```
import pandas as pd
import numpy as np
import sys
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import StandardScaler, PowerTransformer
```

Load data

```
if 'google.colab' in sys.modules:
    from google.colab import drive
    drive.mount('/content/drive')
    DATA_PATH = '/content/drive/MyDrive/DL Project/data/'
else:
    DATA_PATH = './Documents/Classes/AML/proj/archive/'
```

```
Mounted at /content/drive
```

```
train = pd.read_feather(DATA_PATH + 'train_data.ftr')
train.head()
# test = pd.read_feather(DATA_PATH + 'test_data.ftr')
```

```
CAT_COL = ['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68']
```

Preprocess

Transform categorical data into numerical data (Laplace smoothing)

```
def cat_to_num(df):
    y_bar = df['target'].mean()

    def apply_laplace_smooth(df, feature_col, k=30):
        G = feature_col.value_counts()
        y_per_G = df.groupby(feature_col.name)['target'].sum()
        x_hat = (k * y_bar + y_per_G) / (k + G)
        return feature_col.map(x_hat)

    df[CAT_COL] = df[CAT_COL].apply(lambda f: apply_laplace_smooth(df, f)).astype(np.float16)

    return df
```

```
train = cat_to_num(train)
train.head()
```

	customer_ID	S_2	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	...	D_137	D_138
0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	0.938477	0.001734	0.008728	1.006836	0.009224	0.124023	0.008774	0.004707	...	NaN	NaN
1	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-04-07	0.936523	0.005775	0.004925	1.000977	0.006153	0.126709	0.000798	0.002714	...	NaN	NaN
2	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-05-28	0.954102	0.091492	0.021652	1.009766	0.006817	0.123962	0.007599	0.009422	...	NaN	NaN
3	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-06-13	0.960449	0.002455	0.013687	1.002930	0.001372	0.117188	0.000685	0.005531	...	NaN	NaN
4	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-07-16	0.947266	0.002483	0.015190	1.000977	0.007607	0.117310	0.004654	0.009308	...	NaN	NaN

5 rows × 191 columns

```
train.to_pickle(DATA_PATH + 'train_cat_transed.pkl')
```

Handle missing values

```
def impute_mean(df):
    imputer = SimpleImputer().set_output(transform='pandas')
```

```

    trans_df = imputer.fit_transform(df.drop(['customer_ID', 'S_2'], axis=1))
    imputed_df = pd.concat([df[['customer_ID', 'S_2']], trans_df], axis=1)

    assert sum(imputed_df.isna().any()) == 0

    return imputed_df
```

```
def impute_knn(df):
    imputer = KNNImputer(n_neighbors=3).set_output(transform='pandas')
    trans_df = imputer.fit_transform(df.drop(['customer_ID', 'S_2'], axis=1))
    imputed_df = pd.concat([df[['customer_ID', 'S_2']], trans_df], axis=1)

    assert sum(imputed_df.isna().any()) == 0

    return imputed_df
```

```
def impute_drop(df):
    # Get all columns that have NaN values
    nan_cols = df.columns[df.isna().any()].tolist()

    # Drop columns that have > 20% NaN values
    df = df.dropna(thresh=df.shape[0]*0.8, axis=1)

    df = df.dropna()
    df = df.reset_index(drop=True)

    # No NaN values at this point
    assert sum(imputed_df.isna().any()) == 0

    return df
```

```
def find_impute_val(df, feature_series):
    feature_name = feature_series.name
    bin_col_name = feature_name + "_bin"

    # 1. Calculate percentiles (with 10 bins) for each feature that has missing values
    df[bin_col_name] = pd.qcut(x=feature_series, q=10, labels=False, duplicates='drop')

    # 2. Calculate target label rates for each bin of each feature
    target_rate_per_bin = df.groupby(bin_col_name)['target'].mean()

    # 3. Determine the bin that the imputed missing values fall inside and assign the target label rate
    for bin_num in range(len(target_rate_per_bin)):
        bool_mask = (df[bin_col_name]==bin_num) & (train[feature_name].isna())
        feature_series.mask(bool_mask, target_rate_per_bin[bin_num])

    df.drop(bin_col_name, axis=1, inplace=True)

    return feature_series
```

```
def handle_missing_values(df, impute_fn):
    filled_df = impute_fn(df)

    nan_col = filled_df.columns[filled_df.isna().any()].tolist()
    filled_df[nan_col] = filled_df[nan_col].apply(lambda f: find_impute_val(df, f))

    assert sum(filled_df.isna().any()) == 0

    return filled_df
```

```
train = handle_missing_values(train, impute_mean)
train.head()
```

	customer_ID	S_2	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	...	D_137	D
0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-03-09	0.938477	0.001734	0.008728	1.006836	0.009224	0.124023	0.008774	0.004707	...	0.014244	0.16

```
train.to_pickle(DATA_PATH + 'train_nan_imputed.pkl')
```

Handle Outliers

3	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	[illegible]	0.960449	0.002455	0.013687	1.002930	0.001372	0.117188	0.000685	0.005531	...	0.014244	0.16

```
train = pd.read_pickle(DATA_PATH + 'train_nan_imputed.pkl')
```

4	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	[illegible]	0.947266	0.002483	0.015190	1.000977	0.007607	0.117310	0.004654	0.009308	...	0.014244	0.16

```
def compute_IQR(feature):
    Q1 = np.percentile(feature, 5, axis=0)
    Q3 = np.percentile(feature, 95, axis=0)
    IQR = Q3 - Q1
    thres = 1.5
    upper_bound = Q3 + thres * IQR
    lower_bound = Q1 - thres * IQR

    return feature.loc[(feature >= lower_bound) & (feature <= upper_bound)]
```

```
def handle_outliers(df, fn):
    trans_df = df.drop(['customer_ID', 'S_2', 'target'], axis=1).apply(lambda f: fn(f))
    return pd.concat([df[['customer_ID', 'S_2', 'target']], trans_df], axis=1).dropna()
```

```
train = handle_outliers(train, compute_IQR)
train.head()
```

	customer_ID	S_2	target	P_2	D_39	B_1	B_2	R_1	S_3	D_41	...	D_136	D_
5	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-08-04	0.0	0.945801	0.001746	0.007866	1.004883	0.004219	0.110962	0.009857	...	0.242772	0.014
6	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-09-18	0.0	0.940918	0.002182	0.018860	1.007812	0.004509	0.103333	0.006603	...	0.242772	0.014
7	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-10-08	0.0	0.914551	0.003029	0.014328	1.000000	0.000263	0.108093	0.009529	...	0.242772	0.014
8	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-11-20	0.0	0.950684	0.009895	0.016891	1.003906	0.001789	0.102783	0.002520	...	0.242772	0.014
9	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-12-04	0.0	0.868652	0.001082	0.001930	1.007812	0.001772	0.100464	0.004627	...	0.242772	0.014

5 rows × 191 columns

```
train.to_pickle(DATA_PATH + 'train_outlier_removed.pkl')
```

Reduce Skewness

```
# Already performed standardization
def reduce_skewness_yeo_johnson(df):
    pt = PowerTransformer().set_output(transform='pandas')
    trans_df = pt.fit_transform(train.drop(['customer_ID', 'S_2', 'target'], axis=1))
    return pd.concat([df[['customer_ID', 'S_2', 'target']], trans_df], axis=1)
```

```
train = reduce_skewness_yeo_johnson(train)
train.head()
```

```
/Users/roxyk/opt/anaconda3/lib/python3.9/site-packages/numpy/core/_methods.py:236: RuntimeWarning: overflow encountered in multiply
  x = um.multiply(x, x, out=x)
/Users/roxyk/opt/anaconda3/lib/python3.9/site-packages/numpy/core/_methods.py:247: RuntimeWarning: overflow encountered in reduce
  ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
```

	customer_ID	S_2	target	P_2	D_39	B_1	B_2	R_1	S_3	D_41	...	D_136
5	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2017-08-04	0.0	1.167299	-0.753275	-0.596688	1.050376	-0.200328	-0.899846	0.779986	...	-1.110223e-16
		2017-										-1.110223e-

```
train.to_pickle(DATA_PATH + 'train_skewness_reduced.pkl')
```

		2017-										-1.110223e-
		11-20										16

Normalization

```
# scaler = StandardScaler().set_output(transform='pandas')
# train = scaler.fit_transform(train)
```

Save processed data

```
train.to_pickle(DATA_PATH + 'train.pkl')
```

```
1
```