A stroke occurs when part of the brain loses its blood supply and stops working. This causes the part of the body that the injured brain controls to stop working. A stroke also is called a cerebrovascular accident, CVA, or "brain attack."

Causes of Stroke :

High blood pressure , Diabetes, Heart disease, Cigarette smoking, Physical inactivity, High cholesterol, Sickle cell disease, Drinking too much alcohol

In [7]:
```python
import numpy as np       # numpy and pandas use to perform data analysis
import pandas as pd
import seaborn as sns    # Seaborn and matplotlib used to perform data visualization
import matplotlib.pyplot as plt
import warnings          # Used to ignore warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,roc_curve)
```

In [8]:
```python
# Loading the dataset using read_csv
df = pd.read_csv('F:\Machine Learning\Data Files\healthcare-dataset-stroke-data.csv')

# head() is used to see first 5 rows of dataset
df.head(5)
```

Out[8]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Exploring the dataset and cleaning it.

In [9]:
```python
# id column will not give any impact on data analysis so we can drop this column.
df.drop('id', axis=1, inplace=True)
```

In [10]:
```python
df.head(2)
```

Out[10]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1

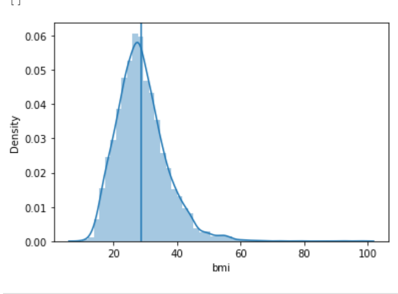
In [11]:
```python
# isnull() is used to check null values.
df.isnull().sum()
```

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
```

In [12]:
```python
# As we can see there are 201 null values in bmi column.
# So we will check whether a column is normally distributed or not.
# If data is normally distributed we will replace null values with mode value
# or else with median.

sns.distplot(df['bmi'])
plt.axvline(df['bmi'].mean())
plt.plot()
```

Out[12]: []



In [13]:
```python
df['bmi'].fillna(df['bmi'].mode()[0], inplace=True)
```

In [14]:
```python
df_num = df.select_dtypes(['int64','float64'])
df_cat = df.select_dtypes('object')
```

In [15]:
```python
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
```

===> Standard Scaling : As we can see our data is not scaled so we will perform Min Max Scaling. Standard Scaling is techique used to scale data, it gives values between 0 to 1.

In [16]:
```python
for col in df_num:
  mm = MinMaxScaler()
  df_num[col] = mm.fit_transform(df_num[[col]])

df_num.head()
```

Out[16]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
0	0.816895	0.0	1.0	0.801265	0.301260	1.0
1	0.743652	0.0	0.0	0.679023	0.210767	1.0
2	0.975586	0.0	1.0	0.234512	0.254296	1.0
3	0.597168	0.0	0.0	0.536008	0.276060	1.0
4	0.963379	1.0	0.0	0.549349	0.156930	1.0

===> Label Encoding : Label Encoding is used to convert categorical data into numeric data.

In [17]:
```python
# In our Dataset we have categorical data as well, so we need to convert this data
# into numerical format as ML understands only numbers.
# For this we will use Label Encoding.

for col in df_cat:
  le = LabelEncoder()
  df_cat[col] = le.fit_transform(df_cat[col])

df_cat.head()
```

Out[17]:

	gender	ever_married	work_type	Residence_type	smoking_status
0	1	1	2	1	1
1	0	1	3	0	2
2	1	1	2	0	2
3	0	1	2	1	3
4	0	1	3	0	2

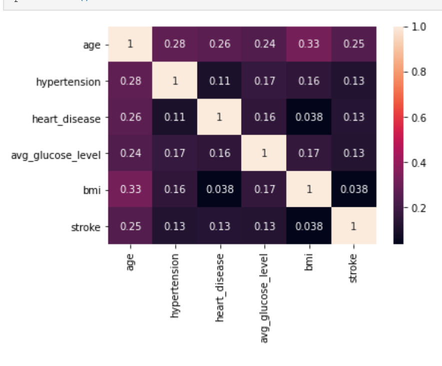
In [18]:
```python
# concat() is used to merge different data.
df_new = pd.concat([df_num, df_cat], axis = 1)
df_new.head()
```

Out[18]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender	ever_married	work_type	Residence_type	smoking_status
0	0.816895	0.0	1.0	0.801265	0.301260	1.0	1	1	2	1	1
1	0.743652	0.0	0.0	0.679023	0.210767	1.0	0	1	3	0	2
2	0.975586	0.0	1.0	0.234512	0.254296	1.0	1	1	2	0	2
3	0.597168	0.0	0.0	0.536008	0.276060	1.0	0	1	2	1	3
4	0.963379	1.0	0.0	0.549349	0.156930	1.0	0	1	3	0	2

Now we have prepared and cleaned data which is ready to Visualize.

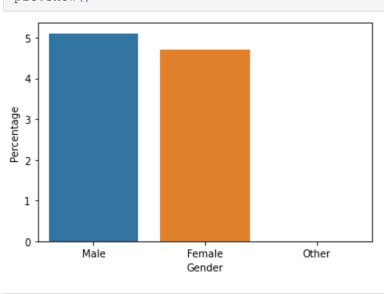
In [19]:
```python
# Checking Corelation
sns.heatmap(df.corr(), annot=True)
plt.show()
```



Data Visualization (% of Stroke vs each columns)

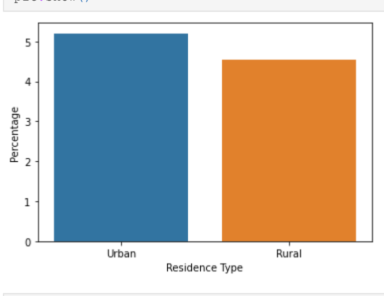
In [20]:
```python
#===== Gender
x_axis = []
y_axis = []

for col in df['gender'].unique():
  data = df[df['gender']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['gender']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel("Gender")
plt.ylabel("Percentage")
plt.show()
```



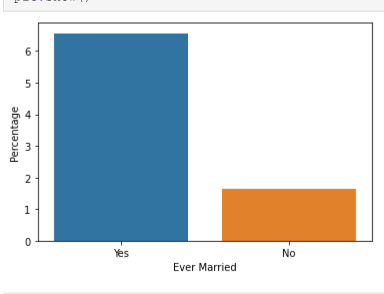
In [21]:
```python
#======= Residence_type
x_axis = []
y_axis = []

for col in df['Residence_type'].unique():
  data = df[df['Residence_type']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['Residence_type']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel('Residence Type')
plt.ylabel("Percentage")
plt.show()
```



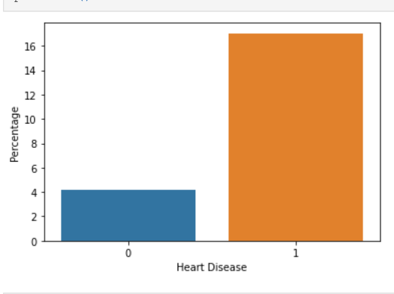
In [22]:
```python
#======= Ever Married
x_axis = []
y_axis = []

for col in df['ever_married'].unique():
  data = df[df['ever_married']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['ever_married']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel("Ever Married")
plt.ylabel("Percentage")
plt.show()
```



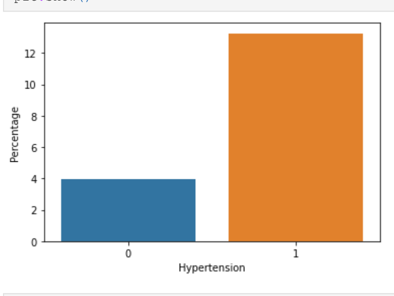
In [23]:
```python
#======= Heart_disease
x_axis = []
y_axis = []

for col in df['heart_disease'].unique():
  data = df[df['heart_disease']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['heart_disease']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel("Heart Disease")
plt.ylabel("Percentage")
plt.show()
```



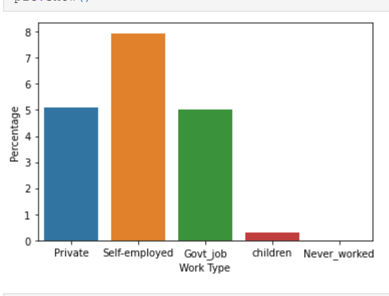
In [24]:
```python
#======= Hypertension
x_axis = []
y_axis = []

for col in df['hypertension'].unique():
  data = df[df['hypertension']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['hypertension']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel("Hypertension")
plt.ylabel("Percentage")
plt.show()
```



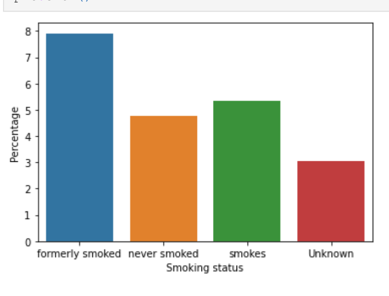
In [25]:
```python
#======= Work Type
x_axis = []
y_axis = []

for col in df['work_type'].unique():
  data = df[df['work_type']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['work_type']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel("Work Type")
plt.ylabel("Percentage")
plt.show()
```



In [26]:
```python
#======= Smoking status
x_axis = []
y_axis = []

for col in df['smoking_status'].unique():
  data = df[df['smoking_status']==col]
  s_yes = len(data[data['stroke']==1])
  total = len(df[df['smoking_status']==col])
  percentage = (s_yes/total)*100
  x_axis.append(col)
  y_axis.append(percentage)
sns.barplot(x_axis,y_axis)
plt.xlabel("Smoking status")
plt.ylabel("Percentage")
plt.show()
```



Conclusion of Data Visualization :

1. Both the Genders have arround 5% chance.

2. People with history of Hypertension and Heart Disease have shown an increased in percentage of Stroke with around 12.5% and 16.5% respectively.

3. Married/Divorced people have a 6.5% chance of stroke.

4. Self Employed people have a higher chance compared to Private and Govt Jobs.

5. Rural and Urban doesn't show much difference.

6. For some reason people who once used to smoke have higher chance compared to people who are still smoking.

We have our Target column as categorical data so we need to check whether a column is balanced or not.

In [27]:
```python
df_new['stroke'].value_counts()
```

Out[27]:
```
0.0    4861
1.0     249
Name: stroke, dtype: int64
```

===> Resampling of data : We have unbalanced data so we need to do resampling of data by using Resampling technique.

In [28]:
```python
from sklearn.model_selection import train_test_split    # used to split data into training and testing part
from collections import Counter
```

In [29]:
```python
x = df_new.drop('stroke', axis=1)
y = df['stroke']
```

In [30]:
```python
df_new['stroke'] = df_new['stroke'].astype(int)
```

In [31]:
```python
df_zero = df_new[df_new['stroke']==0]
df_one = df_new[df_new['stroke']==1]
print(len(df_zero),len(df_one))
```

```
4861 249
```

In [32]:
```python
((249+4861)/2) - 249
```

Out[32]: 2306.0

In [33]:
```python
df_new_2 = pd.concat([df_one, df_zero.sample(n=2300)], axis=0)
```

In [34]:
```python
df_new_2.shape
```

Out[34]: (2549, 11)

Model Building

Dividing data into training and testing

In [35]:
```python
x = df_new_2.drop('stroke', axis=1)
y = df_new_2['stroke']

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3,random_state=1)
```

Here we have worked with 4 models.

1. Logistic Regression

2. Decision Tree

3. Random Forest

4. SVM

In [36]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [37]:
```python
def find_best_model(x, y):
    models = {
        'logistic_regression': {
            'model': LogisticRegression(solver='lbfgs', multi_class='auto'),
            'parameters': {
                'C': [1,5,10]
               }
        },

        'decision_tree': {
            'model': DecisionTreeClassifier(splitter='best'),
            'parameters': {
                'criterion': ['gini', 'entropy'],
                'max_depth': [5,10]
            }
        },

        'random_forest': {
            'model': RandomForestClassifier(criterion='gini'),
            'parameters': {
                'n_estimators': [10,15,20,50,100,200]
            }
        },

        'svm': {
            'model': SVC(gamma='auto'),
            'parameters': {
                'C': [1,10,20],
                'kernel': ['rbf','linear']
            }
        }

    }

    scores = []
    cv_shuffle = ShuffleSplit(n_splits=5, test_size=0.20, random_state=0)

    for model_name, model_params in models.items():
        gs = GridSearchCV(model_params['model'], model_params['parameters'], cv = cv_shuffle, return_train_scor
        gs.fit(x, y)
        scores.append({
            'model': model_name,
            'best_parameters': gs.best_params_,
            'score': gs.best_score_
        })

    return pd.DataFrame(scores, columns=['model','best_parameters','score'])

find_best_model(x_train, y_train)
```

Out[37]:

	model	best_parameters	score
0	logistic_regression	{'C': 10}	0.897479
1	decision_tree	{'criterion': 'gini', 'max_depth': 5}	0.889636
2	random_forest	{'n_estimators': 15}	0.893557
3	svm	{'C': 1, 'kernel': 'rbf'}	0.896919

Note: Since the Random Forest algorithm has the highest accuracy, we futher fine tune the model using hyperparameter optimization.

In [38]:
```python
# Using cross_val_score for gaining average accuracy
from sklearn.model_selection import cross_val_score
scores = cross_val_score(RandomForestClassifier(n_estimators=20, random_state=0), x_train, y_train, cv=5)
print('Average Accuracy : {}%'.format(round(sum(scores)*100/len(scores)), 3))
```

```
Average Accuracy : 90%
```

In [39]:
```python
# Creating Random Forest Model
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(x_train, y_train)
```

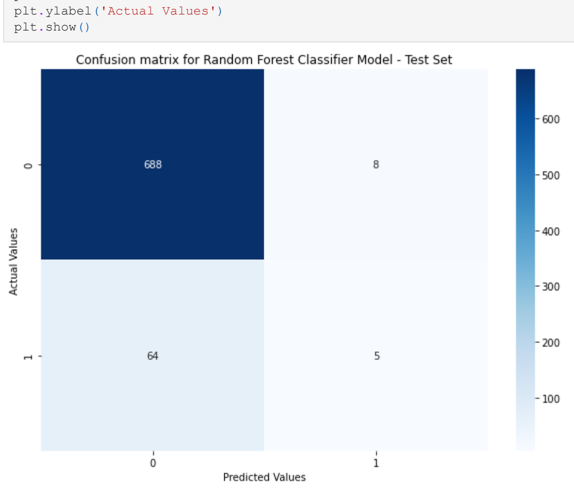
Out[39]: RandomForestClassifier(n_estimators=20, random_state=0)

Model Evaluation

In [40]:
```python
# Creating a confusion matrix
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
y_pred = classifier.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[40]:
```
array([[688,   8],
       [ 64,   5]], dtype=int64)
```

In [41]:
```python
# Plotting the confusion matrix
plt.figure(figsize=(10,7))
p = sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion matrix for Random Forest Classifier Model - Test Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



In [42]:
```python
# Accuracy Score
score = round(accuracy_score(y_test, y_pred),4)*100
print("Accuracy on test set: {}%".format(score))
```

```
Accuracy on test set: 90.59%
```

In [43]:
```python
# After evaluating model we will move for checking prediction for data.
x = df_new_2.drop('stroke',axis=1)
y = df_new_2['stroke']

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3,random_state=1)
# Creating Random Forest Model
classifier = RandomForestClassifier(n_estimators=20)
classifier.fit(x_train, y_train)
```

Out[43]: RandomForestClassifier(n_estimators=20)

In [44]:
```python
# # prediction 1  [age,hypertension, heart_disease, avg_glucose_level, bmi,
#                   #  gender,ever_married,     work_type,Residence_type,smoking_status]
prediction = classifier.predict([[67,0,1,228.69,36.6,1,1,2,1,1]])
print(prediction)
if prediction:
  print('Oops! You have Stroke Disease.')
else:
  print("Great! You don't have Stroke.")
```

```
[1]
Oops! You have Stroke Disease.
```

In [45]:
```python
# # prediction 2    age,hypertension, heart_disease, avg_glucose_level, bmi,
#                   #  gender,ever_married,     work_type,Residence_type,smoking_status
prediction = classifier.predict([[3,0,0,95.12,18,1,0,4,0,0]])
print(prediction)
if prediction:
  print('Oops! You have Stroke Disease.')
else:
  print("Great! You don't have Stroke.")
```

```
[0]
Great! You don't have Stroke.
```

In [46]:
```python
# # prediction 3    [age,hypertension, heart_disease, avg_glucose_level, bmi,
#                   #  gender,ever_married,     work_type,Residence_type,smoking_status]
prediction = classifier.predict([[1, 67, 0, 1, 1, 2, 0, 228.96, 36.6, 1]])
print(prediction)
if prediction:
  print('Oops! You have Stroke Disease.')
else:
  print("Great! You don't have Stroke.")
```

```
[0]
Great! You don't have Stroke.
```

In [ ]: