Data Processing Training

Problem 1: Ordinal Encoding

```
# from sklearn.preprocessing import OrdinalEncoder

enc = OrdinalEncoder()

dataset[["Geography","Gender"]] = enc.fit_transform(dataset[["Geography","Gender"]])

print(dataset)
```

Problem 2: Column Transformer

I didn't know exactly, but I guess the previous values of X, may effect on result. so we create an numpy array from pure output of ct.fit_transform().

Imports

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```



```
np.set_printoptions(threshold=sys.maxsize)
```

Read data

```
dataset = pd.read_csv('./bp1.csv')
dataset = dataset.drop(['RowNumber', 'CustomerId', 'Surname'], axis = 1)
# check = dataset['EstimatedSalary'].isnull().values.any()
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:,10].values
# print(X)
print(dataset)
# print(check)
```

```
      CreditScore Geography  Gender  ...  IsActiveMember  EstimatedSalary  Exited
0             619    France  Female  ...               1        101348.88       1
1             608     Spain  Female  ...               1        112542.58       0
2             502    France  Female  ...               0        113931.57       1
3             699    France  Female  ...               0         93826.63       0
```

Imputation

```
# from sklearn.impute import SimpleImputer
# imp = SimpleImputer(missing_values=np.nan, strategy='mean')
# imp = imp.fit(X[:,:9])
# X[:,:9] = imp.transform(X[:,:9])
```

Encoding

```
# from sklearn.preprocessing import OrdinalEncoder
# enc = OrdinalEncoder()
# dataset[["Geography","Gender"]] = enc.fit_transform(dataset[["Geography","Gender"]])
# print(dataset)
```

```
      CreditScore  Geography  Gender  ...  IsActiveMember  EstimatedSalary  Exited
0             619        0.0     0.0  ...               1        101348.88       1
1             608        2.0     0.0  ...               1        112542.58       0
2             502        0.0     0.0  ...               0        113931.57       1
3             699        0.0     0.0  ...               0         93826.63       0
4             850        2.0     0.0  ...               1         79084.10       0
...           ...        ...     ...  ...             ...              ...     ...
9995          771        0.0     1.0  ...               0         96270.64       0
                                1.0  ...               1        101699.77       0
                                0.0  ...               1         42085.58       1
                                1.0  ...               0         92888.52       1
9999          792        0.0     0.0  ...               0         38190.78       0

[10000 rows x 11 columns]
```

Saved successfully!

```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
enc = enc.fit(X[:,1])
X[:, 1] = enc.transform(X[:, 1])

enc = enc.fit(X[:,2])
X[:, 2] = enc.transform(X[:, 2])
print(X[:10, :])
```

```
[[619 0 0 42 2 0.0 1 1 1 101348.88]
 [608 2 0 41 1 83807.86 1 0 1 112542.58]
 [502 0 0 42 8 159660.8 3 1 0 113931.57]
 [699 0 0 39 1 0.0 2 0 0 93826.63]
 [850 2 0 43 2 125510.82 1 1 1 79084.1]
 [645 2 1 44 8 113755.78 2 1 0 149756.71]
 [822 0 1 50 7 0.0 2 1 1 10062.8]
 [376 1 0 29 4 115046.74 4 1 0 119346.88]
 [501 0 1 44 4 142051.07 2 0 1 74940.5]
 [684 0 1 27 2 134603.88 1 1 1 71725.73]]
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("encoder", OneHotEncoder(), [1,2,7,8])], remainder='passthrough')
X = np.array(ct.fit_transform(X), dtype=np.str)
print(X[:10, :])
```

```
[['1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '619' '42' '2'
  '0.0' '1' '101348.88']
 ['0.0' '0.0' '1.0' '1.0' '0.0' '1.0' '0.0' '0.0' '1.0' '608' '41' '1'
  '83807.86' '1' '112542.58']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '502' '42' '8'
  '159660.8' '3' '113931.57']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '0.0' '699' '39' '1'
  '0.0' '2' '93826.63']
 ['0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '850' '43' '2'
  '125510.82' '1' '79084.1']
 ['0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '1.0' '0.0' '645' '44' '8'
  '113755.78' '2' '149756.71']
 ['1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '822' '50' '7'
  '0.0' '2' '10062.8']
 ['0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '376' '29' '4'
  '115046.74' '4' '119346.88']
 ['1.0' '0.0' '0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '501' '44' '4'
  '142051.07' '2' '74940.5']
 ['1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '684' '27' '2'
  '134603.88' '1' '71725.73']]
```

split train and test

Saved successfully!

```
train_test_split
train_test_split(X, Y, test_size = 0.20)
print(X_train[:10, :])
```

```
[['0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '743' '36' '8'
  '92716.96' '1' '33693.78']
 ['0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '1.0' '0.0' '800' '38' '1'
  '0.0' '2' '51553.43']
 ['0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '636' '31' '9'
  '80844.69' '2' '74641.9']
 ['0.0' '0.0' '1.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '553' '39' '1'
  '142876.98' '2' '44363.42']
 ['1.0' '0.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '1.0' '611' '30' '9'
  '0.0' '2' '148887.69']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '711' '39' '3'
  '152462.79' '1' '90305.97']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '1.0' '0.0' '449' '37' '6'
  '0.0' '2' '82176.48']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '656' '48' '9'
  '0.0' '2' '85240.61']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '670' '45' '5'
  '47884.92' '1' '54340.24']
 ['1.0' '0.0' '0.0' '1.0' '0.0' '1.0' '0.0' '0.0' '1.0' '772' '35' '9'
  '0.0' '1' '25448.31']]
```

Standardizing

```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.fit_transform(X_train)
print(X_train[:10, :])
```

```
[[-0.99600797 -0.58119931  1.73436329  1.08948952 -1.08948952 -0.64589732
   0.64589732 -0.96801137  0.96801137  0.9624188  -0.27763781  1.02846547
   0.25440441 -0.90722244 -1.16841886]
 [-0.99600797 -0.58119931  1.73436329 -0.91786105  0.91786105 -0.64589732
   0.64589732  1.03304572 -1.03304572  1.55297499 -0.08760509 -1.40721356
  -1.23252086  0.81589711 -0.8574222 ]
 [-0.99600797  1.72058015 -0.57658047  1.08948952 -1.08948952 -0.64589732
   0.64589732 -0.96801137  0.96801137 -0.14616913 -0.75271962  1.37641962
   0.06400582  0.81589711 -0.45537417]
 [-0.99600797 -0.58119931  1.73436329  1.08948952 -1.08948952 -0.64589732
   0.64589732  1.03304572 -1.03304572 -1.00610183  0.00741128 -1.40721356
   1.0588333   0.81589711 -0.98262449]
 [ 1.00400803 -0.58119931 -0.57658047 -0.91786105  0.91786105 -0.64589732
   0.64589732 -0.96801137  0.96801137 -0.405185   -0.84773598  1.37641962
  -1.23252086  0.81589711  0.83749511]
 [ 1.00400803 -0.58119931 -0.57658047  1.08948952 -1.08948952 -0.64589732
   0.64589732  1.03304572 -1.03304572  0.63087848  0.00741128 -0.71130527
   1.21256335 -0.90722244 -0.18260995]
 [ 1.00400803 -0.58119931 -0.57658047  1.08948952 -1.08948952 -0.64589732
   0.64589732  1.03304572 -1.03304572 -2.08360785 -0.18262145  0.33255718
  -1.23252086  0.81589711 -0.32417175]
 [ 1.00400803 -0.58119931 -0.57658047  1.08948952 -1.08948952 -0.64589732
   0.64589732 -0.96801137  0.96801137  0.06104357  0.86255854  1.37641962
  -1.23252086  0.81589711 -0.27081493]
 [ 1.00400803 -0.58119931 -0.57658047  1.08948952 -1.08948952 -0.64589732
   0.64589732 -0.96801137  0.96801137  0.20609245  0.57750945 -0.01539697
  -1.23252086  0.81589711 -0.0889445]
 [ 1.00400803 -0.58119931 -0.57658047  1.08948952 -1.08948952  1.5482337
  -1.5482337  -0.96801137  0.96801137  1.26287721 -0.37265417  1.37641962
  -1.23252086 -0.90722244 -1.31200026]]
```

Saved successfully!

Saved successfully!