1.

## 1. Load the Dataset and Summarize Unique Values

```
# Import necessary libraries
from pandas import read_csv
from numpy import unique, loadtxt

# Define the location of the dataset
path = './oil-spill.csv'

# Load the dataset using pandas
df = read_csv(path, header=None)

# Summarize the number of unique values in each column using pandas
print("Unique values in each column using pandas:")
print(df.nunique())

# Load the dataset using numpy
data = loadtxt(path, delimiter=',')

# Summarize the number of unique values in each column using numpy
print("Unique values in each column using numpy:")
for i in range(data.shape[1]):
    num = len(unique(data[:, i]))
    percentage = float(num) / data.shape[0] * 100
    print('%d, %d, %.1f%%' % (i, num, percentage))
```

## 2. Summarize and Delete Columns with Low Variance

```
# Summarize the percentage of unique values for each column using numpy
print("Columns with less than 1% unique values:")
for i in range(data.shape[1]):
    num = len(unique(data[:, i]))
    percentage = float(num) / data.shape[0] * 100
    if percentage < 1:
        print('%d, %d, %.1f%%' % (i, num, percentage))

# Delete columns where the number of unique values is less than 1% of the rows
using pandas
```

```
print("Shape of dataset before dropping columns:")
print(df.shape)

# Get the number of unique values for each column
counts = df.nunique()

# Record columns to delete
to_del = [i for i, v in enumerate(counts) if (float(v) / df.shape[0] * 100) <
1]

# Drop columns
df.drop(to_del, axis=1, inplace=True)

print("Shape of dataset after dropping columns:")
print(df.shape)
```

## 3. Feature Selection Using Variance Threshold

```
# Import additional necessary libraries
from numpy import arange
from sklearn.feature_selection import VarianceThreshold
from matplotlib import pyplot

# Split data into inputs and outputs
data = df.values
X = data[:, :-1]
y = data[:, -1]
print("Shape of input and output data:")
print(X.shape, y.shape)

# Define thresholds to check
thresholds = arange(0.0, 0.55, 0.05)

# Apply transform with each threshold and store the results
results = []
for t in thresholds:
    transform = VarianceThreshold(threshold=t)
    X_sel = transform.fit_transform(X)
    n_features = X_sel.shape[1]
    print('>Threshold=%.2f, Features=%d' % (t, n_features))
    results.append(n_features)

# Plot the threshold vs the number of selected features
pyplot.plot(thresholds, results)
pyplot.xlabel('Threshold')
pyplot.ylabel('Number of Features Selected')
pyplot.show()
```

## 4. Check and Remove Duplicates

```
# Check for duplicates in the dataset
dups = df.duplicated()

# Report if there are any duplicates
if dups.any():
    print("Duplicate rows found:")
    print(df[dups])

# Delete rows of duplicate data
print("Shape of dataset before removing duplicates:")
print(df.shape)

df.drop_duplicates(inplace=True)

print("Shape of dataset after removing duplicates:")
print(df.shape)
```

2.2.Use Linear Regression to predict house prices based on features like area, number of bedrooms, and storeys.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pandas as pd

# Load dataset
data = pd.read_csv('Housing.csv'
X = data[['area', 'bedrooms', 'bathrooms','storeys']]
y = data['price']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Create linear regression modelmodel = LinearRegression()
```

```
model.fit(X_train, y_train)
# Predict and evaluate
predictions = model.predict(X_test)
print(predictions)
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

3.Assuming a set of documents that need to be classified, use the naive Bayesian Classifier model to perform this task. Calculate the accuracy, precision, and recall for your data set.

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
classification_report
# Load the 20 Newsgroups dataset
newsgroups = fetch_20newsgroups(subset='all')
X, y = newsgroups.data, newsgroups.target

# Transform the text data into TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english')
X_tfidf = vectorizer.fit_transform(X)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2,
random_state=42)
# Create and train the Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train, y_train)

# Predict and evaluate

predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='macro')
recall = recall_score(y_test, predictions, average='macro')
classification_rep = classification_report(y_test, predictions,
target_names=newsgroups.target_names)
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print('\nClassification Report:\n', classification_rep)
```

4.Demonstrate the working of the Random forest algorithm. Use an appropriate data set for building and apply this knowledge to classify a new sample.

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

digits = load_digits()
X, y = digits.data, digits.target
# Split the data into training and testing sets

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

# Create and train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f'Test accuracy: {accuracy}')

# Classify a new sample
sample_index = 8  # Change this to classify a different sample
new_sample = X_test[sample_index].reshape(1, -1)  # Reshape to match the input
format of the model
predicted_label = model.predict(new_sample)
actual_label = y_test[sample_index]

print(f'Predicted label: {predicted_label[0]}, Actual label: {actual_label}')

# Visualize the sample and its prediction
plt.figure(figsize=(4, 4))
plt.imshow(new_sample.reshape(8, 8), cmap=plt.cm.gray_r,
interpolation='nearest')
plt.title(f'Predicted: {predicted_label[0]}, Actual: {actual_label}')
plt.show()
```

5.Use the Wine dataset for classification with a Support Vector Machine (SVM) model

```python
import numpy as np
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Load the Wine dataset
wine = load_wine()
data, target = wine.data, wine.target
print(wine)

# Split the data into training and testing sets

train_data, test_data, train_target, test_target = train_test_split(data,
target, test_size=0.2, random_state=42)

# Scale the data

scaler = StandardScaler()
train_data = scaler.fit_transform(train_data)
test_data = scaler.transform(test_data)

# Create and train the SVM model

model = svm.SVC(kernel='linear')
model.fit(train_data, train_target)


# Predict and evaluate

predictions = model.predict(test_data)
accuracy = accuracy_score(test_target, predictions)
print(f'Test accuracy: {accuracy}')

# Classify a new sample

sample_index = 0  # Change this to classify a different sample
new_sample = test_data[sample_index].reshape(1, -1)  # Reshape to match the
input format of the model
predicted_label = model.predict(new_sample)
actual_label = test_target[sample_index]
print(f'Predicted label: {predicted_label[0]}, Actual label: {actual_label}')
```

```
# Visualize the sample

# For the Wine dataset, we will plot the sample along with the classification
results

feature_names = wine.feature_names
target_names = wine.target_names

# Plot the sample's feature values

plt.figure(figsize=(10, 6))
plt.barh(feature_names, test_data[sample_index])
plt.title(f'Predicted: {target_names[predicted_label[0]]}, Actual:
{target_names[actual_label]}')
plt.xlabel('Scaled Feature Values')
plt.show()
```