tdonato3 Add files via upload History

1 contributor

1851 lines (1851 sloc) | 51.4 KB

Import libraries

In [ ]:

```
import os
import zipfile
import numpy as np
import pandas as pd
import seaborn as sns
import lightgbm as lgbm
import statsmodels.api as sm
import statsmodels.formula.api as sms
import matplotlib.pyplot as plt

from patsy import dmatrices
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
from sklearn.preprocessing import PolynomialFeatures
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import (
    mean_squared_error,
    r2_score,
    mean_absolute_error,
    mean_absolute_percentage_error,
)
from sklearn.model_selection import (
    RandomizedSearchCV,
    train_test_split,
)
from statsmodels.tools.tools import add_constant
```

Instructions about how to load the datasets

A total of 40 datasets, comprising 971,520 entries and spanning 56 columns, were obtained from the subsequent webpages: (a) Primary dataset: The dataset titled Real Estate listings in the US was sourced from the kaggle.com* webpage, collated from https://www.realtor.com/, a real estate listing website. (b) Complementary dataset: To augment the primary dataset, 38 recommended datasets were retrieved from the https://www.redfin.com/ webpage. Redfin, a real estate brokerage, provides access to timely and trustworthy housing market data. (3) Demographic dataset: To explore the population demographics of housing areas, an additional dataset was obtained from the Azure Open Datasets**.

- https://www.kaggle.com/datasets/ahmedshahriarsakib/usa-real-estate-dataset/data
- https://learn.microsoft.com/en-us/azure/open-datasets/dataset-us-population-zip?tabs=azureml-opendatasets

These datasets were stored in Team-20 GitHub project repository under the folder Data.

- https://github.gatech.edu/MGT-6203-Fall-2023-Canvas/Team-20

The next steps need to be followed to load the data.

1. Clone Team-20 GitHub project in your machine.
2. Unzip the .zip file "real_estate -redfin.zip".
3. Update the file paths below according to where the repository was cloned.

Real estate dataset

In [ ]:

```
with zipfile.ZipFile(".../Repositories/Team-20/Data/real_estate -redfin.zip"
    zip_ref.extractall(".../Repositories/Team-20/Data")

df1 = pd.read_csv(".../Repositories/Team-20/Data/real_estate -redfin.csv")
df1.head(2)
```

In [ ]:

```
print(df1.dtypes, df1.shape)
```

Demographic dataset

In [ ]:

```
df2 = pd.read_csv(".../Repositories/Team-20/Data/Demographic.csv", encoding=
df2.head(2)
```

In [ ]:

```
print(df2.dtypes,df2.shape)
```

Homogenize ZIP code information between both datasets

In [ ]:

```
df1["zip"] = df1["zip"].str.replace('[^\dA-Za-z]', '').astype(float)
df2["zip"] = df2["zip"].astype(float)
```

Concatenate dataframes

In [ ]:

```
df = pd.merge(df1, df2, how="left", on="zip")
```

Convert states names into their corresponding codes

In [ ]:

```
df['state'] = df['state'].replace(
    {
        'Virgin Islands':'VI',
        'South Carolina':'SC',
        'Tennessee':'TN',
        'Virginia':'VA',
        'Wyoming':'WY',
```

```
        'Georgia':'GA',
        'West Virginia':'WV',
    }
)
df.replace('South Carolina', 'SC', inplace=True)
```

Rename variables

In [ ]:

```
df = df[[
    "state",
    "bed",
    "bath",
    "acre_lot",
    "city_x",
    "house_size",
    "price",
    "status",
    "SALE TYPE",
    "PROPERTY TYPE",
    "YEAR BUILT",
    "DAYS ON MARKET",
    "$/SQUARE FEET",
    "HOA/MONTH",
    "LATITUDE",
    "LONGITUDE",
    "population",
    "density",
    "county_fips",
    "imprecise",
    "military",
]].rename(
    columns={
        "city_x": "city",
        "SALE TYPE": "sale_type",
        "PROPERTY TYPE": "property_type",
        "YEAR BUILT": "year_built",
        "DAYS ON MARKET": "days_on_market",
        "$/SQUARE FEET": "square_feet",
        "HOA/MONTH": "hoa_month",
        "LATITUDE": "latitude",
        "LONGITUDE": "longitude",
    }
)
```

Dataset variables quantitative information before outlier removal

In [ ]:

```
df.describe()
```

In [ ]:

```
df.dtypes
```

Dataset variables

In [ ]:

```
df.columns
```

Prepare and filter columns

In [ ]:

```
df = df[[
    'state', 'bed', 'bath', 'acre_lot', 'city', 'house_size', 'price',
    'status', 'sale_type', 'property_type', 'year_built', 'days_on_market',
    'square_feet', 'hoa_month', 'latitude', 'longitude', 'population',
    'density', 'county_fips', 'imprecise', 'military'
]]
df.dtypes
```

In [ ]:

```
df[['bed', 'bath', 'acre_lot', 'city', 'house_size', 'price','days_on_market
```

Convert Object columns to Factor

In [ ]:

```
df[[
    "status", "state", "city", "sale_type", "property_type", "imprecise", "m
]] = df[[
    "status", "state", "city", "sale_type", "property_type", "imprecise", "m
]].replace(np.nan,"?").astype("category")
```

Plot boxplots of variables before outliers removal

In [ ]:

```
lst = list(df[['bed','bath','acre_lot','house_size','price','days_on_market'

df = df.reset_index(drop=True)
for column in lst:
    plt.figure(figsize=(10, 4))
    plt.subplot(131)
    sns.boxplot(df[column])
    plt.show()
```

Delete outliers

In [ ]:

```
df3 = df.copy()
zscores_bed = (df3["bed"]-df3["bed"].mean())/df3["bed"].std()
zscores_bath = (df3["bath"]-df3["bath"].mean())/df3["bath"].std()
zscores_acre_lot = (df3["acre_lot"]-df3["acre_lot"].mean())/df3["acre_lot"].
zscores_house_size = (df3["house_size"]-df3["house_size"].mean())/df3["house

zscores_year_built = (df3["year_built"]-df3["year_built"].mean())/df3["year_
zscores_days_on_market = (df3["days_on_market"]-df3["days_on_market"].mean()
zscores_square_feet = (df3["square_feet"]-df3["square_feet"].mean())/df3["sq
zscores_hoa = (df3["hoa_month"]-df3["hoa_month"].mean())/df3["hoa_month"].st
zscores_lat = (df3["latitude"]-df3["latitude"].mean())/df3["latitude"].std()
zscores_lng = (df3["longitude"]-df3["longitude"].mean())/df3["longitude"].st
```

```
df3["not_outlier_bed"] = zscores_bed.abs() < 3
df3["not_outlier_bath"] = zscores_bath.abs() < 3
df3["not_outlier_acre_lot"] = zscores_acre_lot.abs() <3
df3["not_outlier_house_size"] = zscores_house_size.abs() < 3
df3["not_outlier_year_built"] = zscores_year_built.abs() < 3
df3["not_outlier_days_on_market"] = zscores_days_on_market.abs() < 3
df3["not_outlier_square_feet"] = zscores_square_feet.abs() < 3
df3["not_outlier_hoa"] = zscores_hoa.abs() < 3
df3["not_outlier_lat"] = zscores_lat.abs() < 3
df3["not_outlier_lng"] = zscores_lng.abs() < 3

D = df3[
    df3.eval(
        'not_outlier_bed and not_outlier_bath and not_outlier_acre_lot and r
    )
].drop(
    columns=[
        "not_outlier_bed",
        "not_outlier_bath",
        "not_outlier_acre_lot",
        "not_outlier_house_size",
        "not_outlier_year_built",
        "not_outlier_days_on_market",
        "not_outlier_square_feet",
        "not_outlier_hoa",
        "not_outlier_lat",
        "not_outlier_lng",
    ]
)
D = D.drop(D.index[D['hoa_month'] >= 3500.0])
D = D.drop(D.index[D['acre_lot'] >= 50.0])
D.head(2)
```

Plot boxplots of variables after outliers removal

In [ ]:

```
lst = list(D[['bed','bath','acre_lot','house_size','price','days_on_market',

D = D.reset_index(drop=True)
for column in lst:
    plt.figure(figsize=(10, 4))
    plt.subplot(131)
    sns.boxplot(D[column])
    plt.show()
```

Dataset variables quantitative information after outlier removal

In [ ]:

```
D[['bed','bath','acre_lot','house_size','price','days_on_market','square_fee
```

In [ ]:

```
b = [
    'AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA','HI', '
    'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV',
    'UT', 'VA', 'VT', 'WA', 'WI', 'WV', 'WY'
]
```

```
L = list(D['state'].unique())
print(L, len(L))
x=[]
for i in b:
    if i not in L:
        x.append(i)
print(x, sep="")
print(len(x))
```

Calculate correlations between variables

In [ ]:

```
Xs = D.loc[:, D.columns!='price']

g = sns.PairGrid(Xs, diag_sharey=False, corner=True)
g.map_lower(sns.scatterplot)
g.map_diag(sns.kdeplot)
```

In [ ]:

```
round(Xs.corr(),2)
```

Convert Categorical Variables to Numerical

In the following code we have transformed status, sale_type and property_type to numerical values in order to check for collinearity, correlation and VIF.

In [ ]:

```
Xs['status_numerical'] = np.where(Xs['status'] == 'for_sale', 1, 0) #1 if fo
Xs['sale_type_numerical'] = np.where(Xs['sale_type'] == 'MLS Listing', 1, 0)

property_type_mapping = {
    'Single Family Residential': 1,
    'Townhouse': 2,
    'Condo/Co-op': 3,
    'Mobile/Manufactured Home': 4,
    'Timeshare': 5,
    'Other': 6
}

Xs['property_type_numerical'] = Xs['property_type'].map(property_type_mappin
Xs['status_numerical'] = Xs['status_numerical'].astype(float)
Xs['sale_type_numerical'] = Xs['sale_type_numerical'].astype(float)
```

Plot correlation heatmap

In [ ]:

```
plt.figure(figsize=(16, 6))
mask = np.triu(np.ones_like(round(Xs.corr(),2), dtype=bool))
heatmap = sns.heatmap(round(Xs.corr(),2), mask=mask, vmin=-1, vmax=1, annot=
heatmap.set_title("Predictors' Correlations Heatmap", fontdict={'fontsize':1
```

In [ ]:

```
corr = round(Xs.corr(),1)
plt.figure(figsize=(16, 6))
mask = np.triu(np.ones_like(round(Xs.corr(),2), dtype=bool))
```

```
mask = np.triu(np.ones_like(round(Xs.corr(),2), dtype=bool))
sns.heatmap(round(Xs.corr(),2), mask=mask, vmin=-1, vmax=1, annot=True, cmap
```

In [ ]:

```
ans=sns.heatmap(round(Xs.corr(),1),  linewidths=1, cmap="Blues", center=0, a
figure = ans.get_figure()
figure.savefig('correlations.png', dpi=800)
```

Calculate VIF

In [ ]:

```
A = D[['bed','bath','acre_lot','house_size','year_built','days_on_market','s
        'latitude','longitude','population','density','county_fips','status_
       'property_type_numerical']]

y1, X1 = dmatrices('bed ~ bath+acre_lot+house_size+latitude+longitude+popula

vif1 = pd.DataFrame()
vif1['VIF'] = [variance_inflation_factor(X1.values, i) for i in range(X1.sha
vif1['variable'] = X1.columns

vif1
```

In [ ]:

```
y2, X2 = dmatrices('bath ~ bed+acre_lot+house_size+latitude+longitude+popula

vif2 = pd.DataFrame()
vif2['VIF'] = [variance_inflation_factor(X2.values, i) for i in range(X2.sha
vif2['variable'] = X2.columns

vif2
```

In [ ]:

```
y3, X3 = dmatrices('house_size ~ bed+acre_lot+bath+latitude+longitude+popula

vif3 = pd.DataFrame()
vif3['VIF'] = [variance_inflation_factor(X3.values, i) for i in range(X3.sha
vif3['variable'] = X3.columns

vif3
```

In [ ]:

```
y4, X4 = dmatrices('hoa_month~ bed+acre_lot+house_size+bath+latitude+longitu

vif4 = pd.DataFrame()
vif4['VIF'] = [variance_inflation_factor(X4.values, i) for i in range(X4.sha
vif4['variable'] = X4.columns

vif4
```

In [ ]:

```
y5, X5 = dmatrices('status_numerical~ bed+acre_lot+house_size+bath+latitude+

vif5 = pd.DataFrame()
vif5['VIF'] = [variance_inflation_factor(X5.values, i) for i in range(X5.sha
vif5['variable'] = X5.columns

vif5
```

Control/Validation/Test sets

In [ ]:

```
X = D.loc[:, D.columns!='price']
y = D["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, ra
print(len(X_train), len(y_train), len(X_test), len(y_test))
```

Impute missing values

In [ ]:

```
imputer_y = SimpleImputer()
imputer_X_int = SimpleImputer(strategy="most_frequent")
imputer_X_float = SimpleImputer()

X_train_int_array = imputer_X_int.fit_transform(
    X_train[[
        "bed",
        "bath",
        "year_built",
        "days_on_market",
        "hoa_month",
        "population",
        "county_fips",
    ]]
)
X_test_int_array = imputer_X_int.transform(
    X_test[[
        "bed",
        "bath",
        "year_built",
        "days_on_market",
        "hoa_month",
        "population",
        "county_fips",
    ]]
)
X_train_int = pd.DataFrame(
    X_train_int_array,
    columns=[
        "bed",
        "bath",
        "year_built",
        "days_on_market",
        "hoa_month",
        "population",
        "county_fips",
    ]
)
X_test_int = pd.DataFrame(
    X_test_int_array,
    columns=[
        "bed",
        "bath",
        "year_built",
        "days_on_market",
        "hoa_month"
```

```
        "hoa_month",
        "population",
        "county_fips",
    ]
)
X_train_float_array = imputer_X_float.fit_transform(
    X_train[[
        "acre_lot",
        "house_size",
        "square_feet",
        "density",
        "latitude",
        "longitude",
    ]]
)
X_test_float_array = imputer_X_float.transform(
    X_test[[
        "acre_lot",
        "house_size",
        "square_feet",
        "density",
        "latitude",
        "longitude",
    ]]
)
X_train_float = pd.DataFrame(
    X_train_float_array,
    columns=[
        "acre_lot",
        "house_size",
        "square_feet",
        "density",
        "latitude",
        "longitude",
    ]
)
X_test_float = pd.DataFrame(
    X_test_float_array,
    columns=[
        "acre_lot",
        "house_size",
        "square_feet",
        "density",
        "latitude",
        "longitude",
    ]
)
```

In [ ]:

```
y_train = imputer_y.fit_transform(y_train.values.reshape(-1, 1))[:,0]
y_test = imputer_y.transform(y_test.values.reshape(-1, 1))[:,0]
```

Append numeric and categorical columns

In [ ]:

```
X_train_categorical = X_train[[
    "status",
    "state",
    "city",
```

```
    "sale_type",
    "property_type",
    "imprecise",
    "military",
]]
X_test_categorical = X_test[[
    "status",
    "state",
    "city",
    "sale_type",
    "property_type",
    "imprecise",
    "military",
]]
X_train_ = pd.concat(
    [
        X_train_int.reset_index(drop=True),
        X_train_float.reset_index(drop=True),
        X_train_categorical.reset_index(drop=True),
    ],
    axis=1
)
X_test_ = pd.concat(
    [
        X_test_int.reset_index(drop=True),
        X_test_float.reset_index(drop=True),
        X_test_categorical.reset_index(drop=True),
    ],
    axis=1
)
X_train_["price"] = y_train
```

Linear regression model

In [ ]:

```
ols_model_without_interaction_terms = sms.ols(
    formula="price ~ state + bed + bath + acre_lot + city + house_size + sta
    data=X_train_
).fit()
```

In [ ]:

```
y_test_pred1 = ols_model_without_interaction_terms.predict(X_test_)
```

Linear regression - model

In [ ]:

```
ols_model_without_interaction_terms.summary()
```

Linear regression - results

In [ ]:

```
mean_squared_error(y_test, y_test_pred1)
```

In [ ]:

```
mae = mean_absolute_error(y_test, y_test_pred1)
mse = mean_squared_error(y_test, y_test_pred1)
```

```
r2 = r2_score(y_test, y_test_pred1)
print('MAE:', mae,'MSE:', mse, 'R2.score',r2)
```

In [ ]:
```
plt.scatter(y_test, y_test_pred1, alpha=0.5, color = 'blue')

plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

plt.show()
```

Linear regression model with nonlinear terms

In [ ]:
```
agg_bed = X_train_.groupby(['bed'])['price'].agg(pd.Series.mean).reset_index

sns.residplot(x='bath', y='price', data=X_train_)
plt.show()
```

In [ ]:
```
agg_bed = X_train_.groupby(['bed'])['price'].agg(pd.Series.mean).reset_index

plt.figure(figsize = (5,4))
plt.plot(agg_bed["bed"], agg_bed["price"], 'b.')
plt.xlabel('bed')
plt.ylabel('price')
plt.title("price vs bed")
plt.show()
```

In [ ]:
```
agg_bath = X_train_.groupby(['bath'])['price'].agg(pd.Series.mean).reset_inc

plt.figure(figsize = (5,4))
plt.plot(agg_bath["bath"], agg_bath["price"], 'b.')
plt.xlabel('bath')
plt.ylabel('price')
plt.title("price vs bath")
plt.show()
```

In [ ]:
```
agg_acre_lot = X_train_.groupby(['acre_lot'])['price'].agg(pd.Series.mean).r

plt.figure(figsize = (5,4))
plt.plot(agg_acre_lot["acre_lot"], agg_acre_lot["price"], 'b.')
plt.xlabel('acre_lot')
plt.ylabel('price')
plt.title("price vs acre_lot")
plt.show()
```

In [ ]:
```
agg_house_size = X_train_.groupby(['house_size'])['price'].agg(pd.Series.mea

plt.figure(figsize = (5,4))
plt.plot(agg_house_size["house_size"], agg_house_size["price"], 'b.')
plt.xlabel('house_size')
plt.ylabel('price')
plt.title("price vs house_size")
plt.show()
```

```
plt.show()
```

In [ ]:

```
agg_density = X_train_.groupby(['density'])['price'].agg(pd.Series.mean).res

plt.figure(figsize = (5,4))
plt.plot(agg_density["density"], agg_density["price"], 'b.', color='blue')
plt.xlabel('density')
plt.ylabel('price')
plt.title("price vs density")
plt.show()
```

In [ ]:

```
agg_density = X_train_.groupby(['square_feet'])['price'].agg(pd.Series.mean)

plt.figure(figsize = (5,4))
plt.plot(agg_density["square_feet"], agg_density["price"], 'b.', color='blue
plt.xlabel('square_feet')
plt.ylabel('price')
plt.title("price vs square_feet")
plt.show()
```

In [ ]:

```
agg_density = X_train_.groupby(['population'])['price'].agg(pd.Series.mean).

plt.figure(figsize = (5,4))
plt.plot(agg_density["population"], agg_density["price"], 'b.', color='blue'
plt.xlabel('population')
plt.ylabel('price')
plt.title("price vs population")
plt.show()
```

In [ ]:

```
polynomial_bed = PolynomialFeatures(degree=2)
polynomial_bath = PolynomialFeatures(degree=2)
polynomial_acre_lot = PolynomialFeatures(degree=2)
polynomial_house_size = PolynomialFeatures(degree=2)
polynomial_density = PolynomialFeatures(degree=2)
```

Linear regression with nonlinear terms - "bed" polynomial terms

In [ ]:

```
polynomial_bed_train_ = polynomial_bed.fit_transform(X_train_[["bed"]])
polynomial_bed_test_ = polynomial_bed.transform(X_test_[["bed"]])

X_train_["polynomial_bed_0"] = polynomial_bed_train_[:,0]
X_train_["polynomial_bed_1"] = polynomial_bed_train_[:,1]
X_train_["polynomial_bed_2"] = polynomial_bed_train_[:,2]

X_test_["polynomial_bed_0"] = polynomial_bed_test_[:,0]
X_test_["polynomial_bed_1"] = polynomial_bed_test_[:,1]
X_test_["polynomial_bed_2"] = polynomial_bed_test_[:,2]
```

Linear regression with nonlinear terms - "bath" polynomial terms

```
polynomial_bath_train_ = polynomial_bath.fit_transform(X_train_[["bath"]])
polynomial_bath_test_ = polynomial_bath.transform(X_test_[["bath"]])

X_train_["polynomial_bath_0"] = polynomial_bath_train_[:,0]
X_train_["polynomial_bath_1"] = polynomial_bath_train_[:,1]
X_train_["polynomial_bath_2"] = polynomial_bath_train_[:,2]

X_test_["polynomial_bath_0"] = polynomial_bath_test_[:,0]
X_test_["polynomial_bath_1"] = polynomial_bath_test_[:,1]
X_test_["polynomial_bath_2"] = polynomial_bath_test_[:,2]
```

Linear regression with nonlinear terms - "acre lot" polynomial terms

```
polynomial_acre_lot_train_ = polynomial_acre_lot.fit_transform(X_train_[["ac
polynomial_acre_lot_test_ = polynomial_acre_lot.transform(X_test_[["acre_lot

X_train_["polynomial_acre_lot_0"] = polynomial_acre_lot_train_[:,0]
X_train_["polynomial_acre_lot_1"] = polynomial_acre_lot_train_[:,1]
X_train_["polynomial_acre_lot_2"] = polynomial_acre_lot_train_[:,2]

X_test_["polynomial_acre_lot_0"] = polynomial_acre_lot_test_[:,0]
X_test_["polynomial_acre_lot_1"] = polynomial_acre_lot_test_[:,1]
X_test_["polynomial_acre_lot_2"] = polynomial_acre_lot_test_[:,2]
```

Linear regression with nonlinear terms - "house size" polynomial terms

```
polynomial_house_size_train_ = polynomial_house_size.fit_transform(X_train_[
polynomial_house_size_test_ = polynomial_house_size.transform(X_test_[["hous

X_train_["polynomial_house_size_0"] = polynomial_house_size_train_[:,0]
X_train_["polynomial_house_size_1"] = polynomial_house_size_train_[:,1]
X_train_["polynomial_house_size_2"] = polynomial_house_size_train_[:,2]

X_test_["polynomial_house_size_0"] = polynomial_house_size_test_[:,0]
X_test_["polynomial_house_size_1"] = polynomial_house_size_test_[:,1]
X_test_["polynomial_house_size_2"] = polynomial_house_size_test_[:,2]
```

Linear regression with nonlinear terms - "population density" polynomial terms

```
polynomial_density_train_ = polynomial_density.fit_transform(X_train_[["dens
polynomial_density_test_ = polynomial_density.transform(X_test_[["density"]]

X_train_["polynomial_density_0"] = polynomial_density_train_[:,0]
X_train_["polynomial_density_1"] = polynomial_density_train_[:,1]
X_train_["polynomial_density_2"] = polynomial_density_train_[:,2]

X_test_["polynomial_density_0"] = polynomial_density_test_[:,0]
X_test_["polynomial_density_1"] = polynomial_density_test_[:,1]
```

```
X_test_["polynomial_density_1"] = polynomial_density_test_[:,1]
X_test_["polynomial_density_2"] = polynomial_density_test_[:,2]
```

In [ ]:

```
ols_model_with_nonlinear_terms = sms.ols(
    formula="price ~ polynomial_bed_0 + polynomial_bed_1 + polynomial_bed_2
    data=X_train_
).fit()
```

In [ ]:

```
y_test_pred2 = ols_model_with_nonlinear_terms.predict(X_test_)
```

Linear regression with nonlinear terms - model

In [ ]:

```
ols_model_with_nonlinear_terms.summary()
```

Linear regression with nonlinear terms - results

In [ ]:

```
mae = mean_absolute_error(y_test, y_test_pred2)
mse = mean_squared_error(y_test, y_test_pred2)
r2 = r2_score(y_test, y_test_pred2)
print('MAE:', mae,'MSE:', mse, 'R2.score',r2)
```

In [ ]:

```
plt.scatter(y_test, y_test_pred2, alpha=0.5, color = 'blue')

plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

plt.show()
```

LightGBM regression

In [ ]:

```
lgbm_model_params = {
        'max_depth': (5, 15),
        'num_leaves': (1000, 5000)
}
lgbm_model_cv = RandomizedSearchCV(
    estimator=lgbm.LGBMRegressor(verbose=0),
    param_distributions=lgbm_model_params,
    cv=3,
    n_iter=3,
    verbose=0,
)
lgbm_model_cv.fit(X_train, y_train)

y_test_pred3 = lgbm_model_cv.predict(X_test)
```

LightGBM regression - results

In [ ]:

```
mae = mean_absolute_error(y_test, y_test_pred3)
mse = mean_squared_error(y_test, y_test_pred3)
r2 = r2_score(y_test, y_test_pred3)
```