data_analysis

October 17, 2023

[ ]:
```
# use this to install pyreadstat, which is used to read .sav files
!pip install pyreadstat
```

1 Data loading and process

[ ]:
```
import pyreadstat

# data path
file_path = 'cumulative_2022_v3_9.sav'

# read data
df, meta = pyreadstat.read_sav(file_path)

# df is a pandas DataFrame (most common data type for data analysis in Python)

# meta is a dict containing metadata, like column names, labels, missing␣
↪values, etc.
```

[ ]:
```
# check the shape of the DataFrame
print( df.shape)
print( 'there are', df.shape[0], 'samples(rows) and', df.shape[1],␣
↪'variables(columns) in the DataFrame\n\n')

# check the head sample of the DataFrame
print(df.head())
```

```
(68224, 1030)
there are 68224 samples(rows) and 1030 features(columns) in the DataFrame


                        Version    Year  VCF0006    VCF0006a  VCF0009x  \
0  ANES_CDF_VERSION:2022-Sep-16  1948.0   1001.0  19481001.0       1.0
1  ANES_CDF_VERSION:2022-Sep-16  1948.0   1002.0  19481002.0       1.0
2  ANES_CDF_VERSION:2022-Sep-16  1948.0   1003.0  19481003.0       1.0
3  ANES_CDF_VERSION:2022-Sep-16  1948.0   1004.0  19481004.0       1.0
4  ANES_CDF_VERSION:2022-Sep-16  1948.0   1005.0  19481005.0       1.0
```

```
   VCF0010x  VCF0011x  VCF0009y  VCF0010y  VCF0011y  …  \
0       1.0       1.0       1.0       1.0       1.0  …
1       1.0       1.0       1.0       1.0       1.0  …
2       1.0       1.0       1.0       1.0       1.0  …
3       1.0       1.0       1.0       1.0       1.0  …
4       1.0       1.0       1.0       1.0       1.0  …

   hardworking_Hispanics  hardworking_Asians  blackInfluence_lifePolitics  \
0                    NaN                 NaN                          NaN
1                    NaN                 NaN                          NaN
2                    NaN                 NaN                          NaN
3                    NaN                 NaN                          NaN
4                    NaN                 NaN                          NaN

   blackInfluence_Politics  VCF9277  VCF9278  sex_orientation  \
0                      NaN      NaN      NaN              NaN
1                      NaN      NaN      NaN              NaN
2                      NaN      NaN      NaN              NaN
3                      NaN      NaN      NaN              NaN
4                      NaN      NaN      NaN              NaN

   bisexalFamilyorFriends  have_healthInsurance  living_withFamily
0                     NaN                   NaN                NaN
1                     NaN                   NaN                NaN
2                     NaN                   NaN                NaN
3                     NaN                   NaN                NaN
4                     NaN                   NaN                NaN

[5 rows x 1030 columns]
```

[ ]:
```python
# check the column names in the DataFrame
print(df.columns)
```

```
Index(['Version', 'Year', 'VCF0006', 'VCF0006a', 'VCF0009x', 'VCF0010x',
       'VCF0011x', 'VCF0009y', 'VCF0010y', 'VCF0011y',
       …
       'hardworking_Hispanics', 'hardworking_Asians',
       'blackInfluence_lifePolitics', 'blackInfluence_Politics', 'VCF9277',
       'VCF9278', 'sex_orientation', 'bisexalFamilyorFriends',
       'have_healthInsurance', 'living_withFamily'],
      dtype='object', length=1030)
```

[ ]:
```python
# the column names in df are not very informative, let's check the detail␣
 ↪variable labels in meta
# we print it and also save it to a txt file
```

```
variable_labels = meta.column_labels
text_file = open("variable_labels.txt", "w")

# we also build a dictionary to map the variable labels in meta to column names␣
↪in df, which may make the feature indexing more conveniently

variable_to_column_dict = {}

for i in range(len(variable_labels)):
    # print(variable_labels[i])
    text_file.write(  variable_labels[i] + '(%s)'%df.columns[i]+ '\n')
    variable_to_column_dict[variable_labels[i]] = df.columns[i]

text_file.close()
```

[ ]:
```
# we further check the meaning(label) of the values for each variable
# similar to variable labels, we print it and also save it to a txt file

value_labels = meta.variable_value_labels
text_file = open("value_labels.txt", "w")

for key in value_labels.keys():
    # print(key, value_labels[key])
    text_file.write(key +'  '+ str( value_labels[key])+ '\n')

text_file.close()
```

[ ]:
```
# other information in meta

num_rows = meta.number_rows
num_cols = meta.number_columns
file_label = meta.file_label

print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_cols}")
print(f"File label: {file_label}\n")

# check the missing values in the DataFrame
print('number of missing values of each variable: ')
print(df.isnull().sum())
```

```
Number of rows: 68224
Number of columns: 1030
File label: None

number of missing values of each variable:
Version                        0
```

```
Year                            0
VCF0006                         0
VCF0006a                        0
VCF0009x                     8280
                              …
VCF9278                     52338
sex_orientation             47942
bisexalFamilyorFriends      48900
have_healthInsurance        42284
living_withFamily           47503
Length: 1030, dtype: int64
```

2 Model 1: regression

[ ]:
```python
# we try a simple multi-variates regression model

# for example, we use the "number_children", "Age_group " and "Family_income"␣
↪to predict "level_politicalInfo_Post"

used_variable = ['number_children', 'Age_group', 'Family_income']
target = 'level_politicalInfo_Post'

#  check the value labels of the used variables and target variable
for var in used_variable:
    print(var, value_labels[var],'\n')
print(target, value_labels[target])
```

```
number_children {0.0: '0. None', 1.0: '1. One', 2.0: '2. Two', 3.0: '3. Three',
4.0: '4. Four', 5.0: '5. Five', 6.0: '6. Six', 7.0: '7. Seven', 8.0: '8. Eight
or more', 9.0: '9. NA; no Pre IW; Panel (1992,1996,2002)'}

Age_group {0.0: '0. NA; DK; RF; no Pre IW', 1.0: '1. 17 - 24', 2.0: '2. 25 -
34', 3.0: '3. 35 - 44', 4.0: '4. 45 - 54', 5.0: '5. 55 - 64', 6.0: '6. 65 - 74',
7.0: '7. 75 - 99 and over (except 1954)'}

Family_income {0.0: '0. DK; NA; refused to answer; no Pre IW', 1.0: '1. 0 to 16
percentile', 2.0: '2. 17 to 33 percentile', 3.0: '3. 34 to 67 percentile', 4.0:
'4. 68 to 95 percentile', 5.0: '5. 96 to 100 percentile'}

level_politicalInfo_Post {0.0: '0. no Post IW; abbrev. Post IW (1984); web  mode
(2012,2016)', 1.0: '1. Very high', 2.0: '2. Fairly high', 3.0: '3. Average',
4.0: '4. Fairly low', 5.0: '5. Very low', 9.0: '9. NA'}
```

[ ]:
```python
# based on the value labels, we only keep the samples with valid values for the␣
↪used variables and target variable
```

```
# remove the missing values
df_used = df[used_variable + [target]].dropna()

# remove the samples with invalid values for the used variables and target␣
↪variable

# remove the samples whose 'number_children' is 9.0
df_used = df_used[df_used['number_children'] != 9.0]

# remove the samples whose 'Age_group' is 0.0
df_used = df_used[df_used['Age_group'] != 0.0]

# remove the samples whose 'Family_income' is 0.0
df_used = df_used[df_used['Family_income'] != 0.0]

# remove the samples whose 'level_politicalInfo_Post' is 0.0
df_used = df_used[df_used['level_politicalInfo_Post'] != 0.0]


print(df_used.shape)

# the summary of the used variables
print(df_used.describe())
```

```
(14598, 4)
       number_children     Age_group  Family_income  level_politicalInfo_Post
count     14598.000000  14598.000000   14598.000000              14598.000000
mean          0.812303      3.459378       2.899986                  3.007261
std           1.070755      1.733047       1.156044                  1.092152
min           0.000000      1.000000       1.000000                  1.000000
25%           0.000000      2.000000       2.000000                  2.000000
50%           0.000000      3.000000       3.000000                  3.000000
75%           2.000000      5.000000       4.000000                  4.000000
max           3.000000      7.000000       5.000000                  5.000000
```

[ ]:
```
# we use the sklearn package to build the regression model

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# split the data into training and testing sets (80% for training and 20% for␣
↪testing)
X_train, X_test, y_train, y_test = train_test_split(df_used[used_variable],␣
↪df_used[target], test_size=0.2, random_state=42)

# build the model
```

```
reg = LinearRegression().fit(X_train, y_train)

# make predictions
y_pred = reg.predict(X_test)

# check the performance of the model
print('Coefficients: \n', reg.coef_)

print('Mean squared error: %.2f'
        % mean_squared_error(y_test, y_pred))

print('Coefficient of determination: %.2f'
        % r2_score(y_test, y_pred))
```

```
Coefficients:
 [ 0.10196896 -0.04577132 -0.32319057]
Mean squared error: 1.08
Coefficient of determination: 0.13
```

3 Model 2: classification

[ ]:
```
# we try a simple classification model

# for example, we use the "number_children", "Age_group " and "Family_income"␣
↪to predict "VCF0302"-- Party Identification of Respondent- Initial Party ID␣
↪Response

used_variable = ['number_children', 'Age_group', 'Family_income']
target = 'VCF0302'


for var in used_variable:
    print(var, value_labels[var],'\n')
#  check the value lables of the target variable
print(target , value_labels[target])
```

```
number_children {0.0: '0. None', 1.0: '1. One', 2.0: '2. Two', 3.0: '3. Three',
4.0: '4. Four', 5.0: '5. Five', 6.0: '6. Six', 7.0: '7. Seven', 8.0: '8. Eight
or more', 9.0: '9. NA; no Pre IW; Panel (1992,1996,2002)'}

Age_group {0.0: '0. NA; DK; RF; no Pre IW', 1.0: '1. 17 - 24', 2.0: '2. 25 -
34', 3.0: '3. 35 - 44', 4.0: '4. 45 - 54', 5.0: '5. 55 - 64', 6.0: '6. 65 - 74',
7.0: '7. 75 - 99 and over (except 1954)'}

Family_income {0.0: '0. DK; NA; refused to answer; no Pre IW', 1.0: '1. 0 to 16
percentile', 2.0: '2. 17 to 33 percentile', 3.0: '3. 34 to 67 percentile', 4.0:
```

'4. 68 to 95 percentile', 5.0: '5. 96 to 100 percentile'}

VCF0302 {1.0: '1. Republican', 2.0: '2. Independent', 3.0: '3. No preference; none; neither', 4.0: '4. Other', 5.0: '5. Democrat', 8.0: '8. DK', 9.0: '9. NA; refused'}

[ ]:
```python
# we first drop samples with missing values,

df_used = df[used_variable + [target]].dropna()

# remove the samples with invalid values for the used variables and target␣
↪variable

# remove the samples whose 'number_children' is 9.0
df_used = df_used[df_used['number_children'] != 9.0]

# remove the samples whose 'Age_group' is 0.0
df_used = df_used[df_used['Age_group'] != 0.0]

# remove the samples whose 'Family_income' is 0.0
df_used = df_used[df_used['Family_income'] != 0.0]

#  then we only select the samples whose target value is {1.0: '1. Republican',␣
↪5.0: '5. Democrat'} (it can be a multi-class classification problem if we␣
↪select more than 2 values, but here we only select 2 values to make it a␣
↪binary classification problem))

df_used = df_used[df_used[target].isin([1.0, 5.0])]

# transform the target variable to binary values
df_used[target] = df_used[target].apply(lambda x: 1 if x == 1.0 else 0)

print(df_used.shape)

# the summary of the used variables
print(df_used.describe())
```

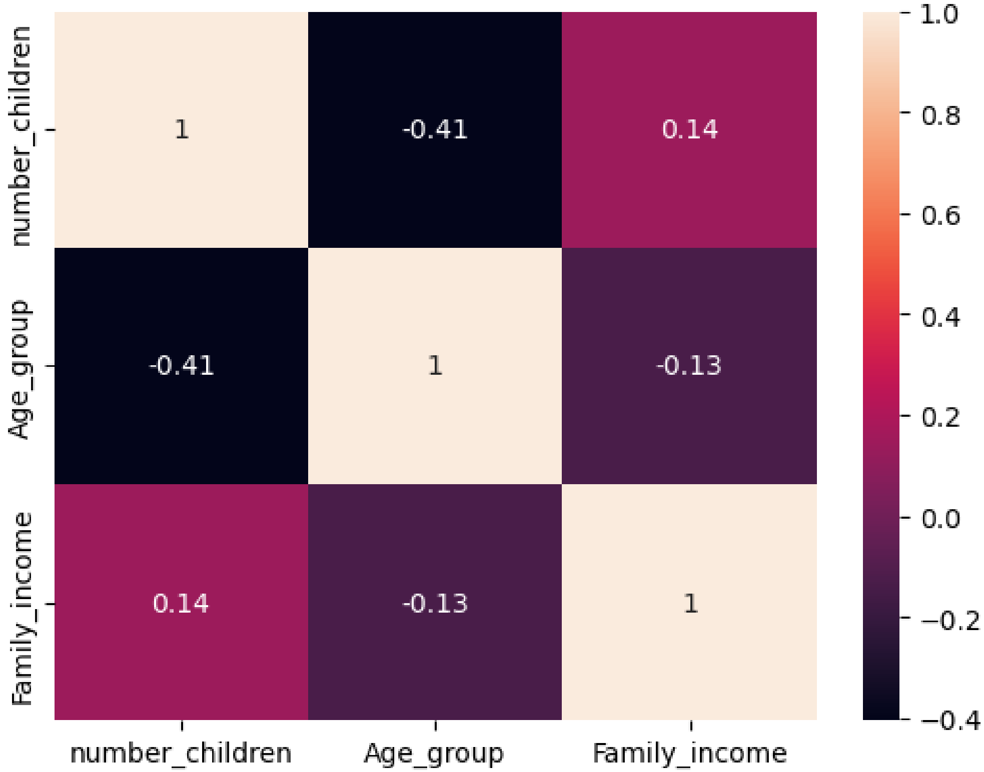
```
(10372, 4)
       number_children     Age_group  Family_income       VCF0302
count     10372.000000  10372.000000   10372.000000  10372.000000
mean          0.745083      3.670748       2.874566      0.386136
std           1.042256      1.765508       1.155694      0.486886
min           0.000000      1.000000       1.000000      0.000000
25%           0.000000      2.000000       2.000000      0.000000
50%           0.000000      3.000000       3.000000      0.000000
75%           1.000000      5.000000       4.000000      1.000000
max           3.000000      7.000000       5.000000      1.000000
```

[ ]:
```python
# we still use the sklearn package to build the classification model by␣
↪logistic regression

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
# use accuracy, F1 score, AUC as the performance metric
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# split the data into training and testing sets(80% for training and 20% for␣
↪testing)
X_train, X_test, y_train, y_test = train_test_split(df_used[used_variable],␣
↪df_used[target], test_size=0.2, random_state=42)

# build the model
clf = LogisticRegression(random_state=0).fit(X_train, y_train)

# make predictions
y_pred = clf.predict(X_test)

# check the performance of the model, report the accuracy
print('Coefficients: \n', clf.coef_)
print('Accuracy: %.2f'
        % accuracy_score(y_test, y_pred))
print('F1 score: %.2f'
        % f1_score(y_test, y_pred))
print('AUC: %.2f'
        % roc_auc_score(y_test, y_pred))
```

```
Coefficients:
 [[-0.07056391 -0.00963245  0.29203705]]
Accuracy: 0.63
F1 score: 0.15
AUC: 0.53
```

[ ]:
```python
# at last, we check the correlation between the three variables

corr_matrix = df[['number_children', 'Age_group', 'Family_income']].corr()
# visualize the correlation matrix

# install the seaborn and matplotlib packages first if you don't have them
# !pip install seaborn
# !pip install matplotlib

import seaborn as sn
import matplotlib.pyplot as plt

sn.heatmap(corr_matrix, annot=True)
```

```
[ ]: <Axes: >
```



4 Model 3: Deep Model with Pytorch

```python
[ ]: # we use pytorch to build a simple neural network to predict the target
     ↪variable based on the used variables

     # the setting is same to the classfication model above: use the
     ↪"number_children", "Age_group " and "Family_income" to predict "VCF0302"--
     ↪Party Identification of Respondent- Initial Party ID Response


     import torch
     import torch.nn as nn
     import torch.nn.functional as F
     import numpy as np

     # split the data into training and testing sets(80% for training and 20% for
     ↪testing)
```

```python
X_train, X_test, y_train, y_test = train_test_split(df_used[used_variable],␣
 ↪df_used[target], test_size=0.2, random_state=42)

# transform the data to torch tensor

X_train = torch.tensor(X_train.values).float()
X_test = torch.tensor(X_test.values).float()
y_train = torch.tensor(y_train.values).float()
y_test = torch.tensor(y_test.values).float()

# build the NN model, a simple fully-connected neural network with 3 hidden␣
 ↪layers
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(3, 10)
        self.fc2 = nn.Linear(10, 10)
        self.fc3 = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x

net = Net()

# define the loss function and optimizer
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)

# train the model

for epoch in range(1000):  # loop over the dataset multiple times

        optimizer.zero_grad()   # zero the gradient buffers

        outputs = net(X_train)
        loss = criterion(outputs.squeeze(), y_train)
        loss.backward()
        optimizer.step()

        if epoch % 100 == 0:
            print(f"Epoch {epoch} loss: {loss.item()}")

print('Finished Training')
```

```python
# make predictions
outputs = net(X_test)
y_pred = outputs.detach().numpy()
y_pred = np.where(y_pred > 0.5, 1, 0)

# check the performance of the model, report the accuracy
print('Accuracy: %.2f'
        % accuracy_score(y_test, y_pred))
print('F1 score: %.2f'
        % f1_score(y_test, y_pred))
print('AUC: %.2f'
        % roc_auc_score(y_test, y_pred))
```

```
Epoch 0 loss: 0.7286853194236755
Epoch 100 loss: 0.6707441806793213
Epoch 200 loss: 0.6615500450134277
Epoch 300 loss: 0.6577353477478027
Epoch 400 loss: 0.6546148657798767
Epoch 500 loss: 0.6523251533508301
Epoch 600 loss: 0.6512188911437988
Epoch 700 loss: 0.6507850885391235
Epoch 800 loss: 0.6505460143089294
Epoch 900 loss: 0.6503548622131348
Finished Training
Accuracy: 0.63
F1 score: 0.24
AUC: 0.55
```