In [1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw8.ipynb")
```

CPSC 330 - Applied Machine Learning

Homework 8: Introduction to Computer vision and Time Series (Lectures 19 and 20)

Due date: see the Calendar.

Imports

In [2]:
```
from hashlib import sha1

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OrdinalEncoder, OneHotEncoder

from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import r2_score
```

Submission instructions

rubric={points:2}
Follow the homework submission instructions.

You may work in a group on this homework and submit your assignment as a group. Below are some instructions on working as a group.

- The maximum group size is 2.
- Use group work as an opportunity to collaborate and learn new things from each other.
- Be respectful to each other and make sure you understand all the concepts in the assignment well.
- It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline.
- You can find the instructions on how to do group submission on Gradescope here.

When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.
4. Make sure that the plots and output are rendered properly in your submitted file.
5. If the .ipynb file is too big and doesn't render on Gradescope, also upload a pdf or html in addition to the .ipynb.

Exercise 1: time series prediction

In this exercise we'll be looking at a dataset of avocado prices. You should start by downloading the dataset and storing it under the `data` folder. We will be forcasting average avocado price for the next week.

In [3]:
```
df = pd.read_csv("data/avocado.csv", parse_dates=["Date"], index_col=0)
df.head()
```

Out[3]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26

In [4]:
```
df.shape
```

Out[4]:
```
(18249, 13)
```

In [5]:
```
df["Date"].min()
```

Out[5]:
```
Timestamp('2015-01-04 00:00:00')
```

In [6]:
```
df["Date"].max()
```

Out[6]:
```
Timestamp('2018-03-25 00:00:00')
```

It looks like the data ranges from the start of 2015 to March 2018 (~2 years ago), for a total of 3.25 years or so. Let's split the data so that we have a 6 months of test data.

In [7]:
```
split_date = '20170925'
df_train = df[df["Date"] <= split_date]
df_test  = df[df["Date"] >  split_date]
```

In [8]:
```
assert len(df_train) + len(df_test) == len(df)
```

1.1 How many time series?

rubric={points:4}

In the Rain in Australia dataset from lecture demo, we had different measurements for each Location.

We want you to consider this for the avocado prices dataset. For which categorical feature(s), if any, do we have separate measurements? Justify your answer by referencing the dataset.

Solution_1.1

Points: 4

For two categorical features, we have separate measurements for the same date: *type* and *region*.

Let take the Date of 2015-12-27 as an example. we can see there are 108 measurements with different combination of the values of *type* and *region* in this single day. There are 54 diffferent regions in the record, and 2 types of avocado.

In [9]:
```
...
df_train.head()
```

Out[9]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26

In [10]:

```
...
df_train[df_train["Date"] == "2015-12-27"]
```

Out[10]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	
0	2015-12-27	0.99	386100.49	292097.36	27350.92	297.90	66354.31	4
0	2015-12-27	1.17	596819.40	40450.49	394104.02	17353.79	144911.10	14
0	2015-12-27	0.97	62909.69	30482.25	2971.94	5894.40	23561.10	2
0	2015-12-27	1.13	450816.39	3886.27	346964.70	13952.56	86012.86	8
...	...	...	...	...	...	...	...	
0	2015-12-27	1.54	1652.19	0.00	73.22	0.00	1578.97	
0	2015-12-27	1.63	2161.84	874.75	17.54	0.00	1269.55	
0	2015-12-27	1.52	549787.59	89709.92	206198.62	5836.04	248043.01	14
0	2015-12-27	1.46	142710.36	29880.32	48416.71	38.63	64374.70	1
0	2015-12-27	1.81	7155.63	1478.79	2629.64	14.10	3033.10	

108 rows × 13 columns

In [11]:
```
...
print(len(df_train[df_train["Date"] == "2015-12-27"]))
print(len(df_train['region'].unique()))
print(len(df_train['type'].unique()))
```

```
108
54
2
```

1.2 Equally spaced measurements?

rubric={points:4}

In the Rain in Australia dataset, the measurements were generally equally spaced but with some exceptions. How about with this dataset? Justify your answer by referencing the dataset.

Solution_1.2

Points: 4

Here to check if there are equally spaced measurements, I go through the time intervals in the different combinations in the two categorical variables which causes multiple time series in our data, rather than compare the time intervals for each of these two variables separately. In the first five attempt, we do see they are equally spaced for these measurements. Then, I further check time intervals in the other combinations, which shows we do have one unequally spcaded measurements: Group ('organic', 'WestTexNewMexico'), which also has a time interval of 14 days and an interval of 21 days.

In [12]:
```
...
# reference: Lecture 20

# here I check the different combinations of two categorical variables that cau
# particularly in this single comparison I fix the type and see the difference
count = 0
for name, group in df_train.groupby(['type', 'region']):
    print("%-30s %s" % (name, group["Date"].sort_values().diff().value_counts()
    print('\n\n')
    count += 1
    if count == 5:
        break
```

```
('conventional', 'Albany')     Date
7 days    142
Name: count, dtype: int64
```

```
('conventional', 'Atlanta')    Date
7 days    142
Name: count, dtype: int64
```

```
('conventional', 'BaltimoreWashington') Date
7 days    142
Name: count, dtype: int64
```

```
('conventional', 'Boise')      Date
7 days    142
Name: count, dtype: int64
```

```
('conventional', 'Boston')     Date
7 days    142
Name: count, dtype: int64
```

In [13]:
```
...
# reference: chatGPT
for name, group in df_train.groupby(['type', 'region']):

    intervals = group['Date'].sort_values().diff().dropna().unique()
    if not len(intervals) == 1 and intervals[0] == pd.Timedelta('7 days'):
        print(f"Group {name} has not only 7 days interval.")
```

```
Group ('organic', 'WestTexNewMexico') has not only 7 days interval.
```

In [14]:
```
...
# reference: chatGPT
for name, group in df_train.groupby(['type', 'region']):
    intervals = group['Date'].sort_values().diff().dropna()

    if len(intervals.unique()) > 1 or intervals.iloc[0] != pd.Timedelta('7 days
        print(f"Group {name} does NOT have only 7 days interval.")
        print("Time interval distribution:")
        print(intervals.value_counts())
        print("\n\n")
```

```
Group ('organic', 'WestTexNewMexico') does NOT have only 7 days interval.
Time interval distribution:
Date
7 days     137
14 days      1
21 days      1
Name: count, dtype: int64
```

In [15]:
```
...
```

Out[15]:
```
Ellipsis
```

1.3 Interpreting regions

rubric={points:4}

In the Rain in Australia dataset, each location was a different place in Australia. For this dataset, look at the names of the regions. Do you think the regions are also all distinct, or are there overlapping regions? Justify your answer by referencing the data.

Solution_1.3

Points: 4

By checking the unique values in the column 'region', we can see there are some overlapping regions: There are a value of 'TotalUS', which I think it might be the sum of other sub-regions, such as 'Northeast', 'Southeast', 'West', 'California', and these also involve the further-divided cities including 'Albany', 'Atlanta', and 'Boston' in the dataset.

In [16]:
```
...
df_train['region'].unique()
```

Out[16]:
```
array(['Albany', 'Atlanta', 'BaltimoreWashington', 'Boise', 'Boston',
       'BuffaloRochester', 'California', 'Charlotte', 'Chicago',
       'CincinnatiDayton', 'Columbus', 'DallasFtWorth', 'Denver',
       'Detroit', 'GrandRapids', 'GreatLakes', 'HarrisburgScranton',
       'HartfordSpringfield', 'Houston', 'Indianapolis', 'Jacksonville',
       'LasVegas', 'LosAngeles', 'Louisville', 'MiamiFtLauderdale',
       'Midsouth', 'Nashville', 'NewOrleansMobile', 'NewYork',
       'Northeast', 'NorthernNewEngland', 'Orlando', 'Philadelphia',
       'PhoenixTucson', 'Pittsburgh', 'Plains', 'Portland',
       'RaleighGreensboro', 'RichmondNorfolk', 'Roanoke', 'Sacramento',
       'SanDiego', 'SanFrancisco', 'Seattle', 'SouthCarolina',
       'SouthCentral', 'Southeast', 'Spokane', 'StLouis', 'Syracuse',
       'Tampa', 'TotalUS', 'West', 'WestTexNewMexico'], dtype=object)
```

In [17]:
```
...
```

Out[17]: Ellipsis

In [18]:
```
...
```

Out[18]: Ellipsis

We will use the entire dataset despite any location-based weirdness uncovered in the previous part.

We will be trying to forecast the avocado price. The function below is adapted from Lecture 19, with some improvements.

In [19]:
```
def create_lag_feature(df, orig_feature, lag, groupby, new_feature_name=None, 
    """
    Creates a new feature that's a lagged version of an existing one.

    NOTE: assumes df is already sorted by the time columns and has unique indic

    Parameters
    ----------
    df : pandas.core.frame.DataFrame
        The dataset.
    orig_feature : str
        The column name of the feature we're copying
    lag : int
        The lag; negative lag means values from the past, positive lag means va
    groupby : list
        Column(s) to group by in case df contains multiple time series
    new_feature_name : str
        Override the default name of the newly created column
    clip : bool
        If True, remove rows with a NaN values for the new feature

    Returns
    -------
    pandas.core.frame.DataFrame
        A new dataframe with the additional column added.

    """

    if new_feature_name is None:
        if lag < 0:
            new_feature_name = "%s_lag%d" % (orig_feature, -lag)
        else:
            new_feature_name = "%s_ahead%d" % (orig_feature, lag)

    new_df = df.assign(**{new_feature_name : np.nan})
    for name, group in new_df.groupby(groupby):
        if lag < 0: # take values from the past
            new_df.loc[group.index[-lag:],new_feature_name] = group.iloc[:lag]|
        else:       # take values from the future
            new_df.loc[group.index[:-lag], new_feature_name] = group.iloc[lag:]
```

```
    if clip:
        new_df = new_df.dropna(subset=[new_feature_name])

    return new_df
```

We first sort our dataframe properly:

In [20]:
```
df_sort = df.sort_values(by=["region", "type", "Date"]).reset_index(drop=True)
df_sort
```

Out[20]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sm B
0	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186
1	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036
2	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651
3	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82	10103
4	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18	9170
...	...	...	...	...	...	...	...	
18244	2018-02-25	1.57	18421.24	1974.26	2482.65	0.00	13964.33	13698
18245	2018-03-04	1.54	17393.30	1832.24	1905.57	0.00	13655.49	13401
18246	2018-03-11	1.56	22128.42	2162.67	3194.25	8.93	16762.57	16510
18247	2018-03-18	1.56	15896.38	2055.35	1499.55	0.00	12341.48	12114
18248	2018-03-25	1.62	15303.40	2325.30	2171.66	0.00	10806.44	10569

18249 rows × 13 columns

We then call `create_lag_feature`. This creates a new column in the dataset `AveragePriceNextWeek`, which is the following week's `AveragePrice`. We have set `clip=True` which means it will remove rows where the target would be missing.

In [21]:
```
df_hastarget = create_lag_feature(df_sort, "AveragePrice", +1, ["region", "type
df_hastarget
```

Out[21]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186
1	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036
2	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651
3	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82	10103
4	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18	9170
...	...	...	...	...	...	...	...	
18243	2018-02-18	1.56	17597.12	1892.05	1928.36	0.00	13776.71	13553
18244	2018-02-25	1.57	18421.24	1974.26	2482.65	0.00	13964.33	13698
18245	2018-03-04	1.54	17393.30	1832.24	1905.57	0.00	13655.49	13401
18246	2018-03-11	1.56	22128.42	2162.67	3194.25	8.93	16762.57	16510
18247	2018-03-18	1.56	15896.38	2055.35	1499.55	0.00	12341.48	12114

18141 rows × 14 columns

Our goal is to predict `AveragePriceNextWeek`.

Let's split the data:

In [22]:
```
df_train = df_hastarget[df_hastarget["Date"] <= split_date]
df_test  = df_hastarget[df_hastarget["Date"] >  split_date]
```

1.4 `AveragePrice` baseline

rubric={points}

Soon we will want to build some models to forecast the average avocado price a week in advance. Before we start with any ML though, let's try a baseline. Previously we used `DummyClassifier` or `DummyRegressor` as a baseline. This time, we'll do something else as a baseline: we'll assume the price stays the same from this week to next week. So, we'll set our prediction of

"AveragePriceNextWeek" exactly equal to "AveragePrice", assuming no change. That is kind of like saying, "If it's raining today then I'm guessing it will be raining tomorrow". This simplistic approach will not get a great score but it's a good starting point for reference. If our model does worse that this, it must not be very good.

Using this baseline approach, what R^2 do you get on the train and test data?

Solution_1.4

Points: 4

The R^2 on the train data is roughly 0.8286 using this baseline approach, and it is roughly 0.7632 on the test data.

In [23]:
```
train_r2 = r2_score(df_train['AveragePriceNextWeek'], df_train['AveragePrice'])

...
```

Out[23]:
```
Ellipsis
```

In [24]:
```
test_r2 = r2_score(df_test['AveragePriceNextWeek'], df_test['AveragePrice'])

...
```

Out[24]:
```
Ellipsis
```

In [25]:
```
...
train_r2
```

Out[25]:
```
0.8285800937261841
```

In [26]:
```
...
test_r2
```

Out[26]:
```
0.7631780188583048
```

In [27]:
```
assert not train_r2 is None, "Are you using the correct variable name?"
assert not test_r2 is None, "Are you using the correct variable name?"
assert sha1(str(round(train_r2, 3)).encode('utf8')).hexdigest() == 'b1136fe2a8
assert sha1(str(round(test_r2, 3)).encode('utf8')).hexdigest() == 'cc24d9a9b56
```

1.5 Forecasting average avocado price

rubric={points:10}

Now that the baseline is done, let's build some models to forecast the average avocado price a week later. Experiment with a few approachs for encoding the

date. Justify the decisions you make. Which approach worked best? Report your test score and briefly discuss your results.

Benchmark: you should be able to achieve R^2 of at least 0.79 on the test set. I got to 0.80, but not beyond that. Let me know if you do better!

Note: because we only have 2 splits here, we need to be a bit wary of overfitting on the test set. Try not to test on it a ridiculous number of times. If you are interested in some proper ways of dealing with this, see for example sklearn's TimeSeriesSplit, which is like cross-validation for time series data.

Solution_1.5

Points: 10

I build several random forest regressor models with different preprocessing on the features: The first one is encoding the date column as a number, that is "days since Jan 4, 2015" (the first date in our dataset), with a test score of 0.70; the second method is encoding the datetime as the month and convert it to a categorical feature, which I get a test score of 0.77; the last way is using lag-based features, which I created 5 lag features for AveragePrice, with the best test score of 0.79. For my best approach (random forest regressor model with lag-based features of AveragePrice), I focus on the lag-based feature for the AvergePrice variable, since I think it could be the only reasonable feature that the ones in the previous periods further affect the ones in the later periods. Also, here I did not try to add some interaction terms to the model since I do not think there are any significant interaction effects between each pair of features in this dataset.

In [28]:
```
...
df_train.head()
```

Out[28]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
0	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186.93
1	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036.04
2	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651.09
3	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82	10103.35
4	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18	9170.82

In [29]:
```
...
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15441 entries, 0 to 18222
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   Date                  15441 non-null  datetime64[ns]
 1   AveragePrice          15441 non-null  float64       
 2   Total Volume          15441 non-null  float64       
 3   4046                  15441 non-null  float64       
 4   4225                  15441 non-null  float64       
 5   4770                  15441 non-null  float64       
 6   Total Bags            15441 non-null  float64       
 7   Small Bags            15441 non-null  float64       
 8   Large Bags            15441 non-null  float64       
 9   XLarge Bags           15441 non-null  float64       
 10  type                  15441 non-null  object        
 11  year                  15441 non-null  int64         
 12  region                15441 non-null  object        
 13  AveragePriceNextWeek  15441 non-null  float64       
dtypes: datetime64[ns](1), float64(10), int64(1), object(2)
memory usage: 1.8+ MB
```

- We do not have missing data here.
- We have both categorical and numeric features.

Define feature types (exclude the date column first):

In [30]:
```
...
numeric_features = ['AveragePrice',
                    'Total Volume',
                    '4046',
                    '4225',
                    '4770',
                    'Total Bags',
                    'Small Bags',
                    'Large Bags',
                    'XLarge Bags',
                    'year']

categorical_features = ['type',
                        'region']

drop_features = ["Date"]

target = ['AveragePriceNextWeek']
```

Define a function for preprocessing:

In [31]:
```
...
# reference: Lectrue 20

def preprocess_features(
    train_df,
    test_df,
```

```
    numeric_features,
    categorical_features,
    drop_features,
    target
):

    all_features = set(numeric_features + categorical_features + drop_features
    if set(train_df.columns) != all_features:
        print("Missing columns", set(train_df.columns) - all_features)
        print("Extra columns", all_features - set(train_df.columns))
        raise Exception("Columns do not match")

    numeric_transformer = make_pipeline(
        SimpleImputer(strategy="median"), StandardScaler()
    )
    categorical_transformer = make_pipeline(
        OneHotEncoder(handle_unknown="ignore", sparse_output=False),
    )

    preprocessor = make_column_transformer(
        (numeric_transformer, numeric_features),
        (categorical_transformer, categorical_features),
        ("drop", drop_features),
    )
    preprocessor.fit(train_df)
    ohe_feature_names = (
        preprocessor.named_transformers_["pipeline-2"]
        .named_steps["onehotencoder"]
        .get_feature_names_out(categorical_features)
        .tolist()
    )
    new_columns = numeric_features + ohe_feature_names

    X_train_enc = pd.DataFrame(
        preprocessor.transform(train_df), index=train_df.index, columns=new_co
    )
    X_test_enc = pd.DataFrame(
        preprocessor.transform(test_df), index=test_df.index, columns=new_colu
    )

    y_train = train_df["AveragePriceNextWeek"]
    y_test = test_df["AveragePriceNextWeek"]

    return X_train_enc, y_train, X_test_enc, y_test, preprocessor
```

In [32]:

```
...
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer

X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_features(
    df_train,
    df_test,
    numeric_features,
    categorical_features,
    drop_features, target
)
```

In [33]:

```
...
X_train_enc.head()
```

Out[33]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
0	-0.432512	-0.234535	-0.229503	-0.222203	-0.214954	-0.232206	-0.229907
1	-0.383676	-0.234440	-0.230948	-0.219448	-0.214272	-0.233587	-0.231513
2	-0.554604	-0.233469	-0.231018	-0.219530	-0.214196	-0.229850	-0.226469
3	-0.823205	-0.233283	-0.230996	-0.218170	-0.213945	-0.230999	-0.228629
4	-0.994133	-0.225747	-0.230668	-0.196131	-0.213811	-0.232627	-0.229930

5 rows × 66 columns

Define a evaluation function (using the random forest regressor):

In [34]:
```
...
def score_rf(preprocessor, train_df, y_train, test_df, y_test, X_train_enc):
    rf_pipe = make_pipeline(preprocessor, RandomForestRegressor(n_estimators=10
    rf_pipe.fit(train_df, y_train)
    print("Train score: {:.2f}".format(rf_pipe.score(train_df, y_train)))
    print("Test score: {:.2f}".format(rf_pipe.score(test_df, y_test)))
```

R^2 score for the random forest model using the features other than the datetime:

In [35]:
```
...
score_rf(preprocessor, df_train, y_train, df_test, y_test, X_train_enc)
```

```
Train score: 0.98
Test score: 0.76
```

I tried three ways of encoding date column:

- Encoding time as a number - create a column of "days since Jan 4, 2015".

In [36]:
```
...
first_day = df_train["Date"].min()

train_df = df_train.assign(
    Days_since=df_train["Date"].apply(lambda x: (x - first_day).days)
)
test_df = df_test.assign(
    Days_since=df_test["Date"].apply(lambda x: (x - first_day).days)
)
```

In [37]:
```
...
X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_features(
    train_df,
    test_df,
    numeric_features + ["Days_since"],
    categorical_features,
    drop_features,
```

```
    target
)
```

In [38]:
```
...
score_rf(preprocessor, train_df, y_train, test_df, y_test, X_train_enc)
```

```
Train score: 0.98
Test score: 0.70
```

- One-hot encoding of the month

In [39]:
```
train_df = df_hastarget[df_hastarget["Date"] <= split_date]
test_df  = df_hastarget[df_hastarget["Date"] >  split_date]
```

In [40]:
```
...
train_df = train_df.assign(
    Month=train_df["Date"].apply(lambda x: x.month_name())
)  # x.month_name() to get the actual string
test_df = test_df.assign(Month=test_df["Date"].apply(lambda x: x.month_name()))
```

In [41]:
```
...
train_df[["Date", "Month"]].sort_values(by="Month")
```

Out[41]:

	Date	Month
1304	2017-04-30	April
9869	2016-04-17	April
9025	2016-04-24	April
3332	2017-04-30	April
3331	2017-04-23	April
...	...	...
11241	2016-09-04	September
11242	2016-09-11	September
11243	2016-09-18	September
11293	2017-09-03	September
18222	2017-09-24	September

15441 rows × 2 columns

In [42]:
```
...
X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_features(
    train_df, test_df,
    numeric_features,
    categorical_features + ["Month"],
    drop_features,
    target
)
```

In [43]:
```
...
score_rf(preprocessor, train_df, y_train, test_df, y_test, X_train_enc)
```

```
Train score: 0.98
Test score: 0.77
```

One-hot encoding seaons

In [44]:
```
def get_season(month):
    # remember this is Australia
    WINTER_MONTHS = ["June", "July", "August"]
    AUTUMN_MONTHS = ["March", "April", "May"]
    SUMMER_MONTHS = ["December", "January", "February"]
    SPRING_MONTHS = ["September", "October", "November"]
    if month in WINTER_MONTHS:
        return "Winter"
    elif month in AUTUMN_MONTHS:
        return "Autumn"
    elif month in SUMMER_MONTHS:
        return "Summer"
    else:
        return "Fall"
```

In [45]:
```
train_df = train_df.assign(Season=train_df["Month"].apply(get_season))
test_df = test_df.assign(Season=test_df["Month"].apply(get_season))
```

In [46]:
```
...
X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_features(
    train_df,
    test_df,
    numeric_features,
    categorical_features + ["Season"],
    drop_features + ["Month"],
    target
)
```

In [47]:
```
...
coeff_df = score_rf(
    preprocessor, train_df, y_train, test_df, y_test, X_train_enc
)
```

```
Train score: 0.98
Test score: 0.76
```

- creating lag-based features

In [48]:
```
...
def create_lag_feature(df, orig_feature, lag):
    """Creates a new df with a new feature that's a lagged version of the origi
    # note: pandas .shift() kind of does this for you already, but oh well I al

    new_df = df.copy()
    new_feature_name = "%s_lag%d" % (orig_feature, lag)
    new_df[new_feature_name] = np.nan
    for (region, type_), group_df in new_df.groupby(["region", "type"]):
        # Assign lagged values to the new feature column
        new_df.loc[group_df.index[lag:], new_feature_name] = group_df.iloc[:-la

    return new_df
```

In [49]:
```
...
df_hastarget_modified = create_lag_feature(df_hastarget, "AveragePrice", 1)
df_hastarget_modified = create_lag_feature(df_hastarget_modified, "AveragePrice
df_hastarget_modified = create_lag_feature(df_hastarget_modified, "AveragePrice
df_hastarget_modified = create_lag_feature(df_hastarget_modified, "AveragePrice
df_hastarget_modified = create_lag_feature(df_hastarget_modified, "AveragePrice
# df_hastarget_modified = create_lag_feature(df_hastarget_modified, "Total Volu

train_df = df_hastarget_modified[df_hastarget_modified["Date"] <= split_date]
test_df  = df_hastarget_modified[df_hastarget_modified["Date"] >  split_date]
```

In [50]:
```
...
X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_features(
    train_df,
    test_df,
    numeric_features + ["AveragePrice_lag1", "AveragePrice_lag2", "AveragePrice
    categorical_features,
    drop_features,
    target
)
```

In [51]:
```
...
lr_coef = score_rf(
    preprocessor, train_df, y_train, test_df, y_test, X_train_enc
)
```

```
Train score: 0.98
Test score: 0.79
```

Here I also tried raising the R^2 score by creating the lag features for Total Volume, but that does not work very well (no significant improvement on the score) even with 3 lag features of this variable.

In [52]:
```
...
```

Out[52]:
```
Ellipsis
```

Exercise 2: Short answer questions

2.1 Time series

rubric={points:6}

The following questions pertain to Lecture 20 on time series data:

1. Sometimes a time series has missing time points or, worse, time points that are unequally spaced in general. Give an example of a real world situation where the time series data would have unequally spaced time points.
2. In class we discussed two approaches to using temporal information: encoding the date as one or more features, and creating lagged versions of features. Which of these (one/other/both/neither) two approaches would struggle with unequally spaced time points? Briefly justify your answer.
3. When studying time series modeling, we explored several ways to encode date information as a feature for the citibike dataset. When we used time of day as a numeric feature, the Ridge model was not able to capture the periodic pattern. Why? How did we tackle this problem? Briefly explain.

Solution_2.1

Points: 6

1. one example could be medical visit records: patients might visit doctors irregularly due to the severity of their body condition or time availability.
2. both of the two approaches could struggle with this problem, more for creating lagged versions of features since encoding the date may omit small gaps in time series data by setting a larger time interval standard, even if significant gaps could not be covered up; while a lag of '1' tend to correspond to very different temporal gaps across the series.
3. that is because we encoded time of day as a numerical variable, so a linear function can only learn a linear function of the time of day, placing 23:00 and 0:00 far apart. We tackled this by encoding this feature as a categorical variable, so that the model can learn each time interval independently rather than assuming the existence of a linear relationship.

2.2 Computer vision

rubric={points:6}

The following questions pertain to Lecture 19 on multiclass classification and introduction to computer vision.

1. How many parameters (coefficients and intercepts) will `sklearn`'s `LogisticRegression()` model learn for a four-class classification problem, assuming that you have 10 features? Briefly explain your answer.
2. In Lecture 19, we briefly discussed how neural networks are sort of like `sklearn`'s pipelines, in the sense that they involve multiple sequential

transformations of the data, finally resulting in the prediction. Why was this property useful when it came to transfer learning?

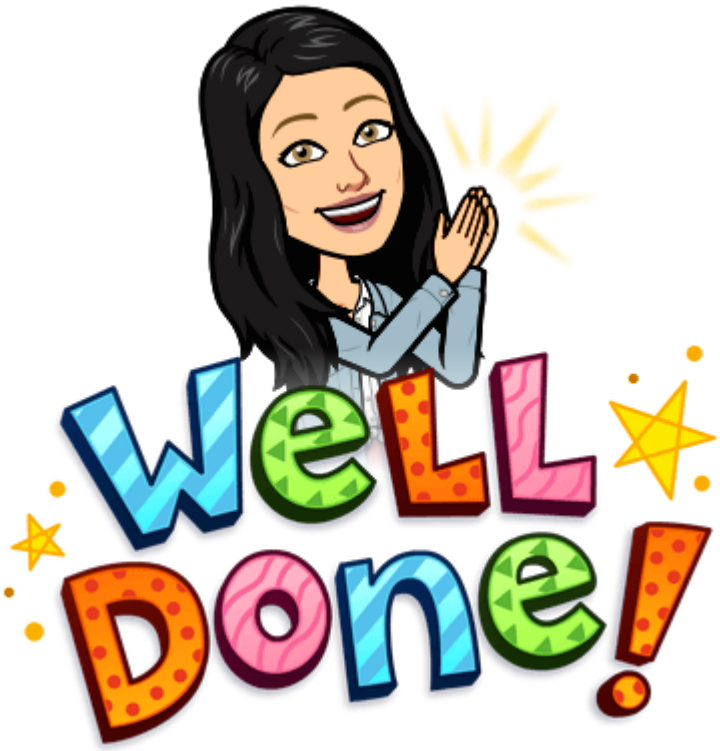
3. Imagine that you have a small dataset with ~1000 images containing pictures and names of 50 different Computer Science faculty members from UBC. Your goal is to develop a reasonably accurate multi-class classification model for this task. Describe which model/technique you would use and briefly justify your choice in one to three sentences.

Solution_2.2

Points: 6

1. each class have its own set of parameters: for 10 features, each class requires 10 coefficients and one intercept, so 4 * (10 + 1) == 44 parameters the LogisticRegression() model will learn.
2. when we use transfer learning, we download a pre-trained model, learn patterns of the data step by step, and fine tune it for our task, which allows us not to save computation and reduce the need for a large dataset.
3. I would use a pre-trained convolutional neural network (CNN), such as the pre-trianed vgg16 model. I would replace the final classification layer of the CNN with a layer suited for 50 classes and fine-tune the network on our dataset. Even if our dataset is not from some Larget datasets like ImageNet, the pre-trianed CNNs have already learnt some general image features from those large dataset, which would be helpful for our smaller dataset.

Before submitting your assignment, please make sure you have followed all the instructions in the Submission instructions section at the top.


WELL
Done!