Data Analyse en Visualisatie HLB Witlox van den Boomen

Voor Corporate Finance

Packages en de CSV importeren

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split
from pandas.plotting import autocorrelation_plot

#Voor lineaire regressie
from sklearn import datasets, linear_model

#Voor random forest
from sklearn.ensemble import RandomForestRegressor

#Voor 3d visualisatie
import plotly.express as px
from IPython.core.display import HTML

#Voor het ARIMA model
import statsmodels as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [3]:
```
winst_verlies_df = pd.read_csv(r'C:\Users\amn\Documents\Internship documents\ExcelCS
winst_verlies_df
```

Out[3]:

	Jaar	Netto-omzet	Inkoopwaarde	Brutomarge	Personeelskosten	Overige kosten	Totale kosten	EBITDA	Afschrijvi
0	2012	1000	500	500	750	250	1000	-500	
1	2013	1500	720	780	825	275	1100	-320	
2	2014	2025	1053	972	908	303	1210	-238	
3	2015	2936	1615	1321	998	333	1331	-10	
4	2016	2936	1644	1292	1098	366	1464	-172	
5	2017	2936	1703	1233	1208	403	1611	-377	
6	2018	4404	2466	1938	1329	443	1772	166	
7	2019	3920	2117	1803	1462	487	1949	-146	
8	2020	6272	3261	3010	1608	536	2144	867	
9	2021	10035	5218	4817	1768	589	2358	2459	

Data verkenning en cleaning

In [4]:
```
winst_verlies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Jaar                      10 non-null     int64  
 1   Netto-omzet               10 non-null     int64  
 2   Inkoopwaarde              10 non-null     int64  
 3   Brutomarge                10 non-null     int64  
 4   Personeelskosten          10 non-null     int64  
 5   Overige kosten            10 non-null     int64  
 6   Totale kosten             10 non-null     int64  
 7   EBITDA                    10 non-null     int64  
 8   Afschrijvingen            10 non-null     int64  
 9   EBIT                      10 non-null     int64  
 10  Kengetallen               10 non-null     int64  
 11  Delta omzet               9 non-null      float64
 12  Brutomarge / omzet        10 non-null     float64
 13  Personeelskosten / omzet  10 non-null     float64
 14  Overige kosten / omzet    10 non-null     float64
 15  Totale kosten / omzet     10 non-null     float64
 16  EBITDA marge              10 non-null     float64
dtypes: float64(6), int64(11)
memory usage: 1.5 KB
```

In [5]:
```
winst_verlies_df.describe()
```

Out[5]:

	Jaar	Netto-omzet	Inkoopwaarde	Brutomarge	Personeelskosten	Overige kosten	To ko
count	10.00000	10.000000	10.000000	10.000000	10.000000	10.000000	10.00(
mean	2016.50000	3796.400000	2029.700000	1766.600000	1195.400000	398.500000	1593.90(
std	3.02765	2664.274102	1387.085197	1282.327069	342.186174	113.920294	456.38
min	2012.00000	1000.000000	500.000000	500.000000	750.000000	250.000000	1000.00(
25%	2014.25000	2252.750000	1193.500000	1037.250000	930.500000	310.500000	1240.25(
50%	2016.50000	2936.000000	1673.500000	1306.500000	1153.000000	384.500000	1537.50(
75%	2018.75000	4283.000000	2378.750000	1904.250000	1428.750000	476.000000	1904.75(
max	2021.00000	10035.000000	5218.000000	4817.000000	1768.000000	589.000000	2358.00(

Onderstaand is te zien dat er 1 NA waarde is, bij delta omzet. Dat is logisch want dit is bij het eerste jaar, daar kan nog geen delta omzet zijn.

In [6]:
```
winst_verlies_df.isna().sum()
```

Out[6]:
```
Jaar                      0
Netto-omzet               0
Inkoopwaarde              0
Brutomarge                0
Personeelskosten          0
Overige kosten            0
Totale kosten             0
```

```
EBITDA                      0
Afschrijvingen              0
EBIT                        0
Kengetallen                 0
Delta omzet                 1
Brutomarge / omzet          0
Personeelskosten / omzet    0
Overige kosten / omzet      0
Totale kosten / omzet       0
EBITDA marge                0
dtype: int64
```

Uit bovenstaande blijkt dat de waarden numeriek en continue zijn. Hiervoor moeten we de juiste algoritmen kiezen, die met deze waarden kunnen werken.

Algoritmen

Voor de algoritmes heb ik gekozen een selectie te maken die toepasselijk is op deze data. Hierbij heb ik besloten om de volgende algoritmen te proberen: Lineaire Regressie, Meerdere Lineaire Regressie en de ARIMA time series. Deze heb ik gekozen zodat ik voorspellingen kan maken voor de komende jaren.

Correlatie zoeken

In [7]:
```
winst_verlies_df.corr()
```

Out[7]:

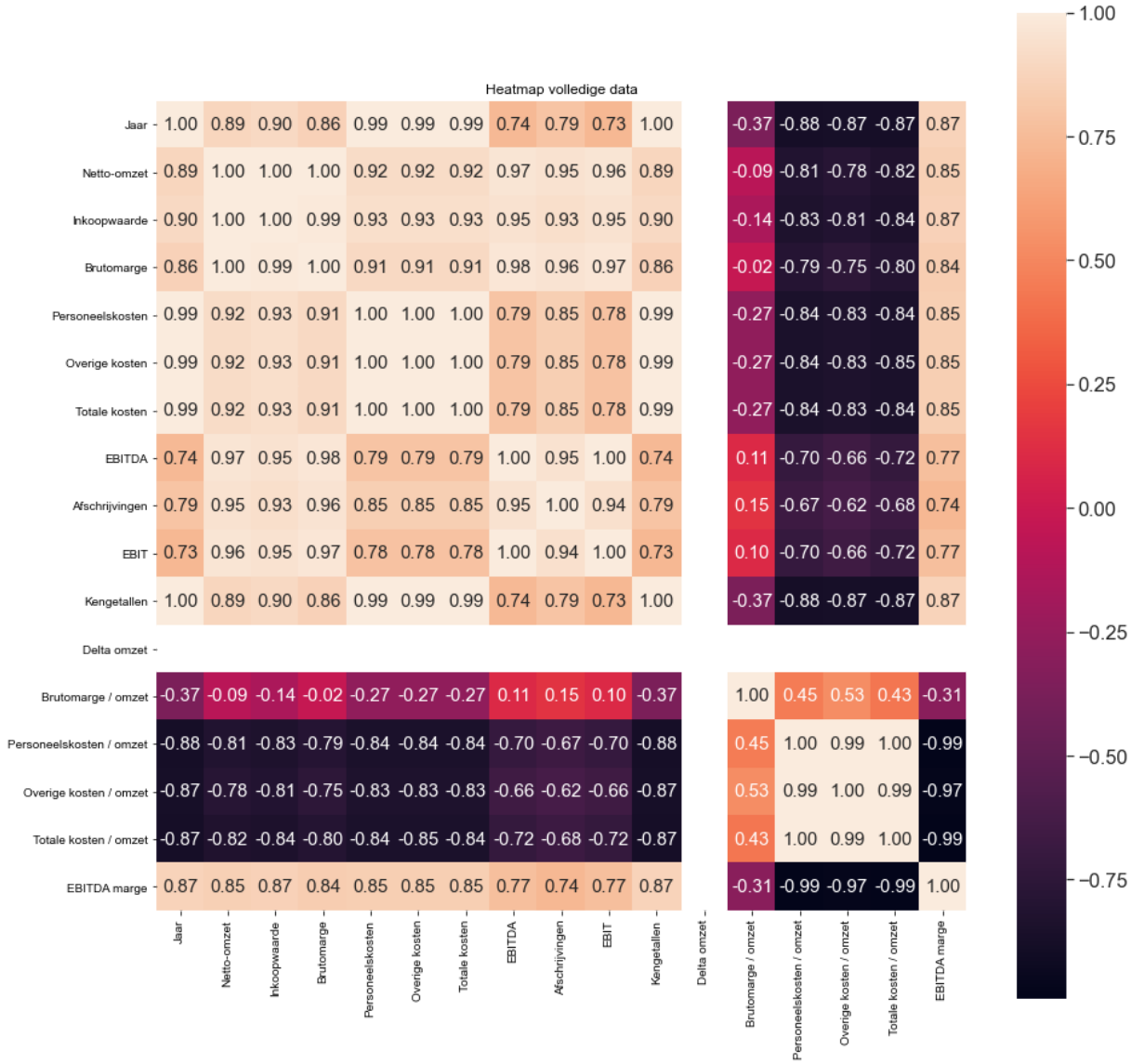
	Jaar	Netto-omzet	Inkoopwaarde	Brutomarge	Personeelskosten	Overige kosten	I
Jaar	1.000000	0.885681	0.901949	0.864434	0.992902	0.993010	0.9
Netto-omzet	0.885681	1.000000	0.998183	0.997878	0.920865	0.920809	0.9
Inkoopwaarde	0.901949	0.998183	1.000000	0.992142	0.931154	0.931142	0.9
Brutomarge	0.864434	0.997878	0.992142	1.000000	0.905942	0.905837	0.9
Personeelskosten	0.992902	0.920865	0.931154	0.905942	1.000000	0.999998	1.(
Overige kosten	0.993010	0.920809	0.931142	0.905837	0.999998	1.000000	0.9
Totale kosten	0.992885	0.921020	0.931312	0.906091	1.000000	0.999998	1.(
EBITDA	0.736499	0.965525	0.951989	0.976231	0.792645	0.792495	0.7
Afschrijvingen	0.791514	0.947662	0.931959	0.960773	0.849630	0.849113	0.8
EBIT	0.728801	0.962659	0.949357	0.973124	0.784539	0.784419	0.7
Kengetallen	1.000000	0.885681	0.901949	0.864434	0.992902	0.993010	0.9
Delta omzet	0.086267	0.418384	0.389724	0.447069	0.153232	0.153467	0.1
Brutomarge / omzet	-0.366163	-0.085551	-0.143363	-0.022710	-0.269148	-0.270089	-0.2
Personeelskosten / omzet	-0.875121	-0.813931	-0.833849	-0.789049	-0.843265	-0.843803	-0.8
Overige kosten / omzet	-0.873912	-0.784540	-0.809321	-0.754520	-0.833974	-0.834589	-0.8

	Jaar	Netto-omzet	Inkoopwaarde	Brutomarge	Personeelskosten	Overige kosten	I
Totale kosten / omzet	-0.874207	-0.822015	-0.840944	-0.798169	-0.844527	-0.845064	-0.8
EBITDA marge	0.867922	0.854875	0.866194	0.839120	0.851129	0.851555	0.8

Zoals al op meerdere momenten in mijn onderzoek is gebleken, kan een afbeelding soms beter inzicht geven dus maak ik gebruik van een heatmap

In [8]:

```
cm = np.corrcoef(winst_verlies_df.values, rowvar = 0)
plt.figure(figsize = (15,15))
plt.title('Heatmap volledige data')
sns.set(font_scale = 1.5)
hm = sns.heatmap(cm,
                 cbar = True,
                 annot = True,
                 square = True,
                 fmt = '.2f',
                 annot_kws = {'size':15},
                 yticklabels = winst_verlies_df.columns,
                 xticklabels = winst_verlies_df.columns,
)
plt.show()
```

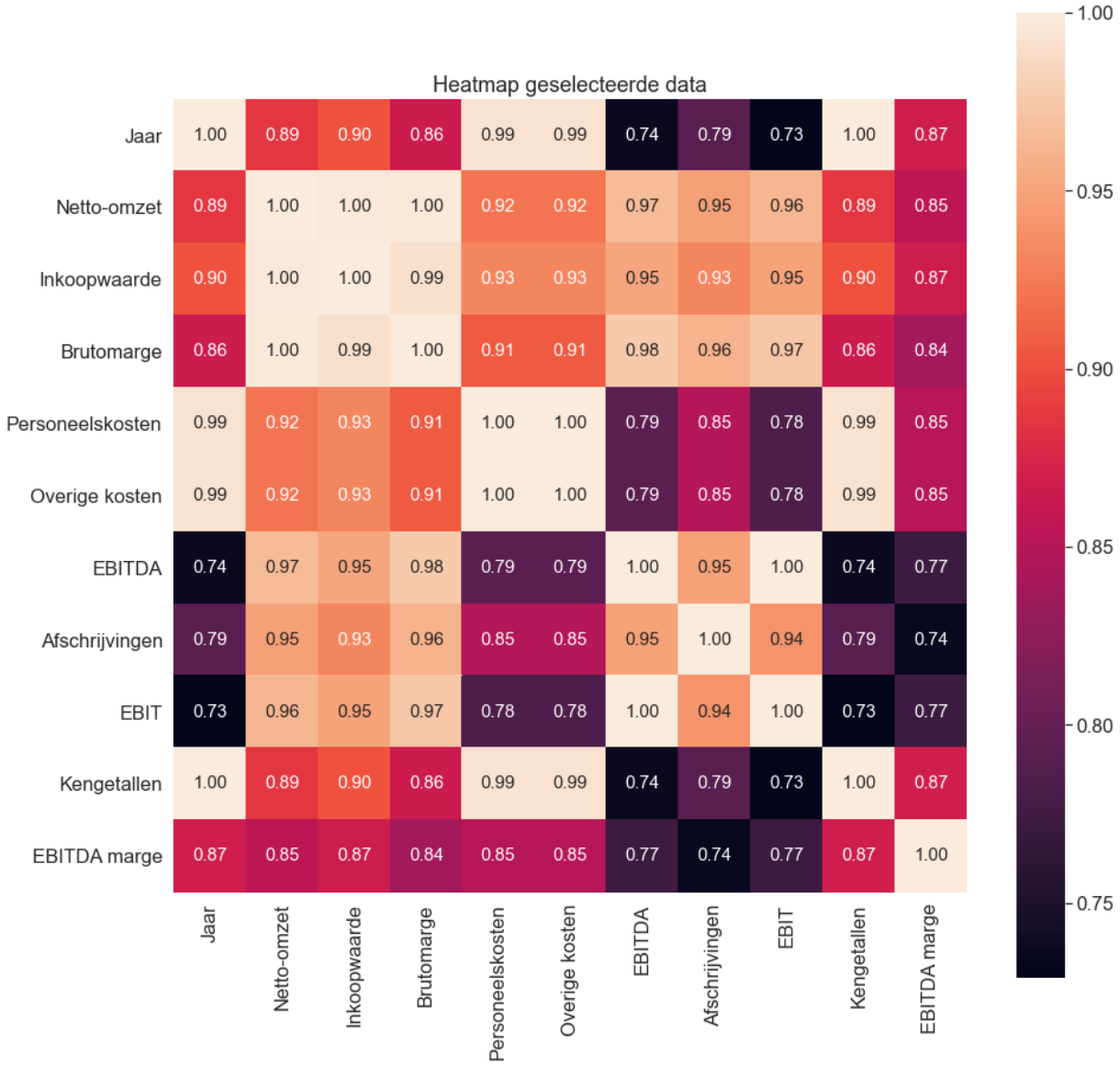


Hoe hoger de correlatie coëfficient, hoe sterker de relatie tussen de twee. Een correlatie van 0 geeft dus ook aan dat er geen relatie is. Een positieve correlatie indiceert een positieve relatie, en vice versa. Het is logisch dat dezelfde kolommen een correlatie van 1 hebben, gezien zij exact dezelfde waarden bevatten. De kolommen waar gekeken wordt naar omzet (bijvoorbeeld overige kosten/omzet en totale kosten/omzet) hebben ook een hoge correlatie, gezien sowieso al een deel van de waarden met elkaar overeenkomt.

Om deze reden is het slimmer om te kijken naar alleen de kolommen die deze niet bevatten: Jaar, Netto-omzet, Inkoopwaarde, Brutomarge, Personeelskosten, Overige kosten, EBITDA, Afschrijvingen, EBIT, Kengetallen, EBITDA marge.

In [9]:
```
cols = ['Jaar', 'Netto-omzet', 'Inkoopwaarde', 'Brutomarge', 'Personeelskosten', 'Ov
cm = np.corrcoef(winst_verlies_df[cols].values, rowvar = 0)
plt.figure(figsize = (15,15))
plt.title('Heatmap geselecteerde data')
sns.set(font_scale = 1.5)
hm = sns.heatmap(cm,
                 cbar = True,
                 annot = True,
                 square = True,
                 fmt = '.2f',
                 annot_kws = {'size':15},
                 yticklabels = cols,
                 xticklabels = cols,
```

```
)
plt.show()
```



Dit geeft al een duidelijker overzicht. Er zijn wel een aantal zaken die duidelijk zijn met behulp van logisch nadenken. Bijvoorbeeld dat de inkoopwaarde en netto omzet een sterke positieve correlatie hebben, of de positieve correlatie bij overige kosten en personeelskosten. Hieruit blijkt al dat er in de data tussen bijna alle kolommen een sterke positieve correlatie is.

Hierdoor is het vergelijken van twee variabelen belangrijk voor het vinden van uitschieters, en bijvoorbeeld het vergelijken van drie variabelen zou kunnen leiden tot andere inzichten

Lineaire Regressie

Eerst is er data nodig voordat er een model gemaakt kan worden. Hiervoor splitten we de data in een trainingset en een testset, zodat we ook data hebben die het model nooit heeft gezien en waar we dus mee kunnen vergelijken.

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(
winst_verlies_df['Jaar'], winst_verlies_df['Netto-omzet'], test_size = 0.3, random_s
```

Nu we de data hebben kunnen we het model maken. Zoals hierboven te zien is gebruiken we de kolommen Netto-omzet en Personeelskosten.

Bij het maken van dit model zijn meerdere mogelijkheden om parameters te tunen.

- fit_intercept is een Boolean (standaard True) die beslist om de intercept te berekenen (True) of deze gelijk te stellen aan nul (False)
- normalize is een Boolean (standaard True) die beslist om de input variabelen wel (True) of niet (False) te normaliseren.
- copy_X is een Boolean (standaard True) die beslist om de input variabelen te kopiëren (True) of te overschrijven (False).
- n_jobs is een integer of None (standaard) dat het aantal jobs gebruikt in de paralelle berekening. None betekent een 'job' en -1 betekent het gebruik van alle processors.

In [11]:
```
lin_regr = linear_model.LinearRegression().fit(X_train.values.reshape(-1, 1), y_trai
```

Nu het model fitted is kunnen we voorspellingen doen. Bijvoorbeeld de Netto omzet van volgend jaar, of over 10 jaar.

In [12]:
```
y_pred_lr = lin_regr.predict(X_test.values.reshape(-1, 1))
```

Nu het model klaar is, is het belangrijk om het model te testen op hoe accuraat deze is. Dit is makkelijk te doen met .score()

In [13]:
```
lin_regr.score(X_test.values.reshape(-1, 1), y_test)
```

Out[13]:
```
0.9336734766357784
```

Een uitkomst van 0.7 of hoger is acceptabel. Bij het basismodel is de score 0.65, dus nog niet op het gewenste niveau. Hiervoor moeten we de parameters tunen.

Na het tunen van de parameters, is het gelukt een score te krijgen van 0.93. Dit is zeer acceptabel.

Dit haalt nog niet weg dat er weinig datapunten zijn om te gebruiken, met een grotere dataset is het mogelijk om betere resultaten te krijgen, waarbij bijvoorbeeld de omzet per maand is bijgehouden, in plaats van jaar.

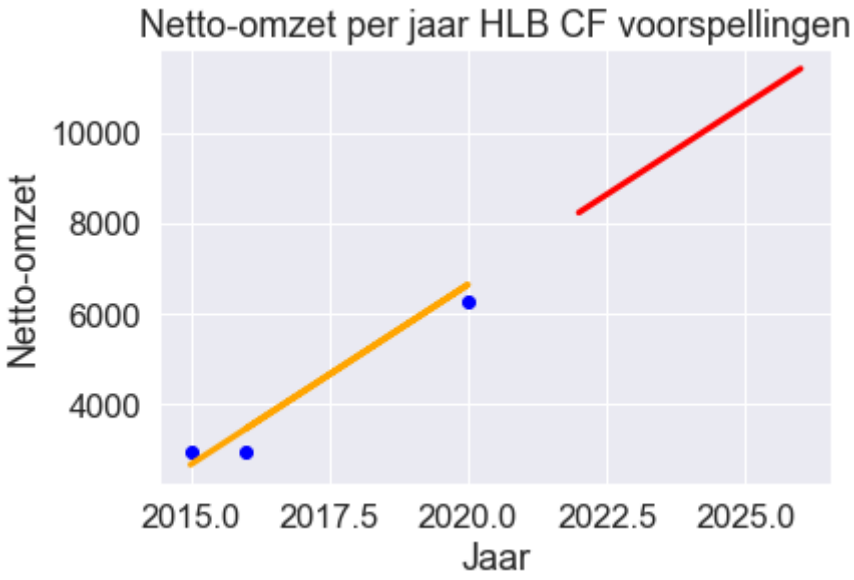
Laten we ook kijken naar de voorspellingen voor de komende 5 jaar

In [14]:
```
X_future = np.array([2022, 2023, 2024, 2025, 2026])
X_future
```

Out[14]:
```
array([2022, 2023, 2024, 2025, 2026])
```

In [15]:
```
y_pred_future_lr = lin_regr.predict(X_future.reshape(-1, 1))
```

In [16]:
```
plt.scatter(X_test, y_test, color = 'blue')
plt.plot(X_future, y_pred_future_lr, color = 'red', linewidth=3)
plt.plot(X_test, y_pred_lr, color = 'orange', linewidth=3)
plt.title('Netto-omzet per jaar HLB CF voorspellingen')
plt.xlabel('Jaar')
plt.ylabel('Netto-omzet')
plt.show()
```



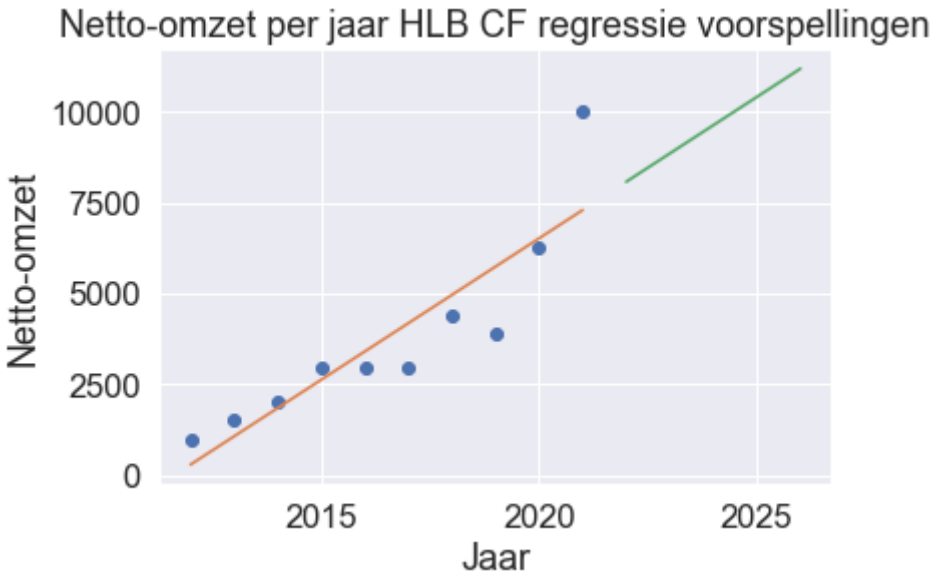
Lineare regressie zonder gebruik van een model

In [17]:
```
x = winst_verlies_df['Jaar']
y = winst_verlies_df['Netto-omzet']
```

In [18]:
```
plt.plot(x, y, 'o')
plt.title('Netto-omzet per jaar HLB CF regressie voorspellingen')
plt.xlabel('Jaar')
plt.ylabel('Netto-omzet')

m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x + b)
plt.plot(X_future, m*X_future+b)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x2686fc23700>]
```



Hieruit volgt bijvoorbeeld dat er een trend te zien is in de jaren tot en met 2020, gezien deze de regressielijn redelijk goed volgen. In 2021 is er een uitschieter.

Multiple Regression Model

Hierbij is het mogelijk om naar meerdere variabelen te kijken, dit zou moeten werken om een

meer accurate voorspelling te maken

Hiervoor kijken we wederom naar de kolommen die we willen gebruiken voor het model

In [19]:
```
winst_verlies_df.columns
```

Out[19]:
```
Index(['Jaar', 'Netto-omzet', 'Inkoopwaarde', 'Brutomarge', 'Personeelskosten',
       'Overige kosten', 'Totale kosten', 'EBITDA', 'Afschrijvingen', 'EBIT',
       'Kengetallen', 'Delta omzet', 'Brutomarge / omzet',
       'Personeelskosten / omzet', 'Overige kosten / omzet',
       'Totale kosten / omzet', 'EBITDA marge'],
      dtype='object')
```

We willen als y variabele de Netto-omzet voorspellen, en bekijken welke variabelen het meeste invloed hierop hebben. Ideaal gezien gebruik je hiervoor zoveel mogelijk variabelen, maar wederom laten we een aantal kolommen buiten beschouwing, net als bij de heatmap.

In [20]:
```
x = np.array(winst_verlies_df[['Inkoopwaarde', 'Brutomarge', 'Personeelskosten', 'Ov
y = np.array(winst_verlies_df['Netto-omzet'])
```

Hier maak ik weer een train en test dataset van:

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_st
```

In [22]:
```
mul_lin_regr = linear_model.LinearRegression().fit(X_train, y_train)
```

In [23]:
```
y_pred_mul = mul_lin_regr.predict(X_test)
```

Wederom checken we de score om te kijken hoe accuraat de voorspellingen zijn

In [24]:
```
mul_lin_regr.score(X_test, y_test)
```

Out[24]:
```
0.9999998999860847
```

Zoals te zien is heeft deze al direct een erg hoge score, dus is dit model zeer accuraat. Dit komt natuurlijk doordat het model alle variabelen meeneemt in de voorspellingen

Gezien het niet mogelijk is om dit model als geheel te visualiseren, maak ik gebruik van een 3D diagram, om 3 variabelen tegenover elkaar te plotten:

In [25]:
```
fig1 = px.scatter_3d(winst_verlies_df, x='Personeelskosten', y='Netto-omzet', z='Afs
HTML(fig1.show())
```

Out[25]: <IPython.core.display.HTML object>

Door middel van het gebruik van kleuren en vormen is het mogelijk nog extra dimensies toe te voegen aan de plot

In [48]:
```
fig2 = px.scatter_3d(winst_verlies_df, x='Personeelskosten', y='Netto-omzet', z='Afs
fig2.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0,
                                           ticks="outside",
                                           ticksuffix=" bills"))
HTML(fig2.show())
```

Out[48]: <IPython.core.display.HTML object>

ARIMA time series

Autoregressive Integrated Moving Average

- AR: Autoregressie
- I: Integrated
- MA: Moving Average

Hierbij worden de parameters als volgt beschreven:

- p: Aantal lag oberservaties in het model, ook wel 'lag order'
- d: Aantal keer dat de observaties zijn gedifferencieerd, ook wel 'degree of differencing'
- q: De grootte van de 'moving average window', ook wel 'order of moving average'

In [35]:
```
ARIMA_df = pd.DataFrame(winst_verlies_df[['Jaar', 'Netto-omzet']]).set_index('Jaar')
```

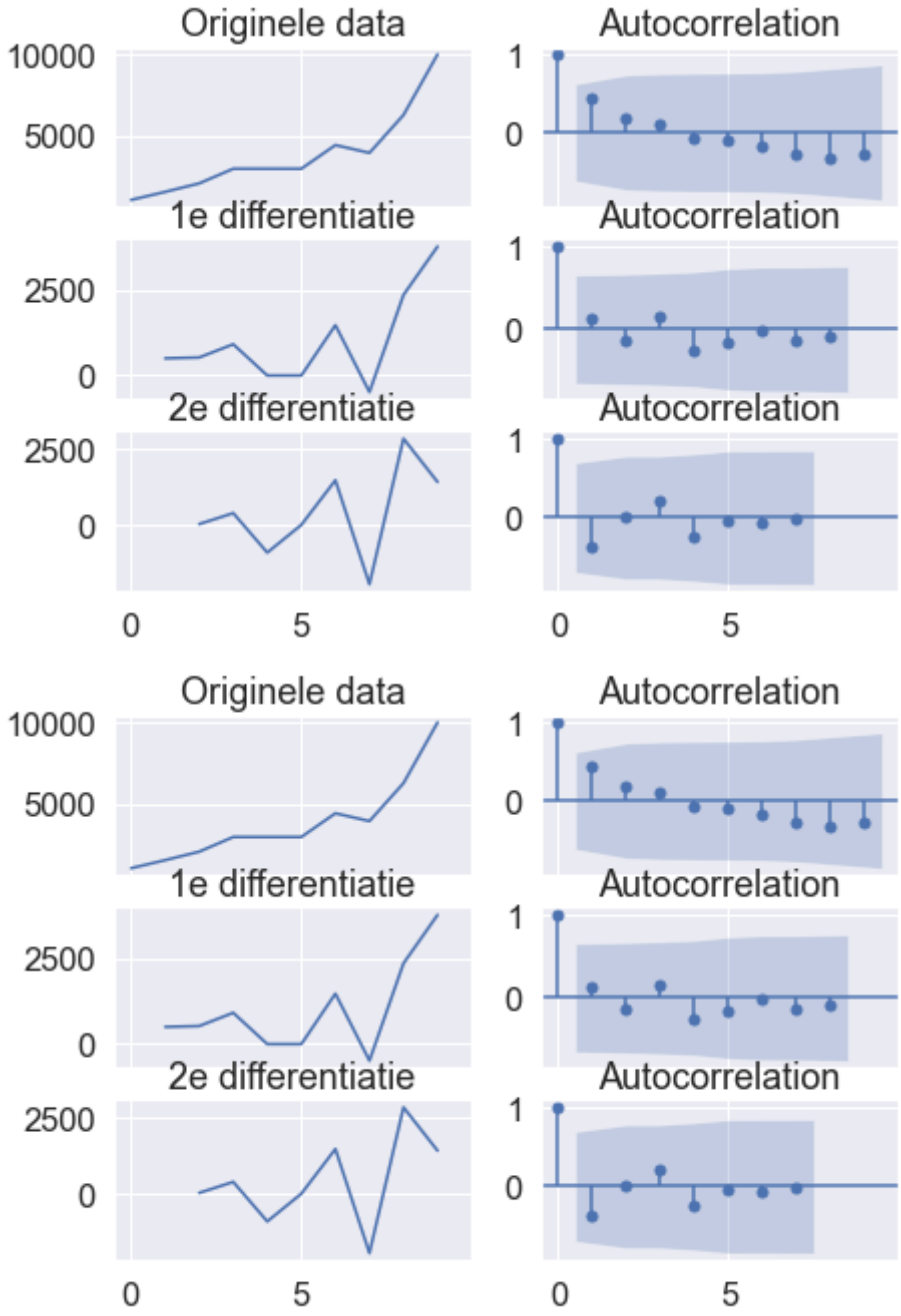
In [36]:
```
ARIMA_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10 entries, 2012 to 2021
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Netto-omzet  10 non-null     int64
dtypes: int64(1)
memory usage: 160.0 bytes
```

Allereerst gaan we kijken naar de 'd' parameter, deze vinden we door het gebruik van autocorrelatie diagrammen. Deze laten zien wanneer de data stationair is (wanneer het gemiddelde rond de 0.0 is)

In [37]:
```
#Originele data
fig, axes = plt.subplots(3,2, sharex = True, figsize=(7,5))
axes[0,0].plot(ARIMA_df.values); axes[0,0].set_title('Originele data')
plot_acf(ARIMA_df.values, ax=axes[0,1])
#1e differentiatie
axes[1,0].plot(ARIMA_df.diff().values); axes[1,0].set_title('1e differentiatie')
plot_acf(ARIMA_df.diff().dropna().values, ax=axes[1,1])
#2e differentiatie
axes[2,0].plot(ARIMA_df.diff().diff().values); axes[2,0].set_title('2e differentiati
plot_acf(ARIMA_df.diff().diff().dropna().values, ax=axes[2,1])
```

Out[37]:

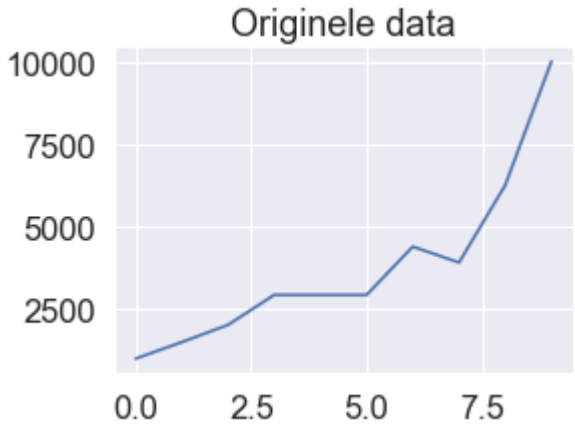


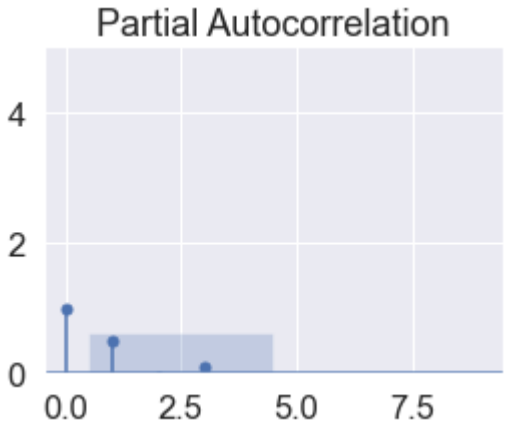
Vanuit bovenstaande grafieken blijkt dat de data niet snel van positief naar negatief veranderd met 1 differentiatie en daar dus al stationair (genoeg) is.

In [38]:

```
fig, axes = plt.subplots(1, 2, sharex=True, figsize=(9,3))
axes[0].plot(ARIMA_df.values); axes[0].set_title('Originele data')
axes[1].set(ylim=(0,5))
plot_pacf(ARIMA_df.dropna().values, ax=axes[1], lags=4)

plt.show()
```





Uit bovenstaande grafiek is te vinden dat voor de p waarde het beste 0 te kiezen is, gezien deze buiten de significance line valt en dus laat zien dat de correlatie tussen de series en de lags het beste is.

Vervolgens kunnen we weer vie het autocorrelation diagram kijken naar de 'q' variabele, deze laat de error van de lagged voorspelling zien. Deze zetten we vooralsnog ook op 0, gezien deze ook buiten de significance line moet vallen.

Conclusie voor nu is dus d = 1, p = 0, q = 0

Hiermee beginnen we met het maken van het model

In [40]:
```
ARIMA_model = ARIMA(ARIMA_df.values, order=(2,1,0)).fit()
```

In de summary kunnen we de waarden beter bekijken. Hierin staan bijvoorbeel de coefficiënt, std error, p waarde etc. Deze uitkomsten kunnen we gebruiken om het model verder te tweaken.

In [41]:
```
print(ARIMA_model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   10
Model:                 ARIMA(2, 1, 0)   Log Likelihood                -77.683
Date:                Fri, 17 Dec 2021   AIC                            161.365
Time:                        11:28:52   BIC                            161.957
Sample:                             0   HQIC                           160.089
                                  - 10
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.6891      0.403      1.712      0.087      -0.100       1.478
ar.L2          0.2482      0.637      0.390      0.697      -1.000       1.496
sigma2      1.804e+06   1.62e+06      1.115      0.265   -1.37e+06    4.98e+06
===================================================================================
Ljung-Box (Q):                        5.22   Jarque-Bera (JB):                 0.63
Prob(Q):                              0.73   Prob(JB):                         0.73
Heteroskedasticity (H):              48.76   Skew:                             0.30
Prob(H) (two-sided):                  0.01   Kurtosis:                         1.85
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

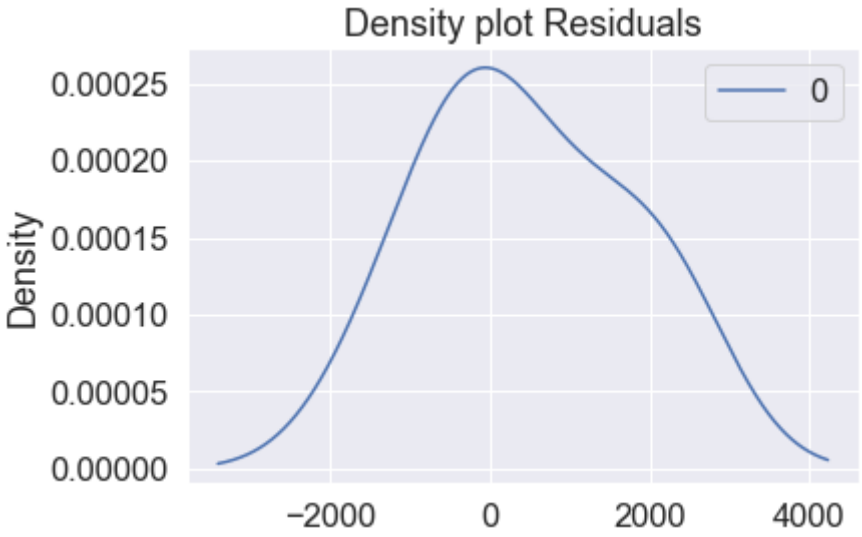
Door het model te zetten naar p,d,q = 2,1,0 krijgen we het beste model. Hier gaan we dan ook mee verder

Vervolgens kijken we naar de residuals, deze laten zien of er nog patronen in het model zitten. Deze wilde je dus niet, want de data moet stationair zijn. Hiervoor kijken we naar de mean en variëteit van residuals, de afstand van de echte datapunten tot de voorspelde datapunten.

In [42]:
```
residuals = pd.DataFrame(ARIMA_model.resid)
residuals.plot()
plt.title('Residuals ARIMA model')
plt.xlabel('Time')
plt.ylabel('Residuals')
```

Out[42]: Text(0, 0.5, 'Residuals')

In [43]:
```
residuals.plot(kind='kde')
plt.title('Density plot Residuals')
plt.show()
```



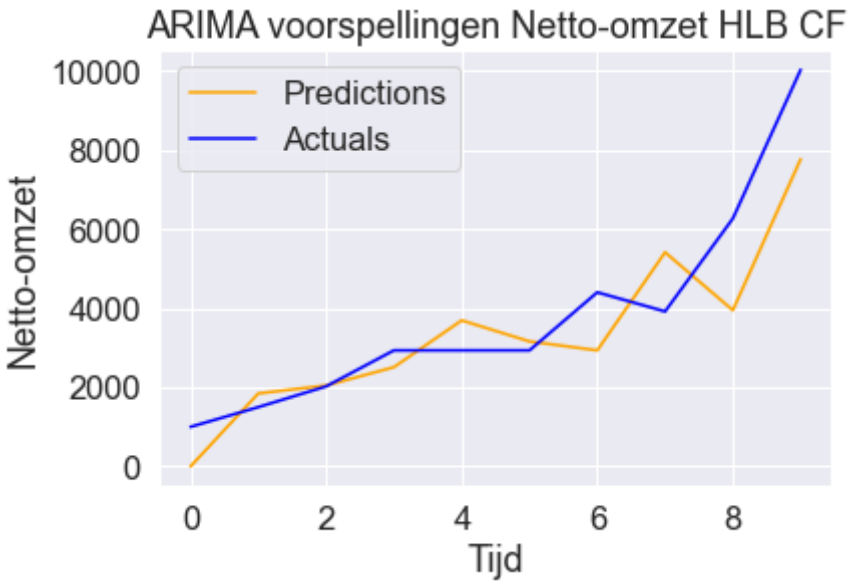
Door te kijken naar de residuals krijgen we nog redelijk weinig info. Doordat de dataset redelijk klein is zijn er weinig datapunten en is het lastig te zien of er wel of geen patroon meer is. Door te kijken naar de density van de residuals, zien we dat de mean rond de 0 ligt, wat betekent dat er niet veel variatie is.

Vervolgens kunnen we de voorspelde waarden plotten tegen de echte waarden.

In [44]:
```
predictions = ARIMA_model.predict()
predictions = predictions.reshape(10,1)
```

In [45]:
```
actuals = winst_verlies_df['Netto-omzet']
```

In [46]:
```
plt.plot(predictions, color = 'Orange', label = 'Predictions')
plt.plot(actuals, color = 'Blue', label = 'Actuals')
plt.title('ARIMA voorspellingen Netto-omzet HLB CF')
plt.ylabel('Netto-omzet')
plt.xlabel('Tijd')
plt.legend()
plt.show()
```



Hieruit volgt dat over het algemeen de voorspellingen enigzins dezelfde trend volgen, maar zeker niet 100% accuraat zijn. Dit komt grotendeels door te weinig datapunten om te vergelijken, maar het feit dat de voorspellingen zo dicht bij elkaar liggen, laat wel zien dat met een grotere dataset, het ARIMA model een goede optie zou zijn om waarden te voorspellen

In [ ]: