Data Mining Lab-2

Mohit Kumar 23IT3028

Dependencies Required:

```
import numpy as np
import pandas as pd
from itertools import combinations
from math import sqrt, log
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

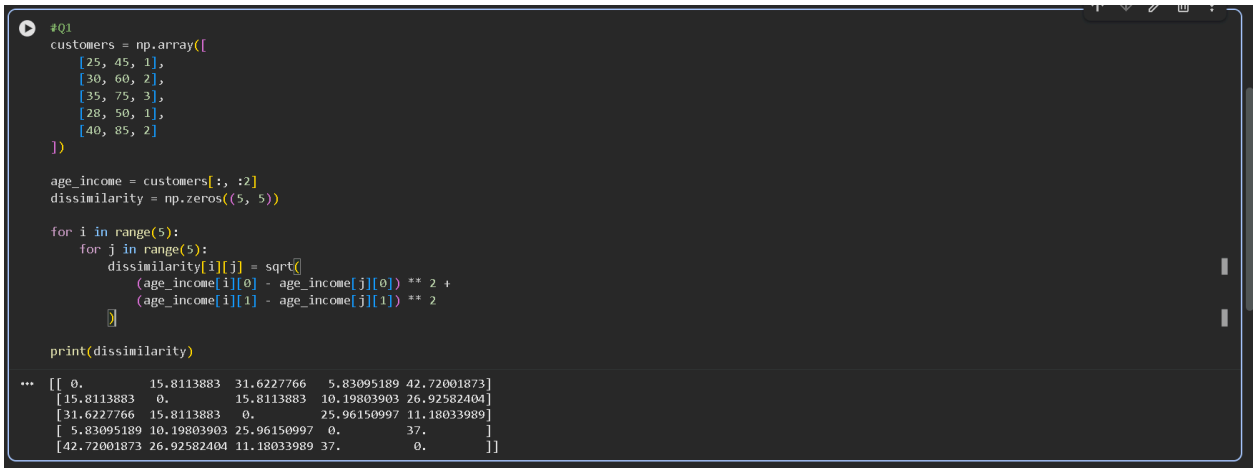
Q1:

Code:

```
customers = np.array([
    [25, 45, 1],
    [30, 60, 2],
    [35, 75, 3],
    [28, 50, 1],
    [40, 85, 2]
])

age_income = customers[:, :2]
dissimilarity = np.zeros((5, 5))

for i in range(5):
    for j in range(5):
        dissimilarity[i][j] = sqrt(
```

```
            (age_income[i][0] - age_income[j][0]) ** 2 +
            (age_income[i][1] - age_income[j][1]) ** 2
        )

print(dissimilarity)
```

Output:



```
#Q1
customers = np.array([
    [25, 45, 1],
    [30, 60, 2],
    [35, 75, 3],
    [28, 50, 1],
    [40, 85, 2]
])

age_income = customers[:, :2]
dissimilarity = np.zeros((5, 5))

for i in range(5):
    for j in range(5):
        dissimilarity[i][j] = sqrt(
            (age_income[i][0] - age_income[j][0]) ** 2 +
            (age_income[i][1] - age_income[j][1]) ** 2
        )

print(dissimilarity)
```

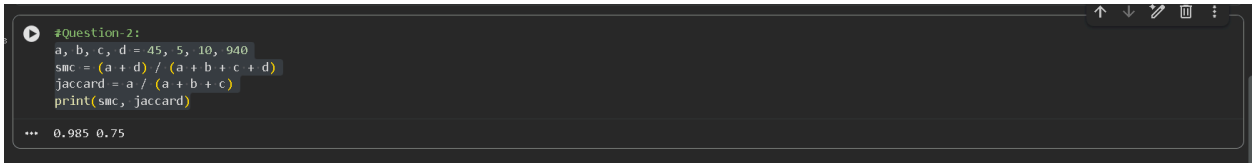
```
[[ 0.         15.8113883  31.6227766   5.83095189 42.72001873]
 [15.8113883   0.         15.8113883  10.19803903 26.92582404]
 [31.6227766  15.8113883   0.         25.96150997 11.18033989]
 [ 5.83095189 10.19803903 25.96150997  0.         37.        ]
 [42.72001873 26.92582404 11.18033989 37.          0.        ]]
```

Q2:

Code:

```
a, b, c, d = 45, 5, 10, 940
smc = (a + d) / (a + b + c + d)
jaccard = a / (a + b + c)
print(smc, jaccard)
```

Output:



```
#Question-2:
a, b, c, d = 45, 5, 10, 940
smc = (a + d) / (a + b + c + d)
jaccard = a / (a + b + c)
print(smc, jaccard)
```

```
0.985 0.75
```

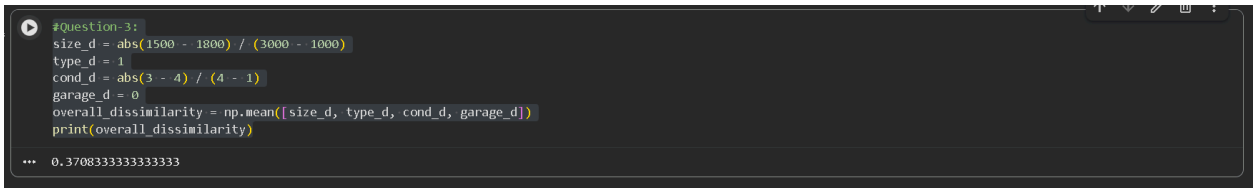
Q3:

Code:

```
#Question-3:
size_d = abs(1500 - 1800) / (3000 - 1000)
type_d = 1
cond_d = abs(3 - 4) / (4 - 1)
```

```
garage_d = 0
overall_dissimilarity = np.mean([size_d, type_d, cond_d, garage_d])
print(overall_dissimilarity)
```

Output:



Q4:

Code:

```
#Question-4
docs = np.array([
    [3, 0, 2, 1, 4],
    [1, 2, 0, 3, 2],
    [2, 1, 1, 2, 3]
])

def cosine(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

for i, j in combinations(range(3), 2):
    print(cosine(docs[i], docs[j]))

A = np.array([0.4, 0.3, 0.2, 0.1, 0.0])
B = np.array([0.3, 0.3, 0.2, 0.1, 0.1])

def kl(p, q):
    return sum(p[i] * log(p[i] / q[i]) for i in range(len(p)) if p[i] > 0)

print(kl(A, B), kl(B, A))
```

Output:

```
#Question-4
docs = np.array([
    [3, 0, 2, 1, 4],
    [1, 2, 0, 3, 2],
    [2, 1, 1, 2, 3]
])

def cosine(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

for i, j in combinations(range(3), 2):
    print(cosine(docs[i], docs[j]))

A = np.array([0.4, 0.3, 0.2, 0.1, 0.0])
B = np.array([0.3, 0.3, 0.2, 0.1, 0.1])

def kl(p, q):
    return sum(p[i] * log(p[i] / q[i]) for i in range(len(p)) if p[i] > 0)

print(kl(A, B), kl(B, A))
```

```
0.6024640760767093
0.9214785982417301
0.8651809126974003
0.11507282898071242 inf
/tmp/ipython-input-3753403888.py:18: RuntimeWarning: divide by zero encountered in scalar divide
  return sum(p[i] * log(p[i] / q[i]) for i in range(len(p)) if p[i] > 0)
```

Q5:

Code:

```
#question-5

data = pd.DataFrame([
    [1, "John Doe", 25, "john@email.com", "2023-02-15", 150.50],
    [2, "Jane Smith", 300, "jane.email@com", "2023/13/45", 200.00],
    [3, "Bob", -5, "bob@company.org", "2023-05-30", "one hundred"],
    [4, "Alice Johnson", 35, "alice@", "2023-07-12", 75.25],
    [5, None, 28, "charlie@test.com", "2023-08-22", 300.00]
], columns=["ID", "Name", "Age", "Email", "Purchase_Date", "Amount"])

data["Name"] = data["Name"].fillna("Unknown")
data["Age"] = data["Age"].clip(18, 100)
data["Purchase_Date"] = pd.to_datetime(data["Purchase_Date"],
errors="coerce").dt.strftime("%Y-%m-%d")
data["Amount"] = pd.to_numeric(data["Amount"], errors="coerce")
print(data)
```

Output:

```
#question-5

data = pd.DataFrame([
    [1, "John Doe", 25, "john@email.com", "2023-02-15", 150.50],
    [2, "Jane Smith", 300, "jane.email@com", "2023/13/45", 200.00],
    [3, "Bob", -5, "bob@company.org", "2023-05-30", "one hundred"],
    [4, "Alice Johnson", 35, "alice@", "2023-07-12", 75.25],
    [5, None, 28, "charlie@test.com", "2023-08-22", 300.00]
], columns=["ID", "Name", "Age", "Email", "Purchase_Date", "Amount"])

data["Name"] = data["Name"].fillna("Unknown")
data["Age"] = data["Age"].clip(18, 100)
data["Purchase_Date"] = pd.to_datetime(data["Purchase_Date"], errors="coerce").dt.strftime("%Y-%m-%d")
data["Amount"] = pd.to_numeric(data["Amount"], errors="coerce")
print(data)
```

```
   ID           Name  Age            Email Purchase_Date  Amount
0   1       John Doe   25    john@email.com    2023-02-15  150.50
1   2     Jane Smith  100    jane.email@com           NaN  200.00
2   3            Bob   18   bob@company.org    2023-05-30     NaN
3   4  Alice Johnson   35           alice@    2023-07-12   75.25
4   5        Unknown   28  charlie@test.com    2023-08-22  300.00
```

Q6:

Code:

```
#Question-6:
salaries = np.array([45000, 52000, 48000, 75000, 82000, 68000, 92000,
55000, 62000, 150000])
minmax = (salaries - salaries.min()) / (salaries.max() - salaries.min())
zscore = (salaries - salaries.mean()) / salaries.std()
decimal = salaries / (10 ** len(str(salaries.max())))

ages = np.array([25, 32, 28, 45, 38, 41, 50, 29, 35, 42])
equal_width = np.digitize(ages, np.linspace(ages.min(), ages.max(), 4))
equal_freq = pd.qcut(ages, 3, labels=False)

print(minmax, zscore, decimal)
print(equal_width, equal_freq)
```

Output:

```
#Question-6:
salaries = np.array([45000, 52000, 48000, 75000, 82000, 68000, 92000, 55000, 62000, 150000])
minmax = (salaries - salaries.min()) / (salaries.max() - salaries.min())
zscore = (salaries - salaries.mean()) / salaries.std()
decimal = salaries / (10 ** len(str(salaries.max())))

ages = np.array([25, 32, 28, 45, 38, 41, 50, 29, 35, 42])
equal_width = np.digitize(ages, np.linspace(ages.min(), ages.max(), 4))
equal_freq = pd.qcut(ages, 3, labels=False)

print(minmax, zscore, decimal)
print(equal_width, equal_freq)
```

```
[0.         0.06666667 0.02857143 0.28571429 0.35238095 0.21904762
 0.44761905 0.0952381  0.16190476 1.        ] [-0.94617596 -0.70878414 -0.84443661  0.07121755  0.30860936 -0.16617427
  0.64774053 -0.60704479 -0.36965297  2.6147013 ] [0.045 0.052 0.048 0.075 0.082 0.068 0.092 0.055 0.062 0.15 ]
[1 1 1 3 2 2 4 1 2 3] [0 1 0 2 1 2 2 0 1 2]
```

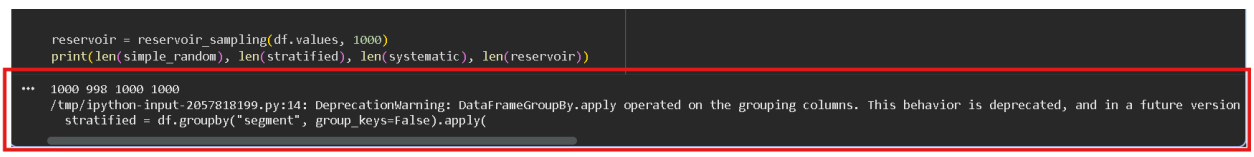
Q7:

Code:

```
#Question-7:
```

```
np.random.seed(42)
n_samples = 10000
df = pd.DataFrame({
    "customer_id": range(n_samples),
    "age": np.random.randint(18, 70, n_samples),
    "income": np.random.normal(50000, 15000, n_samples),
    "purchases": np.random.poisson(5, n_samples),
    "segment": np.random.choice(["A", "B", "C", "D"], n_samples, p=[0.1,
0.3, 0.4, 0.2])
})

simple_random = df.sample(1000)

stratified = df.groupby("segment", group_keys=False).apply(
    lambda x: x.sample(int(len(x) / n_samples * 1000))
)

systematic = df.iloc[::10]

def reservoir_sampling(stream, k):
    reservoir = []
    for i, item in enumerate(stream):
        if i < k:
            reservoir.append(item)
        else:
            j = np.random.randint(0, i + 1)
            if j < k:
                reservoir[j] = item
    return reservoir

reservoir = reservoir_sampling(df.values, 1000)
print(len(simple_random), len(stratified), len(systematic),
len(reservoir))
```

Output:



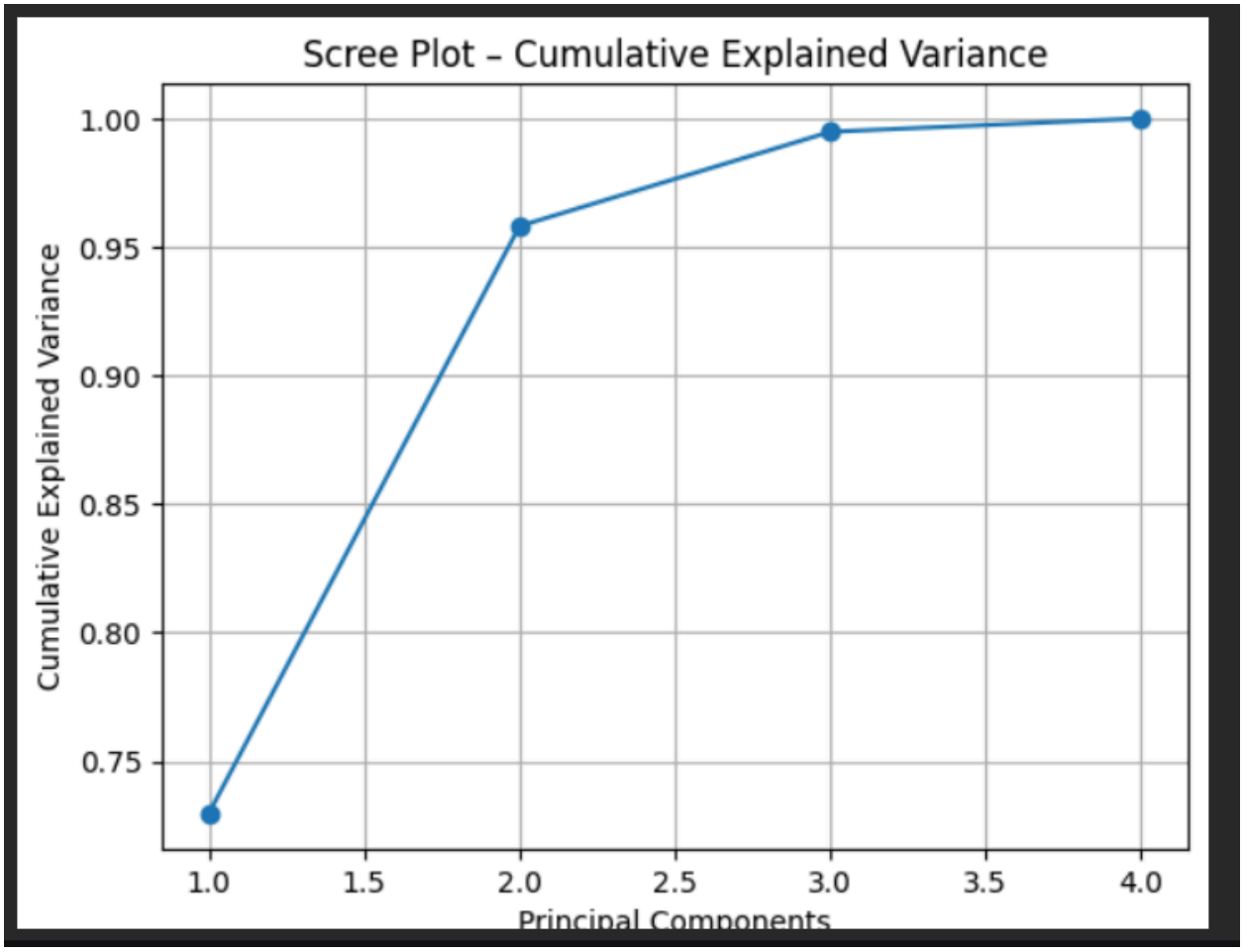
Q8:

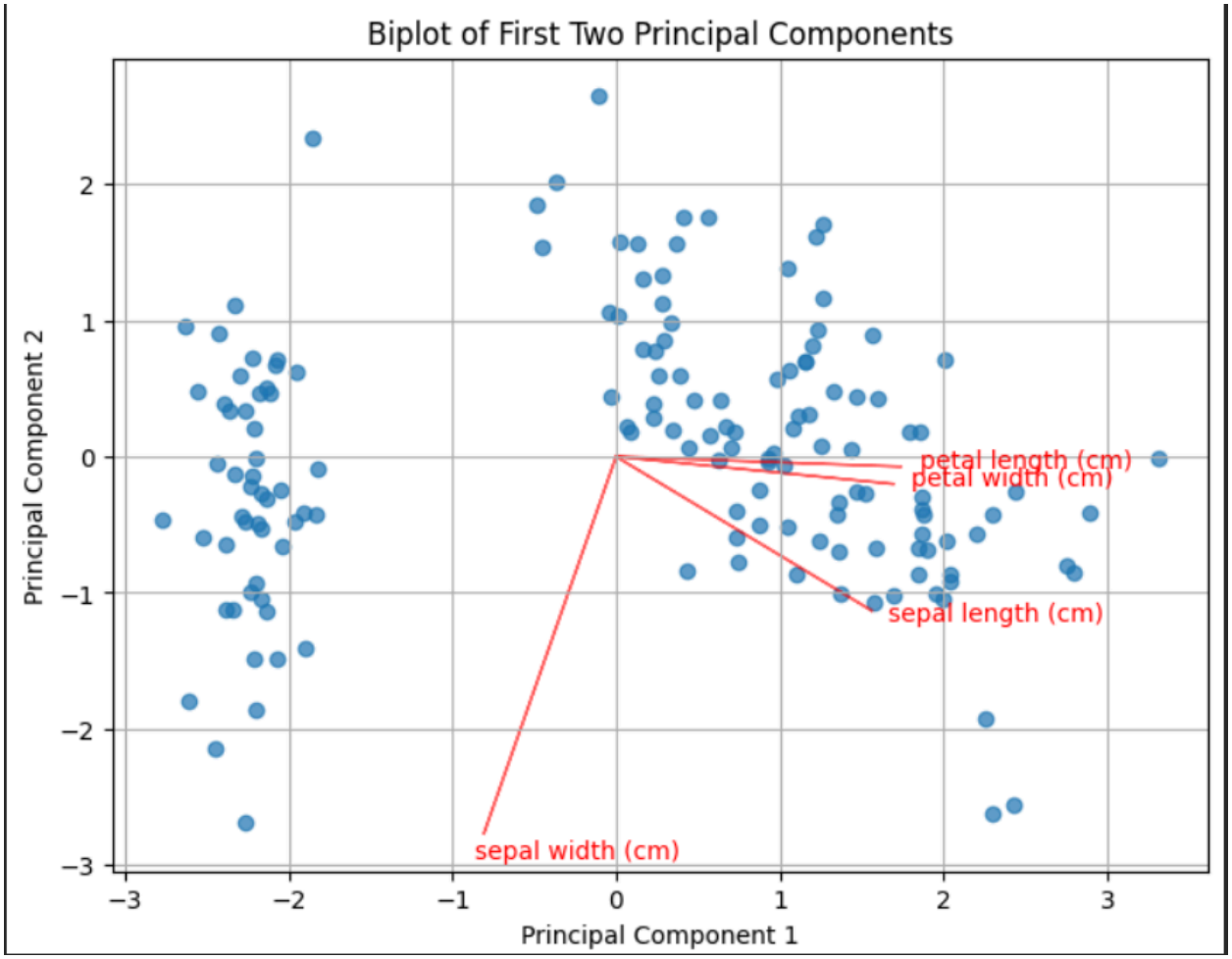
Code:

```
#Question-8:


iris = load_iris()
X = iris.data
feature_names = iris.feature_names

X_std = StandardScaler().fit_transform(X)

cov_matrix = np.cov(X_std.T)
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

explained_variance = eigenvalues / np.sum(eigenvalues)

plt.plot(
    range(1, len(explained_variance) + 1),
    np.cumsum(explained_variance),
    marker='o'
)
plt.xlabel("Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Scree Plot - Cumulative Explained Variance")
plt.grid(True)
plt.show()

pca = PCA()
X_pca_sklearn = pca.fit_transform(X_std)

X_pca_manual = X_std @ eigenvectors[:, :2]

plt.figure(figsize=(8, 6))
plt.scatter(X_pca_manual[:, 0], X_pca_manual[:, 1], alpha=0.7)

for i in range(len(feature_names)):
```

```
    plt.arrow(
        0, 0,
        eigenvectors[i, 0] * 3,
        eigenvectors[i, 1] * 3,
        color='red',
        alpha=0.6
    )
    plt.text(
        eigenvectors[i, 0] * 3.2,
        eigenvectors[i, 1] * 3.2,
        feature_names[i],
        color='red'
    )

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("Biplot of First Two Principal Components")
plt.grid(True)
plt.show()
```

Output:


Scree Plot – Cumulative Explained Variance
Cumulative Explained Variance
1.00
0.95
0.90
0.85
0.80
0.75
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Principal Components



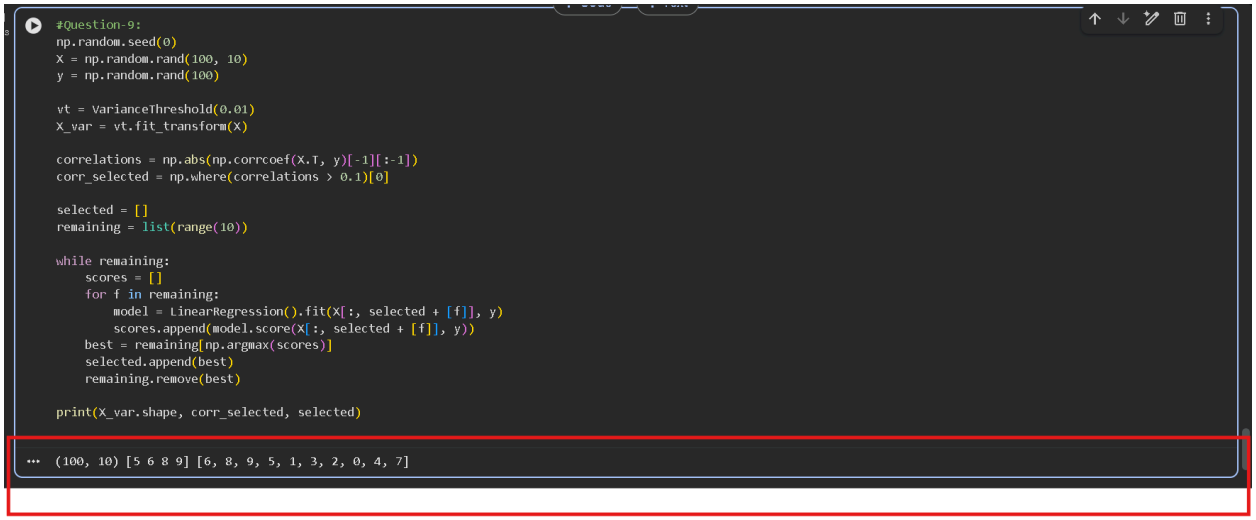
Question-9:

Code:

```
#Question-9:
np.random.seed(0)
X = np.random.rand(100, 10)
y = np.random.rand(100)

vt = VarianceThreshold(0.01)
X_var = vt.fit_transform(X)

correlations = np.abs(np.corrcoef(X.T, y)[-1][:-1])
corr_selected = np.where(correlations > 0.1)[0]

selected = []
remaining = list(range(10))
```

```
while remaining:
    scores = []
    for f in remaining:
        model = LinearRegression().fit(X[:, selected + [f]], y)
        scores.append(model.score(X[:, selected + [f]], y))
    best = remaining[np.argmax(scores)]
    selected.append(best)
    remaining.remove(best)

print(X_var.shape, corr_selected, selected)
```

Output:



Google Collab Link: MohitKumar_23IT3028_Datamining_lab-2