1. **WAPP distribution of wine quality, calculates skewness and kurtosis, and applies a log transformation to the alcohol column. .**

```
from scipy.stats import skew, kurtosis
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np

df=pd.read_csv('winequality_red.csv')
df.head(2)
sns.countplot(x='quality', data=df, palette='viridis')
plt.title("Distribution of Wine Quality")
plt.xlabel("Wine Quality")
plt.ylabel("Count")
plt.show()
a_s=skew(df['alcohol'])
a_k=kurtosis(df['alcohol'])
print(a_k)
print(a_s)
df['log_a']=np.log(df['alcohol'])
df
```

2.Loads the Titanic dataset, shows basic information, checks for missing values, calculates basic statistics, and imputes missing values for the Age column.

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

df = pd.read_csv('titanic.csv')

print("First 5 rows:\n", df.head())

print("\nDataset Info:")
print(df.info())

print("\nMissing Values:\n", df.isnull().sum())
print("\nStatistical Summary:\n", df.describe())

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

print("\nMissing Values after imputation:\n", df.isnull().sum())
```

Write a program in python to print the number of outliers. Generate 200 samples, from a normal distribution, centered around the value 100, with a standard deviation of 5.

```
import numpy as np

data = np.random.normal(100, 5, 200)

threshold = 2

mean = np.mean(data)
std_dev = np.std(data)

z_scores = (data - mean) / std_dev

outliers = np.where(np.abs(z_scores) >
threshold)[0]

print("Number of outliers:", len(outliers))
print("Outlier indices:", outliers)
```

Consider following observations/data. And apply simple linear regression and find out estimated coefficients b1 and b1 Also analyze the performance of the model
x = np.array([1,2,3,4,5,6,7,8])
y = np.array([7,14,15,18,19,21,26,23])

```
import numpy as np
from sklearn import linear_model

X = np.array([0, 1, 2, 3, 4]).reshape((-1,1))
y = np.array([2, 3, 5, 4, 6])

model = linear_model.LinearRegression()
model.fit(X, y)

b0 = model.intercept_
b1 = model.coef_[0]

print("b0 = ",b0)
print("b1 = ",b1)
```

Consider following observations/data. And apply simple linear regression and find out estimated coefficients b0 and b1.(use numpy package)
x= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,11,13]
y = ([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,16, 18]

```
import numpy as np

x = np.array([0, 1, 2, 3, 4])
y = np.array([2, 3, 5, 4, 6])

n = len(x)
mean_x = np.mean(x)
mean_y = np.mean(y)

SS_xy = np.sum(x * y) - n * mean_x * mean_y
SS_xx = np.sum(x * x) - n * mean_x * mean_x

b1 = SS_xy / SS_xx
b0 = mean_y - b1 * mean_x

print("Estimated coefficient b1 (slope):",b1)
print("Estimated coefficient b0 (y-intercept):",b0)
```

Write a python program to implement multiple Linear Regression model for a car dataset. Dataset can be downloaded from: https://www.w3schools.com/python/python_ml_multiple_regression.asp

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("Car.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[2300, 1300]])
print(predictedCO2)

b0 = regr.intercept_
b1 = regr.coef_[0]
b2 = regr.coef_[1]

print("b0 = ",b0)
print("b1 = ",b1)
print("b2 = ",b2)
```

Write a Python program to build an SVM model to Cancer dataset.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score
from sklearn.metrics import recall_score, classification_report

dataset=pd.read_csv('breast_cancer_dataset.csv')
X = dataset.iloc[:, :-2].values
y = dataset.iloc[:, 30].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("\n Model Evaluation on Test Data:")
print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print("\n Classification Report:")
print(classification_report(y_test, y_pred))
```

Consider following dataset weather= ['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy', 'Sunny','Overcast','Overcast','Rainy'] temp

```
weather=['Sunny','Sunny','Overcast','Rainy',
      'Rainy','Rainy','Overcast','Sunny',
      'Sunny','Rainy','Sunny','Overcast',
      'Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool',
    'Cool','Mild','Cool','Mild','Mild','Mild',
    'Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes',
    'No','Yes','Yes','Yes','Yes','Yes','No']

from sklearn import preprocessing
le = preprocessing.LabelEncoder()

w=le.fit_transform(weather)
t=le.fit_transform(temp)
p=le.fit_transform(play)

features = list(zip(w,t))
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(features,p)
predicted = model.predict([[0,2]])
print("Predicted Value:",predicted)
```

WAPP build Decision Tree Classifier for shows.csv from pandas and predict class label for show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7?

```
import pandas
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

dataset = pandas.read_csv("Actors.csv")

d = {'UK': 0, 'USA': 1, 'N': 2}
dataset['Nationality'] = dataset['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
dataset['Go'] = dataset['Go'].map(d)

x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

dtree = DecisionTreeClassifier()
dtree = dtree.fit(x, y)

tree.plot_tree(dtree,feature_names=x)

print(dtree.predict([[40, 10, 7, 1]]))
```

WAPP to implement heirarchical Agglomerative clustering algo

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

dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values

import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()

from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, metric = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)

plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Write a python program to implement complete data pre-processing

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split

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

df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Salary'].fillna(df['Salary'].mean(), inplace=True)

le = LabelEncoder()
df['Category'] = le.fit_transform(df['Category'])

scaler = MinMaxScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])

X = df.drop("Target", axis=1)
y = df["Target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
print("y_train:", y_train.shape)
print("y_test:", y_test.shape)
```

Write a python program to demonstrate the use of PCA

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("iris.csv")
X = df.iloc[:, :-1]
y = df['target']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis',
edgecolor='k', s=150)
plt.title('PCA on Iris Dataset (Using CSV File)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Target Class')
plt.show()
```

Write a python program to implement LDA use any synthetic dataset.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA

df = pd.read_csv("iris.csv")
X = df.iloc[:, :-1]
y = df['target']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

lda = LDA(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)

plt.figure(figsize=(8, 6))
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='viridis', edgecolor='k',
s=150)
plt.title('LDA on Iris Dataset (Using CSV File)')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
plt.colorbar(label='Target Class')
plt.show()
```

Write a python program applies t-SNE

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE

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

X = df.iloc[:, :-1]   # Feature columns
y = df['target']      # Target column

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

tsne = TSNE(n_components=2, perplexity=30, n_iter=1000,
random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis',
edgecolor='k', s=150)
plt.title('t-SNE on Iris Dataset (Using CSV File)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(label='Target Class')
plt.show()
```

Convert categorical values into numeric format

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

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

categorical_cols = df.select_dtypes(include='object').columns

le = LabelEncoder()
for col in categorical_cols:
   df[col] = le.fit_transform(df[col])

print(df.head())
```

Rescale data between 0 and 1 (use inbuilt dataset)

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

data = load_iris()
X = data.data

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

scaled_df = pd.DataFrame(X_scaled, columns=data.feature_names)
print(scaled_df.head())
```

Train-test split

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print("Training shape:", X_train.shape)
print("Testing shape:", X_test.shape)
```

Find all null values and remove them

```
import pandas as pd

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

print("Null values before removing:")
print(df.isnull().sum())

df_clean = df.dropna()

print("\nNull values after removing:")
print(df_clean.isnull().sum())
```