SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19TEL7014 **DATE:**

COURSE NAME: Machine Learning **CLASS:** Final Year B.Tech

EXPERIMENT NO. 1

CO1 Solve real-world problems using suitable machine learning techniques.

TITLE: Data Preprocessing

AIM / OBJECTIVE:

To perform data preprocessing in terms of handling, missing data, removing outliers, eliminating duplicate rows and modifying the datatype, etc.

DESCRIPTION OF EXPERIMENT:

Python is an easy-to-learn programming language, which makes it the most preferred choice for beginners in Data Science, Data Analytics, and Machine Learning. It also has a great community of online learners and excellent data-centric libraries. With so much data being generated, it becomes important that the data we use for Data Science applications like Machine Learning and Predictive Modeling is clean. But what do we mean by clean data? And what makes data dirty in the first place? Dirty data simply means data that is erroneous. Duplicacy of records, incomplete or outdated data, and improper parsing can make data dirty. This data needs to be cleaned. Data cleaning (or data cleansing) refers to the process of "cleaning" this dirty data, by identifying errors in the data and then rectifying them. Data cleaning is an important step in and Machine Learning project, and we will cover some basic data cleaning techniques (in Python) in this article.

Cleaning Data in Python

We will now separate the numeric columns from the categorical columns.

Missing values

We will start by calculating the percentage of values missing in each column, and then storing this information in a DataFrame.

Drop observations

One way could be to drop those observations that contain any null value in them for any of the columns. This will work when the percentage of missing values in each column is very less.

Remove columns (features)

Another way to tackle missing values in a dataset would be to drop those columns or features that have a significant percentage of values missing.

Impute missing values

There is still missing data left in our dataset. We will now impute the missing values in each numerical column with the median value of that column.

Outliers

An outlier is an unusual observation that lies away from the majority of the data. Outliers can affect the performance of a Machine Learning model significantly.

Duplicate records

Data can sometimes contain duplicate values. It is important to remove duplicate records from your dataset before you proceed with any Machine Learning project. In our data, since the ID column is a unique identifier, we will drop duplicate records by considering all but the ID column.

Fixing data type

Often in the dataset, values are not stored in the correct data type. This can create a problem in later stages, and we may not get the desired output or may get errors while execution.

OBSERVATIONS:

In [1]:

```
#transformation
```

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:

```
df=pd.read_csv('.\\dataset\\train.csv')
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30471 entries, 0 to 30470
Columns: 292 entries, id to price_doc
dtypes: float64(119), int64(157), object(16)
memory usage: 67.9+ MB
```

In [5]:

```
df_numeric = df.select_dtypes(include=[np.number])
df_non_numeric = df.select_dtypes(exclude=[np.number])
```

In [7]:

```
#missing values
df.isnull().sum()
```

Out[7]:

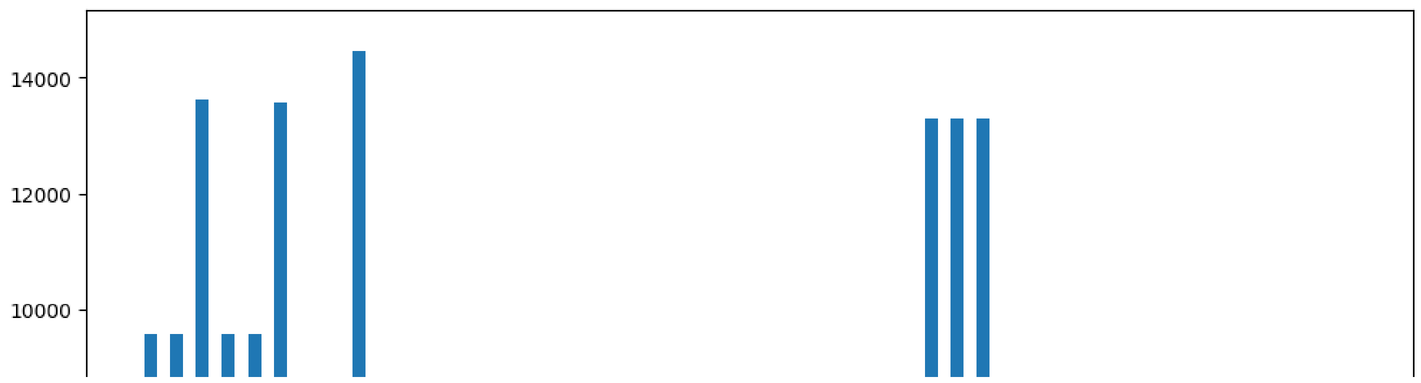
```
id                      0
timestamp               0
full_sq                 0
life_sq              6383
floor                 167
                     ...
mosque_count_5000       0
leisure_count_5000      0
sport_count_5000        0
market_count_5000       0
price_doc               0
Length: 292, dtype: int64
```

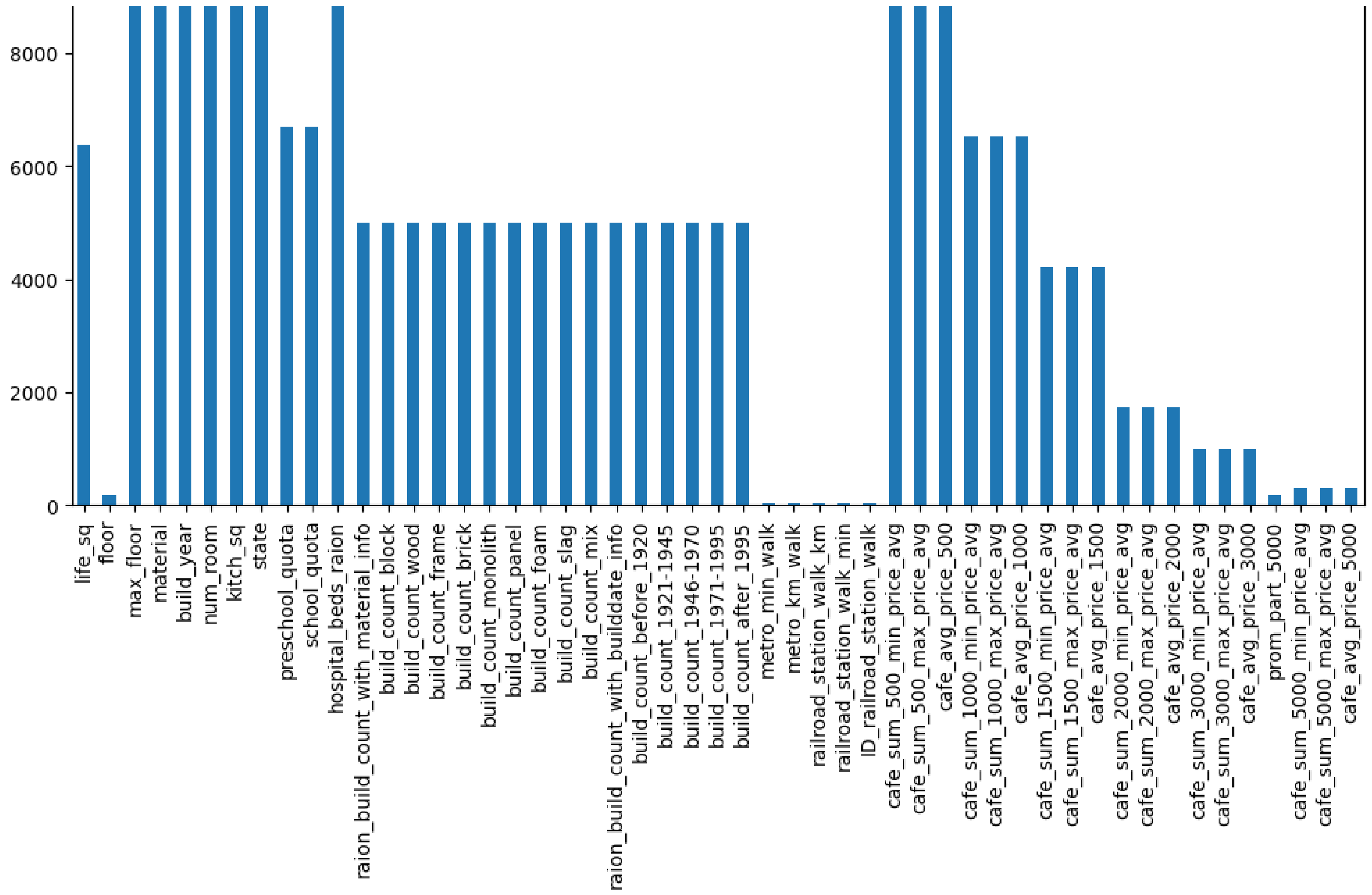
In [12]:

```
df.isnull().sum()[df.isnull().sum()>0].plot(kind='bar', figsize=(12,8))
```

Out[12]:

```
<Axes: >
```





In [22]:

```
#handel missing values
#1. drop

missing_values = df.isnull().sum()
s = list(missing_values[(missing_values > 0) & (missing_values < 0.5 * len(df) / 100)].i
ndex)
print(s)
no_na_df=df.dropna(subset=s)
no_na_df.info()
```

```
['metro_min_walk', 'metro_km_walk', 'railroad_station_walk_km', 'railroad_station_walk_mi
n', 'ID_railroad_station_walk']
<class 'pandas.core.frame.DataFrame'>
Index: 30446 entries, 0 to 30470
Columns: 292 entries, id to price_doc
dtypes: float64(119), int64(157), object(16)
memory usage: 68.1+ MB
```

In [27]:

```
#2. drop col

s = list(missing_values[missing_values > 40 * len(df) / 100].index)

col_dropped_df=no_na_df.drop(columns=s)
print(len(col_dropped_df.columns))
```

```
286
```

In [28]:

```
#3. default values

df_numeric = col_dropped_df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
for col in numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:
        med = col_dropped_df[col].median() #impute with the median
        col_dropped_df[col] = col_dropped_df[col].fillna(med)
```

In [29]:

```
df_non_numeric = col_dropped_df.select_dtypes(exclude=[np.number])
numeric_cols = df_non_numeric.columns.values
for col in numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:
        med = col_dropped_df[col].mode() #impute with the mode
        col_dropped_df[col] = col_dropped_df[col].fillna(med)
```

In [31]:

```
col_dropped_df.isnull().sum().sum()
```

Out[31]:

```
0
```

In [41]:

```
#outliers
print(col_dropped_df.columns)

col_dropped_df.life_sq.describe()
```

```
Index(['id', 'timestamp', 'full_sq', 'life_sq', 'floor', 'max_floor',
       'material', 'num_room', 'kitch_sq', 'product_type',
       ...
       'cafe_count_5000_price_2500', 'cafe_count_5000_price_4000',
       'cafe_count_5000_price_high', 'big_church_count_5000',
       'church_count_5000', 'mosque_count_5000', 'leisure_count_5000',
       'sport_count_5000', 'market_count_5000', 'price_doc'],
      dtype='object', length=286)
```

Out[41]:

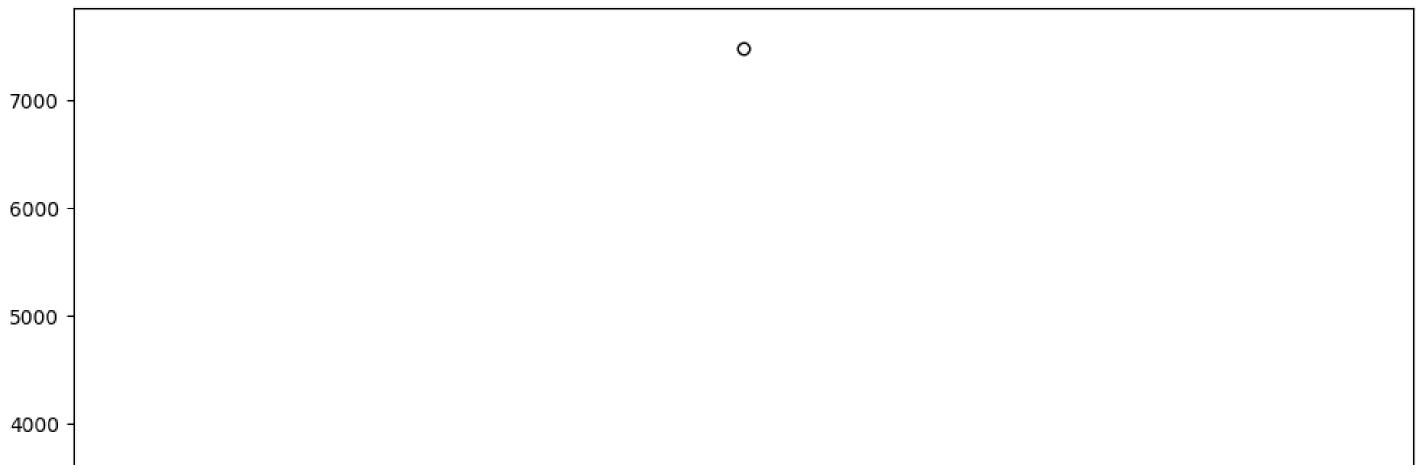
```
count    30446.000000
mean        33.482658
std         46.538609
min          0.000000
25%         22.000000
50%         30.000000
75%         38.000000
max       7478.000000
Name: life_sq, dtype: float64
```

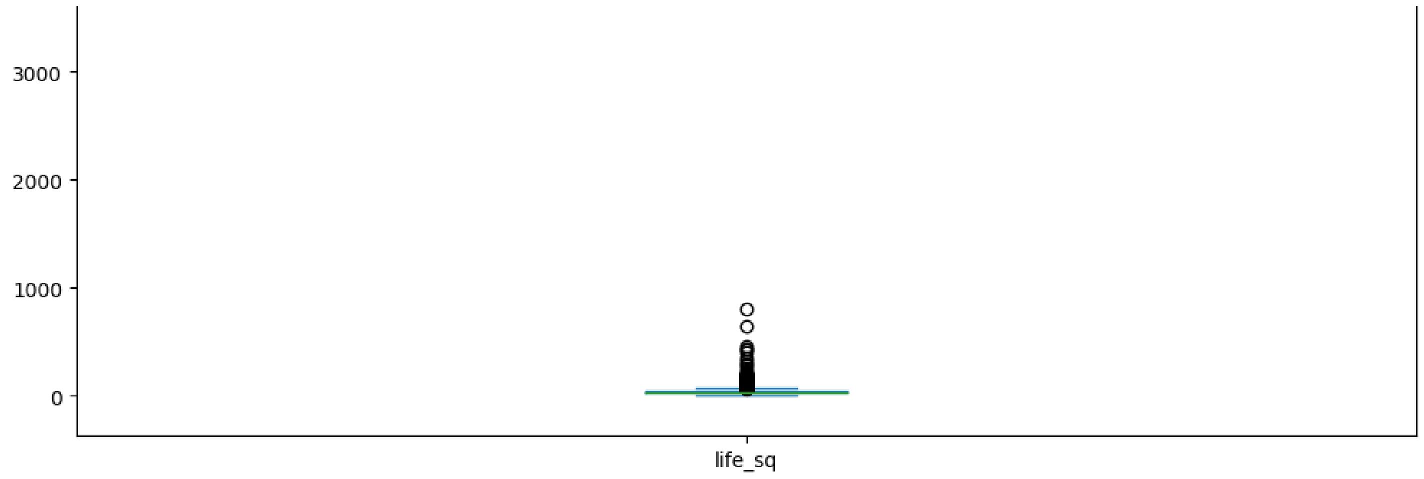
In [44]:

```
col_dropped_df.life_sq.plot(kind='box', figsize=(12, 8))
```

Out[44]:

```
<Axes: >
```





In [43]:

```
col_dropped_df = col_dropped_df.loc[df.life_sq < 7478]
```

In [33]:

```
#dupes

col_dropped_df.drop_duplicates(inplace=True)
col_dropped_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 30446 entries, 0 to 30470
Columns: 286 entries, id to price_doc
dtypes: float64(113), int64(157), object(16)
memory usage: 66.7+ MB
```

In [34]:

```
#fix data types

col_dropped_df['timestamp'] = pd.to_datetime(col_dropped_df.timestamp, format='%Y-%m-%d'
)
```

CONCLUSION:

Data cleaning and preprocessing are fundamental steps in the data science and machine learning pipeline. By using pandas and the techniques covered in this experiment, we can ensure that our data is clean, consistent, and ready for further analysis and modeling. These skills are essential for anyone working with real-world datasets, as they enable us to uncover meaningful insights and build robust data-driven solutions.

In conclusion, our experiment on data cleaning and preprocessing using Python's pandas library has provided us with valuable insights and techniques for preparing raw data for analysis and modeling. We employed several important data cleaning and preprocessing methods, including df.dropna(), df.drop_duplicates(), handling outliers, and managing data types.