Phase 4 Code Challenge

This code challenge is designed to test your understanding of the Phase 4 material. It covers:

- Principal Component Analysis
- Clustering
- Time Series
- Natural Language Processing

Read the instructions carefully. You will be asked both to write code and to answer short answer questions.

Code Tests

We have provided some code tests for you to run to check that your work meets the item specifications. Passing these tests does not necessarily mean that you have gotten the item correct - there are additional hidden tests. However, if any of the tests do not pass, this tells you that your code is incorrect and needs changes to meet the specification. To determine what the issue is, read the comments in the code test cells, the error message you receive, and the item instructions.

Short Answer Questions

For the short answer questions...

- *Use your own words*. It is OK to refer to outside resources when crafting your response, but *do not copy text from another source*.
- *Communicate clearly*. We are not grading your writing skills, but you can only receive full credit if your teacher is able to fully understand your response.
- *Be concise*. You should be able to answer most short answer questions in a sentence or two. Writing unnecessarily long answers increases the risk of you being unclear or saying something incorrect.

In [2]:
```
! pip install sklearn
! pip install pandas
! pip install matplotlib
```

```
Collecting sklearn
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Collecting scikit-learn
  Downloading scikit_learn-0.24.2-cp39-cp39-manylinux2010_x86_64.whl (23.
8 MB)
     |████████████████████████████████| 23.8 MB 36.6 MB/s eta 0:00:01
Collecting joblib>=0.11
  Downloading joblib-1.0.1-py3-none-any.whl (303 kB)
     |████████████████████████████████| 303 kB 121.5 MB/s eta 0:00:01
Collecting scipy>=0.19.1
  Downloading scipy-1.7.1-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_6
4.whl (28.5 MB)
     |████████████████████████████████| 28.5 MB 67.2 MB/s eta 0:00:01
|████████████████████            | 18.0 MB 67.2 MB/s eta 0:00:01
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.2.0-py3-none-any.whl (12 kB)
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.9/
site-packages (from scikit-learn->sklearn) (1.21.2)
Building wheels for collected packages: sklearn
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl si
ze=1316 sha256=9e8e66de73baffe2303cb2c1d46b1b52fad5c71369f67df687a305ee85
b29b46
  Stored in directory: /home/jovyan/.cache/pip/wheels/e4/7b/98/b6466d71b8
d738a0c547008b9eb39bf8676d1ff6ca4b22af1c
Successfully built sklearn
Installing collected packages: threadpoolctl, scipy, joblib, scikit-lear
n, sklearn
Successfully installed joblib-1.0.1 scikit-learn-0.24.2 scipy-1.7.1 sklea
rn-0.0 threadpoolctl-2.2.0
Collecting pandas
  Downloading pandas-1.3.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (11.5 MB)
     |████████████████████████████████| 11.5 MB 49.6 MB/s eta 0:00:01
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/p
ython3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/lib/python3.9/
site-packages (from pandas) (1.21.2)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.9/s
ite-packages (from pandas) (2021.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.9/site-
packages (from python-dateutil>=2.7.3->pandas) (1.16.0)
Installing collected packages: pandas
Successfully installed pandas-1.3.2
Collecting matplotlib
  Downloading matplotlib-3.4.3-cp39-cp39-manylinux1_x86_64.whl (10.3 MB)
     |████████████████████████████████| 10.3 MB 36.7 MB/s eta 0:00:01
Requirement already satisfied: numpy>=1.16 in /opt/conda/lib/python3.9/si
te-packages (from matplotlib) (1.21.2)
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python
3.9/site-packages (from matplotlib) (2.4.7)
```

```
Collecting pillow>=6.2.0
  Downloading Pillow-8.3.1-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_
64.whl (3.0 MB)
     |████████████████████████████████| 3.0 MB 87.7 MB/s eta 0:00:01
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/pyt
hon3.9/site-packages (from matplotlib) (2.8.2)
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.1-cp39-cp39-manylinux1_x86_64.whl (1.2 MB)
     |████████████████████████████████| 1.2 MB 104.3 MB/s eta 0:00:01
Requirement already satisfied: six in /opt/conda/lib/python3.9/site-packa
ges (from cycler>=0.10->matplotlib) (1.16.0)
Installing collected packages: pillow, kiwisolver, cycler, matplotlib
Successfully installed cycler-0.10.0 kiwisolver-1.3.1 matplotlib-3.4.3 pi
llow-8.3.1
```

In [3]:
```python
# Run this cell without changes to import the necessary libraries

from numbers import Number
import matplotlib, sklearn, scipy, pickle
import numpy as np
import pandas as pd

%matplotlib inline
```

Part 1: Principal Component Analysis [Suggested Time: 15 minutes]

In this part, you will use Principal Component Analysis on the wine dataset.

In [4]:
```
# Run this cell without changes

# Relevant imports
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load data
wine = load_wine()
X, y = load_wine(return_X_y=True)
X = pd.DataFrame(X, columns=wine.feature_names)
y = pd.Series(y)
y.name = 'class'

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra

# Scaling
scaler_1 = StandardScaler()
X_train_scaled = pd.DataFrame(scaler_1.fit_transform(X_train), columns=X_tr

# Inspect the first five rows of the scaled dataset
X_train_scaled.head()
```

Out[4]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflav
0	-1.104538	-0.530902	-0.136257	-0.374157	-1.294014	-1.017096	-0.444344	
1	-0.608849	-0.792240	-0.573221	-0.217310	4.793609	0.421716	0.331268	
2	1.170548	-0.471890	1.611596	-0.091832	0.660038	1.141122	1.036369	
3	-1.371448	1.559801	0.118638	0.410080	-1.218858	0.997241	1.096806	
4	-0.443619	0.000204	-0.573221	-0.374157	-0.316988	-0.985122	-1.290465	

1.1) Create a PCA object `wine_pca` and fit it using `X_train_scaled`.

Use parameter defaults with `n_components=0.9` and `random_state=1` for your classifier. You must use the Scikit-learn PCA (docs [here (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)).

Starter Code

```
wine_pca = PCA(
```

In [7]:
```
# your code here
wine_pca = PCA(n_components=0.9, random_state=1)
wine_pca.fit(X_train_scaled)
```

Out[7]:
```
PCA(n_components=0.9, random_state=1)
```

In [8]:
```
# This test confirms that you have created a PCA object named wine_pca

assert type(wine_pca) == PCA

# This test confirms that you have set random_state to 1

assert wine_pca.get_params()['random_state'] == 1

# This test confirms that wine_pca has been fit

sklearn.utils.validation.check_is_fitted(wine_pca)
```

1.2) Create a numeric variable `wine_pca_ncomps` containing the number of components in `wine_pca`

Hint: Look at the list of attributes of trained `PCA` objects in the [scikit-learn documentation (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

Starter Code

```
wine_pca_ncomps =
```

In [13]:
```
# your code here
wine_pca_ncomps= wine_pca.n_components_
wine_pca_ncomps
```

Out[13]:
```
8
```

In [14]:
```
# This test confirms that you have created a numeric variable named wine_pc

assert isinstance(wine_pca_ncomps, Number)
```

1.3) Short Answer: Is PCA more useful or less useful when you have high multicollinearity among your features? Explain why.

PCA is is useful when you have high multicollinearity because by projecting the data into a new feature space and thereby reducing the dimensionality of the dataset, PCA ensures that the new components do not have strong correlations with each other and that the variance captured by multicollinear features is mostly preserved in the new components.

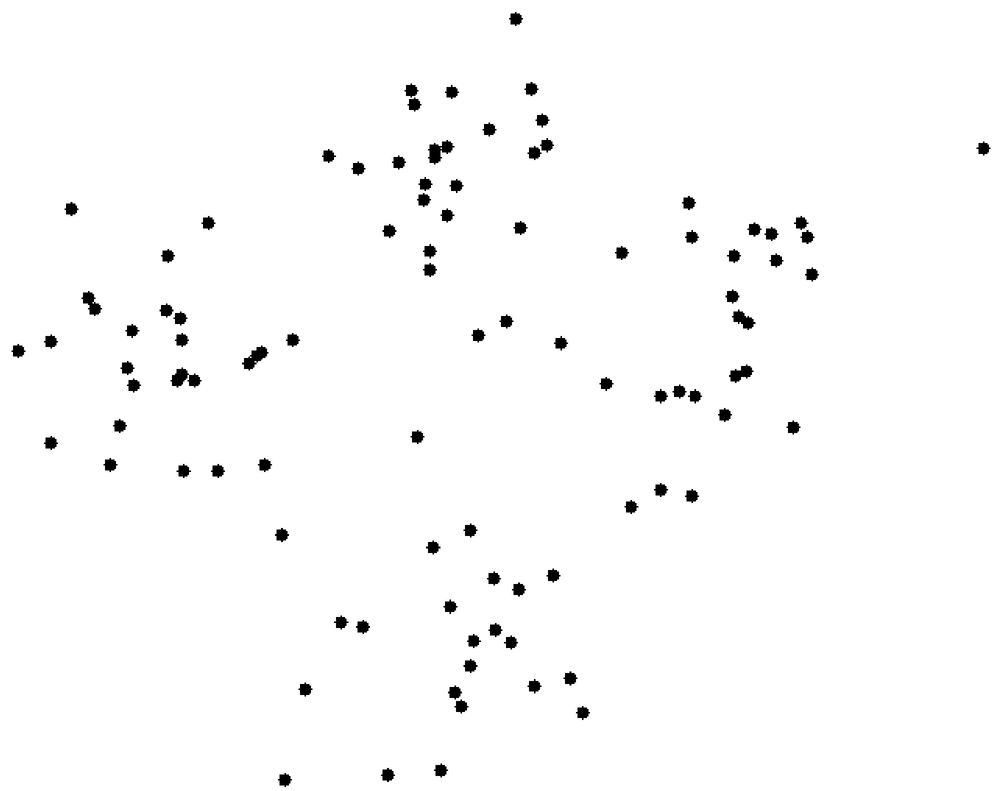
Part 2: Clustering [Suggested Time: 20 minutes]

In this part, you will answer general questions about clustering.

In [16]:
```
# Run this cell without changes

from sklearn.cluster import KMeans
```

2.1) Short Answer: Describe the steps of the k-means clustering algorithm.

Hint: Refer to the animation below, which visualizes the process.



YOUR ANSWER HERE

2.2) Write a function `get_labels()` that meets the requirements below to find `k` clusters in a dataset of features `X`, and return the cluster assignment labels for each row of `X`.

Review the doc-string in the function below to understand the requirements of this function.

Hint: Within the function, you'll need to:

- instantiate a [scikit-learn KMeans object (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html), using `random_state = 1` for reproducibility
- fit the object to the data
- return the cluster assignment labels for each row of `X`

Starter Code - replace None with appropriate code

```
def get_labels(k, X):
    """
    Finds the labels from a k-means clustering model

    Parameters:
    -----------
    k: float object
        number of clusters to use in the k-means clustering model
    X: Pandas DataFrame or array-like object
        Data to cluster

    Returns:
    --------
    labels: array-like object
        Labels attribute from the k-means model

    """

    # Instantiate a k-means clustering model with random_state=1 an
d n_clusters=k
    kmeans = None

    # Fit the model to the data
    None

    # Return the predicted labels for each row in the data produced
by the model
    return None
```

In [17]:
```
# your code here
def get_labels(k, X):
    """
    Finds the labels from a k-means clustering model

    Parameters:
    -----------
    k: float object
        number of clusters to use in the k-means clustering model
    X: Pandas DataFrame or array-like object
        Data to cluster

    Returns:
    --------
    labels: array-like object
        Labels attribute from the k-means model

    """

    # Instantiate a k-means clustering model with random_state=1 and n_clus
    kmeans = KMeans(n_clusters=k, random_state=1)

    # Fit the model to the data
    kmeans.fit(X)

    # Return the predicted labels for each row in the data produced by the
    return kmeans.labels_
```

In [18]:
```
# This test confirms that you have created a function named get_labels

assert callable(get_labels)

# This test confirms that get_labels can take the correct parameter types

get_labels(1, np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5]]))
```

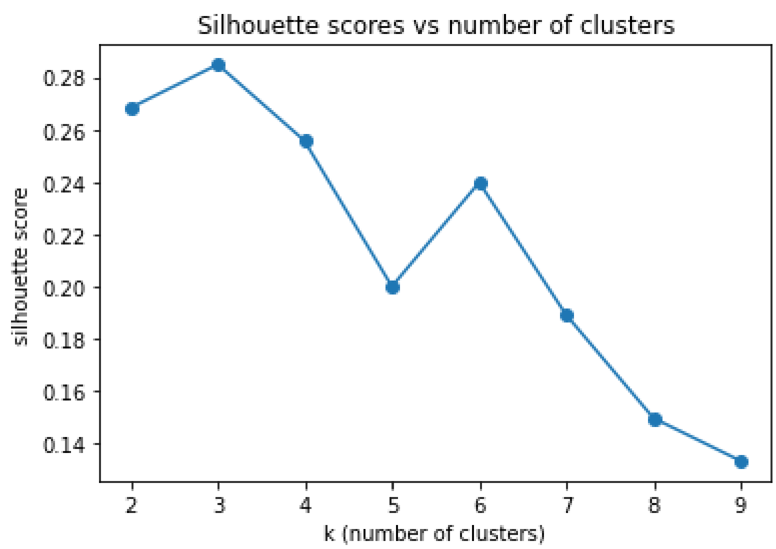
Out[18]:
```
array([0, 0, 0], dtype=int32)
```

The next cell uses your `get_labels` function to cluster the wine data, looping through all k values from 2 to 9. It saves the silhouette scores for each k value in a list `silhouette_scores`.

In [19]:
```
# Run this cell without changes

from sklearn.metrics import silhouette_score

# Preprocessing is needed. Scale the data
scaler_2 = StandardScaler()
X_scaled = scaler_2.fit_transform(X)

# Create empty list for silhouette scores
silhouette_scores = []

# Range of k values to try
k_values = range(2, 10)

for k in k_values:
    labels = get_labels(k, X_scaled)
    score = silhouette_score(X_scaled, labels, metric='euclidean')
    silhouette_scores.append(score)
```

Next, we plot the silhouette scores obtained for each different value of k, against k, the number of clusters we asked the algorithm to find.

In [20]:
```
# Run this cell without changes

import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(k_values, silhouette_scores, marker='o')
plt.title('Silhouette scores vs number of clusters')
plt.xlabel('k (number of clusters)')
plt.ylabel('silhouette score');
```



2.3) Create numeric variable `wine_nclust` containing the value of k you would choose based on the above plot of silhouette scores.

Starter Code

```
wine_nclust =
```

In [21]:
```
# your code here
wine_nclust=3
```

In [22]:
```
# This test confirms that you have created a numeric variable named wine_nc

assert isinstance(wine_nclust, Number)
```

Part 3: Natural Language Processing [Suggested Time: 20 minutes]

In this third section we will attempt to classify text messages as "SPAM" or "HAM" using TF-IDF Vectorization.

In [24]:
```
! pip install nltk
```

```
Collecting nltk
  Downloading nltk-3.6.2-py3-none-any.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 34.7 MB/s eta 0:00:01
Requirement already satisfied: joblib in /opt/conda/lib/python3.9/site-pa
ckages (from nltk) (1.0.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.9/site-pack
ages (from nltk) (4.61.2)
Collecting regex
  Downloading regex-2021.8.3-cp39-cp39-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl (732 kB)
     |████████████████████████████████| 732 kB 92.7 MB/s eta 0:00:01
Collecting click
  Downloading click-8.0.1-py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 940 kB/s s eta 0:00:01
Installing collected packages: regex, click, nltk
Successfully installed click-8.0.1 nltk-3.6.2 regex-2021.8.3
```

In [25]:
```
# Run this cell without changes

# Import necessary libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
import string
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize

# Generate a list of stopwords
nltk.download('stopwords')
stops = stopwords.words('english') + list(string.punctuation)

# Read in data
df_messages = pd.read_csv('./spam.csv', usecols=[0,1])

# Convert string labels to 1 or 0
le = LabelEncoder()
df_messages['target'] = le.fit_transform(df_messages['v1'])

# Examine our data
print(df_messages.head())

# Separate features and labels
X = df_messages['v2']
y = df_messages['target']

# Create test and train datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5,
```

```
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

     v1                                                 v2  target
0   ham  Go until jurong point, crazy.. Available only ...       0
1   ham                      Ok lar... Joking wif u oni...       0
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...       1
3   ham  U dun say so early hor... U c already then say...       0
4   ham  Nah I don't think he goes to usf, he lives aro...       0
```

3.1) Create CSR matrices `tf_idf_train` and `tf_idf_test` by using a `TfidfVectorizer` with stop word list `stops` to vectorize `X_train` and `X_test`, respectively.

Besides using the stop word list, use paramater defaults for your TfidfVectorizer. Refer to the documentation about [TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html).

Starter Code

```
vectorizer =

tf_idf_train =

tf_idf_test =
```

In [27]:
```
# your code here
vectorizer = TfidfVectorizer(stop_words=stops)
tf_idf_train = vectorizer.fit_transform(X_train)
tf_idf_test = vectorizer.transform(X_test)
```

In [28]:
```
# These tests confirm that you have created CSR matrices tf_idf_train and t

assert type(tf_idf_train) == scipy.sparse.csr.csr_matrix
assert type(tf_idf_test) == scipy.sparse.csr.csr_matrix
```

3.2) Create an array `y_preds` containing predictions from an untuned `RandomForestClassifier` that uses `tf_idf_train` and `tf_idf_test`.

Use parameter defaults with `random_state=1` for your classifier. Refer to the documentation on [RandomForestClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html).

Starter Code

```
classifier =

y_preds =
```

In [30]:
```
# your code here

classifier = RandomForestClassifier(random_state=1)
classifier.fit(tf_idf_train, y_train)
y_preds = classifier.predict(tf_idf_test)
```

In [31]:
```
# This test confirms that you have created an array named y_preds

assert type(y_preds) == np.ndarray
```

3.3) Short Answer: What would it mean if the word "genuine" had the highest TF-IDF value of all words in one document from our test data?

It would mean that "genuine" frequently appeared in the one document and that it wasn't a common word in the others.

Part 4: Time Series [Suggested Time: 20 minutes]

In this part you will analyze the price of one stock over time. Each row of the dataset has four prices tracked for each day:

- Open: The price when the market opens.
- High: The highest price over the course of the day.
- Low: The lowest price over the course of the day.
- Close: The price when the market closes.

In [32]:
```
# Run this cell without changes

stocks_df = pd.read_csv('./stocks_5yr.csv')
stocks_df.head()
```

Out[32]:

	open	high	low	close	date
0	15.07	15.12	14.63	14.75	February 08, 2013
1	14.89	15.01	14.26	14.46	February 11, 2013
2	14.45	14.51	14.10	14.27	February 12, 2013
3	14.30	14.94	14.25	14.66	February 13, 2013
4	14.94	14.96	13.16	13.99	February 14, 2013

4.1) For `stocks_df`, create a DatetimeIndex from the `date` column.

The resulting DataFrame should not have a `date` column, only `open`, `high`, `low`, and `close` columns.

Hint: First convert the `date` column to Datetime datatype, then set it as the index.

Starter Code

```
stocks_df['date'] =
```

In [33]:
```
stocks_df['date'] = pd.to_datetime(stocks_df.date)
stocks_df.set_index('date', inplace=True)
```

In [34]:
```
# This test confirms that stocks_df has a DatetimeIndex

assert type(stocks_df.index) == pd.core.indexes.datetimes.DatetimeIndex

# This test confirms that stocks_df only has `open`, `high`, `low`, and `cl

assert list(stocks_df.columns) == ['open', 'high', 'low', 'close']
```

4.2) Create a DataFrame `stocks_monthly_df` that resamples `stocks_df` each month with the 'MS' DateOffset to calculate the mean of the four features over each month.

Refer to the [resample documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.resample.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.resample.html).

Starter Code

```
stocks_monthly_df =
```

In [38]:
```
stocks_monthly_df = stocks_df.resample('MS').mean()
```

In [39]:
```
# This test confirms that you have created a DataFrame named stocks_monthly

assert type(stocks_monthly_df) == pd.DataFrame

# This test confirms that stocks_monthy_df has the correct dimensions

assert stocks_monthly_df.shape == (61,4)
```

4.3) Create a matplotlib figure `rolling_open_figure` containing a line graph that visualizes the rolling quarterly mean of open prices from `stocks_monthly_df`.

You will use this graph to determine whether the average monthly open stock price is stationary or not.

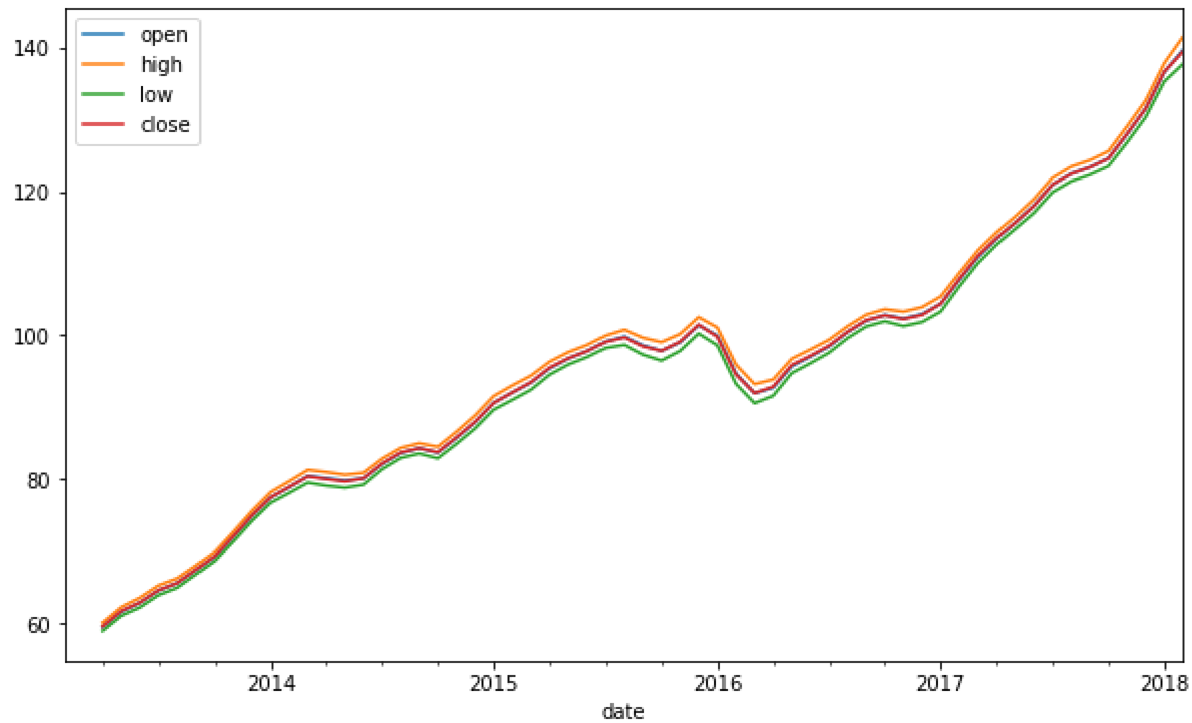
Hint: use a window size of 3 to represent one quarter of a year

Starter Code

```
rolling_open_figure, ax = plt.subplots(figsize=(10, 6))
```

In [44]:
```
rolling_open_figure, ax = plt.subplots(figsize=(10,6))
stocks_monthly_df.rolling(window=3).mean().plot(ax=ax)
```

Out[44]: <AxesSubplot:xlabel='date'>



In [42]:
```
# This test confirms that you have created a figure named rolling_open_figu

assert type(rolling_open_figure) == plt.Figure

# This test confirms that the figure contains exactly one axis

assert len(rolling_open_figure.axes) == 1
```

4.4) Short Answer: Based on your graph from Question 4.3, does the monthly open stock price look stationary? Explain your answer.

No, the montly open stock price has an upwards trend which indicates that it's not stationary.

In [ ]:
```

```