```
import pandas as pd
import numpy as np

# Define the headers since the data does not have any
headers = ["symboling", "normalized_losses", "make", "fuel_type", "aspiration",
           "num_doors", "body_style", "drive_wheels", "engine_location",
           "wheel_base", "length", "width", "height", "curb_weight",
           "engine_type", "num_cylinders", "engine_size", "fuel_system",
           "bore", "stroke", "compression_ratio", "horsepower", "peak_rpm",
           "city_mpg", "highway_mpg", "price"]

# Read in the CSV file and convert "?" to NaN
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases
                    /autos/imports-85.data",
                 header=None, names=headers, na_values="?" )
df.head()
```

	symboling	normalized_losses	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	w
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	
4	2	164.0	audi	gas	std	four	sedan	4wd	front	

```
df.dtypes
```

```
symboling              int64
normalized_losses    float64
make                  object
fuel_type             object
aspiration            object
num_doors             object
body_style            object
drive_wheels          object
engine_location       object
wheel_base           float64
length               float64
width                float64
height               float64
curb_weight            int64
engine_type           object
num_cylinders         object
engine_size            int64
fuel_system           object
bore                 float64
stroke               float64
compression_ratio    float64
horsepower           float64
peak_rpm             float64
city_mpg               int64
highway_mpg            int64
price                float64
dtype: object
```

```
obj_df = df.select_dtypes(include=['object']).copy()
obj_df.head()
```

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_type	num_cylinders	fue
0	alfa-	gas	std	two	convertible	rwd	front	dohc	four	

```
obj_df[obj_df.isnull().any(axis=1)]
```

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_type	num_cylinders	fu
27	dodge	gas	turbo	NaN	sedan	fwd	front	ohc	four	
63	mazda	diesel	std	NaN	sedan	fwd	front	ohc	four	

```
obj_df["num_doors"].value_counts() # 4 is so common so fill with 4 simply
```

```
four    114
two      89
Name: num_doors, dtype: int64
```

```
obj_df = obj_df.fillna({"num_doors": "four"})
```

Approach #1 - Find and Replace

```
obj_df["num_cylinders"].value_counts()
```

```
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: num_cylinders, dtype: int64
```

```
cleanup_nums = {"num_doors":     {"four": 4, "two": 2},
```

```
                "num_cylinders": {"four": 4, "six": 6, "five": 5, "eight": 8,
                                  "two": 2, "twelve": 12, "three":3 }}

obj_df.replace(cleanup_nums, inplace=True)
obj_df.head()
```

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_type	num_cylinders	fue
0	alfa-romero	gas	std	2	convertible	rwd	front	dohc	4	
1	alfa-romero	gas	std	2	convertible	rwd	front	dohc	4	
2	alfa-romero	gas	std	2	hatchback	rwd	front	ohcv	6	
3	audi	gas	std	4	sedan	fwd	front	ohc	4	

```
obj_df.dtypes   # those columns are converted to numbers
```

```
make               object
fuel_type          object
aspiration         object
num_doors           int64
body_style         object
drive_wheels       object
engine_location    object
engine_type        object
num_cylinders       int64
fuel_system        object
dtype: object
```

Approach #2 - Label Encoding

```
'''
```

```
Label encoding is simply converting each value in a column to a number
convertible -> 0
hardtop -> 1
hatchback -> 2
sedan -> 3
wagon -> 4

One trick you can use in pandas is to convert a column to a category, then use those ca
'''
```

```
'\nLabel encoding is simply converting each value in a column to a number\nconver
tible -> 0\nhardtop -> 1\nhatchback -> 2\nsedan -> 3\nwagon -> 4\n\nOne trick you
can use in pandas is to convert a column to a category, then use those category v
```

```
obj_df["body_style"] = obj_df["body_style"].astype('category')
obj_df.dtypes
```

```
make                 object
fuel_type            object
aspiration           object
num_doors             int64
body_style         category
drive_wheels         object
engine_location      object
engine_type          object
num_cylinders         int64
fuel_system          object
dtype: object
```

```
# Then you can assign the encoded variable to a new column using the cat.codes accessor
obj_df["body_style_cat"] = obj_df["body_style"].cat.codes
obj_df.head()
```

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_type	num_cylinders	fue
0	alfa-romero	gas	std	2	convertible	rwd	front	dohc	4	
1	alfa-romero	gas	std	2	convertible	rwd	front	dohc	4	
2	alfa-	gas	std	2	hatchback	rwd	front	ohcv	6	

Approach #3 - One Hot Encoding

Label encoding has the advantage that it is straightforward but it has the disadvantage that the numeric values can be “misinterpreted” by the algorithms. For example, the value of 0 is obviously less than the value of 4 but does that really correspond to the data set in real life?

Pandas supports this feature using get_dummies. This function is named this way because it creates dummy/indicator variables (aka 1 or 0).

Hopefully a simple example will make this more clear. We can look at the column drive_wheels where we have values of 4wd , fwd or rwd . By using get_dummies we can convert this to three columns with a 1 or 0 corresponding to the correct value:

```
pd.get_dummies(obj_df, columns=["drive_wheels"]).head()
```

	make	fuel_type	aspiration	num_doors	body_style	engine_location	engine_type	num_cylinders	fuel_system	body
0	alfa-	gas	std	2	convertible	front	dohc	4	mpfi	

The new data set contains three new columns:

drive_wheels_4wd drive_wheels_rwd drive_wheels_fwd This function is powerful because you can pass as many category columns as you would like and choose how to label the columns using prefix . Proper naming will make the rest of the analysis just a little bit easier.

3	audi	gas	std	4	sedan	front	ohc	4	mpfi	

```
pd.get_dummies(obj_df, columns=["body_style", "drive_wheels"], prefix=["body", "drive"]
```

	make	fuel_type	aspiration	num_doors	engine_location	engine_type	num_cylinders	fuel_system	body_style_cat
0	alfa-romero	gas	std	2	front	dohc	4	mpfi	0
1	alfa-romero	gas	std	2	front	dohc	4	mpfi	0
2	alfa-romero	gas	std	2	front	ohcv	6	mpfi	2
3	audi	gas	std	4	front	ohc	4	mpfi	3
4	audi	gas	std	4	front	ohc	5	mpfi	3

The other concept to keep in mind is that get_dummies returns the full dataframe so you will need to filter out the objects using select_dtypes when you are ready to do the final analysis.

One hot encoding, is very useful but it can cause the number of columns to expand greatly if you have very many unique values in a column. For the number of values in this example, it is not a problem. However you can see how this gets really challenging to manage when you have many more options.

Approach #4 - Custom Binary Encoding

```
obj_df["engine_type"].value_counts()
```

```
ohc      148
ohcf      15
ohcv      13
l         12
dohc      12
rotor      4
dohcv      1
Name: engine_type, dtype: int64
```

For the sake of discussion, maybe all we care about is whether or not the engine is an Overhead Cam (OHC) or not. In other words, the various versions of OHC are all the same for this analysis. If this is the case, then we could use the str accessor plus np.where to create a new column the indicates whether or not the car has an OHC engine.

```
obj_df["OHC_Code"] = np.where(obj_df["engine_type"].str.contains("ohc"), 1, 0)

obj_df[["make", "engine_type", "OHC_Code"]].head()    # This approach can be really use
```

	make	engine_type	OHC_Code
0	alfa-romero	dohc	1
1	alfa-romero	dohc	1
2	alfa-romero	ohcv	1
3	audi	ohc	1
4	audi	ohc	1

Scikit-Learn

```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
obj_df["make_code"] = lb_make.fit_transform(obj_df["make"])
obj_df[["make", "make_code"]].head(11)
```

	make	make_code
0	alfa-romero	0
1	alfa-romero	0
2	alfa-romero	0
3	audi	1
4	audi	1
5	audi	1
6	audi	1
7	audi	1
8	audi	1
9	audi	1
10	bmw	2

```
from sklearn.preprocessing import LabelBinarizer

lb_style = LabelBinarizer()
lb_results = lb_style.fit_transform(obj_df["body_style"])
pd.DataFrame(lb_results, columns=lb_style.classes_).head()
```

	convertible	hardtop	hatchback	sedan	wagon
0	1	0	0	0	0
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	1	0

```
#BACKWARD DIFFERENCE ENCODING

#POLYNOMIAL ENCODING

#MEAN ENCODING



pd.get_dummies(df_train['Crop_Type'])
df_train=pd.get_dummies(data=df_train, columns=['Crop_Type','Soil_Type','Pesticide_Use_

# types 1, 2, 3 are window glass
# types 5, 6, 7 are household glass
glass['household'] = glass.glass_type.map({1:0, 2:0, 3:0, 5:1, 6:1, 7:1})
#glass.head()

import pandas as pd
df=pd.read_csv('https://raw.githubusercontent.com/fivethirtyeight/data/master/biopics/b

df.head()
```

https://raw.githubusercontent.com/fivethirtyeight/data/master/biopics/biopics.csv

	title	site	country	year_release	box_office	director	number_of_subjects	subject
0	10 Rillington Place	http://www.imdb.com/title/tt0066730/	UK	1971	-	Richard Fleischer	1	John Christie
1	12 Years a Slave	http://www.imdb.com/title/tt2024544/	US/UK	2013	$56.7M	Steve McQueen	1	Solomon Northup
2	127 Hours	http://www.imdb.com/title/tt1542344/	US/UK	2010	$18.3M	Danny Boyle	1	Aron Ralston
3	1987	http://www.imdb.com/title/tt2833074/	Canada	2014	-	Ricardo Trogi	1	Ricardo Trogi
4	20 Dates	http://www.imdb.com/title/tt0138987/	US	1998	$537K	Myles Berkowitz	1	Myles Berkowitz

df.dtypes

```
title                object
site                 object
country              object
year_release          int64
box_office           object
director             object
number_of_subjects    int64
subject              object
type_of_subject      object
race_known           object
subject_race         object
person_of_color       int64
subject_sex          object
lead_actor_actress   object
dtype: object
```