PROGRAM-01

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
housing_data=fetch_california_housing(as_frame=True)
df=housing_data.frame
print(df.head())
print("\n dataset Summary:")
print(df.describe())
print("\n Basic information about dataset:")
print(df.info())
print(df.shape)
print("\n Missing Values in each  column:")
print(df.isnull().sum())
numerical_features=df.select_dtypes(include=[np.number]).columns
print(numerical_features)
# for col in numerical_features:
#     plt.figure(figsize=(10,6))
#     df[col].plot(kind='hist',title=col,bins=60,edgecolor='black')
#     plt.ylabel('Frequency')
#     plt.show()
plt.figure(figure=(12,8))
df.hist(figsize=(12,8),bins=30,edgecolor='black')
plt.suptitle("Histograms",fontsize=16)
plt.ylabel('Frequency')
plt.show()

for col in numerical_features:
    plt.figure(figsize=(6,6))
    sns.boxplot(df[col],color='blue')
    plt.title(col)
    plt.ylabel(col)
    plt.show()

print("Outliers detection:")
outliers_summary={}
for feature in numerical_features:
    Q1=df[feature].quantile(0.25)
    Q3=df[feature].quantile(0.75)
    IQR=Q3-Q1
    lower_bound=Q1-1.5*IQR
    upper_bound=Q3+1.5*IQR
    outliers=df[(df[feature]<lower_bound)|(df[feature]>upper_bound)]
    outliers_summary[feature]=len(outliers)
    print(f" {feature}:{len(outliers)} outliers")
```

PROGRAM-02

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
housing_data = fetch_california_housing(as_frame=True)
df = housing_data.frame
print(df.head())
print("\nBasic Information about Dataset:")
print(df.info())
print("\nSummary Statistics:")
print(df.describe())
print("\nMissing Values in Each Column:")
variable_meaning = {
"MedInc": "Median income in block group",
"HouseAge": "Median house age in block group",
"AveRooms": "Average number of rooms per household",
"AveBedrms": "Average number of bedrooms per household",
"Population": "Population of block group",
"AveOccup": "Average number of household members",
"Latitude": "Latitude of block group",
"Longitude": "Longitude of block group",
"Target": "Median house value (in $100,000s)"
}
variable_data = pd.DataFrame(list(variable_meaning.items()), columns=["Feature",
"Description"])
print("\nVariable Meaning Table:")
print(variable_data)
correlation_matrix = df.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
sns.pairplot(df, diag_kind='kde', plot_kws={'alpha': 0.7})
plt.show()
print("\nKey Insights:")
print("1. The dataset has", df.shape[0], "rows and", df.shape[1], "columns.")
print("2. No missing values were found in the dataset.")
print("3. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.")
```

PROGRAM- 3

```
import numpy as np
import pandas as pd
data = pd.read_csv('weather.csv')
print("Data set is --- \n")
print(data)
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
print("\nThe most specific hypothesis : ['0', '0', '0', '0', '0', '0']\n")
for i, val in enumerate (target) :
    if val == "yes" :
        specific_h = concepts[i]
        break
for i, h in enumerate (concepts) :
    if target[i] == "yes" :
        for x in range(len(specific_h)) :
            if h[x] == specific_h[x] :
                pass
            else :
                specific_h[x] = "?"
        print(f"Iteration {i} : {specific_h}")
print("\nMaximally specific hypothesis is \n", specific_h)
```

PROGRAM-4

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

iris=datasets.load_iris()
X=iris.data
y=iris.target
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
print("Covariance matrix is:")
cov_matrix=np.cov(X_scaled.T)
print(cov_matrix)
eigenvalues,eigenvectors=np.linalg.eig(cov_matrix)
print("Eigenvalues:",eigenvalues)
print("Eigenvectors:\n",eigenvectors)
fig=plt.figure(figsize=(8,6))
```

```
ax=fig.add_subplot(111,projection='3d')
colors=['red','green','blue']
labels=iris.target_names
for i in range(len(colors)):

ax.scatter(X_scaled[y==i,0],X_scaled[y==i,1],X_scaled[y==i,2],color=colors[i],label=labels[i])
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')
ax.set_title('3D Visualization of Iris Data Before PCA')
plt.legend()
plt.show()
U,S,Vt=np.linalg.svd(X_scaled,full_matrices=False)
print("Singular Values:",S)
pca=PCA(n_components=2)
X_pca=pca.fit_transform(X_scaled)
explained_variance=pca.explained_variance_ratio_
print(f"Explained Variance by PC1:{explained_variance[0]:.2f}")
print(f"Explained Variance by PC2:{explained_variance[1]:.2f}")
plt.figure(figsize=(8,6))
for i in range(len(colors)):
    plt.scatter(X_pca[y==i,0],X_pca[y==i,1],color=colors[i],label=labels[i])
plt.xlabel('Principal Components 1')
plt.ylabel('Principal Components 2')
plt.title('PCA on Iris Dataset (Dimensinality Reduction)')
plt.legend()
plt.grid()
plt.show()
fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(111,projection='3d')
for i in range(len(colors)):
    ax.scatter(X_scaled [y == i, 0], X_scaled [y == i, 1], X_scaled [y == i, 2], color=colors[i],
label=labels[i])
for i in range(3):
    ax.quiver(0, 0, 0, eigenvectors [i, 0], eigenvectors [i, 1], eigenvectors [i, 2], color='black',
length=1)
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')
ax.set_title('3D Data with Eigenvectors')
plt.legend()
plt.show()
```

PROGRAM-5

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
np.random.seed(0)
data=np.random.rand(100)
labels=np.zeros(100)
labels[:50]=np.where(data[:50]<=0.5,1,2)
test_labels=np.where(data[:50]<=0.5,1,2)
train_data=data[:50].reshape(-1,1)
train_labels=labels[:50]
test_data=data[:50].reshape(-1,1)
print("Train Data\n:",train_data.flatten())
print("Train Labels\n:",test_data.flatten())
print("Test Data\n:",test_data.flatten())
K_values=[1,2,3,4,5,20,30]
for K in K_values:
    Knn=KNeighborsClassifier(n_neighbors=K)
    Knn.fit(train_data,train_labels)
    predicted_labels=Knn.predict(test_data)
    accuracy=accuracy_score(test_labels,predicted_labels)*100
    print(f"\nK={K}")
    print("predicted Labels\n:",predicted_labels)
    print(f"Accuracy\n:{accuracy:.2f}%")
```