Code for Catboost.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
ll drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
import pandas as pd
housing = pd.read_csv('/content/drive/MyDrive/housing.csv')
```

In [ ]:
```
y = housing.iloc[:, -2]
X = pd.concat([housing.iloc[:,:-2], housing.iloc[:, -1]],axis=1)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [ ]:
```
# Missing Automation

from sklearn.base import TransformerMixin
class NullValueImputer(TransformerMixin):
    def __init__(self):
        None
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        for column in X.columns.tolist():
            if column in X.columns[X.dtypes==object].tolist():
                X[column] = X[column].fillna(X[column].mode())
            elif column in ['total_bedrooms', 'population', 'households']:
                X['total_rooms_cat'] = pd.qcut(X['total_rooms'], 4)
                X[column] = X[column].fillna(X.groupby('total_rooms_cat')[c
                X.drop('total_rooms_cat', axis=1, inplace=True)
            else:
                X[column] = X[column].fillna(X[column].median())
        return X
```

In [ ]:
```
#For catboost, do not perform ohe
from sklearn.pipeline import Pipeline
data_pipeline = Pipeline([('null_imputer', NullValueImputer())])
X_train_cat = data_pipeline.fit_transform(X_train)
X_test_cat = data_pipeline.transform(X_test)
```

In [ ]:
```
X_train_cat
```

Out[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
2072	-119.84	36.77	6.0	1853.0	473.0	1397.0
10600	-117.80	33.68	8.0	2032.0	349.0	862.0
2494	-120.19	36.60	25.0	875.0	214.0	931.0
4284	-118.32	34.10	31.0	622.0	229.0	597.0
16541	-121.23	37.79	21.0	1922.0	373.0	1130.0
...	...	...	...	...	...	...
1099	-121.90	39.59	20.0	1465.0	278.0	745.0
18898	-122.25	38.11	49.0	2365.0	504.0	1131.0
11798	-121.22	38.92	19.0	2531.0	461.0	1206.0
6637	-118.14	34.16	39.0	2776.0	840.0	2546.0
2575	-124.13	40.80	31.0	2152.0	462.0	1259.0

16512 rows × 9 columns

In [ ]:
```
from catboost import CatBoostRegressor

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
from sklearn.model_selection import KFold, cross_val_score
kfold = KFold(n_splits=10, shuffle=True, random_state=0)
def regression_model(model):

    scores = cross_val_score(model, X_train_transformed, y_train, scoring=

    rmse = (-scores)**0.5

    # Return mean score
    return rmse.mean()
```

2. Overview

In [ ]:
```
scores = cross_val_score(CatBoostRegressor(cat_features=[8], silent=True),

rmse_cat = (-scores)**0.5
print(rmse_cat.mean())
```

```
45598.23234795042
```

Hyper parameter tuning for CatBoost

In [ ]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error as MSE
import numpy as np
```

In [ ]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=0)
def grid_search(params):

    grid_reg = GridSearchCV(CatBoostRegressor(cat_features=[8],silent=True

    grid_reg.fit(X_train_cat, y_train)

    best_params = grid_reg.best_params_
    best_score=grid_reg.best_score_
    rmse=np.sqrt(-best_score)
    print("Best params:", best_params)
    print("Best score:",rmse.round(2))
```

In [ ]:
```
def randomized_search(params):

    rand_reg = RandomizedSearchCV(CatBoostRegressor(cat_features=[8], sile
                                  cv=kfold, n_jobs=-1)

    rand_reg.fit(X_train_cat, y_train)

    best_model = rand_reg.best_estimator_
    best_params = rand_reg.best_params_
    print("Best params:", best_params)
    best_score = np.sqrt(-rand_reg.best_score_)
    print("Training score: {:.3f}".format(best_score))

    return best_model
```

In [ ]:
```
def model_evaluation(model):
    y_train_pred = model.predict(X_train_cat)
    y_pred = model.predict(X_test_cat)

    train_rmse = np.sqrt(MSE(y_train, y_train_pred))
    test_rmse = np.sqrt(MSE(y_test, y_pred))

    print('Train rmse:', train_rmse)
    print('Test rmse:', test_rmse)
```

In [ ]:
```
grid_search(params={'max_depth': [3, 5, 7, 8, None]})
```

```
Best params: {'max_depth': 8}
Best score: 45137.97
```

In [ ]:
```
grid_search(params={'n_estimators': [200, 300, 500, 700, 1000]})
```

```
Best params: {'n_estimators': 1000}
Best score: 45623.43
```

In [ ]:
```
grid_search(params={'min_child_samples': range(1,6)})
```

```
Best params: {'min_child_samples': 1}
Best score: 45623.43
```

In [ ]:
```
grid_search(params={'learning_rate':[0.01, 0.03, 0.05, 0.1, 0.2]})
```

```
Best params: {'learning_rate': 0.1}
Best score: 45026.66
```

In [ ]:
```
grid_search(params={'subsample':[0.5, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'subsample': 0.8}
Best score: 45623.43
```

In [ ]:
```
grid_search(params={'colsample_bylevel':[0.5, 0.7, 0.8, 0.9, 1]})
```

```
Best params: {'colsample_bylevel': 1}
Best score: 45623.43
```

In [ ]:
```
cat_best = randomized_search(params={'max_depth': [7, 8, 9], 'n_estimators
                           'min_child_samples': range(1,4), 'learning_rate'
                           'subsample': [0.75, 0.8, 0.85], 'colsample_byleve
```

```
Best params: {'subsample': 0.8, 'n_estimators': 1500, 'min_child_samples':
2, 'max_depth': 8, 'learning_rate': 0.1, 'colsample_bylevel': 1}
Training score: 44531.676
```

In [ ]:
```
model_evaluation(cat_best)
```

```
Train rmse: 21550.328160137928
Test rmse: 44765.48360141058
```

In [ ]:
```
sorted_idx=np.argsort(cat_best.feature_importances_)
sorted_col=X_train_cat.columns[sorted_idx]
sorted_imp=cat_best.feature_importances_[sorted_idx]
feature_dict=dict(zip(sorted_col,sorted_imp))
imp_cat=list(feature_dict.items())[::-1]
print(imp_cat)
```

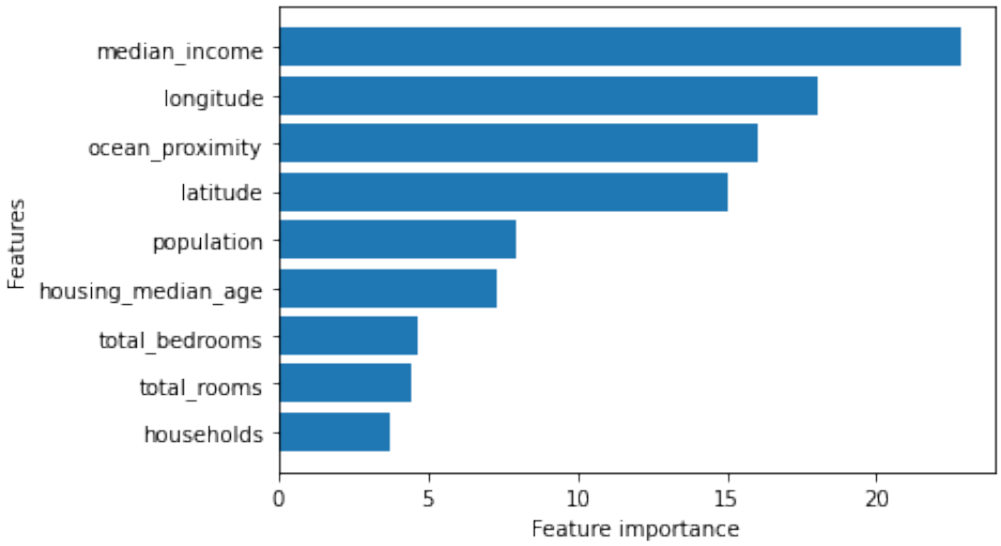
```
[('median_income', 22.813658111400798), ('longitude', 18.00594415959237), (
'ocean_proximity', 15.99740601288843), ('latitude', 15.050464273341351), ('
population', 7.976313721759035), ('housing_median_age', 7.333526502443315),
('total_bedrooms', 4.654168091880431), ('total_rooms', 4.448569617959448),
('households', 3.719949508734845)]
```

In [ ]:

```
features=sorted_col[-10:]
feature_imp=sorted_imp[-10:]
print(sorted_imp)
print(feature_imp)
import matplotlib.pyplot as plt

plt.barh(features,feature_imp)
plt.xlabel('Feature importance')
plt.ylabel('Features')
plt.show()
```

```
[ 3.71994951  4.44856962  4.65416809  7.3335265   7.97631372 15.05046427
 15.99740601 18.00594416 22.81365811]
[ 3.71994951  4.44856962  4.65416809  7.3335265   7.97631372 15.05046427
 15.99740601 18.00594416 22.81365811]
```



In [ ]:

```
import numpy as np

def partial_cat_dependency(model, X, features, f_id):
    X_temp = X.copy()
    grid = np.unique(X_temp.iloc[:, f_id])
    y_pred = np.zeros(len(grid))

    for i, val in enumerate(grid):
        X_temp.iloc[:, f_id] = val
        y_pred[i] = model.predict(X_temp.iloc[:, :features]).mean()

    return grid, y_pred
```

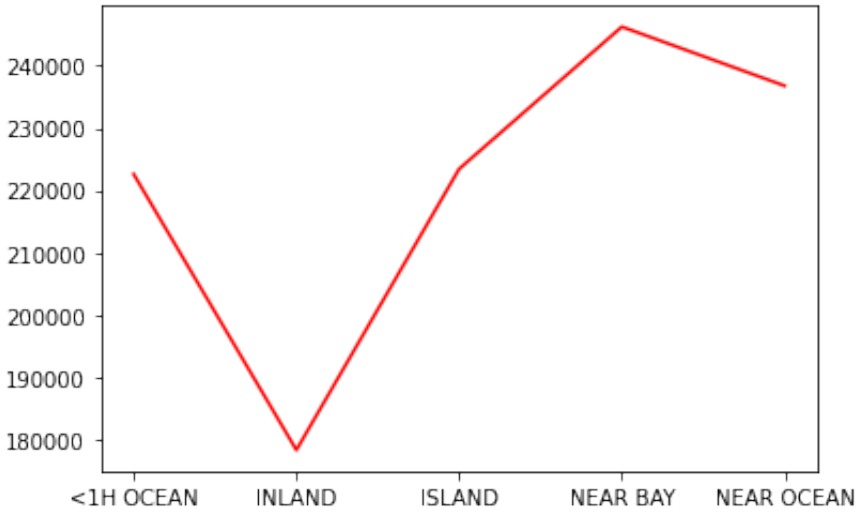
In [ ]:
```
features = X_train_cat.shape[1]

f_id = 8

grid, y_pred = partial_cat_dependency(cat_best, X_train_cat, features, f_i
```

In [ ]:
```
import matplotlib.pyplot as plt

plt.plot(grid, y_pred, '-', color = 'red')
plt.show()
```



Automation

In [74]:
```
from sklearn.pipeline import Pipeline
cat_pipeline = Pipeline([('null_imputer', NullValueImputer()),
                         ('cat',CatBoostRegressor(cat_features=[8], silent=
                                                  n_estimators=1500, min_cl
                                                  learning_rate=0.1, colsar
```

In [75]:
```
cat_pipeline.fit(X_train,y_train)
y_pred_cat=cat_pipeline.predict(X_test_cat)
rmse_cat = MSE(y_test, y_pred_cat)**0.5
print(rmse_cat)
```

```
44765.48360141058
```