Housing Market Model (Team TNA)

In [16]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will l
import re

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sav
```

```
/kaggle/input/effects-of-policy-on-the-housing-market/sample_submissions.csv
/kaggle/input/effects-of-policy-on-the-housing-market/train.csv
/kaggle/input/effects-of-policy-on-the-housing-market/test.csv
```

Uploading and splitting of data

In [17]:

```
# Paths
input_path = '/kaggle/input/effects-of-policy-on-the-housing-market/'

# Load data # IDs removed from data # Names removed due to being relatively
train_df = pd.read_csv(os.path.join(input_path, 'train.csv'))
test_df = pd.read_csv(os.path.join(input_path, 'test.csv'))

X_train, y_train = train_df.drop(columns = train_df.columns[0:2]).drop(colum
```

Exploratory Data Analysis

- `name` contains multiple values that should be split into a list to be placed into certain columns, and the first value appears to be categorical.

In [18]:

```
X_train.head()
# float(re.sub("[^0-9.]","",n))
```

Out[18]:

	id	name	neighborhood_overview	host_id	host_name
0	19792418	Home in Vancouver · ★4.75 · 1 bedroom · 1 bed ...	Everything you need is nearby. <br /><br />Hig...	57488206	Jessi
1	1015650685503221866	Guest suite in Vancouver · ★New · 2 bedrooms ·...	NaN	139792573	Daniel
2	35265562	Guest suite in Vancouver · ★4.85 · 2 bedrooms ...	Beautiful neighbourhood close to prosperous Ma...	265504225	Alex
3	911948980885194155	Home in Vancouver · ★5.0 · 1 bedroom · 1 bed ·...	We are located in a quiet residential neighbor...	22595056	Raymond
4	46069251	Guest suite in Vancouver · ★4.93 · 1 bedroom ·...	Kitsilano at it's best! Short walk to all the ...	65683877	Yendi

5 rows × 41 columns

Data type for each column

- A `FunctionTransformer()` will be required to convert the data type for the categorical variables to string. The numerical variables should be scaled with a `StandardScaler()`.
- Some numerical variables are presented as strings rather than numbers (e.g. `host_response_rate`, `host_acceptance_rate`, `price`) and thus will need to be converted from their original data type with a different `FunctionTransformer()`.

In [19]:

```
X_train.dtypes
```

Out[19]:

```
id                                  int64
name                               object
neighborhood_overview              object
host_id                             int64
host_name                          object
host_response_time                 object
host_response_rate                 object
host_acceptance_rate               object
host_is_superhost                  object
host_listings_count                 int64
host_total_listings_count           int64
neighbourhood                      object
neighbourhood_cleansed             object
latitude                          float64
longitude                         float64
property_type                      object
room_type                          object
accommodates                        int64
beds                              float64
amenities                          object
price                              object
minimum_nights                      int64
maximum_nights                      int64
minimum_nights_avg_ntm            float64
maximum_nights_avg_ntm            float64
availability_30                     int64
availability_60                     int64
availability_90                     int64
availability_365                    int64
number_of_reviews                   int64
number_of_reviews_ltm               int64
review_scores_rating              float64
review_scores_accuracy            float64
review_scores_cleanliness         float64
review_scores_checkin             float64
review_scores_communication       float64
review_scores_location            float64
review_scores_value               float64
instant_bookable                   object
calculated_host_listings_count      int64
reviews_per_month                 float64
dtype: object
```

Testing the need for imputation within data

- Some of the data is missing, and thus imputation is required.
- `name` contains information on `bathrooms`, `bedrooms`, `beds` and `review_scores_rating`, while the `bathrooms` and `bedrooms` columns consist entirely of `NaN` data, so `name` will need to be modified to extract such information and allocate them accordingly.

In [20]:

```
X_train.isna().any()
```

```
Out[20]: id                                False
         name                              False
         neighborhood_overview              True
         host_id                           False
         host_name                         False
         host_response_time                 True
         host_response_rate                 True
         host_acceptance_rate               True
         host_is_superhost                  True
         host_listings_count               False
         host_total_listings_count         False
         neighbourhood                      True
         neighbourhood_cleansed            False
         latitude                          False
         longitude                         False
         property_type                     False
         room_type                         False
         accommodates                      False
         beds                               True
         amenities                         False
         price                              True
         minimum_nights                    False
         maximum_nights                    False
         minimum_nights_avg_ntm            False
         maximum_nights_avg_ntm            False
         availability_30                   False
         availability_60                   False
         availability_90                   False
         availability_365                  False
         number_of_reviews                 False
         number_of_reviews_ltm             False
         review_scores_rating               True
         review_scores_accuracy             True
         review_scores_cleanliness          True
         review_scores_checkin              True
         review_scores_communication        True
         review_scores_location             True
         review_scores_value                True
         instant_bookable                  False
         calculated_host_listings_count    False
         reviews_per_month                  True
         dtype: bool
```

Categories within most categorical variables

- `name` is meant to be a categorical variable, but has not been transformed to allow for that until later.
- `host_response_time` and `host_is_superhost` will require imputation due to containing `nan`.
- `amenities` would theoretically require extraction of consistent features from `neighborhood_overview`.

In [21]:

```
# print(X_train["name"].unique())
print(X_train["host_response_time"].unique()) # Missing values
print(X_train["host_is_superhost"].unique()) # Boolean, missing values
print(X_train["neighbourhood"].unique())
print(X_train["neighbourhood_cleansed"].unique())
print(X_train["property_type"].unique())
print(X_train["room_type"].unique())
# print(X_train["amenities"].unique()) # All empty
print(X_train["instant_bookable"].unique()) # Boolean
```

```
[nan 'within a few hours' 'within an hour' 'within a day'
 'a few days or more']
['f' 't' nan]
['Vancouver, British Columbia, Canada' nan
 'Delta, British Columbia, Canada'
 'Vancouver bc, British Columbia, Canada'
 'Vancouver , British Columbia, Canada' 'Vancouver, Canada'
 'West Vancouver, BC , Canada' 'Vancouver, British Columbia (BC), Canada']
['Hastings-Sunrise' 'Sunset' 'Riley Park' 'Kitsilano' 'Downtown'
 'Fairview' 'Dunbar Southlands' 'Kensington-Cedar Cottage' 'Marpole'
 'Mount Pleasant' 'West End' 'Grandview-Woodland' 'Renfrew-Collingwood'
 'Downtown Eastside' 'Oakridge' 'South Cambie' 'West Point Grey'
 'Arbutus Ridge' 'Killarney' 'Victoria-Fraserview' 'Kerrisdale'
 'Strathcona' 'Shaughnessy']
['Entire home' 'Entire guest suite' 'Entire condo' 'Entire rental unit'
 'Entire townhouse' 'Private room in guest suite' 'Private room in home'
 'Private room in townhouse' 'Private room in condo'
 'Private room in rental unit' 'Entire loft' 'Entire serviced apartment'
 'Shared room in rental unit' 'Private room in guesthouse'
 'Private room in villa' 'Room in boutique hotel' 'Entire guesthouse'
 'Casa particular' 'Shared room in home' 'Private room in tiny home'
 'Private room in loft' 'Tiny home' 'Entire timeshare'
 'Entire vacation home' 'Camper/RV' 'Tower' 'Entire cabin'
 'Room in aparthotel' 'Room in bed and breakfast'
 'Private room in casa particular' 'Private room in bed and breakfast'
 'Entire villa' 'Entire bungalow' 'Room in hotel' 'Entire place'
 'Shared room in condo' 'Entire cottage' 'Private room in bungalow'
 'Private room in boat' 'Private room in camper/rv' 'Boat'
 'Shared room in villa' 'Private room in serviced apartment'
 'Private room in castle' 'Cave' 'Shared room in loft']
['Entire home/apt' 'Private room' 'Shared room' 'Hotel room']
['f' 't']
```

Most continuous variables

- From the data that can be seen, `beds` and several of the review score variables are missing significant amounts of data.
- Due to differing scales for most columns, `StandardScaler()` will be required.

In [22]:

```
display(X_train.describe())
```

	id	host_id	host_listings_count	host_total_listings_count	
count	5.352000e+03	5.352000e+03	5352.000000	5352.000000	535
mean	4.398265e+17	1.916746e+08	11.917601	18.213191	4
std	4.317929e+17	1.808806e+08	45.576743	61.783016	
min	1.318800e+04	4.662000e+03	1.000000	1.000000	4
25%	3.365765e+07	2.637491e+07	1.000000	1.000000	4
50%	5.987246e+17	1.318506e+08	2.000000	3.000000	4
75%	8.769719e+17	3.428838e+08	5.000000	8.000000	4
max	1.044548e+18	5.505990e+08	466.000000	1037.000000	4

8 rows × 27 columns

Extraction of Data and Preprocessing

Methods to convert data within function transformers

In [23]:

```
# Convert categorical column data type to string
from sklearn.preprocessing import FunctionTransformer
# https://stackoverflow.com/questions/59476179/is-it-possible-to-change-panc
def to_categorical(x):
    return pd.DataFrame(x).astype("string")
fun_tr = FunctionTransformer(to_categorical)
```

In [24]:

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTra
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.compose import make_column_transformer
from sklearn.feature_extraction.text import CountVectorizer

# Table transformation functions
def nameterms(x):
    if isinstance(x, (list)) or " · " not in x:
        return x
    else:
        return x.split(" · ")

def strtoint(x, pos):
    if isinstance(x, list) and len(x) > abs(pos):
        new = x[pos]
    else:
        return np.nan
    n = re.sub("[^0-9.]","",new)
    if n == "":
        return np.nan
    else:
        return float(n)

def strtofloat(x):
    if isinstance(x, list) and x and len(x) != 4:
        rate = x[1]
    else:
        return np.nan
    ratenew = re.sub("[^0-9.]","", str(rate))
    if ratenew == "":
        return np.nan
    else:
        return float(ratenew)

def strtopercent(x):
    return pd.DataFrame(x).map(lambda val: float(val.replace('%', '')) / 100

def strtomoney(x):
    return pd.DataFrame(x).map(lambda val: float(val.replace('$', '').replac
```

Creation of `NameTransformer`

- Extract data from `name` column for use later in the table.
- `X_train_transformed` and `test_df_transformed` created to test transformers `NameTransformer` and later `preprocessor`.

In [25]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class NameTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        transformed_df = X.copy()
        transformed_df['name'] = transformed_df['name'].map(nameterms)
        transformed_df['bathrooms'] = transformed_df['name'].map(lambda x: s
        transformed_df['bedrooms'] = transformed_df['name'].map(lambda x: st
        transformed_df['beds'] = transformed_df['name'].map(lambda x: strtoi
        transformed_df['review_scores_rating'] = transformed_df['name'].map(
        transformed_df['name'] = transformed_df['name'].apply(lambda x: x[0]
        return transformed_df

# Create the pipeline
pipeline_transformer = make_pipeline(
    NameTransformer(),
)

X_train_transformed = pipeline_transformer.fit_transform(X = X_train)
test_df_transformed = pipeline_transformer.fit_transform(test_df)

X_train_transformed.head()
```

Out[25]:

	id	name	neighborhood_overview	host_id	host_name
0	19792418	Home in Vancouver	Everything you need is nearby. <br /><br />Hig...	57488206	Jessi
1	1015650685503221866	Guest suite in Vancouver	NaN	139792573	Daniel
2	35265562	Guest suite in Vancouver	Beautiful neighbourhood close to prosperous Ma...	265504225	Alex
3	911948980885194155	Home in Vancouver	We are located in a quiet residential neighbor...	22595056	Raymond
4	46069251	Guest suite in Vancouver	Kitsilano at it's best! Short walk to all the ...	65683877	Yendi

5 rows × 43 columns

Preprocessing of features by column

- `percentage_features` and `currency_features` need to be converted from non-numeric values to numeric values. Due to `percentage_features` and `currency_features` being strings before transformation, their imputers both use "most_frequent" as their imputation strategy.
- Since we did not know of any way to extract data from `neighborhood_overview` into `amenities`, both were dropped.
- `id`, `neighborhood_overview`, `host_id` and `host_name` are used more for identification than statistics and thus are removed.

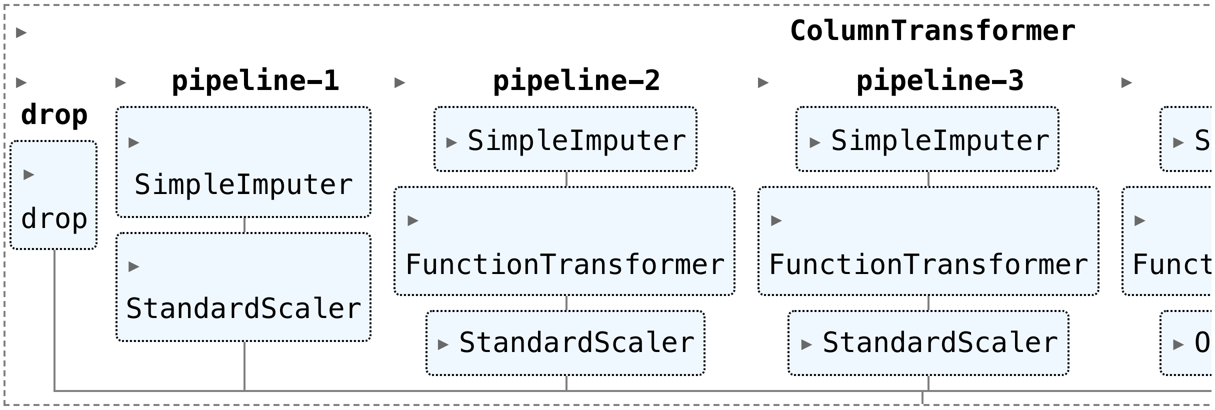
In [26]:

```
numeric_features = ["host_listings_count", "host_total_listings_count", "lat
percentage_features = ["host_response_rate", "host_acceptance_rate"]
currency_features = ["price"]
binary_features = ["host_is_superhost","instant_bookable",]
categorical_features = ["name", "host_response_time", "neighbourhood_cleanse
# vector_features = ["neighborhood_overview",]

# drop_features = ["id","host_id", "host_name", "neighbourhood"]
drop_features = ["id", "neighborhood_overview","host_id", "host_name", "neig

numeric_transformer = make_pipeline(
    SimpleImputer(strategy="mean"),
    StandardScaler(),
)
percentage_transformer = make_pipeline(
    SimpleImputer(strategy="most_frequent"), # Since most data is string bef
    FunctionTransformer(strtopercent, validate=False),
    StandardScaler(),
)
currency_transformer = make_pipeline(
    SimpleImputer(strategy="most_frequent"), # Since most data is string bef
    FunctionTransformer(strtomoney, validate=False),
    StandardScaler(),
)
binary_transformer = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    fun_tr,
    OneHotEncoder(handle_unknown="ignore", sparse_output=False, drop = "if_b
)
categorical_transformer = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    fun_tr,
    OneHotEncoder(handle_unknown="infrequent_if_exist", sparse_output=False)
)
# vector_transformer = make_pipeline(
#     CountVectorizer(stop_words="english"),
#     LogisticRegression(random_state=123)
# )

preprocessor = make_column_transformer(
    ("drop", drop_features),
    (numeric_transformer, numeric_features),
    (percentage_transformer, percentage_features),
    (currency_transformer, currency_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
    # (vector_transformer, vector_features),
)
preprocessor.fit(X_train_transformed)
display(preprocessor)
```



In [27]:
```
categorical_features_new = list(preprocessor.named_transformers_["pipeline-5
new_columns = (
    numeric_features + percentage_features + currency_features + binary_feat
)
X_train_enc = pd.DataFrame(
    preprocessor.transform(X_train_transformed), index=X_train_transformed.i
)
X_train_enc.head()
```

Out[27]:

	host_listings_count	host_total_listings_count	latitude	longitude	accommodates
0	-0.195679	-0.246259	0.996106	1.472784	-0.743460
1	-0.239566	-0.230072	-1.949916	0.522785	0.724263
2	-0.239566	-0.278633	-0.486575	0.367678	1.213504
3	-0.239566	-0.278633	-1.922130	0.378483	-0.743460
4	-0.217623	-0.246259	-0.163904	-1.319147	0.235022

5 rows × 143 columns

Cross-validation Across Various Regression Models

- For optimisation of regression models, will use R^2 as statistical measure of model quality to measure effectiveness in predicting future values.
- `mean_std_cross_val_scores()` provides both the mean and the standard deviation of the cross validations, allowing for a better understanding of the model's effectiveness and consistency.

In [28]:

```python
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_val_score,
    cross_validate,
    train_test_split,
)

def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i

    return pd.Series(data=out_col, index=mean_scores.index)
```

In [29]:

```
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, HistGradientBoostingRegr

results = {}
scoring_metric = ["neg_root_mean_squared_error","r2","neg_mean_absolute_perc
```

Initial ridge, decision tree, random forest and tree-based ensemble models

In [30]:

```
# Ridge regression model
# Hyperparameters: alpha
pipe_ridge = make_pipeline(pipeline_transformer, preprocessor, Ridge(alpha =
# pipe_ridge.fit(X_train, y_train)

results["Ridge"] = mean_std_cross_val_scores(
    pipe_ridge, X_train, y_train, cv = 10, return_train_score = True, scorin
)

# Decision tree
# Hyperparameters: max_depth
# https://ken-hoffman.medium.com/decision-tree-hyperparameters-explained-491
pipe_dt = make_pipeline(pipeline_transformer, preprocessor, DecisionTreeRegr
# pipe_dt.fit(X_train, y_train)
results["Decision tree"] = mean_std_cross_val_scores(
    pipe_dt, X_train, y_train, cv = 10, return_train_score = True, scoring=s
)

# Random forest
# Hyperparameters: n_estimators, max_depth, max_features
pipe_rf = make_pipeline(
    pipeline_transformer, preprocessor, RandomForestRegressor(n_jobs=-1, ran
)
# pipe_rf.fit(X_train, y_train)
results["Random forest"] = mean_std_cross_val_scores(
    pipe_rf, X_train, y_train, cv = 10, return_train_score = True, scoring=s
)

# Tree-based ensemble model (HistGradientBoostingClassifier due to large dat
# Hyperparameters: learning_rate, max_depth
pipe_sklearn_histGB = make_pipeline(
    pipeline_transformer,
    preprocessor,
    HistGradientBoostingRegressor(random_state=76),
)
# pipe_sklearn_histGB.fit(X_train, y_train)
```

```
results["sklearn_histGB"] = mean_std_cross_val_scores(
    pipe_sklearn_histGB, X_train, y_train, cv = 10, return_train_score = Tru
)
# pd.DataFrame(results).T
```

```
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

Ridge regression hyperparameter optimisation

In [31]:

```
# Ridge regression model
# Hyperparameters: alpha
param_grid_ridge = {
    "ridge__alpha": [0.01, 0.1, 1, 10, 100],
}
random_search_ridge = RandomizedSearchCV(
    pipe_ridge, param_distributions = param_grid_ridge, n_iter=10,
    n_jobs = -1, return_train_score = True, random_state = 76,scoring='r2'
)
random_search_ridge.fit(X_train, y_train)

best_params_ridge = random_search_ridge.best_params_
best_score_ridge = random_search_ridge.best_score_

print("best_params_ridge: " + str(best_params_ridge))
print("best_score_ridge: " + str(best_score_ridge))

pipe_ridge_opt = make_pipeline(
    pipeline_transformer,
    preprocessor,
    Ridge(
        alpha = best_params_ridge["ridge__alpha"],
        max_iter = 200,
    )
)
pipe_ridge_opt.fit(X_train, y_train)
```

```
results["Ridge (Optimised)"] = mean_std_cross_val_scores(pipe_ridge_opt, X_t
# pd.DataFrame(results).T
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/model_selection/_search.py:3
05: UserWarning: The total space of parameters 5 is smaller than n_iter=10.
Running 5 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
best_params_ridge: {'ridge__alpha': 100}
best_score_ridge: 0.2835346371356892
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

Decision tree hyperparameter optimisation

In [32]:

```
# Decision tree
# Hyperparameters: max_depth
# https://ken-hoffman.medium.com/decision-tree-hyperparameters-explained-491
param_grid_dt = {
    "decisiontreeregressor__max_depth": np.arange(1, 20, 4),
}
random_search_dt = RandomizedSearchCV(
    pipe_dt, param_distributions = param_grid_dt, n_iter=10,
    n_jobs = -1, return_train_score = True, random_state = 76, scoring='r2'
)
random_search_dt.fit(X_train, y_train)

best_params_dt = random_search_dt.best_params_
best_score_dt = random_search_dt.best_score_
print("best_params_dt: " + str(best_params_dt))
print("best_score_dt: " + str(best_score_dt))

pipe_dt_opt = make_pipeline(
    pipeline_transformer,
    preprocessor,
    DecisionTreeRegressor(
        max_depth = best_params_dt["decisiontreeregressor__max_depth"],
        random_state=76
    )
)
results["Decision tree (Optimised)"] = mean_std_cross_val_scores(
    pipe_dt_opt, X_train, y_train, cv = 10, return_train_score = True, scori
)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/model_selection/_search.py:3
05: UserWarning: The total space of parameters 5 is smaller than n_iter=10.
Running 5 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
best_params_dt: {'decisiontreeregressor__max_depth': 5}
best_score_dt: 0.242331494810347
```

```
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

Random forest hyperparameter optimisation

In [33]:

```
# Random forest
# Hyperparameters: n_estimators, max_depth, max_features, class_weight
param_grid_rf = {
    "randomforestregressor__max_depth": np.arange(1, 20, 4),
    "randomforestregressor__n_estimators": [100, 200, 300, 400, 500],
    "randomforestregressor__max_features": [10, 15, 20, 25, 30, 35, 40],
}
random_search_rf = RandomizedSearchCV(
    pipe_rf, param_distributions = param_grid_rf, n_iter=10,
    n_jobs = -1, return_train_score = True, random_state = 76, scoring='r2'
)
random_search_rf.fit(X_train, y_train)

best_params_rf = random_search_rf.best_params_
best_score_rf = random_search_rf.best_score_
print("best_params_rf: " + str(best_params_rf))
print("best_score_rf: " + str(best_score_rf))

pipe_rf_opt = make_pipeline(
    pipeline_transformer,
    preprocessor,
    RandomForestRegressor(
        n_estimators = best_params_rf["randomforestregressor__n_estimators"]
        max_depth = best_params_rf["randomforestregressor__max_depth"],
        max_features = best_params_rf["randomforestregressor__max_features"]
        n_jobs=-1, random_state=76,
    )
)
results["Random forest (Optimised)"] = mean_std_cross_val_scores(
    pipe_rf_opt, X_train, y_train, cv = 10, return_train_score = True, scori
)
```

```
best_params_rf: {'randomforestregressor__n_estimators': 400, 'randomforestre
gressor__max_features': 35, 'randomforestregressor__max_depth': 5}
best_score_rf: 0.2854408171022718
```

```
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

Tree-based ensemble hyperparameter optimisation

In [34]:

```
# Tree-based ensemble model (HistGradientBoostingRegressor due to large data
# Hyperparameters: learning_rate, max_depth
param_grid_sklearn_histGB = {
    "histgradientboostingregressor__max_depth": np.arange(1, 20, 4),
    "histgradientboostingregressor__learning_rate": [0.1, 0.3, 0.5, 0.7, 0.9
}
random_search_sklearn_histGB = RandomizedSearchCV(
    pipe_sklearn_histGB, param_distributions = param_grid_sklearn_histGB, n_
    n_jobs = -1, return_train_score = True, random_state = 76,scoring='r2'
)
random_search_sklearn_histGB.fit(X_train, y_train)

best_params_sklearn_histGB = random_search_sklearn_histGB.best_params_
best_score_sklearn_histGB = random_search_sklearn_histGB.best_score_

print("best_params_sklearn_histGB: " + str(best_params_sklearn_histGB))
print("best_score_sklearn_histGB: " + str(best_score_sklearn_histGB))

pipe_sklearn_histGB_opt = make_pipeline(
    pipeline_transformer,
    preprocessor,
    HistGradientBoostingRegressor(
        learning_rate = best_params_sklearn_histGB["histgradientboostingregr
        max_depth = best_params_sklearn_histGB["histgradientboostingregresso
        random_state=76
    ),
)
results["sklearn_histGB (Optimised)"] = mean_std_cross_val_scores(
    pipe_sklearn_histGB_opt, X_train, y_train, cv = 10, return_train_score =
)
```

```
best_params_sklearn_histGB: {'histgradientboostingregressor__max_depth': 1,
'histgradientboostingregressor__learning_rate': 0.3}
best_score_sklearn_histGB: 0.29169275617122853
```

```
/tmp/ipykernel_30/3623116645.py:34: FutureWarning: Series.__getitem__ treati
ng keys as positions is deprecated. In a future version, integer keys will a
lways be treated as labels (consistent with DataFrame behavior). To access a
value by position, use `ser.iloc[pos]`
  out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
```

Results from All Models

- `pipe_dt` is likely to have overfitted on the training data as its `train_r2` score is unrealistically high, and its `test_r2` score dropped into the negatives.

- `pipe_sklearn_histGB_opt` provides the greatest `test_r2`, while `pipe_rf_opt` provides a similar `test_r2` with a smaller standard deviation.

- `test_neg_root_mean_squared_error` and `test_neg_mean_absolute_percentage_error` have smaller absolute values for `pipe_sklearn_histGB_opt`.

- As such, `pipe_sklearn_histGB_opt` is arguably slightly more optimal for prediction, along with requiring less time for fitting and scoring.

In [35]:

```
pd.DataFrame(results).T.sort_values('test_r2', ascending=False)
```

Out[35]:

	fit_time	score_time	test_neg_root_mean_squared_error	train_neg_r
sklearn_histGB (Optimised)	0.255 (+/- 0.006)	0.031 (+/- 0.001)	-1120.762 (+/- 33.061)	
Random forest (Optimised)	1.433 (+/- 0.105)	0.126 (+/- 0.006)	-1125.229 (+/- 35.699)	
Ridge (Optimised)	0.243 (+/- 0.065)	0.052 (+/- 0.016)	-1127.742 (+/- 34.152)	
Ridge	0.248 (+/- 0.073)	0.047 (+/- 0.005)	-1131.805 (+/- 34.525)	
Random forest	3.206 (+/- 0.104)	0.060 (+/- 0.001)	-1153.308 (+/- 28.898)	
sklearn_histGB	0.955 (+/- 0.029)	0.032 (+/- 0.001)	-1155.255 (+/- 30.932)	
Decision tree (Optimised)	0.138 (+/- 0.002)	0.024 (+/- 0.001)	-1168.827 (+/- 44.247)	
Decision tree	0.237 (+/- 0.027)	0.024 (+/- 0.001)	-1629.723 (+/- 33.542)	

Production of Results

In [36]:

```
# Given test_r2, pipe_rf_opt provides the best prediction
pipe_sklearn_histGB_opt.fit(X_train, y_train)
y_preds = pd.DataFrame(pipe_sklearn_histGB_opt.predict(test_df).T, columns=[
ids = pd.DataFrame(test_df["id"], columns=['id'])

df = pd.concat([ids, y_preds], axis=1)
# df.head()
df.info()
df.to_csv('submission.csv',index=False)
print('Generated Submission file')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1339 entries, 0 to 1338
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               1339 non-null   int64  
 1   monthly_revenue  1339 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 21.0 KB
Generated Submission file
```