LESSON

Data Cleaning

Lesson Goals

- Examine data for potential issues.
- Identify and fill in missing values.
- Identify and correct incorrect values.
- Remove low variance columns.
- Identify potential outliers.
- Correct incorrect data types.
- Remove special characters and clean categorical variables.
- Identify and remove duplicate records.

Introduction

When working with data sets, you will find that they often require a bit of cleaning. Whether Pandas originally read the data types incorrectly, records are duplicated, the data contains special characters or missing value, or there are slightly different references to the same entity, every data analyst must know how to clean the data they are working with before analyzing it. In this lesson, you will learn about some of the most common problems that make data messy and methods for correcting those problems and cleaning your data.

The data set we are going to be using for this lesson is a messy version of the vehicles data set we worked with in the previous lesson. Let's import this version of our data set so that we can then practice cleaning it up.

```
1   data = pd.read_csv('data sets/vehicles
    /vehicles_messy.csv')
```

Copy

One of the first things we want to do is examine the data and look for any potential issues. Some of the things we are interested in identifying in the data at this stage include:

- Missing values
- Special characters
- Incorrect values
- Extreme values or outliers
- Duplicate records
- Incorrect data types

The presence of these may cause problems when it's time to analyze the data, so we want to make sure we address them beforehand. We can start by visually inspecting the data using the `head` method, which will show us the first 5 rows of data.

```
1   data.head()
```

Copy

	barrels08	barrelsA08	charge120	charge240	city08	city08U	cityA08	cityA08U	cityCD	cityE	...	mfrCode	c240Dscr	charge240b	c240bDscr	createdOn
0	15.696	0.000	0.000	0.000	19	0.000	0	0.000	0.000	0.000	...	NaN	NaN	0.000	NaN	Tue Jan 01 00:00:00 EST 2013
1	29.965	0.000	0.000	0.000	9	0.000	0	0.000	0.000	0.000	...	NaN	NaN	0.000	NaN	Tue Jan 01 00:00:00 EST 2013
2	12.208	0.000	0.000	0.000	23	0.000	0	0.000	0.000	0.000	...	NaN	NaN	0.000	NaN	Tue Jan 01 00:00:00 EST 2013
3	29.965	0.000	0.000	0.000	10	0.000	0	0.000	0.000	0.000	...	NaN	NaN	0.000	NaN	Tue Jan 01 00:00:00 EST 2013
4	17.348	0.000	0.000	0.000	17	0.000	0	0.000	0.000	0.000	...	NaN	NaN	0.000	NaN	Tue Jan 01 00:00:00 EST 2013

Missing Values

From this initial view, we can see that our data set contains some columns that have missing values in them and others that seem to have a lot of zero values. Let's see how prevalent missing values are in our data. We can use the Pandas `isnull` method to check whether the value in each field is missing (null) and return either True or False for each field. We can use the `sum` method to total up the number of True values by column, and then we can add a condition using square brackets that will filter the data and show

← Go back Copy URL for Students Edit in Studio

```
1   null_cols = data.isnull().sum()                                    Copy
2   null_cols[null_cols > 0]
3
4   cylinders       123
5   displ           120
6   drive          1189
7   eng_dscr      15403
8   trany            11
9   guzzler       35562
10  trans_dscr    22796
11  tCharger      32657
12  sCharger      37177
13  atvType       34771
14  fuelType2     36435
15  rangeA        36440
16  evMotor       37281
17  mfrCode       30818
18  c240Dscr      37806
19  c240bDscr     37807
20  startStop     31705
21  dtype: int64
```

We can see that some columns have relatively few null values while others have tens of thousands of nulls. For fields that have a lot of null values, you will often have to make a judgement call. If you don't think the information is going to be very useful to your analysis, then you would remove those columns from your data frame. In Pandas, we can do that using the `drop` method. For our purposes, let's remove the columns that have more than 10,000 null values in them. We will add these column names to a list, and then we will pass those columns to the `drop` method and indicate that we want columns (not rows) dropped by setting the axis parameter to 1.

```
1   drop_cols = list(null_cols[null_cols > 10000].index)              Copy
2   data = data.drop(drop_cols, axis=1)
```

This leaves us with just a handful of remaining columns that have null values. Of the columns that remain, it looks like the cylinders column and the *displ* column have a

```
null_displ = data[(data['displ'].isnull()==True)]
null_displ = null_displ[['year', 'make', 'model',
'trany', 'drive','fuelType','cylinders', 'displ']]
null_displ
```

Copy

	year	make	model	trany	drive	fuelType	cylinders	displ
7138	2000	Nissan	Altra EV	NaN	NaN	Electricity	nan	nan
7139	2000	Toyota	RAV4 EV	NaN	2-Wheel Drive	Electricity	nan	nan
8143	2001	Toyota	RAV4 EV	NaN	2-Wheel Drive	Electricity	nan	nan
8144	2001	Ford	Th!nk	NaN	NaN	Electricity	nan	nan
8146	2001	Ford	Explorer USPS Electric	NaN	2-Wheel Drive	Electricity	nan	nan
8147	2001	Nissan	Hyper-Mini	NaN	NaN	Electricity	nan	nan
9212	2002	Toyota	RAV4 EV	NaN	2-Wheel Drive	Electricity	nan	nan
9213	2002	Ford	Explorer USPS Electric	NaN	2-Wheel Drive	Electricity	nan	nan
10329	2003	Toyota	RAV4 EV	NaN	2-Wheel Drive	Electricity	nan	nan
21413	1985	Subaru	RX Turbo	Manual 5-spd	4-Wheel Drive	Regular	nan	nan
21414	1985	Subaru	RX Turbo	Manual 5-spd	4-Wheel Drive	Regular	nan	nan

We can see that most of the time, cylinders is null when displ is null and that the most of the records where both fields are null have a fuel type of Electricity. This makes sense, as electric cars do not have cylinders and can therefore not have any displacement. In this case, it would make sense to replace these null values with zeros. Pandas makes it easy to do that with the `fillna` method.

```
data[['displ', 'cylinders']] = data[['displ',
'cylinders']].fillna(0)
```

Copy

In this example, we filled the nulls in with zeros, but there are other strategies for filling in nulls. Depending on the circumstances, you might want to replace nulls with the column mean or mode values. Once you get more advanced, you can even use a variety of predictive imputation methods.

Challenge: Now that we have filled those null values in with zeros, there are only two columns in the data set that still have null values: trany and drive. Use what you have learned in this section to investigate and potentially fill in the remaining null values.

In addition to null values, we also want to try to identify any values that seem incorrect. For example, in the previous section, we learned that a vehicle without cylinders should not have displacement and vice versa. Let's check to see if there are any cases that violate these rules.

```
test = data[(data['cylinders']==0) &
(data['displ']!=0)]
test[['year', 'make', 'model', 'trany',
'drive','fuelType','cylinders', 'displ']]
```

Copy

	year	make	model	trany	drive	fuelType	cylinders	displ
21506	1986	Mazda	RX-7	Manual 5-spd	Rear-Wheel Drive	Regular	0.000	1.300

Here we have identified a vehicle with a regular gasoline engine that reportedly does not have any cylinders but does have a value for displacement. The way we would correct this would be to either perform some domain research or ask a domain expert to find out how many actual cylinders this vehicle had. Alternatively, you can also try to look at similar vehicles in the data set and determine the most likely value for this field.

Suppose that using one of the aforementioned methods, we found out that this vehicle actually has a 4 cylinder engine. Once we have this information, we can use the `loc` method to update that specific value in the data frame.

```
data.loc[(data['cylinders']==0) & (data['displ']!=0),
'cylinders'] = 4
```

Copy

Challenge: Try to find other values that might be incorrect in the data set based on what you know about automobiles and correct them.

Low Variance Columns

When analyzing data, we want the fields we are working with to be informative, and we

the potential to not be as informative as columns that have a variety of different values in them.

Let's try to identify columns where at least 90% of the values are the same so that we can remove them from our data set. To do this, we are going to create an empty list called low_variance that will eventually contain the names of columns that fit our criteria. We will then write a for loop that will take the minimum and the 90th percentile value for all the numeric columns in our data set (identified via the `_get_numeric_data` method). If the 90th percentile and the minimum are equal to each other, that means that at least 90% of the values in that column are the same and we will append that column name to our low_variance list.

```
low_variance = []

for col in data._get_numeric_data():
    minimum = min(data[col])
    ninety_perc = np.percentile(data[col], 90)
    if ninety_perc == minimum:
        low_variance.append(col)

print(low_variance)

['barrelsA08', 'charge120', 'charge240', 'cityA08', 'cityA08U',
'cityCD', 'cityE', 'cityUF', 'co2A', 'co2TailpipeAGpm',
'combA08', 'combA08U', 'combE', 'combinedCD', 'combinedUF',
'fuelCostA08', 'ghgScoreA', 'highwayA08', 'highwayA08U',
'highwayCD', 'highwayE', 'highwayUF', 'phevBlended', 'range',
'rangeCity', 'rangeCityA', 'rangeHwy', 'rangeHwyA', 'UCityA',
'UHighwayA', 'charge240b', 'phevCity', 'phevHwy', 'phevComb']
```

Copy

This returned 34 columns that we could potentially eliminate due to not having high enough variability to be informative. Of course, before we do this, we should check the values that do exist in these fields to confirm that they are not very informative. Once they have been checked, we can use the the `drop` method like we did earlier in this lesson to remove those columns from our data frame.

Copy

Extreme Values and Outliers

Now that we have removed low variance columns, we should look for outliers, or extreme values, in the columns that remain. These outliers can influence our aggregations when we are analyzing data later, so we want to make sure we address them during our data cleaning stage.

A common method for identifying outliers is one that leverages the interquartile range (IQR). Once the IQR is calculated, it is multiplied by a constant (typically 1.5) and lower and upper bounds are established at:

- 25th Percentile - (IQR x 1.5)
- 75th Percentile + (IQR x 1.5)

Any values outside this range are potential outliers and should be investigated.

Let's look at how we would do this for our data set using Python. We will use the Pandas `describe` function to easily calculate the 25th and 75th percentiles for every column and transpose the results so that we can easily reference the values in calculating the interquartile ranges.

```
1   stats = data.describe().transpose()
2   stats['IQR'] = stats['75%'] - stats['25%']
3   stats
```

Copy

	count	mean	std	min	25%	50%	75%	max	IQR
barrels08	37843.000	17.533	4.576	0.060	14.331	17.348	20.601	47.087	6.270
city08	37843.000	17.941	6.660	6.000	15.000	17.000	20.000	138.000	5.000
co2TailpipeGpm	37843.000	473.180	122.189	0.000	388.000	467.737	555.438	1269.571	167.438
comb08	37843.000	20.196	6.623	7.000	17.000	19.000	23.000	124.000	6.000
cylinders	37843.000	5.719	1.779	0.000	4.000	6.000	6.000	16.000	2.000
displ	37843.000	3.308	1.372	0.000	2.200	3.000	4.300	8.400	2.100
fuelCost08	37843.000	1882.060	510.280	550.000	1500.000	1850.000	2200.000	5800.000	700.000
highway08	37843.000	24.105	6.963	9.000	20.000	24.000	27.000	111.000	7.000
id	37843.000	19019.286	11034.785	1.000	9461.500	18923.000	28570.500	38173.000	19109.000
UCity	37843.000	22.587	9.350	0.000	18.000	21.000	25.139	197.577	7.139
UHighway	37843.000	33.619	10.048	0.000	27.100	33.000	38.110	159.100	11.010
youSaveSpend	37843.000	-2658.999	2553.098	-22250.000	-4250.000	-2500.000	-750.000	4000.000	3500.000

we established, and appending those results to our outlier data frame.

```
outliers = pd.DataFrame(columns=data.columns)

for col in stats.index:
    iqr = stats.at[col,'IQR']
    cutoff = iqr * 1.5
    lower = stats.at[col,'25%'] - cutoff
    upper = stats.at[col,'75%'] + cutoff
    results = data[(data[col] < lower) |
                   (data[col] > upper)].copy()
    results['Outlier'] = col
    outliers = outliers.append(results)
```

Copy

Our outliers data frame should now be populated with records that you can investigate further and determine whether they should be kept in the data or dropped. The Outlier column we added before appending the results for the column to the outliers data frame will let you know what column in each record contained the outlier. If you find that this method is returning too many results, you can be more stringent with your cutoff criteria (e.g. increasing the constant by which you multiply the IQR to 3 instead of 1.5).

Data Type Correction

One common problem that is often overlooked is incorrect data types. This typically occurs when there is a numeric variable that should actually be represented as a categorical variable. The way to check the data type of each column in Pandas is by using the `dtypes` method.

Copy

← Go back Copy URL for Students Edit in Studio

```
4   city08              int64
5   city08U           float64
6   co2                 int64
7   co2TailpipeGpm    float64
8   ...
9   VClass             object
10  year                int64
11  youSaveSpend        int64
12  createdOn          object
13  modifiedOn         object
14  dtype: object
```

Pandas currently has the year column stored as integers, but what if we wanted the year to be stored as a categorical variable (object) instead? We could easily change that data type using the `astype` method and then check that it changed using the `dtypes` method again just on that field.

```
1   data['year'] = data['year'].astype('object')
2   data['year'].dtype
3
4   dtype('O')
```

Copy

You can apply this technique to any column whose data type you would like to change.

Cleaning Text and Removing Special Characters

The presence of special characters in our fields has the potential to make analyzing our data challenging. Imagine not being able to perform calculations on a numeric field because it was currently represented as an object data type due to the fact that it had a dollar sign ($) in it. Similarly, imagine having a categorical field where you could not group records that belong in the same group together because in one field you are grouping by, terms that refer to the same thing are sometimes hyphenated. In cases like this, it is necessary to remove special characters so that we can properly analyze the data.

```
print(set(data['trany']))

{nan, 'Automatic 4-spd', 'Auto (AV-S6)', 'Auto(AM6)', 'Automatic
(S6)', 'Auto(AM-S8)', 'Manual 7-spd', 'Auto(A1)', 'Manual 6-spd',
'Auto(AV-S6)', 'Manual 3-spd', 'Auto(L3)', 'Automatic (S4)',
'Automatic (variable gear ratios)', 'Auto(L4)', 'Manual 4-spd',
'Auto(AV-S7)', 'Automatic 6-spd', 'Auto (AV-S8)', 'Automatic
(S9)', 'Manual(M7)', 'Auto(AM8)', 'Automatic 8-spd', 'Automatic
6spd', 'Automatic (S5)', 'Auto(AM-S9)', 'Automatic (AM6)',
'Automatic (AM5)', 'Automatic (AV)', 'Auto(AV-S8)', 'Automatic
5-spd', 'Automatic (A6)', 'Manual 5 spd', 'Auto(AM-S7)',
'Automatic (A1)', 'Auto (AV)', 'Auto(AM5)', 'Auto(AM7)',
'Automatic 9-spd', 'Auto(AM-S6)', 'Manual 5-spd', 'Automatic
(S8)', 'Automatic (AV-S6)', 'Automatic 7-spd', 'Manual 4-spd
Doubled', 'Automatic (S7)', 'Automatic 3-spd'}
```

Copy

We can see that there are instances that refer to the same thing, but would not get grouped together due to special characters (e.g. Automatic 6-spd and Automatic 6spd). So let's remove all hyphens from this column with the help of the `str.replace` method and then print unique values again to ensure they were removed.

```
4   {nan, 'Auto(AM6)', 'Auto(AVS7)', 'Automatic (S6)', 'Manual 4spd
    Doubled', 'Auto(AVS8)', 'Auto(A1)', 'Manual 5spd', 'Manual 3spd',
    'Auto(L3)', 'Automatic (S4)', 'Automatic (variable gear ratios)',
    'Auto(L4)', 'Automatic (S9)', 'Auto(AMS8)', 'Manual 7spd',
    'Manual(M7)', 'Auto(AM8)', 'Automatic 6spd', 'Automatic (AVS6)',
    'Auto (AVS8)', 'Automatic 4spd', 'Automatic 3spd', 'Automatic
    (S5)', 'Auto (AVS6)', 'Automatic (AM6)', 'Automatic (AM5)',
    'Automatic (AV)', 'Automatic 9spd', 'Automatic 8spd', 'Automatic
    (A6)', 'Manual 5 spd', 'Manual 4spd', 'Automatic (A1)', 'Auto
    (AV)', 'Auto(AM5)', 'Manual 6spd', 'Auto(AMS6)', 'Auto(AMS9)',
    'Auto(AM7)', 'Automatic (S8)', 'Automatic 5spd', 'Auto(AVS6)',
    'Automatic (S7)', 'Auto(AMS7)', 'Automatic 7spd'}
```

You will also notice that in some cases Automatic is abbreviated to Auto and in other cases it is spelled out. We can make that more consistent by using the same technique. While we are at it, let's also attempt to remove parentheses and make spacing more consistent.

```
1   data['trany'] = data['trany'].str.replace('Automatic',
    'Auto')
2   data['trany'] = data['trany'].str.replace('Auto\(',
    'Auto ')
3   data['trany'] = data['trany'].str.replace('Manual\(', 'Manual ')
4   data['trany'] = data['trany'].str.replace('\(', '')
5   data['trany'] = data['trany'].str.replace('\)', '')
6   print(set(data['trany']))
7
8   {nan, 'Auto AM5', 'Auto AVS8', 'Auto 7spd', 'Manual 4spd
    Doubled', 'Auto A1', 'Auto AMS7', 'Auto 8spd', 'Auto 9spd',
    'Manual 5spd', 'Auto AMS9', 'Manual M7', 'Manual 3spd', 'Auto
    AV', 'Auto S4', 'Auto AVS6', 'Auto S6', 'Manual 7spd', 'Auto
    AMS8', 'Auto 6spd', 'Auto L3', 'Auto AM6', 'Auto L4', 'Auto
    AVS7', 'Auto S8', 'Auto 5spd', 'Manual 4spd', 'Auto 3spd', 'Auto
    S5', 'Auto S7', 'Auto A6', 'Manual 5 spd', 'Auto variable gear
    ratios', 'Auto S9', 'Manual 6spd', 'Auto AM8', 'Auto AM7', 'Auto
    AMS6', 'Auto 4spd'}
```

Copy

Finding and Removing Duplicates

The final topic we are going to cover in this lesson is how to identify and remove duplicate rows (or rows that refer to the same entity) in our data. When trying to identify duplicates, we will use the columns (or attributes) of the data to help us determine what entities are similar enough to be considered the same entity. We want to start with all the columns we currently have available to us and work our way toward a lesser number of attributes in an intuitive fashion. In this process, the act of dropping duplicated records is easy, but identifying the correct attributes for comparison and which records to drop is sometimes quite challenging.

The first thing we will do is attempt to drop any duplicate records, considering all the columns we currently have in the data set. Pandas provides us with the ability to do that via the `drop_duplicates` method. We will use the `len` method to calculate the number of rows in the data set both before and after removing duplicates and then print the number of rows dropped.

```
before = len(data)
data = data.drop_duplicates()
after = len(data)
print('Number of duplicate records dropped: ', str(before - after))

Number of duplicate records dropped:  0
```

Copy

This tells us that there were no records that matched exactly across all columns. However, if we reduce the number of columns in our data that we are interested in, we can try again and have a higher likelihood of finding duplicate records. In the example below, we will select a subset of columns, remove all other columns, and then use the `drop_duplicates` method to drop any duplicate records based on the remaining columns.

Copy

```
    'VClass','fuelType','barrels08',
3                    'city08', 'highway08', 'comb08',
    'co2TailpipeGpm', 'fuelCost08']
4
5   data = data[select_columns].drop_duplicates()
6   after = len(data)
7   print('Number of duplicate records dropped: ', str(before -
    after))
8
9   Number of duplicate records dropped:  885
```

With the reduced number of columns, we were able to identify and drop 885 duplicate records.

Summary

In this lesson, we have learned many techniques for cleaning our data that will make it easier to analyze later. We started the lesson by learning how to examine the data for potential issues. We quickly moved on to finding missing values and incorrect values and correcting them. From there, we learned how to remove low variance columns and identify potential outliers. After that, we learned how to change a column's data type and clean our categorical variables in addition to learning how to identify and remove duplicate records. The techniques you have learned in this lesson are ones that you will find yourself revisiting and utilizing almost every time you prepare to analyze a data set, as most data sets you will encounter will require some level of cleaning.

Mark lesson as completed