Memory Optimization using Pandas Library

The performance of Python program with respect large dataset can be enhanced by using Pandas library.

Pandas native techniques can be practiced to optimize the performance by appropriately utilze the memory and optimize the performance.

In this blog we shall learn few basic practices which allow us to handle large datasets in an efficient way.

Tip 1.

While we are working with large datasets, one of the simplest approach to apply is to identify the specific variables that are of interest and load only those variables for processing rather importing the entire dataset into memory.

In [1]:
```python
import pandas as pd

df = pd.read_csv("sample.csv")

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                   Non-Null Count   Dtype 
---  ------                   --------------   ----- 
 0   Age                      300000 non-null  int64 
 1   Gender                   300000 non-null  object
 2   AppointmentRegistration  300000 non-null  object
 3   ApointmentData           300000 non-null  object
 4   DayOfTheWeek             300000 non-null  object
 5   Status                   300000 non-null  object
 6   Diabetes                 300000 non-null  int64 
 7   Alcoolism                300000 non-null  int64 
 8   HiperTension             300000 non-null  int64 
 9   Handcap                  300000 non-null  int64 
 10  Smokes                   300000 non-null  int64 
 11  Scholarship              300000 non-null  int64 
 12  Tuberculosis             300000 non-null  int64 
 13  Sms_Reminder             300000 non-null  int64 
 14  AwaitingTime             300000 non-null  int64 
dtypes: int64(10), object(5)
memory usage: 34.3+ MB
```

In [2]:
```python
df.info(verbose=False, memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Columns: 15 entries, Age to AwaitingTime
dtypes: int64(10), object(5)
memory usage: 120.2 MB
```

The option = `deep` in info() function is used to perform a real memory usage. The calculation is performed at the cost of computational resources. If we do not use `deep` option then the memory usage is based on the column dtype and number of rows. Assuming values consume the same memory amount for corresponding dtypes are calculated.

The memory consumption for the sample dataset is 120.2 MB.

The question we need to ask ourselfs is:

- Q. Do we require the entire data for processing?

Optimize the Memory Usage

Assume our interest is only for the columns: `Age` and `Status`.

- Why not import only these two columns?

This approach will reduce the memory consuption drastically.

In [3]:
```
# Selecting only the required columns to be imported into Python memory

df = pd.read_csv("sample.csv", usecols=["Age", "Status"])

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   Age     300000 non-null  int64 
 1   Status  300000 non-null  object
dtypes: int64(1), object(1)
memory usage: 4.6+ MB
```

- The `usecols` parameter of read_csv() function can filter out all other columns and import only the required fields.
- This will allow us to utilize the memory in an optimized manner which can enhance the performance of your code.

In [4]:
```
df.info(verbose=False, memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Columns: 2 entries, Age to Status
dtypes: int64(1), object(1)
memory usage: 20.6 MB
```

Tip 2.

Choose the Appropriate Data types.

In Python programming the standard data types are used. Every column is automatically infered for the data types based on the data it holds.

Each of these standard data types have predefined structure and storage defined.

Lets discuss the numerical data types and their memory consumption.

	Data Type	Description
1	bool	Boolean type (True or False) stored as a byte
2	int	Default integer type (int64 or int32)
3	int8	Byte (-128 to 127)
4	int16	Integer (-32768 to 32767)
5	int32	Integer (-2147483648 to 2147483647)
6	int64	Integer (-9223372036854775808 to 9223372036854775807)
7	uint8	Unsigned integer (0 to 255)
8	uint16	Unsigned integer (0 to 65535)
9	uint32	Unsigned integer (0 to 4294967295)
10	uint64	Unsigned integer (0 to 18446744073709551615)
11	float	Shorthand for float64.
12	float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
13	float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
14	float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
15	complex	complex number, 128
16	complex64	Complex number, represented by two 32-bit floats
17	complex128	Complex number, represented by two 64-bit floats

In [10]:

```
import pandas as pd

df = pd.read_csv("sample.csv")

df.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                   Non-Null Count   Dtype 
---  ------                   --------------   ----- 
 0   Age                      300000 non-null  int64 
 1   Gender                   300000 non-null  object
 2   AppointmentRegistration  300000 non-null  object
 3   ApointmentData           300000 non-null  object
 4   DayOfTheWeek             300000 non-null  object
 5   Status                   300000 non-null  object
 6   Diabetes                 300000 non-null  int64 
 7   Alcoolism                300000 non-null  int64 
 8   HiperTension             300000 non-null  int64 
 9   Handcap                  300000 non-null  int64 
 10  Smokes                   300000 non-null  int64 
 11  Scholarship              300000 non-null  int64 
 12  Tuberculosis             300000 non-null  int64 
 13  Sms_Reminder             300000 non-null  int64 
 14  AwaitingTime             300000 non-null  int64 
dtypes: int64(10), object(5)
memory usage: 120.2 MB
```

In [11]:

```
min_val = df.min(numeric_only = True)
max_val = df.max(numeric_only = True)

print(min_val)
```

```
print('\n')
print(max_val)
```

```
Age              -2
Diabetes          0
Alcoolism         0
HiperTension      0
Handcap           0
Smokes            0
Scholarship       0
Tuberculosis      0
Sms_Reminder      0
AwaitingTime   -398
dtype: int64


Age             113
Diabetes          1
Alcoolism         1
HiperTension      1
Handcap           4
Smokes            1
Scholarship       1
Tuberculosis      1
Sms_Reminder      2
AwaitingTime     -1
dtype: int64
```

Importing the entire sample data consumes 120.2 MB of memory. To optimize the memory utilization we can alter the data types from int64 to int32, int16, or int8 as appropriate.

For Example: `Age` column consists of positive 2 digit numbers, so we can **typecast** `Age` to int8 or uint8.

In [12]:
```
df.Age.memory_usage(deep=True)
```

Out[12]:
```
2400128
```

In [13]:
```
# Typecast

df.Age = data.Age.astype('int8')

df.Age.memory_usage(deep=True)
```

Out[13]:
```
300128
```

Observe the difference in the size of the data post typecasting it to int8.

From **2.28 MB** it has reduced to **0.28 MB**.

Typecasting all the numeric columns will reduce the overall memory consumption for the dataframe.

In [14]:
```
df.Age = df.Age.astype('int8')
df.Diabetes = df.Diabetes.astype('int8')
df.Alcoolism = df.Alcoolism.astype('int8')
df.HiperTension = df.HiperTension.astype('int8')
df.Handcap = df.Handcap.astype('int8')
df.Smokes = df.Smokes.astype('int8')
df.Scholarship = df.Scholarship.astype('int8')
df.Tuberculosis = df.Tuberculosis.astype('int8')
df.Sms_Reminder = df.Sms_Reminder.astype('int8')
df.AwaitingTime = df.AwaitingTime.astype('int8')
```

In [15]:
```python
df.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype 
---  ------                    --------------   ----- 
 0   Age                       300000 non-null  int8  
 1   Gender                    300000 non-null  object
 2   AppointmentRegistration   300000 non-null  object
 3   ApointmentData            300000 non-null  object
 4   DayOfTheWeek              300000 non-null  object
 5   Status                    300000 non-null  object
 6   Diabetes                  300000 non-null  int8  
 7   Alcoolism                 300000 non-null  int8  
 8   HiperTension              300000 non-null  int8  
 9   Handcap                   300000 non-null  int8  
 10  Smokes                    300000 non-null  int8  
 11  Scholarship               300000 non-null  int8  
 12  Tuberculosis              300000 non-null  int8  
 13  Sms_Reminder              300000 non-null  int8  
 14  AwaitingTime              300000 non-null  int8  
dtypes: int8(10), object(5)
memory usage: 100.2 MB
```

Applying this strategy has brought down the size of the data from 120.2 MB to 100.2 MB

Further we can also convert the values into boolean type if the data is binary in nature. Use `unique()` or `value_counts()` functions to verify the object columns.

Tip3.

Reduce the memory consumption of a catergorical values by renameing the values.

In [20]:
```python
import pandas as pd

df = pd.read_csv("sample.csv")

df.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype 
---  ------                    --------------   ----- 
 0   Age                       300000 non-null  int64 
 1   Gender                    300000 non-null  object
 2   AppointmentRegistration   300000 non-null  object
 3   ApointmentData            300000 non-null  object
 4   DayOfTheWeek              300000 non-null  object
 5   Status                    300000 non-null  object
 6   Diabetes                  300000 non-null  int64 
 7   Alcoolism                 300000 non-null  int64 
 8   HiperTension              300000 non-null  int64 
 9   Handcap                   300000 non-null  int64 
 10  Smokes                    300000 non-null  int64 
 11  Scholarship               300000 non-null  int64 
 12  Tuberculosis              300000 non-null  int64 
 13  Sms_Reminder              300000 non-null  int64 
 14  AwaitingTime              300000 non-null  int64 
```

```
dtypes: int64(10), object(5)
memory usage: 120.2 MB
```

In [21]:
```python
# Example:
df.DayOfTheWeek.unique()
```

Out[21]:
```
array(['Wednesday', 'Tuesday', 'Thursday', 'Friday', 'Monday', 'Saturday',
       'Sunday'], dtype=object)
```

In [22]:
```python
df.DayOfTheWeek.memory_usage(deep=True)
```

Out[22]:
```
19276698
```

The 'DayOfTheWeek' column is of (19276698 bytes) 18.38 MB, this is due to the full form of the weekday. The size can be reduced by altering the full form with a short form represetation. This can be achived by using a datatype called `category`. The data which is repeated in non-numerical column is stored in a comparatively compact representation.

In [23]:
```python
# Example:
df.DayOfTheWeek = df.DayOfTheWeek.astype('category')

df.DayOfTheWeek.memory_usage(deep=True)
```

Out[23]:
```
300877
```

The memory consumption has reduced to **0.28 MB** from **18.38 MB**, this is a huge compression, especially if we have a big data in terms of the number of rows.

Tip4.

Conversion of Date columns to Datetime will impact the memory usage very effectively. The values are inferred as string/object type by pandas library by default.

In [24]:
```python
import pandas as pd

df = pd.read_csv("sample.csv")

df.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                   Non-Null Count   Dtype 
---  ------                   --------------   ----- 
 0   Age                      300000 non-null  int64 
 1   Gender                   300000 non-null  object
 2   AppointmentRegistration  300000 non-null  object
 3   ApointmentData           300000 non-null  object
 4   DayOfTheWeek             300000 non-null  object
 5   Status                   300000 non-null  object
 6   Diabetes                 300000 non-null  int64 
 7   Alcoolism                300000 non-null  int64 
 8   HiperTension             300000 non-null  int64 
 9   Handcap                  300000 non-null  int64 
 10  Smokes                   300000 non-null  int64 
 11  Scholarship              300000 non-null  int64 
 12  Tuberculosis             300000 non-null  int64 
 13  Sms_Reminder             300000 non-null  int64 
 14  AwaitingTime             300000 non-null  int64 
```

```
dtypes: int64(10), object(5)
memory usage: 120.2 MB
```

In [25]:
```python
df.AppointmentRegistration.memory_usage(deep=True)
```

Out[25]:
```
23100128
```

The column `AppointmentRegistration` is inferred as object type by default. Changing this column into datetime will bring the memory consmuption drastically down. The current memory usage is rounded to **22 MB**. Lets convert this data to datetime and measure the consmption.

In [29]:
```python
# Example:

df.AppointmentRegistration = pd.to_datetime(data.AppointmentRegistration)
```

In [30]:
```python
df.AppointmentRegistration.memory_usage(deep=True)
```

Out[30]:
```
2400128
```

Post the conversion to datetime the memory consumption has come down to **2.28 MB** from **22 MB**.

Conclusion:

Let us implement all the techniques that we have discussed in this blog on the sample data and see the over all effect in memory utilization.

In [36]:
```python
# Example:
import pandas as pd

data = pd.read_csv("sample.csv")

data.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                   Non-Null Count   Dtype 
---  ------                   --------------   ----- 
 0   Age                      300000 non-null  int64 
 1   Gender                   300000 non-null  object
 2   AppointmentRegistration  300000 non-null  object
 3   ApointmentData           300000 non-null  object
 4   DayOfTheWeek             300000 non-null  object
 5   Status                   300000 non-null  object
 6   Diabetes                 300000 non-null  int64 
 7   Alcoolism                300000 non-null  int64 
 8   HiperTension             300000 non-null  int64 
 9   Handcap                  300000 non-null  int64 
 10  Smokes                   300000 non-null  int64 
 11  Scholarship              300000 non-null  int64 
 12  Tuberculosis             300000 non-null  int64 
 13  Sms_Reminder             300000 non-null  int64 
 14  AwaitingTime             300000 non-null  int64 
dtypes: int64(10), object(5)
memory usage: 120.2 MB
```

In [32]:
```python
# Typecasting the Numeric values to appropriate data type (int8)
numeric_features = data.select_dtypes(exclude = ['object'])

numeric_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Age            300000 non-null  int64
 1   Diabetes       300000 non-null  int64
 2   Alcoolism      300000 non-null  int64
 3   HiperTension   300000 non-null  int64
 4   Handcap        300000 non-null  int64
 5   Smokes         300000 non-null  int64
 6   Scholarship    300000 non-null  int64
 7   Tuberculosis   300000 non-null  int64
 8   Sms_Reminder   300000 non-null  int64
 9   AwaitingTime   300000 non-null  int64
dtypes: int64(10)
memory usage: 22.9 MB
```

In [33]:

```
numeric_features = numeric_features.astype('int8')

numeric_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Age            300000 non-null  int8
 1   Diabetes       300000 non-null  int8
 2   Alcoolism      300000 non-null  int8
 3   HiperTension   300000 non-null  int8
 4   Handcap        300000 non-null  int8
 5   Smokes         300000 non-null  int8
 6   Scholarship    300000 non-null  int8
 7   Tuberculosis   300000 non-null  int8
 8   Sms_Reminder   300000 non-null  int8
 9   AwaitingTime   300000 non-null  int8
dtypes: int8(10)
memory usage: 2.9 MB
```

Typecasting to 'int8' has reduced the memory usage by **20 MB**.

Lets target the date columns now.

In [49]:

```
categorical_features = data.select_dtypes(include = ['object'])

categorical_features.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 5 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Gender                    300000 non-null  object
 1   AppointmentRegistration   300000 non-null  object
 2   ApointmentData            300000 non-null  object
 3   DayOfTheWeek              300000 non-null  object
 4   Status                    300000 non-null  object
dtypes: object(5)
memory usage: 97.3 MB
```

In [50]:

```
# Typecasting Data fields
categorical_features.AppointmentRegistration = pd.to_datetime(df.AppointmentRegistration

categorical_features.ApointmentData = pd.to_datetime(df.ApointmentData)
```

In [51]:
```
categorical_features.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 5 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Gender                   300000 non-null  object
 1   AppointmentRegistration  300000 non-null  datetime64[ns, UTC]
 2   ApointmentData           300000 non-null  datetime64[ns, UTC]
 3   DayOfTheWeek             300000 non-null  object
 4   Status                   300000 non-null  object
dtypes: datetime64[ns, UTC](2), object(3)
memory usage: 57.9 MB
```

In [52]:
```
# Typecasting to catergorical values
categorical_features.DayOfTheWeek = categorical_features.DayOfTheWeek.astype('category')
```

In [53]:
```
categorical_features.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 5 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Gender                   300000 non-null  object
 1   AppointmentRegistration  300000 non-null  datetime64[ns, UTC]
 2   ApointmentData           300000 non-null  datetime64[ns, UTC]
 3   DayOfTheWeek             300000 non-null  category
 4   Status                   300000 non-null  object
dtypes: category(1), datetime64[ns, UTC](2), object(2)
memory usage: 39.8 MB
```

In [56]:
```
# Concate all the fields into a Dataframe
data_compressed = pd.concat([categorical_features, numeric_features], axis=1)
```

In [57]:
```
# Memory consumption of the typecasted data
data_compressed.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 15 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Gender                   300000 non-null  object
 1   AppointmentRegistration  300000 non-null  datetime64[ns, UTC]
 2   ApointmentData           300000 non-null  datetime64[ns, UTC]
 3   DayOfTheWeek             300000 non-null  category
 4   Status                   300000 non-null  object
 5   Age                      300000 non-null  int8
 6   Diabetes                 300000 non-null  int8
 7   Alcoolism                300000 non-null  int8
 8   HiperTension             300000 non-null  int8
 9   Handcap                  300000 non-null  int8
 10  Smokes                   300000 non-null  int8
 11  Scholarship              300000 non-null  int8
 12  Tuberculosis             300000 non-null  int8
 13  Sms_Reminder             300000 non-null  int8
 14  AwaitingTime             300000 non-null  int8
dtypes: category(1), datetime64[ns, UTC](2), int8(10), object(2)
memory usage: 42.6 MB
```

- The typecasting technique can help reduce the memory usage to some extent.

- We have successfully shrinked the memory usage from **120.2 MB** to **42.6 MB**.
- There are certain limitations though with Pandas library especially if the dataset is very large when compared to the machines RAM capacity.
- Pandas as is not an idea library to handle large datasets. There are supportive packages that can help Pandas to scale up to deal with large datasets.
  - Dask
  - Modin
  - Vaex

In [ ]:
```

```