In [1]:
```
import pandas as pd
```

In [2]:
```
mobile=pd.read_excel("Mobile_data.csv.xlsx")
mobile
```

Out[2]:

	battery_power	clock_speed	fc	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	price_range
0	842	2.2	1	7	0.6	188	2	2	20	756	2549	9	7	19	1
1	1021	0.5	0	53	0.7	136	3	6	905	1988	2631	17	3	7	2
2	563	0.5	2	41	0.9	145	5	6	1263	1716	2603	11	2	9	2
3	615	2.5	0	10	0.8	131	6	9	1216	1786	2769	16	8	11	2
4	1821	1.2	13	44	0.6	141	2	14	1208	1212	1411	8	2	15	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	0.5	0	2	0.8	106	6	14	1222	1890	668	13	4	19	0
1996	1965	2.6	0	39	0.2	187	4	3	915	1965	2032	11	10	16	2
1997	1911	0.9	1	36	0.7	108	8	3	868	1632	3057	9	1	5	3
1998	1512	0.9	4	46	0.1	145	5	5	336	670	869	18	10	19	0
1999	510	2.0	5	45	0.9	168	6	16	483	754	3919	19	4	2	3

2000 rows × 15 columns

In [3]:
```
mobile.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   battery_power  2000 non-null   int64  
 1   clock_speed    2000 non-null   float64
 2   fc             2000 non-null   int64  
 3   int_memory     2000 non-null   int64  
 4   m_dep          2000 non-null   float64
 5   mobile_wt      2000 non-null   int64  
 6   n_cores        2000 non-null   int64  
 7   pc             2000 non-null   int64  
 8   px_height      2000 non-null   int64  
 9   px_width       2000 non-null   int64  
 10  ram            2000 non-null   int64  
 11  sc_h           2000 non-null   int64  
 12  sc_w           2000 non-null   int64  
 13  talk_time      2000 non-null   int64  
 14  price_range    2000 non-null   int64  
dtypes: float64(2), int64(13)
memory usage: 234.5 KB
```

In [4]:
```
mobile.describe()
```

Out[4]:

	battery_power	clock_speed	fc	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.
mean	1238.518500	1.522250	4.309500	32.046500	0.501750	140.249000	4.520500	9.916500	645.108000	1251.515500	2124.
std	439.418206	0.816004	4.341444	18.145715	0.288416	35.399655	2.287837	6.064315	443.780811	432.199447	1084.
min	501.000000	0.500000	0.000000	2.000000	0.100000	80.000000	1.000000	0.000000	0.000000	500.000000	256.
25%	851.750000	0.700000	1.000000	16.000000	0.200000	109.000000	3.000000	5.000000	282.750000	874.750000	1207.
50%	1226.000000	1.500000	3.000000	32.000000	0.500000	141.000000	4.000000	10.000000	564.000000	1247.000000	2146.
75%	1615.250000	2.200000	7.000000	48.000000	0.800000	170.000000	7.000000	15.000000	947.250000	1633.000000	3064.
max	1998.000000	3.000000	19.000000	64.000000	1.000000	200.000000	8.000000	20.000000	1960.000000	1998.000000	3998.

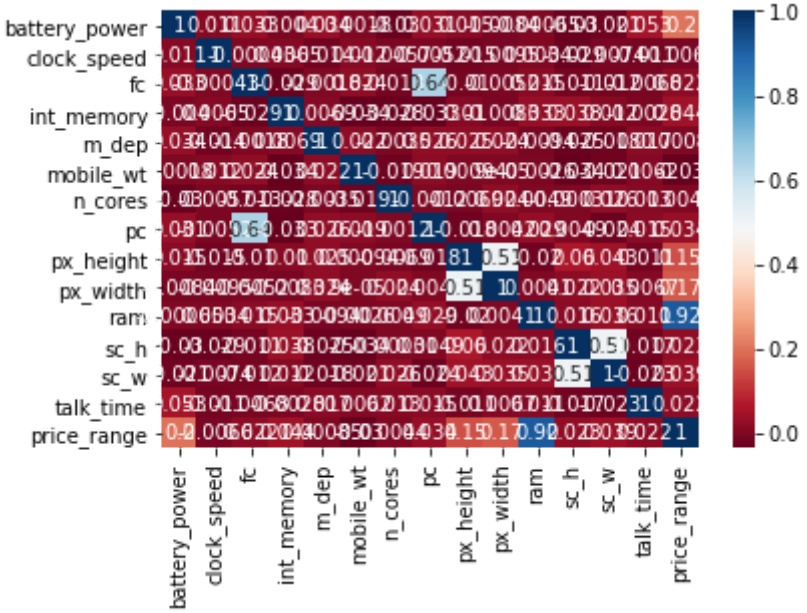
In [5]:
```
mobilecorr=mobile.corr()
mobilecorr
```

Out[5]:

ck_speed	fc	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	price_r
0.011482	0.033334	-0.004004	0.034085	0.001844	-0.029727	0.031441	0.014901	-0.008402	-0.000653	-0.029959	-0.021421	0.052510	0.20
1.000000	-0.000434	0.006545	-0.014364	0.012350	-0.005724	-0.005245	-0.014523	-0.009476	0.003443	-0.029078	-0.007378	-0.011432	-0.00
-0.000434	1.000000	-0.029133	-0.001791	0.023618	-0.013356	0.644595	-0.009990	-0.005176	0.015099	-0.011014	-0.012373	-0.006829	0.02
0.006545	-0.029133	1.000000	0.006886	-0.034214	-0.028310	-0.033273	0.010441	-0.008335	0.032813	0.037771	0.011731	-0.002790	0.04
-0.014364	-0.001791	0.006886	1.000000	0.021756	-0.003504	0.026282	0.025263	0.023566	-0.009434	-0.025348	-0.018388	0.017003	0.00
0.012350	0.023618	-0.034214	0.021756	1.000000	-0.018989	0.018844	0.000939	0.000090	-0.002581	-0.033855	-0.020761	0.006209	-0.03
-0.005724	-0.013356	-0.028310	-0.003504	-0.018989	1.000000	-0.001193	-0.006872	0.024480	0.004868	-0.000315	0.025826	0.013148	0.00
-0.005245	0.644595	-0.033273	0.026282	0.018844	-0.001193	1.000000	-0.018465	0.004196	0.028984	0.004938	-0.023819	0.014657	0.03
-0.014523	-0.009990	0.010441	0.025263	0.000939	-0.006872	-0.018465	1.000000	0.510664	-0.020352	0.059615	0.043038	-0.010645	0.14
-0.009476	-0.005176	-0.008335	0.023566	0.000090	0.024480	0.004196	0.510664	1.000000	0.004105	0.021599	0.034699	0.006720	0.16
0.003443	0.015099	0.032813	-0.009434	-0.002581	0.004868	0.028984	-0.020352	0.004105	1.000000	0.015996	0.035576	0.010820	0.91
-0.029078	-0.011014	0.037771	-0.025348	-0.033855	-0.000315	0.004938	0.059615	0.021599	0.015996	1.000000	0.506144	-0.017335	0.02
-0.007378	-0.012373	0.011731	-0.018388	-0.020761	0.025826	-0.023819	0.043038	0.034699	0.035576	0.506144	1.000000	-0.022821	0.03
-0.011432	-0.006829	-0.002790	0.017003	0.006209	0.013148	0.014657	-0.010645	0.006720	0.010820	-0.017335	-0.022821	1.000000	0.02
-0.006606	0.021998	0.044435	0.000853	-0.030302	0.004399	0.033599	0.148858	0.165818	0.917046	0.022986	0.038711	0.021859	1.00

In [17]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [8]:
```
sns.heatmap(mobilecorr,annot=True,cmap="RdBu");
```

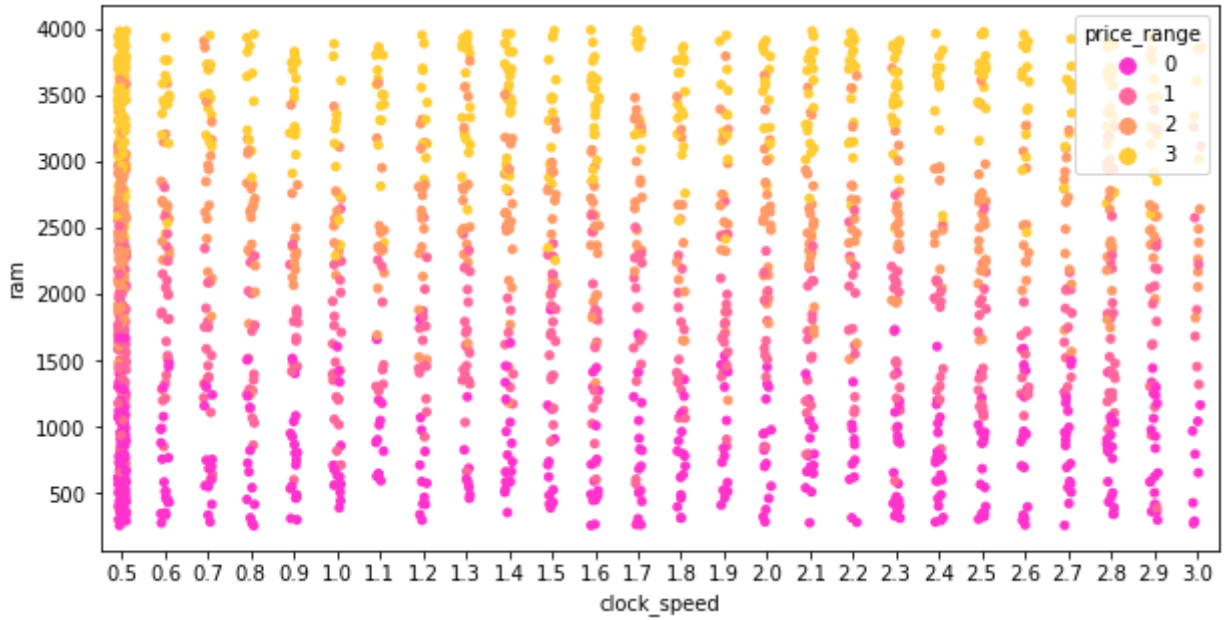


In [9]:
```
x_ind=mobile.drop("price_range",axis=1)
y_dep=mobile.price_range
```

In [10]:
```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
```

In [11]:
```
x_train, x_test, y_train, y_test = train_test_split(x_ind, y_dep, test_size=0.3, random_state=2)
```

In [18]:
```
plt.figure(figsize=(10,5))
sns.stripplot(x='clock_speed',y='ram',data=mobile,palette='spring',hue='price_range');
```



In [21]:
```
from sklearn.preprocessing import StandardScaler
se=StandardScaler()
```

In [22]:
```
x_train=se.fit_transform(x_train)
x_test=se.fit_transform(x_test)
```

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [25]:
```
mobile.shape
```

Out[25]:
```
(2000, 15)
```

In [26]:
```
import numpy as np
```

In [27]:
```
np.sqrt(2000)
```

Out[27]:
```
44.721359549995796
```

In [29]:
```
KNN=KNeighborsClassifier(n_neighbors=45,p=3,metric="euclidean")
```

In [30]:
```
KNN.fit(x_train,y_train)
```

Out[30]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=45, p=3)
```

In [32]:
```
y_pred=KNN.predict(x_test)
y_pred
```

Out[32]:
```
array([0, 2, 0, 2, 2, 2, 3, 2, 1, 2, 1, 0, 1, 2, 1, 3, 1, 1, 1, 3, 0, 2,
       0, 1, 3, 2, 2, 0, 0, 1, 0, 2, 0, 0, 1, 0, 3, 0, 2, 2, 0, 1, 2, 2,
       3, 0, 2, 1, 1, 2, 1, 3, 3, 2, 3, 0, 2, 0, 1, 1, 0, 2, 1, 2, 0, 1,
       0, 0, 0, 2, 1, 0, 1, 2, 0, 2, 2, 2, 3, 2, 2, 2, 1, 1, 0, 3, 1, 2,
       0, 1, 0, 3, 0, 0, 1, 2, 1, 3, 2, 0, 0, 3, 0, 1, 2, 3, 3, 2, 1, 1,
       3, 2, 1, 3, 2, 0, 0, 0, 2, 2, 0, 3, 3, 0, 2, 1, 2, 2, 0, 0, 1, 3,
       2, 1, 0, 3, 3, 0, 2, 1, 1, 3, 3, 1, 3, 1, 0, 2, 0, 1, 1, 2, 1, 2,
       0, 3, 0, 3, 0, 0, 0, 1, 3, 0, 1, 3, 1, 3, 1, 0, 3, 1, 0, 3, 1, 1,
       1, 1, 3, 3, 2, 2, 1, 1, 1, 0, 0, 0, 3, 1, 1, 0, 1, 1, 3, 2, 0, 1,
       3, 1, 1, 3, 3, 0, 3, 2, 1, 2, 3, 3, 3, 2, 0, 1, 2, 2, 0, 0, 3, 0,
       0, 3, 0, 0, 2, 0, 1, 3, 1, 1, 2, 3, 3, 0, 2, 0, 0, 2, 0, 2, 2, 2,
       2, 0, 2, 3, 0, 3, 0, 0, 1, 0, 2, 3, 2, 1, 1, 3, 3, 0, 0, 0, 3, 3,
       0, 3, 1, 1, 1, 1, 2, 0, 1, 0, 0, 0, 2, 0, 3, 0, 2, 3, 0, 2, 2, 1,
       1, 2, 2, 3, 0, 2, 2, 0, 3, 0, 2, 0, 1, 0, 3, 0, 1, 3, 2, 2, 3, 3,
       1, 0, 2, 1, 0, 1, 1, 2, 2, 2, 1, 0, 1, 0, 2, 2, 2, 2, 2, 3, 0, 3,
       2, 3, 2, 1, 3, 3, 0, 2, 2, 2, 3, 1, 3, 0, 0, 2, 0, 1, 1, 2, 1, 2,
       3, 1, 2, 2, 3, 1, 0, 1, 2, 3, 3, 1, 1, 2, 0, 0, 3, 2, 0, 3, 1, 0,
       1, 3, 0, 2, 2, 2, 2, 1, 1, 0, 2, 2, 1, 0, 3, 3, 1, 1, 0, 1, 2, 0,
       3, 0, 3, 3, 0, 3, 2, 2, 0, 3, 2, 1, 3, 0, 2, 0, 2, 0, 2, 0, 3, 3,
       0, 1, 0, 1, 2, 1, 0, 1, 2, 3, 3, 2, 2, 3, 1, 1, 2, 3, 3, 1, 2, 3,
       2, 3, 0, 2, 1, 1, 0, 3, 0, 2, 3, 0, 3, 3, 2, 3, 0, 1, 0, 2, 2, 0,
       0, 2, 2, 2, 2, 1, 2, 0, 2, 0, 2, 0, 0, 3, 3, 0, 3, 0, 2, 1, 2, 3,
       2, 0, 1, 2, 0, 1, 2, 0, 2, 2, 0, 2, 1, 2, 1, 3, 2, 1, 2, 2, 1, 3,
       2, 3, 0, 0, 3, 0, 3, 0, 3, 1, 1, 0, 2, 3, 1, 2, 0, 1, 2, 2, 2, 0,
       1, 2, 2, 2, 3, 0, 1, 2, 0, 3, 3, 0, 2, 2, 1, 0, 2, 1, 1, 3, 2, 0,
       2, 0, 2, 2, 0, 0, 1, 1, 0, 0, 2, 0, 0, 2, 1, 3, 3, 2, 1, 2, 0, 1,
       2, 0, 1, 1, 0, 2, 1, 3, 1, 2, 0, 2, 0, 0, 0, 0, 0, 3, 2, 3, 1, 0,
       2, 1, 1, 3, 3, 0], dtype=int64)
```

In [33]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
```

In [34]:
```
confusion_matrix(y_test,y_pred)
```

Out[34]:
```
array([[125,  23,   0,   0],
       [ 40,  85,  34,   0],
       [  2,  32,  92,  21],
       [  0,   0,  44, 102]], dtype=int64)
```

In [35]:
```
accuracy_score(y_test,y_pred)
```

Out[35]:
```
0.6733333333333333
```

In [45]:
```
error_rate=[]
for i in range(1,50):
    knn_new=KNeighborsClassifier(n_neighbors=i)
    knn_new.fit(x_train,y_train)
    y_pred1=knn_new.predict(x_test)
    error_rate.append(np.mean(y_pred1 != y_test))
```

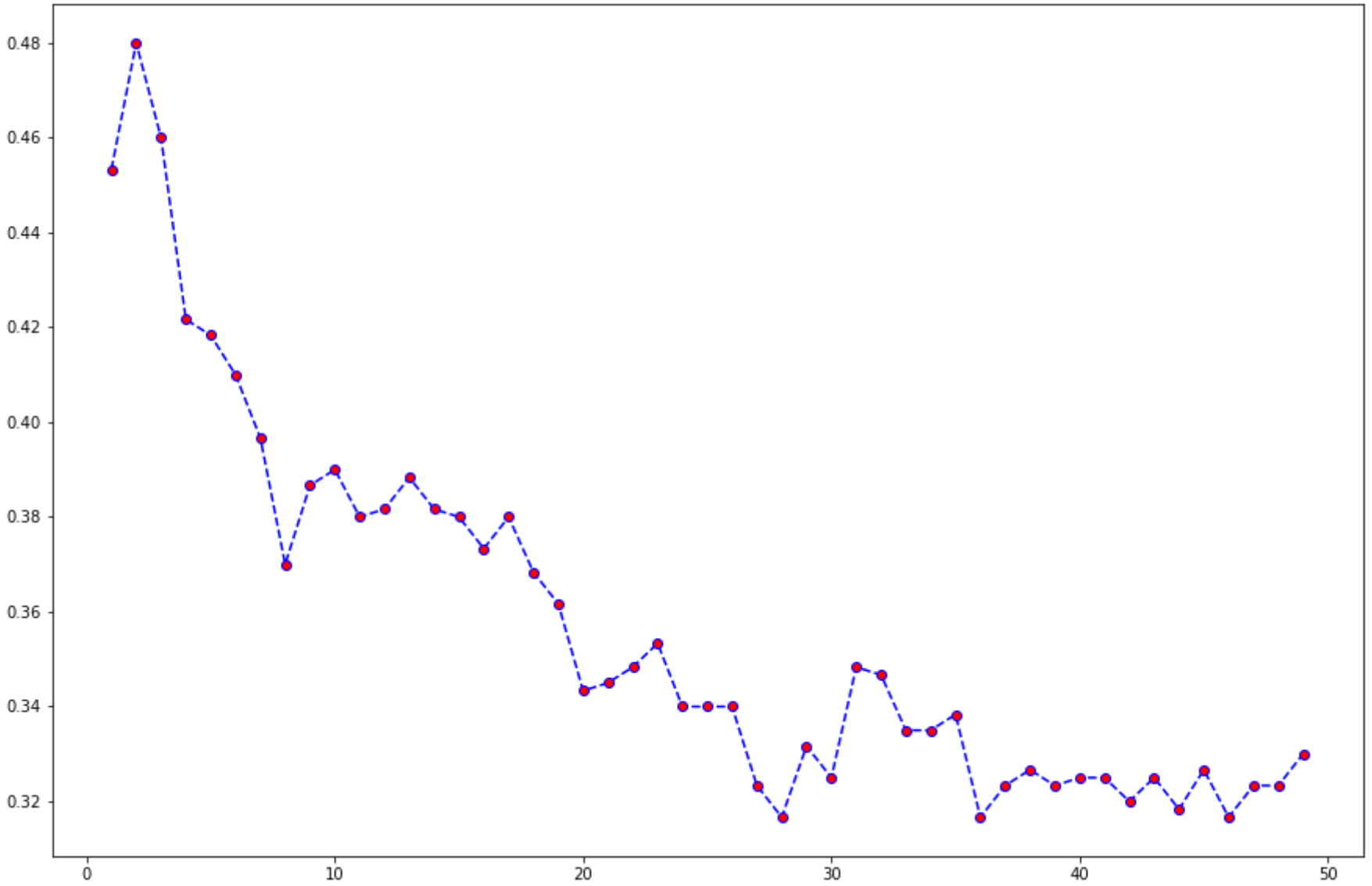
In [46]:
```
error_rate
```

Out[46]:
```
[0.4533333333333333,
 0.48,
 0.46,
 0.4216666666666667,
 0.41833333333333333,
 0.41,
 0.39666666666666667,
 0.37,
 0.38666666666666666,
 0.39,
 0.38,
 0.38166666666666665,
 0.3883333333333333,
 0.38166666666666665,
 0.38,
 0.37333333333333335,
 0.38,
 0.36833333333333335,
 0.3616666666666667,
 0.3433333333333333,
 0.345,
 0.34833333333333333,
 0.35333333333333333,
 0.34,
 0.34,
 0.34,
 0.3233333333333333,
 0.31666666666666665,
 0.33166666666666667,
 0.325,
 0.34833333333333333,
 0.3466666666666667,
 0.335,
 0.335,
 0.3383333333333333,
 0.31666666666666665,
 0.3233333333333333,
 0.32666666666666666,
 0.3233333333333333,
```

```
0.325,
0.325,
0.32,
0.325,
0.31833333333333336,
0.32666666666666666,
0.31666666666666665,
0.3233333333333333,
0.3233333333333333,
0.33]
```

In [47]:
```
plt.figure(figsize=(15,10))
plt.plot(range(1,50),error_rate,color='blue',linestyle='dashed',
        marker='o',markerfacecolor='red')
```

Out[47]:
```
[<matplotlib.lines.Line2D at 0x182d9114910>]
```



In [92]:
```
KNN=KNeighborsClassifier(n_neighbors=47,p=3,metric="euclidean")
```

In [93]:
```
KNN.fit(x_train,y_train)
```

Out[93]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=47, p=3)
```

In [94]:
```
y_pred=KNN.predict(x_test)
```

In [95]:
```
accuracy_score(y_test,y_pred)
```

Out[95]:
```
0.6766666666666666
```

by taking nearest k value as 47 i got final accuracy of 0.676.Accuracy improved by point values as 0.003

In [ ]:
```

```