Pandas

统计列中的个数

```
app_train['TARGET'].value_counts()
```

列中个数画柱状图

```
app_train['TARGET'].astype(int).plot.hist()
```

统计NA个数并按照从大到小输出

```
# Function to calculate missing values by column# Funct
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
```

统计数据类型

```
# Number of each type of column
app_train.dtypes.value_counts()
```

某个类别的unique统计

```
# Number of unique classes in each object column
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

LabelEncoder(sklearn.preprocessing)

只有两个变量

```
# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in app_train:
    if app_train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(app_train[col].unique())) <= 2:
            # Train on the training data
            le.fit(app_train[col])
            # Transform both training and testing data
            app_train[col] = le.transform(app_train[col])
            app_test[col] = le.transform(app_test[col])
```

```
            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```

One-hot encoding

```
# one-hot encoding of categorical variables
app_train = pd.get_dummies(app_train)
app_test = pd.get_dummies(app_test)

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

保持训练集和测试集col相同

```
train_labels = app_train['TARGET']

# Align the training and testing data, keep only columns present in both dataframes
app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

# Add the target back in
app_train['TARGET'] = train_labels

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

某列中值替换

```
app_train['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
```

计算correlation并排列

```
# Find correlations with the target and sort
correlations = app_train.corr()['TARGET'].sort_values()

# Display correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))
```

matplotlib画柱状图

```
# Set the style of plots
plt.style.use('fivethirtyeight')
#用plt.style.available可以找到所有的可用style
# Plot the distribution of ages in years
plt.hist(app_train['DAYS_BIRTH'] / 365, edgecolor = 'k', bins = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```

matplotlib+seaborn画密度图

```
plt.figure(figsize = (5, 4))

# KDE plot of loans that were repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, 'DAYS_BIRTH'] / 365, label = 'target == 0')

# KDE plot of loans which were not repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, 'DAYS_BIRTH'] / 365, label = 'target == 1')

# Labeling of plot
plt.xlabel('Age (years)'); plt.ylabel('Density'); plt.title('Distribution of Ages');
```

等间隔分组

```
# Age information into a separate dataframe
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
```

```
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365

# Bin the age data
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70, num = 11))
age_data.head(10)
```

groupby后会出现新的index，画柱状图

```
plt.figure(figsize = (4, 4))

# Graph the age bins and the average of the target as a bar plot
plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])

# Plot labeling
plt.xticks(rotation = 75); plt.xlabel('Age Group (years)'); plt.ylabel('Failure to Repay (%)')
plt.title('Failure to Repay by Age Group');
```

热力图

```
plt.figure(figsize = (5, 5))

# Heatmap of correlations
sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True, vmax = 0.6)
plt.title('Correlation Heatmap');
```

一图中放多张整合

```
plt.figure(figsize = (10, 12))

# iterate through the sources
for i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']):

    # create a new subplot for each source
    plt.subplot(3, 1, i + 1)
    # plot repaid loans

sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, source], label = 'target == 0')
    # plot loans that were not repaid

sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, source], label = 'target == 1')

    # Label the plots
     plt.title('Distribution of %s by Target Value' % source)
          plt.xlabel('%s' % source); plt.ylabel('Density');

plt.tight_layout(h_pad = 2.5)
```

grid图

```
# Create the pairgrid object
grid = sns.PairGrid(data = plot_data, size = 3, diag_sharey=False,
                    hue = 'TARGET',
                        vars = [x for x in list(plot_data.columns) if x != 'TARGET'])

# Upper is a scatter plot
grid.map_upper(plt.scatter, alpha = 0.1)

# Diagonal is a histogram
grid.map_diag(sns.kdeplot)

# Bottom is density plot
grid.map_lower(sns.kdeplot, cmap = plt.cm.OrRd_r);

plt.suptitle('Ext Source and Age Features Pairs Plot', size = 32, y = 1.05);
```

去除掉某一列

```
poly_features = poly_features.drop(columns = ['TARGET'])
```

Handling missing values(sklearn.preprocessing Imputer)

```
# imputer for handling missing values
from sklearn.preprocessing import Imputer
imputer = Imputer(strategy = 'median')
poly_target = poly_features['TARGET']
poly_features = poly_features.drop(columns
= ['TARGET'])
# Need to impute missing values
poly_features = imputer.fit_trans-
form(poly_features)
poly_features_test = imputer.trans-
form(poly_features_test)
```

Polynomial features

```
from sklearn.preprocessing import Polyno-
mialFeatures
# Create the polynomial object with speci-
fied degree
poly_transformer = PolynomialFeatures(de-
gree = 3)
# Train the polynomial features
poly_transformer.fit(poly_features)
# Transform the features
poly_features = poly_transformer.trans-
form(poly_features)
poly_features_test = poly_transformer.-
transform(poly_features_test)
print('Polynomial Features shape: ',
poly_features.shape)
poly_transformer.get_feature_names(in-
put_features = ['EXT_SOURCE_1', 'EXT_-
SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH'])
[:15]
# Create a dataframe of the features
poly_features = pd.DataFrame(poly_fea-
tures,
                                  columns =
poly_transformer.get_feature_names(['EXT_S
OURCE_1', 'EXT_SOURCE_2',

'EXT_SOURCE_3', 'DAYS_BIRTH']))
# Add in the target
poly_features['TARGET'] = poly_target
# Find the correlations with the target
poly_corrs = poly_features.corr()['TAR-
GET'].sort_values()
```

missing value and Z-value

```
from sklearn.preprocessing import MinMaxS-
caler, Imputer
# Drop the target from the training data
if 'TARGET' in app_train:
        train = app_train.drop(columns =
['TARGET'])
else:
    train = app_train.copy()
# Feature names
features = list(train.columns)
# Copy of the testing data
test = app_test.copy()
# Median imputation of missing values
imputer = Imputer(strategy = 'median')
# Scale each feature to 0-1
scaler = MinMaxScaler(feature_range = (0,
1))
# Fit on the training data
imputer.fit(train)
# Transform both training and testing data
train = imputer.transform(train)
test = imputer.transform(app_test)
# Repeat with the scaler
scaler.fit(train)
train = scaler.transform(train)
test = scaler.transform(test)
print('Training data shape: ',
train.shape)
print('Testing data shape: ', test.shape)
```

LogisticRegression

```
from sklearn.linear_model import Logistic-
Regression
# Make the model with the specified regu-
larization parameter
log_reg = LogisticRegression(C = 0.0001)
# Train on the training data
log_reg.fit(train, train_labels)
# Make predictions
# Make sure to select the second column
only
log_reg_pred = log_reg.predict_proba(test)
[:, 1]
```

Random Forest

```
from sklearn.ensemble import RandomForest-
Classifier
# Make the random forest classifier
random_forest = RandomForestClassifi-
er(n_estimators = 100, random_state = 50,
verbose = 1, n_jobs = -1)
# Train on the training data
random_forest.fit(train, train_labels)
# Extract feature importances
feature_importance_values = random_for-
est.feature_importances_
feature_importances = pd.DataFrame({'fea-
ture': features, 'importance': feature_im-
portance_values})
# Make predictions on the test data
predictions = random_forest.predict_pro-
ba(test)[:, 1]
#show feature importance
def plot_feature_importances(df):
    """
     Plot importances returned by a model.
This can work with any measure of
     feature importance provided that high-
er importance is better.

    Args:
           df (dataframe): feature impor-
tances. Must have the features in a column
          called `features` and the impor-
tances in a column called `importance

    Returns:
        shows a plot of the 15 most impor-
tance features

           df (dataframe): feature impor-
tances sorted by importance (highest to
lowest)
          with a column for normalized im-
portance
        """
      # Sort features according to impor-
tance
        df = df.sort_values('importance',
ascending = False).reset_index()
    # Normalize the feature importances to
add up to one
          df['importance_normalized'] =
df['importance'] / df['importance'].sum()
     # Make a horizontal bar chart of fea-
ture importances
    plt.figure(figsize = (10, 6))
    ax = plt.subplot()
      # Need to reverse the index to plot
most important on top
      ax.barh(list(reversed(list(df.index[:
15]))),

df['importance_normalized'].head(15),
             align = 'center', edgecolor =
'k')
    # Set the yticks and labels

ax.set_yticks(list(reversed(list(df.index[
:15]))))
          ax.set_yticklabels(df['fea-
ture'].head(15))
    # Plot labeling
      plt.xlabel('Normalized Importance');
plt.title('Feature Importances')
    plt.show()
    return df
# Show the feature importances for the de-
fault features
feature_importances_sorted = plot_fea-
ture_importances(feature_importances)
```