imports

In [1]:
```
# Commons
import os
import pandas as pd
import numpy as np
import random

# Visualization
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from zlib import crc32
import seaborn as sns
import folium
from folium.plugins import MarkerCluster

# Metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.metrics import mean_squared_error

# Models
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from xgboost import XGBRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer

# Ensamble model
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import StackingRegressor

# Exporting
from sklearn2pmml.pipeline import PMMLPipeline
from sklearn_pandas import DataFrameMapper
from sklearn.impute import SimpleImputer
from sklearn2pmml import sklearn2pmml
```

1. Fetch data

In [2]:
```
"""
Fetch .csv data from directories.
"""

csv_path = os.path.join('data')
all_files = os.listdir(csv_path)
csv_files = [f for f in all_files if f.count('rent') == 0 and f.count('.csv'
df_list = []

for csv in csv_files:
    file_path = os.path.join(csv_path, csv)
    try:
        # Try reading the file using default UTF-8 encoding
        df = pd.read_csv(file_path)
        df_list.append(df)
    except Exception as e:
        print(f"Could not read file {csv}, error: {e}")

data = pd.concat(df_list, ignore_index=True)

print(f"Samples amount: {len(data)}")
data.head()
```

```
Samples amount: 88690
```

Out[2]:

	id	city	type	squareMeters	rooms	floor
0	a01d82c9529f98a54d64b9e061c9a73b	szczecin	apartmentBuilding	105.00	4.0	3.0
1	8373aa373dbc3fe7ca3b7434166b8766	szczecin	tenement	73.02	3.0	2.0
2	7d0c31d5409caab173571cce3dcdf702	szczecin	blockOfFlats	68.61	3.0	4.0
3	3eaa36a59b9354206703b5f6b2f2ff1d	szczecin	blockOfFlats	42.00	2.0	1.0
4	027b30cebbc49faf3094421b741ddd56	szczecin	blockOfFlats	45.50	2.0	4.0

5 rows × 28 columns

2. Data cleaning

In [3]:
```
"""
Check data type.
"""

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88690 entries, 0 to 88689
Data columns (total 28 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    88690 non-null  object 
 1   city                  88690 non-null  object 
 2   type                  69759 non-null  object 
 3   squareMeters          88690 non-null  float64
 4   rooms                 88690 non-null  float64
 5   floor                 73118 non-null  float64
 6   floorCount            87695 non-null  float64
 7   buildYear             74554 non-null  float64
 8   latitude              88690 non-null  float64
 9   longitude             88690 non-null  float64
 10  centreDistance        88690 non-null  float64
 11  poiCount              88690 non-null  float64
 12  schoolDistance        88625 non-null  float64
 13  clinicDistance        88406 non-null  float64
 14  postOfficeDistance    88587 non-null  float64
 15  kindergartenDistance  88595 non-null  float64
 16  restaurantDistance    88488 non-null  float64
 17  collegeDistance       86343 non-null  float64
 18  pharmacyDistance      88583 non-null  float64
 19  ownership             88690 non-null  object 
 20  buildingMaterial      53217 non-null  object 
 21  condition             22585 non-null  object 
 22  hasParkingSpace       88690 non-null  object 
 23  hasBalcony            88690 non-null  object 
 24  hasElevator           84317 non-null  object 
 25  hasSecurity           88690 non-null  object 
 26  hasStorageRoom        88690 non-null  object 
 27  price                 88690 non-null  int64  
dtypes: float64(16), int64(1), object(11)
memory usage: 18.9+ MB
```

In [4]:
```
"""
Clean NaN and duplicates. Drop columns with lack of data.
"""

dropped_columns = ['id','type', 'floor', 'buildYear', 'condition', 'building
data.drop(dropped_columns, axis=1, inplace=True)
data.dropna(inplace=True)
data = data.drop_duplicates().reset_index(drop=True)
```

In [5]:
```
"""
Check number of null values.
"""

print(f"Dataset amount: {len(data)}")
data.isna().sum()
```

```
Dataset amount: 48429
```

Out[5]:
```
city                    0
squareMeters            0
rooms                   0
floorCount              0
latitude                0
longitude               0
centreDistance          0
poiCount                0
schoolDistance          0
clinicDistance          0
postOfficeDistance      0
kindergartenDistance    0
restaurantDistance      0
collegeDistance         0
pharmacyDistance        0
ownership               0
hasParkingSpace         0
hasBalcony              0
hasElevator             0
hasSecurity             0
hasStorageRoom          0
price                   0
dtype: int64
```

3. Analize cleared data

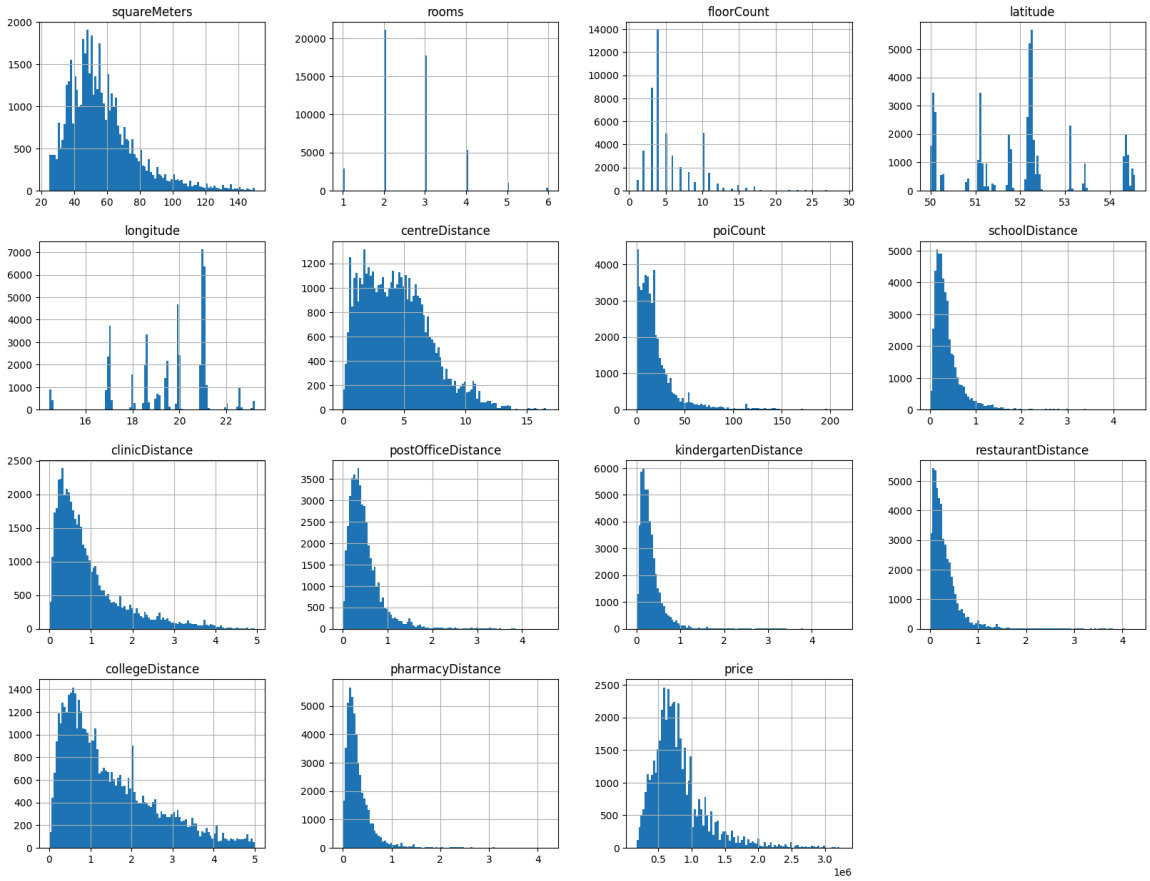
In [6]:
```
"""
Statistics.
"""

data.describe()
```

Out[6]:

	squareMeters	rooms	floorCount	latitude	longitude	centreDistance
count	48429.000000	48429.000000	48429.000000	48429.000000	48429.000000	48429.000000
mean	57.004412	2.618947	5.558281	51.967173	19.513522	4.426425
std	20.478809	0.888424	3.413816	1.292583	1.749187	2.740056
min	25.000000	1.000000	1.000000	49.982200	14.466500	0.010000
25%	43.000000	2.000000	3.000000	51.103234	18.544580	2.190000
50%	53.000000	3.000000	4.000000	52.192840	19.925278	4.140000
75%	66.000000	3.000000	7.000000	52.324180	20.991830	6.150000
max	150.000000	6.000000	29.000000	54.571500	23.203984	16.740000

In [7]:

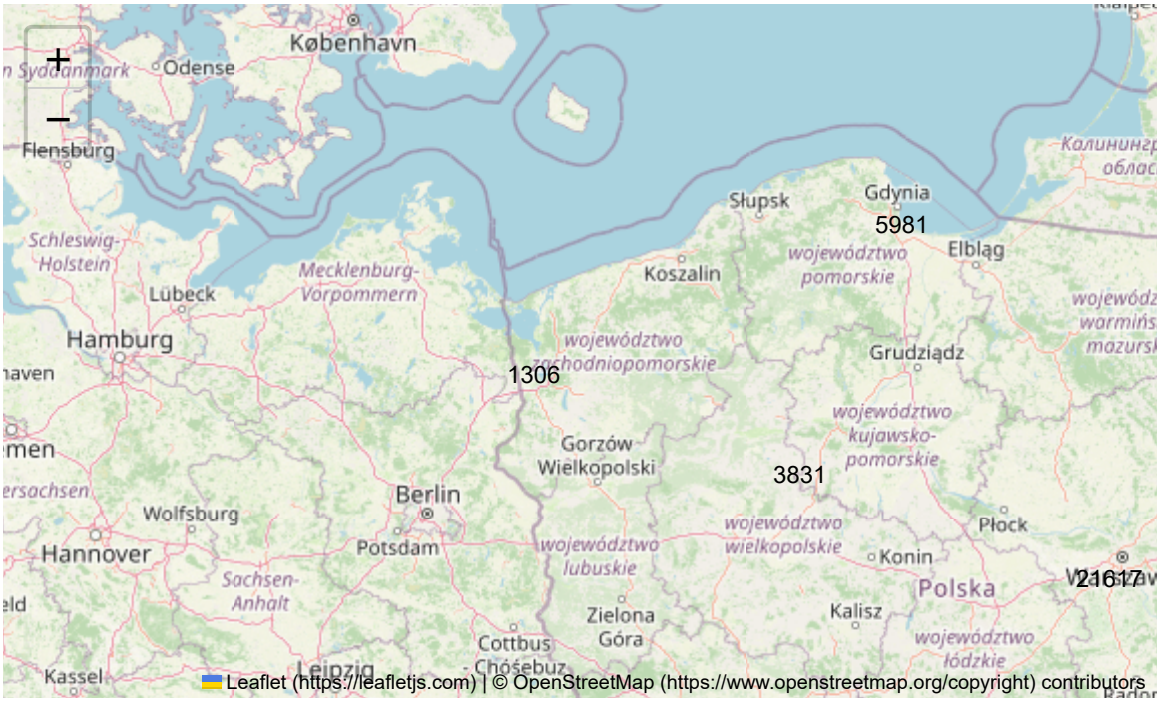
```
"""
Histograms.
"""

%matplotlib inline
data.hist(bins=100, figsize=(20,15))
plt.show()
```



In [8]:

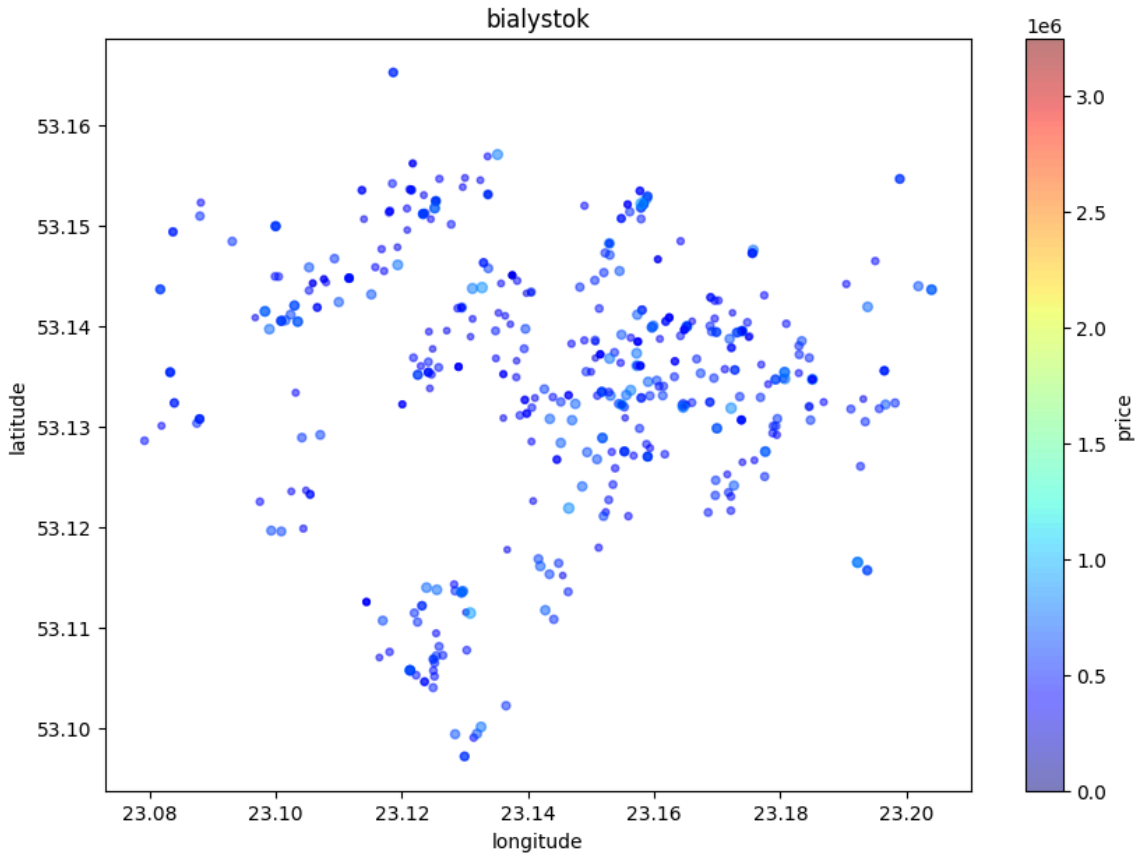
```
"""
Locations on map.
This kind of map is loading quite long, but I think its worth it.
"""

locations = list(zip(data['latitude'], data['longitude']))
m = folium.Map(
    location=[np.mean(data['latitude']), np.mean(data['longitude'])],
    zoom_start=6
)

marker_cluster = MarkerCluster(locations=locations)
marker_cluster.add_to(m)
m
```

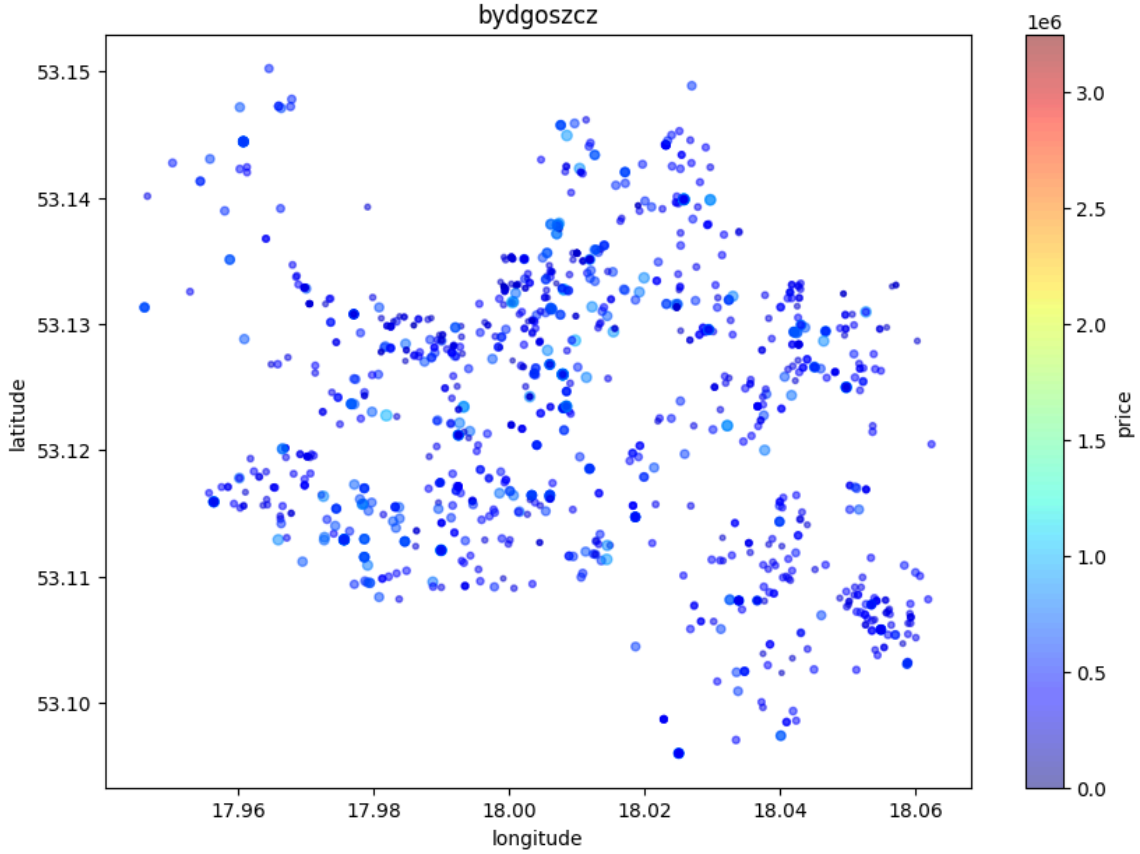
Out[8]:

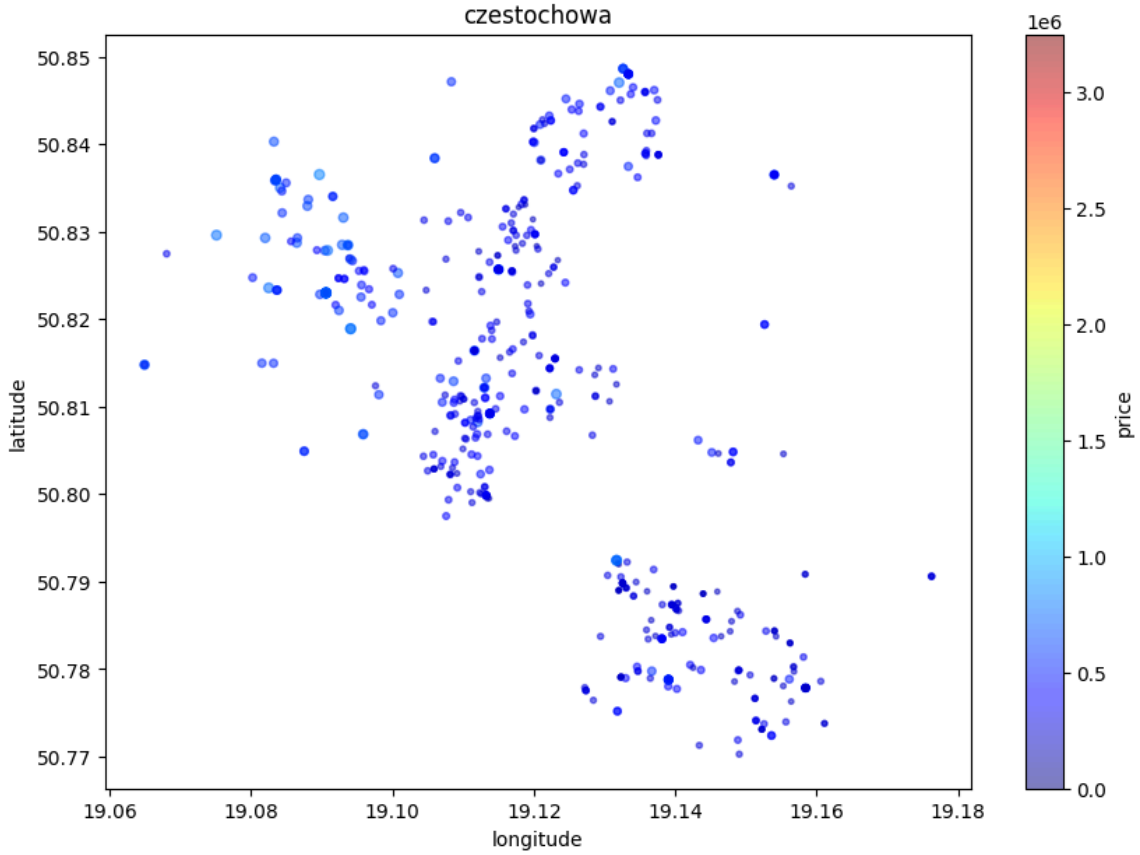


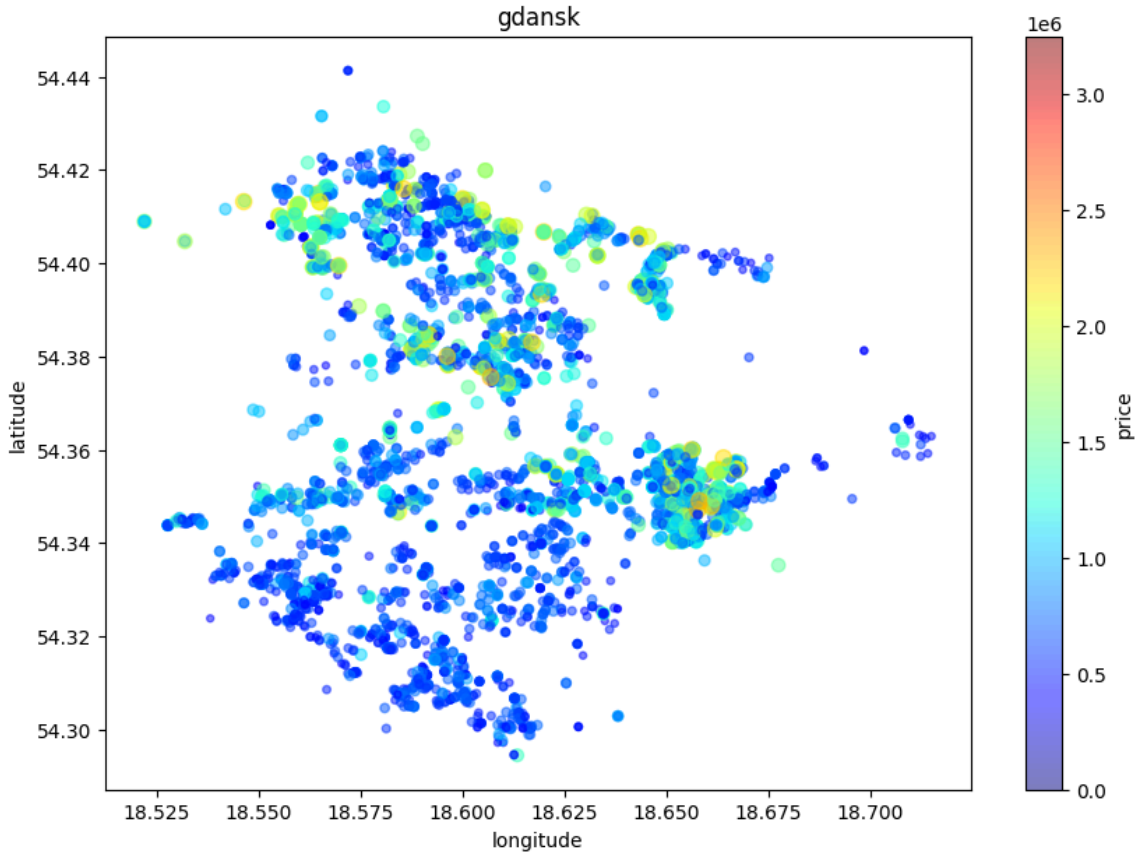
In [9]:

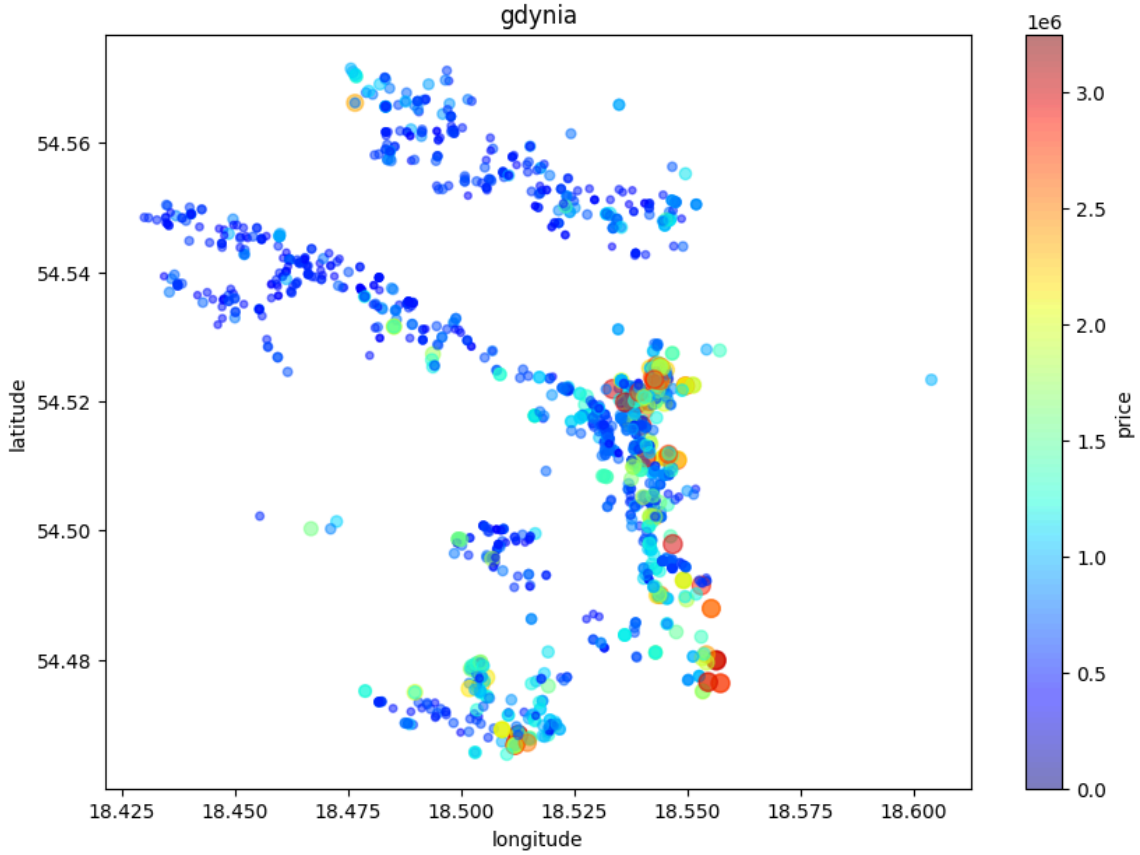
```
"""
Locations based on price.
"""

cities = np.unique(data['city'])
max_price = max(data['price'])
norm = Normalize(vmin=0, vmax=max_price)

for city in cities:
    subset = data[data['city'] == city]
    subset.plot(
        kind='scatter',
        x='longitude',
        y='latitude',
        title=city,
        alpha=0.5,
        s=100 * subset['price'] / max_price,
        figsize=(10, 7),
        c='price',
        norm=norm,
        cmap=plt.get_cmap('jet'),
        colorbar=True
    )
```

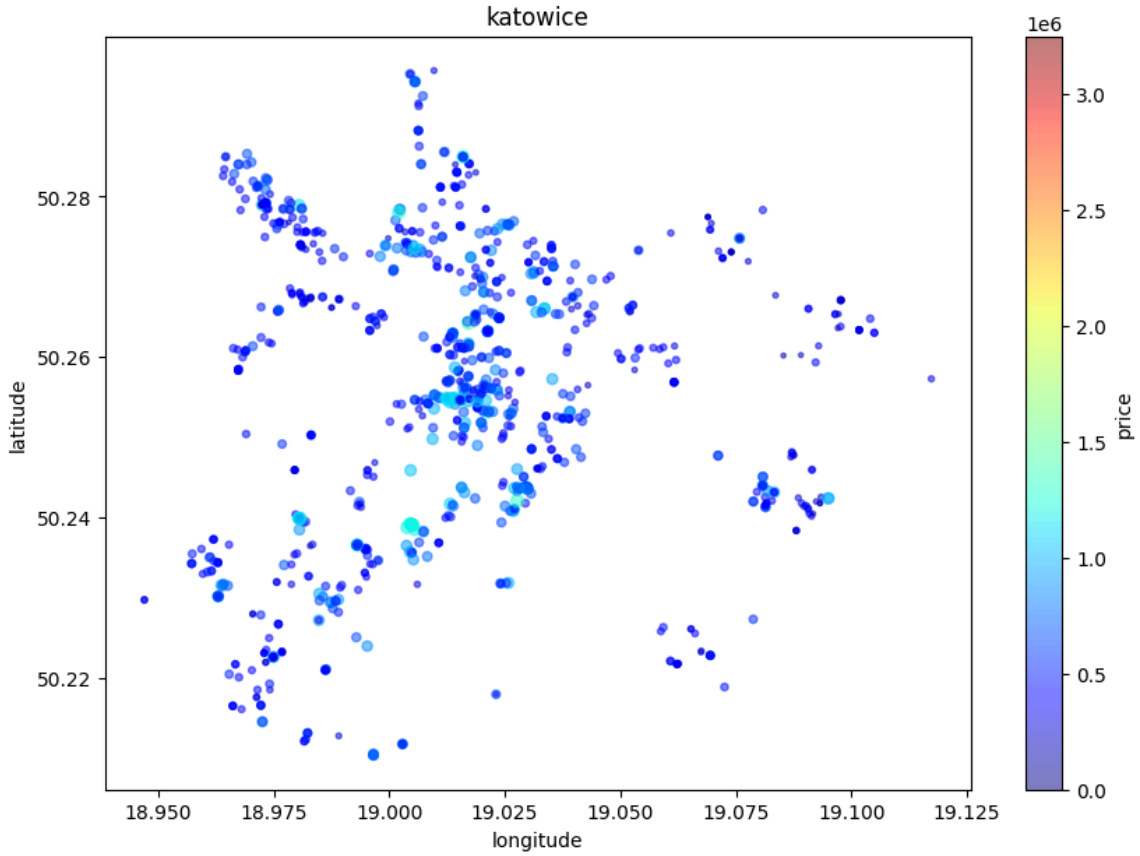


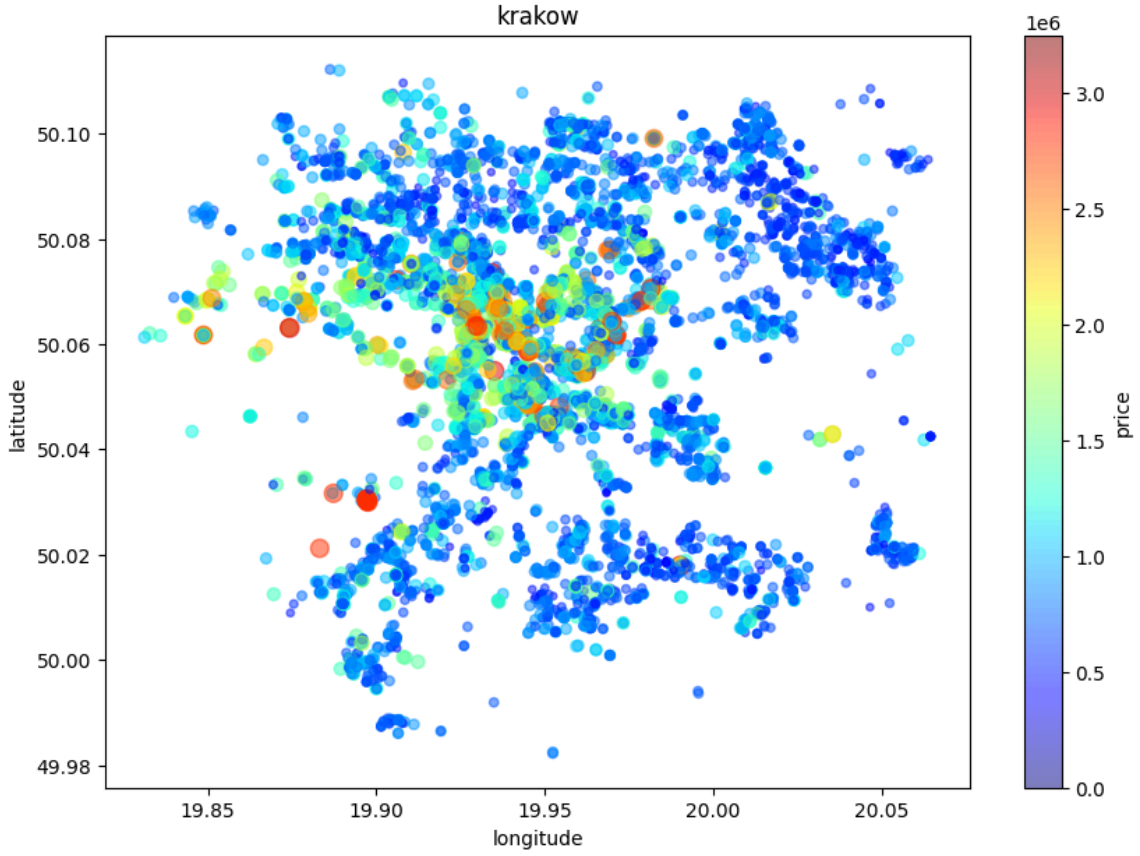

bydgoszcz
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
53.15
53.14
53.13
53.12
53.11
53.10
latitude
17.96
17.98
18.00
18.02
18.04
18.06
longitude

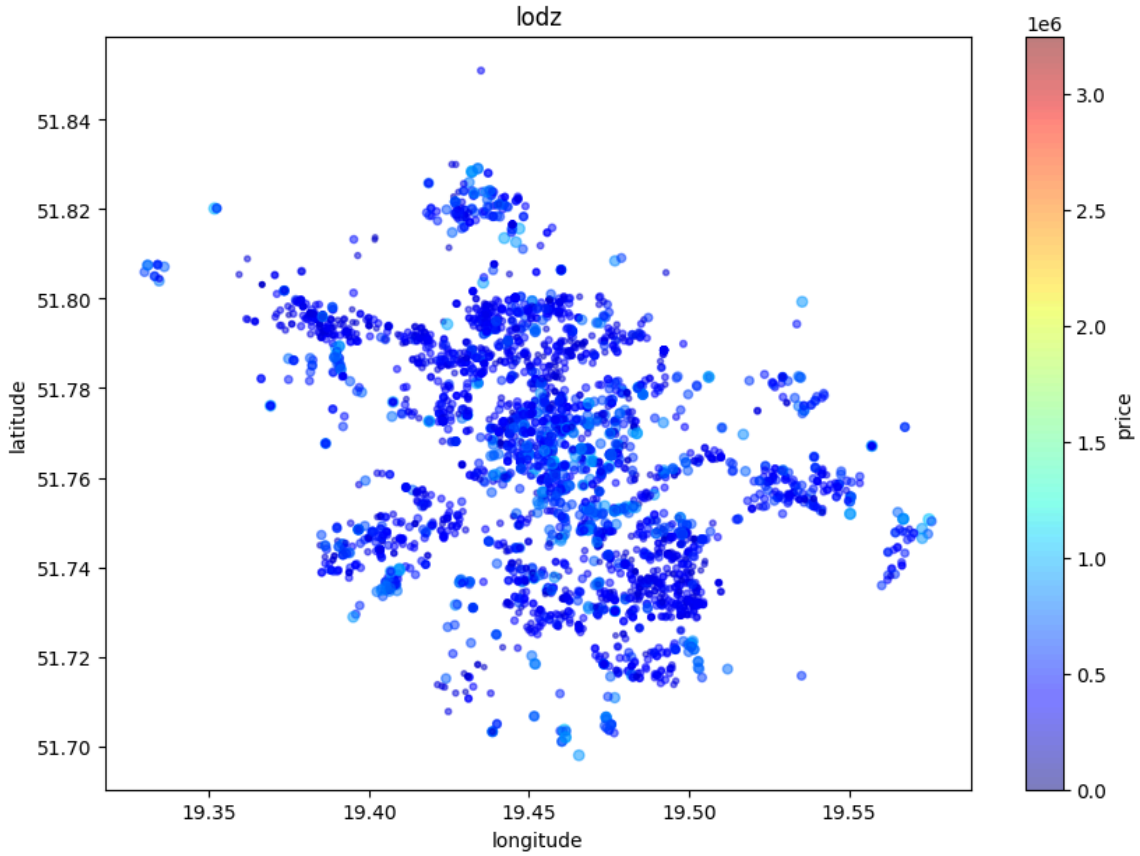

czestochowa
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
50.85
50.84
50.83
50.82
50.81
50.80
50.79
50.78
50.77
latitude
19.06
19.08
19.10
19.12
19.14
19.16
19.18
longitude

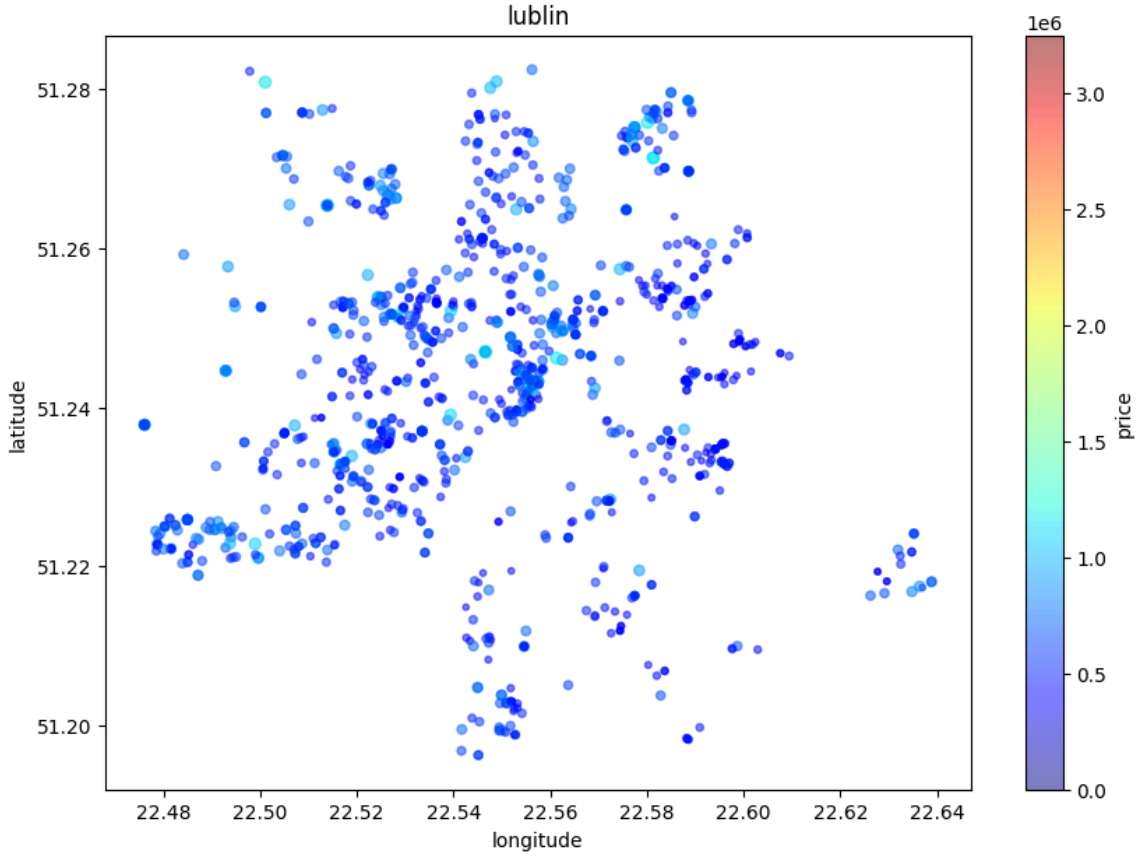

gdansk
54.44
54.42
54.40
54.38
54.36
54.34
54.32
54.30
latitude
18.525
18.550
18.575
18.600
18.625
18.650
18.675
18.700
longitude
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price

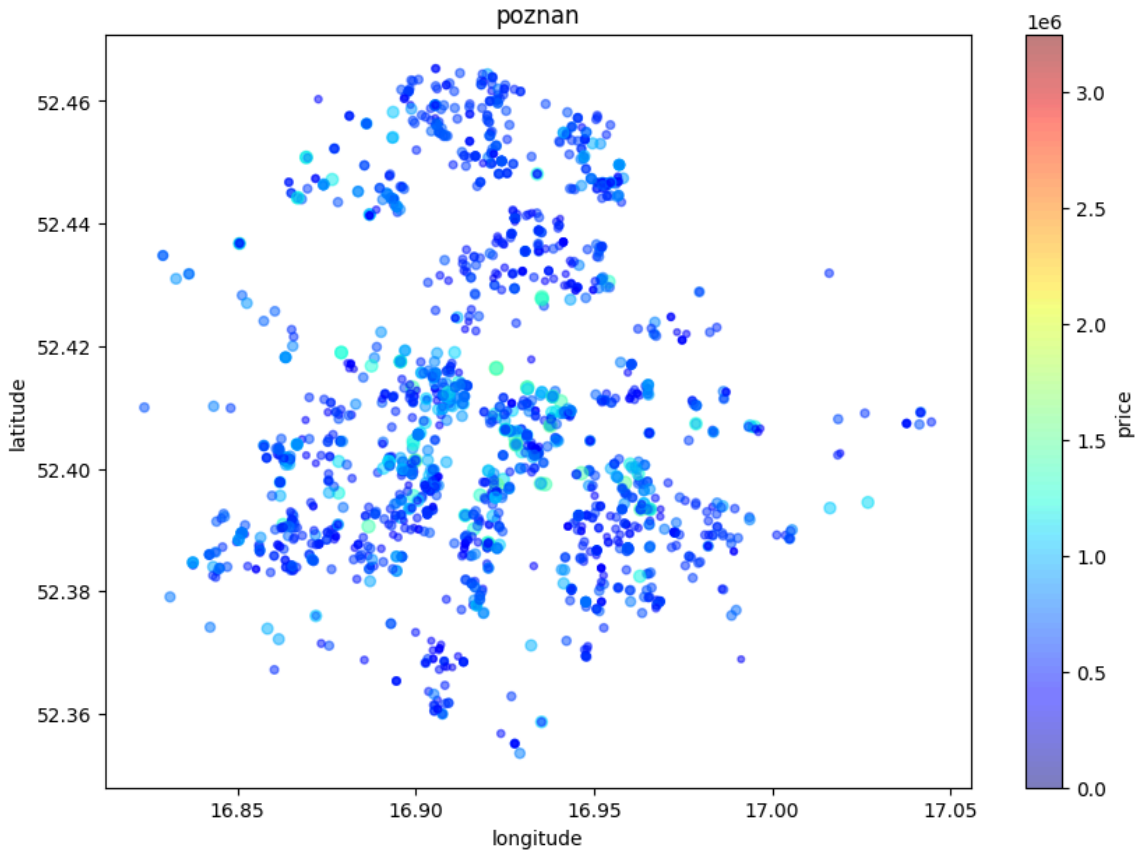

gdynia
54.56
54.54
54.52
54.50
54.48
latitude
18.425
18.450
18.475
18.500
18.525
18.550
18.575
18.600
longitude
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price

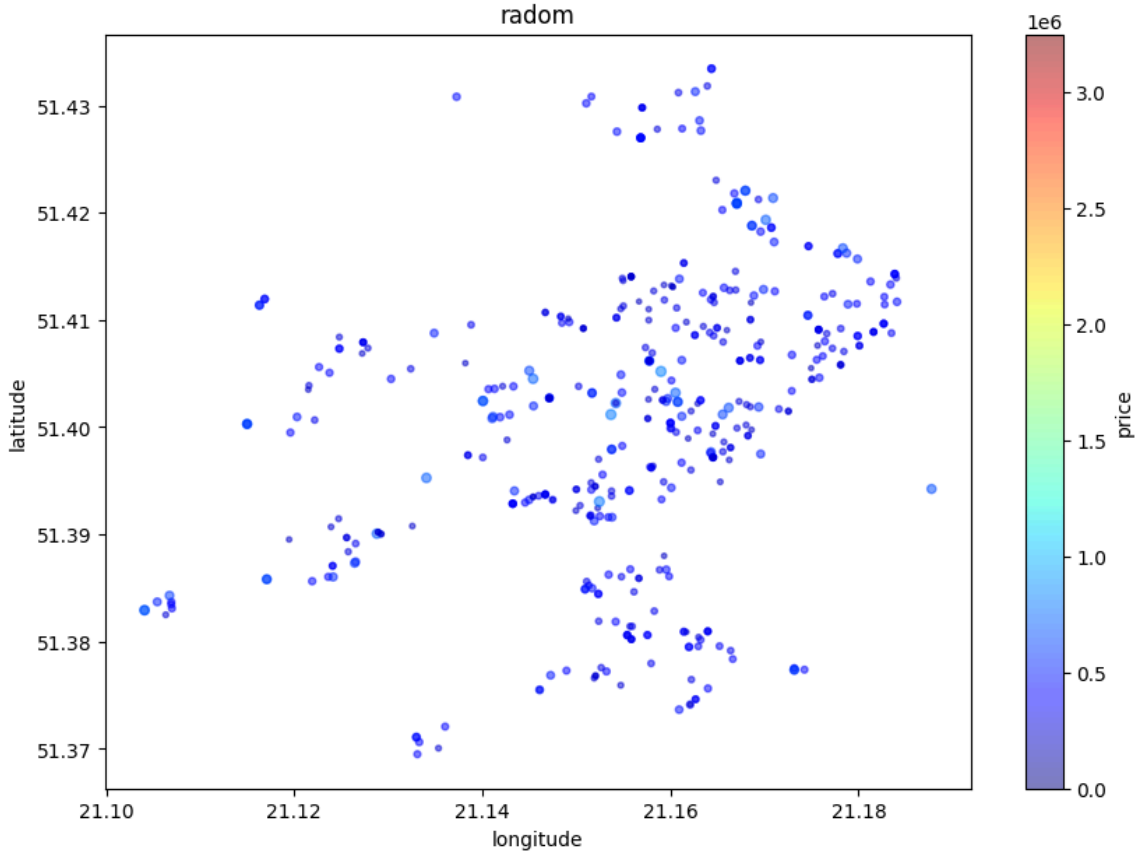

katowice
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
50.28
50.26
50.24
50.22
latitude
18.950
18.975
19.000
19.025
19.050
19.075
19.100
19.125
longitude

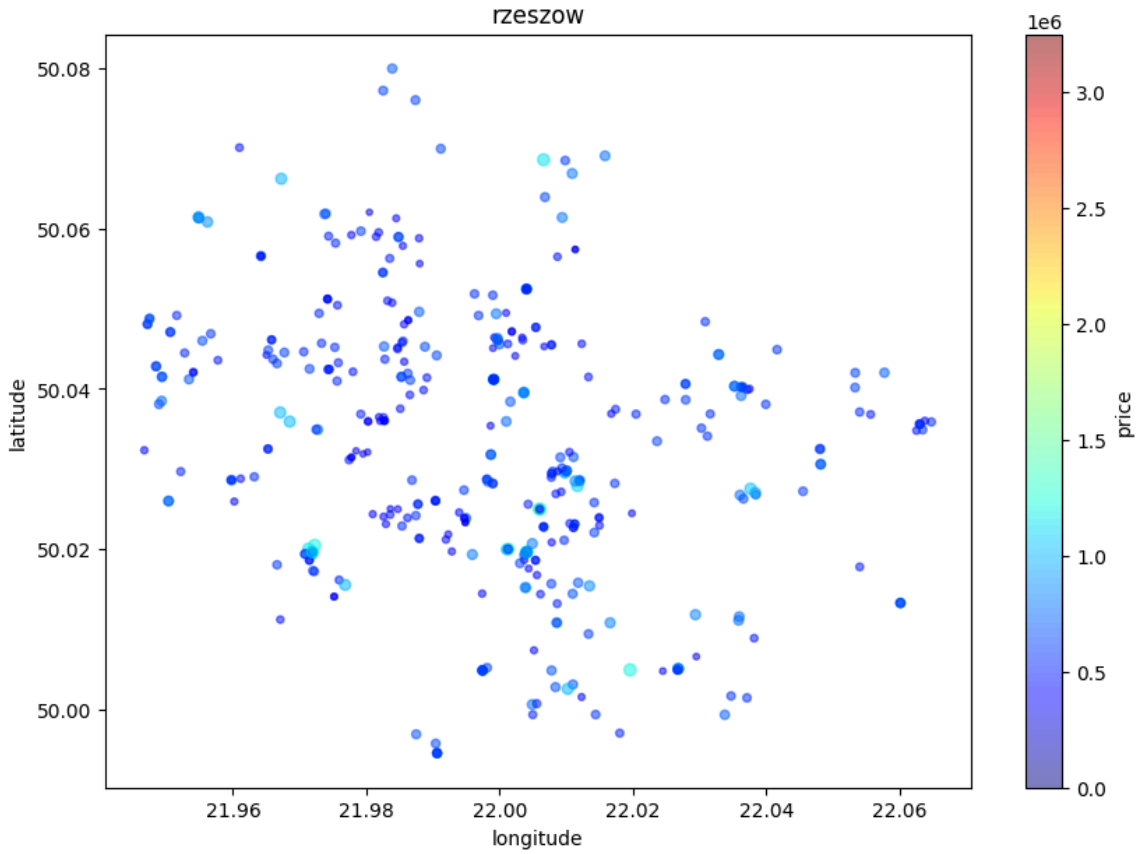

krakow
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
50.10
50.08
50.06
50.04
50.02
50.00
49.98
latitude
19.85
19.90
19.95
20.00
20.05
longitude

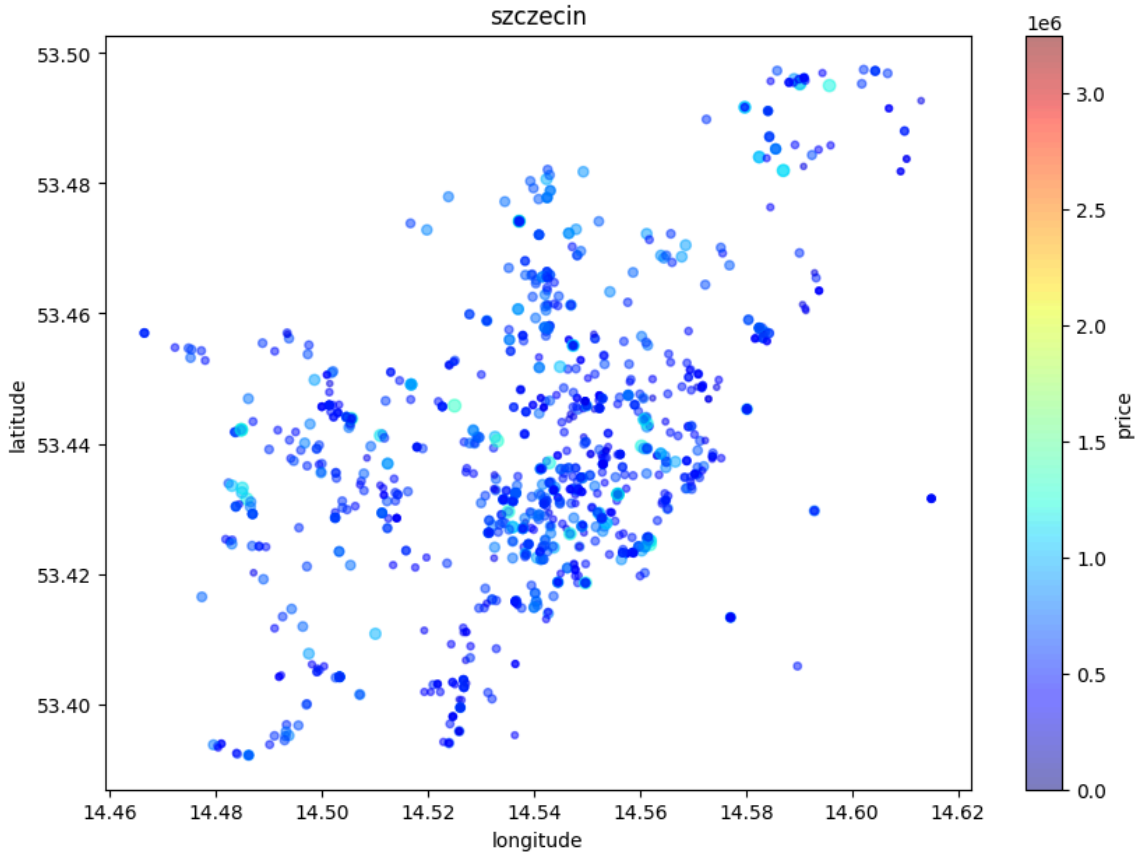

lodz
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
51.84
51.82
51.80
51.78
51.76
51.74
51.72
51.70
latitude
19.35
19.40
19.45
19.50
19.55
longitude

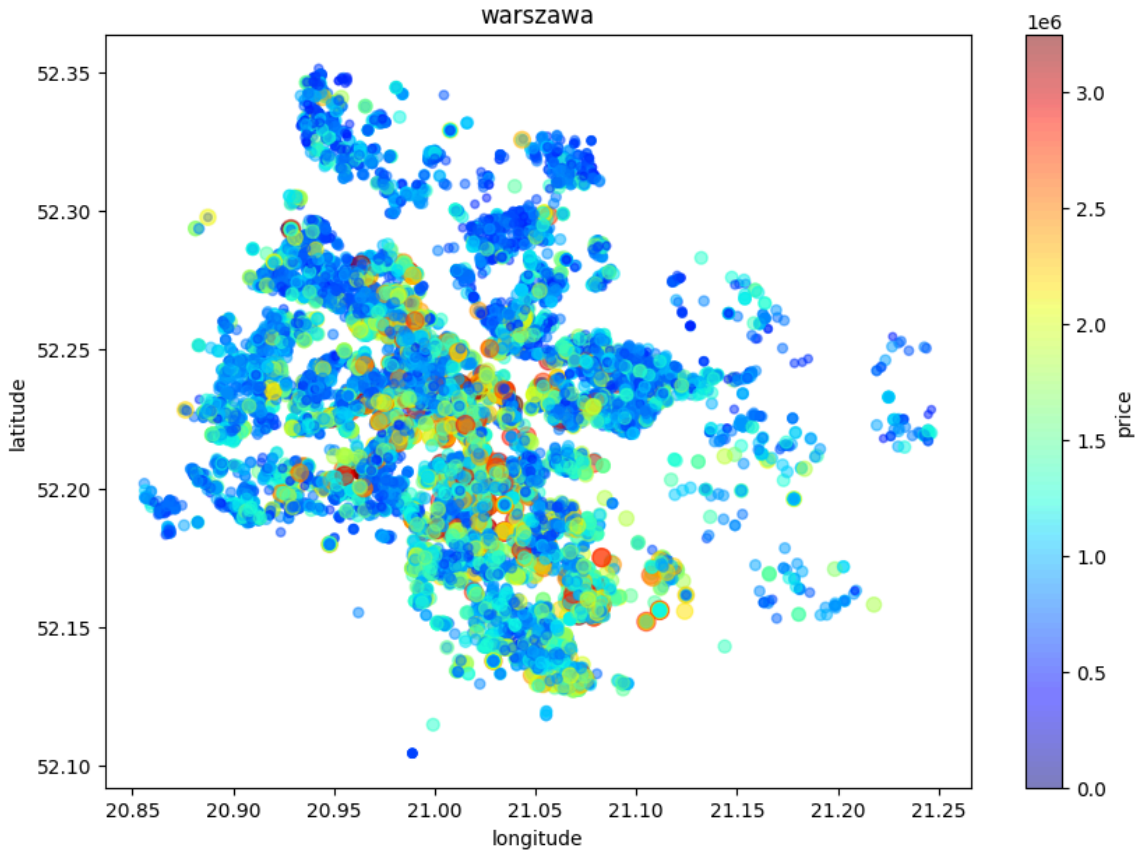

lublin
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
51.28
51.26
51.24
51.22
51.20
latitude
22.48
22.50
22.52
22.54
22.56
22.58
22.60
22.62
22.64
longitude

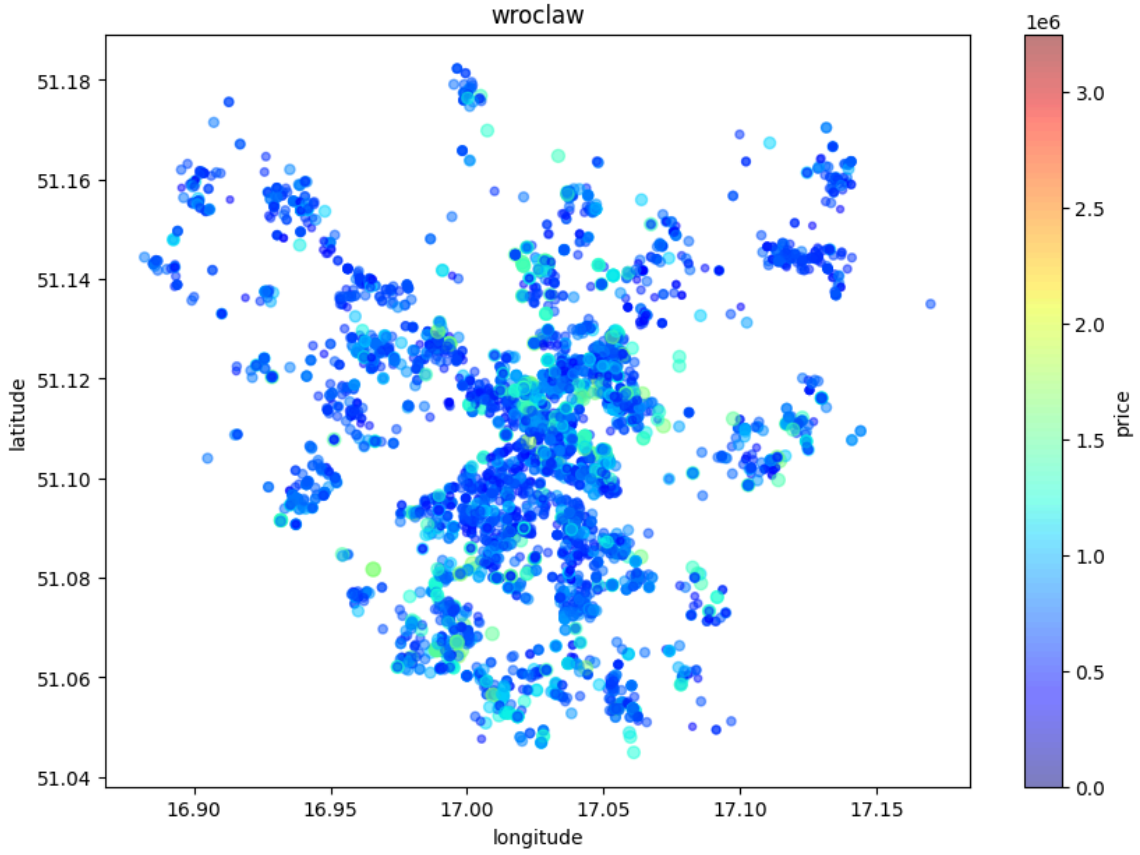

poznan
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
52.46
52.44
52.42
52.40
52.38
52.36
latitude
16.85
16.90
16.95
17.00
17.05
longitude


radom
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
51.43
51.42
51.41
51.40
51.39
51.38
51.37
latitude
21.10
21.12
21.14
21.16
21.18
longitude


rzeszow
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
50.08
50.06
50.04
50.02
50.00
latitude
21.96
21.98
22.00
22.02
22.04
22.06
longitude

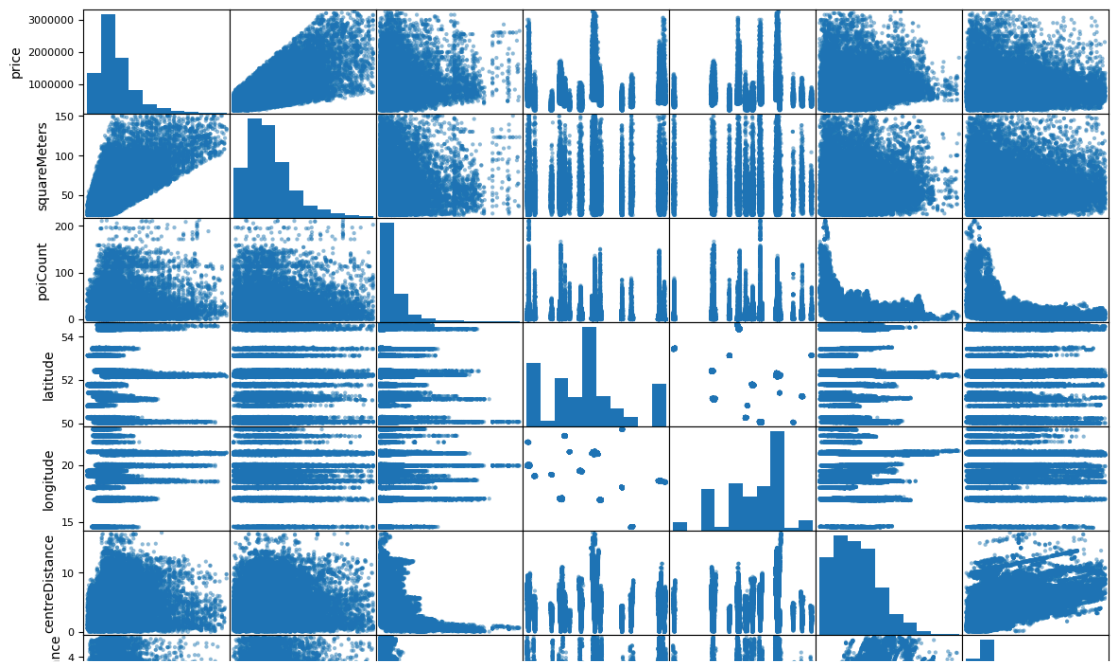

szczecin
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
53.50
53.48
53.46
53.44
53.42
53.40
latitude
14.46
14.48
14.50
14.52
14.54
14.56
14.58
14.60
14.62
longitude


warszawa
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
52.35
52.30
52.25
52.20
52.15
52.10
latitude
20.85
20.90
20.95
21.00
21.05
21.10
21.15
21.20
21.25
longitude


wroclaw
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
51.18
51.16
51.14
51.12
51.10
51.08
51.06
51.04
latitude
16.90
16.95
17.00
17.05
17.10
17.15
longitude

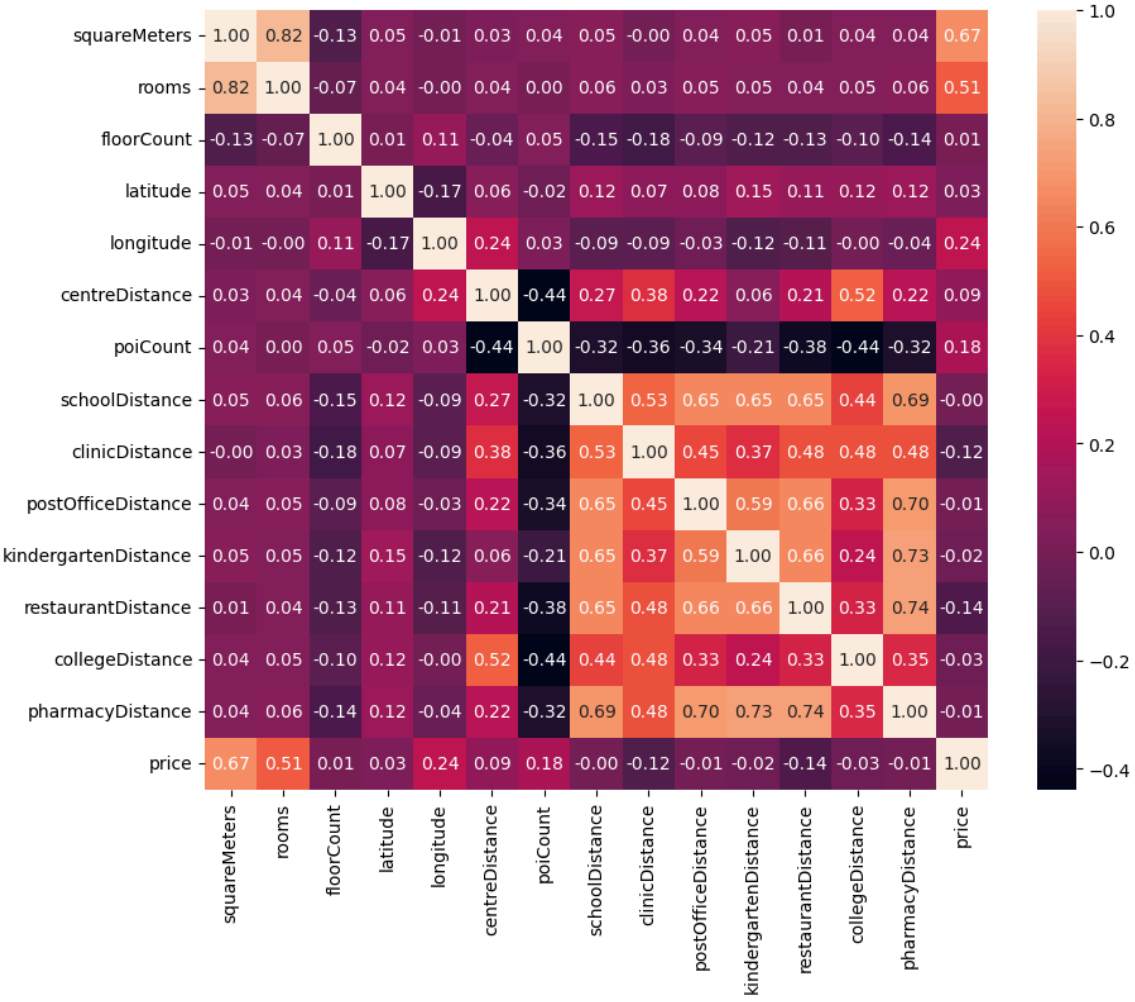
In [10]:

```
"""
Corelation matrix.
"""

%matplotlib inline
from pandas.plotting import scatter_matrix

attributes = ['price', 'squareMeters', 'poiCount', 'latitude', 'longitude',
scatter_matrix(data[attributes], figsize=(14,10))
plt.show()
```



In [11]:
```
"""
Heatmap.
"""

plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(numeric_only=True), annot=True, fmt = '.2f')
```

Out[11]: <AxesSubplot:>



4. Prepare data

In [12]:
```
"""
One-hot coding for cities.
"""

city_dummies = pd.get_dummies(data['city'])
data = data.drop('city', axis=1)
data = pd.concat([data, city_dummies], axis=1)
```

In [13]:
```
"""
Exclude non-numeric columns.
"""
numerical_columns = data.select_dtypes(include=['number']).columns
data = data[numerical_columns]
```

In [14]:
```
"""
Split data to X and y.
"""

test_ratio = 0.2 # 20% of amount is test data
y = data['price']
X = data.drop(columns=['price'])
rand_val = random.randint(0, 111)
min_y = min(y)
max_y = max(y)
```

In [15]:
```
"""
Choose interesting columns.
"""


"""
15 features
"""
X_15f = X.copy()


"""
7 features
"""
columns_7f = ['squareMeters', 'rooms', 'floorCount', 'latitude', 'longitude'
       'centreDistance', 'bialystok', 'bydgoszcz', 'czestochowa', 'gdansk',
       'gdynia', 'katowice', 'krakow', 'lodz', 'lublin', 'poznan', 'radom',
       'rzeszow', 'szczecin', 'warszawa', 'wroclaw']
X_7f = X[columns_7f].copy()


"""
4 features
"""
columns_4f = ['squareMeters', 'latitude', 'longitude', 'bialystok', 'bydgosz
       'czestochowa', 'gdansk', 'gdynia', 'katowice', 'krakow', 'lodz',
       'lublin', 'poznan', 'radom', 'rzeszow', 'szczecin', 'warszawa',
       'wroclaw']
X_4f = X[columns_4f].copy()
```

In [16]:
```
"""
Check columns and data example.
"""

X_15f.head()
```

Out[16]:

	squareMeters	rooms	floorCount	latitude	longitude	centreDistance	poiCount	schoolD
0	105.00	4.0	4.0	53.431503	14.485820	5.06	1.0	
1	73.02	3.0	3.0	53.452222	14.553333	3.24	9.0	
2	68.61	3.0	4.0	53.456213	14.583222	3.94	7.0	
3	42.00	2.0	3.0	53.468056	14.538333	5.19	9.0	
4	45.50	2.0	4.0	53.438165	14.563200	1.65	18.0	

5 rows × 29 columns

5. Metrics functions

We measure the models by coefficient of determination (`R^2`):
https://en.wikipedia.org/wiki/Coefficient_of_determination
(https://en.wikipedia.org/wiki/Coefficient_of_determination)

and `RMSLE`: https://www.kaggle.com/code/carlolepelaars/understanding-the-metric-rmsle
(https://www.kaggle.com/code/carlolepelaars/understanding-the-metric-rmsle)

In [17]:

```
def print_coefficients(_y_pred, _y_test):
    print("\nR^2: " + "{:.3f}".format(r2_score(_y_test, _y_pred)))
    print("\nRMSLE: " + "{:.3f}".format(mean_squared_log_error(_y_test, _y_p


def print_diff_plot(_y_pred, _y_test, title):
    y_diff = pd.DataFrame({'Actual':_y_test, 'Predicted':_y_pred})
    sns.set(style='darkgrid')
    fig = plt.figure(figsize=(16,8))
    y_diff = y_diff.reset_index()
    y_diff = y_diff.drop(['index'],axis=1)
    plt.plot(y_diff[:25])
    plt.title(title)
    plt.legend(['Actual','Predicted'])

def print_covergence_plot(_y_pred, _y_test):
    y_diff = pd.DataFrame({'Actual':_y_test, 'Predicted':_y_pred})
    sns.set(style='darkgrid')
    jplot = sns.jointplot(
        xlim=(0, max_y),
        ylim=(0, max_y),
        x='Actual',
        y='Predicted',
        data=y_diff,
        kind='reg',
        color='k'
    )
    sns.scatterplot(
        data=y_diff,
        x='Actual',
        y='Predicted',
        ax=jplot.ax_joint,
    )

def print_accuracy_diagram(_y_pred, _y_test):
    y_pred_bins = pd.qcut(_y_test, q=8, retbins=True)[1]
    y_pred_bins = np.insert(y_pred_bins, 0, 0)
    y_pred_bins = np.append(y_pred_bins, 5 * max_y)
    y_pred_bins = [int(bin) for bin in y_pred_bins]

    y_pred_cut = pd.cut(_y_pred, bins=y_pred_bins, labels=y_pred_bins[:-1])
    y_test_cut = pd.cut(_y_test, bins=y_pred_bins, labels=y_pred_bins[:-1])

    ConfusionMatrixDisplay.from_predictions(
        y_pred_cut,
        y_test_cut,
        cmap='Oranges',
        xticks_rotation='vertical',
        ax=None
    )
```

6. Build and test simple models

In [18]:
```
"""
K-Mean regressor for every attribute (reg_km_15f)
"""
"""
parameters={
    'n_neighbors': 5,
    'weights': 'uniform',
    'algorithm': 'auto',
    'leaf_size': 30,
    'p': 2,
    'metric': 'minkowski',
    'metric_params': None,
    'n_jobs': None,
}

reg_km_15f = KNeighborsRegressor(**parameters)

X_train, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test

reg_km_15f.fit(X_train, y_train)
y_pred = reg_km_15f.predict(X_test)

print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "K-Means Regressor")
print_covergence_plot(y_pred, y_test)
"""
```

Out[18]:
```
'\nparameters={\n    \'n_neighbors\': 5,\n    \'weights\': \'uniform\',\n
\'algorithm\': \'auto\',\n    \'leaf_size\': 30,\n    \'p\': 2,\n    \'met
ric\': \'minkowski\',\n    \'metric_params\': None,\n    \'n_jobs\': Non
e,\n}\n\nreg_km_15f = KNeighborsRegressor(**parameters)\n\nX_train, X_tes
t, y_train, y_test = train_test_split(X_15f, y, test_size=test_ratio, rand
om_state=rand_val)\n\nreg_km_15f.fit(X_train, y_train)\ny_pred = reg_km_15
f.predict(X_test)\n\nprint_coefficients(y_pred, y_test)\nprint_diff_plot(y
_pred, y_test, "K-Means Regressor")\nprint_covergence_plot(y_pred, y_test)
\n'
```

In [19]:
```
"""
Gradient boosting regressor for every attribute (reg_gb_15f)
"""
"""
parameters={
    'loss': 'squared_error',
    'learning_rate': 0.1,
    'n_estimators': 100,
    'subsample': 1.0,
    'criterion': 'friedman_mse',
    'min_samples_split': 2,
    'min_samples_leaf': 1,
    'max_depth': 3,
    'max_features': None,
    'alpha': 0.9
}

reg_gb_15f = GradientBoostingRegressor(**parameters)

X_train, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test

reg_gb_15f.fit(X_train, y_train)
y_pred = reg_gb_15f.predict(X_test)

print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Gradient-Boosting Regressor")
print_covergence_plot(y_pred, y_test)
"""
```

Out[19]:
```
'\nparameters={\n    \'loss\': \'squared_error\',\n    \'learning_rate\':
0.1,\n    \'n_estimators\': 100,\n    \'subsample\': 1.0,\n    \'criterion
\': \'friedman_mse\',\n    \'min_samples_split\': 2,\n    \'min_samples_le
af\': 1,\n    \'max_depth\': 3,\n    \'max_features\': None,\n    \'alpha
\': 0.9\n}\n\nreg_gb_15f = GradientBoostingRegressor(**parameters)\n\nX_tr
ain, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test_r
atio, random_state=rand_val)\n\nreg_gb_15f.fit(X_train, y_train)\ny_pred =
reg_gb_15f.predict(X_test)\n\nprint_coefficients(y_pred, y_test)\nprint_di
ff_plot(y_pred, y_test, "Gradient-Boosting Regressor")\nprint_covergence_p
lot(y_pred, y_test)\n'
```

7. Build and text XGBoosing models.

In [20]:
```
"""
Create cross-validation model for XGB regressor.
"""

"""
Default:
parameters={
    'booster': 'gbtree',
    'learning_rate': 0.1,
    'n_estimators': 100,
    'max_depth': 3,
    'min_child_weight': 1,
    'subsample': 1,
    'colsample_bytree': 1,
    'gamma': 0,
    'reg_alpha': 0,
    'reg_lambda': 1,
    'objective': 'binary:logistic',
    'n_jobs': 1,
    'nthread': None,
    'max_delta_step': 0,
    'colsample_bytree': 1,
    'colsample_bylevel': 1,
    'scale_pos_weight': 1,
    'base_score': 0.5,
    'random_state': 0,
    'seed': None
}

"""
parameters = {
    'learning_rate': [0.05, 0.1, 0.20],
    'n_estimators': [100, 400, 800],
    'max_depth': [3, 6, 9],
    'min_child_weight': [1, 10, 100]
}

xgb_reg = XGBRegressor()
reg = RandomizedSearchCV(
    estimator=xgb_reg,
    param_distributions=parameters,
    cv=5,
    scoring='neg_mean_squared_log_error',
    n_jobs=-1,
    verbose=2
)
```

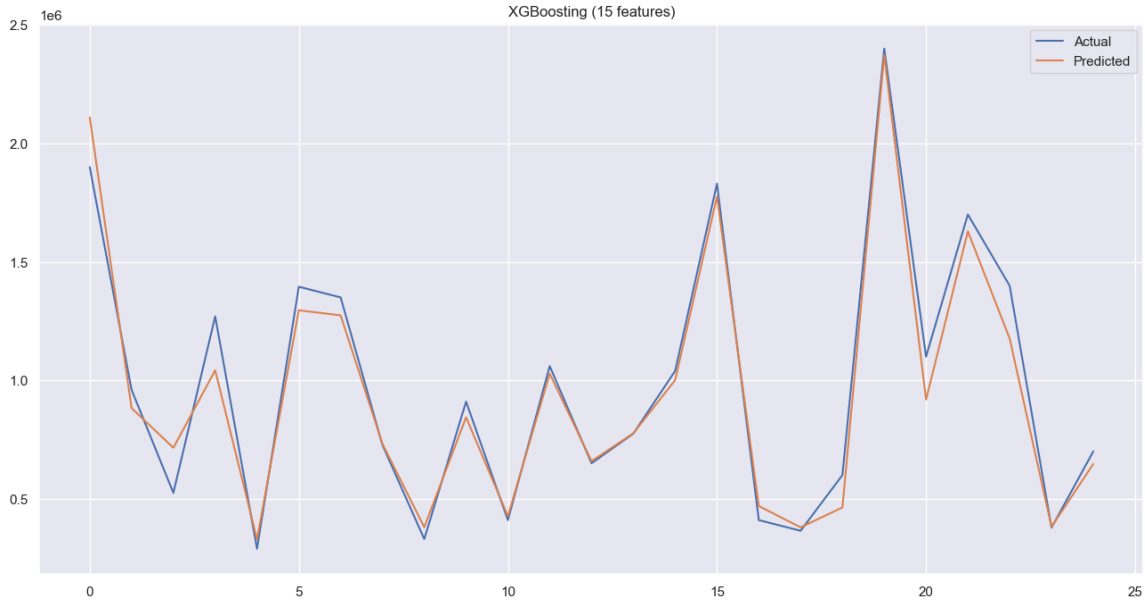
In [21]:

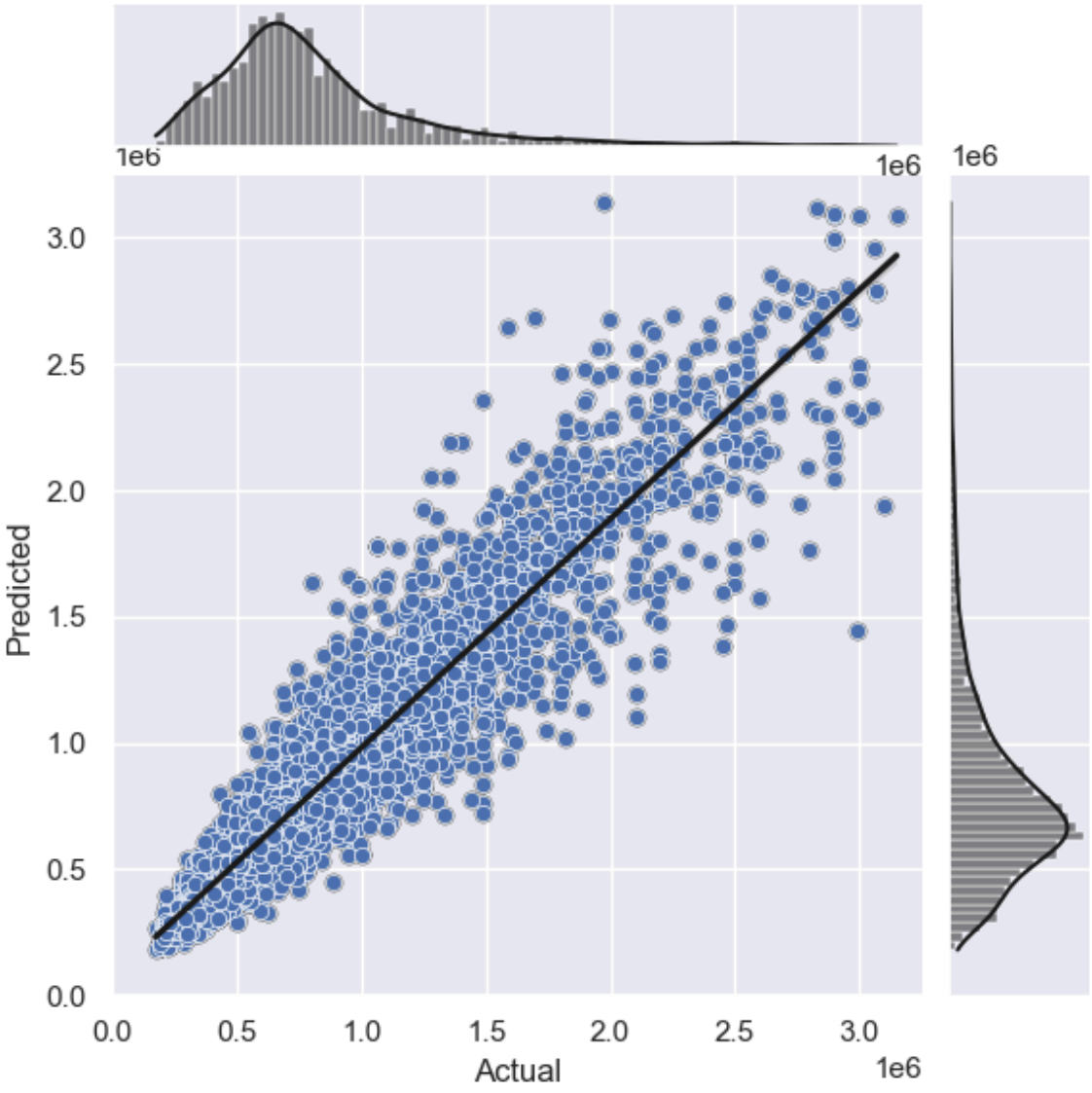
```
"""
Fit XGB regressor for every attribute (reg_xgb_15f)
"""
X_train, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_xgb_15f = XGBRegressor(**reg.best_params_)
reg_xgb_15f.fit(X_train, y_train)
y_pred = reg_xgb_15f.predict(X_test)
```

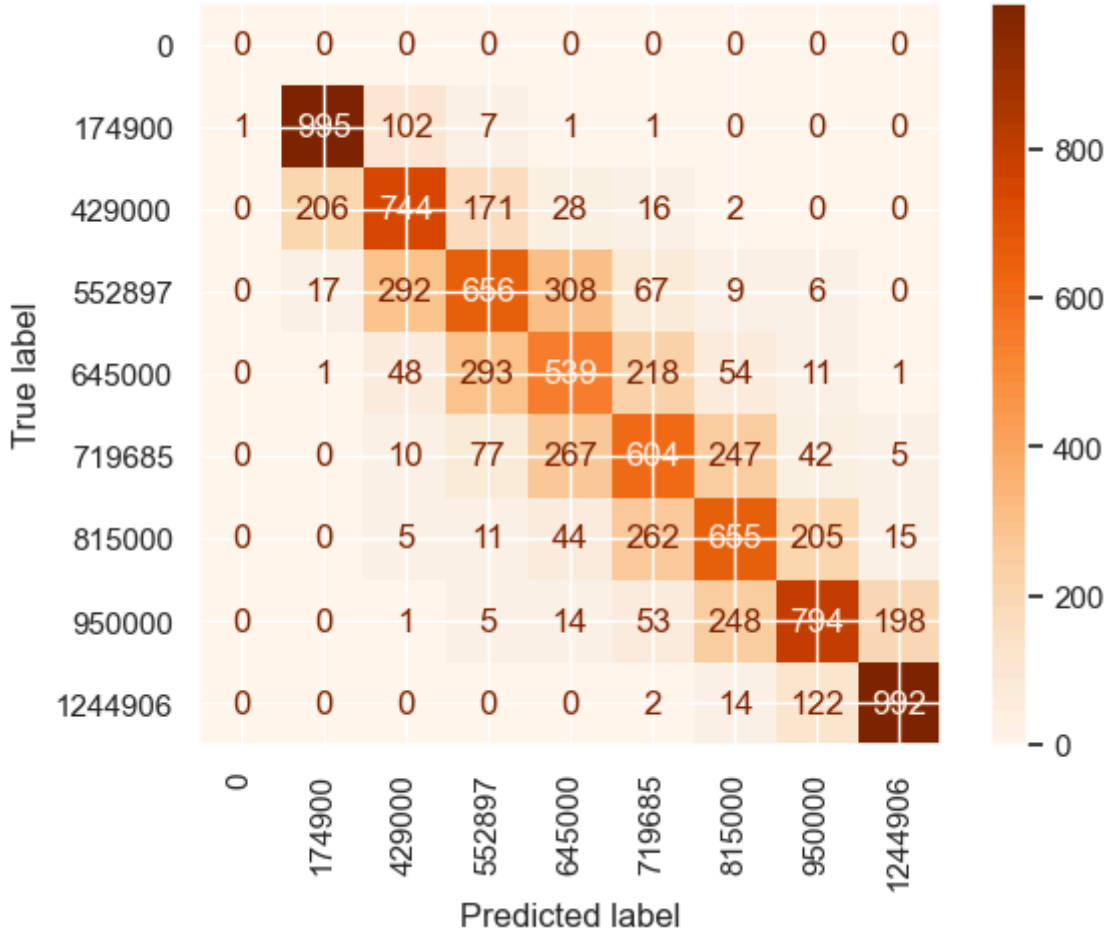
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 800, 'min_child_weight': 1, 'max_de
pth': 9, 'learning_rate': 0.1}
```

In [22]:
```
"""
Test reg_xgb_15f
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "XGBoosting (15 features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.903

RMSLE: 0.018
```




1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
Actual

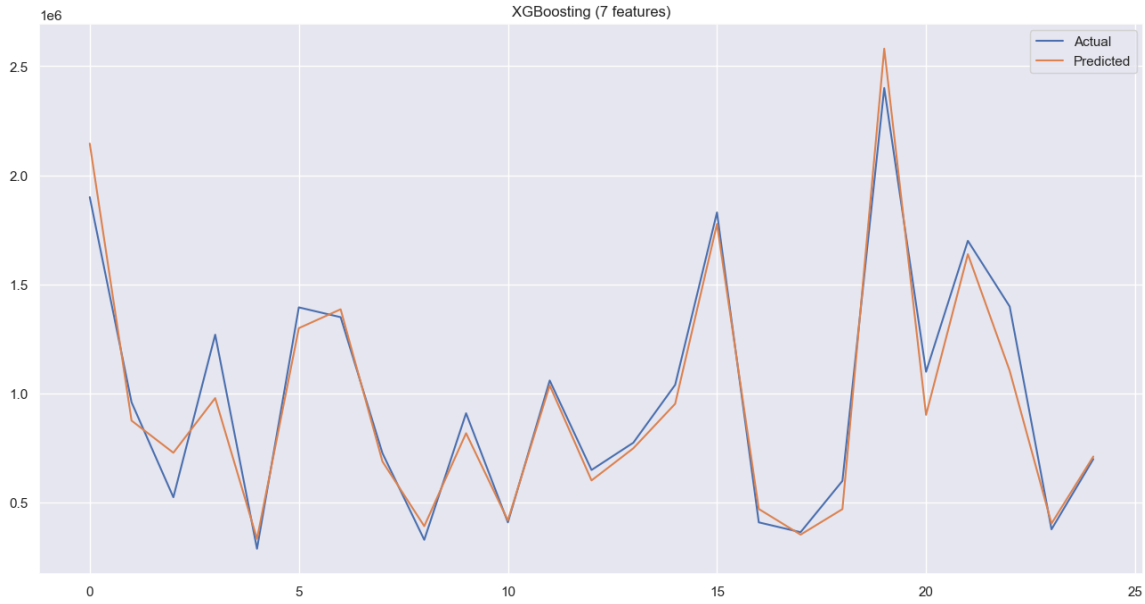


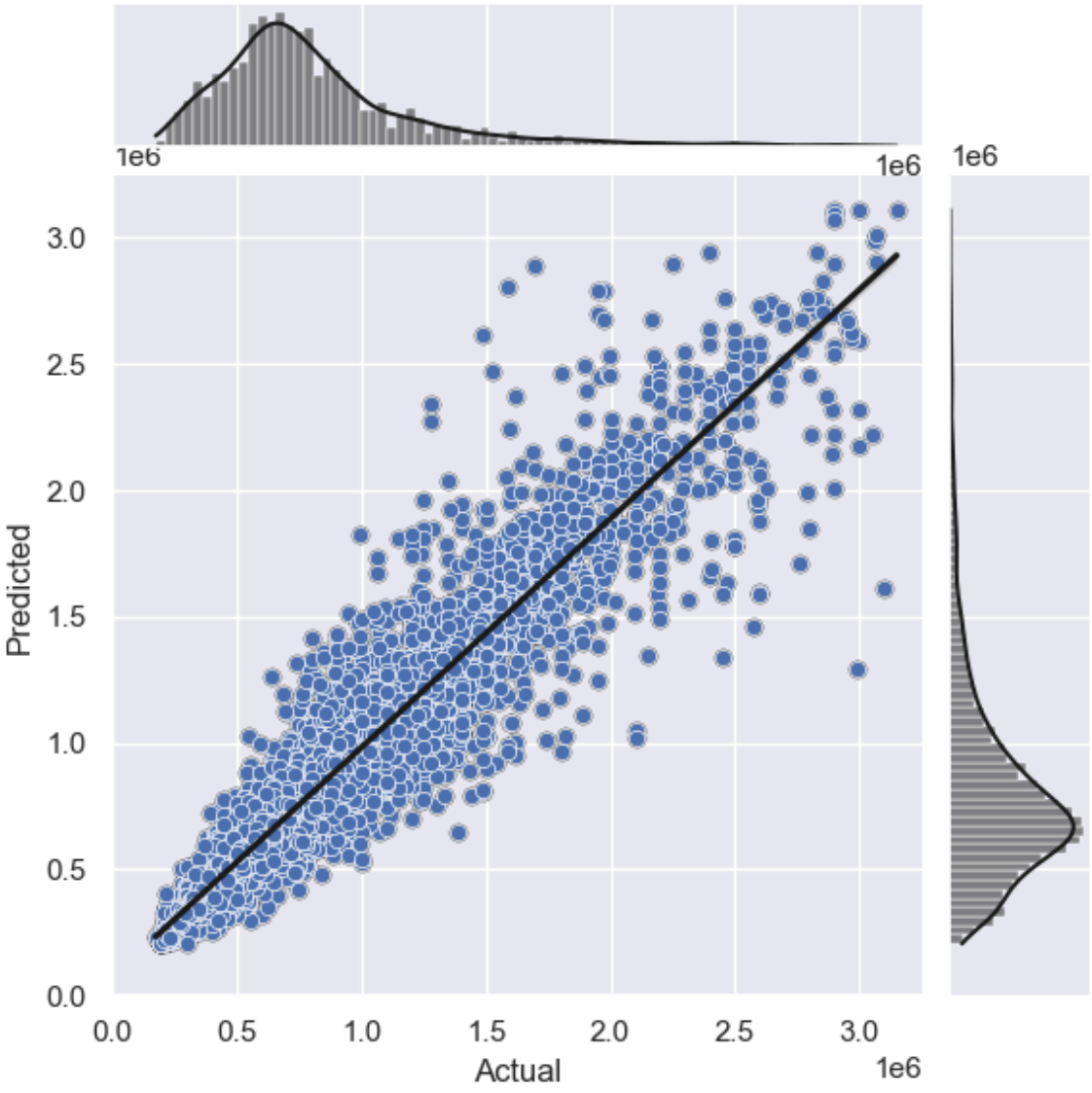
In [23]:
```
"""
Fit XGB regressor for every attribute (reg_xgb_7f)
"""
X_train, X_test, y_train, y_test = train_test_split(X_7f, y, test_size=test_

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_xgb_7f = XGBRegressor(**reg.best_params_)
reg_xgb_7f.fit(X_train, y_train)
y_pred = reg_xgb_7f.predict(X_test)
```

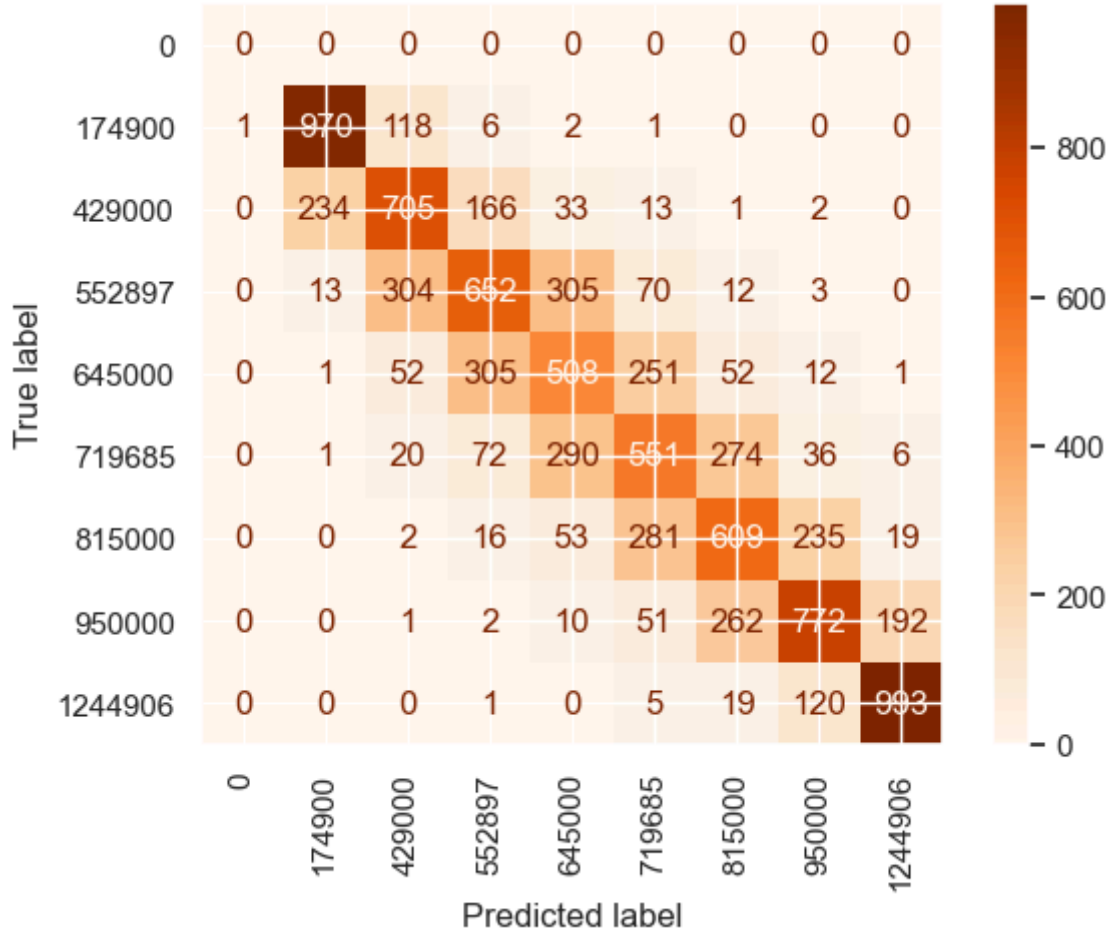
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 800, 'min_child_weight': 1, 'max_de
pth': 9, 'learning_rate': 0.05}
```

In [24]:
```
"""
Test reg_xgb_7f
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "XGBoosting (7 features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.900

RMSLE: 0.019
```




1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6

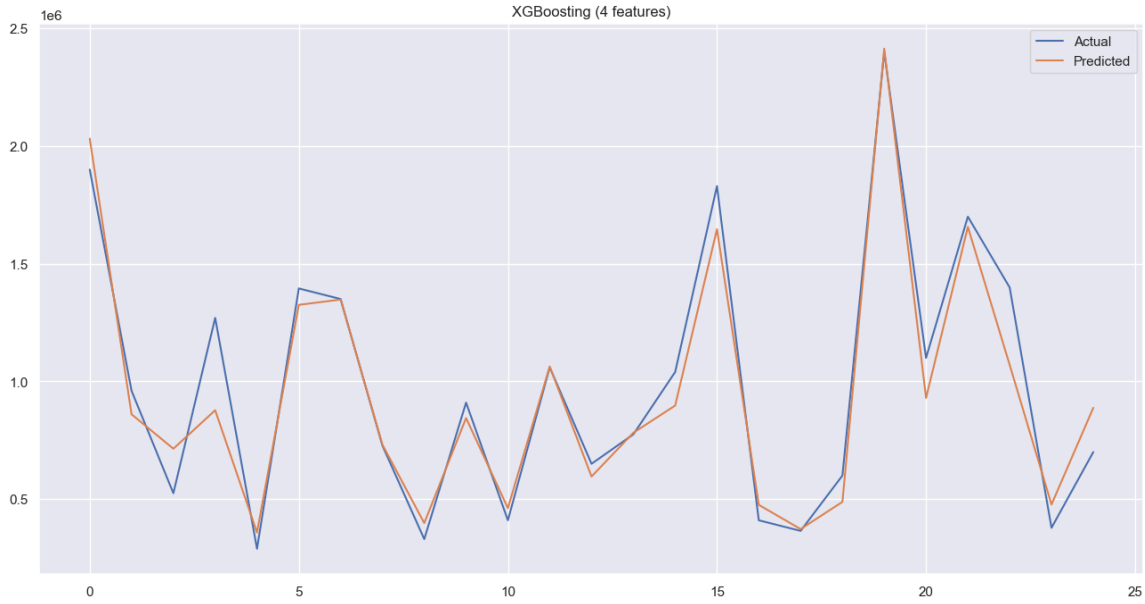


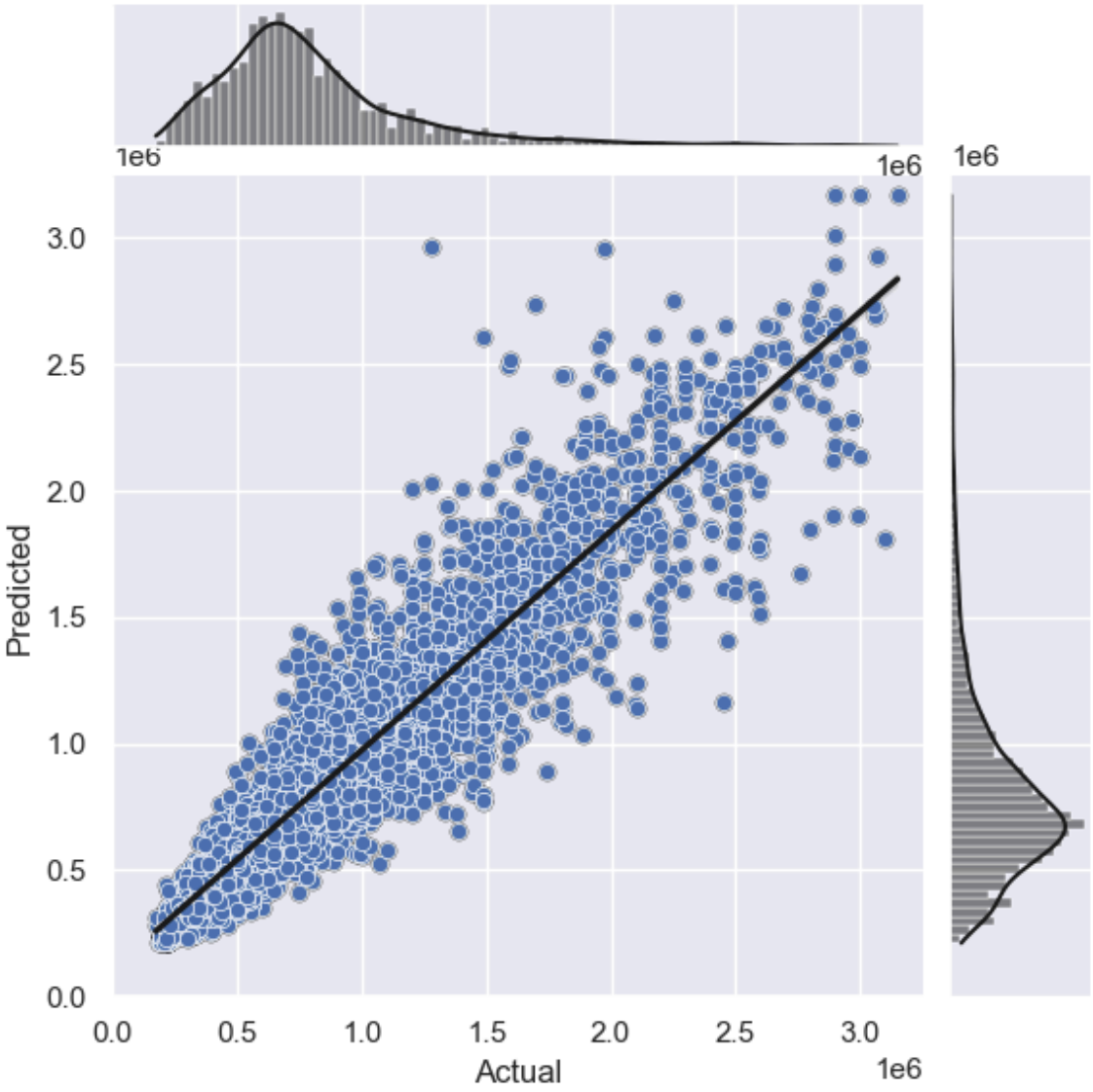
In [25]:
```
"""
Fit XGB regressor for every attribute (reg_xgb_4f)
"""
X_train, X_test, y_train, y_test = train_test_split(X_4f, y, test_size=test_

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_xgb_4f = XGBRegressor(**reg.best_params_)
reg_xgb_4f.fit(X_train, y_train)
y_pred = reg_xgb_4f.predict(X_test)
```

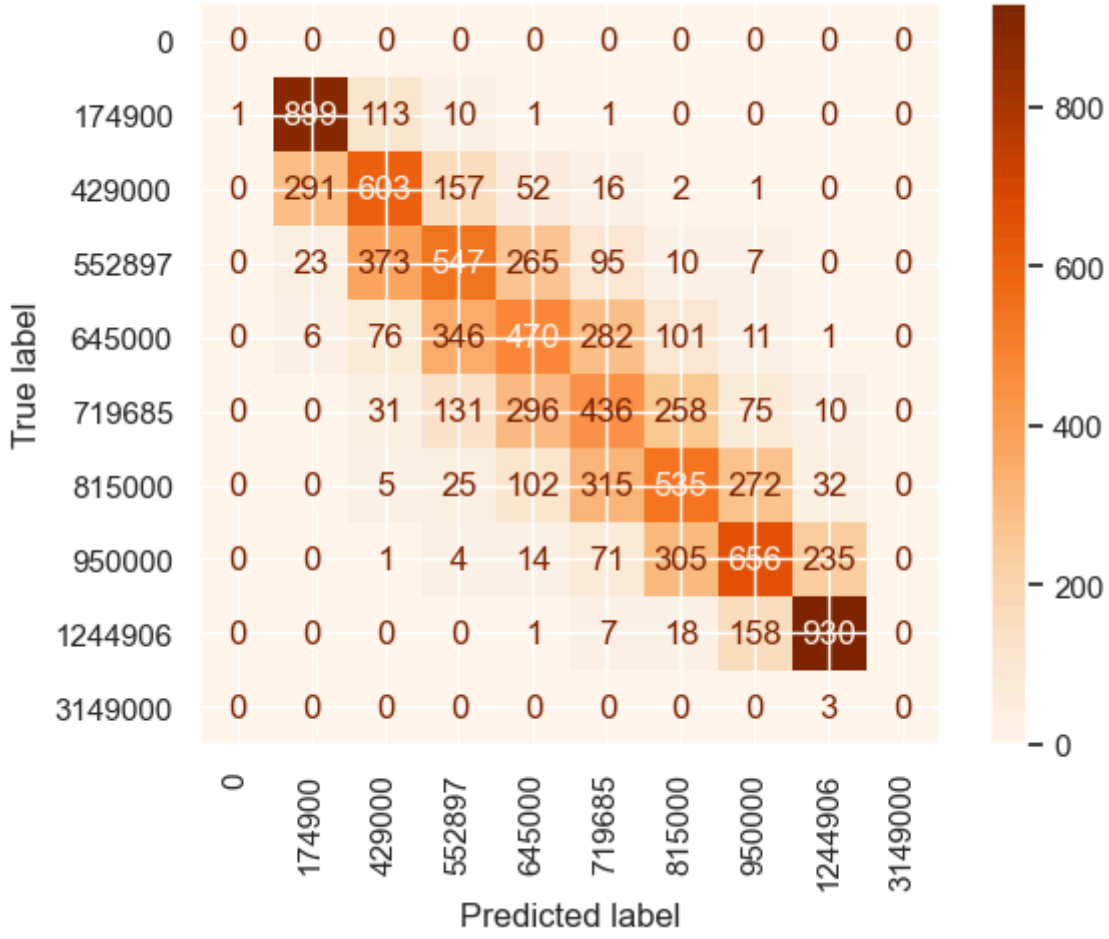
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 100, 'min_child_weight': 1, 'max_de
pth': 9, 'learning_rate': 0.1}
```

In [26]:
```
"""
Test reg_xgb_4f
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "XGBoosting (4 features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.872

RMSLE: 0.026
```




1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6

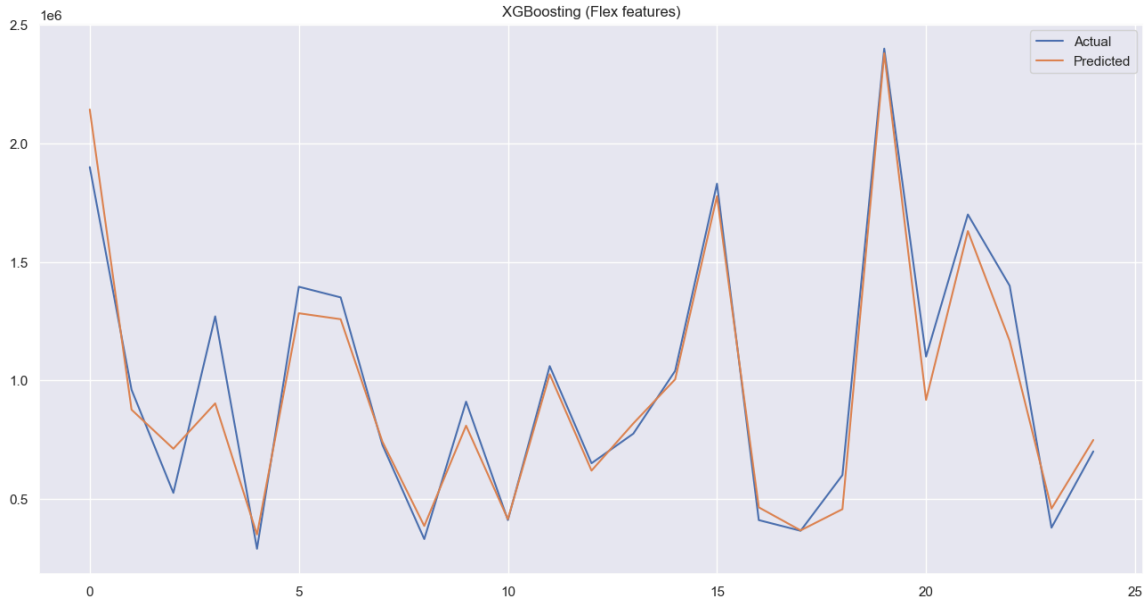


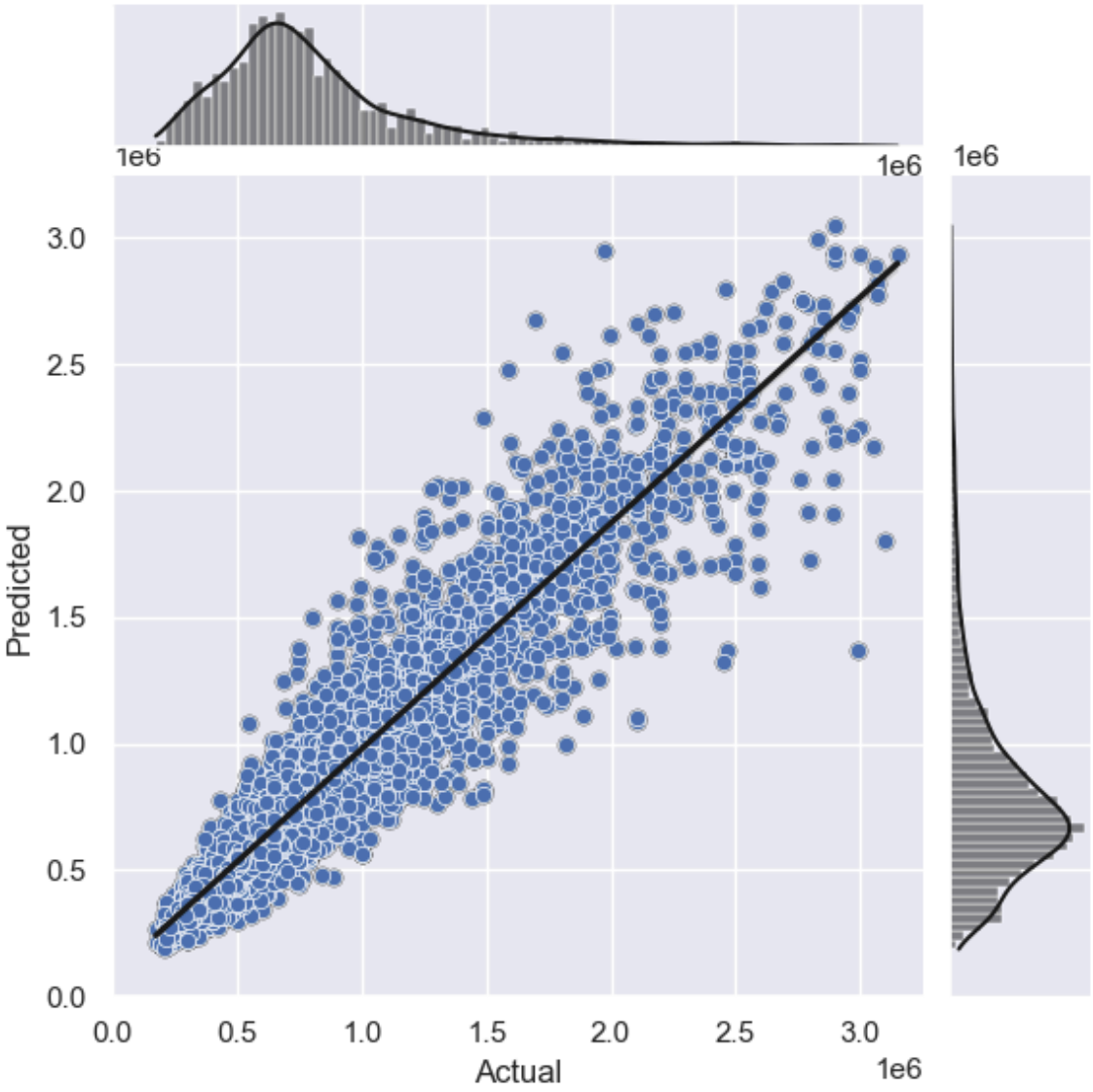
In [27]:
```
"""
Fit XGB regressor for every attribute (reg_xgb_flex)
"""
X_train, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_xgb_flex = XGBRegressor(**reg.best_params_)
preprocessing = DataFrameMapper([(X_train.keys(), [SimpleImputer(missing_val
pipeline_xgb = PMMLPipeline([("preprocessing", preprocessing), ('regressor',

pipeline_xgb.fit(X_train, y_train)
y_pred = pipeline_xgb.predict(X_test)
```

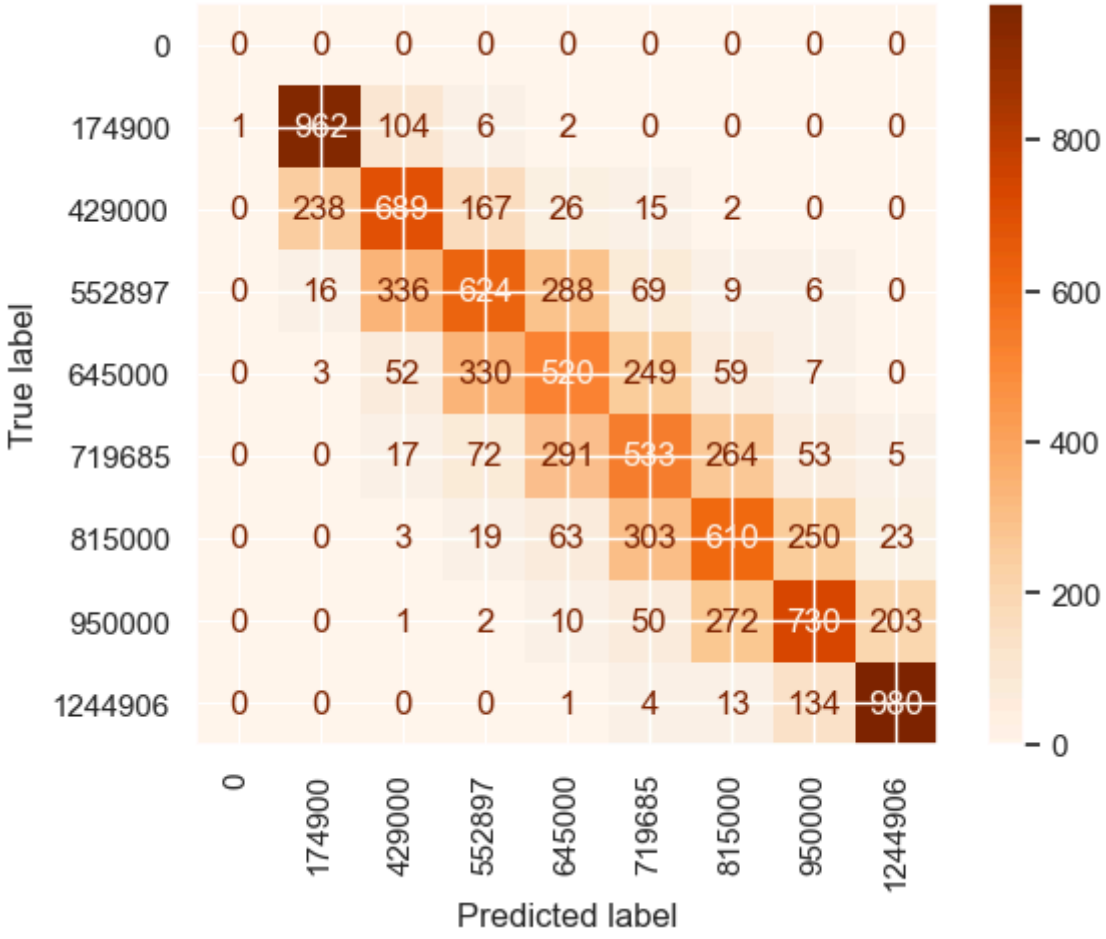
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 400, 'min_child_weight': 1, 'max_de
pth': 9, 'learning_rate': 0.05}
```

In [28]:
```
"""
Test reg_xgb_flex
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "XGBoosting (Flex features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.897

RMSLE: 0.020
```




1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6


True label
0
174900
429000
552897
645000
719685
815000
950000
1244906
0 0 0 0 0 0 0 0 0
1 962 104 6 2 0 0 0 0
0 238 689 167 26 15 2 0 0
0 16 336 624 288 69 9 6 0
0 3 52 330 520 249 59 7 0
0 0 17 72 291 533 264 53 5
0 0 3 19 63 303 610 250 23
0 0 1 2 10 50 272 730 203
0 0 0 0 1 4 13 134 980
0
174900
429000
552897
645000
719685
815000
950000
1244906
Predicted label
800
600
400
200
0

8. Build best random-forest models

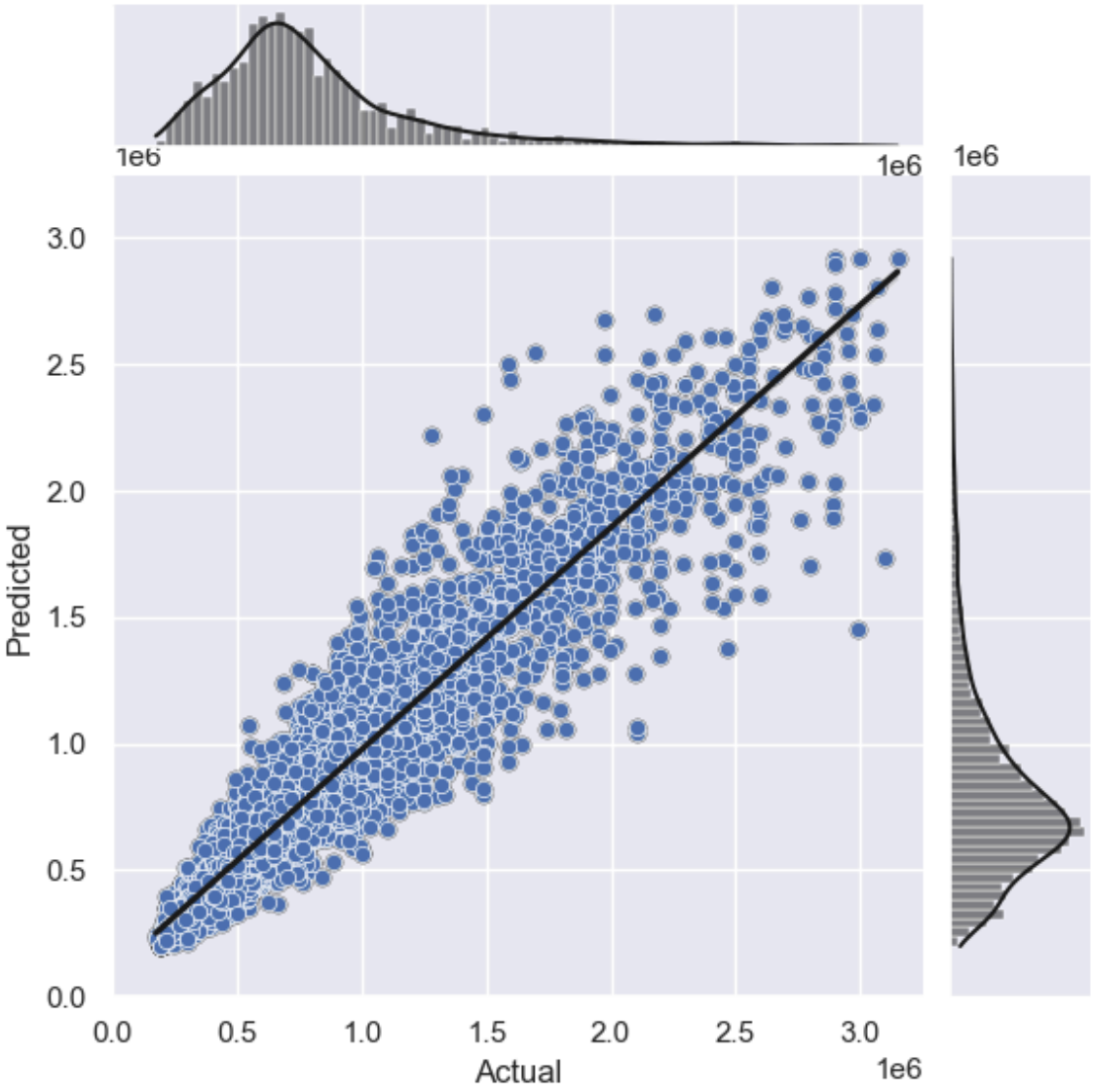
In [29]:
```
"""
Create cross-validation model for random forest regressor.
"""

"""
Default:
parameters={
    'n_estimators': 100,
    'criterion': 'mse',
    'max_depth': None,
    'min_samples_split': 2,
    'min_samples_leaf': 1,
    'max_features': 'auto',
    'bootstrap': True
}
"""

parameters={
    'max_depth': [None, 5, 10, 15],
    'n_estimators': [10, 25, 50, 100],
    'criterion':['squared_error', 'poisson', 'friedman_mse'],
    'min_samples_split': [2, 4, 8],
    'min_samples_leaf': [1, 2, 3],
    'max_features': [0.5, 0.75, 1.0]
}

rf_reg = RandomForestRegressor()
reg = RandomizedSearchCV(
    estimator=rf_reg,
    param_distributions=parameters,
    cv=5,
    scoring='neg_mean_squared_log_error',
    n_jobs=-1,
    verbose=2
)
```

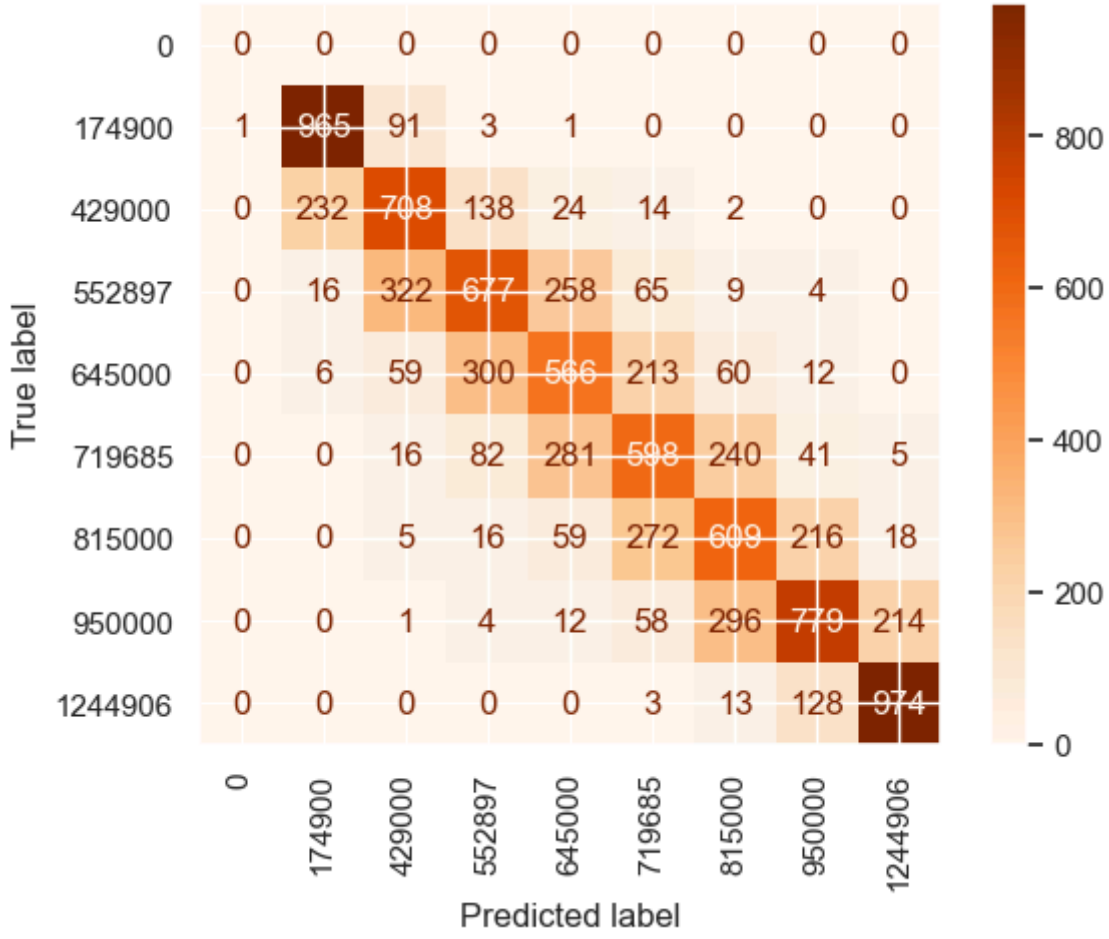
In [30]:
```
"""
Fit random forest regressor for every attribute (reg_rf_15f)
"""
X_train, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_rf_15f = RandomForestRegressor(**reg.best_params_)
reg_rf_15f.fit(X_train, y_train)
y_pred = reg_rf_15f.predict(X_test)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 100, 'min_samples_split': 4, 'min_s
amples_leaf': 2, 'max_features': 0.5, 'max_depth': None, 'criterion': 'squ
ared_error'}
```

In [31]:
```
"""
Test reg_rf_15f
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Random-Forest Regressor (15 features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.898

RMSLE: 0.019
```


1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6

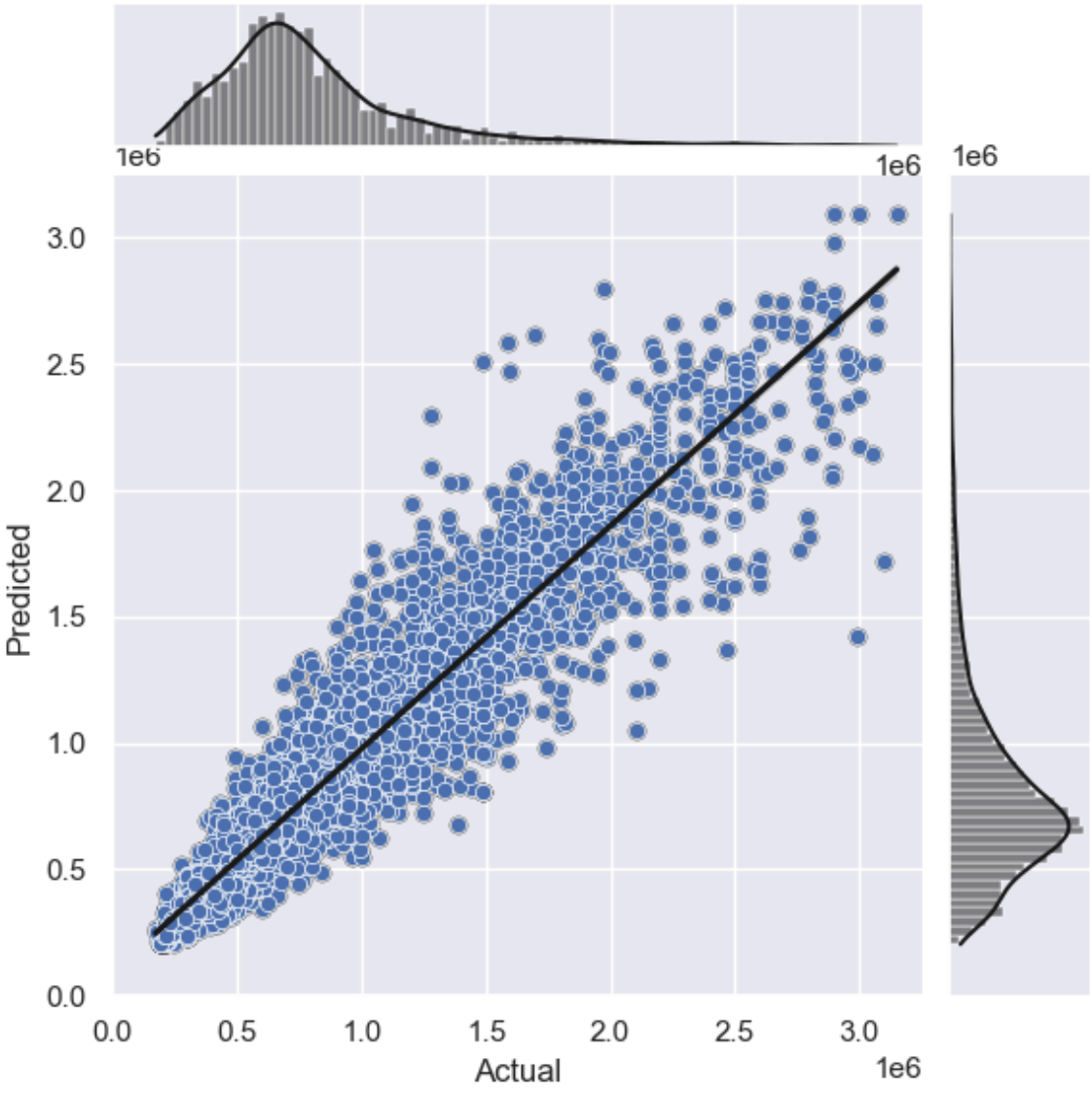


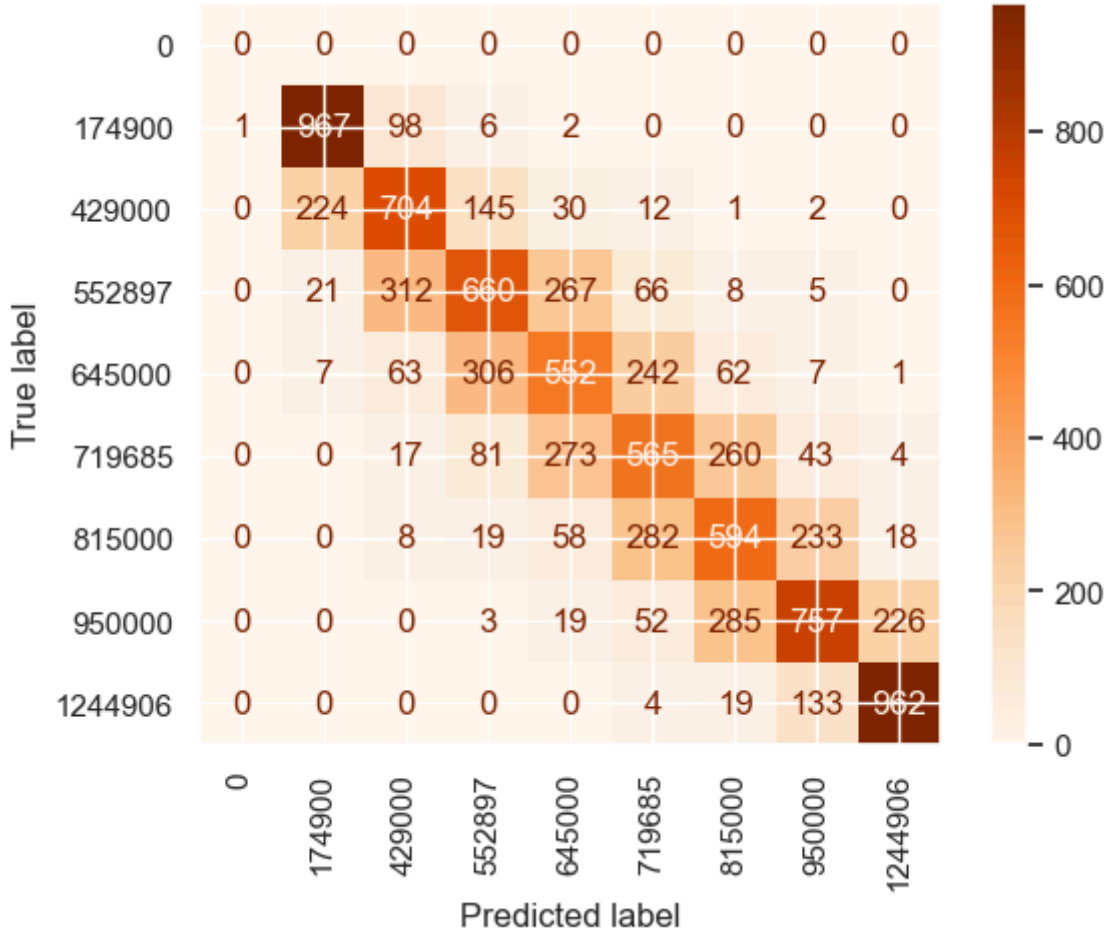
In [32]:
```
"""
Fit random forest regressor for 7 features (reg_rf_7f)
"""
X_train, X_test, y_train, y_test = train_test_split(X_7f, y, test_size=test_

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_rf_7f = RandomForestRegressor(**reg.best_params_)
reg_rf_7f.fit(X_train, y_train)
y_pred = reg_rf_7f.predict(X_test)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 50, 'min_samples_split': 8, 'min_sa
mples_leaf': 3, 'max_features': 0.75, 'max_depth': None, 'criterion': 'poi
sson'}
```

In [33]:
```
"""
Test reg_rf_7f
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Random-Forest Regressor (7 features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.894

RMSLE: 0.020
```


1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6

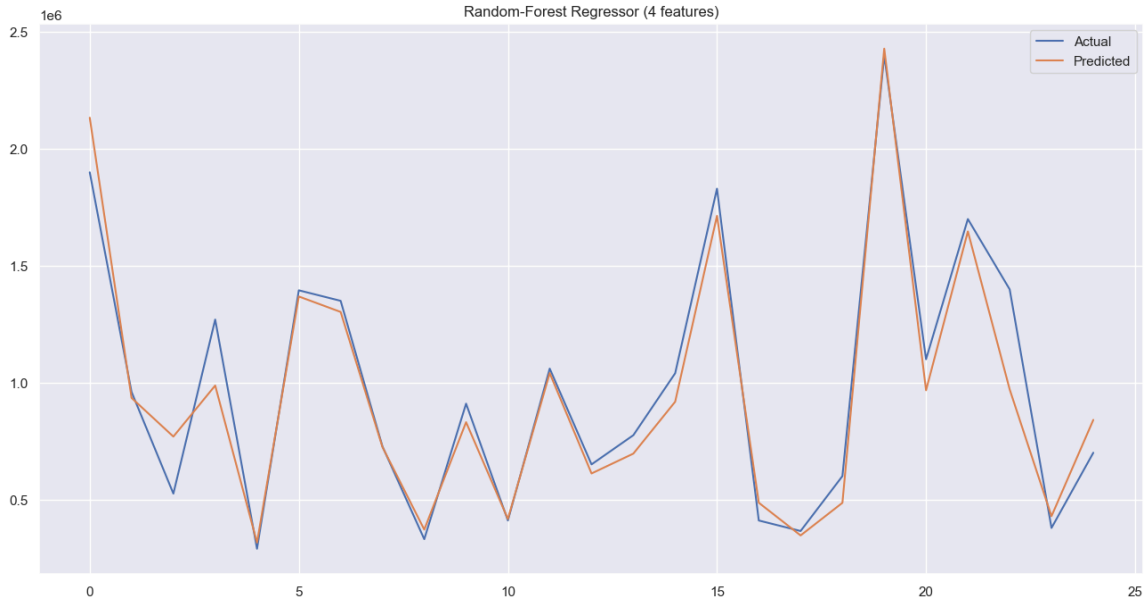


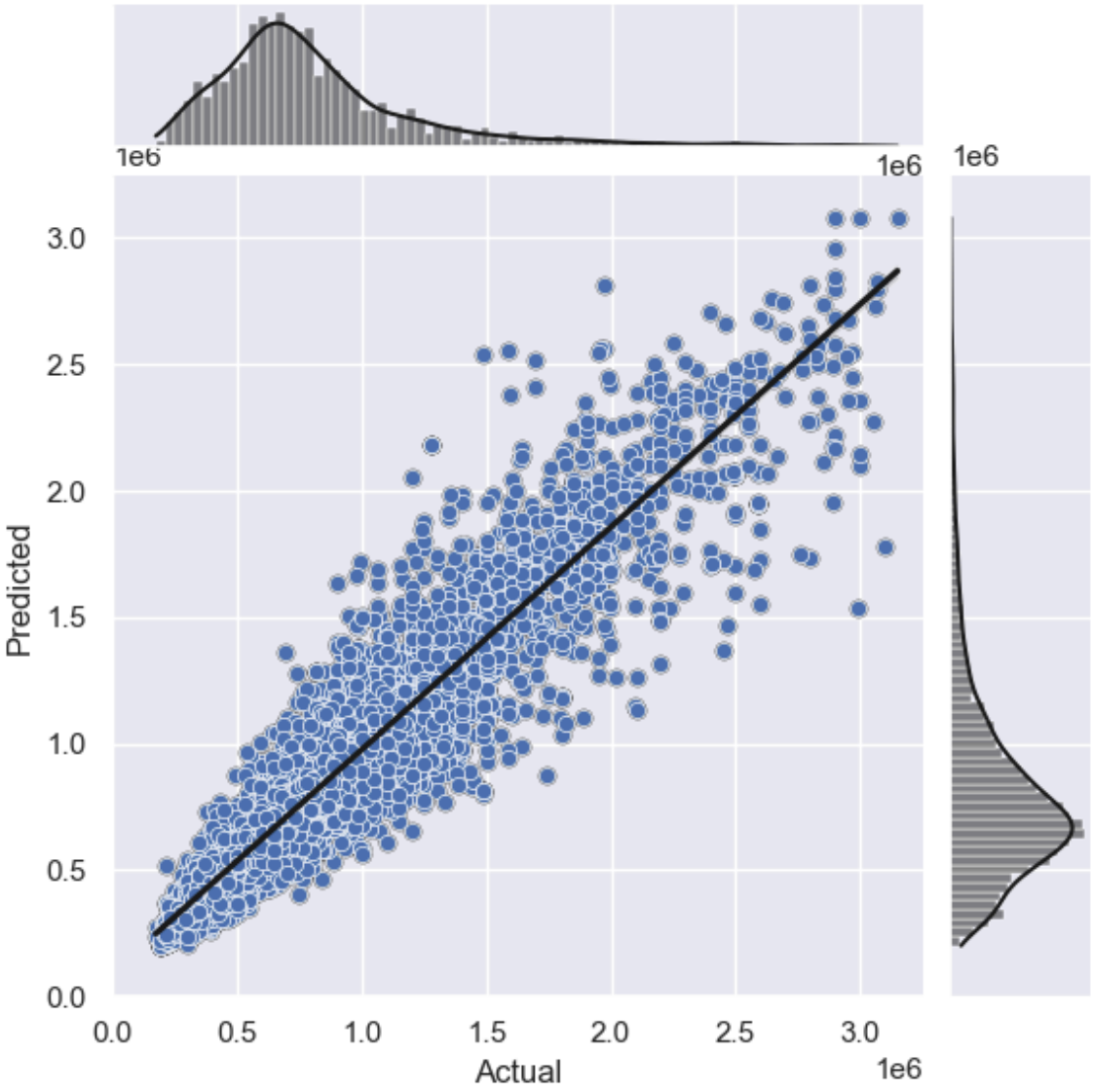
In [34]:
```
"""
Fit random forest regressor for 4 features (reg_rf_4f)
"""
X_train, X_test, y_train, y_test = train_test_split(X_4f, y, test_size=test_

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_rf_4f = RandomForestRegressor(**reg.best_params_)
reg_rf_4f.fit(X_train, y_train)
y_pred = reg_rf_4f.predict(X_test)
```

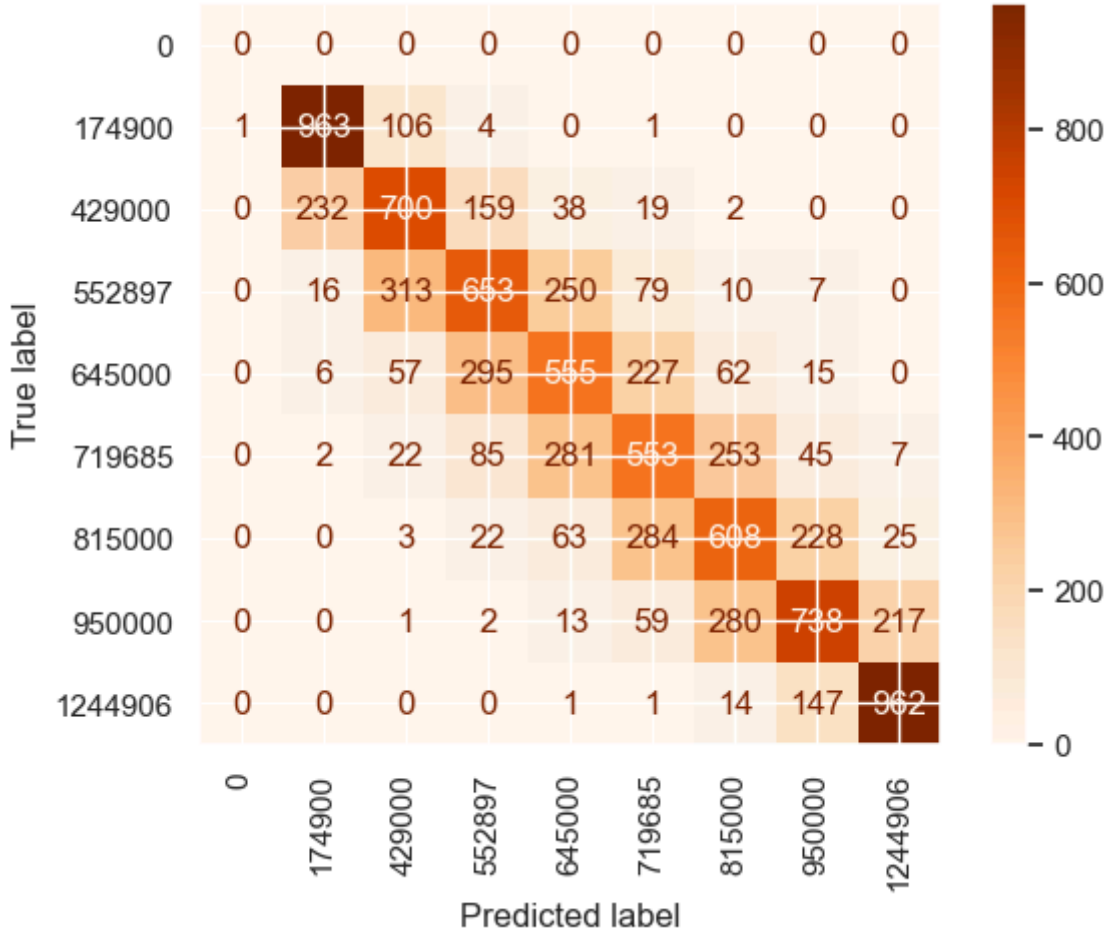
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_s
amples_leaf': 2, 'max_features': 0.5, 'max_depth': None, 'criterion': 'fri
edman_mse'}
```

In [35]:
```
"""
Test reg_rf_4f
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Random-Forest Regressor (4 features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.892

RMSLE: 0.021
```




1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6



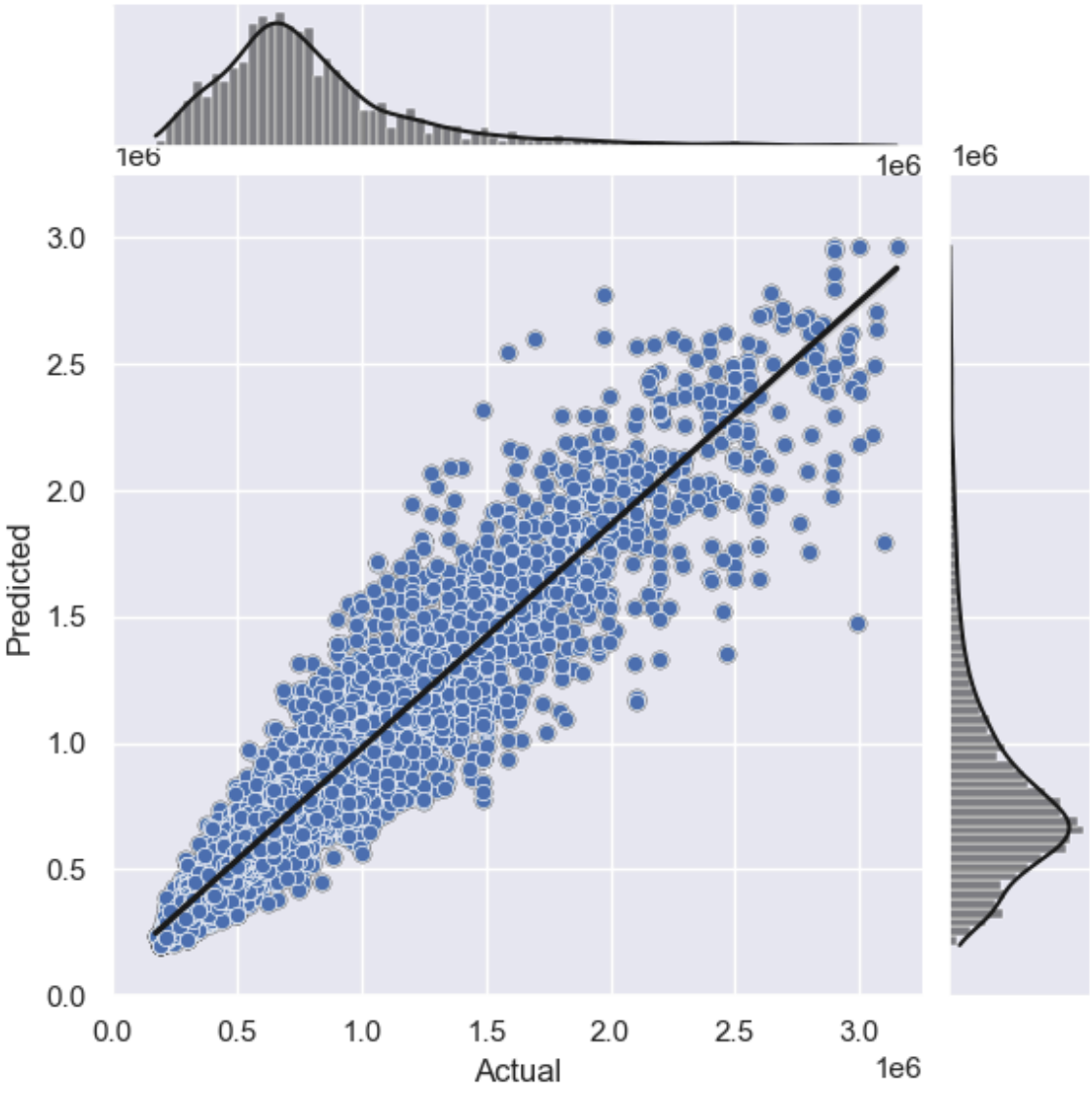
In [36]:

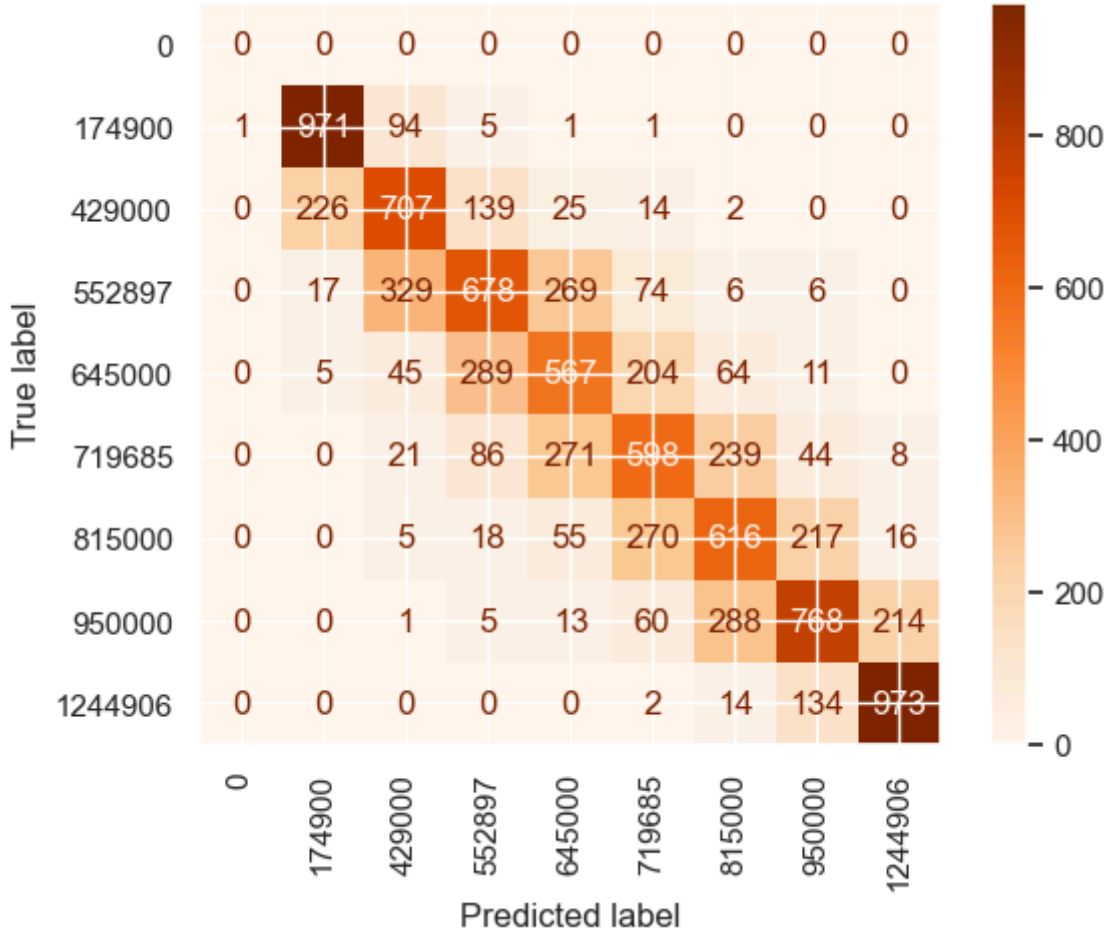
```
"""
Fit random forest regressor for every attribute (reg_rf_flex)
"""
X_train, X_test, y_train, y_test = train_test_split(X_15f, y, test_size=test

reg.fit(X_train, y_train)
print("Best hiperparameters: " + str(reg.best_params_))

reg_rf_flex = RandomForestRegressor(**reg.best_params_)
preprocessing = DataFrameMapper([(X_train.keys(), [SimpleImputer(missing_val
pipeline_rf = PMMLPipeline([("preprocessing", preprocessing), ('regressor',

pipeline_rf.fit(X_train, y_train)
y_pred = pipeline_rf.predict(X_test)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hiperparameters: {'n_estimators': 100, 'min_samples_split': 8, 'min_s
amples_leaf': 1, 'max_features': 0.75, 'max_depth': None, 'criterion': 'po
isson'}
```

In [37]:
```
"""
Test reg_rf_flex
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Random-Forest Regressor (Flex features)")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.899

RMSLE: 0.019
```


1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6



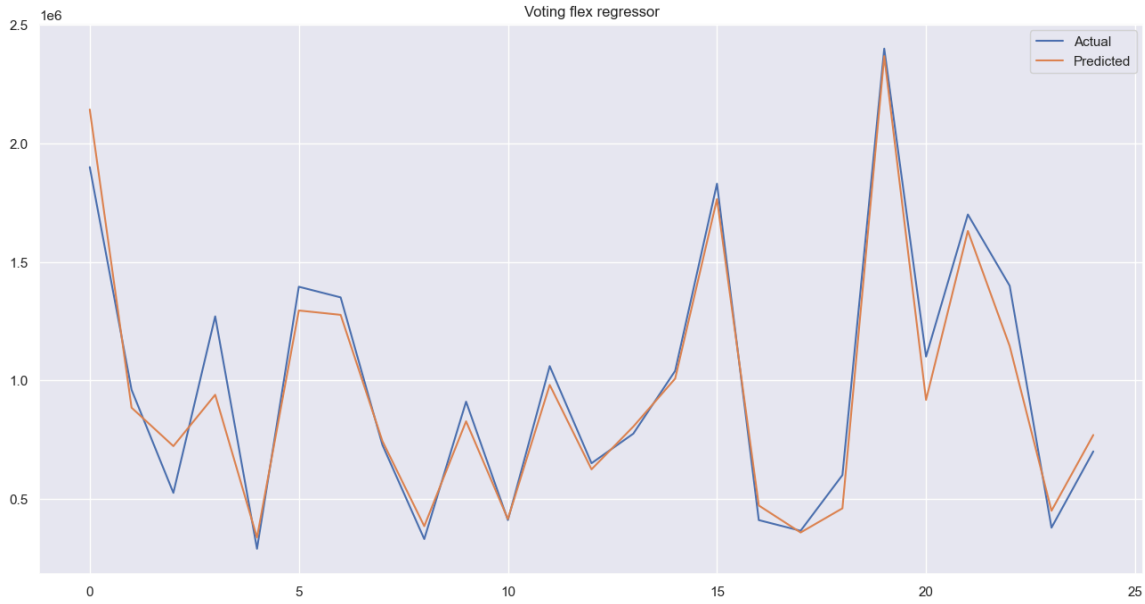
9. Build and test ensemble models

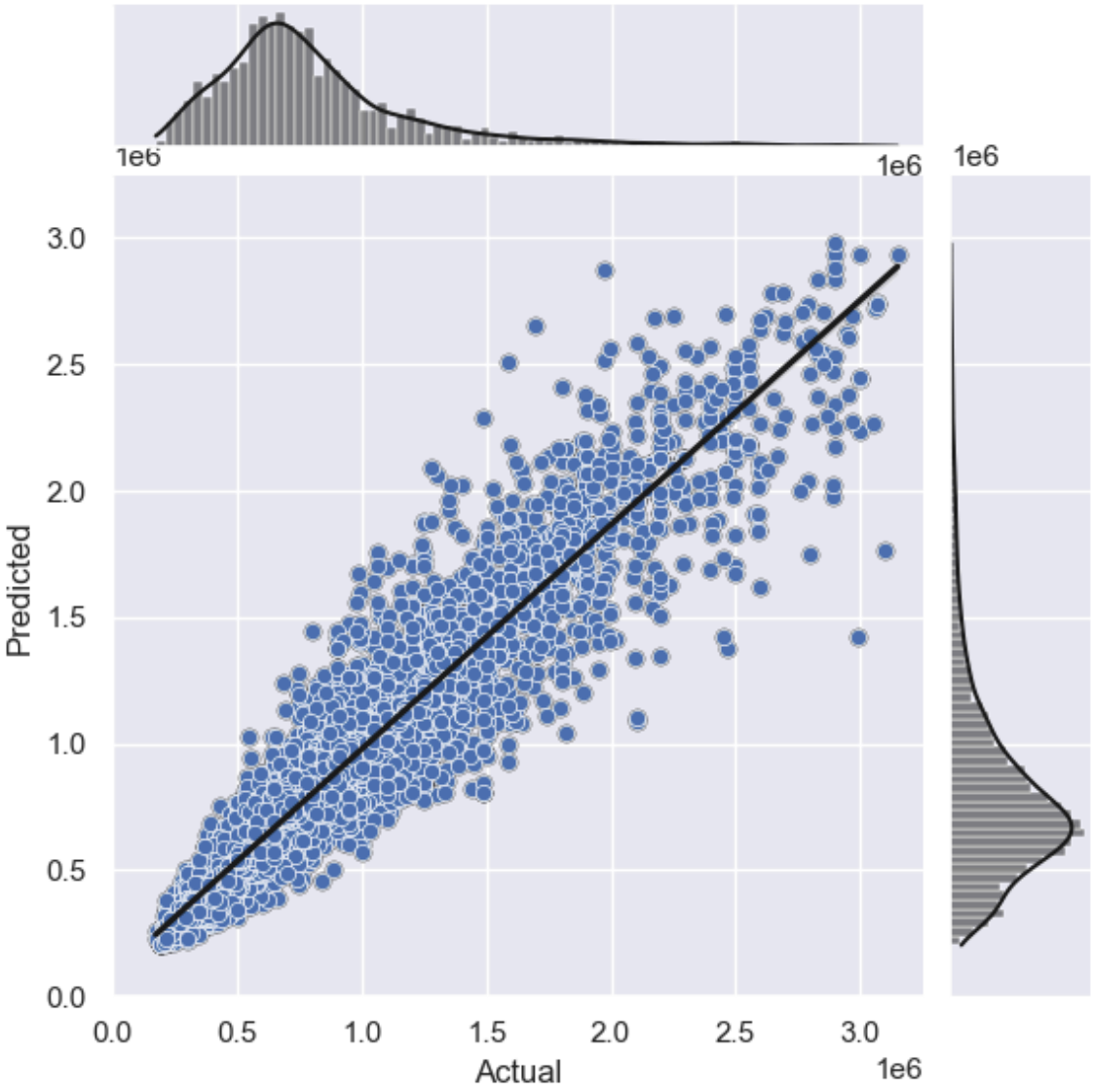
In [38]:
```
"""
Fit and test voting flex model
"""
reg_voting_flex = VotingRegressor(estimators=[('rf', pipeline_rf), ('xgb', p
pipeline_voting_flex = PMMLPipeline([('regressor', reg_voting_flex)])
pipeline_voting_flex.fit(X_train, y_train)
y_pred = pipeline_voting_flex.predict(X_test)
```

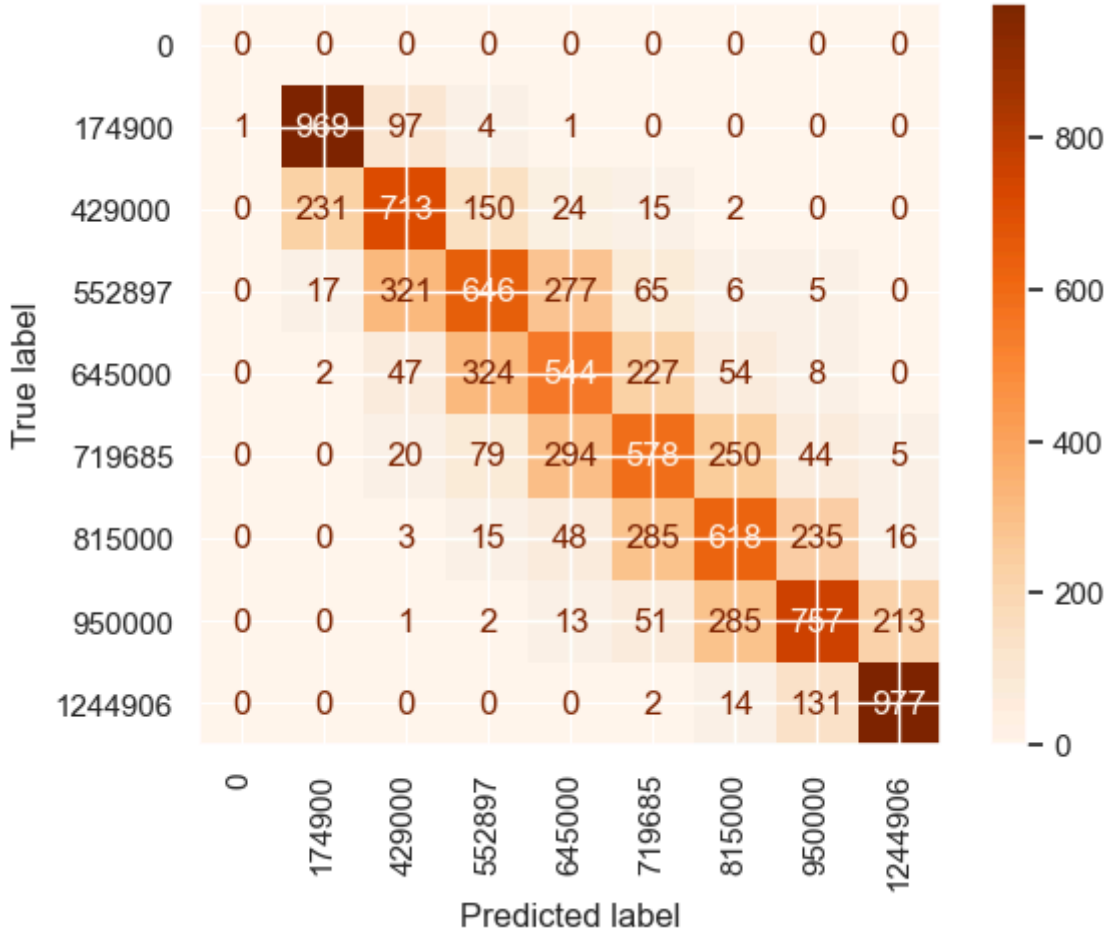
```
C:\Users\dgajd\AppData\Local\Programs\Python\Python39\lib\site-packages\sk
learn2pmml\pipeline\__init__.py:72: UserWarning: y is missing target field
name(s)
  warnings.warn("y is missing target field name(s)")
C:\Users\dgajd\AppData\Local\Programs\Python\Python39\lib\site-packages\sk
learn2pmml\pipeline\__init__.py:72: UserWarning: y is missing target field
name(s)
  warnings.warn("y is missing target field name(s)")
```

In [39]:
```
"""
Test voting flex model
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Voting flex regressor")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.901

RMSLE: 0.019
```



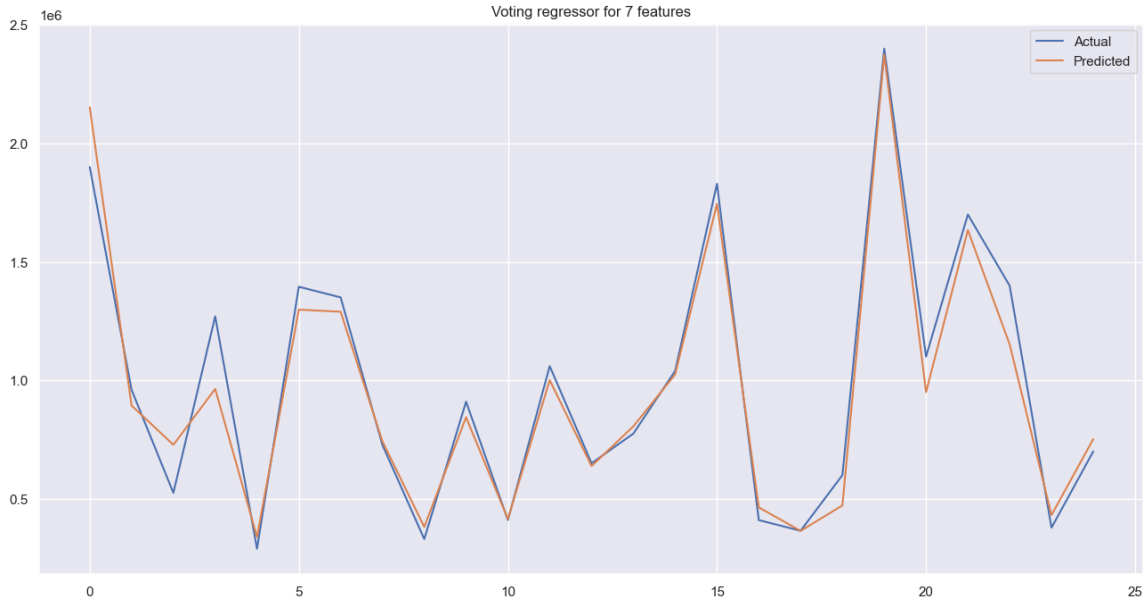

1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6

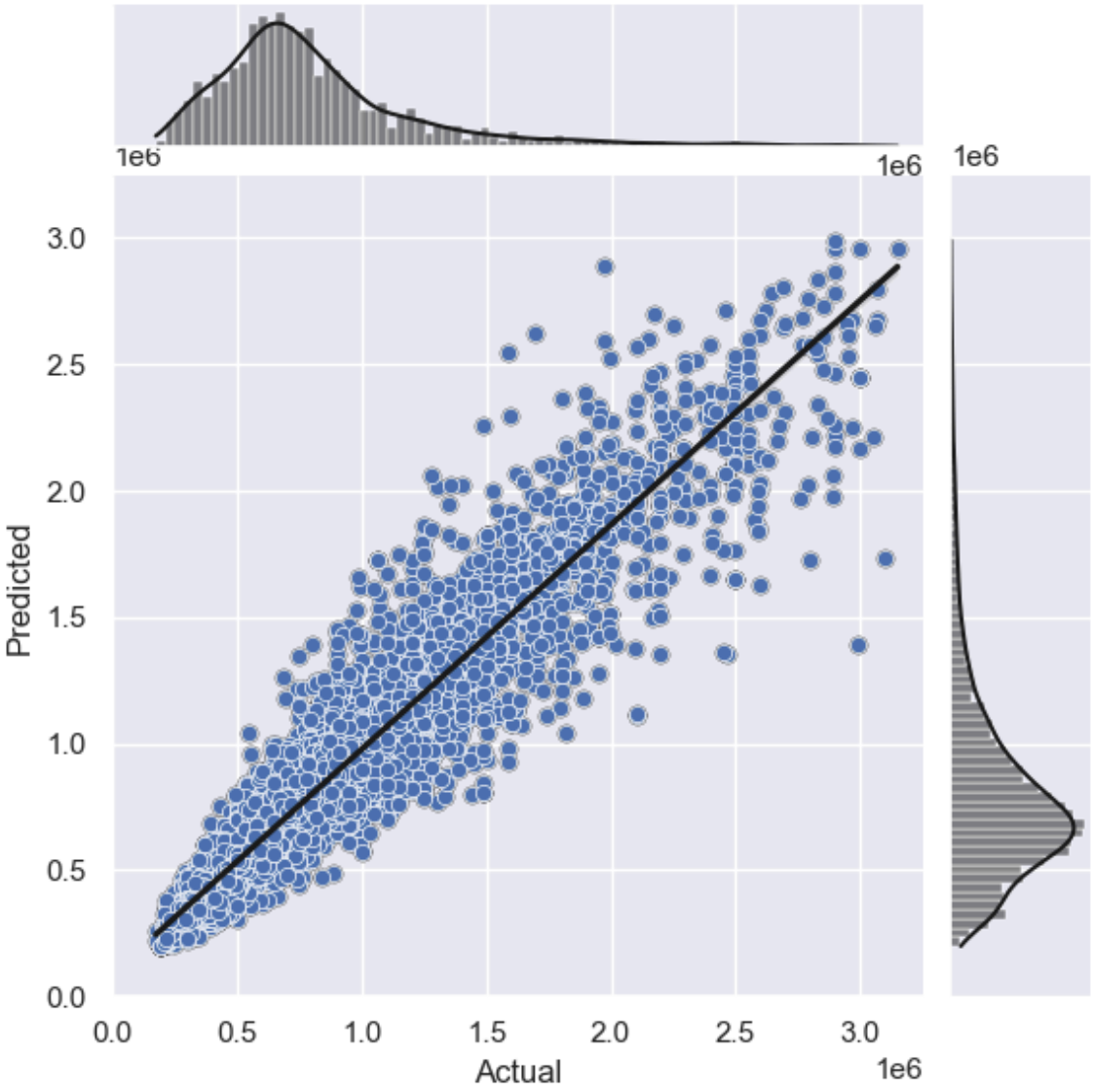


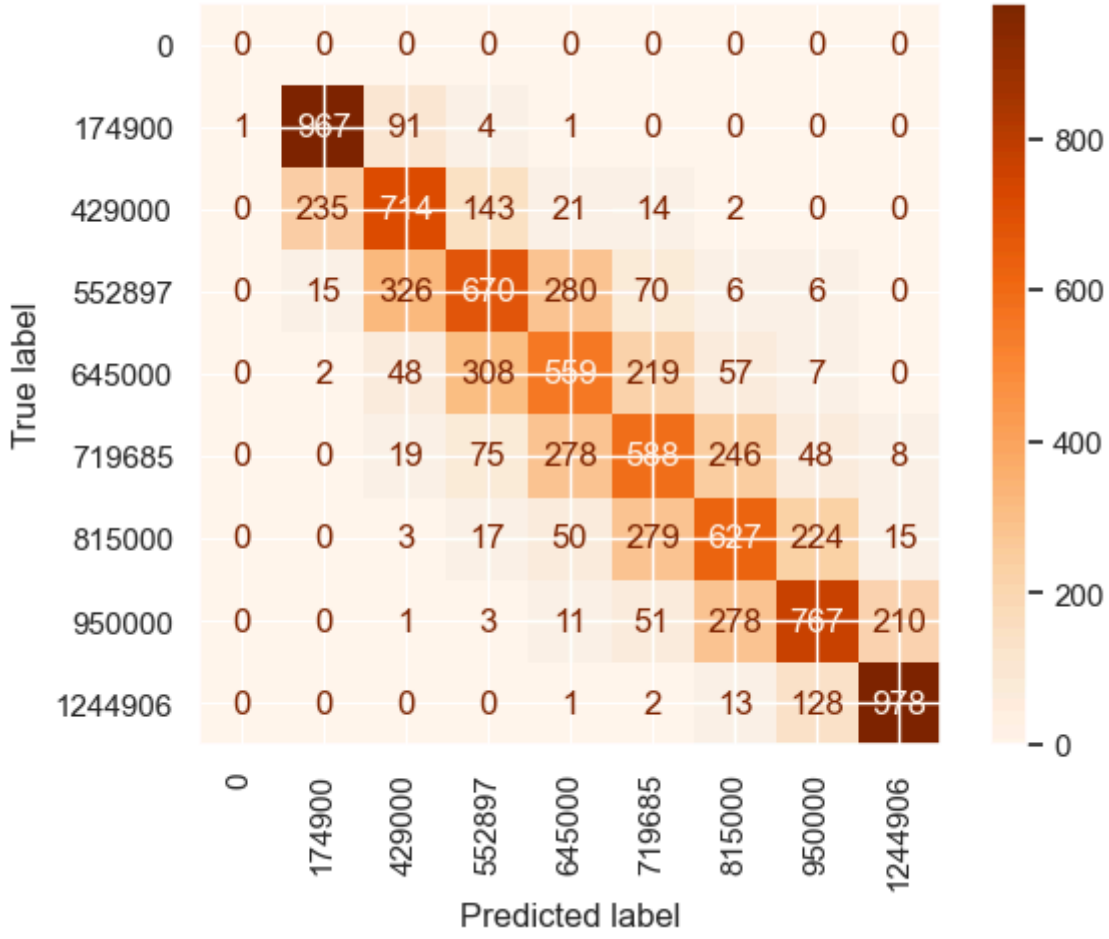
In [53]:
```
"""
Fit and test voting model for 7 features
"""
reg_voting_7f = VotingRegressor(estimators=[('rf', reg_rf_7f), ('xgb', reg_x
pipeline_voting_7f = PMMLPipeline([('regressor', reg_voting_7f)])
pipeline_voting_7f.fit(X_train, y_train)
y_pred = pipeline_voting_7f.predict(X_test)
```

In [54]:
```
"""
Test voting model for 7 features
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Voting regressor for 7 features")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.902

RMSLE: 0.018
```



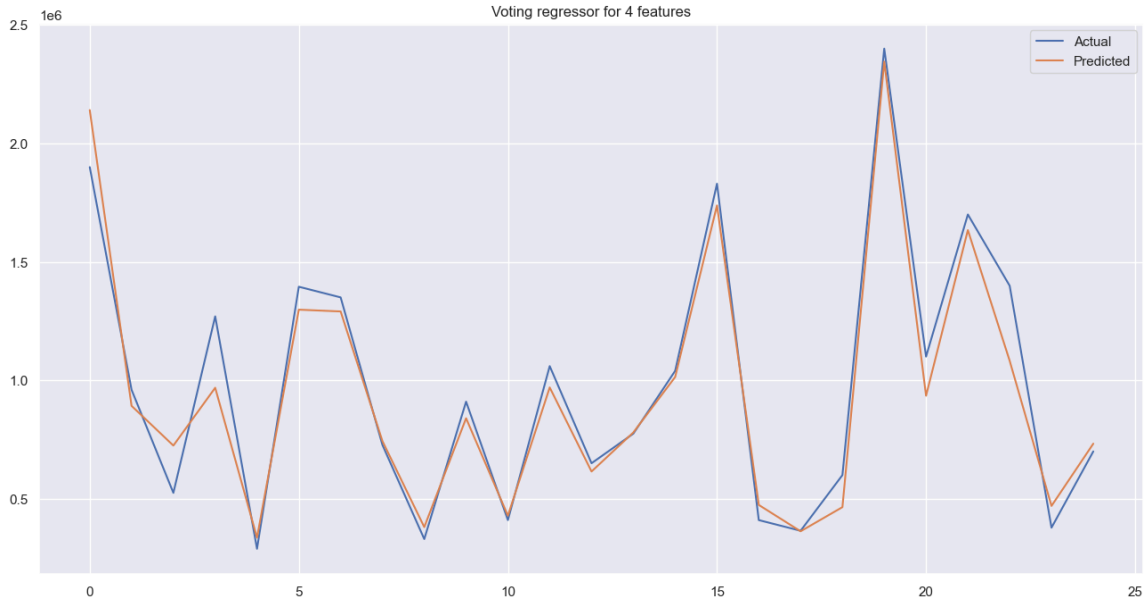

1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6

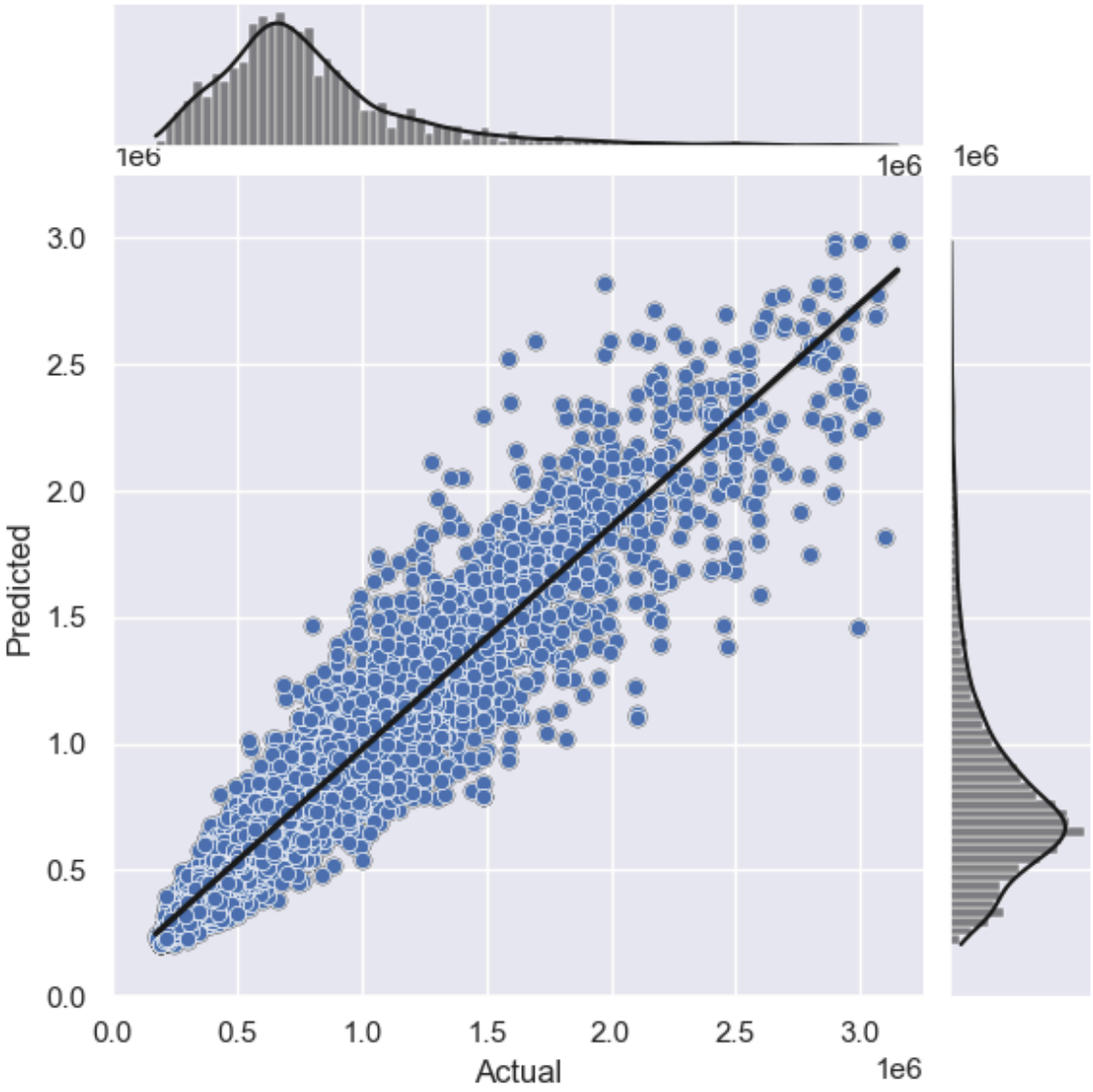


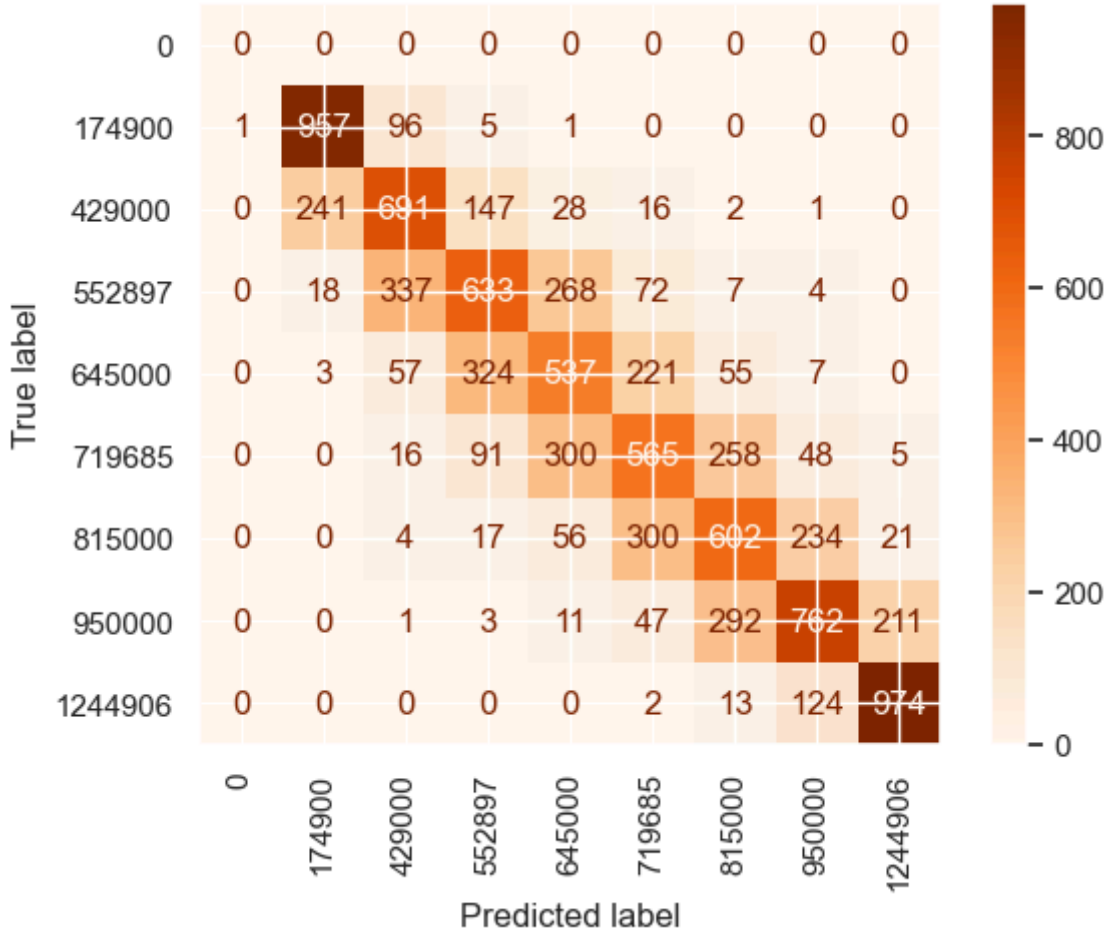
In [55]:
```
"""
Fit and test voting model for 4 features
"""
reg_voting_4f = VotingRegressor(estimators=[('rf', reg_rf_4f), ('xgb', reg_x
pipeline_voting_4f = PMMLPipeline([('regressor', reg_voting_4f)])
pipeline_voting_4f.fit(X_train, y_train)
y_pred = pipeline_voting_4f.predict(X_test)
```

In [56]:
```
"""
Test voting model for 4 features
"""
print_coefficients(y_pred, y_test)
print_diff_plot(y_pred, y_test, "Voting regressor for 4 features")
print_covergence_plot(y_pred, y_test)
print_accuracy_diagram(y_pred, y_test)
```

```
R^2: 0.899

RMSLE: 0.019
```




1e6
1e6
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Actual
1e6



10. Tests

In [58]:
```
X_test_subset = X_test[['latitude', 'longitude', 'floorCount']]
X_test_subset.reset_index(drop=True, inplace=True)
X_test_subset = X_test_subset.loc[0]
print(X_test_subset)


df = pd.DataFrame([X_test_subset], columns=list(X_train.keys()))
test_y = pipeline_voting_flex.predict(df)
print(f"result flex: {test_y}")
```

```
latitude      52.14379
longitude     21.05748
floorCount     4.00000
Name: 0, dtype: float64
result flex: [550740.52059987]
```

11. Data required for every model

In [59]:
```
print("Data for 15-features or flex mode")
print(list(X_train.keys()))

print("\nData for 7-features model")
print(columns_7f)

print("\nData for 4-features model")
print(columns_4f)
```

```
Data for 15-features or flex mode
['squareMeters', 'rooms', 'floorCount', 'latitude', 'longitude', 'centreDi
stance', 'poiCount', 'schoolDistance', 'clinicDistance', 'postOfficeDistan
ce', 'kindergartenDistance', 'restaurantDistance', 'collegeDistance', 'pha
rmacyDistance', 'bialystok', 'bydgoszcz', 'czestochowa', 'gdansk', 'gdyni
a', 'katowice', 'krakow', 'lodz', 'lublin', 'poznan', 'radom', 'rzeszow',
'szczecin', 'warszawa', 'wroclaw']

Data for 7-features model
['squareMeters', 'rooms', 'floorCount', 'latitude', 'longitude', 'centreDi
stance', 'bialystok', 'bydgoszcz', 'czestochowa', 'gdansk', 'gdynia', 'kat
owice', 'krakow', 'lodz', 'lublin', 'poznan', 'radom', 'rzeszow', 'szczeci
n', 'warszawa', 'wroclaw']

Data for 4-features model
['squareMeters', 'latitude', 'longitude', 'bialystok', 'bydgoszcz', 'czest
ochowa', 'gdansk', 'gdynia', 'katowice', 'krakow', 'lodz', 'lublin', 'pozn
an', 'radom', 'rzeszow', 'szczecin', 'warszawa', 'wroclaw']
```

12. Export model to PMML

In [60]:
```
sklearn2pmml(pipeline_voting_4f, "reg_4f.pmml", with_repr=True)
sklearn2pmml(pipeline_voting_7f, "reg_7f.pmml", with_repr=True)
sklearn2pmml(pipeline_voting_flex, "reg_flex.pmml", with_repr=True)
```