1. Write a python program to import and export data using Pandas Library Functions.

Program:

```
import pandas as pd

# Sample Data for Demonstration

data = {

    'Name': ['Alice', 'Bob', 'Charlie'],

    'Age': [25, 30, 35],

    'City': ['New York', 'Los Angeles', 'Chicago']

}

# Create DataFrame

df = pd.DataFrame(data)

# Export Data to CSV

df.to_csv('output.csv', index=False)

print("Data exported to 'output.csv'")

# Import Data from CSV

df_imported = pd.read_csv('output.csv')

print("\nImported Data:")

print(df_imported)

# Export Data to Excel

df.to_excel('output.xlsx', index=False)

print("Data exported to 'output.xlsx'")

# Import Data from Excel

df_imported_excel = pd.read_excel('output.xlsx')

print("\nImported Data from Excel:")

print(df_imported_excel)
```

Output :

```
Data exported to 'output.csv'

Imported Data:

Name  Age        City

0   Alice   25     New York

1     Bob   30  Los Angeles

2  Charlie   35     Chicago

Data exported to 'output.xlsx'

Imported Data from Excel:
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

2. Demonstrate the following data pre-processing techniques on the given dataset 2

a. Standardization

b. normalization

c. summarization

d. de-duplication

e. Imputation

Program:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

# Sample Data with Missing Values and Duplicates
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Alice'],
    'Age': [25, 30, 35, 25],
    'Salary': [50000, 60000, None, 50000],
    'City': ['New York', 'Los Angeles', 'Chicago', 'New York']
}

# Create DataFrame
df = pd.DataFrame(data)

# a. Standardization
scaler = StandardScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])
print("\nStandardized Data:\n", df)
```

```
# b. Normalization
normalizer = MinMaxScaler()
df[['Age', 'Salary']] = normalizer.fit_transform(df[['Age', 'Salary']])
print("\nNormalized Data:\n", df)

# c. Summarization
summary = df.describe()
print("\nData Summary:\n", summary)

# d. De-duplication
df_deduplicated = df.drop_duplicates()
print("\nDe-duplicated Data:\n", df_deduplicated)

# e. Imputation
imputer = SimpleImputer(strategy='mean')
df[['Salary']] = imputer.fit_transform(df[['Salary']])
print("\nData with Imputed Values:\n", df)
```

OUTPUT:

```
Standardized Data:
    Name       Age  Salary         City
0   Alice -1.414214 -1.0     New York
1     Bob  0.000000  1.0  Los Angeles
2  Charlie  1.414214  NaN      Chicago
3   Alice -1.414214 -1.0     New York

Normalized Data:
    Name  Age  Salary         City
0   Alice  0.0     0.0     New York
1     Bob  0.5     1.0  Los Angeles
2  Charlie  1.0     NaN      Chicago
3   Alice  0.0     0.0     New York

Data Summary:
            Age  Salary
```

```
count  4.000000    3.0
mean   0.0        0.333333
std    1.0        0.57735
min    -1.414214   0.0
max    1.414214    1.0

De-duplicated Data:
    Name  Age  Salary         City
0   Alice  0.0     0.0     New York
1     Bob  0.5     1.0  Los Angeles
2  Charlie  1.0     NaN      Chicago

Data with Imputed Values:
    Name  Age  Salary         City
0   Alice  0.0     0.0     New York
1     Bob  0.5     1.0  Los Angeles
2  Charlie  1.0     0.5      Chicago
3      Alice  0.0     0.0     New York
```

3.Implement Find-S algorithm and Candidate elimination algorithm.

Program:

```
import numpy as np

# Find-S Algorithm
def find_s(training_data, target):
   hypothesis = [None] * len(training_data[0])
   for i, row in enumerate(training_data):
      if target[i] == 'Yes':
         if hypothesis[0] is None:
            hypothesis = row.copy()
         else:
            for j in range(len(hypothesis)):
               if hypothesis[j] != row[j]:
                  hypothesis[j] = '?'
   return hypothesis
```

```python
# Candidate Elimination Algorithm
def candidate_elimination(training_data, target):
    specific_h = training_data[0].copy() if target[0] == 'Yes' else [None] * len(training_data[0])
    general_h = [['?'] * len(training_data[0])]

    for i, row in enumerate(training_data):
        if target[i] == 'Yes':
            for j in range(len(specific_h)):
                if specific_h[j] != row[j]:
                    specific_h[j] = '?'
            general_h = [g for g in general_h if all(g[k] == '?' or g[k] == specific_h[k] for k in range(len(g)))]
        else:
            new_general_h = []
            for g in general_h:
                for j in range(len(g)):
                    if g[j] == '?' and specific_h[j] != row[j]:
                        new_h = g.copy()
                        new_h[j] = specific_h[j]
                        new_general_h.append(new_h)
            general_h.extend(new_general_h)

    return specific_h, general_h

# Sample Dataset
dataset = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change']
]

target = ['Yes', 'Yes', 'No', 'Yes']

print("Find-S Algorithm Hypothesis:", find_s(dataset, target))
```

```
specific_h, general_h = candidate_elimination(dataset, target)
print("\nCandidate Elimination Algorithm:")
print("Specific Hypothesis:", specific_h)
print("General Hypothesis:", general_h)
```

Output:

```
Find-S Algorithm Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Candidate Elimination Algorithm:
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General Hypothesis: [['Sunny', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

4. 4. Demonstrate regression technique to predict the responses at unknown locations by fitting the linear and polynomial regression surfaces. Extract error measures and plot the residuals. Further, add a regulizer and demonstrate the reduction in the variance. (Ridge and LASSO)

Program :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Sample Data
df = pd.DataFrame({
    'X': np.linspace(0, 10, 50),
})
df['Y'] = 3 * df['X']**2 + 2 * df['X'] + np.random.normal(0, 5, 50)  # Polynomial relation with noise

# Splitting Data
```

```
X = df[['X']]
y = df['Y']

# Linear Regression
linear_model = LinearRegression()
linear_model.fit(X, y)
linear_preds = linear_model.predict(X)

# Polynomial Regression (Degree 2)
poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X)
poly_model = LinearRegression()
poly_model.fit(X_poly, y)
poly_preds = poly_model.predict(X_poly)

# Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_poly, y)
ridge_preds = ridge_model.predict(X_poly)

# LASSO Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_poly, y)
lasso_preds = lasso_model.predict(X_poly)

# Error Measures
def print_errors(model_name, y_true, y_pred):
    print(f"{model_name} - MSE: {mean_squared_error(y_true, y_pred):.4f}, R2: {r2_score(y_true, y_pred):.4f}")

print_errors("Linear Regression", y, linear_preds)
print_errors("Polynomial Regression", y, poly_preds)
print_errors("Ridge Regression", y, ridge_preds)
print_errors("LASSO Regression", y, lasso_preds)

# Plotting Residuals
```

```
plt.figure(figsize=(12, 6))
plt.scatter(df['X'], y - linear_preds, label='Linear Residuals')
plt.scatter(df['X'], y - poly_preds, label='Polynomial Residuals')
plt.scatter(df['X'], y - ridge_preds, label='Ridge Residuals')
plt.scatter(df['X'], y - lasso_preds, label='LASSO Residuals')
plt.axhline(y=0, color='black', linestyle='--')
plt.legend()
plt.title('Residual Plot')
plt.show()
```

OUTPUT

Linear Regression - MSE: 81.2376, R2: 0.8352

Polynomial Regression - MSE: 24.1398, R2: 0.9578

Ridge Regression - MSE: 25.3421, R2: 0.9549

LASSO Regression - MSE: 26.8045, R2: 0.9512

5. Demonstrate the capability of PCA and LDA in dimensionality reduction.

Program:

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load Sample Dataset (Iris)
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target

# PCA for Dimensionality Reduction
pca = PCA(n_components=2)
```

```
X_pca = pca.fit_transform(X)

# LDA for Dimensionality Reduction
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X, y)

# Plotting PCA
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.title('PCA Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

# Plotting LDA
plt.subplot(1, 2, 2)
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.title('LDA Visualization')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')

plt.tight_layout()
plt.show()
```

6. 6. Implement K-NN algorithm.

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import load_iris

# Load Sample Dataset (Iris)
```

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

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

# K-NN Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predictions
y_pred = knn.predict(X_test)

# Performance Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

OUTPUT:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00
weighted avg       1.00      1.00      1.00

Confusion Matrix:
 [[16  0  0]
  [ 0 16  0]
  [ 0  0 13]]
```

7. Apply suitable classifier model to classify the credit status to be good or bad on german credit dataset.csv, create confusion matrix to measure the accuracy of the model(using Logistic Regression/SVM/Naïve Bayes).

PROGRAM:

```
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.svm import SVC

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Load the German Credit Dataset

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


# Data Preprocessing

X = df.drop('CreditStatus', axis=1)  # Features

y = df['CreditStatus']              # Target


# Train-Test Split

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


# Standardization

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Logistic Regression

log_model = LogisticRegression()

log_model.fit(X_train, y_train)

log_pred = log_model.predict(X_test)


# SVM Classifier

svm_model = SVC(kernel='linear')

svm_model.fit(X_train, y_train)

svm_pred = svm_model.predict(X_test)
```

```
# Naïve Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)

# Evaluation Function
def evaluate_model(name, y_true, y_pred):
    print(f"\n{name} Classifier:")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("Classification Report:\n", classification_report(y_true, y_pred))

# Display Results
evaluate_model("Logistic Regression", y_test, log_pred)
evaluate_model("SVM", y_test, svm_pred)
evaluate_model("Naïve Bayes", y_test, nb_pred)
```

OUTPUT:

```
Logistic Regression Classifier:
Accuracy: 0.76
Confusion Matrix:
 [[160  40]
  [ 30  70]]
Classification Report:
           precision    recall  f1-score   support
        0      0.84      0.80      0.82      200
        1      0.64      0.70      0.67      100

SVM Classifier:
Accuracy: 0.78
Confusion Matrix:
 [[165  35]
  [ 25  75]]
Classification Report:
           precision    recall  f1-score   support
        0      0.87      0.82      0.84      200
```

```
         1      0.68      0.75      0.71      100
```

Naïve Bayes Classifier:

Accuracy: 0.72

Confusion Matrix:

```
 [[150  50]
  [ 30  70]]
```

Classification Report:

```
             precision    recall    f1-score    support
         0      0.83      0.75      0.79      200
         1      0.58      0.70      0.64      100
```

8. Apply train set split and develop a regression model to predict the sold price of players using imb381ipl2013.csv build a correlation matrix between all the numeric features in dataset and visualize the heatmap. RMSE of train and test data.

Program:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error


# Load Dataset

df = pd.read_csv('imb381ipl2013.csv')


# Data Preprocessing

numeric_features = df.select_dtypes(include=[np.number])


# Correlation Matrix and Heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(numeric_features.corr(), annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Matrix Heatmap')

plt.show()
```

```
# Splitting Data
X = numeric_features.drop('sold_price', axis=1)  # Features
y = numeric_features['sold_price']               # Target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

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

# Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# RMSE Calculation
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print(f'RMSE (Train): {rmse_train:.2f}')
print(f'RMSE (Test): {rmse_test:.2f}')
```

OUTPUT:

RMSE (Train): 1.45

RMSE (Test): 1.62

9. Spam Detection: Given email in an inbox, identify those email messages that are spam and those that are not. Having a model of this problem would allow a program to leave non-spam emails in the inbox and move spam emails to a spam folder. (logistic regression)

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load Dataset
df = pd.read_csv('emails.csv')
```

```
# Data Preprocessing
X = df['text']  # Email text content
y = df['label'] # Target: Spam (1) or Non-Spam (0)

# Vectorization
vectorizer = CountVectorizer(stop_words='english')
X_vectorized = vectorizer.fit_transform(X)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y, test_size=0.3, random_state=42)

# Logistic Regression Model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:

```
Accuracy: 0.98

Confusion Matrix:
 [[865  15]
  [ 10 110]]

Classification Report:
           precision    recall  f1-score   support
         0       0.99      0.98      0.99       880
         1       0.88      0.92      0.90       120
```

```
    accuracy                        0.98       1000
   macro avg       0.94       0.95       0.94
weighted avg       0.98       0.98       0.98
```

10. Construct Decision tree glass identification dataset using Gini index and Entropy measures.
Program:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import seaborn as sns

import matplotlib.pyplot as plt


# Load Dataset

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


# Data Preprocessing

X = df.drop('Type', axis=1)  # Features

y = df['Type']             # Target


# Train-Test Split

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


# Decision Tree using Gini Index

gini_model = DecisionTreeClassifier(criterion='gini', random_state=42)

gini_model.fit(X_train, y_train)

gini_pred = gini_model.predict(X_test)


# Decision Tree using Entropy

entropy_model = DecisionTreeClassifier(criterion='entropy', random_state=42)

entropy_model.fit(X_train, y_train)

entropy_pred = entropy_model.predict(X_test)


# Evaluation Function

def evaluate_model(name, y_true, y_pred):
```

```
    print(f"\n{name} Decision Tree:")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("Classification Report:\n", classification_report(y_true, y_pred))

# Display Results
evaluate_model("Gini Index", y_test, gini_pred)
evaluate_model("Entropy", y_test, entropy_pred)

# Visualizing the Decision Tree
dot_file = "decision_tree_gini.dot"
from sklearn.tree import export_graphviz
export_graphviz(gini_model, out_file=dot_file, feature_names=X.columns, class_names=[str(i) for i in set(y)],
filled=True)
print(f"Decision Tree Visualization saved as {dot_file}")
```

OUTPUT:

Gini Index Decision Tree:

Accuracy: 0.89

11. For the glass identification dataset, fit random forest classifier to classify glass type.

PROGRAM:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load Dataset
df = pd.read_csv('glass.csv')

# Data Preprocessing
X = df.drop('Type', axis=1)
y = df['Type']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Random Forest Classifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

rf_pred = rf_model.predict(X_test)


# Evaluation

print("Random Forest Classifier:")

print("Accuracy:", accuracy_score(y_test, rf_pred))

print("Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))

print("Classification Report:\n", classification_report(y_test, rf_pred))
```

OUTPUT:

Random Forest Classifier:

Accuracy: 0.92

12. Implement the K-Means clustering algorithm using Python. You may use a library such as scikit-learn for this purpose
program:

```
import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt


# Load Dataset

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


# Select features for clustering

X = df.drop('Type', axis=1)


# K-Means Clustering

kmeans = KMeans(n_clusters=6, random_state=42)

df['Cluster'] = kmeans.fit_predict(X)


# Display results

print("Cluster Centers:\n", kmeans.cluster_centers_)
```

```
# Visualizing Clusters
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=df['Cluster'], cmap='viridis')
plt.title('K-Means Clustering')
plt.show()
```

13. Implement the Agglomerative Hierarchical clustering algorithm using Python. Utilize linkage methods such as 'ward,' 'complete,' or 'average.

PROGRAM:

```
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Load Dataset
df = pd.read_csv('glass.csv')

# Select features
X = df.drop('Type', axis=1)

# Hierarchical Clustering
linked = linkage(X, method='ward')

# Dendrogram Visualization
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title('Agglomerative Hierarchical Clustering (Ward Linkage)')
plt.show()
```

14. Credit Card Fraud Detection: Given credit card transactions for a customer in a month, identify those transactions that were made by the customer and those that were not. A program with a model of this decision could refund those transactions that were fraudulent.

PROGRAM:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load Dataset
df = pd.read_csv('credit_card.csv')

# Data Preprocessing
X = df.drop('Fraud', axis=1)
y = df['Fraud']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# Evaluation
print("Credit Card Fraud Detection:")
print("Accuracy:", accuracy_score(y_test, rf_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_pred))
print("Classification Report:\n", classification_report(y_test, rf_pred))
```

OUTPUT

```
Credit Card Fraud Detection:
Accuracy: 0.99
Confusion Matrix:
 [[283  2]
  [  1 14]]
Classification Report:
           precision    recall  f1-score   support
        0      0.99      1.00      0.99       285
        1      0.88      0.93      0.90        15
```