Clustering

- K-mean clustering
  - Scaler and Inverse Scaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data[0])
#scaler
trans_data = scaler.transform(data[0])
#inverse scaler
org_data = scaler.inverse_transform(trans_data)
```

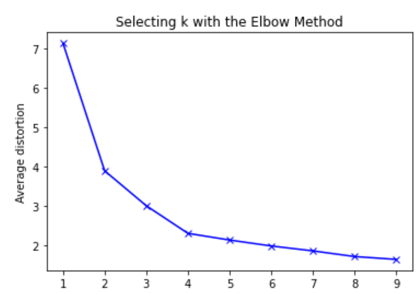
  - Create the clusters.

```
from sklearn.cluster import KMeans
#created 4 clusters
kmeans = KMeans(n_clusters=4)
#fitting model
kmeans.fit(data[0])
#show center coordinate of each group
kmeans.cluster_centers_
#show group label of each point
kmeans.labels_
#prediction
kmeans.predict(data[0])
```

  - Elbow method

```
import numpy as np
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt
K = range(1, 10)
meandistortions = []
for k in K:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(data[0])
     meandistortions.append(sum(np.min(cdist(data[0],
     kmeans.cluster_centers_, 'euclidean'),
     axis=1)) / data[0].shape[0])

plt.plot(K, meandistortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Average distortion')
plt.title('Selecting k with the Elbow Method')
plt.show()
```



So, the best n_cluster is 4

Association Rules

- Apriori

```
from mlxtend.frequent_patterns import apriori
# Build up the frequent items , basket_sets is a one hot vector
frequent_itemsets = apriori(basket_sets, min_support=0.07, use_colnames=True)
```

- Association rules

```
from mlxtend.frequent_patterns import association_rules
# Create the rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
# Display the rules
rules[ (rules['lift'] >= 6) & (rules['confidence'] >= 0.8) ]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.094388	0.096939	0.079082	0.837838	8.642959	0.069932	5.568878
3	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.096939	0.094388	0.079082	0.815789	8.642959	0.069932	4.916181
17	(SET/6 RED SPOTTY PAPER PLATES)	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.127551	0.132653	0.102041	0.800000	6.030769	0.085121	4.336735
18	(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.137755	0.127551	0.122449	0.888889	6.968889	0.104878	7.852041
19	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.127551	0.137755	0.122449	0.960000	6.968889	0.104878	21.556122
20	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.102041	0.127551	0.099490	0.975000	7.644000	0.086474	34.897959
21	(SET/6 RED SPOTTY PAPER CUPS, SET/6 RED SPOTTY...	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.122449	0.132653	0.099490	0.812500	6.125000	0.083247	4.625850
22	(SET/20 RED RETROSPOT PAPER NAPKINS, SET/6 RED...	(SET/6 RED SPOTTY PAPER CUPS)	0.102041	0.137755	0.099490	0.975000	7.077778	0.085433	34.489796

CNN

- Cifar10
  - Generated random seed

```
# The below is necessary for starting Numpy generated random numbers
# in a well-defined initial state.

np.random.seed(42)

# The below is necessary for starting core Python generated random numbers
# in a well-defined state.

rn.seed(12345)
tf.random.set_seed(12345)
```

  - Define the model

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation = 'softmax'))
```

  - Define callback

```
# Define a ModelCheckpoint callback to persist the model after every epoch.
cp_callback = tf.keras.callbacks.ModelCheckpoint('checkpoint.hdf5', verbose=1,
    save_weights_only=True,monitor='val_accuracy', mode='max', save_best_only=True)
```

  - Compile model

```
# Compile, i.e. configure the network and the sparse_categorical_crossentropy loss.
# Furthermore, keep track of the model's accuracy metric and have it printed out during traini
ng.
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

  - Train model

```
# Actually train the model for 15 epochs show the model all the training data 15 times
# To be able to judge the model's performance, also provide the test data. It will be evaluate
d after
# every epoch and the results will be printed to the console.
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test),
   callbacks=[cp_callback])
```

- Prime Minister
  - Define model

```
# Three steps to Convolution
# 1. Convolution
# 2. Activation
# 3. Pooling
# Repeat Steps 1,2,3 for adding more hidden layers

# 4. After that make a fully connected network
# This fully connected network gives ability to the CNN
# to classify the samples

model = Sequential()

model.add(Conv2D(32, 6, input_shape=(90,75,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
BatchNormalization(axis=-1)
model.add(Conv2D(64, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
BatchNormalization(axis=-1)

model.add(Flatten())
model.add(Dense(128))
BatchNormalization()
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
```

  - Compile and Train model

```
model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
#set seed to produce same result
from numpy.random import seed
seed(4)
from tensorflow import set_random_seed
set_random_seed(20)

model.fit(X_train, Y_train,
          batch_size=256, epochs=20,
          verbose=1,
          validation_data=(X_test, Y_test))
```

RNN

- LSTM(Stock Price prediction)
    - Normalize and Generate Data

```
df = normalize_data(df)

def load_data(stock, seq_len):
    n_features = len(stock.columns) #count columns of stock df 4 features open/low/high/close
    data = stock.to_numpy()   #change to matrix numpy array
    sequence_length = seq_len + 1 #5+1
    result = []

    for index in range(len(data) - sequence_length): #1762 data but exclude sequence len , 0 
to 1762-6 => 1756
        result.append(data[index: index + sequence_length]) # construct table with 1756 2d 
arrays, each with [6,nfeature] (1756,6,4)

    result = np.array(result)
    row = round(0.9 * result.shape[0]) # 90% of 1756 data
    train = result[:int(row), :] # select first 90% as train data = 1580 data

    x_train = train[:, :-1] #(1580,5,4) , x is previous 5 days data with 4 columns
    y_train = train[:, -1][:,-1] #(1580), y is the close price of the sixth days

    x_test = result[int(row):, :-1]  # test is remaining rows 10% of data (176,5,4)
    y_test = result[int(row):, -1][:,-1]#(176,)

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], n_features)) # reshape 
again (1580,5,4)
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], n_features))

    return [x_train, y_train, x_test, y_test]

# set feature dimension and look back period
n_features = 4
prev_days = 5
X_train, y_train, X_test, y_test = load_data(df, prev_days)
```

- Build model

```
def build_model(layers):
    p = 0.2 #drop out 20%
    model = Sequential() #sequential type

    model.add(LSTM(256, input_shape=(layers[1], layers[0]), return_sequences=True)) # 5 previo
us days ,4 features
    model.add(Dropout(p)) #dropout between layer

    model.add(LSTM(256, input_shape=(layers[1], layers[0]), return_sequences=False))
    model.add(Dropout(p))

    model.add(Dense(128,activation='relu'))
    model.add(Dense(1,activation='linear'))

    model.compile(loss='mse',optimizer='adam', metrics=['mse'])

    return model

model = build_model([n_features, prev_days, 1])
```

- Train model

```
#setseed to produce same result
from numpy.random import seed
seed(5)
from tensorflow import set_random_seed
set_random_seed(20)

# Checkpoint for call back function
from keras.callbacks import ModelCheckpoint
filepath="/content/weights.best.hdf5" #for print only best model

checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mod
e='min')
callbacks_list = [checkpoint]

#validation split select 10% of data to validate (last 10% sequence)
history = model.fit(X_train, y_train, batch_size=32, epochs=20, validation_split=0.1, verbose=
1, callbacks =callbacks_list)
```

- Compile model

```
# Load weights
print(filepath)
model.load_weights(filepath)
# Compile model (required to make predictions)
model.compile(loss='mse',optimizer='adam', metrics=['mse'])
print("Created model and loaded weights from file")
```

- Denormalize prediction

```
df = pd.read_csv(datapath, index_col = 0)
df["mv close"] = df.close
df.drop(['volume', 'close'], 1, inplace=True)
df = df[df.symbol == SYM]
df.drop(['symbol'],1,inplace=True)

def denormalize(df, normalized_value):
    df = df['mv close'].values.reshape(-1,1)
    normalized_value = normalized_value.reshape(-1,1)

    min_max_scaler = preprocessing.MinMaxScaler()
    _ = min_max_scaler.fit_transform(df)
    denorm = min_max_scaler.inverse_transform(normalized_value)
    return denorm

new_pred = denormalize(df, predict)
newy_test = denormalize(df, y_test)
```

- SARIMAX(PM 2.5)
  - Fit model with exogenous data

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
exog_columns = ['Temp(C)']
best_order = (0, 1, 1)
best_seasonal_order = (1, 1, 0, 12)
mod = SARIMAX(train['PM2.5(µg/m3)'],
              exog=train[exog_columns],
              order=best_order,
              seasonal_order=best_seasonal_order,
              enforce_stationarity=False,
              enforce_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])
```

  - Predict on test

```
# concatenate test and validation set for continuously prediction
test_exog = pd.concat((valid[exog_columns], test[exog_columns]), axis=0)
pred = results.get_prediction(start=test.index[0], end=test.index[-
1], exog=test_exog, dynamic=False)
pred_ci = pred.conf_int()
```