In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import seaborn as sns
from plotly import __version__
from plotly.offline import download_plotlyjs,init_notebook_mode,iplot,plot
import cufflinks as cf
init_notebook_mode(connected=True)
cf.go_offline()
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
pd.options.display.float_format = '{:.2f}'.format
```

In [3]:

```
df_train = pd.read_csv('train_2v.csv')
df_test = pd.read_csv('test_2v.csv')
```

In [4]:

```
df_train.head(2)
```

Out[4]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	30669	Male	3.00	0	0	No	children	Rural	95.12	18.00	NaN	0
1	30468	Male	58.00	1	0	Yes	Private	Urban	87.96	39.20	never smoked	0

In [5]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43400 entries, 0 to 43399
Data columns (total 12 columns):
id                   43400 non-null int64
gender               43400 non-null object
age                  43400 non-null float64
hypertension         43400 non-null int64
heart_disease        43400 non-null int64
ever_married         43400 non-null object
work_type            43400 non-null object
Residence_type       43400 non-null object
avg_glucose_level    43400 non-null float64
bmi                  41938 non-null float64
smoking_status       30108 non-null object
stroke               43400 non-null int64
dtypes: float64(3), int64(4), object(5)
memory usage: 4.0+ MB
```

In [6]:

```
def findMissingValue(df):
    for fn in df.columns:
        targetNum = len(df)
        x= df[fn].describe()[0]
        if x !=targetNum:
            missingValue = targetNum-x
            percentOfMV = round(float((missingValue/targetNum)*100),2)
            print(fn + ' has missing value = '+str(missingValue)+' ('+str(percentOfMV)+'%)')
        else:
            print(fn+ ' = No Missing Value')
```

In [7]:

```
findMissingValue(df_train)
```

```
id = No Missing Value
gender = No Missing Value
age = No Missing Value
hypertension = No Missing Value
heart_disease = No Missing Value
ever_married = No Missing Value
work_type = No Missing Value
Residence_type = No Missing Value
avg_glucose_level = No Missing Value
bmi has missing value = 1462.0 (3.37%)
smoking_status has missing value = 13292 (30.63%)
stroke = No Missing Value
```

In [8]:

```
findMissingValue(df_test)
```

```
id = No Missing Value
gender = No Missing Value
age = No Missing Value
hypertension = No Missing Value
heart_disease = No Missing Value
ever_married = No Missing Value
work_type = No Missing Value
Residence_type = No Missing Value
avg_glucose_level = No Missing Value
bmi has missing value = 591.0 (3.18%)
smoking_status has missing value = 5751 (30.92%)
```

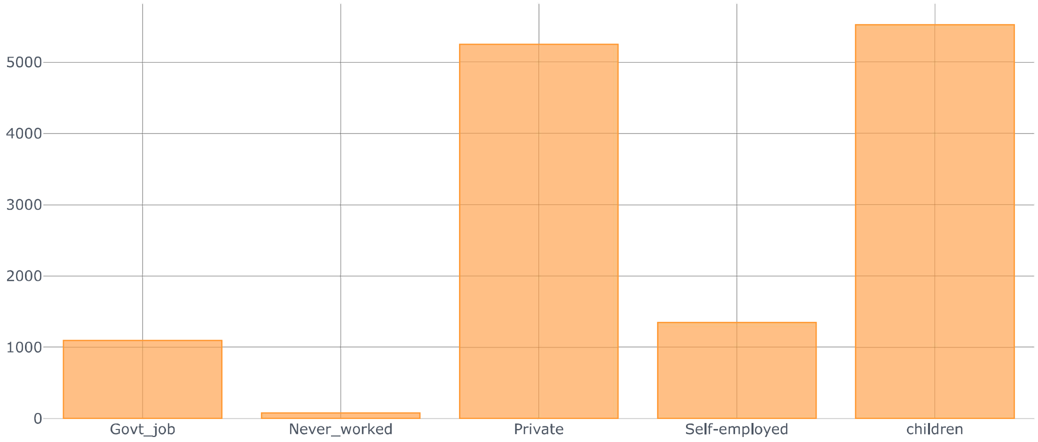
Dealing With Missing Values:

```
In [9]: df_train[df_train['smoking_status'].isna()].count()[0]
Out[9]: 13292
```

```
In [10]: df_test[df_test['smoking_status'].isna()].count()[0]
Out[10]: 5751
```

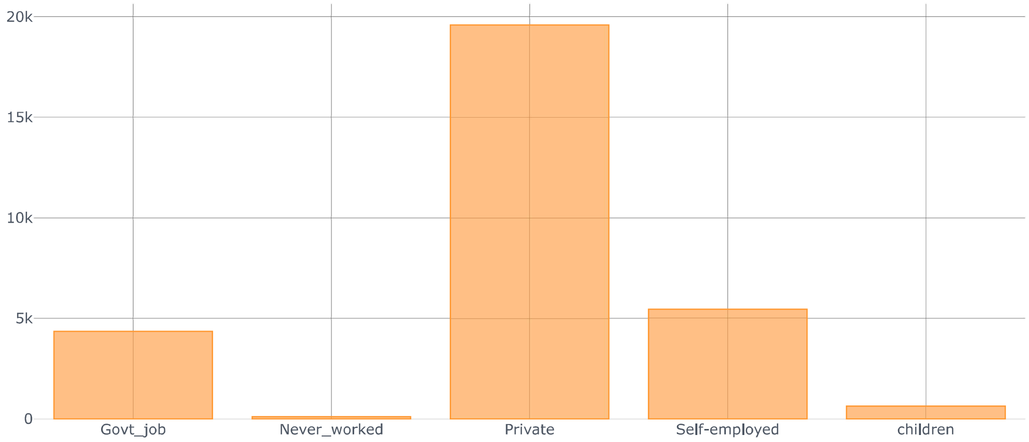
```
In [11]: df_train[df_train['smoking_status'].isna()]['work_type'].unique()
Out[11]: array(['children', 'Private', 'Never_worked', 'Govt_job', 'Self-employed'],
               dtype=object)
```

```
In [12]: df_train[df_train['smoking_status'].isna()].groupby('work_type')['stroke'].count().iplot(kind='bar')
```



Export to plot.ly »

```
In [13]: df_train[df_train['smoking_status'].notna()].groupby('work_type')['stroke'].count().iplot(kind='bar')
```



Export to plot.ly »

Since 30% value is missing, I am considering the that portion as unknown under 'smoking_status' column

```
In [14]: df_train['smoking_status'].fillna(value='unknown',inplace=True)
```

```
In [15]: df_test['smoking_status'].fillna(value='unknown',inplace=True)
```

In [16]:

```
findMissingValue(df_train)
```

```
id = No Missing Value
gender = No Missing Value
age = No Missing Value
hypertension = No Missing Value
heart_disease = No Missing Value
ever_married = No Missing Value
work_type = No Missing Value
Residence_type = No Missing Value
avg_glucose_level = No Missing Value
bmi has missing value = 1462.0 (3.37%)
smoking_status = No Missing Value
stroke = No Missing Value
```

In [17]:

```
findMissingValue(df_test)
```

```
id = No Missing Value
gender = No Missing Value
age = No Missing Value
hypertension = No Missing Value
heart_disease = No Missing Value
ever_married = No Missing Value
work_type = No Missing Value
Residence_type = No Missing Value
avg_glucose_level = No Missing Value
bmi has missing value = 591.0 (3.18%)
smoking_status = No Missing Value
```

Since 'bmi' column has missing value of 3.18%, I am droping those value as we don't know requered parameter to calculate bmi and I also don't want to consider mean/meadian/mode here. And also, as the missing value is below 5%, I considered droping them.

In [18]:

```
df_train.dropna(inplace=True)
```

In [19]:

```
df_test.dropna(inplace=True)
```

In [ ]:

In [20]:

```
findMissingValue(df_train)
```

```
id = No Missing Value
gender = No Missing Value
age = No Missing Value
hypertension = No Missing Value
heart_disease = No Missing Value
ever_married = No Missing Value
work_type = No Missing Value
Residence_type = No Missing Value
avg_glucose_level = No Missing Value
bmi = No Missing Value
smoking_status = No Missing Value
stroke = No Missing Value
```

In [21]:

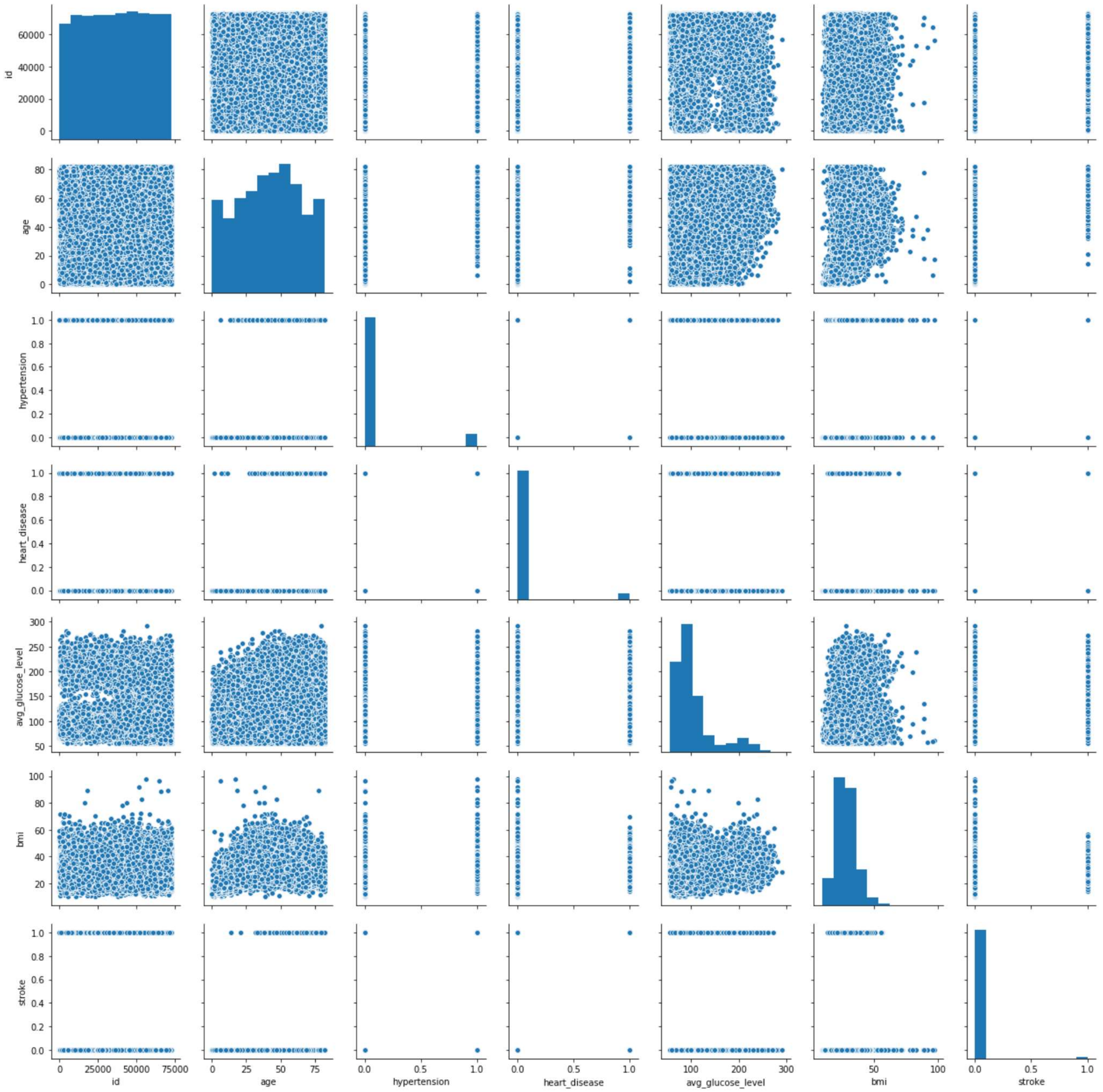
```
findMissingValue(df_test)
```

```
id = No Missing Value
gender = No Missing Value
age = No Missing Value
hypertension = No Missing Value
heart_disease = No Missing Value
ever_married = No Missing Value
work_type = No Missing Value
Residence_type = No Missing Value
avg_glucose_level = No Missing Value
bmi = No Missing Value
smoking_status = No Missing Value
```

Exploratory Data Analysis

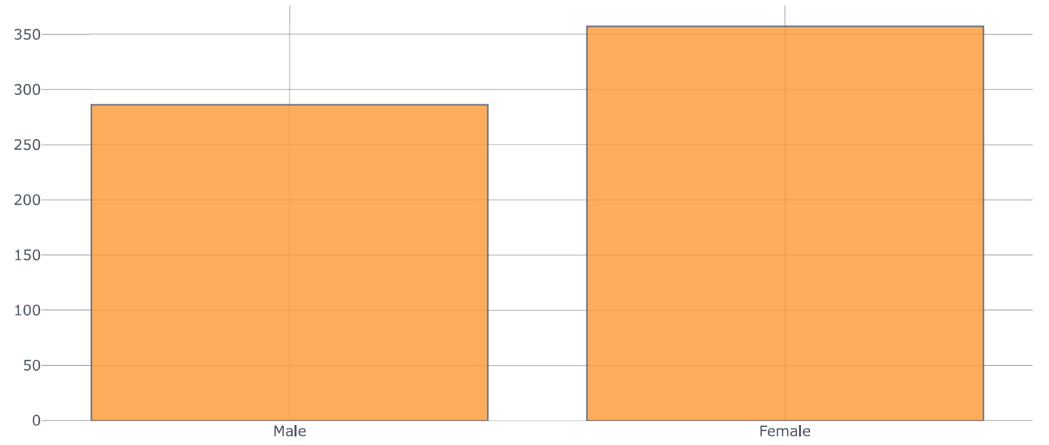
In [22]:
```
sns.pairplot(df_train)
```

Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x21535f0b888>
```



In [23]:

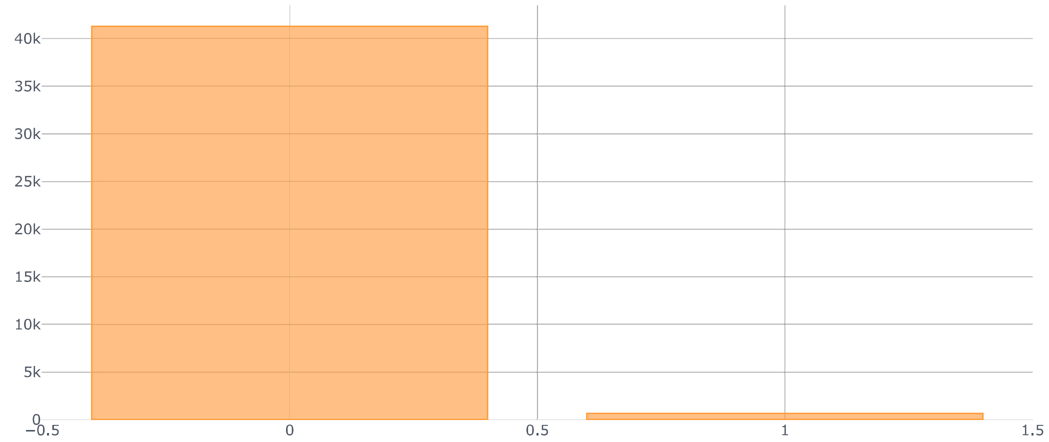
```
df_train[df_train['stroke']==1]['gender'].iplot(kind='hist')
```



Export to plot.ly »

In [24]:

```
df_train['stroke'].value_counts().iplot(kind='bar')
```



Export to plot.ly »

In [25]:

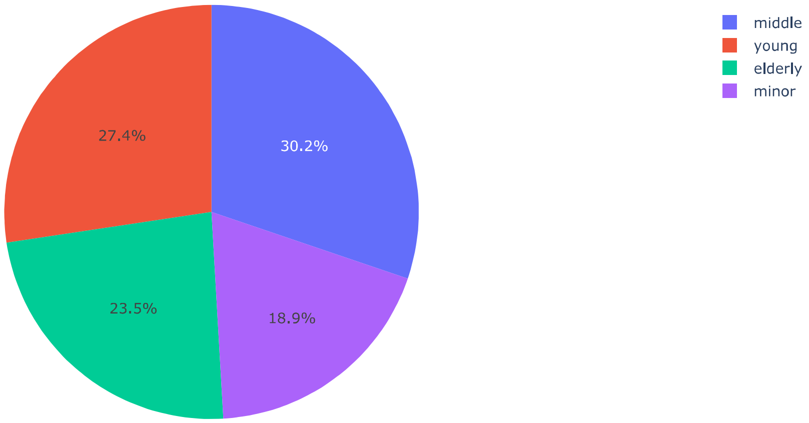
```
minor = df_train[df_train['age']<=18]
young = df_train[(df_train['age']>=19) & (df_train['age']<=40)]
middle = df_train[(df_train['age']>=41) & (df_train['age']<=60)]
elderly = df_train[df_train['age']>=61]
```

In [ ]:

In [26]:
```
import plotly.graph_objects as go

labels = ['minor','young','middle','elderly']
values = [(len(minor)/41938)*100,(len(young)/41938)*100,(len(middle)/41938)*100,(len(elderly)/41938)*100]

fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
fig.show()
```



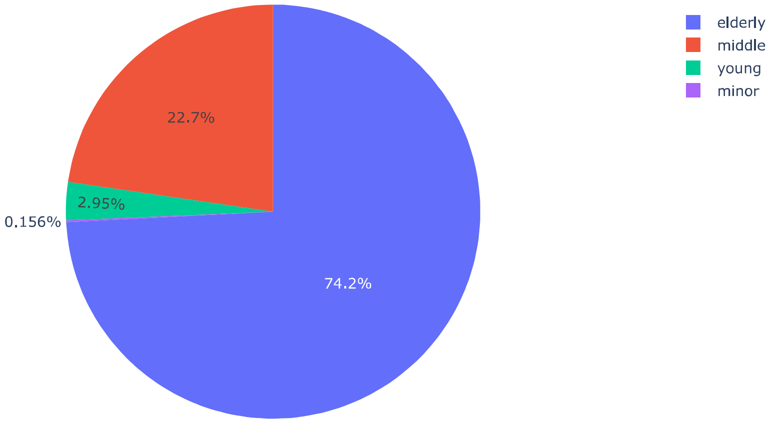
In [27]:
```
minor_stroke = df_train[(df_train['age']<=18) & (df_train['stroke']==1)]
young_stroke = df_train[(df_train['age']>=19) & (df_train['age']<=40) & (df_train['stroke']==1)]
middle_stroke = df_train[(df_train['age']>=41) & (df_train['age']<=60) & (df_train['stroke']==1)]
elderly_stroke = df_train[(df_train['age']>=61) & (df_train['stroke']==1)]
```

In [ ]:

In [28]:
```
import plotly.graph_objects as go

labels = ['minor','young','middle','elderly']
values = [(len(minor_stroke)/643)*100,(len(young_stroke)/643)*100,(len(middle_stroke)/643)*100,(len(elderly_stroke)/643)*100]

fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
fig.show()
```



Predictive Analysis

In [29]:
```
df = pd.get_dummies(df_train)
```

```
In [30]: df=df.drop('id',axis=1)
```

```
In [31]: df2 = pd.get_dummies(df_test)
```

```
In [32]: df2=df2.drop('id',axis=1)
```

```
In [ ]:
```

```
In [33]: X= df.drop(['stroke'],axis=1)
         y= df['stroke']
```

```
In [34]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import classification_report, confusion_matrix
```

```
In [35]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

```
In [36]: from imblearn.ensemble import BalancedBaggingClassifier
         from sklearn.linear_model import LogisticRegression

         #Create an object of the classifier.
         bbc_lr = BalancedBaggingClassifier(base_estimator=LogisticRegression(),
                                         sampling_strategy='auto',
                                         replacement=False,
                                         random_state=1)

         y_train = df['stroke']
         X_train = df.drop(['stroke'], axis=1, inplace=False)

         #Train the classifier.
         bbc_lr.fit(X_train, y_train)


         prediction = bbc_lr.predict(X_test)

         bbc_lr.score(X_test,y_test)
         print('The Logistic Regression Accuracy is {:.2f} %'.format(bbc_lr.score(X_test,y_test)*100))

         print('\n')

         print(classification_report(y_test,prediction))
         print('\n')
         print(confusion_matrix(y_test,prediction))
```

```
The Logistic Regression Accuracy is 75.35 %


              precision    recall  f1-score   support

           0       1.00      0.75      0.86      8252
           1       0.05      0.84      0.10       136

    accuracy                           0.75      8388
   macro avg       0.52      0.80      0.48      8388
weighted avg       0.98      0.75      0.84      8388



[[6206 2046]
 [  22  114]]
```

```
In [ ]:
```

In [37]:

```
from sklearn.tree import DecisionTreeClassifier

#Create an object of the classifier.
bbc_dt = BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                sampling_strategy='auto',
                                replacement=False,
                                random_state=0)

y_train = df['stroke']
X_train = df.drop(['stroke'], axis=1, inplace=False)

#Train the classifier.
bbc_dt.fit(X_train, y_train)

prediction = bbc_dt.predict(X_test)

bbc_dt.score(X_test,y_test)
print('The Decision Tree Accuracy is {:.2f} %'.format(bbc_dt.score(X_test,y_test)*100))

print('\n')

print(classification_report(y_test,prediction))
print('\n')
print(confusion_matrix(y_test,prediction))
```

```
The Decision Tree Accuracy is 82.75 %


              precision    recall  f1-score   support

           0       1.00      0.83      0.90      8252
           1       0.08      0.96      0.15       136

    accuracy                           0.83      8388
   macro avg       0.54      0.89      0.53      8388
weighted avg       0.98      0.83      0.89      8388



[[6810 1442]
 [   5  131]]
```

In [38]:

```
from sklearn.ensemble import RandomForestClassifier
#Create an object of the classifier.
bbc_rf = BalancedBaggingClassifier(base_estimator=RandomForestClassifier(),
                                sampling_strategy='auto',
                                replacement=False,
                                random_state=0)

y_train = df['stroke']
X_train = df.drop(['stroke'], axis=1, inplace=False)

#Train the classifier.
bbc_rf.fit(X_train, y_train)

prediction = bbc_rf.predict(X_test)

bbc_rf.score(X_test,y_test)
print('The Random Forest Accuracy is {:.2f} %'.format(bbc_rf.score(X_test,y_test)*100))

print('\n')

print(classification_report(y_test,prediction))
print('\n')
print(confusion_matrix(y_test,prediction))
```

```
The Random Forest Accuracy is 79.59 %


              precision    recall  f1-score   support

           0       1.00      0.79      0.88      8252
           1       0.07      0.96      0.13       136

    accuracy                           0.80      8388
   macro avg       0.54      0.88      0.51      8388
weighted avg       0.98      0.80      0.87      8388



[[6545 1707]
 [   5  131]]
```

In [39]:

```
from sklearn.svm import SVC
#Create an object of the classifier.
bbc_sv = BalancedBaggingClassifier(base_estimator=SVC(random_state=1),
                                sampling_strategy='auto',
                                replacement=False,
                                random_state=1)

y_train = df['stroke']
X_train = df.drop(['stroke'], axis=1, inplace=False)

#Train the classifier.
bbc_sv.fit(X_train, y_train)

prediction = bbc_sv.predict(X_test)

bbc_sv.score(X_test,y_test)
print('The Support Vector Accuracy is {:.2f} %'.format(bbc_sv.score(X_test,y_test)*100))

print('\n')

print(classification_report(y_test,prediction))
print('\n')
print(confusion_matrix(y_test,prediction))
```

```
The Support Vector Accuracy is 82.84 %


              precision    recall  f1-score   support

           0       1.00      0.83      0.90      8252
           1       0.08      0.88      0.14       136

    accuracy                           0.83      8388
   macro avg       0.54      0.85      0.52      8388
weighted avg       0.98      0.83      0.89      8388



[[6829 1423]
 [  16  120]]
```

In [40]:

```
# Saving Model
import pickle
saved_model = pickle.dumps(bbc_sv)
```

In [41]:

```
# Load the Pickled model
bbc_sv_from_pickle = pickle.loads(saved_model)
```

In [42]:

```
# Using the Loaded pickle model to make predictions
df2['stroke']= bbc_sv_from_pickle.predict(df2)
```

In [43]:

```
df2.head()
```

Out[43]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_Female	gender_Male	gender_Other	ever_married_No	ever_married_Yes	...	work_type_Private	work_ty e
0	80.00	0	0	83.84	21.10	0	1	0	0	1	...	1	
1	74.00	0	1	179.50	26.00	1	0	0	0	1	...	0	
2	14.00	0	0	95.16	21.20	1	0	0	1	0	...	0	
3	28.00	0	0	94.76	23.40	0	1	0	1	0	...	1	
4	63.00	0	0	83.57	27.60	1	0	0	0	1	...	0	

5 rows × 22 columns

In [ ]:

In [44]:

```
df_test['stroke']= df2['stroke']
```

In [45]:

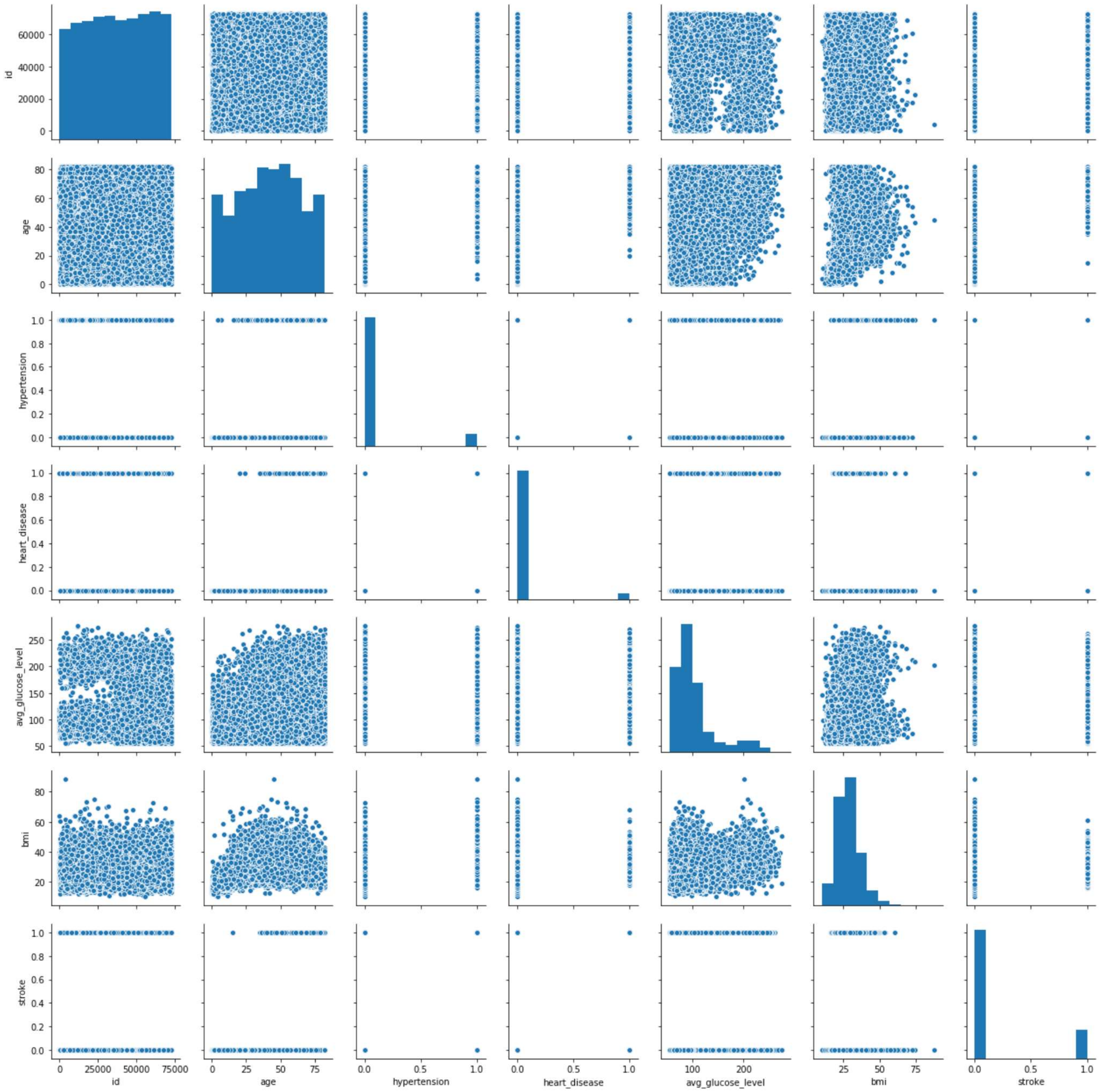
```
df_test.head()
```

Out[45]:

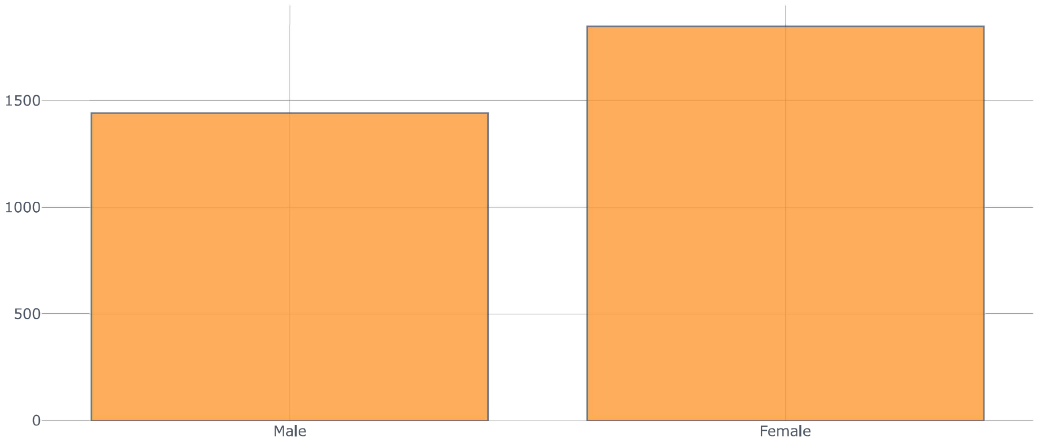
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	36306	Male	80.00	0	0	Yes	Private	Urban	83.84	21.10	formerly smoked	1
1	61829	Female	74.00	0	1	Yes	Self-employed	Rural	179.50	26.00	formerly smoked	1
2	14152	Female	14.00	0	0	No	children	Rural	95.16	21.20	unknown	0
3	12997	Male	28.00	0	0	No	Private	Urban	94.76	23.40	unknown	0
4	40801	Female	63.00	0	0	Yes	Govt_job	Rural	83.57	27.60	never smoked	0

```
In [46]: sns.pairplot(df_test)

Out[46]: <seaborn.axisgrid.PairGrid at 0x2153db1a808>
```

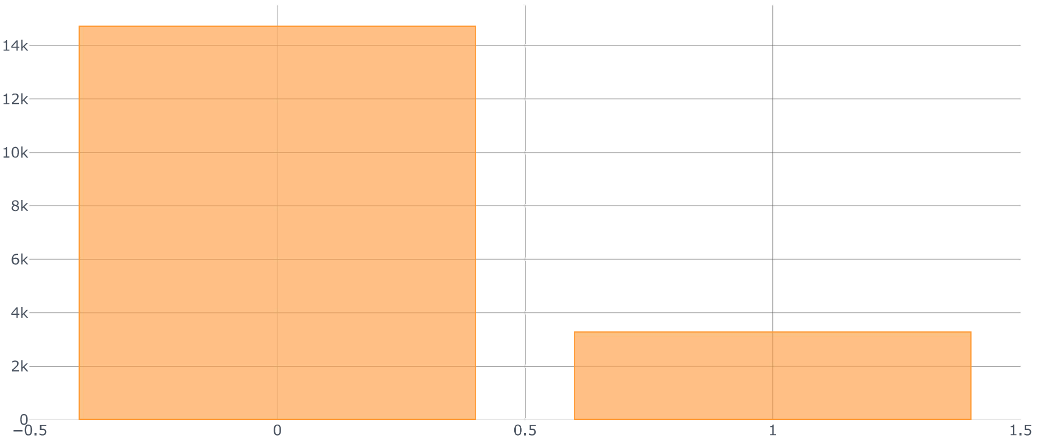


In [47]:
```
df_test[df_test['stroke']==1]['gender'].iplot(kind='hist')
```



Export to plot.ly »

In [48]:
```
df_test['stroke'].value_counts().iplot(kind='bar')
```



Export to plot.ly »

In [49]:
```
len(df_test['age'])
```

Out[49]:
```
18010
```

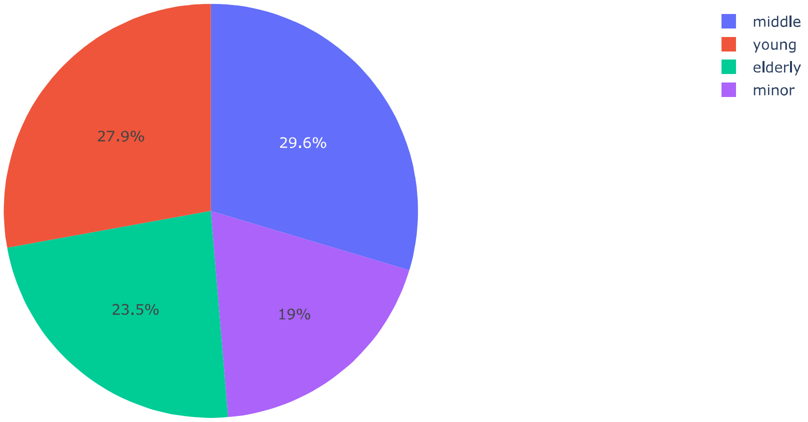
In [50]:
```
minor = df_test[df_test['age']<=18]
young = df_test[(df_test['age']>=19) & (df_test['age']<=40)]
middle = df_test[(df_test['age']>=41) & (df_test['age']<=60)]
elderly = df_test[df_test['age']>=61]
```

In [51]:

```
import plotly.graph_objects as go

labels = ['minor','young','middle','elderly']
values = [(len(minor)/len(df_test['age']))*100,(len(young)/len(df_test['age']))*100,(len(middle)/len(df_test['age']))*100,(len(elderly)/len(df
_test['age']))*100]

fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
fig.show()
```



In [52]:

```
minor_stroke = df_test[(df_test['age']<=18) & (df_test['stroke']==1)]
young_stroke = df_test[(df_test['age']>=19) & (df_test['age']<=40) & (df_test['stroke']==1)]
middle_stroke = df_test[(df_test['age']>=41) & (df_test['age']<=60) & (df_test['stroke']==1)]
elderly_stroke = df_test[(df_test['age']>=61) & (df_test['stroke']==1)]
```

In [53]:

```
len(df_test[df_test['stroke']==1])
```

Out[53]:

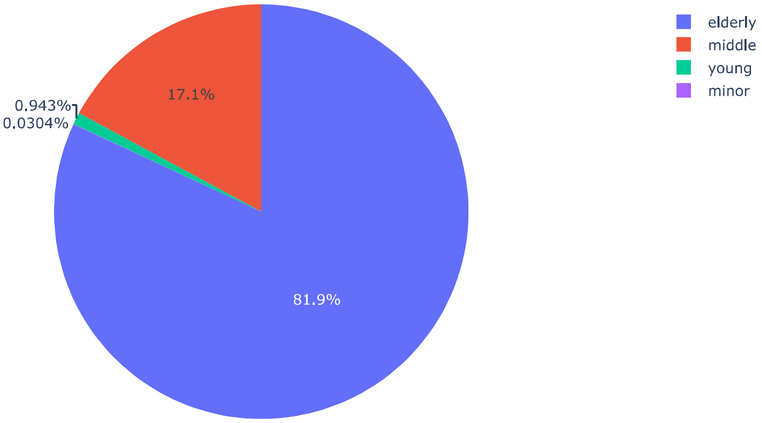
```
3286
```

In [54]:

```
import plotly.graph_objects as go

labels = ['minor','young','middle','elderly']
values = [(len(minor_stroke)/len(df_test[df_test['stroke']==1]))*100,(len(young_stroke)/len(df_test[df_test['stroke']==1]))*100,(len(middle_st
roke)/len(df_test[df_test['stroke']==1]))*100,(len(elderly_stroke)/len(df_test[df_test['stroke']==1]))*100]

fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
fig.show()
```


elderly
middle
young
minor