1) Data Wrangling, I
Perform the following operations using Python on any open-source dataset (e.g., data.csv)
1. Import all the required Python Libraries.
2. Locate an open-source data from the web (e.g., https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site).
3. Load the Dataset into pandas dataframe.
4. Data Preprocessing: check for missing values in the data using pandas isnull (), describe () function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitative variables in Python.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv'
df = pd.read_csv(url)

missing_values = df.isnull().sum()
initial_stats = df.describe()
df_shape = df.shape
df_dtypes = df.dtypes

df['Age'] = df['Age'].fillna(df['Age'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
df['Cabin'] = df['Cabin'].fillna('Unknown')

df['Survived'] = df['Survived'].astype('category')
df['Pclass'] = df['Pclass'].astype('category')
df['Sex'] = df['Sex'].astype('category')
df['Embarked'] = df['Embarked'].astype('category')

df_encoded = pd.get_dummies(df, columns=['Sex', 'Embarked', 'Pclass'], drop_first=True)

print(missing_values)
print(initial_stats)
print(df_shape)
print(df_dtypes)
print(df_encoded.head())
```

2) Data Wrangling II
Create an “Academic performance” dataset of students and perform the following operations using Python.
1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.
2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.
3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.preprocessing import StandardScaler

data = {
    'Student_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Math_Score': [88, 92, 95, 100, 67, 73, np.nan, 85, 110, 89],
    'Science_Score': [78, 85, 90, 75, 70, 80, 82, 95, 120, np.nan],
    'English_Score': [80, 79, 85, 83, 77, 75, 70, 88, 60, 90],
    'Attendance_Percent': [95, 98, 100, 60, 88, 85, 90, 92, 105, 75]
}

df = pd.DataFrame(data)

print("Missing Values:")
print(df.isnull().sum())

df['Math_Score'].fillna(df['Math_Score'].mean(), inplace=True)
df['Science_Score'].fillna(df['Science_Score'].mean(), inplace=True)

print("\nAfter Handling Missing Values:")
print(df)

numeric_cols = ['Math_Score', 'Science_Score', 'English_Score', 'Attendance_Percent']
z_scores = np.abs(stats.zscore(df[numeric_cols]))
df_no_outliers = df[(z_scores < 3).all(axis=1)]

print("\nAfter Removing Outliers:")
print(df_no_outliers)

df_no_outliers['Log_English_Score'] = np.log1p(df_no_outliers['English_Score'])

print("\nAfter Log Transformation on English_Score:")
print(df_no_outliers[['English_Score', 'Log_English_Score']])
```

3) Descriptive Statistics - Measures of Central Tendency and variability
Perform the following operations on any open-source dataset (e.g., data.csv)
1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.
2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris-versicolor' of iris.csv dataset.

```python
import pandas as pd
import numpy as np
import seaborn as sns

# Part 1: Grouped Summary Statistics
df = sns.load_dataset("titanic")

grouped_stats = df.groupby('class')['age'].agg(['count', 'mean', 'median', 'min', 'max',
'std']).reset_index()
print("Summary Statistics for 'age' grouped by 'class':")
print(grouped_stats)

age_list = df.dropna(subset=['class', 'age']).groupby('class')['age'].apply(list).reset_index()
print("\nList of ages for each class:")
print(age_list)

# Part 2: Iris Dataset Statistics
iris = sns.load_dataset('iris')

setosa = iris[iris['species'] == 'setosa']
versicolor = iris[iris['species'] == 'versicolor']
virginica = iris[iris['species'] == 'virginica']

print("\nSetosa Statistics:")
print(setosa.describe(percentiles=[.25, .5, .75]))

print("\nVersicolor Statistics:")
print(versicolor.describe(percentiles=[.25, .5, .75]))

print("\nVirginica Statistics:")
print(virginica.describe(percentiles=[.25, .5, .75]))
```

4) Data Analytics I
Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load data
df = pd.read_csv('bostonhousing.csv')

# Drop missing values
df.dropna(inplace=True)

# Features and Target
X = df.drop('medv', axis=1)
y = df['medv']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Training
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction and Evaluation
y_pred = model.predict(X_test)
print(f'Mean Squared Error (MSE): {mean_squared_error(y_test, y_pred):.2f}')
print(f'R-squared (R²): {r2_score(y_test, y_pred):.2f}')
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)

# Predicting on New Data
new_data = np.array([[0.1, 18, 2.31, 0, 0.4, 6.1, 45, 6.5, 5, 300, 15, 400, 10]])
new_prediction = model.predict(new_data)
print(f'Predicted Home Price for New Data: {new_prediction[0]:.2f}')
```

5) Data Analytics II

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

# Load data
df = pd.read_csv('Social_Network_Ads.csv')
X = df[['Age', 'EstimatedSalary']]
y = df['Purchased']

# Scaling and splitting
X_train, X_test, y_train, y_test = train_test_split(StandardScaler().fit_transform(X), y,
test_size=0.25, random_state=0)

# Model
model = LogisticRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

# Metrics
cm = confusion_matrix(y_test, y_pred)
TN, FP, FN, TP = cm.ravel()
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Results
print(f"Confusion Matrix:\n{cm}")
print(f"TP:{TP} FP:{FP} TN:{TN} FN:{FN}")
print(f"Accuracy:{accuracy:.2f} Error Rate:{error_rate:.2f} Precision:{precision:.2f}
Recall:{recall:.2f}")
```

6) Data Analytics III

1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset.

2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

```python
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

7) Text Analytics

1. Extract Sample document and apply following document preprocessing methods:
Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.

2. Create representation of document by calculating Term Frequency and Inverse Document Frequency

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')


# Sample document
document = "The quick brown fox jumps over the lazy dog. The dog was not amused by the fox."

# Tokenization
tokens = word_tokenize(document)

# POS Tagging
pos_tags = pos_tag(tokens)

# Stop words removal
stop_words = set(stopwords.words('english'))
tokens_no_stopwords = [word for word in tokens if word.lower() not in stop_words]

# Stemming
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in tokens_no_stopwords]

# Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens_no_stopwords]

# Term Frequency and Inverse Document Frequency
corpus = [document]
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(corpus)

print("Tokenized Document:", tokens)
print("POS Tagging:", pos_tags)
print("Tokens without Stopwords:", tokens_no_stopwords)
print("Stemmed Tokens:", stemmed_tokens)
print("Lemmatized Tokens:", lemmatized_tokens)
print("TF-IDF Matrix:\n", tfidf_matrix.toarray())
```

8) Data Visualization I

1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.

2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

```python
import seaborn as sns
import matplotlib.pyplot as plt

titanic = sns.load_dataset('titanic')

sns.pairplot(titanic)
plt.show()

sns.histplot(titanic['fare'], kde=True)
plt.title('Distribution of Ticket Fare')
plt.xlabel('Fare')
plt.ylabel('Frequency')
plt.show()
```

9) Data Visualization II

1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age')

2. Write observations on the inference from the above statistics.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Load Titanic dataset
titanic = sns.load_dataset('titanic')

# Box plot for distribution of age with respect to gender and survival status
sns.boxplot(x='sex', y='age', hue='survived', data=titanic)
plt.title('Age Distribution by Gender and Survival Status')
plt.show()
```

10) Data Visualization III
Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as:
1. List down the features and their types (e.g., numeric, nominal) available in the dataset.
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
3. Create a boxplot for each feature in the dataset.
4. Compare distributions and identify outliers.

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Iris dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ["sepal_length", "sepal_width", "petal_length", "petal_width", "species"]
df = pd.read_csv(url, names=columns)
# df=sns.load_dataset('iris')

# 1. List down the features and their types
features_types = df.dtypes
print("Features and their types:\n", features_types)

# 2. Create a histogram for each feature
df.drop("species", axis=1).hist(figsize=(10, 8))
plt.suptitle('Histogram of Features')
plt.show()

# 3. Create a boxplot for each feature
sns.boxplot(data=df.drop("species", axis=1), orient="h", palette="Set2")
plt.title('Boxplot of Features')
plt.show()

# 4. Compare distributions and identify outliers
sns.pairplot(df, hue="species", palette="Set1")
plt.suptitle('Pairplot of Features with Species', y=1.02)
plt.show()
```

B 4) Write a simple program to find a number is positive, negative or positive in SCALA using Apache Spark framework.

```
import org.apache.spark.sql.SparkSession

object NumberCheck {
  def main(args: Array[String]): Unit = {
    // Step 1: Create SparkSession
    val spark = SparkSession.builder()
      .appName("Number Check")
      .master("local[*]") // Run locally with all cores
      .getOrCreate()

    // Step 2: Input list of numbers (you can modify or get input dynamically)
    val numbers = List(10, -5, 0, 42, -8)

    // Step 3: Parallelize the list using Spark
    val rdd = spark.sparkContext.parallelize(numbers)

    // Step 4: Map each number to its type (positive, negative, or zero)
    val resultRDD = rdd.map { num =>
      if (num > 0) s"$num is Positive"
      else if (num < 0) s"$num is Negative"
      else s"$num is Zero"
    }

    // Step 5: Collect and print results
    resultRDD.collect().foreach(println)

    // Step 6: Stop Spark session
    spark.stop()
  }
}
```

B 4) Write a simple program for calculator in SCALA using Apache Spark framework.

```
import org.apache.spark.sql.SparkSession

object SparkCalculator {
  def main(args: Array[String]): Unit = {

    // Step 1: Initialize SparkSession
    val spark = SparkSession.builder()
      .appName("Spark Calculator")
      .master("local[*]") // Use all available cores
      .getOrCreate()

    // Step 2: Sample list of tuples (number1, number2)
    val numberPairs = List((10, 2), (7, 0), (5, 3), (12, 4))

    // Step 3: Parallelize data using RDD
    val rdd = spark.sparkContext.parallelize(numberPairs)

    // Step 4: Perform operations
    val resultRDD = rdd.map { case (a, b) =>
      val add = a + b
      val sub = a - b
      val mul = a * b
      val div = if (b != 0) a.toDouble / b else "undefined (division by zero)"

      s"For numbers ($a, $b): Add = $add, Sub = $sub, Mul = $mul, Div = $div"
    }

    // Step 5: Show results
    resultRDD.collect().foreach(println)

    // Step 6: Stop Spark
    spark.stop()
  }
}
```

B 4) Write a simple program for prime number in SCALA using Apache Spark framework.

```
import org.apache.spark.sql.SparkSession

object PrimeNumberCheck {
  def main(args: Array[String]): Unit = {

    // Step 1: Initialize SparkSession
    val spark = SparkSession.builder()
      .appName("Prime Number Check")
      .master("local[*]")  // Use all available cores
      .getOrCreate()

    // Step 2: List of numbers to check for primality
    val numbers = List(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)

    // Step 3: Parallelize the list using Spark
    val rdd = spark.sparkContext.parallelize(numbers)

    // Step 4: Check each number for primality
    val resultRDD = rdd.map { num =>
      val isPrime = if (num < 2) false
      else (2 until num).forall(x => num % x != 0)

      if (isPrime) s"$num is Prime" else s"$num is Not Prime"
    }

    // Step 5: Collect and print results
    resultRDD.collect().foreach(println)

    // Step 6: Stop Spark session
    spark.stop()
  }
}
```

B 1) Write a code in JAVA for a simple WordCount application that counts the number of occurrences of each word in a given input set using the Hadoop MapReduce framework on local-standalone set-up.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```java
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String line = value.toString();
            String[] words = line.split("\\s+");
            for (String w : words) {
                word.set(w);
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable result = new IntWritable();

        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: WordCount <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

B 2) Design a distributed application using MapReduce which processes a log file of a system.

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class LogProcessor {

    public static class LogMapper extends Mapper<Object, Text, Text, IntWritable> {
        private Text logLevel = new Text();
        private final static IntWritable one = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String line = value.toString();
            if (line.contains("ERROR")) {
                logLevel.set("ERROR");
            } else if (line.contains("WARN")) {
                logLevel.set("WARN");
            } else if (line.contains("INFO")) {
                logLevel.set("INFO");
            } else {
                return; // Ignore non-relevant log entries
            }
            context.write(logLevel, one);
        }
    }

    public static class LogReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int count = 0;
            for (IntWritable val : values) {
                count += val.get();
            }
            result.set(count);
            context.write(key, result);
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: LogProcessor <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "log processor");

        job.setJarByClass(LogProcessor.class);
        job.setMapperClass(LogMapper.class);
        job.setReducerClass(LogReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```