Problem Statement:

We have raw CSV downloaded from kaggle https://www.kaggle.com/amitabhajoy/bengaluru-house-price-data

We want to create a website that can give price prediction on given features that uses our pre-trained model

Steps to follow:

1: Explore the data

We will use .head, .unique,.share methods of pandas to explore our data.

2: We will clean our data

We will drop columns that we do not need to train our train, domain knowledge is used for that. We will check null values and drop them

Now we have a problem. No of beds has multiple unique values like, no of beds, bds, bhk. So we will use token (" ")empty space before the no of bedrooms to uniquely indentify no of beds in our custom column

Square feet column also has a range, we want single value so, we will add that range and devide it by 2.

We will use **convert_sqft_to_num()**function for that

We will add a custom column known as price per square feet.

Using simplcity sake, we will put all registered apartments locations with less than 10 mentioning under other category. We will do one hot encoding later on, it will help us with having fewer dummy locations

We do outlier removal according to our business domain knowledge
We use standard deviation

We are also removing properties with same location but 2 bedroom apartment price is more than 3 bedroom apartment. We also remove apartments whose 2 bedroom mean square price is less then of apartment with 1 mean square foot. We also remove outlier with unusual no of bathrooms as compared to no of bedrooms

3: Make dummy values using one hot encoding and concatenate those dummy values with our clean dataframe

4: We use train_test_split with K-validation, we divide our dataset further into 5 datasets

5: We train our data on multiple models using GridSearchCV and find the best model

6: We test our model

7: We save our model

In [12]

```
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))
```

We have issue that in some places Data frame bhk(bedroom) column has 2 bedroom and 2 bhk values. To make it consistent to bhk. We are consistant with bhk format. We are taking the value of bedrooms from dataframe using (“ ”) space as a token as taking value before it as our value of no of bedrooms

In [13]

```
def is_float(x):
    try:
        float(x)
    except:
        return False
    return True
```

In[15]

```
df3[~df3['total_sqft'].apply(is_float)].head(10)
```

We are trying to convert total_sqft value into float by using float(x) and catching those values which we are unable to convert in except block. We are using negation sign **~** to invert the functionality of this code. In other words this code will return us the data points for which this conversion is not possible

In[20}

```
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
```

We are multiplying the price by 100000 because the price given at the columns are in lacks and we want to calculate price per square feet

In[22]

```
df5.location = df5.location.apply(lambda x: x.strip())
location_stats = df5['location'].value_counts(ascending=False)
```

.Strip() method is used to strip a word of any extra trailing or leading extra spaces to avoid errors related to strings later on

.value_counts method is used to count instances of unique values in locations column. (ascending=false) so. We will get values in decending. Which means value with most instances will be at the top and value with lowest instances will be at the bottom

In[27]

```
location_stats_less_than_10 = location_stats[location_stats<=10]
```

We are creating a column 'location_stats_less_than_10' so that we can know which areas have less than 10 listings of property. We will mark them as other locations so later when we are doing hardencoding. We have less dummy variables.

In[29]

```
df5.location = df5.location.apply(lambda x: 'other' if x in
location_stats_less_than_10 else x)
```

We marked those locations with less than 10 listings as others here

In[31]

```
df5[df5.total_sqft/df5.bhk<300].head()
```

According to our project domain knowledge, we want to keep minimum apartment square feet dimensions to 300 square feet per apartment

In[32]

```
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
```

Creating new data frame with condition that only square feet area apartments are selected who have equal to or greater 300 square feet area for bed

In[35]

```
def remove_pps_outliers(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) &
(subdf.price_per_sqft<=(m+st))]
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df7 = remove_pps_outliers(df6)
df7.shape
```

Outlier removal using standard deviation. We are using one standard deviation for this

key is used as a variable name to represent the unique value in the 'location' column that is currently being processed.

In[36]

```
def plot_scatter_chart(df,location):
    bhk2 = df[(df.location==location) & (df.bhk==2)]
    bhk3 = df[(df.location==location) & (df.bhk==3)]
    matplotlib.rcParams['figure.figsize'] = (15,10)
    plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 BHK', s=50)
    plt.scatter(bhk3.total_sqft,bhk3.price,marker='+', color='green',label='3
BHK', s=50)
    plt.xlabel("Total Square Feet Area")
    plt.ylabel("Price (Lakh Indian Rupees)")
    plt.title(location)
    plt.legend()

plot_scatter_chart(df7,"Rajaji Nagar")
```

Plotting graph for 2 bed and 3 bed apartments. In the given code, matplotlib.rcParams is used to modify the default parameters of Matplotlib's plotting system. Specifically, the line

In[38]

```
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
```

```
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                'std': np.std(bhk_df.price_per_sqft),
                'count': bhk_df.shape[0]
            }
        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk-1)
            if stats and stats['count']>5:
                exclude_indices = np.append(exclude_indices,
bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.values)
    return df.drop(exclude_indices,axis='index')
df8 = remove_bhk_outliers(df7)
# df8 = df7.copy()
df8.shape
```

We are doing mean std and count for the dataframe bhk column and when we are doing bhk for 2 beds. We are clearning 1 bhk beds of more price than apartments with 2 bhk

In[50]

Again try to understand this line

Grid search cv:

Grid search algorithm is a hyperparameter tuning technique used in machine learning to find the optimal combination of hyperparameters for a given model. Hyperparameters are the adjustable parameters of a model that are not learned from the data, but rather set by the user prior to training.

Grid search algorithm works by creating a grid of all possible combinations of hyperparameters and then evaluating the model performance for each combination using cross-validation. The performance metric used can vary depending on the problem being solved, but commonly used metrics include accuracy, precision, recall, and F1 score.

In[60]

```
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor

def find_best_model_using_gridsearchcv(X,y):
    algos = {
```

```
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [1,2],
                'selection': ['random', 'cyclic']
            }
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv,
return_train_score=False)
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores,columns=['model','best_score','best_params'])

find_best_model_using_gridsearchcv(X,y)
```

This line of code is performing a grid search using cross-validation to find the best set of hyperparameters for a machine learning model.

GridSearchCV is a function from the sklearn.model_selection module that performs an exhaustive search over a specified parameter grid, using cross-validation to evaluate the performance of each combination of hyperparameters. config['model'] is the machine learning model object or pipeline that will be used for the search.

config['params'] is a dictionary of hyperparameters and their possible values to be searched over. cv is the number of cross-validation folds to use. return_train_score=False specifies that the function should not return the training scores for each fold.