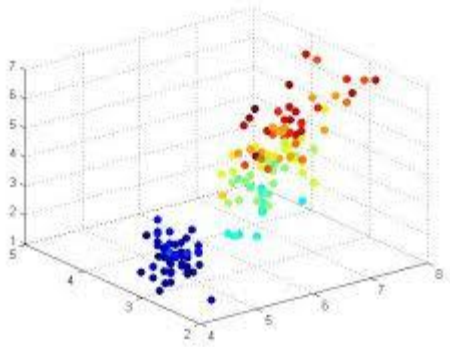
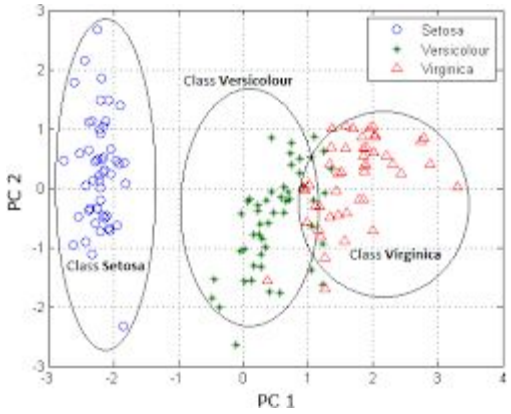
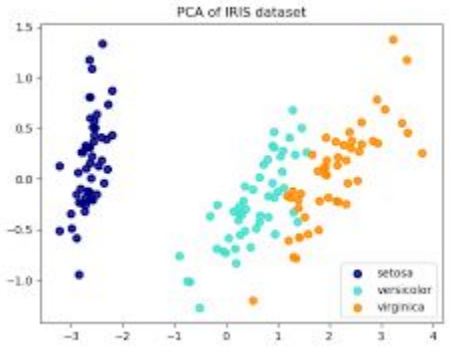
Iris dataset







Imputation of missing values

- Many real world datasets contain missing values, often encoded as blanks, NaNs or other placeholders.
- Such datasets however are incompatible with scikit-learn estimators which assume that all values in an array are numerical, and that all have and hold meaning.
- A basic strategy to use incomplete datasets is to discard entire rows and/or columns containing missing values. However, this comes at the price of losing data which may be valuable (even though incomplete).
- A better strategy is to impute the missing values, i.e., to infer them from the known part of the data.

Univariate vs. Multivariate Imputation

- One type of imputation algorithm is univariate, which imputes values in the i-th feature dimension using only non-missing values in that feature dimension .
- By contrast, multivariate imputation algorithms use the entire set of available feature dimensions to estimate the missing values

Imputation of missing values

INTERNSHIPSTUDIO

Univariate feature imputation

- The SimpleImputer class missing values with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
SimpleImputer()
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[4.          2.        ]
 [6.          3.666...]
 [7.          6.        ]]
```

scikit learn

Imputation of missing values

INTERNSHIPSTUDIO

Univariate feature imputation

- **The SimpleImputer** class also supports sparse matrices:

```
>>> import scipy.sparse as sp
>>> X = sp.csc_matrix([[1, 2], [0, -1], [8, 4]])
>>> imp = SimpleImputer(missing_values=-1, strategy='mean')
>>> imp.fit(X)
SimpleImputer(missing_values=-1)
>>> X_test = sp.csc_matrix([[-1, 2], [6, -1], [7, 6]])
>>> print(imp.transform(X_test).toarray())
[[3. 2.]
 [6. 3.]
 [7. 6.]]
```

Sparse matrix is a **matrix** which contains very few non-zero elements. When a **sparse matrix** is represented with a 2-dimensional array, we waste a lot of space to represent that **matrix**..

scikit learn

Imputation of missing values

Univariate feature imputation

- The **SimpleImputer** class also supports categorical data represented as string values or pandas categoricals when using the 'most_frequent' or 'constant' strategy:

```
>>> import pandas as pd
>>> df = pd.DataFrame([["a", "x"],
...                    [np.nan, "y"],
...                    ["a", np.nan],
...                    ["b", "y"]], dtype="category")
...
>>> imp = SimpleImputer(strategy="most_frequent")
>>> print(imp.fit_transform(df))
[['a' 'x']
 ['a' 'y']
 ['a' 'y']
 ['b' 'y']]
```

Imputation of missing values

INTERNSHIPSTUDIO

Multivariate feature imputation

- A more sophisticated approach is to use the IterativeImputer class, which models each feature with missing values as a function of other features, and uses that estimate for imputation.

```
>>> import numpy as np
>>> from sklearn.experimental import enable_iterative_imputer
>>> from sklearn.impute import IterativeImputer
>>> imp = IterativeImputer(max_iter=10, random_state=0)
>>> imp.fit([[1, 2], [3, 6], [4, 8], [np.nan, 3], [7, np.nan]])
IterativeImputer(random_state=0)
>>> X_test = [[np.nan, 2], [6, np.nan], [np.nan, 6]]
>>> # the model learns that the second feature is double the first
>>> print(np.round(imp.transform(X_test)))
[[ 1.  2.]
 [ 6. 12.]
 [ 3.  6.]]
```

scikit learn

INTERNSHIPSTUDIO

- Q.1 What is the need for Missing Value Imputation?
- Q.2 What is Univariate Imputation?
- Q.3 What is Multivariate Imputation?
- Q.4 What does SimpleImputer do ?

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is used to

- To give insight into a data set.
- Understand the underlying structure.
- Extract important parameters and relationships that hold between them.
- Test underlying assumptions.
  - How to ensure you are ready to use machine learning algorithms in a project?
  - How to choose the most suitable algorithms for your data set?
  - How to define the feature variables that can potentially be used for machine learning?

Exploratory Data Analysis (EDA) helps to answer all these questions, ensuring the best outcomes for the project. It is an approach for summarizing, visualizing, and becoming intimately familiar with the important characteristics of a data set.

Exploratory Data Analysis

INTERNSHIPSTUDIO

Understanding EDA using sample Data set

- Example- EDA on Wine Quality data set

```
import pandas as pd
df = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
df.head()
```

Running above script in jupyter notebook, will give output something like below –

```
import pandas as pd
df = pd.read_csv(r"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv", sep=';')
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56		

scikit learn

Exploratory Data Analysis

We can get the total number of rows and columns from the data set using ".shape" like below –

```
df.shape
```

```
df.shape
(1599, 12)
```

Use info() function- To find what all columns it contains, of what types and if they contain any value in it or not

```
df.info()
```

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Exploratory Data Analysis

describe() function- provides the count, mean, standard deviation, minimum and maximum values and the quantities of the data.

```
df.describe()
```

```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995680	3.210000	0.550000	9.500000	5
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8

In "quality" score scale, 1 comes at the bottom .i.e. poor and 10 comes at the top .i.e. best.
From above we can conclude, none of the observation score 1(poor), 2 and 9, 10(best) score. All the scores are between 3 to 8.

```
df.quality.unique()
```

```
df.quality.unique()
```

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

One-Hot Encoding

INTERNSHIPSTUDIO

- A one hot encoding is a representation of categorical variables as binary vectors.
- This requires that the categorical values be mapped to integer values.
- Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

If we had the sequence:

```
1 'red', 'red', 'green'
```

We could represent it with the integer encoding:

```
1 0, 0, 1
```

And the one hot encoding of:

```
1 [1, 0]
2 [1, 0]
3 [0, 1]
```

A one hot encoding allows the representation of categorical data to be more expressive.
Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.

One-Hot Encoding

One Hot Encode with scikit-learn

In this example, we will assume the case where you have an output sequence of the following 3 labels:

```
"cold"
"warm"
"hot"
```

An example sequence of 10 time steps may be:

```
cold, cold, warm, cold, hot, hot, warm, cold, warm, hot
```

This would first require an integer encoding, such as 1, 2, 3. This would be followed by a one hot encoding of integers to a binary vector with 3 values, such as [1, 0, 0].

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

- LabelEncoder of creating an integer encoding of labels
- OneHotEncoder for creating a one hot encoding of integer encoded values.

One-Hot Encoding


```python
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold', 'warm', 'hot
values = array(data)
print(values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)
# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
# invert first example
inverted = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
print(inverted)
```

```
['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']

[0 0 2 0 1 1 2 0 2 1]

[[ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 0.  1.  0.]]

['cold']
```

INTERNSHIPSTUDIO

- **Q.1 What is the role of EDA in Machine Learning?**
- **Q.2 What is One-Hot Encoding?**
- **Q.3 What is the role of One-Hot Encoding in Machine Learning?**
- **Q.4 df.describe() is used for?**
- **Q.5 df.head() is used for?**

scikit learn