Main Report

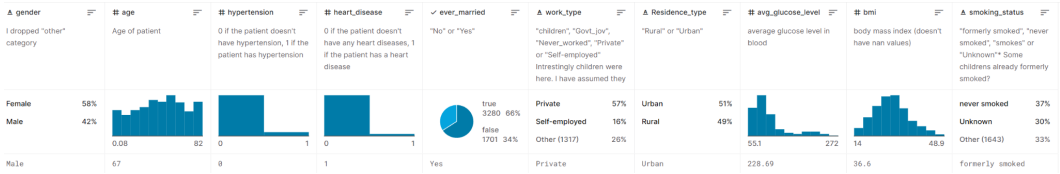
Ahmad Almasri - 30114233

Stage-1

Source of Data:

https://www.kaggle.com/datasets/jillanisofttech/brain-stroke-dataset

No Copyrights. This data set represents records of people who had or had not had a brain-stroke before. It consists of 10 columns. The columns are gender, age, hypertension , heart_disease, ever_married, work_type, Residence_type, avg_glucose_level, bmi, smoking_status and stroke.



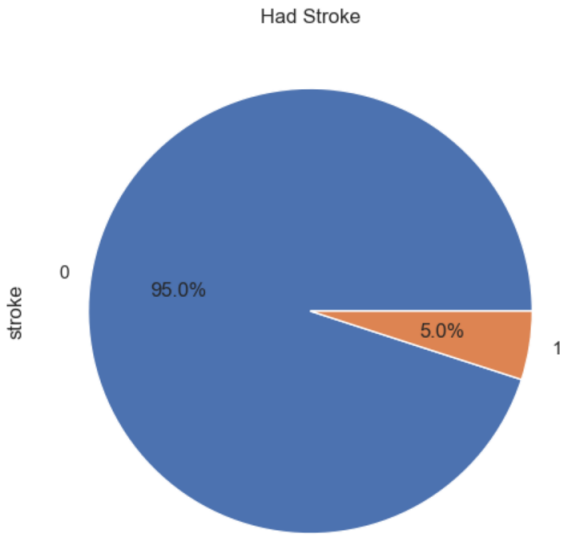
cleaning data:

All records were filled and there are no empty cells in this data. This data set met the requirements for this assignment. However, I decided to convert 3 string columns into numerical columns because the string in these columns are binary (0 or 1), and converting them from now would enhance the efficiency of the learning model that we want to build and would reduce the number of string inputs to our model. The columns that I changed are gender, ever_married, Residence_type.

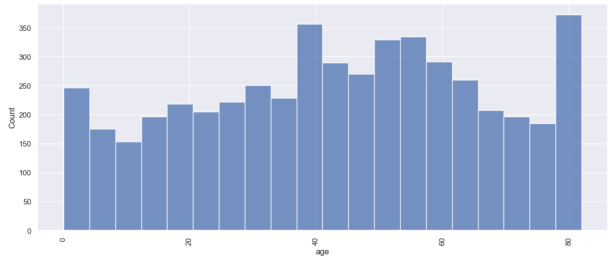
	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	67.0	0	1	1	Private	1	228.69	36.6	formerly smoked	1
1	0	80.0	0	1	1	Private	0	105.92	32.5	never smoked	1
2	1	49.0	0	0	1	Private	1	171.23	34.4	smokes	1
3	1	79.0	1	0	1	Self-employed	0	174.12	24.0	never smoked	1
4	0	81.0	0	0	1	Private	1	186.21	29.0	formerly smoked	1

Visualizations:

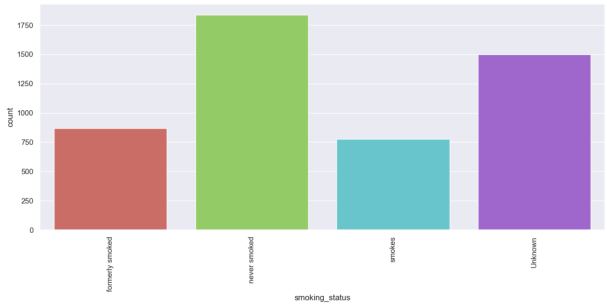
- Pie chart: represent the percentage of people who had or had not had a brain-stroke before.



- Histogram: counter for age groups in the data set.



- Par Chart: Divide the people in the data into groups based on smoking_status.



Stage-2

Columns:

The number of numerical inputs is 8. They are gender, age, hypertension, heart_disease, ever_married, Residence_type, avg_glucose_level, bmi. The output column is the stroke column and it is a binary value. The aim of this model is to provide a tool of prediction based on data that we have, and I want to see if we can predict if a group of data might lead us to a result such as a stroke or no stroke.

```
X=df.drop(['stroke', 'smoking_status', 'work_type'],axis=1)
y=df['stroke']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.25,random_state=3)
X.shape
```

Splitting Data:

The data for training is 75%, and the data for testing is 25%. My decision is to provide more data for training to help the model in generalization. Moreover, we have a considerably good amount for training that should lead to a similar result as the training's result.

Model:

The model is the same exact model used in MINIST.ipynb, but I have changed the hyper-parameters. I used 3 layers. The first one is the input layer. The second is the hidden layers. This layer consists of 128 neurons. The last layer is the output, and the number of neurons is 1 since it is a binary not a category classification.

```
print("--Make model--")
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(8, input_shape=(8,)),
  tf.keras.layers.Dense(32, activation='tanh'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Hyper-parameters:

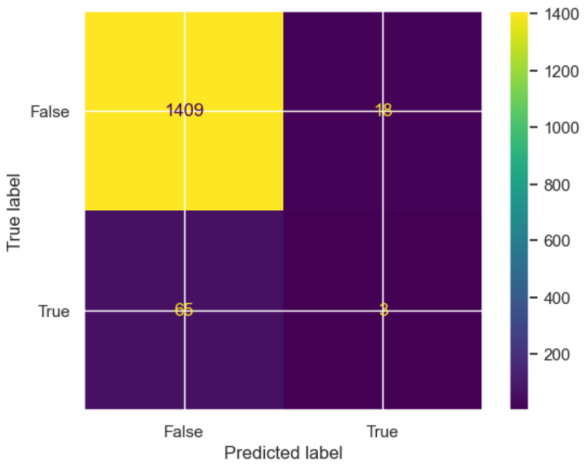
I have changed the hyper-parameters; therefore it is not the same as MINIST. The reason for changing them is that my output is a binary data with 0 or 1. For the input and hidden layer, we use the TANH as an activation function, and the last layer uses the SIGMOID. The used optimizer is ADAM, and for loss function, my choice was binary_crossentropy. The metric is accuracy. The number of epochs is 2.

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print("--Fit model--")
model.fit(X_train, y_train, epochs=2, verbose=2)
```

Result:

The train accuracy is 94% with loss at 0.2189 and the test is 94% with loss at 0.2203.

Stage-3

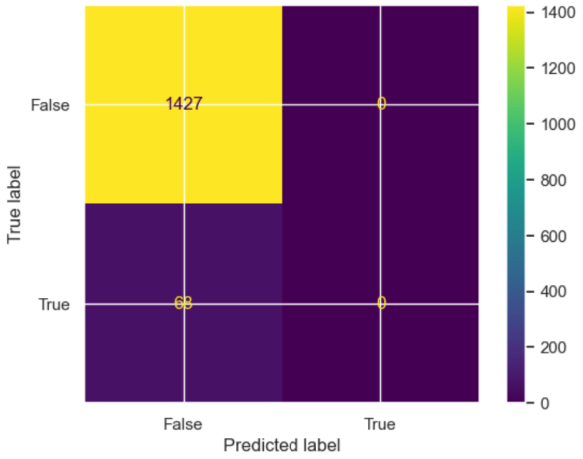


This is a 4 panel graph representing true-positives, true-negatives, false-positives, false-negatives. This is the result before making the new model.

Additional Changes:

I added a new hidden layer with 32 neurons. I increased the number of epochs to 5. I added a dropout layer with a probability equal to 10%. I changed the metrics to binary_accuracy. Moreover, I give more data for training and now the percent of training data is 30%. The new accuracy is 95.5% for testing data.

```
print("--Make model--")
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(8, input_shape=(8,), activation='tanh' ) )
model.add(tf.keras.layers.Dense(32, activation='tanh'))
model.add(tf.keras.layers.Dense(32, activation='tanh'))
model.add(tf.keras.layers.Dropout(0.20))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```



Before the true-negative was 18 but now it is 0.

Stage-4

New Changes:

I used regularization to add a penalty to the error function. I used tf.keras.layers.Dropout(Probability), which leads to ignoring random numbers of neurons and I decrease its value from 20 to 10.

```
print("--Make model--")
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(10, input_shape=(10,), activation='tanh', kernel_regularizer=tf.keras.regularizers.L2(0.001)) )
model.add(tf.keras.layers.Dense(32, activation='tanh', kernel_regularizer=tf.keras.regularizers.L2(0.001)))
model.add(tf.keras.layers.Dropout(0.20))
model.add(tf.keras.layers.Dense(32, activation='tanh', kernel_regularizer=tf.keras.regularizers.L2(0.001)))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

overfitting/under-fitting concerns:

First, this dataset does not lead to an overfitting/underfitting problem. The accuracy of the training and testing are both very near to each other. The number of epochs and the batch size provide enough time and learning experience for the model to not underfitting. The dropout layer and regularization play the role of preventing the result from overfitting.

```
def handle_non_num_data(df):
    columns = df.columns.values

    for column in columns:
        text_digit_values = {}

        def convert_to_int(val):
            return text_digit_values[val]

        if df[column].dtype != np.int64 and df[column].dtype != np.float64:
            column_cont = df[column].values.tolist()
            unique_elem = set(column_cont)

            x = 0
            for unique in unique_elem:
                if unique not in text_digit_values:
                    text_digit_values[unique] = x
                    x+=1
            df[column] = list(map(convert_to_int, df[column]))

    return df
df = handle_non_num_data(df4)
df.head(5)
```

Model decisions:

There was no need for me to use one-hot-encoding or other approach of handling non-numerical input because all of the strings thatI have are related to numbers. In other words, strings can be converted to numbers and still have the full meaning. For instance, converting string values to numbers such as 1, 2, 3 …etc. I think one-hot-encoding might need a more complex string input. I made a function in Python called handle_non_num_data to convert the string to digits. Finally, I ended up with 95.5% accuracy for testing data including all the input fields. The training accuracy, 94.8%, was not far from the testing one.