ME-793 Assignment 3

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [58]:
```
data = pd.read_csv('Periodic-table-final.csv')
data.head()
```

Out[58]:

	Symbol	Atomic Number	Electronegativity	Atomic Radii (pm)	Thermal Conductivity	Density	Crystal System
0	H	1	2.20	53.0	0.1805	0.000090	Simple Hexagonal
1	He	2	NaN	31.0	0.1513	0.000179	Face Centered Cubic
2	Li	3	0.98	167.0	85.0000	0.534000	Body Centered Cubic
3	Be	4	1.57	112.0	190.0000	1.850000	Simple Hexagonal
4	B	5	2.04	87.0	27.0000	2.340000	Simple Trigonal

In [27]:
```
print("Total number of rows:", data.shape[0])
print("Total number of columns:", data.shape[1])
```

```
Total number of rows: 118
Total number of columns: 7
```

In [28]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118 entries, 0 to 117
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Symbol                  118 non-null    object
 1   Atomic Number           118 non-null    int64
 2   Electronegativity       96 non-null     float64
 3   Atomic Radii (pm)       86 non-null     float64
 4   Thermal Conductivity    94 non-null     float64
 5   Density                 105 non-null    float64
 6   Crystal System          113 non-null    object
dtypes: float64(4), int64(1), object(2)
memory usage: 6.6+ KB
```

In [59]:
```
data = data.dropna(subset=['Crystal System'])
print("After dropping the rows for which crystal system data is not available, total number of rows:", data.
```

```
After dropping the rows for which crystal system data is not available, total number of rows: 113
```

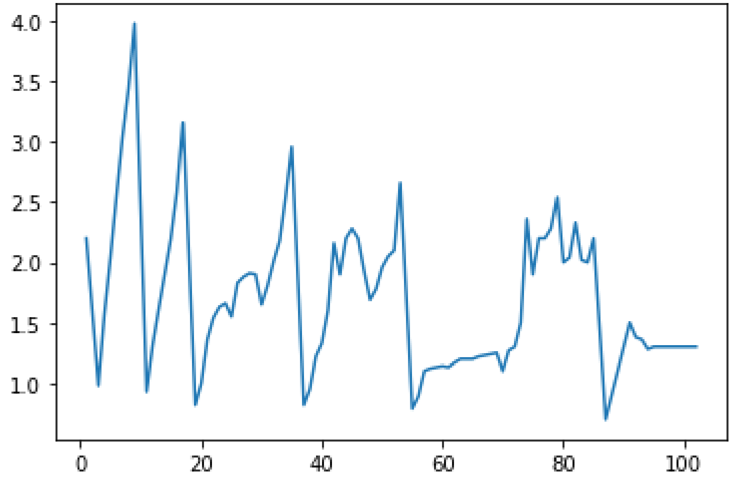
In [60]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 113 entries, 0 to 117
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Symbol                  113 non-null    object
 1   Atomic Number           113 non-null    int64
 2   Electronegativity       96 non-null     float64
 3   Atomic Radii (pm)       86 non-null     float64
 4   Thermal Conductivity    94 non-null     float64
 5   Density                 105 non-null    float64
 6   Crystal System          113 non-null    object
dtypes: float64(4), int64(1), object(2)
memory usage: 7.1+ KB
```

Plot these values on the Y-axis vs. elements on the X-axis. You can do a separate plot for each of these features.

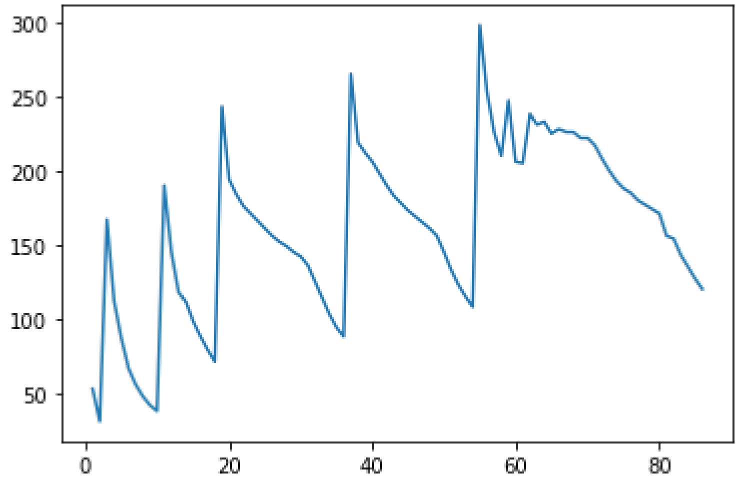
In [62]:
```
# Plotting eletronegativity v/s Elements
data_el = data[data["Electronegativity"].notnull()]
plt.plot(data_el['Atomic Number'], data_el['Electronegativity'])
```

Out[62]:
```
[<matplotlib.lines.Line2D at 0x7f5842f27190>]
```



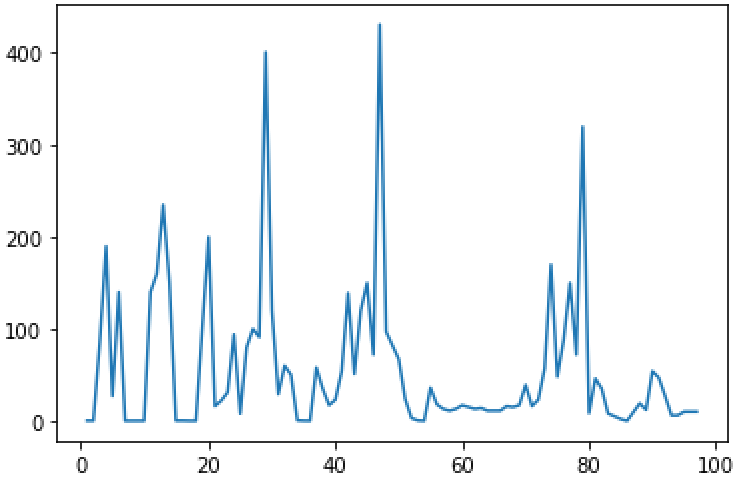
In [32]:
```
# Plotting radii v/s Elements
data_rd = data[data["Atomic Radii (pm)"].notnull()]
plt.plot(data_rd['Atomic Number'], data_rd['Atomic Radii (pm)'])
```

Out[32]: [<matplotlib.lines.Line2D at 0x7f584393b7f0>]



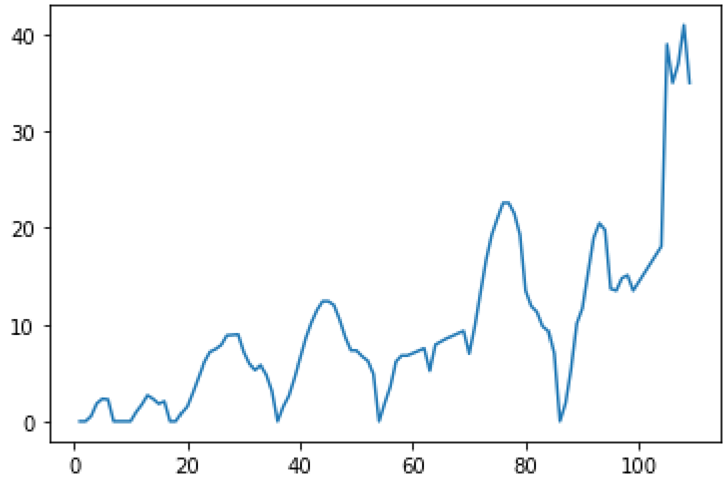
In [33]:
```
# Plotting thermal conductivity v/s Elements
data_tc = data[data['Thermal Conductivity'].notnull()]
plt.plot(data_tc['Atomic Number'], data_tc['Thermal Conductivity'])
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x7f58438ac1f0>]
```



In [34]:
```
# Plotting Density v/s Elements
data_ds = data[data['Density'].notnull()]
plt.plot(data_ds['Atomic Number'], data_ds['Density'])
```

Out[34]:
```
[<matplotlib.lines.Line2D at 0x7f5843890400>]
```



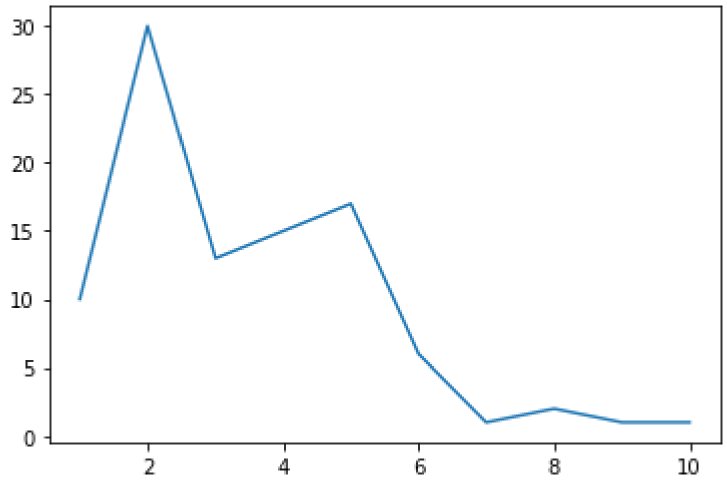
Arrange in increasing order, divide the span of the values of each of these into 10 equal size bins, count the number of elements in each bin and plot number of elements on the Y-axis vs. bins on the X-axis.

In [35]:
```python
def get_bin_counts(data, col):
    data = data.sort_values(by=[col])
    col_range = data[col].max() - data[col].min()
    bin_size = col_range/10
    bins = []
    for i in list(np.arange(data[col].min(), data[col].max(), bin_size)):
        if i+bin_size > data[col].max():
            bins.append([i, data[col].max()])
            break
        else:
            bins.append([i, i+bin_size])
    counts = [0]*10
    for i, bin_ in enumerate(bins):
        idx = 0
        for j, val in enumerate(data[col][idx:]):
            if val >= bin_[0] and val < bin_[1]:
                counts[i] += 1
            if val > bin_[1]:
                idx = j
                break
    counts[i] += 1 # to add the element count having the maximum value
    return bins, counts
```

In [36]:
```
# Bins for electronegativity values
en_bins, en_counts = get_bin_counts(data, 'Electronegativity')
print(en_bins)
plt.plot(np.arange(1,11,1), en_counts)
```

```
[[0.7, 1.028], [1.028, 1.356], [1.356, 1.6840000000000002], [1.6840000000000002, 2.012], [2.0120000000000005, 2.3400000000000003], [2.3400000000000003, 2.668], [2.668, 2.996], [2.9960000000000004, 3.3240000000000003], [3.3240000000000007, 3.6520000000000006], [3.652000000000001, 3.98]]
```

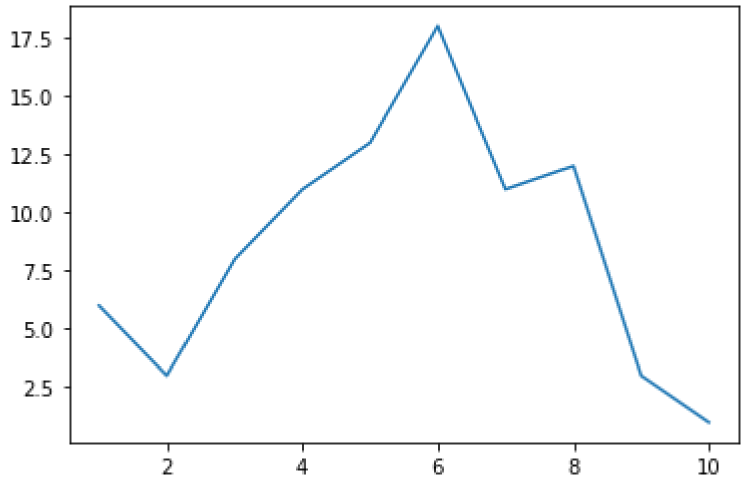
Out[36]:
```
[<matplotlib.lines.Line2D at 0x7f5843863dc0>]
```



In [37]:
```
# Bins for Atomic Radii Values
ar_bins, ar_counts = get_bin_counts(data, 'Atomic Radii (pm)')
print(ar_bins)
plt.plot(np.arange(1,11,1), ar_counts)
```

```
[[31.0, 57.7], [57.7, 84.4], [84.4, 111.10000000000001], [111.10000000000001, 137.8], [137.8, 164.5], [164.
5, 191.2], [191.20000000000002, 217.9], [217.90000000000003, 244.60000000000002], [244.60000000000002, 271.
3], [271.3, 298.0]]
```

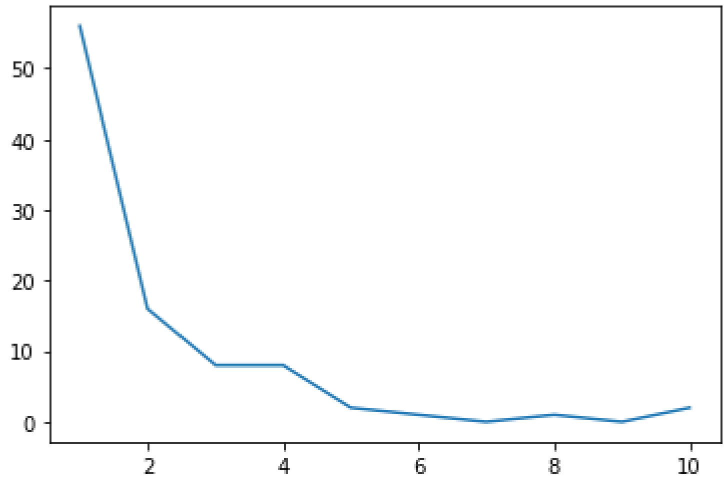
Out[37]:
```
[<matplotlib.lines.Line2D at 0x7f58437ce580>]
```



In [38]:
```
# Bins for Thermal Conductivity values
tc_bins, tc_counts = get_bin_counts(data, 'Thermal Conductivity')
print(tc_bins)
plt.plot(np.arange(1,11,1), tc_counts)
```

```
[[0.00361, 43.003249000000004], [43.003249000000004, 86.00288800000001], [86.002888, 129.002527], [129.0025
2700000001, 172.00216600000002], [172.00216600000002, 215.00180500000002], [215.00180500000002, 258.00144
4], [258.001444, 301.001083], [301.001083, 344.000722], [344.000722, 387.000361], [387.000361, 430.0]]
```

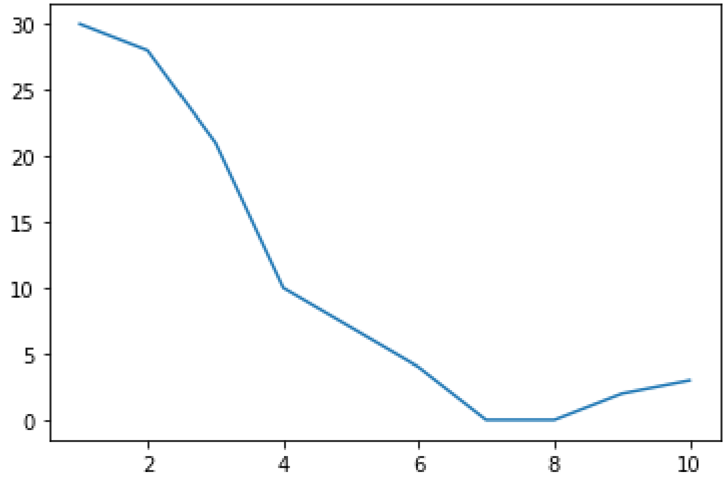
Out[38]:
```
[<matplotlib.lines.Line2D at 0x7f5843734760>]
```



In [39]:
```
# Bins for Density values
d_bins, d_counts = get_bin_counts(data, 'Density')
print(d_bins)
plt.plot(np.arange(1,11,1), d_counts)
```

```
[[8.99e-05, 4.10008091], [4.10008091, 8.20007192], [8.200071920000001, 12.300062930000001], [12.30006293000
0001, 16.40005394], [16.40005394, 20.50004495], [20.50004495, 24.60003596], [24.60003596, 28.70002697], [2
8.70002697, 32.80001798], [32.80001798, 36.90000899], [36.900008989999996, 41.0]]
```

Out[39]: [<matplotlib.lines.Line2D at 0x7f5843718610>]



Make bins of crystal systems, count the number of elements falling in each crystal system and plot the number of elements in each bin on Y-axis vs. bin on the X-axis. Analyze and describe your observations in the context of probability distributions.

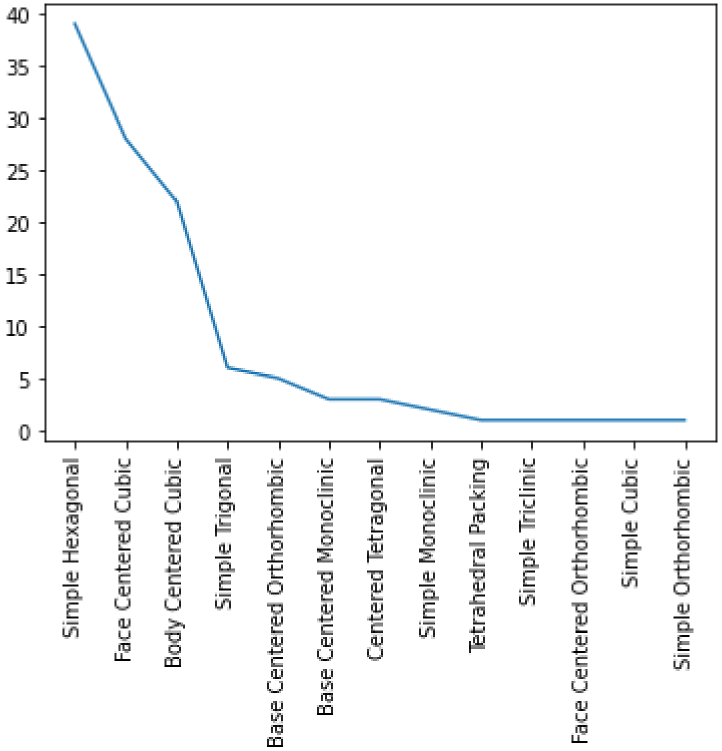
In [50]:
```
cs_counts = data['Crystal System'].value_counts()
cs_df = pd.DataFrame(cs_counts)
cs_df
```

Out[50]:

	Crystal System
Simple Hexagonal	39
Face Centered Cubic	28
Body Centered Cubic	22
Simple Trigonal	6
Base Centered Orthorhombic	5
Base Centered Monoclinic	3
Centered Tetragonal	3
Simple Monoclinic	2
Tetrahedral Packing	1
Simple Triclinic	1
Face Centered Orthorhombic	1
Simple Cubic	1
Simple Orthorhombic	1

In [57]:
```
plt.plot(cs_df.index, cs_df['Crystal System'])
plt.xticks(cs_df.index,rotation=90)
```

Out[57]:
```
([<matplotlib.axis.XTick at 0x7f5843003ac0>,
  <matplotlib.axis.XTick at 0x7f5843003a90>,
  <matplotlib.axis.XTick at 0x7f5843003670>,
  <matplotlib.axis.XTick at 0x7f5842fb9580>,
  <matplotlib.axis.XTick at 0x7f5842fb9cd0>,
  <matplotlib.axis.XTick at 0x7f5842fbf460>,
  <matplotlib.axis.XTick at 0x7f5842fbfbb0>,
  <matplotlib.axis.XTick at 0x7f5842fbfaf0>,
  <matplotlib.axis.XTick at 0x7f5842fb9970>,
  <matplotlib.axis.XTick at 0x7f5842fc5610>,
  <matplotlib.axis.XTick at 0x7f5842fc5d60>,
  <matplotlib.axis.XTick at 0x7f5842fcc4f0>,
  <matplotlib.axis.XTick at 0x7f5842fccc40>],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



If you were to apply PCA on your master data set collected in Q. 1, what could you find? Any interesting trends / observation?

In [15]:
```
data = data.drop(['Crystal System', 'Symbol'], axis=1)
data = data.dropna()
```

In [16]:
```
data_arr = data.to_numpy()
data_arr.shape
```

Out[16]:
```
(79, 5)
```

In [17]:
```
def standardize(X):
    mu = sum(X)/len(X)
    var = sum((X-mu)**2)/len(X)
    z = (X-mu)/var**0.5
    return z
```

In [11]:
```
def pca(data):
    data_norm = standardize(data)
    A = np.dot(data_norm.transpose(), data_norm) / (data_norm.shape[0]-1)
    w,v = np.linalg.eig(A)
    proj_data = np.dot(data_norm, v)
    return v.transpose(), proj_data, w
```

In [18]:
```
P, proj_data, eigvals = pca(data_arr)
```

In [19]:
```
print(P.shape)
print("Principal vector matrix: ", P)
```

```
(5, 5)
Principal vector matrix:  [[ 0.54465094 -0.42367771  0.59022067 -0.00488899  0.41888926]
 [-0.32947094 -0.5891888   0.3122437  -0.27184001 -0.61066548]
 [ 0.29105597  0.23253941 -0.09552645 -0.92288116 -0.01941424]
 [-0.6723125  -0.17709473  0.01921448 -0.27262865  0.66478401]
 [-0.24101155  0.62282954  0.73800334  0.00656471 -0.09646122]]
```

In [20]:
```
print(len(eigvals))
print("Eigenvalues: ", eigvals)
```

```
5
Eigenvalues:  [2.33200843 1.42784973 1.05140657 0.1581375  0.09470033]
```

In [23]:
```
print(proj_data.shape)
print("Projected Data: ", proj_data)
```

```
(79, 5)
Projected Data:  [[-3.05929356e+00  5.78519687e-01  5.92470391e-01  3.45906811e-01
  -6.05474478e-01]
 [-9.20609521e-01  2.00404508e+00 -9.59081413e-01  4.60992179e-01
  -2.60177458e-01]
 [-1.79558768e+00  6.40618670e-01 -1.78728618e+00  7.23146390e-02
  -4.61752187e-01]
 [-2.31116385e+00  5.17882230e-01  2.20659989e-01  4.91705838e-01
  -3.76281389e-01]
 [-2.85794990e+00 -4.38479128e-01 -7.81153360e-01 -5.84269515e-02
  -1.51932134e-01]
 [-3.44858066e+00 -2.61473630e-01  9.67455843e-01 -5.29121097e-02
   1.95508931e-01]
 [-3.77908807e+00 -6.90838701e-01  1.13947174e+00 -1.94662747e-01
   4.66561284e-01]
 [-4.18067798e+00 -1.23792486e+00  1.35896765e+00 -3.74550384e-01
   9.01630050e-01]
 [-4.24396057e-01  1.84750401e+00 -1.52428943e+00  1.35950042e-01
  -7.79347897e-02]
```

In [21]:
```
print(np.cov(proj_data.T))
```

```
[[ 2.33200843e+00  6.97266482e-16 -1.25635963e-18  5.91676652e-16
   2.36691447e-16]
 [ 6.97266482e-16  1.42784973e+00 -1.32839449e-15  1.78151305e-16
   1.37251834e-16]
 [-1.25635963e-18 -1.32839449e-15  1.05140657e+00  4.18971970e-16
  -1.37035688e-16]
 [ 5.91676652e-16  1.78151305e-16  4.18971970e-16  1.58137504e-01
  -5.96044454e-17]
 [ 2.36691447e-16  1.37251834e-16 -1.37035688e-16 -5.96044454e-17
   9.47003287e-02]]
```

In [22]:
```
for i, val in enumerate(eigvals):
    print("% Variance explained by PC {}: ".format(i+1), val/sum(eigvals))
```

```
% Variance explained by PC 1:  0.46049786758985654
% Variance explained by PC 2:  0.2819551356845397
% Variance explained by PC 3:  0.20761952605870237
% Variance explained by PC 4:  0.031227152594959708
% Variance explained by PC 5:  0.018700318071941918
```

It can be seen that 94% of variance contained in the data can be explained by the first 3 principal components only. So, the data projected by the remaining 2 PCs can be dropped for further analyses.