```python
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```python
import gc
import os
import logging
import datetime
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import lightgbm as lgb
from tqdm import tqdm_notebook
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.model_selection import StratifiedKFold
warnings.filterwarnings('ignore')
```

load data

```python
train = pd.read_csv('../input/santander-customer-transaction-
prediction-dataset/train.csv')
test = pd.read_csv('../input/santander-customer-transaction-
prediction-dataset/test.csv')
```

Data exploration

```
# preview
print(train.head())
print(test.head())
```

```
train.shape , test.shape
```

Train contains:

ID_code (string); target; 200 numerical variables, named from var_0 to var_199;

Test contains:

ID_code (string); 200 numerical variables, named from var_0 to var_199;

```
# check missing values and unique values
def missing_data(data):
    total = data.isnull().sum().sort_values(ascending = False)
    percent = 
(data.isnull().sum()/data.isnull().count()*100).sort_values(ascending 
= False)
    return pd.concat([total, percent], axis=1, keys=['Total', 
'Percent'])

def unique_values(data):
    total = data.count()
    tt = pd.DataFrame(total)
    tt.columns = ['Total']
    uniques = []
    for col in data.columns:
        unique = data[col].nunique()
        uniques.append(unique)
    tt['Uniques'] = uniques
    return tt
```

```
missing_data(train)
```

```
missing_data(test)
```

There are no missing data in train and test datasets. Let's check the numerical values in train and test dataset.

```
train.describe()
```

```
test.describe()
```

observations here:

standard deviation is relatively large for both train and test variable data;

min, max, mean, sdt values for train and test data looks quite close;

mean values are distributed over a large range.

```
sns.countplot(train['target'])
```

The target values are unbalanced.

Density plot of features

```
def plot_feature_distribution(df1, df2, label1, label2, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(10,10,figsize=(18,22))

    for feature in features:
        i += 1
        plt.subplot(10,10,i)
        sns.distplot(df1[feature], hist=False,label=label1)
        sns.distplot(df2[feature], hist=False,label=label2)
        plt.xlabel(feature, fontsize=9)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', which='major', labelsize=6, pad=-6)
        plt.tick_params(axis='y', which='major', labelsize=6)
    plt.show();
```

```
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
features = train.columns.values[2:102]
plot_feature_distribution(t0, t1, '0', '1', features)
```

```
features = train.columns.values[102:202]
plot_feature_distribution(t0, t1, '0', '1', features)
```

We can observe that there is a considerable number of features with significant different distribution for the two target values. For example, var_0, var_1, var_2, var_5, var_9, var_13, var_106, var_109, var_139 and many others.

Also some features, like var_2, var_13, var_26, var_55, var_175, var_184, var_196 shows a distribution that resambles to a bivariate distribution.

We will take this into consideration in the future for the selection of the features for our prediction model.

Le't s now look to the distribution of the same features in parallel in train and test datasets.

```
features = train.columns.values[2:102]
plot_feature_distribution(train, test, 'train', 'test', features)
```

```
features = train.columns.values[102:202]
plot_feature_distribution(train, test, 'train', 'test', features)
```

The train and test seems to be well ballanced with respect with distribution of the numeric variables.

Distribution of mean and std

```
# mean per rows
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of mean values per row in the train and test
set")
sns.distplot(train[features].mean(axis=1),color="green",
kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=1),color="blue",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# mean per colunms
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train and
test set")
sns.distplot(train[features].mean(axis=0),color="magenta",kde=True,bin
s=120, label='train')
sns.distplot(test[features].mean(axis=0),color="darkblue",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# std per rows
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of mean values per row in the train and test
set")
sns.distplot(train[features].std(axis=1),color="green",
kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=1),color="blue",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# std per columns
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of mean values per row in the train and test
set")
sns.distplot(train[features].std(axis=0),color="green",
kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=0),color="blue",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# mean per rows, group by target
t0 = train.loc[train['target'] == 0]
```

```
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per row in the train set")
sns.distplot(t0[features].mean(axis=1),color="red", kde=True,bins=120,
label='target = 0')
sns.distplot(t1[features].mean(axis=1),color="blue",
kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()

# mean per colums, group by target
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train set")
sns.distplot(t0[features].mean(axis=0),color="green",
kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].mean(axis=0),color="darkblue",
kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```

Distribution of min and max

```
# min per row
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of min values per row in the train and test
set")
sns.distplot(train[features].min(axis=1),color="red",
kde=True,bins=120, label='train')
sns.distplot(test[features].min(axis=1),color="orange",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# min per colunm
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of min values per row in the train and test
set")
sns.distplot(train[features].min(axis=0),color="red",
kde=True,bins=120, label='train')
sns.distplot(test[features].min(axis=0),color="orange",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# max per row
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of min values per row in the train and test
set")
sns.distplot(train[features].max(axis=1),color="red",
kde=True,bins=120, label='train')
```

```
sns.distplot(test[features].max(axis=1),color="orange",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# max per colunms
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of min values per row in the train and test
set")
sns.distplot(train[features].max(axis=0),color="red",
kde=True,bins=120, label='train')
sns.distplot(test[features].max(axis=0),color="orange",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# min value per row, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16, 6))
plt.title('Distribution of min values per row in the training set')
sns.distplot(t0[features].min(axis=1), color='orange', kde=True,
bins=120, label='Target=0')
sns.distplot(t1[features].min(axis=1), color='blue', kde=True,
bins=120, label='Target=1')
plt.show()

# min value per colunm, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16, 6))
plt.title('Distribution of min values per row in the training set')
sns.distplot(t0[features].min(axis=0), color='orange', kde=True,
bins=120, label='Target=0')
sns.distplot(t1[features].min(axis=0), color='blue', kde=True,
bins=120, label='Target=1')
plt.show()

# max value per row, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16, 6))
plt.title('Distribution of min values per row in the training set')
sns.distplot(t0[features].max(axis=1), color='orange', kde=True,
bins=120, label='Target=0')
sns.distplot(t1[features].max(axis=1), color='blue', kde=True,
bins=120, label='Target=1')
plt.show()
```

```
# max value per colunm, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16, 6))
plt.title('Distribution of min values per row in the training set')
sns.distplot(t0[features].min(axis=0), color='orange', kde=True,
bins=120, label='Target=0')
sns.distplot(t1[features].min(axis=0), color='blue', kde=True,
bins=120, label='Target=1')
plt.show()
```

Distribution of skew and kurtosis

```
# skew per row
plt.figure(figsize=(16,6))
plt.title("Distribution of skew per row in the train and test set")
sns.distplot(train[features].skew(axis=1),color="red",
kde=True,bins=120, label='train')
sns.distplot(test[features].skew(axis=1),color="orange",
kde=True,bins=120, label='test')
plt.legend()
plt.show()
```

```
# skew per column
plt.figure(figsize=(16,6))
plt.title("Distribution of skew per column in the train and test set")
sns.distplot(train[features].skew(axis=0),color="red",
kde=True,bins=120, label='train')
sns.distplot(test[features].skew(axis=0),color="orange",
kde=True,bins=120, label='test')
plt.legend()
plt.show()
```

kurtosis values per rows

```
plt.figure(figsize=(16,6))
plt.title("Distribution of kurtosis per row in the train and test
set")
sns.distplot(train[features].kurtosis(axis=1),color="darkblue",
kde=True,bins=120, label='train')
sns.distplot(test[features].kurtosis(axis=1),color="yellow",
kde=True,bins=120, label='test')
plt.legend()
plt.show()
```

```
# kurtosis values per columns
plt.figure(figsize=(16,6))
plt.title("Distribution of kurtosis per column in the train and test
set")
sns.distplot(train[features].kurtosis(axis=0),color="darkblue",
kde=True,bins=120, label='train')
```

```
sns.distplot(test[features].kurtosis(axis=0),color="yellow",
kde=True,bins=120, label='test')
plt.legend()
plt.show()

# skew values per row, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of skew values per row in the train set")
sns.distplot(t0[features].skew(axis=1),color="red", kde=True,bins=120,
label='target = 0')
sns.distplot(t1[features].skew(axis=1),color="blue",
kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()

# skew value per column, group by target
plt.figure(figsize=(16,6))
plt.title("Distribution of skew values per column in the train set")
sns.distplot(t0[features].skew(axis=0),color="red", kde=True,bins=120,
label='target = 0')
sns.distplot(t1[features].skew(axis=0),color="blue",
kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()

# kurtosis value per row, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of kurtosis values per row in the train set")
sns.distplot(t0[features].kurtosis(axis=1),color="red",
kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].kurtosis(axis=1),color="blue",
kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()

# kurtosis value per colunm, group by target
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
plt.figure(figsize=(16,6))
plt.title("Distribution of kurtosis values per row in the train set")
sns.distplot(t0[features].kurtosis(axis=0),color="red",
kde=True,bins=120, label='target = 0')
sns.distplot(t1[features].kurtosis(axis=0),color="blue",
kde=True,bins=120, label='target = 1')
plt.legend(); plt.show()
```

Feature correlation

```
correlations =
train[features].corr().abs().unstack().sort_values(kind="quicksort").r
```

```
eset_index()
correlations = correlations[correlations['level_0'] !=
correlations['level_1']]
# least 5 correlation
correlations.head(10)
```

```
# most 5 correlation
correlations.tail(10)
```

The correlation between the features is very small.

Check duplicated values per column

```
features = train.columns.values[2:202]
unique_max_train = []
unique_max_test = []
for feature in features:
    values = train[feature].value_counts()
    unique_max_train.append([feature, values.max(), values.idxmax()])
    values = test[feature].value_counts()
    unique_max_test.append([feature, values.max(), values.idxmax()])
```

```
# training set duplicated values
np.transpose((pd.DataFrame(unique_max_train, columns=['Feature', 'Max
duplicates', 'Value'])).\
            sort_values(by = 'Max duplicates',
ascending=False).head(15))
```

```
# testing set duplicated values
np.transpose((pd.DataFrame(unique_max_test, columns=['Feature', 'Max
duplicates', 'Value'])).\
            sort_values(by = 'Max duplicates',
ascending=False).head(15))
```

Same columns in train and test set have the same or very close number of duplicates of same or very close values. This is an interesting pattern that we might be able to use in the future.

Create some aggregation features

```
idx = features = train.columns.values[2:202]
for df in [test, train]:
    df['sum'] = df[idx].sum(axis=1)
    df['min'] = df[idx].min(axis=1)
    df['max'] = df[idx].max(axis=1)
    df['mean'] = df[idx].mean(axis=1)
    df['std'] = df[idx].std(axis=1)
    df['skew'] = df[idx].skew(axis=1)
    df['kurt'] = df[idx].kurtosis(axis=1)
    df['med'] = df[idx].median(axis=1)
```

```
train.head()
```

```
test.head()

def plot_new_feature_distribution(df1, df2, label1, label2, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(2,4,figsize=(18,8))

    for feature in features:
        i += 1
        plt.subplot(2,4,i)
        sns.kdeplot(df1[feature], bw=0.5,label=label1)
        sns.kdeplot(df2[feature], bw=0.5,label=label2)
        plt.xlabel(feature, fontsize=11)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', which='major', labelsize=8)
        plt.tick_params(axis='y', which='major', labelsize=8)
    plt.show();

t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
features = train.columns.values[202:]
plot_new_feature_distribution(t0, t1, 'target: 0', 'target: 1',
features)

features = train.columns.values[202:]
plot_new_feature_distribution(train, test, 'train', 'test', features)

print('Train and test columns: {} {}'.format(len(train.columns),
len(test.columns)))

param = {
    'bagging_freq': 5,
    'bagging_fraction': 0.4,
    'boost_from_average':'false',
    'boost': 'gbdt',
    'feature_fraction': 0.05,
    'learning_rate': 0.01,
    'max_depth': -1,
    'metric':'auc',
    'min_data_in_leaf': 80,
    'min_sum_hessian_in_leaf': 10.0,
    'num_leaves': 13,
    'num_threads': 8,
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1
}
features = [c for c in train.columns if c not in ['ID_code',
'target']]
target = train['target']
```

```
folds = StratifiedKFold(n_splits=10, shuffle=True, randomstate=1111)
oof = np.zeros(len(train))
predictions = np.zeros(len(test))
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values,
target.values)):
    print("Fold {}".format(fold_))
    trn_data = lgb.Dataset(train.iloc[trn_idx][features],
label=target.iloc[trn_idx])
    val_data = lgb.Dataset(train.iloc[val_idx][features],
label=target.iloc[val_idx])

    num_round = 1000000
    clf = lgb.train(param, trn_data, num_round, valid_sets =
[trn_data, val_data], verbose_eval=1000, early_stopping_rounds = 3000)
    oof[val_idx] = clf.predict(train.iloc[val_idx][features],
num_iteration=clf.best_iteration)

    fold_importance_df = pd.DataFrame()
    fold_importance_df["Feature"] = features
    fold_importance_df["importance"] = clf.feature_importance()
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df,
fold_importance_df], axis=0)

    predictions += clf.predict(test[features],
num_iteration=clf.best_iteration) / folds.n_splits

print("CV score: {:<8.5f}".format(roc_auc_score(target, oof)))

cols = (feature_importance_df[["Feature", "importance"]]
        .groupby("Feature")
        .mean()
        .sort_values(by="importance", ascending=False)[:150].index)
best_features =
feature_importance_df.loc[feature_importance_df.Feature.isin(cols)]

plt.figure(figsize=(14,28))
sns.barplot(x="importance", y="Feature",
data=best_features.sort_values(by="importance",ascending=False))
plt.title('Features importance (averaged/folds)')
plt.tight_layout()
plt.savefig('FI.png')

sub_df = pd.DataFrame({"ID_code":test["ID_code"].values})
sub_df["target"] = predictions
sub_df.to_csv("submission.csv", index=False)
```