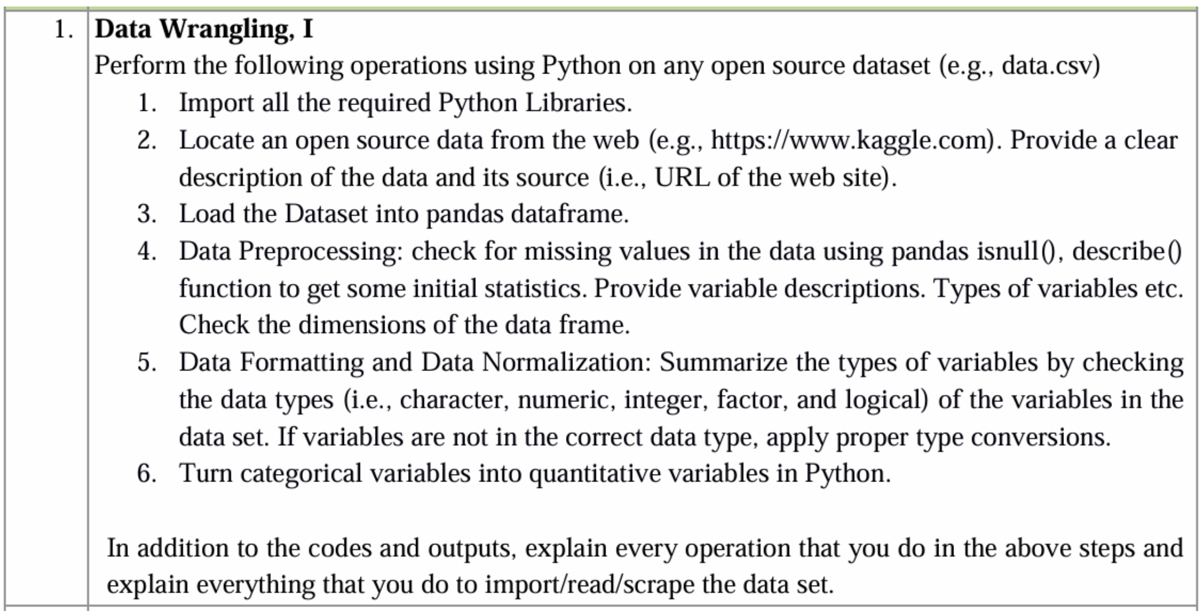
DSBDA All Practicals

A1



1. **Data Wrangling, I**

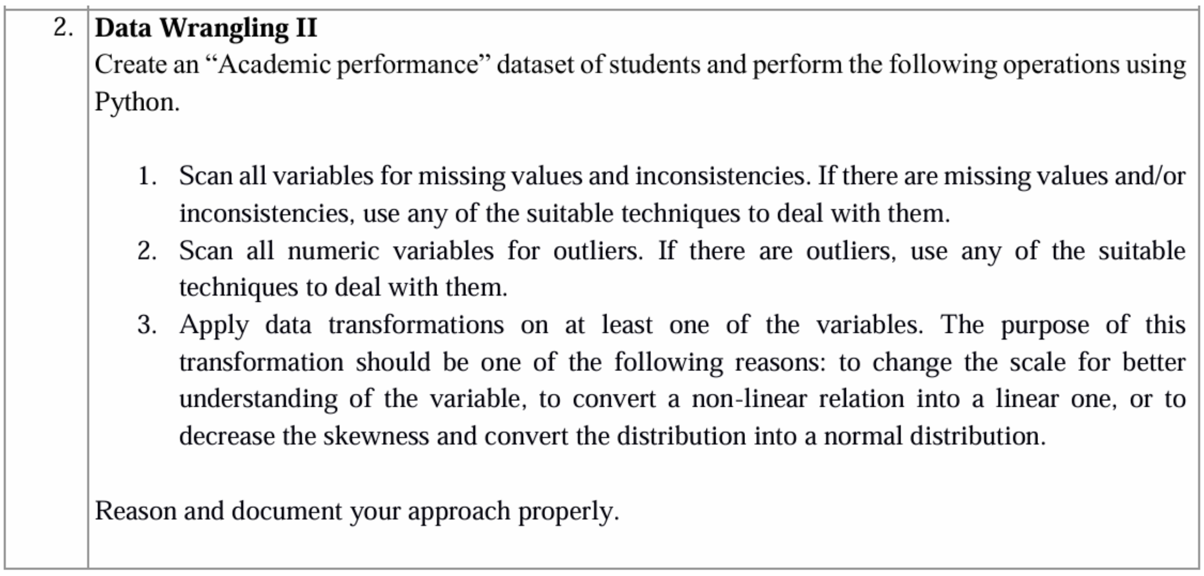
Perform the following operations using Python on any open source dataset (e.g., data.csv)

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.

In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

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

"""Loading the dataset into dataframe"""

df=pd.read_csv("iris.csv")

"""Data Preprocessing"""

df.describe()

df.head()

df.tail()

df.tail(3)

df.info()

df.shape

df.size

df.duplicated()

df.duplicated().sum()
df.isnull()

df.isnull().sum()

df.notnull()

df.notnull().sum()

df["sepal.length"].isnull()

y=df.drop(["petal.length"], axis=1) # axis=1 column. For row, axis=0
print(y)

"""Turn categorical variables into quamtitative variables"""

df['variety'].replace(['Setosa', 'Versicolor', 'Virginica'], [0, 1, 2], inplace=True)
print(df)

#======================================================================

df=df.dropna()

df.isnull().sum()

"""Data type of variables"""

df["Fare"].astype(int)
```

A2



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. Reason and document your approach properly.

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

```
np.random.seed(50)    #for consistency

data = {
    'Student_id': range(1, 51),
    'Name': ['Student_' + str(i) for i in range(1, 51)],
    'Age': np.random.randint(18, 25, size=50),
    'Gender': np.random.choice(['Male', 'Female'], size=50),
    'Scores': [np.random.randint(50, 100, size=3).tolist() for _ in range(50)]
,
    'Attendance': np.random.randint(20,100,size=50),
    'Grade': np.random.choice(['A', 'B', 'C', 'D', 'F'], size=50)
}

df = pd.DataFrame(data)

print(df)

def assign_grade(scores):
    avg_score = np.mean(scores)

    if avg_score > 90:
        return 'A'
    elif avg_score > 80:
        return 'B'
    elif avg_score > 70:
        return 'C'
    elif avg_score > 60:
        return 'D'
    else:
        return 'F'

df['Grade'] = df['Scores'].apply(assign_grade)

#Introduce missing values and inconsistencies
df = pd.DataFrame(data)
df.loc[8, 'Age'] = np.nan
df.loc[29, 'Age'] = np.nan
df.loc[35, 'Age'] = np.nan
df.loc[11, 'Scores'] = None
df.loc[19, 'Scores'] = None
df.loc[9, 'Attendance'] = 105   #invalid percentage
df.loc[15, 'Grade'] = 'Z'   #invalid grade

df.head(20)

"""# Handling Missing Values"""
```

```python
missing_values = df.isnull().sum()    #check missing values
invalid_attendance = df[(df['Attendance'] < 0) | (df['Attendance'] > 100)]
invalid_grades = df[~df['Grade'].isin(['A', 'B', 'C', 'D', 'F'])]

missing_values, invalid_attendance, invalid_grades

df['Age'] = df['Age'].fillna(df['Age'].median())    #fill by median

df['Attendance'] = df['Attendance'].apply(lambda x: 100 if x > 100 else (0 if
x < 0 else x))

def handle_invalid_scores(scores):
    if scores is None:
        return [0, 0, 0]

    return [max(0, min(100, score)) for score in scores]

df['Scores'] = df['Scores'].apply(handle_invalid_scores)

def assign_grade(scores):
    avg_score = np.mean(scores)
    if avg_score > 90:
        return 'A'
    elif avg_score > 80:
        return 'B'
    elif avg_score > 70:
        return 'C'
    elif avg_score > 60:
        return 'D'
    else:
        return 'F'

df['Grade'] = df['Scores'].apply(assign_grade)
df['Grade'] = df['Grade'].apply(lambda x: x if x in ['A', 'B', 'C', 'D', 'F']
else 'F')

df.head(20)

"""# Handling Outliers"""

#Adding outiers
df.loc[5, 'Age'] = 35
df.loc[5, 'Age'] = 50
df.loc[5, 'Age'] = 65
df.loc[10, 'Attendance'] = 200
df.loc[12, 'Attendance'] = 175
df.loc[12, 'Attendance'] = 166
```

```python
print("DataFrame with Outliers:")
print(df.iloc[5:20])

def handle_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df[column] = df[column].apply(lambda x: upper_bound if x > upper_bound els
e (lower_bound if x < lower_bound else x))

handle_outliers_iqr(df, 'Age')
handle_outliers_iqr(df, 'Attendance')

print(df.iloc[5:20])

"""# Data Transformation using Min-Max Scaling"""

df['Scaled_Attendance'] = (df['Attendance'] - df['Attendance'].min()) / (df['A
ttendance'].max() - df['Attendance'].min())

print("DataFrame with Min-Max Scaling on 'Attendance':")
print(df[['Attendance', 'Scaled_Attendance']].head(20))
```

A3

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. Provide the codes with outputs and explain everything that you do in this step.

```
import pandas as pd

data = {
    'age': [25, 30, 22, 40, 55, 60, 33, 28, 45, 50],
    'income': [50000, 60000, 45000, 70000, 80000, 90000, 65000, 62000, 75000,
85000],
    'age_group': ['20-30', '30-40', '20-30', '40-50', '50-60', '50-60', '30-
40', '20-30', '40-50', '50-60']
}

df = pd.DataFrame(data)

# Group the data by age_group and compute summary statistics for 'income'
summary_stats = df.groupby('age_group')['income'].describe()

print(summary_stats)

median_income = df.groupby('age_group')['income'].median()

print("Median Income by Age Group:")
print(median_income)

print("Column Names:", df.columns)

import pandas as pd

# Modified dataset with repeated values
data = {
    'age': [25, 30, 25, 40, 55, 60, 33, 28, 45, 50, 25, 30, 28, 30, 25],
    'income': [50000, 60000, 50000, 70000, 80000, 90000, 65000, 62000, 75000,
85000, 50000, 60000, 62000, 70000, 75000],
    'age_group': ['20-30', '30-40', '20-30', '40-50', '50-60', '50-60', '30-
40', '20-30', '40-50', '50-60', '20-30', '30-40', '20-30', '30-40', '20-30']
}

df = pd.DataFrame(data)

# Calculate the mode for each column
mode_age = df['age'].mode()
mode_income = df['income'].mode()
print(f"Mode of Age: {mode_age.values}")
print(f"Mode of Income: {mode_income.values}")

"""**iris.csv**"""

import pandas as pd

df = pd.read_csv('iris.csv')
```

```python
# Show basic structure of the data
print(df.head())

# Group by species and display summary statistics
summary_stats_species = df.groupby('variety').describe()

# Additionally, we can compute specific percentiles and statistics
percentiles = df.groupby('variety').quantile([0.25, 0.5, 0.75])

# Display summary statistics and percentiles
summary_stats_species = df.groupby('variety').describe()

print("\nPercentiles by Species:")
print(percentiles)

summary_stats_species = df.groupby('variety')['sepal.width'].describe()

print("\nSummary Statistics by Species for Sepal Width:")
print(summary_stats_species)

# Group by species and compute the median for numeric columns
median_values = df.groupby('variety').median()

print("Median Values by Species:")
print(median_values)

median_sepal_length = df.groupby('variety')['sepal.length'].median()
print("Median Sepal Length by Species:")
print(median_sepal_length)

mode_width = df['sepal.width'].mode()
print(f"Mode of Width: {mode_width.values}")
```

A4

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. The objective is to predict the value of prices of the house using the given features.

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

df = pd.read_csv("/content/sample_data/BostonHousing.csv")
```

```
df.head()

X = df[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax'
,
       'ptratio', 'b', 'lstat']]
Y = df['medv']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3, rand
om_state = 42)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(y_pred)

from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mead squared Error is:")
print(rmse)

print("Training accuracy is:")
lr.score(X_train, y_train)

print("Testing accuracy is:")
lr.score(X_test, y_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
print("R2_score: ", r2_score(y_test, y_pred))
print("mean_squared_error: ", mean_squared_error(y_test, y_pred))
print("mean_absolute_error: ", mean_absolute_error(y_test, y_pred))

y_pred = np.array(y_pred)
y_test = np.array(y_test)
data = {
    'Actual_value' : y_test.flatten(),
    'Predicted_value' : y_pred.flatten()
}
df1 = pd.DataFrame(data)
print(df1)
```

```
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual vs Predicted Prices')
plt.grid(True)
plt.show()
```

A5

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 numpy as np
import pandas as pd

data =  pd.read_csv('/content/Social_Network_Ads.csv')
data.head(5)

data.info()

data.describe()

data.isnull().sum()

data.shape

# Optional: Convert Gender to numeric
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

# Features and Target
X = df[['Gender', 'Age', 'EstimatedSalary']]
y = df['Purchased']

# x = data.iloc[:,2:4]

# y = data.iloc[:,4]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, rand
om_state=42)

from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
x_train = scale.fit_transform(x_train)
```

```
x_test = scale.transform(x_test)

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 0,solver = 'lbfgs')
lr.fit(x_train,y_train)
pred = lr.predict(x_test)

print('Expected Output:',pred[:10])
print('-'*15)
print('Predicted Output:\n',y_test[:10])



#evaluate the model
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Extract values
TN, FP, FN, TP = cm.ravel()

# Metrics
accuracy = accuracy_score(y_test, y_pred) # TP+TN / TP+TN+FP+FN
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred) # TP / TP+FP
recall = recall_score(y_test, y_pred) # TP / TP+FN

print(f"True Positives (TP): {TP}")
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")
print(f"Accuracy: {accuracy:.2f}")
print(f"Error Rate: {error_rate:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")

#visualize
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

A6

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.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
 recall_score
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("/content/iris.csv")
print(df.head())

X = df.drop('variety', axis=1)
y = df['variety']

#train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=42)

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

#train NaiveBayes model
model = GaussianNB()
model.fit(X_train_scaled, y_train)

#predict
y_pred = model.predict(X_test_scaled)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
sns.heatmap(cm_df, annot=True, cmap='Blues', fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

classes = model.classes_
```

```python
print("\nClasses:", classes)

# TP, FP, FN, TN for each class
for i in range(len(classes)):
    TP = cm[i, i]
    FP = cm[:, i].sum() - TP
    FN = cm[i, :].sum() - TP
    TN = cm.sum() - (TP + FP + FN)

    print(f"\nFor class '{classes[i]}':")
    print(f"True Positive (TP): {TP}")
    print(f"False Positive (FP): {FP}")
    print(f"False Negative (FN): {FN}")
    print(f"True Negative (TN): {TN}")

# #evaluate the model
# cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
# cm_df = pd.DataFrame(cm, index=model.classes_, columns=model.classes_)

# # Plot Confusion Matrix
# sns.heatmap(cm_df, annot=True, cmap='Blues', fmt='d')
# plt.title('Confusion Matrix')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
# plt.show()

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

print(f"Accuracy: {accuracy:.2f}")
print(f"Error Rate: {error_rate:.2f}")
print(f"Precision (Macro): {precision:.2f}")
print(f"Recall (Macro): {recall:.2f}")
```

A7

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

import nltk
```

```
file=open('/content/ds.txt','r')
#keep file outside sample data for colab and in same folder for jn
#for jn
#file=open('ds.txt','r')

nltk.download('all')

file

content=file.read()
content

from nltk.tokenize import sent_tokenize

sentence = sent_tokenize(content)
sentence

from nltk.tokenize import RegexpTokenizer
tokenizer=RegexpTokenizer(f"\w+")
words=tokenizer.tokenize(content)
words

tokenizer=RegexpTokenizer(f"\s")
words1=tokenizer.tokenize(content)
words1

#POS tagging and stop word removing
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stopWords=set(stopwords.words('english'))

print(stopWords)

for sen in sentence:
    Words=word_tokenize(sen)
    filteredWords=[word.lower() for word in Words if word.lower() not in stopW
ords]
    print(f"words without stopwords{filteredWords}")
    print(f"words with stopwords{Words}")

#POS tagging
for sen in sentence:
    Words=word_tokenize(sen)
    filteredWords=[word.lower() for word in Words if word.lower() not in stopW
ords]
    tagged = nltk.pos_tag(filteredWords)
    print(tagged)

#OR
tagged = nltk.pos_tag(words)
print(tagged)

from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

for sen in sentence:
    Words=word_tokenize(sen)
    filteredWords=[word.lower() for word in Words if word.lower() not in stopW
ords]

    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in filteredWords]

    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filteredWords]

    print("\nOriginal Tokens:", Words)
    print("Stemmed Tokens:", stemmed_tokens)
    print("Lemmatized Tokens:", lemmatized_tokens)

#OR
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in words]

lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in words]

print("\nOriginal Tokens:", Words)
print("Stemmed Tokens:", stemmed_tokens)
print("Lemmatized Tokens:", lemmatized_tokens)

#the difference between first and second is that in first approach we performe
d it on filtered words and in the second we performed it on simple array of wo
rds

# 6. TF-IDF Calculation
# TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform([content])
```

```python
# Show the TF-IDF scores for the terms in the document
terms = vectorizer.get_feature_names_out()
print("\nTF-IDF Scores:")
for term, score in zip(terms, tfidf_matrix.toarray()[0]):
    print(f"{term}: {score}")
```

A8

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 numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
df.head()

df.info()

df.describe()

df.shape

sns.histplot(x='fare',data=df)
#sns.set(rc={'figure.figsize':(5,5)})

sns.displot(x='age',data=df,bins=70)

sns.catplot(x='survived', data=df, kind='count', hue='pclass')

plt.figure(figsize=(5,4))
vp=sns.violinplot(x='class',y='age',palette='rainbow',data=df)
plt.show()

plt.figure(figsize=(6,5))
ab=sns.stripplot(x='class',y='age',data=df)
plt.show()

plt.figure(figsize=(5,4))
wt=sns.swarmplot(x='class',y='age',data=df)
```

```
plt.show()

plt.figure(figsize=(5,4))
kl=sns.scatterplot(x='age',y='fare',data=df,hue='survived')
plt.show()

plt.figure(figsize=(6,4))
ad=sns.countplot(x='class',data=df)
for i in ad.containers:
  ad.bar_label(i)
plt.show()
```

A9

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.

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

df = sns.load_dataset('titanic')
df.head()

plt.figure(figsize=(8,4))
sns.boxplot(x='sex', y='age', data=df, hue='survived')
plt.title("Distribution of age with respect to each gender and Survival Status
")
plt.xlabel("Sex")
plt.ylabel("Age")
plt.show()

"""Observations

We created a box plot of variables 'age & 'sex' & used survival as the hue

There we visualized three variables Age, Sex & Survival. Two out of these are
categorical and one is numerical

Now in addition to the information about the age of each gender, you can also
see the distribution of passengers who survived

For instance, you can see that among the male passengers, on average more youn
ger people survived as compared to older ones
"""

ax=sns.barplot(x='sex',y='age',data=df, palette='Set2', hue='survived')
```

```python
vp=sns.violinplot(x='sex',y='age',data=df, hue= 'survived')

ab=sns.stripplot(x='sex',y='age',data=df)
```

A10

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 seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset('iris')

df

df.head()

df.info()

print(df.dtypes)

sns.histplot(x='sepal_length', data=df)

fig, axes = plt.subplots(2,2, figsize=(15, 10))
fig.suptitle('Histogram')
sns.histplot(x='sepal_length', data=df, ax=axes[0,0])
sns.histplot(x='sepal_width', data=df, ax=axes[0,1])
sns.histplot(x='petal_length', data=df, ax=axes[1,0])
sns.histplot(x='petal_width', data=df, ax=axes[1,1])

fig, axes = plt.subplots(2,2, figsize=(15, 10))
fig.suptitle('Boxplot')
sns.boxplot(y='sepal_length', x='species', data=df, ax=axes[0,0])
sns.boxplot(y='sepal_width', x='species', data=df, ax=axes[0,1])
sns.boxplot(y='petal_length', x='species', data=df, ax=axes[1,0])
sns.boxplot(y='petal_width', x='species', data=df, ax=axes[1,1])

"""https://dev.to/thalesbruno/subplotting-with-matplotlib-and-seaborn-5ei8

Conclusion
```

```

The Iris dataset consists of four numerical features (sepal length, sepal widt
h, petal length, and petal width) and one categorical feature (species), which
 includes three classes: Setosa, Versicolor, and Virginica.

Histograms and boxplots revealed the distribution of each feature, with sepal
width showing some outliers, identified using the IQR method.

Violin plots demonstrated the variation of different features across species,
showing that Setosa is distinct from the other two species, while Versicolor a
nd Virginica overlap in some features.

The pair plot indicated strong correlations, especially between petal length a
nd petal width, confirming that these features contribute significantly to spe
cies classification.

The analysis helped visualize data distribution, identify outliers, and unders
tand feature relationships, providing insights for further classification task
s.
"""
```