Box_Cox

```
def boxcox_trans(x,y):
    """
    https://github.com/tgsmith61591/skutil
    x, y: numpy.array
    """
    from skutil.preprocessing import BoxCoxTransformer
    import pandas as pd
    X = np.array([x, y]).transpose()
    Xdf = pd.DataFrame.from_records(data=X)
    transformer = BoxCoxTransformer(as_df=False).fit(Xdf)
    newX = transformer.transform(Xdf)[:,0]
    return newX
```

Cramer's V

if V>0.7, remove one feature

```
def cramers_corrected_stat(x1, x2):
    '''
    x1, x2: pandas.Series
    '''
    import scipy.stats as ss
    import pandas as pd
    import numpy as np
    confusion_matrix = pd.crosstab(x1, x2)
    chi2 = ss.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2 - (k-1)*(r-1)/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min(kcorr-1, rcorr-1))
```

VIF

if VIF>5, remove one feature

```
def vif(X):
    """
    Parameters:
    -----------
    X : pands.DataFrame
        DataFrame containing multiple variables and observations for predictors.

    Returns:
    --------
    vif : pandas.DataFrame
        vif values between two predictors.
    """
    colnames = X.columns
    values = X.values.T
    length = X.shape[1]
    rs = np.corrcoef(values)**2
    for i in range(length):
        rs[i ,i] = 0
    vif_values = 1 / (1 - rs)
    for j in range(length):
        vif_values[j, j:] = 0
    vif = pd.DataFrame(data = vif_values,
                       columns = colnames,
                       index = colnames)
    return vif
```

Correlation ratio

if eta > 0.6, then drop the dicrete variable.

```
def corr_ratio(cv, dv):
    """
    cv: numpy array
        continuous variable

    dv: numpy array
        discrete variable
    """
    import numpy as np
    mean = np.mean(cv)
    uniq = np.unique(dv)
    sub_mean = []
    sub_count = []
    numerator = 0
    part_denominator = 0
    for sub in uniq:
        pos = dv == sub
        count = sum(pos)
        sub_count.append(count)
        sub_cv = cv[pos]
        part_denominator += np.var(sub_cv) * count
        sub_mean.append(np.mean(sub_cv))
    if len(sub_count) == len(sub_mean):
        for i in range(len(sub_count)):
            counti = sub_count[i]
            meani = sub_mean[i]
            numerator += counti * (meani - mean)**2
    else:
        print("Lengths of 'sub_count' and 'sub_mean' must be same.")
    eta2 = numerator / (part_denominator + numerator)
    return np.sqrt(eta2)
```