Exploratory Data Analysis

Dr. Muhammad Nouman Durrani

Exploratory data analysis (EDA)

- Exploratory data analysis is an approach for analyzing data sets to summarize their main characteristics, often with visual methods

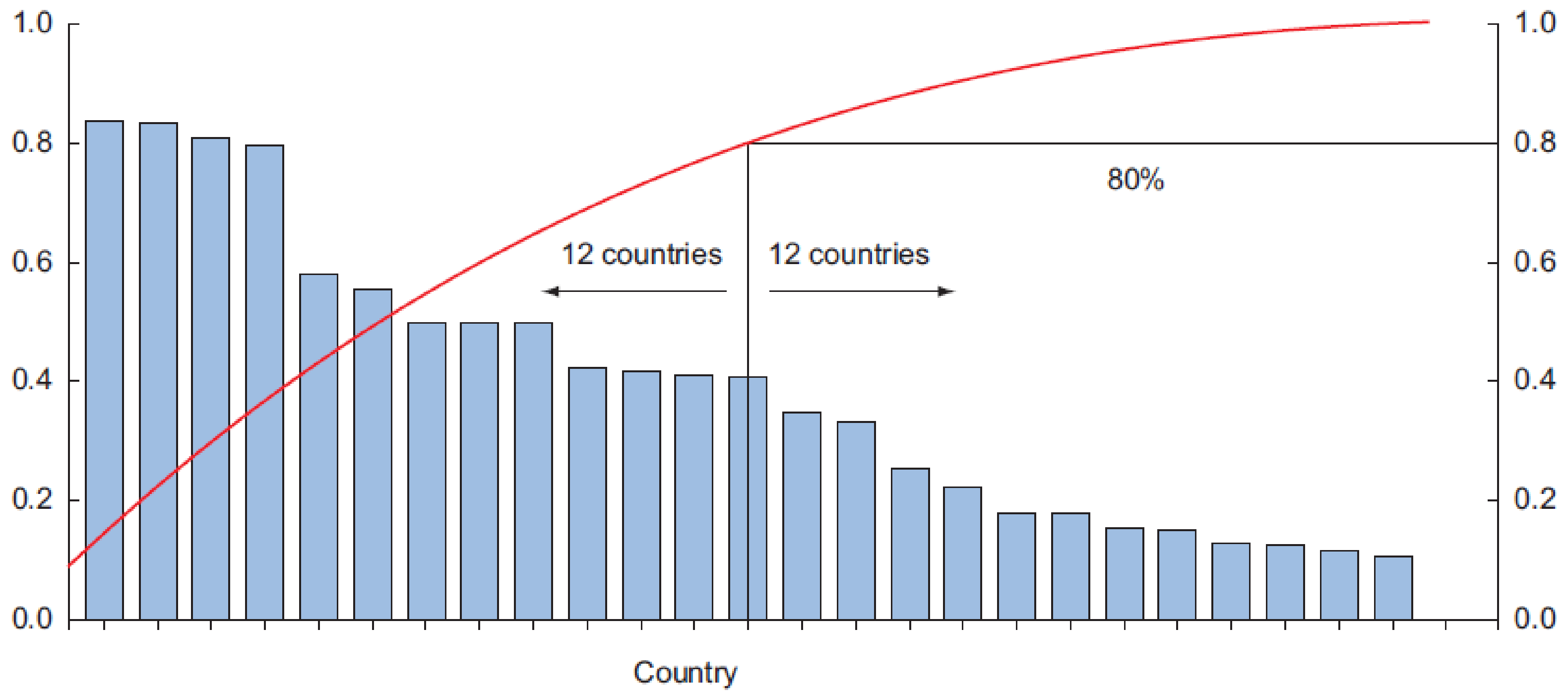
- EDA is statisticians way of *story telling* where you explore data, find patterns and tells insights

Exploratory data analysis (EDA)

- EDA usually employs graphical techniques to get better understanding of data, such as Histograms, Line charts, Box plots etc.
  - However, some other techniques like tabulation, clustering, simple model building can also be used
- Often undetected errors in data are discovered in this phase

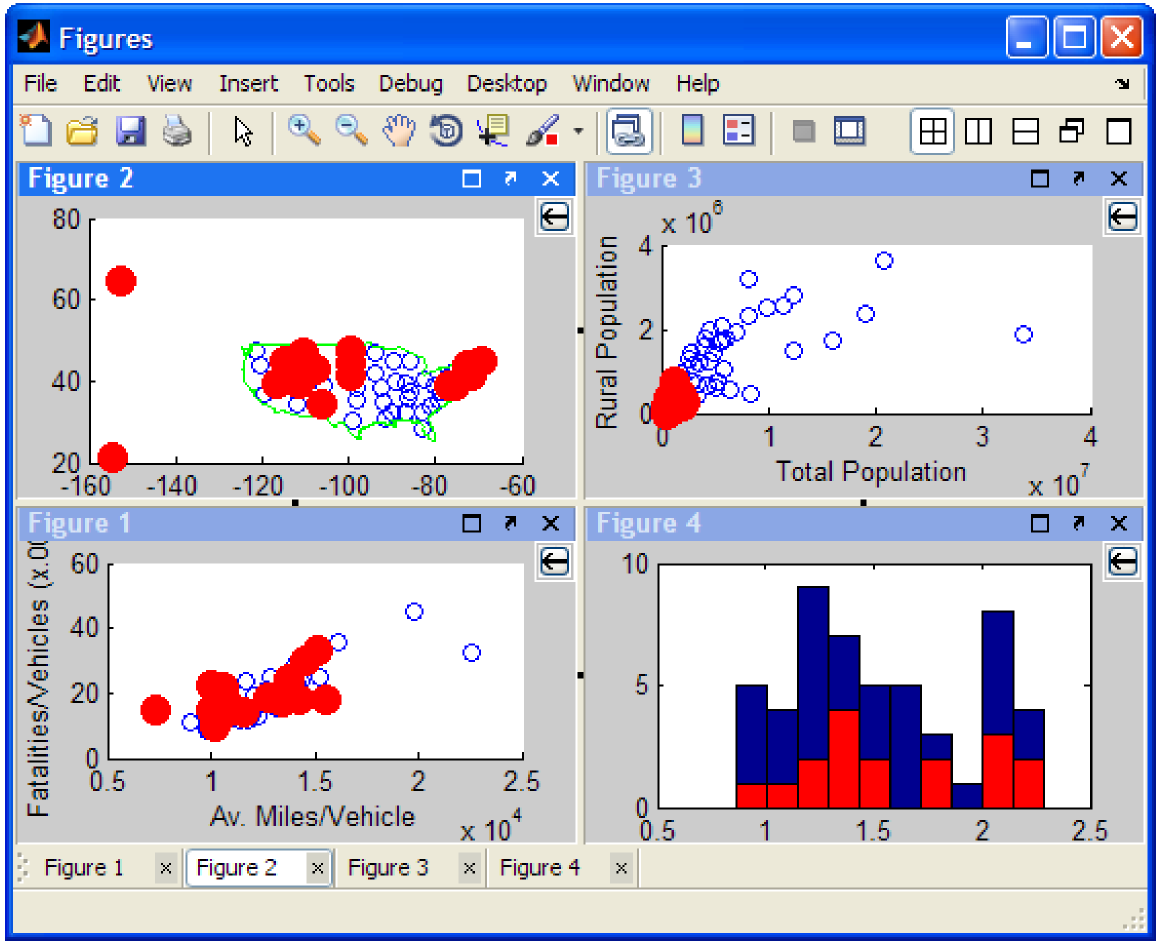
Exploratory data analysis (EDA)

- Overlay plots is also a useful EDA technique, such as Pareto diagram
  - Overlay Plot allows one or more new plot curve(s) to be drawn over existing plots using data from a Parametric, Lookup, Arrays, or Integral table.



Exploratory data analysis (EDA)

- Brushing and Linking (IVA)



Pandas Library

- Pandas is a Python library that provides extensive means for data analysis
- Data scientists often work with data stored in table formats like .csv, .tsv, or .xlsx
- Pandas makes it very convenient to load, process, and analyze such tabular data using SQL-like queries
- In conjunction with Matplotlib and Seaborn, Pandas provides a wide range of opportunities for visual analysis of tabular data

Pandas Library

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages
0	KS	128	415	382-4657	no	yes	25
1	OH	107	415	371-7191	no	yes	26
2	NJ	137	415	358-1921	no	no	0
3	OH	84	408	375-9999	yes	no	0

- The main data structures in Pandas are implemented with Series and DataFrame classes
- Series is a one-dimensional indexed array of some fixed data type
- DataFrame is a two-dimensional data structure - a table - where each column contains data of the same type
  - It as a dictionary of Series instances
  - Rows correspond to instances (examples, observations, etc.), and
  - Columns correspond to features of these instances.

Importing Libraries

Code:

```
import numpy as np
import pandas as pd

# we don't like warnings
# you can comment the following 2 lines if you'd like to

import warnings
warnings.filterwarnings('ignore')
```

Analyzing a dataset on the churn rate of telecom operator clients

- Read the data (using read_csv), and take a look at the first 5 lines using the head method

Code:

```
df = pd.read_csv('Desktop/Python Tutorial/telecom_churn.csv')
df.head()
```

Each row corresponds to one client, an instance, and columns are features of this instance.

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10.0	3	2.70
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13.7	3	3.70
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12.2	5	3.29
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6.6	7	1.78
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10.1	3	2.73

Retrieving data

If we need the first or the last line of the data frame, we can use the df[:1] or df[-1:] construct:

Code:

df[12:15]

df[-1:]

```
df[12:15]
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
12	IA	168	408	363-1107	no	no	0	128.8	96	21.90	...
13	MT	95	510	394-8006	no	no	0	156.6	88	26.62	...
14	IA	62	415	366-9238	no	no	0	120.7	70	20.52	...

3 rows × 21 columns

```
df[-1:]
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...

Data dimensionality, feature names, and feature types

Code:

Data Dimensionality:
print(df.shape)

Printing out column names using columns:
print(df.columns)

```
print(df.shape)
```

```
(3333, 21)
```

```
print(df.columns)
```

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

Feature types:

Use the info() method to output some general information about the dataframe.

Code:

print(df.info())

- bool, int64, float64 and object are the data types of our features
- With this same method, we can easily see if there are any missing values

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
state                     3333 non-null object
account length            3333 non-null int64
area code                 3333 non-null int64
phone number              3333 non-null object
international plan        3333 non-null object
voice mail plan           3333 non-null object
number vmail messages     3333 non-null int64
total day minutes         3333 non-null float64
total day calls           3333 non-null int64
total day charge          3333 non-null float64
total eve minutes         3333 non-null float64
total eve calls           3333 non-null int64
total eve charge          3333 non-null float64
total night minutes       3333 non-null float64
total night calls         3333 non-null int64
total night charge        3333 non-null float64
total intl minutes        3333 non-null float64
total intl calls          3333 non-null int64
total intl charge         3333 non-null float64
customer service calls    3333 non-null int64
churn                     3333 non-null bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.1+ KB
None
```

The number of individuals or items moving out of a collective group over a specific period.

Changing the Feature Type

- Change the column type with the astype method
- Let's apply this method to the churn feature to convert it into int64
- The describe method shows basic statistical characteristics of each numerical feature (int64 and float64 types)
  - number of non-missing values, mean, standard deviation, range, median, 0.25 and 0.75 quartiles

Code:

```
df['churn'] = df['churn'].astype('int64')
df.describe()
```

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037	100.107711
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847	19.568609
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000	33.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000	87.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000	100.000000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000	113.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000	175.000000

Statistics of non-numerical features

Explicitly indicate data types of interest in the include parameter

Code:

df.describe(include=['object', 'bool'])

```
df.describe(include=['object', 'bool'])
```

	state	phone number	international plan	voice mail plan
count	3333	3333	3333	3333
unique	51	3333	2	2
top	WV	350-9720	no	no
freq	106	1	3010	2411

Statistics of Categorical (type object) and Boolean (type bool) features

- For categorical (type object) and boolean (type bool) features we can use the value_counts method
- Let's have a look at the distribution of Churn

Code:

```
df['churn'].value_counts()
```

```
df['churn'].value_counts(normalize=True)
```

```
df['churn'].value_counts()
```

```
0    2850
1     483
Name: churn, dtype: int64
```

```
df['churn'].value_counts(normalize=True)
```

```
0    0.855086
1    0.144914
Name: churn, dtype: float64
```

DataFrame Sorting

- A DataFrame can be sorted by the value of one of the variables (i.e columns).
- For example, we can sort by Total day charge (use ascending=False to sort in descending order)

Code:

df.sort_values(by='total day charge', ascending=False).head()

```
df.sort_values(by='total day charge', ascending=False).head()
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge
365	CO	154	415	343-5709	no	no	0	350.8	75	59.64	...	94	18.40	253.9	100	11.43
985	NY	64	415	345-9140	yes	no	0	346.8	55	58.96	...	79	21.21	275.4	102	12.39
2594	OH	115	510	348-1163	yes	no	0	345.3	81	58.70	...	106	17.29	217.5	107	9.79
156	OH	83	415	370-9116	no	no	0	337.4	120	57.36	...	116	19.33	153.9	114	6.93
605	MO	112	415	373-2053	no	no	0	335.5	77	57.04	...	109	18.06	265.0	132	11.93

Sort by multiple columns:

Code:

```
df.sort_values(by=['churn', 'total day charge'],
       ascending=[True, False]).head()
```

total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	customer service calls	churn
53.65	...	71	17.76	260.1	123	11.70	12.1	3	3.27	3	0
53.35	...	103	12.55	192.7	97	8.67	10.1	7	2.73	3	0
52.77	...	123	5.65	246.5	99	11.09	9.2	10	2.48	4	0
52.68	...	89	17.03	183.5	105	8.26	14.2	2	3.83	1	0
52.36	...	128	21.06	152.9	103	6.88	7.4	3	2.00	1	0

Single Column Statistics: Indexing and retrieving data

(a) To get a single column, we can use a DataFrame['Name'] construction

What is the proportion of churned users in our dataframe?

Code:

df[‘churn'].mean()

```
df['churn'].mean()
```

```
0.14491449144914492
```

Single Column Statistics: Indexing and retrieving data

(b) Boolean indexing with one column:

- The syntax is df[P(df['Name'])], where P is some logical condition that is checked for each element of the Name column.
- The result of such indexing is the DataFrame consisting only of rows that satisfy the P condition on the Name column.

What are average values of numerical features for churned users?

Code:

df[df['churn'] == 1].mean()

```
df[df['churn'] == 1].mean()
```

```
account length            102.664596
area code                 437.817805
number vmail messages       5.115942
total day minutes         206.914079
total day calls           101.335404
total day charge           35.175921
total eve minutes         212.410145
total eve calls           100.561077
total eve charge           18.054969
total night minutes       205.231677
total night calls         100.399586
total night charge          9.235528
total intl minutes         10.700000
total intl calls            4.163561
total intl charge           2.889545
customer service calls      2.229814
churn                       1.000000
dtype: float64
```

Multiple Column Statistics: Indexing and retrieving data

How much time (on average) do churned users spend on the phone during daytime?

Code:

```
df[df['churn'] == 1]['total day minutes'].mean()
```

What is the maximum length of international calls among loyal users (Churn == 0) who do not have an international plan?

Code:

```
df[(df['churn'] == 1) & (df['international plan']== 'yes')]['total intl minutes'].max()
```

```
df[df['churn'] == 1]['total day minutes'].mean()
```

```
206.91407867494814
```

```
df[(df['churn'] == 1) & (df['international plan']== 'yes')]['total intl minutes'].max()
```

```
20.0
```

Indexing and retrieving data

(c) DataFrames indexing by column name (label) or row name (index) or by the serial number of a row

- The **loc method** is used for indexing by name, while **iloc()** is used for indexing by number
- In the first case below, we say "give us the values of the rows with index from 0 to 5 (inclusive) and columns labeled from State to Area code (inclusive)"
- In the second case, we say "give us the values of the first five rows in the first three columns" (as in a typical Python slice: the maximal value is not included)

Code:

df.loc[0:5, 'State':'Area code']

df.iloc[0:4, 0:4]

```
df.loc[0:5, 'state':'area code']
```

	state	account length	area code
0	KS	128	415
1	OH	107	415
2	NJ	137	415
3	OH	84	408
4	OK	75	415
5	AL	118	510

```
df.iloc[0:4, 0:4]
```

	state	account length	area code	phone number
0	KS	128	415	382-4657
1	OH	107	415	371-7191
2	NJ	137	415	358-1921
3	OH	84	408	375-9999
4	OK	75	415	330-6626

Indexing and retrieving data

(d) If we need the first or the last line of the data frame, we can use the df[:1] or df[-1:] construct:

Code:

df[12:15] # Index 12 to 15

df[-1:]

```
df[12:15]
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
12	IA	168	408	363-1107	no	no	0	128.8	96	21.90	...
13	MT	95	510	394-8006	no	no	0	156.6	88	26.62	...
14	IA	62	415	366-9238	no	no	0	120.7	70	20.52	...

3 rows × 21 columns

```
df[-1:]
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...

Applying Functions to Cells, Columns and Rows

To apply functions to each column, use apply():

The apply method can also be used to apply a function to each row.

- To do this, specify axis=1

Code:

df.apply(np.max)

How to a particular Row?
Home Task

```
df.apply(np.max)
```

```
state                          WY
account length                243
area code                     510
phone number             422-9964
international plan            yes
voice mail plan               yes
number vmail messages          51
total day minutes           350.8
total day calls               165
total day charge            59.64
total eve minutes           363.7
total eve calls               170
total eve charge            30.91
total night minutes           395
total night calls             175
total night charge          17.77
total intl minutes             20
total intl calls               20
total intl charge             5.4
customer service calls          9
churn                           1
dtype: object
```

Applying Functions to Cells, Columns and Rows

Lambda function:

If we need to select all states starting with W, we can do it like this:

Code:

df[df[‘state'].apply(lambda state: state[0] == 'W')].head()

```
df[df['state'].apply(lambda state: state[0] == 'W')].head()
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	...
26	WY	57	408	357-3817	no	yes	39	213.0	115	36.21	...
44	WI	64	510	352-1237	no	no	0	154.0	67	26.18	...
49	WY	97	415	405-7146	no	yes	24	133.2	135	22.64	...
54	WY	87	415	353-3759	no	no	0	151.0	83	25.67	...

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

x = lambda a, b : a * b
print(x(5, 6))

Applying Functions to Cells, Columns and Rows

Map function:

- The map method can be used to replace values in a column by passing a dictionary of the form {old_value: new_value} as its argument:

Code:

```
d = {'No' : False, 'Yes' : True}
df['International plan'] = df['International plan'].map(d)
df.head()
```

The same thing can be done with the replace method:

```
df = df.replace({'Voice mail plan': d})
df.head()
```

```
d = {'no' : False, 'yes' : True}
df['international plan'] = df['international plan'].map(d)
df.head()
```

```
df = df.replace({'voice mail plan': d})
df.head()
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages
0	KS	128	415	382-4657	False	True	25
1	OH	107	415	371-7191	False	True	26
2	NJ	137	415	358-1921	False	False	0

Grouping Data

```
columns_to_show = ['total day minutes', 'total eve minutes',
                   'total night minutes']

df.groupby(['churn'])[columns_to_show].describe(percentiles=[])
```

	total day minutes						total eve minutes					
	count	mean	std	min	50%	max	count	mean	std	min	50%	max
churn												
0	2850.0	175.175754	50.181655	0.0	177.2	315.6	2850.0	199.043298	50.292175	0.0	199.6	361.8
1	483.0	206.914079	68.997792	0.0	217.6	350.8	483.0	212.410145	51.728910	70.9	211.3	363.7

df.groupby(by=grouping_columns)[columns_to_show].function()

- First, the groupby method divides the grouping_columns by their values.
  - They become a new index in the resulting dataframe.
- Then, columns of interest are selected (columns_to_show).
  - If columns_to_show is not included, all non groupby clauses will be included.
- Finally, one or several functions are applied to the obtained groups per selected columns.

Code:

```
columns_to_show = ['total day minutes', 'total eve minutes’, total night minutes']

df.groupby(['churn'])[columns_to_show].describe(percentiles=[])
```

Grouping Data

Passing a list of functions to agg():

Code:

```
columns_to_show = ['total day minutes', 'total eve minutes',
                   'total night minutes']

df.groupby(['churn'])[columns_to_show].agg([np.mean, np.std, np.min, np.max])
```

	total day minutes				total eve minutes			
	mean	std	amin	amax	mean	std	amin	amax
churn								
0	175.175754	50.181655	0.0	315.6	199.043298	50.292175	0.0	361.8
1	206.914079	68.997792	0.0	350.8	212.410145	51.728910	70.9	363.7

Summary tables

- Suppose we want to see how the observations in our sample are distributed in the context of two variables - Churn and International plan
- To do so, we can build a contingency table using the crosstab method

Code:

```
pd.crosstab(df['Churn'], df['International plan'])

pd.crosstab(df['Churn'], df['Voice mail plan'], normalize=True)
```

```
pd.crosstab(df['churn'], df['international plan'])
```

international plan	False	True
churn		
0	2664	186
1	346	137

```
pd.crosstab(df['churn'], df['voice mail plan'], normalize=True)
```

voice mail plan	False	True
churn		
0	0.602460	0.252625
1	0.120912	0.024002

```python
df.pivot_table(['total day calls', 'total eve calls', 'total night calls'],
               ['area code'], aggfunc='std') #sum , mean, maximum, minimum or something else.
```

area code	total day calls	total eve calls	total night calls
408	19.694243	19.433974	19.407893
415	20.251354	20.090270	19.428139
510	20.098565	20.080271	19.983527

Pivot tables

Pivot_table method takes the following parameters:

- values – a list of variables to calculate statistics for
- index – a list of variables to group data by
- aggfunc – what statistics we need to calculate for groups:
  - sum, mean, maximum, minimum or something else

Let's look at the average number of day, evening, and night calls by area code:

Code:

df.pivot_table(['total day calls', 'total eve calls', 'total night calls'], ['area code'], aggfunc='std')
#sum , mean, maximum, minimum or something else.

DataFrame transformations

total intl minutes	total intl calls	total intl charge	customer service calls	churn	Total calls
10.0	3	2.70	1	0	303
13.7	3	3.70	1	0	332
12.2	5	3.29	0	0	333
6.6	7	1.78	2	0	255
10.1	3	2.73	3	0	359

Adding columns to a DataFrame:

For example: Calculate the total number of calls for all users

- Create the total_calls Series and paste it into the DataFrame

Code:

```
total_calls = df['Total day calls'] + df['Total eve calls'] + \
            df['Total night calls'] + df['Total intl calls']
df.insert(loc=len(df.columns), column='Total calls', value=total_calls)
df.head()

# loc parameter is the number of columns after which to insert the Series object
# we set it to len(df.columns) to paste it at the very end of the dataframe
```

DataFrame transformations

```
df['Total charge'] = df['total day charge'] + df['total eve charge'] + \
                     df['total night charge'] + df['total intl charge']
df.head()
```

Adding columns to a DataFrame:

- Add a column without creating an intermediate Series instance

Code:

```
df['Total charge'] = df['total day charge'] + df['total eve charge'] + \
                df['total night charge'] + df['total intl charge']
df.head()
```

total intl minutes	total intl calls	total intl charge	customer service calls	churn	Total calls	Total charge
10.0	3	2.70	1	0	303	75.56
13.7	3	3.70	1	0	332	59.24
12.2	5	3.29	0	0	333	62.29

DataFrame transformations

Delete columns or rows

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61

- Use the drop method, passing the required indexes and the axis parameter (1 if you delete columns, and nothing or 0 if you delete rows)
- The inplace argument tells whether to change the original DataFrame.
  - With inplace=False, the drop method doesn't change the existing DataFrame and returns a new one with dropped rows or columns.
  - With inplace=True, it alters the DataFrame.

Code:

```
# get rid of just created columns
df.drop(['Total charge', 'Total calls'], axis=1, inplace=True)
# and here's how you can delete rows
df.drop([1, 2]).head()
```

Predicting telecom churn Using a crosstab contingency table

How churn rate is related to the International plan feature.

- Using a crosstab contingency table and
- Also through visual analysis with Seaborn

Code:

```
pd.crosstab(df['churn'], df['international plan'], margins=True)
```

International plan	False	True	All
Churn			
0	2664	186	2850
1	346	137	483
All	3010	323	3333

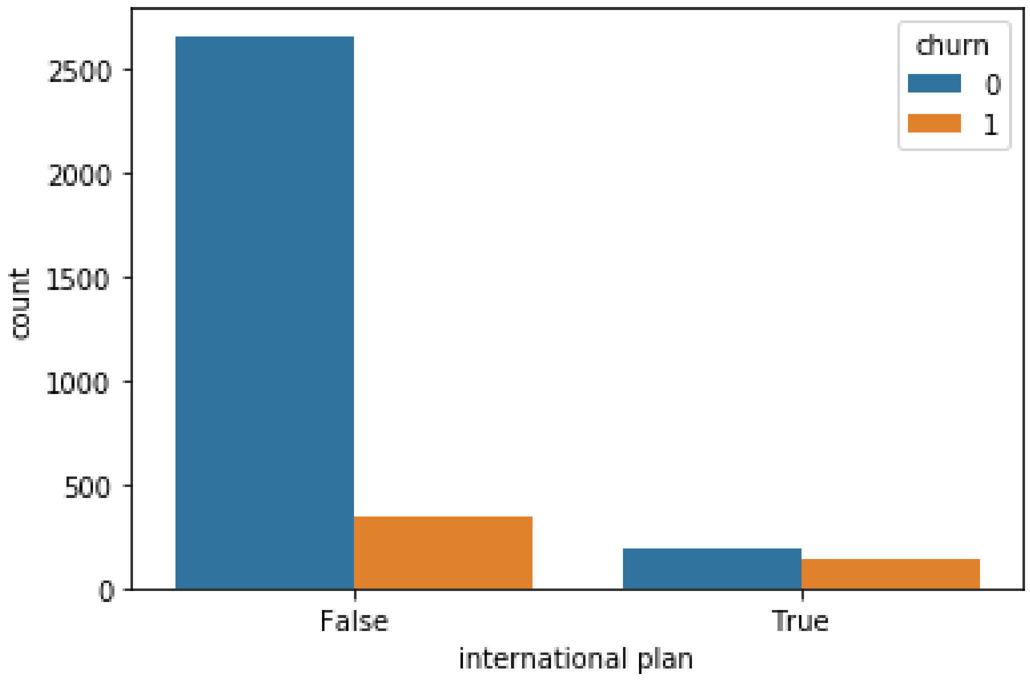
Predicting telecom churn Visual analysis with Seaborn

Code:

```
# some imports to set up plotting
import matplotlib.pyplot as plt
# pip install seaborn
import seaborn as sns

sns.countplot(x='international plan', hue='churn', data=df);
```

```
sns.countplot(x='international plan', hue='churn', data=df);
```



Predicting telecom churn Using a crosstab contingency table

Code:

pd.crosstab(df['churn'], df['customer service calls'], margins=True)

```
pd.crosstab(df['churn'], df['customer service calls'], margins=True)
```

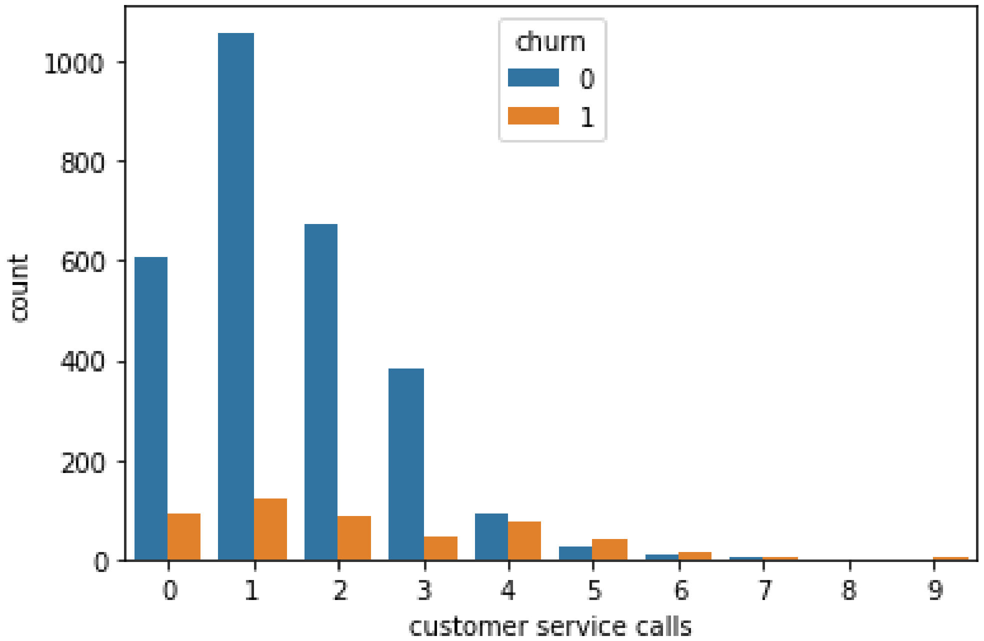
customer service calls	0	1	2	3	4	5	6	7	8	9	All
churn											
0	605	1059	672	385	90	26	8	4	1	0	2850
1	92	122	87	44	76	40	14	5	1	2	483
All	697	1181	759	429	166	66	22	9	2	2	3333

Predicting telecom churn Visual analysis with Seaborn

Code:

sns.countplot(x='Customer service calls', hue='Churn', data=df);

```
sns.countplot(x='customer service calls', hue='churn', data=df);
```



Predicting telecom churn Using a crosstab contingency table

```
df['Many_service_calls'] = (df['customer service calls'] > 3).astype('int')

pd.crosstab(df['Many_service_calls'], df['churn'], margins=True)
```

churn	0	1	All
Many_service_calls			
0	2721	345	3066
1	129	138	267
All	2850	483	3333

Code:

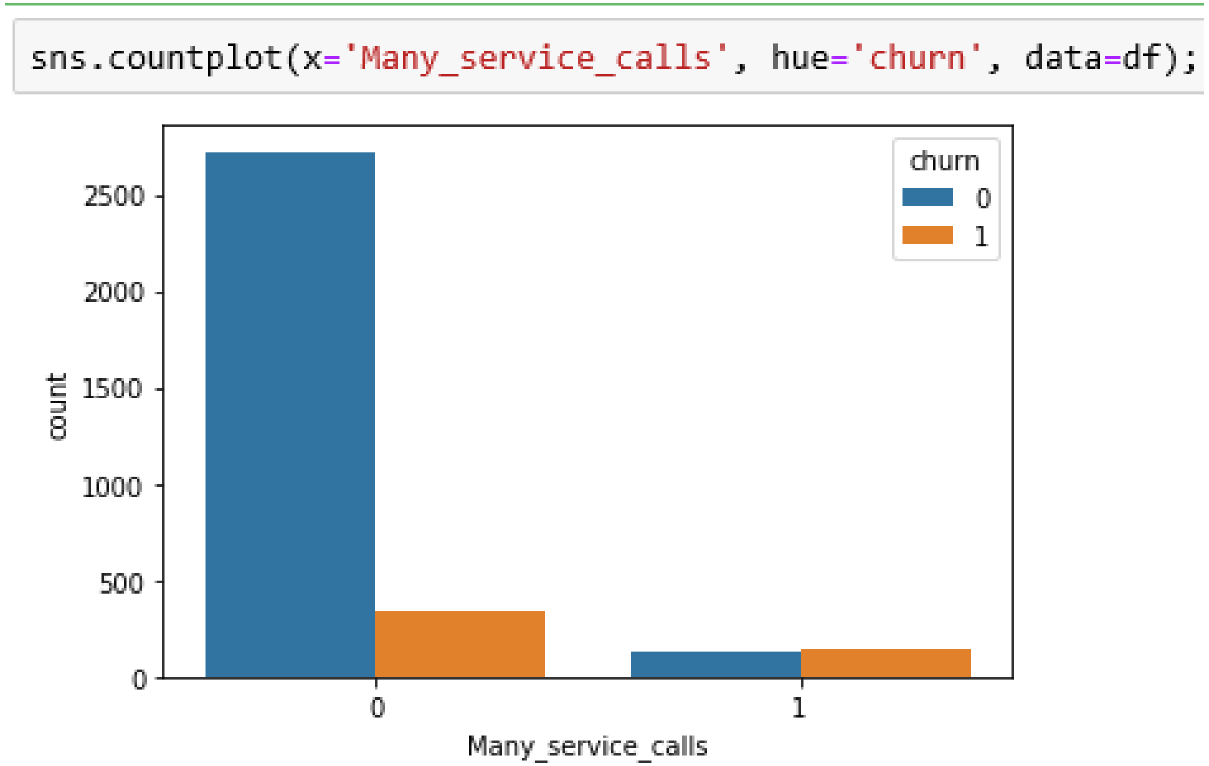
```
df['Many_service_calls'] = (df['customer service calls'] > 3).astype('int')

pd.crosstab(df['Many_service_calls'], df['churn'], margins=True)
```

Predicting telecom churn Visual analysis with Seaborn

Code:

```
sns.countplot(x='Many_service_calls', hue='churn', data=df);
```



Predicting telecom churn Using a crosstab contingency table

- Contingency table that relates Churn with both International plan and freshly created Many_service_calls.

Code:

pd.crosstab(df['Many_service_calls'] & df['international plan'] , df['churn'])

```
pd.crosstab(df['Many_service_calls'] & df['international plan'] , df['churn'])
```

churn	0	1
row_0		
False	2841	464
True	9	19

EDA on another Dataset

Preparations

For the preparation, lets first import the necessary libraries and load the files needed for our EDA

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline

plt.style.use('bmh')
```

Preparations

Code:
```
df = pd.read_csv('train.csv')
df.head()

df.info()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley
0	1	60	RL	65.0	8450	Pave	NaN
1	2	20	RL	80.0	9600	Pave	NaN
2	3	60	RL	68.0	11250	Pave	NaN
3	4	70	RL	60.0	9550	Pave	NaN
4	5	60	RL	84.0	14260	Pave	NaN

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id                  1460 non-null int64
MSSubClass          1460 non-null int64
MSZoning            1460 non-null object
LotFrontage         1201 non-null float64
LotArea             1460 non-null int64
Street              1460 non-null object
Alley               91 non-null object
```

- Some features won't be relevant in our exploratory analysis as there are too much missing values (such as Alley and PoolQC)
- Better to concentrate on the ones which can give us real insights

Preparations

- Remove Id and
- Features with 30% or less NaN values

```python
# df.count() does not include NaN values
df2 = df[[column for column in df if df[column].count() / len(df) >= 0.3]]
del df2['Id']
print("List of dropped columns:", end=" ")
for c in df.columns:
    if c not in df2.columns:
        print(c, end=", ")
print('\n')
df = df2
```

```
List of dropped columns: Id, Alley, PoolQC, Fence, MiscFeature,
```

Code:


```python
# df.count() does not include NaN values
df2 = df[[column for column in df if df[column].count() / len(df) >= 0.3]]
del df2['Id']
print("List of dropped columns:", end=" ")
for c in df.columns:
    if c not in df2.columns:
        print(c, end=", ")
print('\n')
df = df2
```

How the housing price is distributed

Code:

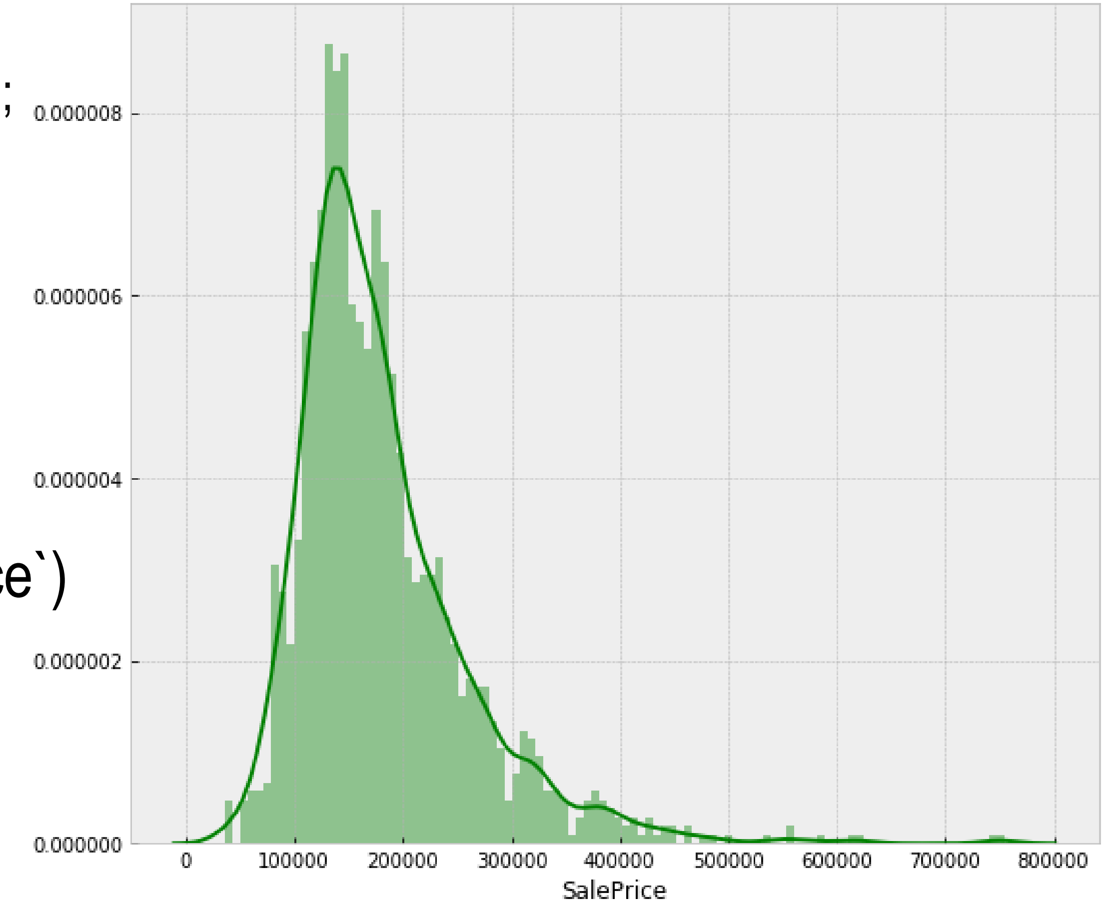
```
print(df['SalePrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(df['SalePrice'], color='g', bins=100, hist_kws={'alpha': 0.4});
```

- Prices are skewed right and some outliers lies above ~500,000
- We will eventually want to get rid of the them to get a normal distribution of the independent variable (`SalePrice`)

How? Home Task


```
print(df['SalePrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(df['SalePrice'], color='g', bins=100, hist_kws={'alpha': 0.4});
```


```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```



Listing Data from a dataset

List all the types of our data from our dataset and take only the numerical ones:

Code:

```
list(set(df.dtypes.tolist()))

df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
```

```
list(set(df.dtypes.tolist()))
```

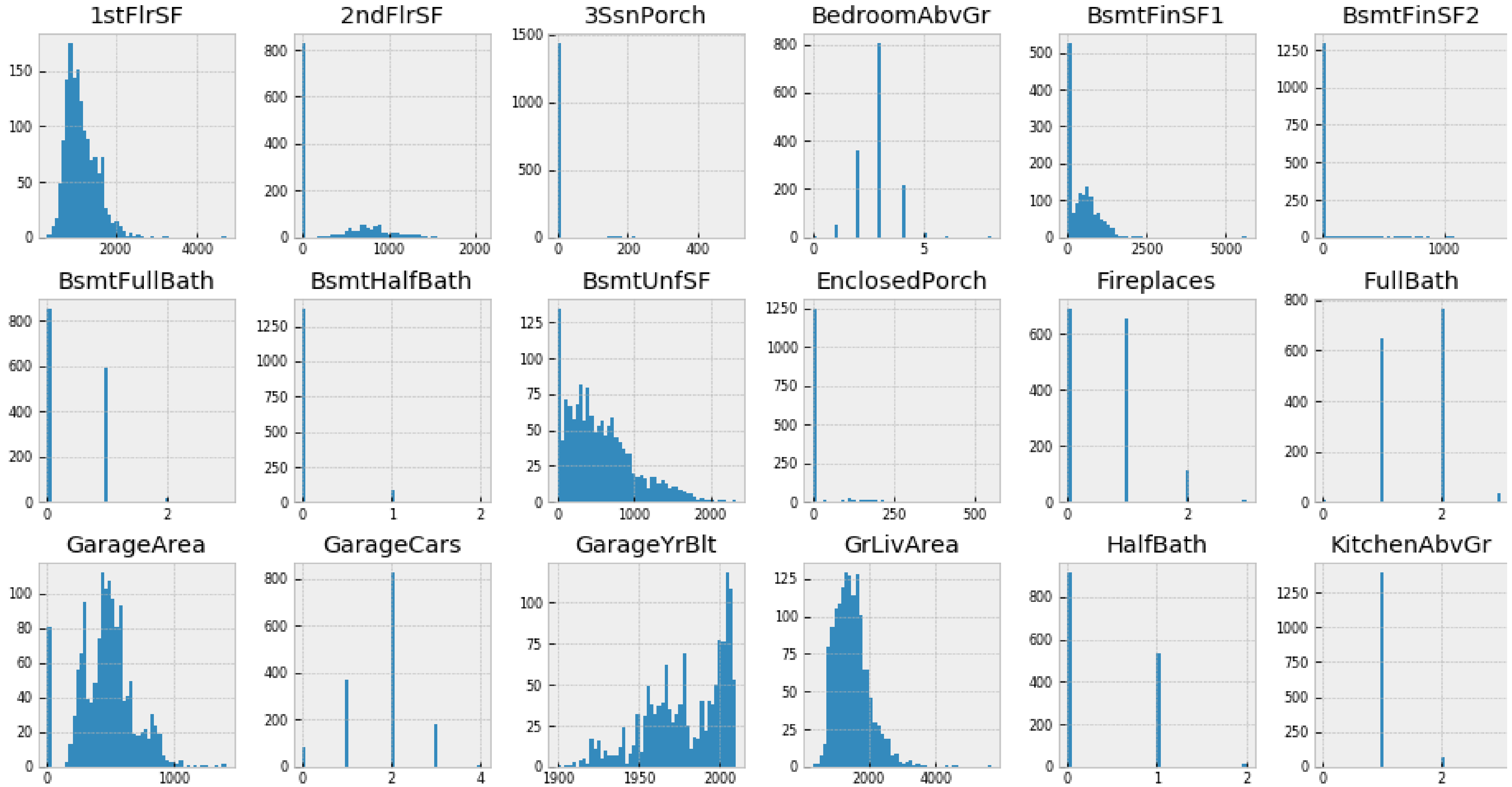
```
[dtype('O'), dtype('float64'), dtype('int64')]
```

```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt
0	60	65.0	8450	7	5	2003
1	20	80.0	9600	6	8	1976
2	60	68.0	11250	7	5	2001
3	70	60.0	9550	7	5	1915
4	60	84.0	14260	8	5	2000

Code:

```
df_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);
# ; avoid having the matplotlib verbose informations
```



Correlation

Features such as `1stFlrSF`, `TotalBsmtSF`, `LotFrontage`, `GrLiveArea`... seems to share a similar distribution to the one we have with `SalePrice`

- Find which features are strongly correlated with SalePrice
- We'll store them in a var called golden_features_list
- We'll reuse our df_num dataset to do so

```
There is 10 strongly correlated values with SalePrice:
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
YearRemodAdd    0.507101
Name: SalePrice, dtype: float64
```

```
df_num_corr = df_num.corr()['SalePrice'][:-1] # -1 because the latest row is SalePrice
golden_features_list = df_num_corr[abs(df_num_corr) > 0.5].sort_values(ascending=False)
print("There is {} strongly correlated values with SalePrice:\n{}".format(len(golden_features_list), golden_features_list))
```

Code:

```
df_num_corr = df_num.corr()['SalePrice'][:-1]                # -1 because the latest row is SalePrice
golden_features_list = df_num_corr[abs(df_num_corr) > 0.5].sort_values(ascending=False)
print("There is {} strongly correlated values with SalePrice:\n{}".format(len(golden_features_list),
golden_features_list))
```