Assignment 1

Data Preprocessing

UCS1729: Data Warehousing and Data Mining

Karthik D
195001047
CSE A

Analysis-Cleaning-Transformation

September 14, 2022

1 Analysis, Cleaning and Transformation of New York City Taxi Trip Data

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[2]:
```
import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

Load data and Show information

[3]:
```
DATA_PATH = "/content/drive/MyDrive/Undergrad/Semester-7/
↪DWDM_Preprocessing-Assignment/Data"
df = pd.read_csv(os.path.join(DATA_PATH, "taxi_trip_data.csv"))
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000000 entries, 0 to 9999999
Data columns (total 17 columns):
 #   Column              Dtype
---  ------              -----
 0   vendor_id           int64
 1   pickup_datetime     object
 2   dropoff_datetime    object
 3   passenger_count     int64
 4   trip_distance       float64
 5   rate_code           int64
 6   store_and_fwd_flag  object
 7   payment_type        int64
 8   fare_amount         float64
 9   extra               float64
```

```
 10  mta_tax              float64
 11  tip_amount           float64
 12  tolls_amount         float64
 13  imp_surcharge        float64
 14  total_amount         float64
 15  pickup_location_id   int64
 16  dropoff_location_id  int64
dtypes: float64(8), int64(6), object(3)
memory usage: 1.3+ GB
None
```

[4]:
```
# Show first 5 rows
print(df.head())
```

```
   vendor_id      pickup_datetime     dropoff_datetime  passenger_count  \
0          2  2018-03-29 13:37:13  2018-03-29 14:17:01                1
1          2  2018-03-29 13:37:18  2018-03-29 14:15:33                1
2          2  2018-03-29 13:26:57  2018-03-29 13:28:03                1
3          2  2018-03-29 13:07:48  2018-03-29 14:03:05                2
4          2  2018-03-29 14:19:11  2018-03-29 15:19:59                5

   trip_distance  rate_code store_and_fwd_flag  payment_type  fare_amount  \
0          18.15          3                  N             1         70.0
1           4.59          1                  N             1         25.0
2           0.30          1                  N             1          3.0
3          16.97          1                  N             1         49.5
4          14.45          1                  N             1         45.5

   extra  mta_tax  tip_amount  tolls_amount  imp_surcharge  total_amount  \
0    0.0      0.0       16.16         10.50            0.3         96.96
1    0.0      0.5        5.16          0.00            0.3         30.96
2    0.0      0.5        0.76          0.00            0.3          4.56
3    0.0      0.5        5.61          5.76            0.3         61.67
4    0.0      0.5       10.41          5.76            0.3         62.47

   pickup_location_id  dropoff_location_id
0                 161                    1
1                  13                  230
2                 231                  231
3                 231                  138
4                  87                  138
```

2 Introduction

The goal of this notebook is to clean and transform the data available for the purpose of later utilizing it in ML algorithms, or for data warehousing purposed.

The data cleaning process can begin to clear out outliers, missing values and other noise which might affect the results of the algorithm.

2.1 Background

Rides following similar paths in the past will likely take similar routes, and rides during the same hours of the day will also likely take roughly the same amount of time. This gives us a sort of rolling average for distance and time to make the calculation easier, what it doesn't give us, is how much of that distance is sitting in traffic, below 12mph, or driving at normal speeds above 12mph, nor does it account for sitting at red lights.

These values are hard to account for. While patterns can be detected when analysing the data through graphs and other visuals, it doesn't make for a very mathematical or repeatable prediction. We'll need the model to detect these patterns quickly and repeatably to get the most accurate predictions possible, which means some data, such as start and end times should be broken down into chunks that are easier for a machine to read such as the number of minutes per trip, the month, day, day of the week, and year (separately) .

2.2 Dataset: NYC Taxi Trip Data - Google Public Data

This data set is a subset of the Google BigQuery public datasets - Nyc yellow taxi cab trips data set containing a random 10,000,000 rows of data. This dataset was extracted and uploaded for the purpose of experimenting with and learning regression models for price prediction. There is also a lot of room for data cleaning, outliers in the data, and plenty of data to work with for more realistic model training, testing, and validation.

The data is publicly accessible at: https://www.kaggle.com/datasets/neilclack/nyc-taxi-trip-data-google-public-data

2.2.1 Data Attributes

column	type	description
vendor_id	text	A code indicating the TPEP provider that provided the record.
pickup_datetime	datetime	The date and time when the meter was engaged.
dropoff_datetime	datetime	The date and time when the meter was disengaged

column	type	description
passenger_count	integer	The number of passengers in the vehicle. This is a driver-entered value
trip_distance	numeric	The elapsed trip distance in miles reported by the taximeter
rate_code	string	The final rate code in effect at the end of the trip
storeandfwd_flag	string	Flag indicates iftrip record was held in vehicle memory before sending to vendor
payment_type	string	A numeric code signifying how the passenger paid for the trip
fare_amount	numeric	The time-and-distance fare calculated by the meter
extra	numeric	Miscellaneous extras and surcharges
mta_tax	numeric	$0.50 MTA tax that is automatically triggered based on the metered rate in use

column	type	description
tip_amount	numeric	Tip amount – Automatically populated for credit card tips
tolls_amount	numeric	Total amount of all tolls paid in the trip
imp_surcharge	numeric	$0.30 improvement surcharge assessed trips at the flag drop
total_amount	numeric	The total amount charged to passengers. Does not include cash tips
pickuplocationid	string	TLC Taxi Zone in which the taximeter was engaged
dropofflocationid	string	TLC Taxi Zone in which the taximeter was disengaged

2.3 Plan for Features

Here are the features of the current dataset that will be kept, as well as a few that will need to be created based on other features:
- pickup_timestamp
- dropoff_timestamp
- trip_distance
- fare_amount
- extra
- mta_tax
- imp_surcharge
- total_amount
- pickup_location_id

- dropoff_location_id

3 Data Analysis

The **correlation matrix** calculates how the change in one value effects a change in the other value, and assigns a value between -1 and 1 to that correlation.
Let's review what those correlation values mean before we move on:

3.0.1 Correlation Matrix

The correlation matrix for all pairs of attributes is represented in the heatmap below:

- **-1:** A very strong negative correlation, when value A moves in one direction, value B moves in the opposite direction.

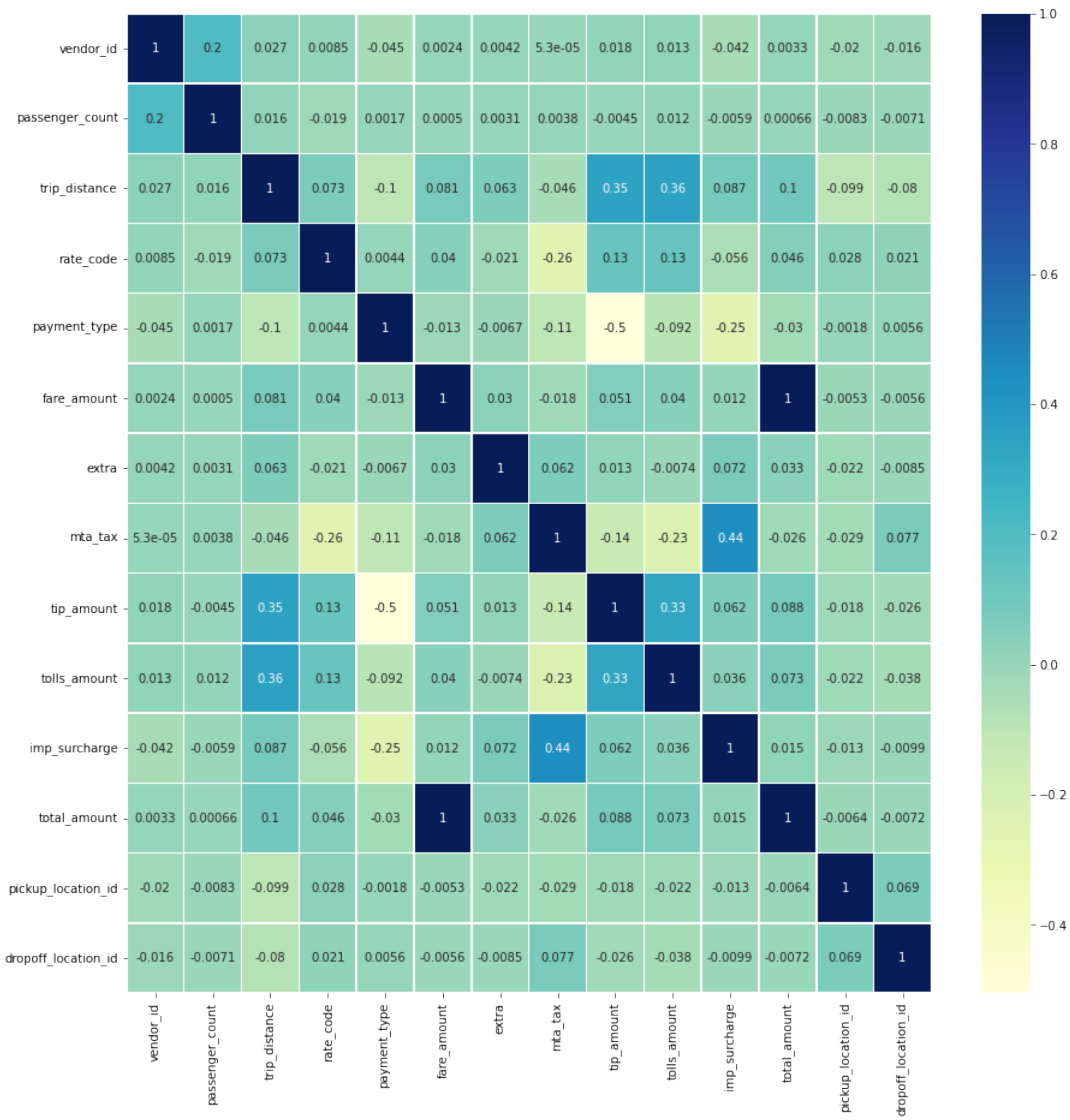
- **0:** No correlation between values A and B, when one moves, the other is not effected.

- **1:** A very strong positive correlation, as you can guess, this is the opposite of the negative correlation above. When value A moves in one direction, value B follows in the same direction.

This value isn't related to the rate of change, only the direction of change. Value A moves up, and value B either stays, moves up, or moves down.

[5]:
```
# Generating the correlation matrix
corr = df.corr()
```

[6]:
```
# Drawing the heatmap
fig, ax = plt.subplots(figsize=(15,15))
ax = sns.heatmap(corr, cmap='YlGnBu', annot=True, linewidths=0.5);
```



There is already a list of known values that we should keeep. The only remaining values are:
- **vendor_id** - Vendor of data provider. This definitely won't be used for anything for our model here
- **rate_code** - The rate code at the end of the trip. Used likely to track certain charges. Has a correlation with tolls and tips but not much with anything else.
- **sotre_and_fwd_flag** - This is simply a fag that indicates whether a value was stored in vehicle memory before being recorded due to a lack of internet connection. This is useless to us, however, it's currently stored as a string and converting it to a value that can appear in a correlation matrix later might serve useful. While not likely, it could be possible that values are different from those not stored in memory, such as having a higher amount of errors, or some upload process might be altering values in an unexpected way.

In the end, only one column is being dropped right off the start and that's **Vendor ID**.

[7]:
```
df = df.drop('vendor_id', axis=1)
df.head()
```

[7]:
```
       pickup_datetime     dropoff_datetime  passenger_count  trip_distance  \
0  2018-03-29 13:37:13  2018-03-29 14:17:01                1          18.15
1  2018-03-29 13:37:18  2018-03-29 14:15:33                1           4.59
2  2018-03-29 13:26:57  2018-03-29 13:28:03                1           0.30
3  2018-03-29 13:07:48  2018-03-29 14:03:05                2          16.97
4  2018-03-29 14:19:11  2018-03-29 15:19:59                5          14.45

   rate_code store_and_fwd_flag  payment_type  fare_amount  extra  mta_tax  \
0          3                  N             1         70.0    0.0      0.0
1          1                  N             1         25.0    0.0      0.5
2          1                  N             1          3.0    0.0      0.5
3          1                  N             1         49.5    0.0      0.5
4          1                  N             1         45.5    0.0      0.5

   tip_amount  tolls_amount  imp_surcharge  total_amount  pickup_location_id  \
0       16.16         10.50            0.3         96.96                 161
1        5.16          0.00            0.3         30.96                  13
2        0.76          0.00            0.3          4.56                 231
3        5.61          5.76            0.3         61.67                 231
4       10.41          5.76            0.3         62.47                  87

   dropoff_location_id
0                    1
1                  230
2                  231
3                  138
4                  138
```

4 Data Cleaning and Analysis

The following data cleaning steps are assessed and applied:

1. **Remove duplicate rows** - Carefully, as we only want to remove duplicate trips, not duplicates within the values themselves. These values are not required to be unique.
2. Check for **missing values**
3. Check for **zeros and empty strings**. These values won't be "missing" but still aren't valid. Very few columns in this data have valid zeros
4. **Validate formatting** of data, especially dates
5. **Strip and normalize strings** - our data doesn't contain any strings, so we can skip this.

4.1 Remove Duplicates

[8]:
```
# Remove duplicates -
# Rename the dataframe from df to td for temporary data, thus not altering the␣
↪original dataframe until much later.
td = df.drop_duplicates()
# less than 1% dropped
print(f"{df.shape[0] - td.shape[0]} duplicate rows dropped. Thats {(df.shape[0]␣
↪- td.shape[0]) / df.shape[0] * 100}%")
print(f"{td.shape[0]} rows remain.")
```

```
607571 duplicate rows dropped. Thats 6.07571%
9392429 rows remain.
```

4.2 Remove Missing Values

[9]:
```
# Checking for missing values
for col in td.columns:
    missing = td[col].isna().sum()
    print(f"Missing values in {col}: {missing}")
```

```
Missing values in pickup_datetime: 0
Missing values in dropoff_datetime: 0
Missing values in passenger_count: 0
Missing values in trip_distance: 0
Missing values in rate_code: 0
Missing values in store_and_fwd_flag: 0
Missing values in payment_type: 0
Missing values in fare_amount: 0
Missing values in extra: 0
Missing values in mta_tax: 0
Missing values in tip_amount: 0
Missing values in tolls_amount: 0
Missing values in imp_surcharge: 0
Missing values in total_amount: 0
Missing values in pickup_location_id: 0
Missing values in dropoff_location_id: 0
```

4.3 Remove Zeros and Empty Strings

[10]:
```
# Checking for zeros in numeric columns
def check_for_zeros(td):
    for col in td.columns:
        zeros = td[td[col] == 0].shape[0]
        print(f"Zeros in {col}:{zeros}")
```

```
check_for_zeros(td)
```

```
Zeros in pickup_datetime:0
Zeros in dropoff_datetime:0
Zeros in passenger_count:85779
Zeros in trip_distance:264896
Zeros in rate_code:0
Zeros in store_and_fwd_flag:0
Zeros in payment_type:0
Zeros in fare_amount:12176
Zeros in extra:5048008
Zeros in mta_tax:285657
Zeros in tip_amount:2062555
Zeros in tolls_amount:6253748
Zeros in imp_surcharge:12433
Zeros in total_amount:5725
Zeros in pickup_location_id:0
Zeros in dropoff_location_id:0
```

Changes applied so far ...

- *passenger_count*, *trip_distance*, *fare_amount* and *total_amount* --- all contain zeros.
- It doesn't appear to be a large amount of the overall data.
- Without distance, we can't determine fare amount, even with distance, it's impossible to know which miles were driven above the 12mph threshold, and which were below.
- There isn't much of a choice but to drop these. However, **total_amount** can be corrected by simply adding all of the charge column values together, so I'll keep and fix these rows.

Dropping rows with 0 values in columns where 0 is not allowed

[11]:
```
# Dropping rows with 0 values in columns where 0 is not allowed
td = td.drop(['passenger_count'], axis=1)
td = td[td['trip_distance'] > 0]
td = td[td['fare_amount'] > 0]

check_for_zeros(td)
```

```
Zeros in pickup_datetime:0
Zeros in dropoff_datetime:0
Zeros in trip_distance:0
Zeros in rate_code:0
Zeros in store_and_fwd_flag:0
Zeros in payment_type:0
Zeros in fare_amount:0
Zeros in extra:4836000
Zeros in mta_tax:216469
Zeros in tip_amount:1886004
```

```
Zeros in tolls_amount:5980138
Zeros in imp_surcharge:1418
Zeros in total_amount:0
Zeros in pickup_location_id:0
Zeros in dropoff_location_id:0
```

After dropping rows with zero values in other columns, there remains no zeros in total_amount, so no corrections are necessary here

[12]:
```
# Checking how much of the original data ramains
remaining = td.shape[0] / df.shape[0] * 100
print(f"Remaining amount of original dataset: {remaining}%")
```

```
Remaining amount of original dataset: 90.99450999999999%
```

4.4 Validating Data Formats

Ensure that the dates are all readable date formats and exist in the same format such as mm/dd/yyyy, for example.

[13]:
```
# Converting to an actual Python/Pandas datetime object ensures that the data␣
↪is a valid datetime.
# Then, we  move on to exploring the datetimes available.
td['pickup_datetime'] = pd.to_datetime(td['pickup_datetime'])
td['dropoff_datetime'] = pd.to_datetime(td['dropoff_datetime'])

print('Done.')
```

```
Done.
```

Inference: All datetime stamps in the dataset are correctly formatted

The datetime columns are now split up into meaninful columns. The only dropoff information we really need to keep is the hour, and even then, only to calculate the length of the trip.

[14]:
```
td['year'] = pd.to_datetime(td['pickup_datetime']).dt.year
td['month'] = pd.to_datetime(td['pickup_datetime']).dt.month
td['day'] = pd.to_datetime(td['pickup_datetime']).dt.day
td['day_of_week'] = pd.to_datetime(td['pickup_datetime']).dt.dayofweek
td['hour_of_day'] = pd.to_datetime(td['pickup_datetime']).dt.hour

print('Done.')
```

```
Done.
```

4.5 Cleaning Date and Time Data

4.5.1 Validate Timestamps

[15]:
```
# Converting the datetime columns to a numpy array for vectorization
pickup_array = td['pickup_datetime'].values
dropoff_array = td['dropoff_datetime'].values
```

4.5.2 Validate Trip Durations

[16]:
```
# Getting the new timedelta, this takes less than a second to complete compared␣
↪to 15+ minutes with apply()
trip_duration = np.subtract(dropoff_array, pickup_array)

# Adding the resulting array to the dataframe in the trip_duration column
td['trip_duration'] = pd.Series(trip_duration)

# Converting the timedelta to number of seconds
td['trip_duration'] = td['trip_duration'].dt.total_seconds()

# Preview the results
td.head()
```

[16]:
```
      pickup_datetime    dropoff_datetime  trip_distance  rate_code  \
0 2018-03-29 13:37:13 2018-03-29 14:17:01          18.15          3
1 2018-03-29 13:37:18 2018-03-29 14:15:33           4.59          1
2 2018-03-29 13:26:57 2018-03-29 13:28:03           0.30          1
3 2018-03-29 13:07:48 2018-03-29 14:03:05          16.97          1
4 2018-03-29 14:19:11 2018-03-29 15:19:59          14.45          1

  store_and_fwd_flag  payment_type  fare_amount  extra  mta_tax  tip_amount  \
0                  N             1         70.0    0.0      0.0       16.16
1                  N             1         25.0    0.0      0.5        5.16
2                  N             1          3.0    0.0      0.5        0.76
3                  N             1         49.5    0.0      0.5        5.61
4                  N             1         45.5    0.0      0.5       10.41

   …  imp_surcharge  total_amount  pickup_location_id  dropoff_location_id  \
0  …            0.3         96.96                 161                    1
1  …            0.3         30.96                  13                  230
2  …            0.3          4.56                 231                  231
3  …            0.3         61.67                 231                  138
4  …            0.3         62.47                  87                  138

   year  month  day  day_of_week  hour_of_day  trip_duration
```

```
0  2018      3   29          3          13        2388.0
1  2018      3   29          3          13        2295.0
2  2018      3   29          3          13          66.0
3  2018      3   29          3          13        3317.0
4  2018      3   29          3          14        3648.0

[5 rows x 21 columns]
```

Now, the datetime columns can be dropped entirely.

[17]:
```python
td.drop(['pickup_datetime', 'dropoff_datetime'], axis=1, inplace=True)
```

Displaying the dataset's current state ...

[18]:
```python
td.head()
```

[18]:
```
   trip_distance  rate_code store_and_fwd_flag  payment_type  fare_amount  \
0          18.15          3                  N             1         70.0
1           4.59          1                  N             1         25.0
2           0.30          1                  N             1          3.0
3          16.97          1                  N             1         49.5
4          14.45          1                  N             1         45.5

   extra  mta_tax  tip_amount  tolls_amount  imp_surcharge  total_amount  \
0    0.0      0.0       16.16         10.50            0.3         96.96
1    0.0      0.5        5.16          0.00            0.3         30.96
2    0.0      0.5        0.76          0.00            0.3          4.56
3    0.0      0.5        5.61          5.76            0.3         61.67
4    0.0      0.5       10.41          5.76            0.3         62.47

   pickup_location_id  dropoff_location_id  year  month  day  day_of_week  \
0                 161                    1  2018      3   29            3
1                  13                  230  2018      3   29            3
2                 231                  231  2018      3   29            3
3                 231                  138  2018      3   29            3
4                  87                  138  2018      3   29            3

   hour_of_day  trip_duration
0           13         2388.0
1           13         2295.0
2           13           66.0
3           13         3317.0
4           14         3648.0
```

Now that the dates have been broken down properly, a higher level of data clean-up can be performed.

- Any trips with a duration of 0 need to be dropped. These trips won't be useful, and are certainly due to a data entry error.

- Investigate what years are available in this dataset, how much of the dataset each year makes up, and begin investigating whether we should keep all years, or only specific years by visualizing trends in fare amounts when compared to trip duration and distance.

[19]:
```
td = td[td['trip_duration'] > 0]
```

[20]:
```
list_of_years = td.year.unique()
print(list_of_years)
```

```
[2018 2009 2017 2019 2008 2020 2003 2002 2001 2029 2032]
```

[21]:
```
for year in list_of_years:
    year_amount = td[td['year'] == year].shape[0]
    total_amount = td.shape[0]

    print(f"{year} makes up {(year_amount / total_amount) * 100}% of the␣
↪dataset")
```

```
2018 makes up 99.99841696039846% of the dataset
2009 makes up 0.0006356143854646263% of the dataset
2017 makes up 0.00020387631231884242% of the dataset
2019 makes up 0.00014391269104859462% of the dataset
2008 makes up 0.0004917016944160317% of the dataset
2020 makes up 1.1992724254049552e-05% of the dataset
2003 makes up 2.3985448508099104e-05% of the dataset
2002 makes up 2.3985448508099104e-05% of the dataset
2001 makes up 2.3985448508099104e-05% of the dataset
2029 makes up 1.1992724254049552e-05% of the dataset
2032 makes up 1.1992724254049552e-05% of the dataset
```

4.6 Eliminate Off-Trend Data

It's clear that this dataset is HEAVILY weighted towards 2018. For that reason, dropping anything from before 2018 can help avoid skewing the data towards old trends, while keeping anything newer than 2018 might reveal new trends.

If a dataset of such massive size consists of 99% of the same year, it's likely that the trips from newer years are either invalid data upon collection, and incomplete enough to actually show any trends.

All rows but 2018 are, therefore, dropped.

[22]:
```
td = td[td['year'] == 2018]
# Evaluate data stats after dropping
td.describe()
```

[22]:
```
       trip_distance     rate_code  payment_type   fare_amount         extra  \
count   8.338257e+06  8.338257e+06  8.338257e+06  8.338257e+06  8.338257e+06
```

```
mean    9.120187e+00  1.154223e+00  1.180907e+00  3.178215e+01  3.469645e-01
std     5.879868e+00  6.330880e-01  4.073165e-01  7.560952e+01  5.659283e-01
min     1.000000e-02  1.000000e+00  1.000000e+00  1.000000e-02 -8.000000e+01
25%     6.030000e+00  1.000000e+00  1.000000e+00  2.350000e+01  0.000000e+00
50%     8.600000e+00  1.000000e+00  1.000000e+00  2.900000e+01  0.000000e+00
75%     1.121000e+01  1.000000e+00  1.000000e+00  3.700000e+01  5.000000e-01
max     7.655760e+03  9.900000e+01  4.000000e+00  1.874365e+05  2.020000e+01

            mta_tax    tip_amount  tolls_amount  imp_surcharge  total_amount  \
count  8.338257e+06  8.338257e+06  8.338257e+06   8.338257e+06  8.338257e+06
mean   4.882261e-01  5.526390e+00  2.174295e+00   2.999538e-01  4.062672e+01
std    8.265593e-02  4.568232e+00  3.748963e+00   3.744167e-03  7.668925e+01
min    0.000000e+00  0.000000e+00 -5.760000e+00   0.000000e+00  3.100000e-01
25%    5.000000e-01  2.000000e+00  0.000000e+00   3.000000e-01  2.915000e+01
50%    5.000000e-01  5.550000e+00  0.000000e+00   3.000000e-01  3.755000e+01
75%    5.000000e-01  7.910000e+00  5.760000e+00   3.000000e-01  4.901000e+01
max    8.080000e+01  4.220000e+02  9.182500e+02   6.000000e-01  1.874378e+05

       pickup_location_id  dropoff_location_id       year         month  \
count        8.338257e+06         8.338257e+06  8338257.0  8.338257e+06
mean         1.528662e+02         1.476428e+02     2018.0  6.459984e+00
std          6.017347e+01         7.560037e+01        0.0  3.423810e+00
min          1.000000e+00         1.000000e+00     2018.0  1.000000e+00
25%          1.320000e+02         8.800000e+01     2018.0  3.000000e+00
50%          1.380000e+02         1.420000e+02     2018.0  6.000000e+00
75%          1.860000e+02         2.290000e+02     2018.0  1.000000e+01
max          2.650000e+02         2.650000e+02     2018.0  1.200000e+01

                day   day_of_week   hour_of_day  trip_duration
count  8.338257e+06  8.338257e+06  8.338257e+06   8.338257e+06
mean   1.576347e+01  2.950375e+00  1.380998e+01   2.210049e+03
std    8.640502e+00  1.930177e+00  6.231820e+00   4.865978e+03
min    1.000000e+00  0.000000e+00  0.000000e+00   1.000000e+00
25%    9.000000e+00  1.000000e+00  1.000000e+01   1.403000e+03
50%    1.600000e+01  3.000000e+00  1.400000e+01   1.835000e+03
75%    2.300000e+01  5.000000e+00  1.900000e+01   2.348000e+03
max    3.100000e+01  6.000000e+00  2.300000e+01   3.200310e+05
```

4.7 Trip Fare - Sanity Checks

The value of *total_amount* should be equal to the sum of the *fare_amount*, *mta_tax*, *tip_amount*, *tolls_amount*, *imp_surcharge* and the *extra*.

Calculating total amounts and dropping rows whose values don't "add up"...

4.7.1 Drop Fare Columns with Negative Values

[23]:
```
init_count = len(td)
td = td[td['fare_amount'] >= 0]
td = td[td['extra'] >= 0]
td = td[td['mta_tax'] >= 0]
td = td[td['tip_amount'] >= 0]
td = td[td['imp_surcharge'] >= 0]
td = td[td['tolls_amount'] >= 0]
final_count = len(td)

print(f"Fraction of dataframe retained: {final_count / init_count * 100}%")
```

```
Fraction of dataframe retained: 99.99989206377305%
```

4.7.2 Verify that the Fare Values Add Up

[24]:
```
# Calculating total amounts and dropping rows whose values don't "add up"...
fare = td['fare_amount'].values
extra = np.add(fare, td['extra'].values)
mta_tax = np.add(extra, td['mta_tax'].values)
tip_amount = np.add(mta_tax, td['tip_amount'].values)
imp_surcharge = np.add(tip_amount, td['imp_surcharge'].values)
calculated_total_amount = np.add(imp_surcharge, td['tolls_amount'].values)

td['calculated_total_amount'] = pd.Series(calculated_total_amount)

# validate calculated total by manually adding all relevant columns and␣
↪comparing to the calculated column
td.head(10)
```

[24]:

	trip_distance	rate_code	store_and_fwd_flag	payment_type	fare_amount	\
0	18.15	3	N	1	70.0	
1	4.59	1	N	1	25.0	
2	0.30	1	N	1	3.0	
3	16.97	1	N	1	49.5	
4	14.45	1	N	1	45.5	
5	11.60	1	N	1	42.0	
6	5.80	1	N	1	24.0	
7	3.38	1	N	1	25.0	
8	16.98	3	N	1	85.0	
9	4.99	1	N	1	22.0	

	extra	mta_tax	tip_amount	tolls_amount	imp_surcharge	total_amount	\
0	0.0	0.0	16.16	10.50	0.3	96.96	
1	0.0	0.5	5.16	0.00	0.3	30.96	

```
2    0.0      0.5        0.76          0.00            0.3          4.56
3    0.0      0.5        5.61          5.76            0.3         61.67
4    0.0      0.5       10.41          5.76            0.3         62.47
5    0.0      0.5       14.57          5.76            0.3         63.13
6    0.0      0.5        4.95          0.00            0.3         29.75
7    0.0      0.5        5.16          0.00            0.3         30.96
8    0.0      0.0       15.00         12.50            0.3        112.80
9    1.0      0.5        4.76          0.00            0.3         28.56

   pickup_location_id  dropoff_location_id  year  month  day  day_of_week  \
0                 161                    1  2018      3   29            3
1                  13                  230  2018      3   29            3
2                 231                  231  2018      3   29            3
3                 231                  138  2018      3   29            3
4                  87                  138  2018      3   29            3
5                  68                  138  2018      3   29            3
6                 100                   87  2018      3   29            3
7                 144                  161  2018      3   29            3
8                  87                    1  2018      3   29            3
9                  13                  161  2018      3   29            3

   hour_of_day  trip_duration  calculated_total_amount
0           13         2388.0                    96.96
1           13         2295.0                    30.96
2           13           66.0                     4.56
3           13         3317.0                    61.67
4           14         3648.0                    62.47
5           14         3540.0                    63.13
6           14         1608.0                    29.75
7           15         2554.0                    30.96
8           15         5267.0                   112.80
9           16         1810.0                    28.56
```

Dropping incorrect `total_amount` values

[25]:
```python
# Dropping incorrect `total_amount` values
init_count = len(td)
td = td[td['total_amount'] != td['calculated_total_amount']]
final_count = len(td)

print(f"Fraction of dataframe retained: {final_count / init_count * 100}%")

# Drop the computed fare value
td.drop('calculated_total_amount', axis=1, inplace=True)

td.describe()
```

```
Fraction of dataframe retained: 99.7802655905653%
```

[25]:
```
       trip_distance     rate_code  payment_type   fare_amount         extra  \
count   8.319926e+06  8.319926e+06  8.319926e+06  8.319926e+06  8.319926e+06
mean    9.126148e+00  1.154471e+00  1.180647e+00  3.179688e+01  3.470340e-01
std     5.882454e+00  6.336688e-01  4.070884e-01  7.558217e+01  5.652676e-01
min     1.000000e-02  1.000000e+00  1.000000e+00  1.000000e-02  0.000000e+00
25%     6.040000e+00  1.000000e+00  1.000000e+00  2.350000e+01  0.000000e+00
50%     8.600000e+00  1.000000e+00  1.000000e+00  2.900000e+01  0.000000e+00
75%     1.122000e+01  1.000000e+00  1.000000e+00  3.700000e+01  5.000000e-01
max     7.655760e+03  9.900000e+01  4.000000e+00  1.874365e+05  2.020000e+01

            mta_tax    tip_amount  tolls_amount  imp_surcharge  total_amount  \
count  8.319926e+06  8.319926e+06  8.319926e+06   8.319926e+06  8.319926e+06
mean   4.881855e-01  5.530809e+00  2.178277e+00   2.999539e-01  4.064989e+01
std    7.630298e-02  4.570137e+00  3.751520e+00   3.741069e-03  7.666306e+01
min    0.000000e+00  0.000000e+00  0.000000e+00   0.000000e+00  3.100000e-01
25%    5.000000e-01  2.000000e+00  0.000000e+00   3.000000e-01  2.915000e+01
50%    5.000000e-01  5.550000e+00  0.000000e+00   3.000000e-01  3.755000e+01
75%    5.000000e-01  7.910000e+00  5.760000e+00   3.000000e-01  4.906000e+01
max    2.150000e+01  4.220000e+02  9.182500e+02   6.000000e-01  1.874378e+05

       pickup_location_id  dropoff_location_id       year         month  \
count        8.319926e+06         8.319926e+06  8319926.0  8.319926e+06
mean         1.528594e+02         1.476394e+02     2018.0  6.460077e+00
std          6.015449e+01         7.559550e+01        0.0  3.423744e+00
min          1.000000e+00         1.000000e+00     2018.0  1.000000e+00
25%          1.320000e+02         8.800000e+01     2018.0  3.000000e+00
50%          1.380000e+02         1.420000e+02     2018.0  6.000000e+00
75%          1.860000e+02         2.290000e+02     2018.0  1.000000e+01
max          2.650000e+02         2.650000e+02     2018.0  1.200000e+01

                day   day_of_week   hour_of_day  trip_duration
count  8.319926e+06  8.319926e+06  8.319926e+06   8.319926e+06
mean   1.576325e+01  2.950070e+00  1.381030e+01   2.209896e+03
std    8.640600e+00  1.930190e+00  6.231147e+00   4.865080e+03
min    1.000000e+00  0.000000e+00  0.000000e+00   1.000000e+00
25%    9.000000e+00  1.000000e+00  1.000000e+01   1.403000e+03
50%    1.600000e+01  3.000000e+00  1.400000e+01   1.835000e+03
75%    2.300000e+01  5.000000e+00  1.900000e+01   2.348000e+03
max    3.100000e+01  6.000000e+00  2.300000e+01   3.200310e+05
```

4.8 Dataframe After Cleanup

[26]:
```python
# Display sample rows from the dataset
td.head()
```

```
[26]:     trip_distance  rate_code store_and_fwd_flag  payment_type  fare_amount  \
     3            16.97          1                  N             1         49.5
     4            14.45          1                  N             1         45.5
     5            11.60          1                  N             1         42.0
     10            5.10          1                  N             1         26.5
     12           11.11          1                  N             1         45.5

         extra  mta_tax  tip_amount  tolls_amount  imp_surcharge  total_amount  \
     3     0.0      0.5        5.61          5.76            0.3         61.67
     4     0.0      0.5       10.41          5.76            0.3         62.47
     5     0.0      0.5       14.57          5.76            0.3         63.13
     10    1.0      0.5        5.65          0.00            0.3         33.95
     12    1.0      0.5       10.61          5.76            0.3         63.67

         pickup_location_id  dropoff_location_id  year  month  day  day_of_week  \
     3                  231                  138  2018      3   29            3
     4                   87                  138  2018      3   29            3
     5                   68                  138  2018      3   29            3
     10                 186                   33  2018      3   29            3
     12                 163                  138  2018      3   29            3

         hour_of_day  trip_duration
     3            13         3317.0
     4            14         3648.0
     5            14         3540.0
     10           16         2585.0
     12           16         4521.0
```

5 Data Transformation and Analysis

The following data transformation steps are applied to the cleansed data, to facilitate further data analysis and mining.

5.1 Construct Composite/Derived Attributes

Construction of composite and derived attributes can greatly aid the the learning and analysis phase of data mining. Simplex relationships across the data attributes that may not be captured by the downstream analysis models, can be expressed through explicity computed attributes through a combination of one or more pre-existing attributes.

Compute and Add Driving Speed A `driving_speed` attribute in addition to the existing data attributes can be useful to analyze traffic data in different geographic locations of the city. Average speed can be directly correlated with the traffic density.

[27]:
```
trip_distance_array = td['trip_distance']
trip_duration_array = td['trip_duration']

# driving_speed = trip_dist (in miles) / trip_duration (in sec) * 3600 sec
driving_speed = np.divide(trip_distance_array, trip_duration_array)*3600

td['driving_speed'] = pd.Series(trip_duration)
```

Compute and Add Tipping Rate `tipping_rate` can help to analyze the general proportion of tipping that cab riders usually pay for their rides.

[28]:
```
tip_amount_array = td['tip_amount']
total_amount_array = td['total_amount']

# tipping rate = tip_amount / total_amount
tipping_rate = np.divide(tip_amount_array, total_amount_array)

td['tipping_rate'] = tipping_rate
```

5.2 Data Normalization

Data Normalization is a typical practice in data mining technique which consists of transforming numeric columns to a standard scale. Since some feature values differ from others multiple times, the features with higher values will dominate the learning and analysis process. Hence, bringing them down to the same scale is useful for faster and meaningful analysis.

Simple `min-max scaling` procedure is adopted in the following cells to normalize specific attributes. Since statistical information about the mean, variance, etc. is not known, more complex and informed scaling procedures cannot be applied.

5.2.1 All Fare Attributes

All cost related columns have very distinct scales. `tip_amount`, for instance, is exteremely low while the `total_fare` is typically a very high value. Hence, the latter would dominate the analysis and model training (for ML) processes. These are normalized here,

[29]:
```
columns = [
    'fare_amount',
    'extra',
    'mta_tax',
    'tip_amount',
    'tolls_amount',
    'total_amount'
]
for column in columns:
```

```
  td[column] = (td[column] - td[column].min()) / (td[column].max() - td[column].
↪min())

td.head()
```

[29]:

```
    trip_distance  rate_code store_and_fwd_flag  payment_type  fare_amount  \
3           16.97          1                  N             1     0.000264
4           14.45          1                  N             1     0.000243
5           11.60          1                  N             1     0.000224
10           5.10          1                  N             1     0.000141
12          11.11          1                  N             1     0.000243

       extra   mta_tax  tip_amount  tolls_amount  imp_surcharge  …  \
3   0.000000  0.023256    0.013294      0.006273            0.3  …
4   0.000000  0.023256    0.024668      0.006273            0.3  …
5   0.000000  0.023256    0.034526      0.006273            0.3  …
10  0.049505  0.023256    0.013389      0.000000            0.3  …
12  0.049505  0.023256    0.025142      0.006273            0.3  …

    pickup_location_id  dropoff_location_id  year  month  day  day_of_week  \
3                  231                  138  2018      3   29            3
4                   87                  138  2018      3   29            3
5                   68                  138  2018      3   29            3
10                 186                   33  2018      3   29            3
12                 163                  138  2018      3   29            3

    hour_of_day  trip_duration   driving_speed tipping_rate
3            13         3317.0 0 days 00:55:17     0.090968
4            14         3648.0 0 days 01:00:48     0.166640
5            14         3540.0 0 days 00:59:00     0.230794
10           16         2585.0 0 days 00:43:05     0.166421
12           16         4521.0 0 days 01:15:21     0.166640

[5 rows x 21 columns]
```

5.3 Numeric Encoding for Categorical String Attributes

Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the models to give and improve the predictions.

5.3.1 Encode `store_and_fwd_flag` Attribute

[30]:
```python
# Numeric encoding for categorical variable
td.store_and_fwd_flag = td.store_and_fwd_flag.astype('category').cat.codes
td.head()
```

[30]:
```
    trip_distance  rate_code  store_and_fwd_flag  payment_type  fare_amount  \
3           16.97          1                   0             1     0.000264
4           14.45          1                   0             1     0.000243
5           11.60          1                   0             1     0.000224
10           5.10          1                   0             1     0.000141
12          11.11          1                   0             1     0.000243

       extra   mta_tax  tip_amount  tolls_amount  imp_surcharge  …  \
3   0.000000  0.023256    0.013294      0.006273            0.3  …
4   0.000000  0.023256    0.024668      0.006273            0.3  …
5   0.000000  0.023256    0.034526      0.006273            0.3  …
10  0.049505  0.023256    0.013389      0.000000            0.3  …
12  0.049505  0.023256    0.025142      0.006273            0.3  …

    pickup_location_id  dropoff_location_id  year  month  day  day_of_week  \
3                  231                  138  2018      3   29            3
4                   87                  138  2018      3   29            3
5                   68                  138  2018      3   29            3
10                 186                   33  2018      3   29            3
12                 163                  138  2018      3   29            3

    hour_of_day  trip_duration   driving_speed tipping_rate
3            13         3317.0 0 days 00:55:17     0.090968
4            14         3648.0 0 days 01:00:48     0.166640
5            14         3540.0 0 days 00:59:00     0.230794
10           16         2585.0 0 days 00:43:05     0.166421
12           16         4521.0 0 days 01:15:21     0.166640

[5 rows x 21 columns]
```

5.4 Discretization

Some of the data attributes represented using continuos ranges can be discretized to simplify the data. They can be used for a classification or categorical analysis, as opposed to a regression analysis.

In most scenarios, this leads to easier, simpler and computationally cheaper analysis. Furthermore, it can be modeled using simpler functions since complex analog variations are discretized. The analysis inferences from both analyses are usually similar in these scenarios.

5.4.1 Bin the trip_distance Attribute

The value of `trip_distance` does not have to be accurate to the level of 2 decimal places to analyze its impact on the travel time in minutes. The value can be binned to every **two-mile interval** to simplify the analysis, whilst preserving the meanifulness of the analysis inferences.

[31]:
```
bins = [ x for x in range(int(td.trip_distance.min()), int(td.trip_distance.
 ↪max())+2, 2) ]
if bins[0]!=0:
  bins = [0] + bins

labels = [ x for x in bins[1:] ]

td['trip_distance_binned'] = pd.cut(td['trip_distance'], bins=bins,␣
 ↪labels=labels)
td.describe()
```

[31]:

	trip_distance	rate_code	store_and_fwd_flag	payment_type
count	8.319926e+06	8.319926e+06	8.319926e+06	8.319926e+06
mean	9.126148e+00	1.154471e+00	4.863986e-03	1.180647e+00
std	5.882454e+00	6.336688e-01	6.957246e-02	4.070884e-01
min	1.000000e-02	1.000000e+00	0.000000e+00	1.000000e+00
25%	6.040000e+00	1.000000e+00	0.000000e+00	1.000000e+00
50%	8.600000e+00	1.000000e+00	0.000000e+00	1.000000e+00
75%	1.122000e+01	1.000000e+00	0.000000e+00	1.000000e+00
max	7.655760e+03	9.900000e+01	1.000000e+00	4.000000e+00

	fare_amount	extra	mta_tax	tip_amount	tolls_amount
count	8.319926e+06	8.319926e+06	8.319926e+06	8.319926e+06	8.319926e+06
mean	1.695875e-04	1.717990e-02	2.270630e-02	1.310618e-02	2.372205e-03
std	4.032416e-04	2.798354e-02	3.548976e-03	1.082971e-02	4.085510e-03
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.253225e-04	0.000000e+00	2.325581e-02	4.739336e-03	0.000000e+00
50%	1.546658e-04	0.000000e+00	2.325581e-02	1.315166e-02	0.000000e+00
75%	1.973469e-04	2.475248e-02	2.325581e-02	1.874408e-02	6.272802e-03
max	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00

	imp_surcharge	…	pickup_location_id	dropoff_location_id	year
count	8.319926e+06	…	8.319926e+06	8.319926e+06	8319926.0
mean	2.999539e-01	…	1.528594e+02	1.476394e+02	2018.0
std	3.741069e-03	…	6.015449e+01	7.559550e+01	0.0
min	0.000000e+00	…	1.000000e+00	1.000000e+00	2018.0
25%	3.000000e-01	…	1.320000e+02	8.800000e+01	2018.0
50%	3.000000e-01	…	1.380000e+02	1.420000e+02	2018.0
75%	3.000000e-01	…	1.860000e+02	2.290000e+02	2018.0
max	6.000000e-01	…	2.650000e+02	2.650000e+02	2018.0

```
              month           day   day_of_week   hour_of_day  trip_duration  \
count  8.319926e+06  8.319926e+06  8.319926e+06  8.319926e+06   8.319926e+06
mean   6.460077e+00  1.576325e+01  2.950070e+00  1.381030e+01   2.209896e+03
std    3.423744e+00  8.640600e+00  1.930190e+00  6.231147e+00   4.865080e+03
min    1.000000e+00  1.000000e+00  0.000000e+00  0.000000e+00   1.000000e+00
25%    3.000000e+00  9.000000e+00  1.000000e+00  1.000000e+01   1.403000e+03
50%    6.000000e+00  1.600000e+01  3.000000e+00  1.400000e+01   1.835000e+03
75%    1.000000e+01  2.300000e+01  5.000000e+00  1.900000e+01   2.348000e+03
max    1.200000e+01  3.100000e+01  6.000000e+00  2.300000e+01   3.200310e+05

                   driving_speed  tipping_rate
count                    8319926  8.319926e+06
mean   0 days 00:36:49.895845828  1.296826e-01
std    0 days 01:21:05.079831993  7.636368e-02
min              0 days 00:00:01  0.000000e+00
25%              0 days 00:23:23  9.090909e-02
50%              0 days 00:30:35  1.664671e-01
75%              0 days 00:39:08  1.666667e-01
max              3 days 16:53:51  9.985929e-01

[8 rows x 21 columns]
```

5.5 Dataframe After Transformations

[32]:
```python
# Display sample rows from the dataset
td.head()
```

[32]:
```
    trip_distance  rate_code  store_and_fwd_flag  payment_type  fare_amount  \
3           16.97          1                   0             1     0.000264
4           14.45          1                   0             1     0.000243
5           11.60          1                   0             1     0.000224
10           5.10          1                   0             1     0.000141
12          11.11          1                   0             1     0.000243

       extra   mta_tax  tip_amount  tolls_amount  imp_surcharge  …  \
3   0.000000  0.023256    0.013294      0.006273            0.3  …
4   0.000000  0.023256    0.024668      0.006273            0.3  …
5   0.000000  0.023256    0.034526      0.006273            0.3  …
10  0.049505  0.023256    0.013389      0.000000            0.3  …
12  0.049505  0.023256    0.025142      0.006273            0.3  …

    dropoff_location_id  year  month  day  day_of_week  hour_of_day  \
3                   138  2018      3   29            3           13
4                   138  2018      3   29            3           14
5                   138  2018      3   29            3           14
10                   33  2018      3   29            3           16
```

```
12                   138  2018      3   29            3            16

    trip_duration   driving_speed  tipping_rate trip_distance_binned
3          3317.0 0 days 00:55:17      0.090968                   18
4          3648.0 0 days 01:00:48      0.166640                   16
5          3540.0 0 days 00:59:00      0.230794                   12
10         2585.0 0 days 00:43:05      0.166421                    6
12         4521.0 0 days 01:15:21      0.166640                   12

[5 rows x 22 columns]
```

6 Finishing Up

The *total_amount* column did a lot more than just clean totals, but it actually checked all of the other total effecting columns at the same time. If any errors occurred in any column, the calculated total would have differed from the calculated total.

Missing *mta_tax*, and incorrect *toll_amount* values are dropped.

[33]:
```
# this is a quick, easy way to de-allocate the memory assigned to df, which␣
↪holds the original dataframe
# this was necessary else the write to csv function of Pandas (to_csv) would␣
↪max out the allowed memory in the notebook environment on Kaggle.
df=[]
```

Save the cleaned dataframe to a CSV.

[34]:
```
td.to_csv(os.path.join(DATA_PATH, 'taxi-trip-data_2018_cleaned.csv'))
print('Done!')
```

```
Done!
```

7 Save the notebook

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

[42]:
```
os.chdir("/content/drive/MyDrive/Undergrad/Semester-7/
↪DWDM_Preprocessing-Assignment")
!ls
!jupyter nbconvert --to PDF "Analysis-Cleaning-Transformation.ipynb"
```

```
1_Analysis.ipynb  Analysis-Cleaning-Transformation.ipynb  Data
2_Cleaning.ipynb  Analysis-Cleaning-Transformation.pdf
[NbConvertApp] Converting notebook Analysis-Cleaning-Transformation.ipynb to PDF
```

```
[NbConvertApp] Support files will be in Analysis-Cleaning-Transformation_files/
[NbConvertApp] Making directory ./Analysis-Cleaning-Transformation_files
[NbConvertApp] Writing 104946 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 216909 bytes to Analysis-Cleaning-Transformation.pdf
```