Data Wrangling I

Perform the following operations using Python on any open source dataset (eg. data.csv)

1. Import all the required Python Libraries.
2. Locate an open source data from the web (eg. https://www.kaggle.com (https://www.kaggle.com)). Provide a clear description of the data and its source (i.e. URL of the web site).
3. Load the Dataset into pandas dataframe.
4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitative variables in Python In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set

By,

Vinayak Jalan

TE B 74

1. Import all the required Python Libraries.

Why do we need pandas, why not our excel?

- **Advantage**

> quickly analyse data & gives you insight
>
> need not to be a programmer

- **Disadvantage**

> Can not handle large amount of data
>
> it may crashes while loading a data
>
> Missing value, cleanind data involves lots of process

- Pandas developed for data analysis
- support Multiple file format
- Time series analysis
- One Script can be used for similar operation again & again

In [127]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#so that we can view the graphs inside the notebook
```

In [128]:

```
#from google.colab import drive
#drive.mount('/content/gdrive')
```

Fundamental Data Types in Pandas

1. Series---- 1 D array with corresponding index
2. Data Frame ---- n D array

In [129]:

```
s1 = pd.Series(range(1,10,1))
```

In [130]:

```
s1
```

Out[130]:

```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
8    9
dtype: int64
```

In [131]:

```
s3 = pd.Series({1:21, 2:13,3:45})
```

In [132]:

```
s3
```

Out[132]:

```
1    21
2    13
3    45
dtype: int64
```

In [133]:

```
s2 = pd.Series([1, 2, 3, 4], index=['p', 'q', 'r','s'], name='one')
```

In [134]:

```
s2
```

Out[134]:

```
p    1
q    2
r    3
s    4
Name: one, dtype: int64
```

In [135]:

```
df1 = pd.DataFrame(s2)
df1
```

Out[135]:

	one
p	1
q	2
r	3
s	4

2.Locate an open source data from the web (e.g. https://www.kaggle.com (https://www.kaggle.com)). Provide a clear description of the data and its source (i.e., URL of the web site).

3.Load the Dataset into pandas data frame

Real power- Import from different formats http://pandas.pydata.org/pandas-docs/version/0.20/io.html (http://pandas.pydata.org/pandas-docs/version/0.20/io.html)

In [136]:

```python
df2 = pd.read_csv("/content/sample_data/california_housing_test.csv")
#dataframe_name = pd.read_<format>(filename)
```

In [137]:

```python
df2.head(10)
```

Out[137]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	6
1	-118.30	34.26	43.0	1510.0	310.0	809.0	2
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	4
3	-118.36	33.82	28.0	67.0	15.0	49.0	
4	-119.67	36.33	19.0	1241.0	244.0	850.0	2
5	-119.56	36.51	37.0	1018.0	213.0	663.0	2
6	-121.43	38.63	43.0	1009.0	225.0	604.0	2
7	-120.65	35.48	19.0	2310.0	471.0	1341.0	4
8	-122.84	38.40	15.0	3080.0	617.0	1446.0	5
9	-118.02	34.08	31.0	2402.0	632.0	2830.0	6

In [138]:

```python
df2.tail(3)
```

Out[138]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
2997	-119.70	36.30	10.0	956.0	201.0	693.0	
2998	-117.12	34.10	40.0	96.0	14.0	46.0	
2999	-119.63	34.42	42.0	1765.0	263.0	753.0	

In [139]:

```python
df2['median_house_value_new']=df2['median_house_value']+111
```

In [140]:

```
df2.tail(3)
```

Out[140]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
2997	-119.70	36.30	10.0	956.0	201.0	693.0	
2998	-117.12	34.10	40.0	96.0	14.0	46.0	
2999	-119.63	34.42	42.0	1765.0	263.0	753.0	

In [141]:

```
# write
# <dataframe's name>.to_<file_format>(<file_name>)
```

In [142]:

```
df2.to_json('data1.json')
```

In [143]:

```
#If our age dataset is an year old
#df[age_now]= df[age]+1
#df[salary_increment]=df[salary]+5000
```

In [144]:

```
#df1['value'] = df1['num']*2
# internally for each value in column num perform each_value*2 and save it as the corre
sponding
# result in the value column
#df1
```

In [145]:

```
len(df2['total_rooms'])
```

Out[145]:

3000

In [146]:

```
df2['total_rooms'].count()
```

Out[146]:

3000

In [147]:

```
df2['total_rooms'].mean()
```

Out[147]:

```
2599.578666666667
```

In [148]:

```
df2['total_rooms'].sum()
```

Out[148]:

```
7798736.0
```

In [149]:

```
df2['total_rooms'].median()
```

Out[149]:

```
2106.0
```

In [150]:

```
df2['total_rooms'].std()
```

Out[150]:

```
2155.59333162558
```

In [151]:

```
df2['total_rooms'].min()
```

Out[151]:

```
6.0
```

In [152]:

```
df2['total_rooms'].max()
```

Out[152]:

```
30450.0
```

In [153]:

```
df2['total_rooms'].describe()
```

Out[153]:

```
count     3000.000000
mean      2599.578667
std       2155.593332
min          6.000000
25%       1401.000000
50%       2106.000000
75%       3129.000000
max      30450.000000
Name: total_rooms, dtype: float64
```

In [154]:

```
df2['total_rooms'].cumsum()
```

Out[154]:

```
0          3885.0
1          5395.0
2          8984.0
3          9051.0
4         10292.0
          ...
2995    7790662.0
2996    7795919.0
2997    7796875.0
2998    7796971.0
2999    7798736.0
Name: total_rooms, Length: 3000, dtype: float64
```

In [155]:

```
# When you give the whole dataframe, then all numerical columns will be analysis
df2.mean()
```

Out[155]:

```
longitude                  -119.589200
latitude                     35.635390
housing_median_age           28.845333
total_rooms                2599.578667
total_bedrooms              529.950667
population                 1402.798667
households                  489.912000
median_income                 3.807272
median_house_value       205846.275000
median_house_value_new   205957.275000
dtype: float64
```

In [156]:

```
df2.describe()
```

Out[156]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
count	3000.000000	3000.00000	3000.000000	3000.000000	3000.000000	3000.0(
mean	-119.589200	35.63539	28.845333	2599.578667	529.950667	1402.7!
std	1.994936	2.12967	12.555396	2155.593332	415.654368	1030.5
min	-124.180000	32.56000	1.000000	6.000000	2.000000	5.0(
25%	-121.810000	33.93000	18.000000	1401.000000	291.000000	780.0(
50%	-118.485000	34.27000	29.000000	2106.000000	437.000000	1155.0(
75%	-118.020000	37.69000	37.000000	3129.000000	636.000000	1742.7!
max	-114.490000	41.92000	52.000000	30450.000000	5419.000000	11935.0(

In [157]:

```
df = pd.read_csv("/content/sample_data/california_housing_test.csv")
```

In [158]:

```
df.describe()
```

Out[158]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
count	3000.000000	3000.00000	3000.000000	3000.000000	3000.000000	3000.0(
mean	-119.589200	35.63539	28.845333	2599.578667	529.950667	1402.7!
std	1.994936	2.12967	12.555396	2155.593332	415.654368	1030.5
min	-124.180000	32.56000	1.000000	6.000000	2.000000	5.0(
25%	-121.810000	33.93000	18.000000	1401.000000	291.000000	780.0(
50%	-118.485000	34.27000	29.000000	2106.000000	437.000000	1155.0(
75%	-118.020000	37.69000	37.000000	3129.000000	636.000000	1742.7!
max	-114.490000	41.92000	52.000000	30450.000000	5419.000000	11935.0(

In [159]:

```
df.columns
```

Out[159]:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

In [160]:

```
df['longitude']
```

Out[160]:

```
0      -122.05
1      -118.30
2      -117.81
3      -118.36
4      -119.67
         ...
2995   -119.86
2996   -118.14
2997   -119.70
2998   -117.12
2999   -119.63
Name: longitude, Length: 3000, dtype: float64
```

In [161]:

```
df.longitude
```

Out[161]:

```
0       -122.05
1       -118.30
2       -117.81
3       -118.36
4       -119.67
          ...
2995    -119.86
2996    -118.14
2997    -119.70
2998    -117.12
2999    -119.63
Name: longitude, Length: 3000, dtype: float64
```

In [162]:

```
df.iloc[:,1:3]
```

Out[162]:

	latitude	housing_median_age
0	37.37	27.0
1	34.26	43.0
2	33.78	27.0
3	33.82	28.0
4	36.33	19.0
...	...	...
2995	34.42	23.0
2996	34.06	27.0
2997	36.30	10.0
2998	34.10	40.0
2999	34.42	42.0

3000 rows × 2 columns

4.Data Preprocessing:

check for missing values in the data using pandas , describe() function to get some initial statistics. Filling missing values using fillna(), replace() and interpolate()

In [163]:

```
# importing pandas as pd
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

data.head(10)
```

Out[163]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
5	Dennis	Male	4/18/1987	1:35 AM	115163	10.125	False	Legal
6	Ruby	Female	8/17/1987	4:20 PM	65476	10.012	True	Product
7	NaN	Female	7/20/2015	10:43 AM	45906	11.598	NaN	Finance
8	Angela	Female	11/22/2005	6:29 AM	95570	18.523	True	Engineering
9	Frances	Female	8/8/2002	6:51 AM	139852	7.524	True	Business Development

In [164]:

```
data.describe()
```

Out[164]:

	Salary	Bonus %
count	1000.000000	1000.000000
mean	90662.181000	10.207555
std	32923.693342	5.528481
min	35013.000000	1.015000
25%	62613.000000	5.401750
50%	90428.000000	9.838500
75%	118740.250000	14.838000
max	149908.000000	19.944000

In [165]:

```
data.isnull()
```

Out[165]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	True
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
995	False	True	False	False	False	False	False	False
996	False	False	False	False	False	False	False	False
997	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	False	False
999	False	False	False	False	False	False	False	False

1000 rows × 8 columns

In [166]:

```
data.notnull()
```

Out[166]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	False
2	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...
995	True	False	True	True	True	True	True	True
996	True	True	True	True	True	True	True	True
997	True	True	True	True	True	True	True	True
998	True	True	True	True	True	True	True	True
999	True	True	True	True	True	True	True	True

1000 rows × 8 columns

In [167]:

```
data.isnull().sum()
```

Out[167]:

```
First Name            67
Gender               145
Start Date             0
Last Login Time        0
Salary                 0
Bonus %                0
Senior Management     67
Team                  43
dtype: int64
```

Filling a null values using fillna()

In [168]:

```
# filling a null values using fillna()
data["Gender"].fillna("No Gender", inplace = True)
```

In [169]:

```
data.isnull().sum()
```

Out[169]:

```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

In [170]:

```
# will replace  Nan value in dataframe with value -99
import numpy as np
data.replace(to_replace = np.nan, value = -99)
```

Out[170]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	-99
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	No Gender	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

In [171]:

```
# filling a missing value with previous ones
data.fillna(method ='pad')
```

Out[171]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Marketing
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	No Gender	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

In [172]:

```
data['Salary'].fillna(int(data['Salary'].mean()), inplace=True)
```

Dropping missing values using dropna()

In [173]:

```
data.dropna(axis=1)
```

Out[173]:

	Gender	Start Date	Last Login Time	Salary	Bonus %
0	Male	8/6/1993	12:42 PM	97308	6.945
1	Male	3/31/1996	6:53 AM	61933	4.170
2	Female	4/23/1993	11:17 AM	130590	11.858
3	Male	3/4/2005	1:00 PM	138705	9.340
4	Male	1/24/1998	4:47 PM	101004	1.389
...	...	...	...	...	...
995	No Gender	11/23/2014	6:09 AM	132483	16.655
996	Male	1/31/1984	6:30 AM	42392	19.675
997	Male	5/20/2013	12:39 PM	96914	1.421
998	Male	4/20/2013	4:45 PM	60500	11.985
999	Male	5/15/2012	6:24 PM	129949	10.169

1000 rows × 5 columns

In [174]:

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                   "B":[None, 2, 54, 3, None],
                   "C":[20, 16, None, 3, 8],
                   "D":[14, 3, None, None, 6]})

# Print the dataframe
df
```

Out[174]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

Syntax: DataFrame.interpolate(method='linear', axis=0, limit=None, inplace=False, limit_direction='forward', limit_area=None, downcast=None, **kwargs)

Parameters : method : {'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'}

In [175]:

```
df.interpolate(method ='linear', limit_direction ='forward')
```

Out[175]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	9.5	4.0
3	3.0	3.0	3.0	5.0
4	1.0	3.0	8.0	6.0

5.Data Formatting and Data Normalization

Data Formatting

In [176]:

```
#remove white space everywhere
text="today is Monday"
#df['Col Name'] = df['Col Name'].str.replace(' ', '')
text.replace(' ','')
```

Out[176]:

'todayisMonday'

In [177]:

```
text=' Today'
text.lstrip()
```

Out[177]:

'Today'

In [178]:

```
text='Today '
text.rstrip()
```

Out[178]:

'Today'

In [179]:

```
text=' Today '
text.strip()
```

Out[179]:

```
'Today'
```

Data Normalization

Scaling or Feature Scaling is the process of changinng the scale of certain features to a common one. This is typically achieved through normalization and standardization.

Normalization is the process of scaling data into a range of [0, 1]. It's more useful and common for regression tasks.

x′ =x−xmin / xmax − xmin

Standardization is the process of scaling data so that they have a mean value of 0 and a standard deviation of 1. It's more useful and common for classification tasks.

x′=x−μ / σ Where μ is mean and σ is standard deviation

In [180]:

```
import pandas
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler

# data values
X = [ [110, 200], [120, 800], [310, 400], [140, 900], [510, 200], [653, 400] ,[310, 880
] ]

# transofrm data
scaler = MinMaxScaler(feature_range=(0,5))
rescaledX = scaler.fit_transform(X)
```

In [181]:

```
X
```

Out[181]:

```
[[110, 200],
 [120, 800],
 [310, 400],
 [140, 900],
 [510, 200],
 [653, 400],
 [310, 880]]
```

In [182]:

```
rescaledX
```

Out[182]:

```
array([[0.        , 0.        ],
       [0.09208103, 4.28571429],
       [1.84162063, 1.42857143],
       [0.27624309, 5.        ],
       [3.68324125, 0.        ],
       [5.        , 1.42857143],
       [1.84162063, 4.85714286]])
```

StandardScaler

In [183]:

```
from sklearn.preprocessing import StandardScaler
import pandas
import numpy

# data values
X = [ [110, 200], [120, 800], [310, 400], [140, 900], [510, 200], [653, 400] ,[310, 880
] ]

# scaler
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
```

In [184]:

```
rescaledX
```

Out[184]:

```
array([[-1.02004521, -1.17792918],
       [-0.96841602,  0.90076937],
       [ 0.01253852, -0.48502966],
       [-0.86515765,  1.24721913],
       [ 1.04512224, -1.17792918],
       [ 1.78341961, -0.48502966],
       [ 0.01253852,  1.17792918]])
```

Normalize data

In [185]:

```
from sklearn.preprocessing import Normalizer
import pandas
import numpy

# data values
X = [ [110, 200], [120, 800], [310, 400], [140, 900], [510, 200], [653, 400] ,[310, 880
] ]

# normalize values
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
```

In [186]:

```
normalizedX
```

Out[186]:

```
array([[0.48191875, 0.87621591],
       [0.14834045, 0.98893635],
       [0.61257167, 0.79041505],
       [0.15370701, 0.98811647],
       [0.9309732 , 0.36508753],
       [0.8527326 , 0.52234769],
       [0.33225942, 0.94318804]])
```

Binary Data Transformation

In [187]:

```
from sklearn.preprocessing import Binarizer
import pandas
import numpy

# data values
X = [ [501, 200], [120, 800], [310, 400], [140, 900], [510, 200], [653, 400] ,[310, 880
] ]

# binarize data
binarizer = Binarizer(threshold=500).fit(X)
binaryX = binarizer.transform(X)
```

In [188]:

```
binaryX
```

Out[188]:

```
array([[1, 0],
       [0, 1],
       [0, 0],
       [0, 1],
       [1, 0],
       [1, 0],
       [0, 1]])
```

6.Turn categorical variables into quantitative variables in Python.

In [189]:

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

Out[189]:

	symboling	normalized_losses	make	fuel_type	aspiration	num_doors	body_style	drive
0	3	NaN	alfa-romero	gas	std	two	convertible	
1	3	NaN	alfa-romero	gas	std	two	convertible	
2	1	NaN	alfa-romero	gas	std	two	hatchback	
3	2	164.0	audi	gas	std	four	sedan	
4	2	164.0	audi	gas	std	four	sedan	

5 rows × 26 columns

In [190]:

```
df.dtypes
```

Out[190]:

```
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
```

In [191]:

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

Out[191]:

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

In [192]:

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

Out[192]:

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	eng
27	dodge	gas	turbo	NaN	sedan	fwd	front	
63	mazda	diesel	std	NaN	sedan	fwd	front	

In [193]:

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

Out[193]:

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

In [194]:

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

In [195]:

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

Out[195]:

make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engine_

Approach #1 - Find and Replace

[https://pbpython.com/categorical-encoding.html (https://pbpython.com/categorical-encoding.html)](https://pbpython.com/categorical-encoding.html)

In [196]:

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

Out[196]:

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

In [197]:

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

In [198]:

```
obj_df = obj_df.replace(cleanup_nums)
obj_df.head()
```

Out[198]:

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engi
0	alfa-romero	gas	std	2	convertible	rwd	front	
1	alfa-romero	gas	std	2	convertible	rwd	front	
2	alfa-romero	gas	std	2	hatchback	rwd	front	
3	audi	gas	std	4	sedan	fwd	front	
4	audi	gas	std	4	sedan	4wd	front	

In [199]:

```
obj_df.dtypes
```

Out[199]:

```
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

In [200]:

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

Out[200]:

```
sedan          96
hatchback      70
wagon          25
hardtop         8
convertible     6
Name: body_style, dtype: int64
```

In [201]:

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

Out[201]:

```
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
```

In [202]:

```
obj_df["body_style_cat"] = obj_df["body_style"].cat.codes
obj_df.head()
```

Out[202]:

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engi
0	alfa-romero	gas	std	2	convertible	rwd	front	
1	alfa-romero	gas	std	2	convertible	rwd	front	
2	alfa-romero	gas	std	2	hatchback	rwd	front	
3	audi	gas	std	4	sedan	fwd	front	
4	audi	gas	std	4	sedan	4wd	front	

Approach #3 - One Hot Encoding

In [203]:

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

Out[203]:

	make	fuel_type	aspiration	num_doors	body_style	engine_location	engine_type	num_
0	alfa-romero	gas	std	2	convertible	front	dohc	
1	alfa-romero	gas	std	2	convertible	front	dohc	
2	alfa-romero	gas	std	2	hatchback	front	ohcv	
3	audi	gas	std	4	sedan	front	ohc	
4	audi	gas	std	4	sedan	front	ohc	

Approach #4 - Scikit-Learn: OrdinalEncoder and OneHotEncoder

In [204]:

```
from sklearn.preprocessing import OrdinalEncoder

ord_enc = OrdinalEncoder()
obj_df["make_code"] = ord_enc.fit_transform(obj_df[["make"]])
obj_df[["make", "make_code"]].head(11)
```

Out[204]:

	make	make_code
0	alfa-romero	0.0
1	alfa-romero	0.0
2	alfa-romero	0.0
3	audi	1.0
4	audi	1.0
5	audi	1.0
6	audi	1.0
7	audi	1.0
8	audi	1.0
9	audi	1.0
10	bmw	2.0

In [205]:

```
from sklearn.preprocessing import OneHotEncoder

oe_style = OneHotEncoder()
oe_results = oe_style.fit_transform(obj_df[["body_style"]])
pd.DataFrame(oe_results.toarray(), columns=oe_style.categories_).head()
```

Out[205]:

	convertible	hardtop	hatchback	sedan	wagon
0	1.0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0

In [206]:

```
obj_df = obj_df.join(pd.DataFrame(oe_results.toarray(), columns=oe_style.categories_))
```

In [207]:

```
obj_df.head()
```

Out[207]:

	make	fuel_type	aspiration	num_doors	body_style	drive_wheels	engine_location	engi
0	alfa-romero	gas	std	2	convertible	rwd	front	
1	alfa-romero	gas	std	2	convertible	rwd	front	
2	alfa-romero	gas	std	2	hatchback	rwd	front	
3	audi	gas	std	4	sedan	fwd	front	
4	audi	gas	std	4	sedan	4wd	front	