Price Prediction of Kalimati Vegetables (Linear Regression)

May 15, 2020

1. Data Analysis and Visualization

[1]:
```
# import require libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

[2]:
```
# reading dataset
df = pd.read_csv('Price.csv')

df.head()
```

[2]:
```
                                                cdate pricetype
0        (   )   . .                     02/25/2018         W
1                    . .                     02/25/2018         W
2                     . .                     02/25/2018         W
3                    . .                     02/25/2018         W
4                . .                     02/25/2018         W
```

[3]:
```
# renaming the column names
df = df.rename(columns={
    '     ': 'Items',
    '   ': 'Unit',
    '   ': 'MinPrice',
    '    ': 'MaxPrice',
    '  ': 'AvgPrice',
    'cdate': 'Date',
    'pricetype': 'PriceType'
})

df.head()
```

[3]:
```
                    Items    Unit MinPrice MaxPrice AvgPrice        Date  \
0        (   )   . .                               02/25/2018
1                    . .                               02/25/2018
```

```
2                       . .                            02/25/2018
3                      . .                            02/25/2018
4                 . .                            02/25/2018

  PriceType
0         W
1         W
2         W
3         W
4         W
```

[4]:
```
# changing nepali date to digit
df['MinPrice'] = df['MinPrice'].map(int)
df['MaxPrice'] = df['MaxPrice'].map(int)
df['AvgPrice'] = df['AvgPrice'].map(int)

df.head()
```

[4]:
```
                  Items     Unit  MinPrice  MaxPrice  AvgPrice        Date  \
0         (   )   . .         30         35        33  02/25/2018
1                     . .         25         30        28  02/25/2018
2                       . .         20         23        22  02/25/2018
3                      . .         18         20        19  02/25/2018
4                 . .         44         46        45  02/25/2018

  PriceType
0         W
1         W
2         W
3         W
4         W
```

[5]:
```
# checking the datatype
df.dtypes
```

[5]:
```
Items          object
Unit           object
MinPrice        int64
MaxPrice        int64
AvgPrice        int64
Date           object
PriceType      object
dtype: object
```

[6]:
```
# converting date object to datetimes
df['Date'] = pd.to_datetime(df['Date'])
```

```python
df.dtypes
```

[6]:
```
Items                object
Unit                 object
MinPrice              int64
MaxPrice              int64
AvgPrice              int64
Date         datetime64[ns]
PriceType            object
dtype: object
```

[7]:
```python
# checking the null values
df.isnull().sum()
```

[7]:
```
Items        0
Unit         0
MinPrice     0
MaxPrice     0
AvgPrice     0
Date         0
PriceType    0
dtype: int64
```

[8]:
```python
# basic info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99302 entries, 0 to 99301
Data columns (total 7 columns):
Items        99302 non-null object
Unit         99302 non-null object
MinPrice     99302 non-null int64
MaxPrice     99302 non-null int64
AvgPrice     99302 non-null int64
Date         99302 non-null datetime64[ns]
PriceType    99302 non-null object
dtypes: datetime64[ns](1), int64(3), object(3)
memory usage: 5.3+ MB
```

[9]:
```python
# describe the numeric value
df.describe()
```

[9]:

	MinPrice	MaxPrice	AvgPrice
count	99302.000000	99302.000000	99302.000000
mean	102.130984	111.899368	107.094751
std	79.376194	82.277694	80.746183
min	1.000000	10.000000	9.000000

```
25%        50.000000      60.000000      55.000000
50%        80.000000      90.000000      85.000000
75%       130.000000     140.000000     135.000000
max      1600.000000    1650.000000    1625.000000
```

[10]:
```
# seperating day, month and year
df['Day'] = df['Date'].dt.day
df['Month'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year

df.head()
```

[10]:
```
                  Items    Unit  MinPrice  MaxPrice  AvgPrice       Date  \
0        (   )   . .        30        35        33 2018-02-25
1                    . .        25        30        28 2018-02-25
2                     . .        20        23        22 2018-02-25
3                     . .        18        20        19 2018-02-25
4                . .        44        46        45 2018-02-25

  PriceType  Day  Month  Year
0         W   25      2  2018
1         W   25      2  2018
2         W   25      2  2018
3         W   25      2  2018
4         W   25      2  2018
```

[11]:
```
# checking the numbers of date
total_dates = len(pd.date_range(df['Date'].min(), df['Date'].max()))

print('Total Dates = ', total_dates)
```

```
Total Dates =  721
```

[12]:
```
# date range
min_date = df['Date'].min()
max_date = df['Date'].max()

print('Staring Date: {}'.format(min_date))
print('Final Date: {}'.format(max_date))
```

```
Staring Date: 2018-02-25 00:00:00
Final Date: 2020-02-15 00:00:00
```

[13]:
```
# calculating the number of wholesale and retail items
df.groupby('PriceType').count()
```

```
[13]:            Items   Unit  MinPrice  MaxPrice  AvgPrice   Date    Day  Month  \
PriceType
R            49646  49646     49646     49646     49646  49646  49646  49646
W            49656  49656     49656     49656     49656  49656  49656  49656

              Year
PriceType
R            49646
W            49656
```

```python
[14]: # expensive and cheap veggis items
expensive_item = df.nlargest(n=1, columns='AvgPrice')['Items']
cheap_item = df.nsmallest(n=1, columns='AvgPrice')['Items']

print('Expensive Item: ', expensive_item)
print('Cheap Item: ', cheap_item)
```

```
Expensive Item:  5431
Name: Items, dtype: object
Cheap Item:  39502      (  )
Name: Items, dtype: object
```

Expensive Veggis is   and cheap veggis is   (   ).

```python
[15]: # finding the unique veggis name
unique_veggis = df['Items'].unique()

unique_veggis
```

```
[15]: array(['      (   )', '        ', '      ', '     ',
       '          ', '  (   )', '   (   )',
       '       ', '        ', '      ', '    (   )',
       '      ', '       ', '     ', '     (   )',
       '     ', '   ', '       ', '       (   )',
       '   ', '   ', '   ', '   ', '     ', '      ',
       '      ', '       ', '       ', '       ', '   ',
       '  ', '  (  )', '     ', '    ', '      ',
       '      ', '   ', '   ', '      ', '   ',
       '     ', '   ', '   ', '   ', '   ', '  (  )',
       '  ', '   ', '   ', '  (   )', '   (   )',
       '  (   )', '   ', '     ', '  (  )', '     ',
       '   (  )', '  (   )', '   ', '   ',
       '       ', '        ', '     ',
       '      ', '        ', '         ',
       '         ', '        ', '       ', '      ',
       '  (  )', '       ', '  ( )', '   ', '     ',
       '   ', '  (   )', '     ', '   ', '   (  )',
       '     ', '   ', '   ', '   ', '   ',
```

```
'      ', '       ( )', '    (    )',
'  ( )', '  ( )', '   (    )',
'       ( )', '       ( )',
'       (  )', '     ( )', '      (  )',
'     ( )', '     ( \u200c  )', '    (    )',
'    ', '     (    )', '     (   )',
'      (  )', ' (   )', '   ( )',
' (   )', '   (   )', '  (   )', '     ( )',
'      (   )', '    (   )', '      (   )',
'   (   )', '  (  )', '      (  )',
'          ', '  (  )', '       (  )',
'  (  )', '  (  )'], dtype=object)
```

[16]:
```python
# number of unique veggis
print('Total unique veggis: {}'.format(df['Items'].nunique()))
```

```
Total unique veggis: 120
```

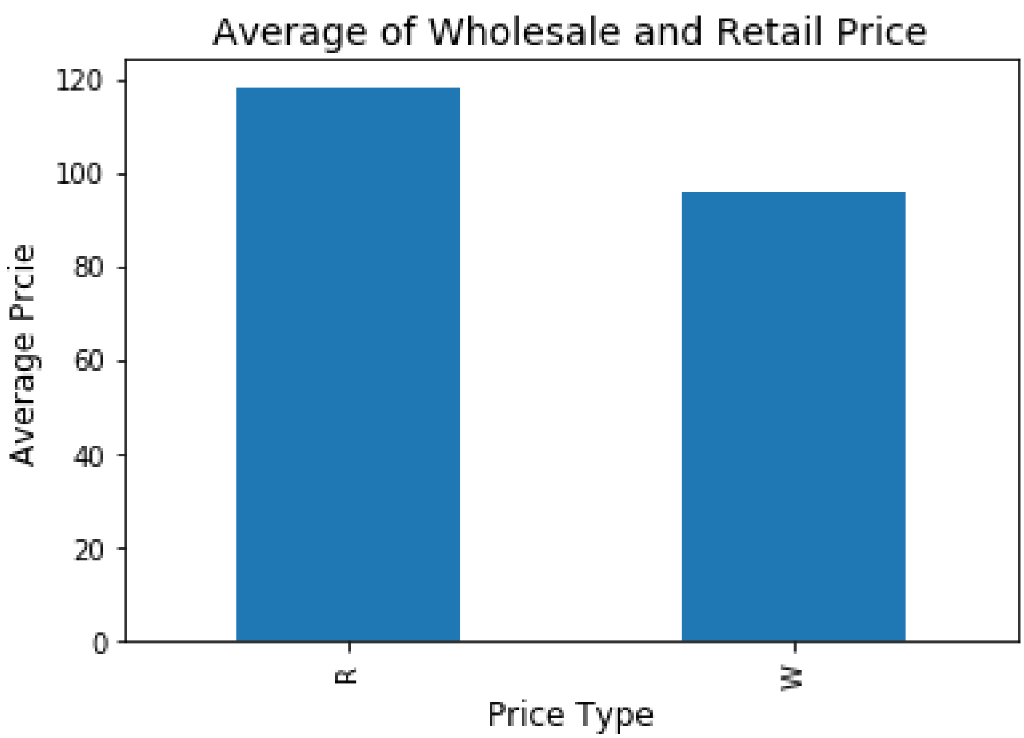
[17]:
```python
# seperating veggis according to the categories
fruits = np.array(['      (   )', '       ', '  ', '  ( )', '  ', '   ', '  ',␣
 ↪' (   )', '   (   )', '   (   )', '  ', '     ', '   (  )', '     ',␣
 ↪'    (  )', ' (   )', '  ', '   ', ' (   )', '   (  )', '  ', '   ', '  ␣
 ↪ ( )', '  ( )', '   (    )', ' (   )', '   ( )', ' (   )', '   (   )', ' ␣
 ↪  (   )', '    (   )', '      (   )', '    (   )', '      (  )', ' (  )'])
veg = np.array(['     ', '     ', '  (  )', '  (  )', '       ', '       ', ' ␣
 ↪  ', '    (  )', '      ', '       ', '     ', '     ( )', '     ', '  ', '  ␣
 ↪  ', '      (  )', '   ', '   ', '   ', '  ', '     ', '       ', '       ',␣
 ↪'      ', '       ', '  ', '  ', '  (  )', '     ', '    ', '       ', '       ',␣
 ↪'  ', '       ', '      ', '   ', '   ', '  ', '     ', '       ', '  (  )',␣
 ↪' ( )', '   ', '     ', '     ', '      ', '   ', '   ', '    ', '     (   )',␣
 ↪' (  )', '    (    )', '    ', '     (    )', '     (   )', '     (  )',␣
 ↪' (   )', '    ( )', '  (  )', '  (  )', '       (  )', '  (  )'])
non_veg = np.array(['     ', '      ( )', '      (  )', '      ( )', '  ␣
 ↪  ( \u200c  )'])
spices = np.array(['         ', '       ', '   ', '   ', '   ', '       ', '   ␣
 ↪  ', '      ', '      ', '       ', '         ', '         ', '       ', '   ␣
 ↪  ', '      ( )', '      ( )', '      (  )', '         '])
```

[18]:
```python
# checking the total size after categorizing
cat_total_items = fruits.size + veg.size + non_veg.size + spices.size

print('Total size after categorizing: {}'.format(cat_total_items))
```

```
Total size after categorizing: 120
```

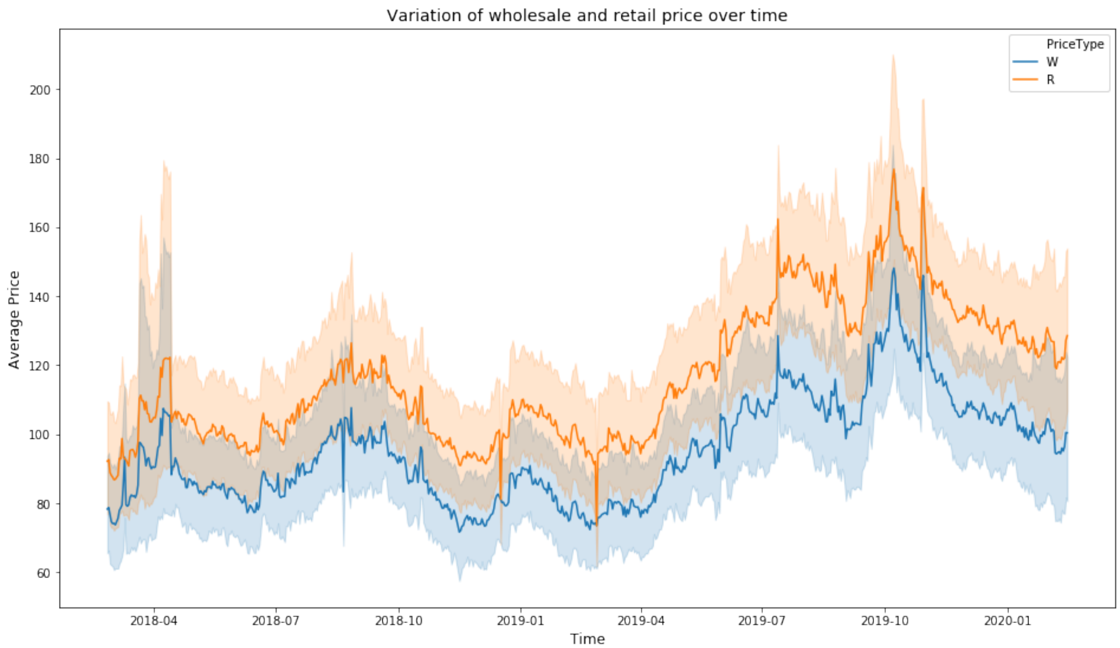
[19]:
```python
# bar plot between retail price and wholesale price
df.groupby(df['PriceType'])['AvgPrice'].mean().plot.bar()
```

```
plt.title('Average of Wholesale and Retail Price', fontsize=14)
plt.xlabel('Price Type', fontsize=12)
plt.ylabel('Average Prcie', fontsize=12)

plt.show()
```



[20]:

```
# variation of wholesale and retail rate over time
plt.figure(figsize=(16, 9))

sns.lineplot(x='Date', y='AvgPrice', data=df, hue='PriceType')

plt.title('Variation of wholesale and retail price over time', fontsize=14)
plt.xlabel('Time', fontsize = 12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```



This shows that the retail price is correlated to the wholesale price.

[21]:
```
# making a seperate dataframe for different categories
def column(dataframe, category_list):
    new_df = pd.DataFrame()

    for item in category_list:
        new_row = df[df['Items'] == item]
        new_df = new_df.append(new_row)

    return new_df



fruits_df = column(df, fruits)
veg_df = column(df, veg)
non_veg_df = column(df, non_veg)
spices_df = column(df, spices)
```

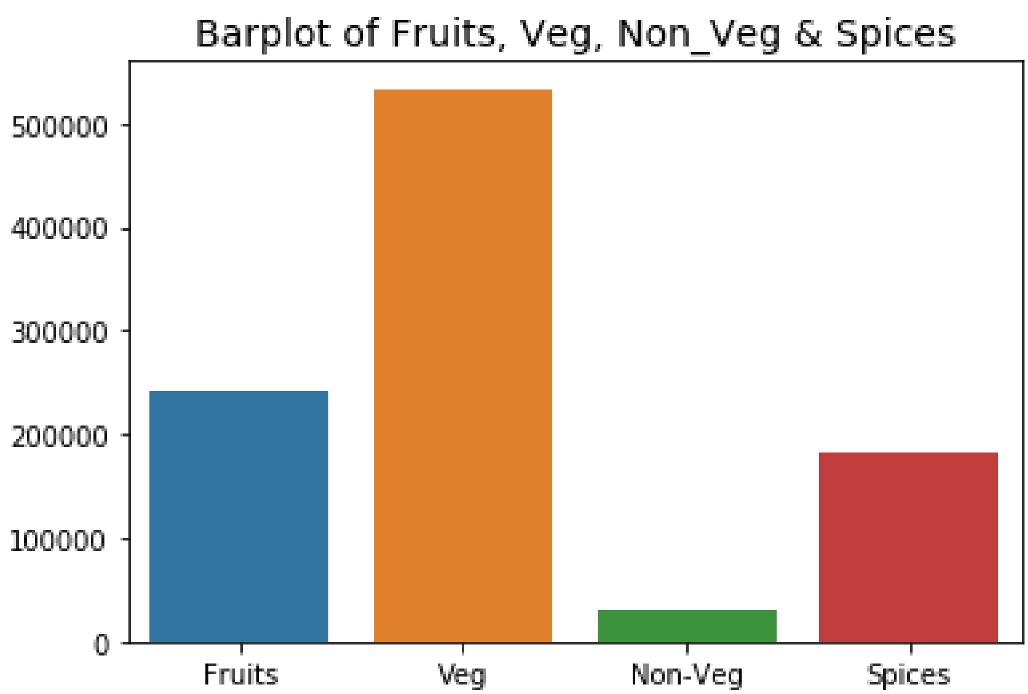
[22]:
```
# checking the size of category dataframe
print('fruits_df size: ', fruits_df.size)
print('veg_df size: ', veg_df.size)
print('non_veg_df size: ', non_veg_df.size)
print('spices_df size: ', spices_df.size)
```

```
fruits_df size:  241780
```

```
veg_df size:  533790
non_veg_df size:  29920
spices_df size:  183260
```

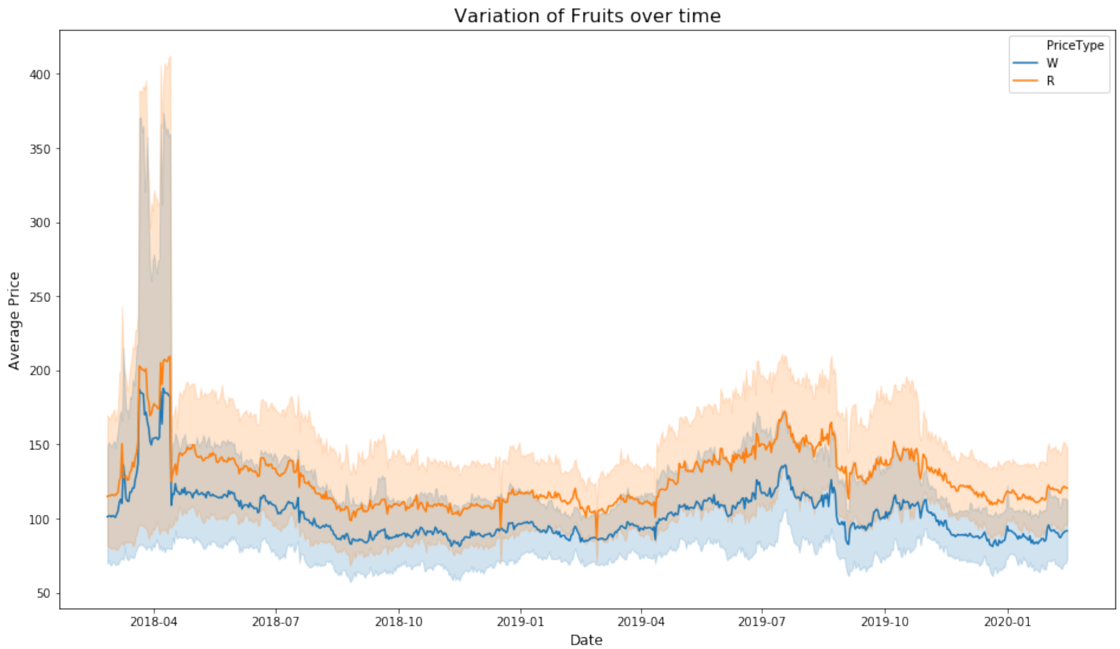
[23]:

```
sns.barplot(x=['Fruits', 'Veg', 'Non-Veg', 'Spices'],
            y=[fruits_df.size, veg_df.size, non_veg_df.size, spices_df.size])

plt.title('Barplot of Fruits, Veg, Non_Veg & Spices', fontsize=14)

plt.show()
```

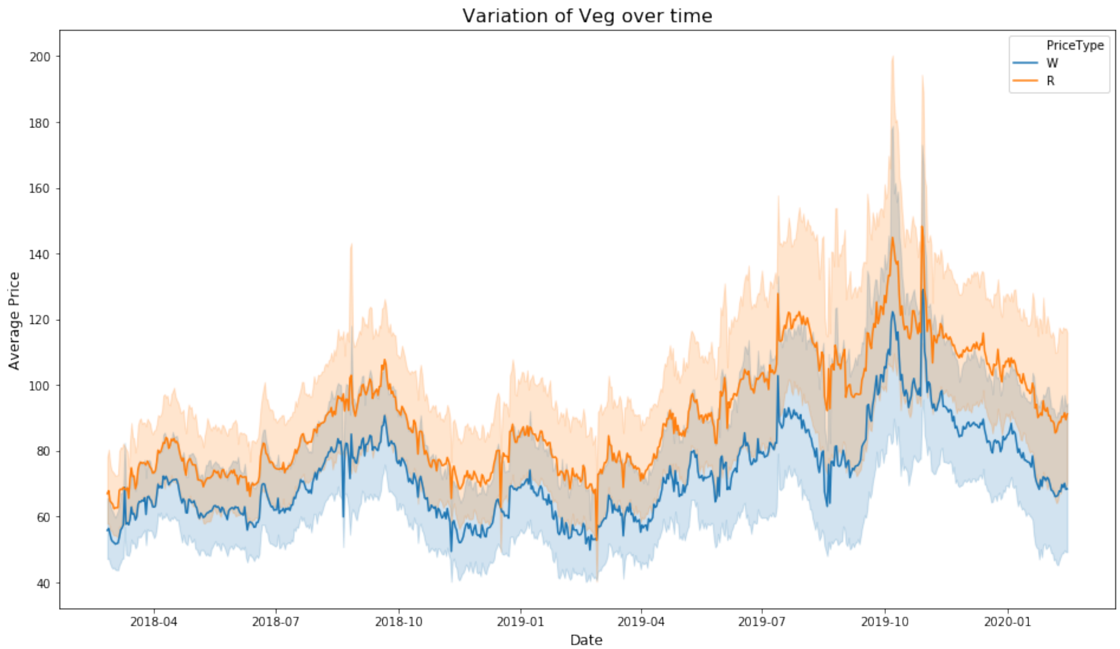


[24]:

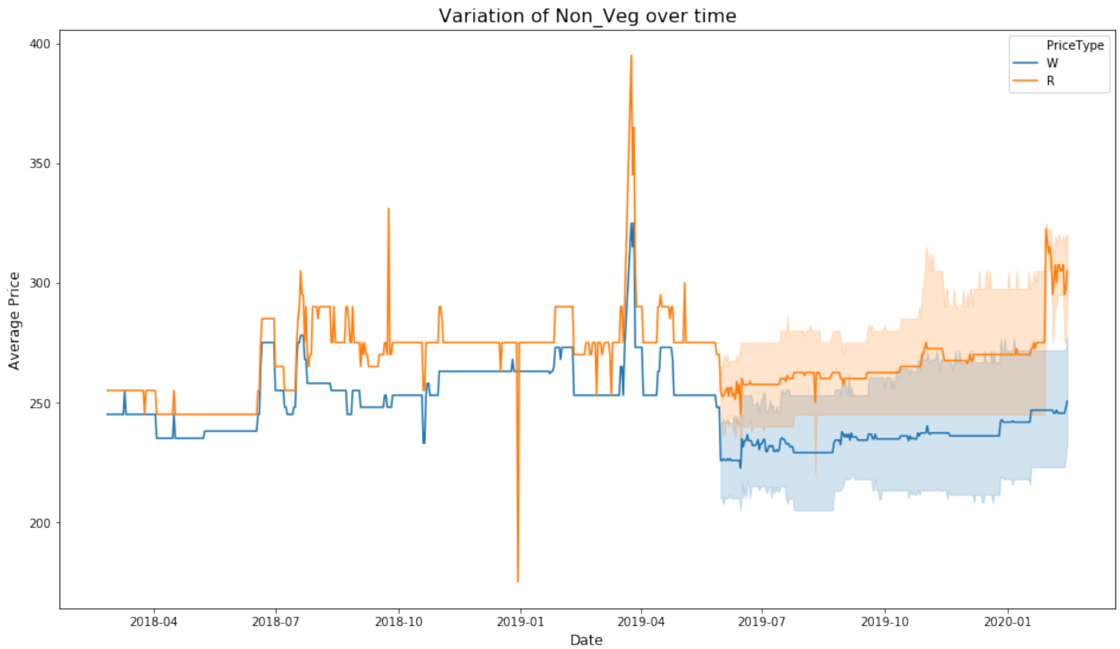
```
# variation of Fruits items over time
plt.figure(figsize=(16, 9))

sns.lineplot(x='Date', y='AvgPrice', data=fruits_df, hue='PriceType')

plt.title('Variation of Fruits over time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```



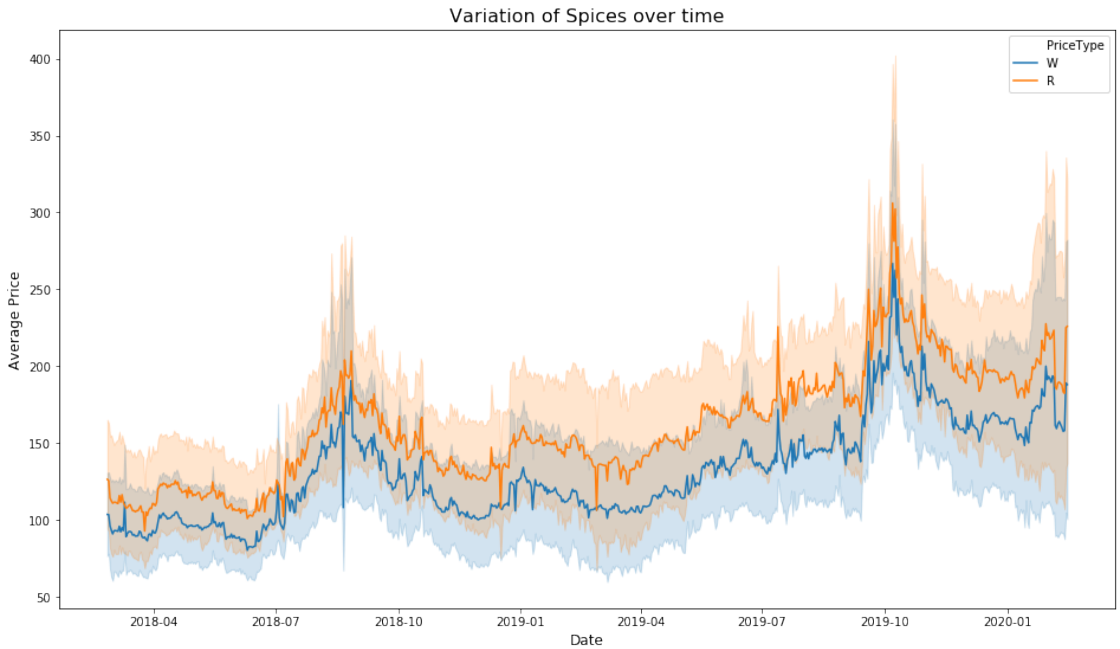
[25]:
```
# variation of Veg items over time
plt.figure(figsize=(16, 9))

sns.lineplot(x='Date', y='AvgPrice', data=veg_df, hue='PriceType')

plt.title('Variation of Veg over time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```



[26]:
```
# variation of Non_veg items over time
plt.figure(figsize=(16, 9))

sns.lineplot(x='Date', y='AvgPrice', data=non_veg_df, hue='PriceType')

plt.title('Variation of Non_Veg over time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```

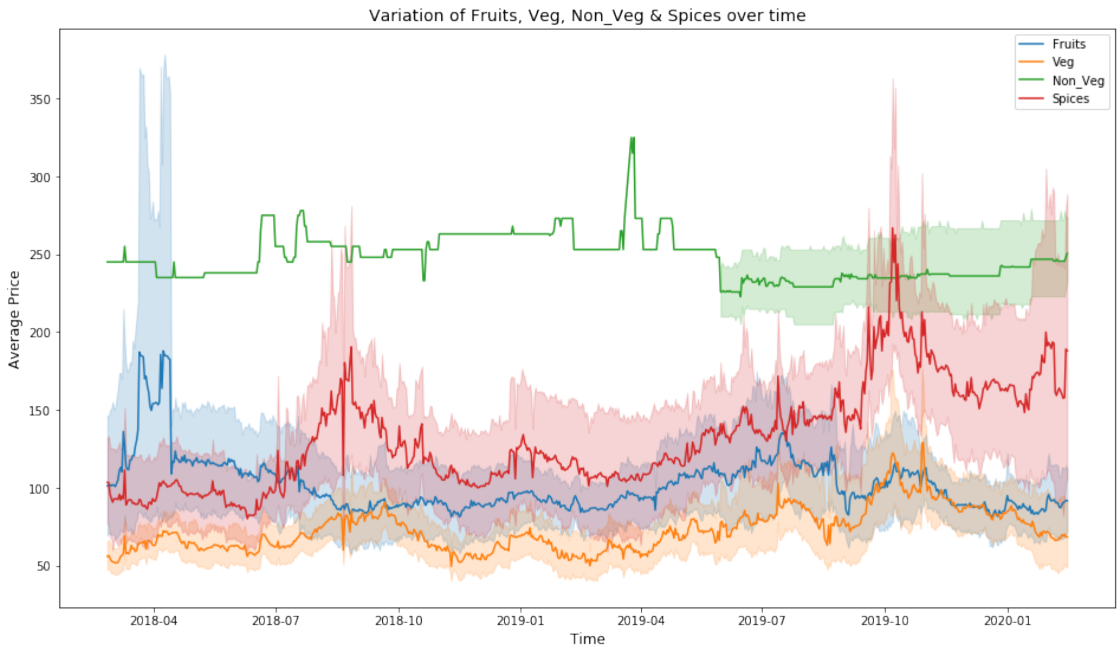


[27]:
```
# variation of Spices items over time
plt.figure(figsize=(16, 9))

sns.lineplot(x='Date', y='AvgPrice', data=spices_df, hue='PriceType')

plt.title('Variation of Spices over time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```



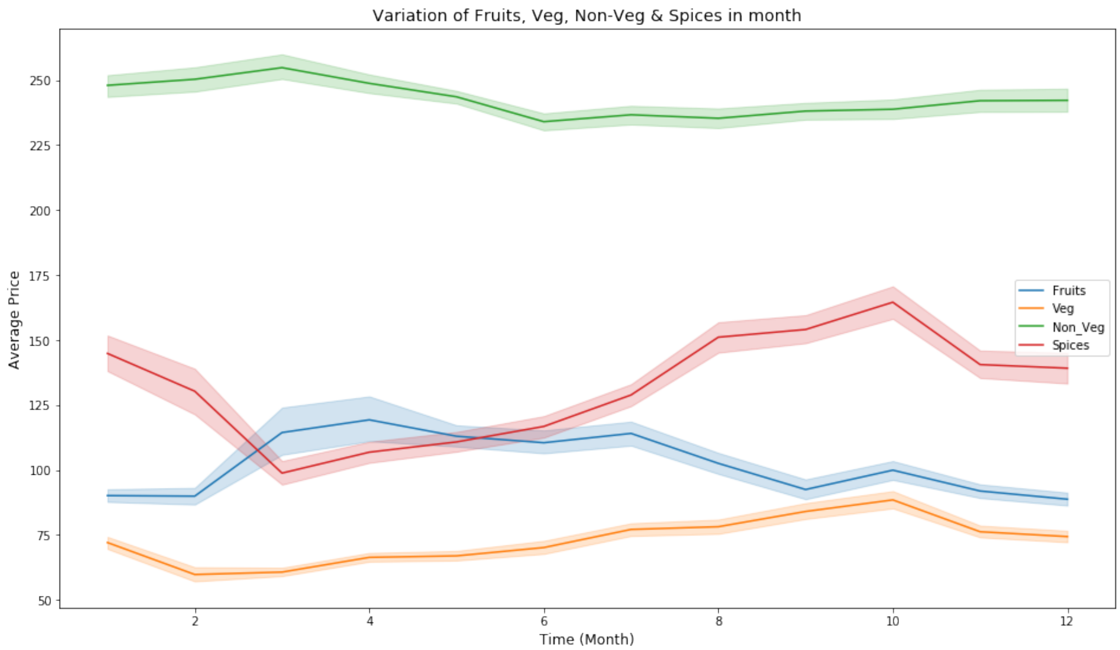
[28]:
```
# creating the wholesale dataframe of categorical data
w_fruits_df = fruits_df[fruits_df['PriceType']=='W']
w_veg_df = veg_df[veg_df['PriceType']=='W']
w_non_veg_df = non_veg_df[non_veg_df['PriceType']=='W']
w_spices_df = spices_df[spices_df['PriceType']=='W']
```

[29]:
```
# variation of all wholesale price of categorical veggis over time
plt.figure(figsize=(16, 9))

sns.lineplot(x='Date', y='AvgPrice', data=w_fruits_df, label='Fruits')
sns.lineplot(x='Date', y='AvgPrice', data=w_veg_df, label='Veg')
sns.lineplot(x='Date', y='AvgPrice', data=w_non_veg_df, label='Non_Veg')
sns.lineplot(x='Date', y='AvgPrice', data=w_spices_df, label='Spices')

plt.title('Variation of Fruits, Veg, Non_Veg & Spices over time', fontsize=14)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.legend()
plt.show()
```



[30]:

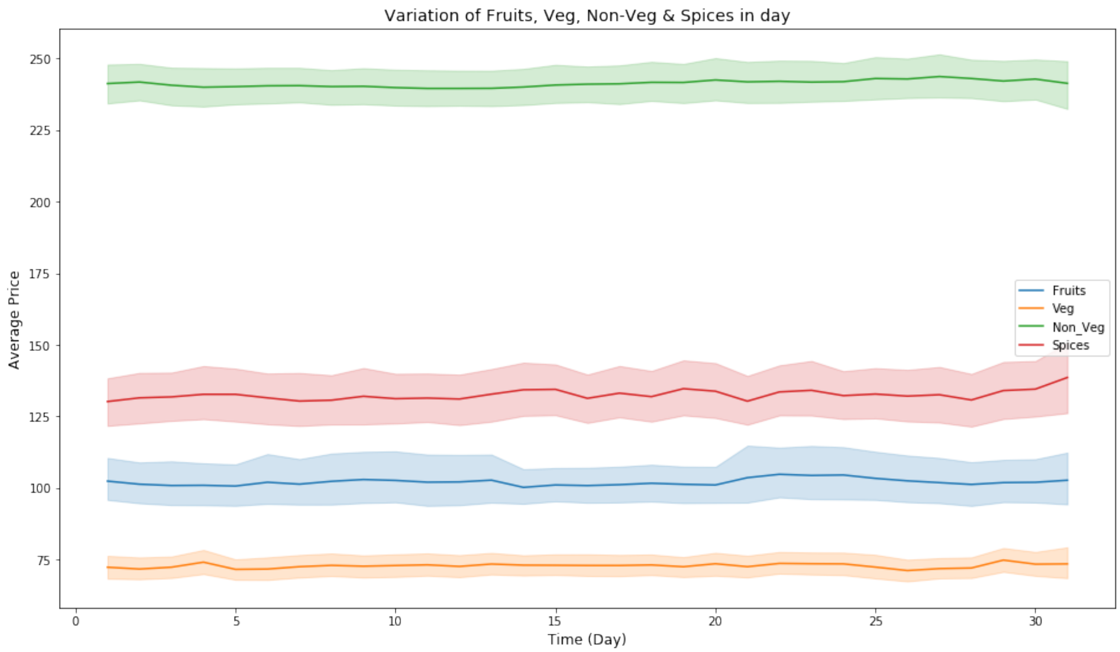
```
# variation of price over month of all categories
plt.figure(figsize=(16, 9))

sns.lineplot(x='Month', y='AvgPrice', data=w_fruits_df, label='Fruits')
sns.lineplot(x='Month', y='AvgPrice', data=w_veg_df, label='Veg')
sns.lineplot(x='Month', y='AvgPrice', data=w_non_veg_df, label='Non_Veg')
sns.lineplot(x='Month', y='AvgPrice', data=w_spices_df, label='Spices')

plt.title('Variation of Fruits, Veg, Non-Veg & Spices in month', fontsize=14)
plt.xlabel('Time (Month)', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```



- `Spices` price decrease at third month and gradually increase over upto October and again decrease.
- `Non-Veg` price decrease from third to sixth month and increase gradually.
- `Veg` pirce increase gradually from second month to October.
- `Fruits` price is maximum at third and starting of sixth month and then decrease gradually.

[31]:

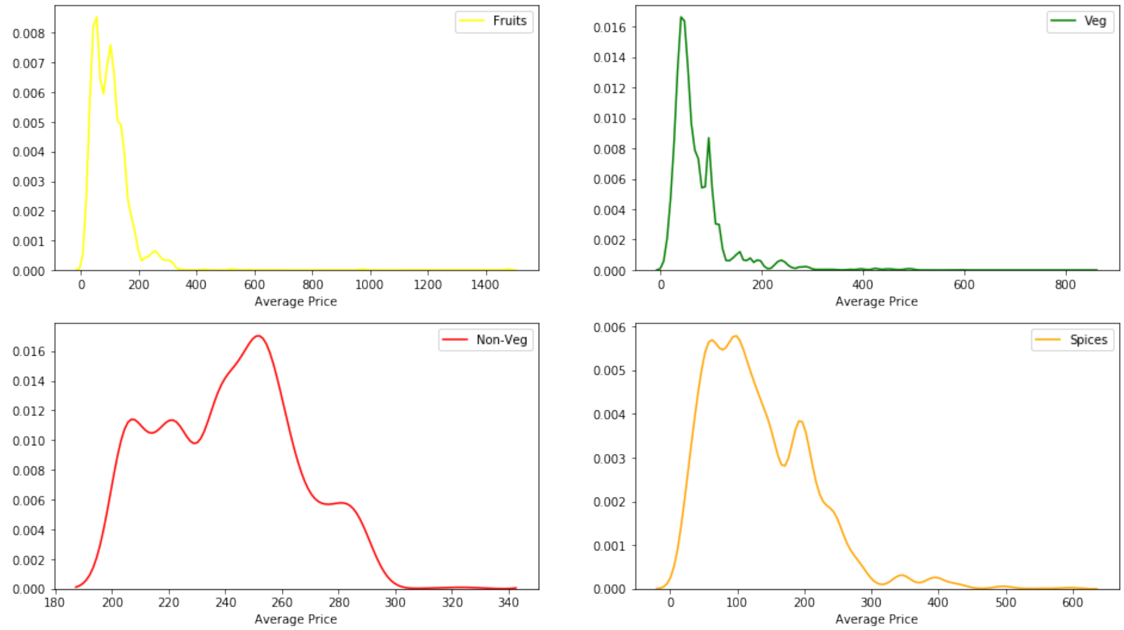
```python
# variation of price over day of all categories
plt.figure(figsize=(16, 9))

sns.lineplot(x='Day', y='AvgPrice', data=w_fruits_df, label='Fruits')
sns.lineplot(x='Day', y='AvgPrice', data=w_veg_df, label='Veg')
sns.lineplot(x='Day', y='AvgPrice', data=w_non_veg_df, label='Non_Veg')
sns.lineplot(x='Day', y='AvgPrice', data=w_spices_df, label='Spices')

plt.title('Variation of Fruits, Veg, Non-Veg & Spices in day', fontsize=14)
plt.xlabel('Time (Day)', fontsize=12)
plt.ylabel('Average Price', fontsize=12)

plt.show()
```



This show that price of all items doesn't variate over day time period.

[32]:
```
# histogram plot of all the category data
plt.figure(figsize=(16, 9))

plt.subplot(2, 2, 1)
sns.distplot(w_fruits_df['AvgPrice'], bins=10, label='Fruits', hist=False,␣
 ↪color='yellow')
plt.xlabel('Average Price')
plt.legend()

plt.subplot(2, 2, 2)
sns.distplot(w_veg_df['AvgPrice'], bins=10, label='Veg', hist=False,␣
 ↪color='green')
plt.xlabel('Average Price')
plt.legend()

plt.subplot(2, 2, 3)
sns.distplot(w_non_veg_df['AvgPrice'], bins=10, label='Non-Veg', hist=False,␣
 ↪color='red')
plt.xlabel('Average Price')
plt.legend()

plt.subplot(2, 2, 4)
sns.distplot(w_spices_df['AvgPrice'], bins=10, label='Spices', hist=False,␣
 ↪color='orange')
```

```python
plt.xlabel('Average Price')
plt.legend()

plt.show()
```



Feature Extraction

[33]:
```python
# copy of dataframe for feature dataframe
feature_df = df.copy()

feature_df.head()
```

[33]:
```
                       Items    Unit  MinPrice  MaxPrice  AvgPrice        Date  \
0        (   )   . .          30          35          33  2018-02-25
1                    . .          25          30          28  2018-02-25
2                      . .          20          23          22  2018-02-25
3                     . .          18          20          19  2018-02-25
4              . .          44          46          45  2018-02-25

  PriceType  Day  Month  Year
0         W   25      2  2018
1         W   25      2  2018
2         W   25      2  2018
3         W   25      2  2018
4         W   25      2  2018
```

Feature - 1

[34]:
```python
# converting wholesale data to 1 and retail to 0
def convert_pricetype(column):
    result = []

    for value in column:
        if value == 'W':
            result.append(1)
        else:
            result.append(0)

    return (result)

feature_df['PriceType'] = convert_pricetype(feature_df['PriceType'])
```

[35]:
```python
feature_df.groupby('PriceType').count()
```

[35]:
```
           Items   Unit  MinPrice  MaxPrice  AvgPrice   Date    Day  Month  \
PriceType
0          49646  49646     49646     49646     49646  49646  49646  49646
1          49656  49656     49656     49656     49656  49656  49656  49656

            Year
PriceType
0          49646
1          49656
```

[36]:
```python
# dropping the unwanted column
feature_df = feature_df.drop(columns=['Unit', 'MinPrice', 'MaxPrice', 'Day',␣
 ↪'Month', 'Year'])

feature_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99302 entries, 0 to 99301
Data columns (total 4 columns):
Items        99302 non-null object
AvgPrice     99302 non-null int64
Date         99302 non-null datetime64[ns]
PriceType    99302 non-null int64
dtypes: datetime64[ns](1), int64(2), object(1)
memory usage: 3.0+ MB
```

Feature - 2

[37]:
```python
# taking category of a veggis as a feature using one-hot encoding
is_f = []
is_v = []
is_n = []
```

```
is_s = []

for value in feature_df['Items']:
    is_f.append(1) if value in fruits else is_f.append(0)
    is_v.append(1) if value in veg else is_v.append(0)
    is_n.append(1) if value in non_veg else is_n.append(0)
    is_s.append(1) if value in spices else is_s.append(0)


feature_df['is_fruits'] = is_f
feature_df['is_veg'] = is_v
feature_df['is_non_veg'] = is_n
feature_df['is_spices'] = is_s

feature_df.head()
```

[37]:

```
                  Items  AvgPrice       Date  PriceType  is_fruits  is_veg  \
0        (   )        33 2018-02-25          1          1       0
1                        28 2018-02-25          1          1       0
2                          22 2018-02-25          1          0       1
3                         19 2018-02-25          1          0       1
4                     45 2018-02-25          1          0       0

   is_non_veg  is_spices
0           0          0
1           0          0
2           0          0
3           0          0
4           0          1
```

Featue - 3

[38]:

```
# using last 3 days price as a features
f_df = []

for v in unique_veggis:
    for ptype in [1, 0]:
        df_v = feature_df[(feature_df['Items']==v) &␣
 ↪(feature_df['PriceType']==ptype)]
        df_v = df_v.sort_values(by='Date', ascending=False)

        # Getting last 3 days price as a feature
        df_v['t-1'] = df_v.shift(-1)['AvgPrice']
        df_v['t-2'] = df_v.shift(-2)['AvgPrice']
        df_v['t-3'] = df_v.shift(-3)['AvgPrice']

        f_df.append(df_v)
```

```python
feature_df = pd.concat(f_df)
feature_df.tail()
```

[38]:

```
                Items  AvgPrice        Date  PriceType  is_fruits  is_veg  \
97480    (   )        290  2020-02-04          0          1       0
97316    (   )        290  2020-02-03          0          1       0
97152    (   )        290  2020-02-02          0          1       0
96988    (   )        305  2020-02-01          0          1       0
96824    (   )        295  2020-01-31          0          1       0

       is_non_veg  is_spices    t-1    t-2    t-3
97480           0          0  290.0  290.0  305.0
97316           0          0  290.0  305.0  295.0
97152           0          0  305.0  295.0    NaN
96988           0          0  295.0    NaN    NaN
96824           0          0    NaN    NaN    NaN
```

Here, we got the null data in our feature dataframe. Now, handling with the null value.

[39]:

```python
# removing NaN values columns
feature_df.dropna(inplace=True)

feature_df.tail()
```

[39]:

```
                Items  AvgPrice        Date  PriceType  is_fruits  is_veg  \
97969    (   )        290  2020-02-07          0          1       0
97807    (   )        290  2020-02-06          0          1       0
97644    (   )        290  2020-02-05          0          1       0
97480    (   )        290  2020-02-04          0          1       0
97316    (   )        290  2020-02-03          0          1       0

       is_non_veg  is_spices    t-1    t-2    t-3
97969           0          0  290.0  290.0  290.0
97807           0          0  290.0  290.0  290.0
97644           0          0  290.0  290.0  290.0
97480           0          0  290.0  290.0  305.0
97316           0          0  290.0  305.0  295.0
```

[40]:

```python
# Randomly shuffeling the data for the linear regression (IID)
feature_df = feature_df.sample(frac=1, random_state=42)
feature_df.tail()
```

[40]:

```
                         Items  AvgPrice        Date  PriceType  is_fruits  \
60430                       48  2019-06-10          0          0
79483                            105  2019-10-15          0          1
```

```
18123                        205  2018-07-13          0           0
74172        (   )        85  2019-09-07          0           1
1344                          52  2018-03-08          1           0

       is_veg  is_non_veg  is_spices    t-1    t-2    t-3
60430       0           0          1   48.0   48.0   48.0
79483       0           0          0  105.0  105.0  105.0
18123       0           0          1  140.0  205.0  255.0
74172       0           0          0   85.0   85.0   75.0
1344        1           0          0   52.0   62.0   62.0
```

Making X (For Feature) and y (For Prediction)

[41]:
```python
X = feature_df[['PriceType', 'is_fruits', 'is_veg', 'is_non_veg', 'is_spices',␣
↪'t-1', 't-2', 't-3']].values
y = feature_df['AvgPrice'].values

print('X shape: ', X.shape)
print('y shape: ', y.shape)
```

```
X shape:  (98582, 8)
y shape:  (98582,)
```

[42]:
```python
X
```

[42]:
```
array([[  0.,   0.,   1., …,  45.,  45.,  45.],
       [  1.,   0.,   0., …,  55.,  58.,  58.],
       [  0.,   0.,   1., …,  75.,  75.,  75.],
       …,
       [  0.,   0.,   0., …, 140., 205., 255.],
       [  0.,   1.,   0., …,  85.,  85.,  75.],
       [  1.,   0.,   1., …,  52.,  62.,  62.]])
```

[43]:
```python
y
```

[43]:
```
array([ 45,  55,  75, …, 205,  85,  52])
```

[44]:
```python
# inserting 1 on the first column of X matrix (for calculation)
X = np.insert(X, 0, values=1, axis=1)

X
```

[44]:
```
array([[  1.,   0.,   0., …,  45.,  45.,  45.],
       [  1.,   1.,   0., …,  55.,  58.,  58.],
       [  1.,   0.,   0., …,  75.,  75.,  75.],
       …,
       [  1.,   0.,   0., …, 140., 205., 255.],
```

```
        [  1.,   0.,   1., …,  85.,  85.,  75.],
        [  1.,   1.,   0., …,  52.,  62.,  62.]])
```

[45]:
```python
# making y 2d for easier calculation
y = y.reshape(-1, 1)
y.shape
```

[45]:
```
(98582, 1)
```

[46]:
```python
# seperating train, test and validation set in (60%, 20%, 20%)
total = len(X)

train = int(0.6 * total)
val = int(0.2 * total)

X_train = X[: train]
y_train = y[: train]

X_val = X[train: train + val]
y_val = y[train: train + val]

X_test = X[train + val: ]
y_test = y[train + val: ]

print('X_train shape: ', X_train.shape)
print('X_val shape: ', X_val.shape)
print('X_test shape: ', X_test.shape)
print('\n')

print('Y_train shape', y_train.shape)
print('Y_val shape', y_val.shape)
print('Y_test shape', y_test.shape)
```

```
X_train shape:  (59149, 9)
X_val shape:  (19716, 9)
X_test shape:  (19717, 9)


Y_train shape (59149, 1)
Y_val shape (19716, 1)
Y_test shape (19717, 1)
```

Normalization

We are using `Min-Max Normalization`.

[47]:
```python
# normalization
```

```
min_v = X_train[:, 6:].min()
max_v = X_train[:, 6:].max()
diff = max_v - min_v

def min_max_normalization(dataset):
    for i in range(0, dataset.shape[0]):
        for j in range(6, dataset.shape[1]):
            dataset[i][j] = (dataset[i][j] - min_v) / diff


min_max_normalization(X_train)
min_max_normalization(X_val)
min_max_normalization(X_test)
```

[48]:
```
print('X_train[0]: \n', X_train[0])
print('X_val[0]: \n', X_val[0])
print('X_test[0]: \n', X_test[0])
```

```
X_train[0]:
 [1.         0.         0.         1.         0.         0.
 0.02227723 0.02227723 0.02227723]
X_val[0]:
 [1.         0.         0.         1.         0.         0.
 0.02227723 0.02227723 0.02227723]
X_test[0]:
 [1.         1.         0.         1.         0.         0.
 0.13985149 0.13985149 0.13985149]
```

[49]:
```
# checking the dimension of all matrix
print('X_train shape: ', X_train.shape)
print('y_train shape: ', y_train.shape)

print('\n X_val shape: ', X_val.shape)
print('y_val shape', X_val.shape)

print('\n X_test shape: ', X_test.shape)
print('y_test shape: ', y_test.shape)
```

```
X_train shape:  (59149, 9)
y_train shape:  (59149, 1)

 X_val shape:  (19716, 9)
y_val shape (19716, 9)

 X_test shape:  (19717, 9)
y_test shape:  (19717, 1)
```

Grid Search parameters with val loss

Calucating the loss using Mean Square Error (MSE).

[50]:
```
# mean square error function
def MSE(h, y):
    return (0.5 * np.average((h - y) ** 2))
```

[51]:
```
# no. of iteration for hyperparameter and learning rate
np.random.seed(42)

params = {
    'num_of_iteration': 1000,
    'learning_rate': 0.01,
}

grid_params = {
    'num_of_iteration': [1000, 1500, 2000, 2500, 3000],
    'learning_rate': [0.001, 0.01, 0.1, 0.9]
}
```

Training Model

[52]:
```
def train_model(X_train, y_train, X_val, y_val, params):
    train_error = []

    # initializing random value of W and reshaping it in 2d
    w = np.random.rand(X_train.shape[1]).reshape(-1, 1)

    # finding w
    for i in range(0, params['num_of_iteration']):
        h_train = np.matmul(X_train, w)
        gradient = np.matmul(np.transpose(X_train), (h_train - y_train)) /␣
↪X_train.shape[0]

        train_loss = MSE(h_train, y_train)
        train_error.append(train_loss)

        w = w - (params['learning_rate'] * gradient)

    h_val = np.matmul(X_val, w)
    val_loss = MSE(h_val, y_val)

    print('For Hyper-parameter', params, val_loss)

    return (w, train_error)
```

[53]:
```
# train the model
train_model(X_train, y_train, X_val, y_val, params)
```

```
For Hyper-parameter {'num_of_iteration': 1000, 'learning_rate': 0.01}
2278.1736512647044
```

[53]:
```
(array([[ 96.08215484],
        [-11.77517748],
        [ 20.42927223],
        [-12.04465243],
        [ 42.57054135],
        [ 46.31434662],
        [ 41.36062755],
        [ 41.71124681],
        [ 40.91766697]]),
 [8874.274199745307,
  8704.049551790878,
  8539.39157613552,
  8380.112852026343,
  8226.032295138362,
  8076.974943198014,
  7932.771748860466,
  7793.259379595276,
  7658.280024343265,
  7527.681206715457,
  7401.315604512763,
  7279.040875352462,
  7160.719488194924,
  7046.218560570852,
  6935.409701316178,
  6828.168858628288,
  6724.376173263434,
  6623.915836701437,
  6526.675954109572,
  6432.548411943243,
  6341.42875002656,
  6253.216037961213,
  6167.812755717203,
  6085.124678263921,
  6005.060764104849,
  5927.533047583806,
  5852.456534835122,
  5779.749103254427,
  5709.33140437093,
  5641.126770006106,
  5575.06112160756,
  5511.062882650651,
  5449.062894004065,
  5388.9943321590345,
  5330.792630225325,
```

5274.395401600353,
5219.742366220979,
5166.775279310631,
5115.437862537226,
5065.6757375004445,
5017.436361469414,
4970.668965294747,
4925.324493421305,
4881.3555459306235,
4838.716322544327,
4797.362568522174,
4757.2515223906075,
4718.341865439896,
4680.593672930021,
4643.968366947454,
4608.42867085702,
4573.938565294822,
4540.463245650111,
4507.969080985717,
4476.423574348351,
4445.795324421769,
4416.05398847733,
4387.170246578088,
4359.115766993965,
4331.863172787047,
4305.386009527409,
4279.6587141012,
4254.656584574045,
4230.355751074056,
4206.733147659936,
4183.766485140863,
4161.434224815929,
4139.715553102046,
4118.590357020218,
4098.03920051118,
4078.0433015523013,
4058.584510048666,
4039.6452864721255,
4021.208681223009,
4003.2583146900624,
3985.778357984955,
3968.7535143285622,
3952.16900106695,
3936.010532295764,
3920.2643020724304,
3904.916968196288,
3889.9556365374224,

3875.3678458956533,
3861.1415533717163,
3847.2651202333304,
3833.7272982593813,
3820.5172165460726,
3807.624368759385,
3795.038600818761,
3782.7500989974183,
3770.7493784251947,
3759.027271980302,
3747.5749195568433,
3736.3837576953492,
3725.4455095640956,
3714.7521752792863,
3704.2960225526695,
3694.0695776554912,
3684.065616688096,
3674.277157144812,
3664.6974497641577,
3655.3199706546884,
3646.1384136871798,
3637.1466831441207,
3628.3388866178125,
3619.709328148677,
3611.252501595622,
3602.9630842306346,
3594.835930550002,
3586.866066294833,
3579.0486826738097,
3571.3791307813062,
3563.852916204287,
3556.4656938115772,
3549.2132627193528,
3542.091561426873,
3535.0966631167057,
3528.224771113875,
3521.47221449856,
3514.8354438671345,
3508.311027236558,
3501.8956460872278,
3495.5860915396465,
3489.3792606603533,
3483.2721528927573,
3477.2618666086323,
3471.3455957762185,
3465.520626740955,
3459.784335115047,

3454.134182772199,
3448.5677149439252,
3443.0825574140276,
3437.6764138079056,
3432.3470629735007,
3427.0923564507625,
3421.910216026647,
3416.7986313727706,
3411.7556577628807,
3406.7794138674954,
3401.868079623059,
3397.0198941731246,
3392.2331538790913,
3387.5062103981854,
3382.8374688263752,
3378.2253859040443,
3373.6684682822834,
3369.165270847772,
3364.714395104247,
3360.3144876086517,
3355.9642384601243,
3351.6623798400196,
3347.407684601258,
3343.198964905319,
3339.0350709052695,
3334.914889473281,
3330.8373429711123,
3326.8013880621224,
3322.8060145633985,
3318.850244336641,
3314.933130216502,
3311.053754975108,
3307.211230321533,
3303.404695935065,
3299.6333185310846,
3295.8962909584925,
3292.192831327597,
3288.5221821674245,
3284.8836096114837,
3281.2764026109935,
3277.699872174668,
3274.1533506341448,
3270.636190934195,
3267.147765946883,
3263.687467808855,
3260.254707280982,
3256.848913129593,

3253.469531528586,
3250.116025481688,
3246.7878742641983,
3243.4845728835508,
3240.205631558059,
3236.9505752132286,
3233.718942995052,
3230.510287799692,
3227.3241758190206,
3224.1601861014624,
3221.017910127638,
3217.89695140029,
3214.796925048028,
3211.7174574424084,
3208.658185827914,
3205.6187579643815,
3202.598831781468,
3199.5980750447443,
3196.616165033024,
3193.652788226544,
3190.707640005643,
3187.7804243595574,
3184.8708536050253,
3181.9786481143424,
3179.1035360525552,
3176.245253123495,
3173.403542324339,
3170.5781537084167,
3167.768844155985,
3164.9753771527003,
3162.19752257552,
3159.435056485809,
3156.687760929364,
3153.9554237431626,
3151.2378383685964,
3148.5348036709543,
3145.846123764965,
3143.1716078461905,
3140.511070028057,
3137.864329184355,
3135.231208797015,
3132.611536808962,
3130.00514548192,
3127.411871258955,
3124.831554631622,
3122.2640400115542,
3119.709175606341,

3117.1668132995596,
3114.6368085348054,
3112.119020203603,
3109.613310537052,
3107.119545001089,
3104.637592195244,
3102.1673237547748,
3099.7086142560524,
3097.2613411251136,
3094.82538454924,
3092.4006273914947,
3089.9869551080965,
3087.584255668535,
3085.1924194783514,
3082.8113393044755,
3080.4409102030468,
3078.081029449625,
3075.731596471718,
3073.3925127835514,
3071.063681922984,
3068.7450093905254,
3066.43640259037,
3064.13777077337,
3061.849024981901,
3059.5700779965523,
3057.300844284563,
3055.0412399499683,
3052.7911826853892,
3050.5505917254027,
3048.319387801451,
3046.097493098232,
3043.8848312115215,
3041.681327107382,
3039.4869070827053,
3037.3014987270526,
3035.125030885737,
3032.957433624126,
3030.7986381930955,
3028.648576995618,
3026.507183554445,
3024.374392480838,
3022.2501394443116,
3020.134361143372,
3018.026995277196,
3015.9279805182327,
3013.837256485694,
3011.754763719899,

3009.6804436574525,
3007.6142386072192,
3005.5560917270795,
3003.505947001424,
3001.463749219384,
2999.42944395375,
2997.402977540569,
2995.3842970594014,
2993.373350314202,
2991.37008581481,
2989.374452759035,
2987.386401015313,
2985.405881105902,
2983.4328441906273,
2981.46724205113,
2979.5090270756127,
2977.558152244072,
2975.6145711139943,
2973.678237806501,
2971.74910699292,
2969.8271338817945,
2967.9122742062796,
2966.004484211945,
2964.1037206449437,
2962.2099407405663,
2960.3231022121277,
2958.443163240214,
2956.5700824622527,
2954.703818962404,
2952.844332261764,
2950.991582308867,
2949.1455294704724,
2947.3061345226392,
2945.4733586420675,
2943.647163397701,
2941.8275107425825,
2940.014363005954,
2938.207682885599,
2936.4074334404036,
2934.6135780831487,
2932.8260805735067,
2931.0449050112557,
2929.270015829683,
2927.501377789191,
2925.738955971083,
2923.982715771539,
2922.2326228957554,

2920.488643352261,
2918.750743447399,
2917.01888977996,
2915.2930492359756,
2913.573188983651,
2911.8592764684563,
2910.1512794083364,
2908.449165789074,
2906.7529038597704,
2905.0624621284537,
2903.377809357818,
2901.6989145610655,
2900.025746997871,
2898.358276170457,
2896.6964718197746,
2895.040303921782,
2893.389742683832,
2891.744758541147,
2890.105322153395,
2888.4714044013504,
2886.842976383649,
2885.220009413616,
2883.6024750161923,
2881.990344924924,
2880.383591079042,
2878.7821856206083,
2877.186100891734,
2875.595309431875,
2874.0097839751843,
2872.4294974479403,
2870.854422966029,
2869.2845338324964,
2867.7198035351526,
2866.1602057442424,
2864.6057143101575,
2863.0563032612213,
2861.5119468015105,
2859.9726193087345,
2858.4382953321647,
2856.9089495906087,
2855.384556970429,
2853.865092523615,
2852.350531465885,
2850.840849174846,
2849.3360211881777,
2847.8360232018717,
2846.3408310684963,

2844.850420795507,
2843.364768543589,
2841.883850625036,
2840.40764350216,
2838.93612378574,
2837.4692682334976,
2836.0070537486044,
2834.54945737822,
2833.096456312061,
2831.648027880996,
2830.2041495556637,
2828.7647989451325,
2827.3299537955677,
2825.899591988934,
2824.473691541721,
2823.052230603694,
2821.635187456664,
2820.2225405132785,
2818.814268315845,
2817.41034953516,
2816.010762969372,
2814.6154875428574,
2813.2245023051137,
2811.8377864296826,
2810.4553192130743,
2809.077080073731,
2807.703048550982,
2806.333204304042,
2804.967527111008,
2803.605996867876,
2802.2485935875766,
2800.895297399026,
2799.5460885461844,
2798.200947387141,
2796.8598543931967,
2795.52279014798,
2794.1897353465547,
2792.8606707945596,
2791.5355774073496,
2790.214436209152,
2788.8972283322337,
2787.5839350160823,
2786.2745376065964,
2784.9690175552887,
2783.667356418499,
2782.3695358566156,
2781.0755376333123,

2779.7853436147866,
2778.49893576902,
2777.216296165038,
2775.937406972179,
2774.662250459381,
2773.3908089944694,
2772.123065043455,
2770.8590011698443,
2769.59860003395,
2768.3418443922224,
2767.0887170965743,
2765.8392010937227,
2764.5932794245377,
2763.3509352233937,
2762.112151717534,
2760.876912226435,
2759.6452001611888,
2758.41699902388,
2757.1922924069763,
2755.971063992724,
2754.753297552552,
2753.5389769464764,
2752.3280861225176,
2751.120609116121,
2749.916530049581,
2748.715833131473,
2747.518502656096,
2746.3245230029065,
2745.1338786359784,
2743.946554103447,
2742.7625340369755,
2741.5818031512144,
2740.4043462432733,
2739.2301481921972,
2738.0591939584406,
2736.8914685833543,
2735.7269571886795,
2734.5656449760304,
2733.407517226401,
2732.2525592996617,
2731.1007566340727,
2729.9520947457872,
2728.8065592283738,
2727.664135752331,
2726.5248100646145,
2725.3885679881623,
2724.2553954214272,

2723.125278337913,
2721.9982027857136,
2720.8741548870576,
2719.753120837852,
2718.635086907235,
2717.520039437131,
2716.407964841809,
2715.298849607441,
2714.1926802916705,
2713.089443523177,
2711.9891260012546,
2710.891714495378,
2709.7971958447865,
2708.705556958068,
2707.6167848127366,
2706.5308664548256,
2705.447788998481,
2704.3675396255485,
2703.290105585176,
2702.215474193415,
2701.1436328328177,
2700.07456895205,
2699.008270065497,
2697.9447237528757,
2696.8839176588494,
2695.825839492646,
2694.770477027678,
2693.7178181011664,
2692.667850613764,
2691.6205625291877,
2690.5759418738453,
2689.533976736473,
2688.494655267769,
2687.457965680033,
2686.4238962468094,
2685.392435302529,
2684.3635712421565,
2683.3372925208355,
2682.313587653548,
2681.292445214758,
2680.273853838076,
2679.257802215913,
2678.2442790991386,
2677.2332732967507,
2676.2247736755344,
2675.218769159735,
2674.215248730721,

2673.214201426663,
2672.2156163422032,
2671.2194826281316,
2670.2257894910676,
2669.234526193137,
2668.2456820516586,
2667.259246438825,
2666.27520878139,
2665.29355856036,
2664.3142853106833,
2663.3373786209413,
2662.362828133047,
2661.390623541939,
2660.4207545952813,
2659.4532110931627,
2658.4879828878043,
2657.525059883258,
2656.5644320351153,
2655.606089350217,
2654.6500218863625,
2653.6962197520234,
2652.744673106053,
2651.7953721574067,
2650.848307164857,
2649.9034684367166,
2648.960846330553,
2648.020431252917,
2647.082213659067,
2646.14618405269,
2645.212332985636,
2644.2806510576456,
2643.35112891608,
2642.4237572556567,
2641.4985268181827,
2640.575428392289,
2639.654452813176,
2638.735590962344,
2637.81883376734,
2636.9041722015013,
2635.9915972836957,
2635.0811000780745,
2634.172671693811,
2633.2663032848586,
2632.361986049697,
2631.4597112310844,
2630.5594701158143,
2629.661254034467,

2628.765054361172,
2627.8708625133586,
2626.9786699515225,
2626.088468178986,
2625.2002487416557,
2624.3140032277947,
2623.429723267784,
2622.5474005338856,
2621.66702674002,
2620.7885936415287,
2619.91209303495,
2619.037516757787,
2618.1648566882877,
2617.2941047452173,
2616.4252528876323,
2615.5582931146646,
2614.693217465295,
2613.83001801814,
2612.9686868912277,
2612.1092162417876,
2611.2515982660298,
2610.3958251989343,
2609.541889314039,
2608.689782923224,
2607.8394983765065,
2606.9910280618287,
2606.1443644048545,
2605.2994998687545,
2604.456426954012,
2603.6151381982077,
2602.7756261758304,
2601.9378834980603,
2601.10190281258,
2600.267676803374,
2599.4351981905256,
2598.604459730023,
2597.7754542135667,
2596.94817446837,
2596.1226133569703,
2595.298763777033,
2594.476618661165,
2593.65617097672,
2592.837413725615,
2592.020339944138,
2591.204942702763,
2590.391215105969,
2589.5791502920474,

2588.7687414329284,
2587.959981733988,
2587.1528644338805,
2586.3473828043443,
2585.543530150033,
2584.7412998083346,
2583.9406851491935,
2583.1416795749374,
2582.344276520098,
2581.5484694512447,
2580.754251866803,
2579.96161729689,
2579.170559303141,
2578.3810714785404,
2577.5931474472513,
2576.8067808644505,
2576.0219654161606,
2575.2386948190842,
2574.456962820438,
2573.676763197793,
2572.8980897589067,
2572.120936341564,
2571.3452968134175,
2570.5711650718245,
2569.7985350436884,
2569.0274006853033,
2568.2577559821952,
2567.4895949489646,
2566.7229116291323,
2565.9577000949844,
2565.1939544474208,
2564.4316688157987,
2563.6708373577835,
2562.9114542591988,
2562.1535137338733,
2561.397010023493,
2560.641937397456,
2559.888290152717,
2559.136062613652,
2558.385249131902,
2557.6358440862346,
2556.887841882397,
2556.141236952974,
2555.3960237572464,
2554.6521967810477,
2553.909750536623,
2553.168679562491,

2552.4289784233047,
2551.6906417097102,
2550.9536640382125,
2550.218040051038,
2549.4837644159993,
2548.750831826354,
2548.01923700068,
2547.288974682734,
2546.5600396413242,
2545.8324266701725,
2545.1061305877884,
2544.381146237336,
2543.657468486504,
2542.935092227377,
2542.214012376307,
2541.4942238737876,
2540.7757216843243,
2540.0585007963086,
2539.342556221894,
2538.627882996873,
2537.914476180547,
2537.202330855611,
2536.491442128023,
2535.7818051268887,
2535.0734150043368,
2534.3662669353985,
2533.66035611789,
2532.9556777722896,
2532.2522271416233,
2531.549999491345,
2530.848990109217,
2530.1491943052,
2529.450607411332,
2528.7532247816107,
2528.0570417918902,
2527.3620538397527,
2526.668256344407,
2525.9756447465697,
2525.2842145083537,
2524.5939611131616,
2523.904880065569,
2523.216966891218,
2522.5302171367057,
2521.8446263694805,
2521.160190177725,
2520.476904170257,
2519.794763976418,

2519.1137652459693,
2518.4339036489823,
2517.7551748757396,
2517.077574636623,
2516.4010986620165,
2515.725742702198,
2515.0515025272407,
2514.3783739269056,
2513.706352710544,
2513.035434706997,
2512.3656157644896,
2511.6968917505383,
2511.0292585518446,
2510.362712074202,
2509.697248242394,
2509.032863000097,
2508.3695523097863,
2507.7073121526328,
2507.0461385284143,
2506.386027455415,
2505.7269749703337,
2505.0689771281864,
2504.412030002214,
2503.7561296837907,
2503.1012722823257,
2502.4474539251755,
2501.794670757552,
2501.14291894243,
2500.4921946604522,
2499.842494109849,
2499.1938135063383,
2498.5461490830403,
2497.8994970903927,
2497.253853796054,
2496.6092154848234,
2495.9655784585475,
2495.322939036039,
2494.681293552985,
2494.0406383618647,
2493.400969831863,
2492.7622843487843,
2492.1245783149693,
2491.487848149212,
2490.8520902866726,
2490.217301178799,
2489.58347729324,
2488.950615113767,

2488.3187111401876,
2487.687761888269,
2487.057763889655,
2486.428713691784,
2485.800607857814,
2485.1734429665366,
2484.547215612304,
2483.921922404946,
2483.2975599696947,
2482.6741249471047,
2482.051613992977,
2481.430023778281,
2480.8093509890814,
2480.189592326456,
2479.5707445064268,
2478.9528042598795,
2478.3357683324903,
2477.7196334846553,
2477.1043964914093,
2476.490054142359,
2475.8766032416047,
2475.264040607671,
2474.652363073434,
2474.041567486048,
2473.4316507068716,
2472.8226096114035,
2472.2144410892065,
2471.607142043835,
2471.000709392772,
2470.395140067355,
2469.790431012704,
2469.18657918766,
2468.5835815647088,
2467.9814351299187,
2467.38013688287,
2466.7796838365866,
2466.1800730174728,
2465.5813014652435,
2464.983366232855,
2464.3862643864486,
2463.789993005275,
2463.1945491816327,
2462.5999300208077,
2462.006132641,
2461.413154173268,
2460.8209917614568,
2460.2296425621435,

2459.6391037445655,
2459.049372490564,
2458.460445994518,
2457.8723214632837,
2457.2849961161314,
2456.6984671846863,
2456.112731912866,
2455.5277875568167,
2454.94363138486,
2454.3602606774266,
2453.777672726998,
2453.1958648380473,
2452.6148343269824,
2452.034578522083,
2451.455094763443,
2450.876380402914,
2450.2984328040498,
2449.721249342041,
2449.1448274036657,
2448.5691643872265,
2447.9942577024985,
2447.4201047706715,
2446.846703024291,
2446.2740499072056,
2445.7021428745134,
2445.1309793924997,
2444.560556938591,
2443.990873001294,
2443.421925080141,
2442.853710685643,
2442.2862273392284,
2441.719472573193,
2441.153443930646,
2440.5881389654573,
2440.023555242206,
2439.4596903361276,
2438.896541833059,
2438.334107329392,
2437.7723844320176,
2437.2113707582776,
2436.6510639359117,
2436.0914616030072,
2435.53256140795,
2434.9743610093738,
2434.4168580761084,
2433.8600502871327,
2433.303935331526,

```
2432.7485109084146,
2432.193774726928,
2431.639724506146,
2431.0863579750558,
2430.533672872496,
2429.9816669471193,
2429.430337957335,
2428.879683671268,
2428.329701866708,
2427.7803903310687,
2427.231746861332,
2426.6837692640097,
2426.1364553550943,
2425.589802960013,
2425.043809913584,
2424.498474059969,
2423.953793252628,
2423.4097653542785,
2422.8663882368446,
2422.323659781418,
2421.781577878212,
2421.2401404265156,
2420.6993453346536,
2420.159190519941,
2419.61967390864,
2419.0807934359154,
2418.5425470457963,
2418.0049326911294,
2417.467948333538,
2416.9315919433798,
2416.3958614997064,
2415.860754990219,
2415.326270411231,
2414.7924057676214,
2414.259159072798,
2413.726528348656,
2413.1945116255347,
2412.6631069421824,
2412.132312345709,
2411.602125891554,
2411.07254564344,
2410.543569673338,
2410.0151960614253,
2409.487422896047,
2408.9602482736773,
2408.433670298882,
2407.907687084277,
```

2407.3822967504934,
2406.8574974261373,
2406.333287247751,
2405.809664359779,
2405.286626914527,
2404.7641730721252,
2404.2423010004927,
2403.7210088753,
2403.2002948799304,
2402.6801572054446,
2402.160594050547,
2401.6416036215446,
2401.123184132315,
2400.6053338042693,
2400.0880508663154,
2399.571333554823,
2399.05518011359,
2398.5395887938066,
2398.0245578540193,
2397.5100855600954,
2396.996170185194,
2396.482810009724,
2395.970003321315,
2395.4577484147812,
2394.946043592089,
2394.434887162323,
2393.924277441651,
2393.414212753292,
2392.904691427483,
2392.3957118014464,
2391.887272219357,
2391.379371032307,
2390.872006598276,
2390.3651772821017,
2389.8588814554405,
2389.3531174967397,
2388.847883791208,
2388.3431787307786,
2387.839000714082,
2387.335348146411,
2386.832219439694,
2386.329613012461,
2385.8275272898113,
2385.3259607033874,
2384.8249116913394,
2384.3243786983,
2383.824360175348,

2383.324854579985,
2382.8258603761,
2382.3273760339416,
2381.82940003009,
2381.331930847425,
2380.834966975098,
2380.338506908502,
2379.842549149244,
2379.3470922051156,
2378.8521345900626,
2378.3576748241594,
2377.863711433577,
2377.3702429505574,
2376.8772679133876,
2376.3847848663636,
2375.892792359772,
2375.4012889498554,
2374.9102731987873,
2374.4197436746463,
2373.929698951387,
2373.440137608812,
2372.9510582325465,
2372.4624594140096,
2371.9743397503908,
2371.486697844621,
2370.999532305345,
2370.5128417468964,
2370.026624789273,
2369.5408800581076,
2369.0556061846455,
2368.5708018057153,
2368.086465563704,
2367.6025961065343,
2367.119192087635,
2366.6362521659184,
2366.1537750057537,
2365.671759276944,
2365.1902036546994,
2364.709106819611,
2364.2284674576294,
2363.74828426004,
2363.268555923434,
2362.7892811496904,
2362.3104586459463,
2361.8320871245764,
2361.3541653031684,
2360.876691904496,

```
 2360.3996656565023,
 2359.9230852922674,
 2359.4469495499907,
 2358.971257172966,
 2358.4960069095587,
 2358.021197513182,
 2357.5468277422738,
 2357.0728963602733,
 2356.5994021356028,
 2356.1263438416345,
 2355.6537202566815,
 2355.181530163964,
 2354.709772351594,
 2354.2384456125483,
 2353.7675487446504,
 2353.2970805505456,
 2352.8270398376803,
 2352.3574254182813,
 2351.8882361093306,
 2351.419470732548,
 2350.9511281143664,
 2350.4832070859134,
 2350.0157064829855,
 2349.548625146033,
 2349.081961920134])
```

Grid Search

[54]:
```
# grid search for hyper-parameter tunning
def grid_search(X_train, y_train, X_val, y_val, grid_params):

    import itertools
    grid = list(itertools.product(grid_params['num_of_iteration'],␣
 ↪grid_params['learning_rate']))

    for g in grid:
        p = {
            'num_of_iteration': g[0],
            'learning_rate': g[1]
        }

        train_model(X_train, y_train, X_val, y_val, p)
```

[55]:
```
grid_search(X_train, y_train, X_test, y_test, grid_params)
```

```
For Hyper-parameter {'num_of_iteration': 1000, 'learning_rate': 0.001}
3754.6487681036897
For Hyper-parameter {'num_of_iteration': 1000, 'learning_rate': 0.01}
```

```
2446.340096156169
For Hyper-parameter {'num_of_iteration': 1000, 'learning_rate': 0.1}
897.9859750677152
For Hyper-parameter {'num_of_iteration': 1000, 'learning_rate': 0.9}
156.78689194347126
For Hyper-parameter {'num_of_iteration': 1500, 'learning_rate': 0.001}
3431.9557441612837
For Hyper-parameter {'num_of_iteration': 1500, 'learning_rate': 0.01}
2247.217125853829
For Hyper-parameter {'num_of_iteration': 1500, 'learning_rate': 0.1}
577.4731036405151
For Hyper-parameter {'num_of_iteration': 1500, 'learning_rate': 0.9}
156.14212819877824
For Hyper-parameter {'num_of_iteration': 2000, 'learning_rate': 0.001}
3263.99920293618
For Hyper-parameter {'num_of_iteration': 2000, 'learning_rate': 0.01}
2089.825105566918
For Hyper-parameter {'num_of_iteration': 2000, 'learning_rate': 0.1}
393.4438106493658
For Hyper-parameter {'num_of_iteration': 2000, 'learning_rate': 0.9}
154.91830566886554
For Hyper-parameter {'num_of_iteration': 2500, 'learning_rate': 0.001}
3148.40885620937
For Hyper-parameter {'num_of_iteration': 2500, 'learning_rate': 0.01}
1954.2598326610225
For Hyper-parameter {'num_of_iteration': 2500, 'learning_rate': 0.1}
288.3581324464173
For Hyper-parameter {'num_of_iteration': 2500, 'learning_rate': 0.9}
153.65567106370497
For Hyper-parameter {'num_of_iteration': 3000, 'learning_rate': 0.001}
3043.58941953272
For Hyper-parameter {'num_of_iteration': 3000, 'learning_rate': 0.01}
1836.580396751928
For Hyper-parameter {'num_of_iteration': 3000, 'learning_rate': 0.1}
228.64345492934012
For Hyper-parameter {'num_of_iteration': 3000, 'learning_rate': 0.9}
152.50416097066324
```

Train loss plot on the best model

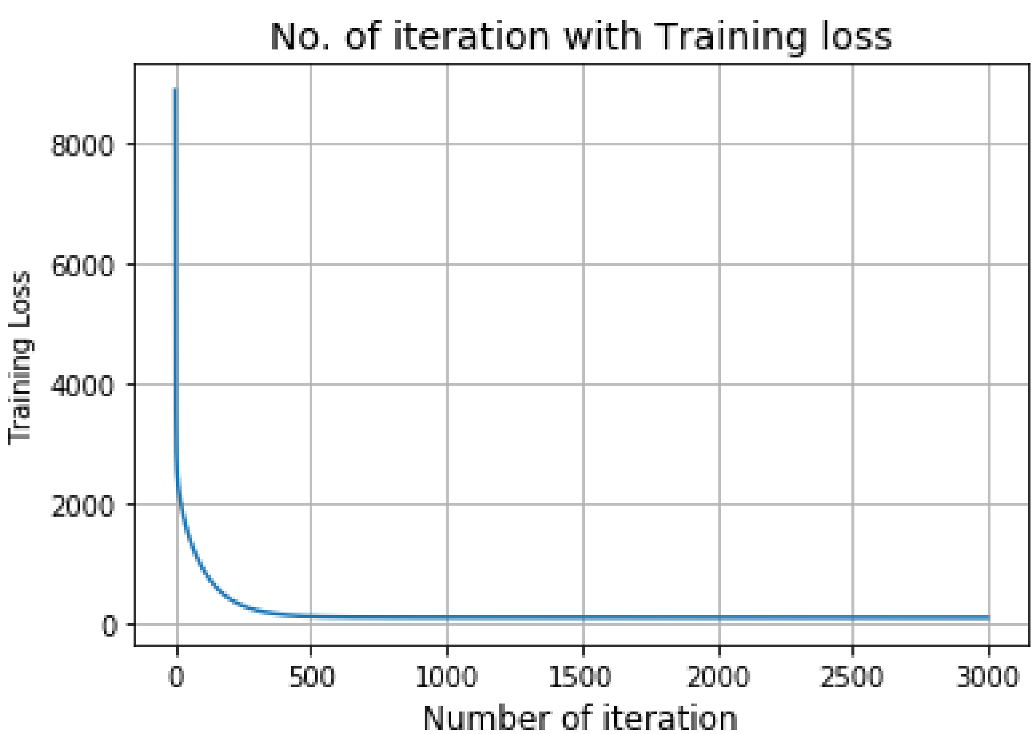
Final model and parameter for calculating weights (by choosing less loss).

[56]:
```
final_params = {
    'num_of_iteration': 3000,
    'learning_rate': 0.9,
}
```

```
model_weights, train_loss = train_model(X_train, y_train, X_val, y_val,␣
↪final_params)

print(model_weights)
```

```
For Hyper-parameter {'num_of_iteration': 3000, 'learning_rate': 0.9}
123.57753974994084
[[ 8.68014376e+00]
 [-1.76892798e-01]
 [ 9.64871272e-01]
 [ 7.32470442e-01]
 [ 1.65068490e+00]
 [ 1.32343908e+00]
 [ 6.37628847e+02]
 [ 5.17166495e+02]
 [ 4.54273803e+02]]
```

[57]:
```
# plotting the train loss with number of iteration
plt.plot(train_loss)

plt.title('No. of iteration with Training loss', fontsize=14)
plt.xlabel('Number of iteration', fontsize=12)
plt.ylabel('Training Loss')

plt.grid()
plt.show()
```



Model Evaluation

For model evaluation, we will use the testing dataset.

[58]:
```
# implementing R2 and adjusted R2
y_mean = np.mean(y_test)
h_test = np.matmul(X_test, model_weights)
n = X_test.shape[0]
k = X_test.shape[1] - 1

SSE = np.average((h_test - y_test) ** 2)
SST = np.average((y_test - y_mean) ** 2)

R2 = 1 - (SSE / SST)
adjusted_R2 = 1 - (SSE / (n - k - 1)) / (SST / (n - 1))
```

[59]:
```
print('R2 score: ', R2)
print('Adjusted R2 score: ', adjusted_R2)
```

```
R2 score:  0.9545621471718602
Adjusted R2 score:  0.9545437027420537
```

Here, our R2 score and adjusted R2 score is colse to 1 which means our model work pretty good for the prediction of price of veggis. This model can be furture better if we map the seasonality of the veggis, exported veggis, demand, etc for more accurate prediction.

[ ]:
```
```