Projektdokumentation

Thema

Analyse einer Datensammlung eines Parkplatzes in St. Peter-Ording

Teilnehmer: Christian, Johannes

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Projektdokumentation zeichnet den Verlauf eines Schulprojektes auf, das als Vorbereitung für die IHK-Projektarbeit diente. Als Datengrundlage wurden uns CSV-Daten zur Verfügung gestellt, welche die Besucherfrequenz eines Strandparkplatzes in St. Peter-Ording dokumentierten.

2 Projektbeschreibung

Unsere Analyse basiert auf Daten des Strandparkplatzes in St. Peter-Ording. Das Hauptziel des Projekts besteht darin, eine Datensammlung mithilfe von Diagrammen visuell verständlich zu machen. Eine Herausforderung ergab sich durch die Dateninkonsistenz im Zeitraum vom 24.03. bis 11.05. Diese haben wir durch eine einfache Vorhersage der Daten behoben, um die Lücke für eine bessere Analyse zu schließen.

3 Projektziel

Das übergeordnete Ziel unseres Projekts ist es, die Daten aus der vorgegebenen Datensammlung durch aussagekräftige Diagramme zu veranschaulichen. Darüber hinaus bietet das Projekt eine Möglichkeit zur Weiterbildung durch einen Einblick in die Datenanalyse mit Jupyter Notebook.

4 Code

Der unten abgebildete Code beschreibt drei Diagramme, die jeweils eine eigene Darstellung der Daten bieten.

Diagramm 1: Der größte Anstieg von Value liegt zwischen {Start Zeit} und {End Zeit}

Diagramm 2: Die Anzahl der Werte über 100

Diagramm 3: Die Anzahl der Werte über 500: {count_500}

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import argrelextrema
import plotly.express as px

# CSV-Datei mit Pandas einlesen (mit Semikolon als Trennzeichen)
data = pd.read_csv('Strandparkplatz_Ording_2021.csv', sep=';')

# Konvertiere 'timestamp' in ein DateTime-Objekt
data['timestamp'] = pd.to_datetime(data['timestamp'])

# Entferne Duplikate im Index
data = data[~data.duplicated(subset='timestamp')]

# Fülle die Lücken zwischen 24.03. und 11.05. durch lineare Interpolation in 15-min-Intervallen
filtered_data = data.set_index('timestamp').resample('15T').asfreq()

# Filtere die Daten, um 'value'-Werte ungleich null zu behalten
non_null_data = filtered_data[filtered_data['value'] != 0].copy()
```

```
# Interpoliere nur die Nicht-Null-Werte
non_null_data['value'] = non_null_data['value'].interpolate(method='time')

# Füge Nullwerte an ihren ursprünglichen Stellen ein
interpolated_data = pd.merge(filtered_data, non_null_data[['value']], how='left', left_index=True,
right_index=True)
interpolated_data['value'] = interpolated_data['value'].fillna(method='ffill')

# Entferne NaN-Werte
filtered_data = interpolated_data.dropna(subset=['value'])

# Erstelle separate DataFrames für Werte über 1000 und über 500
data_over_1000 = filtered_data[filtered_data['value'] > 1000]
data_over_500 = filtered_data[(filtered_data['value'] <= 1000) & (filtered_data['value'] > 500)]
data_below_500 = filtered_data[filtered_data['value'] <= 500]

# Anzahl der Werte über 1000 und 500
count_1000 = len(data_over_1000)
count_500 = len(data_over_500)

# Punktdiagramm erstellen und Anpassungen vornehmen
fig = px.scatter(filtered_data, x=filtered_data.index, y='value', title='Punktdiagramm mit
Regressionslinie, Median, Durchschnitt und Markierungen',
                 labels={'value': 'Wert'}, color_discrete_map={'value': 'blue'})
fig.update_traces(marker=dict(size=6, opacity=0.6))

# Werte über 1000 und über 500 markieren
fig.add_scatter(x=data_over_1000.index, y=data_over_1000['value'], mode='markers',
marker=dict(color='green', size=10), name='Werte über 1000')
fig.add_scatter(x=data_over_500.index, y=data_over_500['value'], mode='markers',
marker=dict(color='yellow', size=10), name='Werte über 500')

# Lineare Regression durchführen, Median und Durchschnitt berechnen und anzeigen
x = np.arange(len(filtered_data.index))
y = filtered_data['value']
```

```
m, b = np.polyfit(x, y, 1)
fig.add_scatter(x=filtered_data.index, y=m * x + b, mode='lines', line=dict(color='red'),
name=f'Regressionslinie: y = {m:.2f}x + {b:.2f}')

median = np.median(data['value'])
average = np.mean(data['value'])
fig.add_annotation(text=f'Median: {median:.2f}', xref='paper', yref='paper', x=0.05, y=0.9,
showarrow=False, font=dict(color='green'))
fig.add_annotation(text=f'Durchschnitt: {average:.2f}', xref='paper', yref='paper', x=0.05, y=0.85,
showarrow=False, font=dict(color='orange'))

# Legende mit Anzahl der Werte über 1000 und 500 erstellen
legend_labels = [f'Werte über 1000: {count_1000}', f'Werte über 500: {count_500}', 'Regressionslinie']
fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1))
fig.update_layout(legend_title="Legende")

# Diagramm anzeigen
fig.update_xaxes(title='Timestamp')
fig.update_yaxes(title='Wert')
fig.show()

# Unterteilung der Daten nach Tageszeit: 0-8 Uhr, 8-16 Uhr und 16-24 Uhr
data_0_8 = filtered_data[(filtered_data.index.hour >= 0) & (filtered_data.index.hour < 8)]
data_8_16 = filtered_data[(filtered_data.index.hour >= 8) & (filtered_data.index.hour < 16)]
data_16_24 = filtered_data[(filtered_data.index.hour >= 16) & (filtered_data.index.hour <= 23)]

# Erstellen von separaten Diagrammen für jede Tageszeit
fig_0_8 = px.scatter(data_0_8, x=data_0_8.index, y='value', color='value',
                     title='Punktdiagramm von 0-8 Uhr', labels={'value': 'Wert'})
fig_0_8.update_traces(marker=dict(size=6, opacity=0.6))
fig_0_8.update_layout(xaxis_title='Timestamp', yaxis_title='Wert')
fig_0_8.show()

fig_8_16 = px.scatter(data_8_16, x=data_8_16.index, y='value', color='value',
                      title='Punktdiagramm von 8-16 Uhr', labels={'value': 'Wert'})
fig_8_16.update_traces(marker=dict(size=6, opacity=0.6))
```

```
fig_8_16.update_layout(xaxis_title='Timestamp', yaxis_title='Wert')
fig_8_16.show()

fig_16_24 = px.scatter(data_16_24, x=data_16_24.index, y='value', color='value',
                       title='Punktdiagramm von 16-24 Uhr', labels={'value': 'Wert'})
fig_16_24.update_traces(marker=dict(size=6, opacity=0.6))
fig_16_24.update_layout(xaxis_title='Timestamp', yaxis_title='Wert')
fig_16_24.show()


#Erstelle Kopien der Daten, um mit den Zeitintervallen zu arbeiten
data_8h = filtered_data.copy(deep=True)
data_8h = filtered_data.copy()
eight_hour_changes = []

# Füge 'timestamp'-Spalte hinzu, falls sie nicht vorhanden ist
if 'timestamp' not in data_8h.columns:
    data_8h['timestamp'] = filtered_data.index

eight_hour_changes = []

# Hier wird die Veränderung zwischen dem ersten und letzten Wert in einem 8-Stunden-Intervall
berechnet
eight_hour_changes = []
for i in range(0, len(data_8h) - 8, 8):
  start_value = data_8h.iloc[i]['value']
  end_value = data_8h.iloc[i + 7]['value']
  eight_hour_changes.append((end_value - start_value) / start_value * 100)

# Finde den Index des größten Anstiegs
max_increase_index = eight_hour_changes.index(max(eight_hour_changes))

# Gib den Zeitrahmen des größten Anstiegs aus
start_time = data_8h.iloc[max_increase_index]['timestamp']
end_time = data_8h.iloc[max_increase_index + 7]['timestamp']

# Filtere die Daten, um Zeilen mit 'value'-Werten gleich null zu erhalten
```

```
# Konvertiere 'timestamp' in ein DateTime-Objekt
data['timestamp'] = pd.to_datetime(data['timestamp'])
# Gruppiere nach Datum und zähle die Anzahl der Werte pro Tag
daily_count = data.groupby(data['timestamp'].dt.date)['value'].count()

# Anzahl der Tage, an denen alle Werte Null sind
total_days_all_zero_values = (daily_count == 0).sum()
print(f'Gesamtanzahl der Tage, an denen alle Werte Null sind: {total_days_all_zero_values}')

print(f"Der größte Anstieg von Value liegt zwischen {start_time} und {end_time}.")
print(f'Die Anzahl der Werte über 1000: {count_1000}')
print(f'Die Anzahl der Werte über 500: {count_500}')
```

5 Projektabschluss

Die drei erstellten Diagramme ermöglichen eine schnelle und effektive Ermittlung der besucherstärksten Zeiten des Strandparkplatzes. Die gesetzten Ziele wurden vollständig erreicht,

und es bietet sich nun die Möglichkeit zu einer vertiefenden Analyse der Diagramme.

6 Lessons Learned

Durch eine gute Planung konnten wir das Projekt zwar im vollen Umfang durchführen, jedoch entstanden die besten Ideen meistens erst bei Fertigstellung des Projekts. Als Lessons Learned konnten wir schließen, dass wir das Projekt früher hätten beginnen sollen.