

2_analyza_cen_BJ_z_excel_datasetu

May 22, 2025

```
[48]: # Import dat z tabulky v EXCELU - cesty a nactení DF

import pandas as pd
from pathlib import Path

# jednou definujete cestu a vytvoříte df
excel_path = Path(
    r"C:\Users\ijttr\OneDrive\Dokumenty\OCEŇOVÁNÍ\_IJK"

    r"\035535-2025 - BD - Praha 7 - Holešovice - Dělnická - Osadní"
    r"\035535-2025 - BD - Praha 7 - Holešovice - Dělnická - Osadní.xlsx"

    #r"\024276 - 2025 - Praha 6 - Dejvice - Hanspaulka - VILA rozdělení na BJ - 01t"
    #r"\024276 - 2025.xlsx"
)
df = pd.read_excel(excel_path, sheet_name="data_byty", engine="openpyxl")

# rychlá kontrola
print(f"Načteno řádků: {df.shape[0]}, sloupců: {df.shape[1]}")

columns_list = df.columns.tolist()
print(columns_list)
```

Načteno řádků: 385, sloupců: 150

['Číslo vkladu', 'Datum podání', 'Datum zplatnění', 'Listina', 'Nemovitost', 'Typ', 'LAT', 'LON', 'Vzdálenost [km]', 'Adresa', 'Cenový údaj', 'Měna', 'Plocha [m2]', 'Typ plochy', 'Popis', 'Okres', 'Kat# území', 'Rok', 'Měsíc', 'nem', 'jednotka', 'byt', 'budova', 'parcela', 'rd', 'garáž', 'SUM Plocha bytů dle řízení [m2]', 'SUM Cena bytů dle řízení [Kč]', 'JC byty [Kč/m2]', 'Q_JC byty', '1', '# Q JC [Kč/m²]', 'Unnamed: 32', 'Unnamed: 33', 'Unnamed: 34', 'Unnamed: 35', 'Unnamed: 36', 'LAT =', 50.103376078, 'LON =', 14.447172818, 'Unnamed: 41', 'Unnamed: 42', 'Unnamed: 43', 'Unnamed: 44', 'Unnamed: 45', 'Unnamed: 46', 'Unnamed: 47', 'Unnamed: 48', 'Unnamed: 49', 'Unnamed: 50', 'Unnamed: 51', 'Unnamed: 52', 'Unnamed: 53', 'Unnamed: 54', 'Unnamed: 55', 'Unnamed: 56', 'Unnamed: 57', 'Unnamed: 58', 'Unnamed: 59', 'Unnamed: 60', 'Unnamed: 61', 'Unnamed: 62', 'Unnamed: 63', 'Unnamed: 64', 'Holešovice', 2, 'vklad', 'datum podání', 'ku', 'I', 'plocha [m2]', 'JC [Kč/m2]', 'JC I [Kč/m2]', 'Unnamed: 65']

```

74', 'Unnamed: 75', 'Unnamed: 76', 'Unnamed: 77', 'Unnamed: 78', 'Unnamed: 79',
'Unnamed: 80', 'Unnamed: 81', 'Unnamed: 82', 'Unnamed: 83', 'Unnamed: 84',
'Unnamed: 85', 'Unnamed: 86', 'Unnamed: 87', 'Unnamed: 88', 'Unnamed: 89',
'Unnamed: 90', 'Unnamed: 91', 'Unnamed: 92', 'Unnamed: 93', 'Unnamed: 94',
'Unnamed: 95', 'Unnamed: 96', 'Unnamed: 97', 'Unnamed: 98', 'Unnamed: 99',
'Unnamed: 100', 'Unnamed: 101', 'Unnamed: 102', 'Unnamed: 103', nan, 'Unnamed:
105', 'Unnamed: 106', 'Unnamed: 107', 'Unnamed: 108', 'Unnamed: 109', 'Unnamed:
110', 'Unnamed: 111', 'Unnamed: 112', 'Unnamed: 113', 'Unnamed: 114', 'Unnamed:
115', 'Unnamed: 116', 'Unnamed: 117', 'Unnamed: 118', 'Unnamed: 119', 'Unnamed:
120', 'K-means', 'M Lokte', 'PairAnlClust', 'Unnamed: 124', 'Unnamed: 125',
'Unnamed: 126', 'Unnamed: 127', 'Unnamed: 128', 'Unnamed: 129', 'Unnamed: 130',
'Unnamed: 131', 'Unnamed: 132', 'Unnamed: 133', 'Unnamed: 134', 'Unnamed: 135',
'Unnamed: 136', 'Unnamed: 137', 'Unnamed: 138', 'Unnamed: 139', 'Unnamed: 140',
'Unnamed: 141', 'Unnamed: 142', 'Unnamed: 143', 'Unnamed: 144', 'Unnamed: 145',
'Unnamed: 146', 'Unnamed: 147', 'RA dle clusteru', 'data se shlukovou analyzou']

```

```

C:\Users\ijtttr\AppData\Roaming\Python\Python312\site-
packages\openpyxl\worksheet\_reader.py:329: UserWarning: Unknown extension is
not supported and will be removed

```

```

warn(msg)

```

```

C:\Users\ijtttr\AppData\Roaming\Python\Python312\site-
packages\openpyxl\worksheet\_reader.py:329: UserWarning: Conditional Formatting
extension is not supported and will be removed

```

```

warn(msg)

```

[65]: *# SQL dotaz pro nacteni dat z DB*

```

query_byty = ""

```

```

DECLARE

```

```

    @cols          NVARCHAR(MAX),
    @alias_pl      NVARCHAR(128),
    @alias_cu      NVARCHAR(128),
    @alias_dp      NVARCHAR(128),
    @alias_jc_int  NVARCHAR(128),
    @alias_jc_out  NVARCHAR(128),
    @alias_cv      NVARCHAR(128),
    @sql           NVARCHAR(MAX);

```

```

-- 1) Seznam sloupců C.* kromě 'plocha','cenovy_udaj','datum_podani' a
↳ 'cislo_vkladu'

```

```

SELECT

```

```

    @cols = STRING_AGG('C.' + QUOTENAME(name), ', ')

```

```

FROM sys.columns

```

```

WHERE object_id = OBJECT_ID('dbo.Valuo_data')

```

```

    AND name NOT IN ('plocha','cenovy_udaj','datum_podani','cislo_vkladu');

```

```

-- 2) Aliasy

```

```

SET @alias_pl      = QUOTENAME('Plocha [m2]');
SET @alias_cu      = QUOTENAME('Cenový údaj');
SET @alias_dp      = QUOTENAME('Datum podání');
SET @alias_jc_int  = 'JC_byty_Kc_m2';
SET @alias_jc_out  = QUOTENAME('JC byty [Kč/m2]');
SET @alias_cv      = QUOTENAME('Číslo vkladu');           -- nový alias

-- 3) Vytvoření dynamického SQL
SET @sql = N'
WITH ValidValuo AS (
    SELECT *
    FROM dbo.Valuo_data
    WHERE cislo_vkladu IN (
        SELECT cislo_vkladu
        FROM dbo.Valuo_data
        GROUP BY cislo_vkladu
        HAVING COUNT(*) = 1
        AND MAX(typ) = ''byt''
    )
),
Computed AS (
    SELECT
        V.*,
        CAST(
            ROUND(
                V.cenovy_udaj
                / NULLIF(SUM(V.plocha) OVER (PARTITION BY V.cislo_vkladu), 0)
            , 0)
        AS DECIMAL(38,0)
        ) AS ' + @alias_jc_int + '
    FROM ValidValuo AS V
)
SELECT
    ' + @cols + ',                               -- všechny ostatní C.* sloupce
    C.cislo_vkladu AS ' + @alias_cv + ',          -- teď přejmenované
    C.plocha      AS ' + @alias_pl + ',
    C.cenovy_udaj AS ' + @alias_cu + ',
    C.datum_podani AS ' + @alias_dp + ',
    C.' + @alias_jc_int + ' AS ' + @alias_jc_out + ',
    K.kat_uzemi   AS kat_uzemi_KN,
    K.upper_zoning_id,
    K.parcel_number,
    K.gml_id,
    K.areaValue_m2,
    K.beginLifespanVersion,
    K.endLifespanVersion,
    K.geometry,

```

```

        K.inspire_localId,
        K.inspire_namespace,
        K.label,
        K.nationalCadastralReference,
        K.refPoint_x,
        K.refPoint_y,
        K.refPoint_lon,
        K.refPoint_lat,
        K.validFrom,
        K.administrativeUnit_href,
        K.administrativeUnit_title,
        K.zoning_href,
        K.zoning_title,
        K.id_valuo
FROM Computed AS C
LEFT JOIN dbo.KN_parcel_data AS K
    ON K.id_valuo = C.id
WHERE
    -- C.kat_uzemi IN ('Dejvice', 'Stodůlky', 'Holešovice')
    C.kat_uzemi IN ('Dejvice', 'Stodůlky', 'Holešovice')
    AND C.datum_podani >= '2021-01-01'
    AND C.mena = 'CZK'
    AND C.plocha > 15
    AND C.plocha < 200
    AND C.' + @alias_jc_int + ' > 70000
    AND C.' + @alias_jc_int + ' < 299000
;';

-- 4) Provedeme
EXEC sp_executesql @sql;

""

```

[66]: *# Oceňované byty - dataframe s udati*

```

import pandas as pd

# Definice DataFrame df_objc s jedním řádkem
df_objc = pd.DataFrame({
    'byt': ['1', '2', '3', '4'],
    'LAT': [50.10185708, 50.10185708, 50.10185708, 50.10185708],
    'LON': [14.3745576, 14.3745576, 14.3745576, 14.3745576],
    'Plocha [m2]': [89.17, 105.17, 138.44, 169.51],
    #'Plocha [m2]': [89.17, 168.17, 168.44, 149.51],

```

```

        'Vzdálenost [km]': [0,0,0,0],
        'Datum podání (dny)': [2700,2700,2700,2700]
    })

# Zobrazení výsledného DataFrame
df_objc

```

```

[66]:   byt      LAT      LON  Plocha [m2]  Vzdálenost [km]  Datum podání (dny)
0    1  50.101857  14.374558      89.17           0         2700
1    2  50.101857  14.374558     105.17           0         2700
2    3  50.101857  14.374558     138.44           0         2700
3    4  50.101857  14.374558     169.51           0         2700

```

```

[67]: # Vložení dat z SQL do DataFrame a dopoctení sloupce Vzdálenost [km] dle GPS
      ↪souradnic ocenovane nemovitosti

```

```

import numpy as np
import pandas as pd
from pathlib import Path
from sqlalchemy import create_engine, text
import urllib.parse

params_conn = urllib.parse.quote_plus(
    "Driver={ODBC Driver 17 for SQL Server};"
    "Server=localhost;"
    "Database=VALUO;"
    "Trusted_Connection=yes;"
)

connection_url = f"mssql+pyodbc:///odbc_connect={params_conn}"
engine = create_engine(connection_url)

# Dotaz je definován externě (proměnná 'query_byty')
# načtou se data z SQL
df_1 = pd.read_sql(query_byty, engine)

# dopoctení sloupce Vzdálenost [km]
# GPS souradnice ocenovane neoovitosti

# Haversineova funkce pro vzdálenost mezi dvěma body (lat, lon) v km
def haversine(lat1, lon1, lat2, lon2):
    R = 6371.0 # poloměr Země v kilometrech
    phi1, phi2 = np.radians(lat1), np.radians(lat2)
    d_phi = np.radians(lat2 - lat1)
    d_lambda = np.radians(lon2 - lon1)
    a = np.sin(d_phi/2)**2 + np.cos(phi1)*np.cos(phi2)*np.sin(d_lambda/2)**2

```

```

    return R * 2 * np.arcsin(np.sqrt(a))

# Vytáhneme unikátní body z ocenovaných bytů
unique_points = df_objc[['LAT', 'LON']].drop_duplicates().values

# Funkce pro min. vzdálenost ke všem těmto bodům
def compute_min_dist(row):
    lat, lon = row['LAT'], row['LON']
    dists = [haversine(lat, lon, lat2, lon2) for lat2, lon2 in unique_points]
    return round(min(dists), 3)

# Vytvoříme kopii původního df_1 a přidáme nový sloupec
df = df_1.copy()
df['Vzdálenost [km]'] = df.apply(compute_min_dist, axis=1)

# Teď máte v `df` přesně to, co jste chtěl - všechny původní sloupce
# + nový sloupec Vzdálenost [km] (km, 3 desetinná místa).

# rychlá kontrola
print(f"Načteno řádků: {df.shape[0]}, sloupců: {df.shape[1]}")

columns_list = df.columns.tolist()
print(columns_list)

```

Načteno řádků: 1679, sloupců: 46

```

['adresa', 'datum_zplatneni', 'GPS_API_info', 'id', 'kat_uzemi', 'KN_WFS_info',
'LAT', 'listina', 'LON', 'mena', 'mesic', 'nemovitost', 'okres', 'popis', 'rok',
'timestamp', 'typ', 'typ_plochy', 'Číslo vkladu', 'Plocha [m2]', 'Cenový údaj',
'Datum podání', 'JC byty [Kč/m2]', 'kat_uzemi_KN', 'upper_zoning_id',
'parcel_number', 'gml_id', 'areaValue_m2', 'beginLifespanVersion',
'endLifespanVersion', 'geometry', 'inspire_localId', 'inspire_namespace',
'label', 'nationalCadastralReference', 'refPoint_x', 'refPoint_y',
'refPoint_lon', 'refPoint_lat', 'validFrom', 'administrativeUnit_href',
'administrativeUnit_title', 'zoning_href', 'zoning_title', 'id_valuo',
'Vzdálenost [km]']

```

[68]: # MAPA 1 - dynamická mapa s pop-upy

```

# Mapy / Ceny / Street View

import folium
import math
import branca.colormap as cm
from IPython.display import display

```

```

import pandas as pd
from folium.plugins import MarkerCluster

# --- 0) Váš Google Maps Embed API KEY -----
API_KEY = "AIzaSyCY2ClgEZNv3HD7q0qYqTsPtXCM0h1RIsm"

# --- 1) Funkce pro výpočet headingu a jeho otočení o 180° -----
def compute_heading(lat, lon, delta=0.0001):
    lat2, lon2 = lat + delta, lon
    dLon = math.radians(lon2 - lon)
    y = math.sin(dLon) * math.cos(math.radians(lat2))
    x = (math.cos(math.radians(lat)) * math.sin(math.radians(lat2))
        - math.sin(math.radians(lat)) * math.cos(math.radians(lat2)) * math.
↪cos(dLon))
    bearing = (math.degrees(math.atan2(y, x)) + 360) % 360
    return (bearing + 180) % 360 # otočíme o 180°

# --- 2) Barevná škála podle JC -----
jc_vals = df['JC byty [Kč/m2]'].dropna()
vmin, vmax = jc_vals.min(), jc_vals.max()
if vmin == vmax:
    vmax = vmin + 1
colormap = cm.LinearColormap(['green', 'yellow', 'red'], vmin=vmin, vmax=vmax)
colormap.caption = 'JC byty [Kč/m²]'

# --- 3) Střed mapy -----
center = [df['LAT'].mean(), df['LON'].mean()]

# --- 4) Inicializace Folium mapy -----
m = folium.Map(location=center, zoom_start=12, control_scale=True)
folium.TileLayer('OpenStreetMap', name='OpenStreetMap').add_to(m)
folium.TileLayer('Esri.WorldImagery', name='Ortofotomap').add_to(m)
mc = folium.plugins.MarkerCluster(name='Byty').add_to(m)

# --- 5) Smyčka pro body s popupem -----
for _, row in df.dropna(subset=['LAT', 'LON']).iterrows():
    lat, lon = row['LAT'], row['LON']
    jc = row['JC byty [Kč/m2]']
    color = colormap(jc)
    cislo = row.get('Číslo vkladu', row.name)
    datum = pd.to_datetime(row['Datum podání']).date()
    vzd = f"{row['Vzdálenost [km]']:.2f} km"
    plocha = f"{row['Plocha [m2]']:.2f} m²"
    jc_txt = f"{jc:,.0f}".replace(',', ' ') + " Kč/m²"
    cena = round(jc * row['Plocha [m2]'])
    cena_txt = f"{int(cena):,d} Kč"
    heading = compute_heading(lat, lon)

```

```

# HTML kontejner bez scrollování, menší rozměry
html = f"""
<div style="
    width: 400px;
    height: 450px;
    overflow: hidden;
    font-size: 14px;
    line-height: 1.4;
">
    <iframe
        width="400" height="300"
        frameborder="0" style="border:0; display:block;"
        src="https://www.google.com/maps/embed/v1/streetview?key={API_KEY}
            &location={lat},{lon}
            &heading={heading:.1f}
            &pitch=0
            &fov=90"
        allowfullscreen>
    </iframe>
    <div style="
        padding: 8px;
        box-sizing: border-box;
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    ">
        <b>Číslo vkladu:</b> {cislo}<br>
        <b>Datum podání:</b> {datum}<br>
        <b>Vzdálenost:</b> {vzd}<br>
        <b>Plocha:</b> {plocha}<br>
        <b>JC:</b> {jc_txt}<br>
        <b>Cena:</b> {cena_txt}
    </div>
</div>
"""

# Vložíme do IFrame přesně těchto rozměrů a přiřadíme Popup
iframe = folium.IFrame(html=html, width=400, height=450)
popup = folium.Popup(iframe, max_width=400)

folium.CircleMarker(
    location=(lat, lon),
    radius=6,
    color=color,
    fill=True, fill_color=color, fill_opacity=0.9,
    popup=popup
).add_to(mc)

```



```
# --- 6) Legenda a přepínač vrstev -----
colormap.add_to(m)
folium.LayerControl().add_to(m)

# --- 7) Vykreslení mapy -----
display(m)
```

<folium.folium.Map at 0x171bf1888f0>

[69]: # MAPA 2 - VŠECHNY BYTY SOUČASNĚ/ Ceny / Street View -

```
import folium
import math
import branca.colormap as cm
from IPython.display import display
import pandas as pd

# --- 0) Váš Google Maps Embed API KEY -----
#API_KEY = "AIzaSyCY2ClgEZNV3HD7q0qYqTsPtαCM0h1RIsM"

# --- 1) Funkce pro výpočet headingu a jeho otočení o 180° -----
def compute_heading(lat, lon, delta=0.0001):
    lat2, lon2 = lat + delta, lon
    dLon = math.radians(lon2 - lon)
    y = math.sin(dLon) * math.cos(math.radians(lat2))
    x = (math.cos(math.radians(lat)) * math.sin(math.radians(lat2))
        - math.sin(math.radians(lat)) * math.cos(math.radians(lat2)) * math.
↪cos(dLon))
    bearing = (math.degrees(math.atan2(y, x)) + 360) % 360
    return (bearing + 180) % 360

# --- 2) Barevná škála podle JC -----
jc_vals = df['JC byty [Kč/m2]'].dropna()
vmin, vmax = jc_vals.min(), jc_vals.max()
if vmin == vmax:
    vmax = vmin + 1
colormap = cm.LinearColormap(['green','yellow','red'], vmin=vmin, vmax=vmax)
colormap.caption = 'JC byty [Kč/m²]'

# --- 3) Střed mapy -----
center = [df['LAT'].mean(), df['LON'].mean()]

# --- 4) Inicializace Folium mapy -----
m = folium.Map(location=center, zoom_start=12, control_scale=True)
folium.TileLayer('OpenStreetMap', name='OpenStreetMap').add_to(m)
folium.TileLayer('Esri.WorldImagery', name='Ortofotomap').add_to(m)
```

```

# --- 5) Smyčka pro body s popupem -----
for _, row in df.dropna(subset=['LAT', 'LON']).iterrows():
    lat, lon = row['LAT'], row['LON']
    jc      = row['JC byty [Kč/m2]']
    color   = colormap(jc)
    cislo   = row.get('Číslo vkladu', row.name)
    datum   = pd.to_datetime(row['Datum podání']).date()
    vzd     = f"{row['Vzdálenost [km]']:.2f} km"
    plocha  = f"{row['Plocha [m2]']:.2f} m²"
    jc_txt  = f"{jc:,.0f}".replace(',', ' ') + " Kč/m²"
    cena    = round(jc * row['Plocha [m2]'])
    cena_txt = f"{int(cena):,d} Kč"
    heading = compute_heading(lat, lon)

    html = f"""
    <div style="
        width: 400px;
        height: 450px;
        overflow: hidden;
        font-size: 14px;
        line-height: 1.4;
    ">
        <iframe
            width="400" height="300"
            frameborder="0" style="border:0; display:block;"
            src="https://www.google.com/maps/embed/v1/streetview?key={API_KEY}
                &location={lat},{lon}
                &heading={heading:.1f}
                &pitch=0
                &fov=90"
            allowfullscreen>
        </iframe>
        <div style="
            padding: 8px;
            box-sizing: border-box;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        ">
            <b>Číslo vkladu:</b> {cislo}<br>
            <b>Datum podání:</b> {datum}<br>
            <b>Vzdálenost:</b> {vzd}<br>
            <b>Plocha:</b> {plocha}<br>
            <b>JC:</b> {jc_txt}<br>
            <b>Cena:</b> {cena_txt}
        </div>
    </div>
    """

```

```

iframe = folium.IFrame(html=html, width=400, height=450)
popup = folium.Popup(iframe, max_width=400)

# Přidáváme přímo na mapu, bez clusteru
folium.CircleMarker(
    location=(lat, lon),
    radius=6,
    color=color,
    fill=True, fill_color=color, fill_opacity=0.9,
    popup=popup
).add_to(m)

# --- 6) Legenda a přepínač vrstev -----
colormap.add_to(m)
folium.LayerControl().add_to(m)

# --- 7) Vykreslení mapy -----
display(m)

```

<folium.folium.Map at 0x172050d6690>

```

[70]: # Základní popisná statistika

# stačí jen využít již načtené df
import matplotlib.pyplot as plt

# vyberete si sloupce
cols = ["JC byty [Kč/m2]", "Plocha [m2]", "Vzdálenost [km]", "Datum podání"]

# příprava

df["Datum podání"] = pd.to_datetime(df["Datum podání"], dayfirst=True,
    ↪errors="coerce")
for c in ["JC byty [Kč/m2]", "Plocha [m2]", "Vzdálenost [km]"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# popisné statistiky
desc = df[["JC byty [Kč/m2]", "Plocha [m2]", "Vzdálenost [km]"]].describe().
    ↪round(2).T
print(desc)

# datumové statistiky
cnt = df["Datum podání"].count()
mn = df["Datum podání"].min().strftime("%Y-%m-%d")
mx = df["Datum podání"].max().strftime("%Y-%m-%d")

```

```
print(f"\nDatum podání - count: {cnt}, min: {mn}, max: {mx}")
```

	count	mean	std	min	25%	50% \
JC byty [Kč/m2]	1679.0	127855.32	30134.17	70547.00	102464.00	126214.00
Plocha [m2]	1679.0	61.52	23.81	21.00	43.00	57.50
Vzdálenost [km]	1679.0	5.38	2.38	0.05	4.36	5.85

	75%	max
JC byty [Kč/m2]	147034.50	235732.00
Plocha [m2]	74.00	185.00
Vzdálenost [km]	7.32	8.21

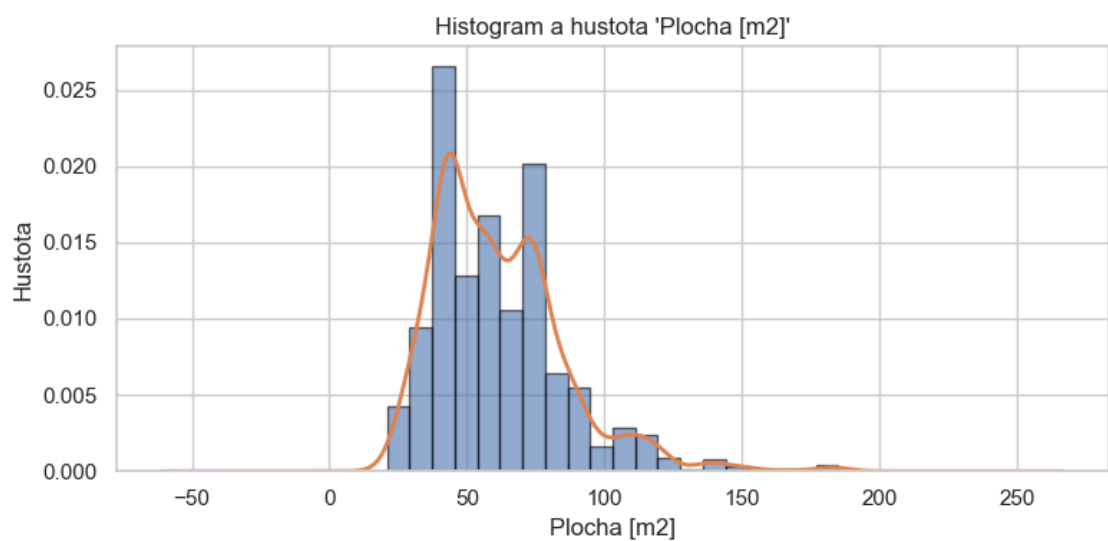
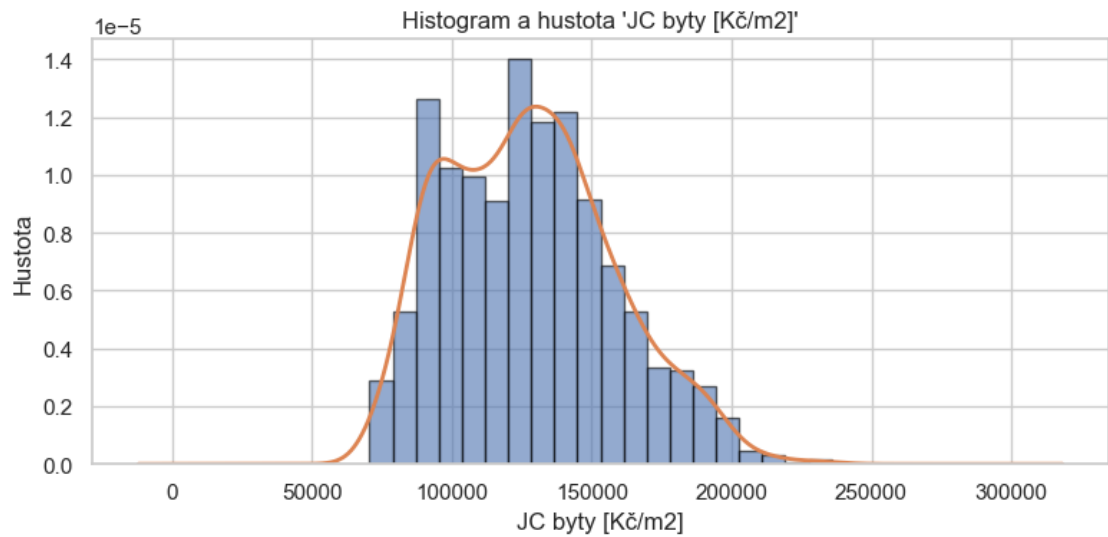
Datum podání - count: 1679, min: 2021-08-04, max: 2025-04-04

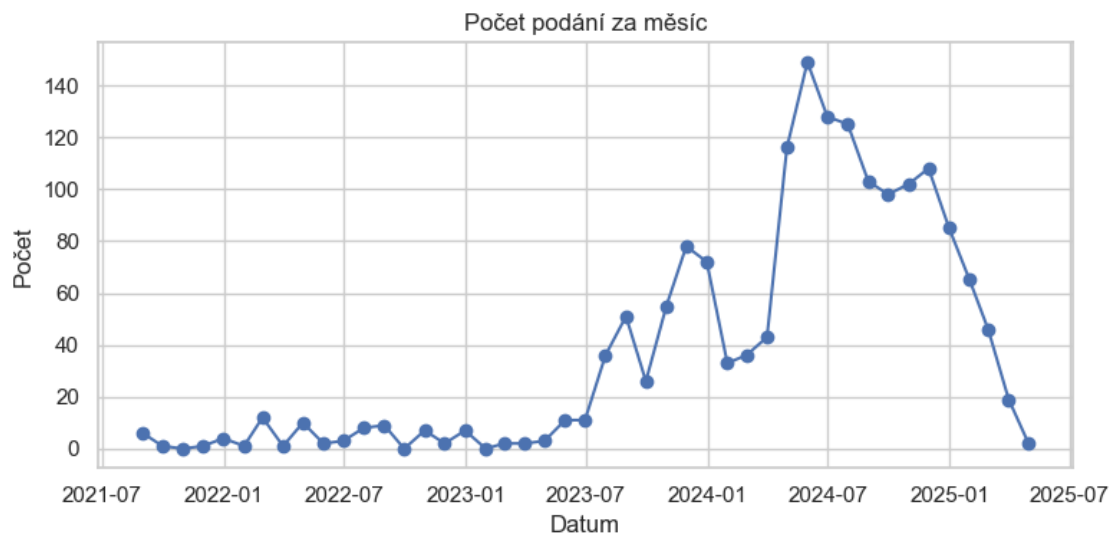
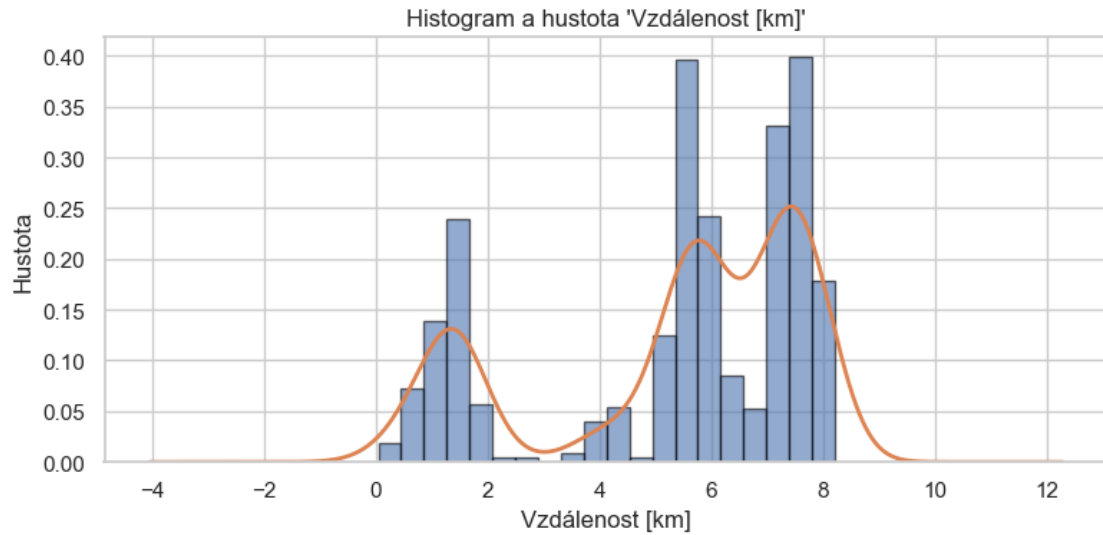
[71]: # Histogramy

```
import numpy as np

# histogramy s hustotní křivkou
for c in ["JC byty [Kč/m2]", "Plocha [m2]", "Vzdálenost [km]"]:
    fig, ax = plt.subplots(figsize=(8,4))
    ax.hist(df[c].dropna(), bins=20, density=True, alpha=0.6, edgecolor='black')
    df[c].dropna().plot.kde(ax=ax, linewidth=2)
    ax.set_title(f"Histogram a hustota '{c}'")
    ax.set_xlabel(c)
    ax.set_ylabel("Hustota")
    plt.tight_layout()
    plt.show()

# časová řada počtu podání za měsíc
df_monthly = df.set_index("Datum podání").resample("M").size()
fig, ax = plt.subplots(figsize=(8,4))
ax.plot(df_monthly.index, df_monthly.values, marker='o')
ax.set_title("Počet podání za měsíc")
ax.set_xlabel("Datum")
ax.set_ylabel("Počet")
plt.tight_layout()
plt.show()
```





```
[72]: # Vztahy jednotlivých parametrů a JC byty [Kč/m2]

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, AutoDateLocator

# Předpoklad: df již načteno dříve a obsahuje sloupce
# "Plocha [m2]", "JC byty [Kč/m2]", "Vzdálenost [km]", "Datum podání"
```

```

# Styly
palette = plt.get_cmap("RdYlGn")
point_size = 60
alpha = 0.7
edgecol = 'k'
sns.set(style="whitegrid")

def plot_scatter(x, y, xlabel, ylabel, title, degree=1, show_poly=True):
    xi, yi = x.values, y.values
    mask = np.isfinite(xi) & np.isfinite(yi)
    xi, yi = xi[mask], yi[mask]
    if len(xi) < 2:
        print(f"Není dost dat pro '{title}'")
        return

    # Lineární regrese
    a, b = np.polyfit(xi, yi, 1)
    r2_lin = np.corrcoef(xi, yi)[0,1]**2

    # Polynomická regrese
    poly = None
    if show_poly and degree > 1 and len(xi) > degree:
        coef_poly = np.polyfit(xi, yi, degree)
        poly = np.poly1d(coef_poly)
        y_pred = poly(xi)
        ss_tot = ((yi - yi.mean())**2).sum()
        ss_res = ((yi - y_pred)**2).sum()
        r2_poly = 1 - ss_res/ss_tot

    xr = np.linspace(xi.min(), xi.max(), 200)
    fig, ax = plt.subplots(figsize=(14,7), dpi=100)
    sc = ax.scatter(xi, yi, c=yi, cmap=palette, s=point_size,
                    edgecolor=edgecol, alpha=alpha)

    # Vykreslit lineární křivku
    ax.plot(xr, a*xr + b, color="red",
            label=f"Lineární:  $y={a:.2f}x+{b:.2f}$ ,  $R^2={r2\_lin:.3f}$ ")

    # Vykreslit polynomickou křivku
    if poly is not None:
        ax.plot(xr, poly(xr), color="blue", linestyle="--",
                label=f"Polynomická (deg {degree}):  $R^2={r2\_poly:.3f}$ ")

    # Kvartily
    Q1, Q2, Q3 = np.percentile(yi, [25,50,75])
    for q, col, pct in [(Q1, "purple", "25%"), (Q2, "orange", "50%"),
                        (Q3, "green", "75%")]:

```

```

        ax.axhline(q, color=col, linestyle=":", linewidth=1.5,
                    label=f"Q{pct} = {q:.2f}")

    # Colorbar a popisky
    cbar = fig.colorbar(sc, ax=ax)
    cbar.set_label(ylabel)
    ax.set_title(title, fontsize=16, fontweight="bold")
    ax.set_xlabel(xlabel, fontsize=14)
    ax.set_ylabel(ylabel, fontsize=14)
    ax.legend(loc="best", fontsize=10)
    ax.grid(linestyle="--", linewidth=0.5)
    plt.tight_layout()
    plt.show()

def plot_time_scatter(dates, values, ylabel, title, degree=1):
    mask = dates.notna() & np.isfinite(values)
    dates = dates[mask]
    y = values[mask].astype(float)
    if len(dates) < 2:
        print(f"Není dost dat pro '{title}'")
        return

    # Převod na ordinal a regrese
    x_ord = dates.map(pd.Timestamp.toordinal).values
    a, b = np.polyfit(x_ord, y, 1)
    r2_lin = np.corrcoef(x_ord, y)[0,1]**2

    poly = None
    if degree > 1 and len(dates) > degree:
        coef_poly = np.polyfit(x_ord, y, degree)
        poly = np.poly1d(coef_poly)
        y_pred = poly(x_ord)
        ss_tot = ((y - y.mean())**2).sum()
        ss_res = ((y - y_pred)**2).sum()
        r2_poly = 1 - ss_res/ss_tot

    dates_range = pd.date_range(dates.min(), dates.max(), periods=200)
    x_ord_range = dates_range.map(pd.Timestamp.toordinal).values

    fig, ax = plt.subplots(figsize=(14,7), dpi=100)
    sc = ax.scatter(dates, y, c=y, cmap=palette, s=point_size,
                    edgecolor=edgecol, alpha=alpha)

    # Lineární trend
    ax.plot(dates_range, a*x_ord_range + b, color="red",
            label=f"Lineární:  $R^2 = {r2_lin:.3f}$ ")

```



```

# Polynomický trend
if poly is not None:
    ax.plot(dates_range, poly(x_ord_range), color="blue", linestyle="--",
            label=f"Polynomická (deg {degree}):  $R^2$ ={r2_poly:.3f}")

# Kvartily
Q1, Q2, Q3 = np.percentile(y, [25,50,75])
for q, col, pct in [(Q1,"purple","25%"), (Q2,"orange","50%"),
                    (Q3,"green","75%")]:
    ax.axhline(q, color=col, linestyle=":", linewidth=1.5,
               label=f"Q{pct} = {q:.2f}")

# Formát osy X
ax.xaxis.set_major_locator(AutoDateLocator())
ax.xaxis.set_major_formatter(DateFormatter("%Y-%m"))
fig.autofmt_xdate()

# Colorbar a popisky
cbar = fig.colorbar(sc, ax=ax)
cbar.set_label(ylabel)
ax.set_title(title, fontsize=16, fontweight="bold")
ax.set_xlabel("Datum podání", fontsize=14)
ax.set_ylabel(ylabel, fontsize=14)
ax.legend(loc="best", fontsize=10)
ax.grid(linestyle="--", linewidth=0.5)
plt.tight_layout()
plt.show()

# Vykreslení všech tří grafů
plot_scatter(
    df["Plocha [m2]"], df["JC byty [Kč/m2]"],
    xlabel="Plocha [m2]", ylabel="JC byty [Kč/m2]",
    title="Plocha [m2] vs. JC byty [Kč/m2]",
    degree=6, show_poly=True
)

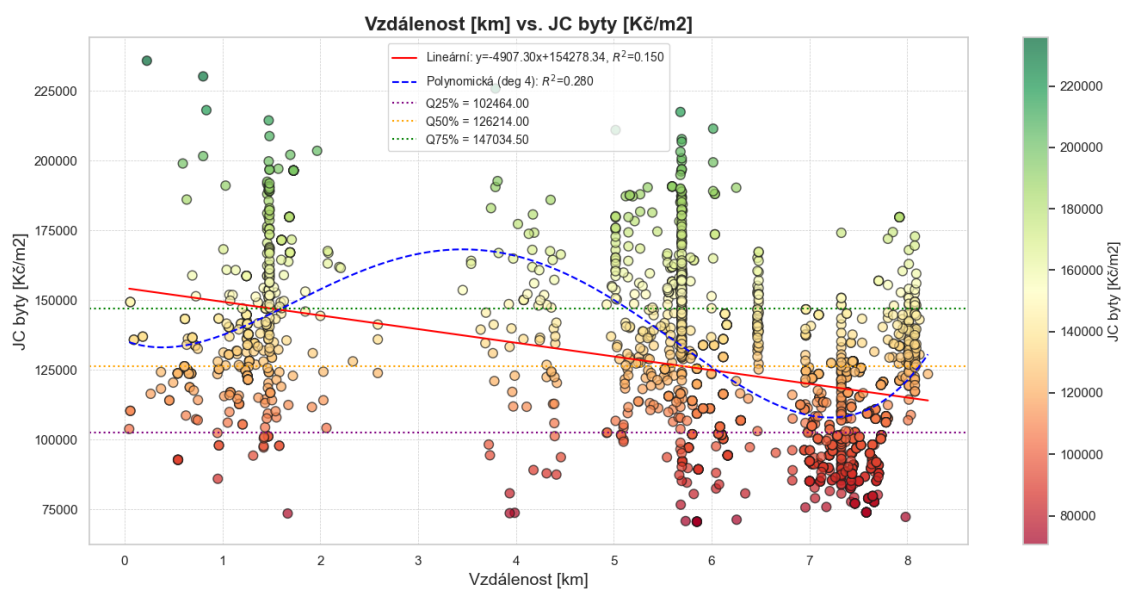
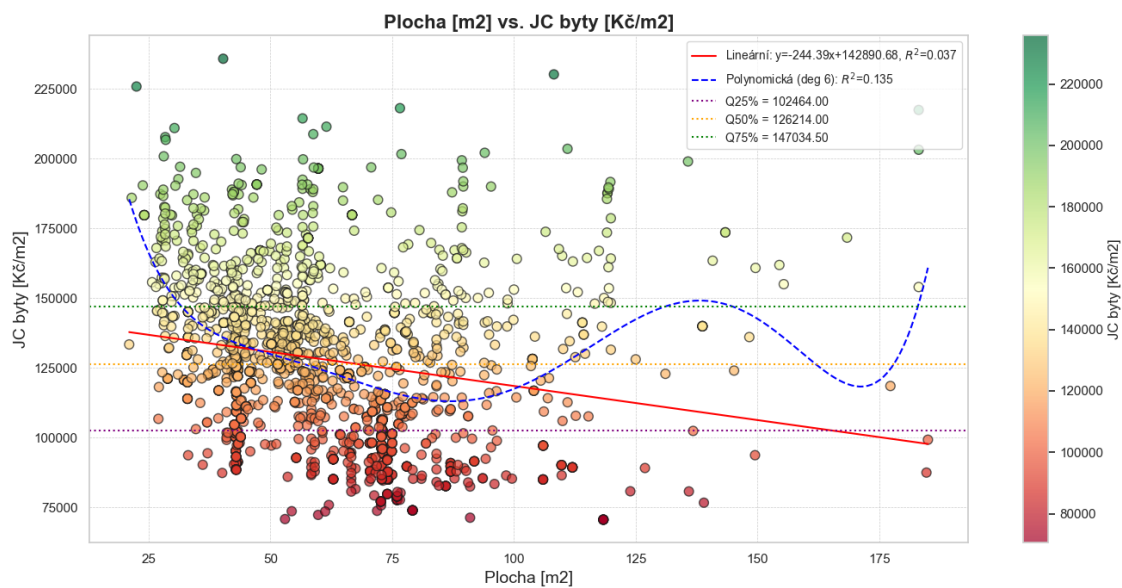
plot_scatter(
    df["Vzdálenost [km]"], df["JC byty [Kč/m2]"],
    xlabel="Vzdálenost [km]", ylabel="JC byty [Kč/m2]",
    title="Vzdálenost [km] vs. JC byty [Kč/m2]",
    degree=4, show_poly=True
)

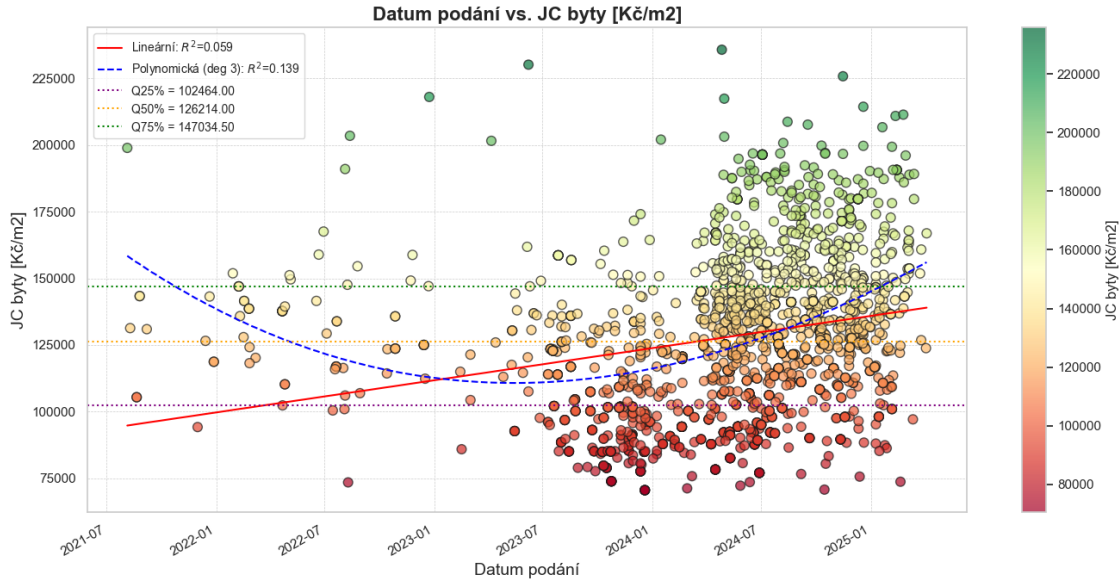
plot_time_scatter(
    df["Datum podání"], df["JC byty [Kč/m2]"],
    ylabel="JC byty [Kč/m2]",
    title="Datum podání vs. JC byty [Kč/m2]",

```

degree=3

)





[73]: # Metoda Lokte a určení optimálního K ... počtu shluků pro shlukovou analýzu

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# --- Předpoklad: df je již načtené z Excelu ---

# 1) Vybereme numerické sloupce
num_cols = ["Plocha [m2]", "JC byty [Kč/m2]", "Vzdálenost [km]"]

# 2) Explicitní převod na čísla
df[num_cols] = df[num_cols].apply(pd.to_numeric, errors="coerce")

# 3) Imputace chybějících hodnot průměrem
imputer = SimpleImputer(strategy="mean")
X_imputed = imputer.fit_transform(df[num_cols])

# 4) Škálování
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# 5) Výpočet SSE pro K = 1...10
```

```

sse = []
k_range = np.arange(1, 11)
for k in k_range:
    km = KMeans(n_clusters=int(k), random_state=42)
    km.fit(X_scaled)
    sse.append(km.inertia_)

# 6) Detekce "elbow" metodou největší vzdálenosti od přímky mezi krajními body
# Souřadnice přímky
x1, y1 = k_range[0], sse[0]
x2, y2 = k_range[-1], sse[-1]
# Vzdálenosti bodů
distances = []
for x0, y0 in zip(k_range, sse):
    # vzdálenost bodu (x0,y0) od přímky (x1,y1)-(x2,y2)
    num = abs((y2 - y1)*x0 - (x2 - x1)*y0 + x2*y1 - y2*x1)
    den = np.sqrt((y2 - y1)**2 + (x2 - x1)**2)
    distances.append(num/den)
# optimal K
optimal_idx = np.argmax(distances)
optimal_k = k_range[optimal_idx]

# 7) Výpis zdůvodnění
print(f"Optimální počet shluků K podle metody lokte: {optimal_k}")
print(f"Maximální vzdálenost od přímky: {distances[optimal_idx]:.2f}")
print("Toto K značí místo, kde se při navyšování K nejméně zlepšuje SSE, tedy
↳ ideální bod lokte.\n")

# 8) Graf metody lokte s vyznačením optimálního K
plt.figure(figsize=(8, 5), dpi=100)
plt.plot(k_range, sse, marker='o', linestyle='-')
plt.axvline(optimal_k, color='red', linestyle='--', label=f'Elbow
↳ (K={optimal_k})')
plt.xticks(k_range)
plt.xlabel('Počet shluků $K$', fontsize=12)
plt.ylabel('SSE (suma čtverců vzdáleností)', fontsize=12)
plt.title('Metoda lokte pro určení optimálního $K$', fontsize=14,
↳ fontweight='bold')
plt.grid(linestyle='--', linewidth=0.5, alpha=0.7)
plt.legend()
plt.tight_layout()
plt.show()

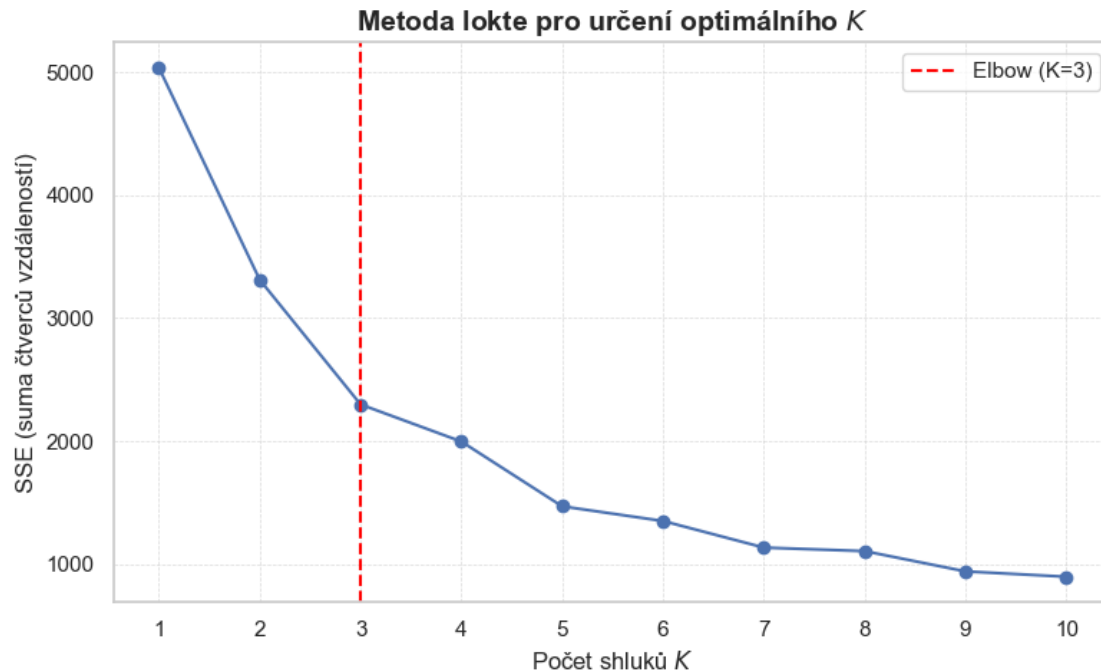
```

Optimální počet shluků K podle metody lokte: 3

Maximální vzdálenost od přímky: 3.95

Toto K značí místo, kde se při navyšování K nejméně zlepšuje SSE, tedy ideální

bod lokte.



```
[74]: # Shluková analýza

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from scipy.stats import linregress
from matplotlib.lines import Line2D

# --- 1) Použijte váš existující DF ---
# Zde nahraďte YOUR_DF názvem proměnné, kterou v notebooku používáte
df_clusters = df.copy()

# --- 2) Připravíme časovou proměnnou ---
df_clusters['Datum podání'] = pd.to_datetime(
    df_clusters['Datum podání'], dayfirst=True, errors='coerce'
)
df_clusters['Datum podání (dny)'] = (
    df_clusters['Datum podání'] - df_clusters['Datum podání'].min()
)
```

```

).dt.days

# --- 3) Definice vstupních sloupců --
features = [
    'LAT', 'LON',
    'JC byty [Kč/m2]', 'Plocha [m2]',
    'Vzdálenost [km]', 'Datum podání (dny)'
]

# --- 4) Imputace chybějících hodnot průměrem --
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(df_clusters[features])

# --- 5) Standardizace --
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# --- 6) K-Means clustering --
# optimal_k musí být předem definováno (např. z metody lokte)
kmeans = KMeans(
    n_clusters=optimal_k,
    random_state=42,
    n_init=50,
    algorithm='elkan'
)
df_clusters['Cluster'] = kmeans.fit_predict(X_scaled)

# --- 7) Výsledek: df_clusters obsahuje nový sloupec 'Cluster' --

# --- 8) Ukázkové vykreslení: mapa všech clusterů --
palette = sns.color_palette(
    'viridis',
    n_colors=len(df_clusters['Cluster'].unique())
)

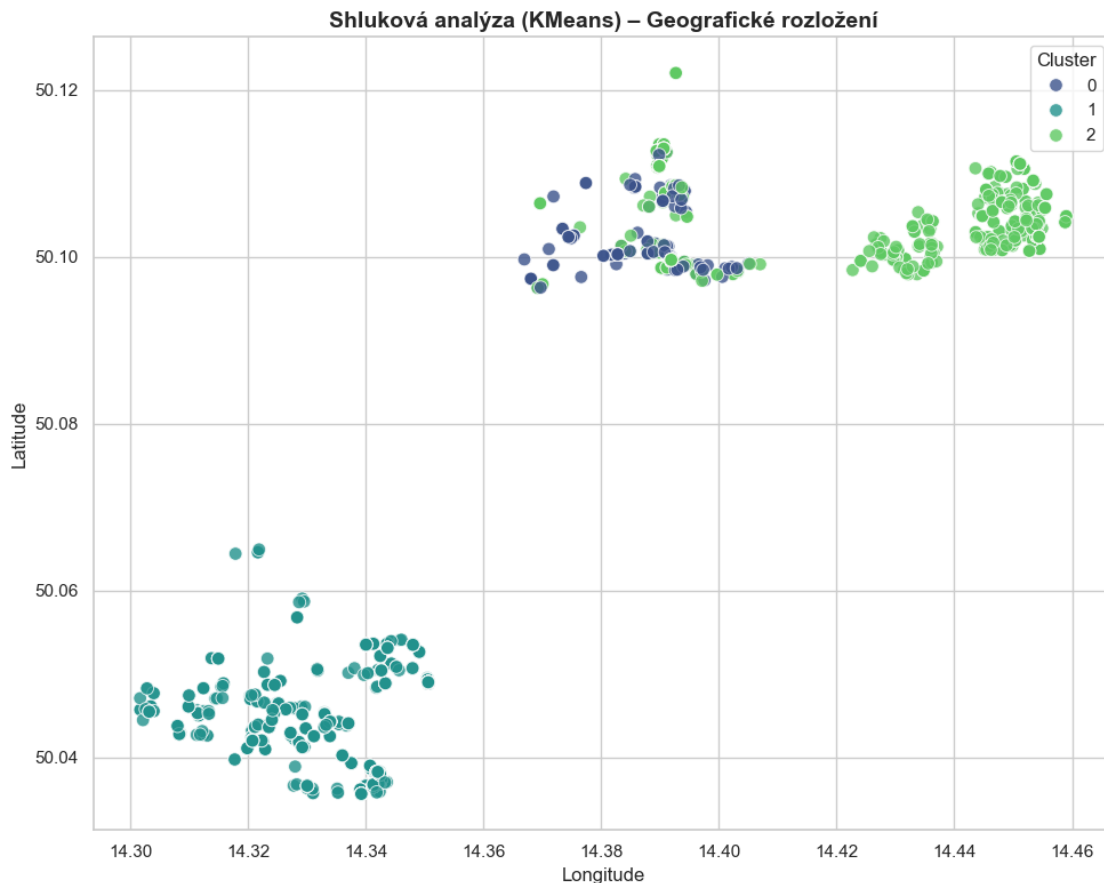
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x='LON', y='LAT',
    data=df_clusters,
    hue='Cluster', palette=palette,
    s=70, alpha=0.8
)

plt.title(
    'Shluková analýza (KMeans) - Geografické rozložení',
    fontsize=14, fontweight='bold'
)

plt.xlabel('Longitude')
plt.ylabel('Latitude')

```

```
plt.legend(title='Cluster', loc='upper right')
plt.tight_layout()
plt.show()
```



```
[75]: # Geografické znázornění shluků

import folium
from folium.plugins import MarkerCluster
from branca.element import Template, MacroElement
import seaborn as sns
from matplotlib import colors as mcolors
from IPython import get_ipython

# Získání DataFrame df_clusters z paměti
ns = get_ipython().user_ns
df_clusters = ns.get('df_clusters')
if df_clusters is None:
    raise ValueError("DataFrame 'df_clusters' není k dispozici. Spustte buňku s ↵
    ↵clusteringem.")
```

```

# Připravíme barvy pro jednotlivé clustery
clusters = sorted(df_clusters['Cluster'].unique())
palette = sns.color_palette('viridis', n_colors=len(clusters))
hex_colors = [mcolors.to_hex(c) for c in palette]

# Odfiltrujeme řádky bez platných souřadnic LAT/LON
df_map = df_clusters.dropna(subset=['LAT', 'LON']).copy()

# Určíme střed mapy pro počáteční pozici
center = [df_map['LAT'].mean(), df_map['LON'].mean()]

# Sestavíme HTML pro legendu (vlevo dole)
legend_items = "".join(
    f'<i style="background:{color};width:10px;height:10px;display:inline-block;
    ↪margin-right:5px;"></i>'
    f'Cluster {idx}<br>'
    for idx, color in zip(clusters, hex_colors)
)
legend_html = f"""
<div style="
    position: fixed;
    bottom: 50px;
    left: 50px;
    width: 140px;
    background-color: white;
    border:2px solid grey;
    z-index:9999;
    font-size:14px;
    padding: 10px;
">
    <b>Legenda clusterů</b><br>
    {legend_items}
</div>
"""

# Vytvoříme MacroElement pro legendu
template = Template(f"{{% macro html(this, kwargs) %}}{legend_html}{{% endmacro %}}")
legend = MacroElement()
legend._template = template

# --- 1) Mapa s MarkerCluster ---
m_cluster = folium.Map(location=center, zoom_start=12, control_scale=True)
# Přidáme dvě vrstvy podkladových map
folium.TileLayer('OpenStreetMap', name='OpenStreetMap').add_to(m_cluster)
folium.TileLayer('Esri.WorldImagery', name='Ortofotomap').add_to(m_cluster)

```



```

# MarkerCluster pro lepší výkon při velkém množství bodů
marker_cluster = MarkerCluster(name='Clustered').add_to(m_cluster)
for _, row in df_map.iterrows():
    folium.CircleMarker(
        location=(row['LAT'], row['LON']),
        radius=5,
        color=hex_colors[int(row['Cluster'])],
        fill=True,
        fill_color=hex_colors[int(row['Cluster'])],
        fill_opacity=0.8
    ).add_to(marker_cluster)

# Přidáme legendu a ovládání vrstev
m_cluster.get_root().add_child(legend)
folium.LayerControl().add_to(m_cluster)

# Zobrazíme mapu s clusterovanými markery
from IPython.display import display
display(m_cluster)

# --- 2) Mapa se všemi body barevně dle clusteru ---
m_points = folium.Map(location=center, zoom_start=12, control_scale=True)
folium.TileLayer('OpenStreetMap', name='OpenStreetMap').add_to(m_points)
folium.TileLayer('Esri.WorldImagery', name='Ortofotomap').add_to(m_points)

for _, row in df_map.iterrows():
    folium.CircleMarker(
        location=(row['LAT'], row['LON']),
        radius=5,
        color=hex_colors[int(row['Cluster'])],
        fill=True,
        fill_color=hex_colors[int(row['Cluster'])],
        fill_opacity=0.9
    ).add_to(m_points)

# Přidáme legendu a ovládání vrstev
m_points.get_root().add_child(legend)
folium.LayerControl().add_to(m_points)

# Zobrazíme mapu se všemi body
display(m_points)

```

<folium.folium.Map at 0x1718aff5b50>

<folium.folium.Map at 0x171ffdd9220>

[76]: # Shluková a vztahová analýza

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from scipy.stats import linregress

import warnings
warnings.filterwarnings("ignore", message="The palette list has more values")

# --- Předpoklad: df_clusters je již načtené a obsahuje sloupce:
# 'LAT', 'LON', 'JC byty [Kč/m2]', 'Plocha [m2]', 'Vzdálenost [km]', 'Datum_
↳ podání'

# 1) Převod a příprava časové proměnné
df_clusters['Datum podání'] = pd.to_datetime(
    df_clusters['Datum podání'], dayfirst=True, errors='coerce'
)
df_clusters['Datum podání (dny)'] = (
    df_clusters['Datum podání'] - df_clusters['Datum podání'].min()
).dt.days

# 2) Definice feature setu
features = [
    'LAT', 'LON',
    'JC byty [Kč/m2]', 'Plocha [m2]',
    'Vzdálenost [km]', 'Datum podání (dny)'
]

# 3) Imputace chybějících hodnot
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(df_clusters[features])

# 4) Standardizace a K-means
X_scaled = StandardScaler().fit_transform(X_imputed)
kmeans = KMeans(
    n_clusters=optimal_k,
    random_state=42,
    n_init=50,
    algorithm='elkan'
).fit(X_scaled)
df_clusters['Cluster'] = kmeans.labels_

```

```

# 5) Paleta pro konzistentní barvy
clusters = sorted(df_clusters['Cluster'].unique())
palette = sns.color_palette('viridis', n_colors=len(clusters))

# 6) Příprava handle pro kvartily (bez samostatného vykreslení)
quartile_handles = [
    plt.Line2D([0],[0], color='green',   linestyle='--', label='Q1'),
    plt.Line2D([0],[0], color='orange',  linestyle='-',  label='Median'),
    plt.Line2D([0],[0], color='red',      linestyle='--', label='Q3'),
]

# --- 1) Celková mapa clusterů (zakomentováno, pokud nepotřebujete) ---
# plt.figure(figsize=(10,8))
# sns.scatterplot(x='LON', y='LAT', data=df_clusters,
#                 hue='Cluster', palette=palette,
#                 s=70, alpha=0.8)
# plt.title('Geografické rozložení všech clusterů', fontsize=14,
#           fontweight='bold')
# plt.xlabel('Longitude')
# plt.ylabel('Latitude')
# plt.legend(title='Cluster', loc='upper right')
# plt.tight_layout()
# plt.show()

# --- 2) Detailní per-cluster grafy s  $R^2$  a tabulkami ---
fig, axs = plt.subplots(len(clusters), 4, figsize=(24, 6*len(clusters)))
if len(clusters) == 1:
    axs = axs.reshape(1, -1)

pairs = [
    ('Plocha [m2]', 'JC byty [Kč/m2]'),
    ('Vzdálenost [km]', 'JC byty [Kč/m2]'),
    ('Datum podání (dny)', 'JC byty [Kč/m2]'),
    ('LON', 'LAT')
]

for i, cluster in enumerate(clusters):
    cd = df_clusters[df_clusters['Cluster']==cluster]
    q1 = cd[features].quantile(0.25)
    q2 = cd[features].quantile(0.50)
    q3 = cd[features].quantile(0.75)

    for j,(x,y) in enumerate(pairs):
        ax = axs[i,j]

        if j < 3:
            # pozadí všech bodů

```

```

sns.scatterplot(x=x, y=y, data=df_clusters, hue='Cluster',
↪palette=palette,
                    ax=ax, legend=False, alpha=0.3, s=40)
# body aktuálního clusteru
sns.scatterplot(x=x, y=y, data=cd, hue='Cluster', palette=palette,
                    ax=ax, legend=False, s=50, edgecolor='k')
# lineární regrese + R²
slope, intercept, r_value, p_value, std_err = linregress(cd[x],
↪cd[y])

sns.regplot(x=x, y=y, data=cd, ax=ax, scatter=False,
            color='red', line_kws={'linewidth':1.5})
ax.text(0.05, 0.85, f"$R^2$={r_value**2:.3f}",
        transform=ax.transAxes, color='red',
        fontsize=10, backgroundcolor='white')
# kvartilové čáry
for label, quart in zip(['Q1', 'Median', 'Q3'], [q1, q2, q3]):
    style = '--' if label != 'Median' else '-'
    color = {'Q1': 'green', 'Median': 'orange', 'Q3': 'red'}[label]
    ax.axvline(quart[x], linestyle=style, color=color)
    ax.axhline(quart[y], linestyle=style, color=color)
else:
    # geografický panel
    sns.scatterplot(x='LON', y='LAT', data=df_clusters, hue='Cluster',
↪palette=palette,
                    ax=ax, legend=False, alpha=0.3, s=40)
    sns.scatterplot(x='LON', y='LAT', data=cd, hue='Cluster',
↪palette=palette,
                    ax=ax, legend=False, s=50, edgecolor='k')
    ax.set_xlabel('Longitude')
    ax.set_ylabel('Latitude')

    ax.set_title(f"Cluster {cluster} - {x} vs {y}", pad=10)

# tabulka mean / median pod grafy
cell_text = [[f"{cd[col].mean():.1f}", f"{cd[col].median():.1f}"] for col
↪in features]
axs[i,0].table(
    cellText=cell_text,
    rowLabels=features,
    colLabels=['Mean', 'Median'],
    loc='bottom',
    bbox=[0.0, -1.0, 1.0, 0.7],
    colWidths=[0.2, 0.2],
    cellLoc='center'
)

```

7) Přidáme legendu kvartilů společně s grafy

```
fig.legend(handles=quartile_handles, loc='upper center', ncol=3,
           title='Kvartily')
plt.subplots_adjust(hspace=1.2, wspace=0.4, bottom=0.15, top=0.93)
plt.show()
```



[77]: # Porovnání lineární regrese a Random Forest pro predikci JC byt

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
```

```

from sklearn.ensemble      import RandomForestRegressor
from sklearn.pipeline      import Pipeline
from sklearn.metrics       import r2_score, mean_squared_error

# 0) Sestavím dataset bez NaN v X ani v y
features = ['LAT', 'LON', 'Plocha [m2]', 'Vzdálenost [km]', 'Datum podání (dny)']
target   = 'JC byty [Kč/m2]'
df_model = df_clusters.dropna(subset=features + [target]).reset_index(drop=True)

X = df_model[features]
y = df_model[target].values

# 1) Rozdělení na trénink/test (20 %)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 2) Pipeline pro oba modely (imputace+škálování+model)
pipe_lin = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('model', LinearRegression())
])
pipe_rf = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('model', RandomForestRegressor(n_estimators=200, random_state=42))
])

# 3) Trénink
pipe_lin.fit(X_train, y_train)
pipe_rf.fit(X_train, y_train)

# 4) Predikce a metriky
y_pred_lin = pipe_lin.predict(X_test)
y_pred_rf  = pipe_rf.predict(X_test)

r2_lin     = r2_score(y_test, y_pred_lin)
rmse_lin   = np.sqrt(mean_squared_error(y_test, y_pred_lin))
r2_rf      = r2_score(y_test, y_pred_rf)
rmse_rf    = np.sqrt(mean_squared_error(y_test, y_pred_rf))

# 5) Křížová validace  $R^2$ 
cv_lin = cross_val_score(pipe_lin, X, y, cv=5, scoring='r2')
cv_rf  = cross_val_score(pipe_rf, X, y, cv=5, scoring='r2')

# 6) Sestav DataFrame pro koeficienty a feature importances

```

```

coef_df = pd.DataFrame({
    'feature': features,
    'coef': pipe_lin.named_steps['model'].coef_
}).set_index('feature').sort_values('coef', key=lambda s: s.abs())

imp_df = pd.DataFrame({
    'feature': features,
    'importance': pipe_rf.named_steps['model'].feature_importances_
}).set_index('feature').sort_values('importance')

# 7) Vykreslení všech grafů
sns.set_style("whitegrid")
fig = plt.figure(constrained_layout=True, figsize=(16,12))
gs = fig.add_gridspec(3, 2, height_ratios=[1,1,0.6])

# 7.1 Scatterry
ax0 = fig.add_subplot(gs[0,0])
sns.scatterplot(x=y_test, y=y_pred_lin, ax=ax0,
                color=sns.color_palette("viridis")[0], alpha=0.7)
ax0.plot([y.min(),y.max()], [y.min(),y.max()], 'r--')
ax0.set_title(f"Lineární regrese ( $R^2$ ={{r2_lin:.3f}}, RMSE={{rmse_lin:,.0f}})")
ax0.set_xlabel("Skutečná cena")
ax0.set_ylabel("Predikovaná cena")

ax1 = fig.add_subplot(gs[0,1])
sns.scatterplot(x=y_test, y=y_pred_rf, ax=ax1,
                color=sns.color_palette("viridis")[3], alpha=0.7)
ax1.plot([y.min(),y.max()], [y.min(),y.max()], 'r--')
ax1.set_title(f"Random Forest ( $R^2$ ={{r2_rf:.3f}}, RMSE={{rmse_rf:,.0f}})")
ax1.set_xlabel("Skutečná cena")
ax1.set_ylabel("Predikovaná cena")

# 7.2 Barploty
ax2 = fig.add_subplot(gs[1,0])
coef_df.plot.barh(ax=ax2, color="steelblue")
ax2.set_title("Koeficienty lineární regrese")
ax2.set_xlabel("Hodnota koeficientu")

ax3 = fig.add_subplot(gs[1,1])
imp_df.plot.barh(ax=ax3, color="seagreen")
ax3.set_title("Feature importance (Random Forest)")
ax3.set_xlabel("Relativní důležitost")

# 7.3 Textové shrnutí
ax4 = fig.add_subplot(gs[2,:])
ax4.axis("off")
summary = (

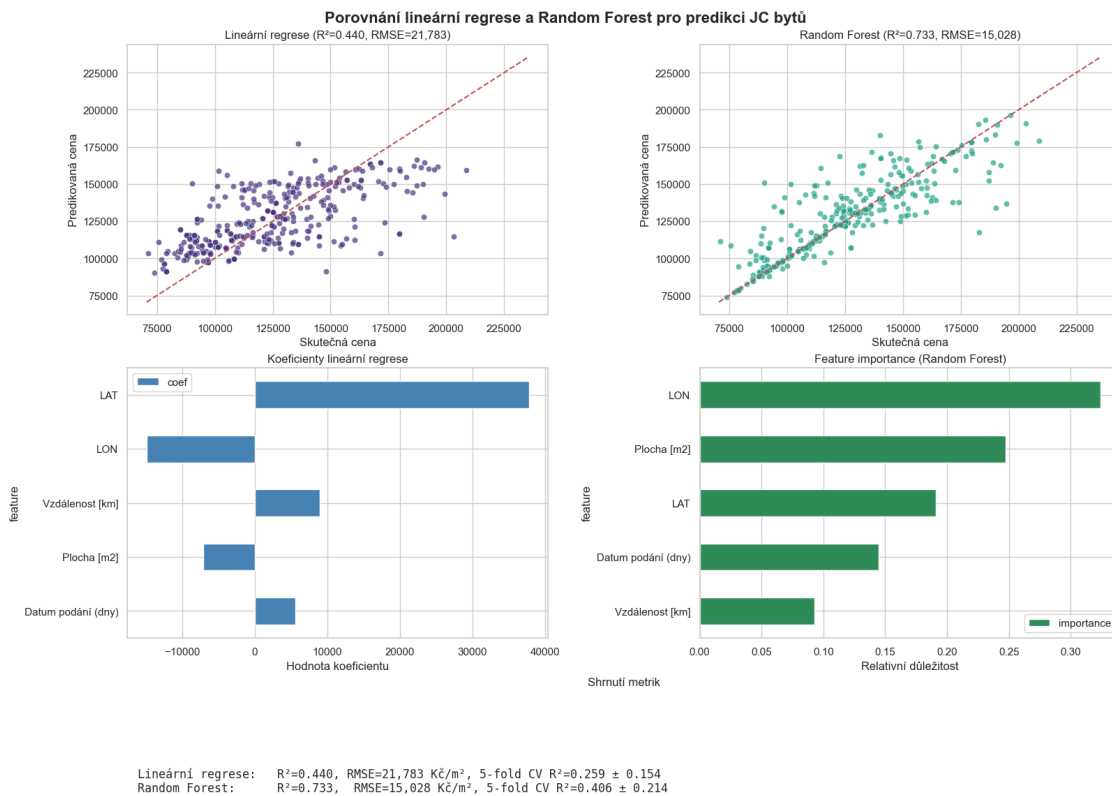
```

```

f"Lineární regrese:   R²={r2_lin:.3f}, RMSE={rmse_lin:,.0f} Kč/m², "
f"5-fold CV R²={cv_lin.mean():.3f} ± {cv_lin.std():.3f}\n"
f"Random Forest:    R²={r2_rf:.3f},  RMSE={rmse_rf:,.0f} Kč/m², "
f"5-fold CV R²={cv_rf.mean():.3f} ± {cv_rf.std():.3f}"
)
ax4.text(0.01, 0.5, summary, va="center", fontsize=12, family="monospace")
ax4.set_title("Shrnutí metrik", pad=15)

plt.suptitle("Porovnání lineární regrese a Random Forest pro predikci JC bytů",
             fontsize=16, fontweight="bold", y=1.02)
plt.show()

```



[78]: *# Analýza reziduálních clusterů pro RF model*

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

```



```

from sklearn.ensemble          import RandomForestRegressor
from sklearn.cluster           import KMeans

import warnings
warnings.filterwarnings("ignore", message="The palette list has more values")

# --- 0) Proměnné -----
# df_clusters : Váš DataFrame s historií transakcí
features      = ['LAT', 'LON', 'Plocha [m2]', 'Vzdálenost [km]', 'Datum podání_
↳(dny)']
target        = 'JC byty [Kč/m2]'

# --- 1) Drop NaN v X i v y -----
df_clean = df_clusters.dropna(subset=features + [target]).reset_index(drop=True)

X_clean = df_clean[features]
y_clean = df_clean[target].values

# --- 2) Natrénujte pipeline RF + scaler -----
pipeline = make_pipeline(
    StandardScaler(),
    RandomForestRegressor(n_estimators=200, random_state=42)
)
pipeline.fit(X_clean, y_clean)

scaler = pipeline.named_steps['standardscaler']
rf      = pipeline.named_steps['randomforestregressor']

# --- 3) Predikce a rezidua -----
X_all = scaler.transform(df_clean[features])
df_clean['pred'] = rf.predict(X_all)
df_clean['resid'] = df_clean[target] - df_clean['pred']

# --- 4) Shlukování podle [pred, resid] -----
k = 4
km = KMeans(n_clusters=k, random_state=42)
df_clean['resid_cluster'] = km.fit_predict(df_clean[['pred', 'resid']])

# --- 5) Vykreslení dashboardu -----
sns.set_style("whitegrid")
fig = plt.figure(constrained_layout=True, figsize=(18,12))
gs = fig.add_gridspec(3,3, height_ratios=[1,1,0.8])

# 5.1) Predikce vs rezidua
ax1 = fig.add_subplot(gs[0,0])
sns.scatterplot(x='pred', y='resid',
                hue='resid_cluster', palette='tab10',

```

```

        data=df_clean, s=40, alpha=0.7, ax=ax1)
ax1.axhline(0, color='black', linestyle='--')
ax1.set(title='Predikce vs. rezidua dle clusteru',
        xlabel='Predikovaná cena (Kč/m²)',
        ylabel='Reziduum')

# 5.2) Boxplot reziduí
ax2 = fig.add_subplot(gs[0,1])
sns.boxplot(x='resid_cluster', y='resid',
            palette='tab10', data=df_clean, ax=ax2)
ax2.set(title='Rozdělení reziduí v clusterech',
        xlabel='Cluster', ylabel='Reziduum')

# 5.3) Histogram reziduí
ax3 = fig.add_subplot(gs[0,2])
sns.histplot(df_clean['resid'], bins=30, kde=True, color='gray', ax=ax3)
ax3.set(title='Histogram reziduí',
        xlabel='Reziduum', ylabel='Počet')

# 5.4) Geografie reziduálních clusterů
ax4 = fig.add_subplot(gs[1,:2])
sns.scatterplot(x='LON', y='LAT',
                hue='resid_cluster', palette='tab10',
                data=df_clean, s=40, alpha=0.7, ax=ax4)
ax4.set(title='Geografie reziduálních clusterů',
        xlabel='Longitude', ylabel='Latitude')

# 5.5) Velikost clusterů
counts = df_clean['resid_cluster'].value_counts().sort_index()
ax5 = fig.add_subplot(gs[1,2])
sns.barplot(x=counts.index, y=counts.values, palette='tab10', ax=ax5)
ax5.set(title='Velikost reziduálních clusterů',
        xlabel='Cluster', ylabel='Počet')

# 5.6) Skutečná vs predikovaná cena dle clusterů
ax6 = fig.add_subplot(gs[2,:])
sns.scatterplot(x=target, y='pred',
                hue='resid_cluster', palette='tab10',
                data=df_clean, s=30, alpha=0.7, ax=ax6)
ax6.plot([df_clean[target].min(), df_clean[target].max()],
        [df_clean[target].min(), df_clean[target].max()],
        'r--')
ax6.set(title='Skutečná vs predikovaná cena dle clusteru',
        xlabel='Skutečná cena (Kč/m²)',
        ylabel='Predikovaná cena (Kč/m²)')

plt.suptitle('Analýza reziduálních clusterů pro RF model',

```

```

        fontsize=16, fontweight="bold", y=1.02)
plt.show()

```

C:\Users\ijttr\AppData\Local\Temp\ipykernel_77160\1389783784.py:63:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```

sns.boxplot(x='resid_cluster', y='resid',
C:\Users\ijttr\AppData\Local\Temp\ipykernel_77160\1389783784.py:85:
FutureWarning:

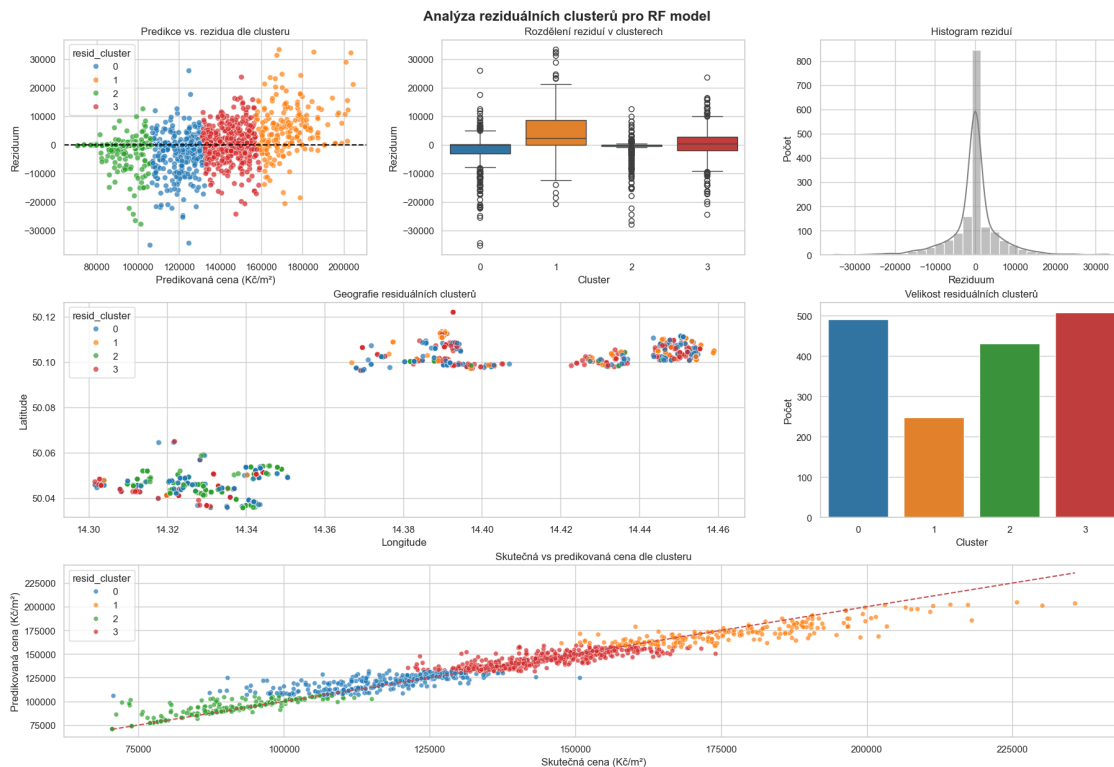
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```

sns.barplot(x=counts.index, y=counts.values, palette='tab10', ax=ax5)

```



[79]: 3 # Ocenění bytů pomocí MODELU Random Forest

```

import numpy as np

```

```

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline      import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble      import RandomForestRegressor
from sklearn.cluster       import KMeans
from IPython               import get_ipython

# --- 0) Získejte data z relace -----
ns = get_ipython().user_ns
df_clusters = ns.get('df_clusters') # historická data s JC
df_objc     = ns.get('df_objc')     # nové záznamy pro predikci

if df_clusters is None or df_objc is None:
    raise ValueError("df_clusters nebo df_objc není definováno. Spustte ↵
    ↵ příslušné buňky dříve.")

# --- 1) Definice features a targetu -----
features = ['LAT', 'LON', 'Plocha [m2]', 'Vzdálenost [km]', 'Datum podání (dny)']
target   = 'JC byty [Kč/m2]'

# --- 2) Připravíme čistou tréninkovou data -----
df_hist = df_clusters.dropna(subset=features + [target]).reset_index(drop=True)
X_hist  = df_hist[features]
y_hist  = df_hist[target].values

# --- 3) Natrénujeme model -----
pipeline = make_pipeline(
    StandardScaler(),
    RandomForestRegressor(n_estimators=200, random_state=42)
)
pipeline.fit(X_hist, y_hist)

# --- 4) Připravíme nové data pro predikci -----
df_new = df_objc.copy()
for col in features:
    # nahradíme čárky tečkami a převedeme na float
    df_new[col] = df_new[col].astype(str).str.replace(',', '.').astype(float)

# --- 5) Predikce JC/m2 -----
X_new = df_new[features]
df_new['Predikovaná JC [Kč/m2]'] = pipeline.predict(X_new).round().astype(int)

# --- 6) Spočteme celkovou cenu bytu -----
df_new['Celková cena [Kč]'] = (

```

```

    df_new['Predikovaná JC [Kč/m2]'] * df_new['Plocha [m2]']
).astype(int)

# --- 7) (Volitelné) Shluk reziduí -----
# Pokud skutečný target v df_new není k dispozici, tento krok lze vynechat.
# km = KMeans(n_clusters=4, random_state=42)
# df_new['Resid_cluster'] = km.fit_predict(df_new[['Predikovaná JC [Kč/
    ↪m2]', 'Celková cena [Kč]']])

# --- 8) Výpis výsledku -----
print(df_new)
# Nebo v Jupyteru přímo:
df_new

```

	byt	LAT	LON	Plocha [m2]	Vzdálenost [km]	Datum podání (dny)	\
0	1	50.101857	14.374558	89.17	0.0	2700.0	
1	2	50.101857	14.374558	105.17	0.0	2700.0	
2	3	50.101857	14.374558	138.44	0.0	2700.0	
3	4	50.101857	14.374558	169.51	0.0	2700.0	

	Predikovaná JC [Kč/m2]	Celková cena [Kč]
0	140080	12490933
1	139620	14683835
2	140029	19385614
3	137496	23306946

[79]:

	byt	LAT	LON	Plocha [m2]	Vzdálenost [km]	Datum podání (dny)	\
0	1	50.101857	14.374558	89.17	0.0	2700.0	
1	2	50.101857	14.374558	105.17	0.0	2700.0	
2	3	50.101857	14.374558	138.44	0.0	2700.0	
3	4	50.101857	14.374558	169.51	0.0	2700.0	

	Predikovaná JC [Kč/m2]	Celková cena [Kč]
0	140080	12490933
1	139620	14683835
2	140029	19385614
3	137496	23306946

[80]:

```

# Ocenění bytů pomocí MODELU Random Forest s aplikací vah pro jednotlivé
    ↪kvartily JC a tím zohlednit stav/kvalitu bytu zaměřením se na odpovídající
    ↪segment dat

'''
Proč váhy?
Kdybychom chtěli ocenit novostavbu (byt v perfektním stavu), nemáme v datech
    ↪sloupec „stav bytu“.
Ale víme, že dražší byty (horní kvartil cen) jsou pravděpodobně v lepším stavu.

```

Proto je necháme při učení modelu dvojmo (váha 2), zatímco běžné byty (nižší ↪ cenový kvartil) budeme brát jednou (váha 1).

Co to dělá?

Když se Random Forest učí, častěji se zaměří na vzory z dražších bytů.

Naučí se lépe předpovídat v té části cenového spektra, která odpovídá stavově ↪ lepším bytům.

Výsledek

Model se stane „naladěnějším“ na data z horního kvartilu, takže když do něj ↪ zadáš parametry novostavby, odhad ceny bude realističtější.

Shrnutí

Kvartily cen nám pomáhají kvazistav bytu nahradit cenou.

Váhy zajistí, že dražší (předpokládaně lépe udržované) byty mají při učení ↪ větší vliv.

To je matematicky správné řešení pro případ, kdy nám chybí explicitní informace ↪ o stavu bytu.

'''

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from IPython import get_ipython

# --- 0) Načtení dat z relace ---
ns = get_ipython().user_ns
srovnaci = ns.get('df_clusters') # srovnávací data s JC
ocenovane = ns.get('df_objc')    # byty k ocenění
if srovnaci is None or ocenovane is None:
    raise ValueError("df_clusters nebo df_objc není definováno.")

# --- 1) Definice features a cíle ---
features = ['LAT', 'LON', 'Plocha [m2]', 'Vzdálenost [km]', 'Datum podání (dny)']
target   = 'JC byty [Kč/m2]'

# --- 2) Příprava historických dat ---
df_hist = srovnaci.dropna(subset=features + [target]).reset_index(drop=True)
y_hist = df_hist[target].values

# --- 3) Výpočet kvartilů a vah ---
Q1 = df_hist[target].quantile(0.25)
Q2 = df_hist[target].quantile(0.50)
```

```

Q3 = df_hist[target].quantile(0.75)
weight_q1, weight_q2, weight_q3, weight_q4 = 0.5, 1.0, 1.5, 2.0

def assign_weight(price):
    if price < Q1: return weight_q1
    if price < Q2: return weight_q2
    if price < Q3: return weight_q3
    return weight_q4
sample_weights = df_hist[target].apply(assign_weight).values

# --- 4) Trénink modelu ---
scaler = StandardScaler().fit(df_hist[features])
rf = RandomForestRegressor(n_estimators=200, random_state=42)
rf.fit(scaler.transform(df_hist[features]), y_hist,
      ↪sample_weight=sample_weights)

# --- 5) Predikce pro ocenované byty ---
df_new = ocenovane.copy()
for col in features:
    df_new[col] = df_new[col].astype(str).str.replace(',', '.').astype(float)
y_new_pred = rf.predict(scaler.transform(df_new[features]))
df_new['JC_pred'] = np.round(y_new_pred).astype(int)
df_new['Celkem'] = (df_new['JC_pred'] * df_new['Plocha [m2]']).astype(int)

# --- 6) Formátování čísel ---
def fmt(x):
    return f"{int(x):,}".replace(',', ' ')
df_new['JC_pred_fmt'] = df_new['JC_pred'].apply(fmt)
df_new['Celkem_fmt'] = df_new['Celkem'].apply(fmt)

# --- 7) Vizualizace výsledků ---
sns.set_style("whitegrid")
fig = plt.figure(constrained_layout=True, figsize=(10,16))
gs = fig.add_gridspec(4,1, height_ratios=[1,1,1,1])

# 7.1 Tabulka ocenovaných bytů
ax0 = fig.add_subplot(gs[0,0])
ax0.axis('off')
cols = ['byt', 'LAT', 'LON', 'Plocha [m2]', 'Vzdálenost [km]', 'Datum podání',
      ↪(dny)', 'JC_pred_fmt', 'Celkem_fmt']
data_vals = df_new[cols].values
norm = plt.Normalize(df_new['JC_pred'].min(), df_new['JC_pred'].max())
cmap = plt.get_cmap('RdYlGn') # červená->žlutá->zelená
cellColours = []
for row in df_new['JC_pred']:
    base = cmap(norm(row))
    light = tuple(1 - 0.8*(1 - np.array(base[:3]))) + (1,)

```

```

        cellColours.append(['white']*6 + [light, light])

table = ax0.table(
    cellText=data_vals,
    colLabels=cols,
    cellColours=cellColours,
    cellLoc='center', loc='center'
)
table.auto_set_font_size(False)
table.set_fontsize(14)
for key, cell in table.get_celld().items():
    cell.set_text_props(weight='bold', fontsize=12)
ax0.set_title('Oceněné byty', pad=20, fontsize=16)

# 7.2-7.4 Grafy s trendy
def plot_with_trend(ax, x, y, label_new):
    sns.scatterplot(x=x, y=y, data=df_hist, label='Srovnávací data', alpha=0.5,
        ax=ax)
    sns.scatterplot(x=x, y='JC_pred', data=df_new, label=label_new,
        color='red', marker='X', s=100, ax=ax)
    m,b = np.polyfit(df_hist[x], df_hist[y], 1)
    xs = np.linspace(df_hist[x].min(), df_hist[x].max(), 100)
    ax.plot(xs, m*xs+b, 'r--', label='Trend')
    ax.set_xlabel(x, fontsize=12)
    ax.set_ylabel(y, fontsize=12)
    ax.legend(fontsize=12)

ax1 = fig.add_subplot(gs[1,0])
plot_with_trend(ax1, 'Datum podání (dny)', target, 'Ocenované byty')
ax1.set_title('Datum podání vs JC', fontsize=14)

ax2 = fig.add_subplot(gs[2,0])
plot_with_trend(ax2, 'Plocha [m2]', target, 'Ocenované byty')
ax2.set_title('Plocha vs JC', fontsize=14)

ax3 = fig.add_subplot(gs[3,0])
plot_with_trend(ax3, 'Vzdálenost [km]', target, 'Ocenované byty')
ax3.set_title('Vzdálenost vs JC', fontsize=14)

plt.show()

```


Oceněné byty

byt	LAT	LON	Plocha [m2]	Vzdálenost [km]	Datum podání (dny)	JC_pred_fmt	Celkem_fmt
1	50.10185708	14.3745576	89.17	0.0	2700.0	135 195	12 055 338
2	50.10185708	14.3745576	105.17	0.0	2700.0	134 614	14 157 354
3	50.10185708	14.3745576	138.44	0.0	2700.0	134 365	18 601 490
4	50.10185708	14.3745576	169.51	0.0	2700.0	133 000	22 544 830

