```
#%%

#import all needed packages and environments
import pandas as pd
import numpy as np
from missingpy import missforest
from sklearn.preprocessing import LabelEncoder
from pandas.api.types import is_string_dtype,is_numeric_dtype
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import
confusion_matrix,accuracy_score,matthews_corrcoef,f1_score,classification_rep
ort
from sklearn.externals import joblib
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.neighbors import DistanceMetric
from sklearn.datasets import make_blobs
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
import warnings
from sklearn.model_selection import train_test_split
warnings.simplefilter("ignore")


%matplotlib inline

#%%

#save my path
my_path ='/Users/chojoon/Downloads '

#%%

#load data
df = pd.read_csv(f'{my_path}/stroke.csv')
df.head(3)


#%%

#check shape of data
df.shape

#%%

#drop ID column
dataframe = df.drop('id', axis = 1)
dataframe.head()

#%%
```

```
#check data types
dataframe.info()

#%%

#check missing values
dataframe.isnull().sum

#%%

#check for missing values
null_vals = dataframe.isnull().sum()
null_vals = pd.DataFrame(null_vals)
null_vals.reset_index(inplace = True)
null_vals.columns = ["Feature","Percent missing"]
plt.figure(figsize = (8,6))
plt.xticks(rotation=45)
sns.barplot(x = "Feature",y ="Percent missing",data = null_vals)

#%%

# Imputation for BMI using mean
dataframe['bmi'] =
dataframe['bmi'].fillna(dataframe.groupby('stroke')['bmi'].transform('mean'))

#%%

#re-check for missing values
null_vals = dataframe.isnull().sum()
null_vals = pd.DataFrame(null_vals)
null_vals.reset_index(inplace = True)
null_vals.columns = ["Feature","Percent missing"]
plt.figure(figsize = (8,6))
plt.xticks(rotation=45)
sns.barplot(x = "Feature",y ="Percent missing",data = null_vals)

#%%

#count for stroked number and unstroked number, and the plot
sns.countplot(dataframe['stroke'])
dataframe['stroke'].value_counts()

#%%

#sum and visualize data
print("differentiate by hypertension")
print(dataframe.groupby('stroke')['hypertension'].value_counts())
print("\n")
print("differentiate by gender")
print(dataframe.groupby('stroke')['gender'].value_counts())
print("\n")
print("differentiate by heart disease")
print(dataframe.groupby('stroke')['heart_disease'].value_counts())
print("\n")
print("differentiate by ever married")
print(dataframe.groupby('stroke')['ever_married'].value_counts())
print("\n")
```

```
print("differentiate by residence type")
print(dataframe.groupby('stroke')['Residence_type'].value_counts())


#%%

#visualize data
fig, (a1,a2,a3) = plt.subplots(1,3, figsize = (16,6))

sns.countplot(x = 'hypertension', hue = 'stroke', data = dataframe, ax=a1)
sns.countplot(x = 'gender', hue = 'stroke', data = dataframe, ax=a2)
a2.set_ylabel("")
sns.countplot(x = 'heart_disease', hue = 'stroke', data = dataframe, ax = a3)
a3.set_ylabel("")


#%%

#plot for age and stroke
print(dataframe.groupby('stroke')['age'].mean())

g = sns.FacetGrid(data=dataframe, col='stroke')
g.map(sns.distplot, 'age')
plt.show()

#%%

#visualize data
sns.catplot(x="gender", y="age", hue='stroke', kind="box", data=dataframe);

#%%

#visualize data
print("differentiate by work type")
print(dataframe.groupby('stroke')['work_type'].value_counts())
print("\n")

print("differentiate by smoking status")
print(dataframe.groupby('stroke')['smoking_status'].value_counts())

#%%

#visualize data
fig, (a1, a2, a3, a4) = plt.subplots(1,4,figsize=(20,6))
sns.countplot(x='ever_married', hue='stroke', data=dataframe, ax=a1);
sns.countplot(x='work_type', hue='stroke', data=dataframe, ax=a2);
a2.set_ylabel("")
sns.countplot(x='Residence_type', hue='stroke', data=dataframe, ax=a3);
a3.set_ylabel("")
sns.countplot(x='smoking_status', hue='stroke', data=dataframe, ax=a4);
a4.set_ylabel("")

#%%

#visualize data
print(dataframe.groupby('stroke')['avg_glucose_level'].mean())
g = sns.FacetGrid(data=dataframe, col='stroke', height=5)
```

```
g.map(sns.distplot, 'avg_glucose_level')
plt.show()

#%%

#visualize data
print(dataframe.groupby('stroke')['bmi'].mean())

sns.catplot(x="stroke", y="bmi", kind="box", data=dataframe);

g = sns.FacetGrid(data=dataframe, col='stroke', height=5)
g.map(sns.distplot, 'bmi')
plt.show()

#%%

#lable encoding
le = LabelEncoder()

en_data = dataframe.apply(le.fit_transform)

en_data.head()

#%%

#split train test set

y = en_data['stroke']
x = en_data.drop('stroke', axis = 1)
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15,
random_state=80, stratify=y)



#%%

#model training
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier()
model_rf.fit(x_train, y_train)
y_pred = model_rf.predict(x_test)


#%%

#check feature importance
model_rf.feature_importances_

#%%

#standardization
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

#%%

from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

print("The Training Score of RandomForestClassifier is:
{:.3f}%".format(model_rf.score(x_train, y_train)*100))
print("\n----------------------------------------------------------------------
--\n")
print("The Confusion Matrix for RandomForestClassifier is:
\n{}\n".format(confusion_matrix(y_test, y_pred)))
print("\n----------------------------------------------------------------------
--\n")
print("The Classification report:
\n{}\n".format(classification_report(y_test, y_pred)))
print("\n----------------------------------------------------------------------
--\n")
print("The Accuracy Score of RandomForestClassifier is:
{}%".format(accuracy_score(y_test, y_pred)*100))

#%% md

# deep learning model

#%%

#Import necessary packages
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import seaborn as sns
import numpy as np
import pandas as pd
from keras.wrappers.scikit_learn import KerasClassifier
%matplotlib inline

#%%

# Build the deep neural network
model = Sequential()
model.add(Dense(15, input_dim=10, init='uniform', activation='relu'))
model.add(Dense(6, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))

#%%

# Compile the DNN

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

#%%
```

```
# Fit the DNN with train data

model.fit(x_train, y_train, validation_data=(x_test, y_test), nb_epoch=50,
batch_size=10)

#%%

# Evaluate the model
scores = model.evaluate(x_test, y_test)
print ("Accuracy: %.2f%%" %(scores[1]*100))

#%% md

# Ensembling

#%%

#read data
my_path = 'C:\\Users\\63336\\Desktop\\info 6105\\final'

#%%

df1 = pd.read_csv(f'{my_path}/stroke.csv',header = None)

#%%

#dataset split
y1 = en_data['stroke']
x1 = en_data.drop('stroke', axis = 1)
from sklearn.model_selection import train_test_split

(X_train1, X_test1, Y_train1, Y_test1) = train_test_split(x1, y1,
test_size=0.15, random_state=1)

#%%

(X_train1, X_val, Y_train1, Y_val) = train_test_split(X_train1, Y_train1,
test_size=0.15, random_state=1)

#%%

#set Validation prediction and Test prediction for DNN
val_pred1 = model.predict(X_val)
test_pred1 = model.predict(X_test1)
val_pred1 = pd.DataFrame(val_pred1)
test_pred1 = pd.DataFrame(test_pred1)
val_pred1.shape

#%%

#set Validation prediction and Test predction for RF
val_pred2 = model_rf.predict(X_val)
test_pred2 = model_rf.predict(X_test1)
val_pred2 = pd.DataFrame(val_pred2)
test_pred2 = pd.DataFrame(test_pred2)

#%%
```

```
#concatenate data
df_val= pd.concat([ val_pred1.reset_index(drop=True), val_pred2], axis = 1)
df_test = pd.concat([X_test1, test_pred1, test_pred2], axis = 1)
df_val.shape

#%%

#Blending by logisticRegression
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression()
model_lr.fit(df_val, Y_val)
#score = logisticRegr.score(df_val, Y_val)
#pinrt(score)
model_lr.score(df_val, Y_val)

#%%
```