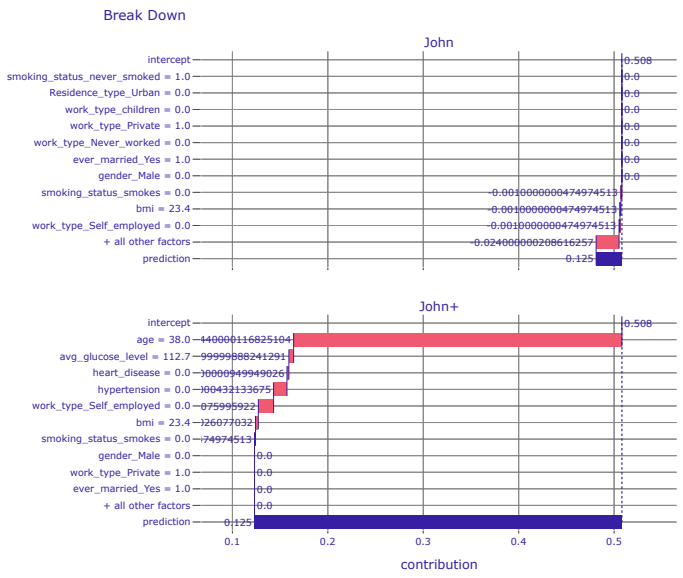
In [105]:
```
bd_john = exp.predict_parts(john, type='break_down')
bd_interactions_john = exp.predict_parts(john, type='break_down_interactions')

sh_mary = exp.predict_parts(mary, type='shap', B = 10)
```

In [106]:
```
bd_john.result.label = "John"
bd_interactions_john.result.label = "John+"

bd_john.result
```

Out[106]:

	variable_name	variable_value	variable	cumulative	contribution	sign	position	label
0	intercept	1	intercept	0.508170	0.508170	1.0	15	John
1	smoking_status_never_smoked	1.0	smoking_status_never_smoked = 1.0	0.508170	0.000000	0.0	14	John
2	Residence_type_Urban	0.0	Residence_type_Urban = 0.0	0.508170	0.000000	0.0	13	John
3	work_type_children	0.0	work_type_children = 0.0	0.508170	0.000000	0.0	12	John
4	work_type_Private	1.0	work_type_Private = 1.0	0.508170	0.000000	0.0	11	John
5	work_type_Never_worked	0.0	work_type_Never_worked = 0.0	0.508170	0.000000	0.0	10	John
6	ever_married_Yes	1.0	ever_married_Yes = 1.0	0.508170	0.000000	0.0	9	John
7	gender_Male	0.0	gender_Male = 0.0	0.508170	0.000000	0.0	8	John
8	smoking_status_smokes	0.0	smoking_status_smokes = 0.0	0.507352	-0.000818	-1.0	7	John
9	bmi	23.4	bmi = 23.4	0.506489	-0.000863	-1.0	6	John
10	work_type_Self_employed	0.0	work_type_Self_employed = 0.0	0.505268	-0.001220	-1.0	5	John
11	hypertension	0.0	hypertension = 0.0	0.502167	-0.003102	-1.0	4	John
12	heart_disease	0.0	heart_disease = 0.0	0.494018	-0.008149	-1.0	3	John
13	avg_glucose_level	112.7	avg_glucose_level = 112.7	0.481764	-0.012254	-1.0	2	John
14	age	38.0	age = 38.0	0.125399	-0.356365	-1.0	1	John
15			prediction	0.125399	0.125399	1.0	0	John

In [107]:
```
bd_john.plot(bd_interactions_john)

```

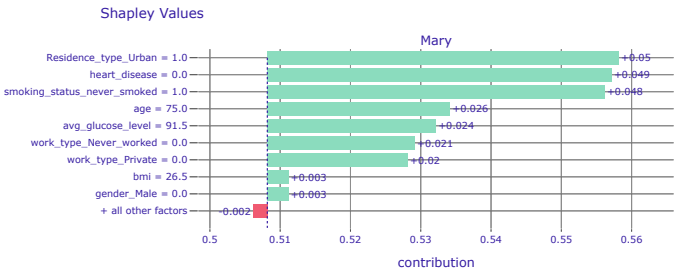


In [108]:
```
sh_mary.result.label = "Mary"
sh_mary.result.loc[sh_mary.result.B == 0, ]
```

Out[108]:

	variable	contribution	variable_name	variable_value	sign	label	B
10	Residence_type_Urban = 1.0	0.050119	Residence_type_Urban	1.0	1.0	Mary	0
2	age = 75.0	0.026045	age	75.0	1.0	Mary	0
0	avg_glucose_level = 91.5	0.023976	avg_glucose_level	91.5	1.0	Mary	0
3	bmi = 26.5	0.003490	bmi	26.5	1.0	Mary	0
7	ever_married_Yes = 1.0	-0.000169	ever_married_Yes	1.0	-1.0	Mary	0
1	gender_Male = 0.0	0.002770	gender_Male	0.0	1.0	Mary	0
6	heart_disease = 0.0	0.048996	heart_disease	0.0	1.0	Mary	0
13	hypertension = 1.0	-0.000479	hypertension	1.0	-1.0	Mary	0
5	smoking_status_never_smoked = 1.0	0.047646	smoking_status_never_smoked	1.0	1.0	Mary	0
4	smoking_status_smokes = 0.0	-0.001848	smoking_status_smokes	0.0	-1.0	Mary	0
12	work_type_Never_worked = 0.0	0.021456	work_type_Never_worked	0.0	1.0	Mary	0
11	work_type_Private = 0.0	0.020197	work_type_Private	0.0	1.0	Mary	0
9	work_type_Self_employed = 1.0	-0.000871	work_type_Self_employed	1.0	-1.0	Mary	0
8	work_type_children = 0.0	0.001236	work_type_children	0.0	1.0	Mary	0

In [109]:
```
sh_mary.plot(bar_width = 16)
```



In [110]:
```
sh_john = exp.predict_parts(john, type='shap', B = 10)
sh_john.result.label = "John"
sh_john.plot(max_vars=5)
```