```
# PYTHON

Answer NO 1:-

 output = {
    "abc": ["def", "ghi", "jkl", "mno", "pqr", "stu", "vwx", "yz"],
    "def": ["ghi", "jkl", "mno", "pqr", "stu", "vwx", "yz"],
    "ghi": ["jkl", "mno", "pqr", "stu", "vwx", "yz"],
    "jkl": ["mno", "pqr", "stu", "vwx", "yz"],
    "mno": ["pqr", "stu", "vwx", "yz"],
    "pqr": ["stu", "vwx", "yz"],
    "stu": ["vwx", "yz"],
    "vwx": ["yz"],
    "yz": ["you are finally here !!!"]
}

for key, value in output.items():
    print(f"{key}: {value}")

ANSWER NO 2:-


 def count_horses(stalls, distance):
    count = 1
    last_stall = stalls[0]
    for stall in stalls:
        if stall - last_stall >= distance:
            count += 1
            last_stall = stall
    return count

def max_min_distance(stalls, k):
    stalls.sort()
    left, right = 1, stalls[-1] - stalls[0] + 1

    while left < right:
        mid = left + (right - left) // 2
        if count_horses(stalls, mid) >= k:
            left = mid + 1
        else:
            right = mid

    return left - 1

 ANSWER NO 3:-

ANSWER NO 4 :-def fourSum(nums, target):
    nums.sort()
    quadruplets = []
    n = len(nums)
```

```
    for i in range(n - 3):
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        for j in range(i + 1, n - 2):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue

            left = j + 1
            right = n - 1

            while left < right:
                current_sum = nums[i] + nums[j] + nums[left] +
nums[right]

                if current_sum == target:
                    quadruplets.append([nums[i], nums[j], nums[left],
nums[right]])

                    while left < right and nums[left] == nums[left +
1]:
                        left += 1
                    while left < right and nums[right] == nums[right -
1]:
                        right -= 1

                    left += 1
                    right -= 1
                elif current_sum < target:
                    left += 1
                else:
                    right -= 1

    return quadruplets



# SQL

ANSWER NO 1:-

To fix this issue, assuming you intend to select all the runners who
have not won any races, you can rewrite the query using a LEFT JOIN to
handle NULL values explicitly:

SELECT r.*
FROM runners r
LEFT JOIN races ra ON r.id = ra.winner_id
WHERE ra.winner_id IS NULL OR ra.winner_id = '';

This approach explicitly handles NULL values and empty strings,
```

```
providing a more reliable way to achieve the desired result of
selecting runners who have not won any races.

ANSWER NO 2:-

 To fetch values from table test_a that are not in test_b without
using the NOT keyword, you can use a LEFT JOIN and filter out the rows
where there is no match in test_b. Here's it is:

 SELECT a.id
FROM test_a a
LEFT JOIN test_b b ON a.id = b.id
WHERE b.id IS NULL;

 ANSWER NO 3:-

 SELECTA
    u.user_id,
    u.username,
    td.training_id,
    COUNT(*) AS times_taken
FROM
    users u
JOIN
    training_details td ON u.user_id = td.user_id
GROUP BY
    u.user_id, td.training_id, td.training_date
HAVING
    COUNT(*) > 1
ORDER BY
    td.training_date DESC;

ANSWER NO 4:-

 SELECT Manager_Id AS Manager_d,
       Emp_name AS Manager,
       AVG(Salary) AS Average_Salary_Under_Manager
FROM Employees
WHERE Manager_Id IS NOT NULL
GROUP BY Manager_Id, Emp_name
ORDER BY Manager_Id;

# STATISTICS

ANSWER NO 1:-

Six Sigma is a statistical concept that originated in the
manufacturing industry and is used to measure and improve the quality
of processes by minimizing defects and variations. The term "Six
Sigma" refers to a process that operates with extremely high accuracy
and precision, with only 3.4 defects per million opportunities.
```

In statistical terms, the Sigma (σ) symbol represents the standard deviation, which measures the amount of variation or dispersion in a set of data. In a Six Sigma process, the goal is to reduce this variation so that the process operates consistently and produces high-quality output.

Here's how Six Sigma is typically applied statistically:

Defining Defects:

Measuring Defects:

Calculating Sigma Level:

Example:

ANSWER NO 2:-

Data that do not adhere to a log-normal or Gaussian distribution can be found across various domains. Here are some examples:

Internet Traffic: The volume of data transferred over the internet often follows a distribution that is far from Gaussian. Internet traffic tends to have bursts and spikes during certain periods, leading to a distribution that is skewed and potentially heavy-tailed.

Income Distribution: The distribution of incomes within a population is typically highly skewed, with a small percentage of individuals earning a disproportionately large share of the total income. This distribution does not conform to a Gaussian or log-normal pattern, as it exhibits a long tail on the higher income side.

Power Grid Usage: The usage of electricity in a power grid can exhibit non-Gaussian behavior. During peak hours, there may be significant spikes in electricity consumption, leading to a distribution that is skewed and potentially has heavy tails.

Customer Purchase Behavior: The amount of money spent by customers in retail stores or online shops can vary widely and often does not follow a Gaussian distribution. There may be a small number of customers making large purchases, while the majority make smaller purchases, resulting in a skewed distribution.

Earthquake Magnitudes: The magnitudes of earthquakes follow a distribution known as the Gutenberg-Richter law, which is characterized by a power-law relationship rather than a Gaussian or log-normal distribution. This means that while small earthquakes are more frequent, larger earthquakes occur less frequently but with potentially significant impact.

```
ANSWER 3:-

The five-number summary in statistics provides a concise summary of
the distribution of a dataset. It consists of the following five
values:

Minimum: The smallest value in the dataset.
First Quartile (Q1): The value below which 25% of the data falls.
Median (Q2): The middle value of the dataset when it is sorted in
ascending order. It divides the dataset into two equal halves.
Third Quartile (Q3): The value below which 75% of the data falls.
Maximum: The largest value in the dataset.
The five-number summary is useful for understanding the spread,
central tendency, and shape of the data. It helps in identifying
outliers and comparing different datasets.

For example, consider the dataset: 3, 7, 8, 9, 10, 12, 15, 18, 20, 25.

The five-number summary would be:

Minimum: 3
Q1: 7.5 (average of 7 and 8)
Median: 10
Q3: 16.5 (average of 15 and 18)
Maximum: 25

ANSWER NO 4:-

pip install numpy pandas matplotlib

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# Creating sample dataset
data = {
    'X': [1, 2, 3, 4, 5],
    'Y': [2, 4, 6, 8, 10]
}

# Creating DataFrame from the dataset
df = pd.DataFrame(data)

# Calculating correlation coefficient
correlation_coefficient = df['X'].corr(df['Y'])

print("Correlation Coefficient:", correlation_coefficient)

# Plotting the dataset
plt.scatter(df['X'], df['Y'], color='blue')
```

```
plt.title('Scatter plot of X and Y')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.show()

# MACHINE LEARNING

ANSWER NO:- 1

ANSWER NO:- 2

ANSWER NO:- 3

Let's start by implementing the steps for training and fine-tuning a
Decision Tree using the wine dataset:

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split,
RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from scipy.stats import randint

# Step 1: Load the wine dataset
wine_data = load_wine()
X = wine_data.data
y = wine_data.target

# Step 2: Split the dataset into train and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 3: Hyperparameter tuning using RandomizedSearchCV
param_dist = {
    'max_depth': randint(1, 10),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'criterion': ['gini', 'entropy']
}

dt_classifier = DecisionTreeClassifier()
random_search = RandomizedSearchCV(dt_classifier,
param_distributions=param_dist, n_iter=100, cv=5, scoring='accuracy',
random_state=42)
random_search.fit(X_train, y_train)

print("Best Parameters:", random_search.best_params_)

# Step 4: Evaluate the model
best_dt_model = random_search.best_estimator_
```

```
y_pred = best_dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Decision Tree:", accuracy)

Now, let's proceed with growing a random forest:


from sklearn.model_selection import ShuffleSplit
from sklearn.base import clone

# Step 1: Create 10 subsets of the training dataset
n_trees = 10
shuffle_split = ShuffleSplit(n_splits=n_trees, train_size=0.8,
random_state=42)

# Step 2: Train 1 decision tree on each subset
trees = []
for train_index, _ in shuffle_split.split(X_train):
    clone_dt = clone(best_dt_model)
    clone_dt.fit(X_train[train_index], y_train[train_index])
    trees.append(clone_dt)

# Step 3: Evaluate all the trees on the test dataset
ensemble_predictions = []
for tree in trees:
    y_pred_tree = tree.predict(X_test)
    ensemble_predictions.append(y_pred_tree)

# Compute the ensemble prediction (majority voting)
ensemble_predictions = np.array(ensemble_predictions)
ensemble_predictions = np.transpose(ensemble_predictions)
ensemble_predictions = [np.argmax(np.bincount(pred)) for pred in
ensemble_predictions]

# Evaluate the ensemble model
ensemble_accuracy = accuracy_score(y_test, ensemble_predictions)
print("Accuracy of Random Forest:", ensemble_accuracy)




# Deep Learning

ANSWER NO 1:-

(a). Implementing Deep Learning (DL) in a real-world application
involves several steps:

1. Define the Problem: Clearly define the problem you want to solve
and determine if DL is the right approach. DL is suitable for tasks
such as image and speech recognition, natural language processing, and
```

```
more.

2. Collect and Preprocess Data: Gather a sufficient amount of labeled
data for training and testing. Preprocess the data to ensure it is in
a suitable format and is representative of the real-world scenarios

3. Choose a DL Framework: Select a deep learning framework such as
TensorFlow, PyTorch, or Keras. These frameworks provide a set of tools
and abstractions to simplify the implementation of neural networks.

4. Design the Neural Network Architecture: Define the architecture of
your neural network. This includes the number and type of layers, the
activation functions, and the connections between neurons

5. Train the Model: Split your dataset into training and testing sets.
Train the model on the training set using an optimization algorithm,
adjusting the weights and biases of the network to minimize the error

6. Validate and Tune: Evaluate the model on the validation set to
ensure it generalizes well to new data. Fine-tune hyperparameters and
architecture based on performance

7. Deploy the Model: Once satisfied with the model's performance,
deploy it to the real-world environment. This could involve
integrating it into a web application, a mobile app, or an embedded
system.

8. Monitor and Update: Regularly monitor the model's performance in
the real-world environment. If necessary, update the model with new
data and retrain it to adapt to changing conditions.
(B). Use of Activation Function:

Introducing Non-Linearity: Activation functions introduce non-
linearities into the network, allowing it to model and understand
complex patterns and relationships in the data.

Learning Complex Representations: Non-linear activation functions
enable the neural network to learn hierarchical and intricate
representations of the input data, which is essential for capturing
features at different levels of abstraction.

Gradient Descent Optimization: Activation functions help in the
optimization process during training by providing gradients that allow
the network to adjust its parameters through backpropagation.

(C). Problem Without Activation Function:- If neural networks had no
activation functions, they would fail to learn the complex non-linear
patterns that exist in real-world data

ANSWER NO 2:-
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

# Design a simple ANN model
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(28 *
28,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
batch_size=128, validation_data=(test_images, test_labels))

ANSWER NO 3:-

let's use the Boston Housing Prices dataset, which is a classic
regression dataset available in scikit-learn.

Here's:

import numpy as np
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the Boston Housing Prices dataset
boston = load_boston()
X = boston.data
y = boston.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer with single neuron for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=32,
verbose=1, validation_split=0.2)

# Evaluate the model
mse = model.evaluate(X_test_scaled, y_test, verbose=0)
print("Mean Squared Error on Test Set:", mse)
```