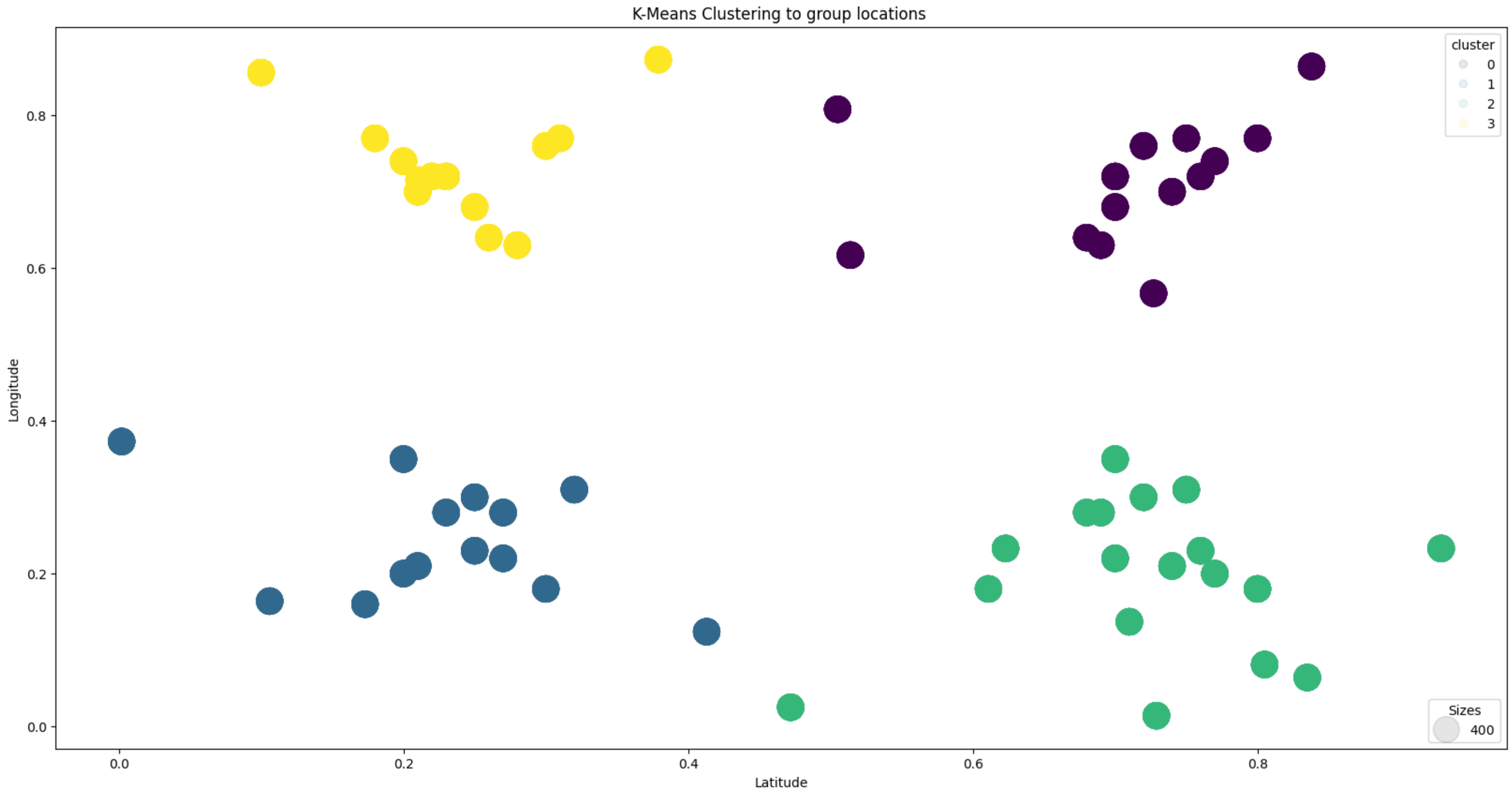
Data preparation - Environemental Solar Data

- Fill the dataset for the 2022 for the months 09 to 12
- Labeling the location ID
- Labeling the Quadrant ID

In [3]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
from imblearn.pipeline import Pipeline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import re



df = pd.read_csv('Datasets/Environment_Solar_Data.csv')
df = df[df.sunshine_hours != 1000]

def process_string(x):
    # Extract the month using a regular expression
#     print(x)
    month = result = x.split('/')[0]
#     print("month", month)
    return month


df['month_id'] = df['Date'].apply(process_string)

df['month_id'] = pd.to_numeric(df['month_id'])


# 60 monthly sunshine hours for every lantitude and longitude (01.2019 to 12.2022 - 60months)
model = KMeans(n_clusters=59, n_init=10)
X = df[['Latitude', 'Longitude']]
model.fit(X)
cluster_labels = model.predict(X)
df['location_id'] = cluster_labels


model = KMeans(n_clusters=4, n_init=10)
X = df[['Latitude', 'Longitude']]

# imputer = SimpleImputer(strategy='mean')
# X_imputed = imputer.fit_transform(X)
model.fit(X)

cluster_labels = model.predict(X)

df['location_quadrants'] = cluster_labels


df.to_csv('csv_outputs/env_data_processed.csv', index=False)




for i in range(1,13):
    value = df.loc[df['month_id'] == i]
    for j in range(0,59):
        location_point = value.loc[value['location_id'] == j]
        mean_sunshine = location_point['sunshine_hours'].mean()
        mean_wind_speed = location_point['Avg.Wind_Speed'].mean()
        mean_land_prices = location_point['land_prices'].mean()

        null_rows = location_point[location_point['sunshine_hours'].isnull()]
        indices = null_rows.index
        if not indices.empty:
            int_value = indices.values[0]
            df.at[int_value, 'sunshine_hours'] = mean_sunshine
            df.at[int_value, 'Avg.Wind_Speed'] = mean_wind_speed
            df.at[int_value, 'land_prices'] = mean_land_prices
            if len(indices.values) == 2:
                int_value = indices.values[1]
                df.at[int_value, 'sunshine_hours'] = mean_sunshine
                df.at[int_value, 'Avg.Wind_Speed'] = mean_wind_speed
                df.at[int_value, 'land_prices'] = mean_land_prices
```

```
df.to_csv('csv_outputs/env_data_processed.csv', index=False)

plt.rcParams['figure.figsize'] = [20, 10]

df = pd.read_csv('csv_outputs/env_data_processed.csv')
fig, ax = plt.subplots()
scatter = ax.scatter(df['Latitude'], df['Longitude'],
                     c=df['location_quadrants'], s=400, marker='o', alpha = 0.1)
# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper right", title="cluster")
ax.add_artist(legend1)

# produce a legend with a cross section of sizes from the scatter
handles, labels = scatter.legend_elements("sizes")
legend2 = ax.legend(handles, labels, loc="lower right", title="Sizes")
ax.set_title('K-Means Clustering to group locations ')
ax.set_xlabel('Latitude')
ax.set_ylabel('Longitude')
plt.show()

centroids = model.cluster_centers_
for i, centroid in enumerate(centroids):
    print('centroid for the Cluster', i)
    print('their Latitude:', centroid[0])
    print('their Longitude:', centroid[1])

# Group the data by cluster and compute summary statistics for each feature
cluster_stats = df.groupby('land_prices').describe()

# Print the summary statistics for each feature
# print(cluster_stats)
```



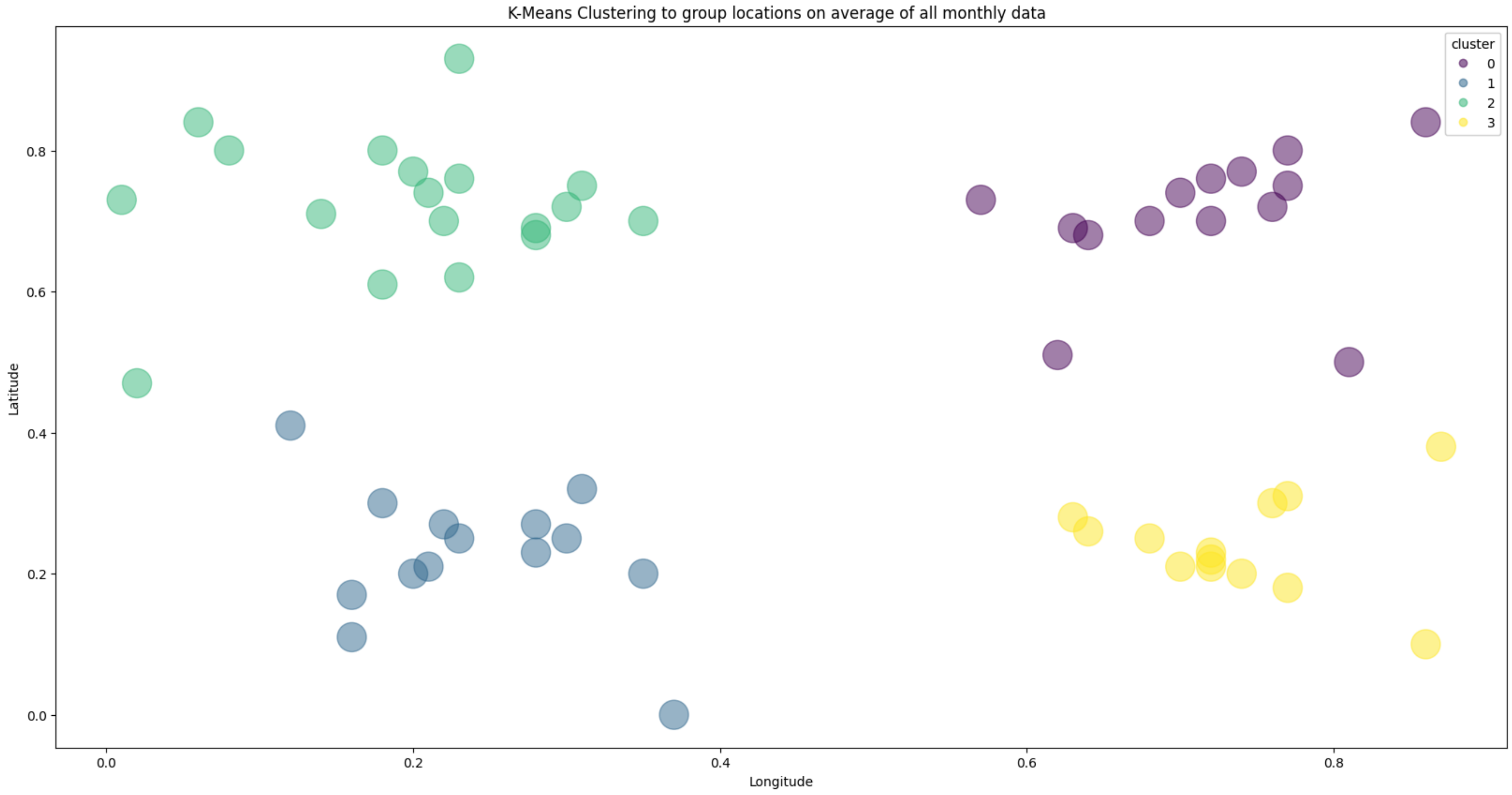

```
centroid for the Cluster 0
their Latitude: 0.7066901072705603
their Longitude: 0.7134600715137069
centroid for the Cluster 1
their Latitude: 0.22814285714285687
their Longitude: 0.2415000000000002
centroid for the Cluster 2
their Latitude: 0.7235555555555557
their Longitude: 0.19594444444444442
centroid for the Cluster 3
their Latitude: 0.2407692307692309
their Longitude: 0.7364615384615386
```

Monthly average of the sunshine hours

In [4]:
```
df = pd.read_csv('csv_outputs/env_data_processed.csv')


location_quadrants = []
landprice_list = []
sunshine_list = []
Latitude = []
Longitude = []
for i in range(0,59):
```

```
    value = df.loc[df['location_id'] == i]
#     print(value.loc[:, ['Date','sunshine_hours', 'location_quadrants']])
#     print(mean_sunshine, mean_landprice)

    sunshine_list.append(round(value['sunshine_hours'].mean(),4))
    landprice_list.append(round(value['land_prices'].mean(),2))
    location_quadrants.append(value['location_quadrants'].mean())
    Latitude.append(round(value['Latitude'].mean(),2))
    Longitude.append(round(value['Longitude'].mean(),2))



#     print(value.loc[:, ['Date','sunshine_hours']])
#     print(mean_sunshine, mean_landprice)


data = list(zip(Longitude, Latitude,
                sunshine_list, landprice_list,
                location_quadrants))
# Convert the list to a data frame
df_new = pd.DataFrame(data, columns=['Longitude',
                                     'Latitude', 'sunshine_avg',
                                     'land_prices',
                                     'location_quadrants'])

# Save the data frame to a CSV file
df_new.to_csv('csv_outputs/average_env_data_processed.csv', index=False)
```

In [5]:
```
df = pd.read_csv('csv_outputs/average_env_data_processed.csv')

fig, ax = plt.subplots()
scatter = ax.scatter(df['Longitude'], df['Latitude'],
                     c=df['location_quadrants'],
                     s=500, marker='o', alpha = 0.5)
# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper right", title="cluster")
ax.add_artist(legend1)

ax.set_title('K-Means Clustering to group locations on average of all monthly data  ')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.show()
```

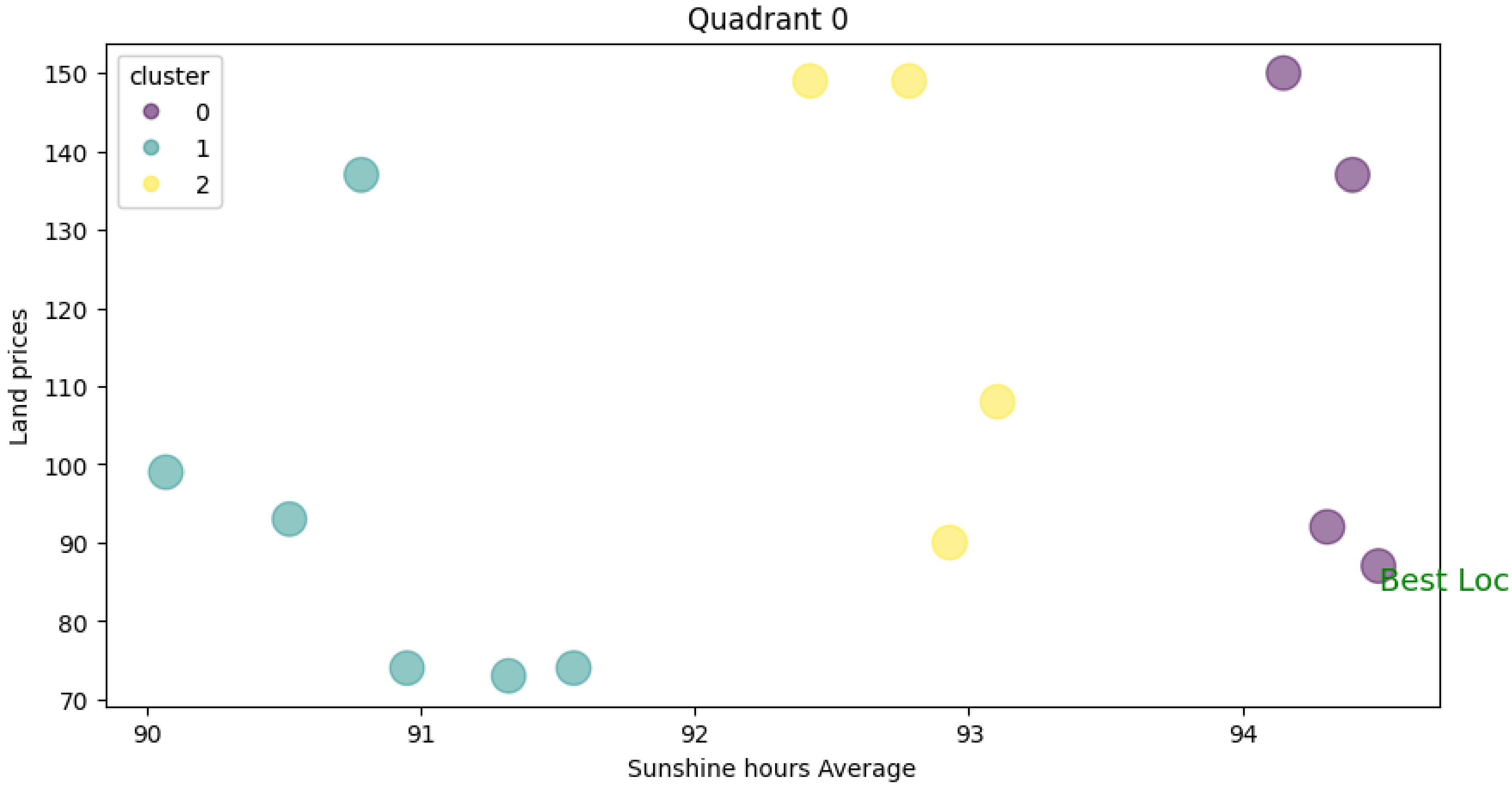


Optimisation technique to find the best points in each quadrants.

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from scipy.optimize import minimize

plt.rcParams['figure.figsize'] = [10, 5]
quadrant = 0

df = pd.read_csv('csv_outputs/average_env_data_processed.csv')
```

```python
value = df.loc[df['location_quadrants'] == quadrant]

X = value[['sunshine_avg']]


model = KMeans(n_clusters=3, n_init=10)

model.fit(X)

cluster_labels = model.predict(X)

value['sunshine_group'] = cluster_labels

# Find the cluster with the highest mean value
cluster_idx = np.argmax(model.cluster_centers_)

# Select the rows of the data frame that belong to the cluster with the highest mean value
high_cluster = value[cluster_labels == cluster_idx]

min_row = high_cluster.nsmallest(1, 'land_prices')

print("The best location in the quadrant {} is in location ({}, {})"
      .format(quadrant,
        min_row['Longitude'].values[0],
        min_row['Latitude'].values[0]))
if min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North East")
elif min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South East")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South West")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North West")

print(" with sunshine hrs of {} Kwh/month and its price is {} euros per m2"
      .format(min_row['sunshine_avg'].values[0],
              min_row['land_prices'].values[0]))

fig, ax = plt.subplots()
# Create a scatter plot
scatter = ax.scatter(value['sunshine_avg'], value['land_prices'],
                     c=cluster_labels, s=200, alpha = .5)

# Add labels and title
ax.set_xlabel('Sunshine hours Average')
ax.set_ylabel('Land prices')
ax.set_title('Quadrant {} '.format(quadrant))
ax.text(min_row['sunshine_avg'].values[0],
        min_row['land_prices'].values[0], 'Best Loc', fontsize=13, color='Green', va='top')


value.to_csv('csv_outputs/cluster_for_quadrant_{}.csv'.format(quadrant), index=False)
min_row.to_csv('csv_outputs/best_of_quadrant_{}.csv'.format(quadrant), index=False)

# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper left", title="cluster")
ax.add_artist(legend1)

# Show the plot
plt.show()
```

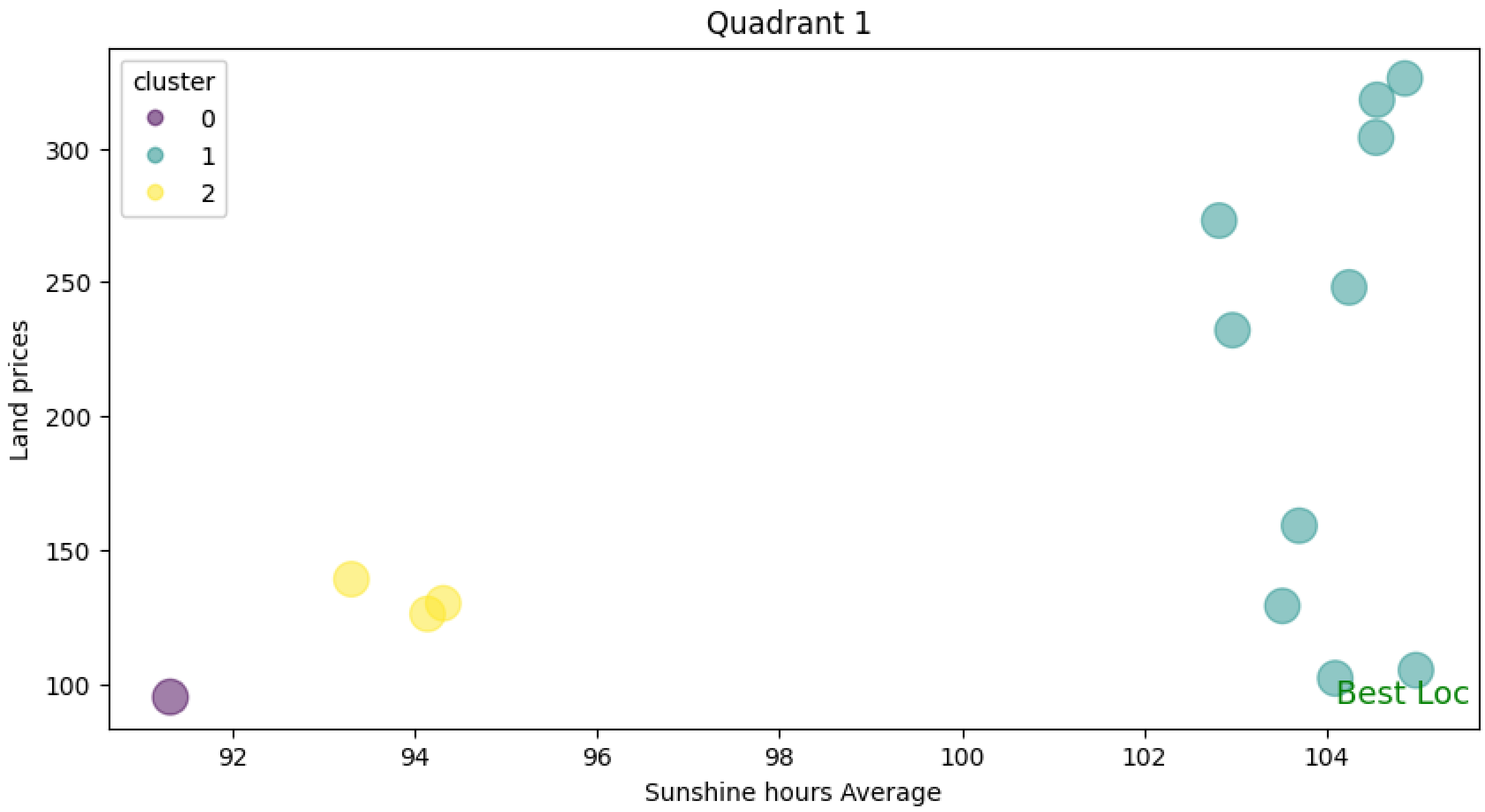
```
/tmp/ipykernel_13852/1355006253.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  value['sunshine_group'] = cluster_labels
The best location in the quadrant 0 is in location (0.74, 0.77)
This is for quadrant North East
 with sunshine hrs of 94.4948 Kwh/month and its price is 87.0 euros per m2
```



In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from scipy.optimize import minimize

plt.rcParams['figure.figsize'] = [10, 5]
quadrant = 1

df = pd.read_csv('csv_outputs/average_env_data_processed.csv')


value = df.loc[df['location_quadrants'] == quadrant]

X = value[['sunshine_avg']]


model = KMeans(n_clusters=3, n_init=10)

model.fit(X)

cluster_labels = model.predict(X)

value['sunshine_group'] = cluster_labels

# Find the cluster with the highest mean value
cluster_idx = np.argmax(model.cluster_centers_)

# Select the rows of the data frame that belong to
# the cluster with the highest mean value
high_cluster = value[cluster_labels == cluster_idx]

min_row = high_cluster.nsmallest(1, 'land_prices')

print("The best location in the quadrant {} is in location ({}, {})"
      .format(quadrant,
    min_row['Longitude'].values[0],
    min_row['Latitude'].values[0]))


if min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North East")
elif min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South East")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South West")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North West")


print(" with sunshine hrs of {} Kwh/month and its price is {} euros per m2"
      .format(min_row['sunshine_avg'].values[0],
    min_row['land_prices'].values[0]))

fig, ax = plt.subplots()
# Create a scatter plot
scatter = ax.scatter(value['sunshine_avg'], value['land_prices'],
                     c=cluster_labels, s=200, alpha = .5)
```

```
# Add labels and title
ax.set_xlabel('Sunshine hours Average')
ax.set_ylabel('Land prices')
ax.set_title('Quadrant {} '.format(quadrant))
ax.text(min_row['sunshine_avg'].values[0],
        min_row['land_prices'].values[0], 'Best Loc',
        fontsize=13, color='Green', va='top')


value.to_csv('csv_outputs/cluster_for_quadrant_{}.csv'.format(quadrant), index=False)
min_row.to_csv('csv_outputs/best_of_quadrant_{}.csv'.format(quadrant), index=False)

# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper left", title="cluster")
ax.add_artist(legend1)

# Show the plot
plt.show()
```

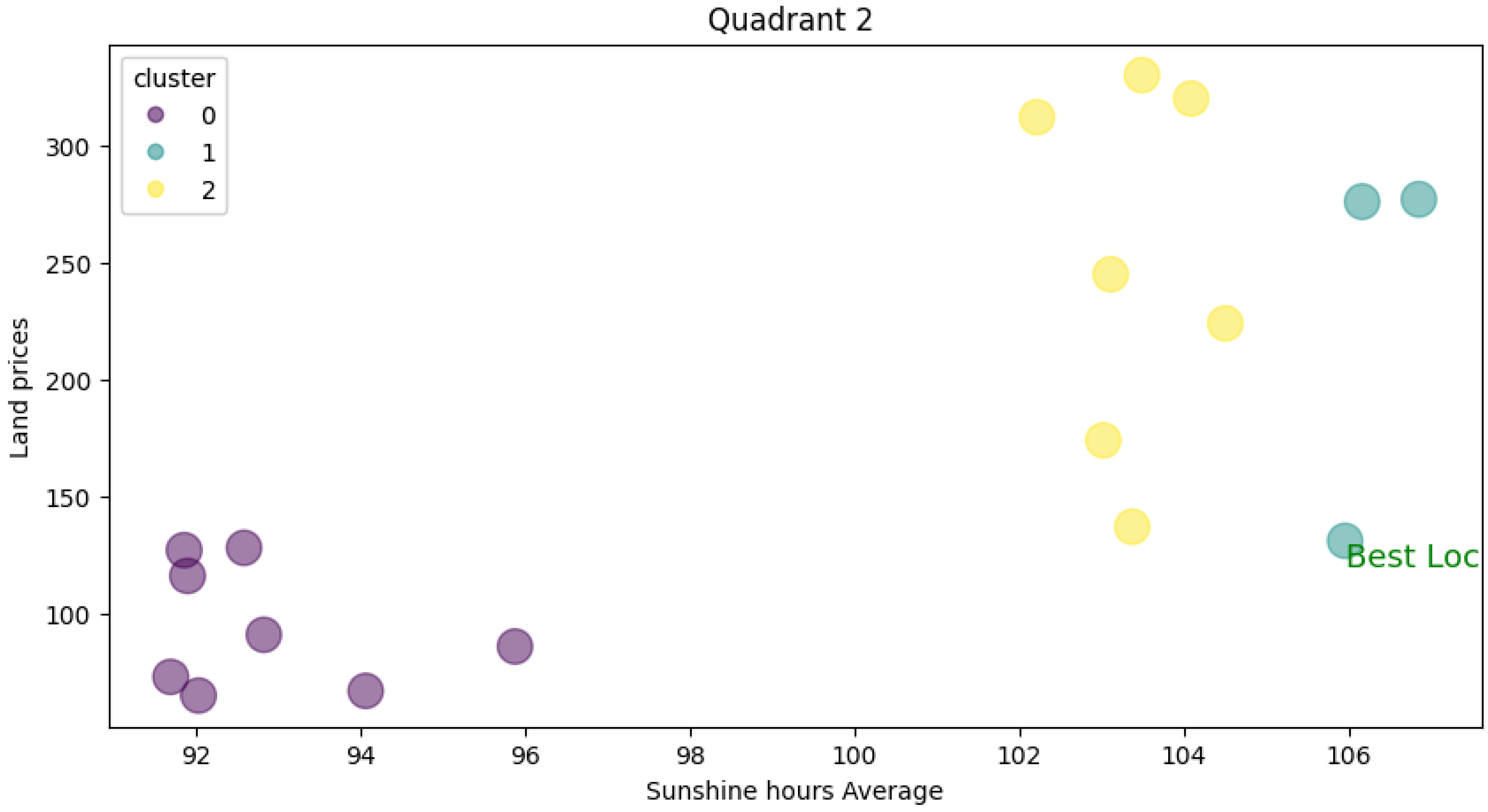
```
/tmp/ipykernel_13852/640240244.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  value['sunshine_group'] = cluster_labels
The best location in the quadrant 1 is in location (0.28, 0.27)
This is for quadrant South West
 with sunshine hrs of 104.0952 Kwh/month and its price is 102.0 euros per m2
```



In [4]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from scipy.optimize import minimize

plt.rcParams['figure.figsize'] = [10, 5]
quadrant = 2

df = pd.read_csv('csv_outputs/average_env_data_processed.csv')


value = df.loc[df['location_quadrants'] == quadrant]

X = value[['sunshine_avg']]


model = KMeans(n_clusters=3, n_init=10)

model.fit(X)

cluster_labels = model.predict(X)

value['sunshine_group'] = cluster_labels

# Find the cluster with the highest mean value
cluster_idx = np.argmax(model.cluster_centers_)
```

```
# Select the rows of the data frame that belong to the cluster with the highest mean value
high_cluster = value[cluster_labels == cluster_idx]

min_row = high_cluster.nsmallest(1, 'land_prices')

print("The best location in the quadrant {} is in location ({}, {})"
      .format(quadrant,
      min_row['Longitude'].values[0],
    min_row['Latitude'].values[0]))


if min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North East")
elif min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South East")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South West")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North West")


print(" with sunshine hrs of {} Kwh/month and its price is {} euros per m2"
      .format(min_row['sunshine_avg'].values[0],
        min_row['land_prices'].values[0]))

fig, ax = plt.subplots()
# Create a scatter plot
scatter = ax.scatter(value['sunshine_avg'], value['land_prices'],
                     c=cluster_labels, s=200, alpha = .5)

# Add labels and title
ax.set_xlabel('Sunshine hours Average')
ax.set_ylabel('Land prices')
ax.set_title('Quadrant {} '.format(quadrant))
ax.text(min_row['sunshine_avg'].values[0],
        min_row['land_prices'].values[0], 'Best Loc',
        fontsize=13, color='Green',  va='top')


value.to_csv('csv_outputs/cluster_for_quadrant_{}.csv'
             .format(quadrant), index=False)
min_row.to_csv('csv_outputs/best_of_quadrant_{}.csv'
               .format(quadrant), index=False)

# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper left", title="cluster")
ax.add_artist(legend1)

# Show the plot
plt.show()
```

```
/tmp/ipykernel_13852/17067273.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  value['sunshine_group'] = cluster_labels
The best location in the quadrant 2 is in location (0.28, 0.68)
This is for quadrant North West
 with sunshine hrs of 105.9501 Kwh/month and its price is 131.0 euros per m2
```

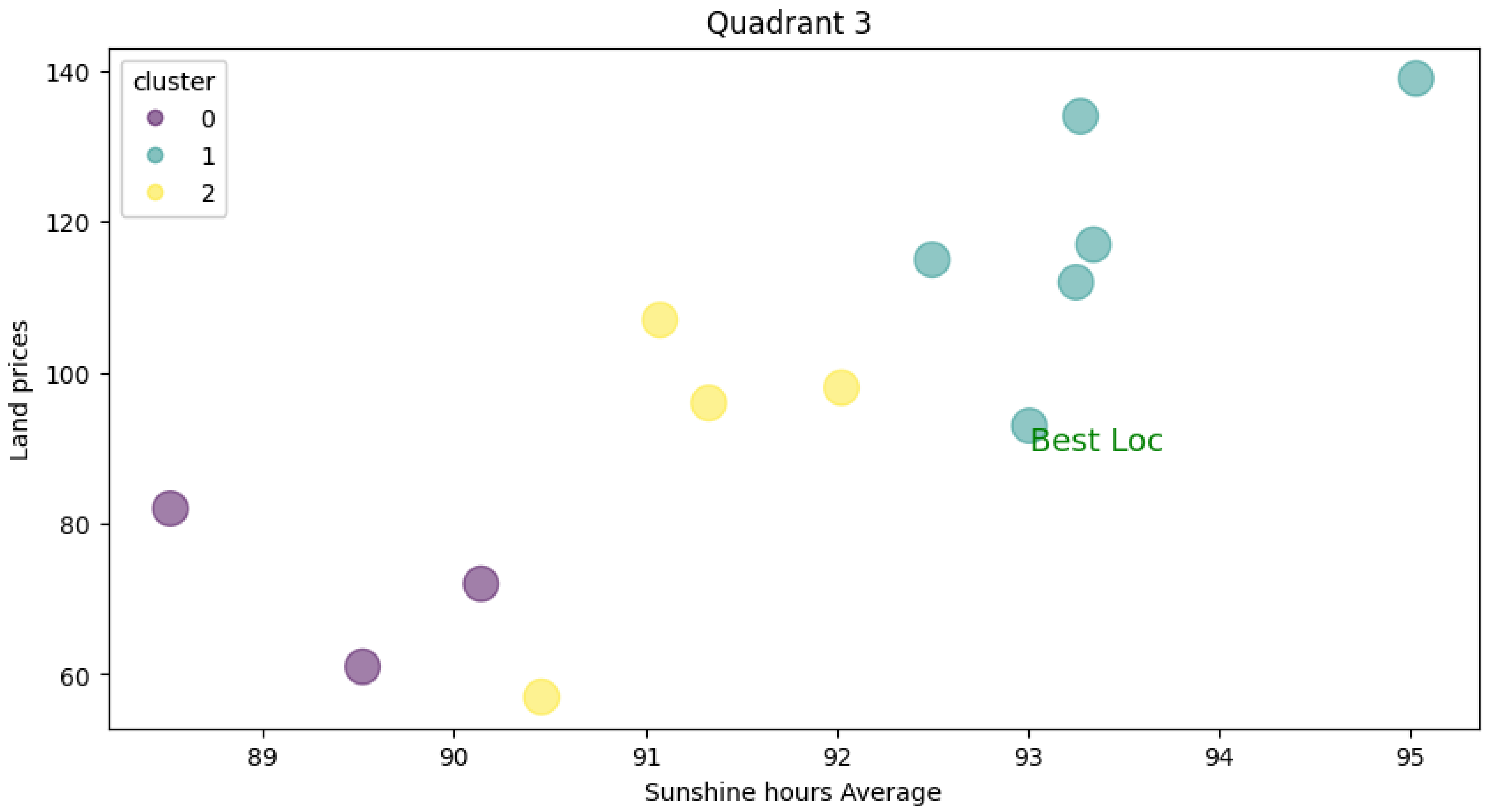


In [5]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from scipy.optimize import minimize

plt.rcParams['figure.figsize'] = [10, 5]
quadrant = 3

df = pd.read_csv('csv_outputs/average_env_data_processed.csv')


value = df.loc[df['location_quadrants'] == quadrant]

X = value[['sunshine_avg']]


model = KMeans(n_clusters=3, n_init=10)

model.fit(X)

cluster_labels = model.predict(X)

value['sunshine_group'] = cluster_labels

# Find the cluster with the highest mean value
cluster_idx = np.argmax(model.cluster_centers_)

# Select the rows of the data frame that belong to the cluster with the highest mean value
high_cluster = value[cluster_labels == cluster_idx]

min_row = high_cluster.nsmallest(1, 'land_prices')

print(" The best location in the quadrant {} is in location ({}, {})"
      .format(quadrant,
    min_row['Longitude'].values[0],
    min_row['Latitude'].values[0]))


if min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North East")
elif min_row['Longitude'].values[0] > 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South East")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] < 0.45 :
    print("This is for quadrant South West")
elif min_row['Longitude'].values[0] < 0.45 and min_row['Latitude'].values[0] > 0.45 :
    print("This is for quadrant North West")


print(" with sunshine hrs of {} Kwh/month and its price is {} euros per m2"
      .format(min_row['sunshine_avg'].values[0],
     min_row['land_prices'].values[0]))

fig, ax = plt.subplots()
# Create a scatter plot
scatter = ax.scatter(value['sunshine_avg'], value['land_prices'],
                     c=cluster_labels, s=200, alpha = .5)
```

```
# Add labels and title
ax.set_xlabel('Sunshine hours Average')
ax.set_ylabel('Land prices')
ax.set_title('Quadrant {} '.format(quadrant))
ax.text(min_row['sunshine_avg'].values[0],
        min_row['land_prices'].values[0], 'Best Loc',
        fontsize=13, color='Green', va='top')


value.to_csv('csv_outputs/cluster_for_quadrant_{}.csv'
             .format(quadrant), index=False)
min_row.to_csv('csv_outputs/best_of_quadrant_{}.csv'
               .format(quadrant), index=False)

# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper left", title="cluster")
ax.add_artist(legend1)

# Show the plot
plt.show()
```

```
/tmp/ipykernel_13852/216255868.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  value['sunshine_group'] = cluster_labels
 The best location in the quadrant 3 is in location (0.7, 0.21)
This is for quadrant South East
 with sunshine hrs of 93.0098 Kwh/month and its price is 93.0 euros per m2
```



Supervised learning - Linear Regression

In [18]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


df = pd.read_csv('Datasets/Installed_Solar_Plants.csv')


# Remove the commas from the strings
df['Generated_energy_kWh_per_a'] = df['Generated_energy_kWh_per_a'].str.replace(',', '')



df['Sunshine_Hours_per_year'] = pd.to_numeric(df['Sunshine_Hours_per_year'])
df['Generated_energy_kWh_per_a'] = pd.to_numeric(df['Generated_energy_kWh_per_a'])

# Create a new column for x1 * x2
df['product_sunshine_panel_size'] = df['Sunshine_Hours_per_year'] * df['Size_Solar_Panel_m2']

# print(df)
df.to_csv('csv_outputs/Installed_Solar_Plants_processed.csv', index=False)
```

Linear Regression with Kfolding technique

In [19]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold


df = pd.read_csv('csv_outputs/Installed_Solar_Plants_processed.csv')

X = df[['product_sunshine_panel_size']]

y = df['Generated_energy_kWh_per_a']

model = LinearRegression()

# Create a k-fold cross-validation iterator
kfold = KFold(n_splits=5, shuffle=True, random_state=1)

# Initialize a list to store the scores
scores = []

# Loop through the folds and fit the model on
# the training data, then score on the test data
for train_index, test_index in kfold.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))

    # Print the coefficients
print('Coefficients:', model.coef_)

# Print the intercept
print('Intercept:', model.intercept_)

# Calculate the mean and standard deviation of the scores
mean_score = np.mean(scores)
std_dev = np.std(scores)

print(f'Mean score: {mean_score}')
print(f'Standard deviation: {std_dev:.3f}')
```

```
Coefficients: [0.21086572]
Intercept: -819.0896819195477
Mean score: 0.9992211407653047
Standard deviation: 0.001
```

Linear Regression with normal data set spliting

```
- We dont use it because of smaller data set and high error
- This is why we use the Kfolding technique
```

In [12]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


df = pd.read_csv('csv_outputs/Installed_Solar_Plants_processed.csv')

X = df[['product_sunshine_panel_size']]

y = df['Generated_energy_kWh_per_a']

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)


model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)


# Print the coefficients
print('Coefficients:', model.coef_)

# Print the intercept
print('Intercept:', model.intercept_)
```

```
print('Mean Absolute Error:', mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
print('R2 Score:', r2_score(y_test, y_pred))
```

```
Coefficients: [0.2106823]
Intercept: 1249.9256050355034
Mean Absolute Error: 6821.184546696939
Mean Squared Error: 56081923.531897224
R2 Score: 0.999836899269818
```

Linear Programming

```
- Solving to find the x0,x1,x2,x3
- By solving for the contraints we get the panel size in each quadrant
```

Minimize cost of solar farm

In [16]:

```
# Linear programming

# From Linear Regression with Kfolding Technique
Coefficients = 0.21086572
Intercept = -819.0896819195477

# Sunshine hours monthy * coefficient *12 months

a11 = 93.01 * Coefficients *12
a12 = 105.95 * Coefficients *12
a13 = 94.494 * Coefficients *12
a14 = 104.0952 * Coefficients *12


# Land price plus the material cost
a21 = 93 + 100
a22 = 131 + 100
a23 = 87 + 100
a24 = 102 + 100


a31 = a32 = a33 = a34 = 0

a41 = a42 = a43 = a44 = 0


b1 = 2000000 - (4*Intercept) # kwh/a
b2 = 2000000 # Euros
b3 = 0
b4 = 0


c1 = a21
c2 = a22
c3 = a23
c4 = a24


A_eq = [[a11, a12, a13, a14],
    [a31, a32, a33, a34],
    [a31, a32, a33, a34],
    [a41, a42, a43, a44]]


b_eq = [b1, b3, b3, b4]

c =[c1, c2, c3, c4]
# print(c)


# define the bounds on the variables
bnds = ((100, 2000), (100, 3000), (100, 3000), (100, 2000))

# define the initial starting point for the variables
x0 = [100, 100, 100, 100]

# import the linprog function from the scipy.optimize library
from scipy.optimize import linprog


# options = {"tol": 8.2e+04,
#             "maxiterint": 10}
# solve the linear programming problem
res = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bnds, method='highs', x0=x0)
```

```
status = res.status
panel_size = res.x
print("Panel size in South east {} m2".format(panel_size[0]))
print("Panel size in South west {} m2".format(panel_size[3]))
print("Panel size in North east {} m2".format(panel_size[2]))
print("Panel size in North west {} m2".format(panel_size[1]))
print("Total panel size in {} m2"
      .format(panel_size[0]+panel_size[1]+panel_size[2]+panel_size[3]))

amount = (a21* panel_size[0]) + (a22* panel_size[1]) + (a23* panel_size[2]) + (a24* panel_size[3])
power_produced = ((a11* panel_size[0])
                    + (a12* panel_size[1])
                    + (a13* panel_size[2])
                    + (a14* panel_size[3])
                    +(4*Intercept))

print("The amount need to build these is {} euros ".format(amount))
print("The Solar power produced {} kwh/a ".format(power_produced))
```

```
Panel size in South east 2000.0 m2
Panel size in South west 2000.0 m2
Panel size in North east 3000.0 m2
Panel size in North west 1075.9303848106345 m2
Total panel size in 8075.930384810635 m2
The amount need to build these is 1599539.9188912567 euros
The Solar power produced 2000000.0 kwh/a
```

Maximize the Solar power Production

In [14]:

```
# Linear programming
# define the objective function


# From Linear Regression with Kfolding Technique
Coefficients = 0.21086572
Intercept = -819.0896819195477

# Sunshine hours monthy * coefficient *12 months

a11 = 93.01 * Coefficients *12
a12 = 105.95 * Coefficients *12
a13 = 94.494 * Coefficients *12
a14 = 104.0952 * Coefficients *12


# Land price plus the material cost
a21 = 93 + 100
a22 = 131 + 100
a23 = 87 + 100
a24 = 102 + 100


a31 = a32 = a33 = a34 = 0

a41 = a42 = a43 = a44 = 0


b1 = 2000000 - (4*Intercept) # kwh/a
b2 = 2000000 # Euros
b3 = 0
b4 = 0



c1 = a11
c2 = a12
c3 = a13
c4 = a14

A_eq = [[a21, a22, a23, a24],
    [a31, a32, a33, a34],
    [a31, a32, a33, a34],
    [a41, a42, a43, a44]]


b_eq = [b2, b3, b3, b4]


# define the bounds on the variables
bnds = ((100, 2000), (100, 3000), (100, 3000), (100, 2000))

# define the initial starting point for the variables
x0 = [100, 100, 100, 100]

# import the linprog function from the scipy.optimize library
```

```python
from scipy.optimize import linprog


# options = {"tol": 8.2e+04,
#             "maxiterint": 10}
# solve the linear programming problem
res = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bnds, method='highs', x0=x0)


status = res.status
panel_size = res.x
print("Panel size in South east {} m2".format(panel_size[0]))
print("Panel size in South west {} m2".format(panel_size[3]))
print("Panel size in North east {} m2".format(panel_size[2]))
print("Panel size in North west {} m2".format(panel_size[1]))
print("Total panel size in {} m2"
      .format(panel_size[0]+panel_size[1]+panel_size[2]+panel_size[3]))

amount = (a21* panel_size[0]) + (a22* panel_size[1]) + (a23* panel_size[2]) + (a24* panel_size[3])
power_produced = ((a11* panel_size[0])
                  + (a12* panel_size[1])
                  + (a13* panel_size[2])
                  + (a14* panel_size[3])
                  + (4*Intercept))

print("The amount need to build these is {} euros ".format(amount))
print("The Solar power produced {} kwh/a ".format(power_produced))
```

```
Panel size in South east 1772.020725388601 m2
Panel size in South west 2000.0 m2
Panel size in North east 3000.0 m2
Panel size in North west 3000.0 m2
Total panel size in 9772.0207253886 m2
The amount need to build these is 2000000.0 euros
The Solar power produced 2468730.286067561 kwh/a
```

In [15]:

```python
# "To reach 3 million KW/a we need approximately"


# Panel size in South east 2890.706950735872 m2
# Panel size in South west 3000.0 m2
# Panel size in North east 3000.0 m2
# Panel size in North west 3000.0 m2
# Total panel size in 11890.706950735872 m2
# The amount need to build these is 2426906.4414920234 euros
```

In [ ]: