ФКН НИУ ВШЭ Москва

Магистратура

Мокрий Юрий, Вальчук Ксения, Поляков Павел

Современные методы анализа данных

Задания по Домашнему Проекту 2018-19

FCS NRU HSE Moscow

MSc Programme

Yury Mokry, Kseniya Valchuk, Pavel Polyakov

“Modern” data analysis subgroup AID

Assignment 1

We used dataset which contains characteristics of various mobile phones. Using these characteristics, we can reveal the price range for a certain mobile phone, and also reveal the most determining factors for mobile phone price.

The dataset was taken from Kaggle data science platform and it can be downloaded from https://www.kaggle.com/iabhishekofficial/mobile-price-classification (https://www.kaggle.com/iabhishekofficial/mobile-price-classification).

There are 2000 objects in the dataset, and each object is described by 21 features, such as random access memory (ram), front and primary camera megapixels, support of 4G and dual sim, etc.

Target variable is price range. There are 4 such ranges.

Analysis of the dataset with characteristics of mobile phones will be important for our group to start our own company. Using this dataset, we can more precisely estimate the prices of our future products.

In [306]:
```
from IPython.display import HTML
```

In [307]:
```
HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
The raw code for this IPython notebook is by default hidden for easier readin
g.
To toggle on/off the raw code, click <a href="javascript:code_toggle()">here</
a>.''')
```

Out[307]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Assignment 2

Let's look at first five objects of our data.

In [3]:
```
%matplotlib inline
import sklearn
import pandas as pd
import matplotlib.pyplot as plt
```

In [4]:
```
data = pd.read_csv('train_mobile.csv')
```

In [5]:
```
data.head()
```

Out[5]:

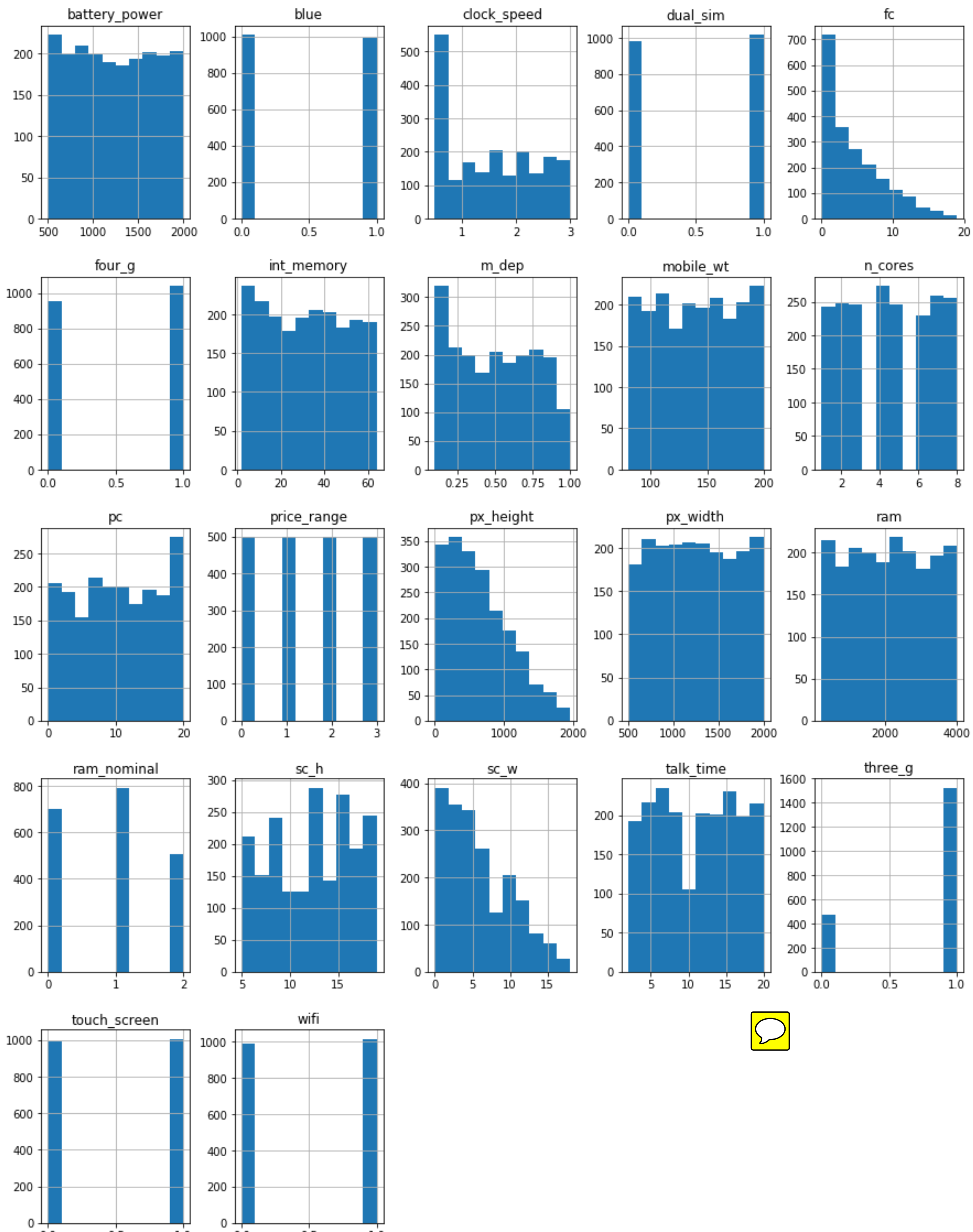
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_c
0	842	0	2.2	0	1	0	7	0.6	188	
1	1021	1	0.5	1	0	1	53	0.7	136	
2	563	1	0.5	1	2	1	41	0.9	145	
3	615	1	2.5	0	0	0	10	0.8	131	
4	1821	1	1.2	0	13	1	44	0.6	141	

5 rows × 21 columns

Let's visualize features.

In [303]:
```
fig = plt.figure(figsize=(15,20))
ax = fig.gca()
data.hist(ax=ax)
plt.show()
```

```
/Users/kaktus/virtualenv/lib/python3.6/site-packages/IPython/core/interactive
shell.py:2961: UserWarning: To output multiple subplots, the figure containin
g the passed axes is being cleared
  exec(code_obj, self.user_global_ns, self.user_ns)
```



For clustering, we select 6 following features:

- sc_h
- clock_speed
- battery_power
- talk_time
- ram

In [7]:
```
SELECTED_FEATURES = [
    "sc_h",
    "clock_speed",
    "battery_power",
    "talk_time",
    "ram"
]
```

We believe that these features are one's of the most significant factors you consider when you want to buy a new telephone.

In [8]:
```
selected_features_data = data[SELECTED_FEATURES].copy()
```

In [9]:
```
selected_features_data.head()
```

Out[9]:

	sc_h	clock_speed	battery_power	talk_time	ram
0	9	2.2	842	19	2549
1	17	0.5	1021	7	2631
2	11	0.5	563	9	2603
3	16	2.5	615	11	2769
4	8	1.2	1821	15	1411

To prepare data for clustering, we should standartize it.

Standartized data:

In [10]:
```
from sklearn.preprocessing import StandardScaler
```

In [11]:

```
values = StandardScaler().fit_transform(selected_features_data) # return numpy
 array, not pandas.DataFrame
standartized_selected_features_data = pd.DataFrame(
    data=values,
    index=selected_features_data.index,
    columns=selected_features_data.columns
)
```

In [12]:

```
standartized_selected_features_data.head()
```

Out[12]:

	sc_h	clock_speed	battery_power	talk_time	ram
0	-0.784983	0.830779	-0.902597	1.462493	0.391703
1	1.114266	-1.253064	-0.495139	-0.734267	0.467317
2	-0.310171	-1.253064	-1.537686	-0.368140	0.441498
3	0.876859	1.198517	-1.419319	-0.002014	0.594569
4	-1.022389	-0.395011	1.325906	0.730240	-0.657666

Now, let's apply K-means to our data.

5 clusters

Let's try 10 random initializations for K-means clusters.

In [13]:
```
from sklearn.cluster import KMeans

best_inertia = float("inf")
for iteration in range(10):
    kmeans = KMeans(init='random', n_init=1, n_clusters=5, random_state=1356 *
 iteration + 13487).fit(
        standartized_selected_features_data)
    print("Attempt {} \nSum of squared distances of samples to their cluster c
enter:".format(iteration),
          kmeans.inertia_)
    if best_inertia > kmeans.inertia_:
        best_kmeans, best_inertia = kmeans, kmeans.inertia_
```

```
Attempt 0
Sum of squared distances of samples to their cluster center: 6214.92721509163
4
Attempt 1
Sum of squared distances of samples to their cluster center: 6125.37376465660
1
Attempt 2
Sum of squared distances of samples to their cluster center: 6218.39833030713
6
Attempt 3
Sum of squared distances of samples to their cluster center: 6166.14206392849
3
Attempt 4
Sum of squared distances of samples to their cluster center: 6230.75041180155
5
Attempt 5
Sum of squared distances of samples to their cluster center: 6189.35209527132
3
Attempt 6
Sum of squared distances of samples to their cluster center: 6232.89232264389
1
Attempt 7
Sum of squared distances of samples to their cluster center: 6134.74517665775
1
Attempt 8
Sum of squared distances of samples to their cluster center: 6152.34088769880
7
Attempt 9
Sum of squared distances of samples to their cluster center: 6125.98866733692
5
```

In [15]:
```
def create_table_of_features_means(data, clusters):
    k_clusters = clusters.max() + 1
    means = []
    for cluster in range(k_clusters):
        cluster_features_means = data.values[clusters == cluster].mean(axis=0)
        means.append(["cluster_{}".format(cluster), (clusters == cluster).sum
(), *cluster_features_means])
    grand_features_means = data.values.mean(axis=0)
    means.append(["grand", data.shape[0], *grand_features_means])
    return pd.DataFrame(
        data=means,
        columns=["mean_type", "objects_in_cluster", *data.columns]
    )
```

In [16]:
```
table_of_features_means = create_table_of_features_means(selected_features_dat
a, clusters=best_kmeans.labels_)
```

Within-cluster and grand means of each chosen feature.

In [17]:
```
table_of_features_means.round(3)
```

Out[17]:

	mean_type	objects_in_cluster	sc_h	clock_speed	battery_power	talk_time	ram
0	cluster_0	362	9.936	2.247	1187.348	6.110	1643.091
1	cluster_1	395	12.544	2.319	1372.365	16.327	2156.190
2	cluster_2	407	7.948	0.852	1266.509	12.587	2377.676
3	cluster_3	416	15.368	1.399	1178.613	8.382	3234.031
4	cluster_4	420	15.317	0.919	1188.955	11.312	1163.955
5	grand	2000	12.306	1.522	1238.518	11.011	2124.213

In [18]:
```
def build_means_relative_differences_table(table):
    values = table.values.copy()
    values[:, 2:] = (values[:, 2:] - values[-1:, 2:]) / values[-1:, 2:] * 100
    return pd.DataFrame(
        data=values,
        columns=table.columns
    )
```

Relative differences between within-cluster and grand means of each chosen feature.

In [19]:
```
build_means_relative_differences_table(table_of_features_means)
```

Out[19]:

	mean_type	objects_in_cluster	sc_h	clock_speed	battery_power	talk_time	ram
0	cluster_0	362	-19.2584	47.6261	-4.13158	-44.5055	-22.6494
1	cluster_1	395	1.93234	52.3727	10.8069	48.2752	1.50535
2	cluster_2	407	-35.413	-44.0083	2.25997	14.315	11.9321
3	cluster_3	416	24.8754	-8.09404	-4.83687	-23.8742	52.2461
4	cluster_4	420	24.46	-39.6414	-4.00186	2.73277	-45.2054
5	grand	2000	0	0	0	0	0

In [20]:
```
best_kmeans_5 = best_kmeans
```

9 clusters

Let's try 10 random initializations for K-means clusters.

In [21]:
```
best_inertia = float("inf")
for iteration in range(10):
    kmeans = KMeans(init='random', n_init=1, n_clusters=9, random_state=179 *
iteration + 57).fit(
        standartized_selected_features_data)
    print("Attempt {} \nSum of squared distances of samples to their cluster c
enter:".format(iteration),
          kmeans.inertia_)
    if best_inertia > kmeans.inertia_:
        best_kmeans, best_inertia = kmeans, kmeans.inertia_
```

```
Attempt 0
Sum of squared distances of samples to their cluster center: 4640.53974028566
3
Attempt 1
Sum of squared distances of samples to their cluster center: 4648.78518213401
1
Attempt 2
Sum of squared distances of samples to their cluster center: 4752.49705730709
05
Attempt 3
Sum of squared distances of samples to their cluster center: 4594.70149978795
6
Attempt 4
Sum of squared distances of samples to their cluster center: 4622.26584016876
6
Attempt 5
Sum of squared distances of samples to their cluster center: 4691.43487442977
6
Attempt 6
Sum of squared distances of samples to their cluster center: 4646.80094442248
8
Attempt 7
Sum of squared distances of samples to their cluster center: 4647.14805965670
7
Attempt 8
Sum of squared distances of samples to their cluster center: 4595.22087579760
7
Attempt 9
Sum of squared distances of samples to their cluster center: 4687.01891529347
7
```

Within-cluster and grand means of each chosen feature.

In [290]:
```
table_of_features_means = create_table_of_features_means(selected_features_dat
a, clusters=best_kmeans.labels_)
table_of_features_means
```

Out[290]:

	mean_type	objects_in_cluster	sc_h	clock_speed	battery_power	talk_time	ram
0	cluster_0	199	13.140704	0.847236	906.618090	6.618090	3132.100503
1	cluster_1	233	15.317597	2.332618	1039.459227	11.527897	3162.442060
2	cluster_2	234	14.064103	1.164103	1625.944444	6.594017	1205.841880
3	cluster_3	217	11.248848	2.426267	1469.811060	16.341014	1397.700461
4	cluster_4	228	10.750000	2.274561	929.881579	6.149123	1383.100877
5	cluster_5	197	8.390863	1.735025	1634.345178	7.319797	2983.517766
6	cluster_6	215	14.851163	1.064186	1657.646512	15.679070	2834.195349
7	cluster_7	262	7.576336	0.942748	1067.274809	14.843511	1954.194656
8	cluster_8	215	15.883721	0.917674	860.497674	13.144186	1294.744186
9	grand	2000	12.306500	1.522250	1238.518500	11.011000	2124.213000

Relative differences between within-cluster and grand means of each chosen feature.

In [23]:
```
build_means_relative_differences_table(table_of_features_means)
```

Out[23]:

	mean_type	objects_in_cluster	sc_h	clock_speed	battery_power	talk_time	ram
0	cluster_0	199	6.77856	-44.3432	-26.7982	-39.8956	47.4476
1	cluster_1	233	24.4675	53.2349	-16.0724	4.69437	48.8759
2	cluster_2	234	14.2819	-23.5275	31.2814	-40.1143	-43.2335
3	cluster_3	217	-8.59426	59.3869	18.6749	48.4063	-34.2015
4	cluster_4	228	-12.6478	49.421	-24.9198	-44.1547	-34.8888
5	cluster_5	197	-31.8176	13.9777	31.9597	-33.5229	40.4529
6	cluster_6	215	20.6774	-30.0912	33.8411	42.3946	33.4233
7	cluster_7	262	-38.4363	-38.0688	-13.8265	34.8062	-8.00383
8	cluster_8	215	29.0677	-39.7159	-30.522	19.3732	-39.0483
9	grand	2000	0	0	0	0	0

Clustering with 5 clusters seems to be more interesting, because clusters look more infomative. Moreover, it is easier to analyze contingency tables with smaller amount of rows.

In [24]:
```
best_kmeans_9 = best_kmeans
```

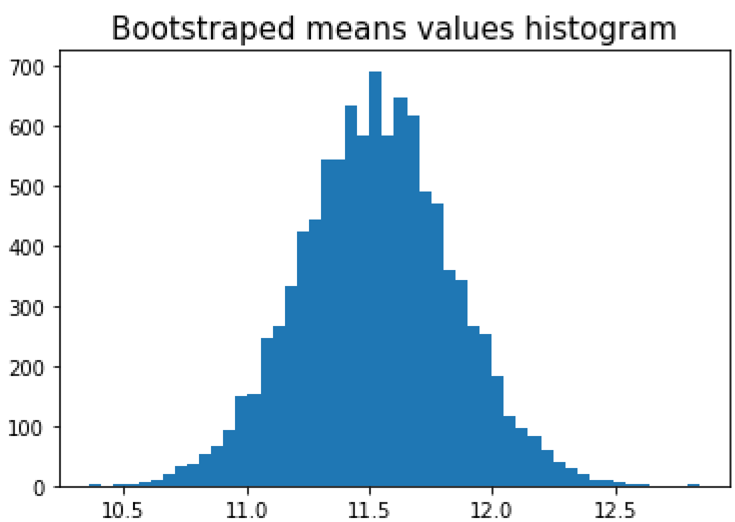
Assignment 3

We chose cluster 1 (K=9) (with the largest average "ram" feature value) and "talk_time" feature, as a feature with the smallest relative difference with grand mean value.

In [25]:
```
values = selected_features_data["talk_time"][best_kmeans.labels_ == 1]
```

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

def bootstrap(values):
    return np.random.choice(values, size=len(values), replace=True)

np.random.seed(132124)
means = np.array([bootstrap(values).mean() for i in range(10000)])
```

In [292]:
```
plt.hist(means, bins=50)
plt.title('Bootstraped means values histogram', size=15)
plt.show()
```



Now, in order to validate within-cluster mean, we build 95% confidence interval for mean using bootstrap samples. There are two ways:

pivotal confidence interval

In [28]:
```
std = np.std(means)
```

In [29]:
```
print(np.mean(means) - 1.96 * std, np.mean(means) + 1.96 * std)
```

```
10.918462488292318 12.137444807845023
```

non-pivotal confidence interval

In [30]:
```
means = np.sort(means)
print(means[round(0.025 * len(means))], means[round(0.975 * len(means))])
```

```
10.918454935622318 12.15450643776824
```

The grand mean lies in both confidence interval, so we can say that the within-cluster mean is not significantly different from the grand mean.

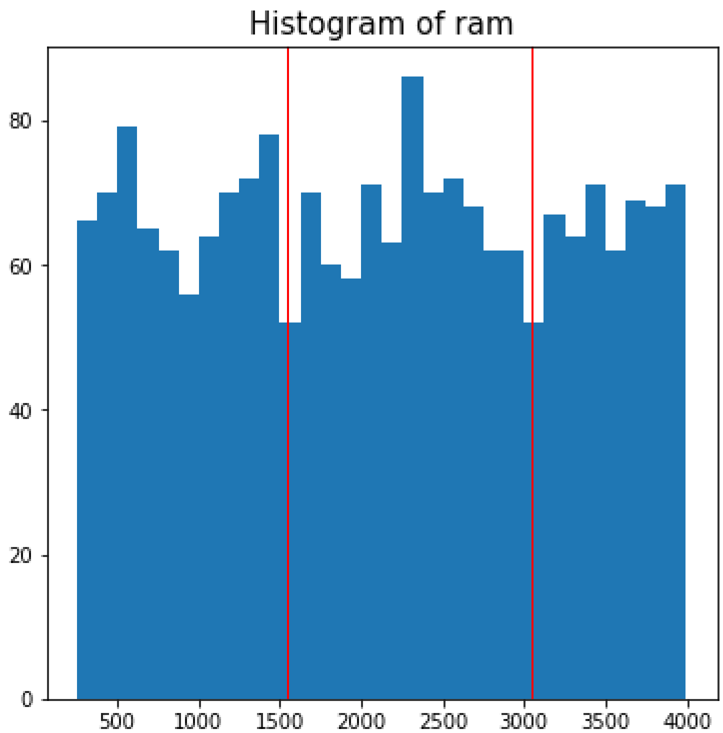
Assignment 4

Our two features choice:

- price_range (it's already a nominal feature with 4 categories, where each category represents continuous of mobile phone prices).
- ram (it's a quantitive feature, we are going to develop a nominal feature)

ram feature

In [305]:
```
plt.figure(figsize=(6, 6))
plt.hist(data['ram'], bins=30)
plt.title('Histogram of ram', size=15)
plt.axvline(1550, color='red', linestyle='-', linewidth=1)
plt.axvline(3050, color='red', linestyle='-', linewidth=1)


plt.show()
```



We chose 1550 and 3050 as border for categories.

In [32]:
```
data['ram_nominal'] = 1
data.loc[data['ram'] < 1550, 'ram_nominal'] = 0
data.loc[data['ram'] > 3050, 'ram_nominal'] = 2
```

In [33]:
```
data[['ram', 'ram_nominal']].head(10)
```

Out[33]:

	ram	ram_nominal
0	2549	1
1	2631	1
2	2603	1
3	2769	1
4	1411	0
5	1067	0
6	3220	2
7	700	0
8	1099	0
9	513	0

Contingency tables for price_range and ram_nominal

In [34]:
```
def create_contingency_table(data, feature_name, clusters):
    categories = data[feature_name].values
    k_categories = categories.max() + 1
    k_clusters = clusters.max() + 1
    coocuriences = []
    for cat in range(k_categories):
        coocuriences.append(["{}_{}".format(feature_name, cat)])
        for cluster in range(k_clusters):
            coocuriences[-1].append(((categories == cat) & (clusters == cluste
r)).sum())
        coocuriences[-1].append(np.sum(coocuriences[-1][1:]))
    coocuriences.append(["total", *list(np.array(coocuriences)[:, 1:].astype(i
nt).sum(axis=0))])
    coocuriences = np.array(coocuriences)
    return pd.DataFrame(
        data=coocuriences,
        columns=[feature_name, *('cluster_{}'.format(i) for i in range(k_clust
ers)), 'total']
    )
```

Contingency table for "ram_nominal" feature

In [35]:
```
create_contingency_table(data, 'ram_nominal', best_kmeans_5.labels_)
```

Out[35]:

	ram_nominal	cluster_0	cluster_1	cluster_2	cluster_3	cluster_4	total
0	ram_nominal_0	185	123	91	0	303	702
1	ram_nominal_1	143	189	201	142	117	792
2	ram_nominal_2	34	83	115	274	0	506
3	total	362	395	407	416	420	2000

Contingency table for "price_range" feature

In [294]:
```
price_contingency_table = create_contingency_table(data, 'price_range', best_k
means_5.labels_)
price_contingency_table
```

Out[294]:

	price_range	cluster_0	cluster_1	cluster_2	cluster_3	cluster_4	total
0	price_range_0	142	78	53	0	227	500
1	price_range_1	114	101	107	27	151	500
2	price_range_2	74	121	131	135	39	500
3	price_range_3	32	95	116	254	3	500
4	total	362	395	407	416	420	2000

Contingencies tables look similar to each other (probably, "price_range" and "ram" are highly correlated features). So, for futher consideration, we decided to choose "price_range" feature.

Quetelet index table for price_range

In [37]:
```
def create_quetelet_table(contingency_table):
    quetelet_table = contingency_table.copy()
    rows, columns = contingency_table.shape
    for i in range(rows - 1):
        for j in range(1, columns - 1):
            N = int(contingency_table.iloc[rows-1, columns-1])
            Nij = int(contingency_table.iloc[i, j])
            Ni = int(contingency_table.iloc[i, columns-1])
            Nj = int(contingency_table.iloc[rows-1, j])
            quetelet_table.iloc[i, j] = ((N * Nij) / (Ni * Nj) - 1) * 100
    return quetelet_table
```

In [38]:
```
price_quetelet_table = create_quetelet_table(price_contingency_table)
price_quetelet_table
```

Out[38]:

	price_range	cluster_0	cluster_1	cluster_2	cluster_3	cluster_4	total
0	price_range_0	56.9061	-21.0127	-47.9115	-100	116.19	500
1	price_range_1	25.9669	2.27848	5.15971	-74.0385	43.8095	500
2	price_range_2	-18.232	22.5316	28.7469	29.8077	-62.8571	500
3	price_range_3	-64.6409	-3.79747	14.0049	144.231	-97.1429	500
4	total	362	395	407	416	420	2000

In [39]:
```
def compute_summary_quetelet_coefficient(contingency_table, quetelet_table):
    contingency_values = contingency_table.values[:-1, 1:-1].astype(float)
    quetelet_values = quetelet_table.values[:-1, 1:-1].astype(float)
    return (quetelet_values * contingency_values).sum() / contingency_values.s
um() / 100
```

Summary Quetelet coefficient and Chi-square association coefficient (in fact, they are equal).

In [40]:
```
summary_quetelet_coeff = compute_summary_quetelet_coefficient(price_contingenc
y_table, price_quetelet_table)
summary_quetelet_coeff
```

Out[40]:
```
0.40445170908038774
```

NX^2 statistic follows the χ^2-distribution with $(5-1)(4-1) = 12$ degrees of freedom.

In [41]:
```
nchi_2 = summary_quetelet_coeff * data.shape[0]
nchi_2
```

Out[41]:
```
808.9034181607755
```

The 95th percentile of χ^2_{12} distribution is 21. So the hypothesis of independence is to be rejected at 95% confidence level.

The number of objects we need to reject hypothesis of independence at 95% confidence level can be estimated as: 95th percentile divided by Chi-square association coefficient.

In [42]:
```
21 / summary_quetelet_coeff
```

Out[42]:
```
51.92214429690071
```

So, it's about 52 objects.

Assignment 5

PCA: Hidden Factor and Data Visualization

Let us select the following 4 features from our dataset : ram (Random Access Memory in Megabytes), int_memory (Internal Memory in Gigabytes), n_cores (Number of cores of processor), and clock_speed (speed at which microprocessor executes instructions. We assume that the memory characteristics and processor performance are the determining factors for mobile phone price.

In [44]:
```
data_short = data[['ram', 'n_cores', 'int_memory', 'clock_speed']]
```

In [264]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 100))
scaler.fit(data_short)
data_scaled = scaler.transform(data_short)
```

In [265]:
```
u, s, v = np.linalg.svd(data_scaled, full_matrices=False)
```

In [266]:
```
u.shape, s.shape, v.shape
```

Out[266]:
```
((2000, 4), (4,), (4, 4))
```

In [267]:
```
first_singular_vector = np.abs(v[:, 0])
```

In [268]:
```
data_short.head()
```

Out[268]:

	ram	n_cores	int_memory	clock_speed
0	2549	2	7	2.2
1	2631	3	53	0.5
2	2603	5	41	0.5
3	2769	6	10	2.5
4	1411	2	44	1.2

In [269]:
```
weights =  first_singular_vector * (1/ np.sum(first_singular_vector))
print('Weights', weights)
```

```
Weights [0.30537398 0.01176073 0.24700128 0.43586401]
```

In [270]:
```
hidden_vector = data_scaled.dot(weights)
```

In [271]:
```
print('Hidden vector:' ,hidden_vector)
```

```
Hidden vector: [50.51122991 40.03556802 35.3624187  ... 44.55329304 30.17750771
 74.01532815]
```

In [274]:
```
np.corrcoef(hidden_vector, y)
```

Out[274]:
```
array([[1.        , 0.45307349],
       [0.45307349, 1.        ]])
```

Obtained hidden factor has positive correlation with price range.

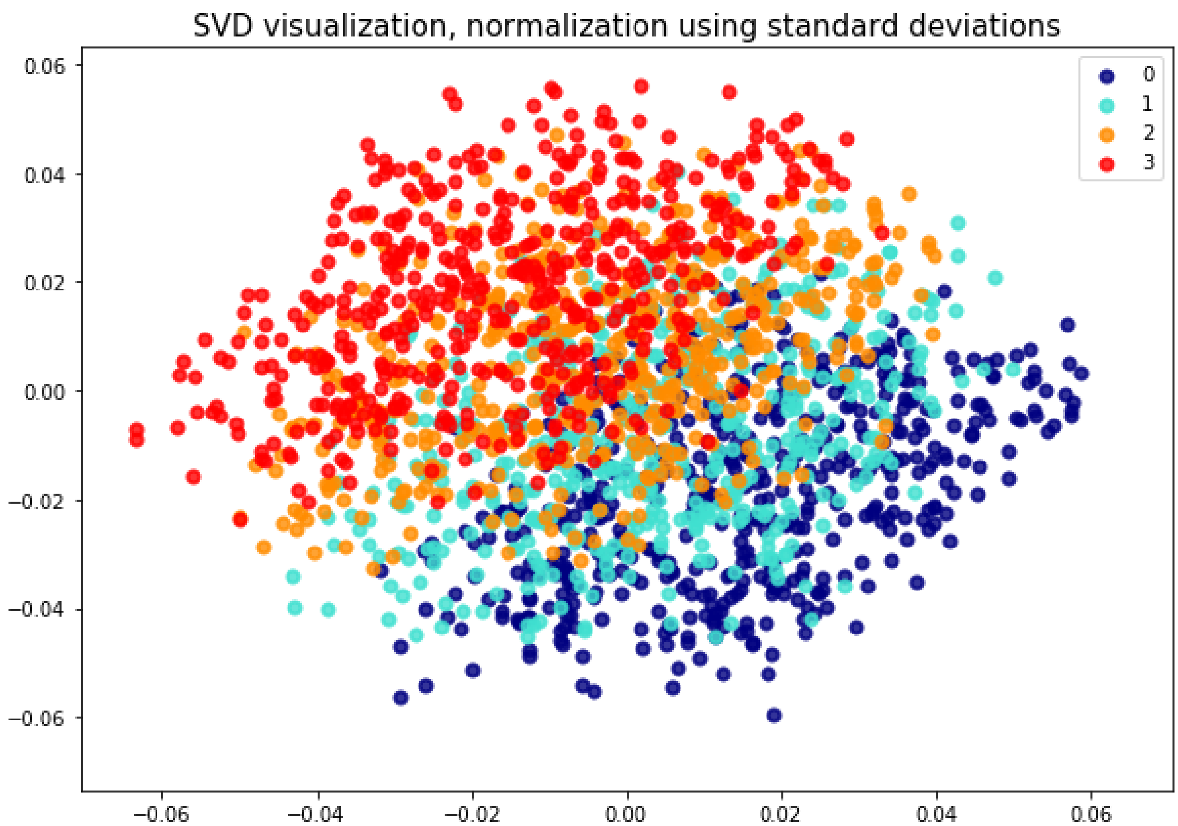
It can be interpretated as performance of mobile phone.

Normalization using standard deviations:

In [285]:
```
scaler = StandardScaler()
scaler.fit(data_short)
data_scaled = scaler.transform(data_short)

u, s, v = np.linalg.svd(data_scaled)

singular_vectors = u[:, 0:2]
```

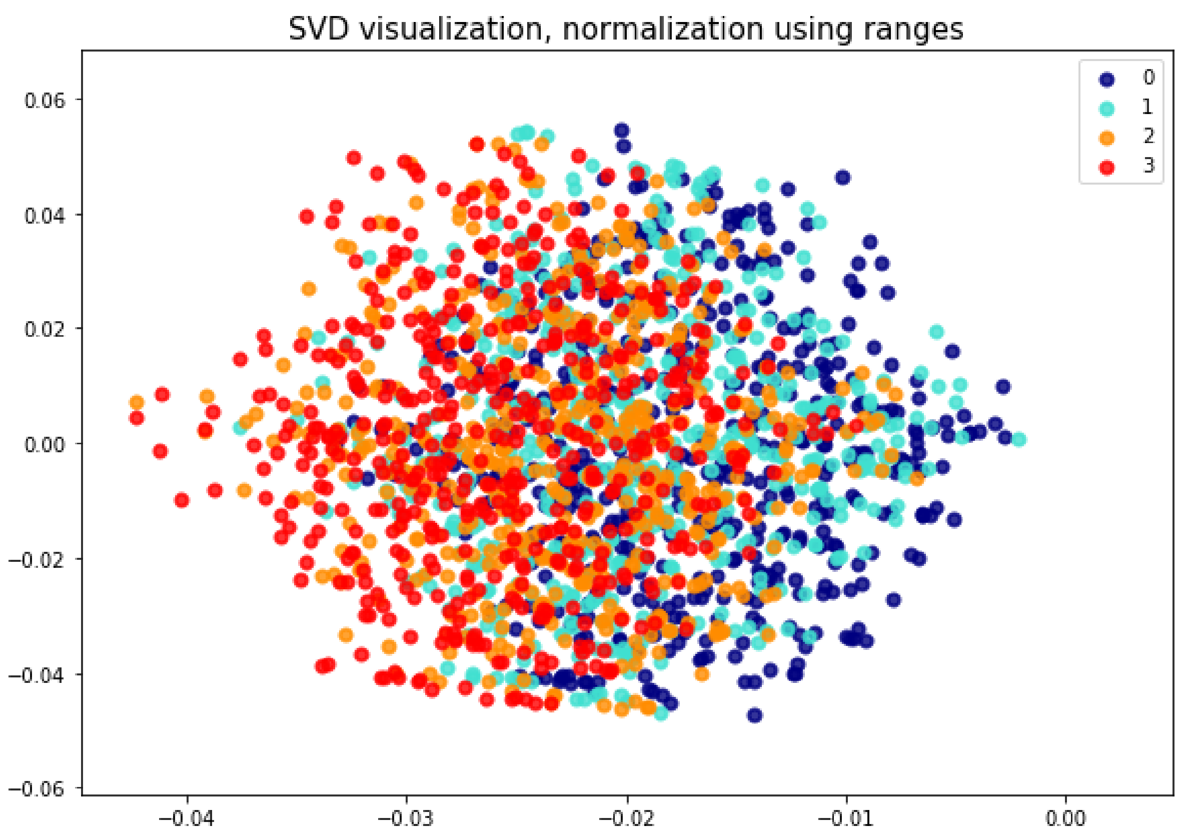
In [286]:
```
fig = plt.figure(figsize=(10, 7))

for color, i, target_name in zip(colors, [0, 1, 2, 3], target_names):
    plt.scatter(singular_vectors[y == i, 0], singular_vectors[y == i, 1], colo
r=color, alpha=.8, lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('SVD visualization, normalization using standard deviations', size=1
5)
```

Out[286]:
```
Text(0.5, 1.0, 'SVD visualization, normalization using standard deviations')
```



In [287]:
```
scaler = MinMaxScaler(feature_range=(0, 100))
scaler.fit(data_short)
data_scaled = scaler.transform(data_short)

u, s, v = np.linalg.svd(data_scaled)

singular_vectors = u[:, 0:2]

fig = plt.figure(figsize=(10, 7))

for color, i, target_name in zip(colors, [0, 1, 2, 3], target_names):
    plt.scatter(singular_vectors[y == i, 0], singular_vectors[y == i, 1], colo
r=color, alpha=.8, lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('SVD visualization, normalization using ranges', size=15)
```

Out[287]: Text(0.5, 1.0, 'SVD visualization, normalization using ranges')



As it can be seen from two different SVD normalizations, normalization which uses standard deviations is better.

It can be explained by the fact that contribution of all features to data scatter for mormalization with standart deviations are equal to each other.

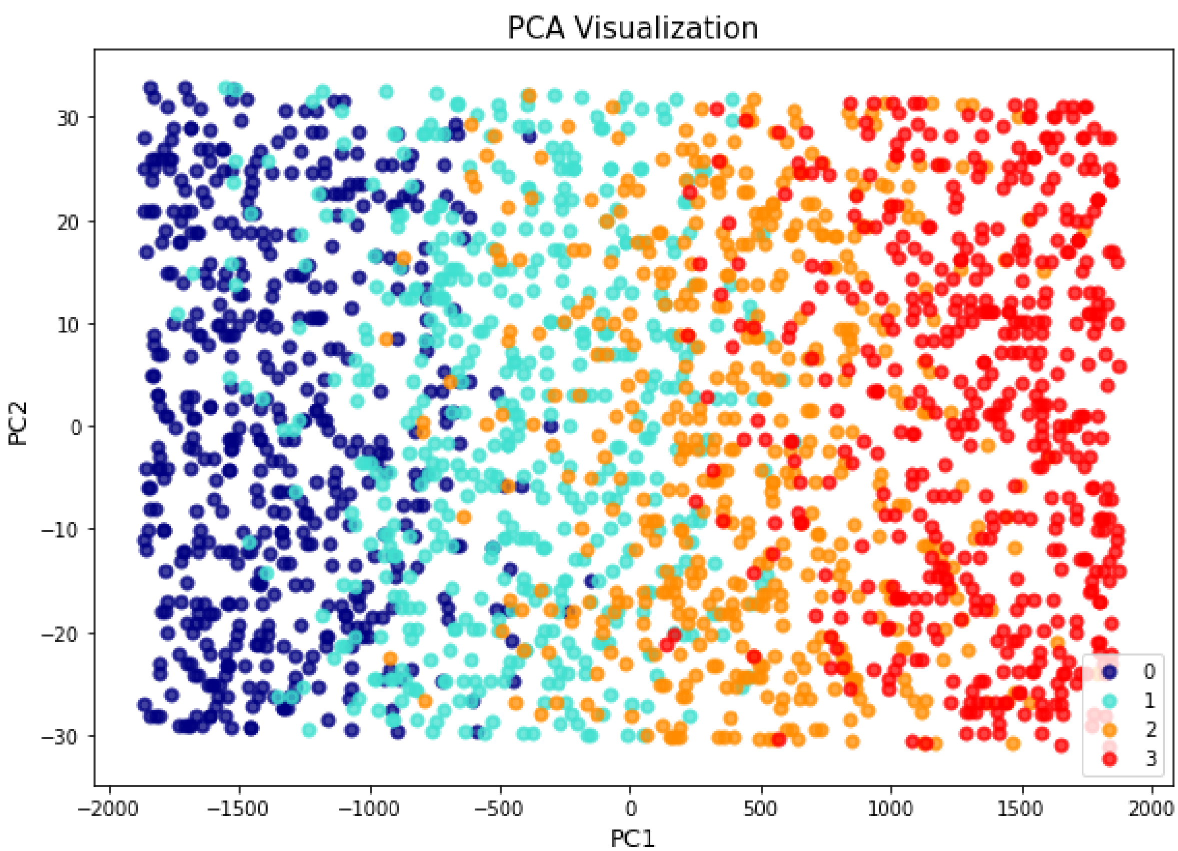
In [126]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
scaler.fit(data_short)
data_scaled = scaler.transform(data_short)

new_scaled = svd.fit_transform(data_short)
```

In [47]:
```
y = data['price_range']
```

In [43]:
```
from sklearn.decomposition import PCA
```

In [78]:
```
fig = plt.figure(figsize=(10, 7))

pca = PCA(n_components=2)
data_tr = pca.fit(data_short).transform(data_short)
colors = ['navy', 'turquoise', 'darkorange', 'red']
target_names = ['0', '1', '2', '3']
lw = 2


fig = plt.figure(figsize=(10, 7))

for color, i, target_name in zip(colors, [0, 1, 2, 3], target_names):
    plt.scatter(data_tr[y == i, 0], data_tr[y == i, 1], color=color, alpha=.75
, lw=lw,
                label=target_name)

plt.xlabel('PC1', size=13)
plt.ylabel('PC2', size=13)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA Visualization', size=15)
```

Out[78]:
```
Text(0.5, 1.0, 'PCA Visualization')

<Figure size 720x504 with 0 Axes>
```



In this visualization, labels 0, 1, 2, 3 mean price range for mobile phones, where 0 -- the cheapest mobiles, 3 -- the most expensive ones.

In [79]:
```
df = pd.DataFrame(pca.explained_variance_ratio_, index=['PC1', 'PC2'], columns
=['Explained Variance Ratio'])

df
```

Out[79]:

	Explained Variance Ratio
PC1	0.999716
PC2	0.000279

First principal component explains 99.9716% of variance.

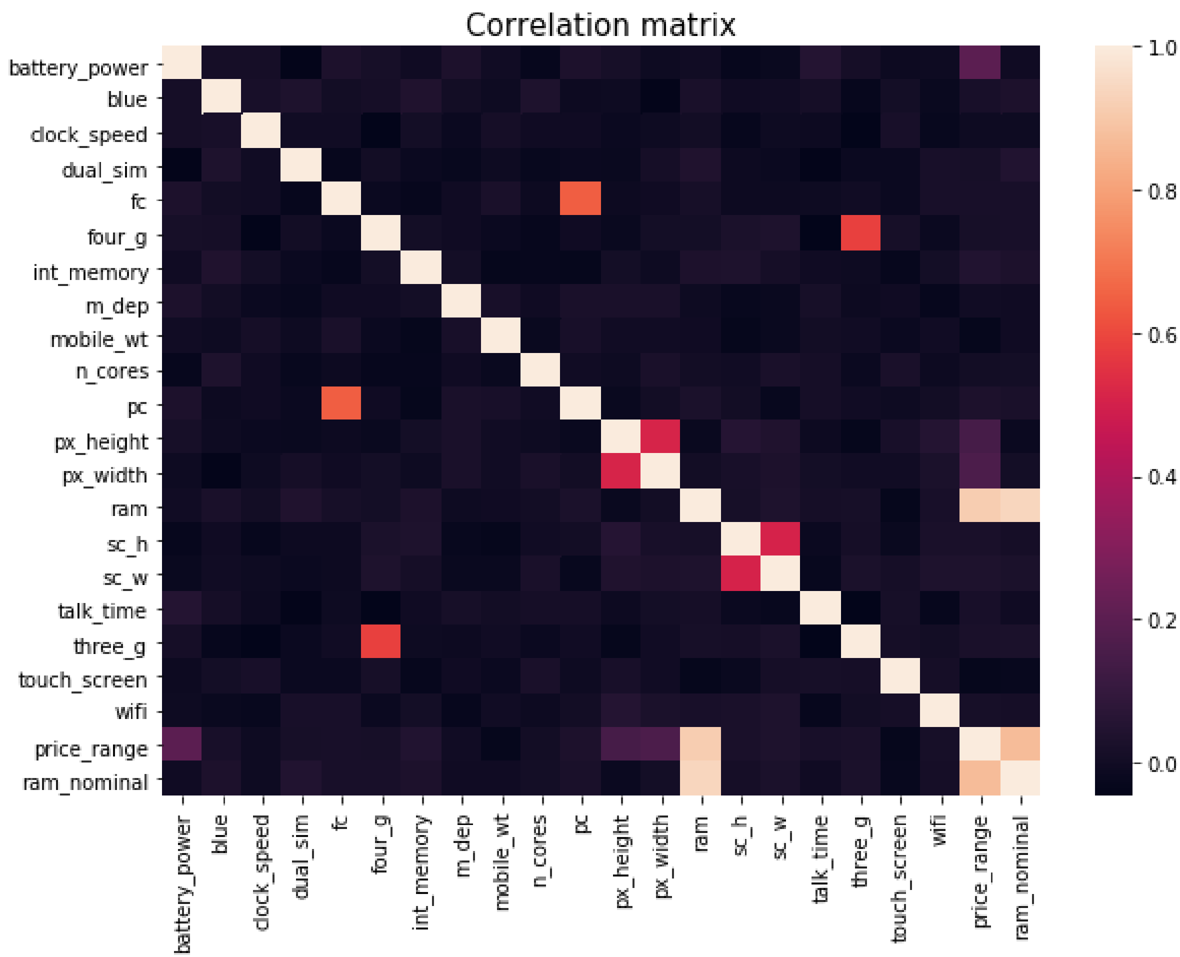
In PCA visualization, the separation between predefined groups is more explicit than for SVD.

Assignment 6

Let's display correlation matrix of features in order to find two appropriate features, for which we scatter-plot will be "linear-like".

In [86]:
```
import seaborn as sns

fig = plt.figure(figsize=(10, 7))

plt.title('Correlation matrix', size=15)
corr = data.corr()
sns.heatmap(corr)
```

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x10f9f63c8>



In [87]:
```
data[['fc', 'pc']].corr()
```

Out[87]:

	fc	pc
fc	1.000000	0.644595
pc	0.644595	1.000000

As it can be seen from visualization of correlation matrix, pc (Primary Camera mega pixels) and fc (Front Camera mega pixels) have high correlation coefficient, and we will display scatter plot for them.

In [89]:
```
from sklearn.linear_model import LinearRegression
```

In [110]:
```
lr = LinearRegression()
```

In [111]:
```
lr.fit(np.array(data['fc']).reshape(-1, 1), np.array(data['pc']).reshape(-1, 1
))
```

Out[111]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [112]:
```
preds = lr.predict(np.array(data['fc']).reshape(-1, 1))
```

In [114]:
```
df = pd.DataFrame([lr.coef_[0][0], lr.intercept_[0]], index=['Slope', 'Interce
pt'], columns=['Linear Regression Parameters'])

df
```

Out[114]:

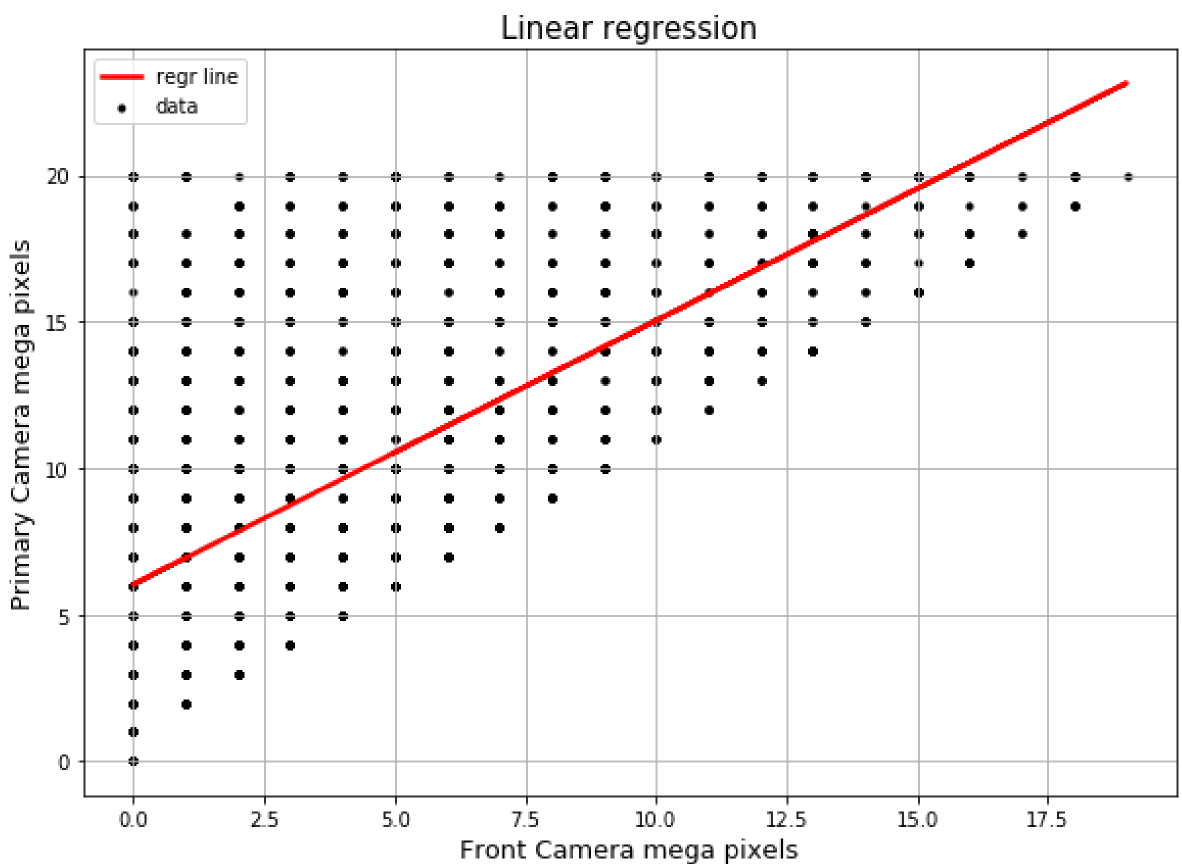
	Linear Regression Parameters
Slope	0.900398
Intercept	6.036233

This means, that when fc increases on one point, than pc increases on 0.9 points.

In [115]:
```
k_learned, b_learned = lr.coef_, lr.intercept_
```

In [118]:
```
y_rg = k_learned*np.array(data['fc']) + b_learned

fig = plt.figure(figsize=(10, 7))

plt.scatter(data['fc'], data['pc'], color='black', label='data', s=10)
plt.plot(np.array(data['fc']), y_rg.flatten(), color='red', label='regr line',
 linewidth=2.5)
plt.grid(True)
plt.legend()

plt.title('Linear regression', size=15)
plt.xlabel('Front Camera mega pixels', size=13)
plt.ylabel('Primary Camera mega pixels', size=13)
```

Out[118]: Text(0, 0.5, 'Primary Camera mega pixels')



As it can be seen from this scatter plot, the dataset we chose is synthetic and unrealistic.

We will provide you with more obvious evidence of why this data is unrealistic.

In [261]:
```
data[['ram', 'price_range']].corr()
```

Out[261]:

	ram	price_range
ram	1.000000	0.917046
price_range	0.917046	1.000000

In [120]:
```
from sklearn.metrics import r2_score

print('Determinacy coefficient: ', r2_score(data['pc'], preds))
```

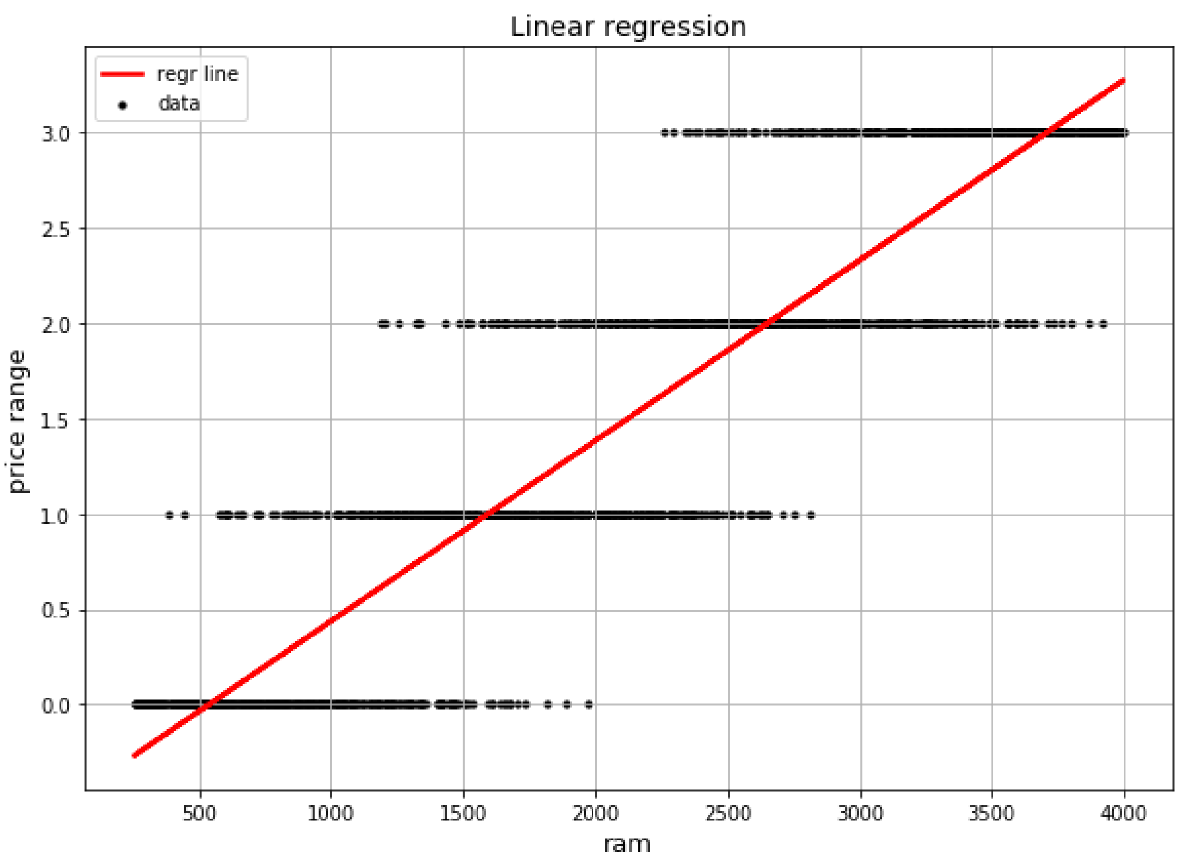
```
Determinacy coefficient:  0.4155030786023818
```

Despite price range is ordinal variable, we can also predict it using linear regression, and then choose the nearest range for predicted continious value.

In [150]:
```
lr.fit(np.array(data['ram']).reshape(-1, 1), np.array(data['price_range']).reshape(-1, 1))

preds = lr.predict(np.array(data['ram']).reshape(-1, 1))

k_learned, b_learned = lr.coef_, lr.intercept_

y_rg = k_learned*np.array(data['ram']) + b_learned
```

In [151]:
```
fig = plt.figure(figsize=(10, 7))

plt.scatter(data['ram'], data['price_range'], color='black', label='data', s=1
0)
plt.plot(np.array(data['ram']), y_rg.flatten(), color='red', label='regr line'
, linewidth=2.5)
plt.grid(True)
plt.legend()

plt.title('Linear regression', size=14)
plt.xlabel('ram', size=13)
plt.ylabel('price range', size=13)
```

Out[151]:
```
Text(0, 0.5, 'price range')
```



In [153]:
```
print('Determinacy coefficient: ', r2_score(data['price_range'], preds))
```

```
Determinacy coefficient:  0.8409728824017988
```

In [137]:
```
lr.fit(data_short, y)
```

Out[137]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Let's predict target value using previously chosen characteristics of memory and processor performance: ram (Random Access Memory in Megabytes), int_memory (Internal Memory in Gigabytes), and n_cores (Number of cores of processor).

In [138]:
```
preds = lr.predict(data_short)
```

In [139]:
```
print('Determinacy coefficient: ', r2_score(y, preds))
```

```
Determinacy coefficient:  0.8412760670637213
```

Let's predict target value using all predictors:

In [141]:
```
df = pd.read_csv('train_mobile.csv')
```

In [143]:
```
df.drop(['price_range'], axis=1, inplace=True)
```

In [144]:
```
lr.fit(df, y)
preds = lr.predict(df)
print('Determinacy coefficient: ', r2_score(y, preds))
```

```
Determinacy coefficient:  0.9186309555753549
```

It can be seen from the determination coefficients, that with even only "ram" predictor, we can obtain good predictions for price range. When we predict target value using all the predictors, 91.8% of variance in price range predictable from given features.

In [222]:
```
from sklearn.metrics import mean_absolute_error
```

In [263]:
```
print ('Relative absolute error: ', mean_absolute_error(y, preds)  * 2000 / np
.sum(np.abs(np.mean(y) - y)))
```

```
Relative absolute error:  0.34676825283446383
```

This means, that error of obtained predictions three times better than if we just use target mean values as predictions.