EXPERIMENT-1

Introduction to Kaggle and how it can be used to enhance visibility:

In the ever-expanding realm of data science and machine learning, Kaggle has emerged as a powerful platform that brings together enthusiasts, professionals, and organizations to explore, collaborate, and compete in solving complex data challenges.

This article delves into the essence of Kaggle, unravelling its features, functionalities, and the diverse array of competitions it hosts.

Through examples of notable competitions, we'll showcase how Kaggle has become a driving force in pushing the boundaries of what is achievable in the field of data science.

What is Kaggle?

Kaggle, founded in 2010, is an online platform that serves as a hub for data science and machine learning enthusiasts.

Acquired by Google in 2017, Kaggle provides a collaborative environment where individuals and teams can access datasets, share code, and participate in machine learning competitions.

It has evolved into a vibrant community that fosters learning, innovation, and problem-solving in the field of data science.

Key Features of Kaggle:

There are few Key feature about Kaggle platform which are given below in short detail:

1. Datasets:

Kaggle offers a vast repository of datasets covering diverse domains.

Users can explore, analyze, and download datasets to work on their own projects or participate in Kaggle competitions.

2. Kernels:

Kaggle Kernels provide an interactive environment for writing and executing code in a variety of languages, including Python and R.

Kernels enable users to share code, analyses, and visualizations, fostering collaboration and learning.

3. Competitions:

Kaggle hosts a wide range of machine learning competitions that challenge participants to tackle real-world problems using provided datasets.

These competitions often come with cash prizes, job opportunities, and the chance to work on cutting-edge problems.

4. Discussions and Forums:

Kaggle's discussion forums allow users to ask questions, share insights, and engage in conversations with a global community of data scientists and machine learning practitioners.

5. Courses and Learning Resources:

Kaggle provides learning resources, including courses and tutorials, to help users enhance their skills in data science, machine learning, and related fields.

Why Kaggle is Required?

Kaggle is considered a valuable and necessary platform in the field of data science and machine learning for several compelling reasons:

1. Real-World Problem Solving:

- Kaggle hosts a variety of competitions that involve solving real-world problems.
- This provides participants with the opportunity to apply their data science and machine learning skills to practical scenarios, gaining hands-on experience.

2. Access to Diverse Datasets:

- Kaggle provides a vast repository of datasets across various domains.
- This allows data scientists to explore diverse datasets, ranging from finance and healthcare to image and text data, enhancing their ability to work on different types of projects.

3. Learning and Collaboration:

- Kaggle fosters a collaborative learning environment. Participants can explore and share code, insights, and best practices through kernels, discussions, and forums.
- This collaborative approach accelerates the learning process and exposes individuals to a wide range of techniques.

4. Benchmarking and Competition:

- Kaggle competitions serve as a benchmark for data science skills.
- By participating in competitions, individuals can assess and benchmark their abilities against a global community of data scientists, gaining insights into best practices and advanced techniques.

5. Community Engagement:

- Kaggle has a vibrant and active community of data scientists, researchers, and industry professionals.
- Engaging with this community allows individuals to network, seek advice, and collaborate on projects.
- The forums provide a platform for discussing challenges, solutions, and the latest developments in the field.

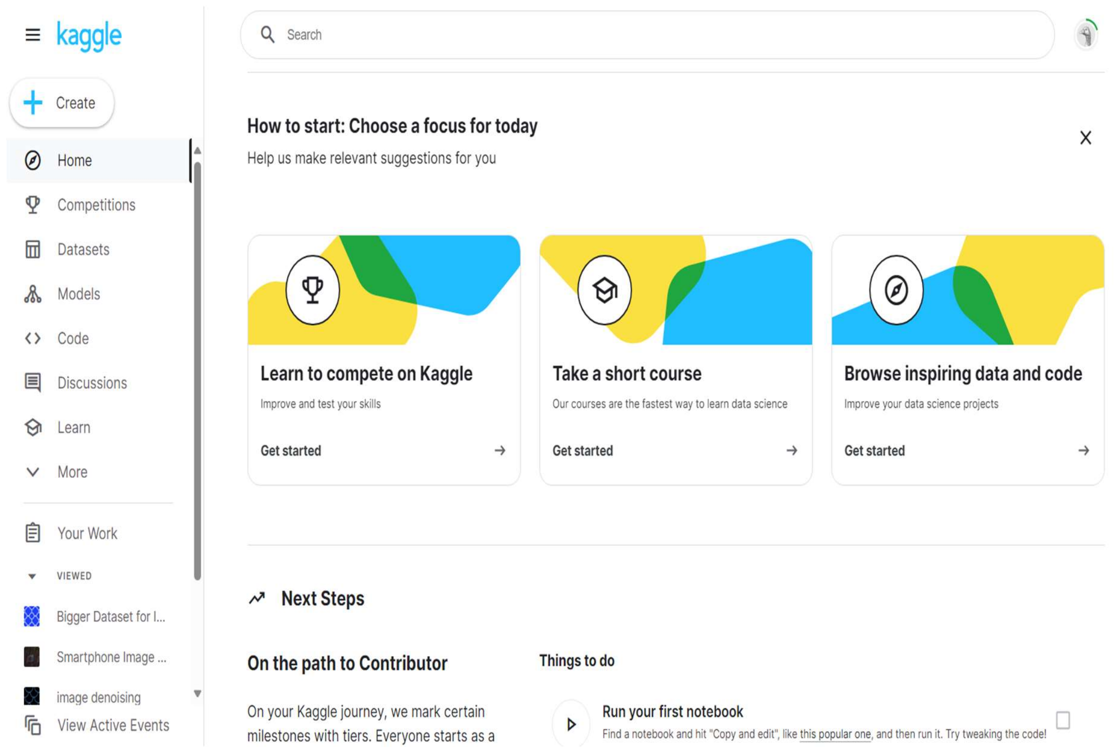
6. Career Opportunities:

- Success in Kaggle competitions can enhance one’s visibility within the data science community.

- Many companies value Kaggle achievements, and participating in competitions can open up job opportunities and collaborations with organizations seeking top-tier data science talent.

How Can Kaggle Enhance Visibility?

- **Skill Enhancement**: Kaggle offers a unique opportunity to acquire and master data science skills. Here's how you can use it effectively:
    - **Programming Languages**: Familiarize yourself with Python and R, as many Kaggle notebooks are written in these languages.
    - **Algorithms**: Understand different types of algorithms and their use-cases.
- **Data Science Competitions**:
    - Kaggle hosts various machine learning problems related to data science projects. By participating, you can:
        - **Solve Real-World Problems**: Work with accurate data and solve predictive modeling issues.
        - **Compete Effectively**: Submit your models and see how they perform on public leaderboards.
        - **Learn from Others**: Explore upvoted kernels and learn from winners' thinking processes[23].
- **Datasets**: Kaggle's extensive collection of datasets allows you to practice and apply theories in real-world scenarios. Access to quality data is crucial for skill development


kaggle
Search
Create
Home
Competitions
Datasets
Models
Code
Discussions
Learn
More
Your Work
VIEWED
Bigger Dataset for I...
Smartphone Image ...
image denoising
View Active Events
How to start: Choose a focus for today
Help us make relevant suggestions for you
Learn to compete on Kaggle
Improve and test your skills
Get started
Take a short course
Our courses are the fastest way to learn data science
Get started
Browse inspiring data and code
Improve your data science projects
Get started
Next Steps
On the path to Contributor
Things to do
Run your first notebook

On your Kaggle journey, we mark certain milestones with tiers. Everyone starts as a

Find a notebook and hit "Copy and edit", like this popular one, and then run it. Try tweaking the code!

EXPERIMENT-2

Program:

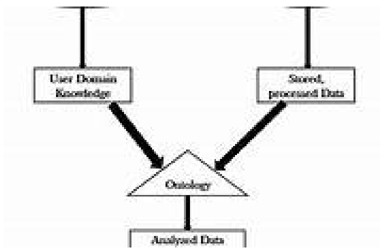
Implementation of program to remove outliers, missing values from the Iris dataset.

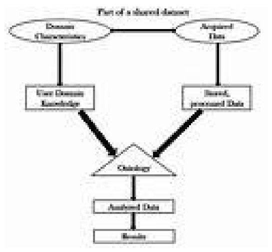
Platform Used: Google Colab.

Theory:

1. **Iris Dataset Overview**:
   - The **Iris dataset** is a well-known dataset in the field of pattern recognition and machine learning.
   - It contains measurements of various features from three different species of iris flowers: **Setosa**, **Versicolor**, and **Virginica**.
   - The features include:
     - **Sepal Length (Cm)**
     - **Sepal Width (Cm)**
     - **Petal Length (Cm)**
     - **Petal Width (Cm)**

   - The goal is to classify the iris flowers based on these features.





2. **Data Preprocessing Steps**:

- **Loading Data**:
    - The program starts by loading the Iris dataset using the Pandas library. The dataset is read from a CSV file named “Iris.csv”.
- **Removing ID Column**:
    - The first column (ID) is removed from the dataset since it does not contribute to the analysis.
- **Checking for Missing Values**:
    - The program checks if there are any missing values in the dataset using the isnull().sum() method.
    - Fortunately, there are no missing values in this dataset.
- **Removing Outliers**:
    - Outliers can affect model performance. The program uses the **Z-score** method to identify and remove outliers.
    - Entries with Z-scores greater than 3 (in absolute value) are considered outliers and filtered out.
- **Standardization**:
    - Standardization scales the features to have a mean of 0 and a standard deviation of 1.
    - The StandardScaler from Scikit-learn is used to standardize the numeric features (sepal and petal measurements).
- **Result**:
    - The cleaned Iris dataset is now ready for further analysis or model training.

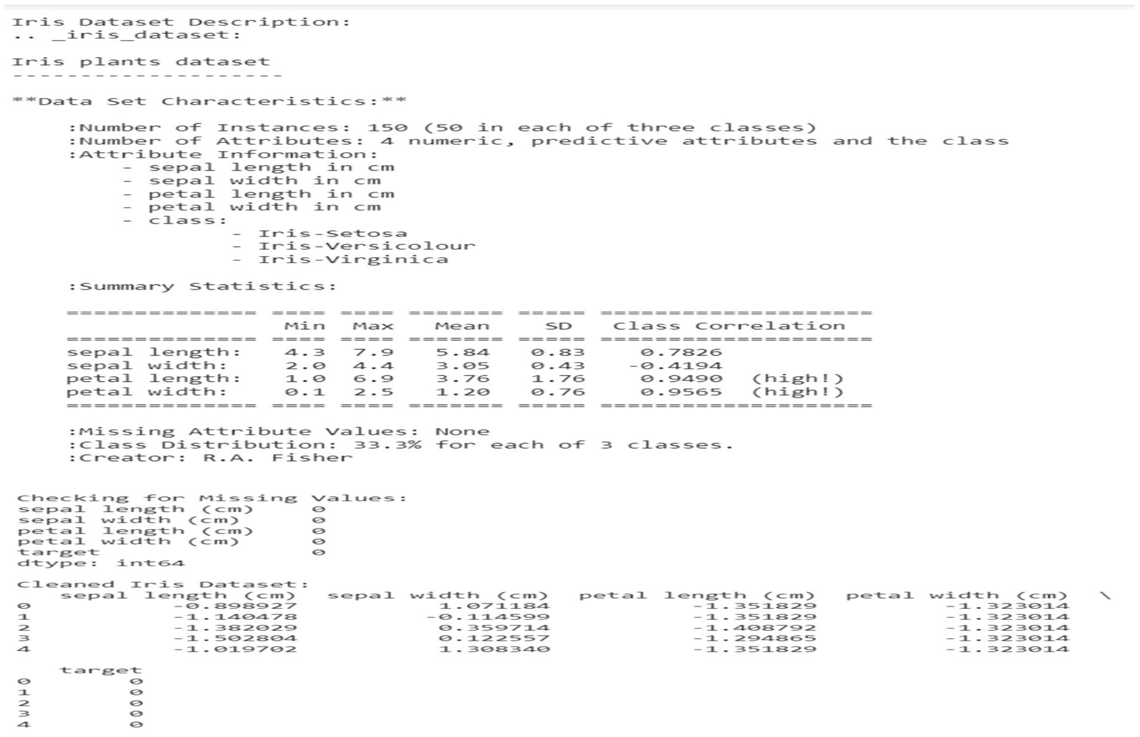
3. **Why Preprocessing Matters**:
    - Data preprocessing is crucial for several reasons:
        - **Quality**: Ensures data quality by handling missing values and outliers.
        - **Model Performance**: Improves model performance by standardizing features.
        - **Robustness**: Helps models handle noisy or irregular data.

Input:

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from scipy import stats
```

```
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
print("Iris Dataset Description:")
print(iris.DESCR)
print("\nChecking for Missing Values:")
print(iris_df.isnull().sum())
iris_df = iris_df.dropna()
z_scores = stats.zscore(iris_df.iloc[:, :-1])
abs_z_scores = abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
iris_df = iris_df[filtered_entries]
scaler = StandardScaler()
iris_df.iloc[:, :-1] = scaler.fit_transform(iris_df.iloc[:, :-1])
print("\nCleaned Iris Dataset:")
print(iris_df.head())
```

Output:



```
Iris Dataset Description:
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher


Checking for Missing Values:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64

Cleaned Iris Dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0          -0.898927          1.071184          -1.351829         -1.323014
1          -1.140478         -0.114599          -1.351829         -1.323014
2          -1.382029          0.359714          -1.408792         -1.323014
3          -1.502804          0.122557          -1.294865         -1.323014
4          -1.019702          1.308340          -1.351829         -1.323014

   target
0       0
1       0
2       0
3       0
4       0
```

EXPERIMENT-3

Program:

Implementation of AND and OR gates in python.

Platform Used: Google Colab.

Theory:

Here's a breakdown of how it works:

1. **perceptron_AND_OR Function**:
   - This function takes **inputs**, **weights**, **threshold**, and **learning_rate** as arguments.
   - It calculates the weighted sum of inputs multiplied by weights.
   - Based on the weighted sum compared to the threshold, it determines the output using a step function (activation function).
   - It calculates the error as the difference between the threshold and the output.
   - Then, it updates the weights using the perceptron learning rule: **weight = weight + learning_rate * error * input**.
   - Finally, it returns the output and the updated weights.
2. **Test for AND Gate**:
   - It initializes the inputs, threshold, learning rate, and weights for the AND gate.
   - It iterates through each input set for the AND gate.
   - For each input set, it calls the **perceptron_AND_OR** function to get the output and updated weights.
   - It prints the input set, output, and updated weights.
3. **Test for OR Gate**:
   - Similar to the test for the AND gate, but using inputs, threshold, learning rate, and weights specific to the OR gate.
4. **Theoretical Explanation**:
   - The perceptron model is a simple binary classifier that learns a linear decision boundary.
   - It calculates the weighted sum of inputs and compares it to a threshold to produce the output.
   - If the weighted sum is greater than or equal to the threshold, it outputs 1; otherwise, it outputs 0.

- The learning rule updates the weights based on the error, which is the difference between the desired output and the actual output.
- The learning rate controls the step size of weight updates and helps in controlling the convergence speed of the algorithm.

Overall, this program demonstrates how a perceptron can learn the weights to perform logical operations like AND and OR gates. It showcases the basic principles of supervised learning and the perceptron learning rule.

Input:

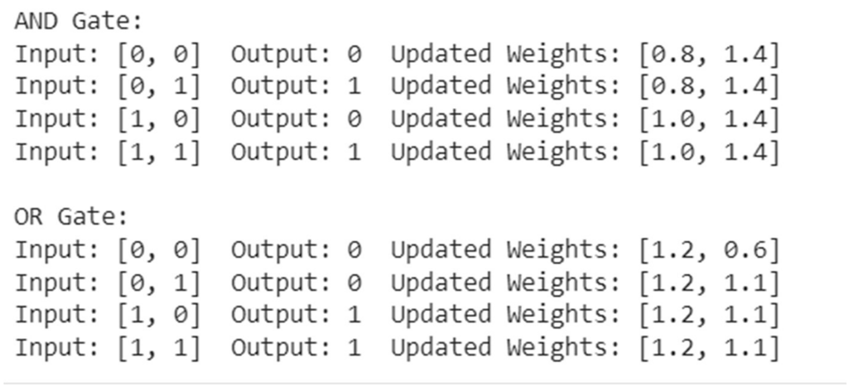
```
def perceptron_AND_OR(inputs, weights, threshold, learning_rate):
    weighted_sum = sum([inputs[i] * weights[i] for i in range(len(inputs))])
    if weighted_sum >= threshold:
        output = 1
    else:
        output = 0
    error = threshold - output
    for i in range(len(weights)):
        weights[i] += learning_rate * error * inputs[i]
    return output, weights
inputs_AND = [[0, 0], [0, 1], [1, 0], [1, 1]]
threshold_AND = 1
learning_rate_AND = 0.2
weights_AND = [0.8, 1.4]
print("AND Gate:")
for input_set in inputs_AND:
    output, weights_AND = perceptron_AND_OR(input_set, weights_AND, threshold_AND,
learning_rate_AND)
    print("Input:", input_set, " Output:", output, " Updated Weights:", weights_AND)
inputs_OR = [[0, 0], [0, 1], [1, 0], [1, 1]]
threshold_OR = 1
learning_rate_OR = 0.5
weights_OR = [1.2, 0.6]
print("\nOR Gate:")
```

for input_set in inputs_OR:

    output, weights_OR = perceptron_AND_OR(input_set, weights_OR, threshold_OR, learning_rate_OR)

    print("Input:", input_set, " Output:", output, " Updated Weights:", weights_OR)

Output:



```
AND Gate:
Input: [0, 0]  Output: 0  Updated Weights: [0.8, 1.4]
Input: [0, 1]  Output: 1  Updated Weights: [0.8, 1.4]
Input: [1, 0]  Output: 0  Updated Weights: [1.0, 1.4]
Input: [1, 1]  Output: 1  Updated Weights: [1.0, 1.4]

OR Gate:
Input: [0, 0]  Output: 0  Updated Weights: [1.2, 0.6]
Input: [0, 1]  Output: 0  Updated Weights: [1.2, 1.1]
Input: [1, 0]  Output: 1  Updated Weights: [1.2, 1.1]
Input: [1, 1]  Output: 1  Updated Weights: [1.2, 1.1]
```