Lab 6:Pandas Data Cleaning

SURUTHI S

225229141

In [58]:
```
import pandas as pd
df = pd.read_csv("BTC-USD.csv")
df.head(10)

```

Out[58]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-02-28	37706.000000	45077.578125	37518.214844	38419.984375	38419.984375	187557375751
1	2022-03-07	38429.304688	42465.671875	37260.203125	37849.664063	37849.664063	175966999156
2	2022-03-14	37846.316406	42316.554688	37680.734375	41247.824219	41247.824219	184097042034
3	2022-03-21	41246.132813	46827.546875	40668.042969	46820.492188	46820.492188	188591889758
4	2022-03-28	46821.851563	48086.835938	44403.140625	46453.566406	46453.566406	223334181931
5	2022-04-04	46445.273438	47106.140625	42021.207031	42207.671875	42207.671875	188557001876
6	2022-04-11	42201.039063	42424.589844	39373.058594	39716.953125	39716.953125	174652159709
7	2022-04-18	39721.203125	42893.582031	38696.191406	39469.292969	39469.292969	184314843516
8	2022-04-25	39472.605469	40713.890625	37585.789063	38469.093750	38469.093750	216681007567
9	2022-05-02	38472.187500	39902.949219	33878.964844	34059.265625	34059.265625	239044762282

In [59]:
```
column_names = df.columns
print(column_names)
df.dtypes
for i in column_names:
    print("{} is unique : {}".format(i,df[i].is_unique))

```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
Date is unique : True
Open is unique : True
High is unique : True
Low is unique : True
Close is unique : True
Adj Close is unique : True
Volume is unique : True
```

In [60]:
```
df.index.values
```

Out[60]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53], dtype=int64)
```

In [61]:
```
0 in df.index.values
```

Out[61]: True

In [62]:
```
df.set_index("Date",inplace=True)

```

In [63]:
```
df.head()
```

Out[63]:

Date	Open	High	Low	Close	Adj Close	Volume
2022-02-28	37706.000000	45077.578125	37518.214844	38419.984375	38419.984375	187557375751
2022-03-07	38429.304688	42465.671875	37260.203125	37849.664063	37849.664063	175966999156
2022-03-14	37846.316406	42316.554688	37680.734375	41247.824219	41247.824219	184097042034
2022-03-21	41246.132813	46827.546875	40668.042969	46820.492188	46820.492188	188591889758
2022-03-28	46821.851563	48086.835938	44403.140625	46453.566406	46453.566406	223334181931

In [64]:
```
columns_to_drop = [column_names[i] for i in [2,3,4]]
```

In [65]:
```
df.drop(columns_to_drop, inplace=True, axis=1)
```

In [66]:
```
df.head()
```

Out[66]:

Date	Open	Adj Close	Volume
2022-02-28	37706.000000	38419.984375	187557375751
2022-03-07	38429.304688	37849.664063	175966999156
2022-03-14	37846.316406	41247.824219	184097042034
2022-03-21	41246.132813	46820.492188	188591889758
2022-03-28	46821.851563	46453.566406	223334181931

In [67]:
```
import numpy as np
```

In [68]:
```
df.iloc[6:8,:] = np.NaN
```

In [69]:
```
df.head(10)
```

Out[69]:

Date	Open	Adj Close	Volume
2022-02-28	37706.000000	38419.984375	1.875574e+11
2022-03-07	38429.304688	37849.664063	1.759670e+11
2022-03-14	37846.316406	41247.824219	1.840970e+11
2022-03-21	41246.132813	46820.492188	1.885919e+11
2022-03-28	46821.851563	46453.566406	2.233342e+11
2022-04-04	46445.273438	42207.671875	1.885570e+11
2022-04-11	NaN	NaN	NaN
2022-04-18	NaN	NaN	NaN
2022-04-25	39472.605469	38469.093750	2.166810e+11
2022-05-02	38472.187500	34059.265625	2.390448e+11

In [70]:
```
df['Volume'] = df['Volume'].fillna(' ')
df.head(10)
```

Out[70]:

Date	Open	Adj Close	Volume
2022-02-28	37706.000000	38419.984375	187557375751.0
2022-03-07	38429.304688	37849.664063	175966999156.0
2022-03-14	37846.316406	41247.824219	184097042034.0
2022-03-21	41246.132813	46820.492188	188591889758.0
2022-03-28	46821.851563	46453.566406	223334181931.0
2022-04-04	46445.273438	42207.671875	188557001876.0
2022-04-11	NaN	NaN	
2022-04-18	NaN	NaN	
2022-04-25	39472.605469	38469.093750	216681007567.0
2022-05-02	38472.187500	34059.265625	239044762282.0

In [71]:
```
df['Adj Close'] = df['Adj Close'].fillna(99)
df.head(10)
```

Out[71]:

Date	Open	Adj Close	Volume
2022-02-28	37706.000000	38419.984375	187557375751.0
2022-03-07	38429.304688	37849.664063	175966999156.0
2022-03-14	37846.316406	41247.824219	184097042034.0
2022-03-21	41246.132813	46820.492188	188591889758.0
2022-03-28	46821.851563	46453.566406	223334181931.0
2022-04-04	46445.273438	42207.671875	188557001876.0
2022-04-11	NaN	99.000000	
2022-04-18	NaN	99.000000	
2022-04-25	39472.605469	38469.093750	216681007567.0
2022-05-02	38472.187500	34059.265625	239044762282.0

In [72]:
```
df['Open'] = df['Open'].fillna(df['Open'].mean())
df.head(10)

```

Out[72]:

Date	Open	Adj Close	Volume
2022-02-28	37706.000000	38419.984375	187557375751.0
2022-03-07	38429.304688	37849.664063	175966999156.0
2022-03-14	37846.316406	41247.824219	184097042034.0
2022-03-21	41246.132813	46820.492188	188591889758.0
2022-03-28	46821.851563	46453.566406	223334181931.0
2022-04-04	46445.273438	42207.671875	188557001876.0
2022-04-11	24583.493277	99.000000	
2022-04-18	24583.493277	99.000000	
2022-04-25	39472.605469	38469.093750	216681007567.0
2022-05-02	38472.187500	34059.265625	239044762282.0

In [73]:
```
df1 = pd.DataFrame(data={'col1':[np.nan,np.nan,2,3,4,np.nan,np.nan]})

```

In [74]:
```
df1.fillna(method='pad', limit=1)

```

Out[74]:

	col1
0	NaN
1	NaN
2	2.0
3	3.0
4	4.0
5	4.0
6	NaN

In [75]:
```
df1.fillna(method='pad', limit=1)

```

Out[75]:

	col1
0	NaN
1	NaN
2	2.0
3	3.0
4	4.0
5	4.0
6	NaN

In [76]:
```
df1.fillna(method = 'bfill')

```

Out[76]:

	col1
0	2.0
1	2.0
2	2.0
3	3.0
4	4.0
5	NaN
6	NaN

In [77]:
```
df1.dropna()
```

Out[77]:

	col1
2	2.0
3	3.0
4	4.0

In [78]:
```
df1.dropna(axis=1)

```

Out[78]:

0
1
2
3
4
5
6

In [79]:
```
df1.dropna(thresh=int(df1.shape[0] * .9), axis=1)
```

Out[79]:

0
1
2
3
4
5
6

In [84]:
```
np.where(df.Open>35000, 1, 0)
```

Out[84]:
```
array([1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [86]:
```
df2 = pd.DataFrame(data={'col1':np.random.randint(0, 10, 10),
'col2':np.random.randint(-10, 10, 10)})
```

In [87]:
```
df2
```

Out[87]:

	col1	col2
0	7	5
1	0	4
2	1	-6
3	3	6
4	1	-6
5	6	9
6	2	2
7	1	2
8	2	5
9	5	4

In [88]:
```
assert(df['col1'] >= 0 ).all() # Should return nothing
```

In [89]:
```
assert(df['col1'] != str).any() # Should return nothing
```

In [92]:
```
import pandas.util.testing as tm
tm.assert_series_equal(df['col1'], df['col2'])
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [92], in <cell line: 2>()
      1 import pandas.util.testing as tm
----> 2 tm.assert_series_equal(df['col1'], df['col2'])

    [... skipping hidden 1 frame]

File ~\anaconda3\lib\site-packages\pandas\_libs\testing.pyx:52, in pandas._libs.testing.assert_almost_equal()

File ~\anaconda3\lib\site-packages\pandas\_libs\testing.pyx:167, in pandas._libs.testing.assert_almost_equal()

File ~\anaconda3\lib\site-packages\pandas\_testing\asserters.py:682, in raise_assert_detail(obj, message, left,
right, diff, index_values)
    679 if diff is not None:
    680     msg += f"\n[diff]: {diff}"
--> 682 raise AssertionError(msg)

AssertionError: Series are different

Series values are different (100.0 %)
[index]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[left]:  [5, 2, 0, 3, 8, 2, 5, 9, 1, 0]
[right]: [2, -9, -6, -3, 0, -3, -4, -4, -5, 7]
```

In [97]:
```
! pip install beautifier
```

```
Collecting beautifier
  Downloading beautifier-0.5.5.tar.gz (19 kB)
Building wheels for collected packages: beautifier
  Building wheel for beautifier (setup.py): started
  Building wheel for beautifier (setup.py): finished with status 'done'
  Created wheel for beautifier: filename=beautifier-0.5.5-py3-none-any.whl size=19302 sha256=08daee28c20b27f838
09f4765df523a4db17b179f29c4384c830d62d37a89c62
  Stored in directory: c:\users\suruthi s\appdata\local\pip\cache\wheels\a2\5e\d4\91e4866f2a5e6a891676692a10efb
00ad61b1e41614e910d21
Successfully built beautifier
Installing collected packages: beautifier
Successfully installed beautifier-0.5.5
```

In [98]:
```
import beautifier
```

In [ ]:
```
print(email.username)
print(email.is_free_email)
Output:
bar.com
foo
False

url = Url(url_string)
print(url.param)
```

In [102]:
```
from beautifier import Email, Url
email_string = 'foo@bar.com'
email = Email(email_string)
print(email.domain)
print(email.username)
print(email.is_free_email)

url_string ='https://github.com/labtocat/beautifier/blob/master/beautifier/ init .py'
url = Url(url_string)
print(url.param)
print(url.username)
print(url.domain)
```

```
bar.com
foo
False
None
{'msg': 'feature is currently available only with linkedin urls'}
github.com
```

In [104]:
```
! pip install ftfy
```

```
Collecting ftfy
  Downloading ftfy-6.1.1-py3-none-any.whl (53 kB)
Requirement already satisfied: wcwidth>=0.2.5 in c:\users\suruthi s\anaconda3\lib\site-packages (from ftfy) (0.
2.5)
Installing collected packages: ftfy
Successfully installed ftfy-6.1.1
```

In [105]:
```
import ftfy
foo = ' ¯\\_(ã\x83\x84)_/ ¯'
bar = '\ufeffParty'
baz = '\001\033[36;44mI’m'
print(ftfy.fix_text(foo))
print(ftfy.fix_text(bar))
print(ftfy.fix_text(baz))
```

```
 ¯\_(ツ)_/ ¯
Party
I'm
```

Lab6. Pandas Data Cleaning Part-II

In [106]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

In [107]:
```
le = LabelEncoder()
df = pd.DataFrame(data = {'col1': ['foo','bar','foo','bar'], 'col2': ['x', 'y', 'x', 'z'], 'col3':[1,2,3,4]})
```

In [108]:
```
df.apply(le.fit_transform)
```

Out[108]:

	col1	col2	col3
0	1	0	0
1	0	1	1
2	1	0	2
3	0	2	3

In [109]:
```
import pandas as pd
df = pd.DataFrame({'A': ['a','b','a'], 'B': ['b','a','c'], 'C': [1, 2, 3]})
df
```

Out[109]:

	A	B	C
0	a	b	1
1	b	a	2
2	a	c	3

In [110]:
```
pd.get_dummies(df, prefix=['col1','col2'])
```

Out[110]:

	C	col1_a	col1_b	col2_a	col2_b	col2_c
0	1	1	0	0	1	0
1	2	0	1	1	0	0
2	3	1	0	0	0	1

In [111]:
```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler(feature_range = (0,1)) # (0,1 ) is default range
df2 = pd.DataFrame({'col1':[5,-41, -67],
 'col2': [23, - 53, -36],
 'col3': [-25,10, 17]})
mm_scaler.fit_transform(df2)
```

Out[111]:
```
array([[1.        , 1.        , 0.        ],
       [0.36111111, 0.        , 0.83333333],
       [0.        , 0.22368421, 1.        ]])
```

In [112]:
```
from sklearn.preprocessing import Binarizer
dfb = pd.DataFrame({'col1': [110, 200],
 'col2': [120, 800],
 'col3': [310, 400]})
bin = Binarizer(threshold=300)
bin.fit_transform(dfb)
```

Out[112]:
```
array([[0, 0, 1],
       [0, 1, 1]], dtype=int64)
```

In [113]:
```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
df = pd.DataFrame({'col1': [7, 2, 3],
 'col2': [4, np.nan, 6],
 'col3': [np.nan, np.nan, 3],
 'col4': [10, np.nan, 9]})
print(df)
imp_mean.fit_transform(df)
```

```
   col1  col2  col3  col4
0     7   4.0   NaN  10.0
1     2   NaN   NaN   NaN
2     3   6.0   3.0   9.0
```

Out[113]:
```
array([[ 7. ,  4. ,  3. , 10. ],
       [ 2. ,  5. ,  3. ,  9.5],
       [ 3. ,  6. ,  3. ,  9. ]])
```

In [115]:
```
! pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0
```

In [116]:
```
import warnings
warnings.filterwarnings('ignore')
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
a = 'Welcome to Bishop Heber College'
b = 'the trainable argument in the Embedding layer is used to specify whether the weights of the layer should
ratio = fuzz.ratio(a, b)
weighted_ratio = fuzz.WRatio(a, b)
unicode_ratio = fuzz.UQRatio(a, b)
print('Ratio =', ratio)
print('Weighted ratio =', weighted_ratio)
print('Unicode ratio =', unicode_ratio)

```

```
Ratio = 21
Weighted ratio = 86
Unicode ratio = 22
```

In [117]:
```
c = a + b
```

In [118]:
```
ex_tract = process.extract('I', c)
ex_tract
```

Out[118]:
```
[('i', 100), ('i', 100), ('i', 100), ('i', 100), ('i', 100)]
```

In [119]:
```
ex_tract_1 = process.extractOne('I', c)
ex_tract_1
```

Out[119]:
```
('i', 100)
```

In [ ]:
```

```