데이터 전처리

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings(action='ignore')
pd.set_option('display.max_columns', 500)
data=pd.read_csv('data.csv', encoding='cp949')

col_id=[]
col_cat=['season']
col_int=['casual', 'registered']
col_float=['temp', 'atemp']
col_bool=['holiday', 'workingday']
col_date=['datetime']

data[col_int]=data[col_int].astype('int', errors='ignore')
data[col_float]=data[col_float].astype('float')
data[col_cat]=data[col_cat].astype('str')
data['datetime']=pd.to_datetime(data['datetime'])

data.descirbe()
data.columns
data.index
data.dtypes
data['count'].value_counts()
```

#중복값 처리

```
data[data.duplicated(keep=False)]
data.drop_duplicates()
data.drop_duplicates(['col1'], keep='last')
```

#결측치 처리

```
data.isnull().sum()
data.fillna(0)
data['count']=data['count'].fillna(data['count'].mean())
```

#이상치 처리

```
def get_outlier(df, columns):
        for column in columns:
                q1=df[column].quantile(.25)
                q3=df[column].quantile(.75)
                iqr=q3-q1
                low=q1-1.5*iqr
                high=q3+1.5*iqr
                df.loc[df[column]<low, column]=low
                df.loc[df[column]>high, column]=high
        return df

cut_data=get_outlier(data, ['humidity', 'casual', 'registered']
```

#날짜변수 처리

```
X_train['datetime']=pd.to_datetime(X_train['datetime'])
X_test['datetime']=pd.to_datetime(X_test['datetime'])
def preprocessing_data(df2):
    df=df2.copy()
    df['year']=df['datetime'].map (lambda x : x.year)
    df['month']=df['datetime'].map(lambda x: x.month)
    df['day']=df['datetime'].map(lambda x: x.day)
    df['weekend']=df['datetime'].map(lambda x : x.dayofweek)
    df['hour']=df['datetime'].map(lambda x : x.hour)
    df['minute']=df['datetime'].map(lambda x : x.minute)
    return df

X_train=preprocessing_data(X_train)
X_train=X_train.drop(['datetime'], axis=1)
X_test=preprocessing_data(X_test)
X_test=X_test.drop(['datetime'], axis=1)
```

#정규화, 표준화
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures
columns=['holiday', 'workingday', 'weather', 'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered']
pre_train_data=X_train[columns]
remain_train_data=X_train.drop(columns, axis=1)
pre_test_data=X_test[columns]
remain_test_data=X_test.drop(columns, axis=1)
scaler=StandardScaler()
scaled_X_train=scaler.fit_transform(pre_train_data)
scaled_X_train=pd.DataFrame(scaled_X_train, columns=pre_train_data.columns, index=pre_train_data.index)
scaled_X_test=scaler.transform(pre_test_data)
scaled_X_test=pd.DataFrame(scaled_X_test, columns=pre_test_data.columns, index=pre_test_data.index)
X_train=pd.concat([scaled_X_train, remain_train_data], axis=1)
X_test=pd.concat([scaled_X_test, remain_test_data], axis=1)
StandardScaler().fit_transform(train_data)
MinMaxScaler().fit_transform(train_data)
np.log1p(train_data)
PolynomialFeatures(degree=2, include_bias=False).fit_transform(train_data)
```

#Dummy화
```
cat_data=train_scale[col_cat]
remain_data=train_scale.drop(col_cat, axis=1)
train_data_dum=pd.get_dummies(train_data)
test_data_dum=pd.get_dummies(test_data)
final_train, final_test=train_data_dum.align(test_data_dum, join='left', axis=1)
```

#Label Encoder
```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
pre_train_data=X_train[['season']]
remain_train_data=X_train.drop(['season'], axis=1)
pre_train_data=encoder.fit_transform(pre_train_data)
pre_train_data=pd.DataFrame(pre_train_data, columns=['season'], index=X_train.index)
X_train=pd.concat([pre_train_data, remain_train_data], axis=1)
pre_test_data=X_test[['season']]
remain_test_data=X_test.drop(['season'], axis=1)
pre_test_data=encoder.fit_transform(pre_test_data)
pre_test_data=pd.DataFrame(pre_test_data, columns=['season'], index=X_test.index)
X_test=pd.concat([pre_test_data, remain_test_data], axis=1)
```

#차원축소
```
from sklearn.decomposition import PCA, TruncatedSVD, NMF, FactorAnalysis
from sklerarn.discriminant_analysis import LinearDiscriminantAnalysis
pca=PCA(n_components=2)
lda=LinearDiscriminantAnalysis(n_components=2)
tsvd=TruncatedSVD(n_components=2)
factor=FactorAnalysis(n_components=2)
nmf=NMF(n_components=2)

pca.fit_transform(scaled_data)
pca_columns=['pca_component+1', 'pca_components_2']
data_pca=pd.DataFrame(data_pca, columns=pca_columns)
```

#상관분석
```
import scipy.stats as spst
data.corr()
spst.spearmanr(data['twitter'], data['revenues']).correlation
spst.kendalltau(data['twitter'], dfata['revenues']).correlation
spst.pearsonr(data['twitter'], data['revenues'])
```

#t-test
```
import scipy.stats as spst
m=spst.ttest_ind(datM, datF, equal_var=False)
t=m.statistic
abs(t)/np.sqrt((len(datM)+len(datF)-2
np.sqrt(t**2)/(t**2)+len(datM)+len(datF)-2
spst.ttest_rel(datM, datF)

spst.f_oneway(datM, datF, datK)
```

분류

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, Y, test_size=0.3, random_state=0)

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

lr_clf=LogisticRegression()
gb_clf=GradientBoostingClassifier()
rf_clf=RandomForestClassifier()
knn_clf=KNeighborsClassifier(n_neighbors=8)
dt_clf=DecisionTreeClassifier()
xgb_clf=XGBClassifier()

from sklearn.model_selection import cross_val_score
import numpy as np

#roc_auc는 이진분류, roc_auc_ovr, roc_auc_ovo
scores_lr=cross_val_score(lr_clf, X, Y, scoring= 'roc_auc_ovr', cv=3)
scores_gb=cross_val_score(gb_clf, X, Y, scoring= 'roc_auc_ovr', cv=3)
scores_rf=cross_val_score(rf_clf, X, Y, scoring='roc_auc_ovr', cv=3)
scores_knn=cross_val_score(knn_clf, X, Y, scoring= 'roc_auc_ovr', cv=3)
scores_dt=cross_val_score(dt_clf, X, Y, scoring='roc_auc_ovr', cv=3)
scores_xgb=cross_val_score(xgb_clf, X, Y, scoring='roc_auc_ovr', cv=3)

print('cross_LogisticRegression 분류기 정확도:', np.round(np.mean(scores_lr),4))
print('corss_KNN 분류기 정확도:', np.round(np.mean(scores_knn),4))
print('cross_Decision Tree 분류기 정확도:', np.round(np.mean(scores_dt),4))
print('cross_Random Forest 분류기 정확도:', np.round(np.mean(scores_rf),4))
print('cross_Gradient Boosting 분류기 정확도:', np.round(np.mean(scores_gb),4))
print('cross_XGB 분류기 정확도:', np.round(np.mean(scores_xgb),4))

LogisticRegression: penalty, C
DecisionTreeClassifier : min_samples_split, min_samples_leaf, max_features, max_depth
RandomForestClassifier : n_estimators, max_features, max_depth, min_samples_leaf, min_samples_split, bootstrap
GradientBoostingClasssifier: learning_rate, n_estimators, learning_rate
XGB: min_child_weight, gamma, max_depth, sub_sample, colsample_bytree, reg_alpha, learning_rate

penalty=['l2', 'l1']
max_features = ['auto', 'sqrt']
bootstrap = [True, False]
C=[0.01, 0.1, 1.5, 10]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
min_child_weight=list(range(50, 100))
max_depth = list(range(3, 10))
n_estimators = list(range(100,200))
learning_rate=np.linspace(0.1, 1, 10)
subsample=np.linspace(0.5, 1, 7)
reg_alpha=np.linspace(0.5, 1, 10)
gamma=np.linspace(0.3, 1, 8)
colsample_bytree=np.linspace(0.5,1,8)

rf_param_grid={'n_estimators': n_estimators, 'max_features': max_features, 'max_depth': max_depth,
               'min_samples_split': min_samples_split, 'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}
xgb_param_grid={'max_depth': max_depth, 'subsample': subsample, 'reg_alpha' : reg_alpha,
                'learning_rate': learning_rate, 'n_estimators': n_estimators, 'gamma' : gamma,
                'min_child_weight': min_child_weight, 'colsample_bytree': colsample_bytree}

from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
import random
import numpy as np

rf_clf=RandomForestClassifier()
xgb_clf=XGBClassifier()

rf_random=RandomizedSearchCV(estimator=rf_clf, param_distributions=rf_param_grid, n_iter=100, cv=3, verbose=2,
                             scoring='roc_auc_ovr', random_state=0, n_jobs=-1, refit=True, return_train_score = True)
xgb_random=RandomizedSearchCV(estimator = xgb_clf, param_distributions = xgb_param_grid, n_iter = 100, cv=3,
                              scoring='roc_auc_ovr', random_state=0, n_jobs=-1, refit=True, return_train_score = True)
```

```
rf_random.fit(X_train, y_train)
xgb_random.fit(X_train, y_train)
rf_random_df=pd.DataFrame(rf_random.cv_results_)
xgb_random_df = pd.DataFrame(xgb_random.cv_results_)
rf_result=rf_random_df.loc[:, ['mean_test_score', 'params']]
xgb_result=xgb_random_df.loc[:, ['mean_test_score', 'params']]

rf_final_score=rf_result.sort_values('mean_test_score', ascending=False).reset_index()
xgb_final_score=xgb_result.sort_values('mean_test_score', ascending=False).reset_index()
rf_scores=rf_final_score['mean_test_score'][0]
xgb_scores=xgb_final_score['mean_test_score'][0]
rf_params=rf_final_score['params'][0]
xgb_params=xgb_final_score['params'][0]

rf_final_clf=RandomForestClassifier(n_jobs=-1, n_estimators=rf_params['n_estimators'],
                                    max_features=rf_params['max_features'], max_depth=rf_params['max_depth'],
                                    min_samples_split=rf_params['min_samples_split'], bootstrap=rf_params['bootstrap'],
                                    min_samples_leaf=rf_params['min_samples_leaf'])

xgb_final_clf=XGBClassifier(n_jobs=-1, eval_metric='auc', n_estimators=xgb_params['n_estimators'],
                            reg_alpha=xgb_params['reg_alpha'], max_depth=xgb_params['max_depth'],
                            learning_rate=xgb_params['learning_rate'], gamma=xgb_params['gamma'],
                            min_child_weight=xgb_params['min_child_weight'], subsample=xgb_params['subsample'],
                            colsample_bytree=xgb_params['colsample_bytree'], use_label_encoder=False)

rf_final_clf.fit(X,Y)
xgb_final_clf.fit(X,Y)

from sklearn.ensemble import VotingClassifier
final_clf=VotingClassifier(estimators=[('RF', rf_final_clf), ('GB', xgb_final_clf)], voting='soft', n_jobs=-1)
voting=final_clf.fit(X, Y)

pred_y_0=pd.Series(voting.predict_proba(X_test)[:,0], name='pred_y_0')
pred_y_1=pd.Series(voting.predict_proba(X_test)[:,1], name='pred_y_1')
pred_y_2=pd.Series(voting.predict_proba(X_test)[:,2], name='pred_y_2')
pred_y_3=pd.Series(voting.predict_proba(X_test)[:,3], name='pred_y_3')
results=pd.concat([pred_y_0, pred_y_1, pred_y_2, pred_y_3], axis=1)
results.index=X_test.index
results.to_csv('ensemble_voting.csv', index=False)


from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, auc, classification_report
tn, fp, fn, tp=confusion_matrix(y_test, pred).ravel()
specificity=tn/(tn+fp)
accuracy_score(y_test, pred)
```

회귀

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, Y, test_size=0.3)

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, VotingRegressor
from xgboost import XGBRegressor
from sklearn.linear_model import Ridge, Lasso, ElasticNet

dt_reg=DecisionTreeRegressor()
rf_reg=RandomForestRegressor()
gb_reg=GradientBoostingRegressor()
xgb_reg=XGBRegressor()
rg_reg=Ridge()
ls_reg=Lasso()
en_reg=ElasticNet()

from sklearn.model_selection import cross_val_score
neg_scores_dt=cross_val_score(dt_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)
neg_scores_rf=cross_val_score(rf_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)
neg_scores_gb=cross_val_score(gb_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)
neg_scores_xgb=cross_val_score(xgb_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)
neg_scores_rg=cross_val_score(rg_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)
neg_scores_ls=cross_val_score(ls_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)
neg_scores_en=cross_val_score(en_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=3)

scores_dt=np.sqrt(-1*neg_scores_dt)
scores_rf=np.sqrt(-1*neg_scores_rf)
scores_gb=np.sqrt(-1*neg_scores_gb)
scores_xgb=np.sqrt(-1*neg_scores_xgb)
scores_rg=np.sqrt(-1*neg_scores_rg)
scores_ls=np.sqrt(-1*neg_scores_ls)
scores_en=np.sqrt(-1*neg_scores_en)

print('cross_Decision Tree 분류기 정확도:', np.round(np.mean(scores_dt),4))
print('cross_RandomForest 분류기 정확도:', np.round(np.mean(scores_rf),4))
print('cross_Gradient Boosting 분류기 정확도:', np.round(np.mean(scores_gb),4))
print('cross_XGB Regressor 분류기 정확도:', np.round(np.mean(scores_xgb),4))
print('cross_Ridge 분류기 정확도:', np.round(np.mean(scores_rg),4))
print('cross_Lasso 분류기 정확도:', np.round(np.mean(scores_ls),4))
print('cross_Elastic Net 분류기 정확도:', np.round(np.mean(scores_en),4))

DecisionTreeRegressor : min_samples_split, min_samples_leaf, max_features, max_depth
RandomForestRegressor : n_estimators, max_features, max_depth, min_samples_leaf, min_samples_split, bootstrap
GradientBoostingRegreissor: learning_rate, n_estimators, learning_rate
XGB: min_child_weight, gamma, max_depth, sub_sample, colsample_bytree, reg_alpha, learning_rate
Ridge, Lasso, Elastic Net: alphas

import random
alphas=[0, 0.05, 0.1, 0.5, 1, 5, 10, 100]
max_features = ['auto', 'sqrt']
bootstrap = [True, False]
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
min_child_weight=list(range(50, 100))
max_depth = list(range(3, 10))
n_estimators = list(range(100,200))
learning_rate=np.linspace(0.1, 1, 10)
subsample=np.linspace(0.5, 1, 7)
reg_alpha=np.linspace(0.5, 1, 10)
gamma=np.linspace(0.3, 1, 8)
colsample_bytree=np.linspace(0.5,1,8)
rf_param_grid={'n_estimators': n_estimators, 'max_features': max_features, 'max_depth': max_depth,
                'min_samples_split': min_samples_split, 'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}
xgb_param_grid={'max_depth': max_depth, 'subsample': subsample, 'reg_alpha' : reg_alpha,
                 'learning_rate': learning_rate, 'n_estimators': n_estimators, 'gamma' : gamma,
                 'min_child_weight': min_child_weight, 'colsample_bytree': colsample_bytree}

from sklearn.model_selection import RandomizedSearchCV
import random

rf_clf=RandomForestRegressor()
xgb_clf=XGBRegressor()
```

```
rf_random=RandomizedSearchCV(estimator=rf_clf, param_distributions=rf_param_grid, n_iter=100, cv=3, verbose=2,
                             scoring='neg_mean_squared_error', random_state=0, n_jobs=-1, refit=True, return_train_score = True)
xgb_random=RandomizedSearchCV(estimator = xgb_clf, param_distributions = xgb_param_grid, n_iter = 100, cv=3,
                              scoring='neg_mean_squared_error', random_state=0, n_jobs=-1, refit=True, return_train_score = True)

rf_random.fit(X_train, y_train)
xgb_random.fit(X_train, y_train)
rf_random_df=pd.DataFrame(rf_random.cv_results_)
xgb_random_df = pd.DataFrame(xgb_random.cv_results_)
rf_result=rf_random_df.loc[:, ['mean_test_score', 'params']]
xgb_result=xgb_random_df.loc[:, ['mean_test_score', 'params']]
rf_final_score=rf_result.sort_values('mean_test_score', ascending=False).reset_index()
xgb_final_score=xgb_result.sort_values('mean_test_score', ascending=False).reset_index()
rf_scores=rf_final_score['mean_test_score'][0]
xgb_scores=xgb_final_score['mean_test_score'][0]
rf_params=rf_final_score['params'][0]
xgb_params=xgb_final_score['params'][0]

rf_final_clf=RandomForestRegressor(n_jobs=-1, n_estimators=rf_params['n_estimators'],
                                   max_features=rf_params['max_features'], max_depth=rf_params['max_depth'],
                                   min_samples_split=rf_params['min_samples_split'], bootstrap=rf_params['bootstrap'],
                                   min_samples_leaf=rf_params['min_samples_leaf'])

xgb_final_clf=XGBRegressor(n_jobs=-1, eval_metric='auc', n_estimators=xgb_params['n_estimators'],
                           reg_alpha=xgb_params['reg_alpha'], max_depth=xgb_params['max_depth'],
                           learning_rate=xgb_params['learning_rate'], gamma=xgb_params['gamma'],
                           min_child_weight=xgb_params['min_child_weight'], subsample=xgb_params['subsample'],
                           colsample_bytree=xgb_params['colsample_bytree'], use_label_encoder=False)

rf_final_clf.fit(X_train,y_train)
xgb_final_clf.fit(X_train,y_train)

from sklearn.ensemble import VotingRegressor
voting=VotingRegressor(estimators=[('RF', rf_final_clf),('XGB', xgb_final_clf)])
voting.fit(X_train, y_train)
pred=voting.predict(X_test)
pred=pred.reshape(-1,1)
result=pd.DataFrame(pred, index=X_test.index).rename(columns={0: 'prediction'})

final=pd.concat([result, y_test], axis=1)
final.to_csv('ensemble_voting.csv', index=False)


from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
r2=r2_score(y_test,y_preds) # R2 : 0~1, 1에 가까울수록 설명력이 높은 것
mae=mean_absolute_error(true,pred)
mse=mean_squared_error(y_test,y_preds) # MSE : 낮을수록 좋음
rmse=np.sqrt(mse) # RMSE : 낮을수록 좋음
def rmsle (y,pred): # RMSLE : 낮을수록 좋음
    log_y=np.log1p(y)
    log_pred=np.log1p(pred)
    squared_error=(log_y-log_pred)**2
    rmsle=np.sqrt(np.mean(squared_error))
    return rmsle
def mape(y_test,y_preds):
    y_test,y_preds=np.array(y_test),np.array(y_preds)
    return np.mean(np.abs((y_test-y_preds)/y_test))*100
```