In [1]:
```
import xgboost as xgb
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
```

In [2]:
```
from datetime import datetime
import pandas as pd
```

In [16]:
```
base_data = pd.read_csv("Final_unprocessed_dataset.csv")

ML_cols = ['Price-Paid', 'Date', 'Full-Postcode', 'Postcode', 'Property-Type', 'Free
           'New-Build',
           'UK region',
           'Town/City', 'Town/Area', 'Region', 'District', 'County',
           'Transaction Type', 'Grid Reference',  # 'Active postcodes',
           'Population',
           'Households',  # 'Nearby districts',
           'Latitude', 'Longitude', 'Easting',
           'Northing',  # 'Postcodes',
           'imputed-meta-data-info',
           'imputed-pcode-and-imputed-info']
```

In [17]:
```
def machine_learning_preprocessing(dataframe):

    #Selecting Columns
    dataframe = dataframe[ML_cols]

    ###Date Feature-Engineering
    dataframe['datetime'] = dataframe['Date'].apply(
    lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M'))

    dataframe['Year'] = dataframe['datetime'].apply(
    lambda x: x.year)

    dataframe.drop(['datetime','Date'], inplace = True, axis =1)


    #Creating lists of columns names that are numeric vs non floats. Will allow for
    cols = dataframe.columns
    print(cols)
    numeric_cols = dataframe._get_numeric_data().columns
    catergorical_cols = list(set(cols) - set(numeric_cols))

    dataframe[catergorical_cols] = dataframe[catergorical_cols].astype('category')

    enc = OrdinalEncoder()
    dataframe.dropna(inplace = True)
    dataframe[catergorical_cols] = enc.fit_transform(dataframe[catergorical_cols])

    return dataframe
```

In [18]:
```
def split_to_Predictors_and_Target(dataframe):
    y = dataframe['Price-Paid']
    X = dataframe.drop('Price-Paid', inplace = False, axis=1)

    return(X,y)
```

In [19]:
```
def split_into_training_and_testing(Predictors,Targets):
    X_train, X_test, y_train, y_test = train_test_split(Predictors, Targets, test_si

    return (X_train, X_test, y_train, y_test)
```

In [20]:
```
processed_data = machine_learning_preprocessing(base_data)
Prediction, Target = split_to_Predictors_and_Target(processed_data)
X_train, X_test, y_train, y_test = split_into_training_and_testing(Prediction,Target
```

```
C:\Users\colin\AppData\Local\Temp/ipykernel_8044/998092653.py:7: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe['datetime'] = dataframe['Date'].apply(
C:\Users\colin\AppData\Local\Temp/ipykernel_8044/998092653.py:10: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  dataframe['Year'] = dataframe['datetime'].apply(
D:\Anaconda\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
Index(['Price-Paid', 'Full-Postcode', 'Postcode', 'Property-Type',
       'Freehold/Leasehold', 'New-Build', 'UK region', 'Town/City',
       'Town/Area', 'Region', 'District', 'County', 'Transaction Type',
       'Grid Reference', 'Population', 'Households', 'Latitude', 'Longitude',
       'Easting', 'Northing', 'imputed-meta-data-info',
       'imputed-pcode-and-imputed-info', 'Year'],
      dtype='object')
D:\Anaconda\lib\site-packages\pandas\core\frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
D:\Anaconda\lib\site-packages\pandas\util\_decorators.py:311: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  return func(*args, **kwargs)
D:\Anaconda\lib\site-packages\pandas\core\frame.py:3678: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  self[col] = igetitem(value, i)
```

In [59]:
```
price_predictor = xgb.XGBRegressor(
    tree_method="gpu_hist",
    enable_categorical=True,
    use_label_encoder=False,
    n_estimators=100,
    max_depth=10,
    max_leaves=100,
    gamma =0.05,
    subsample = 0.5,
    learning_rate=0.05)
```

In [60]:
```
price_predictor.fit(X_train, y_train)
```

Out[60]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=True,
             eval_metric=None, gamma=0, gpu_id=0, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=10, max_leaves=100, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)
```

In [61]:
```
price_predictor.score(X_test,y_test)
```

Out[61]:
```
0.08871795917733405
```

In [45]:
```
predicted_price = price_predictor.predict(X_train)
```

In [46]:
```
print(predicted_price[0])
```

```
153963.52
```

In [62]:
```
price_predictor.feature_importances_
```

Out[62]:
```
array([0.04298985, 0.01616368, 0.06579047, 0.16376412, 0.02040799,
       0.0378144 , 0.01336269, 0.12363151, 0.036701  , 0.04834237,
       0.01048755, 0.01812071, 0.03696876, 0.05280218, 0.04304762,
       0.02802225, 0.03827516, 0.06477173, 0.04436739, 0.        ,
       0.        , 0.09416855], dtype=float32)
```

In [30]:
```
price_predictor.save_model("xgboost_price_prediction_model.json")
```

In [31]:
```
from sklearn.tree import DecisionTreeRegressor
clf = DecisionTreeRegressor(random_state=0)
clf.fit(X_train, y_train)
clf.score(X_test,y_test)
```

Out[31]:
```
-0.6271896148814902
```

In [32]:
```
from sklearn.ensemble import GradientBoostingRegressor
clf = GradientBoostingRegressor(random_state=0)
clf.fit(X_train, y_train)
clf.score(X_test,y_test)
```

Out[32]:
```
0.08137233399778177
```

In [ ]:
```

```