Progetto di esame: Stroke Prediction from clinical dataset

Programmazione di Applicazioni Data Intensive
Laurea in Ingegneria e Scienze Informatiche
DISI - Università di Bologna, Cesena

Samuele Bertani - Matricola 0000889633

samuele.bertani@studio.unibo.it

Obbiettivo

Lo scopo di questo progetto è quello di predire, attraverso un dataset che andremo ad analizzare, se un paziente ha avuto uno ictus, dato rappresentato da una variabile discreta binaria.

Predisposizione

Andiamo subito a fare il setup delle librerie che verrano utilizzate per il progetto:

In [3]:

```
import numpy as np
import pandas as pd
import seaborn as splot # libreria non utilizzata nelle esercitazioni ma volevo mostrare le mie competenze nel suo utilizzo
import matplotlib.pyplot as plt
%matplotlib inline
```

Importiamo poi il dataset che andremo a valutare:

In [4]:

```
dataset = pd.read_csv("https://raw.githubusercontent.com/SamueleBertani/Samuele/main/healthcare-dataset-stroke-data.csv", sep=",")
```

Controlliamo che il documento sia stato importato con successo:

In [5]:

```
dataset.head()
```

Out[5]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smok
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	nev
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	nev
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	nev

Introduzione Dataset

Andiamo ora a analizzare e comprendere il dataset.

Il dataset è stato scaricato dal seguente link di Kaggle: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

Il dataset è utilizzabile per moditivi educativi e il creatore da citare è https://www.kaggle.com/fedesoriano.

Esso è composto da inserimenti di 12 colonne dove ognuno raffigura un paziente analizzato, le features indicano dati sul singolo paziente e sono di diversa tipologia. Andiamo ora ad analizzarle:

1. **id:** unique identifier
2. **gender:** "Male", "Female" or "Other"
3. **age:** age of the patient
4. **hypertension:** 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5. **heart_disease:** 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6. **ever_married:** "No" or "Yes"
7. **work_type:** "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8. **Residence_type:** "Rural" or "Urban"
9. **avg_glucose_level:** average glucose level in blood
10. **bmi:** body mass index
11. **smoking_status:** "formerly smoked", "never smoked", "smokes" or "Unknown"*
12. **stroke:** 1 if the patient had a stroke or 0 if not

In particolare lo scopo del progetto sarà predirre la colonna 12 ovvero se il paziente ha avuto un ictus o no

Data Cleaning

Osserviamo quanto è grande il dataset e cerchiamo di camprendere eventuali pattern osservabili.

Per prima cosa osserviamo le dimensioni:

In [6]:

```
dataset.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 1.8 MB
```

In [7]:

```
dataset.describe()
```

Out[7]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

Notiamo subito che nonostante le basse dimensioni di poco più di 5000 righe il file pesa 1.8MB che è un peso considerevole. Analizzando i `DataType` osserviamo che essi sono di tre tipologie `int64, object e float64`.

Sappiamo pero che le features che sono di tipo object sono in realtà categorie quindi andiamo a cambiare il tipo di dato.

In [8]:

```
custom_dtypes = {
    "gender": "category",
    "ever_married": "category",
    "work_type": "category",
    "Residence_type": "category",
    "smoking_status": "category",
}
dataset = pd.read_csv("https://raw.githubusercontent.com/SamueleBertani/Samuele/main/healthcare-dataset-stroke-data.csv", dtype=custom_dtypes)
```

Osserviamo ora il peso decisamente diminuito:

In [9]:

```
dataset.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   id                 5110 non-null   int64   
 1   gender             5110 non-null   category
 2   age                5110 non-null   float64 
 3   hypertension       5110 non-null   int64   
 4   heart_disease      5110 non-null   int64   
 5   ever_married       5110 non-null   category
 6   work_type          5110 non-null   category
 7   Residence_type     5110 non-null   category
 8   avg_glucose_level  5110 non-null   float64 
 9   bmi                4909 non-null   float64 
 10  smoking_status     5110 non-null   category
 11  stroke             5110 non-null   int64   
dtypes: category(5), float64(3), int64(4)
memory usage: 306.1 KB
```

Altra ottimizzazione che possiamo fare è quella di rimuovere il campo ID perche inutile e rindondante:

In [10]:

```
dataset.drop('id', axis=1, inplace=True)
dataset.head()
```

Out[10]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_stat
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	forme smok

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_stat
1	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smok
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smok
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smok
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smok

Analizziamo ora la presenza di valori nulli all'interno del dataset:

In [11]:

```
dataset.isna().sum()
```

Out[11]:

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

Sono presenti diversi valori nulli ma sono limitati alla sola colonna `BMI`.

Valutando la composizione dei valori nulli e sapendo che la quantità di dati è abbastanza limitata, si prende la decisione di utilizzare la media nei valori di BMI sconosciuti; al posto di eliminare le righe interessate e le colonne.

Questo anche perche la media sarebbe 28.893237 e la mediana 28.100000 dati molto simili che ci fa capire anche grazie alla std di 7.854067 che i valori non si discostano troppo.

In [12]:

```
dataset["bmi"] = dataset["bmi"].fillna(dataset["bmi"].mean())
dataset.isna().sum()
```

Out[12]:

```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

In questo modo si va certamente a modificare la precisione in maniera negativa ma si conservano i dati. (NDR: come vedremo in seguito cio non sarà un problema perche i dati verranno scartati durante l'analisi)

Osservando inoltre `gender` si nota che oltre le due scelte maschio e femmina è presente la categoria `other`.

In [13]:

```
dataset["gender"].value_counts()
```

Out[13]:

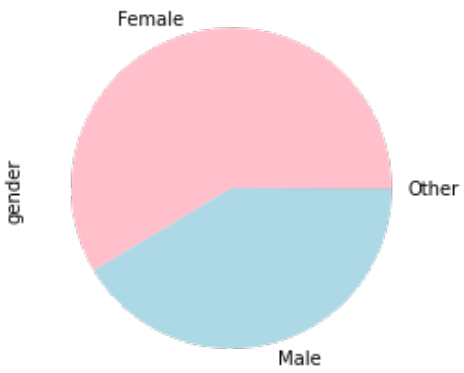
```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

In [14]:

```
dataset["gender"].value_counts().plot.pie(colors=[ "pink","lightblue","red"])
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83639ad350>
```



Come vediamo la percentuale di Other è bassisima rispetto alle altre scelte e essa risulterebbe solo uno spreco di spazio e prestazioni quando trasmormata per il training quindi eliminiamo la riga in cui è presente

In [15]:

```
dataset =dataset.drop(dataset[dataset["gender"]=="Other"].index.values)
dataset["gender"].value_counts()
```

Out[15]:

```
Female    2994
Male      2115
Other        0
Name: gender, dtype: int64
```

Quindi trasformo la variabile maschio e femmina in interi 1 o 0

In [16]:

```
dataset["gender"] = [int(elem) for elem in (dataset["gender"]=="Male")]
```

é però inoltre importante guardare la presenza di valori sconosciuti (paragonabili a valori nulli) in `smoking_status` (non visualizzabili con `isna()`)

Questi rappresentano una grande quantità:

In [17]:

```
(dataset["smoking_status"]=="Unknown").sum()
```

Out[17]:

```
1544
```

Togliamo dalle possibilità l'eliminazione delle righe perche graverebbe molto sul quantitativo dei dati. é ipotizzabile la rimozione della colonna ma da ricerche su internet e dai pattern che si evideniano sui grafici i fumatori sembrano avere una probabilità maggiore di avere un ictus.

Vista la scarsità dei dati e la motivazione espost sopra si decide di mantenerla la feature anche con se il dato è

Visto la scarsità dei dati e le motivazione esposte sopra si decide di mantenerle la feature anche con se il dato è parziale.

Grazie all'enconder utilizzato il rischio che crei bias è minore e circoscribile guardando i parametri

Sono stati analizzati anche gli altri dati (con particolare attenzione agli estremi) che sembrano rientrare tutti in una ottica di buon senso. Gli unici dubbi derivano dagli estremi di BMI che risultano un po particolari

Data Explorarion

Analizziamo ora possibili pattern visibili e identificabili

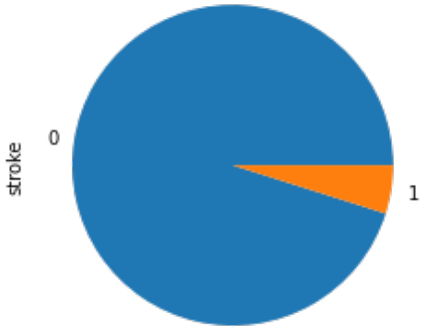
La prma cosa importante che osserviamo è che i dati sono altamente sbilanciati rispetto la variabile da predire.

In [18]:

```
dataset["stroke"].value_counts().plot.pie()
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f836394f210>
```



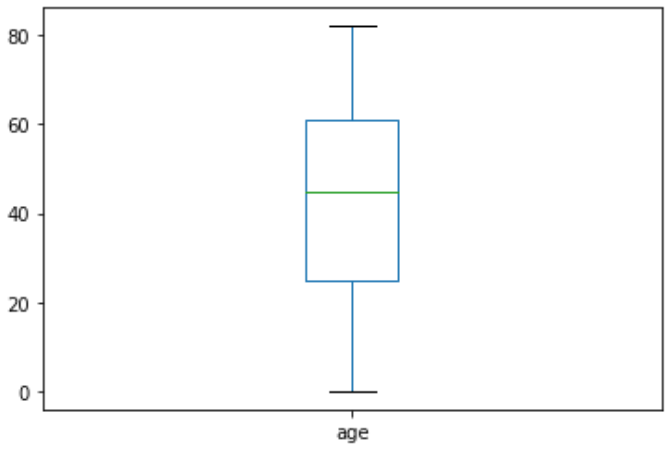
Come si vede i paziendi analizzzati che hanno avuto un ictus sono inferiori al 5% e questo creera problematiche di fitting nella creazione del modello

In [19]:

```
dataset["age"].plot.box()
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8363460110>
```



Notiamo inoltre che l'eta è abbastanza distribuita e che rispecchia con appprossimazione l'età di uno stato

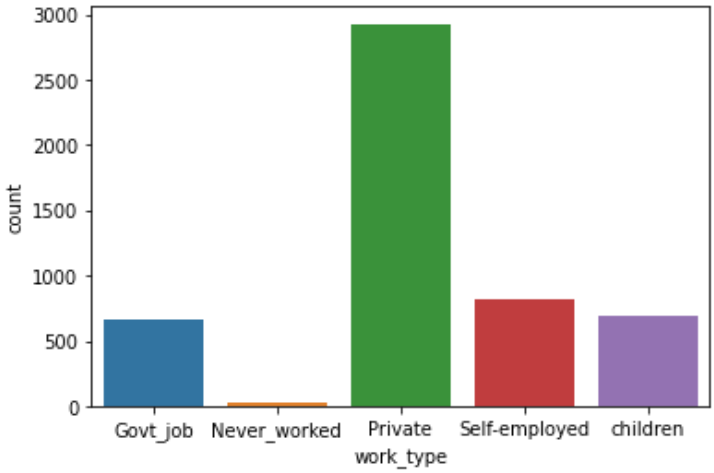
In [20]:

```
splot.countplot(dataset['work type'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8363447a10>
```



Facendo inoltre un semplice controllo sui valori i dati sembrano avere senso rispetto aall'eta delle persene.

Si noti l'alta quantità di persone che lavorano nell'ambito privato che non rispecchia alcuna media statele ma può comunque essere credibile

In [21]:

```
print((dataset["age"] < 20).value_counts())
print((dataset["work_type"] == "children").value_counts())
```

```
False    4143
True      966
Name: age, dtype: int64
False    4422
True      687
Name: work_type, dtype: int64
```

Si utilizza poi tramite la libreria di Seaborn una mappa di calore per evidenziare gia diverse possibili correlazioni.

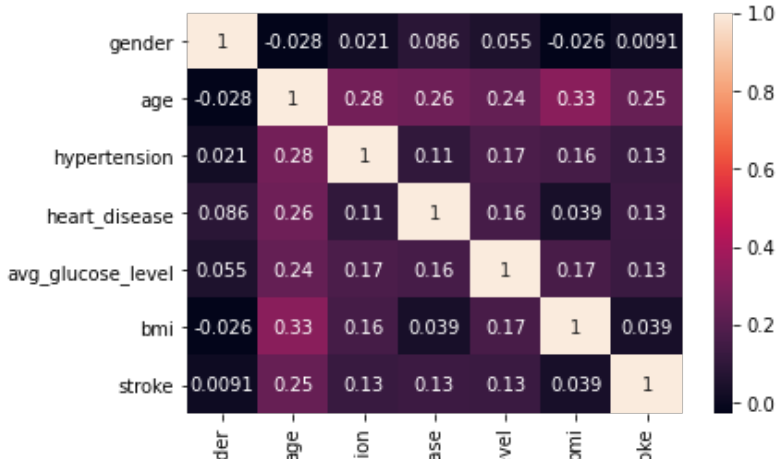
In questo modo sono però solo visibili delle correlazioni nelle variabili numeriche escludendo le altre.

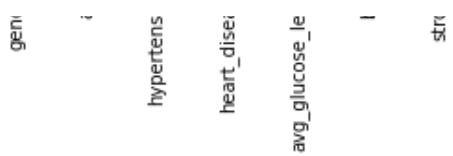
In [22]:

```
splot.heatmap(dataset.corr(method='pearson'), annot=True)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83633d9310>
```





Non sono presenti correlazione dirette ben identificabili, ma si puo subito identicifare il fatto che i casi di ictus sono relazionati almeno in parte all'età.

In particolare la colonna dell'età sembra essere un buon mappatore rispetto alle altre features e questo ci sembra veritiero grazie alle ricerche scientifiche gia presenti. (le persone anziane hanno piu probabilità di avere questo tipo di disturbi)

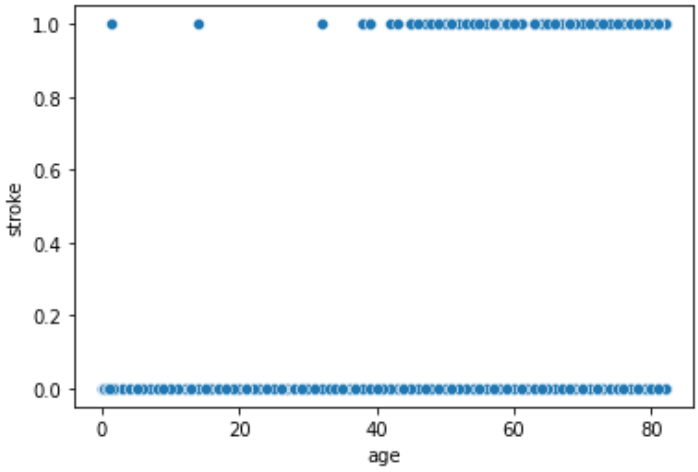
Per questo motivo approfondiamo con un grafico fra età e stroke

In [23]:

```
splot.scatterplot(x=dataset['age'], y=dataset['stroke'])
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f835aa99590>
```

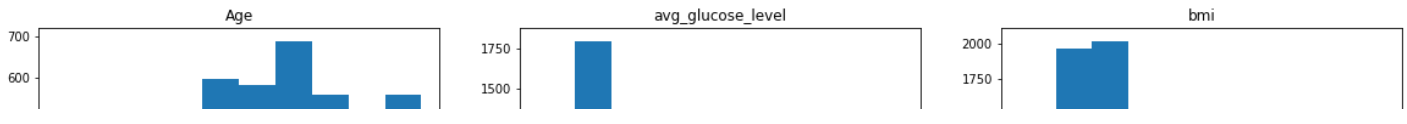


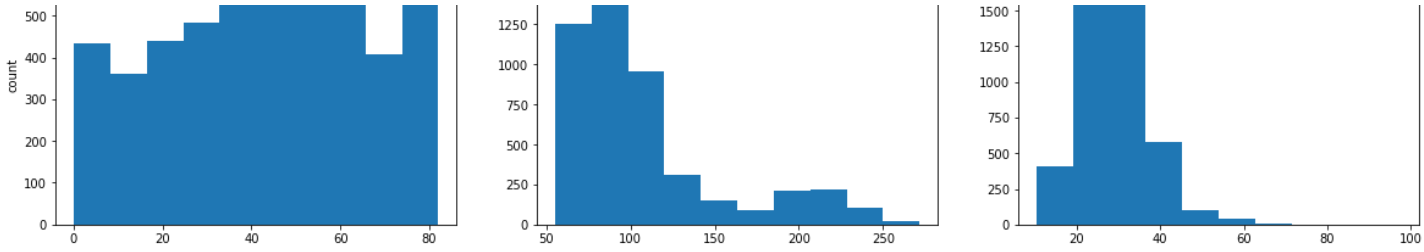
Vediamo subito che la maggior parte dei casi positivi sono concentrati tutti dai 40 agli 80 anni e che quindi l'età è effettivamente un indicatore

Visualizziamo ora gli altri valori per osservare se ci è sfuggito qualcosa

In [24]:

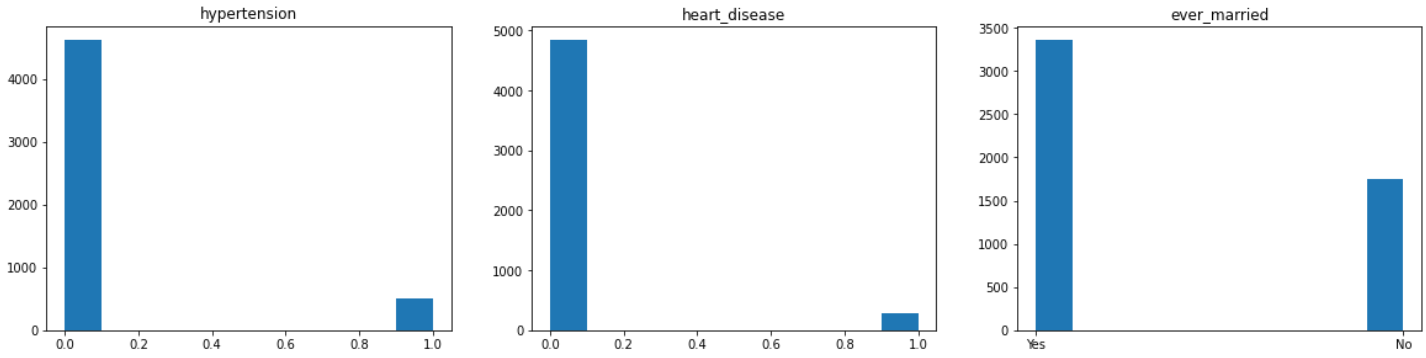
```
plt.figure(figsize=(20, 10))

plt.subplot(2, 3, 1)
plt.title('Age')
plt.hist(dataset['age'])
plt.ylabel('count')


plt.subplot(2, 3, 2)
plt.title('avg_glucose_level')
plt.hist(dataset['avg_glucose_level'])


plt.subplot(2, 3, 3)
plt.title('bmi')
plt.hist(dataset['bmi'])


plt.show()
```





In [25]:

```
plt.figure(figsize=(20, 10))

plt.subplot(2, 3, 1)
plt.title('hypertension')
plt.hist(dataset['hypertension'])

plt.subplot(2, 3, 2)
plt.title('heart_disease')
plt.hist(dataset['heart_disease'])

plt.subplot(2, 3, 3)
plt.title('ever_married')
plt.hist(dataset['ever_married'])

plt.show()
```



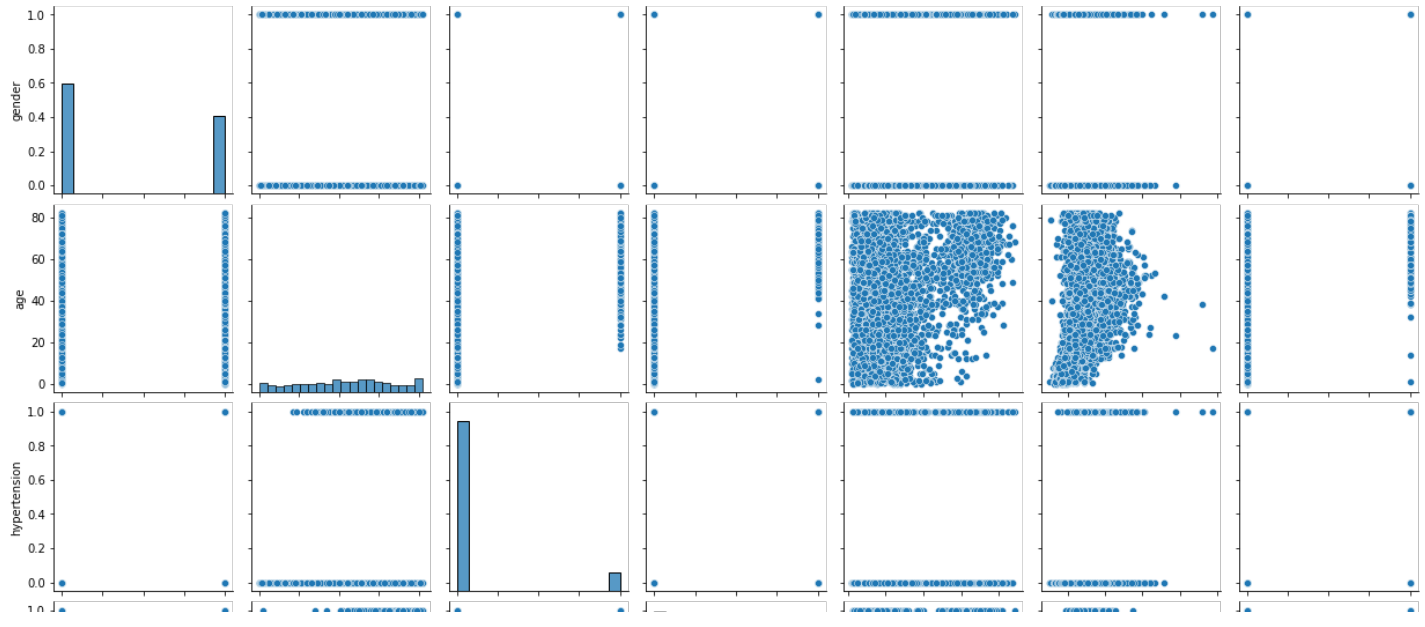
é possibile vedere correlazioni e l'insieme dei grafici anche travverso pairplot che può aiutare a comprendere il dataset. In realtà diversi grafici risultanto poco chiari in questo modo essendo molte variabili binare (0 o 1) come risultato

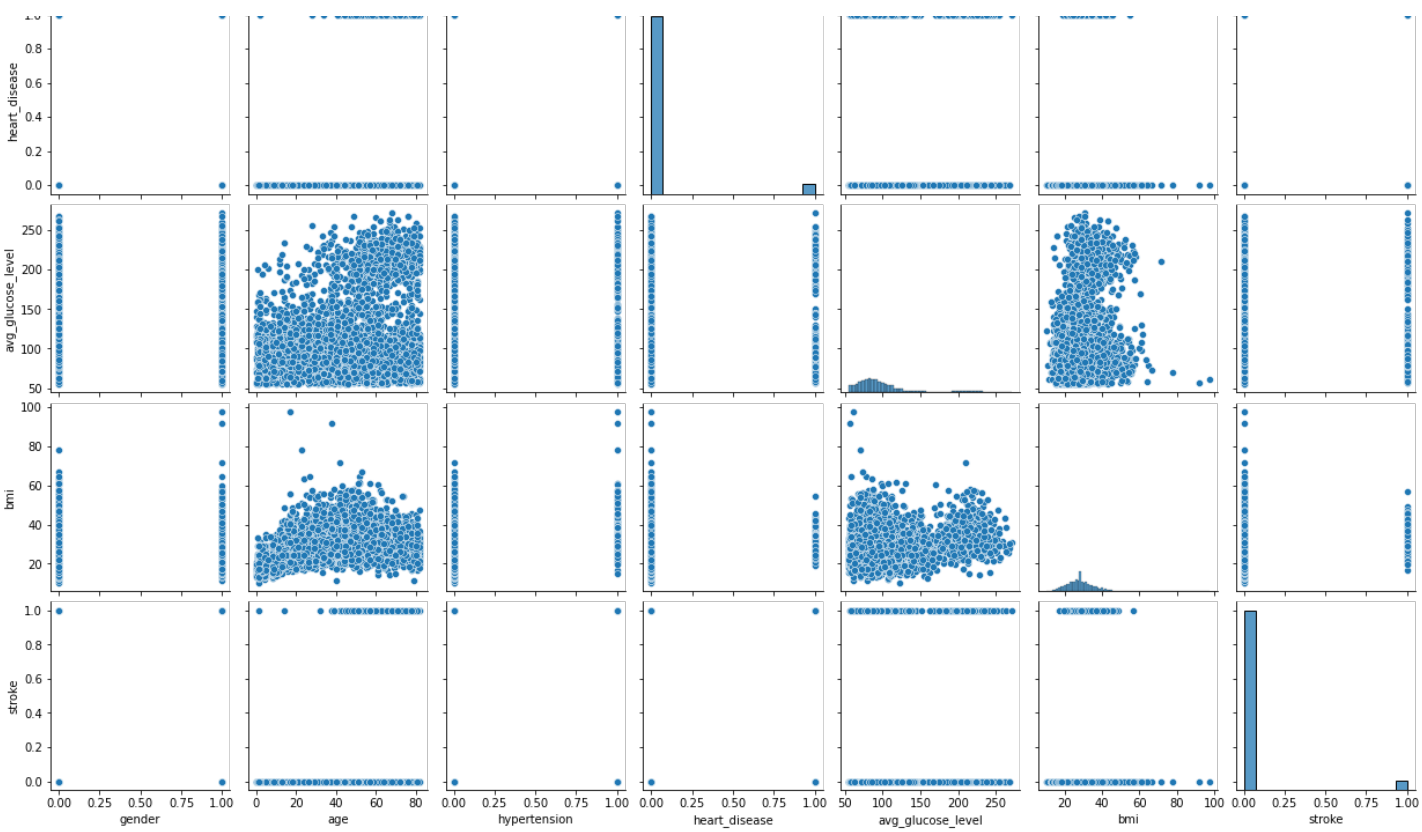
In [26]:

```
splot.pairplot(dataset,)
```

Out[26]:

```
<seaborn.axisgrid.PairGrid at 0x7f835a99e4d0>
```





Osservando il grafico dell'BMI si notano valori molto fuori dalla media della popolazione e alcuni assurdamente alti.

Facendo una ulteriore ricerca su internet i valori anormali sono sotto i 20 e sopra i 25 e sono inesistenti valori oltre i 35.

Qui la pagina wikepedia a riguardo

Si sceglie quindi di eliminare totalmente la colonna, anche se si ipotizzava potesse essere correlata, perchè reputata con informazioni non adeguate e con errori di misurazione

In [27]:

```
dataset.drop("bmi" , axis=1)
```

Out[27]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	smoking_status
0	1	67.0	0	1	Yes	Private	Urban	228.69	formerly smoked
1	0	61.0	0	0	Yes	Self-employed	Rural	202.21	never smoked
2	1	80.0	0	1	Yes	Private	Rural	105.92	never smoked
3	0	49.0	0	0	Yes	Private	Urban	171.23	smokes
4	0	79.0	1	0	Yes	Self-employed	Rural	174.12	never smoked
...	...	...	...	...	...	...	...	...	...
5105	0	80.0	1	0	Yes	Private	Urban	83.75	never smoked
5106	0	81.0	0	0	Yes	Self-employed	Urban	125.20	never smoked
5107	0	35.0	0	0	Yes	Self-employed	Rural	82.99	never smoked
5108	1	51.0	0	0	Yes	Private	Rural	166.29	formerly smoked
5109	0	44.0	0	0	Yes	Govt_job	Urban	85.28	Unknown

5109 rows × 10 columns

Features engineering e predisposizone dei dati

Dividiamo i dati: in primo luogo isoliamo la variabile da prevedere e poi tramite lo slit creiamo due dataset per fare il training e per fare il test

In [28]:

```
from sklearn.model_selection import train_test_split

X = dataset.drop("stroke" , axis=1)
y = dataset["stroke"]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42,s
huffle=True)
```

il dataset comprende molte variabili categoriche (fortunatamente con poche categorie).

La nostra strategia consiste nel convertire le variabili categoriche in numeriche tramite il one-hot encoding.

Per prima cosa dividiamo le variabili categoriche da quelle numeriche.

Prima prendiamo le categoriche e creiamo i nuovi dataset

In [29]:

```
categorical_vars = ["ever_married","work_type","Residence_type","smoking_status"]

X_train_cat = X_train[categorical_vars]
X_test_cat = X_test[categorical_vars]
```

Poi facciamo lo stesso con quelle numeriche con valori

In [30]:

```
numerical_vars = ["gender","age", "hypertension", "heart_disease", "avg_glucose_level"]
X_train_num = X_train[numerical_vars]
X_test_num = X_test[numerical_vars]
```

Creiamo poi l'encoder e e lo scaler che andremo ad utilizzare

Qui utilizziamo `drop="first"` in modo da eliminare una possibilità e quindi diminuire la grandezza. Questo è buono perche alcune variabili categoriche hanno poche categorie. In questo diminuiamo la dimensione della tabella di piu di 10 000 valori a discapito però di una perdita di precisone dovuto a bias introdotti

In [31]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
encoder = OneHotEncoder(drop="first", sparse=False)
```

Infine standardizziamo i dati numerici (da notare che la maggior parte delle variabili non ha bisogno di essere standardizzata essendo gia binaria) e trasformiamo i categorici

In [32]:

```
X_train = np.c_[
    scaler.fit_transform(X_train_num),
    encoder.fit_transform(X_train_cat)
]

X_test = np.c_[
```

```
    scaler.transform(X_test_num),
    encoder.transform(X_test_cat)
]

X_train.size
```

Out[32]:

```
50064
```

Model observation

Ci accertiamo che i dati siano ben composti facendo un rapido test con la regressione Perceptron

In [33]:

```
from sklearn.linear_model import Perceptron

model_lin_default = Perceptron()
model_lin_default.fit(X_train,y_train)

model_lin_default.score(X_test,y_test)
```

Out[33]:

```
0.9419439008480104
```

Provando ora con una logistic regression semplice ci accorgiamo che è uno strumento migliore. Inoltre valutiamo velocemente che i dati sono stati manipolati in modo adeguato e che le tempistiche di addestramento (accesso ai dati) sono abbastanza rapide visto il volume

In [32]:

```
from sklearn.linear_model import LogisticRegression

model_log_default = LogisticRegression()
model_log_default.fit(X_train,y_train)

model_log_default.score(X_test,y_test)
```

Out[32]:

```
0.9425962165688193
```

In [33]:

```
model_log_default.coef_
```

Out[33]:

```
array([[-0.02908158,  1.67808358,  0.09831342,  0.07317024,  0.18755317,
        -0.37232736, -0.04515981,  0.10589277, -0.33596934,  0.66831858,
         0.12355926, -0.14182257, -0.16884594,  0.18084291]])
```

Etichettiamo i vari coefficenti

In [34]:

```
pd.Series(
    model_log_default.coef_[0][:5],
    index=numerical_vars
)
```

Out[34]:

```
gender              -0.029082
age                  1.678084
hypertension         0.098313
heart_disease        0.073170
avg_glucose_level    0.187553
```

```
dtype: float64
```

In [35]:

```
pd.Series(
    model_log_default.coef_[0][5:],
    index=encoder.get_feature_names(X_train_cat.columns)
)
```

Out[35]:

```
ever_married_Yes                 -0.372327
work_type_Never_worked           -0.045160
work_type_Private                 0.105893
work_type_Self-employed          -0.335969
work_type_children                0.668319
Residence_type_Urban              0.123559
smoking_status_formerly smoked   -0.141823
smoking_status_never smoked      -0.168846
smoking_status_smokes             0.180843
dtype: float64
```

Vediamo che il secondo valore che corrisponde all'età ha alta correlazione come mostrava l'heatmap

é inoltre interessante vedere che lo score è gia molto alto e il modello creato potrebbe sembrare ben addestrato. In realtà con po' di attenzione e grazie all'analisi dati fatta prima, si potrebbe supporre, che il modello anche se ha delle buone metriche esse non sono sufficenti. Per testare cio lo si compara velocemente a due "modelli" tutti negativi o positivi.

In [36]:

```
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix

def model_information(y, y_pred) :

    print("Precision:  " + str(precision_score(y, y_pred, pos_label=1)) )
    print("Recall:     " + str(recall_score(y, y_pred)) )
    print("F1_Score:   " + str(f1_score(y, y_pred, average="macro")) )
    print("Differenza semplice")
    print((y==y_pred).value_counts())
```

In [37]:

```
model_information(y_test, model_log_default.predict(X_test))
```

```
Precision:  1.0
Recall:     0.011235955056179775
F1_Score:   0.496326164874552
Differenza semplice
True     1445
False      88
Name: stroke, dtype: int64
```

In [38]:

```
print("\n random model all 1")
model_information(y_test,np.ones(y_test.size))

print("\n random model all 0")
model_information(y_test,np.zeros(y_test.size))
```

```
 random model all 1
Precision:  0.05805609915198956
Recall:     1.0
F1_Score:   0.05487053020961775
Differenza semplice
False    1444
True       89
Name: stroke, dtype: int64

 random model all 0
```

```
Precision:  0.0
Recall:     0.0
F1_Score:   0.485052065838092
Differenza semplice
True     1444
False      89
Name: stroke, dtype: int64
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefined
MetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Si nota subito che le metriche sono decisamente piu basse nei array completamente omogeni. Pero in quello composto da tutti zero si è riusciti a predirre esattamente un risultato in meno rispetto al logisticRegression. Questo perche i dati sono altamente sbilanciati.

Andiamo ora a fare una analisi dei modelli incentrata ad vere uno score maggiore.

In [39]:

```
from sklearn.model_selection import StratifiedKFold

k_fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Per prima cosa utilizziamo la StratifiedKFold per avere insiemi piu consistenti e quindi migliori risultati.

In [40]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc
import plotly.express as px

from sklearn.metrics import f1_score

def KFold_gridSearch_model(model, grid, X_t, y_t):
    grid_model = GridSearchCV(model, grid, cv=k_fold, n_jobs=-1,)

    grid_model.fit(X_t, y_t)

    score = grid_model.score(X_test, y_test)

    print(f"Best score:     {grid_model.best_score_}\n")
    print(f"Best params:    {grid_model.best_params_}\n")
    print(f"Best estimator: {grid_model.best_estimator_}\n")
    print(f"Score in test set: {score}\n")
    print()
    y_pred = grid_model.predict(X_test)
    model_information(y_test, y_pred)
    print()
    splot.heatmap(confusion_matrix(y_test, y_pred), annot=True)


    #Per una spiegazione  leggere l'approfondimento sotto
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)

    fig = px.area(
        x=fpr, y=tpr,
        title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
        labels={'x':'False Positive Rate', 'y':'True Positive Rate'},
        width=700, height=500
    )
    fig.add_shape(
        type='line', line={'dash':'dash'},
        x0=0, x1=1, y0=0, y1=1
    )

    fig.update_yaxes(scaleanchor="x", scaleratio=1)
```

```
    fig.update_xaxes(constrain='domain')
    fig.show()

    print("*"*50)

    return grid_model
```

Approfondimento:

AUC - ROC Curve

Fra le informazioni che andiamo a osservare nel grafico una di queste è il AUC - ROC Curve graph (anche detto AUROC). Il grafico mostra AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) che sono indicatori che mostrano come il modello interpreta le classi da prevedere.

Una spiegazaione rapida ma ben comprensibile la si puo trovare qui
https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Logic Regression

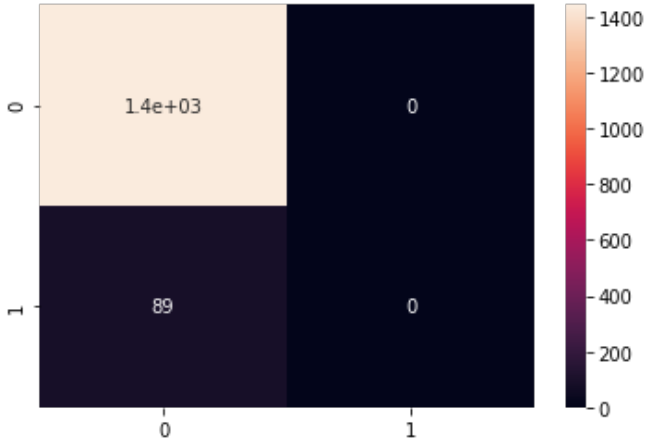
In [41]:

```
from sklearn.pipeline import Pipeline


lr_model = Pipeline([
    ("lr", LogisticRegression(n_jobs=-1, random_state=42))
])


lr_grid = {
    "lr__class_weight": [None,"balanced"],
    "lr__penalty": [None,"l1", "l2"],
    "lr__C": np.logspace(-5, 3, 8),
    "lr__fit_intercept": [False, True]
}


lr = KFold_gridSearch_model(lr_model, lr_grid, X_train, y_train)
```

```
Best score:     0.9552571866735521

Best params:    {'lr__C': 1e-05, 'lr__class_weight': None, 'lr__fit_intercept': True, 'lr
__penalty': 'l2'}

Best estimator: Pipeline(memory=None,
         steps=[('lr',
                 LogisticRegression(C=1e-05, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=100,
                                    multi_class='auto', n_jobs=-1, penalty='l2',
                                    random_state=42, solver='lbfgs', tol=0.0001,
                                    verbose=0, warm_start=False))],
         verbose=False)

Score in test set: 0.9419439008480104


Precision:  0.0
Recall:     0.0
F1_Score:   0.485052065838092
Differenza semplice
True     1444
False      89
Name: stroke, dtype: int64
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefined
MetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
```

****



Oversampling e bilanciamento dei dati

Come vediamo però anche lo score si alza fino 0,955 la matrice di confusione mostra e il grafico mostano che cio che in realtà fa il modello: prevede che ogni utente non abbia l'ictus e in questo modo crea un buon risultato.

Facendo una ricerca ho trovato che oltre l'oversampling delle persone con ictus o l'undersampling di persone senza in maniera manuale esistono metodi che danno maggiori risultati introducendo meno bias.

In [42]:

```
from imblearn.over_sampling import SMOTE
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning:
```

```
The module is deprecated in version 0.21 and will be removed in version 0.23 since we've
dropped support for Python 2.7. Please rely on the official version of six (https://pypi.
org/project/six/).

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:

The sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed in v
ersion 0.24. The corresponding classes / functions should instead be imported from sklear
n.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the p
rivate API.
```

SMOTE - che sta per Synthetic Minority Oversampling Technique è una tecnica di oversampling dove vengono generati dati sintetici della classe piu piccola. L'algoritmo aiuta a evitare problemi di overfitting creati da un oversampling causale

Qui un link per capire meglio: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

In [43]:

```
print("Prima dell' OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Prima dell' OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))

sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

print('Dopo aver fatto l OverSampling, the shape of train_X: {}'.format(X_train_res.shape
))
print('Dopo aver fatto l OverSampling, the shape of train_y: {} \n'.format(y_train_res.sh
ape))

print("Dopo aver fatto l OverSampling, counts of label '1': {}".format(sum(y_train_res==1
)))
print("Dopo aver fatto l OverSampling, counts of label '0': {}".format(sum(y_train_res==0
)))
```

```
Prima dell' OverSampling, counts of label '1': 160
Prima dell' OverSampling, counts of label '0': 3416

Dopo aver fatto l OverSampling, the shape of train_X: (6832, 14)
Dopo aver fatto l OverSampling, the shape of train_y: (6832,)

Dopo aver fatto l OverSampling, counts of label '1': 3416
Dopo aver fatto l OverSampling, counts of label '0': 3416
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function safe_indexing is deprecated; safe_indexing is deprecated in version 0.22 and wil
l be removed in version 0.24.
```

Ora proviamo ad utilizzare lo stesso modello sui nuovi dati. Ci aspettiamo in parte un peggioramento dello score dato dall'introduzione di dati sintetici in parte un miglioramento nella classificazione di persone con ictus dato dal numero maggiore di dati presenti di questo tipo

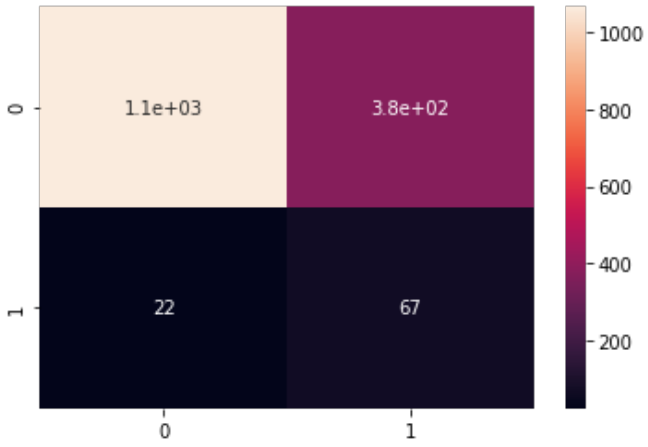
In [44]:

```
lr = KFold_gridSearch_model(lr_model, lr_grid, X_train_res, y_train_res)
```

```
Best score:     0.8062064935398526

Best params:    {'lr__C': 71.96856730011514, 'lr__class_weight': None, 'lr__fit_intercept
': False, 'lr__penalty': 'l2'}

Best estimator: Pipeline(memory=None,
         steps=[('lr',
                 LogisticRegression(C=71.96856730011514, class_weight=None,
                                    dual=False, fit_intercept=False
```

```
                                   dual=False, fit_intercept=False,
                                   intercept_scaling=1, l1_ratio=None,
                                   max_iter=100, multi_class='auto', n_jobs=-1,
                                   penalty='l2', random_state=42,
                                   solver='lbfgs', tol=0.0001, verbose=0,
                                   warm_start=False))],
         verbose=False)

Score in test set: 0.7397260273972602


Precision:  0.15090090090090091
Recall:     0.7528089887640449
F1_Score:   0.54694320152227
Differenza semplice
True     1134
False     399
Name: stroke, dtype: int64
```

****



Lo score è diminuito notevolmente (anche se gli altri parametri sono aumentati). A questo punto si prende una decisione soggettiva ovvero quello di utilizzare comunque i dati con l'Oversample. Questo perche rispecchiano meglio l'utilizzo effettivo che sarà fatto con i modelli creati. Riflettendoci è certamente importante avere una

buona precisione ma immaginiamo che questo software possa essere utilizzato per mettere in stato di allerta persone a rischio di ictus. Questo significa che è sicuramente meglio allertare delle persone in piu che magari non corrono il rischio che non allertare delle persone che invece lo corrono. Con questi dati lo stesso modello come si vede dai due grafici allerta molte piu persone e quindi ci da in fondo un margine di savataggio maggiore. Cerchiamo ora di migliorarlo.

SVC

SVC (Support Vector Machine) determina l'iperpiano di separazione ottimale trasformando lo spazio dei dati in modo che le classi diventino separabili linearmente,

I parametre delle Grid

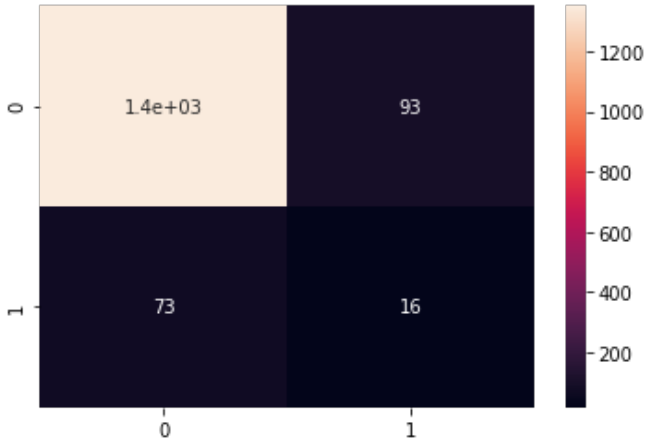
- **C controlla l'overfitting, più il valore è alto piu si restringe il margine riducendo il numero di errori di training.**
- **Il kernel usato è il RBF (radial basis function) ed è una delle piu comune usati con questo modello. Qui la formula e alcune informazioni aggiutive https://en.wikipedia.org/wiki/Radial_basis_function_kernel**

In [45]:

```
from sklearn.svm import SVC

svm_model = Pipeline([
    ("svc", SVC(random_state=42,))
])


svm_grid = {
    "svc__class_weight": [None, "balanced"],
    'svc__C': np.logspace(1, 5, 5),
    'svc__gamma': ['scale'],
    'svc__kernel': ['rbf']

    },


svc = KFold_gridSearch_model(svm_model, svm_grid, X_train_res, y_train_res)
```

```
Best score:     0.9524267293416557

Best params:    {'svc__C': 10000.0, 'svc__class_weight': None, 'svc__gamma': 'scale', 'sv
c__kernel': 'rbf'}

Best estimator: Pipeline(memory=None,
         steps=[('svc',
                 SVC(C=10000.0, break_ties=False, cache_size=200,
                     class_weight=None, coef0=0.0,
                     decision_function_shape='ovr', degree=3, gamma='scale',
                     kernel='rbf', max_iter=-1, probability=False,
                     random_state=42, shrinking=True, tol=0.001,
                     verbose=False))],
         verbose=False)

Score in test set: 0.8917155903457273


Precision:  0.14678899082568808
Recall:     0.1797752808988764
F1_Score:   0.551868052914078
Differenza semplice
True     1367
False     166
Name: stroke, dtype: int64
```

****



KNeighborsClassifier

Un modello che attribuisce la classe target, a seconda delle delle feature "vicine" spazialmente a quella da determinare andando quindi ad osservere k vicini (che sono presubilmente casi simili)

I parametre delle Grid

- **n_neighbors: Il numero dei vicini che verranno considerati per l'osservazione**
- **weights: Il tipo di metrica che viene utilizzata per valutare i vicini**

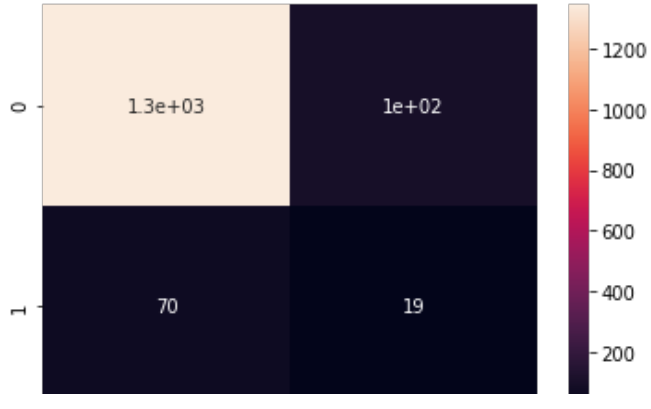
In [46]:

```
from sklearn.neighbors import KNeighborsClassifier

knc_model = Pipeline([
    ("knc", KNeighborsClassifier(n_jobs=-1))
])



knc_grid = {
        'knc__n_neighbors': range(1, 10),
        'knc__weights': ['uniform', 'distance']
            }
```

```
knc = KFold_gridSearch_model(knc_model, knc_grid, X_train_res, y_train_res)
```

```
Best score:     0.953746157732056

Best params:    {'knc__n_neighbors': 1, 'knc__weights': 'uniform'}

Best estimator: Pipeline(memory=None,
         steps=[('knc',
                 KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                      metric='minkowski', metric_params=None,
                                      n_jobs=-1, n_neighbors=1, p=2,
                                      weights='uniform'))],
         verbose=False)

Score in test set: 0.8884540117416829


Precision:  0.15833333333333333
Recall:     0.21348314606741572
F1_Score:   0.5609825945842747
Differenza semplice
True     1362
False     171
Name: stroke, dtype: int64
```

****



DecisionTreeClassifier

Gli alberi decisionali costituiscono un approccio differente ai modelli visti precedentemente infatti la classificazione avviene in base ad una serie di decisioni "semplici", basate ciascuna su una sola variabile.

Quindi ciascuna decisione porta ad un ramo diverso dell'albero.

I parametre delle Grid

- **Split, ovvero quanto viene diviso**
- **Leaf, ovvero il numero delle foglie**
- **Depth, Ovvero la profondità massima che l'albero puo avere (L'opzione None toglie il limite)**

In [47]:

```
encoder.get_feature_names(X_train_cat.columns)
numerical_vars
name_vars = np.append(numerical_vars, np.asarray(encoder.get_feature_names(X_train_cat.c
olumns)))
name_vars
```

Out[47]:

```
array(['gender', 'age', 'hypertension', 'heart_disease',
       'avg_glucose_level', 'ever_married_Yes', 'work_type_Never_worked',
       'work_type_Private', 'work_type_Self-employed',
       'work_type_children', 'Residence_type_Urban',
       'smoking_status_formerly smoked', 'smoking_status_never smoked',
       'smoking_status_smokes'], dtype=object)
```

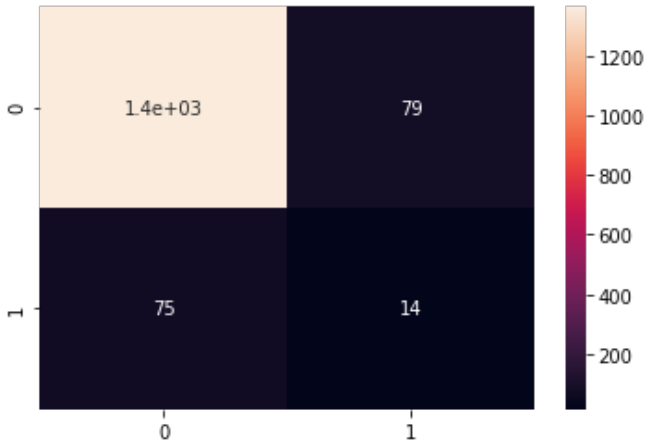
In [48]:

```
from sklearn.tree import DecisionTreeClassifier

tree_model = Pipeline([
    ("tree", DecisionTreeClassifier(random_state=42))
])

#print(tree_model.get_params())

tree_grid = {
            "tree__class_weight": [None, "balanced"],
             'tree__min_samples_split': range(2, 5),
             'tree__min_samples_leaf': range(1, 5),
             'tree__max_depth': [None] + [i for i in range(2, 7)],
             }

tree = KFold_gridSearch_model(tree_model, tree_grid, X_train_res, y_train_res)
```

```
Best score:     0.9385215723545075

Best params:    {'tree__class_weight': None, 'tree__max_depth': None, 'tree__min_samples_
leaf': 1, 'tree__min_samples_split': 3}

Best estimator: Pipeline(memory=None,
         steps=[('tree',
                 DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                        criterion='gini', max_depth=None,
                                        max_features=None, max_leaf_nodes=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=3,
                                        min_weight_fraction_leaf=0.0,
                                        presort='deprecated', random_state=42,
                                        splitter='best'))],
         verbose=False)
```

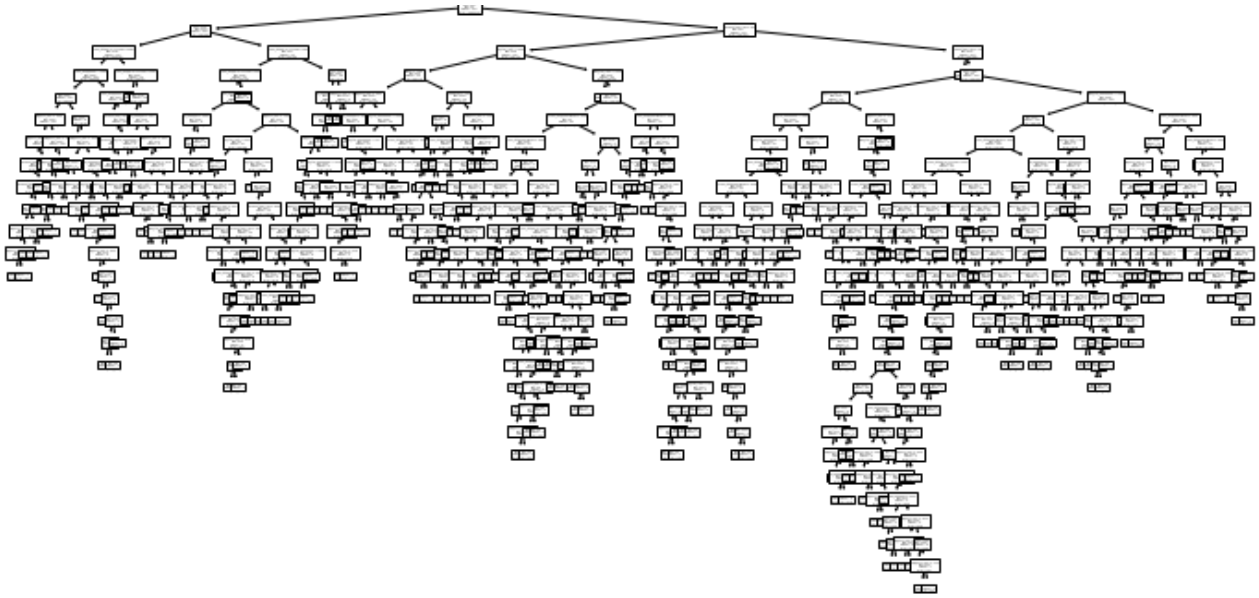
```
Score in test set: 0.8995433789954338


Precision:  0.15053763440860216
Recall:     0.15730337078651685
F1_Score:   0.5502240477968633
Differenza semplice
True     1379
False     154
Name: stroke, dtype: int64
```

****



In [49]:

```
from sklearn.tree import plot_tree

best_model_to_plot = DecisionTreeClassifier(random_state=42, max_depth= None, min_sample
s_split= 3)
best_model_to_plot.fit(X_train_res, y_train_res)

plt.figure(figsize=(12, 6))
plot_tree(best_model_to_plot, feature_names= name_vars);
```



Come vediamo l'albero creato è enorme peche la max depth (parametro nel migliore risultato della grid) è impostato a None e quindi l'albero si estende per quanto gli è possibile

RandomForestClassifier

Il funzionamento è simile al `DecisionTreeClassifier` ma vengono generati piu alberi (da qui il nome) e nel momento della decisione di quale "strada" prendere vengono interpellati tutti gli alberi competenti per poi trovare optare per una soluzione media. In questo modo questo modello anche se piu pesante del precente risulta migliore nelle problematiche di overfitting.

I parametre delle Grid

- **N_estimators, ovvero i numeri di alberi che verranno interpellati**
- **Split, ovvero quanto viene diviso**
- **Depth, Ovvero la profondità massima che l'albero puo avere (L'opzione None toglie il limite)**

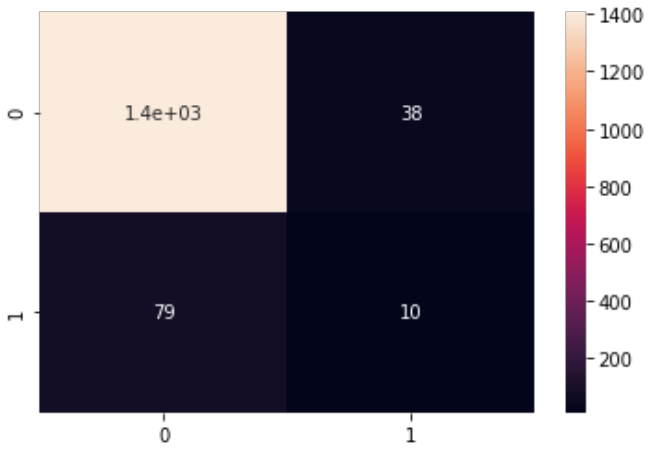
In [50]:

```
from sklearn.ensemble import RandomForestClassifier

forest_model = Pipeline([
    ("forest", RandomForestClassifier(n_jobs=-1, random_state=42))
])



forest_grid = {
              "forest__class_weight": [None, "balanced"],
             'forest__n_estimators': range(1, 10),
             'forest__min_samples_split': range(1, 5),
             'forest__max_depth': [None] + [i for i in range(1, 5)],
             }

forest = KFold_gridSearch_model(forest_model, forest_grid, X_train_res, y_train_res)
```

```
Best score:     0.9603313126642863

Best params:    {'forest__class_weight': 'balanced', 'forest__max_depth': None, 'forest__
min_samples_split': 2, 'forest__n_estimators': 8}

Best estimator: Pipeline(memory=None,
         steps=[('forest',
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight='balanced',
                                        criterion='gini', max_depth=None,
                                        max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
```

```
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1, min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=8, n_jobs=-1,
                                       oob_score=False, random_state=42,
                                       verbose=0, warm_start=False))],
         verbose=False)

Score in test set: 0.923679060665362


Precision:  0.20833333333333334
Recall:     0.11235955056179775
F1_Score:   0.5530200138060622
Differenza semplice
True     1416
False     117
Name: stroke, dtype: int64
```

```
**************************************************
```



Multi-layer perceptron

- **Il *multi-layer perceptron* è una *rete neurale* molto semplice che basa il suo modello di apprendimento sulla combinazione di molteplici nodi (ogni nodo non è altro che una regressione lineare con una funzione di attivazione)**

Non è un modello particolarmente appropriato perche solitamente usatto per il processamento di dati non strutturati, come immagini e testi.

I parametre delle Grid

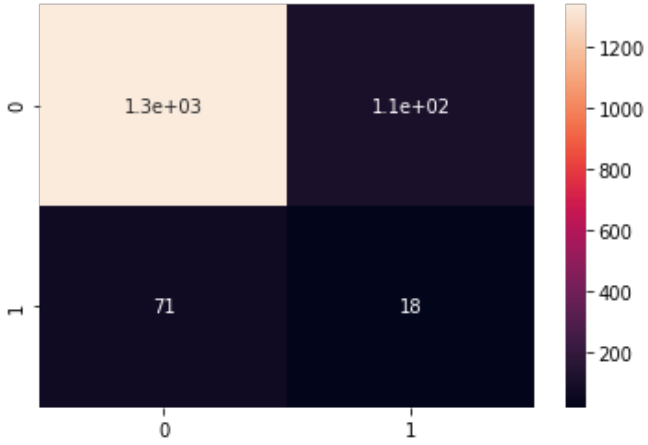
- **Activation: la funzione di attivazione (es. Relu => return np.maximum(0, x))**
- **hidden_layer_sizes: Il numero di layer e quanti nodi hanno**
- **Batch_size: la grandezza dei batch che vengono usati, ovvero i gruppi casuali osservati**

In [51]:

```
from sklearn.neural_network import MLPClassifier

mlpc_model = Pipeline([
    ("mlpc", MLPClassifier(max_iter=1000))
])

mlpc_grid = {
    "mlpc__activation": ["relu", "identity"],
    "mlpc__hidden_layer_sizes": [16, 32, (16, 8), (16, 32, 16)],
    "mlpc__batch_size": [100, 200],

}

mlpc = KFold_gridSearch_model(mlpc_model, mlpc_grid, X_train_res, y_train_res)
```

```
Best score:     0.9418862859931675

Best params:    {'mlpc__activation': 'relu', 'mlpc__batch_size': 100, 'mlpc__hidden_layer
_sizes': (16, 32, 16)}

Best estimator: Pipeline(memory=None,
         steps=[('mlpc',
                 MLPClassifier(activation='relu', alpha=0.0001, batch_size=100,
                               beta_1=0.9, beta_2=0.999, early_stopping=False,
                               epsilon=1e-08, hidden_layer_sizes=(16, 32, 16),
                               learning_rate='constant',
                               learning_rate_init=0.001, max_fun=15000,
                               max_iter=1000, momentum=0.9, n_iter_no_change=10,
                               nesterovs_momentum=True, power_t=0.5,
                               random_state=None, shuffle=True, solver='adam',
                               tol=0.0001, validation_fraction=0.1,
                               verbose=False, warm_start=False))],
         verbose=False)

Score in test set: 0.8838878016960209


Precision:  0.144
Recall:     0.20224719101123595
F1_Score:   0.5529059784247159
Differenza semplice
True     1355
False     178
Name: stroke, dtype: int64
```

****



Confronto fra i vari modelli

Tutti i modelli addestrati hanno dato buoni risultati, che indipendentemente dal modello e con l'uso dei dati con oversampling, si sono dimostrati coerenti fra loro. I valori degli score sono compresi fra i 0,93 e i 0,96 nelle versioni con grid search e migliori parametri sui dati di test. (Tranne la logisticRegression che ha avuto uno score di 0,8)

Il miglior risultato è stato raggiunto da RandomForestClassifier con uno score del 0.9158512720156555 tramite i parametri `'forest__max_depth': None, 'forest__min_samples_split': 3, 'forest__n_estimators': 9`

é però inoltre importante ribadire che anche altri valori sono molto importanti infatti obiettivo dato per questo software è quello di allarmare persone che potrebbero avere l'ictus, avendo quindi poca preoccupazione di allarmare persone che non corrono questo rischio eccessivamente.

Ora analizziamo un modello randomico per osservere le differenze con quelli addestrati da noi

In [52]:

```
from sklearn.dummy import DummyClassifier

random = DummyClassifier(strategy="uniform", random_state=42)
random.fit(X_train_res, y_train_res)

random_score = random.score(X_train, y_train)

print(f"Score: {random_score}");
print()
```

```
print(f"F1 score: {f1_score(y_test, random.predict(X_test))}");
print(f"Precision score: {precision_score(y_test, random.predict(X_test))}")
print(f"Recall score: {recall_score(y_test, random.predict(X_test))}")
```

```
Score: 0.4988814317673378

F1 score: 0.11515863689776733
Precision score: 0.06430446194225722
Recall score: 0.550561797752809
```

Ci accorgiamo che i risultati sono decisamente piu bassi e che i valori non si avvicinano ai modelli che abbiamo addestrato.

Problemi di Overfitting e sbilanciamento

Considerando l'analisi svolta il maggiore problema è stato quello di classificare nel modo adeguato le persone con ictus e non viceversa. Per affrontare il problema si è cercato tramite Smote di bilanciare i dati, anche se con un risultato non eccelso.

Una prima soluzione provata ma che ha dato scarsi risultati è stata quella di inserire `class_weight="balanced"` per un ribilanciamento da parte del modello.

Una soluzione potrebbe essere quella di trattare il problema non come un problema di classificazione ma di regressione per avere una variabile continua come risultato finale. Con essa si potrebbe impostare una approssimazione come ad esempio `if x > 0.3: x=1` per avere una maggiore sicurezza e un campione maggiore nelle persone da avvisare sbilanciando i modelli verso i risultati con ictus. O comunque comunicare al paziente la percentuale direttamente o attraverso un sistema di scala del pericolo.

In [53]:

```
models = [lr, svc, knc, tree, forest, mlpc]
name_model = ["logic regression", "Support Vector Classifier", "KNeighborsClassifier" ,
"DecisionTree Classifier", "RandomForestClassifier" , "Multi-layer perceptron"]
len(models)
```

Out[53]:

```
6
```

In [54]:

```
def overfitting_valutation(model_to_test, name_model):
  train = model_to_test.best_score_
  test = model_to_test.score(X_test, y_test)
  diff =  train - test
  diff_perc = 100 - test/train *100

  print("++++++++++++++++++++++++++++++++++++")
  print(name_model)
  print(f"Score sui dati del train:   {train}")
  print(f"Score sui dati del test:    {test}")
  print(f"differenza fra i due score: {diff}")
  print(f"differenza percentuale:     {diff_perc} %")
```

In [55]:

```
for i in range(0,len(models)):
  overfitting_valutation(models[i],name_model[i])
```

```
++++++++++++++++++++++++++++++++++++
logic regression
Score sui dati del train:   0.8062064935398526
Score sui dati del test:    0.7397260273972602
differenza fra i due score: 0.0664804661425924
differenza percentuale:     8.246084182563848 %
++++++++++++++++++++++++++++++++++++
Support Vector Classifier
Score sui dati del train:   0.9524267293416557
```

```
Score sui dati del test:    0.8917155903457273
differenza fra i due score: 0.06071113899592839
differenza percentuale:     6.374363205649814 %
++++++++++++++++++++++++++++++++++
KNeighborsClassifier
Score sui dati del train:   0.953746157732056
Score sui dati del test:    0.8884540117416829
differenza fra i due score: 0.06529214599037303
differenza percentuale:     6.84586202115176 %
++++++++++++++++++++++++++++++++++
DecisionTree Classifier
Score sui dati del train:   0.9385215723545075
Score sui dati del test:    0.8995433789954338
differenza fra i due score: 0.03897819335907371
differenza percentuale:     4.153148367307907 %
++++++++++++++++++++++++++++++++++
RandomForestClassifier
Score sui dati del train:   0.9603313126642863
Score sui dati del test:    0.923679060665362
differenza fra i due score: 0.0366522519989243
differenza percentuale:     3.81662573276283 %
++++++++++++++++++++++++++++++++++
Multi-layer perceptron
Score sui dati del train:   0.9418862859931675
Score sui dati del test:    0.8838878016960209
differenza fra i due score: 0.0579984842971466
differenza percentuale:     6.157694953164167 %
```

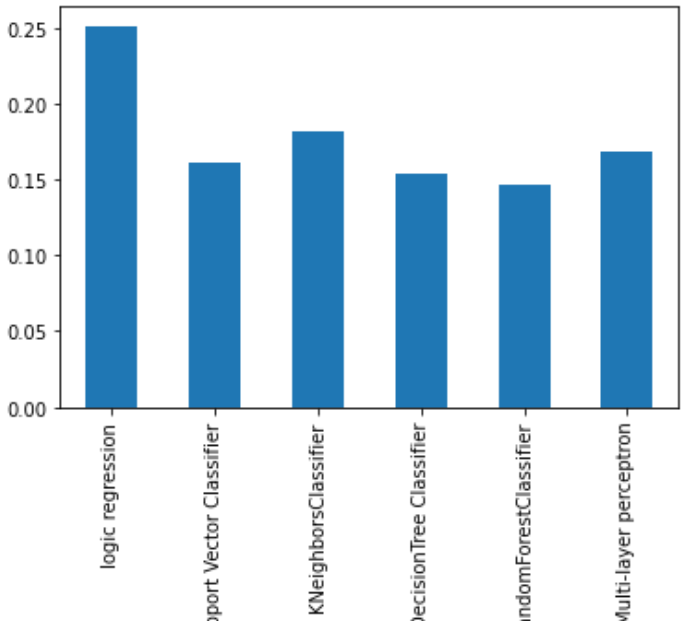
Come si nota da queste misurazioni in cui è stato confrontato il punteggio fra i dati di training e i dati di test alcuni modelli hanno altissimi punteggi sui dati di train e quindi in parte soffrono di overfitting. Si nota che comunque la differenza non è altissima e nei casi migliori la differenza è poco sopra al 4 % che risulta un valore accettabile

In [58]:

```
f1_scores = np.zeros(6)

for i in range(0,len(models)):
  f1_scores[i] = f1_score(y_test, models[i].predict(X_test))

f1_plt = pd.Series(
    f1_scores,
    index=name_model
)

f1_plt.plot.bar()
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f72cc0491d0>
```



Come si nota da questo secondo grafico l' F1 score di tutti i modelli è abbastanza basso con un leggero ma significativo valore maggiore nella logic regression. Una soluzione per aumentare questi valori sarebbe quella di utilizzare come parametro di riferimento all'interno tramite griglia `"F1"`. Soluzione alternativa sarebbe utilizzare una ricerca di griglia a piu fattori, scelta che non si è decisa di prendere perche non obiettivo principale dell'esercitazione.

Confronto con altri progetti su questo dataset

Su Kaggle il dataset che abbiamo utilizzato non è certo uno dei piu famosi ma ci sono state comunque diverse sottomissioni da parte di utenti con lo stesso obiettivo di prevedere lo stroke.

Osservando velocemente i loro lavori ci si accorge che i punteggi fatti in termini di score sono quasi equiparabili anche se leggermente piu alti (toccando poco sopra i 0,95)

Considerazioni Finali

Per l'addestramento dei modelli si è deciso di utilzzare tutti i dati non scartando colonne tranne quella riguardante BMI (e quella dell'ID). Inoltre non sono state eliminate righe se non una per eliminare il genere Other.

é stato preso questo approccio perche i dati considerati erano pochi e soprattutto dentro un problema complesso. Inoltre visto la piccola quantità dei dati è esigua, i modelli utilizzati non richiedono molto tempo di addestramento e concludono la fase di fit() in pochi secondi al massimo.

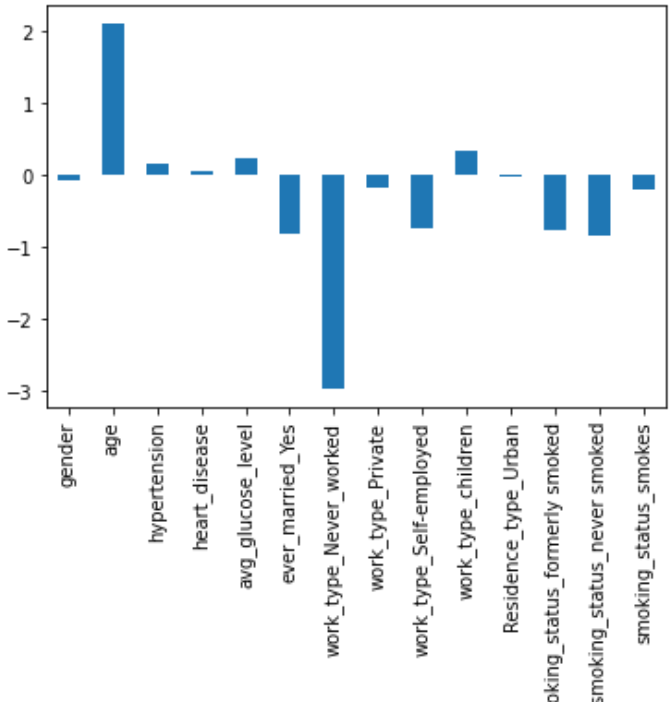
Per questo motivo si valuta che il KNeighborsClassifier sia il modello migliore anche se non il piu rapido (anche nel fit), non prendendo come metrica i tempi di eseguzione e l'hardware.

In [57]:

```
lr.best_estimator_.named_steps["lr"].coef_

coef = pd.Series(
    lr.best_estimator_.named_steps["lr"].coef_[0],
    index=name_vars
)

coef.plot.bar()
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f72d53a1ed0>
```



Riguardando i coefficenti del primo modello addestrato si riconferma che l'età ha un ruolo fondamentale nella decisione dei modelli. Le ricerche scientifiche confermano cio con numerosi studi.

é inoltre importante dire che ad accentuare questo parametro probabilmente è colpa della misurazione stessa: i pazienti con età bassa hanno avuto meno tempo per essere colpiti da un ictus e quindi meno probabilità.

Proprio per questo motivo non si può fare troppo affidamento ai modelli addestrati perche difficili da valutre essendo i dati per forza di cosa scorretti. Cio è dovuto in parte alla casualità della malattia ma anche al fatto le misurazioni non sono mai state aggiornate e quindi potrebbe capitare che una persona con una età avanzata, problemi cardiaci e fumatore al tempo dell'intervista non risulti aver avuto un ictus. Ma essere colpito dal problema pochi giorni dopo all'inserimento dei dati nel database.