Data Wrangling

Steps involved

1. Importing necessary packages *:*

This allows us to use the underlying functions and methods and easily just by using the alias created.

For ex: Importing pandas package can be done as follows -

```
import pandas as pd
```

2. Importing data from a csv file in its raw form *:*

The first step involved was to import the raw data from file to a dataframe using pandas `read_csv()` utility.

```
sample_dataframe = pd.read_csv("../data/sample.csv")
```

```
flipkart_df=
pd.read_csv("../data/flipkart_com-ecommerce_sample.csv")
```

3. Examine the imported dataframe:

`.info()` - This gives a high level summary of the data that lies in the dataframe. For example - the type of data in each column, the number of non null entries, coerced data types, etc.

```
sample_dataframe.info()
```

```
flipkart_df.info()
```

`.head()` - examine the first few rows, to get the gist of type of values for each column. For example, null values can be identified at early stages.

```
sample_dataframe.head()
```

```
flipkart_df.head()
```

`.columns` - can get details on the column names and the no.of columns in the dataset. This was useful to clean up the column names - for example - If the column names have unnecessary spaces or can reassign new column names to be precise on what kind of info is depicted by that column.

```
sample_dataframe.columns
```

```
flipkart_df.columns
```

`.describe()` - this gives a high level summary statistics of all quantitative columns.

```
sample_dataframe.describe()
```

```
flipkart_df.describe()
```

`.value_counts()` - to identify frequency counts for categorical data

```
sample_dataframe.sample_categorical_column.value_counts()
```

```
flipkart_df.product_category_tree.value_counts()
```

`.unique()` - To get the unique values of a particular column. For instance, if a column has null values and we are not sure about how the null values are stored, we can identify it using this.

```
sample_dataframe.column_categorical.unique()
```

```
flipkart_df.product_name.unique()
```

4. Identifying and choosing the appropriate Index:

After examining the raw dataset using the above methods, if a particular column has a unique identifier for each row/case/observation/subject and describes an appropriate meaning to each observation, then this column can be conveniently used to set as index leveraging `pandas set_index()` method or can be performed while doing the `read_csv()` import with `index_col` attribute.

Reference: `flipkart_df Index - "uniq_id"`

```
# Setting the index using the index_col attribute of read_csv() utility
flipkart_df = pd.read_csv("../data/flipkart_com-ecommerce_sample.csv",index_col=0)

# Setting the index using set_index() dataframe method
```

```
# Using inplace as True, so it alters the dataframe, without having to create a new dataframe
flipkart_df.set_index('uniq_id',inplace=True)
```

After examining the dataframe in step 3, the most appropriate unique identifier for this dataset would be “`product identifier (pid)`” column, as this uniquely identifies each observation. So, we can re-set the index to “`pid`” column (*after removing any duplicates - explained in later part of this document*).

Reference: `flipkart_df new Index - "pid"`

```
# Setting the index using set_index() dataframe method
# Using inplace as True, so it alters the dataframe, without having to create a new dataframe
# This automatically drops any existing index
flipkart_df.set_index('pid',inplace=True)
```

5. Verify the correctness of data type for each column:

examine using the `info()` method.

(i). *Span the use of category type*:

If a column has categorical data and its datatype is 'object', this can be changed to 'category' type - as its good to use for memory efficiency. This can be done easily using `.astype('category')`.

Reference: `flipkart_df column: "product_category_tree"`

```
flipkart_df.product_category_tree = flipkart_df.product_category_tree.astype('category')
```

(ii). Deal with coerced types :

If a numeric column (for eg: price) is of dtype: object then it means - the data of that column needs to be examined further. This is often the case that some of the missing values may be set to some string(eg: 'missing') instead of NaN, as a result the data type is *coerced* to 'object' in other words, string. This can be handled by using `.to_numeric()` method on the column. But, before using this conversion "null" values need to be handled using any of the techniques above

Reference: `flipkart_df column: "overall_rating"`

```
# If errors= 'coerce' then invalid parsing will be set as NaN
flipkart_df.overall_rating
=pd.to_numeric(flipkart_df['overall_rating'],errors='coerce')
```

(iii). Handling datetime objects :

Date columns need to be imported as pandas datetime objects as it would make the dataset easier to access for further analysis of data. We can do this using pandas `to_datetime()` method or can be performed while doing the `read_csv()` import with `parse_dates` attribute.

Reference: `flipkart_df column : "crawl_timestamp"`

```
# Using the parse_dates attribute if read_csv() utility
pd.read_csv("../data/flipkart_com-ecommerce_sample.csv",index_col=0
,
                           parse_dates=['crawl_timestamp'])

# Using to_datetime()
flipkart_df.crawl_timestamp=pd.to_datetime(flipkart_df['crawl_times
tamp']).dt.strftime('%Y-%m-%d %H:%M:%S')
```

6. Handle "null" values :

These can often be represented as - empty strings, NaN's, NA, 'missing' or in any other format. Can be examined by using `head()`, `unique()`, `isnull()`, `values` etc,.

Following can be used to handle null/empty values :
1. `dropna()` - Delete records with null entries, not a great choice since most of the data is lost.
2. `fillna()` - Broadcast null entries with a particular value.
3. `Custom Python functions` - Impute using custom generated values
4. `{ }` - Broadcast using Dict comprehensions
5. `.map()` - Using other column data as a basis to fill in the null entries.

Reference: `flipkart_df column : "brand"`

```
flipkart_df['brand']=[str(k).split()[0] if v is np.NaN else k for
k,v in zip(flipkart_df['product_name'],flipkart_df['brand'])]
```

Example for using `.map()`

```
dataframe['name']= dataframe['code'].map({k: v for k, v in
zip(dataframe['code'], dataframe['name']) if v is not ''})
```

7. Handling duplicates :

Duplicates are removed using `drop_duplicates()` method. If dropping is based only on a particular column, that column name can be provided as a value to the subset attribute.
For example: In `flipkart_df` - "`pid`" column has duplicate entries (meaning has the same product details). In this case we can safely discard the duplicate entries retaining one entry and removing the duplicate entry.

Reference: `flipkart_df column : "pid"`

```
flipkart_df                                                        =
flipkart_df.drop_duplicates(subset='pid',keep='first')
```

8. Normalize embedded json values :

Some of the column values may contain embedded json as their which is required to normalize to make the dataset tidy and ready for analysis.

Reference: `flipkart_df column : product_specifications[0]`

```
json_data = {'product_specification': [{'key': 'Number of Contents
in Sales Package',
  'value': 'Pack of 3'},
 {'key': 'Fabric', 'value': 'Cotton Lycra'},
 {'key': 'Type', 'value': 'Cycling Shorts'},
```

```
 {'key': 'Pattern', 'value': 'Solid'},
 {'key': 'Ideal For', 'value': "Women's"},
 {'value': 'Gentle Machine Wash in Lukewarm Water, Do Not Bleach'},
 {'key': 'Style Code', 'value': 'ALTHT_3P_21'},
 {'value': '3 shorts'}]}

json_normalize(json_data)
```

9. Joining dataframes :

After retrieving the json data as columns, it needs to be joined back onto the original dataframe on which the analysis will be performed.

Reference: `flipkart_df and product_specifications join`

```
flipkart_df=flipkart_df.join(product_specification,how='left')
```

10. Reindex columns :

To reorder the columns in a dataframe we can use the `.reindex()` method of dataframe.

Reference: `flipkart_df`

```
flipkart_df = flipkart_df.reindex(columns=['crawl_timestamp',
'product_category','brand', 'product_name','retail_price',
'discounted_price','final_price', 'is_FK_Advantage_product',
'description','product_url','image', 'product_rating',
'product_color', 'product_ideal_for', 'product_occasion'])
```