In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

In [2]:
```python
# .rolling(30).mean()
# sns.set()
# '{:.0%}.format(n) for n in...'
```

1) Suppose that Tolya likes avocados and decides to look at the price dynamics of this product in the US. There is the following data set from the Hass Avocado Board 🥑: Date - date. AveragePrice is the average price of an avocado. Total Volume - number of avocados sold 4046 - number of avocados sold PLU *4046 4225 - number of avocados sold PLU 4225 4770 - number of avocados sold PLU 4770 Total Bags - Total Bags Small Bags - small bags Large Bags - large XLarge Bags - very large Type - Regular or Organic year - year Region - the city or region of the destination*PLU - product lookup code *Dataframe does not contain data for each day, but for the end of each week. For each date, there are several observations differing in terms of type of avocado and region of sale. Suppose we are not interested in this separation, so the avocado_mean records the aggregated data, where: avocado_mean = pd.read_csv("path", index_col=0, squeeze=True, parse_dates=['Date'])

```
Date
2015-01-04 1.301296
2015-01-11 1.370648
2015-01-18 1.391111
2015-01-25 1.397130
2015-02-01 1.247037
Name: AveragePrice, dtype: float64
```

In [3]:
```python
avocado_mean = pd.read_csv("C:/Users/stask/Analitics_Karpov/Module6/Lesson/avocado_mean.csv",
                           index_col=0, parse_dates=['Date'])
avocado_mean.head()
```

Out[3]:

Date	AveragePrice
2015-01-04	1.301296
2015-01-11	1.370648
2015-01-18	1.391111
2015-01-25	1.397130
2015-02-01	1.247037

2) Calculate the moving average of the avocado price (AveragePrice) with a window of 3 (window). What is the maximum value? Round up the answer to 2 decimal places.

In [19]:
```python
avocado_mean.rolling(3).mean().round(2).head()
```

Out[19]:

Date	AveragePrice
2015-01-04	NaN
2015-01-11	NaN
2015-01-18	1.35
2015-01-25	1.39
2015-02-01	1.35

3) Great! In addition to avocados, Tolya likes to look at graphs. Now it is time to see how the graph will change depending on the chosen window size. Your task is to plot a moving average with different values of the window parameter (2, 4, 10, 50), look at the changes, and relate the pictures below to the corresponding window size. (task3.png)

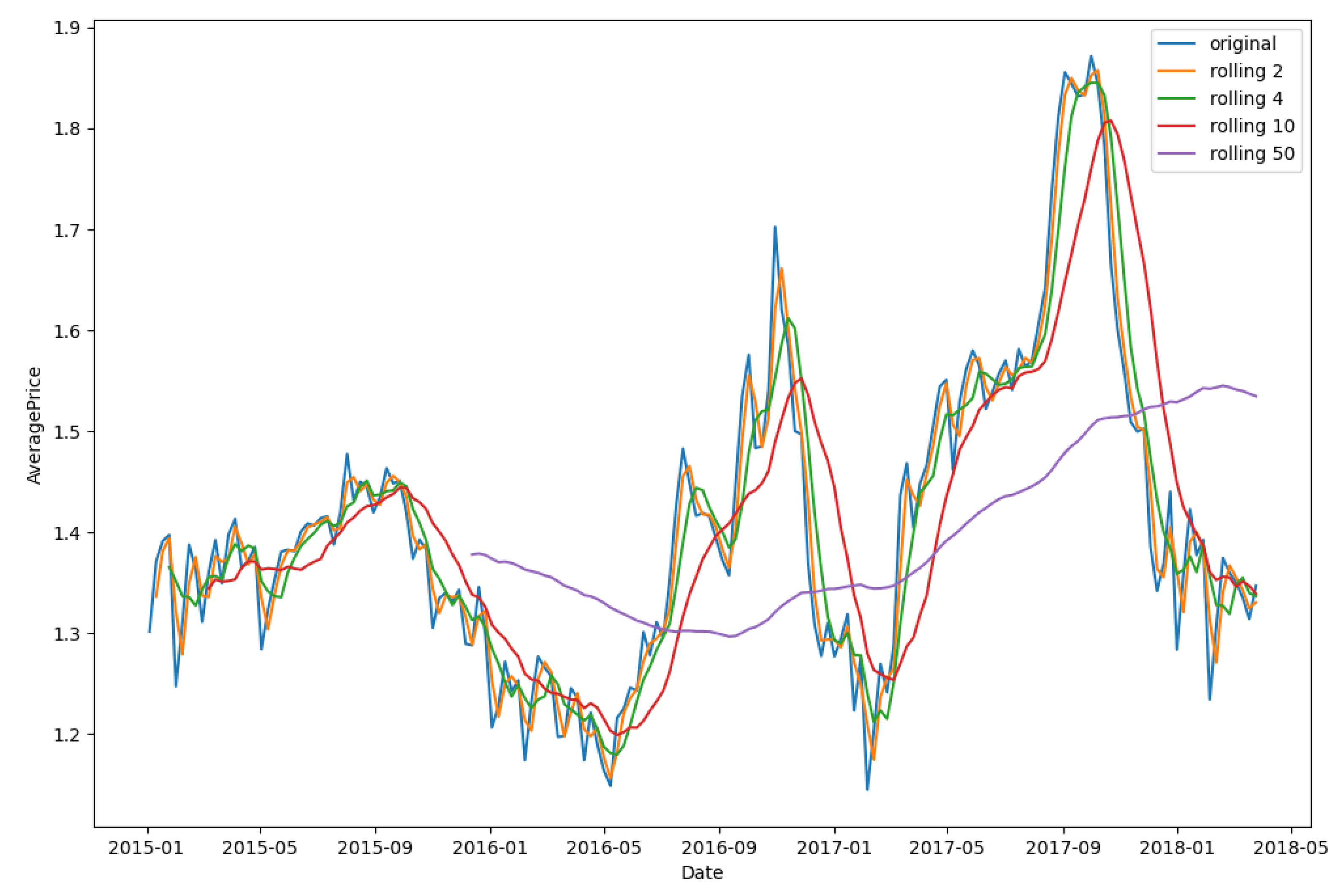
In [5]:
```python
avo_mean_2 = avocado_mean.rolling(2).mean()
avo_mean_4 = avocado_mean.rolling(4).mean()
avo_mean_10 = avocado_mean.rolling(10).mean()
avo_mean_50 = avocado_mean.rolling(50).mean()
```

In [6]:
```python
plt.figure(figsize=(12,8))

sns.lineplot(x=avocado_mean.index, y=avocado_mean.AveragePrice, label='original')
sns.lineplot(x=avo_mean_2.index, y=avo_mean_2.AveragePrice, label='rolling 2')
sns.lineplot(x=avo_mean_4.index, y=avo_mean_4.AveragePrice, label='rolling 4')
sns.lineplot(x=avo_mean_10.index, y=avo_mean_10.AveragePrice, label='rolling 10')
sns.lineplot(x=avo_mean_50.index, y=avo_mean_50.AveragePrice, label='rolling 50')

plt.legend()
```

Out[6]: `<matplotlib.legend.Legend at 0x25b66dbfd00>`



5) ⭐Task with asterisk!⭐ Use the avocado_mean.csv aggregated avocado price data and apply ewm() with parameter span=2 to get an exponential moving average. Record the result in avocado_ewm.

In [20]:
```python
avocado_ewm = avocado_mean.ewm(span=2).mean()
avocado_mean.merge(avocado_ewm, on='Date').tail()
```

Out[20]:

Date	AveragePrice_x	AveragePrice_y
2018-02-25	1.359630	1.356212
2018-03-04	1.350185	1.352194
2018-03-11	1.335093	1.340793
2018-03-18	1.313704	1.322734
2018-03-25	1.346852	1.338812

6) ⭐Task with asterisk!⭐ And one more step to consolidate the material. Import complete data, (avocado_full.csv) specifying index_col=0 (use first column as index). For avocados of organic type in Chicago (region), calculate a moving average with window 4 and an exponential moving average with span=4. Plot the graphs and then fill in the blanks. Round up the numbers to three decimal places after the point.

In [8]:
```python
avocado_full = pd.read_csv("C:/Users/stask/Analitics_Karpov/Module6/Lesson/avocado_full.csv",
                           index_col=0,
                           parse_dates=['Date'])
avocado_full.describe()
```

Out[8]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	year
count	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	18249.000000	18249.000000
mean	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974e+04	2.396392e+05	1.821947e+05	5.433809e+04	3106.426507	2016.147899
std	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05	9.862424e+05	7.461785e+05	2.439660e+05	17692.894652	0.939938
min	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	2015.000000
25%	1.100000	1.083858e+04	8.540700e+02	3.008780e+03	0.000000e+00	5.088640e+03	2.849420e+03	1.274700e+02	0.000000	2015.000000
50%	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900e+02	3.974383e+04	2.636282e+04	2.647710e+03	0.000000	2016.000000
75%	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420e+03	1.107834e+05	8.333767e+04	2.202925e+04	132.500000	2017.000000
max	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439e+06	1.937313e+07	1.338459e+07	5.719097e+06	551693.650000	2018.000000

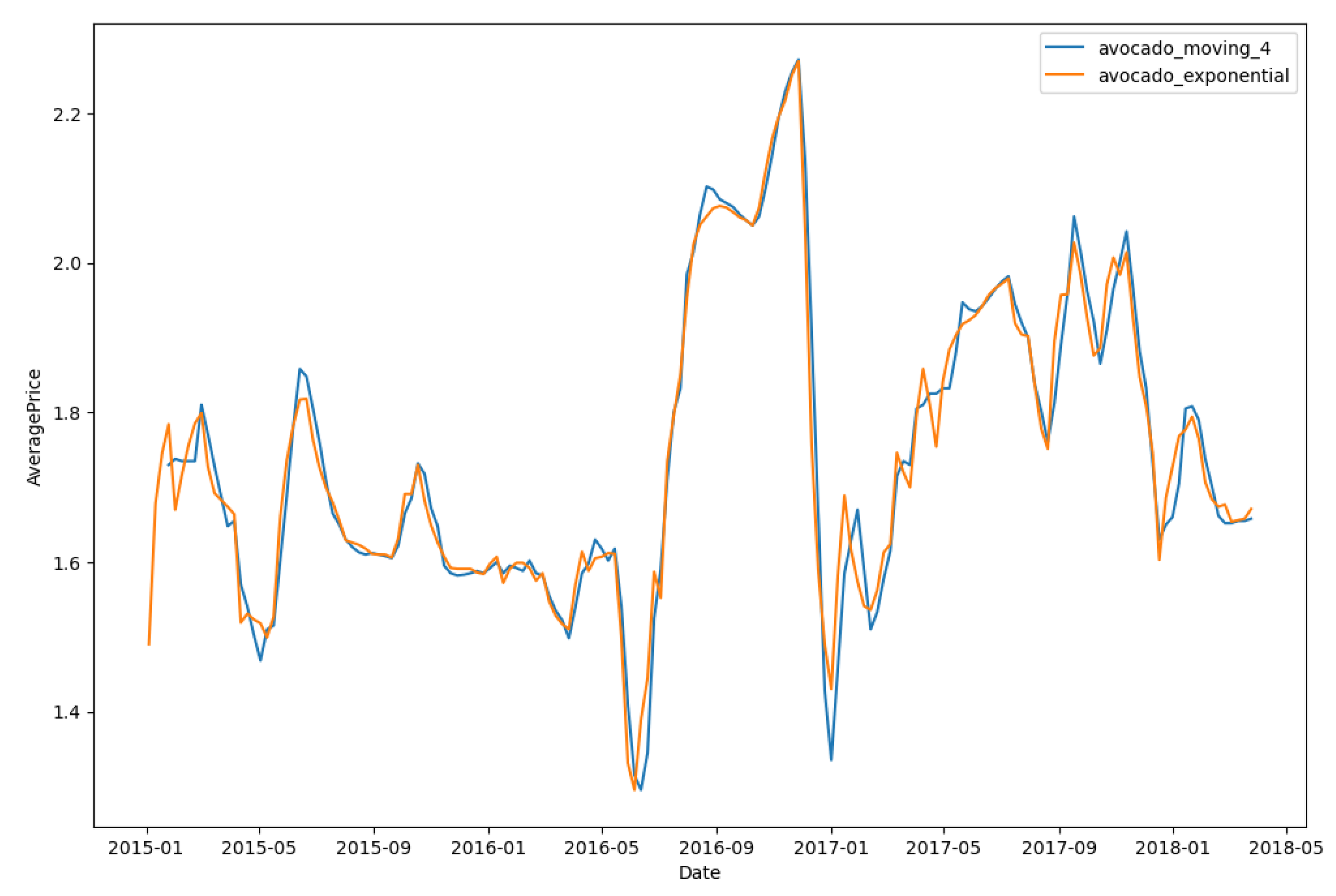
In [21]:
```python
avocado_organic_chicago = avocado_full.query('(type == "organic") and (region == "Chicago")').AveragePrice
avocado_organic_chicago.head()
```

Out[21]:
```
Date
2015-01-04    1.49
2015-01-11    1.79
2015-01-18    1.81
2015-01-25    1.83
2015-02-01    1.52
Name: AveragePrice, dtype: float64
```

In [10]:
```python
avocado_moving_4 = avocado_organic_chicago.rolling(4).mean().round(3).reset_index()
avocado_exponential = avocado_organic_chicago.ewm(span=4).mean().round(3).reset_index()

plt.figure(figsize=(12,8))
sns.lineplot(x=avocado_moving_4.Date, y=avocado_moving_4.AveragePrice, label='avocado_moving_4')
sns.lineplot(x=avocado_exponential.Date, y=avocado_exponential.AveragePrice, label='avocado_exponential')
```

Out[10]: `<AxesSubplot:xlabel='Date', ylabel='AveragePrice'>`



In [11]:
```python
avocado_organic_chicago.loc['2017-02-05':'2017-03-01']
```

Out[11]:
```
Date
2017-02-05    1.49
2017-02-12    1.53
2017-02-19    1.60
2017-02-26    1.69
Name: AveragePrice, dtype: float64
```

7) 💀 Difficult task! 💀 Next we're going to work with the data on delays in making deals! The delays file contains information about which companies made deals with whom and for how much, and most importantly how late they were in doing so. Here is the data sabbet

```
    client_idcompany_id delay revenue
0 2389 4240 -1 days -14:57:48.000000000 705436
1 7614 9544 -4 days -02:14:23.000000000 859266
2 2052 2427 -0 days -06:08:57.000000000 812416
3 9635 8054 -0 days -23:40:19.000000000 162312
4 8154 5503 -1 days -13:21:53.000000000 658844
```

Let's first convert delay to timedelta format. Note that the column will not translate that easily - you will have to remove the -. Save data to df variable, format the column as needed (hint - remove -) and convert it to timedelta type. May need: pd.to_timedelta() pd.Series.str

In [12]:
```python
df = pd.read_csv('C:/Users/stask/Analitics_Karpov/Module6/Lesson/delays.csv')
df.head()
```

Out[12]:

	client_id	company_id	delay	revenue
0	2389	4240	-1 days -14:57:48.000000000	705436
1	7614	9544	-4 days -02:14:23.000000000	859266
2	2052	2427	-0 days -06:08:57.000000000	812416
3	9635	8054	-0 days -23:40:19.000000000	162312
4	8154	5503	-1 days -13:21:53.000000000	658844

In [13]:
```python
# Remove the '-' character from the 'delay' column
df['delay'] = df['delay'].str.replace('-', '')

# Convert 'delay' column to pandas timedelta type
df['delay'] = pd.to_timedelta(df['delay'])

df.dtypes
```

Out[13]:
```
client_id               int64
company_id              int64
delay         timedelta64[ns]
revenue                 int64
dtype: object
```

8) Time is good, but you would also like to break it up into intervals. Split the time into 3 intervals and put the corresponding values in the delay_categorical column May need: pd.cut().

In [14]:
```python
# Calculate the intervals using pd.cut() and assign labels
intervals = pd.cut(df['delay'], bins=3, labels=['Interval 1', 'Interval 2', 'Interval 3'])

# Create 'delay_categorical' column with interval labels
df['delay_categorical'] = intervals
df.head(5)
```

Out[14]:

	client_id	company_id	delay	revenue	delay_categorical
0	2389	4240	1 days 14:57:48	705436	Interval 1
1	7614	9544	4 days 02:14:23	859266	Interval 3
2	2052	2427	0 days 06:08:57	812416	Interval 1
3	9635	8054	0 days 23:40:19	162312	Interval 1
4	8154	5503	1 days 13:21:53	658844	Interval 1

9) 💀 Difficult task! 💀 The breakdown is certainly not bad, but we want slightly more integer values and more human names. The arguments of the pd.cut function will help us with that! Redefine column delay_categorical so that the values in it are 'less than 1 day' - time from 0 to 1 day '1-2 days' - from 1 to 2 days '2 to 3 days' - 2 to 3 days 'more than 3 days' - more than 3 days For example, for the following example client_id company_id delayed revenue 14345 54631 2 days 13:34:45 1453 54434 54834 1 days 08:26:00 453245 34905 49834 0 days 00:26:03 14543 The result will be client_id company_id delay revenue delay_categorical 14345 54631 2 days 13:34:45 145345 2-3 days 54434 54834 1 days 08:26:00 453245 1-2 days 34905 49834 0 days 00:26:03 14543 less than 1 day May be needed: pd.cut() pd.to_timedelta()

In [16]:
```python
bins = [pd.to_timedelta('0d'),
        pd.to_timedelta('1d'),
        pd.to_timedelta('2d'),
        pd.to_timedelta('3d'),
        pd.to_timedelta('100d'),
        ]
labels = ['less than 1 day',
          'from 1 day to 2 days',
           'from 2 days to 3 days',
           'more than 3 days'
          ]
```

In [17]:
```python
df['delay_new'] = pd.cut(df.delay,
      bins = bins,
      labels = labels
      )
df.head()
```

Out[17]:

	client_id	company_id	delay	revenue	delay_categorical	delay_new
0	2389	4240	1 days 14:57:48	705436	Interval 1	from 1 day to 2 days
1	7614	9544	4 days 02:14:23	859266	Interval 3	more than 3 days
2	2052	2427	0 days 06:08:57	812416	Interval 1	less than 1 day
3	9635	8054	0 days 23:40:19	162312	Interval 1	less than 1 day
4	8154	5503	1 days 13:21:53	658844	Interval 1	from 1 day to 2 days

10) Perfect! Now build an interactive barplot with how often transactions are delayed, figure out the rarest and most frequent choices.

In [18]:
```python
delay_counts = df['delay_new'].value_counts()

# Create the interactive bar plot
fig = px.bar(delay_counts, x=delay_counts.index, y=delay_counts.values)

# Set plot title and axis labels
fig.update_layout(
    title="Frequency of Delay Categories",
    xaxis_title="Delay Categories",
    yaxis_title="Count"
)

# Show the plot
fig.show()
```