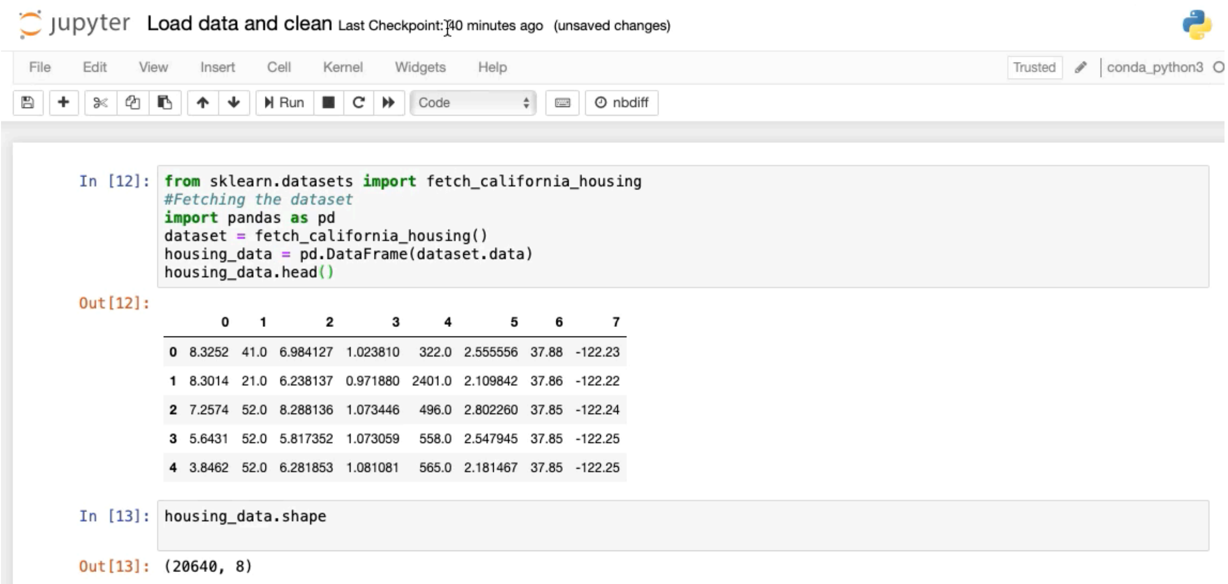
Whizlabs - ML Specialty Exam Course - Data Engineering

https://www.whizlabs.com/learn/course/aws-mls-practice-tests

1. **DATA ENGINEERING**

Loading data from scikit learn data repository in Jupiter



In [12]:
```
from sklearn.datasets import fetch_california_housing
#Fetching the dataset
import pandas as pd
dataset = fetch_california_housing()
housing_data = pd.DataFrame(dataset.data)
housing_data.head()
```

Out[12]:

	0	1	2	3	4	5	6	7
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

In [13]:
```
housing_data.shape
```

Out[13]: (20640, 8)

8 features, 20,640 features

AWS Machine Learning - Data Engineering

Handle Missing Data

- Null value replacement - Several approaches to the problem of handling missing data
  - Do nothing
  - Remove the entire record
  - Mode/median/average value replacement
  - Most frequent value
  - Model-based imputation
    - K-Nearest Neighbors
    - Regression
    - Deep Learning
  - Interpolation / Extrapolation
  - Forward filling / Backward filling
  - Hot deck imputation

Case	Attributes			Decision
	Temperature	Headache	Nausea	Flu
1	high	?	yes	yes
2	very_high	yes	no	yes
3	?	no	no	no
4	normal	yes	?	no
5	?	yes	yes	yes

XGboost can impute missing values

AWS Machine Learning - Handling Missing Data

Null Value Replacement - Which Method Should You Use

- Do nothing and let your algorithm either replace them through imputation (XGBoost) or just ignore them as LightGBM does with its *use_missing=false* parameter
  - Some algorithms will throw an error if they find missing values (LinearRegression)
- Or, replace all missing values

Case	Attributes			Decision
	Temperature	Headache	Nausea	Flu
1	high	?	yes	yes
2	very_high	yes	no	yes
3	?	no	no	no
4	normal	yes	?	no
5	?	yes	yes	yes

Using Pandas to REMOVE ROWS or Observations that don't have Values:

jupyter Drop Features with Missing Values Last Checkpoint: 8 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | conda_python3

In [ ]:

```python
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from math import sqrt
import random
import numpy as np
random.seed(0)

#Fetching the dataset
import pandas as pd
dataset = fetch_california_housing()
train, target = pd.DataFrame(dataset.data), pd.DataFrame(dataset.target)
train.columns = ['zero','one','two','three','four','five','six','seven']
train.insert(loc=len(train.columns), column='target', value=target)

#Randomly replace 40% of the first column with NaN values
column = train['zero']
missing_pct = int(column.size * 0.4)
i = [random.choice(range(column.shape[0])) for _ in range(missing_pct)]
column[i] = np.NaN
train
```

In [19]:

```python
train.shape
```

Out[19]: (20640, 9)

40% Randomly of 1st feature now had NaN value

```python
train
```

Out[21]:

	zero	one	two	three	four	five	six	seven	target
0	NaN	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	NaN	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	NaN	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	NaN	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	NaN	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611
10	3.2031	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815
11	3.2705	52.0	4.772480	1.024523	1504.0	2.049046	37.85	-122.26	2.418
12	NaN	52.0	5.322650	1.012821	1098.0	2.346154	37.85	-122.26	2.135
13	NaN	52.0	4.000000	1.097701	345.0	1.982759	37.84	-122.26	1.913
14	NaN	52.0	4.262903	1.009677	1212.0	1.954839	37.85	-122.26	1.592
15	2.1250	50.0	4.242424	1.071970	697.0	2.640152	37.85	-122.26	1.400

We want to drop the rows that have a NaN

=> use dropna() method

```
In [22]: train.shape
Out[22]: (20640, 9)

In [23]: ######################################################
         # Remove observations that have missing values
         # Will drop all rows that have any missing values.
         ######################################################
         train.dropna(inplace=True)
         train.shape
Out[23]: (13783, 9)
```

Now only have 13k feature from the 20k

Other Imputations Techniques

AWS Machine Learning - Handling Missing Data

Median/Average Value Replacement

- Replace the missing values with a simple median, or mean
  - Reflection of the other values in the feature
  - Doesn't factor correlation between features
  - Can't use on categorical features

Case		Attributes		Decision
	Temperature	Headache	Temperature	Flu
1	high	?	99.9	yes
2	very_high	yes	100.3	yes
3	?	no	98.6	no
4	normal	yes	?	no
5	?	yes	101.0	yes

Example via Code

jupyter Impute using SimpleImpute mean Last Checkpoint: Yesterday at 6:42 PM (autosaved)

```
In [ ]: from sklearn.datasets import fetch_california_housing
        from sklearn.linear_model import LinearRegression
        from sklearn.model_selection import StratifiedKFold
        from sklearn.metrics import mean_squared_error
        from math import sqrt
        import random
        import numpy as np
        random.seed(0)

        #Fetching the dataset
        import pandas as pd
        dataset = fetch_california_housing()
        train, target = pd.DataFrame(dataset.data), pd.DataFrame(dataset.target)
        train.columns = ['zero','one','two','three','four','five','six','seven']
        train.insert(loc=len(train.columns), column='target', value=target)

        #Randomly replace 40% of the first column with NaN values
        column = train['zero']
        missing_pct = int(column.size * 0.4)
        i = [random.choice(range(column.shape[0])) for _ in range(missing_pct)]
        column[i] = np.NaN
        train

In [ ]:
```

```
column[i] = np.NaN
train
```

Out[3]:

	zero	one	two	three	four	five	six	seven	target
0	NaN	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	NaN	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	NaN	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	NaN	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	NaN	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611
10	3.2031	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815
11	3.2705	52.0	4.772480	1.024523	1504.0	2.049046	37.85	-122.26	2.418

Now impute values with **SimpleImputer()** from Scikitlearn
Can call it with “mean”, “median”,…

ravel() method is to unravel into a vector that we can use to replace the train[zero] features

Now the missing values are replaced by the **mean**: 3.87

In [4]:

```
#Impute the values using scikit-learn SimpleImpute Class
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean') #for options other than mean imputation replace
imputer = imputer.fit(train[['zero']])
train['zero'] = imputer.transform(train[['zero']]).ravel()
train
```

Out[4]:

	zero	one	two	three	four	five	six	seven	target
0	3.87794	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.30140	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	3.87794	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.64310	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.87794	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.03680	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.65910	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	3.87794	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	3.87794	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.69120	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611

Now using the **median** => 3.5497

In [6]:

```
#Impute the values using scikit-learn SimpleImpute Class
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='median') #for options other th
imputer = imputer.fit(train[['zero']])
train['zero'] = imputer.transform(train[['zero']]).ravel()
train
```

Out[6]:

	zero	one	two	three	four	five	six	seven	target
0	3.5497	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	3.5497	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.5497	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	3.5497	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	3.5497	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611
10	3.2031	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815

Now using the **most_frequent** => 3.1250

```
In [10]: #Impute the values using scikit-learn SimpleImpute Class
         from sklearn.impute import SimpleImputer
         imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent') #for options other than mean imputation
         imputer = imputer.fit(train[['zero']])
         train['zero'] = imputer.transform(train[['zero']]).ravel()
         train
```

Out[10]:

	zero	one	two	three	four	five	six	seven	target
0	3.1250	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	3.1250	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.1250	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	3.1250	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	3.1250	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267

AWS Machine Learning - Handling Missing Data

Most Frequent Value

- Replace missing values with the most frequently occurring value in the feature
  - Doesn't factor correlation between features
  - Works with categorical features
  - Can introduce bias into your model

Case		Attributes		Decision
	Temperature	Headache	Temperature	Flu
1	high	?	99.9	yes
2	very_high	yes	100.3	yes
3	?	no	98.6	no
4	normal	yes	?	no
5	?	yes	101.0	yes

Now using **Model Based** imputation

AWS Machine Learning - Handling Missing Data

Model-Based Imputation

- Use a machine learning algorithm to impute the missing values
  - K-Nearest Neighbors
    - Uses ‘feature similarity’ to predict missing values
  - Regression
    - Predictors of the variable with missing values identified via correlation matrix
    - Best predictors are selected and used as independent variables in a regression equation
    - Variable with missing data is used as the target variable
  - Deep Learning
    - Works very well with categorical and non-numerical features

Now let’s see model based imputation in code:
Same dataset to start with:

In [1]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
from math import sqrt
import random
import numpy as np
random.seed(0)

#Fetching the dataset
import pandas as pd
dataset = fetch_california_housing()
train, target = pd.DataFrame(dataset.data), pd.DataFrame(dataset.target)
train.columns = ['zero','one','two','three','four','five','six','seven']
train.insert(loc=len(train.columns), column='target', value=target)

#Randomly replace 40% of the first column with NaN values
column = train['zero']
missing_pct = int(column.size * 0.4)
i = [random.choice(range(column.shape[0])) for _ in range(missing_pct)]
column[i] = np.NaN
train
```

Out[1]:

	zero	one	two	three	four	five	six	seven	target
0	NaN	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	NaN	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	NaN	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697

Now we use the **KNN** algorithm to impute the missing values
In this case, using 2 neighbors

```python
In [2]: #Impute the values using scikit-learn KNNImputer Class
        #Install the KNNImputer pip package in the current Jupyter kernel
        import sys
        !{sys.executable} -m pip install --upgrade pip
        !{sys.executable} -m pip install missingpy
        from missingpy import KNNImputer
        #Replace missing feature values using K-Nearest Neighbors
        imputer = KNNImputer(n_neighbors=2, weights="uniform")
        imputer.fit_transform(train[['zero']])
        train['zero'] = imputer.transform(train[['zero']]).ravel()
        train
```

5	4.03680	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.65910	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	3.87794	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	3.87794	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.69120	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611
10	3.20310	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815
11	3.27050	52.0	4.772480	1.024523	1504.0	2.049046	37.85	-122.26	2.418
12	3.87794	52.0	5.322650	1.012821	1098.0	2.346154	37.85	-122.26	2.135
13	3.87794	52.0	4.000000	1.097701	345.0	1.982759	37.84	-122.26	1.913
14	3.87794	52.0	4.262903	1.009677	1212.0	1.954839	37.85	-122.26	1.592

Notes: KNN only works with numerical values

Till now, we discussed missing value treatment using kNNImputer for continuous variables. Below, we create a data frame with missing values in categorical variables. **For imputing missing values in categorical variables, we have to encode the categorical values into numeric values as kNNImputer works only for numeric variables.** We can perform this using a mapping of categories to numeric variables.

Other methods for Imputation

AWS Machine Learning - Handling Missing Data

Other Methods

- Interpolation / Extrapolation
  - Estimate values from other observations within the range of a discrete set of known data points
- Forward filling / Backward filling
  - Fill the missing value by filling it from the preceding value or the succeeding value
- Hot deck imputation
  - Randomly choosing the missing value from a set of related and similar variables