Verdifastsettelse av fritidsboliger

In [1]:

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
import pandas as pd
pd.options.display.float_format = '{:.4f}'.format
from pandas_summary import DataFrameSummary
import numpy as np
from catboost import CatBoostRegressor, Pool
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection.partial_dependence import plot_partial_dependence
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.dummy import DummyRegressor
from sklearn.tree import export_graphviz
from IPython.display import Image
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import shap
shap.initjs()
from pycebox.ice import ice, ice_plot
import lime.lime_tabular
from lime import submodular_pick
```

Innlasting av data, og initiell datautforskning (EDA)

Datasettet som er brukt i denne notebooken er et fritt tilgjengelig datasett fra vanlig boligsalg i Melbourne i Australia. Datasettet er hentet fra data-science platformen Kaggle, og finnes tilgjengelig her (https://www.kaggle.com/anthonypino/melbourne-housing-market)

In [2]:

```
datapath = "MELBOURNE_HOUSE_PRICES_LESS.csv"
df = pd.read_csv(datapath, parse_dates=["Date"])
```

Oppsummering av data

Forklaring av kolonneverdier

Suburb: Suburb / forstad

Address: Address

Rooms: Number of rooms

Price: Price in Australian dollars

Method: S - property sold; SP - property sold prior; PI - property passed in; PN - sold prior not disclosed; SN - sold not disclosed; NB - no bid; VB - vendor bid; W - withdrawn prior to auction; SA - sold after auction; SS - sold after auction price not disclosed. N/A - price or highest bid not available.

Type:

- h - house,cottage,villa, semi,terrace
- u - unit, duplex
- t - townhouse

SellerG: Real Estate Agent

Date: Date sold

Distance: Distance from CBD in Kilometres / Avstand fra sentrum

Regionname: General Region (West, North West, North, North east ...etc)

Propertycount: Number of properties that exist in the suburb.

CouncilArea: Governing council for the area

In [3]:

```
summ = DataFrameSummary(df).summary().T
summ.sort_values(by=["types", "count", "uniques"])
```

Out[3]:

	count	mean	std	min	25%	50%
Type	NaN	NaN	NaN	NaN	NaN	NaN
Regionname	NaN	NaN	NaN	NaN	NaN	NaN
Method	NaN	NaN	NaN	NaN	NaN	NaN
CouncilArea	NaN	NaN	NaN	NaN	NaN	NaN
Suburb	NaN	NaN	NaN	NaN	NaN	NaN
SellerG	NaN	NaN	NaN	NaN	NaN	NaN
Address	NaN	NaN	NaN	NaN	NaN	NaN
Date	NaN	NaN	NaN	NaN	NaN	NaN
Price	48433.0000	997898.2415	593498.9190	85000.0000	620000.0000	830000.0000
Rooms	63023.0000	3.1106	0.9576	1.0000	3.0000	3.0000
Distance	63023.0000	12.6848	7.5920	0.0000	7.0000	11.4000
Postcode	63023.0000	3125.6739	125.6269	3000.0000	3056.0000	3107.0000
Propertycount	63023.0000	7617.7281	4424.4232	39.0000	4380.0000	6795.0000

Ut ifra statistikken, kan vi observere blant annet:

- Mange av featurene er kategoriske, enkelte med høy kardinalitet
- Postcode har tallverdi, men er en kategorisk variabel
- 23% av radene mangler pris

Vi velger å fjerne radene som inneholder hus uten oppgitt salgspris.

In [4]:

```
df = df.dropna(subset=["Price"])
```

In [5]:

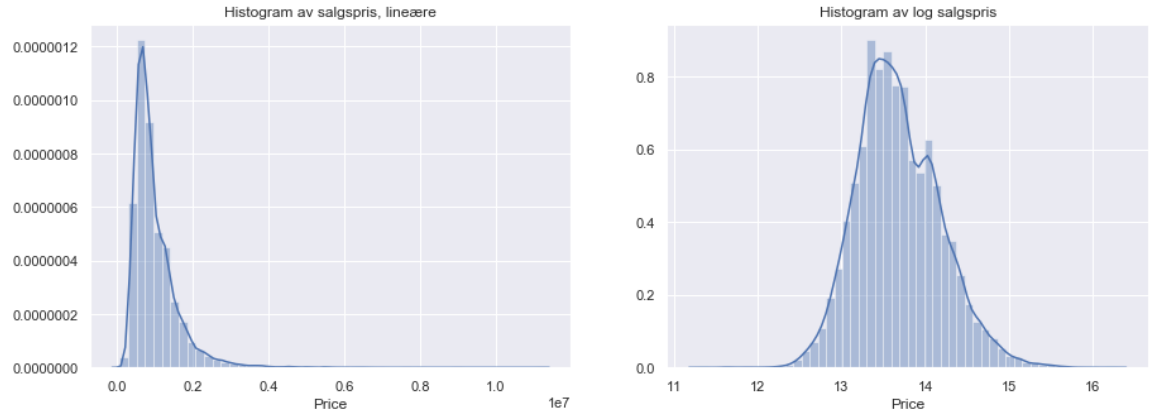
```
df.head()
```

Out[5]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Postcode	R
0	Abbotsford	49 Lithgow St	3	h	1490000.0000	S	Jellis	2017-01-04	3067	
1	Abbotsford	59A Turner St	3	h	1220000.0000	S	Marshall	2017-01-04	3067	
2	Abbotsford	119B Yarra St	3	h	1420000.0000	S	Nelson	2017-01-04	3067	
3	Aberfeldie	68 Vida St	3	h	1515000.0000	S	Barry	2017-01-04	3040	
4	Airport West	92 Clydesdale Rd	2	h	670000.0000	S	Nelson	2017-01-04	3042	

In [6]:

```
fig, ax = plt.subplots(1,2, figsize=(15,5))
sns.distplot(df["Price"], ax=ax[0]).set_title("Histogram av salgspris, lineære");
sns.distplot(df["Price"].apply(lambda x: np.log(x)), ax=ax[1]).set_title("Histogram av
 log salgspris");
```



Ut ifra statistikken, kan vi observere blant annet:

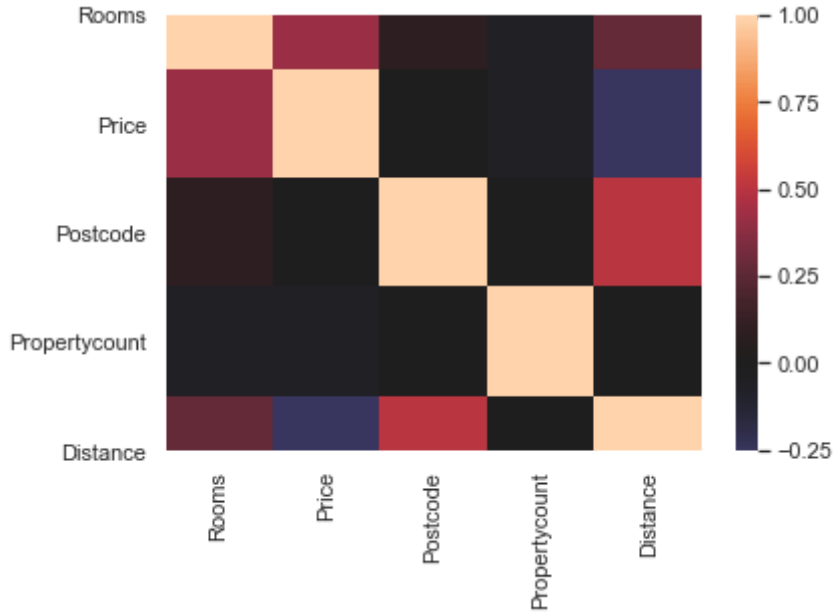
- Mange av featurene er kategoriske, enkelte med høy kardinalitet
- Postcode har tallverdi, men er en kategorisk variabel
- 23% av radene mangler pris

Eksplorativ dataanalyse (EDA)

Ved å se på korrelasjonsmatrisen til dataen, kan vi se om det finnes noen åpenbare sammenhenger mellom salgspris og noen av de andre featurene

In [7]:

```
sns.heatmap(df.corr(), center=0);
```

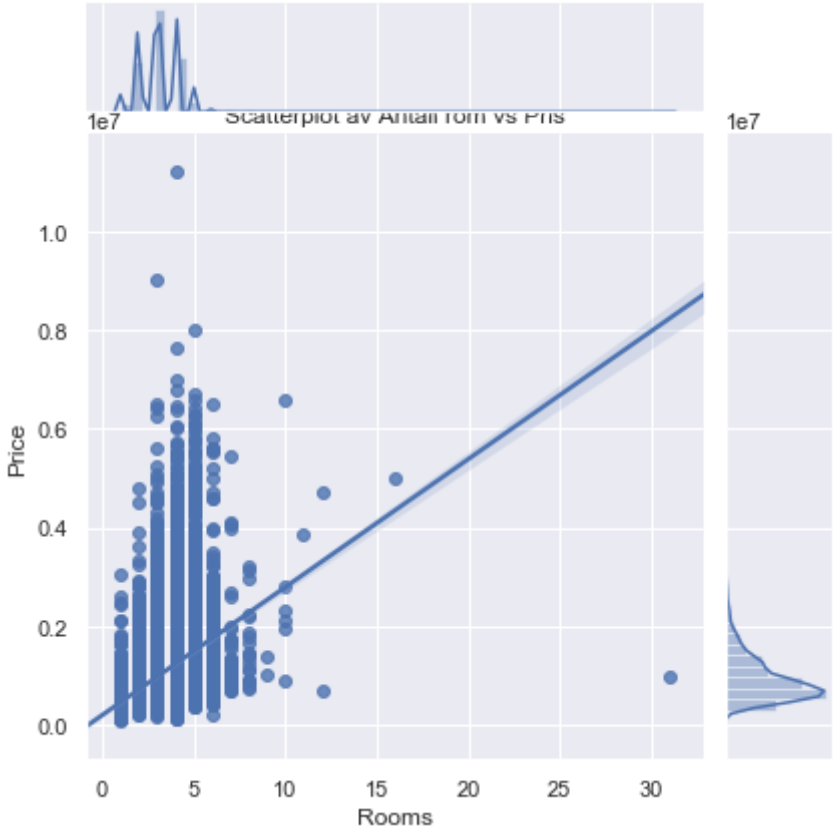


Inspeksjon av korrelerte features

På figurene nedenfor er det vist det parvise forholdet mellom feature og target-verdi. En regresjonslinje er tilpasset hvert enkelt plot.

In [8]:

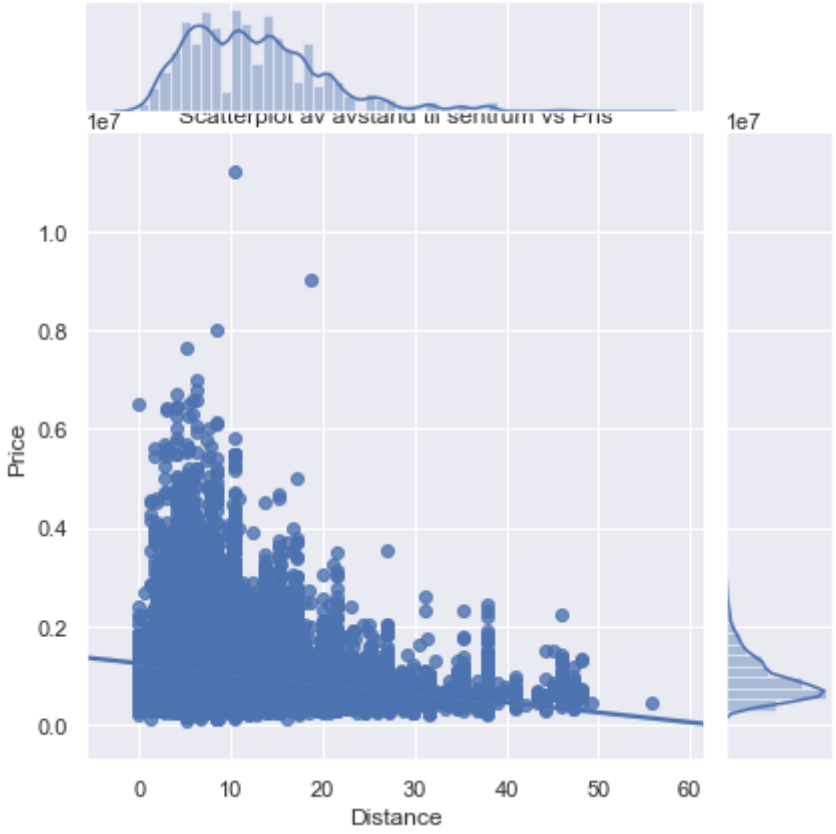
```
g1 = sns.JointGrid(x="Rooms", y="Price", data=df)
g1.plot(sns.regplot, sns.distplot)
plt.title("Scatterplot av Antall rom vs Pris");
```



Vi kan observere at pris har positiv korrelasjon med antall rom i boligen.

In [9]:

```
g2 = sns.JointGrid(x="Distance", y="Price", data=df)
g2.plot(sns.regplot, sns.distplot)
plt.title("Scatterplot av avstand til sentrum vs Pris");
```



Vi kan observere at avstand til sentrum korrelerer negativt med pris

Feature engineering

Fiks datatyper

In [10]:

```
categorical_feature_names = ["Suburb", "Type", "Method", "Regionname", "CouncilArea"]
df[categorical_feature_names] = df[categorical_feature_names].astype("category")
```

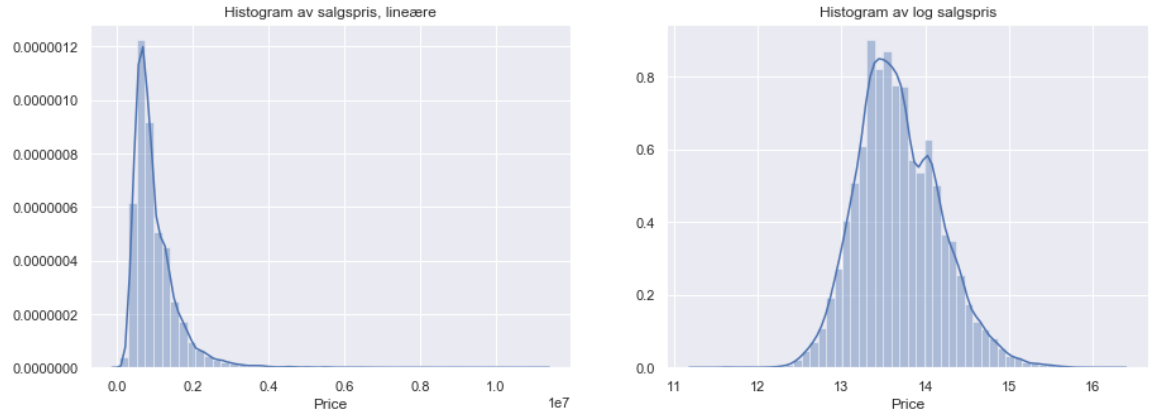
Gjør om dato til år og måned

In [11]:

```
df["Month"] = df["Date"].dt.month
df["Year"] = df["Date"].dt.year
```

In [12]:

```
fig, ax = plt.subplots(1,2, figsize=(15,5))
sns.distplot(df["Price"], ax=ax[0]).set_title("Histogram av salgspris, lineære");
sns.distplot(df["Price"].apply(lambda x: np.log(x)), ax=ax[1]).set_title("Histogram av
 log salgspris");
```



Gjør om til log-pris

Som vist på figuren ovenfor, har salgsprisen til boligene i en linæer skale en lang "hale" mot de høyere prisene. En bedre måte å predikere prisen vil være å ta en logaritmisk transformasjon av salgsprisen. På denne måten kan vi som vist til høyre i figuren ovenfor få en mer normalfordelt variabel

In [13]:

```
df["LogPrice"] = df["Price"].apply(np.log)
```

In [14]:

```
columns_to_drop = ["Address", "Date", "Price", "SellerG", "LogPrice"]
```

In [15]:

```
df.head()
```

Out[15]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Postcode
0	Abbotsford	49 Lithgow St	3	h	1490000.0000	S	Jellis	2017-01-04	3067
1	Abbotsford	59A Turner St	3	h	1220000.0000	S	Marshall	2017-01-04	3067
2	Abbotsford	119B Yarra St	3	h	1420000.0000	S	Nelson	2017-01-04	3067
3	Aberfeldie	68 Vida St	3	h	1515000.0000	S	Barry	2017-01-04	3040
4	Airport West	92 Clydesdale Rd	2	h	670000.0000	S	Nelson	2017-01-04	3042

Spesifiser kolonner som ikke skal brukes for å trene modellen

- Address har meget høy kardinalitet, og vil mest sannsynlig ikke inneholde signaler
- Vi har allerede hentet ut år og måned fra datokolonnen. Modellene vi bruker klarer ikke bruke datetime kolonner
- Vi ønsker å se bort ifra hvilken selger som har solgt huset
- Vi må også fjerne målvariablen (Price og LogPrice) fra treningssettet

In [16]:

```
columns_to_drop = ["Address", "Date","SellerG" ,"Price", "LogPrice"]
```

Evalueringskriterier og Metrikker

Kvalitetsmetrikker

Vi har valgt å bruke kvalitetsmetrikken som beskrives i SSBs modell for beregning av boligformue; «Omtrent X% av de estimerte markedsverdiene ligger innenfor +/- Y% av observert verdi».

Evalueringsmetrikker

For å evaluere modellytelse for en regresjonsmodell har vi valgt å benytte evalueringsmetrikkene beskrevet under Forklaring av variabler:

- y er den faktiske salgsprisen av boligen
- $\hat{y}$ er den predikerte salgsprisen fra modellen
- $\bar{y}$ er gjennomsnitsverdien av salgsprisen
- log betegner den naturlige logaritmen
- k er en faktor som brukes under utregning av WALE

Winsorized Absolute Log Error (WALE)

WALE er en nyttig metrikk i tilfeller der store outliers i datasettet vil føre til at andre metrikker vil domineres av disse. Ved å sette k=0.4=ln(100 % + 50 %), vil alle salgspriser som predikeres til å være mer enn +/- 50 % avkortes til dette maksimale avviket.

WALE er definert på følgende måte:
\begin{align} WALE = \frac{1}{N}\sum_{i=1}^{N}min(|log(\hat{y}_i) - log(y_i)|, k) \end{align}

Tolkning: Jo lavere tall, jo bedre.

WALE har etter vår kunnskap ingen implementasjon i noen open-source bibliotek, og er definert som en python-funksjon nedenfor, der $k=0.4$

In [17]:

```
def winsorized_absolute_log_error(y_true, y_pred, epsilon=1e-7):
    return np.mean(np.minimum(np.abs(np.log(y_pred+epsilon) - np.log(y_true+epsilon)),
0.4))
```

Root Mean Squared Error (RMSE)

RMSE er en metrikk vil "straffe" prediksjoner som bommer stort \begin{align} RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \end{align}

Tolkning: Jo lavere tall, jo bedre.

Mean Absolute Error (MAE)

Fordelen med MAE, er at man intuitivt kan forholde seg til verdien, som sier hvor mye modellen i gjennomsnitt bommer på faktisk salgspris. \begin{align} MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i| \end{align}

Tolkning: Samme enheter som målvariablen (Salgspris i Australske Dollar). Jo lavere tall, jo bedre.

R^2-score / Coefficient of determination

R^2 beskriver andelen varians som forklares av de uavhengige variablene i modellen. Metrikken gir oss en indikasjon på "goodness of fit", og dermed et mål på hvor godt modellen klarer å predikere usett data riktig. \begin{align} R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \end{align}

Tolkning: Tar verdien 0-1, der 0 betyr at modellen gjør det like godt som å alltid predikere gjennomsnittet, mens 1 betyr at modellens prediksjoner passer perfekt til dataen.

Rammeverket scikit-learn har gode implementasjoner av disse metrikkene, som vil brukes i denne notebooken

Vi definerer også en funksjon for å regne ut alle metrikkene

In [18]:

```
def get_metrics(true, pred, method="unknown", print_metrics=True, log_target=True):
    if log_target:
        true, pred = np.exp(true), np.exp(pred)

    WALE = winsorized_absolute_log_error(true,pred)
    RMSE = np.sqrt(mean_squared_error(true, pred))
    MAE = mean_absolute_error(true, pred)
    R2 = r2_score(true, pred)
    return_dict = {"WALE": WALE, "RMSE": RMSE, "MAE": MAE, "R2": R2, "Method": method}
    if print_metrics:
        for key, value in return_dict.items():
            print(f"{key}: {value}")
    return return_dict
```

Oppdeling av trening, validering og holdout-test-sett

Vi velger å bruke 80% av dataen til modellering/trening, 10% av dataen brukes til å validere modellen, mens 10% brukes som holdout-test-set for å evaluere modellen. Ofte benytter man seg kun av et treningssett og et testsett. For noen av maskinlæringsmodellene vil deler av testsettet brukes til å finjustere hyperparameterne til modellen. For slike modeller bør man også ha et såkalt *holdout testsett* som man evaluerer når modellen er ferdig *finjustert*. Måten vi deler opp dette er på tid, der vi bruker den eldste dataen til å modellere, mens nyeste data brukes til å validere og evaluere

In [19]:

```
df = df.sort_values(by="Date")
holdout_test_idx = df.shape[0] - int(df.shape[0] * 0.1)
val_idx = df.shape[0] - int(df.shape[0] * 0.1) * 2
```

In [20]:

```
def split_dataset(dataframe, val_index, holdout_test_index):
    train = dataframe.iloc[:val_index]
    val = dataframe.iloc[val_index:holdout_test_index]
    holdout_test = dataframe.iloc[holdout_test_index:]
    return train, val, holdout_test
```

Prediksjon av salgspris

Baseline-modell

Vi setter opp en baseline-modell, som alltid bør bli "slått" av en modell vi lager:

- En modell som alltid predikerer gjennomsnittet av salgsprisen

In [21]:

```
dummy_train, dummy_val, dummy_test = split_dataset(df, val_idx, holdout_test_idx)
```

In [22]:

```
dummy = DummyRegressor(strategy="mean")
dummy.fit(dummy_train, dummy_train["LogPrice"]);
```

In [23]:

```
dummy_pred = dummy.predict(dummy_test)
```

Metrikker for baseline-modell

In [24]:

```
dummy_metrics = get_metrics(dummy_test["LogPrice"], dummy_pred, method="Baseline")
```

```
WALE: 0.2729600649523902
RMSE: 547051.137129804
MAE: 364438.6487996448
R2: -0.034421471832561945
Method: Baseline
```

Fremgangsmåte 1: Lineær regresjon, med one-hot encoding av kategoriske variabler

En lineær-regresjonsmodell forsøker å finne de koeffisientene til input-featurene som minimerer summen av kvadratisk feil (Sum of squared errors).
\begin{align} \hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_N x_N \end{align}

der β_0 er intercept, $\beta_1-\beta_N$ er de lærte koeffisientene til featurene, og x_1-x_N er verdien til featurene

Preprossesering av kategoriske variabler

For å kunne bruke kategoriske variabler i en lineær regresjon må man transformere disse. Mens numeriske variabler har en rekkefølge og innbyrdes ordning, har ikke kategoriske variabler den samme tolkningen. Variablen *Suburb*, eller forstad i dette datasettet har ikke en innbyrdes ordning, og vi vet ikke f.eks. hvordan forstadene *Essendon* og *Brunswick* forholder seg til hverandre. En måte å gjøre denne encodingen er ved å bruke one-hot-encoding, der hver ulike verdi variablen tar representeres med en egen kolonne, som sier om verdien er tilstede i raden eller ikke.

One-hot-encoding

In [25]:

```
dummies = pd.get_dummies(df[categorical_feature_names])
non_dummies = df.drop(columns=categorical_feature_names)
lr_df = non_dummies.join(dummies)
```

In [26]:

```
dummies.iloc[:5,:10]
```

Out[26]:

	Suburb_Abbotsford	Suburb_Aberfeldie	Suburb_Airport West	Suburb_Albanvale	Suburb_
56441	0	0	0	0	
56456	0	0	0	0	
56455	0	0	0	0	
56454	0	0	0	0	
56452	0	0	0	0	

Inndeling i trening, validering og test

In [27]:

```
lr_train, lr_val, lr_holdout_test = split_dataset(lr_df, val_idx, holdout_test_idx)
lr_X_train = lr_train.drop(columns=columns_to_drop)
lr_y_train = lr_train["LogPrice"]

lr_X_val = lr_val.drop(columns=columns_to_drop)
lr_y_val = lr_val["LogPrice"]

lr_X_test = lr_holdout_test.drop(columns=columns_to_drop)
lr_y_test = lr_holdout_test["LogPrice"]
```

In [28]:

```
lr_n_rows, lr_n_columns = lr_X_train.shape
```

In [29]:

```
print(lr_n_columns)
```

```
426
```

Vi ender altså opp med et datasett som inneholder 426 features

Modellering

Først trener vi en lineær regresjonsmodell

In [30]:

```
linreg = LinearRegression()
linreg.fit(lr_X_train, lr_y_train);
```

Vi kan så undersøke intercept og koeffisientene til linreg-modellen

In [31]:

```
linreg.intercept_
```

Out[31]:

```
52345614.8756702
```

... og de 10 første av de 426 koeffisientene til lineærregresjonen

In [32]:

```
linreg.coef_[:10]
```

Out[32]:

```
array([ 1.99556952e-01, -2.47597720e+03,  1.64778393e+02,  3.55218439e+05,
        7.24546667e-03,  9.19909510e-02,  2.57897089e+06, -1.30345746e+06,
       -2.64517866e+06,  2.85626371e+05])
```

Evaluering

Så bruker vi modellen til å predikere test-settet vårt. I tabellen nedenfor kan vi se faktisk verdi (true) samt predikert verdi(pred) og absoluttverdien av differansen mellom faktisk og predikert verdi

In [33]:

```
lr_pred = linreg.predict(lr_X_test.values)
```

Klipping av ugyldige verdier

For at metrikkene som er valgt skal gi mening, klipper vi alle verdiene som er negative, til å være 0 og alle som er over e^{20} til å være e^{20}

In [34]:

```
lr_pred = np.clip(lr_pred, a_min=0, a_max=20)
```

In [35]:

```
lr_res = pd.DataFrame({'true': lr_y_test, 'pred': lr_pred})
lr_res["diff"] = lr_res["true"] - lr_res["pred"]
lr_res["absDiff"] = (lr_res["true"] - lr_res["pred"]).abs()
lr_res.sort_values(by="absDiff", inplace=True)
```

Fem tilfeller der modellen treffer best

In [36]:

```
lr_res.head()
```

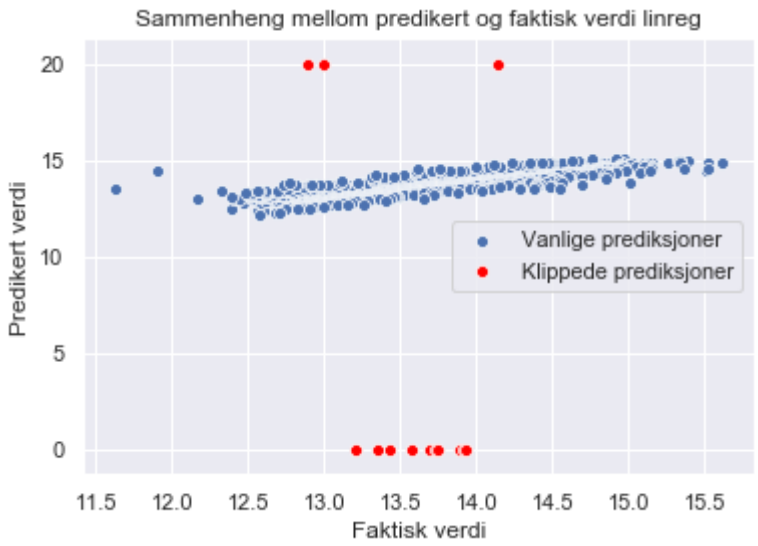
Out[36]:

	true	pred	diff	absDiff
15239	14.1520	14.1520	-0.0000	0.0000
44115	13.4588	13.4589	-0.0000	0.0000
25018	13.8605	13.8604	0.0001	0.0001
24772	13.7332	13.7330	0.0002	0.0002
43781	13.5798	13.5796	0.0002	0.0002

I plottet under kan vi se sammenhengen mellom predikert verdi og faktisk verdi. Helst ville vi sett en diagonal linje. Prediksjonene som er klippet er markert som røde prikker under

In [39]:

```
clipped_data = lr_res[(lr_res["pred"] == 0) | (lr_res["pred"] == 20)]
sns.scatterplot("true", "pred", data=lr_res, label="Vanlige prediksjoner");
sns.scatterplot("true", "pred", data=clipped_data, color="red", label="Klippede prediks
joner")
plt.xlabel("Faktisk verdi"); plt.ylabel("Predikert verdi")
plt.title("Sammenheng mellom predikert og faktisk verdi linreg");
```



Metrikker på testdata

In [40]:

```
lr_metrics = get_metrics(lr_y_test, lr_pred, "Simple LinReg, testdata")
```

```
WALE: 0.18147206196649954
RMSE: 12060558.228787877
MAE: 501171.85294012993
R2: -501.7782663500737
Method: Simple LinReg, testdata
```

Metrikker på treningsdata

Hvis vi sammenligner metrikkene evaluert på testsettet med metrikkene evaluert på treningssettet, kan vi undersøke hvor godt modellen klarer å predikere nye data

In [41]:

```
lr_train_pred = linreg.predict(lr_X_train)
lr_train_pred = np.clip(lr_train_pred, a_min=0, a_max=20)
lr_train_metrics = get_metrics(lr_y_train, lr_train_pred, "Simple LinReg, treningsdata"
)
```

```
WALE: 0.1668797575377578
RMSE: 1085124.2585205953
MAE: 196045.24517332885
R2: -2.2257969809341285
Method: Simple LinReg, treningsdata
```

Vi ser at metrikkene er mye dårligere på testsettet enn på treningssettet. Dette tyder på at modellen er dårlig til å generalisere.

Konklusjon fremgangsmåte 1: Lineær regresjon, med one-hot encoding av kategoriske variabler

Vi ser at med en "naiv" lineær regresjonsmodell som bruker alle featurene, klarer ikke modellen å generalisere godt nok, slik at prediksjon av usette boliger vil være dårlig.

Fremgangsmåte 2: Lineær regresjonsmodell med utvalgte og transformerte variabler basert på random forest-innsikt

Denne fremgangsmåten ble foreslått i konseptbeskrivelsen i konkurransegrunnlaget. Formålet med denne fremgangsmåten er å bruke en random forest-modell for å forstå sammenhengene mellom features i dataen, for så å transformere disse featurene og til slutt bruke de transformerte featurene i en lineær regresjonsmodell. Vi bruker her scikit-learn (https://scikit-learn.org/stable/index.html) sin implementasjon av Random Forest (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html).

Preprossesering

Før dataen kan mates inn i modellen, må gjøre om kategoriske variabler representert som strenger om til heltall

In [42]:

```
columns_to_encode = ["Suburb", "Type", "Method", "Regionname", "CouncilArea"]
rfs_enc = OrdinalEncoder()
rfs_df = df.copy()
rfs_df[columns_to_encode] = rfs_enc.fit_transform(df[columns_to_encode])
rfs_df[columns_to_encode] = rfs_df[columns_to_encode].astype(int)
```

Oppdeling i trening, validering og holdout-testsett

In [43]:

```
rfs_train, rfs_valid, rfs_holdout_test = split_dataset(rfs_df, val_idx, holdout_test_id
x)
rfs_X_train = rfs_train.drop(columns=columns_to_drop)
rfs_y_train = rfs_train["LogPrice"]

rfs_X_test = rfs_holdout_test.drop(columns=columns_to_drop)
rfs_y_test = rfs_holdout_test["LogPrice"]
```

Modellering

Vi bruker sklearn-implementasjonen av Random Forest for å trene opp en modell. I denne eksempelnotebooken utfører vi ikke et såkalt *grid search*- å finne kombinasjonen av hyperparameter som gir best resultat, da det er en meget tidkrevende prosess. Vi utfører heller ikke en såkalt *k-fold* kryssvalidering av samme årsak

In [44]:

```
rfs = RandomForestRegressor(n_estimators=100)
rfs.fit(rfs_X_train, rfs_y_train);
```

Evaluering

In [45]:

```
rfs_pred = rfs.predict(rfs_X_test)
rfs_res = pd.DataFrame({"true": rfs_y_test, "pred": rfs_pred})
rfs_res["absDiff"] = (rfs_res["true"] - rfs_res["pred"]).abs()
rfs_res.sort_values(by="absDiff", inplace=True)
```

De fem tilfellene der modellen predikerer nærmest faktisk salgspris

In [46]:

```
rfs_res.head()
```

Out[46]:

	true	pred	absDiff
13895	13.6423	13.6423	0.0001
51180	13.4075	13.4076	0.0001
18281	13.5008	13.5006	0.0002
18116	13.8205	13.8202	0.0003
13219	13.4045	13.4050	0.0005

De fem tilfellene der modellen predikerer lengst unna faktisk salgspris

In [47]:

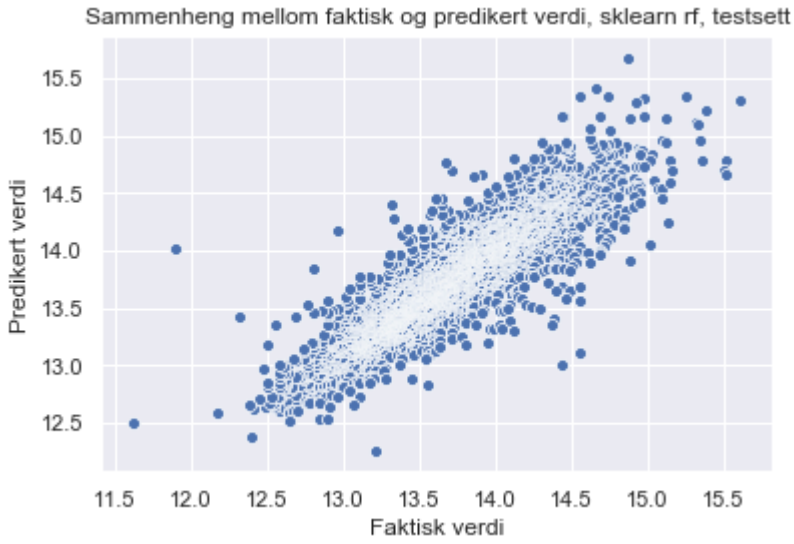
```
rfs_res.tail()
```

Out[47]:

	true	pred	absDiff
22232	12.3239	13.4299	1.1061
22059	12.9598	14.1659	1.2061
51318	14.4307	13.0048	1.4259
20749	14.5527	13.1071	1.4456
20532	11.9050	14.0086	2.1036

In [48]:

```
sns.scatterplot("true", "pred", data=rfs_res); plt.xlabel("Faktisk verdi"); plt.ylabel(
"Predikert verdi")
plt.title("Sammenheng mellom faktisk og predikert verdi, sklearn rf, testsett");
```



Metrikker for Random forest-modell

In [49]:

```
rfs_metrics = get_metrics(rfs_y_test, rfs_pred, "Random Forest Sklearn")
```

```
WALE: 0.15796474836554558
RMSE: 306143.5367391559
MAE: 182608.60984074103
R2: 0.6760396446044945
Method: Random Forest Sklearn
```

Forklaringer

Feature importance

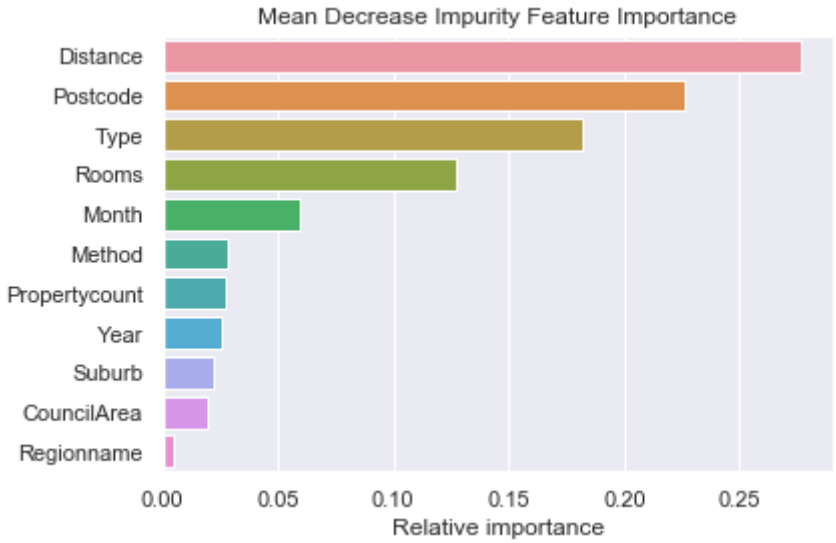
De fleste Random Forest-implementasjonene har muligheter for å hente ut en såkalt "feature importance", som kort forklart forteller hvor "viktig" hver uavhengige variabel / feature var for resultatet

Mean Decrease Impurity / gini importance

Ved å telle antall ganger en feature brukes til å splitte en node i et desisjonstre, vektet med antall eksempler den splitter, vil vi få mean decrease impurity, også kalt *gini importance*

In [50]:

```
rfs_fi = list(zip(rfs_X_test.columns, rfs.feature_importances_))
rfs_fi.sort(key=lambda x: x[1])
rfs_name, rfs_feature_importance = zip(*rfs_fi)
mdi_fi_df = pd.DataFrame({"feature": rfs_name, "fi": rfs_feature_importance}).sort_valu
es(by="fi", ascending=False)
sns.barplot(y="feature", x="fi", data=mdi_fi_df).set(title="Mean Decrease Impurity Feat
ure Importance");
plt.xlabel("Relative importance"); plt.ylabel("");
```



Vi kan i feature importance-plottet over se at det er spesielt fire features som er viktige med tanke på prediksjonen:

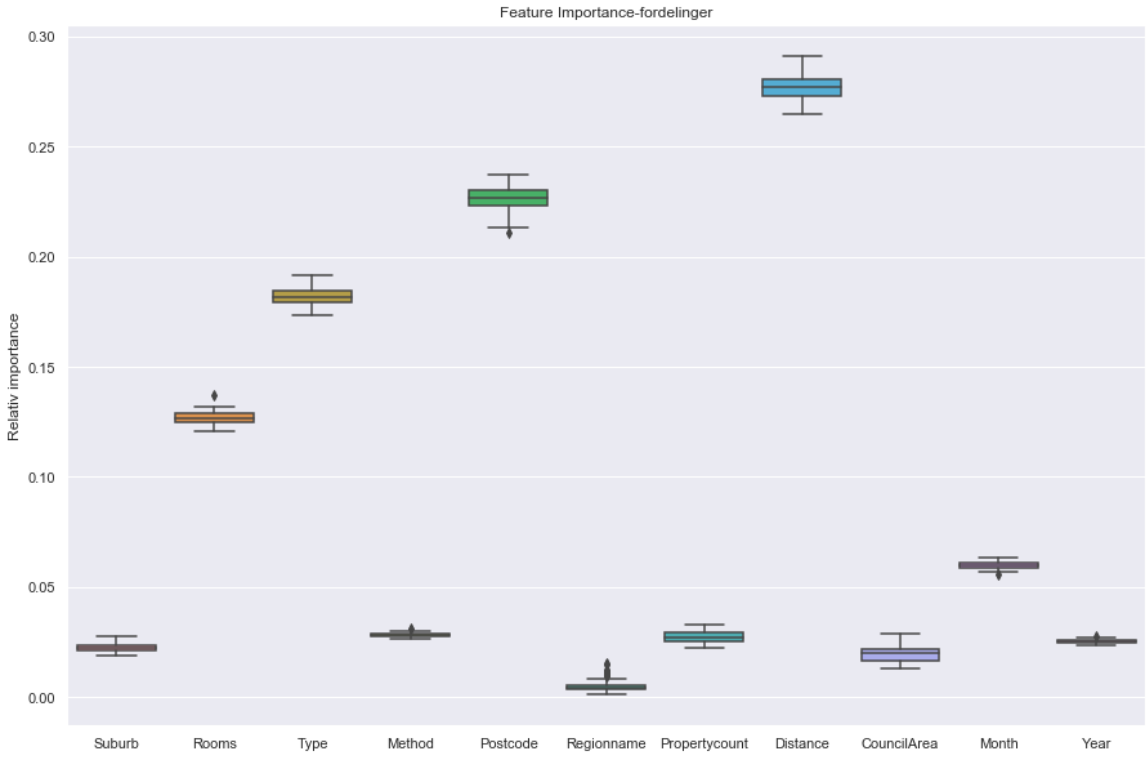
- Avstand til sentrum (Distance)
- Postnummer (Postcode)
- Hustype (Type)
- Antall rom (Rooms)

Feature importance-fordelinger mellom trær i random forest

Feature importance-figuren vist ovenfor viser snittet over alle desisjonstrærne i random-foresten, men sier oss ikke noe om variansen. Nedenfor har vi visualisert fordelingen for hver feature over alle trær i random-forest'en.

In [51]:

```
plt.figure(figsize=(15,10))
all_feat_imp_df = pd.DataFrame(data=[tree.feature_importances_ for tree in rfs], column
s=rfs_X_train.columns)
(sns.boxplot(data=all_feat_imp_df).set(title="Feature Importance-fordelinger", ylabel=
"Relativ importance"));
```



Som vi kan se over, er det for de viktigste featurene lite varians i importance. Dette betyr at selv om vi ikke har gjort en k-fold kryssvalidering, kan vi være ganske sikre på at dette er en generell trend i dataen, og ikke tilfeldigheter funnet under modelleringen av random forest-modellen.

Grafisk trevisualisering med beslutningsstier

Med random forest-modeller kan vi visualisere beslutningsstiene som blir dannet under modelleringen. Av praktiske årsaker har vi begrenset tredybden til 3 i figuren nedenfor. I hver node kan vi se fire linjer med informasjon. Disse beskriver følgende:

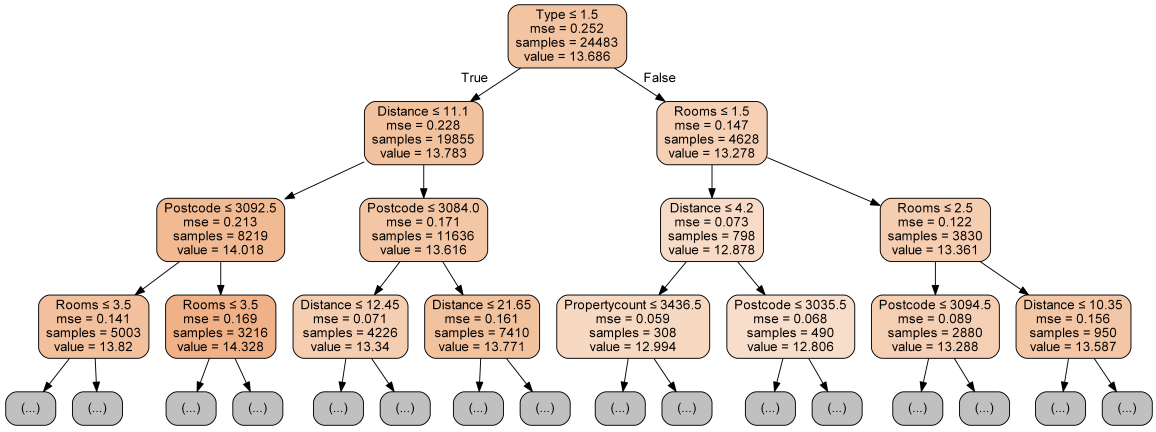
- Første rad: Split-kriteriet for noden
- Andre rad: Mean squared error (mse) i noden
- Tredje rad: Antall eksempler i noden
- Fjerde rad: Gjennomsnittlig salgspris innad i noden

In [52]:

```
depths =  [tree for tree in rfs.estimators_]
tree = depths[1]
export_graphviz(tree, out_file='tree.dot', feature_names=rfs_X_train.columns.tolist(),
filled=True, rounded=True, special_characters=True, max_depth=3)

!dot -Tpng tree.dot -o tree.png -Gdpi=600
Image(filename="tree.png")
```

Out[52]:



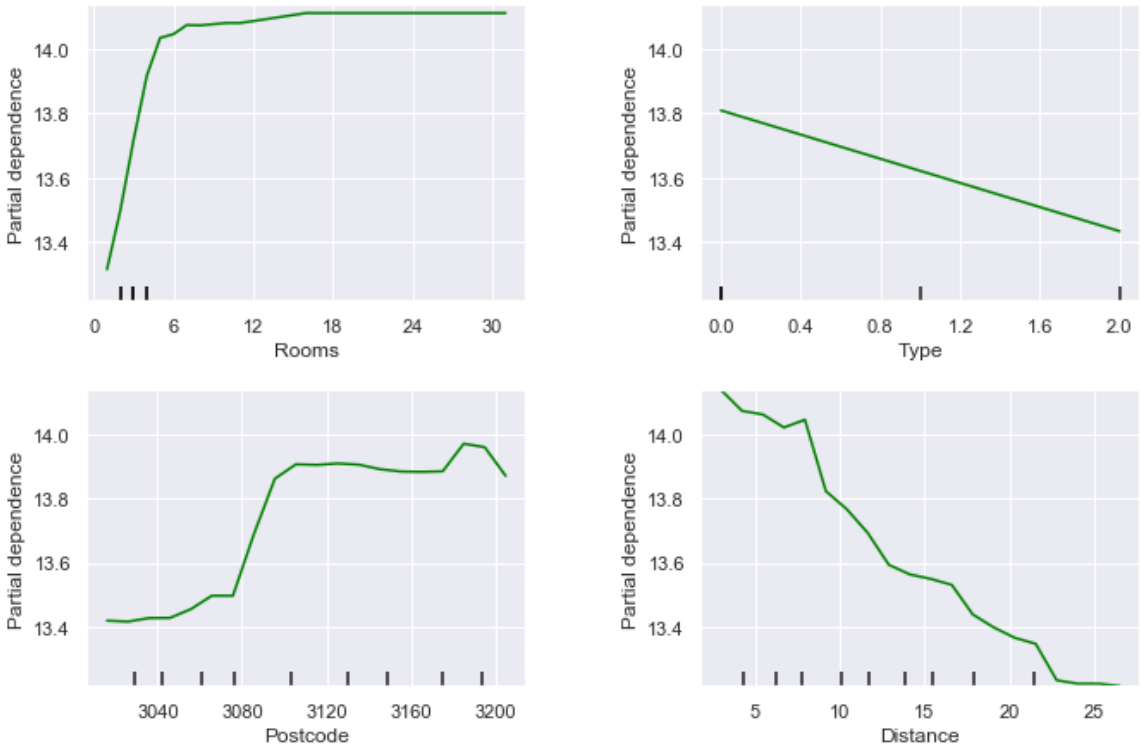
Partial Dependance Plots (PDP)

Plot av denne types viser avhengigheten mellom målvariablen (salgspris) og et sett med target features, samtidig som den ser bort ifra bidrag fra andre features. Figurene under viser PDP-plot for de fire featurene som ble ansett som viktigst fra feature importance

NB: Kjøring av cellen under kan ta noen minutter

In [53]:

```
plot_partial_dependence(rfs, rfs_X_train, features=[1,2,4,7], feature_names=rfs_X_test.
columns.tolist(), grid_resolution=20, n_cols=2)
fig = plt.gcf()
fig.set_size_inches(10,10)
```



Transformasjon av features for lineær regresjon

En måte å fordedre lineærregresjonen, kan være å utføre transformasjoner på featurene

In [54]:

```
transformed_linreg_columns = ["LogPrice", "Rooms", "Postcode", "Distance"]
lrt_df = df[transformed_linreg_columns].copy()
lrt_df["Type"] = rfs_df["Type"]
```

Obervasjoner for feature *Rooms*: Separer boliger der rom > 6 i egen kategori

Det ser ut som salgspris stiger relativt lineært sammen med antall rom mens antall rom er opp til 6. Med flere rom enn dette stiger ikke prisen like mye, og flater ut ved ca 16 rom.

Vi kan sette alle boliger som har lik eller flere rom enn 6 til å være konstant

In [55]:

```
lrt_df.loc[lrt_df["Rooms"] >= 6, "Rooms"] = 6
```

Observasjoner for feature *Postcode*: Grupper Postcode i to kategorier

Det kan virke som hus med postnummer under 3100 har en betydelig lavere salgspris enn hus med postnummer høyere eller lik 3100. Selv om man i utgangspunktet skulle tro at postnummer var en kategorisk feature, virker som det er et mønster i postnummeret og salgsprisen for hus med disse postnummerne. Vi kan dele de inn i to grupper, over eller under postnummer 3100

In [56]:

```
lrt_df["Postcode_ge_3100"] = 1
lrt_df.loc[lrt_df["Postcode"] < 3100, "Postcode_ge_3100"] = 0
```

Observasjoner for feature *Distance*

Vi ser at distanse til sentrum (Distance) er tilnærmet lineær med salgspris, med negativ koeffisient. Vi velger å ikke utføre noen transformasjoner med denne, og lar den være som den er

Observasjoner for feature *Type*

Type, som beskriver hvilken type hus som er solgt, er en kategorisk variabel. Samtidig ser det ut som det er en helt lineær funksjon med negativt fortegn. Vi utfører ingen transformasjoner for denne variablen.

In [57]:

```
rfs_enc.categories_[1]
```

Out[57]:

```
array(['h', 't', 'u'], dtype=object)
```

Hvis vi invers-transformerer de heltalls-enkodede verdiene, finner vi at:

- Verdi = 0 tilsvarer h - som er en av **house, cottage, villa, semi, terrace**
- Verdi = 1 tilsvarer t - som er **townhouse**
- Verdi = 2 tilsvarer u - som er en av **unit, duplex**

Det betyr at hus i kategori H er dyrere en kategori T, som igjen er dyrere enn U

Lineær regresjon med transformerte features

Med våre nye, og forhåpentligvis forbedrede features forsøker vi nå å lage en ny linreg-modell

In [58]:

```
lrt_features = ["Rooms", "Postcode_ge_3100", "Distance", "Type"]
```

In [59]:

```
lrt_train, lrt_val, lrt_holdout_test = split_dataset(lrt_df, val_idx, holdout_test_idx)
lrt_X_train = lrt_train[lrt_features]
lrt_y_train = lrt_train["LogPrice"]

lrt_X_val = lrt_val[lrt_features]
lrt_y_val = lrt_val["LogPrice"]

lrt_X_test = lrt_holdout_test[lrt_features]
lrt_y_test = lrt_holdout_test["LogPrice"]
```

In [60]:

```
linreg_t = LinearRegression()
linreg_t.fit(lrt_X_train, lrt_y_train);
```

In [61]:

```
lrt_pred = linreg_t.predict(lrt_X_test)
lrt_res = pd.DataFrame({"true": lrt_y_test, "pred": lrt_pred})
lrt_res["absDiff"] = (lrt_res["true"] - lrt_res["pred"]).abs()
lrt_res.sort_values(by="absDiff", inplace=True)

lrt_pred = np.clip(lrt_pred, None, 20)
```

Metrikker for lineær regresjon med feature-transformasjon

In [62]:

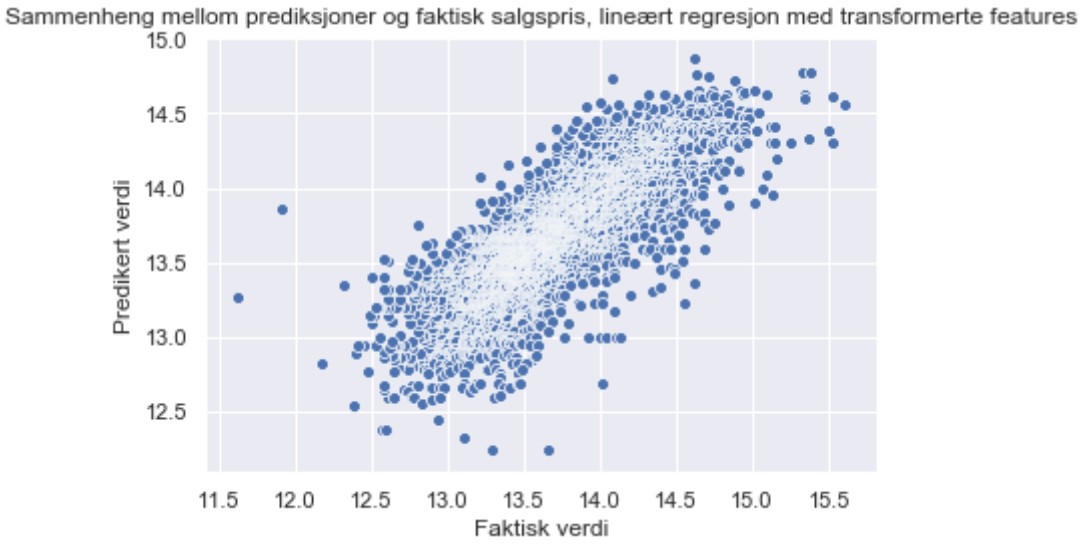
```
lrt_metrics = get_metrics(lrt_y_test, lrt_pred, "linreg Transformed")
```

```
WALE: 0.19814299681454736
RMSE: 352070.42944756284
MAE: 220834.77163038505
R2: 0.5715493790815853
Method: linreg Transformed
```

Sammenheng mellom prediksjoner og faktisk salgspris

In [63]:

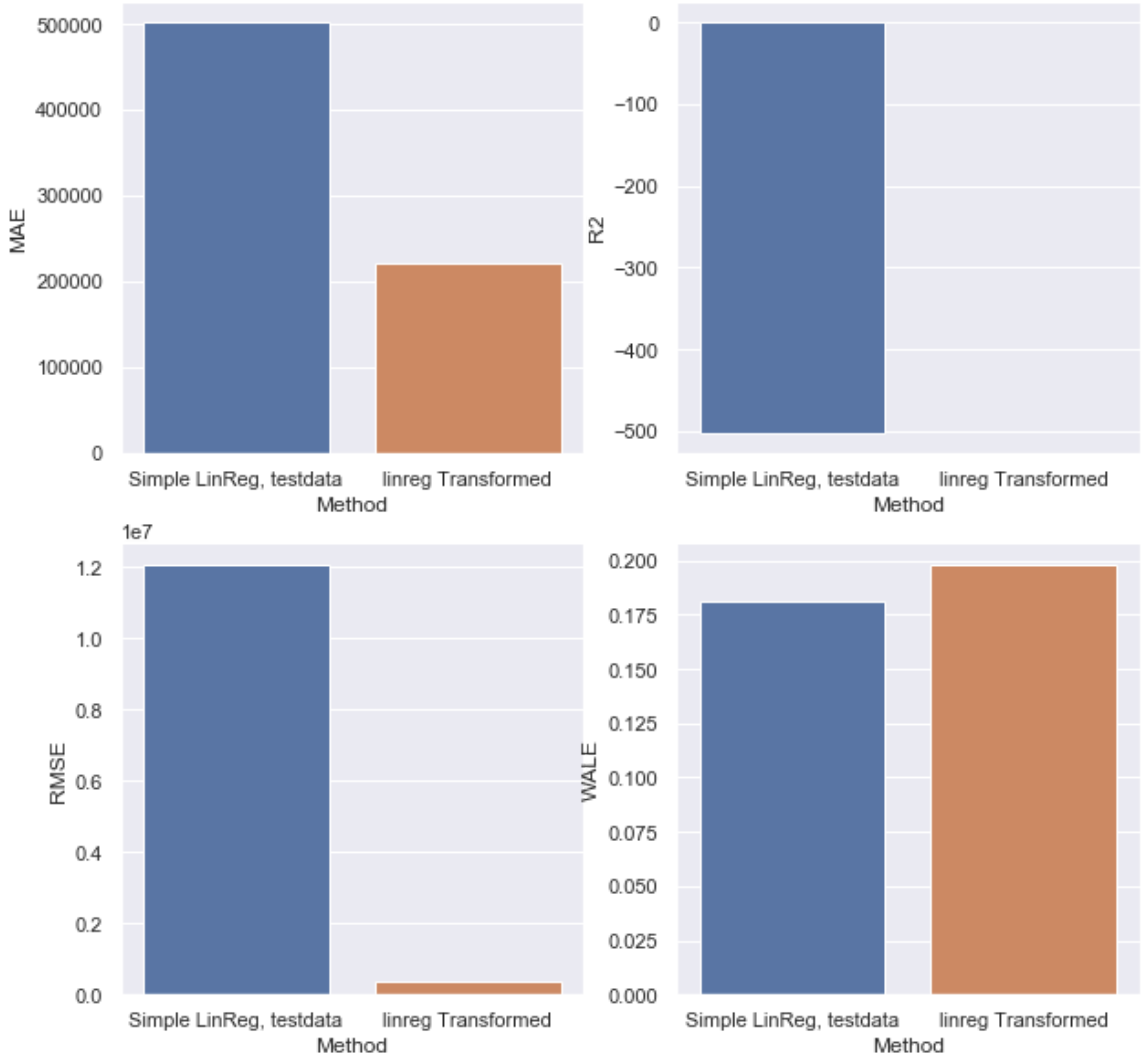
```
sns.scatterplot("true", "pred", data=lrt_res); plt.xlabel("Faktisk verdi"); plt.ylabel(
"Predikert verdi")
plt.title("Sammenheng mellom prediksjoner og faktisk salgspris, lineært regresjon med t
ransformerte features");
```



Sammenligning av lineær regresjon før og etter feature-transformasjoner

In [64]:

```
metrics_trans = pd.DataFrame([lr_metrics, lrt_metrics])
fig, axs = plt.subplots(2,2, figsize=(10,10))
axs = axs.flatten()
for idx, col in enumerate(metrics_trans.drop(columns="Method").columns):
    sns.barplot(y=col, x="Method", data=metrics_trans, ax=axs[idx])
```



Som vi kan se i figurene ovenfor, har vi med liner-regresjonsmodellen med transformerte features fått en mye bedre modell enn den naive varianten der vi bruker alle features. WALE er noe lavere for den opprinnelige modellen, men RMSE og MAE er betydelig lavere for den transformerte modellen. Konklusjonen er denne framgangsmåten, som bruker transformerte features predikerer boligpriser bedre enn en lineær regresjonsmodell som er trent på alle featurene i datasettet. Den beste modellen bruker dessuten kun fire features fra datasettet.

Fremgangsmåte 3: Gradient boosting-modell med økt innsikt og forklarbarhet

CatBoost er en maskinlæringsalgoritme som er utviklet av Yandex. Algoritmen baserer seg på gradient-boosting over desisjonstrær. Mer info kan finnes her (https://github.com/catboost/catboost). Blant fordelene med CatBoost finner vi:

- Meget god ytelse sammenlignet med tilsvarende implementasjoner
- Støtte for både numeriske og kategoriske features
- GPU-, og CPU-støtte
- Enkelt grensesnitt

Preprossesering

For å kunne bruke en catboost-modell, trenger vi ikke å utføre one-hot encoding av featurene. Vi må derimot gjøre om strings til heltall

In [65]:

```
columns_to_encode = ["Suburb", "Type", "Method", "Postcode", "Regionname", "CouncilAre
a"]
rfc_enc = OrdinalEncoder()
rfc_df = df.copy()
rfc_df[columns_to_encode] = rfc_enc.fit_transform(df[columns_to_encode])
rfc_df[columns_to_encode] = rfc_df[columns_to_encode].astype(int)
```

Oppdeling i trening, validering og holdout-testsett

In [66]:

```
rfc_train, rfc_valid, rfc_holdout_test = split_dataset(rfc_df, val_idx, holdout_test_id
x)

rfc_X_train = rfc_train.drop(columns=columns_to_drop)
rfc_y_train = rfc_train["LogPrice"]

rfc_X_valid = rfc_valid.drop(columns=columns_to_drop)
rfc_y_valid = rfc_valid["LogPrice"]

rfc_X_test = rfc_holdout_test.drop(columns=columns_to_drop)
rfc_y_test = rfc_holdout_test["LogPrice"]
```

In [67]:

```
cat_col_idx = np.array([rfc_X_train.columns.get_loc(c) for c in columns_to_encode])
```

Modellering

Catboost har muligheten til å plotte trenings-losset mens vi trener modellen. Dette er et nyttig hjelpemiddel for å overvåke treningsprosessen. Med RMSE som optimeringsmetrikk, ønsker vi at modellen skal ha synkende RMSE jo flere iterasjoner av gradient-boosting algoritmen vi utfører. RMSE vil til slutt gå asymptotisk mot en verdi, og når RMSE ikke synker med økende antall iterasjoner anses treningsprosessen som ferdig. Dette kan observeres i figuren nedenfor

In [68]:

```
rfc = CatBoostRegressor(iterations=500, loss_function="RMSE", task_type="GPU")
```

In [69]:

```
rfc.fit(X=rfc_X_train, y=rfc_y_train, cat_features=cat_col_idx, eval_set=(rfc_X_valid,
rfc_y_valid), silent=True, plot=True);
```

Evaluering

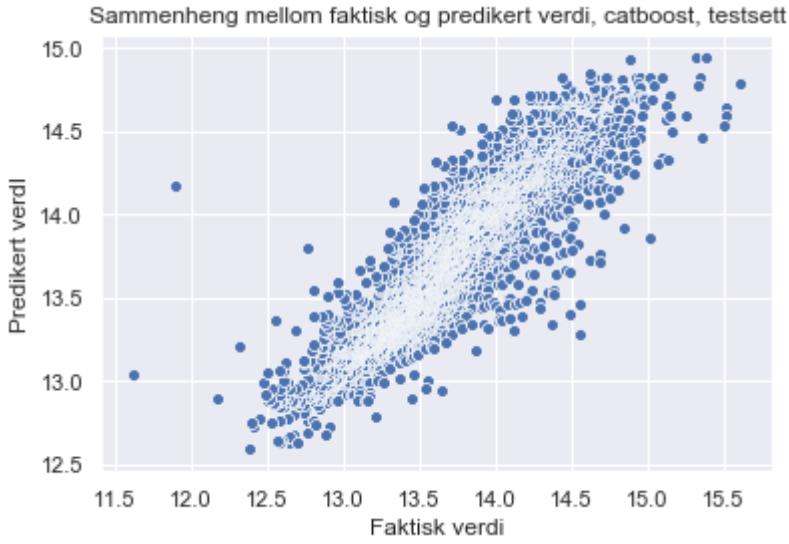
In [70]:

```
rfc_pred = rfc.predict(rfc_X_test)
rfc_res = pd.DataFrame({"true": rfc_y_test, "pred": rfc_pred})
rfc_res["absDiff"] = (rfc_res["true"] - rfc_res["pred"]).abs()
rfc_res.sort_values(by="absDiff", inplace=True)
#rfc_res
```

Vi ser i plottet under at sammenhengen mellom faktisk og predikert verdi ser mer ut som en rett linje. Vi kan observere enkelte tilfeller der prediksjonen har bommet. Det kan virke som modellen ofte undervurderer den faktiske prisen

In [71]:

```
sns.scatterplot("true", "pred", data=rfc_res); plt.xlabel("Faktisk verdi"); plt.ylabel(
"Predikert verdI")
plt.title("Sammenheng mellom faktisk og predikert verdi, catboost, testsett");
```



Metrikker

In [72]:

```
rfc_metrics = get_metrics(rfc_y_test, rfc_pred, "Gradient Boosting, CatBoost")
```

```
WALE: 0.15227122529946738
RMSE: 290076.9224115076
MAE: 172599.49543551833
R2: 0.7091506920768744
Method: Gradient Boosting, CatBoost
```

Forklaringer

Mer komplekse modeller, som for eksempel catboost kan være vanskeligere å forklare. Man kan få ut det som kalles "feature-importance", som sier noe om hvilke features som har mest å si for prediksjonen. Hvis man derimot vil forklare ett eksempel (en rad), er det vanskeligere å få en god forklaring med slike modeller.

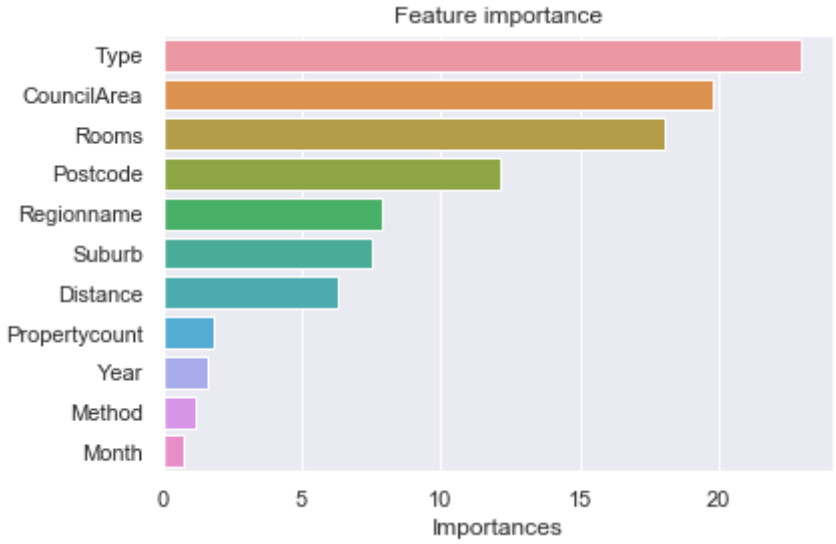
Heldigvis har det vært forsket mye på metoder for å kunne forklare prediksjonene til komplekse maskinlæringsmodeller I denne seksjonen vil vi demonstrere to rammeverk for forklaring av maskinlæringsmodeller:

- SHapley Additive exPlanations (SHAP)
- Local Interpretable Model-agnostic Explanations (LIME)

Feature importance

In [73]:

```
fi = rfc.get_feature_importance(prettified=True)
sns.barplot(y="Feature Index", x="Importances", data=fi);
plt.title("Feature importance"), plt.ylabel("");
```



Individual Conditional Expectation (ICE) plots

Et ICE plot viser en linje per rad/eksempel, som viser hvordan akkurat dette eksemplets prediksjon endrer seg når en feature endrer seg. På figuren nedenfor kan vi se hvordan modellen predikerer at salgsprisen endrer seg, når vi endrer antall rom. Vi kan observere at mesteparten av eksemplene følger den samme formen. Det vil si at prisen alltid øker for boliger med 5-6 rom, og etter dette stagnerer. Mens PDP-plot kun forteller oss gjennomsnittlig forhold mellom feature og prediksjon, forteller ICE oss forholdet mellom feature og prediksjon for alle eksempelene. Vi kan også inkludere et PDP-plot i ICE-plot, og illustreres i figurene nedenfor som en tykkere linje i midten av figurene.

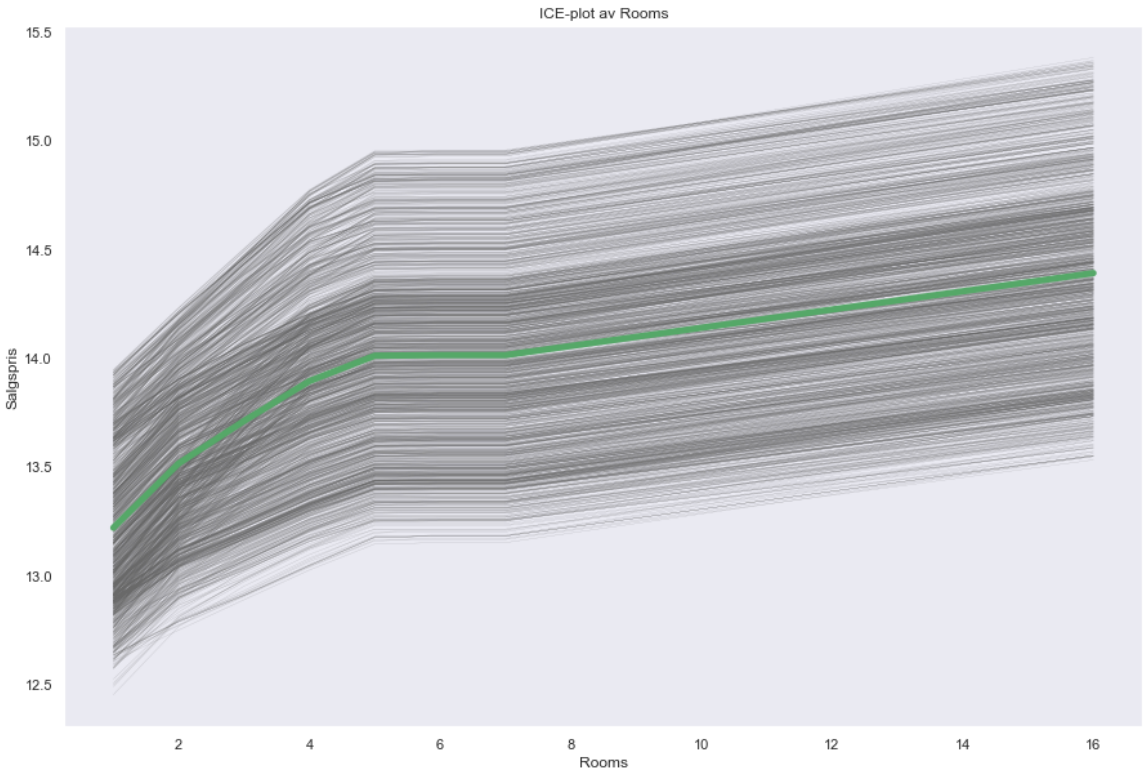
Merk: Vi har kun laget ICE-plot for et subsett av radene i datasettet, da operasjonen med å lage et ICE-plot kan være tidkrevende for datamaskinen

In [74]:

```
rooms_ice_df = ice(data=rfc_X_train.sample(2000), column="Rooms", predict=rfc.predict)
```

In [75]:

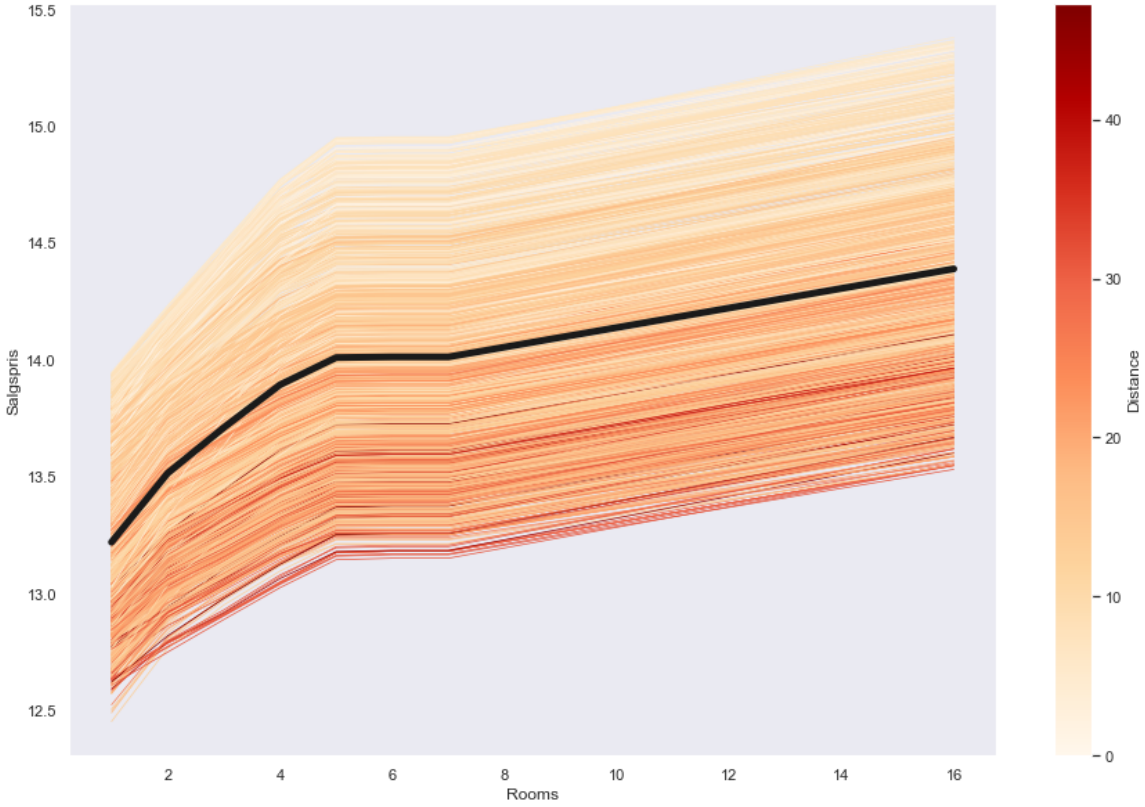
```
ice_plot(rooms_ice_df, c="dimgray", linewidth=0.1, plot_pdp=True, pdp_kwargs={'c':'g',
'linewidth':5})
plt.grid([])
fig = plt.gcf()
fig.set_size_inches(15,10); plt.xlabel("Rooms"); plt.ylabel("Salgspris"); plt.title("IC
E-plot av Rooms");
```



Vi kan med ICE-plots også se legge til en annen feature i figuren, for å se hvordan to features interagerer med hverandre. Nedenfor ser vi hvordan antall rom påvirker salgsprisen. Vi ser også hvor stor avstand fra sentrum disse husene har. Som vi kan se fra figuren under, kan det virke som hus med stor avstand fra sentrum stort sett har normal prisstigning, og har lavest salgspris. For hus med kort avstand til sentrum, som representeres av de øverste linjene i figuren under, kan det se ut som det er en større prisstigning per rom enn andre hus

In [76]:

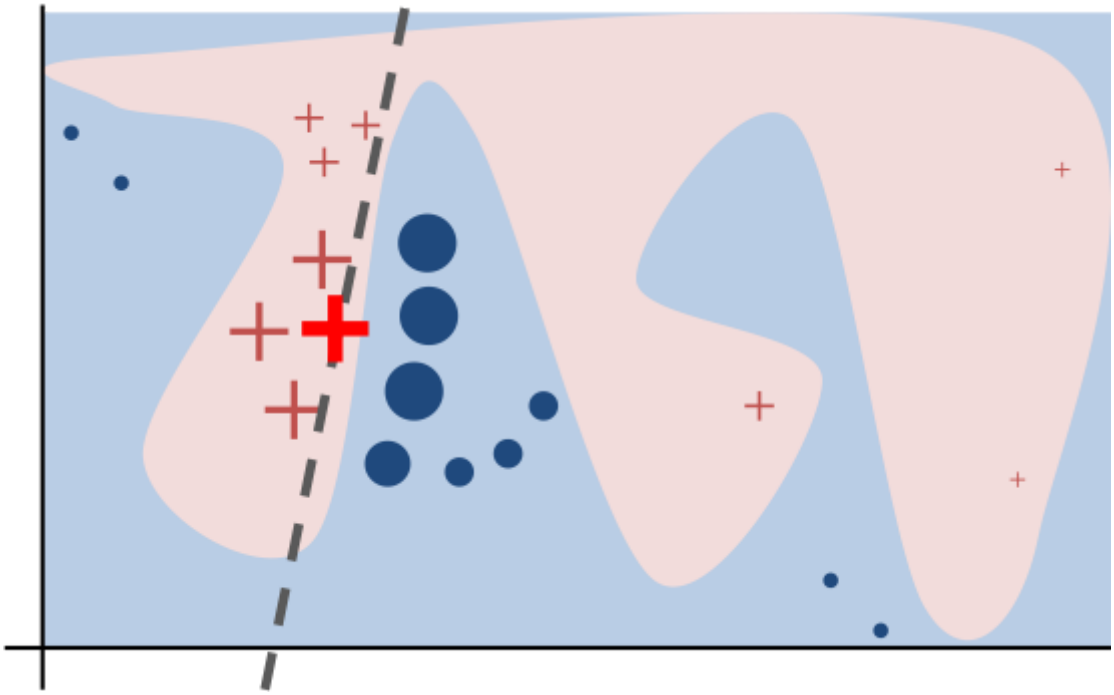
```
cmap2 = plt.get_cmap('OrRd')
ice_plot(rooms_ice_df, linewidth=0.7, color_by='Distance', cmap=cmap2, plot_pdp=True,
         pdp_kwargs={'c': 'k', 'linewidth': 5})
wt_vals = rooms_ice_df.columns.get_level_values('Distance').values
sm = plt.cm.ScalarMappable(cmap=cmap2,norm=plt.Normalize(vmin=wt_vals.min(),vmax=wt_val
s.max()))
sm._A = []
fig = plt.gcf()
fig.set_size_inches(15,10);plt.colorbar(sm, label='Distance');plt.ylabel('Salgspris');p
lt.xlabel('Rooms');plt.grid([])
```



LIME

LIME er et modell-agnostisk rammeverk. Det vil si at det ikke har noe å si hvilken maskinlæringsmodell man bruker. Det LIME gjør, er at den lager en lokal lineær approksimasjon av et eksempel. Selv om den globale modellen kan være kompleks, vil den lokale forklaringen være lettere å approksimere.

Intuisjonen bak LIME vises i bildet under. Modellens desisjonsfunksjon er representert med den blå/rosa bakgrunnen, og er helt klart ikke-lineær. Det store røde krysset er eksemplet som vi ønsker å forklare. Vi sampler så eksempler i nærheten av eksemplet vi ønsker å forklare. Vi lærer så en lineær modell (striplet linje) som approksimerer modellen godt i nærheten av dette eksemplet.



La oss prøve å forklare eksemplet der catboost-modellen predikerer nærmest faktisk salgspris

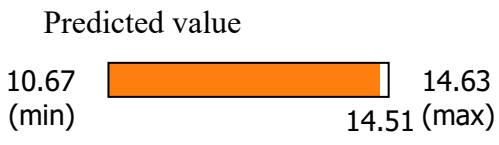
In [77]:

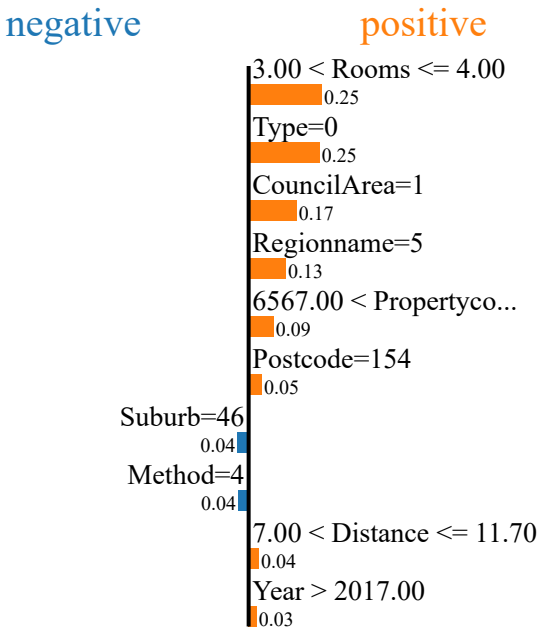
```
rfc_explainer = lime.lime_tabular.LimeTabularExplainer(rfc_X_train.values, categorical_
features=cat_col_idx, feature_names=rfc_X_train.columns, class_names="LogPrice", mode=
"regression")
```

In [78]:

```
idx = 0
best_pred_index = rfc_res.iloc[idx].name
rfc_best_exp = rfc_explainer.explain_instance(rfc_X_test.loc[best_pred_index], rfc.pred
ict)
print(rfc_res.iloc[idx])
rfc_best_exp.show_in_notebook()
```

```
true      14.5087
pred      14.5086
absDiff    0.0000
Name: 27827, dtype: float64
```

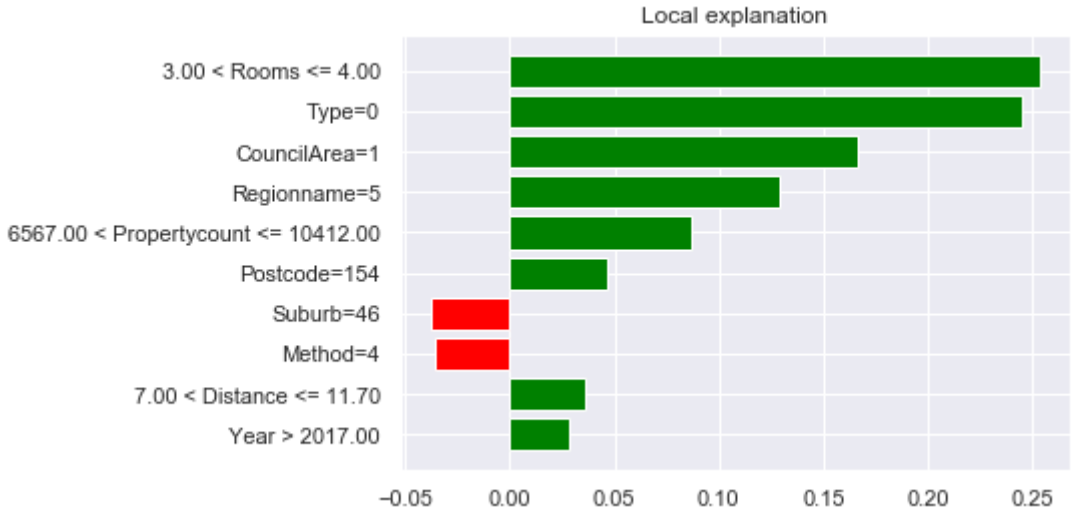




Feature	Value
Rooms	4.00
Type=0	True
CouncilArea=1	True
Regionname=5	True
Propertycount	6938.00
Postcode=154	True
Suburb=46	True
Method=4	True
Distance	10.30

In [79]:

```
rfc_best_exp.as_pyplot_figure();
```



I figurene ovenfor ser vi en forklaring på hvorfor modellen predikerte slik den gjorde. Måten å tolke figurene er å se på stolpene som går enten til venstre eller høyre, som bidragene fra de individuelle featurene for å trekke prisen opp eller ned. nullpunktet er gjenomsnittet av prediksjoner

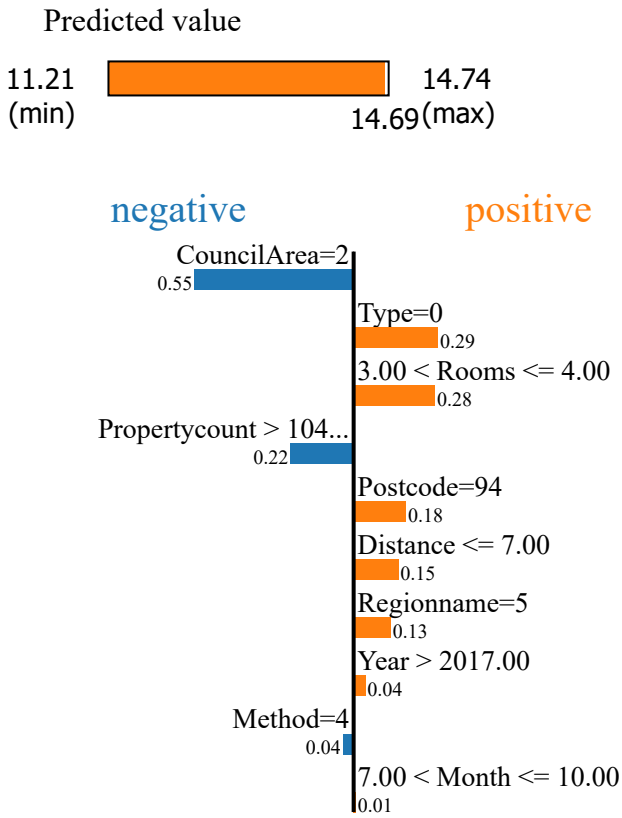
LIME submodular pick

Med LIME submodular pick forsøker vi å finne et representativt utvalg av eksempler som forklarer modellen globalt

Kjøring av cellen under kan ta noe tid

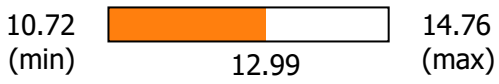
In [80]:

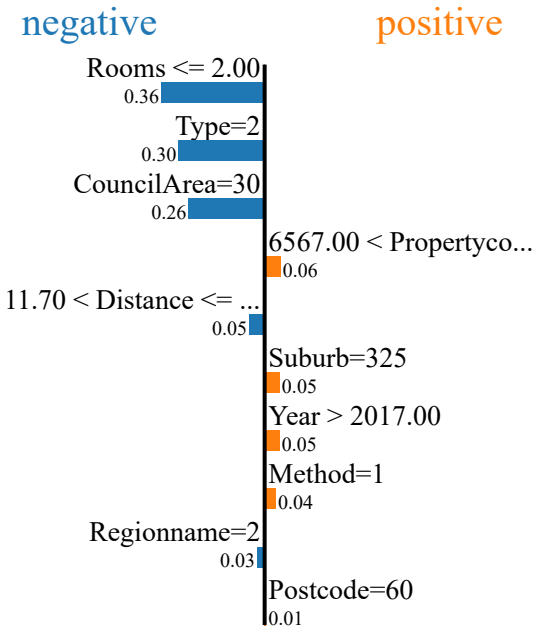
```
sp_obj = submodular_pick.SubmodularPick(rfc_explainer, rfc_X_test.values, rfc.predict)
[exp.show_in_notebook() for exp in sp_obj.sp_explanations];
```



Feature	Value
CouncilArea=2	True
Type=0	True
Rooms	4.00
Propertycount	11308.00
Postcode=94	True
Distance	5.30
Regionname=5	True
Year	2018.00
Method=4	True

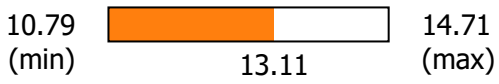
Predicted value

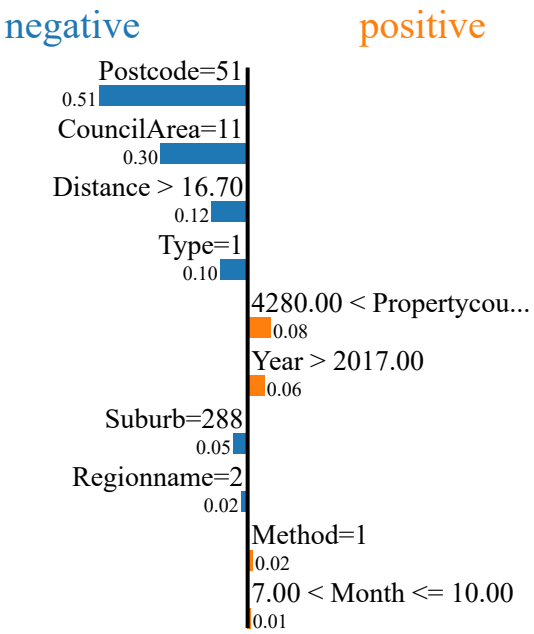




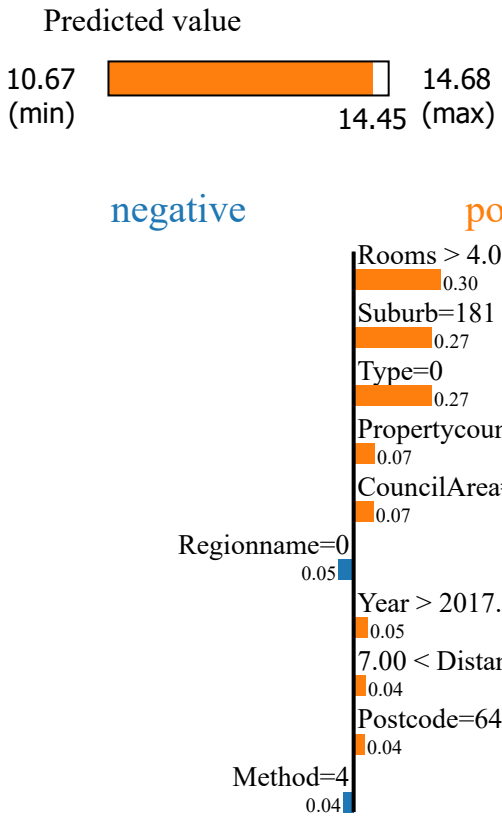
Feature	Value
Rooms	2.00
Type=2	True
CouncilArea=30	True
Propertycount	7955.00
Distance	15.30
Suburb=325	True
Year	2018.00
Method=1	True
Regionname=2	True

Predicted value



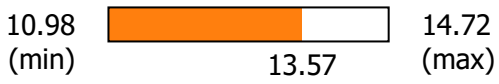


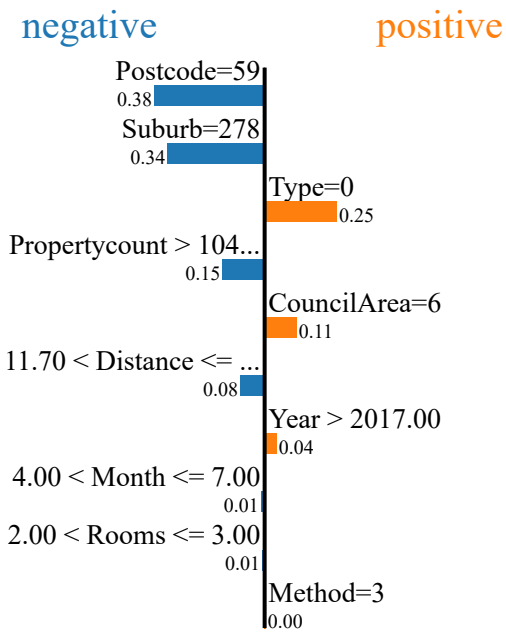
Feature	Value
Postcode=51	True
CouncilArea=11	True
Distance	20.60
Type=1	True
Propertycount	5833.00
Year	2018.00
Suburb=288	True
Regionname=2	True
Method=1	True



Feature	Value
Rooms	5.00
Suburb=181	True
Type=0	True
Propertycount	1554.00
CouncilArea=0	True
Regionname=0	True
Year	2018.00
Distance	7.80
Postcode=64	True

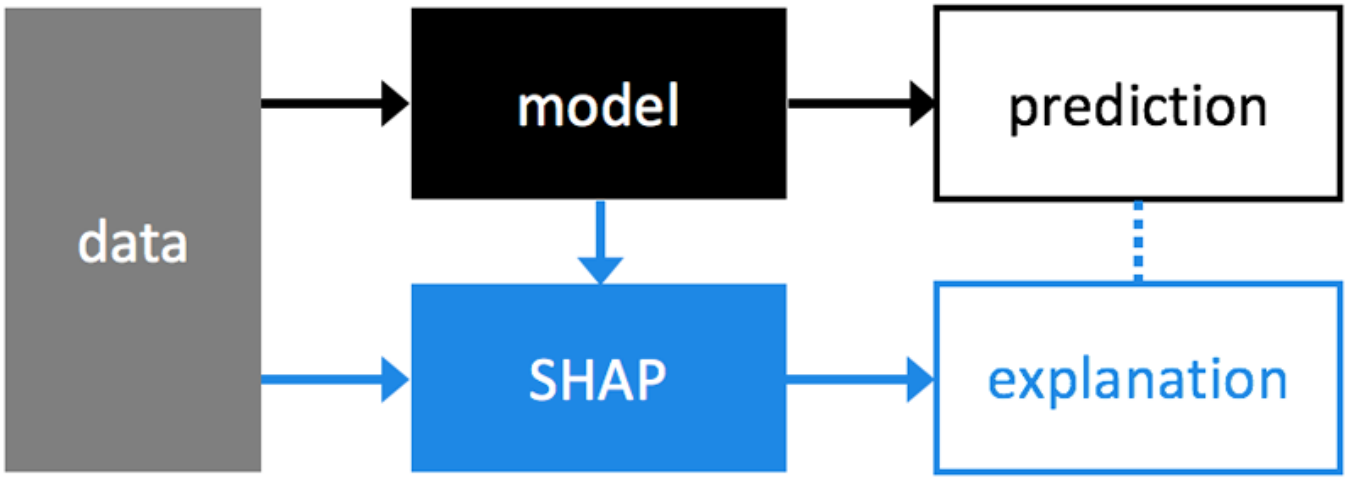
Predicted value





Feature	Value
Postcode=59	True
Suburb=278	True
Type=0	True
Propertycount	21650.00
CouncilArea=6	True
Distance	12.00
Year	2018.00
Month	7.00
Rooms	3.00

Forklaringer med SHAP



SHapley Additive exPlanations (SHAP) (https://github.com/slundberg/shap) er et rammeverk som har som mål å kunne forklare prediksjonene fra alle maskinlæingsmodeller, uavhengig av algoritme. SHAP bruker spillteori med lokale forklaringer for å kunne forklare prediksjonene

In [81]:

```
explainer = shap.TreeExplainer(rfc)
```

In [82]:

```
shap_values = explainer.shap_values(Pool(rfc_X_train, rfc_y_train, cat_features=cat_col
_idx))
```

Force plots

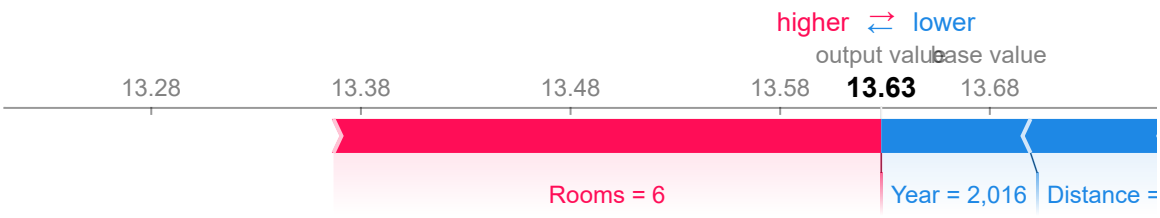
Force plots kan vise oss effekten hver feature har på modellens prediksjon. Features anses som å dytte prediksjonen enten høyere eller lavere bort fra base-verdien (gjennomsnittlig output)

In [83]:

```
print(f"True value: {rfc_pred[0]}")
shap.force_plot(explainer.expected_value, shap_values[0,:], rfc_X_train.iloc[0])
```

```
True value: 14.010278327354552
```

Out[83]:

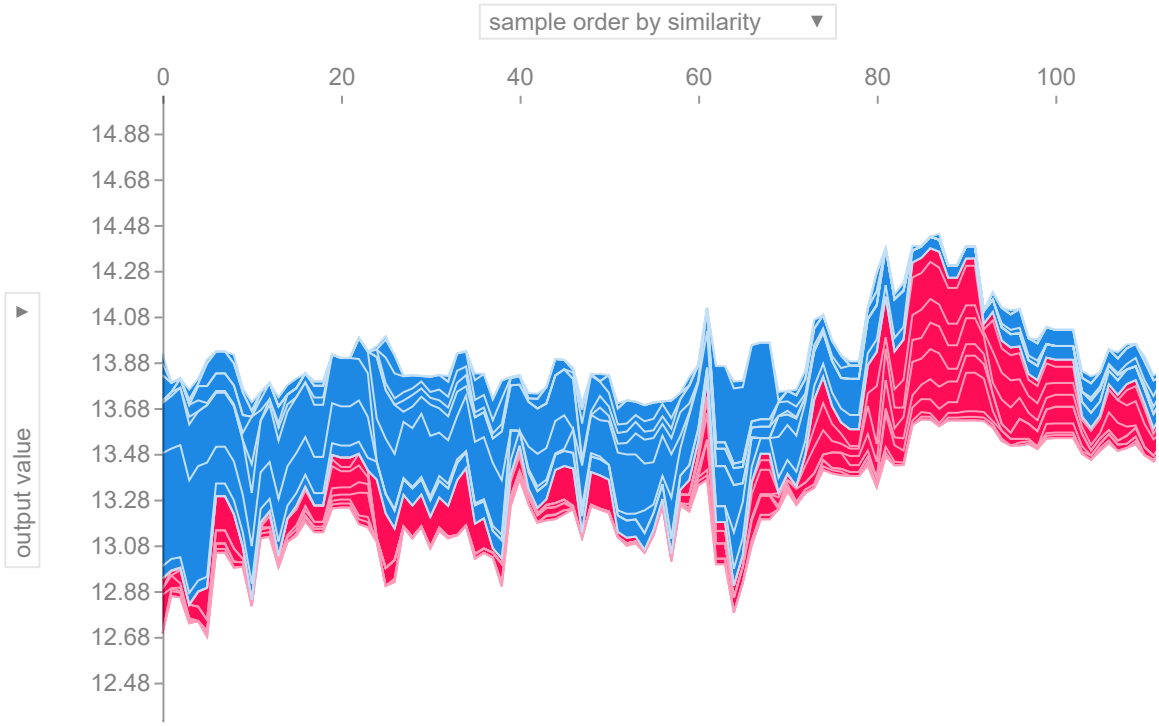


Vi kan også se forklaringen for alle eksemplene i datasettet vårt, som vist under

In [84]:

```
shap.force_plot(explainer.expected_value, shap_values[:200], rfc_X_train.iloc[:200,:])
```

Out[84]:

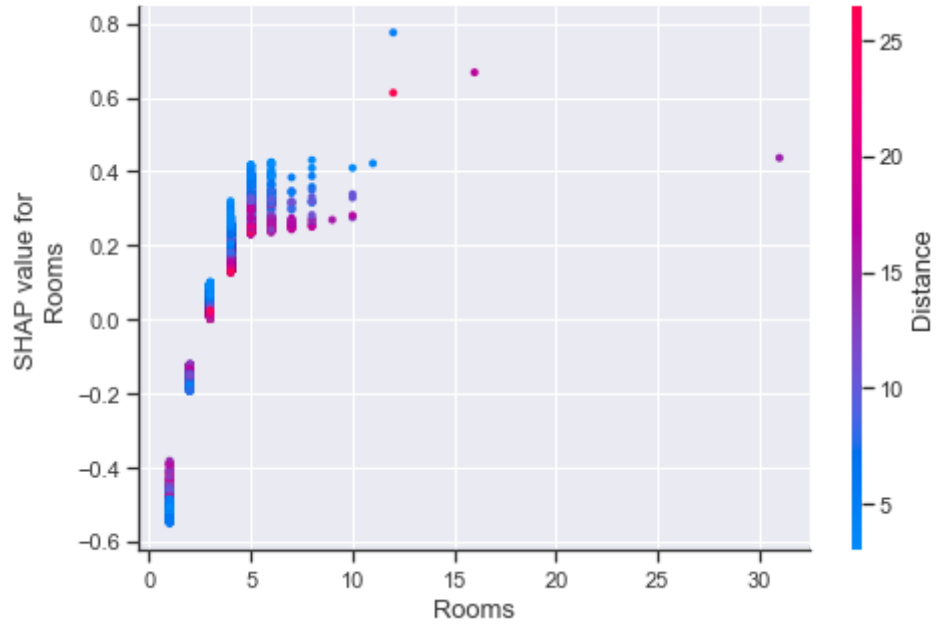


Dependance plots i SHAP

SHAP lager PDP-plots, der den featurene som har høyest joint dependancy blir visualisert med farge

In [85]:

```
shap.dependence_plot("Rooms", shap_values, rfc_X_train)
```



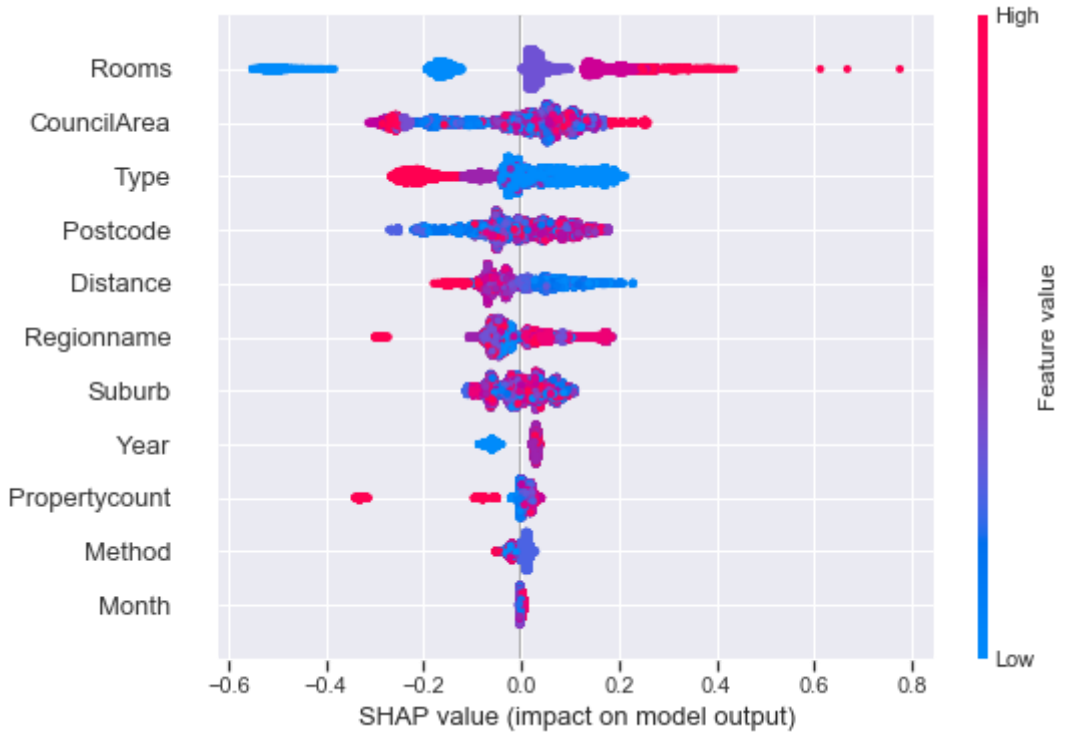
I plottet over kan vi se noen interessante observasjoner. Antall rom øker prisen frem til rundt 6 rom, som er observert tidligere. Men vi ser også at for hus med få rom bidrar kort distanse til sentrum til lavere pris, mens for hus med høyere antall rom ser vi det motsatte; at hus med lang avstand til sentrum er billigere enn de som ligger nærmere sentrum

Summary plot i SHAP

SHAP kan brukes til å oppsummere hvilke features med hvilke verdier som bidrar mest til å påvirke prediksjonen

In [86]:

```
shap.summary_plot(shap_values, rfc_X_train)
```



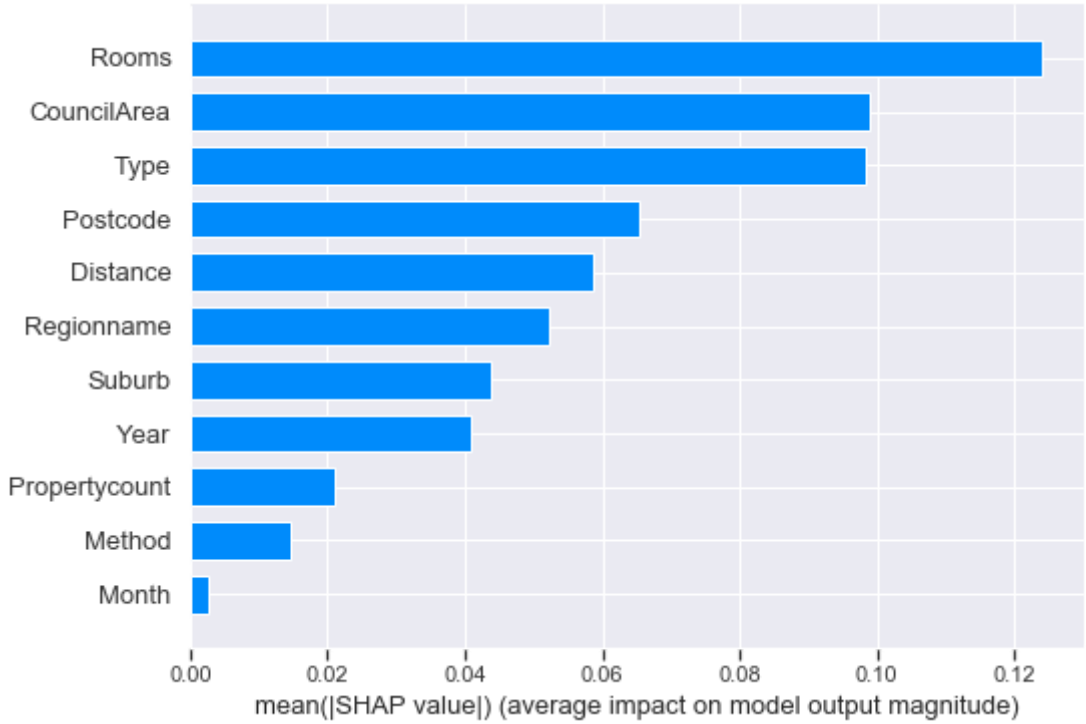
I figuren over ser vi blant annet:

- Antall rom har mye og si, og flere rom drar prisen høyere
- Type bolig har mye å si for prisen
- Boliger som ligger nærmere sentrum er stort sett dyrere enn hus som ligger lengre bort fra sentrum

Vi kan også ta gjennomsnittlig absoluttverdi av shap-verdiene for hver feature for å få et barplot over de viktigste featurene for prediksjonen som ble gjort

In [87]:

```
shap.summary_plot(shap_values, rfc_X_train, plot_type="bar")
```



Sammenligning av fremgangsmåter

Sammenligning av evalueringsmetrikker

In [88]:

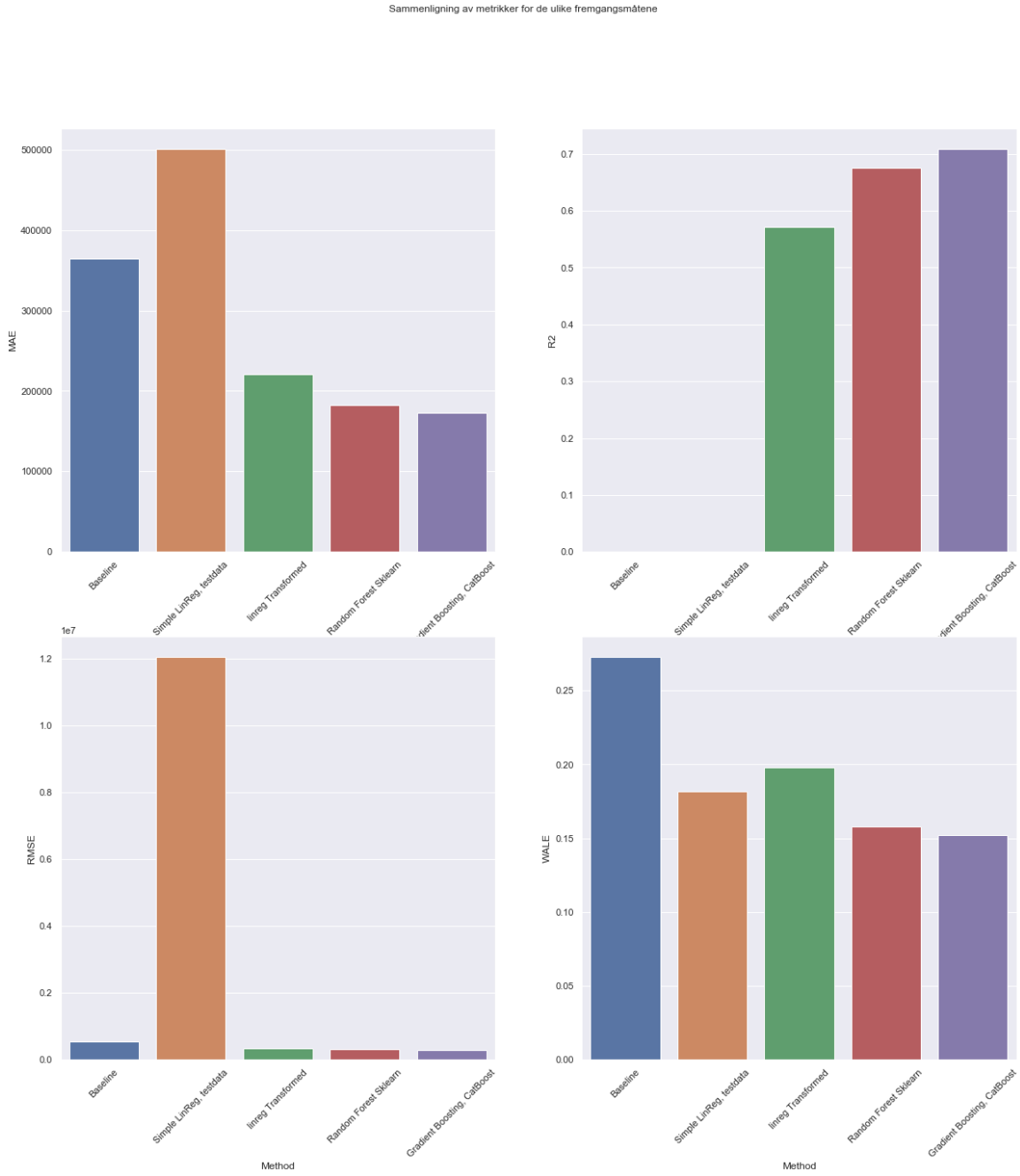
```
metrics_df = pd.DataFrame([dummy_metrics, lr_metrics, lrt_metrics, rfs_metrics, rfc_met
rics])
```

In [89]:

```
metrics_df["R2"] = metrics_df["R2"].apply(lambda x: np.clip(x, 0, None))
```

In [90]:

```
fig, axs = plt.subplots(2,2, figsize=(20,20))
axs = axs.flatten()
for idx, col in enumerate(metrics_df.drop(columns="Method").columns):
    axs[idx].tick_params(axis='x', rotation=45)
    sns.barplot(y=col, x="Method", data=metrics_df, ax=axs[idx])
fig.suptitle("Sammenligning av metrikker for de ulike fremgangsmåtene");
```



Tolkning

Oppfriskning på metrikker

- **Lavere er bedre**: MAE, RMSE, WALE
- **Høyere er bedre**: R2-score

Med tanke på det kjente kompromisset mellom modellens ytelse og forklarbarhet, ser vi at selv komplekse modeller kan bli forklarbare, samtidig som modellytelsen øker. Av alle maskinlæringsmodellene som er gjennomgått i denne notebooken, ser vi at CatBoost har best ytelse av alle modellene. Vi har også sett at ved hjelp av rammeverk som SHAP og LIME, kan selv komplekse modellers prediksjoner gjøres forklarbare.

Man trenger nødvendigvis ikke å ofre modellpresisjon for å utvikle modeller som kan forklares og forstås intuitivt av mennesker.

Sammenligning av residualer

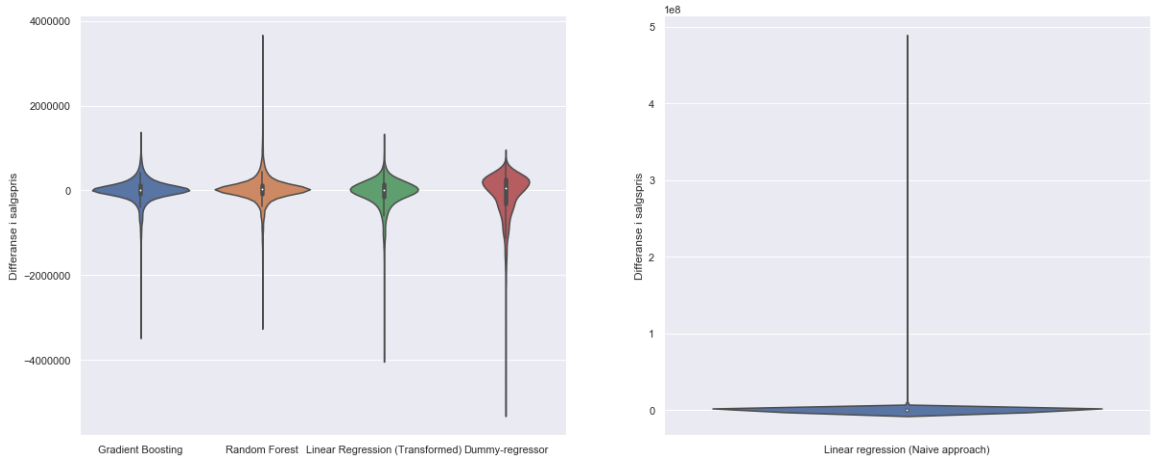
Med metrikker som RMSE og MAE får vi kun ut gjennomsnittsavvik for residualene. Vi kan også visualisere alle residualene som violinplots (https://seaborn.pydata.org/generated/seaborn.violinplot.html). Vi kan da bedre så på variansen av residualene per modell, som vist i figuren under

In [91]:

```
def get_diffprice_lin(true, pred):
    return np.exp(pred) - np.exp(true)
diff_rfc = get_diffprice_lin(rfc_y_test, rfc_pred)
diff_dummy = get_diffprice_lin(dummy_test["LogPrice"], dummy_pred)
diff_rfs = get_diffprice_lin(rfs_y_test, rfs_pred)
diff_lr = get_diffprice_lin(lr_y_test, lr_pred)
diff_lrt = get_diffprice_lin(lrt_y_test, lrt_pred)
diff_df = pd.DataFrame({"Gradient Boosting": diff_rfc, "Random Forest": diff_rfs, "Line
ar Regression (Transformed)": diff_lrt, "Dummy-regressor": diff_dummy})
lr_df = pd.DataFrame({"Linear regression (Naive approach)": diff_lr})
```

In [92]:

```
fig, axs = plt.subplots(1,2, figsize=(20,8))
sns.violinplot(data=diff_df, ax=axs[0]).set_ylabel("Differanse i salgspris")
sns.violinplot(data=lr_df, orient="v", ax=axs[1]).set_ylabel("Differanse i salgspris");
```



Vi kan i figuren over se fordelingen for residualene av de ulike modellene. Vi ser at de fleste modellene har flest verdier rundt 0, men at de i noen tilfeller predikerer enten alt for høye verdier, eller alt for lave verdier.

Kvalitetsmetrikker for de ulike modellene

In [93]:

```
def is_within(true, pred, limit=20):
    true = np.exp(true)
    pred = np.exp(pred)
    mape = np.abs(pred - true) / true * 100
    denom = mape.shape[0]
    nom = mape[mape <= limit].shape[0]
    return nom / denom * 100
```

In [94]:

```
limit = 20
rfc_q = is_within(rfc_y_test, rfc_pred, limit)
dummy_q = is_within(dummy_test["LogPrice"], dummy_pred, limit)
lr_q = is_within(lr_y_test, lr_pred, limit)
lrt_q = is_within(lrt_y_test, lrt_pred, limit)
rfs_q = is_within(rfs_y_test, rfs_pred, limit)
```

In [95]:

```
for name, pct in zip(["Baseline", "Linreg (Naive)", "Linreg (Transformed)", "Random For
est", "Gradient Boosting"], [dummy_q, lr_q, lrt_q, rfs_q, rfc_q]):
    print(f"For modell '{name}' ligger {pct:.0f}% av estimert markedverdi innenfor +/-
{limit}% av observert verdi")
```

```
For modell 'Baseline' ligger 31% av estimert markedverdi innenfor +/- 20%
av observert verdi
For modell 'Linreg (Naive)' ligger 58% av estimert markedverdi innenfor +/
- 20% av observert verdi
For modell 'Linreg (Transformed)' ligger 55% av estimert markedverdi innen
for +/- 20% av observert verdi
For modell 'Random Forest' ligger 68% av estimert markedverdi innenfor +/-
20% av observert verdi
For modell 'Gradient Boosting' ligger 70% av estimert markedverdi innenfor
+/- 20% av observert verdi
```

Med tanke på kvalitetsmetrikken, gjør modellene vi har vist i denne notebooken det meget bra, til tross for begrensede datakilder og begrenset tid brukt på å finjustere modellene. Vi har i denne eksmpel-notebooken heller ikke knyttet inn andre datakilder, slik det er beskrevet i konseptbeskrivelsen i konkurransegrunnlaget. Den beste modellen bommer fremdeles på 30 % av boligene. Dette skyldes mest sannsynlig at prisen på disse bestemmes av eksogene variabler som ikke er kjent for modellen. Det kan være spesifikke lokale forhold som ikke fanges opp, som for eksempel vakker natur, fasiliteter i nærheten. Det kan også skyldes forskjeller i ytre/indre standard for boligen, f.eks nyopppusset bad, parkett vs. linoleum, velholdte ytre vegger og tak eller stelt hage vs. vilniss.