INDEX

NAME: Hanisha. P STD: AIDS SEC: ROLL NO.: 96 SUB PODS

S.No.	Date	Title	Page No.	Teacher's Sign / Remarks
1.	19.7.25	Numpy Commands	10	
2.	19.7.25	Pandas Commands	10	
3.	22.7.25	Titanic Dataset & Convert into dataframe	16	
4.	22.7.25	Data Preprocessing	10	
5.	26.7.25	Sales prediction & Customer Segmentation using linear Regression & K means Clustering	10	
6.	29.7.25	Text Preprocessing & Analysis Pipeline.		
7.	5.8.25	NLP		
8.	9.9.25	Exploratory Data Analysis		
9.	16.9.25	Clustering		
10.	20.9.25	Dashboards with Tableau using Google Sheets		

19-7-25 NUMPY

A simple program to implement one dimensional array using numpy.

1. import numpy
a = numpy.array([10,20,30,40,50])
print(a)

o/p: [10, 20, 30, 40, 50]

2. from numpy import *
a = array([10,20,30,40,50])
print(a)

o/p: [10, 20, 30, 40, 50]

3. from numpy import *
a = array([10,30,40.5,50,100])
print(a)

o/p: [10.0, 30.0, 40.5, 50.0, 100.0]

4. import numpy as np
a = np.linspace(1,10,10)
print(a)

o/p: [1. 2. 3. 4. 5. 6. 7. 8. 9. 10.]

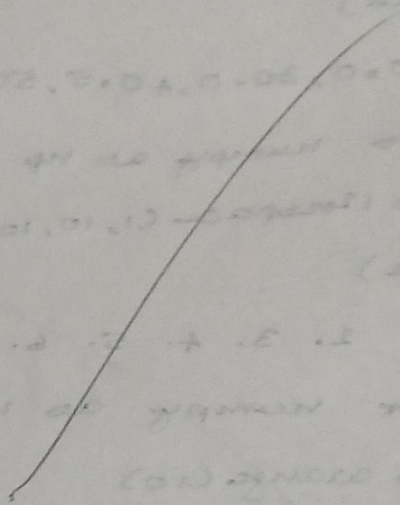
5. import numpy as np
a = np.arange(10)
b = np.arange(5,10)
c = np.arange(10,1,-1)
print(a)
print(b)
print(c)

o/p: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2]

o/p: [0., 0., 0., 0., 0.]
[1., 1., 1., 1., 1.]

o/p: [10, 20, 30, 40, 50]
[5, 15, 25, 35, 45]
[50, 100, 150, 200, 250]
[2. 4. 6. 8. 10.]

o/p: [45, 5, 6, 7, 8] (Aliased)
[45, 5, 6, 7, 8] (Independent)



o/p: [[2, 4, 6]
[6, 8, 10]]

o/p: 1
2
(2, 3)
6
Usually 4 (bytes)

6. import numpy as np

```
k = np.zeros(5)
R = np.ones(5)
print(k)
print(R)
```

MATHEMATICAL OPERATIONS

```
import numpy as np
k = np.array([10, 20, 30, 40, 50])
print(k+5)
print(k-5)
print(k*5)
print(k/5)
```

ALIASING VS COPY

```
k = np.array([3, 5, 6, 7, 8])
h = k
k[0] = 45
print(h)
h = k.copy()
k[0] = 100
print(h)
```

2D Array

```
x = np.array([[2, 4, 6], [6, 8, 10]])
print(x)

A = np.array([5, 6, 7, 8])
R = np.array([[4, 5, 6], [7, 8, 9]])
print(A.ndim)
print(R.ndim)
print(R.shape)
print(R.size)
print(R.itemsize)
```

o/p: 6 rows, 0 columns

1 row, 12 columns

o/p: Random Garbage values

[[0.0] [0.0] [0.0]]

[[1 1] [1 1] [1 1]]

o/p: [2 3 4 8 9 10]

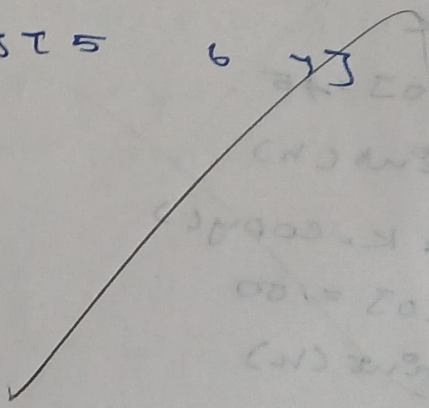
[2 3 4 8 9 10]

[[2 3 4]]

[8 9 10]]

o/p:

[0 1 2] [3 4] [5 6 7]



o/p: [[+1 . -1 .]

[-1 . +1 .]]

Reshaping

```
d = np.array([1,5,6,7], [5,6,7,8], [7,8,9,6])
print(d.reshape(6,12))
print(d.reshape(1,12))
```

Special Arrays

```
np.eye(3)
np.empty([3,2], int)
np.zeros([3,2], int)
np.ones([3,2], int)
```

Joining Arrays

```
a = np.array([2,3,4])
b = np.array([8,9,10])
print(np.concatenate([a,b]))
print(np.hstack((a,b)))
print(np.vstack((a,b)))
```

Splitting Arrays

```
x = np.arange(8)
x1, x2, x3 = np.split(x, [3,5])
print(x1, x2, x3)
a = np.arange(16).reshape((4,4))
left, right = np.hsplit(a,2)
top, bottom = np.vsplit(a,2)
```

Covariance

```
x = np.array([0, 1, 2])
y = np.array([2, 1, 0])
print(np.cov(x, y))
```

O/P: Cars parkings

0	BMW	3
1	Volvo	7
2	Ford	2

O/P:

0	10
1	20
2	30
3	40
4	50

dtype: int 64

O/P: Series Representation

0	10
1	20
2	30
3	40
4	50

dtype: int 64

Array Representation: [10 20 30 40 50]

Index Object: RangeIndex (start=0, stop=5, step=1)

O/P:

x	1
y	4
z	2

dtype: int64

O/P:

a	10
b	20
c	30
d	40
e	50

dtype: int64

19.4.25 PANDAS

1. import pandas
mydataset = {
'cars': ["BMW", "Volvo", "Ford"],
'passings': [3, 7, 2]
}
myvar = pandas.DataFrame(mydataset)
print(myvar)

2. import pandas as pd
data = [10, 20, 30, 40, 50]
series = pd.Series(data)
print(series)

3. import pandas as pd
data = [10, 20, 30, 40, 50]
series = pd.series(data)
print("Series Representation & It's series")
array_representation = series.values
index_object = series.index
print("Array Representation:", array_representation)
print("Index Object:", index_object)

4. a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)

5. data = [10, 20, 30, 40, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.series(data, index = index_labels)
print(series)

O/P: 20

a	10
c	30
e	50

dtype: int64

O/P: series > 25 =

0	False
1	False
2	True
3	True
4	True

dtype: bool

Filtered series:

2	30
3	40
4	50

dtype: int64

Series after scalar Multiplication:

0	20
1	40
2	60
3	80
4	100

dtype: int64

O/P: series =

a	10
b	20
c	30
d	40
e	50

dtype: int64

6. print (series['b'])
print (series[['a', 'c', 'e']])

7. gt = series > 25
print ("series > 25 = \n", gt)
series_filtered = series[series > 25]
print ("Filtered Series\n", series_filtered)
series_multiplied = series * 2
print ("\n Series after Scalar Multiplication: \n",
series_multiplied)

8. import numpy as np
print ("series = \n", series)
series_sqrt = np.sqrt (series)
print ("\n Series after Applying Square Root: \n",
series_sqrt)

Series after Applying Square Root:

a	3.162278
b	4.472136
c	5.477226
d	6.324555
e	7.071068

dtype: float64

O/P: California 38332521
Texas 26448193
New York 19651127
Florida 19552860
Illinois 12882135
dtype: int64

O/P: True
False

O/P:

a	10.0
b	NaN
c	30.0
d	NaN
e	50.0

dtype: float64

9. Population_dict = {'California': 38332521,
'Texas': 26448193,
'New York': 19651127,
'Florida': 19552860,
'Illinois': 12882135}

population = pd.Series(population_dict)
print(population)

10. print('California' in population)
print('Canada' in population)

11. data = [10, npnan, 30, npnan, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series_with_nan = pd.Series(data, index = index_labels)
print(series_with_nan)
print("\n Checking for Nan values:\n",
series_with_nan.isnull())
print("\n Checking for non-NaN values:\n",
series_with_nan.notnull())

Checking for Null values:

a	False
b	True
c	False
d	True
e	False

dtype: bool

Checking for Non-Null values:

a	True
b	False
c	True
d	False
e	True

dtype: bool

O/P: Population Series:

California	38332521
Texas	26448193
New York	19651127
Florida	19552860
Illinois	12882135

dtype: int64

Area Series

California	423967
Texas	695662
New York	141297
Florida	170312
Illinois	149995
North Carolina	123456

dtype: int64

12. area_dict = {'California': 423967, 'Texas': 695662,
'New York': 141297, 'Florida': 170312, 'Illinois':
149995, 'North Carolina': 123456}

area = pd.Series(area_dict)

Population_density = population / area

print("Population Density:", population_density)

Population Density:

California	80.413926
Florida	114.806121
Illinois	85.883763
New York	139.076746
North Carolina	NaN
Texas	38.018740

dtype: float64

df: Total Population

California	38332521
Texas	26448193
New York	19552860
Illinois	12882135

dtype: int64

df:

d	4.5
b	4.2
a	-5.3
c	3.1

dtype: float64

a	-5.3
b	7.2
c	3.6
d	4.5
e	NaN

dtype: float64

13. male_population_dict = {'California': 19000, 'Texas': 13000, 'New York': 9500, 'Florida': 9600, 'Illinois': 6500}

female_population_dict = {'California': 19223521, 'Texas': 13143193, 'New York': 10151124, 'Florida': }

male_population = pd.Series(male_population_dict)

female_population = pd.Series(female_population_dict)

total_population = male_population + female_population

print(total_population)

14. obj = pd.Series([1.5, 7.2, -5.3, 3.6], index = ['d', 'b', 'a', 'c'])

print(obj)

obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

print(obj2)

EX NO: 3
22.7.25

LOAD THE TITANIC DATASET AND CONVERT INTO DATA FRAME

AIM:

To perform basic preprocessing and Exploratory data analysis (EDA) on the Titanic dataset using pandas, seaborn, and sklearn tools.

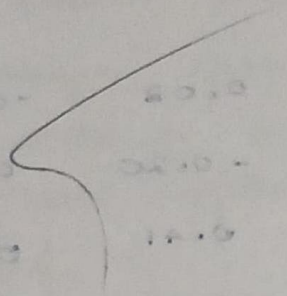
PROCEDURE:

* Load Data into Data Frame.
* Preview Data: Use .head().
* Check Info: .info() and .isnull().sum()
* Handle Missing Age: Apply .ffill() & .bfill() to 'Age'.
* Handle Missing Cabin: Fill with "unknown".
* Remove Duplicates: Use .drop_duplicates().
* Encode 'Sex': Use LabelEncoder.
* Scale 'Fare': Use Standard Scaler
* Pair Plot: For 'Pclass', 'Sex', 'Age', 'SibSp'.
* Heatmap: For 'Pclass', 'Age', 'SibSp', Parch', 'Fare'.

CODE:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,
    StandardScaler
df = sns.load_dataset('titanic')
print("First 5 rows:")
print(df.head())
print("\n Dataset info:")
df.info()
print("\n Missing values:\n", df.isnull().sum())
df['Age_ffill'] = df['age'].ffill()
df['Age_bfill'] = df['age'].bfill()
df['deck'] = df['deck'].cat.add_categories('unkno
df['deck'] = df['deck'].fillna('unknown', limit=5)
df.drop_duplicates(inplace=True)
le = LabelEncoder()
df['sex_encoded'] = le.fit_transform(df['sex'])
scaler = StandardScaler()
df['fare_scaled'] = scaler.fit_transform(df[['fare']])
selected_features = ['pclass', 'sex_encoded', 'age',
                     'sibsp']
sns.pairplot(df[selected_features].dropna())
plt.suptitle("Pair Plot", y=1.02)
plt.show()

corr_features = ['pclass', 'age', 'sibsp', 'parch', 'fare']
corr_matrix = df[corr_features].corr()
```

```
plt.figure(figsize = (8, 6))
sns.heatmap(corr_matrix, annot=True, cmap=
                                "coolwarm", fmt =".2f")
plt.title("Correlation Heatmap")
plt.show()
```



RESULT:

Thus the basic preprocessing & EDA on the titanic dataset using pandas, Seaborn, sklearn tools.

Ex No: 4
Date: 22.7.25

TITANIC DATASET ANALYSIS AND PREPROCESSING USING PANDAS SIMPLE IMPUTER

AIM:

To load Titanic dataset from csv, handle missing values using simple imputer, analyze key passenger features, filter passenger based on conditions, and prepare data for model training and testing.

PROCEDURE:

* load titanic.csv into a data frame.
* Explore dataset shape into summary statistics.
* Use simple imputer to fill missing Age.
* Fill missing cabin with "unknown" & embarked with mode.
* Visualize passenger class distribution with countplot.
* Filter passengers by genders, survival class, age, family abroad and survival status.
* Identify top oldest survivors & zeros for passengers.
* Split training and testing sets.

PROGRAM:

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

Correlation Heatmap

Pclass	+1.00	-0.37	0.09	0.02	-0.55
age	-0.37	1.00	-0.32	-0.20	0.09
SibSp	0.09	-0.32	1.00	0.41	0.15
Parch	0.02	-0.20	0.41	1.00	0.21
fare	-0.55	0.09	0.15	0.21	1.00
	Pclass	age	SibSp	Parch	fare

```
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
df = sns.load_dataset('titanic')
df['age'] = SimpleImputer(strategy='mean').fit_transform(df[['age']])
df['deck'] = df['deck'].cat.add_categories('Unknown')
df['deck'] = df['deck'].fillna('Unknown')
df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])
sns.countplot(x='pclass', data=df); plt.title('Passenger class distribution')
plt.show()
print("Females who survived:", df[(df.sex=="female") & (df.survived==1)].index.tolist())
print("3rd class passengers under 18:", df[(df.pclass==3) & (df.age<18)].index.tolist())
print("1st class passengers older than 40:", df[(df.pclass==1) & (df.age>40)].index.tolist())
print("1st class passengers older than 40 who survived:", df[(df.pclass==1) & (df.age>40) & (df.survived==1)].index.tolist())
print("Male passengers who paid far > 100:", df[(df.sex=="male") & (df.fare>100)].index.tolist())
print("Passengers embarked at 'C' and 1st Class:", df[(df.embarked=='C') & (df.pclass==1)].index.tolist())
print("Passengers with max min 2 siblings/spouses abroad:", df[df.sibsp>2].index.tolist())
```

```
print("Top 5 oldest passengers who survived:")
print(df[df['survived'] == 1].sort_values('age',
ascending = False).head(5)[['age']])
print("passengers who paid zero fare:",
df[df['fare'] == 0].index.tolist())
train_df, test_df = train_test_split(df,
test_size = 0.2, random_state = 42)
print(f"Training set size: {len(train_df)}
Testing set size: {len(test_df)}").
```

REPORT:

The program successfully identifies passengers with zero fare & efficiently splits the dataset into 80% training & 20% testing sets, ensuring reproducibility & readiness for machine learning tasks.

EX NO: 5 SALES PREDICTION AND CUSTOMER
DATE: 28.7.25 SEGMENTATION USING LINEAR REGRESSION AND K MEANS CLUSTERING

AIM:

To predict sales based on advertising budgets across TV, Radio & Newspaper using linear Regression. Additionally, to identify distinct patterns in advertising spend and sales through K-means clustering.

ALGORITHM:

* Read the advertising.csv file containing TV, Radio, Newspaper, budgets and sales.
* Prepare data: select features and target, then split into training & testing splits.
* Train and evaluate the model on training data, predict sales on test data and calculate Mean Squared Error (MSE).
* Scale features & determine data into 3 groups to find patterns.
* Visualize features: plot actual vs predicted sales & visualize clusters to interpret model performance & segments.

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('advertising.csv')
print(df.head())
print(df.describe())
X = df[['TV', 'Radio', 'Newspaper']]
y = df['Sales']
X_train, X_test, y_train, y_test = train_test_split
        (X, y, test_size=0.2, random_state=0)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Linear Regression MSE:", mse)
plt.figure(figsize=(8,5))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Linear Regression: Actual vs
                                predicted")

plt.show()
```

```
scaler = StandardScaler()
scaled = scaler.fit_transform(df['TV', 'Radio',
                               'Newspaper'])
K_means = KMeans(n_clusters = 3, random_state)
df['Cluster'] = K_means.fit_predict(scaled)
plt.figure(figsize = (8, 6))
sns.scatterplot(data = df, x = 'TV', y = 'Sales',
                hue = 'Cluster', palette = "Set 2")
plt.title("K-Means Clustering: TV budget
                               vs Sales")
plt.show().
```

RESULT:

The linear Regression model achieved a Mean Squared Error (MSE) of 4.052, showing a good fit between actual & predicted sales. K-Means clustering grouped the data into 3 distinct clusters, revealing clear patterns in TV advertising budgets & sales.

EX NO: 6
DATE: 29.07.25

TEXT PREPROCESSING AND ANALYTICS PIPELINE

AIM:

To clean & preprocess Amazon review texts for analysis, then identify the most frequent words.

ALGORITHM:

* Load Dataset

Import Amazon reviews from amazon reviews.csv.

* Clean Text:

Convert text to lowercase, remove punctuation & special characters then tokenize using spacy.

* Remove stopwords:

Filter out common stopwords & punctuation tokens.

* Apply Cleaning:

Process the entire reviews text column to generate cleaned tokens.

* Analyze Frequency:

Flatten all tokens & count word frequencies using Counter.

Output:

we got this usp for my band who is an (OTR).....

1. I'm a professional OTR truck driver, and I roll.....
2. well, what can I say. I've had his wife in m.....
3. Not going to write a long review, ever magit....

Name: review Text, dtype: Object

review Text

0 we got this aps for my band who is an (OTR)

1. I'm a professional OTR truck driver & I trou...
2. well what can I say.
3. Not going to write a long review.
4. I've had mine 500 a year & here what we go.

Cleaned_tokens

0 [got, gps, husband, otr, road, ...]

1 [ive, professional, otr, truck, driver]

2 [ive, unit, truck, days, pros]

3 [going, write, long, review]

4 [ive, year, got, route, true]

Top 15 frequent words in Amazon Reviews:

[('c', 4, 3203), ('book', 1447), ('the', 962),
('books', 805), ('kindle', 561),
('seven', 473), ('like', 452), ('read', 451),
('great', 422), ('use', 420)]

PROGRAM:

```
import pandas as pd
import re
import spacy
nlp = spacy.load("en_core_web_sm")
df = pd.read_csv("amazon_reviews.csv")
print(df['Review Text'].head())
def clean_text_spacy(text):
if pd.isnull(text):
        return []
text = text.lower()
text = re.sub(r'[^\w\s]', '', text)
text = text.encode("ascii", "ignore").decode("ascii")
doc = nlp(text)
tokens = [token.text for token in doc
  if not token.is_stop & not token.is_
                                  punct]
return tokens
df['cleaned_tokens'] = df['review Text'].
                apply(clean_text_spacy)
print(df['review Text', 'cleaned_tokens']
                .head(5)).
```

RESULT:

Thus cleaned tokens contain meaningful words without noise, and the top is frequent words reflect key terms from the review.

NLP

Ex: 5

DATE: 5.9.25

AIM:

To analyze text using Part of Speech (POS) tagging (with spaCy) and for grammatical structure identification.

ALGORITHM:

* Load language model. Load en_core in spaCy for POS tagging.
* Text Preprocessing Pass the input text to spaCy & extract POS tags.
* Prepare Corpus Collect sample documents & append the user query.
* Vectorization Convert documents into TF-IDF vectors.
* Similarity & Ranking - Compute Cosine similarity between query & documents, then rank by score.

o/p:

POS Tagging:

AI → PROPN

driven → VERB

Platforms → NOUN

Personalize → VERB

learning → VERB

Paths → NOUN

and → CCOUNT

help → VERB

Students → NOUN

grasp → VERB

Concepts → NOUN

faster → ADV

→ PUNCT

Top relevant documents:

Score : 0.16 → AI helps automate grading & administrative tasks in schools.

Score : 0.10 → Intelligent tutoring systems adapt to each students learning style.

PROGRAM:

```
pip install spacy
import spacy
nlp = spacy.load("en_core_web_sm")
text = "AI-driven platforms personalize learning paths & help students
group concepts faster."
doc = nlp(text)
for token in doc:
    print(f"{token.text:15} -> {token.pos_}")
from sklearn.metrics.pairwise import
                    cosine_similarity
documents = [
    "AI tools analyze student
Performance & provide real time
feedback" Intelligent tutoring system
adapts to each students learning
style".
query = "How does AI support
students in learning?"
corpus = documents + [query]
```

```
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform
                    (corpus)
similarities = cosine_similarity(tfidf_matr
[-1], tfidf_matrix[:-1]).flatten()
ranked_docs = sorted(zip(similarities,
        documents), reverse=True)
print("\nTop relevant documents:\n")
for score, doc in ranked_docs:
        print(f"Score: {score:.2f}\n{doc}")
```

RESULT:

Thus, we have analyzed text using POS & for grammatical structure identification.

6/9/25

Ex no:6 Exploratory Data Analysis

DATE: 9.9.25 with Pytorch

AIM:

To explore and analyze the Netflix dataset using Pytorch in order to uncover insights about content types, release trends, countries of production & genre distributions.

ALGORITHM:

* Load datasets & scale features as needed.

* Apply Kmeans clustering & Mall Customers data & use the Elbow method to find the optimal clusters.

* Perform multiple KMeans clusterings & the wine dataset with varying cluster counts.

* Bind a similarity matrix from base clusterings & apply spectral clustering for ensemble labels.

OUTPUT:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Data columns (total 12 columns):

#	Column	Non-Null Count	Dtype
0	show_id	8807	non-null object
1	type	8807	non-null object
2	title	8807	non-null object
3	Director	6173	non-null object
4	cast	7982	non-null object
5	country	7982	non-null object
6	date_added	8797	non-null object

dtypes: int64(1), object(11)
memory usage: 825.8+ KB
None

	show_id	type	title	director	cast	country
1		Ama Qamata	Khosi Ngema	Gail Mabalane	Thaban	South Africa
2		Sami Bouajila	Tracy Gotoas	Samuel Jay	Nabi	NaN
3.		Mayur More	Jitendra Kumar	Rajan Raj	Alam K	India

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import
                    StandardScaler
df = pd.read_csv("netflix_titles.csv")
print(df.info())
print(df.head())
print(df.describe(include="all"))
print("Number of unique countries:",
                df['country'].unique())
print(df['type'].value_counts())
print(df['release_year'].value_counts().head(
print(df.groupby(['country','type']).size().
        sort_values(ascending=False).
        head(10))
df['date_added'] = pd.to_datetime(df
        ['date_added'], format="mixed",
        errors="coerce")
df.set_index('date_added', inplace=True)
monthly_content = df.resample('M').size()
```

	date_added	release_year	rating	duration
0	September 25, 2021	2020	PG-13	90 min
1	September 24, 2021	2021	TV-MA	2 Seasons
2	September 24, 2021	2021	TV-MA	1 Season
3	September 24, 2021	2021	TV-MA	1 Season
4	September 24, 2021	2021	TV-MA	2 Seasons

listed_in

0 Documentaries

1 International TV Shows, TV Dramas

2 Crime TV Shows, TV Act

3 Docuseries, Reality TV

4 International TV Shows, Romantic TV Shows

description

0 As his country mourns the end of his life, former...

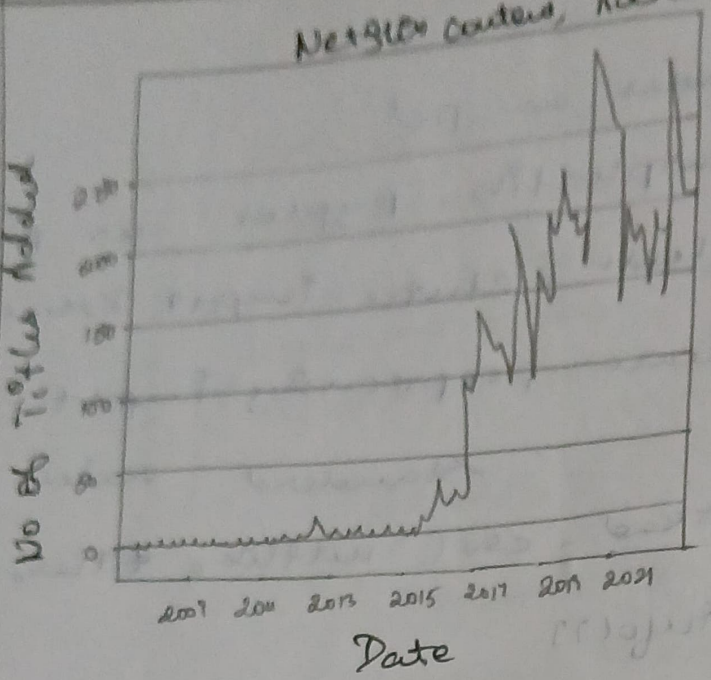
1 After crossing paths at a party, a Cape Town t...

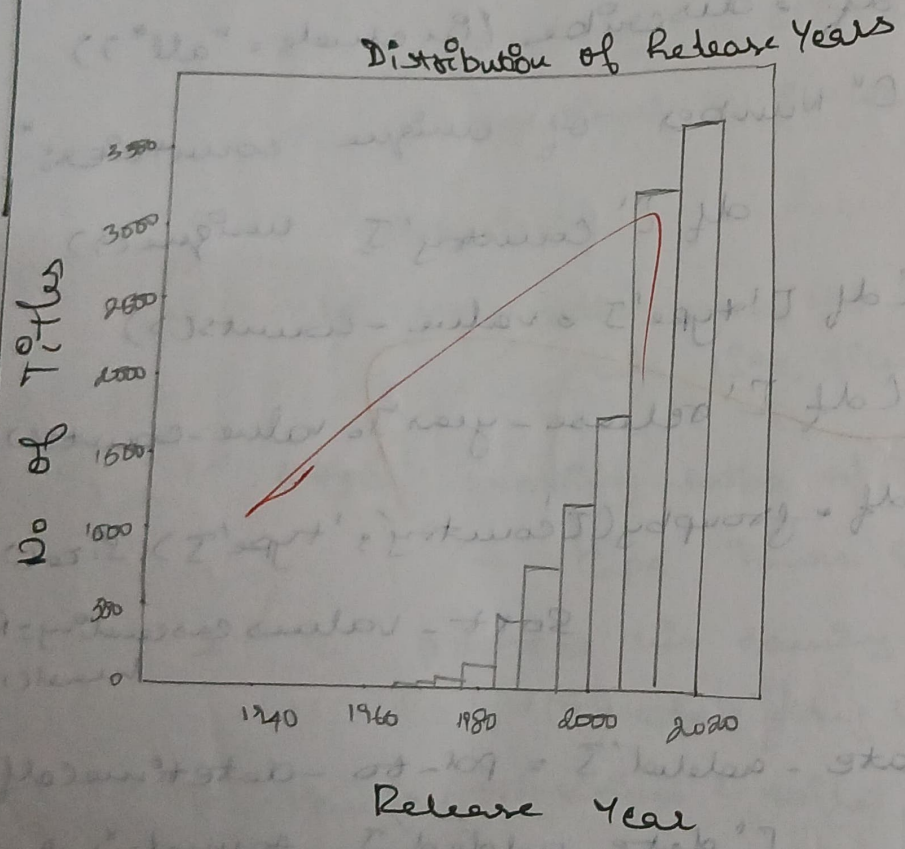
2 To protect his family from a powerful drug lor...

3 Feuds, flirtations & toilet talk go down amo...

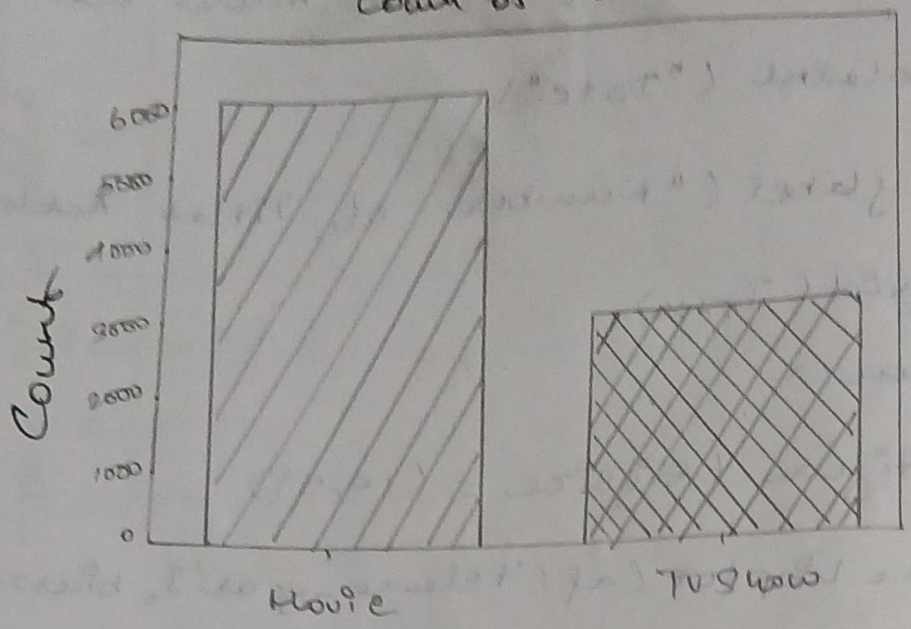
4 In a city of coastal contrasts, two women trust...

```
plt.title("Netflix Content Added Over Time")
plt.xlabel("Date")
plt.ylabel("Number of Titles Added")
plt.grid(True)
plt.show()
plt.figure(figsize=(10,5))
sns.histplot(df['release_year'], bins=30, kde=False)
plt.title("Distribution of Release Years")
plt.xlabel("Release Year")
plt.ylabel("Number of Titles")
plt.show()
sns.countplot(data=df, x="type", palette="Set2")
plt.title("Count of Movies & TV Shows")
plt.show()
top_countries = df['country'].value_counts().head(10)
top_countries.plot(kind="bar", color="skyblue")
plt.title("Top 10 countries by Number of Titles")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
genres = df["listed_in"].str.split(', ', expand=True).stack()
```

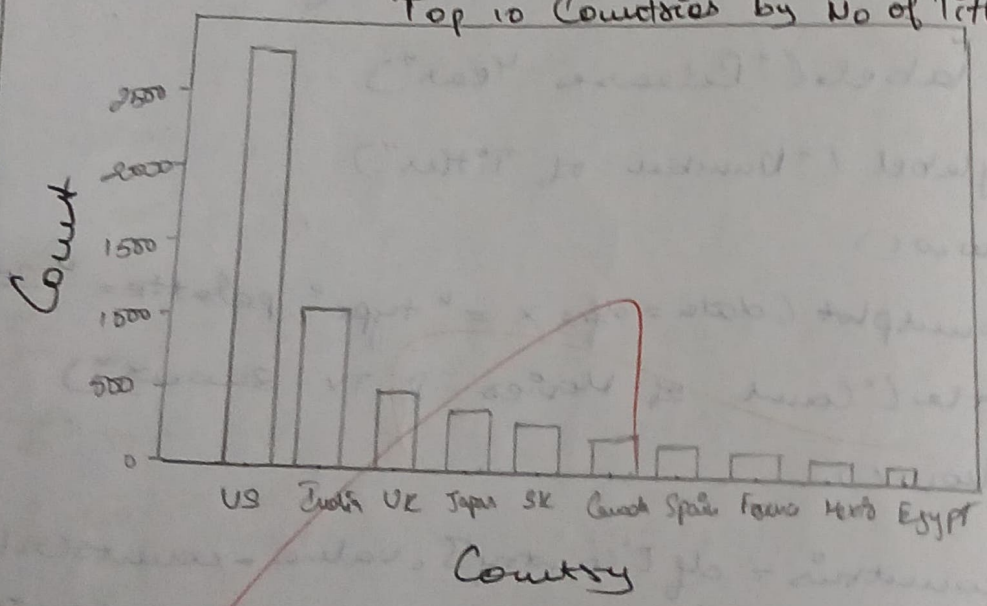

No of Titles Added
0
Date


Distribution of Release Years
No of Titles
1940
1960
1980
2000
2020
Release Year

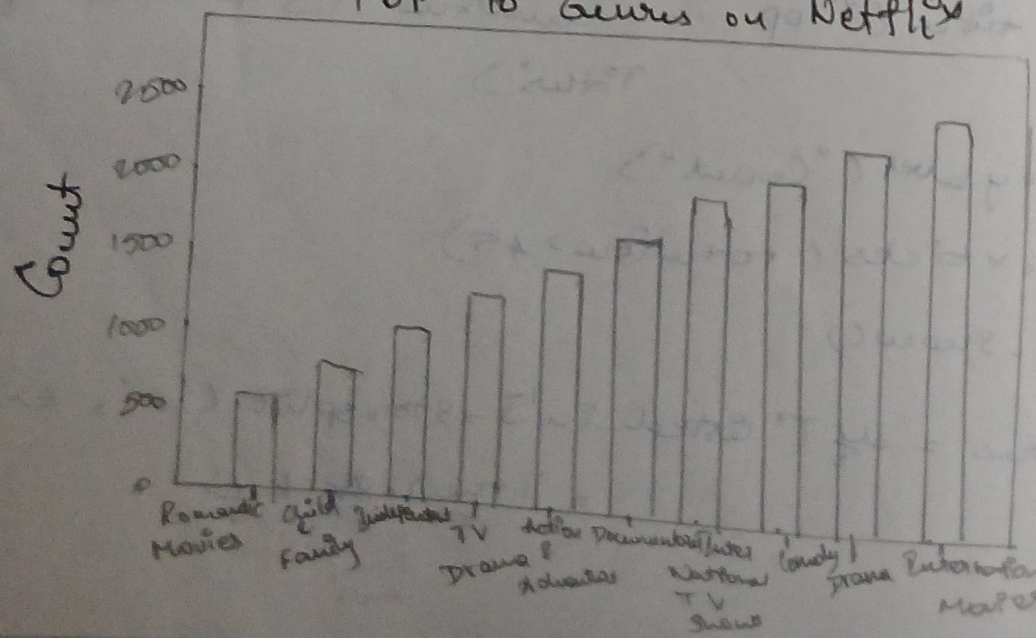
Count of Movies & TV Shows



Top 10 Countries by No of Titles



Top 10 Genres on Netflix



```
top_genres = genres.value_counts().head(10)
top_genres.plot(kind='barh', colour='coral')
plt.title("Top 10 Genres on Netflix")
plt.xlabel("Count")
plt.gca().invert_yaxis()
plt.show()
```

RESULT:

Thus implement & analyze the Netflix dataset using Python is implement & executed successfully.

EX NO: 4

DATE: 16.9.25

CLUSTERING

AIM:

* Mall Customers: To group customers using K Means clustering & find the optimal number of clusters with the Elbow Method.

* Wine Dataset: To apply CSPA ensemble clustering with spectral clustering & visualize clusters using PCA.

ALGORITM:

* MALL CUSTOMERS (K Means):

* Load Dataset & select features.

* Apply K Means Clustering.

* Use Elbow Method to determine best K.

* Visualize Custers with Scatter plot.

* WINE DATASET:

* Load & Standardize data.

* Run K-Means with different k values.

* Build similarity matrix.

* Apply spectral clustering.

* Evaluate with silhouette score.

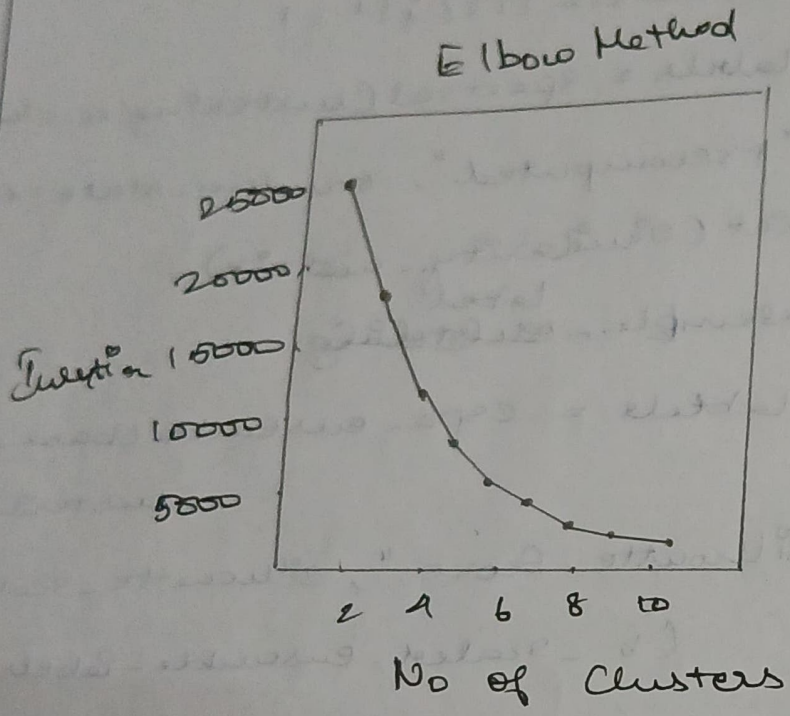
* Reduce dimensions with PCA and plot clusters.

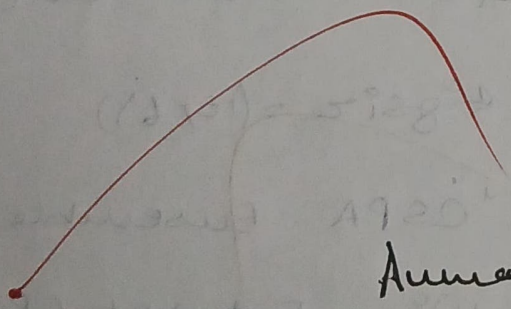
CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import
                    Standard Scaler
import seaborn as sns
df = pd.read_csv("Mall_Customers.csv")
kmeans = KMeans(n_clusters = 5, random_state=42)
df['Cluster'] = kmeans.fit_predict()
distortions=[]
for i in range(1,11):
 km = KMeans(n_cluster=9)
km.fit(df['Annual Income(k$)', 'Spending
                    Score(1-100)'])

distortions.append(km.inertia_)
```

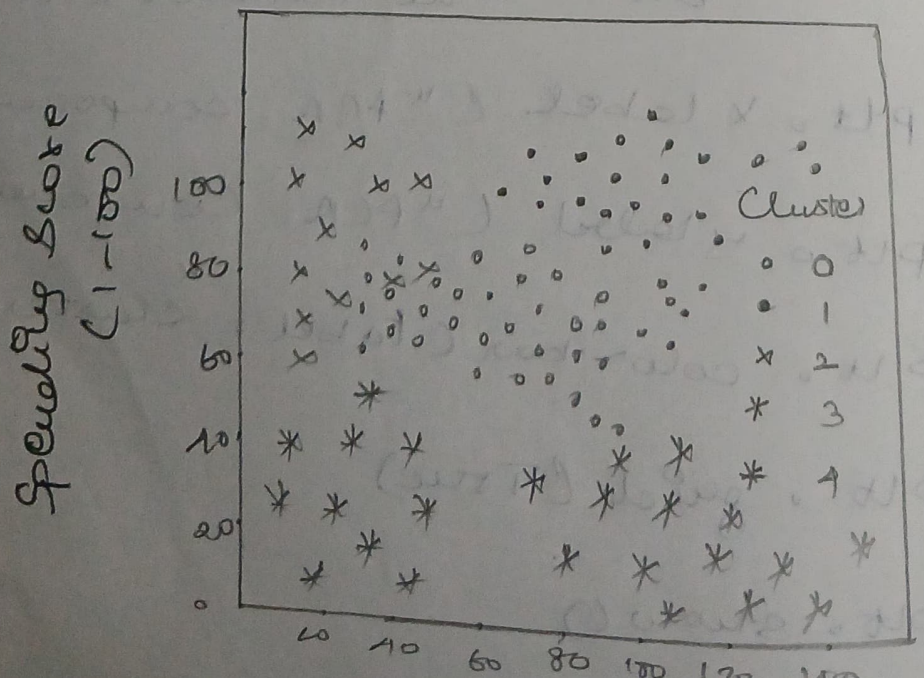
```
plt.plot(range(1,11), distortions, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()
sns.scatterplot(data=df, x="Annual Income(k$)",
                y="Spending Score(1-100)", hue="Cluster",
                palette="Set2")
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
wine = load_wine()
X = pd.DataFrame(wine.data, columns=wine.feature_names)
X_scaled = StandardScaler().fit_transform(X)
base_clusterings = []
for k in [2, 3, 4, 5]:
    km = KMeans(n_clusters=k, random_state=42)
    base_clusterings.append(km.fit_predict(X_scaled))
```

```
for clustering in clusterings:
    for i in range(n_samples):
        for j in range(n_samples):
            if clustering[i] == clustering[j]:
                similarity_matrix[i][j] += 1
base_labels = SpectralClustering(n_clusters=3,
    affinity="precomputed", random_state=42).
    fit_predict(similarity_matrix)
return ensemble_labels
ensemble_labels = cspa_ensemble(base_
                                clusterings)
print("Silhouette Score:", silhouette_score
                    (b_scaled, ensemble_labels))

pca = PCA(n_components=2)
b_pca = pca.fit_transform(b_scaled)
plt.figure(figsize=(10,6))
plt.title("CSPA Ensemble Clustering
    on Wine Dataset (PCA-reduced)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label="Cluster label")
plt.grid(True)
plt.show()
```





Annual Income (K$)



RESULT:

* In the Mall Customers dataset, customers are successfully grouped into distinct clusters showing spending behaviour patterns.

* In the wine dataset, CSPA ensemble clustering improved clustering quality confirmed by a good Silhouette score & clear cluster separation in PCA visualization.

Ex No: 8 Interactive Dashboards with Tableau using Google Sheets

AIM:

To create an interactive Tableau dashboard that integrates data from Google sheets & visualizes it using bar charts, line plots and scatter plots, allowing dynamic exploration of sales or demographic data.

MATERIALS REQUIRED:

* Tableau Desktop or Tableau Public
* Google account with Google Sheets.
* Sample Sales or demographic dataset on Google Sheets.
* Internet connection.

PROCEDURE:

* Prepare the data.
* Connect Tableau to Google sheets.
* Create Individual visualizations.

=> Bar charts (Sales by product or Region)

⇒ Line plot (sales over time)

⇒ Scatter plot (customer Demographics or profit vs sales)

* Build the Dashboard.

* Publish or share.

RESULT:

Created an Interactive Tableau dashboard linked to Google sheets, allowing users to explore sales trends, compare product performance & analyze demographics.