**■ 피처 처리**

• sklearn.preprocessing

**•• LabelEncoder**

- encoder = LabelEncoder()

encoder.fit(items)

labels = encoder.transform(items)

- encoder.classes\_

- encoder.inverse\_transform()

**•• OneHotEncoder**

- oh.encoder = OneHotEncoder()

oh\_encoder.fit(labels)

oh\_labels = oh\_encoder.transform(labels)

**•• StandardScaler**

- scaler = StandardScaler()

scaler.fit(iris\_df)

iris\_scaled = scaler.transform(iris\_df)

- iris\_scaled = scaler.fit\_transform(iris\_df)

**•• MinMaxScaler**

- scaler = MinMaxScaler()

scaler.fit(iris\_df)

iris\_scaled = scaler.transform(iris\_df)

**•• Binarizer**

- binarizer = Binarizer(threshold=1.1)

binarizer.fit\_transform(X)

- 보통 predict\_proba와 같이 사용함

pred\_proba = lr\_clf.predict\_proba(X\_test)

**•• LabelBinarizer**

- lb\_brand\_name = LabelBinarizer(sparse\_output=True)

X\_brand\_name.fit\_transform(mercari\_df[‘brand\_name’]

• sklearn.feature\_selection

• sklearn.feature\_extraction

• imblearn.over\_sampling

**•• SMOTE**

- smote = SMOTE(random\_state=0)

X\_train\_over, y\_train\_over = smote.fit\_sample(X\_train, y\_train)

•

**■ 피처 처리 & 차원 축소**

• sklearn.decomposition

**•• PCA**

- pca = PCA(n\_components=2)

pca.fit(iris\_scaled)

iris\_pca = pca.transform(iris\_scaled)

- pca.explained\_variance\_ratio\_ ; 변동성 비율(분산) 출력

**•• TruncatedSVD**

- tsvd = TruncatedSVD(n\_components=2)

tsvd.fit(iris\_ftrs)

iris\_tsvd = tsvd.transform(iris\_ftrs)

**•• NMF**

- nmf = NMF(n\_components=2)

nmf.fit(iris\_ftrs)

iris\_nmf = nmf.transform(iris\_ftrs)

**•• PolynomialFeatures**

- poly = PolynomialFeatures(degree=2)

poly.fit(X)

poly\_ftr = poly.transform(X)

- poly\_ftr = PolynomialFeatures(degree=3).fit\_transform(X)

• sklearn.discriminant\_analysis

**•• LinearDiscriminantAnalysis**

- lda = LinearDiscriminantAnalysis(n\_components=2)

lda.fit(iris\_scaled, iris.target)

iris\_lda = lda.transform(iris\_scaled)

• numpy.linalg

**•• svd**

- U, Sigma, Vt = svd(a)

• scipy.sparse.linalg

**•• svds**

- U-tr, Sigma\_tr, Vt\_tr = svds(matrix, k=4)

**■ 데이터 분리, 검증 & 파라미터 튜닝**

• sklearn.model\_selection

**•• train\_test\_split**

- X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris\_data.data, iris\_data.target,

test\_size=0.2, random\_state=121)

**•• KFold**

- kfold = KFold(n\_splits=5)

train\_index, test\_index in kfold.split(features)

X\_train, X\_test = features[train\_index], feature[test\_index]

y\_train, y\_test = label[train\_index], label[test\_index]

**•• StratifiedKFold**

- skfold = StratifiedKFold(n\_splits=3)

train\_index, test\_index in skfold.split(features, label)

X\_train, X\_test = features[train\_index], feature[test\_index]

y\_train, y\_test = label[train\_index], label[test\_index]

**•• cross\_val\_score**

- scores = cross\_val\_score(dt\_clf, data, label, scoring=’accuracy’, cv=3)

- rmse\_list = np.sqrt(-cross\_val\_score(model, X\_features, y\_target,

scoring="neg\_mean\_squared\_error", cv=5))

rmse\_avg = np.mean(rmse\_list)

- dt\_clf, lasso, ridge

**•• cross\_validate**

**•• GridSearchCV**

- parameters = { ‘max\_depth’ : [6,8,10,12], ‘min\_samples\_split’ : [2,3,4] }

grid\_dtree = GridSearchCV(dt\_clf, param\_grid = parameters, cv=3, refit=True, n\_jobs=-1,

verbose=1)

- grid\_model = GridSearchCV(model, param\_grid=parmas,

scoring='neg\_mean\_squared\_error', cv=5)

grid\_model.fit(X\_features, y\_target)

- grid\_dtree.cv\_results\_

- grid\_dtree.best\_params\_

- grid\_dtree.best\_score\_

- grid\_dtree.best\_estimator\_ : refit를 하면 최적성능의 하이퍼파라미터로 학습해 저장됨

**■ 평가**

• sklearn.metrics

**•• accuracy\_score**

- accuracy\_socre(y\_test, pred)

**•• confusion\_matrix**

- confusion\_matrix(y\_test, pred)

**•• precision\_score**

- precision\_score(y\_test, pred)

**•• recall\_score**

- recall\_score(y\_test, pred)

**•• precision\_recall\_curve**

- pred\_proba\_class1 = lr\_clf.predict\_proba(X\_test)[:,1]

- precision\_recall\_curve(y\_test, pred\_proba\_class1)

**•• f1\_score**

- f1\_score(y\_test, pred)

**•• roc\_curve**

- fprs, tprs, thresholds = roc\_curve(y\_test, pred\_proba\_c1)

- fprs = feature, tprs = target

**•• roc\_auc\_score**

- pred\_proba = lr\_clf.pred\_proba(X\_test)

- roc\_score = roc\_auc\_score(y\_test, pred\_proba)

**•• mean\_squared\_error**

- mean\_squared\_error(y, pred)

**•• mean\_absolute\_error**

- mae\_val = mean\_absolute\_error(y, pred)

**•• silhouette\_samples**

- score\_samples = silhouette\_samples(iris.data, irisDF[‘cluster’], metric=’euclidean’)

**•• silhouette\_score**

- sihouette\_score(iris.data, irisDF[‘cluster’], metric=’euclidean’)

• sklearn.metrics.pairwise

**•• cosine\_similarity**

- similarity\_simple\_pair = cosine\_similarity(feature\_vect\_simple, feature\_vect\_simple)

**■ ML 알고리즘**

• sklearn.ensemble

**•• VotingClassifier**

- vo\_clf = VotingClassifier( estimators=[(‘LR’, lr\_clf), (‘KNN’, knn\_clf)], voting=’soft’)

- voting = ‘soft’ or ‘hard’

- vo\_clf.fit(X\_train, y\_train) ; pred = vo\_clf.predict(X\_test)

**•• BaggingClassifier**

- bag = BaggingClassifier(random\_state=1)

**•• ExtraTreesClassifier**

- ext = ExtraTreesClassifier(random\_state=1)

**•• AdaBoostClassifier**

- ada = AdaBoostClassifier(random\_state=1)

n\_estimators = [10, 100, 200, 500]

learning\_rate = [0.001, 0.01, 0.1, 0.5, 1, 1.5, 2]

hyperparams = {'n\_estimators': n\_estimators, 'learning\_rate': learning\_rate}

**•• RandomForestClassifier**

- rf\_clf1 = RandomForestClassifier(n\_estimators=300, max\_features=’auto’, max\_depth=10,

min\_samples\_leaf=8, min\_samples\_split=8, random\_state=0)

- rf\_clf1.feature\_importances\_

**•• GradientBoostingClassifier**

- GradientBoostingClassifier(loss = ‘deviance’, learning\_rate=0.1, n\_estimators = 100,

subsample = 1)

• neighbors

•• KNeighborsClassifier

•xgboost

**•• XGBClassifier**

- xgb\_wrapper = XGBClassifier(n\_estimators=400, learning\_rate=0.1, max\_depth=3)

- evals = [(X\_test, y\_test)]

xgb\_wrapper.fit(X\_train, y\_train, early\_stopping\_rounds=100, eval\_metric=”logloss”,

eval\_set = evals, verbose=True)

xgb\_wrapper\_pred = wgb\_wrapper.predict(X\_test)

**•• plot\_importance**

- fig, ax = plt.subplots(figsize=(10,12))

plot\_importance(xgb\_wrapper, ax=ax)

• lightgbm

**•• LGBMClassifier**

• sklearn.linear\_model

**•• LinearRegression**

- LinearRegression(fit\_intercept=True, normalize=False, copy\_X=True, n\_jobs=1)

- lr.coef\_

- lr.intercept\_

**•• Ridge**

- ridge = Ridge(alpha=10)

**•• Lasso**

- lasso = Lasso(alpha=10)

**•• ElasticNet**

- ElasticNet(alpha=10, l1\_ratio=0.7)

**•• LogisticRegression**

- lr\_clf = LogisticRegression(penalty = ‘l2’, C = ‘0.01’)

• sklearn.naive\_bayes

• sklearn.neighbnors

• sklearn.svm

**•• SVC**

- svc = SVC(probability=True)

Cs = [0.01, 0.1, 1, 5, 10, 15, 20, 50]

gammas = [0.001, 0.01, 0.1]

hyperparams = {'C': Cs, 'gamma' : gammas}

• sklearn.tree

**•• DecisionTreeClassifier**

- DecisionTreeClassifier(min\_samples\_split= 4, min\_samples\_leaf=3, max\_features=3,

max\_depth=3, max\_leaf\_nodes=3, random\_state=156)

- dt\_clf.feature\_importances\_

• sklearn.cluster

**•• KMeans**

- kmeans = KMeans(n\_clusters=8, init=’k-means++’, max\_iter=300, random\_state=0)

kmeans.fit(iris\_df)

- kmeans.labels\_

- kmeans.cluster\_centers\_

**•• MeanShift**

- meanshift = MeanShift(bandwidth=1)

cluster\_labels = meanshift.fit\_predict(X)

- centers = meanshift.cluster\_centers\_

- estimate\_bandwidth

bandwidth = estimate\_bandwidth(X, quantile=0.2)

**•• DBSCAN**

- dbscan = DBSCAN(eps=0.6, min\_samples=8, metric=’euclidean’)

dbscan\_labels = dbscan.fit\_predict(iris.data)

• sklearn.mixture

**•• GaussianMixture**

- gmm = GaussianMixture(n\_components=3, random\_state=0)

gmm.fit(iris.data)

gmm\_cluster\_labels = gmm.predict(iris.data)

**■ 유틸리티**

• sklearn.pipeline

**•• Pipeline**

- model = Pipeline([(‘poly’, PolynomialFeatures(degree=3)), (‘linear’, LinearRegression())])

model = model.fit(X, y)

- coefficients = pipeline.named\_steps[‘linear\_regression’].coef\_

**■ 텍스트 분석**

• nltk

**•• sent\_tokenize**

- sentences = sent\_tokenize(text=text\_sample)

**•• word\_tokenize**

- words = word\_tokenize(sentence)

**•• corpus.stopwords.words(‘english’)**

- stopwords = corpus.stopwords.words(‘english’)

- nltk.download(‘stopwords’)

• nltk.stem

**•• LancasterStemmer**

- stemmer = LancaterStemmer()

print(stemmer.stem(‘working’), stemmer.stem(‘works’), stemmer.stem(‘worked’)))

**•• WordNetLemmatizer**

- lemma = WordNetLemmatizer()

print(lemma.lemmatize(‘amusing’, ‘v’), lemma.lemmatize(‘amuses’, ‘v’),

lemma.lemmatize(‘amused’, ‘v’))

print(lemma.lemmatize(‘happier’, ‘a’), lemma.lemmatize(‘happiest’, ‘a’)

• sklearn.feature\_extraction.text

**•• CountVectorizer**

- cnt\_vect = CountVectorizer()

cnt\_vect.fit(X\_train, ?y\_train?)

X\_train\_cnt\_vect = cnt\_vect.transform(X\_train)

X\_test\_cnt\_vect = cnt\_vect.transform(X\_test)

- train으로 학습한 걸로 test를 합습시켜야 함

**•• TfidfVectorizer**

- tfidf\_vect = TfidfVectorizer(tokenizer=LemNormalize, stop\_words=’english’,

ngram\_range=(1,2), min\_df=0.05, max\_df=300 or 0.85))

tfidf\_vect.fit(X\_train)

X\_train\_tfidf\_vect = tfidf\_vect.transform(X\_train)

X\_test\_tfidf\_vect = tfidf\_vect.transform(X\_test)

- tokenizer=LemNormalize or tw\_tokenizer 함수를 만들어 집어 넣음, 빼도 됨

- train으로 학습한 걸로 test를 합습시켜야 함

• nltk.corpus

**•• wordnet as wn**

- term = ‘present’

synsets = wn.synsets(term)

- POS = synset.lexname()

- Definition = synset.definition()

- Lemmas = synset.lemma\_names()

- entity.path\_similarity(compared\_entity)

**•• sentiwordnet as swn**

- senti\_synsets = list(swn.senti\_synsets(‘slow’))

- father = swn.senti\_synset(‘father.n.01’)

father.pos\_score()

father.neg\_score()

father.obj\_score()

• nltk.sentiment.vader

**•• SentimentIntensityAnalyzer**

- senti\_analyzer = SentimentIntensityAnalyzer()

- senti\_scores = senti\_analyzer.polarity\_scores(train\_df[‘review’][0])

• decomposition

**•• LatentDirichletAllocation**

- lda = LatentDirichletAllocation(n\_components=8, random\_state=0)

lda.fit(feat\_vect)

- lda.components\_

- LDA는 Count기반의 벡터화만 사용함

• konlpy.tag

**•• Twitter**

- twitter = Twitter()

tokens\_ko = twitter.morphs(text)

**■ 추천(Surprise)**

• surprise

**•• SVD**

- algo = SVD(n\_epochs=20, n\_factors=50, random\_state=0))

algo.fit(trainset)

- predictions = algo.test(testset)

- uid = str(196)

iid = str(302)

pred = algo.predict(uid, iid)

**•• accuracy**

- accuracy.rmse(predictions)

**•• Reader**

- reader = Reader(line\_format=’user item rating timestamp’, sep=’,’, rating\_scale=(0.5, 5))

data = Dataset.load\_from\_file(‘./~~~~/ratings\_noh.csv’, reader=reader)

• surprise.model\_selection

**•• train\_test\_split**

- trainset, testset = train\_test\_split(data, test\_size=.25, random\_state=0)

**•• GridSearchCV**

- param\_grid = {‘n\_epochs’ : [20, 40], ‘n\_factors’ : [50, 100]}

gs = GridSearchCV(SVD, param\_grid, measures=[‘rmse’, ‘mae’], cv=3)

gs.fit(data)

- gs.best\_score[‘rmse’]

- gs.best\_params[‘rmse’]

• surprise.dataset

**•• DatasetAutoFolds**

- data\_folds = DatasetAutoFolds(ratings\_file=’./~~~~/ratings\_noh.csv’, reader=reader)

- trainset = data\_folds.build\_full\_trainset()

**■ 기타 유용한 함수**

• pd.DataFrame(data=iris.data, columns=iris.feature\_names) ; 데이터프레임 형식으로 변환

• label.reshape(-1,1) ; series형태처럼 변환

• oh\_labels.toarray() ; 배열로 변환

• .tolist() ; 리스트로 변환

• pd.get\_dummies(X\_features, columns=[‘year’, ‘month’, ‘hour’, ‘holiday’]) ; 1,0으로 변환

• iris\_df\_scaled.min() ; 최소값

• iris\_df\_scaled.max() ; 최대값

• max([1,2,3]) ; 1,2,3 중에 가장 큰 3 출력

• max(‘Chevrolet’) ; 가장 늦게 오는 알파벳 출력

• math.ceil(0.5) ; 0.5를 반올림해서 1

• math.floor(0.5) ; 0.5를 내림해서 0

• titanic\_df[‘Age’].fillna(titanic\_df[‘Age’].mean(), inplace=True) ; 빈칸 채우기

• .fillna(method=’pad’) ; fillna(method=’bfill’) ; 앞이나 뒤에꺼 따라서 채우기

• titanic\_df[‘Cabin’].value\_counts() ; 요소와 요소갯수 출력

• titanic\_df[‘Cabin’].str[:1] ;

• titanic\_df.gruopby([‘Sex’, ‘Survived’])[‘Survived’].count()

• titanic\_df[‘Age\_cat’] = titanic\_df[‘Age’].apply(lambda x : get\_category(x))

• .apply(lambda x : (‘ ‘).join(x)) ; 각각의 원소를 공백 1개로 합친다.

• .join(‘, ‘) ; 콤마로 하나로 합친다.

• .split(‘, ‘) ; 콤마로 나눠서 각각의 원소로 만든다.

• .values ; 원소 추출

• class\_name = classifier.\_\_class\_\_.\_\_name\_\_

• cust\_df.drop(‘ID’, axis=1, inplace=True)

• cust\_df[‘var3’].replace(-99999, 2, inplace=True)

• corr = card\_df.corr()

• sns.heatmap(corr, cmap=’RdBu’, annot=False, annot\_kws={‘size’:20})

• np.percentile(fraud.values, 25) ; np..percentile(fraud.values, 75)

• np.transpose(pred)

• np.concatenate((knn\_train, rf\_train, dt\_train, ada\_train), axis=1)

• pd.concat((train, test))

pd.concat([df\_1, df\_2], axis=1) ; 데이터 프레임을 합치며 컬럼을 붙인다.

• df\_first.melt(id\_vars=”지역”, var\_name=”기간”, value\_name=”평당분양가격”)

• np.log1p()

• np.expm1()

• bike\_df[‘datatime’] = bike\_df.datatime.apply(pd.to\_datatime)

• bike\_df[‘year’] = bike\_df.datatime.apply(lambda x : x.year)

• bike\_df[‘month’] = bike\_df.datatime.apply(lambda x : x.month)

• bike\_df[‘day’] = bike\_df.datatime.apply(lambda x : x.day)

• bike\_df[‘hour’] = bike\_df.datatime.apply(lambda x : x.hour)

• .days ; 날짜를 일수로 변환

• .copy()

• from scipy.stats import skew ; .skew(x)

• from scipy import sparse ;

- sparse\_coo = sparse.coo\_matrix((data, (row\_pos, col\_pos)))

- sparse\_csr = sparse.csr\_matrix((data2, col\_pos, row\_pos\_ind))

• from scipy.sparse import hstack ‘ X\_features\_sparse = hstack(sparse\_matrix\_list).tocsr()

• .tocsr()

• sparse\_coo.toarray()

• np.dot()

• np.round(a, 3)

• np.diag()

• len()

• import re ; re.sub(“[^a-zA-Z]”, “ “, x) ; .apply(lambda x : re.sub(“[^a-zA-Z]”, “ “, x)

• .argsort()[:, ::-1]

• from ast import literal\_eval

- movies\_df[‘genres’] = movies\_df[‘genres’].apply(literal\_eval)

• m = movies\_df[‘vote\_count’].quantile(0.6)

• .apply(~~~~~, axis = 1)

• ratings\_matrix = ratings.pivot\_table(‘rating’, index=’userId’, columns=’movieId’)

• pd.merge(ratings, movies, on=’movieId’)

pd.merge(left=user\_usage, right=user\_device, on='use\_id', how='outer', indicator=True)

indicator를 True로 하면 \_merge 컬럼이 생성되며 어떻게 합쳐졌는지를 알 수 있다.

indicator = ‘변수이름’ ; indicator = ‘indicator\_info’ 로 하면 변수명도 바뀐다.

• pd.merge(df\_left\_2, df\_right\_2, how='inner', on='KEY', suffixes=('\_left', '\_right'))

on=’KEY’를 중심으로 중복되는 컬럼의 뒷에 글자를 붙여서 출력된다.

출처: https://rfriend.tistory.com/258 [R, Python 분석과 프로그래밍의 친구 (by R Friend)]

• ratings\_matrix.transpose()

• Q.T

• .sum(axis=1)

• sort\_values([‘Fare’, ‘Survived’, ‘Pclass’], ascending=[False, False, False])

• cust\_df.rename(columns={‘InvoiceData’:’Recency’}))

• .rstrip()

• .lstrip()

• .strip()

• try: ~~~; except Keyerror: pass

• .str.extract('([A-Za-z]+)\.')

• pd.crosstab([trian\_df[‘Inicial’], train\_df[‘Pcalss’]], [train\_df[‘Sex’], train\_df[‘Survived’],

margins=all or True).T.style.background\_gradient(cmap=’summer\_r’)

• reset\_index(drop=True)

• .groupby([‘Age\_cat’], as\_index=True).groups ; .gorups.keys() ; .groups.values()

.groupby([‘Age\_cat’]).first() ; .count() ; mean()

.groupby([‘Age\_cat’])[[‘column1’]] ; 컬럼을 한번 더 대괄호로 묶으면 데이터프레임으로 나옴

• train\_df[‘Embarked’].dropna(axis=0, how=’all’, thresh=2, subset=[‘second score’,

fourth score’], inplace=True)

• pd.Series(data, index=[1,2,3])

• from pandas import Series ; Series(data, index=[1,2,3]) ; Series(dict)

Series(list) ; Series(dict)

• .columns

• ax[0].set\_title(‘~~’, y=1.02)

• .skew()

• .describe(include=’all’)

• .dtypes ; ()을 넣을 때는 각 행이 있어야 할 수 있다.(Series는 할 수 없음)

house\_df.dtypes[house\_df.dtypes !=

• .info()

• .any() ; np.any(value <= 0.6) 하나의 값이 참이면 참

• .all() ‘ np.all(value <= 0.6) 모든 값이 참이면 참

• .factorize()

• pd.cut(train[‘Age’], 8) ; 구간으로 나눈다.

• pd.cut(train[‘Age’], bins=[0, 1, 3, 10, 18, 65, 9, 9], labels=[‘Baby’, ‘Todler’, ‘Kid’, ‘Teens’, ‘adult’, ‘elderly’])

• pd.qcut(train[‘Age’], 8) ; 구간과 상관없이 갯수로 나re눈다.

• mask[np.triu\_indices\_from(mask)] = True

• pd.set\_option(“display.max\_columns”, 8)

• .sort\_index(axis=1)

• df[‘column’].isin([‘C50’]) ; df[‘column’].isin([‘C50’, ‘C85]) ;해당 값에 해당 글자가 포함되면 True

• .to\_frame() ; Series를 DataFrame으로 변환

• .nlargest(5)

• .insert(3, ‘New1’, 0) ; 새로운 컬럼을 원하는 열에 집어 넣음

• .str.lower()

• .str.upper()

• .replace([‘m2’, ‘w2’, ‘m1’], [‘1,2,3’], inplace=True)

• data.astype({‘Age’:’int’})

• data[‘CustomerID’].astype(int)

• .get\_group(‘adult’)

• temp['Age'] = temp.groupby('Initial')['Age'].apply(lambda x: x.fillna(x.mean())) ; 빈칸채우기

• .ljust(10) ; 왼쪽 정렬

• .rjust(10) ; 오른쪽 정렬

• a = lambda x, y : x\*\*y ; a(2,3) ; z = (lambda x,y : x+y) (10,20)

• 2 % 1 나머지 , // 몫

• filter(function, iterable) ; list(filter(lambda x, y : x+y, range(-5,10)))

• gbc\_imp.merge(xgb\_imp, on='Feature') ; key를 중심으로 join한다.(vlookup 같은것)

• pd.merge(df1, df2)

• np.sqrt()

• Series.str.contains(‘oh|son’) ; 특정 문자열을 포함하는 요소 찾아줌, True/False

• Series.str.startswith() ; 특정 문자열이 처음에 포함되는지

• Series.str.endwith() ; 특정 문자열이 마지막에 포함되는지

• list = set(list) ; 리스트 중복제거

• a.keys() ; 딕셔너리 키값을 list형태로 변환

• .reshape(-1,1) ; Series 형태처럼 변환을 시켜주는 것

• .append() ; 데이터 프레임끼리도 append가 됨

append(ignore\_index=True) ; Series가 아닌 것으로 넣을 경우

list.append() ; list의 경우 list = list.append() 하지 않고 list.append() 한다.

• import datetime as dt

• retail\_df.groupby(‘CustomerID’).agg(aggreegations) ;

aggreegations = {'duration':[min, max, sum],

'network\_type':'count',

'date':['first', lambda x: max(x)-min(x)]}

• sns.heatmap(corr, annot=True, fmt=’ .1g’)

• pred\_proba = lr\_clf.predict\_proba(X\_test)

• import json

- with open(‘~~~.json’, ‘w’) as st\_json: ; 인코딩

st\_json = json.dump(data, indent=4, sort\_keys=True) ; indent=들여쓰기, sort\_keys=키로 정렬

- with open(‘~~~’, ‘r’) as st\_json: ; 디코딩

st\_python = json.loads(st\_json)

• df.set\_index(‘column’)

• df.drop\_duplicates(‘column’, keep=’first’ ; ‘last’ ; ‘False’) ; 1개만 남기가 중복 제거

• df.duplicated([‘key1’, ‘key2’], keep=’first’ ; ‘last’ ; ‘False’) ; 중복값 확인

• array = array.extend(array2) ; 리스트 합칠 때 사용

• list.index(‘a’) ; list.index(value, start, end) ; 리스트에서 원하는 문자의 인덱스 찾기

• df[‘column’].value\_counts().sort\_values(ascending=True).plot.bar(ax=ax[0,0]]) ; 컬럼 1개

df[[‘column1’, ‘column2’]].groupby([‘column1’], as\_index=True).count().plot.bar() ; 컬럼 2개 이상

sns.barplot(data=df, x=’column1’, y=’column2’, ax=ax[0,0]) ; 그래프 그리는 3가지 방법

• list.isalnum() ; 영문자나 숫자로 되어 있으면 True

list.isalpha() ; 영문자로 되어 있으면 True

list.isdigit() ; 숫자로 되어 있으면 True

• df.iloc[[0]] ; df.iloc[:, [0]] ; iloc에서 대괄호를 두번하면 Series가 아닌 DataFrame으로 나온다.

df.loc[df[‘column’] == ‘Suger beet’, [‘Area’]]

• .str.startswith(‘keyword’) ; 처음에 keyword가 나오면 True

.str.endswith(‘keyword’) ; 마지막에 keyword가 나오면 True

• raw\_phone.groupby(['month', 'item']).agg(aggregation\_logic).columns.droplevel(level=0, axis=0)

• df.set\_index(‘column1’, drop=False, inplace=True) ; 해당 컬럼을 인덱스로 지정하기

df.set\_index(‘column2’, append=True) ; 인덱스 추가하기

df.loc[‘Bob’] ; df.at[‘Bob’, ‘age’] ; 값 추출

• sort\_index(inplace=True) ; 중복된 인덱스가 제거됨

• df.asfreq(‘H’) ; df.asfreq(‘W’) ; df.asfreq(‘D’) ; 인덱스의 날짜가 나눠짐

**■ 오분류표 해석**

• 정확도(Accuracy) = TN + TP / TN + FP + FN + TP

• 정밀도(Precision) = TP / FP + TP

- 실제 Negative 음성인 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 상대적으로 더 중요한 지표이다. (스팸 판단 모델)

• 재현율(Recall), 민감도 = TP / FN + TP

- 실제 Positive 양성인 데이터 예측을 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 상대적으로 더 중요한 지표이다.(암 판단 모델 / 보험사기, 금융사기 적발 모델)

- 보통 재현율이 정밀도보다 상대적으로 중요한 업무가 많다.