Analyse classificatie & regressie

Classificatie

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusi
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier


pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Vragen classificatie

1. Noem een voorbeeld uit de praktijk waarin jullie algoritme wordt gebruikt. Het Random forest model wordt gebruikt in de Finaciele sector voor fraudedetecite. Het kan patronen identificeren die wijzen op verdachte activiteiten, zoals ongebruikelijke transacties of creditcardfraude.

2. Hoe werkt het algoritme conceptueel? Wat zijn de belangrijkste stappen?

- Bootstrapping: Maakt willekeurige subsets van de trainingsgegevens door exemplaren met vervanging te selecteren.
- Kenmerkselectie: Selecteerd willekeurige kenmerken bij elke knoop in elke boom om overfitting te verminderen.
- Opbouwen van Beslissingsbomen: Trained beslissingsbomen op de subsets van gegevens en kenmerken.
- Voorspellingen van Elke Boom: Gebruikt elke boom om voorspellingen te doen op nieuwe gegevens.
- Aggregatie van Voorspellingen: Combineert de voorspellingen van alle bomen om de uiteindelijke voorspelling van het Random Forest te verkrijgen, bijvoorbeeld door het nemen van gemiddelden (voor regressie) of meerderheid (voor classificatie).

3. Wat zijn de voor- en nadelen van jullie algoritme? In welke situaties werkt het heel goed en wanneer juist niet?

Voordelen:

- Random Forest heeft vaak een hoge voorspellende nauwkeurigheid, zelfs

zonder veel afstemming van hyperparameters.
- Door het gebruik van meerdere bomen en bootstrapping is Random Forest robuust tegen overfitting, overfitting gebeurt snel bij kleine datasets, wat bij ons het geval is.

Nadelen:

- Het maken van voorspellingen kan relatief lang duren, vooral bij grote aantallen bomen en kenmerken.

Wanneer werkt Random forest wel goed:

- Random Forest werkt goed bij complexe taken waarin er veel interacties en niet-lineaire relaties tussen kenmerken zijn.

Wanneer werkt Random forest niet goed:

- Als snelle voorspellingen cruciaal zijn, kan Random Forest minder geschikt zijn vanwege de combinatie van resultaten over meerdere bomen.

In [ ]:
```
# Hier wordt het bestand ingeladen en wordt de naam 'data' gegeven.
data = pd.read_csv('./data/classification/data.csv')
```

Verkenende analyse

In [ ]:
```
# Hier word gekeken hoe het dataframe eruit ziet
data.head()
```

Out[ ]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean
0	842302	M	17.99	10.38	122.80	1001.0
1	842517	M	20.57	17.77	132.90	1326.0
2	84300903	M	19.69	21.25	130.00	1203.0
3	84348301	M	11.42	20.38	77.58	386.1
4	84358402	M	20.29	14.34	135.10	1297.0

In [ ]:
```
# Hier wordt gekeken welke kolommen er allemaal in de dataframe staan
data.columns
```

Out[ ]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean
',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_
se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [ ]:
```
# Wat is het type van de data in het dataframe?
# Als het dataframe een kolom heeft met veel Nan waarden of missing value
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      569 non-null    int64
 1   diagnosis               569 non-null    object
 2   radius_mean             569 non-null    float64
 3   texture_mean            569 non-null    float64
 4   perimeter_mean          569 non-null    float64
 5   area_mean               569 non-null    float64
 6   smoothness_mean         569 non-null    float64
 7   compactness_mean        569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
 12  radius_se               569 non-null    float64
 13  texture_se              569 non-null    float64
 14  perimeter_se            569 non-null    float64
 15  area_se                 569 non-null    float64
 16  smoothness_se           569 non-null    float64
 17  compactness_se          569 non-null    float64
 18  concavity_se            569 non-null    float64
 19  concave points_se       569 non-null    float64
 20  symmetry_se             569 non-null    float64
 21  fractal_dimension_se    569 non-null    float64
 22  radius_worst            569 non-null    float64
 23  texture_worst           569 non-null    float64
 24  perimeter_worst         569 non-null    float64
 25  area_worst              569 non-null    float64
 26  smoothness_worst        569 non-null    float64
 27  compactness_worst       569 non-null    float64
 28  concavity_worst         569 non-null    float64
 29  concave points_worst    569 non-null    float64
 30  symmetry_worst          569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32             0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [ ]:
```
# Zitten er nan values in het dataframe?
data.isna().sum()
```

Out[ ]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

In [ ]:
```
# In deze tabel kan gezien worden dat alle minimale waarden positief zijn
data.describe()
```

Out[ ]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	sr
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

In [ ]:
```
# Unnamed 32 is een kolom met alleen maar nan waardes, deze is dus niet n
# Daarnaast is id geen kenmerk of het goedaardig of kwaadaardig is, maar
data.drop(columns=['Unnamed: 32', 'id'], inplace=True)
```

In [ ]:
```
# M = malignant (kwaadaardig) B = begnin (goedaardig)
# Wat is ongeveer de verhoding van goedaardig en kwaadaardig
print(data.diagnosis.value_counts())
data.diagnosis.value_counts(normalize=True)
```

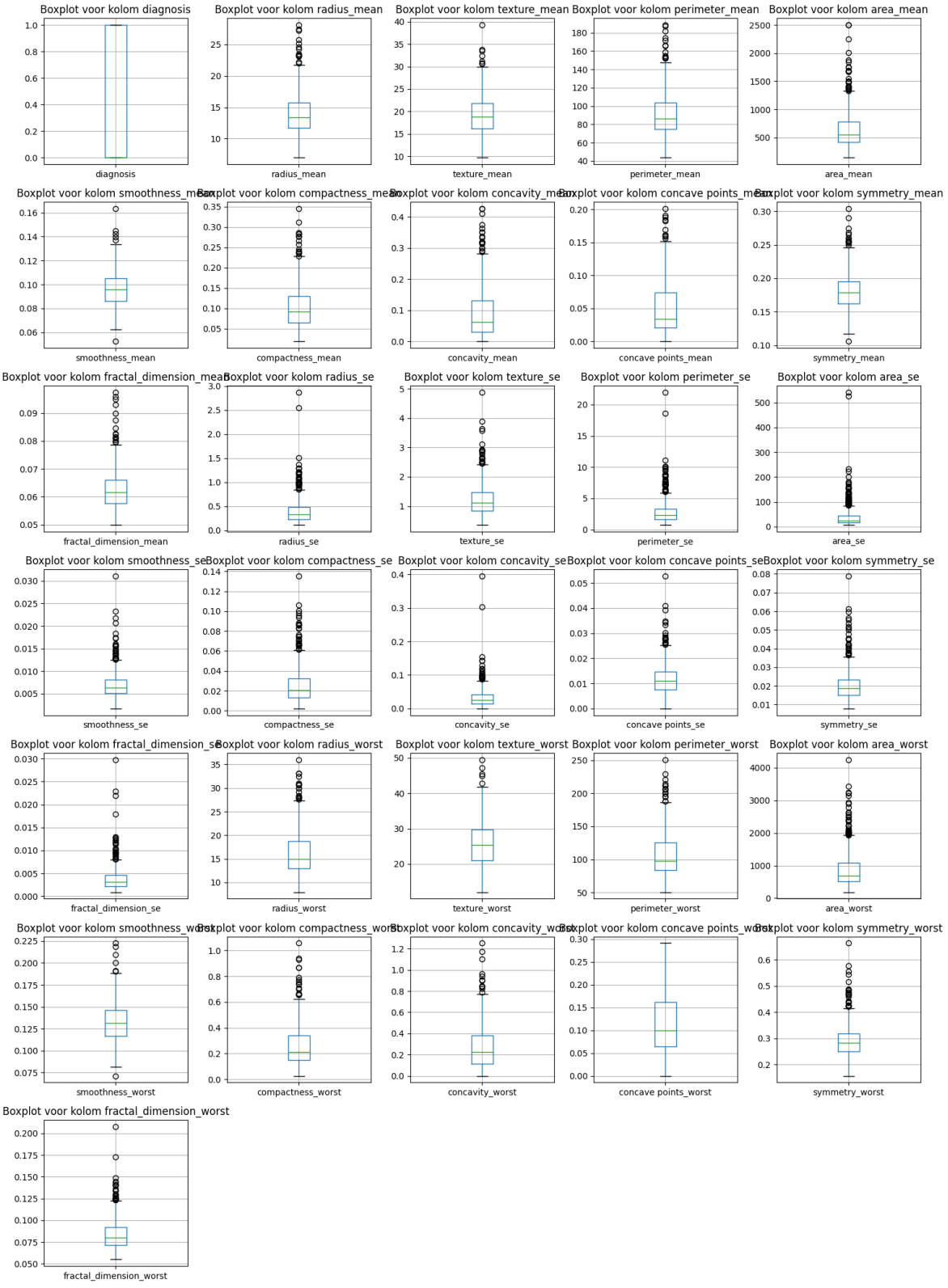
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

Out[ ]:
```
diagnosis
B    0.627417
M    0.372583
Name: proportion, dtype: float64
```

In [ ]:
```
# voor machine learning moet alles een numerieke waarde zijn, dus kwaadaa
data['diagnosis'] = data['diagnosis'].replace({'M': 1, 'B': 0})
```

In de Boxplots hieronder kan gezien worden dat er bij sommige kolommen uitschieters zijn. Deze zijn echter vaak niet alleen, hierdoor nemen wij aan dat dit geen meetfouten zijn en dat het model hiervan kan leren.

In [ ]:
```
aantal_kolommen = len(data.columns)
aantal_kolommen_per_rij = 5
# Afgeronde deling om ervoor te zorgen dat alle kolommen worden gedekt
aantal_rijen = -(-aantal_kolommen // aantal_kolommen_per_rij)

# Genereer boxplots in een rij van 4
fig, axs = plt.subplots(
    aantal_rijen, aantal_kolommen_per_rij, figsize=(15, 3*aantal_rijen))

# Flatten de axs array als het meer dan één rij heeft
axs = axs.flatten()

for i, column in enumerate(data.columns):
    plt.sca(axs[i])
    data.boxplot(column=column)
    plt.title(f'Boxplot voor kolom {column}')

# Verwijder ongebruikte subplots als het aantal kolommen niet een veelvou
for j in range(i+1, len(axs)):
    axs[j].axis('off')

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



In [ ]:
```
# De correlatie maken tussen de verschillende kolommen
corr = data.corr()
```

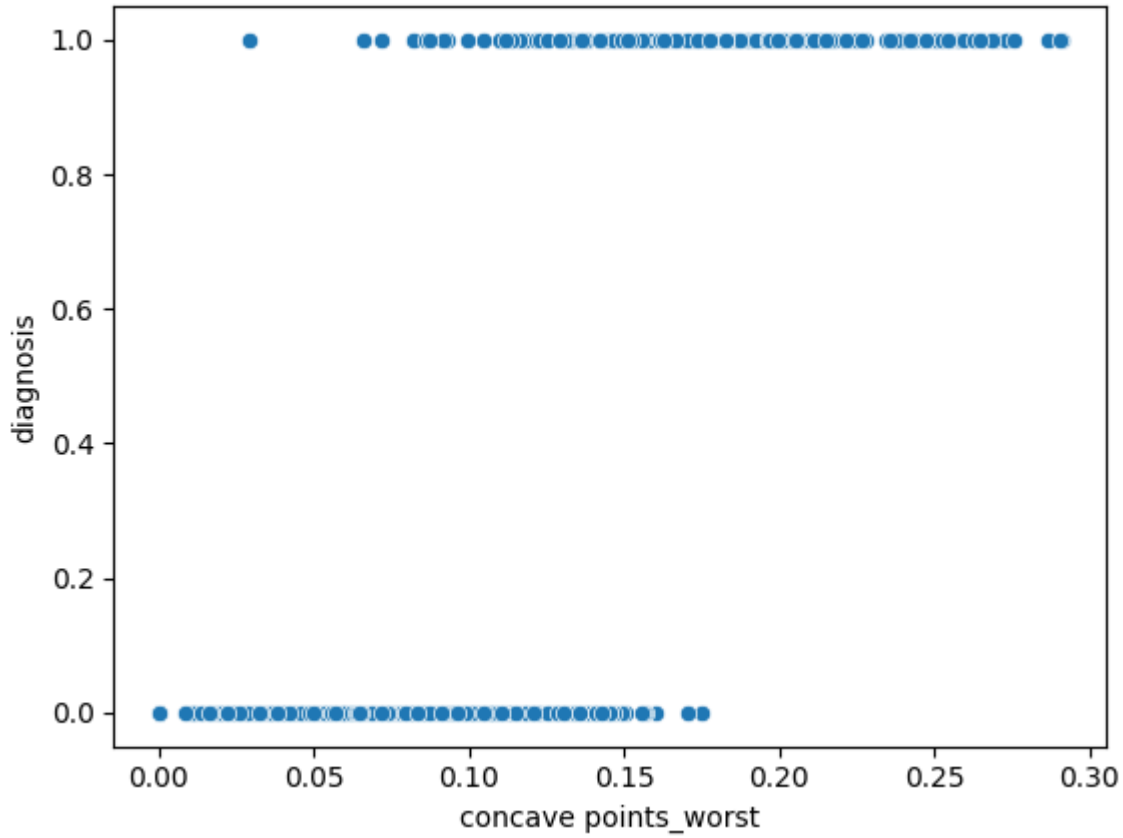
In [ ]:
```
# Hier kan gezien worden welke kolommen de hoogste correlatie hebben met d
# Het spreekt voor zich dat de kolom diagnosis precies een correlatie hee
# concave points_worst heeft dus de hoogste correlatie
corr['diagnosis'].sort_values(ascending=False)
```

Out[ ]:
```
diagnosis                 1.000000
concave points_worst      0.793566
perimeter_worst           0.782914
concave points_mean       0.776614
radius_worst              0.776454
perimeter_mean            0.742636
area_worst                0.733825
radius_mean               0.730029
area_mean                 0.708984
concavity_mean            0.696360
concavity_worst           0.659610
compactness_mean          0.596534
compactness_worst         0.590998
radius_se                 0.567134
perimeter_se              0.556141
area_se                   0.548236
texture_worst             0.456903
smoothness_worst          0.421465
symmetry_worst            0.416294
texture_mean              0.415185
concave points_se         0.408042
smoothness_mean           0.358560
symmetry_mean             0.330499
fractal_dimension_worst   0.323872
compactness_se            0.292999
concavity_se              0.253730
fractal_dimension_se      0.077972
symmetry_se              -0.006522
texture_se               -0.008303
fractal_dimension_mean   -0.012838
smoothness_se            -0.067016
Name: diagnosis, dtype: float64
```

In [ ]:
```
# Het maken van een heatmap zodat er overzichtelijk gezien kan worden wel
px.imshow(corr)
```

In [ ]:
```
# spreiding van de meest gecoreleerde variable met diagnosis,
# Hier kan al een beetje gezien worden dat als de concave points_worst >0
sns.scatterplot(data, x='concave points_worst', y='diagnosis')
```

Out[ ]:
```
<Axes: xlabel='concave points_worst', ylabel='diagnosis'>
```



Het classificatie model

Voor het classifiseren van borstkanker is de accuratiteit belangrijker dan de snelheid van het model. Daarnaast is het geen grote dataset dus in overfitting een gevaar. Hierdoor gebruiken wij het randomforest calssifatie model.

De code met 1 variabele

In [ ]:
```
# Hier wordt de y (wat moet er voorspeld worden) en de x (de features(met
# - bepaald. De y = de diagnose en de x = concave points_worst (de variab
# Bij X wordt .reshape(-1, 1) gebruikt. Hiermee moet python zelf de vorm
# X = data.drop('diagnosis', axis=1)
X = np.array(data['concave points_worst']).reshape(-1, 1)

y = np.array(data['diagnosis'])
```

In [ ]:
```
# De data set aan het verdelen tussen een trai
# In dataset en een test dataset,
# hier wordt 20% de test dataset zodat de dataset niet overgefit wordt
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)
```

In [ ]:
```
# Hier wordt een instantie van het random forest classifier model gemaakt
# Hier zouden ook verschillen Hyperparameters ingesteld kunnen worden,
# echter zijn de standaard instellingen vaak een goed uitgangspunt.

rf_model = RandomForestClassifier(random_state=42)
```

In [ ]:
```
rf_model.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [ ]:
```
# Hier wordt er met het getrainde model de waarden voorspeld op de test da
predictions = rf_model.predict(X_test)
```

In [ ]:
```
# Hier wordt de nauwkeurigheid berekend
accuracy = accuracy_score(y_test, predictions)
print(f"Nauwkeurigheid: {accuracy}")

# Andere evaluatiemetingen, zoals de precisie en de f1score
# Daarnaast wordt ook een matrix laten zien waar de voorspellingen goed zi
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
Nauwkeurigheid: 0.847953216374269
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       108
           1       0.78      0.81      0.80        63

    accuracy                           0.85       171
   macro avg       0.84      0.84      0.84       171
weighted avg       0.85      0.85      0.85       171

[[94 14]
 [12 51]]
```

Met 1 variabele is het model dus ongeveer 84% van de keren correct. Dit is best hoog, echter is dit ook de variabele die die hoogste correlatie heeft met de diagnose. Deze nauwkeurigheid is dus wel realistisch.

De code met meerde variabelen

Hieronder is een stuk code geschreven om variabelen toe te voegen bij de X. Het stoppen van het toevoegen van variabelen stopt als de nauwkeurigheid niet meer hoger wordt.

In [ ]:
```
# De variable die wij willen voorspellen, dus de diagnose
y2 = data["diagnosis"]

# Hier wordt een lege lijst aangemaakt om de nauwkeurigheid van de versch
accuracy_list = []


# We beginnn met de hoogst gecorreleerde kolom omdat dit ook de variabele
selected_columns = data.corr()["diagnosis"].sort_values(
    ascending=False).index[1:2].tolist()

# Print de geselecteerde kolommen om te controleren of ze overeenkomen me
print("Selected Columns:", selected_columns)

# Hier wordt de beste combinaties van kolommen en de nauwkeurigheid van d
# Deze waarden kunnen later geprint worden zodat dit in een duidelijke co
best_columns = selected_columns
```

```
best_accuracy = 0.0

# Hier wordt de tolerantie aangemaakt. De tolerantie is hier 0.001, dus wa
# of slechter wordt stopt het model.
tolerance = 0.001

# Hier wordt een loop gemaakt die doorgaat totdat de nauwkeurigheid niet m
while True:
    # Hier wordt de volgende best presterende kolom toegevoegd aan de kol
    best_performing_column = None
    best_performing_accuracy = 0.0

    # Itereer over de kolommen die niet in de geselecteerde kolommen zitt
    for column in data.columns.difference(["diagnosis"] + selected_columns
        current_columns = selected_columns + [column]
        X2 = data[current_columns]

        # Hier wordt de data in een train en een test dataset gesplits. D
        X2_train, X2_test, y2_train, y2_test = train_test_split(
            X2, y2, test_size=0.3, random_state=42)

        # hier wordt het model aangemaakt en wordt het getraind.
        rf_model = RandomForestClassifier(random_state=42)
        rf_model.fit(X2_train, y2_train)

        # Hier worden de voorspellingen op de test dataste gedaan
        # Met deze voorspellingen kan ook de nauwkeurigheid berekend word
        predictions = rf_model.predict(X2_test)

        # Hier wordt de Nauwkeurigheid berekend
        # Op basis van deze nauwkeurigheid wordt er gekeken of het aantal
        # of dat dit de optimale combinatie is van variabelen.
        current_accuracy = accuracy_score(y2_test, predictions)

        # Hier wordt een if statement gemaakt om te kijken of de kolom be
        if current_accuracy > best_performing_accuracy:
            best_performing_accuracy = current_accuracy
            best_performing_column = column

    # Hier wordt berekend of de toevoeging van een nieuwe kolom de nauwke
    if best_performing_column is not None:
        selected_columns.append(best_performing_column)
        accuracy_list.append(best_performing_accuracy)
        print(
            f"Model met {len(selected_columns)} kolommen: {selected_columns

        # Hier wordt de beste combinatie van kolommen veranderd als de hu
        if best_performing_accuracy > best_accuracy:
            best_accuracy = best_performing_accuracy
            best_columns = selected_columns
        else:
            # Als de nauwkeurigheid minder verbeterd dan wat de tolerantie
            if best_accuracy - best_performing_accuracy < tolerance:
                break
    else:
        # Als er helemaal geen verbetering in de nauwkeurigheid wordt gez
        break

# Hier worden de uiteindelijke resulaten geprint zodat dit gezien kan wor
print(
```

```
    f"\nUiteindelijk geselecteerde kolommen: {best_columns}, Nauwkeurighe
```

```
Selected Columns: ['concave points_worst']
Model met 2 kolommen: ['concave points_worst', 'area_worst'], Nauwkeurighe
id: 0.9532163742690059
Model met 3 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst'], Nauwkeurigheid: 0.9649122807017544
Model met 4 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst', 'area_mean'], Nauwkeurigheid: 0.9707602339181286
Model met 5 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst', 'area_mean', 'fractal_dimension_worst'], Nauwkeurigheid: 0.97660818
71345029
Model met 6 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean'], Nau
wkeurigheid: 0.9707602339181286
Model met 7 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean', 'are
a_se'], Nauwkeurigheid: 0.9707602339181286
Model met 8 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean', 'are
a_se', 'compactness_se'], Nauwkeurigheid: 0.9707602339181286
Model met 9 kolommen: ['concave points_worst', 'area_worst', 'smoothness_w
orst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean', 'are
a_se', 'compactness_se', 'radius_se'], Nauwkeurigheid: 0.9707602339181286
Model met 10 kolommen: ['concave points_worst', 'area_worst', 'smoothness_
worst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean', 'ar
ea_se', 'compactness_se', 'radius_se', 'concave points_se'], Nauwkeurighei
d: 0.9707602339181286
Model met 11 kolommen: ['concave points_worst', 'area_worst', 'smoothness_
worst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean', 'ar
ea_se', 'compactness_se', 'radius_se', 'concave points_se', 'symmetry_wors
t'], Nauwkeurigheid: 0.9766081871345029

Uiteindelijk geselecteerde kolommen: ['concave points_worst', 'area_worst
', 'smoothness_worst', 'area_mean', 'fractal_dimension_worst', 'concave po
ints_mean', 'area_se', 'compactness_se', 'radius_se', 'concave points_se',
'symmetry_worst'], Nauwkeurigheid: 0.9766081871345029
```

Wat hier opvalt is dat de nauwkeurigheid met 5 kolommen en met 11 kolommen hetzelfde is, namelijk 0.9766081871345029. De combinatie van kolommen die wij kiezen zijn de 11 kolommen. Dit omdat het model dan betrouwbaarder wordt omdat er meer kolommen worden meegenomen, terwijl het model niet overgefit wordt op deze dataset omdat de totale dataset 30 kolommen bevat. Wat ons ook opvalt is dat de kolommen ook niet op volgorde van correlatie zijn die wij hebben gevonden bij de data verkenning, zie de code hieronder als herinnering. Dit is voor ons een verassing dus hebben wij gekeken of de top 5 variabelen met de hoogste correlatie een betere nauwkeurigheid krijgen dan de 5 die een nauwkeurigheid hebben gekregen van 0.9766081871345029.

In [ ]:
```
data.corr()["diagnosis"].sort_values(ascending=False).head(10)
```

Out[ ]:
```
diagnosis                1.000000
concave points_worst     0.793566
perimeter_worst          0.782914
concave points_mean      0.776614
radius_worst             0.776454
perimeter_mean           0.742636
area_worst               0.733825
radius_mean              0.730029
area_mean                0.708984
concavity_mean           0.696360
Name: diagnosis, dtype: float64
```

In [ ]:
```
# Dit zijn de 5 kollommen met de hoogste correlatie tot de diagnose
selected_columns = ['concave points_worst', 'perimeter_worst',
                    'concave points_mean', 'radius_worst', 'perimeter_mean
data_selected = data[selected_columns]

# Hier wordt de dataset met de 5 kolommen gesplitst in test en train data
X3 = data_selected.drop('diagnosis', axis=1)
y3 = data_selected['diagnosis']
X3_train, X3_test, y3_train, y3_test = train_test_split(
    X3, y3, test_size=0.3, random_state=42)

# Hier wordt het random forest classifier model geinitieerd en wordt de ra
# Daarnaast wordt het model ook getrained
clf = RandomForestClassifier(random_state=42)
clf.fit(X3_train, y3_train)

# Hier wordt de diagnose voorspeld. Aan de hand van deze voorspelling kan
y3_pred = clf.predict(X3_test)

# Hier wordt de nauwkeurigheid berekend, met wat het daadwerkelijk is en w
accuracy = accuracy_score(y3_test, y3_pred)
print(f'Nauwkeurigheid: {accuracy}')

# In het classificatie rapport kan de kwaliteit van het omdel beter bekeke
print('Classificatierapport:')
print(classification_report(y3_test, y3_pred))

# In de verwarringsmatrix kan gezien worden waar de fouten en goede voorsp
# Waar het goedaardig voorspeld is en waar het goedaardig is.
print('Verwarringsmatrix:')
print(confusion_matrix(y3_test, y3_pred))
```

```
Nauwkeurigheid: 0.9532163742690059
Classificatierapport:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       108
           1       0.95      0.92      0.94        63

    accuracy                           0.95       171
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171

Verwarringsmatrix:
[[105   3]
 [  5  58]]
```

De combinatie van deze 5 kolommen geeft dus een nauwkeurigheid van 0.9532163742690059. Dit is lager dat de combinatie van 5 kolommen die wij hiervoor hebben gevonden.

De kolommen waarmee het model uiteindelijk het beste werkt zijn dus: 'concave points_worst', 'area_worst', 'smoothness_worst', 'area_mean', 'fractal_dimension_worst', 'concave points_mean', 'area_se', 'compactness_se', 'radius_se', 'concave points_se', 'symmetry_worst'. Deze kolommen geven dus een nauwkeurigheid van 0.9766081871345029.

Regression

Laden data

In [ ]:
```
import pandas as pd

df = pd.read_csv('./data/regression/train.csv')
df[['SalePrice', 'GrLivArea']].head()
```

Out[ ]:

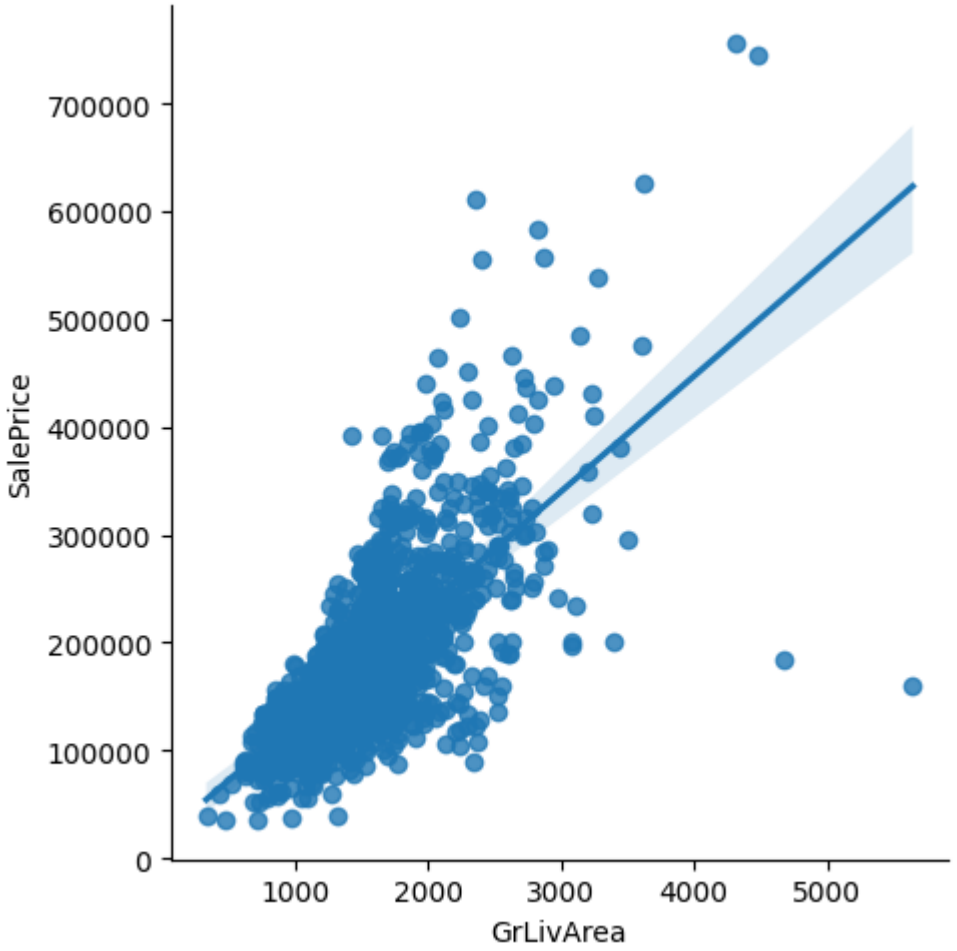
	SalePrice	GrLivArea
0	208500	1710
1	181500	1262
2	223500	1786
3	140000	1717
4	250000	2198

Verkennende analyse

In [ ]:
```
import plotly.express as px
import seaborn as sns

corr = df.select_dtypes('number').corr()

fig = px.imshow(corr)
fig.update_layout(
    yaxis={"tickfont": {"size": 5}},
    xaxis={"tickfont": {"size": 5}})

fig.show()
```

In [ ]:
```
sns.lmplot(df, x="GrLivArea", y="SalePrice")
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd9a961bfd0>
```



Trainen regressiemodellen

In [ ]:
```
import statsmodels.api as sm
```

In [ ]:
```
# Ordinary Least Squares regression
reg = sm.OLS(df.SalePrice, sm.add_constant(df.GrLivArea)).fit()
reg.summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared:**	0.502
Model:	OLS	**Adj. R-squared:**	0.502
Method:	Least Squares	**F-statistic:**	1471.
Date:	Wed, 22 Nov 2023	**Prob (F-statistic):**	4.52e-223
Time:	15:02:11	**Log-Likelihood:**	-18035.
No. Observations:	1460	**AIC:**	3.607e+04
Df Residuals:	1458	**BIC:**	3.608e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.857e+04	4480.755	4.144	0.000	9779.612	2.74e+04
GrLivArea	107.1304	2.794	38.348	0.000	101.650	112.610

Omnibus:	261.166	**Durbin-Watson:**	2.025
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3432.287
Skew:	0.410	**Prob(JB):**	0.00
Kurtosis:	10.467	**Cond. No.**	4.90e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.9e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [ ]:
```
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
```

KNeighbors with one input variable

In [ ]:
```
y = np.array(df.SalePrice)
X = np.array(df['GrLivArea']).reshape(-1, 1)

reg = KNeighborsRegressor().fit(X, y)

print(f"De regressiescore van dit model is: {reg.score(X,y)}")
print("met Y = SalePrice en X = GrLivArea")
```

```
De regressiescore van dit model is: 0.6040086934190917
met Y = SalePrice en X = GrLivArea
```

Prediction with user input

In [ ]:
```
reg.predict([[1400], [1800]])
```

Out[ ]:
```
array([166205.2, 222280. ])
```

KNeighbors tested on same data as trained (overfitted)

KNeirestNeighbours is een simpel algoritme dat vaak gebruikt wordt om classificatieproblemen op te lossen. Het werkt door het bereken van afstanden van het te voorspellen punt tot andere punten. In de praktijk kan dit bijvoorbeeld gebruikt worden om te voorspellen welke soort een bloem is op basis van de lengte van de bladeren en de dikte van de stam. Voor regressieproblemen zal het midden van de matchende punten gebruikt worden als voorspelling. Het voordeel van KNearestNeighbours is dat het een relatief simpel algoritme is en dat het snel te trainen valt, het nadeel is dat het niet erg slim is en niet in alle situaties goed werkt.

In [ ]:
```
# Defineer x parameters
x_params = ['GrLivArea', 'OverallQual', 'KitchenAbvGr']

# Defineer data als numpy arrays, dit kan gebruikt worden voor het trainer
y = np.array(df.SalePrice)
X = np.array(df[x_params])

# Train het model met de eerder gedefineerde data
reg = KNeighborsRegressor(weights="distance").fit(X, y)

print(f"De regressiescore van dit model is: {reg.score(X,y)}")
print(f"met Y = SalePrice en de X parameters: {x_params}")
```

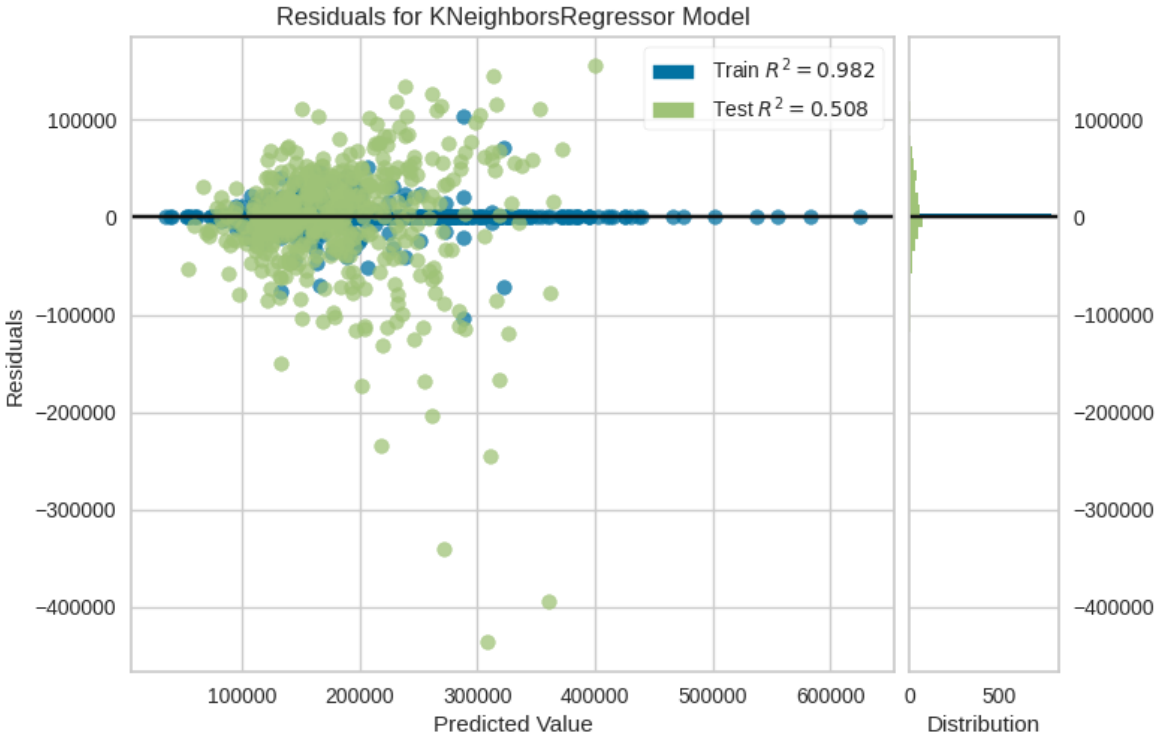
```
De regressiescore van dit model is: 0.9816175256270805
met Y = SalePrice en de X parameters: ['GrLivArea', 'OverallQual', 'Kitche
nAbvGr']
```

KNeighbors train/test split, prove of overfitting

In [ ]:
```
x_params = ['GrLivArea', 'OverallQual', 'KitchenAbvGr']

y = np.array(df.SalePrice)
X = np.array(df[x_params])

# Split de data in een training en een test dataset zodat later de nauwkeu
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)

reg = KNeighborsRegressor(weights="distance").fit(X_train, y_train)

print(f"De regressiescore van dit model is: {reg.score(X_test,y_test)}")
print(f"met Y = SalePrice en de X parameters: {x_params}")
```

```
De regressiescore van dit model is: 0.5075977757034607
met Y = SalePrice en de X parameters: ['GrLivArea', 'OverallQual', 'Kitche
nAbvGr']
```

In [ ]:
```
from yellowbrick.regressor import ResidualsPlot

x_params = ['GrLivArea', 'OverallQual', 'KitchenAbvGr']

y = np.array(df.SalePrice)
X = np.array(df[x_params])

# Split de data in een training en een test dataset zodat later de nauwke
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)

model = KNeighborsRegressor(weights="distance")
visualizer = ResidualsPlot(model)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualiz
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
visualizer.show()
```



Out[ ]:
```
<Axes: title={'center': 'Residuals for KNeighborsRegressor Model'}, xlabe
l='Predicted Value', ylabel='Residuals'>
```