Data

In [4]:
```
DATAPATH = Path('C:/Users/zizhe/Desktop/Leo Zhao/10 Stroke Prediction')
```

In [5]:
```
train = pd.read_csv(DATAPATH/'train.csv').drop(columns='id')
test = pd.read_csv(DATAPATH/'test.csv').drop(columns='id')

train.head()
```

Out[5]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_l
0	Male	28.0	0	0	Yes	Private	Urban	7
1	Male	33.0	0	0	Yes	Private	Rural	7
2	Female	42.0	0	0	Yes	Private	Rural	10
3	Male	56.0	0	0	Yes	Private	Urban	6
4	Female	24.0	0	0	No	Private	Rural	7

Columns details:

1. `id` : unique identifier
2. `gender` : categorical = "Male", "Female" or "Other"
3. `age` : float = (0.08 - 82)
4. `hypertension` : bool = (0, 1)
5. `heart_disease` : bool = (0, 1)
6. `ever_married` : categorical = "No", "Yes" (will be converted into bool)
7. `work_type` : categorical = "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8. `Residence_type` : categorical = "Rural" or "Urban"
9. `avg_glucose_level` : float = average glucose level in blood
10. `bmi` : float = body mass index
11. `smoking_status` : categorical = "formerly smoked", "never smoked", "smokes" or "Unknown"
12. `stroke` : bool = (0, 1)

Initial Visualization and Thoughts

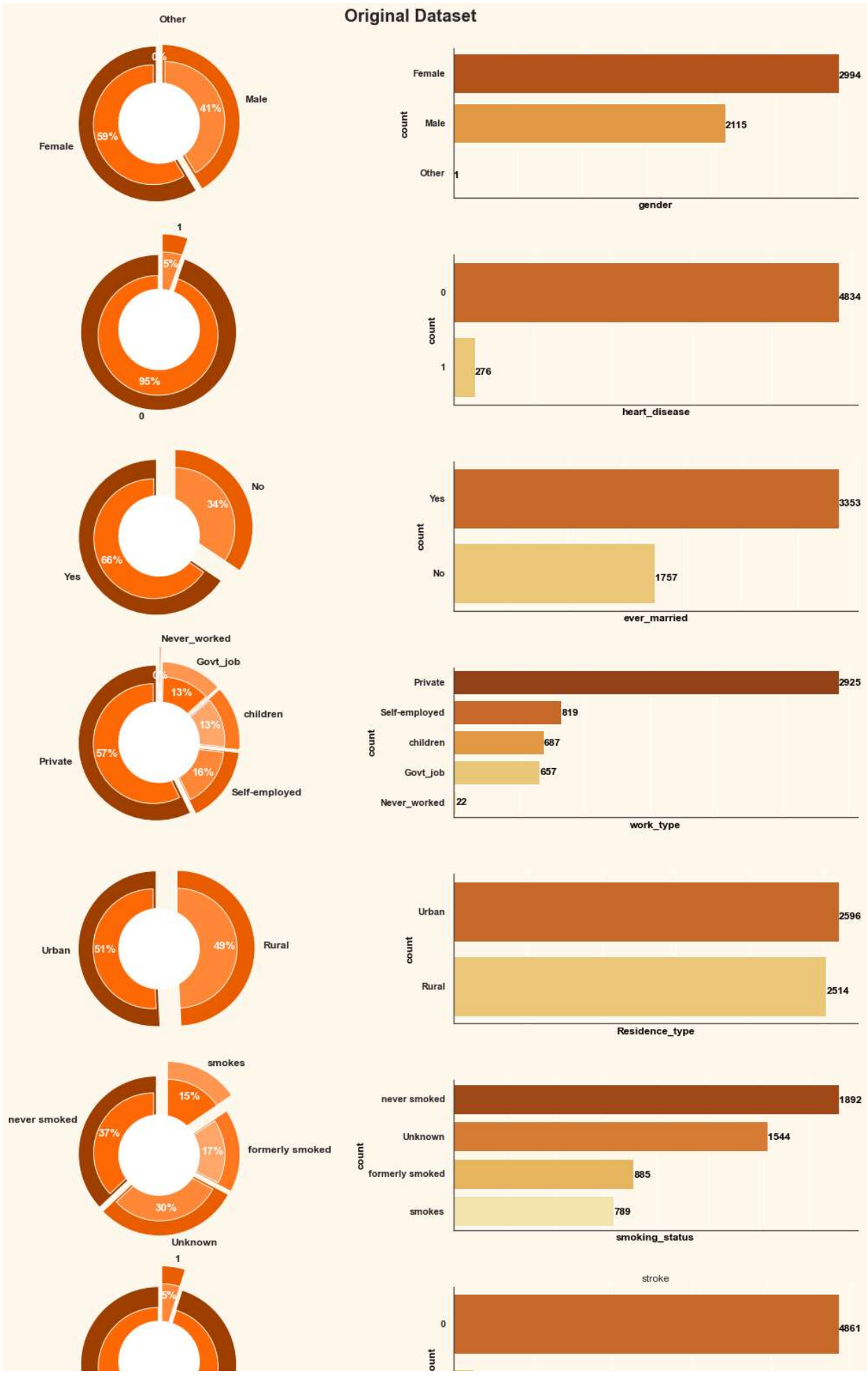
In [6]:
```
def plot_count(df: pd.core.frame.DataFrame, col_list: list, title_name: str='Train') -
    """Draws the pie and count plots for categorical variables.

    Args:
        df: train or test dataframes
        col_list: a list of the selected categorical variables.
```

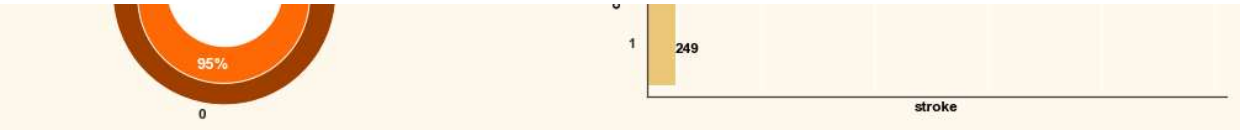
```
        title_name: 'Train' or 'Test' (default 'Train')

    Returns:
        subplots of size (len(col_list), 2)
    """
    f, axes = plt.subplots(len(col_list), 2, figsize=(15, 24))
    plt.subplots_adjust(wspace=0)

    for col_name, ax in zip(col_list, axes):
        s1 = df[col_name].value_counts()
        N = len(s1)

        outer_sizes = s1
        inner_sizes = s1/N

        outer_colors = ['#9E3F00', '#eb5e00', '#ff781f', '#ff9752', '#ff9752']
        inner_colors = ['#ff6905', '#ff8838', '#ffa66b']

        ax[0].pie(
            outer_sizes,colors=outer_colors,
            labels=s1.index.tolist(),
            startangle=90,frame=True, radius=1.3,
            explode=([0.05]*(N-1) + [.3]),
            wedgeprops={ 'linewidth' : 1, 'edgecolor' : 'white'},
            textprops={'fontsize': 12, 'weight': 'bold'}
        )

        textprops = {
            'size':13,
            'weight': 'bold',
            'color':'white'
        }

        ax[0].pie(
            inner_sizes, colors=inner_colors,
            radius=1, startangle=90,
            autopct='%1.f%%',explode=([.1]*(N-1) + [.3]),
            pctdistance=0.8, textprops=textprops
        )

        center_circle = plt.Circle((0,0), .68, color='black',
                                   fc='white', linewidth=0)
        ax[0].add_artist(center_circle)

        x = s1
        y = s1.index.astype(str)

        sns.barplot(
            x=x, y=y, ax=ax[1],
            palette='YlOrBr_r', orient='horizontal'
        )

        ax[1].spines['top'].set_visible(False)
        ax[1].spines['right'].set_visible(False)
        ax[1].tick_params(
            axis='x',
            which='both',
            bottom=False,
            labelbottom=False
        )
```

```
        for i, v in enumerate(s1):
            ax[1].text(v, i+0.1, str(v), color='black',
                       fontweight='bold', fontsize=12)

        plt.title(col_name)
        plt.setp(ax[1].get_yticklabels(), fontweight="bold")
        plt.setp(ax[1].get_xticklabels(), fontweight="bold")
        ax[1].set_xlabel(col_name, fontweight="bold", color='black')
        ax[1].set_ylabel('count', fontweight="bold", color='black')

    f.suptitle(f'{title_name} Dataset', fontsize=20, fontweight='bold')
    plt.tight_layout()
    plt.show()

def pair_plot(df: pd.core.frame.DataFrame, title_name: str, hue: str) -> None:
    """Draws the pairplot for the selected dataframe.

    Args:
        df: train, test or combined dataframes
        title_name: any string title
        hue: a specified categorical column name

    Returns:
        pairplots
    """
    s = sns.pairplot(df, hue=hue, palette=['#9E3F00', '#eb5e00'])
    s.fig.set_size_inches(16, 12)
    s.fig.suptitle(title_name, y=1.08)
    plt.show()

def plot_correlation_heatmap(df: pd.core.frame.DataFrame, title_name: str='Train corre
    """Draws the correlation heatmap plot.

    Args:
        df: train or test dataframes
        title_name: 'Train' or 'Test' (default 'Train correlation')

    Returns:
        subplots of size (len(col_list), 2)
    """

    corr = df.corr()
    fig, axes = plt.subplots(figsize=(20, 10))
    mask = np.zeros_like(corr)
    mask[triu_indices_from(mask)] = True
    sns.heatmap(corr, mask=mask, linewidths=.5, cmap='YlOrBr_r', annot=True)
    plt.title(title_name)
    plt.show()
```

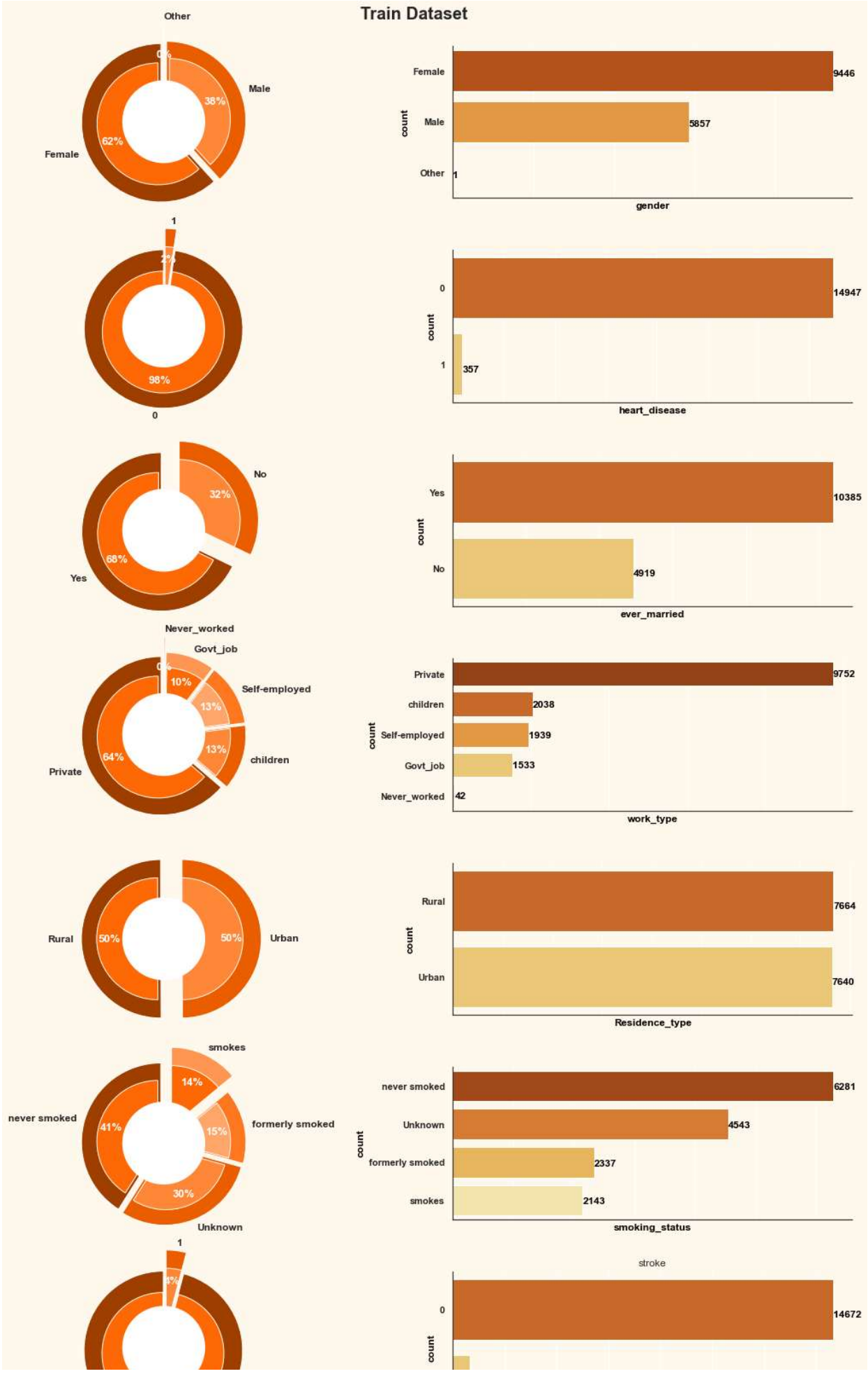
In [11]:
```
origin = pd.read_csv('C:/Users/zizhe/Desktop/Leo Zhao/10 Stroke Prediction/healthcare-
```

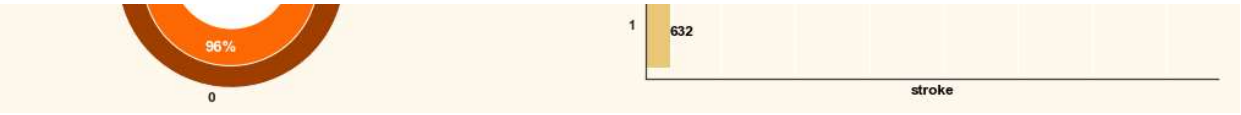
In [12]:
```
plot_count(origin, selected_columns, 'Original')
```


Original Dataset
Other
0%
Male
41%
Female
59%
Female
2994
Male
2115
Other
1
count
gender
1
5%
95%
0
0
4834
1
276
count
heart_disease
No
34%
Yes
66%
Yes
3353
No
1757
count
ever_married
Never_worked
0%
Govt_job
13%
children
13%
Private
57%
Self-employed
16%
Private
2925
Self-employed
819
children
687
Govt_job
657
Never_worked
22
count
work_type
Urban
51%
Rural
49%
Urban
2596
Rural
2514
count
Residence_type
smokes
15%
never smoked
37%
formerly smoked
17%
30%
Unknown
never smoked
1892
Unknown
1544
formerly smoked
885
smokes
789
count
smoking_status
1
5%
stroke
0
4861
count



In [7]:
```
selected_columns = ['gender', 'heart_disease', 'ever_married', 'work_type',
                    'Residence_type', 'smoking_status', 'stroke']
plot_count(train, selected_columns)
```


Train Dataset
Other
Male
Female
0%
38%
62%
Female
Male
Other
9446
5857
1
gender
1
0
2%
98%
0
1
14947
357
heart_disease
No
Yes
32%
68%
Yes
No
10385
4919
ever_married
Never_worked
Govt_job
Self-employed
children
Private
0%
10%
13%
13%
64%
Private
children
Self-employed
Govt_job
Never_worked
9752
2038
1939
1533
42
work_type
Rural
Urban
50%
50%
Rural
Urban
7664
7640
Residence_type
smokes
never smoked
formerly smoked
Unknown
14%
41%
15%
30%
never smoked
Unknown
formerly smoked
smokes
6281
4543
2337
2143
smoking_status
1
0
2%
98%
stroke
0
14672
count

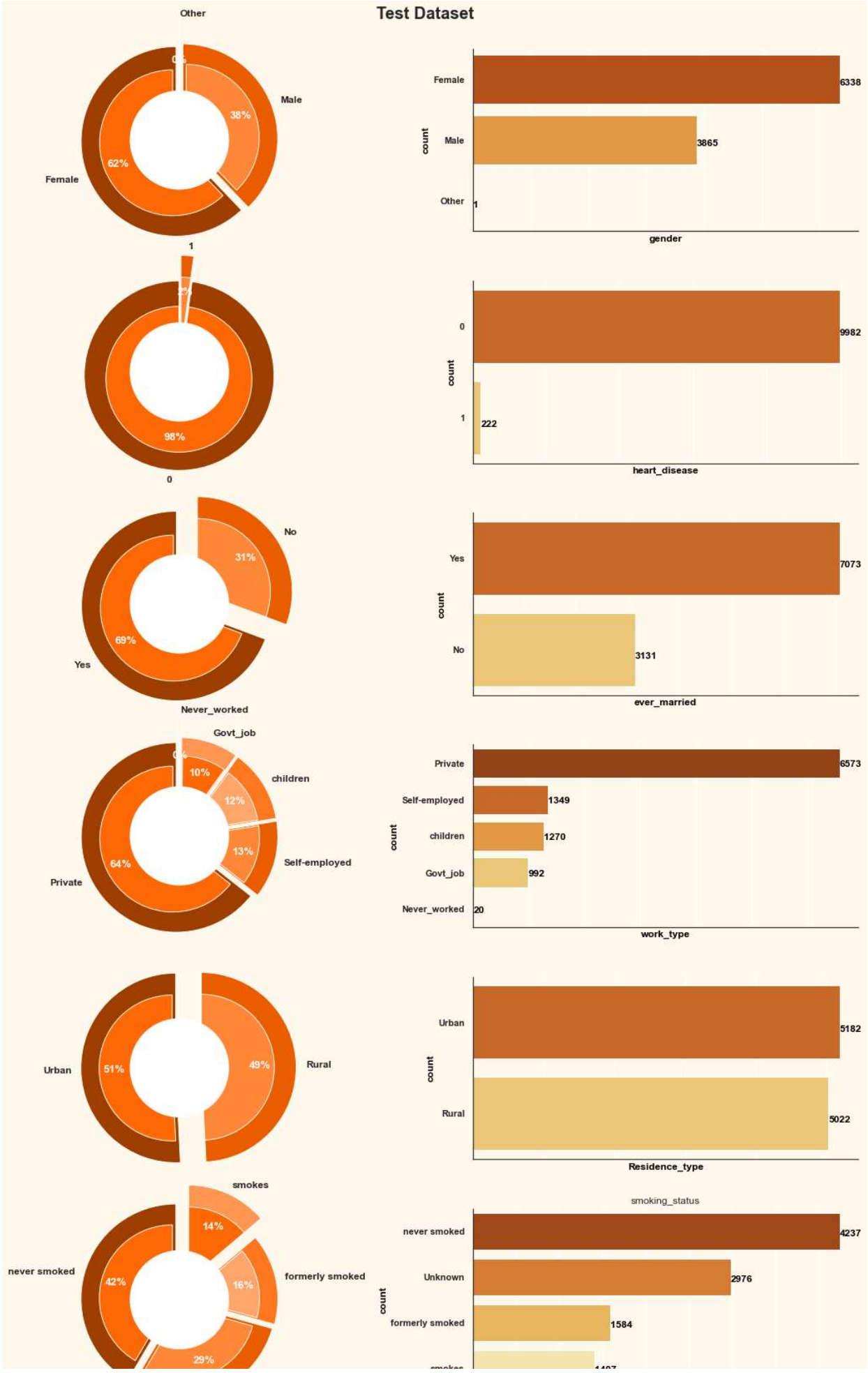


Initial observations:

1. There are more females than males (62% vs 38%) in the dataset.
2. Those who have heart disease are only 2% of the data.
3. ~2/3 of the population was married at some point of their life or still married.
4. 64% of the population are privately employed.
5. Residence type is perfectly balanced (city people vs 'villagers' 50% - 50%).
6. 29% People ever smoked, 30% is unsure about it, and 41% pretend they never.
7. There is 4% of strokes in the dataset where heart disease is only 2%. That is interesting.

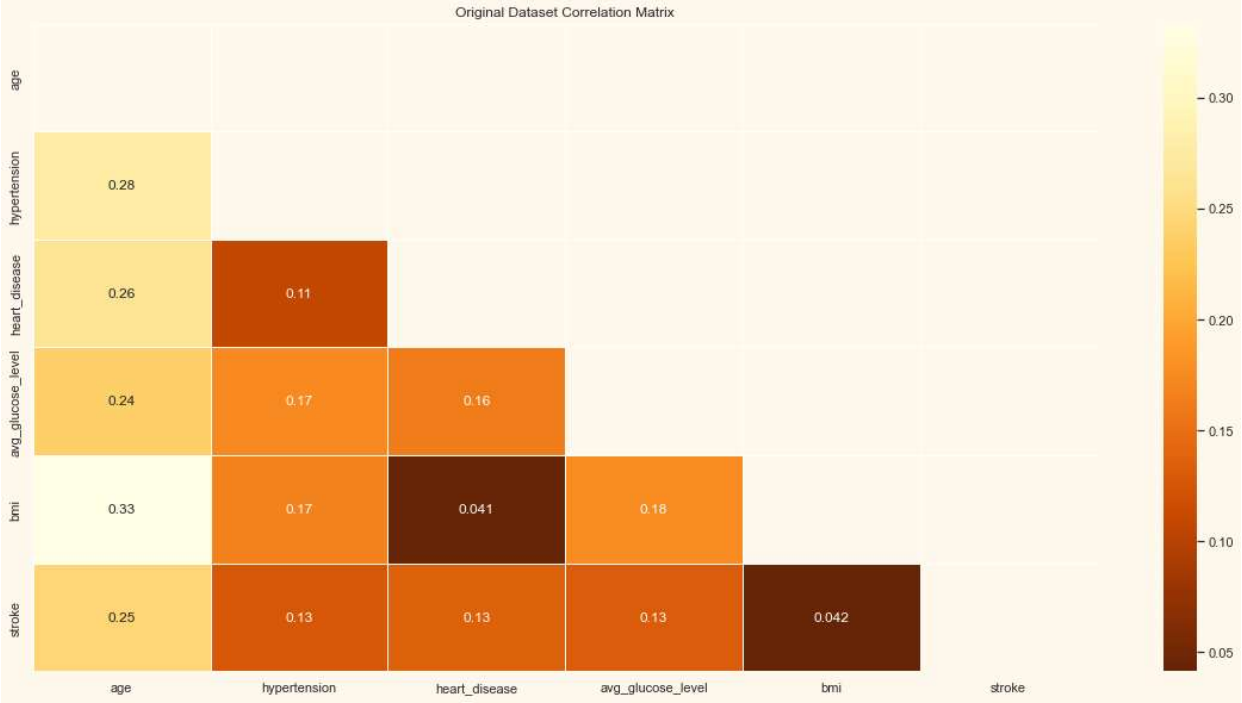
- p.s. we have class imbalance
- p.s.s. there is `1 other in the gender` self-identified as other, let it be so (2023 reality)

In [8]:
```
plot_count(test, selected_columns[:-1], title_name='Test')
```

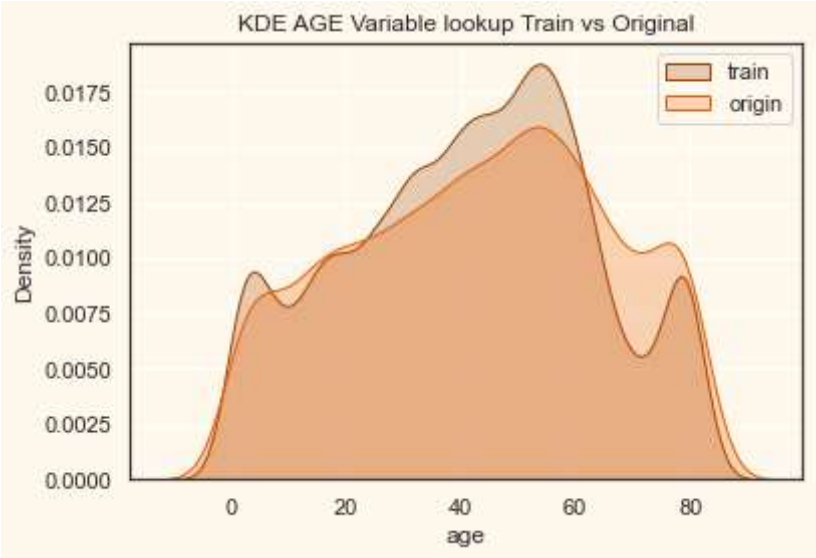

Test Dataset
Other
Male
38%
Female
62%
1
Female
Male
Other
6338
3865
1
count
gender
2%
98%
0
0
1
9982
222
count
heart_disease
No
31%
Yes
69%
Never_worked
Yes
No
7073
3131
count
ever_married
Govt_job
10%
children
12%
13%
Self-employed
Private
64%
Private
Self-employed
children
Govt_job
Never_worked
6573
1349
1270
992
20
count
work_type
Urban
51%
49%
Rural
Urban
Rural
5182
5022
count
Residence_type
smokes
14%
never smoked
42%
16%
formerly smoked
29%
smoking_status
never smoked
Unknown
formerly smoked
4237
2976
1584
count

In [13]:
```
train['is_original'] = 0
origin['is_original'] = 1
train_original_combined = pd.concat([train, origin]).reset_index(drop=True)
pair_plot(train_original_combined, title_name='Train vs Original Dataset', hue='is_ori
```

In [14]:
```
plot_correlation_heatmap(origin.drop(columns=['is_original']), title_name='Original Da
```



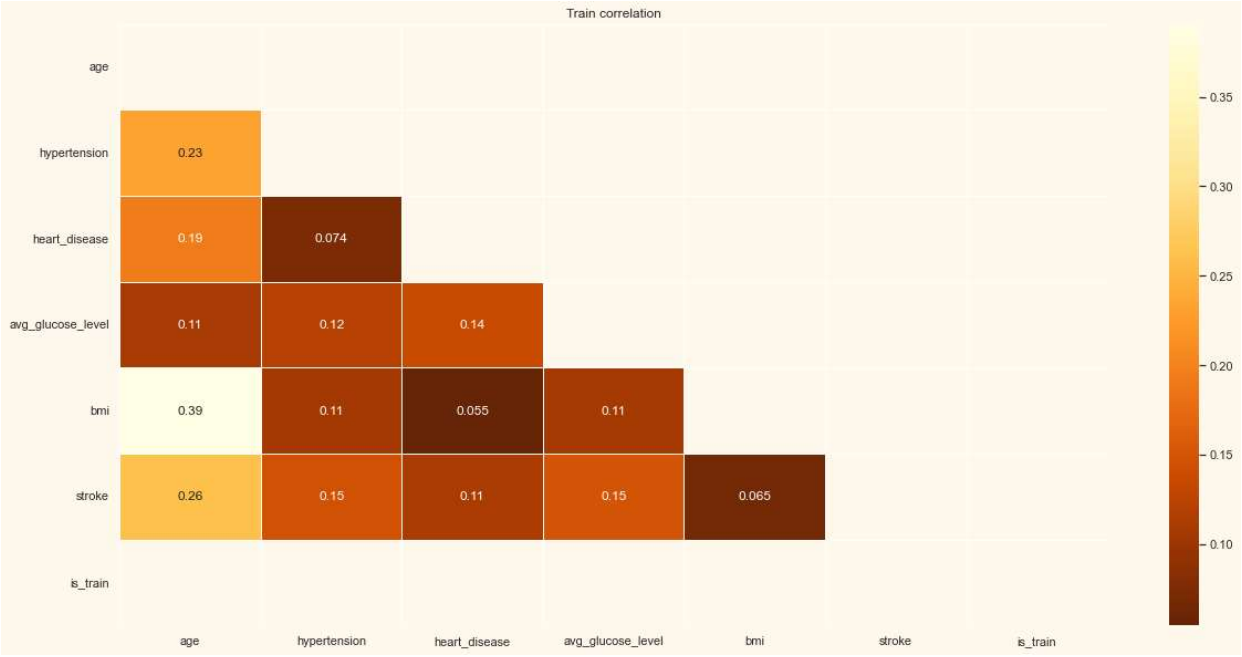
In [15]:
```
sns.kdeplot(train.age, color='#9E3F00', shade='True', label='train')
sns.kdeplot(origin.age, color='#eb5e00', shade=True, label='origin')
plt.legend()
plt.title('KDE AGE Variable lookup Train vs Original')
plt.show()
```

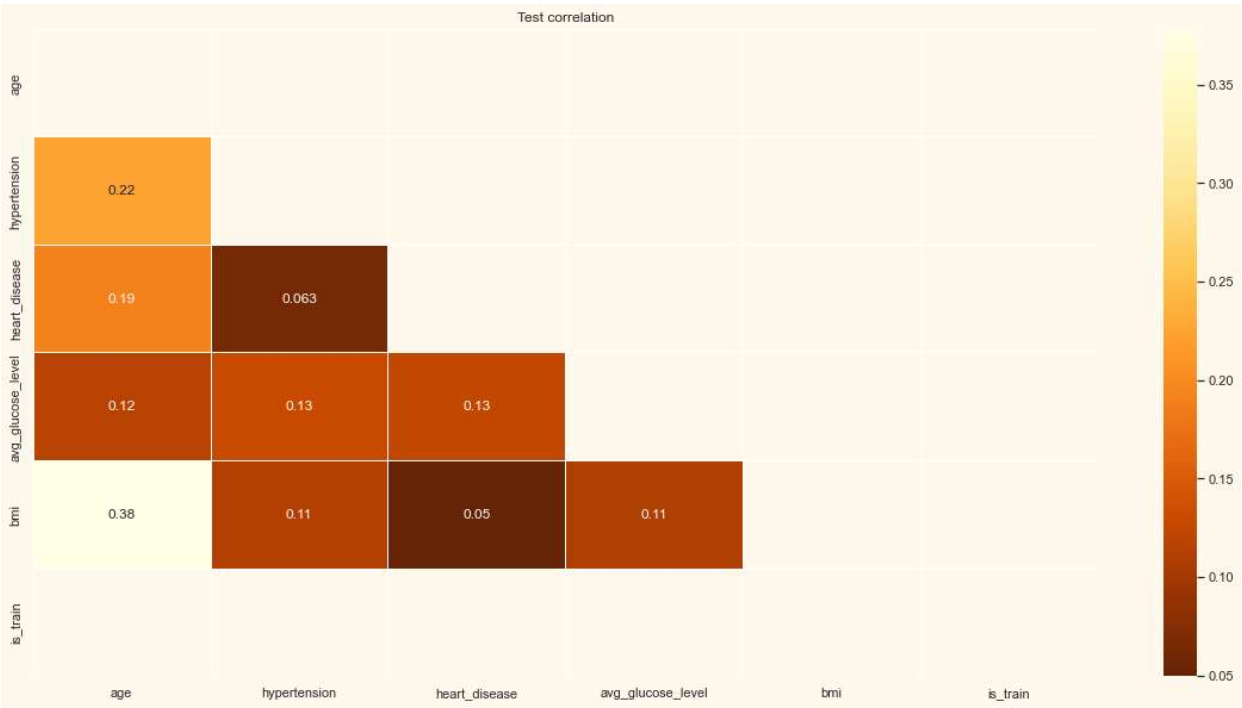


Pairplots notes:

1. Stroke becomes a major risk after ~age of 30. Older the person is higher the risk.
2. `bmi` for `stroke` is between ~20-60 range. Google says that repeated studies estimate that each unit increase in body mass index (BMI) increases the risk of stroke by 5 percent.
3. Strokes are more likely with high average glucose level.
4. Test dataset follows train dataset distribution almost perfectly.

In [10]:
```
plot_correlation_heatmap(train)
plot_correlation_heatmap(test, 'Test correlation')
```





Correlation plot notes:

1. The correlation between numeric features is weak.
2. `bmi` and `age` somewhat correlate. `bmi` grows with age.
3. `stroke` somewhat correlates with `BMI` and `age` which seems logical but still the correlation is weak.

Pytorch Model

In [20]:
```
# CFG
config = {
    'seed': 15,
    'num_cls': 1,
```

```
    'num_folds': 10,
    'lr': 9.8e-3,
    'wd': 1e-4,
    'plateau_factor': .5,
    'plateau_patience': 4,
    'batch': 256,
    'epochs': 2,
    'early_stopping': 9,
    'sample_size': 0
}

######################################################################
# Custom Dataset
######################################################################

class TrainDataset(Dataset):
    def __init__(self, x, y):
        """
        Defines PyTorch dataset.
        :param x: torch.Tensor
        :param y: torch.Tensor
        """
        self.x = torch.Tensor(x)
        self.y = torch.Tensor(y).unsqueeze(1)

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

    def __len__(self):
        return self.x.shape[0]


######################################################################
# Model
######################################################################

class Model(nn.Module):
    def __init__(self, in_features, num_cls):
        super().__init__()
        self.hidden_size = [2048, 1024, 512, 256]
        self.dropout_value = [0.25, 0.25, 0.25, 0.25]

        self.bn1 = nn.BatchNorm1d(in_features)
        self.fc1 = nn.Linear(in_features, self.hidden_size[0])

        self.bn2 = nn.BatchNorm1d(self.hidden_size[0])
        self.dropout2 = nn.Dropout(self.dropout_value[0])
        self.fc2 = nn.Linear(self.hidden_size[0], self.hidden_size[1])

        self.bn3 = nn.BatchNorm1d(self.hidden_size[1])
        self.dropout3 = nn.Dropout(self.dropout_value[1])
        self.fc3 = nn.Linear(self.hidden_size[1], self.hidden_size[2])

        self.bn4 = nn.BatchNorm1d(self.hidden_size[2])
        self.dropout4 = nn.Dropout(self.dropout_value[2])
        self.fc4 = nn.Linear(self.hidden_size[2], self.hidden_size[3])

        self.bn5 = nn.BatchNorm1d(self.hidden_size[3])
        self.dropout5 = nn.Dropout(self.dropout_value[3])
        self.fc5 = nn.utils.weight_norm(nn.Linear(self.hidden_size[3], num_cls))
```

```
    def forward(self, x):
        x = self.bn1(x)
        x = F.relu(self.fc1(x))

        x = self.bn2(x)
        x = self.dropout2(x)
        x = F.relu(self.fc2(x))

        x = self.bn3(x)
        x = self.dropout3(x)
        x = F.relu(self.fc3(x))

        x = self.bn4(x)
        x = self.dropout4(x)
        x = F.relu(self.fc4(x))

        x = self.bn5(x)
        x = self.dropout5(x)
        x = self.fc5(x)
        return x

######################################################################
# Early Stopping, init_weights, seed
######################################################################

class EarlyStopper(object):
    def __init__(self, patience: int):
        self.wait_counter = 0
        self.val_loss = 0
        self.val_loss_best = 1
        self.patience = patience

    def update(self, valid_loss, model_state_dict, fold):
        self.val_loss = valid_loss
        if self.val_loss < self.val_loss_best:
            self.val_loss_best = self.val_loss
            self.wait_counter = 0
#             torch.save(model_state_dict, f'./model/best_fold{fold + 1}.pth')
            print(f'\n[INFO] The best model has been saved.\n')
        else:
            self.wait_counter += 1
            if self.wait_counter > self.patience:
#                 torch.save(model_state_dict, f'./model/last_fold{fold + 1}.pth')
                print(f"\n[INFO] There's been no improvement "
                      f"in val_loss. Early stopping has been invoked.")
                return 'EarlyStop'

def init_weights(layer):
    if isinstance(layer, nn.Linear):
        nn.init.xavier_normal_(layer.weight.data)


def set_seed(seed):
    """
    Set a seed for the result reproducibility.
    """
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
```

```
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

######################################################################
# Custom Loops
######################################################################

def train_loop(train_loader, model, criterion, optimizer, epoch, device, fold):
    model.train()
    stream = tqdm(train_loader)
    loss_cumsum = 0
    for i, (x, y) in enumerate(stream, start=1):
        output = model(x)
        loss = criterion(output, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_cumsum += loss.item()
        desc = "Fold: {fold}. Epoch: {epoch}. Train. {loss:.4f}"
        stream.set_description(
            desc.format(fold=fold+1, epoch=epoch, loss=loss_cumsum/(i+1))
        )
    loss_avg = loss_cumsum/len(train_loader)
    return loss_avg


def valid_loop(val_loader, model, criterion, epoch, device, fold):
    model.eval()
    stream = tqdm(val_loader)
    loss_cumsum = 0
    with torch.no_grad():
        for i, (x, y) in enumerate(stream, start=1):
            output = model(x)
            loss = criterion(output, y)
            loss_cumsum += loss.item()
            desc = "Fold: {fold}. Epoch: {epoch}. Valid. {loss:.4f}"
            stream.set_description(
                desc.format(fold=fold+1, epoch=epoch, loss=loss_cumsum/(i+1))
            )
    loss_avg = loss_cumsum/len(train_loader)
    return loss_avg
```

Training

In [21]:

```
X = pd.get_dummies(train.drop(columns=['is_train', 'stroke']), drop_first=True).to_num
y = train.stroke.to_numpy()

print(f'[INFO] Train: {X.shape}, Test: {y.shape}')
skf = StratifiedKFold(n_splits=config['num_folds'], shuffle=True, random_state=config[
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

for fold, (idx_train, idx_val) in enumerate(skf.split(X, y)):
    if fold == 1: break

    # Seed, model and weights.
    set_seed(config['seed'])
    model = Model(in_features=X.shape[1], num_cls=config['num_cls'])
```

```
    model.apply(init_weights)
    model = model.to(device)

    # Loss, optimizer and scheduler.
    loss_tr = nn.BCEWithLogitsLoss()
    loss_vl = nn.BCEWithLogitsLoss()

    optimizer = optim.Adam(model.parameters(), lr=config['lr'], weight_decay=config['w
    scheduler = ReduceLROnPlateau(optimizer=optimizer, factor=config['plateau_factor']
                                  patience=config['plateau_patience'], mode='max', ver

    X_train, y_train = X[idx_train, :], y[idx_train]
    X_valid, y_valid = X[idx_val, :], y[idx_val]
    print(f'[INFO]: X_tr: {X_train.shape}, X_val: {X_valid.shape}.')

    train_loader = DataLoader(TrainDataset(X_train, y_train), batch_size=config['batch
    valid_loader = DataLoader(TrainDataset(X_valid, y_valid), batch_size=config['batch
    early_stopper = EarlyStopper(patience=config['early_stopping'])
    del X_train, y_train, X_valid, y_valid

    for epoch in range(1, config['epochs'] + 1):
        train_loss = train_loop(train_loader, model, loss_tr, optimizer, epoch, device
        valid_loss = valid_loop(valid_loader, model, loss_vl, epoch, device, fold)
        scheduler.step(valid_loss)
        stop = early_stopper.update(valid_loss, model.state_dict(), fold)
        if stop:
            break
```

```
[INFO] Train: (15304, 16), Test: (15304,)
[INFO]: X_tr: (13773, 16), X_val: (1531, 16).
  0%|          | 0/54 [00:00<?, ?it/s]
  0%|          | 0/6 [00:00<?, ?it/s]
[INFO] The best model has been saved.

  0%|          | 0/54 [00:00<?, ?it/s]
  0%|          | 0/6 [00:00<?, ?it/s]
[INFO] The best model has been saved.
```

Inference

In [29]:

```
X_test = pd.get_dummies(test.drop(columns=['is_train']), drop_first=True).to_numpy()
test_loader = DataLoader(TrainDataset(x=X_test, y=np.ones((X_test.shape[0], 1))),
                         batch_size=config['batch'], shuffle=False)

list_ = []
with torch.no_grad():
    for x_batch, _ in tqdm(test_loader):
        x_batch = x_batch.to(device)
        y_test_pred = model(x_batch.float())
        y_test_pred = torch.sigmoid(y_test_pred).cpu().numpy()
        list_.extend(y_test_pred)
    preds = np.array(list_)


# Ensure 'preds' is a 1-dimensional array or list
preds = np.array(preds).flatten()  # Convert to 1D array if it's not already
```

```python
# Create an index (e.g., using range or any appropriate index)
index = range(len(preds))

# Create a DataFrame with an index and a 'stroke' column
Final Result = pd.DataFrame({'stroke': preds}, index=index)

# Save the DataFrame to a CSV file
Final Result.to_csv('Final Result.csv', index=False)

# Optionally, display the DataFrame
print(Final Result)
```

```
  0%|          | 0/40 [00:00<?, ?it/s]
        stroke
0     0.045887
1     0.114798
2     0.007156
3     0.027158
4     0.011949
...        ...
10199 0.009711
10200 0.026210
10201 0.006683
10202 0.009789
10203 0.006939

[10204 rows x 1 columns]
```