Vedlegg 2 - Databehandling

Gruppe 3 - Høst 2021

Modellering av solfanger. Her lages en model basert på data fra mars til juli 2020. Med gjeldene krav fra ASHRAE-93.
De predikerte verdiene er plottet mot målte verdier, og samlignet med en benchmarkmodell med 50% virkningsgrad.
Nøkkeltall:
Datapunkter i 2020: 586
$RMSE_{model} = 16.460$
$RMSE_{benchmark} = 28.095$

In [1]:
```
# Import av nødvendige moduler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math

from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
```

In [2]:
```
# Laster inn datasettet for 2020
data = pd.read_csv("Måledata_mars_juli_2020.csv")
data.Time = pd.to_datetime(data.Time)
data
```

Out[2]:

	Time	Irradiance (W/m2)	Ambient Temperature (C)	Inlet Temperature (C)	Mass flow (g/s)	Outlet Temperature (C)
0	2020-03-01 00:01:00	0.0	0.4	15.6	107.0	23.6
1	2020-03-01 00:02:00	0.0	0.4	15.6	107.0	23.6
2	2020-03-01 00:03:00	0.0	0.4	15.6	107.0	23.6
3	2020-03-01 00:04:00	0.0	0.4	15.6	107.0	23.6
4	2020-03-01 00:05:00	0.0	0.4	15.6	107.0	23.6
...	...	...	...	...	...	...
220314	2020-07-31 23:55:00	-1.0	17.7	25.5	0.0	26.9
220315	2020-07-31 23:56:00	-1.0	17.8	25.5	0.0	26.9
220316	2020-07-31 23:57:00	-1.0	17.8	25.5	0.0	26.9
220317	2020-07-31 23:58:00	-1.0	17.8	25.5	0.0	26.9
220318	2020-07-31 23:59:00	-1.0	17.8	25.5	0.0	26.9

220319 rows × 6 columns

Krav for irradians

In [3]:
```
# Lagrer datapunkter med irradians som møter standarden ASHRAE-93:
def find_irradiance(data):
    try:
        data_irra = data[data["Irradiance (W/m2)"] > 790]
    except:
        data_irra = data[data["Irradience (W/m2)"] > 790]
    return data_irra.to_numpy() # Lagrer datapunktene som numpy-array
```

Overlappende 15-minutters intervaller

In [4]:
```
# Deler datasettet med høy nok irradians inn i 15-minutters intervaller:
def find_15_min_intervall(np_data):
    info = []
    for i, (tid, irradiance, Ta, Ti, ms, To) in enumerate(np_data[15:]):
        if tid - np_data[i][0] == pd.Timedelta("15m"):
            # save all 15 minutes inside infomration
            info.append(np_data[i:i+15])
    info = np.array(info)
    return info # Lagrer 15-minutters intervallene i en numpy-array.
```

Krav for steady-state

$Irradiance \pm 32$
$T_a \pm 1.5$
$T_i \pm 1$

Irradians, T_a og T_o

In [5]:
```
# Finner alle intervaller som tilferedsstiller krav om steady state i hhv. irradians,
# omfivelsestemperatur (Ta), og vanntemperatur ut av kollektr (To).
def steady_state_krav(info):
    innenfor_krav = []

    for intervall15min in info:
        ir_mean, Ta_mean, Ti_mean, _, To_mean = (np.mean(intervall15min[:,1:], axis=0))
        if (np.max(abs(ir_mean - intervall15min[:,1])) <= 32 and
            np.max(abs(Ta_mean - intervall15min[:,2])) <= 1.5 and
            np.max(abs(To_mean - intervall15min[:,5])) <= 1 and
            np.min(intervall15min[:,4]) >= 50):

            innenfor_krav.append(intervall15min)
    innenfor_krav = np.array(innenfor_krav)
    return innenfor_krav # Lagrer datapunkter i steady-state i numpy-array.
```

Sletter duplikater i 15-minutters intervaller

In [6]:
```
# Sletter duplikater av datapunkter slik at det blir
# "rene" 15-minutters intervaller etter hverandre i rekkefølge.
def delete_duplicates(innenfor_krav):
    femtenmin_split = np.array(innenfor_krav[0:2])

    for i,v in enumerate(innenfor_krav[2:]):
        print(f"{i} av {len(innenfor_krav[2:])}",end="\r")
        if not np.isin(v[:,0], femtenmin_split[:, :, 0]).any():
            femtenmin_split = np.append(femtenmin_split, [v], axis=0)

    femtenmin_split = femtenmin_split[1:]
    return femtenmin_split
```

Massestrøm: mean $\pm$ 1std

Beholder massestrømmen som er mest like ved å finne snittet og standardavviket.

In [7]:
```
def remove_ms_std(femtenmin_split):
    snitt = np.mean(femtenmin_split[:, :, 4].flatten())
    std = np.std(femtenmin_split[:, :, 4].flatten())

    femtenmin_split = femtenmin_split[(abs(femtenmin_split[:,:,4] - snitt) /std <= 1).any(axis=1)]
    print(f"snitt: {snitt:.2f}, std:{std:.2f}")
    return femtenmin_split
```

Behandling av datasettet

In [8]:
```
data_irra = find_irradiance(data)                          # Sorterer ut irradians-krav.
info = find_15_min_intervall(data_irra)                    # Deler inn i overlappende 15-minutters intervaller.
innenfor_krav = steady_state_krav(info)                    # Sikrer steady-state for irradians, Ta og To.
femtenmin_split = delete_duplicates(innenfor_krav)         # Fjerner duplikater i 15-minutters intervaller.
ready_15_min_intervall = remove_ms_std(femtenmin_split)    # Sikrer steady-state i massestrøm.
```

```
snitt: 374.29, std:42.89
```

Modellering

Finne variabel (X) veridene for hvert momentanpunkt:

(1) $\frac{T_i - T_a}{G_T}$ [Tm2/W]

Finne virkningsgrad ved å se max effekt og Q

(2) $\eta = \frac{Q}{GA}$

Effekt i vannet

c = 4.183 j/g K
(3) $Q = \dot{m}c\Delta T$ [W]

Max effekt

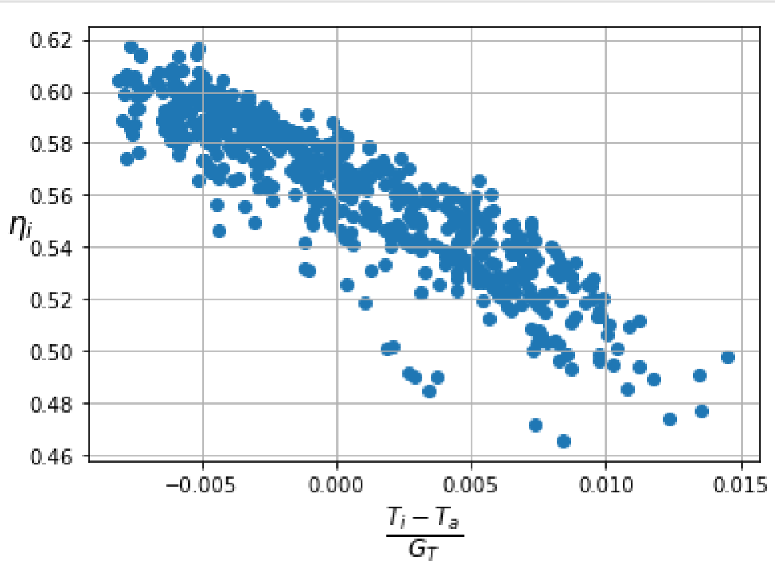
Solfanger er A=67,2 m2
(4) $GA = G * A$ [W]

In [9]:
```
def formulas(femtenmin_split):
    X = []
    Q = []
    GA = []
    for i in femtenmin_split:
        ir_mean, Ta_mean, Ti_mean, m_s_mean, To_mean = np.mean(i[:, 1:], axis=0)

        # X formelen (1)
        x = (Ti_mean - Ta_mean)/ir_mean
        X.append(x)

        # Q formelen (3)
        q = m_s_mean * 4.183 * (To_mean - Ti_mean)
        Q.append(q)

        # GA foremlen (4)
        ga = ir_mean * 67.2
        GA.append(ga)


    X = np.array(X)
    Q = np.array(Q)
    GA = np.array(GA)
    n = Q/GA # (2)
    return X, Q, GA, n
```

In [10]:
```
X, Q, GA, n = formulas(ready_15_min_intervall) # Lagrer verdier for X-komponent, vannets varme, irradians og ma
```

Plotter 15-minutters intervallene som datapunkter:

In [11]:
```
plt.scatter(X[n>0.2],n[n>0.2])
plt.grid()
plt.xlabel("$\\frac{T_{i} - T_{a}}{G_{T}}$", fontsize=17)
plt.ylabel("$\eta_{i}$", fontsize=15).set_rotation(0)
plt.show()
```



Lineær regresjon av modell

Kalkulerer r2-score for å sjekke at den lineære regresjonen følger trenden til de faktiske datapunktene.

In [12]:
```
reg = LinearRegression().fit(X[n>0.2].reshape(-1,1),n[n>0.2].reshape(-1,1))
print(f"r2score: {reg.score(X[n>0.2].reshape(-1,1),n[n>0.2].reshape(-1,1))}")
```

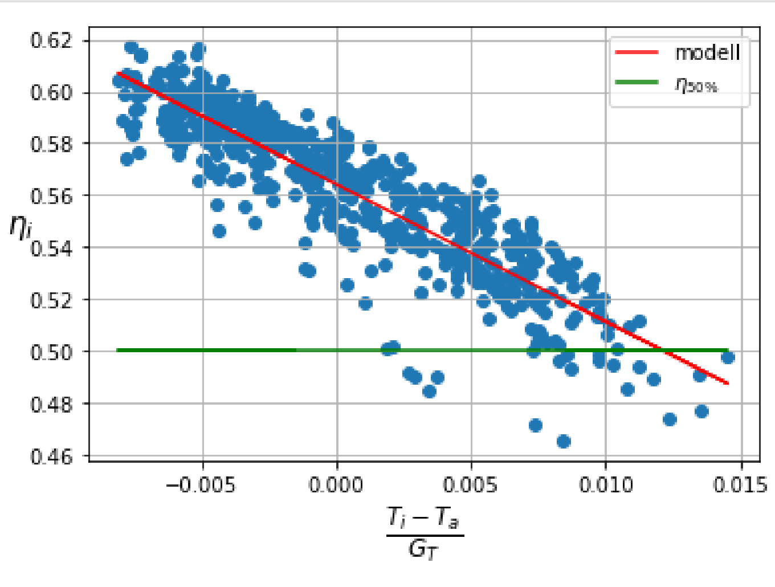
```
r2score: 0.7941282898548367
```

Fjerner outliere for virkningsgrad lavere enn 0,2

In [13]:
```
pred = reg.predict(X[n>0.2].reshape(-1,1))
```

Plotter lineær regresjon og modell med konstant 50% virkningsgrad

In [14]:
```
plt.scatter(X[n>0.2],n[n>0.2])
plt.plot(X[n>0.2],pred, c="red", label="modell")
plt.plot(X[n>0.2],[0.5 for i in range(len(pred))], c="green", label="$\eta_{50\%}$")
plt.grid()
plt.xlabel("$\\frac{T_{i} - T_{a}}{G_{T}}$", fontsize=17)
plt.ylabel("$\eta_{i}$", fontsize=15).set_rotation(0)
plt.legend()
plt.show()
```



In [15]:
```
print(f"b_1: {reg.coef_}.. \t = Fr*Ul")
print(f"b0: {reg.intercept_}..\t = Fr(τα)")

Fr = reg.intercept_ / (0.83 * 0.95)
print(f"Fr er {Fr}")

Ul  = -reg.coef_[0][0] / Fr
print(f"Ul er {Ul}")
```

```
b_1: [[-5.27614314]]..    = Fr*Ul
b0: [0.5642758]..         = Fr(τα)
Fr er [0.71563196]
Ul er [7.37270475]
```

Beregne energiproduksjon per dag for 2021

In [16]:
```
# Laster inn data og korrigerer kollonnenavn:
data = pd.read_csv("Måledata_mars_juli_2021.csv")
try:
    data = data.rename(columns={"Irradience (W/m2)":"Irradiance (W/m2)"})
except:
    print("Heter Irradiance (W/m2) allerede")
```

In [17]:
```
# Ordner tidsindeks for å dele data inn i dager:
data.Time = pd.to_datetime(data.Time)
data.index = data.Time
data = data.drop(columns=["Time"])
```

Energi målt

$Q = \dot{m}c\Delta T$

In [18]:
```
# Oppretter ny kollonne med beregning av målt energi fra massestrøm, spesifikk varmekapasitet og temperatutendi
data["energi [kWh]"] = data["Mass flow (g/s)"] * 4.183 * (data["Outlet Temperature (C)"] - data["Inlet Temperat
```

Energi predikert

$\eta_{pred} = \frac{Q}{GA} = Fr * (\tau\alpha) - Fr * Ul\frac{T_i - T_a}{G}$

$Q_{pred} = Fr * (\tau\alpha) * GA - Fr * Ul(T_i - T_a) * A$

$\eta = \frac{Q}{GA} = \frac{\dot{m}c\Delta T}{GA}$

In [19]:
```
# Verdier fra 2020-modell:
Fr = 0.71563196
Ul = 7.37270475
tau = 0.83
alpha = 0.95
areal = 67.2

# Predikerer effekt og kilowattimer for 2021 med 2020-modell og 50%-modell:
data["pred Q [W]"] = Fr * (tau * alpha) * data["Irradiance (W/m2)"] * areal - Fr * Ul * (data["Inlet Temperatur
data["pred energi [kWh]"] = data["pred Q [W]"] * (1/60) * 1e-3
data["dårligmodell [kWh]"] = data['Irradiance (W/m2)'] * areal * 0.5 * (1/60) * 1e-3
```

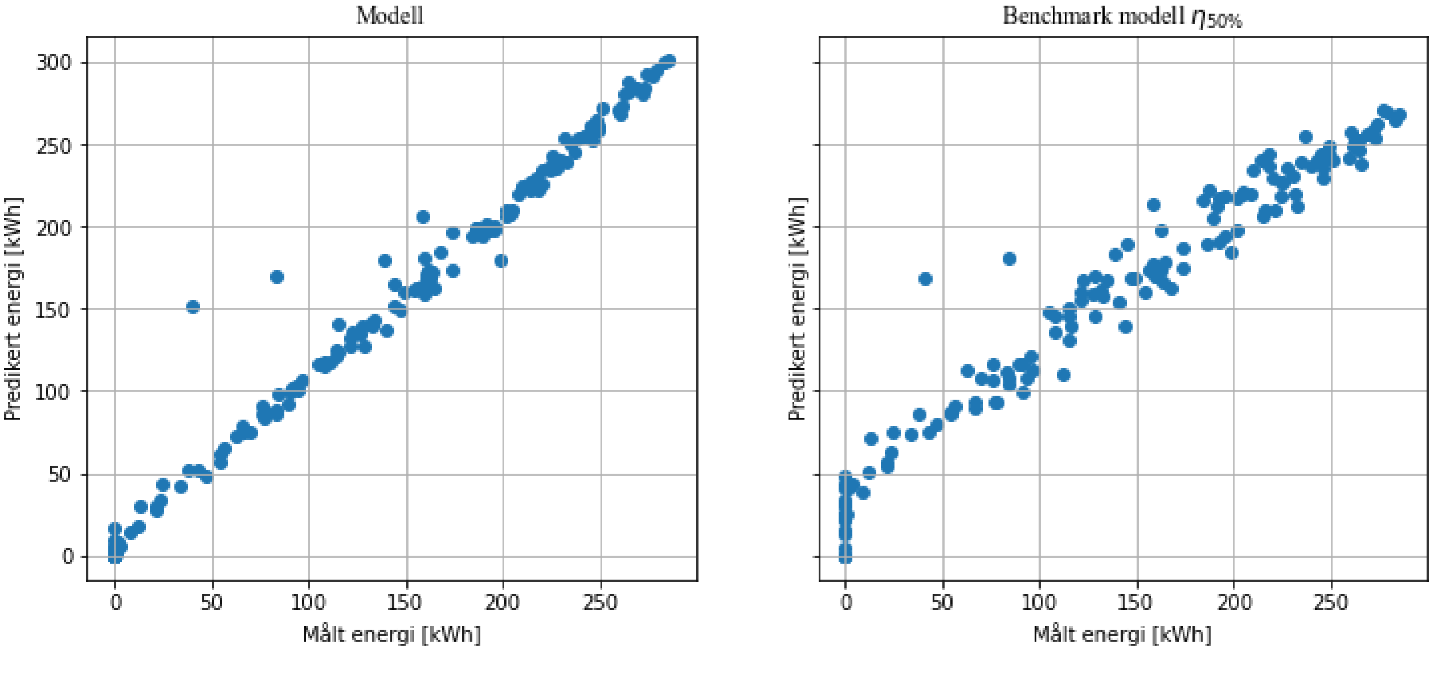
Antar at alle predikerte verdier der Q blir mindre enn 0 ikke skjer i virkeligheten. Setter derfor alle negative verdier til 0.

In [20]:
```
t = data["pred energi [kWh]"].to_numpy()
t[t < 0] = 0
data["pred energi [kWh]"] = t
```

In [21]:
```
# Samler det i dagsintervall og summerer opp energien
data_per_dag = data.resample("1D").sum()
```

Plotter sammenligning av predikert og målt energi per dag for 2020-modell og 50%-modell

In [22]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey = True, figsize=(12,5))
ax1.scatter(data_per_dag["energi [kWh]"], data_per_dag["pred energi [kWh]"])
ax1.set_title("Modell", fontname="Times New Roman")
ax2.scatter(data_per_dag["energi [kWh]"], data_per_dag["dårligmodell [kWh]"])
ax2.set_title("Benchmark modell $\eta_{50\%}$", fontname="Times New Roman")
ax1.grid(); ax2.grid()

ax1.set_xlabel("Målt energi [kWh]"); ax2.set_xlabel("Målt energi [kWh]")
ax1.set_ylabel("Predikert energi [kWh]"); ax2.set_ylabel("Predikert energi [kWh]")
fig.suptitle("Målt verdier mot predikert verdier daglig energi", fontsize=19, fontname="Times New Roman")
plt.show()
```



Finner RMS-verdier for modellene for estimat av modellenes kvalitet

Beregner gjennomsnittlig avvik fra linja som bekriver gjennomsnittet gjennom datapunktene.

In [23]:
```
print(f"RMSE for egen modell: {math.sqrt(mean_squared_error(data_per_dag['energi [kWh]'], data_per_dag['pred en
print(f"RMSE for benchmark modell: {math.sqrt(mean_squared_error(data_per_dag['energi [kWh]'], data_per_dag['då
```

```
RMSE for egen modell: 16.460
RMSE for benchmark modell: 28.095
```