Report of Capstone: Nice Price or No Dice

Cong, Yuxi

Apr, 2023

Statement

Shoppers often look to the holiday season for the best prices for consumer goods, particularly electronics. Shoppers are usually willing to spend money on products that they want to have and also enjoy the discount that comes with it.

I'd love to build a tool to let people use it to find out when is the best buying time. I'll use machine learning as my tool to build that.For single customers, they can use this as a tool before shopping and save money and the time to find a better deal. In the future, I want to take this as an extension on browsers, so shoppers can use this tool before they go online shopping and save them money instantly.

Background

It is a great problem to apply data science techniques. During the process of solving and analyzing the dataset that I have, machine learning of data science is the tool to make this process more accurate. Only by using machine learning, it allows me to process large datasets on my laptop. Meanwhile, I can try different models on the same datasets to find out the best one. Without data science techniques, doing all of these processes on hand will take me several weeks instead of days.

I did research online and tried to find a tool that has the same functions. Unfortunately, they are all price tracking websites. I can find out one product's historic price but without trends over the years. This obstacle inspired me to find out a way of determining if a given item is likely to be discounted. I set out to build a model that could predict if there will be a deal on holiday.

Details on dataset

The original data is the price of the product on Amazon.us. However, it is difficult to pull out data from Amazon directly. The reason is that I need data from some categories, but Amazon won’t allow for gathering all a product’s information by category or prices over time.

Here are two websites that can help me in finding these key data. They are CamelCamelCamel and Keepa. CamelCamelCamel shows a product's historical price by searching the name or ASIN(Amazon Standard Identification Number) is one of the unique product identifiers used for managing the Amazon product catalog. Keepa provides ASIN values for any category of product.

To obtain usable data, I first used Keepa to pull off a list of products with their ASINs, then extracted historical price data from CamelCamelCamel. Lastly, I used Python to run some code to help me gather them into one dataset.

First, to get the right format for columns and data, I used Python to define a custom function that changed the name, and the datatype for all the price columns.

Next, I removed duplicates of columns. To make further analysis step easier, I binned all price data by week using the lowest value for each bin as the output. The following code catties out the cleaning/transformation described above.

Cleaning and preprocessing

Now I have a folder that contains 1315 csv files. Each of them represents the historical price of one product. The goal for cleaning and preprocessing is to get them all in one dataset. Before doing the next step, EDA is requested for the dataset, which means to make the dataset in a format that I can do analysis on.

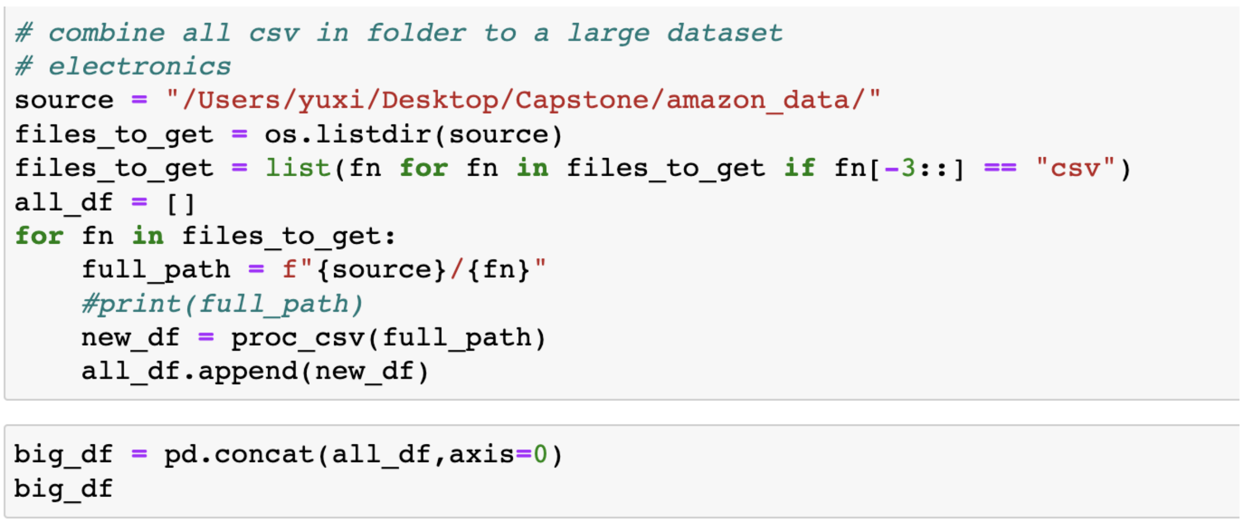
First, to use Python to define a custom function. Inside of this function, change the datatype for the columns that contain date and rename some columns.

Then remove duplicates of columns. To make further analysis step easier, I convert the actual date to week of the year, using the lowest price to represent the price of that week.

```python
def proc_csv(pathname):

    filename = os.path.basename(pathname)
    asin = filename.split(".")[0]
    df = pd.read_csv(pathname)

    # get columns
    df["asin"] = asin # get asin
    df["0"] = pd.to_datetime(df["0"]) #convert to datetime
    df["week"] = df["0"].dt.isocalendar().week # get week number
    df["year"] = df["0"].dt.isocalendar().year # get year number
    df["time"] = df["year"].astype(str)+"-"+ df["week"].astype(str)
    df.drop(["0"],axis= 1)


    # combine and sort
    df.rename({"1": "price"},axis=1, inplace = True)
    test = df.groupby(["time","asin"])["price"].min()
    test = test.reset_index()


    return test
```

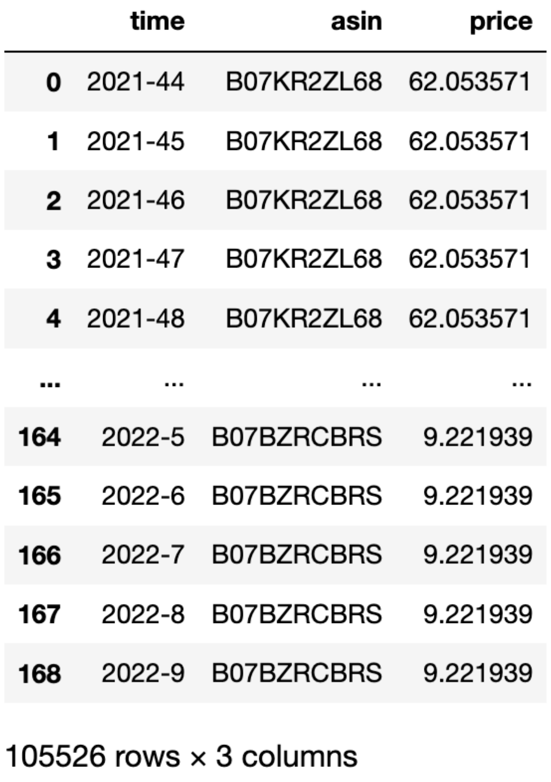
```python
result = proc_csv('~/Desktop/Capstone/amazon_data/B0B2DRWLMB.png.csv')
```

Second, I built another Python pipeline to combine all csv files from folder to one dataset.



```python
# combine all csv in folder to a large dataset
# electronics
source = "/Users/yuxi/Desktop/Capstone/amazon_data/"
files_to_get = os.listdir(source)
files_to_get = list(fn for fn in files_to_get if fn[-3::] == "csv")
all_df = []
for fn in files_to_get:
    full_path = f"{source}/{fn}"
    #print(full_path)
    new_df = proc_csv(full_path)
    all_df.append(new_df)
```

```python
big_df = pd.concat(all_df,axis=0)
big_df
```

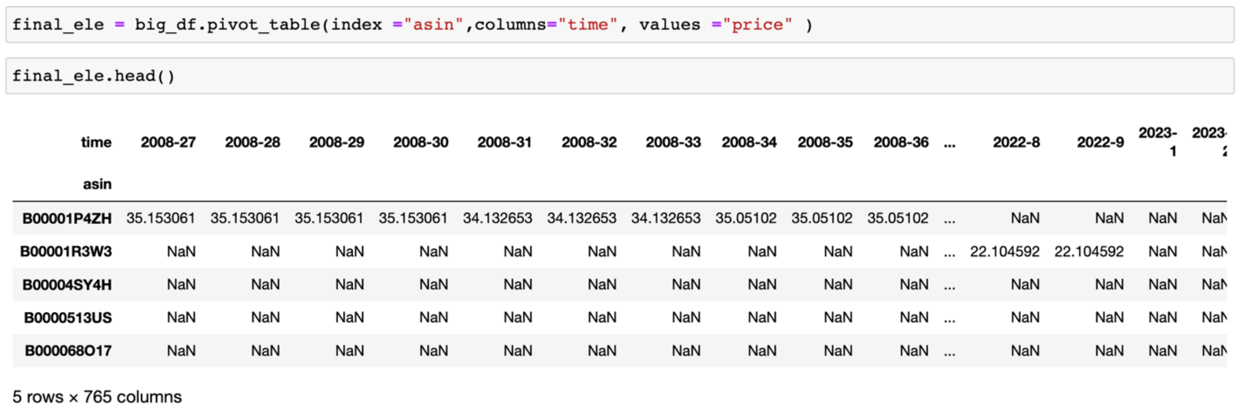


	time	asin	price
0	2021-44	B07KR2ZL68	62.053571
1	2021-45	B07KR2ZL68	62.053571
2	2021-46	B07KR2ZL68	62.053571
3	2021-47	B07KR2ZL68	62.053571
4	2021-48	B07KR2ZL68	62.053571
...	...	...	...
164	2022-5	B07BZRCBRS	9.221939
165	2022-6	B07BZRCBRS	9.221939
166	2022-7	B07BZRCBRS	9.221939
167	2022-8	B07BZRCBRS	9.221939
168	2022-9	B07BZRCBRS	9.221939

105526 rows × 3 columns

Next, I will convert the dataset above, which can be considered "long" format, to a "wide" table with unique weeks as columns and unique ASIN values as rows. This transformation was accomplished using the pivot_table function. Using the

Python function can do the pivot dataset for me. After this, I get a dataset containing 691 rows as unique ASIN's and 765 columns as unique time points.

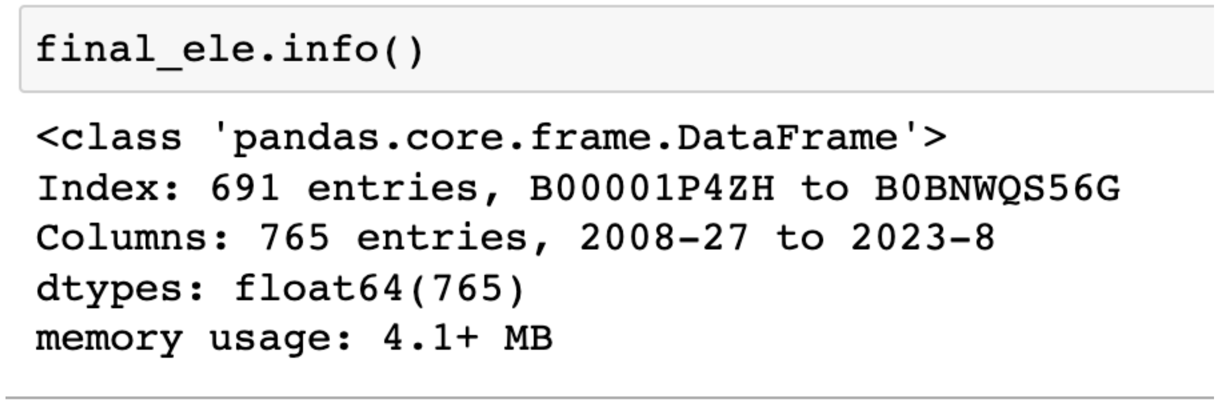


```
final_ele = big_df.pivot_table(index ="asin",columns="time", values ="price" )
```

```
final_ele.head()
```

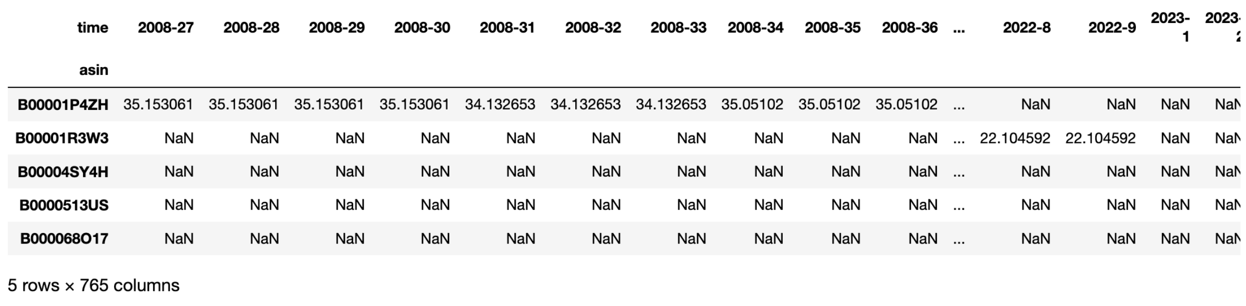
time	2008-27	2008-28	2008-29	2008-30	2008-31	2008-32	2008-33	2008-34	2008-35	2008-36	...	2022-8	2022-9	2023-1	2023-
asin															
B00001P4ZH	35.153061	35.153061	35.153061	35.153061	34.132653	34.132653	34.132653	35.05102	35.05102	35.05102	...	NaN	NaN	NaN	Na
B00001R3W3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	22.104592	22.104592	NaN	Na
B00004SY4H	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
B0000513US	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
B000068O17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na

5 rows × 765 columns



```
final_ele.info()
```

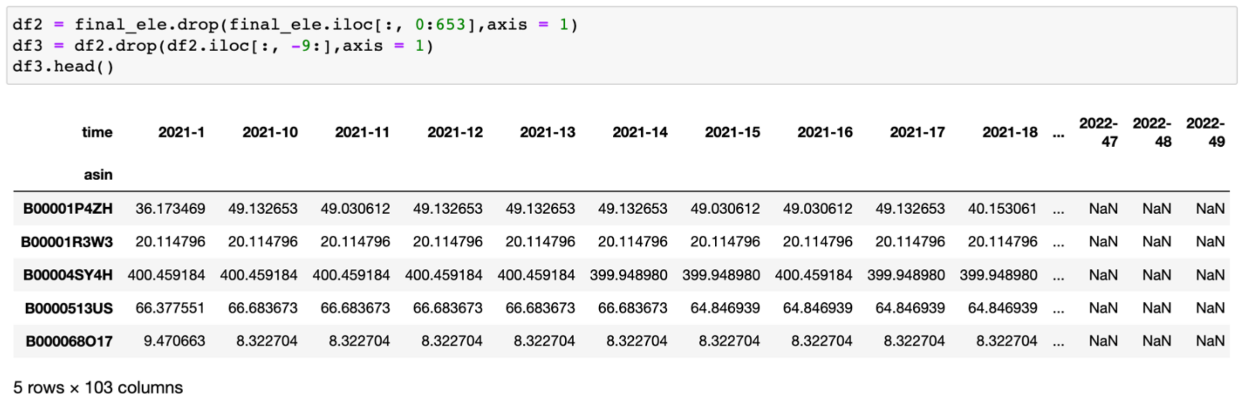
```
<class 'pandas.core.frame.DataFrame'>
Index: 691 entries, B00001P4ZH to B0BNWQS56G
Columns: 765 entries, 2008-27 to 2023-8
dtypes: float64(765)
memory usage: 4.1+ MB
```



time	2008-27	2008-28	2008-29	2008-30	2008-31	2008-32	2008-33	2008-34	2008-35	2008-36	...	2022-8	2022-9	2023-1	2023-
asin															
B00001P4ZH	35.153061	35.153061	35.153061	35.153061	34.132653	34.132653	34.132653	35.05102	35.05102	35.05102	...	NaN	NaN	NaN	Na
B00001R3W3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	22.104592	22.104592	NaN	Na
B00004SY4H	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
B0000513US	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
B000068O17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na

5 rows × 765 columns

The date from the dataset is 2008, I don't need such long period data to run models on. The next step is to remove these columns, only keep the columns that have 2021 and 2022 prices.

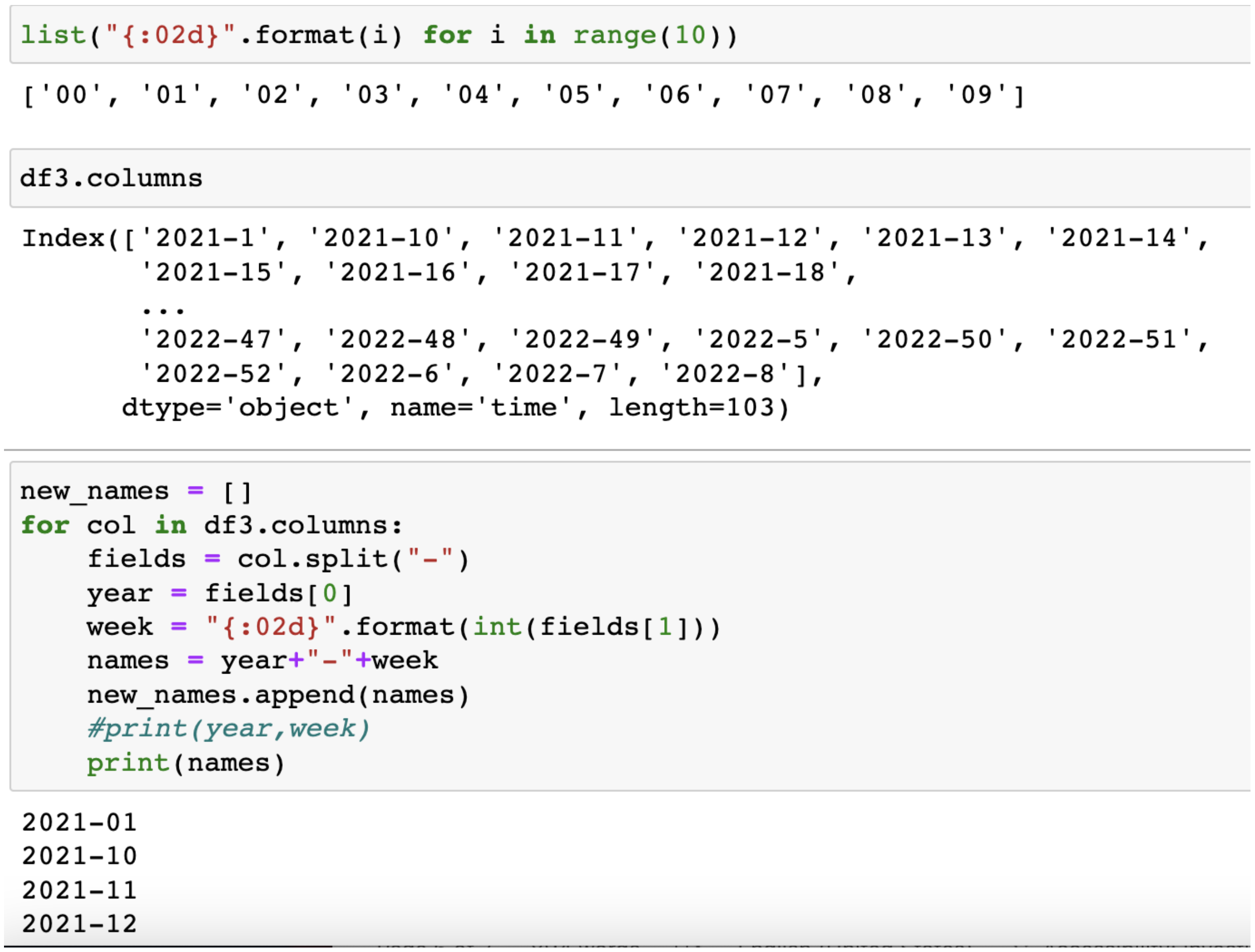


```
df2 = final_ele.drop(final_ele.iloc[:, 0:653],axis = 1)
df3 = df2.drop(df2.iloc[:, -9:],axis = 1)
df3.head()
```

time	2021-1	2021-10	2021-11	2021-12	2021-13	2021-14	2021-15	2021-16	2021-17	2021-18	...	2022-47	2022-48	2022-49
asin														
B00001P4ZH	36.173469	49.132653	49.030612	49.132653	49.132653	49.132653	49.030612	49.030612	49.132653	40.153061	...	NaN	NaN	NaN
B00001R3W3	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	...	NaN	NaN	NaN
B00004SY4H	400.459184	400.459184	400.459184	400.459184	400.459184	399.948980	399.948980	400.459184	399.948980	399.948980	...	NaN	NaN	NaN
B0000513US	66.377551	66.683673	66.683673	66.683673	66.683673	66.683673	64.846939	64.846939	64.846939	64.846939	...	NaN	NaN	NaN
B000068O17	9.470663	8.322704	8.322704	8.322704	8.322704	8.322704	8.322704	8.322704	8.322704	8.322704	...	NaN	NaN	NaN

5 rows × 103 columns

Finally, it is the dataset that I want, but the order in columns is counterintuitive. For example, the column label 2021-1 corresponds to the 1st week of 2021, but is followed by the column labeled 2021-10, the 10th week of 2021. I will use the following Python function to rename and reorder columns to be more sensible.



```
list("{:02d}".format(i) for i in range(10))
```

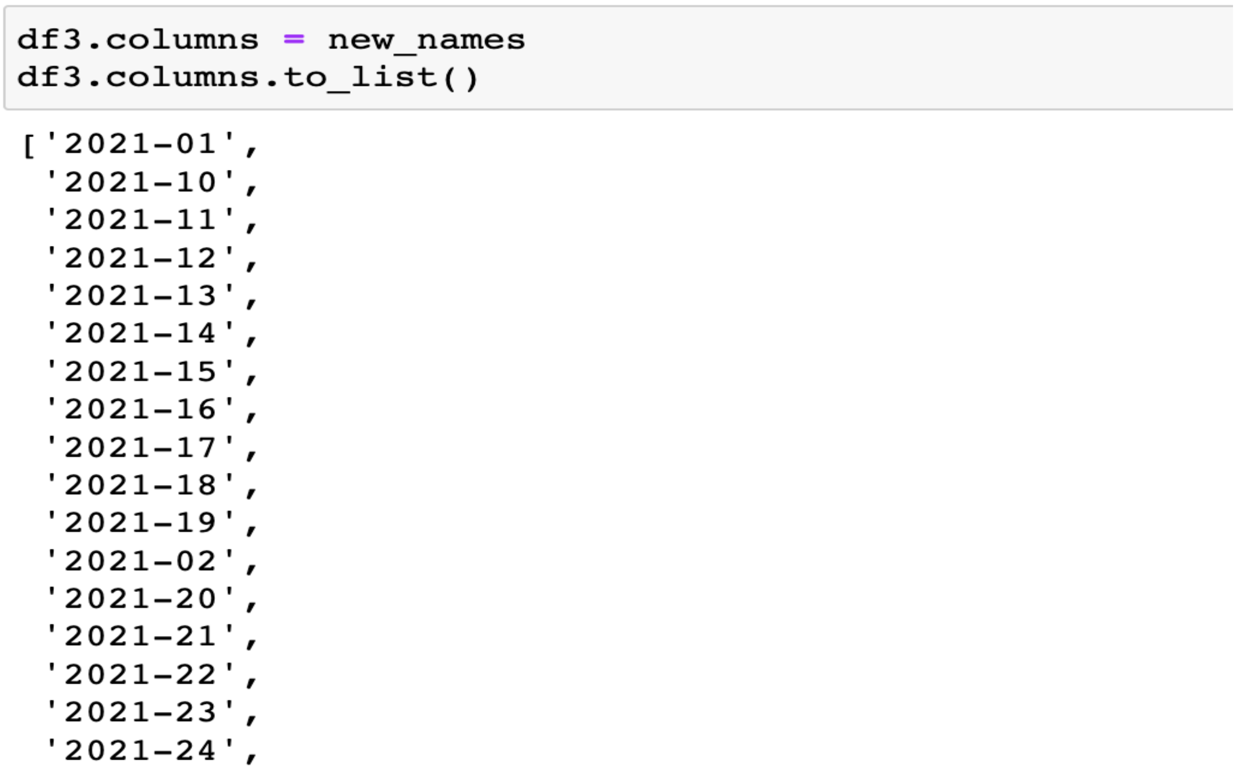
```
['00', '01', '02', '03', '04', '05', '06', '07', '08', '09']
```

```
df3.columns
```

```
Index(['2021-1', '2021-10', '2021-11', '2021-12', '2021-13', '2021-14',
       '2021-15', '2021-16', '2021-17', '2021-18',
       ...
       '2022-47', '2022-48', '2022-49', '2022-5', '2022-50', '2022-51',
       '2022-52', '2022-6', '2022-7', '2022-8'],
      dtype='object', name='time', length=103)
```

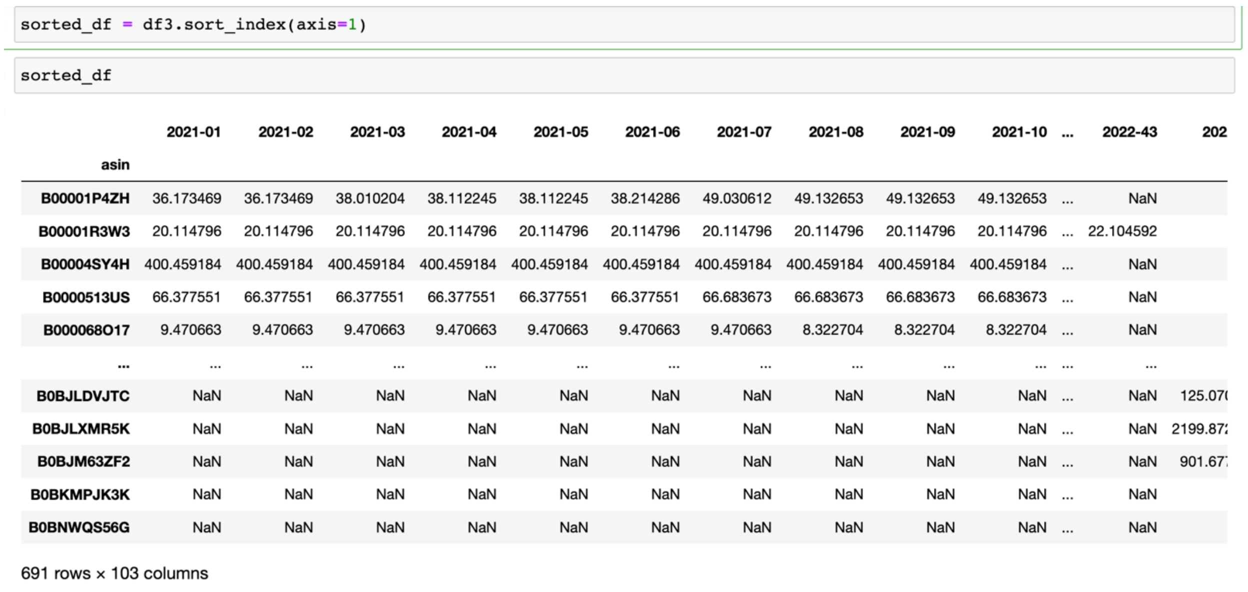
```
new_names = []
for col in df3.columns:
    fields = col.split("-")
    year = fields[0]
    week = "{:02d}".format(int(fields[1]))
    names = year+"-"+week
    new_names.append(names)
    #print(year,week)
    print(names)
```

```
2021-01
2021-10
2021-11
2021-12
```



```
df3.columns = new_names
df3.columns.to_list()
```

```
['2021-01',
 '2021-10',
 '2021-11',
 '2021-12',
 '2021-13',
 '2021-14',
 '2021-15',
 '2021-16',
 '2021-17',
 '2021-18',
 '2021-19',
 '2021-02',
 '2021-20',
 '2021-21',
 '2021-22',
 '2021-23',
 '2021-24',
```



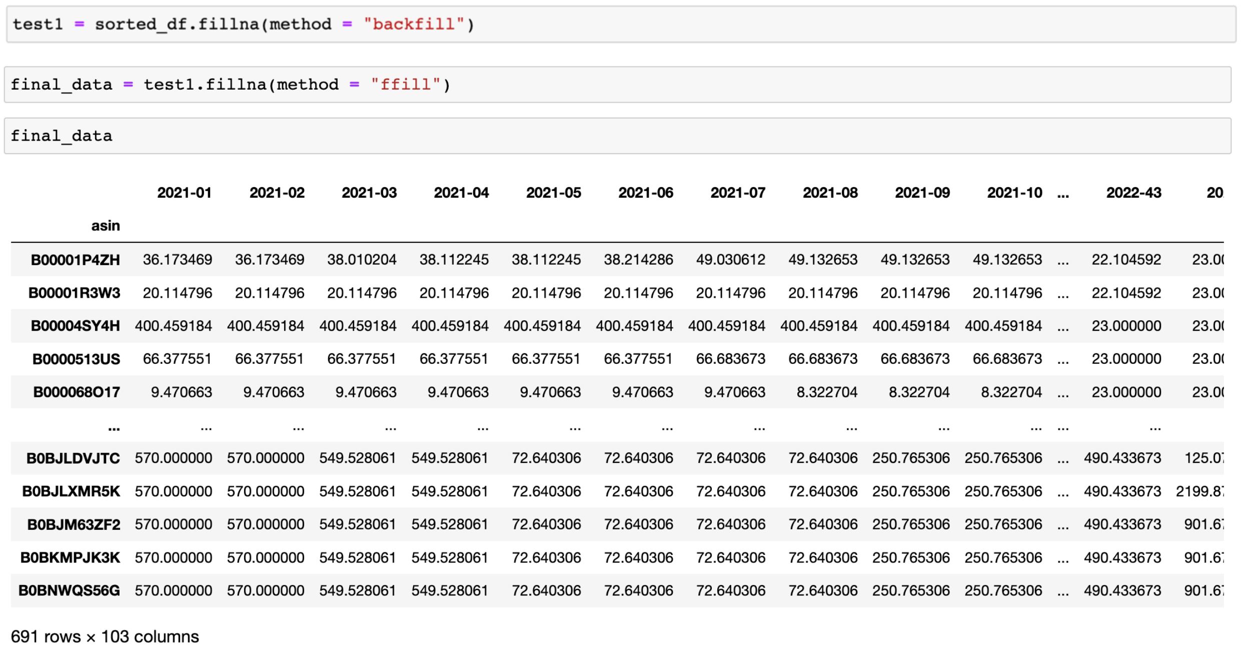
```
sorted_df = df3.sort_index(axis=1)
```

```
sorted_df
```

	2021-01	2021-02	2021-03	2021-04	2021-05	2021-06	2021-07	2021-08	2021-09	2021-10	...	2022-43	202
asin													
B00001P4ZH	36.173469	36.173469	38.010204	38.112245	38.112245	38.214286	49.030612	49.132653	49.132653	49.132653	...	NaN	
B00001R3W3	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	...	22.104592	
B00004SY4H	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	...	NaN	
B0000513US	66.377551	66.377551	66.377551	66.377551	66.377551	66.377551	66.683673	66.683673	66.683673	66.683673	...	NaN	
B000068O17	9.470663	9.470663	9.470663	9.470663	9.470663	9.470663	9.470663	8.322704	8.322704	8.322704	...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
B0BJLDVJTC	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	125.07
B0BJLXMR5K	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	2199.87
B0BJM63ZF2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	901.67
B0BKMPJK3K	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
B0BNWQS56G	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	

691 rows × 103 columns

Before running any machine learning models, the null values need to be removed and replaced with actual values. The reason is that there are prices of certain products, if they don't have them filled, models can not be running on. I used a combination of forward and back filling to remove NaN values.



```
test1 = sorted_df.fillna(method = "backfill")
```

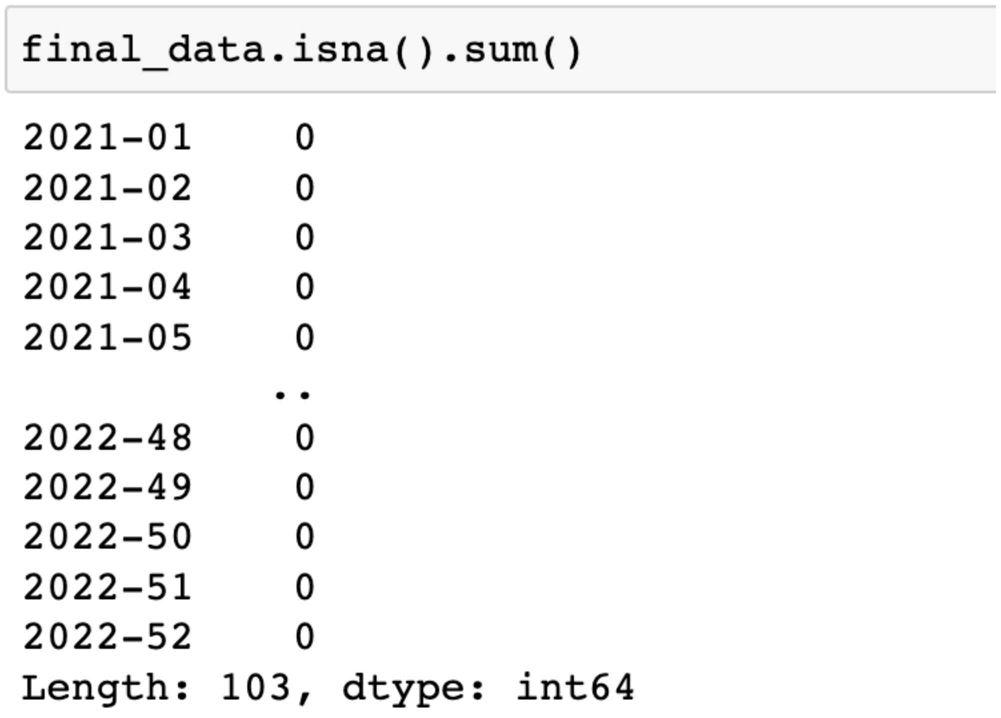
```
final_data = test1.fillna(method = "ffill")
```

```
final_data
```

	2021-01	2021-02	2021-03	2021-04	2021-05	2021-06	2021-07	2021-08	2021-09	2021-10	...	2022-43	20
asin													
B00001P4ZH	36.173469	36.173469	38.010204	38.112245	38.112245	38.214286	49.030612	49.132653	49.132653	49.132653	...	22.104592	23.0
B00001R3W3	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	20.114796	...	22.104592	23.0
B00004SY4H	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	400.459184	...	23.000000	23.0
B0000513US	66.377551	66.377551	66.377551	66.377551	66.377551	66.377551	66.683673	66.683673	66.683673	66.683673	...	23.000000	23.0
B000068O17	9.470663	9.470663	9.470663	9.470663	9.470663	9.470663	9.470663	8.322704	8.322704	8.322704	...	23.000000	23.0
...	...	...	...	...	...	...	...	...	...	...	...	...	
B0BJLDVJTC	570.000000	570.000000	549.528061	549.528061	72.640306	72.640306	72.640306	72.640306	250.765306	250.765306	...	490.433673	125.0
B0BJLXMR5K	570.000000	570.000000	549.528061	549.528061	72.640306	72.640306	72.640306	72.640306	250.765306	250.765306	...	490.433673	2199.8
B0BJM63ZF2	570.000000	570.000000	549.528061	549.528061	72.640306	72.640306	72.640306	72.640306	250.765306	250.765306	...	490.433673	901.6
B0BKMPJK3K	570.000000	570.000000	549.528061	549.528061	72.640306	72.640306	72.640306	72.640306	250.765306	250.765306	...	490.433673	901.6
B0BNWQS56G	570.000000	570.000000	549.528061	549.528061	72.640306	72.640306	72.640306	72.640306	250.765306	250.765306	...	490.433673	901.6

691 rows × 103 columns

Then, check if there are still some missing values in this dataset.



```
final_data.isna().sum()
```

```
2021-01     0
2021-02     0
2021-03     0
2021-04     0
2021-05     0
           ..
2022-48     0
2022-49     0
2022-50     0
2022-51     0
2022-52     0
Length: 103, dtype: int64
```

Perfect! Now the dataset is ready for the next step.

Insights, modeling, and results

The models that I choose to run are Logistic regression, random forest, decision tree and KNN (K-Nearest Neighbors Algorithm).

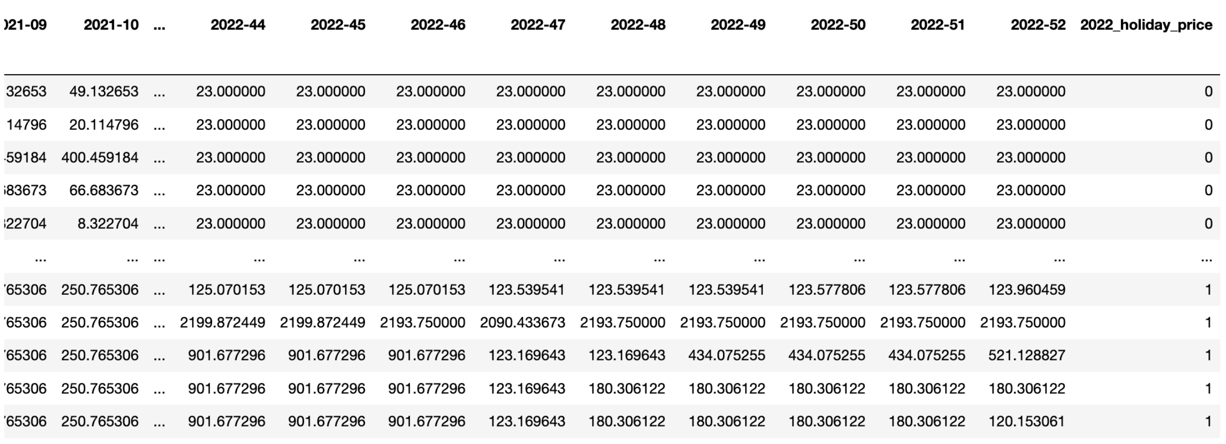
1. Logistic Regression

To create a new column as a binary column. By comparing the price of Black Friday in the year of 2021 and 2022 and the week before Black Friday, if the price on Black Friday is not higher than the week before, return 1 in the new column, otherwise return 0.

Take this new column as target of logistic regression, and the other week's prices as features to see if they have a significant relationship.

```
# 2022 week 47
final_data["2022_holiday_price"] = final_data.apply(lambda x: 1 if x["2022-47"] < x["2022-46"] else 0, axis=1)
```

```
final_data
```



21-09	2021-10	...	2022-44	2022-45	2022-46	2022-47	2022-48	2022-49	2022-50	2022-51	2022-52	2022_holiday_price
32653	49.132653	...	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	0
14796	20.114796	...	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	0
59184	400.459184	...	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	0
83673	66.683673	...	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	0
22704	8.322704	...	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	23.000000	0
...	...	...	...	...	...	...	...	...	...	...	...	...
65306	250.765306	...	125.070153	125.070153	125.070153	123.539541	123.539541	123.539541	123.577806	123.577806	123.960459	1
65306	250.765306	...	2199.872449	2199.872449	2193.750000	2090.433673	2193.750000	2193.750000	2193.750000	2193.750000	2193.750000	1
65306	250.765306	...	901.677296	901.677296	901.677296	123.169643	123.169643	434.075255	434.075255	434.075255	521.128827	1
65306	250.765306	...	901.677296	901.677296	901.677296	123.169643	180.306122	180.306122	180.306122	180.306122	180.306122	1
65306	250.765306	...	901.677296	901.677296	901.677296	123.169643	180.306122	180.306122	180.306122	180.306122	120.153061	1

```
#2022 w47

X = final_data.drop(["2022_holiday_price","2021-47","2022-47"],axis=1)
y = final_data["2022_holiday_price"]
```

Before running the model on data, I did a train and test data split. I took 70% of the dataset as training, and the rest 30% as testing.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state =11)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s = scaler.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression

# 1. Instantiate the model
log_reg = LogisticRegression(C=1, penalty='l2', random_state=2)

# 2. Fit the model
model = log_reg.fit(X_train_s, y_train)
```

```
print(log_reg.score(X_test_s, y_test))
```

```
0.8605769230769231
```

The accuracy of logistic regression is 86%.

2. Random Forest

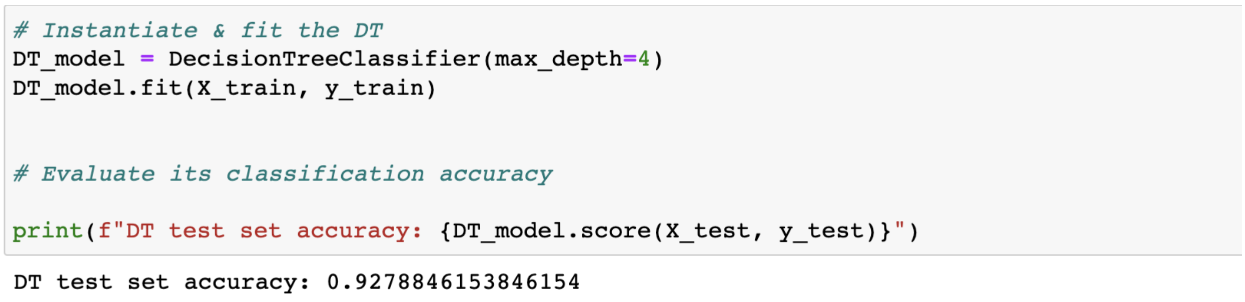
```
from sklearn.ensemble import RandomForestClassifier
```

```
rfc = RandomForestClassifier(n_estimators=35,max_depth=10,random_state=42)

# Fit the model to the training data
rfc.fit(X_train, y_train)

# Predict the class labels for the test data
y_pred_rfc = rfc.predict(X_test)

# Evaluate the model's performance
print("Accuracy:", rfc.score(X_test, y_test))
```

```
Accuracy: 0.8990384615384616
```

The accuracy of a random forest model is 89.9%. I adjusted the hyperparameter several times to find out the best result. When n_esstimators equal to35 and max_depth equals to 4, the accuracy is the highest.

3. Decision Tree

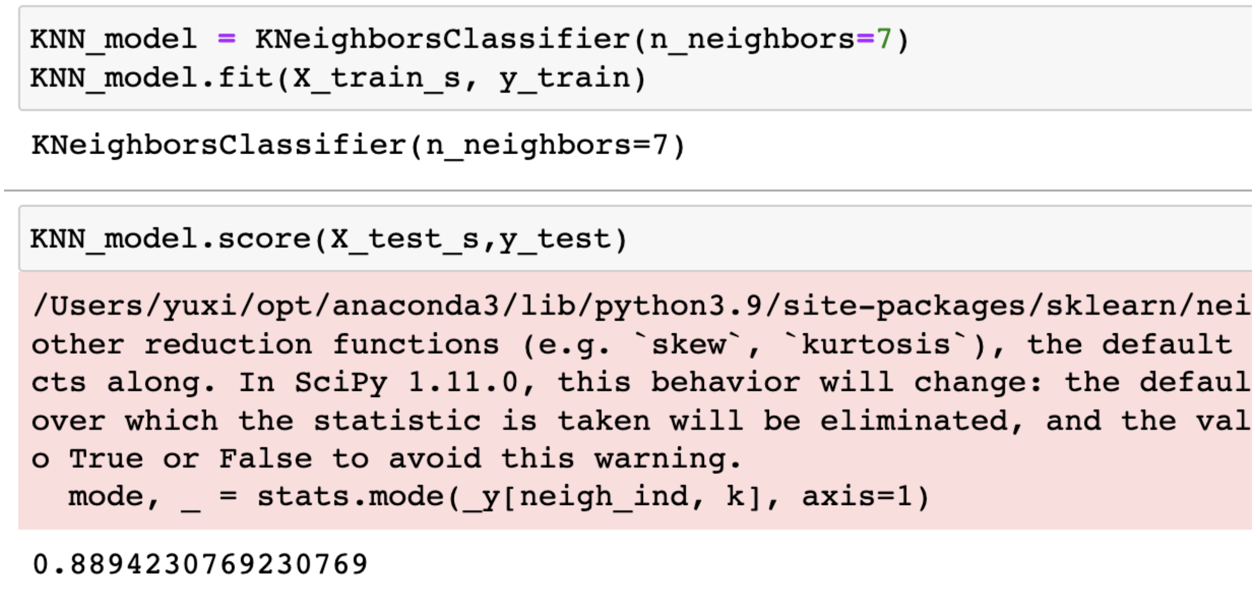


```
# Instantiate & fit the DT
DT_model = DecisionTreeClassifier(max_depth=4)
DT_model.fit(X_train, y_train)


# Evaluate its classification accuracy

print(f"DT test set accuracy: {DT_model.score(X_test, y_test)}")
```

```
DT test set accuracy: 0.9278846153846154
```

The accuracy of the decision tree is 92.7%.

4. KNN (K-Nearest Neighbors Algorithm)



```
KNN_model = KNeighborsClassifier(n_neighbors=7)
KNN_model.fit(X_train_s, y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

```
KNN_model.score(X_test_s,y_test)
```

```
/Users/yuxi/opt/anaconda3/lib/python3.9/site-packages/sklearn/nei
other reduction functions (e.g. `skew`, `kurtosis`), the default
cts along. In SciPy 1.11.0, this behavior will change: the defaul
over which the statistic is taken will be eliminated, and the val
o True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

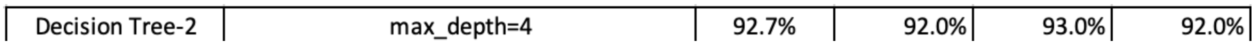
```
0.8894230769230769
```

The accuracy of KNN is 88.9%.

Findings

Decision tree model got a better accuracy at 92.7% on testing data. Precision at 92%, recall at 93% and F-1 score at 92%.

Features don't have the columns of black Friday week for two years. Maybe there is some relationship between the 2022 black Friday deal and previous year's black Friday deal. I brought 2021's black Friday week columns back and ran a decision tree again.



Decision Tree-2	max_depth=4	92.7%	92.0%	93.0%	92.0%

Accuracy and precision stayed the same as last tree, recall and F-1 score got 1% up. As a result, the previous year's deal doesn't affect this year's deal.

Also, I plotted both two trees to take a deeper look. They both have their first split on the second week after black Friday, which is week 49. It means that week is the most important feature for predicting black Friday deals.

Conclusions

Here is my conclusion about this project:

- The previous year's Black Friday deal won't be a key to find out if there is a deal this year
- The second week after Black Friday is the most important feature to predict if there is a Black Friday's deal

Future and next step

- I'm planning to make this idea as a browser extension. It will be used by people doing online shopping. By clicking on this extension's button, it will have a pop-up window telling them "Buy now, best deal!" or "Don't buy it now, wait for black Friday".
- I want to add more features for this model. Such as number in stock and launch time of product.