

Projektrapport for kurs Matematisk Modelering MA1487 HT24

Olga Egorova, oleg22

Introduction

I projektet förväntas vi att plocka data från en open API, beskriva hämtade data och berbeta de med olika statistiska metoder. Datamängder för projektet hämtas från SMHI Open Data API Docs - Meteorological Observations. Detta API tillhandahåller meteorologiska data som beskriver väderförändringar på olika platser i Sverige. API:et erbjuder flera olika typer av mätningar, inklusive temperatur (parameter 1) och relativ luftfuktighet (parameter 6). Dessa parametrar kan användas för att, till exempel, förstå klimatförhållanden och variationer över tid.

För projektet ville jag hämta begränsad mängd av data. Eftersom temperatur och relativt luftfuktighet mäts varje timme, data från sista tre dygn ska innehålla 72 mätningar. Det mängd av data är översiktligt och tillräckligt stor för flera statistiska metoder. Därför välde jag dessa parametrar för analys. I arbete används data från tre meteorologiska stationer: Halmstad flygplats, Uppsala Flygplats och Umeå Flygplats. Stationer väljs eftersom båda valda parametrar finns att hämta i API. Flera andra stationer hade inte data för dessa parametrar.

Temperaturen mäts i grader Celsius ($^{\circ}\text{C}$), medan relativ luftfuktighet anges i procent (%). Genom att använda dessa data kan jag utföra olika typer av statistiska analyser, inklusive visualiseringar av frekvensfördelningar, beräkningar av medelvärden och standardavvikelser, samt jämförelser mellan stationer och parametrar. Detta arbete syftar till att demonstrera hur statistiska metoder kan användas för att bearbeta och tolka meteorologiska data.

Uppgift 1. Databeskrivning

All kod som jag använder för att hämta och bearbeta data finns i GitHub. För att hämta en ny dataset, skaffa alla tabeller och figurer tillräckligt att använda fyll 'ALL_CODE.py' Datamängder plockas med följande kod:

```
import json

# This part i inactivated because i am working with downloaded data
# Downloads data from three stations and for two parameters
for key in PARAMS.keys():
    for station, id in STATIONS.items():
        data_url = (f'https://opendata-download-metobs.smhi.se'
                    f'/api/version/1.0/parameter/'
                    f'{key}/station/{id}/period/{PERIOD}/data.json')
```

```

response = requests.get(data_url)
response.raise_for_status() # Check if the request succeeded

result = json.loads(response.content)
save_path = f'data/{id}_{key}.json'
with open(save_path, "w", encoding="utf-8") as file:
    json.dump(result, file, indent=4, ensure_ascii=False)

```

Som resultat genereras en JSON-fil för varje kombination av station och parameter. Varje fil innehåller över 2500 mätpunkter, vilket ger en omfattande datamängd för analys. För statistisk bearbetning används data från de senaste 72 timmarna. Urvalet av data görs med hjälp av följande kod:

```

import datetime
import json
import pytz
import numpy as np

# variables
STATIONS = {
    'Halmstad flygplats': 62410,
    'Uppsala Flygplats': 97530,
    'Umeå Flygplats': 140480
}

# parameters to download (parameter_id:parameter_name)
PARAMS = {1:["TEMPERATUR", "°C"], 6:["LUFTFUKTIGHET", "%"]}
# Extract the required period (three days) from downloaded data
measured_points = 72 # Number of points to include
three_days = {}
data_rows = []

# Process data for each parameter and station
for param_id, parameter in PARAMS.items():
    three_d_station = {}
    for name, station_id in STATIONS.items():
        file_path = f'data/{station_id}_{param_id}.json'
        with open(file_path, 'r') as file:
            data = json.load(file)

        # Sort data by timestamp and select the last N points
        sorted_data = sorted(
            data.get("value", []),
            key=lambda x: datetime.fromtimestamp(x["date"] / 1000,
            tz=pytz.timezone("Europe/Stockholm"))
       )[-measured_points:]

```

```

# Prepare station data and append rows for further processing
stat_set = {}
for item in sorted_data:
    check = item['quality'] in ['G', 'Y']
    new_value = float(item['value']) if check else np.nan
    stat_set[item['date']] = new_value
    data_rows.append({
        'time': datetime.fromtimestamp(item['date'] / 1000,
                                         tz=pytz.timezone("Europe/Stockholm")),
        'station_name': name,
        'parameter': PARAMS[param_id][0],
        'value': new_value
    })
    three_d_station[name] = stat_set
three_days[param_id] = three_d_station

```

Det skaffas två objekt med samma innehåll: 1. 'tree-days' - En nästlad ordbok (dictionary) som innehåller filtrerad och sorterad data för de senaste 72 timmarna för varje parameter och station. Strukturen ser ut så här:

```

{
    param_id: { # Parameterens ID
        station_name: { # Stationens namn
            timestamp: värde, # Tidsstämpel och tillhörande mätvärde
            ...
        },
        ...
    },
    ...
}

```

Nycklar: param_id (parameter-ID) och station_name (stationens namn). Värden: För varje station skapas en ordbok där nyckeln är en tidsstämpel och värdet är ett numeriskt mätvärde (eller numpy.nan om kvaliteten inte är godkänd). Det objekt används i flesta fall. 2. 'data_rows' - En lista med rader där varje rad är en ordbok med information om en specifik mätpunkt. Strukturen ser ut så här: [{ 'time': datetime-objekt, # Tidsstämpel som datetime-objekt 'station_name': 'stationens namn', # Namn på mätstationen 'parameter': 'parameterens namn', # Parameterens namn (t.ex. temperatur eller luftfuktighet) 'value': numeriskt värde # Mätvärde eller numpy.nan }, ...] Sista objektet är lättare att omvandla till pandas objekt. För att lättare operera med data jag skaffar också kombinerad objekt 'df_three'.

```

import utils # some funktionalitet

# Convert the list of dictionaries into a pandas DataFrame objekt
df_three = pd.DataFrame(data_rows)
df_three.columns.str.replace(' ', '\n')

```

```
# save to markdown file to be able show in the presentation
utils.save_to_mdfile(df_three, 'dataframe.md', 'statistics')
```

Urval av hämtade datamängd presenteras i Tabel 1. Fullständigt tabell finns på GitHub

Tabel 1. Sammansätta data

	time	station_name	parameter	value
0	2024-12-15	Halmstad	TEMPERATUR	7.8
	18:00:00+01:00	flygplats		
1	2024-12-15	Halmstad	TEMPERATUR	8.1
	19:00:00+01:00	flygplats		
132	2024-12-18	Uppsala	TEMPERATUR	-3.7
	06:00:00+01:00	Flygplats		
133	2024-12-18	Uppsala	TEMPERATUR	-3.2
	07:00:00+01:00	Flygplats		
214	2024-12-18	Umeå Flygplats	TEMPERATUR	-3.4
	16:00:00+01:00			
215	2024-12-18	Umeå Flygplats	TEMPERATUR	-3.1
	17:00:00+01:00			
216	2024-12-15	Halmstad	LUFTFUKTIGHET	98
	18:00:00+01:00	flygplats		
217	2024-12-15	Halmstad	LUFTFUKTIGHET	95
	19:00:00+01:00	flygplats		
429	2024-12-18	Umeå Flygplats	LUFTFUKTIGHET	95
	15:00:00+01:00			
430	2024-12-18	Umeå Flygplats	LUFTFUKTIGHET	95
	16:00:00+01:00			
431	2024-12-18	Umeå Flygplats	LUFTFUKTIGHET	96
	17:00:00+01:00			

För att testa om vissa tidpunkter på en av stationer saknar av mätningar används följande kod:

```
# Count NaN values per station_name and parameter
nan_counts = df_three.groupby(['station_name', 'parameter'])['value'].apply(
    lambda x: x.isna().sum()
).reset_index()

# Give name for columns
nan_counts.columns = ['station_name', 'parameter', 'Missing values']
utils.save_to_mdfile(nan_counts, "nan_counts.md", "statistics")
```

I data sim jag använder inga mätningar saknas (Tabel 2.):

Tabel 2. Missade data för alla parameter:

	station_name	parameter	Missing values
0	Halmstad flygplats	LUFTFUKTIGHET	0
1	Halmstad flygplats	TEMPERATUR	0
2	Umeå Flygplats	LUFTFUKTIGHET	0
3	Umeå Flygplats	TEMPERATUR	0
4	Uppsala Flygplats	LUFTFUKTIGHET	0
5	Uppsala Flygplats	TEMPERATUR	0

Uppgift 2. Beskrivande statistik

För att snabbt räkna ut statistiska egenskaper jag använder describe() metod för pandas objekt.

```
descriptive_stats = df_three.  
    groupby(['station_name', 'parameter'])['value'].describe()
```

Resultat presenterad i Tabel 3:

Tabell 3. Beskrivande statistik för alla stationer och parametrar

station, parameter, enheter	count	mean	std	min	25%	50%	75%	max
Halmstad flygplats, LUFTFUKTIGHET, %	72	91.47	5.98	75	90	93	96	99
Halmstad flygplats, TEMPERATUR °C	72	6.91	0.93	4.4	6.38	7	7.43	8.9
Umeå Flygplats, LUFTFUKTIGHET, %	72	88.38	4.1	81	85	88	91.25	96
Umeå Flygplats, TEMPERATUR, °C	72	-	5.68	-	-	-	-	-1.3
Uppsala Flygplats, LUFTFUKTIGHET, %	72	78.01	14.14	57	64	77.5	87.25	100
Uppsala Flygplats, TEMPERATUR, °C	72	1.27	2.48	-4.7	0.18	1.9	2.72	6.6

Tabellen att alla stationer har 72 mätningpunkter för båda parameter. Medelvärde avviker inte mycket från medianen med undantag för temperatur i Umeå. Första och tredje kvartiler avstar ganska lika mycket från median, men minimala och maximala värde avstar inte lika mycket från median.

Det är svårt att säga om data är normalfördelat enbart från resultat av tabellen. Jag skaffar därför plottar som visar fördelningen.

Uppgift 3. Beskrivande plottar

Beskrivande statistik kan visualiseras med hjälp av ladogrammar, som visar medelvärde, kvariler, 95% spridningen och avvikande värde i urvalet.

Följande kod skaffar ladogrammer för varje station och parameter. Jag välde att köra Shapiro-Wilk test och adda resultat till ladogrammer, ser Figur 2. Shapiro-Wilk test en av mest använda tester för att jämföra urvalet med normalfördelat mängden med samma parametrar. Vid p-värde mindre än 5% kan sägas att urvalets data inte normalfördelade.

```
# Arrayer to iterate through
stations = df_three['station_name'].unique()
parameters = df_three['parameter'].unique()

# Set up the figure
fig, axes = plt.subplots(2, 3, figsize=(12, 4 * 2)) # 2 rows, 3 columns

# Array to save results of the Shapiro-Wilk test
ShapiroW = []
# Loop over stations and parameters
for i, parameter in enumerate(parameters):
    for j, station in enumerate(stations):
        # check stations name
        condition_station = df_three['station_name'] == station
        # check parameter
        condition_parameter = df_three['parameter'] == parameter
        # mask to filter
        mask = condition_station & condition_parameter
        data_filtered = df_three[mask]

        # Perform Shapiro-Wilk normality test
        stat, p_value = sci.shapiro(data_filtered['value'])
        ShapiroW.append({
            'Station': station,
            'Parameter': parameter,
            'Shapiro-Wilk Statistic': round(stat, 5),
            'P-value': round(p_value, 5),
            'Normal Distribution (p > 0.05)': 'Yes' if p_value > 0.05 else 'No'
        })
        ax = axes[i, j]

# the boxplot
```

```

sns.boxplot(
    ax=ax,
    data=data_filtered,
    x='station_name', # Same station on x-axis
    y='value',
    palette=[COLORS[j]], # Assign unique color for the station
    width=0.3,
    dodge=False
)
# y-labels
ax.set_ylabel((f'{parameter}'
               f'{'°C' if parameter == 'TEMPERATUR' else '%'}'),
              fontsize=8
              )
# Remove the x-axis label
ax.set_xlabel('')

# Annotate p-value on the plot
ax.text(
    0.8,
    0.13,
    (f'p={p_value:.5f}'),
    transform=ax.transAxes,
    fontsize=10,
    ha='center',
    color='red' if p_value < 0.05 else 'black'
)
plt.suptitle("Databeskrivning med ladogrammar", fontsize=16)

# Adjust layout to make space for the title
plt.tight_layout(rect=[0, 0, 1, 0.95])

plt.savefig('img/box_plot/all.png')
plt.show()

```

Förklaring till Figur 1. Figuren visar boxplottar för olika stationer och parametrar. De parametrar som visas och respektiva stationer skrivs under varje subplot. Motsvarande enheter anges med etiketter till y-axes. Den horisontal linjen i varje box representerar medianen, uppe och nedre gränsen på boxarna visar det interkvartila intervallet (IQR) och morrhåren sträcker sig till minimi- och maximivärdena inom $1,5 * \text{IQR}$. Små sirklar visar avvikande värde. För varje boxplott anges ett resultat från Shapiro-Wilk-testet, den hjälper att bedöma om data följer en normalfördelning. Ett p-värde under 0,05 indikerar att data inte följer en normalfördelning, och detta markeras med rött i diagrammet.

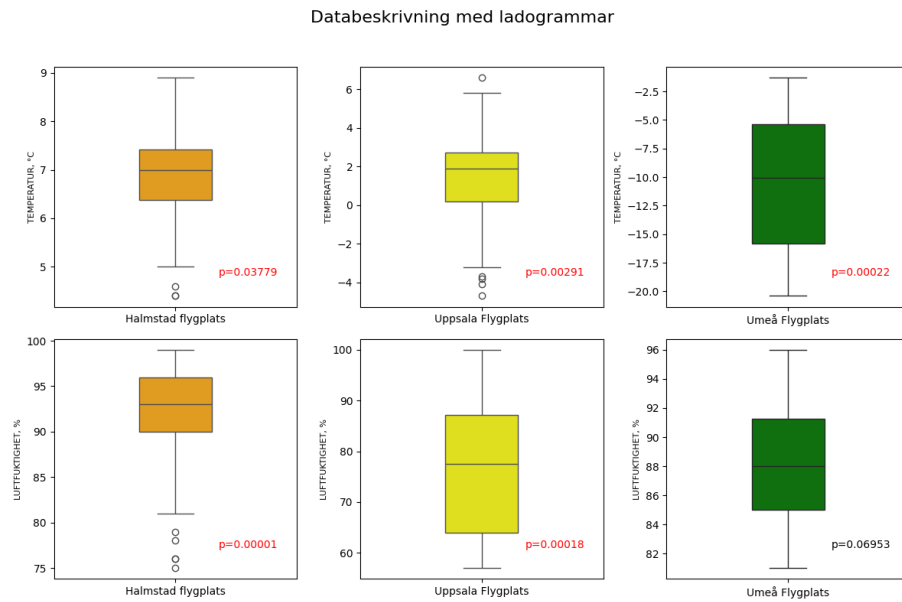


Figure 1: Ladogrammar.

Tabell 4. Resultater av Shapiro-Wilk test

Station	Parameter	Shapiro-Wilk Statistic	P- value	Normal Distribution (p > 0.05)
Halmstad flygplats	LUFTFUKTIGHET	0.884	0	No
Halmstad flygplats	TEMPERATUR	0.964	0.038	No
Umeå Flygplats	LUFTFUKTIGHET	0.969	0.07	Yes
Umeå Flygplats	TEMPERATUR	0.92	0	No
Uppsala Flygplats	LUFTFUKTIGHET	0.918	0	No
Uppsala Flygplats	TEMPERATUR	0.944	0.003	No

Med dessa plottar och Shapiro-Wilk test testar jag nulhypotes: *är data normalfördelad?* Ldogrammar och Shapiro-Wilk test för normality tillåtar förkasta nulhypotes om att temperatur spridning är normal fördelad. Sannolikheten att nulhypotes stämmer är mindre än 5% och därmed för alla tre platsar och därmed är sannolikhet för typ II fel (att felaktigt förkasta null hupotes) är ganska liten.

Samma påstående stämmer för relativt lyft fuktighet med undantag för relativt luftfuktighet i Umeå flygplats, där null hypotes kan inte förkastas utan hög risk på fel numer 2, eftersom p-värde 7% och högre än 5%.

Fördelning av data i urvalet kan visualiseras även med stapeldiagram. Figur med stapeldiagrammar skapas med följande koden:

```
# initiate figure
plt.figure(figsize=(8, 6))

# Prepare the custom blue square legend handle
text = 'Blue color shows samples distribution'
blue_square = Line2D([0], [0],
                    marker='s',
                    color='w',
                    markerfacecolor='blue',
                    markersize=8,
                    label=text
                    )

# Prepare the legend for the normal distribution
normal_dist_line = Line2D([0], [0],
                        color='orange',
                        lw=2,
                        label="Normal Distribution"
                        )

# variable to chose what norm dist line vill be shown in the legend
normal_dist_added = False
# Iterate through all stations and parameters
for i, station in enumerate(stations):
    for j, parameter in enumerate(parameters):
        condition_station = df_three['station_name'] == station
        condition_parameter = df_three['parameter'] == parameter
        mask = condition_station & condition_parameter
        # filter data for each station and parameter
        data = df_three[mask]

# Subplot indexing: 3 rows for 3 stations and 2 columns for 2 parameters
plt.subplot(3, 2, i * len(parameters) + j + 1)

sns.histplot(data['value'],
            kde=True,
            bins=24,
            color="blue",
            edgecolor="black")
```

```

# Calculate the mean and standard deviation
mean = data['value'].mean()
std_dev = data['value'].std()

# Generate x values for normal distribution (range around the data's values)
x = np.linspace(data['value'].min(), data['value'].max(), 100)

# Calculate the normal distribution values (PDF)
y = sci.norm.pdf(x, mean, std_dev)
# Add normal distribution with the same parameters to the subplot
plt.plot(x, y * len(data) * (x[1] - x[0]), color='orange')

# add title and axes
plt.title(f"{station}", fontsize=10)
# Conditionally set the xlabel depending on the parameter
if parameter == 'TEMPERATUR':
    plt.xlabel(f"{parameter.lower()} (°C)")
else:
    plt.xlabel(f"{parameter.lower()} (%)")

plt.ylabel("Frekvens")
# Create a global legend outside the subplots (top)
fig = plt.gcf() # Get the current figure

fig.legend(handles=[blue_square,
                    normal_dist_line],
            loc='upper center',
            bbox_to_anchor=(0.5, 0.99),
            ncol=2,
            fontsize='small'
            )

plt.tight_layout()
plt.subplots_adjust(top=0.85)
# Save and show the plot
plt.savefig("img/frekvenser/alla.png")

```

Grafiska fördelningar visas i Figur 2.

Förklaring till Figur 2. Den här figuren visualiserar frekvensfördelningen av temperatur- och relativ luftfuktighetsdata från flera stationer i Sverige, inklusive Halmstad Flygplats, Uppsala Flygplats och Umeå Flygplats. Fördelningen visas som histogram med Kernel Density Estimation (KDE), och varje subplot motsvarar en kombination av station och parameter. De blå staplarna representerar frekvensfördelningen av mätningarna, där varje stapel representerar ett specifikt värdeintervall. Blå linjen visar Kernel density estimation, eller den up-

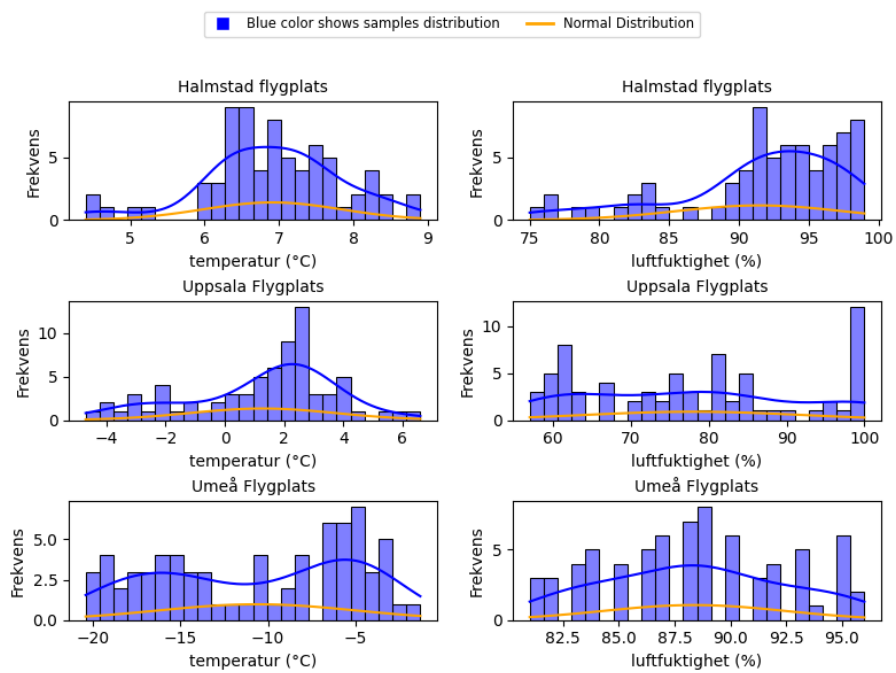


Figure 2: Frekvensfördelning.

pskattade sannolikhetsdensiteten för mätvärdena. Normalfördelningskurvan är en orange linje. Denna kurva beräknas med hjälp av medelvärde och standardavvikelsen för värdena i datasetet för varje station och parameter. Normalfördelningen läggs till för att visuellt jämföra hur den faktiska datadistributionen överensstämmer med den teoretiska normalfördelningen.

Figurer 1 och 2 visar att spridningen i alla datamängder avviker från Normal-spridningen. Shapiro-Wilk dock säger att dataspridning av relativt luftfuktighet i Umeå närmar sig mest till normalt. Stapelldiagrammar visar också att relativt luftfuktighet förändras inte likadant med temperaturförändring vid varje station. Här ifrån tar jag slutsatsen att det är inte korrekt att utföra statistiska tester på sammansatta data från alla stationer.

Q-Q plottar

Det finns ett annat sätt att visualisera avvikelse från eller liknande till normalfördelning, nämligen kvantil_kvantil plot(1). Q-Q plottar skaffas ed följande koden:

```
fig, axes = plt.subplots(2, 3, figsize=(10, 3 * 2))
# Loopa through all stations and parameters
for i, station in enumerate(stations):
    for j, parameter in enumerate(parameters):
        condition_station = df_three['station_name'] == station
        condition_parameter = df_three['parameter'] == parameter
        mask = condition_station & condition_parameter
        # Filter for station and oarameter
        data = df_three[mask]
        numeric_data = data['value'].dropna()
        # Create Q-Q plots
        ax = axes[j, i]
        sci.probplot(numeric_data, dist="norm", plot=ax)
        l_unit = 'temperatur, °C' if parameter == 'TEMPERATUR' else 'humidity, %'
        ax.set_ylabel(l_unit, fontsize=8)
        # Add titel
        ax.set_title(f"Q-Q plot: {station} - {parameter}", fontsize=8)
        ax.get_lines()[1].set_color('red') # Give lene for the
plt.tight_layout()
plt.savefig('img/q_q_plot/all.png')
plt.close()
```

Resultat visas på Figur 3.

Förklaring till Figur 3. På plottar jämförs dataset från olika stationer och parametrar mot den teoretiska normalfördelningen. En Q-Q plot (Quantile-Quantile plot) jämför de empiriska kvantilerna från den faktiska datan med

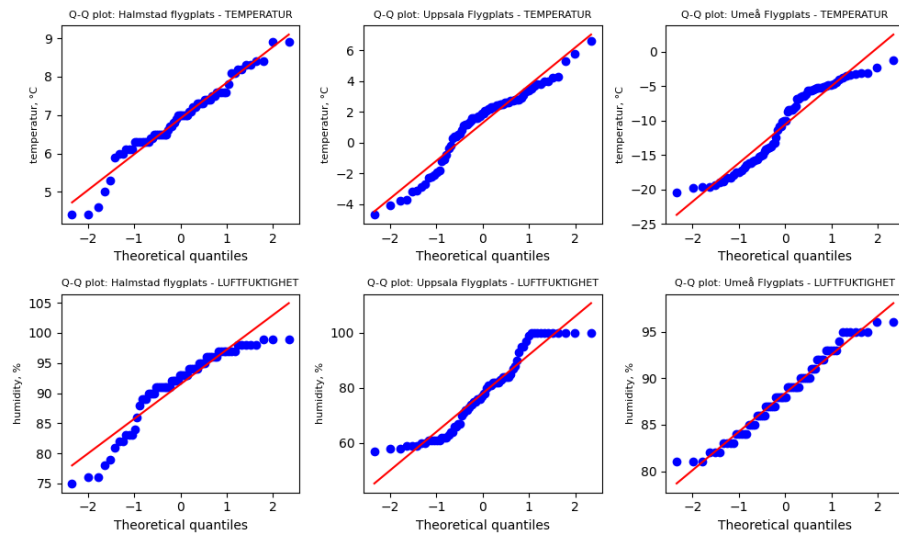


Figure 3: Q_Q plottar.

de teoretiska kvantilerna från en normalfördelning. I På X-axeln visas normalfördelnings kvantiler, på Y-axeln visas kvantiler från respektiv datamängd (Tabel 3a[b][### Tabel 3b])

Tolkning av figuren:

- **Punkterna ligger nära en rak linje:** Detta tyder på att datan är nära en normalfördelning.
- **Punkterna avviker från linjen:**
- **Punkterna böjer sig uppåt vid svansarna:** Detta kan indikera att datan har för mycket extrema värden, vilket tyder på en tyngre svans än normalfördelningen (t.ex. en t-fördelning).
- **Punkterna böjer sig nedåt vid svansarna:** Detta kan tyda på att datan har för få extrema värden och inte har lika många extremvärden som en normalfördelning.
- **Punkterna är ojämnt fördelade eller böjer sig i mitten:** Detta kan indikera en snedvridning (skewness) i datan, vilket innebär att den inte är symmetrisk.

Denna typ av figur används för att snabbt bedöma om datan följer en normalfördelning, vilket kan vara användbart i statistiska tester eller när du ska välja lämpliga modeller för analysen.

Liksom tidigare testar visar figur att närmast till normalfördelningen är data från relativt luftfuktighet i Umeå flygplats.

Jag testade att ta bort mest avvikande värde från dataset för att se om det hjälper att nå normalfördelning. visar om extrema värde tas bort, men fördelningen fortfarande avviker från normalt (data inte visad).

Uppgift 4: Linjär regression

För att se med vilka data ska jag arbeta vill jag först titta på hur data korrelerar med varandra. Därför skaffar jag några plottar, som visas på Figur 4.

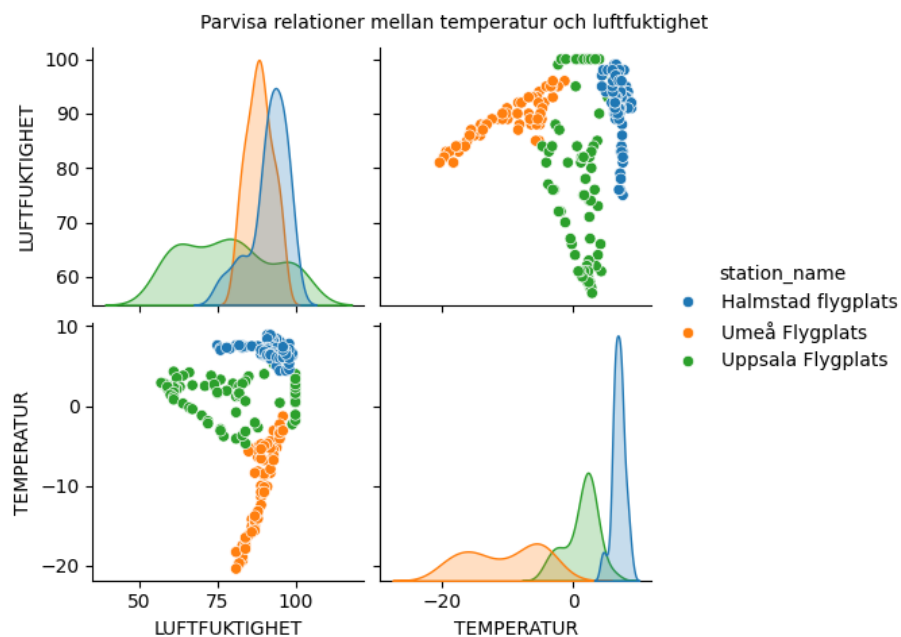


Figure 4: Korrelationer.

Figuren visar att temperatur och luftfuktighet i Umeå flygplats korrelerar, men det kan inte sägas att luftfuktighet och temperatur korrelerar i två andra stationer. Plot var skaffat med följande kod:

```
# Pivot three_days
pivote_df = df_three.pivot_table(index=['time', 'station_name'],
                                  columns='parameter',
                                  values='value').reset_index()

# create paired plot
sns.pairplot(pivote_df, hue='station_name')
plt.subplots_adjust(top=0.9)
```

```

plt.suptitle("Parvisa relationer mellan temperatur och luftfuktighet",
             fontsize=10,
             y=0.95)
plt.savefig("img/correlation/param_param.png")
plt.show()
plt.close()

```

Jag skaffar även ett annat plot, som visar mera detaljer:

```

# Pairplot
fig = sns.pairplot(combined_data, height=1.8, diag_kind='kde')
# Adjust font size and axis
for ax in fig.axes.flatten():
    # Get current x and y axis labels
    xlabel = ax.get_xlabel()
    ylabel = ax.get_ylabel()

    # Remove "flygplats" (case-insensitive)
    xlabel = re.sub(r'(?i)flygplats', '', xlabel).strip()
    # Replace "TEMPERATUR_" with "TEMP_"
    xlabel = xlabel.replace("TEMPERATUR_", "°C, TEMP_").strip()
    # Replace "LUFTFUKTIGHET_" with "FUKT_"
    xlabel = xlabel.replace("LUFTFUKTIGHET_", "%, FUKT_").strip()

    # Remove "flygplats" (case-insensitive)
    ylabel = re.sub(r'(?i)flygplats', '', ylabel).strip()
    # Replace "TEMPERATUR_" with "TEMP_"
    ylabel = ylabel.replace("TEMPERATUR_", "°C, TEMP_").strip()
    # Replace "LUFTFUKTIGHET_" with "FUKT_"
    ylabel = ylabel.replace("LUFTFUKTIGHET_", "%, FUKT_").strip()

    # Set the modified labels with font size
    ax.set_xlabel(xlabel, fontsize=6)
    ax.set_ylabel(ylabel, fontsize=6)

    ax.set_ylabel(ylabel.replace("LUFTFUKTIGHET_", "%, FUKT_").strip(), fontsize=6)

    # Set font size for tick labels
    ax.tick_params(axis='x', labelsz=5)
    ax.tick_params(axis='y', labelsz=5)

plt.suptitle("Pairwise Relationships for Parameters and Stations",
             y=0.99,
             fontsize=16)
plt.subplots_adjust(hspace=0.2, wspace=0.2, top=0.9)
plt.savefig('img/correlation/all_pairwise_relationships.png')
plt.show()

```

```
plt.close()
```

Parvisa relationer för parametrar och stationer

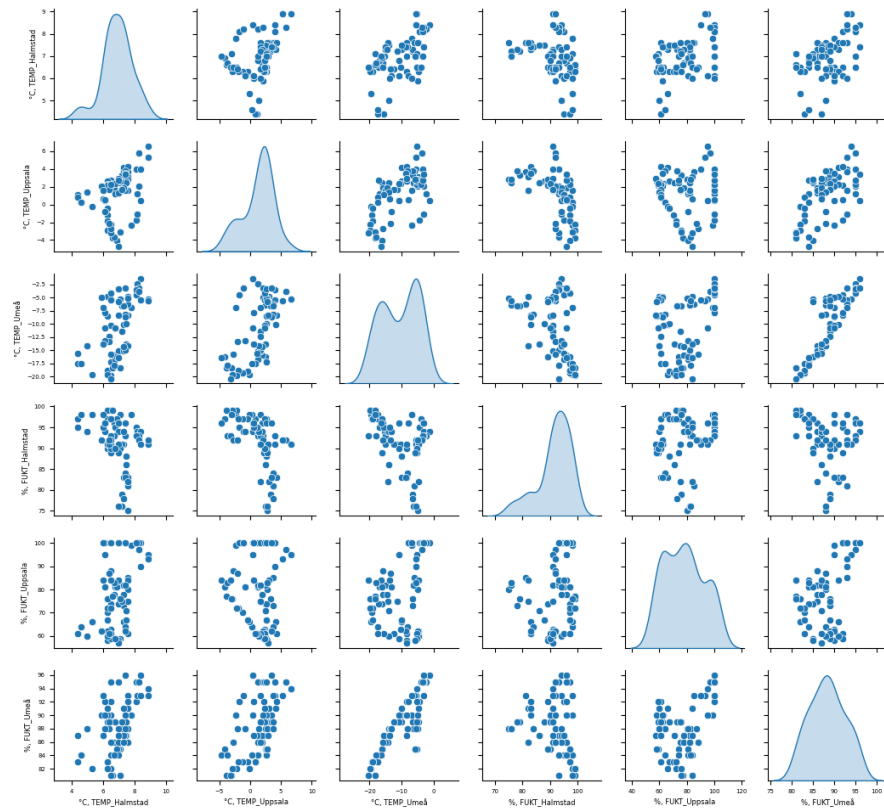


Figure 5: Alla möjliga parvisa korrelationer.

Detta plot visar igen, att det kan finnas direkt samband mellan relativt luftfuktighet och temperatur i Umeå. Jag skapar också matris som visar hur korrelerar en parameter från en station med alla andra parameter-station kombinationer.

```
# Calculate the correlation matrix
correlation_matrix = combined_data.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
ax = sns.heatmap(
    correlation_matrix,
    annot=True, # Avoid cluttering with too many annotations
    cmap=LinearSegmentedColormap.from_list("CustomCmap", COLORS, N=256),
```



```

cbar=True
)
# Adjust font sizes for station names
ax.tick_params(axis='x', labelsizes=8)
ax.tick_params(axis='y', labelsizes=8)
plt.title("Korrelationsmatris för alla parametrar och stationer", fontsize=14)
plt.tight_layout()
plt.savefig('img/correlations/all_correlations.png')
plt.show()
plt.close()

```

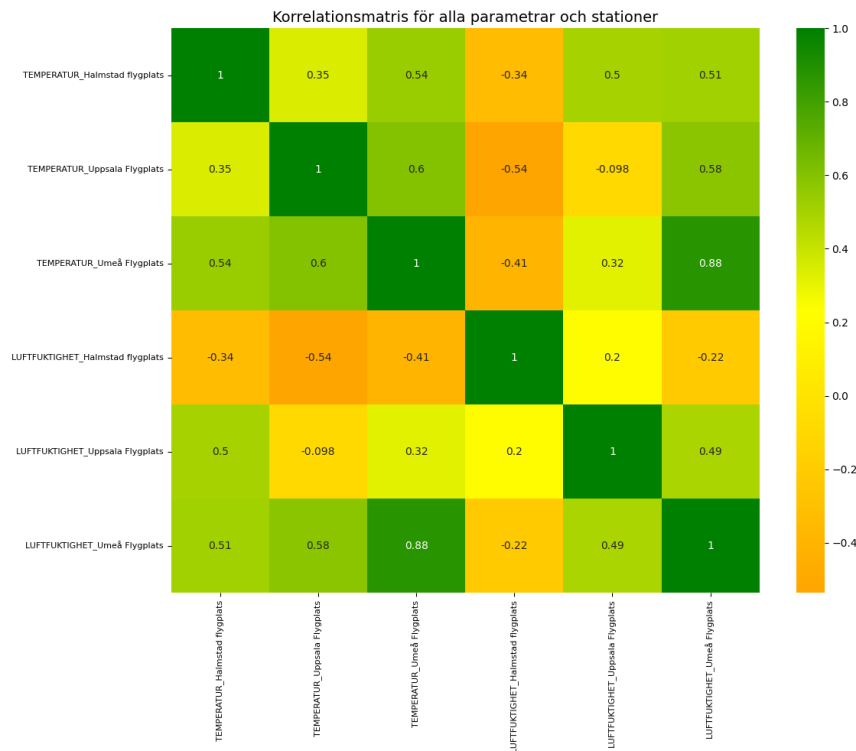


Figure 6: Korrelation matris

Förklaring till Figur 6. I figur visas korrelations koeffitienter mellan olika dataset. På axlar visas olika parameter-station kombinationer. Samma kombinationer visas på båda axlarna. Korrelationskoeffitienter mellan parar visas med text i färgade kvadrater. Färgskala visas till höger.

Korrelationmatrisen visar samma som två figurer innan (Figur 4 och Figur 5). Tydlig korrelation finns bara mellan relativt luftfuktighet och temperatur i Umeå flygplats. Därför väljer jag att utforska samband mellan relativt luftfuktighet

och temperatur bara i Umeå vidare.

Jag skaffar regressions modell med hjälp av maskinlearning. 50% av data används som tränings dataset de resterande 50% används som testdataset.

```
# Get training ang testing datasets
fraktion = 0.5
train = combined_data.sample(frac=fraktion, random_state=1)
test = combined_data.drop(train.index)

# # Extract X (independent variable) and y (dependent variable) from the dataframe
# Title for the plot
X_train = train[column_name1].values.reshape(-1, 1)
y_train = train[column_name2].values
X_test = test[column_name1].values.reshape(-1, 1)
y_test = test[column_name2].values

model = LinearRegression().fit(X_train, y_train)
pred = model.predict(X_test)

# Räkna ut MSE
mse = np.mean((pred - y_test)**2)
b = model.coef_[0]
a = model.intercept_

# Calculate residuals of test data
residuals = pred - y_test
stat_linjar, p_linjar = sci.shapiro(residuals)

# Use statsmodel for calculating confidens interval
# Lägg till konstant för intercept
X_train_with_const = sm.add_constant(X_train)
ols_model = sm.OLS(y_train, X_train_with_const).fit()
conf_int_params = ols_model.conf_int(alpha=0.05)

# calculate confidens interval
a_ci = conf_int_params[0] # Första raden: Intercept
b_ci = conf_int_params[1] # Andra raden: Lutning

plt.figure(figsize=(10, 6))
# Train data
plt.scatter(X_train, y_train, color="orange", label='Träningsdata', alpha=0.6)
# Test data
plt.scatter(X_test, y_test, color="blue", label='Testdata', alpha=0.6)

# Title
```

```

sns.regplot(x=column_name1,
            y=column_name2,
            data=combined_data,
            scatter=False,
            line_kws={'color': 'red', 'label': f'Y = {a:.2f} + {b:.2f} * X'},
            ci=95)

x_plot = np.linspace(-20, -1, 100).reshape(-1,1)
draw_plot = model.predict(x_plot)

# Regression line for predictions (testdata)
plt.plot(x_plot, draw_plot, color='g', label='Test Data Prediction', linewidth=2)

# show Regression equation and confidence interval
plt.text(0.5, 0.89,
        (f'Lineärmodel: y = {a:.2f} + {b:.2f}* X\n'
         f'95% konfidens interval för a: [{a_ci[0]:.2f}, {a_ci[1]:.2f}]\n'
         f'95% konfidens interval för b: [{b_ci[0]:.2f}, {b_ci[1]:.2f}]'),
        ha='center',
        va='center',
        transform=plt.gca().transAxes,
        fontsize=10,
        color='red'
        ((f'Shapiro-Wilk test for residualernas spridning: p = {p_linjar:.4f}, '
         f'\nstatistics={stat_linjar:.4f}\n'))
)
plt.title(f'Prognos av luftfuktighet baserat på temperatur i Umeå\n'
         f'Mean squared error: {mse:.2f}\nFraktion: {fraktion}')
plt.xlabel("Temperatur, °C")
plt.ylabel("Relativt Luftfuktighet, %")
plt.legend(loc='best', fontsize=8)
plt.savefig('img/regression/Conf_int_regr_prediction_Umea_temp_luft.png')
#plt.show()
plt.close()

```

Förklaring till figure 7. Figuren visar den linjära regressionsmodellen som förutsäger relativ luftfuktighet på Umeå flygplats på grund av temperatur. x-axeln visar temperatur i Celciumgrader, y-axeln visar relativ luftfuktighet i procent. Orange punkter visar data som använts för att ta fram prediktionsmodellen, blå punkter visar data som använts för att testa prediktionsmodellen. Röd linje representerar prediktionsmodellen och grön linje representerar den modell som skulle erhållas med testdataset. Ekvationen för prediktionsmodellen, 95% konfedencsintervaller för linjär slop och den fria element, samnt resultat av Shapiro-Wilk test för normalsprindning av residualerna, visas i röd text. Grön

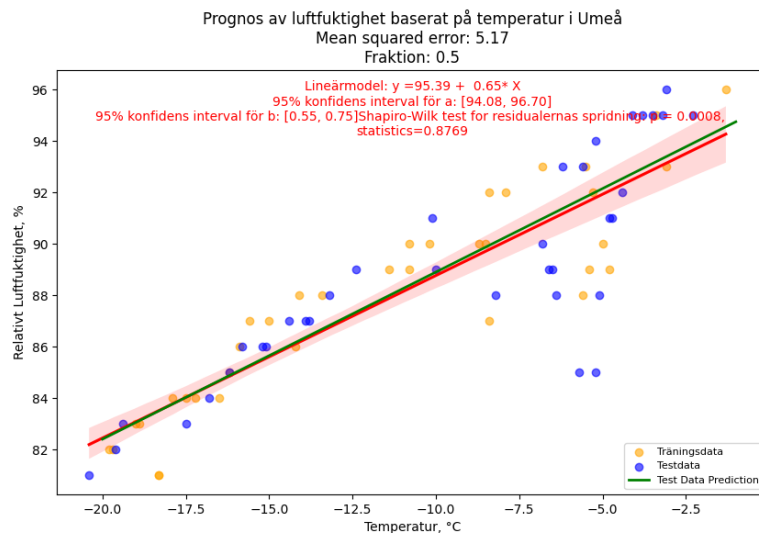


Figure 7: Linjäregression för relativt luftfuktighet i Umeå flygplats

linja visar korrelation model som skulle skapas om test dataset skulle användas i stället av trainings dataset.

Linjär regression är en linja $y = bx + a$. I vårt fall y föreställer alla data för relativt luftfuktighet, och x är datarray med temperaturer från respektive tidspunkter. b kalkuleras som $\text{sum}((x[i] - \text{medel}(x))(y[i] - \text{medel}(y)))/\text{sum}((x[i] - \text{medel}(x))^2)$ a kalkuleras som $\text{medel}(y) - b * \text{medel}(x)$ 95% interval kalkulering är korrekt ifall residualerna är normalfördelad. Shapiro-Wilk test visar att probabilitet för normalfördelning av residualer är väldigt litet, bara 0.08% som tillåter förkasta hypotes om normalfördelning av residualer. Det betyder att linjär model inte kan prediktera luftfuktighet på grund av temperatur.

För att skaffa bättre model i detta fall kan jag försöka att ta bort avvikande värde, eller transformera den beroende variable(2). Som det visas på ladogram, finns det inte värde som avviker mycket. Jag väljer därför att transformera den beroende variabel, relativt luftfuktighet.

Uppgift 5: Transformera data

Jag testar om logmodifiering av relativt luftfuktighet kan hjälpa att skapa bättre model. Relativtluftfuktighet är alltid positivt, därför modifiering kan göras direkt.

```
# Regression with log relativt humidity
# Relative humidity is always pozitiv
y_train_log = np.log(y_train)
```

```

y_test_log = np.log(y_test)

# create liniar regression pf the log data
log_y_model = LinearRegression()
log_y_model.fit(X_train, y_train_log)

# Make prediction of the test data
#x_for_log_y = np.linspace(-1.0, 3.01, 100)
pred_log_y = log_y_model.predict(X_test)
pred_y_by_log = np.exp(pred_log_y)

b_Y_log = log_y_model.coef_[0]
a_Y_log = log_y_model.intercept_

# Create model line use x_plot for prediction
#x_for_log_y = np.linspace(-25,-3, 100)
drow_y_log_model = log_y_model.predict(x_plot)

# Calculate MSE
mse_log_y = np.mean(np.exp(pred_log_y - y_test_log)**2)
print(mse_log_y)
plot_text= ((f'MSE liniarregressoin, original dataset: {mse:.2f}\n'
             f'MSE logtransformerad y, dataset utan avvikande värde: {mse_log_y:.5f}'))

plt.scatter(X_train, y_train_log, label='Träningsdata')
plt.scatter(X_test, y_test_log, label='Test data')
plt.plot(x_plot, drow_y_log_model,
         label='Linjär regression log domän',
         color='g',
         linewidth=3
        )
plt.legend()
plt.text(-18.5, 4.51,
        f"log(Y) = {a_Y_log:.2f} + {b_Y_log:.2f} * X",
        fontsize=8,
        color="red"
       )
plt.title("Prognos av relativt luftfuktighet (log transformerad y, %)")
plt.xlabel("temperatur, °C")
plt.ylabel("relativt luftfuktighet [log %]")
plt.savefig('img/regression/log_new_transform_FUKT_Umeå_all.png')
plt.show()
plt.close()

# Create model line in the original domain

```

```

drow_y_log_model_exp = np.exp(log_y_model.predict(x_plot))

# Calculate MSE in original domain
mse_log_y = np.mean((pred_y_by_log - y_test)**2)
# Calculate residualer
residual_log_y = y_test - np.exp(pred_log_y)

p_log_y, stat_log_y = sci.shapiro(residual_log_y)
stderr_log_y = np.std(residual_log_y)
# dispersion skattning
n = len(residual_log_y)
mean_x = sum(X_test)/n
residual_variance = sum(r**2 for r in residual_log_y) / (n- 2)
square_sum = sum((X_test[i] - mean_x) ** 2 for i in range(n))[0]
# standard error för koeffitienter
se_b = math.sqrt(residual_variance / square_sum)
# Compute the denominator of the second term (se_a)
sum_squared_diffs = sum((X_test[i] - mean_x) ** 2 for i in range(n))

# Compute the second term in the variance formula
variance_term = mean_x**2 / sum_squared_diffs

# Compute the full variance factor
variance_factor = 1 / n + variance_term

# Compute the standard error
se_a = math.sqrt(residual_variance * variance_factor)

# Plotting
plt.scatter(X_train, y_train, label='Träningsdata (original scale)')
plt.scatter(X_test, y_test, label='Test data (original scale)')
plt.plot(x_plot,
         drow_y_log_model_exp,
         label='Log-y-transformerad regression i original domän',
         color='g',
         linewidth=3
        )

# t-värde för 95% interval
t_critical = sci.t.ppf(0.975, df=n - 2)
a_error = t_critical * se_a
b_error = t_critical * se_b
plot_text = ((f'y = exp({a_Y_log:.2f} + {b_Y_log:.2f} * X)\n'
              f'Shapiro-Wilk test, p = {p_log_y:.2f}\n'
              f'95% konfidensinterval för a: '
              f'{(a_Y_log - a_error):.2f} < a < '

```

```

f' {(a_Y_log + a_error):.2f} \n'
f'95% konfidens interval för b: '
f' {(b_Y_log - b_error):.2f} < b < '
f' {(b_Y_log + b_error):.2f} '
)
plt.legend(fontsize=8)
plt.text(-12.5, 81, plot_text, fontsize=8, color='red')
plt.title((f'Prognos av relativt luftfuktighet \n'
f'model är skapad med logtransformerad y och transformerad tillbaka'))
plt.xlabel("temperatur, °C")
plt.ylabel("relativt luftfuktighet [%]")
plt.savefig('img/regression/back_new_log_transform_FUKT_Umeå_All.png')
plt.show()
plt.close()

```

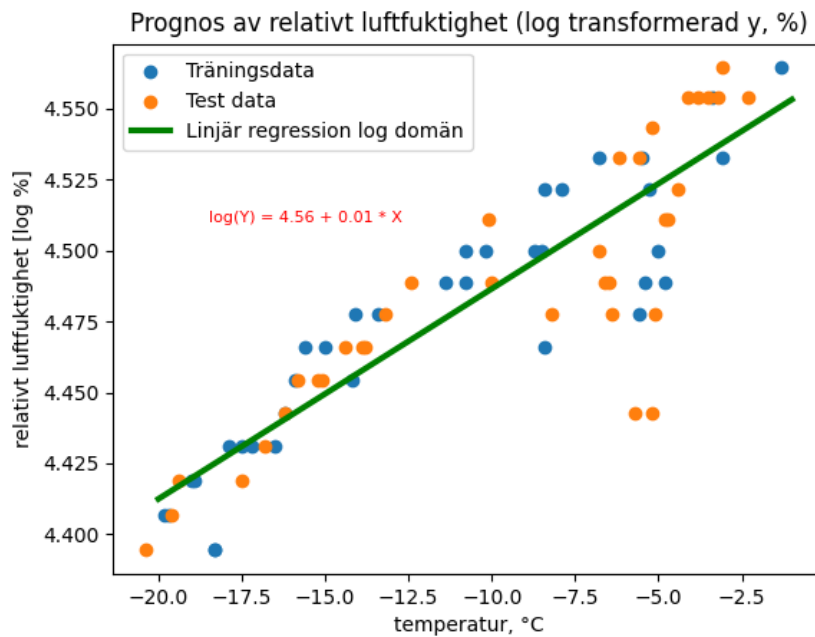


Figure 8: Här visas model prediktion med transformerade y-värde:

Förklaring för Figuren 8. Figuren visar linjärregression som predikterar logaritm av relativluftfuktighet på grund av temperatur. X-axel visar temperatur, Y-axel visar logritmerad relativt luftfuktighet. Ekvationen visas med röd text på plotten

Prognos av relativt luftfuktighet
 model är skapad med logtransformerad y och transformerad tillbaka

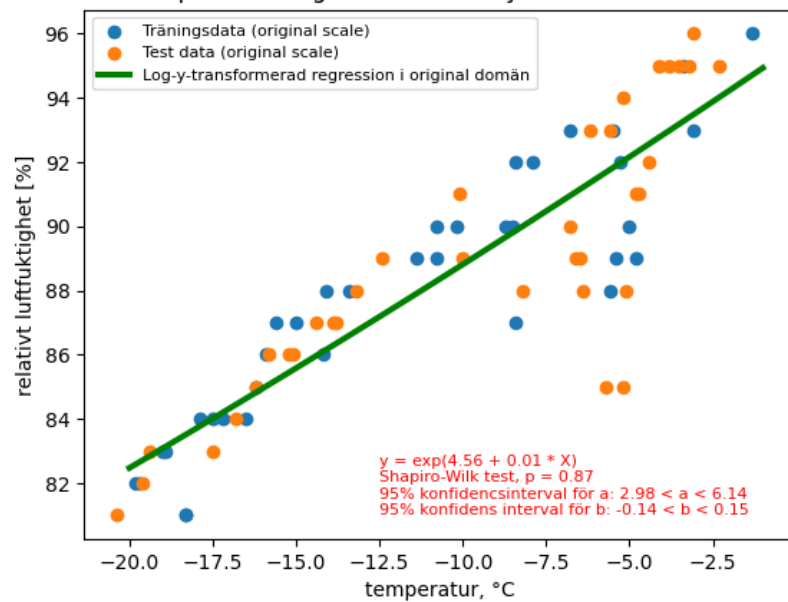


Figure 9: Rrediktion relativt-lyftfuktighets med originala fultighet värde leg-model

Förklaring för Figuren 9. Figuren visar linjärregression som predikterar logaritm av relativluftfuktighet på grund av temperatur. X-axel visar temperatur, Y-axel relativt luftfuktighet. Ekvationen visas med röd text på plotten. Det visas också 95% gränser för a och b koefficienter.

För att jämföra modeller skapar jag en plot som visar båda modeller:

```
plt.scatter(X_train, y_train, label='Träningsdata')
plt.scatter(X_test, y_test, label='Test data')
plt.plot(x_plot,
         draw_y_log_model_exp,
         label='Linjär regression exponentiell i log(y)',
         color='green',
         linewidth=3)
plt.plot(x_plot,
         draw_plot,
         label='Linjär regression originela data',
         color="red",
         linewidth=3)
plt.legend()
plot_text= ((f'MSE linjärregressoin, original dataset: {mse:.2f}\n'
             f'MSE logtransformerad y, dataset utan avvikande värde: {mse_log_y:.2f}'))
plt.title("Två modeller i en plot", fontsize=10)
plt.xlabel("temperatur, °C", fontsize=8)
plt.ylabel("relativt luftfuktighet, %", fontsize=8)
plt.text(-15.0, 80.8, plot_text, color="red", fontsize=8)
plt.savefig('img/regression/tva_modeller.png')
plt.show()
```

Förklaring för Figur 10. Figuren visar båda modeller: linjärregression och återtransformerad log-model. Det visas också MSE för båda modeller med röd text.

Sammanfattande resultat av transformationer visas på följande plot:

Förklaring till Figur 11. Figur visar tre modeller som predikterar relativt luftfuktighet med temperatur i Umeå Flugplats. För alla tre modeller användes 68 mätpunkter (data utan två högsta temperaturvärde och två lägsta temperatur vrde). Linjärregression visas med grön linjen, model skaffade med logtransformerade luftfuktighetsvärde visas som grön linja och model skaffad med logtransformerade temperaturvärde visas som en röd linja. MSQ för alla modeller visas med röd text. Första MSE värde kommer från linjärregressions modellen som var skaffad på grun av original dataset. Det är tydligt att logaritmisk transformation av temperaturvärde hjälper inte skaffa bättre modellen. MSE från sisenämnda modellen är betydligt högre än MSE från två andra modller.

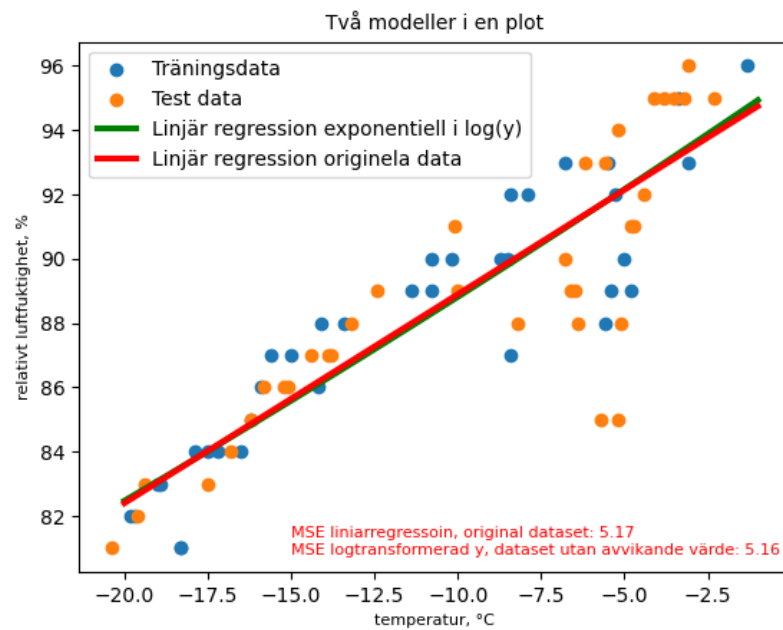


Figure 10: Två modeller

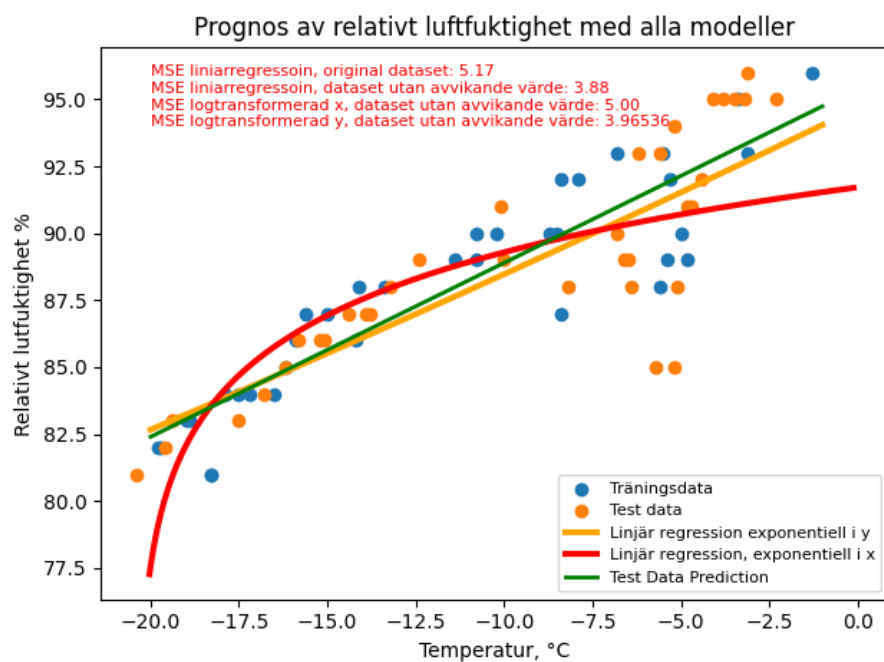


Figure 11: Tre modeller med transformerade data utan data som innehåller två högsta och två lägsta temperaturvärde

Sammanfattning

I projektet analyserades data från tre meteorologiska stationer i Sverige. Programmen tillåter hämta data och spara de i separata filler. Från filer data kan hämtas för bearbetning. Data för temperatur och relativt lyftfuktighet från tre dygn (72 timmar) analyserades. Resultater visar att fördelningen av datamängden inte liknar normalfördelningen. Korrelationstestar visar tillåter göra slutsats att luftfuktighet för det mesta korrelerar inte liniart med temperatur. Det enda starka korrelation fanns mellan relativt lyftfuktighet och temperatur i Umeå. Det skapades två olika regressionsmodeller: direkt liniärregression och liniärregression med logaritmerad relativt lyftfuktighet. Den andra modellen är mera "grundande" eftersom fördelning av residualerna kan räknas som normalt. Visualisering av modeller visar att de liknar varandra och därför kan liniärregressions modellen vara mera preferabel på grund av sin enkelhet.

Använd litteratur:

1. Körner, S., & Wahlgren, L. (2006). Statistisk dataanalys (4., [omarb.] uppl.). Studentlitteratur.
2. Wilk, M. B., & Gnanadesikan, R. (1968). Probability plotting methods for the analysis of data. *Biometrika*, 55(1), 1–17.
3. Osborne, J., (2002) "Notes on the use of data transformations", *Practical Assessment, Research, and Evaluation* 8(1): 6. doi: <https://doi.org/10.7275/4vng-5608>