Praca_domowa_4

May 3, 2021

1 Praca domowa 4

Autor: Bartosz Sawicki

[128]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from dalex import datasets

from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.model_selection import  RandomizedSearchCV, train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.svm import SVR, SVC
```

1.1 Apartments

[33]:
```
apartments = pd.concat([datasets.load_apartments(),datasets.
 ↪load_apartments_test()])
```

[49]:
```
ap_X_train, ap_X_test, ap_y_train, ap_y_test = train_test_split(
    apartments.drop(['m2_price'], axis=1),
    apartments.loc[:,'m2_price'],
    random_state=123)
```

1.1.1 EDA

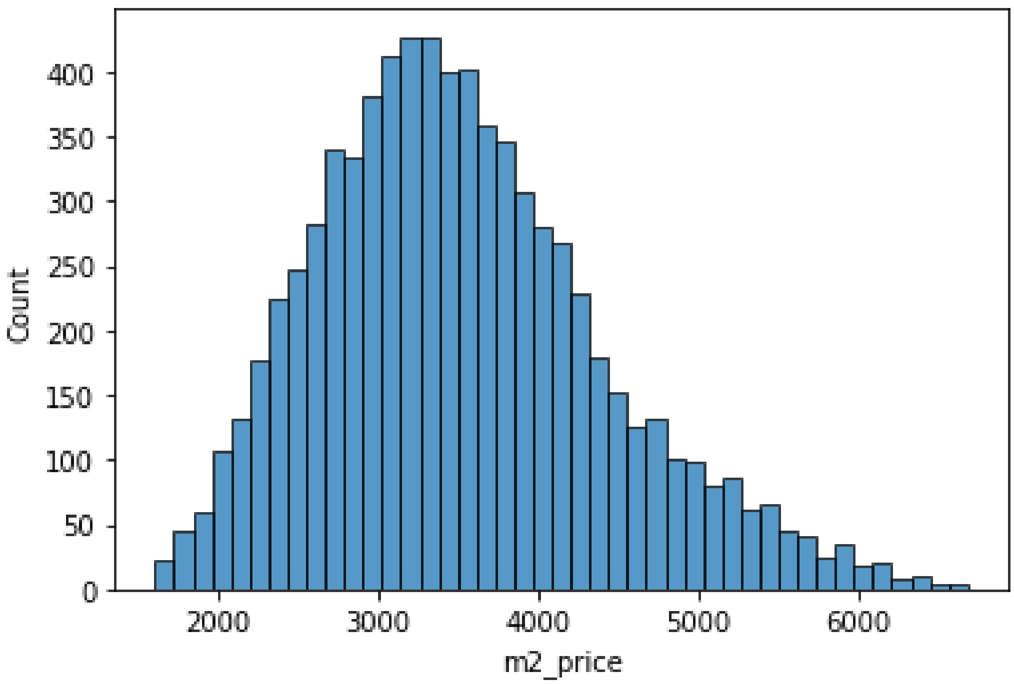
[54]:
```
apartments.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   m2_price           10000 non-null  int64
 1   construction_year  10000 non-null  int64
```

```
 2   surface            10000 non-null  int64
 3   floor              10000 non-null  int64
 4   no_rooms           10000 non-null  int64
 5   district           10000 non-null  object
dtypes: int64(5), object(1)
memory usage: 546.9+ KB
```

[41]:
```
sns.histplot(ap_y_train)
```

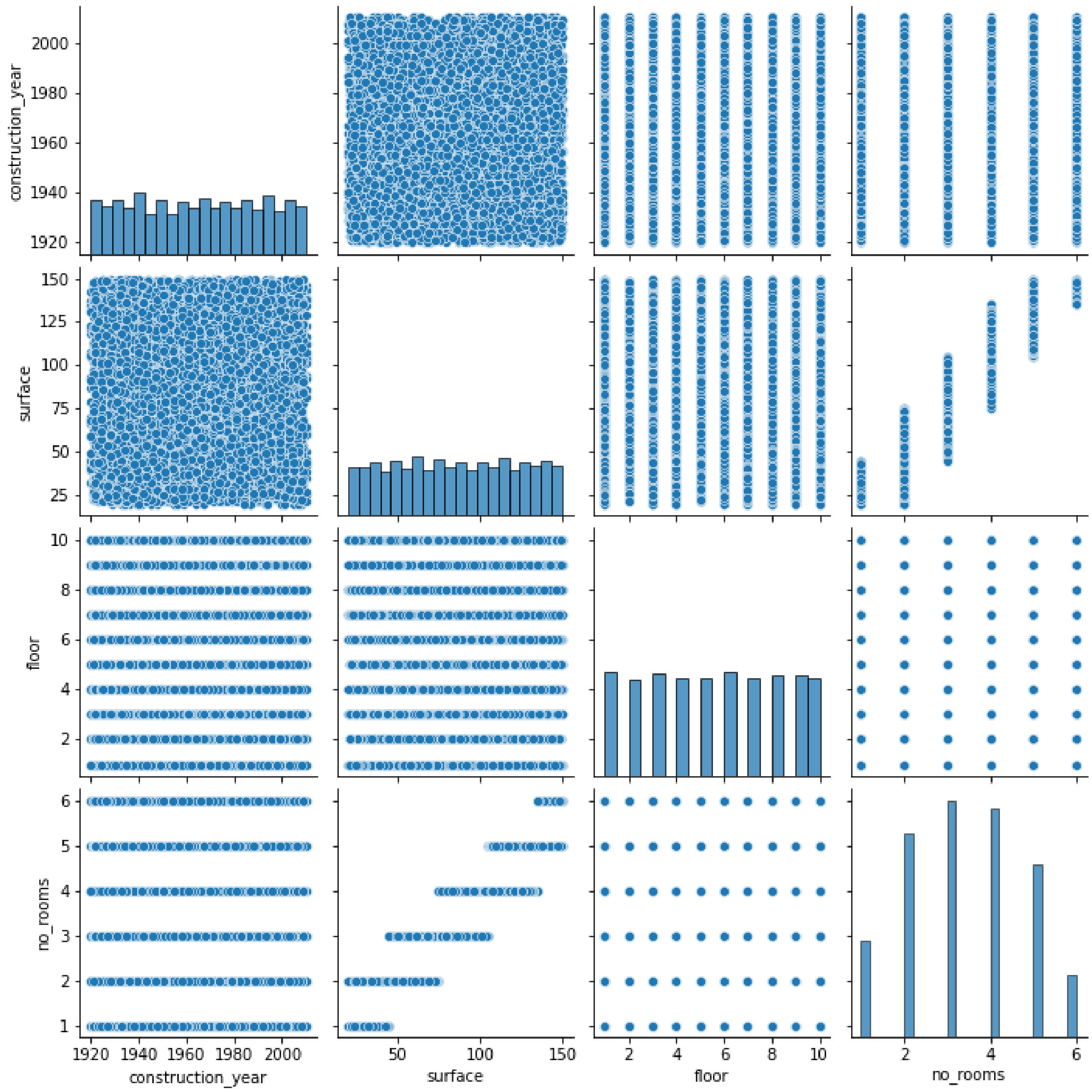
[41]: <AxesSubplot:xlabel='m2_price', ylabel='Count'>



Zmienna celu ma lekko prawoskośny rozkład, ale nie powinno być to problemem w zadaniu regresji.

[50]:
```
sns.pairplot(ap_X_train)
```

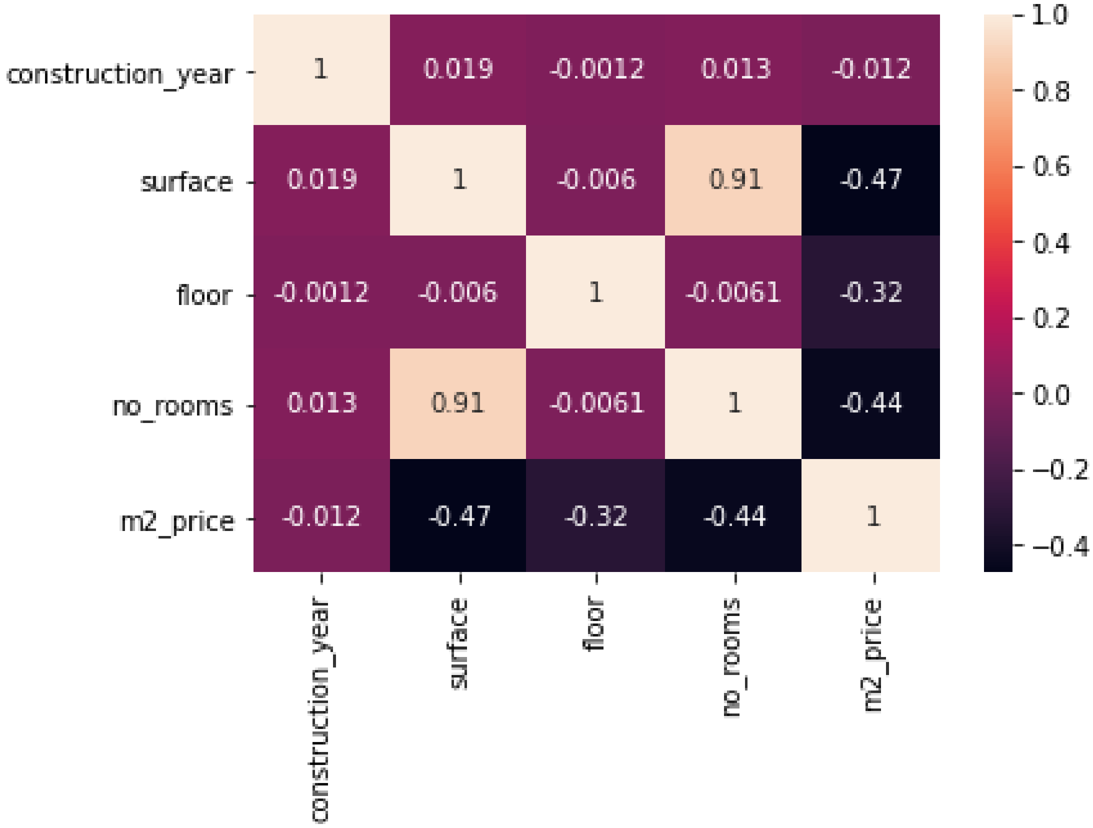
[50]: <seaborn.axisgrid.PairGrid at 0x7f37e6301d30>



Wszystkie zmienne mają dosyć równomierny rozkład, liczba pokoji jest skorelowana z powierzchnią

[53]:
```
ap_train = pd.concat([ap_X_train, ap_y_train], axis=1)
sns.heatmap(ap_train.corr(), annot=True)
```

[53]: <AxesSubplot:>



Usuniemy jedną ze skorelowanych zmiennych

[57]:
```
ap_X_test.drop(['no_rooms'], axis=1, inplace=True)
ap_X_train.drop(['no_rooms'], axis=1, inplace=True)
```

1.2 Mobile price classification

Źródło: https://www.kaggle.com/iabhishekofficial/mobile-price-classification.

Zbiór danych o modelach telefonów komórkowych z etykietą kategorii cenowej. Należy przewidzieć do jakiej kategorii należy telefon.

[93]:
```
mobiles_train = pd.read_csv('data/train.csv')
mobiles_test = pd.read_csv('data/test.csv')
```

[94]:
```
mobiles_test.columns
```

[94]:
```
Index(['id', 'battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc',
       'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc',
       'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi'],
```

```
      dtype='object')
```

[95]:
```
mobiles_train.columns
```

[95]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

W zbiorze danych `test` brakuje zmiennej celu, więc jest dla nas bezużyteczny. Podzielimy zbiór train na treningowy i testowy.

[96]:
```
mobiles = pd.read_csv('data/train.csv')
mob_X = mobiles.drop(['price_range'], axis=1)
mob_y = mobiles.loc[:,'price_range']
```

[98]:
```
mob_X_train, mob_X_test, mob_y_train, mob_y_test = train_test_split(mob_X,␣
↪mob_y, random_state=123, stratify=mob_y)
```

1.2.1 EDA

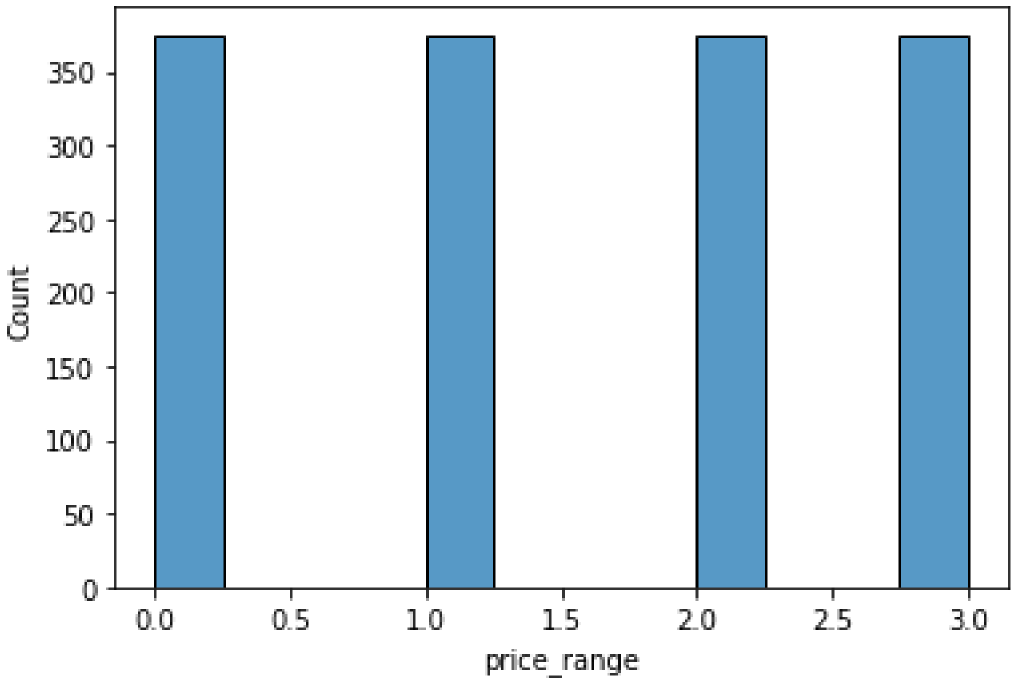
[99]:
```
mob_X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1500 entries, 1336 to 1352
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  1500 non-null   int64
 1   blue           1500 non-null   int64
 2   clock_speed    1500 non-null   float64
 3   dual_sim       1500 non-null   int64
 4   fc             1500 non-null   int64
 5   four_g         1500 non-null   int64
 6   int_memory     1500 non-null   int64
 7   m_dep          1500 non-null   float64
 8   mobile_wt      1500 non-null   int64
 9   n_cores        1500 non-null   int64
 10  pc             1500 non-null   int64
 11  px_height      1500 non-null   int64
 12  px_width       1500 non-null   int64
 13  ram            1500 non-null   int64
 14  sc_h           1500 non-null   int64
 15  sc_w           1500 non-null   int64
 16  talk_time      1500 non-null   int64
 17  three_g        1500 non-null   int64
```

 18 touch_screen   1500 non-null   int64
 19 wifi   1500 non-null   int64
dtypes: float64(2), int64(18)
memory usage: 246.1 KB

[100]:
```
sns.histplot(mob_y_train)
```

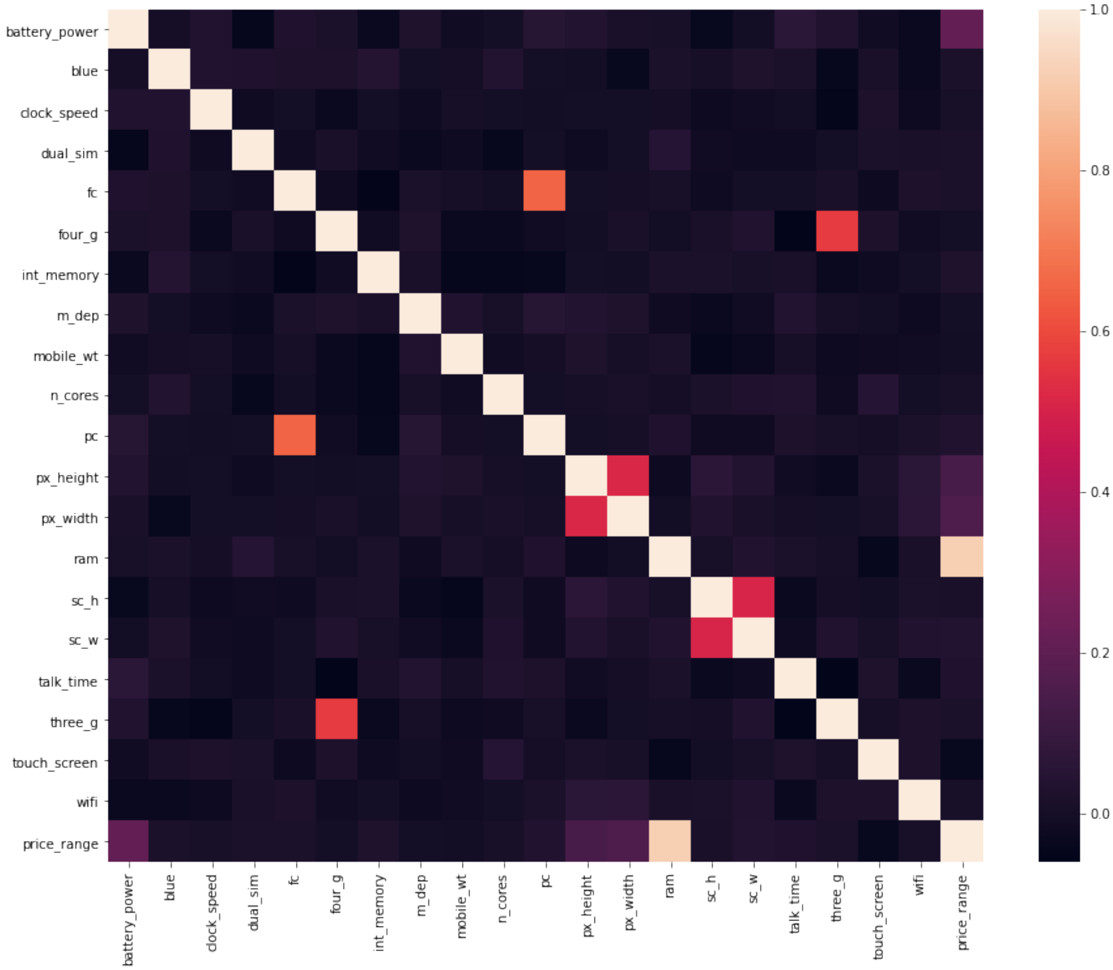
[100]: <AxesSubplot:xlabel='price_range', ylabel='Count'>



[101]:
```
mob_train = pd.concat([mob_X_train, mob_y_train], axis=1)
fig, ax = plt.subplots(figsize=(15,12))

sns.heatmap(ax=ax, data=mob_train.corr())
```

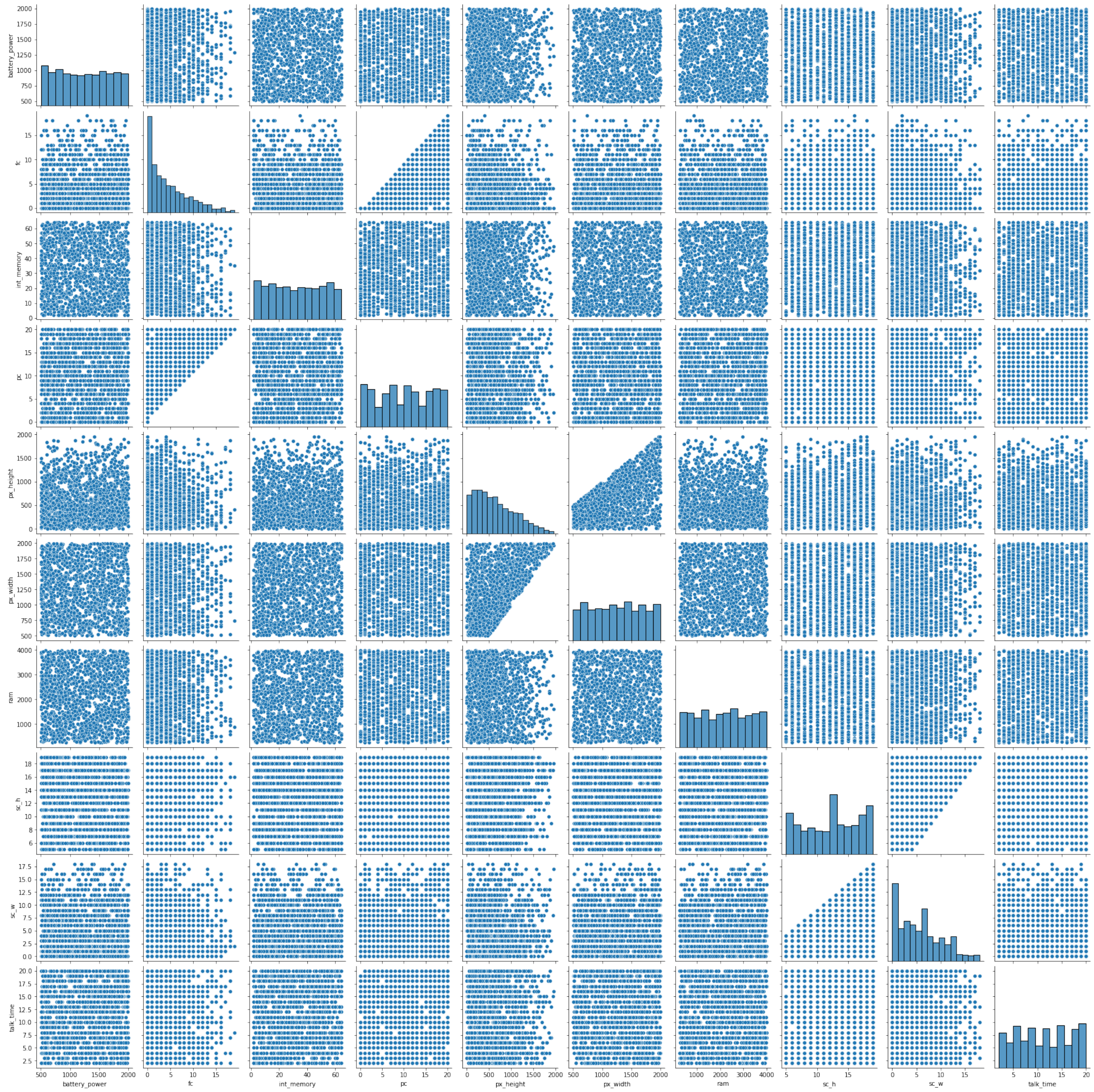
[101]: <AxesSubplot:>



- pc skorelowany z fc (pc - primary camera, fc - front camera). Rozdzielczość aparatów podstawowego i przedniego.
- px_height i px_width skorelowane (szer. i wys. ekranu w pixelach)
- Na cenę ma wpływ:
  - pamięć RAM
  - pojemność baterii
  - rozdzielczość ekranu

[102]:
```
cols = ['battery_power', 'fc', 'int_memory', 'pc', 'px_height', 'px_width',␣
↪'ram', 'sc_h', 'sc_w', 'talk_time']
sns.pairplot(mob_X_train.loc[:,cols])
```

[102]: <seaborn.axisgrid.PairGrid at 0x7f37c6d2d2b0>



Usuniemy z modelu skorelowane zmienne

[103]:
```
mob_X_test = mob_X_test.drop(['fc', 'px_height', 'sc_h'], axis=1)
mob_X_train = mob_X_train.drop(['fc', 'px_height', 'sc_h'], axis=1)
```

1.3 Tworzenie modeli

1.3.1 Apartments

[121]:
```
ap_scaling_clf = make_pipeline(OneHotEncoder(sparse=False), StandardScaler(), SVR())
ap_clf = make_pipeline(OneHotEncoder(), SVR())
```

```
ap_scaling_clf.fit(ap_X_train, ap_y_train)
ap_clf.fit(ap_X_train, ap_y_train)
```

[121]: Pipeline(steps=[('onehotencoder', OneHotEncoder()), ('svr', SVR())])

[122]:
```
display(mean_squared_error(ap_y_test, ap_scaling_clf.predict(ap_X_test)))
display(mean_squared_error(ap_y_test, ap_clf.predict(ap_X_test)))
```

```
836907.1480178521
```

```
787026.2580606752
```

SVM Regressor osiągnął mniejszy MSE, gdy nie skalował danych. Zaprzecza to tezie postawionej w artykule.

1.3.2 Mobiles

[124]:
```
mob_scaling_clf = make_pipeline(StandardScaler(), SVC())
mob_clf = SVC()

mob_scaling_clf.fit(mob_X_train, mob_y_train)
mob_clf.fit(mob_X_train, mob_y_train)
```

[124]: SVC()

[127]:
```
display(accuracy_score(mob_y_test, mob_scaling_clf.predict(mob_X_test)))
display(accuracy_score(mob_y_test, mob_clf.predict(mob_X_test)))
```

```
0.836
```

```
0.884
```

Podobnie jest w tym przypadku. Lepszy wynik uzyskuje klasyfikator bez skalowania.

1.4 Tuning hiperparametrów

1.4.1 Apartments

[132]:
```
ap_scaling_clf.get_params().keys()
```

[132]:
```
dict_keys(['memory', 'steps', 'verbose', 'onehotencoder', 'standardscaler',
'svr', 'onehotencoder__categories', 'onehotencoder__drop',
'onehotencoder__dtype', 'onehotencoder__handle_unknown',
'onehotencoder__sparse', 'standardscaler__copy', 'standardscaler__with_mean',
'standardscaler__with_std', 'svr__C', 'svr__cache_size', 'svr__coef0',
'svr__degree', 'svr__epsilon', 'svr__gamma', 'svr__kernel', 'svr__max_iter',
'svr__shrinking', 'svr__tol', 'svr__verbose'])
```

[150]:
```
distributions = dict(svr__gamma=['scale', 'auto', .05, .1, .25],
                     svr__C=[.01, .05, .1, .25, .5, 2, 5],
```

```
                      svr__kernel=['poly'],
                      svr__degree=[x for x in range(1,6)])
ap_scaling_clf_rs = RandomizedSearchCV(ap_scaling_clf, distributions,
                                       random_state=123,␣
 ↪scoring='neg_mean_squared_error',
                                       n_jobs=-1, verbose=5, n_iter=5)
search = ap_scaling_clf_rs.fit(ap_X_train, ap_y_train)
search.best_params_
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

[150]:
```
{'svr__kernel': 'poly', 'svr__gamma': 0.1, 'svr__degree': 1, 'svr__C': 2}
```

[151]:
```
ap_best_clf = search.best_estimator_
display(mean_squared_error(ap_y_test, ap_best_clf.predict(ap_X_test)))
```

```
3824.8873003131343
```

Uzyskaliśmy dużą poprawę MSE dla SVR (z 836907 do 3824)

1.4.2 Mobiles

[155]:
```
distributions = dict(svc__gamma=['scale', 'auto', .05, .1, .25],
                     svc__C=[.01, .05, .1, .25, .5, 2, 5, 10, 25],
                     svc__kernel=['poly'],
                     svc__degree=[x for x in range(1,6)])
mob_scaling_clf_rs = RandomizedSearchCV(mob_scaling_clf, distributions,
                                        random_state=123, scoring='accuracy',
                                        n_jobs=-1, verbose=5)
mob_search = mob_scaling_clf_rs.fit(mob_X_train, mob_y_train)
mob_search.best_params_
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[155]:
```
{'svc__kernel': 'poly', 'svc__gamma': 0.1, 'svc__degree': 1, 'svc__C': 25}
```

[156]:
```
mob_best_clf = mob_search.best_estimator_
display(accuracy_score(mob_y_test, mob_best_clf.predict(mob_X_test)))
```

```
0.888
```

Poprawiliśmy skuteczność z 83% do prawie 89%

1.5 Wnioski

Zastosowanie tuningu hiperparametrów może znacząco poprawić wynik modelu. RandomizedSearchCV pozwala na szybkie sprawdzenie różnych kombinacji hiperparametrów, które póżniej można udoskonalić poprzez GridSearch