In [75]:

```
print('Logistic Regression Mean Absolute Error:', lreg_mae)
print('Logistic Regression Mean squared Error:', lreg_mse)
print('Logistic Regression  Squared Error:', lreg_rse)
print('Logistic Regression R2 Score:',lreg_r2)
```

```
Logistic Regression Mean Absolute Error: 0.3925
Logistic Regression Mean squared Error: 0.4275
Logistic Regression  Squared Error: 0.653834841531101
Logistic Regression R2 Score: 0.6603351955307262
```

Creating & Training KNN Model

In [63]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X_train,y_train)
```

Out[63]:

```
KNeighborsClassifier(n_neighbors=15)
```

In [64]:

```
knn.score(X_test,y_test)
```

Out[64]:

```
0.935
```

Elbow Method For optimum value of K

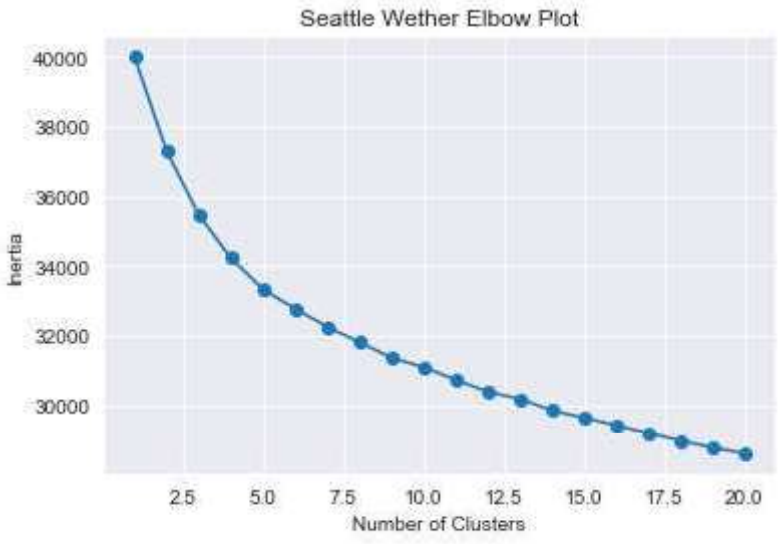
In [65]:

```
# Scale Data
ss2=StandardScaler()
sc2=ss2.fit_transform(X)
```

In [66]:

```
# Create Elbow Plot
from sklearn.cluster import KMeans
inertia_ls = [] # save inertia values
for k in range(1, 21): # test different n_clusters between 1-10
    km = KMeans(n_clusters=k)
    km.fit(sc2) # apply KMeans
    inertia_ls.append(km.inertia_) # Append inertia to list

# Plot Elbow
plt.plot([i for i in range(1, 21)], inertia_ls, marker="o")
plt.title("Seattle Wether Elbow Plot")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia");
```



In [67]:

```
# Apply KMeans and Plot KMeans Results and Actual Results
km=KMeans(n_clusters=4)
preds=km.fit_predict(sc2) # make labe and prediction in single step
preds
```

Out[67]:

```
array([0, 1, 1, ..., 3, 3, 2])
```

submition

In [68]:

```
# Submit 'price_range into 'test.csv''
dft['price_range']=predt
```

In [69]:

```
1 dft
```

Out[69]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobi
0	1	1043	1	1.8	1	14	0	5	0.1	
1	2	841	1	0.5	1	4	1	61	0.8	
2	3	1807	1	2.8	0	1	0	27	0.9	
3	4	1546	0	0.5	1	18	1	25	0.5	
4	5	1434	0	1.4	0	11	1	49	0.5	
...	...	...	...	...	...	...	...	...	...	
995	996	1700	1	1.9	0	0	1	54	0.5	
996	997	609	0	1.8	1	0	0	13	0.9	
997	998	1185	0	1.4	0	1	1	8	0.5	
998	999	1533	1	0.5	1	0	0	50	0.4	
999	1000	1270	1	0.5	0	4	1	35	0.1	

1000 rows × 22 columns

In [70]:

```
1 X = dft
2 y=dft['price_range']
3 lr=LinearRegression()
4 lr.fit(X, y)
5 preds=lr.predict(X)
```

In [71]:

```
1 preds
```

Out[71]:

```
array([ 2.50883263e+00,  2.87989950e+00,  2.14844694e+00,  3.43977805e+00,
        1.24696405e+00,  2.74099010e+00,  3.57846240e+00,  1.01709872e+00,
        2.61425050e+00,  1.31181916e-03,  3.34678317e+00,  3.21180892e+00,
       -1.18076932e-02,  4.31602512e-01,  1.99770214e+00,  4.36139198e-01,
        1.88343584e+00,  1.16755315e+00,  3.11185049e+00,  1.59825340e+00,
        7.97397033e-01,  2.48614093e+00,  1.32109854e+00,  1.32081164e+00,
        3.19431650e+00,  2.71927083e-02,  2.04695701e+00, -3.44412163e-01,
        2.62805038e+00,  4.06415649e-02,  2.16280246e+00,  2.13950004e-01,
        2.84582156e+00,  1.93636932e-01,  5.48663486e-01,  9.72005643e-01,
        2.77195881e+00,  1.21900904e+00,  2.16387545e+00,  9.53917132e-01,
        1.09759815e+00,  1.73748505e+00, -3.99622402e-01, -3.56458507e-01,
        4.05275273e-04,  8.74976993e-01,  3.75447859e-01,  2.60299414e+00,
        1.05382235e+00,  2.15539201e+00,  1.38975774e+00, -5.71754498e-03,
        2.33163862e+00,  4.07984847e-01,  3.12809132e+00,  6.67434961e-01,
        3.05549475e+00,  1.08203021e+00,  7.10295027e-01,  3.14308419e+00,
        3.18164183e+00,  2.52382677e+00,  2.44608828e-01,  1.44291148e+00,
        6.37488050e-01,  1.10412698e+00,  1.41791822e+00,  2.44023139e+00,
        9.45163761e-01.  1.69363777e+00.  1.06808811e+00.  1.96410683e+00.
```

In [72]:

```
1 len(preds)
```

Out[72]:

```
1000
```

In [73]:

```
1 # Compute Accuracy Score of KMean Labels with True Labels
2 dft["price_rangep"] =preds
```

In [74]:

```
1 dft
```

Out[74]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobi
0	1	1043	1	1.8	1	14	0	5	0.1	
1	2	841	1	0.5	1	4	1	61	0.8	
2	3	1807	1	2.8	0	1	0	27	0.9	
3	4	1546	0	0.5	1	18	1	25	0.5	
4	5	1434	0	1.4	0	11	1	49	0.5	
...	...	...	...	...	...	...	...	...	...	
995	996	1700	1	1.9	0	0	1	54	0.5	
996	997	609	0	1.8	1	0	0	13	0.9	
997	998	1185	0	1.4	0	1	1	8	0.5	
998	999	1533	1	0.5	1	0	0	50	0.4	
999	1000	1270	1	0.5	0	4	1	35	0.1	

1000 rows × 23 columns

In [ ]:

```
1
```