**data-processing.py**

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from shutil import rmtree
from os.path import exists, isdir
from os import makedirs

DATA_NAME = "original_data.csv"
OUT_DATA_PREFIX = "/data"
PNN_OUTPUT_DIR = "./out-pnn"
PNN_REFERENCE_OUTPUT_DIR = PNN_OUTPUT_DIR + OUT_DATA_PREFIX
TRAIN_OUTPUT_DIR = "./out-train"
TRAIN_REFERENCE_OUTPUT_DIR = TRAIN_OUTPUT_DIR + OUT_DATA_PREFIX
TEST_OUTPUT_DIR = "./out-test"
TEST_REFERENCE_OUTPUT_DIR = TEST_OUTPUT_DIR + OUT_DATA_PREFIX

PNN_REFERENCE_COUNT_BY_CLASS = {
    "smurf":           0.6,
    "neptune":         0.6,
    "normal":          0.6,
    "back":            1.0,
    "satan":           1.0,
    "ipsweep":         1.0,
    "portsweep":       1.0,
    "warezclient":     1.0,
    "teardrop":        1.0,
    "pod":             1.0,
    "nmap":            1.0,
    "guess_passwd":    1.0,
    "buffer_overflow": 1.0,
    "land":            1.0,
    "warezmaster":     1.0,
    "imap":            1.0,
    "rootkit":         1.0,
    "loadmodule":      1.0,
    "ftp_write":       1.0,
    "multihop":        1.0,
    "phf":             1.0,
    "perl":            1.0,
    "spy":             1.0
}
TEST_SIZE = 0.25
REAL_DATA_COUNT_PER_CLASS = 8000
RANDOM_STATE = 1450

# Read data
data = pd.read_csv(DATA_NAME)

# Show column info
print(">>> Original data >>>")
for column in data.columns:
    unique_values_count = data[column].nunique()
```

```
    unique_values = data[column].unique()
    print(f"Column: {column}")
    print(f"Unique Values Count: {unique_values_count}")
    print(f"Unique Values: {unique_values}")
    print()
print(">>> Column types >>>\n", data.dtypes, "\n")

# Show unique classes and their count
print(">>> Class | Count >>>")
print(data["label"].value_counts(), "\n")

# Drop lnum_outbound_cmds and is_host_login columns since values are same for all rows
data = data.drop(columns = ["lnum_outbound_cmds", "is_host_login"])

# Convert protocol_type, service, flag columns to numeric
nonnumeric_columns = ["protocol_type", "service", "flag"]
label_encoder = LabelEncoder()
for column in nonnumeric_columns:
    data[column] = label_encoder.fit_transform(data[column])

# normalize data
for column_name in data.columns:
    if column_name != 'label':
        column = data[column_name]
        min_val = column.min()
        max_val = column.max()
        data[column_name] = (column - min_val) / (max_val - min_val)

# Show column info
print(">>> Data: removed useless columns, all numeric >>>")
for column in data.columns:
    unique_values_count = data[column].nunique()
    unique_values = data[column].unique()
    print(f"Column: {column}")
    print(f"Unique Values Count: {unique_values_count}")
    print(f"Unique Values: {unique_values}")
    print()
print(">>> Column types >>>\n", data.dtypes, "\n")

# Show unique classes and their count
print(">>> Class | Count >>>")
print(data["label"].value_counts(), "\n")

# Delete old run data and recreate out directory tree
if exists(PNN_OUTPUT_DIR):
    rmtree(PNN_OUTPUT_DIR)
makedirs(PNN_REFERENCE_OUTPUT_DIR)
if exists(TRAIN_OUTPUT_DIR):
    rmtree(TRAIN_OUTPUT_DIR)
makedirs(TRAIN_REFERENCE_OUTPUT_DIR)
if exists(TEST_OUTPUT_DIR):
    rmtree(TEST_OUTPUT_DIR)
makedirs(TEST_REFERENCE_OUTPUT_DIR)

# data export
property_count = len(data.columns.tolist()) - 1
```

```
data.insert(0, "property_count", property_count)
rows_by_class = dict(tuple(data.groupby("label")))
count_by_class = { "pnn": 0, "train" : 0, "test": 0 }
for k,k_data in rows_by_class.items():
    k_data.drop(columns = ["label"], inplace = True)
    if k_data.shape[0] > REAL_DATA_COUNT_PER_CLASS:
        # select real count from k_data
        k_data_real = k_data.sample(n = REAL_DATA_COUNT_PER_CLASS, random_state = RANDOM_STATE)

        # split
        k_data_train, k_data_test = train_test_split(k_data_real, test_size = TEST_SIZE, random_state = RANDOM_STATE)

        # take pnn data from train data
        k_data_pnn = k_data_train.sample(frac = PNN_REFERENCE_COUNT_BY_CLASS[k], random_state = RANDOM_STATE)

        # save data
        count_by_class["pnn"] += k_data_pnn.shape[0]
        k_data_pnn.to_csv(PNN_REFERENCE_OUTPUT_DIR + "/" + k + "-" + str(k_data_pnn.shape[0]) + ".csv",
                          header = False, index = True, lineterminator = ',')
        count_by_class["train"] += k_data_train.shape[0]
        k_data_train.to_csv(TRAIN_REFERENCE_OUTPUT_DIR + "/" + k + "-" + str(k_data_train.shape[0]) + ".csv",
                           header = False, index = True, lineterminator = ',')
        count_by_class["test"] += k_data_test.shape[0]
        k_data_test.to_csv(TEST_REFERENCE_OUTPUT_DIR + "/" + k + "-" + str(k_data_test.shape[0]) + ".csv",
                           header = False, index = True, lineterminator = ',')
    else:
        # save whole data as pnn, train, test
        count_by_class["pnn"] += k_data.shape[0]
        k_data.to_csv(PNN_REFERENCE_OUTPUT_DIR + "/" + k + "-" + str(k_data.shape[0]) + ".csv",
                          header = False, index = True, lineterminator = ',')
        count_by_class["train"] += k_data.shape[0]
        k_data.to_csv(TRAIN_REFERENCE_OUTPUT_DIR + "/" + k + "-" + str(k_data.shape[0]) + ".csv",
                           header = False, index = True, lineterminator = ',')
        count_by_class["test"] += k_data.shape[0]
        k_data.to_csv(TEST_REFERENCE_OUTPUT_DIR + "/" + k + "-" + str(k_data.shape[0]) + ".csv",
                           header = False, index = True, lineterminator = ',')

# save metadata
with open(PNN_OUTPUT_DIR + "/info.csv", "w", newline = "\n") as info:
    info.write(",".join([str(property_count),
                         str(count_by_class["pnn"]),
                         str(len(data["label"].unique()))]))
with open(TRAIN_OUTPUT_DIR + "/info.csv", "w", newline = "\n") as info:
    info.write(",".join([str(property_count),
                         str(count_by_class["train"]),
                         str(len(data["label"].unique()))]))
with open(TEST_OUTPUT_DIR + "/info.csv", "w", newline = "\n") as info:
    info.write(",".join([str(property_count),
                         str(count_by_class["test"]),
```

```
159                     str(len(data["label"].unique()))])
```