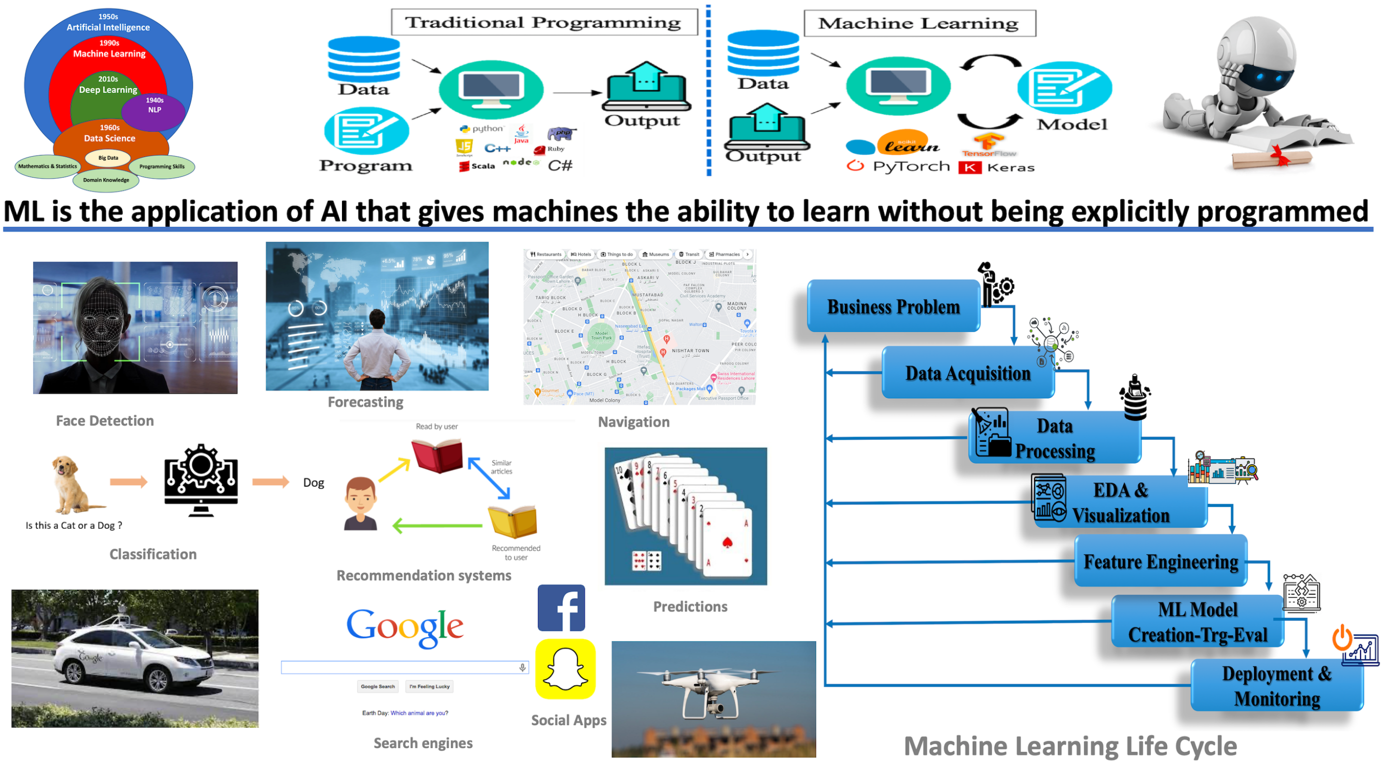
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.11 (Data Preprocessing: Extracting and Combining Information)

Open in Colab
(https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)



ML is the application of AI that gives machines the ability to learn without being explicitly programmed

In [ ]:

```

```

```
In [ ]:
```

Learning agenda of this notebook

- Overview of Data Pre-Processing and Feature Engineering
  - Feature Extraction
    - Extracting information from Student ID
    - Extracting information from a DateTime Column
  - Merging Features

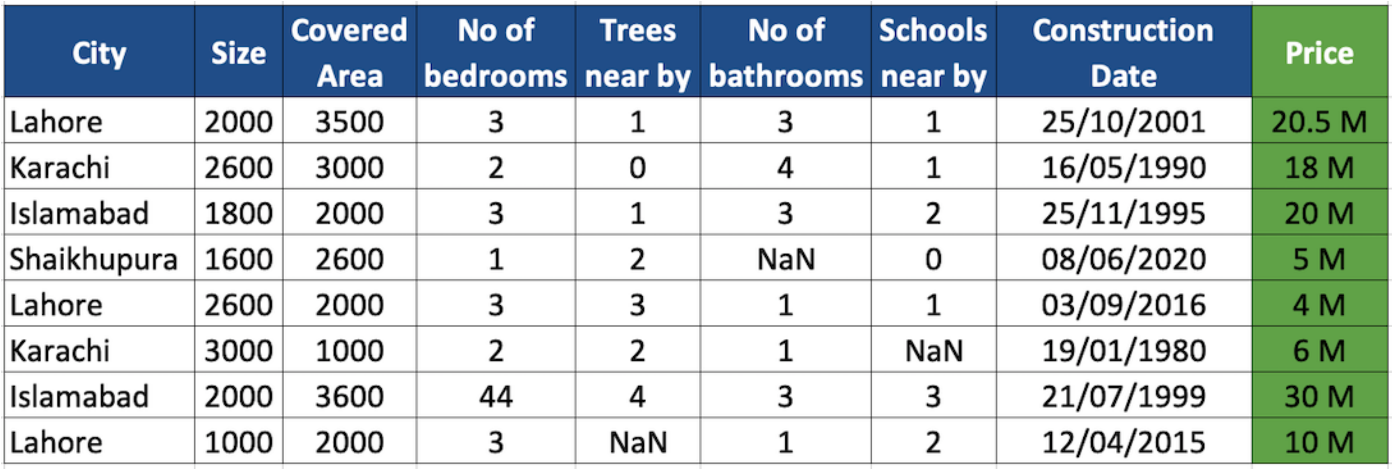
```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

1. Overview of Data Pre-Processing and Feature Engineering

- `Data Preprocessing` involves actions that we need to perform on the dataset in order to make it ready to be fed to the machine learning model.
- `Feature Engineering` is the process of using domain knowledge to extract features from raw data via data mining techniques.



City	Size	Covered Area	No of bedrooms	Trees near by	No of bathrooms	Schools near by	Construction Date	Price
Lahore	2000	3500	3	1	3	1	25/10/2001	20.5 M
Karachi	2600	3000	2	0	4	1	16/05/1990	18 M
Islamabad	1800	2000	3	1	3	2	25/11/1995	20 M
Shaikhupura	1600	2600	1	2	NaN	0	08/06/2020	5 M
Lahore	2600	2000	3	3	1	1	03/09/2016	4 M
Karachi	3000	1000	2	2	1	NaN	19/01/1980	6 M
Islamabad	2000	3600	44	4	3	3	21/07/1999	30 M
Lahore	1000	2000	3	NaN	1	2	12/04/2015	10 M

- Pre-processing package of sklearn provides a bundle of utility functions and transformer classes for data preprocessing (will cover later).
  - **Detecting and handling outliers**

        - Univariate (Z-Score, IQR, Percentiles)
        - Multivariate Analysis (Depth-based, Distance-based, Density-based methods)
        - Trimming, Capping/Winsorization, Discritization
    - **Missing values Imputation**
        - Univariate Imputation (Panda's `fillna()` method, Sklearn's `SimpleImputer()` transformer)
        - Multivariate Imputation (Sklearn's `IterativeImputer()` and `KNNImputer()` transformers)
    - **Encoding Categorical Features**
        - Encode Nominal i/p features using Pandas `get_dummies()` and Scikit-Learn's `OneHotEncoder()`
        - Encode Ordinal i/p features using Scikit-Learn's `OrdinalEncoder()`
        - Encode Ordinal o/p label using Scikit-Learn's `LabelEncoder()`
    - **Feature Scaling**
        - Use numPy to perform maxabs, minmax, standard and robust scaling
        - Use Sklearn's `MaxAbsScalar` , `MinMaxScalar` , `StandardScalar` , `RobustScalar` transformers
    - **Extracting Information**
        - Use Sklearn's `CountVectorizer` , `DictVectorizer` , `TfidfVectorizer` , and `TfidfTransformer`
    - **Combining Information**
        - Use `FeatureUnion` , `Pipeline` , `PCA`

In [ ]:

```

```

In [ ]:

```

```

2. Extracting Information from Student ID

In [1]:

```python
import pandas as pd
import numpy as np
df = pd.read_csv('datasets/extracting-data.csv')
df.head()
```

Out[1]:

	id	cgpa	scholarship
0	BDSF22M512	3.69	yes
1	BSEF19M025	2.50	no
2	BCSF19A541	3.80	yes
3	BDSF22M511	2.60	no
4	BITF21A012	3.00	no

In [ ]:

```python

```

In [2]:

```python
type(df['id'])
```

Out[2]:

```
pandas.core.series.Series
```

In [3]:

```python
df['id']
```

Out[3]:

```
0     BDSF22M512
1     BSEF19M025
2     BCSF19A541
3     BDSF22M511
4     BITF21A012
5     BSEF21M521
6     BSEF22M028
7     BDSF22A519
8     BSEF20M020
9     BDSF22M521
10    BITF19M026
11    BSEF20M012
12    BDSF22M507
13    BDSF22A525
14    BCSF21A014
15    BCSF19M527
Name: id, dtype: object
```

In [4]:

```python
df['id'][0].upper()
```

Out[4]:

```
'BDSF22M512'
```

In [5]:

```
df['id'].str.upper()
```

Out[5]:

```
0     BDSF22M512
1     BSEF19M025
2     BCSF19A541
3     BDSF22M511
4     BITF21A012
5     BSEF21M521
6     BSEF22M028
7     BDSF22A519
8     BSEF20M020
9     BDSF22M521
10    BITF19M026
11    BSEF20M012
12    BDSF22M507
13    BDSF22A525
14    BCSF21A014
15    BCSF19M527
Name: id, dtype: object
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [6]:

```python
#Extract Degree
df['degree'] = df['id'].str[0:3:1]

#Extract Batch
df['batch'] = df['id'].str[3:6:1]

#Extract Session
df['session'] = df['id'].str[6:7:1]

#Extract Rollno
df['rollno'] = df['id'].str[7::1].astype(dtype=np.uint16)

#Extract Campus
df['campus'] = np.where(df['rollno']>500, 'new-campus', 'old-campus')
df
```

Out[6]:

	id	cgpa	scholarship	degree	batch	session	rollno	campus
0	BDSF22M512	3.69	yes	BDS	F22	M	512	new-campus
1	BSEF19M025	2.50	no	BSE	F19	M	25	old-campus
2	BCSF19A541	3.80	yes	BCS	F19	A	541	new-campus
3	BDSF22M511	2.60	no	BDS	F22	M	511	new-campus
4	BITF21A012	3.00	no	BIT	F21	A	12	old-campus
5	BSEF21M521	3.10	no	BSE	F21	M	521	new-campus
6	BSEF22M028	3.75	yes	BSE	F22	M	28	old-campus
7	BDSF22A519	3.79	yes	BDS	F22	A	519	new-campus
8	BSEF20M020	3.25	no	BSE	F20	M	20	old-campus
9	BDSF22M521	3.90	yes	BDS	F22	M	521	new-campus
10	BITF19M026	2.85	no	BIT	F19	M	26	old-campus
11	BSEF20M012	3.10	no	BSE	F20	M	12	old-campus
12	BDSF22M507	3.00	no	BDS	F22	M	507	new-campus
13	BDSF22A525	2.90	no	BDS	F22	A	525	new-campus
14	BCSF21A014	2.70	no	BCS	F21	A	14	old-campus
15	BCSF19M527	3.85	yes	BCS	F19	M	527	new-campus

> From the single `id` column we have created five new columns

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

3. Extracting Information from DateTime Feature

a. Load Dataset

- Stock Market Dataset related to Crypto-Currency (Bitcoin is a digital currency created in 2009)
- Ethrium is a transactional token that facilitate operations on Ethrium Network and uses BlockChain development to replace storage of consumer data.

In [7]:

```
import numpy as np
import pandas as pd
import datetime
df = pd.read_csv('datasets/cryptodata.csv')
df.head(10)
```

Out[7]:

	Date	Symbol	Open	High	Low	Close	Volume
0	2020-03-13 08-PM	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
1	2020-03-13 07-PM	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2	a	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
3	2020-03-13 05-PM	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
4	2020-03-13 04-PM	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
5	2020-03-13 03-PM	ETHUSD	128.39	128.90	116.06	124.85	7378976.00
6	2020-03-13 02-PM	ETHUSD	134.03	137.90	125.50	128.39	3733916.89
7	2020-03-13 01-PM	ETHUSD	131.35	140.95	128.99	134.03	9582732.93
8	2020-03-13 12-PM	ETHUSD	128.93	134.60	126.95	131.35	3906590.52
9	2020-03-13 11-AM	ETHUSD	132.60	133.17	126.01	128.93	3311080.29

Recap of Python's Built-in `time` Module:

In [8]:

```
import time
```

In [9]:

```
print(dir(time))
```

```
['_STRUCT_TM_ITEMS', '__doc__', '__loader__', '__name__', '__package_
_', '__spec__', 'altzone', 'asctime', 'ctime', 'daylight', 'get_clock_
info', 'gmtime', 'localtime', 'mktime', 'monotonic', 'monotonic_ns',
'perf_counter', 'perf_counter_ns', 'process_time', 'process_time_ns',
'sleep', 'strftime', 'strptime', 'struct_time', 'time', 'time_ns', 'ti
mezone', 'tzname', 'tzset']
```

In [10]:

```
# Number of seconds passed since UNIX epoch
# Midnight January 01, 1970
time.time()
```

Out[10]:

```
1686105755.124831
```

In [11]:

```
time.ctime(1685236161.0406349)
```

Out[11]:

```
'Sun May 28 06:09:21 2023'
```

In [12]:

```
time.ctime(time.time())
```

Out[12]:

```
'Wed Jun  7 07:42:35 2023'
```

In [13]:

```
time.ctime(0)
```

Out[13]:

```
'Thu Jan  1 05:00:00 1970'
```

In [ ]:

```

```

In [ ]:

```

```

Recap of Python's Built-in `datetime` Module:

In [14]:

```
import datetime
```

In [15]:

```
print(dir(datetime))
```

```
['MAXYEAR', 'MINYEAR', '__all__', '__builtins__', '__cached__', '__doc
__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'date', 'datetime', 'datetime_CAPI', 'sys', 'time', 'timedelta', 'time
zone', 'tzinfo']
```

In [16]:

```
datetime.datetime(2023,1,25)
```

Out[16]:

```
datetime.datetime(2023, 1, 25, 0, 0)
```

In [ ]:

```

```

In [ ]:

```

```

In [17]:

```
df
```

Out[17]:

	Date	Symbol	Open	High	Low	Close	Volume
0	2020-03-13 08-PM	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
1	2020-03-13 07-PM	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2	a	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
3	2020-03-13 05-PM	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
4	2020-03-13 04-PM	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...	...	...	...	...	...	...	...
23669	2017-07-01 03-PM	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
23670	2017-07-01 02-PM	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
23671	2017-07-01 01-PM	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
23672	2017-07-01 12-PM	ETHUSD	275.01	275.01	271.00	274.83	824362.87
23673	2017-07-01 11-AM	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 7 columns

In [18]:

```
df.dtypes
```

Out[18]:

```
Date          object
Symbol        object
Open         float64
High         float64
Low          float64
Close        float64
Volume       float64
dtype: object
```

In [19]:

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

Out[19]:

```
'2020-03-13 08-PM'
```

In [ ]:

```

```

b. Convert the Datatype of Date Column to Datetime

- Pandas `pd.to_datetime()` method is used to convert its only required argument `arg` to a Timestamp object.

  ```
  pd.to_datetime(arg, format=None, errors='raise',)
  ```

- Where,
  - `arg` can be a string, Series, int, datetime, list, tuple, 1-d array, DataFrame/dict-like object to convert
  - `errors` {'ignore', 'raise', 'coerce'}, default 'raise'
    - If `raise` , then invalid parsing will raise an exception.
    - If `coerce` , then invalid parsing will be set as NaT.
    - If `ignore` , then invalid parsing will return the input
  - `format` : Used if the `arg` is not in the format as expected by the method

> For details of datetime formats visit: https://pandas.pydata.org/docs/reference/api/pandas.Period.strftime.html (https://pandas.pydata.org/docs/reference/api/pandas.Period.strftime.html)

Pass an appropriate format string to the format argument of the pd.to_datetime() method. The format string need to be prepared as per the string date format. Visit this link to see for Format codes:

> Let us pass this column/series to the `pd.to_datetime()` method to convert the datatype to `datetime64`

`pd.Timestamp` Attributes and Methods

`Series.dt.year` : Returns the year of datetime object

`Series.dt.month` : Returns month as January=1, December=12

`Series.dt.month_name()` : Returns month as string

`Series.dt.day` : Returns day of the month

`Series.dt.hour` : Returns hours

`Series.dt.minute` : Returns minutes

`Series.dt.second` : Returns seconds

`Series.dt.dayofweek` : Returns number representing the day

`Series.dt.day_name()` : Returns name of the day as string (Sunday being 0)

Convert a String date to a datetime object:

In [20]:

```
# Let's try to convert 6 March 2022 from string to datetime object
pd.to_datetime('06-03-2022')
```

Out[20]:

```
Timestamp('2022-06-03 00:00:00')
```

In [21]:

```
pd.to_datetime('06-03-2022').month
```

Out[21]:

```
6
```

> `pd.to_datetime()` method expects the string date as `month-day-year` , while we in Pakistan normally use `day-month-year`

In [22]:

```
pd.to_datetime('06-03-2022', format='%d-%m-%Y')
```

Out[22]:

```
Timestamp('2022-03-06 00:00:00')
```

In [23]:

```
pd.to_datetime('06-03-2022', format='%d-%m-%Y').month
```

Out[23]:

```
3
```

In [ ]:

```

```

Convert a String datetime to a datetime object:

In [24]:

```
pd.to_datetime('06-03-2022 08-AM', format='%d-%m-%Y %I-%p')
```

Out[24]:

```
Timestamp('2022-03-06 08:00:00')
```

In [ ]:

```

```

What Happens when the string is not a valid datetime string:

- Use of argument `errors` {‘ignore’, ‘raise’, ‘coerce’}, default ‘raise’

- If `raise` , then invalid parsing will raise an exception.
- If `coerce` , then invalid parsing will be set as NaT.
- If `ignore` , then invalid parsing will return the input

In [25]:

```
#pd.to_datetime('06-03-2022 0aa8-AM', format='%d-%m-%Y %I-%p', errors='raise')
```

In [26]:

```
pd.to_datetime('06-03-2022 0aa8-AM', format='%d-%m-%Y %I-%p', errors='ignore')
```

Out[26]:

```
'06-03-2022 0aa8-AM'
```

In [27]:

```
pd.to_datetime('06-03-2022 0aa8-AM', format='%d-%m-%Y %I-%p', errors='coerce')
```

Out[27]:

```
NaT
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Convert data type of Datetime Column of Crypto Dataset from String to Datetime:

In [28]:

```
df
```

Out[28]:

	Date	Symbol	Open	High	Low	Close	Volume
0	2020-03-13 08-PM	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
1	2020-03-13 07-PM	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2	a	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
3	2020-03-13 05-PM	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
4	2020-03-13 04-PM	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...	...	...	...	...	...	...	...
23669	2017-07-01 03-PM	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
23670	2017-07-01 02-PM	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
23671	2017-07-01 01-PM	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
23672	2017-07-01 12-PM	ETHUSD	275.01	275.01	271.00	274.83	824362.87
23673	2017-07-01 11-AM	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 7 columns

In [29]:

```
# Change the datatype of Date column from string to Datetime object
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d %I-%p', errors='coerce'
df
```

Out[29]:

	Date	Symbol	Open	High	Low	Close	Volume
0	2020-03-13 20:00:00	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
1	2020-03-13 19:00:00	ETHUSD	119.51	132.02	117.10	129.94	7579741.09
2	NaT	ETHUSD	124.47	124.85	115.50	119.51	4898735.81
3	2020-03-13 17:00:00	ETHUSD	124.08	127.42	121.63	124.47	2753450.92
4	2020-03-13 16:00:00	ETHUSD	124.85	129.51	120.17	124.08	4461424.71
...	...	...	...	...	...	...	...
23669	2017-07-01 15:00:00	ETHUSD	265.74	272.74	265.00	272.57	1500282.55
23670	2017-07-01 14:00:00	ETHUSD	268.79	269.90	265.00	265.74	1702536.85
23671	2017-07-01 13:00:00	ETHUSD	274.83	274.93	265.00	268.79	3010787.99
23672	2017-07-01 12:00:00	ETHUSD	275.01	275.01	271.00	274.83	824362.87
23673	2017-07-01 11:00:00	ETHUSD	279.98	279.99	272.10	275.01	679358.87

23674 rows × 7 columns

In [30]:

```
# Verify
df.dtypes
```

Out[30]:

```
Date      datetime64[ns]
Symbol            object
Open             float64
High             float64
Low              float64
Close            float64
Volume           float64
dtype: object
```

In [ ]:

```

```

Accessing/Extracting Information from a Datetime Object

In [31]:

```
df['Date'][0]
```

Out[31]:

```
Timestamp('2020-03-13 20:00:00')
```

In [32]:

```
df['Date'][0].year
```

Out[32]:

```
2020
```

In [33]:

```
df['Date'][0].month
```

Out[33]:

```
3
```

In [34]:

```
df['Date'][0].day
```

Out[34]:

```
13
```

In [35]:

```
df['Date'][0].hour
```

Out[35]:

```
20
```

In [36]:

```
df['Date'][0].dayofweek
```

Out[36]:

```
4
```

In [37]:

```
df['Date'][0].day_name()
```

Out[37]:

```
'Friday'
```

In [38]:

```
df['Date'][0].month_name()
```

Out[38]:

```
'March'
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Accessing/Extracting Information from a Datetime Series Object

In [39]:

```
#df['Date'].year
```

In [40]:

```
df['Date'].dt.year
```

Out[40]:

```
0        2020.0
1        2020.0
2           NaN
3        2020.0
4        2020.0
          ...  
23669    2017.0
23670    2017.0
23671    2017.0
23672    2017.0
23673    2017.0
Name: Date, Length: 23674, dtype: float64
```

In [41]:

```
df['Date'].dt.day_name()
```

Out[41]:

```
0          Friday
1          Friday
2             NaN
3          Friday
4          Friday
           ...   
23669    Saturday
23670    Saturday
23671    Saturday
23672    Saturday
23673    Saturday
Name: Date, Length: 23674, dtype: object
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Add Seven additional Columns in the Dataframe by Extracting Information from a Date Column

In [42]:

```python
df.head(2)
```

Out[42]:

	Date	Symbol	Open	High	Low	Close	Volume
0	2020-03-13 20:00:00	ETHUSD	129.94	131.82	126.87	128.71	1940673.93
1	2020-03-13 19:00:00	ETHUSD	119.51	132.02	117.10	129.94	7579741.09

In [43]:

```python
# Extract day of the month
df['day'] = df['Date'].dt.day
# Extract Month
df['month'] = df['Date'].dt.month
# Extract Year
df['year'] = df['Date'].dt.year
# Extract month name
df['month_name'] = df['Date'].dt.month_name()
# day of week - name
df['dow_name'] = df['Date'].dt.day_name()
# is weekend?
df['date_is_weekend'] = np.where(df['dow_name'].isin(['Saturday', 'Sunday']), 1,

df.head()
```

Out[43]:

	Date	Symbol	Open	High	Low	Close	Volume	day	month	year	month_na
0	2020-03-13 20:00:00	ETHUSD	129.94	131.82	126.87	128.71	1940673.93	13.0	3.0	2020.0	Ma
1	2020-03-13 19:00:00	ETHUSD	119.51	132.02	117.10	129.94	7579741.09	13.0	3.0	2020.0	Ma
2	NaT	ETHUSD	124.47	124.85	115.50	119.51	4898735.81	NaN	NaN	NaN	N
3	2020-03-13 17:00:00	ETHUSD	124.08	127.42	121.63	124.47	2753450.92	13.0	3.0	2020.0	Ma
4	2020-03-13 16:00:00	ETHUSD	124.85	129.51	120.17	124.08	4461424.71	13.0	3.0	2020.0	Ma

In [ ]:

```python

```

In [ ]:

```python

```

Let us find the oldest and newest record in the dataframe

In [44]:

```
df['Date'].max()
```

Out[44]:

```
Timestamp('2020-03-13 20:00:00')
```

In [45]:

```
df['Date'].min()
```

Out[45]:

```
Timestamp('2017-07-01 11:00:00')
```

In [46]:

```
df['Date'].max() - df['Date'].min()
```

Out[46]:

```
Timedelta('986 days 09:00:00')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

4. Combining Information from multiple columns to a single column

In [47]:

```
# Import the dataset using sklearn built-in `titanic dataset`
import numpy as np
import pandas as pd
from sklearn import datasets

titanic = datasets.fetch_openml(name='titanic', version=1)
df = pd.DataFrame(titanic.data, columns=titanic.feature_names)
df['target'] = titanic.target
df
```

Out[47]:

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	b
0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	
1	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	
2	1.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
3	1.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
4	1.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	N
...	...	...	...	...	...	...	...	...	...	...	
1304	3.0	Zabour, Miss. Hileni	female	14.5000	1.0	0.0	2665	14.4542	None	C	N
1305	3.0	Zabour, Miss. Thamine	female	NaN	1.0	0.0	2665	14.4542	None	C	N
1306	3.0	Zakarian, Mr. Mapriededer	male	26.5000	0.0	0.0	2656	7.2250	None	C	N
1307	3.0	Zakarian, Mr. Ortin	male	27.0000	0.0	0.0	2670	7.2250	None	C	N
1308	3.0	Zimmerman, Mr. Leo	male	29.0000	0.0	0.0	315082	7.8750	None	S	N

1309 rows × 14 columns

VARIABLE DESCRIPTIONS

`Pclass` -> Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd).

`survival` -> Survival (0 = No; 1 = Yes)

`name` -> Name

`sex` -> Sex
`age` -> Age
`sibsp` -> Number of Siblings/Spouses Aboard
`parch` -> Number of Parents/Children Aboard
`ticket` -> Ticket Number
`fare` -> Passenger Fare (British pound)
`cabin` -> Cabin
`embarked` -> Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
`boat` -> Lifeboat
`body` -> Body Identification Number
`home.dest` -> Home/Destination

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Drop unnecessary Columns

In [48]:

```
df.drop(columns=['name','pclass','ticket','fare','embarked', 'boat', 'body','hom
df
```

Out[48]:

	sex	age	sibsp	parch
0	female	29.0000	0.0	0.0
1	male	0.9167	1.0	2.0
2	female	2.0000	1.0	2.0
3	male	30.0000	1.0	2.0
4	female	25.0000	1.0	2.0
...	...	...	...	...
1304	female	14.5000	1.0	0.0
1305	female	NaN	1.0	0.0
1306	male	26.5000	0.0	0.0
1307	male	27.0000	0.0	0.0
1308	male	29.0000	0.0	0.0

1309 rows × 4 columns

In [ ]:

```python

```

In [ ]:

```python

```

Add new Column/Feature

Both `sibsp` and `parch` relate to traveling with family. I'll combine these two variables/columns into one categorical variable, which represents if a person is traveling alone or not.

In [49]:

```python
df['travel_alone'] = np.where((df['sibsp']+df['parch'])>0, 1, 0)
```

In [50]:

```python
df.sample(5)
```

Out[50]:

	sex	age	sibsp	parch	travel_alone
623	female	6.0	4.0	2.0	1
364	male	19.0	0.0	0.0	0
1062	male	21.0	0.0	0.0	0
1179	male	NaN	1.0	9.0	1
525	male	23.0	0.0	0.0	0

In [51]:

```python
df.drop(['sibsp','parch'], axis=1, inplace=True)
df
```

Out[51]:

	sex	age	travel_alone
0	female	29.0000	0
1	male	0.9167	1
2	female	2.0000	1
3	male	30.0000	1
4	female	25.0000	1
...	...	...	...
1304	female	14.5000	1
1305	female	NaN	1
1306	male	26.5000	0
1307	male	27.0000	0
1308	male	29.0000	0

1309 rows × 3 columns

In [ ]:

```python

```