```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor

import warnings
warnings.filterwarnings('ignore')



df = pd.read_csv('/content/Zillow.csv')
df.head()
```

	price	unformattedPrice	address	addressStreet	addressCity	addressState	addressZipcode	beds	baths	area	latitude	longitu
0	445000	445000	8916 Mountain Shadows Cv APT B, Austin, TX 78735	8916 Mountain Shadows Cv APT B	Austin	TX	78735	3.0	3.0	1802.0	30.269207	-97.862
1	1995000	1995000	1701 Alguno Rd, Austin, TX 78757	1701 Alguno Rd	Austin	TX	78757	4.0	4.0	3443.0	30.333755	-97.734
2	929900	929900	1800 Kinney Ave, Austin, TX 78704	1800 Kinney Ave	Austin	TX	78704	2.0	2.0	1318.0	30.252070	-97.768
3	6495000	6495000	2407 Pemberton Pl, Austin, TX 78703	2407 Pemberton Pl	Austin	TX	78703	5.0	6.0	5000.0	30.290514	-97.753
4	365000	365000	11701 Lansdowne Rd, Austin, TX 78754	11701 Lansdowne Rd	Austin	TX	78754	4.0	2.0	2127.0	30.355240	-97.612

```
df.shape
```

```
(800, 20)
```

```
to_remove = []
for col in df.columns:

    # Removing columns having only one value.
    if df[col].nunique() == 1:
        to_remove.append(col)

    # Removing columns with more than 90% of the
    # rows as null values.
    elif (df[col].isnull()).mean() > 0.60:
        to_remove.append(col)

print(len(to_remove))
```

```
4
```

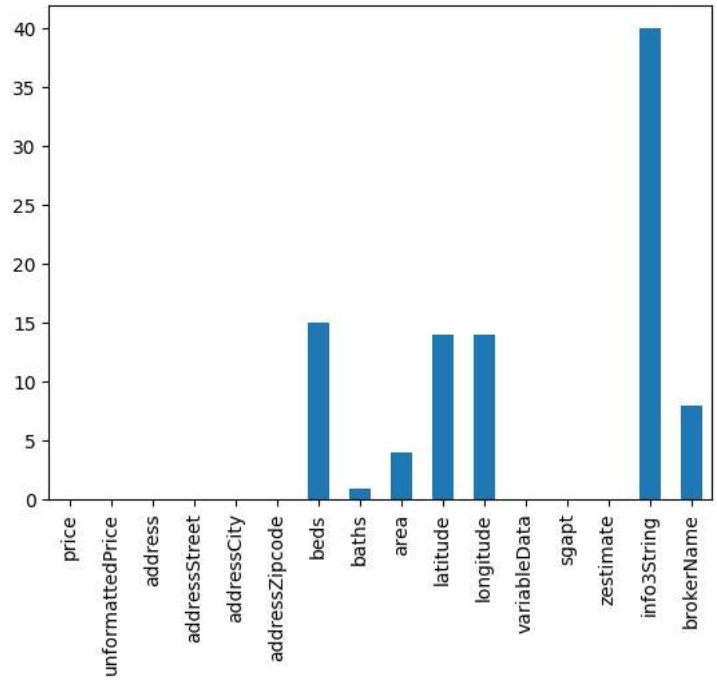
```
df.drop(to_remove,
        axis=1,
```

```
        inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   price             800 non-null    int64  
 1   unformattedPrice  800 non-null    int64  
 2   address           800 non-null    object 
 3   addressStreet     800 non-null    object 
 4   addressCity       800 non-null    object 
 5   addressZipcode    800 non-null    int64  
 6   beds              785 non-null    float64
 7   baths             799 non-null    float64
 8   area              796 non-null    float64
 9   latitude          786 non-null    float64
 10  longitude         786 non-null    float64
 11  variableData      800 non-null    object 
 12  sgapt             800 non-null    object 
 13  zestimate         800 non-null    int64  
 14  info3String       760 non-null    object 
 15  brokerName        792 non-null    object 
dtypes: float64(5), int64(4), object(7)
memory usage: 100.1+ KB
```

```
df.isnull().sum().plot.bar()
plt.show()
```




```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].fillna(df[col].mode()[0])
    elif df[col].dtype == np.number:
        df[col] = df[col].fillna(df[col].mean())
```

```
df.isnull().sum().sum()
```

```
0
```

```
ints, objects, floats = [], [], []
```

```
for col in df.columns:
    if df[col].dtype == float:
        floats.append(col)
    elif df[col].dtype == int:
        ints.append(col)
    else:
        objects.append(col)

len(ints), len(floats), len(objects)
```

```
(4, 5, 7)
```

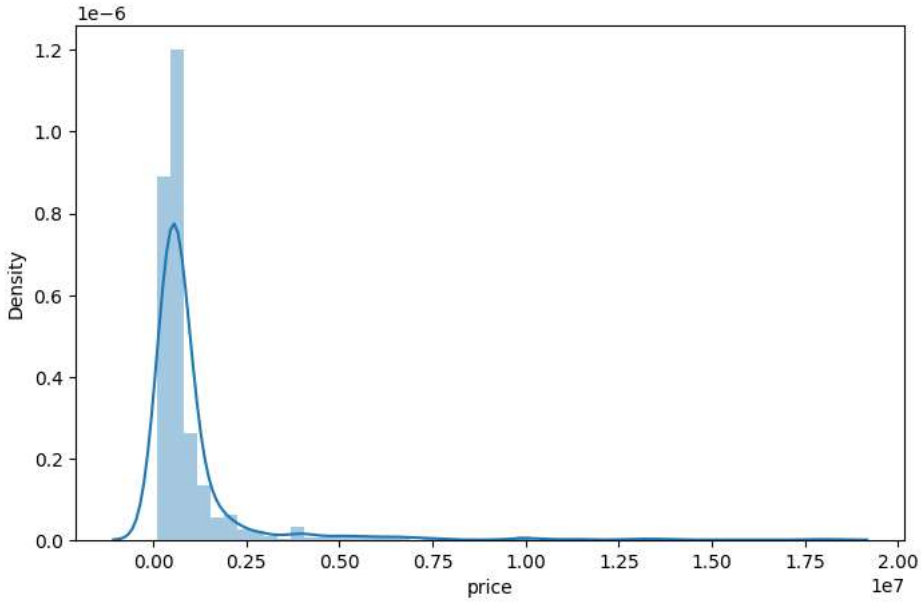
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   price             800 non-null    int64  
 1   unformattedPrice  800 non-null    int64  
 2   address           800 non-null    object 
 3   addressStreet     800 non-null    object 
 4   addressCity       800 non-null    object 
 5   addressZipcode    800 non-null    int64  
 6   beds              800 non-null    float64
 7   baths             800 non-null    float64
 8   area              800 non-null    float64
 9   latitude          800 non-null    float64
 10  longitude         800 non-null    float64
 11  variableData      800 non-null    object 
 12  sgapt             800 non-null    object 
 13  zestimate         800 non-null    int64  
 14  info3String       800 non-null    object 
 15  brokerName        800 non-null    object 
dtypes: float64(5), int64(4), object(7)
memory usage: 100.1+ KB
```

```
for col in objects:
    print(col, ' -> ', df[col].nunique())
    print(df[col].unique())
    print()
```

```
 'Kes and Kanch LLC'  'Horizon Realty'  'Nobles Realty Group, LLC'
'Harrison-Pearson Assoc. Inc' 'JBGoodwin REALTORS LT'
'Epperson Realty Group LLC' 'James Goldrick' 'Meador Realtors'
'Todd Hower Realty LLC' 'Crawford Realty Inc' 'Select Austin Real Estate'
'TOWERS.net' 'Brookfield Residential' 'Kevin Kent Real Estate'
'Hendricks Real Estate' 'Miguel Barrutia, BROKER'
'Linda Welsh Realty GRP' 'NICE MOVE REALTY' 'Texas Dream Realtors'
'Devora Realty' 'NB Regal Real Estate, LLC' 'Kopa Real Estate'
'Minto Real Estate' 'Smart Source Realty' 'Anders Pierce Realty, LLC'
'RE/MAX Austin Skyline' 'Stanberry REALTORS' 'Sherman & Co., Realtors'
"M.E. 'Gene' Johnson REALTY" 'Fourth Dimension Group'
'Solomon Group Real Estate' 'McAllister & Associates'
'White Label Realty' 'Ensor & Co., Realtors' 'Dochen, REALTORS'
'The One Realty' 'MidCentury Realty' 'Rudder Realty, Inc.'
'BluEarth Realty LLC' 'Robert Griffice' 'AustinRealEstate.com'
'David Brodsky Properties' 'Uptown Realty LLC' 'Agency Texas Inc'
'InTown Builders' 'McNabb & Company Real Estate S' 'Green City Realty'
'LindenDwell Real Estate' 'The Agency Austin' 'Pearson Properties'
'Nest Easy Realty' 'Upside Realty' 'Urban to Suburban Realty'
'All Access Austin' 'Tate Property' 'Bedrock Realty'
'Ely Properties, Inc.']
```

```
plt.figure(figsize=(8, 5))
sb.distplot(df['price'])
plt.show()
```




```
plt.figure(figsize=(8, 5))
sb.boxplot(df['price'])
plt.show()
```

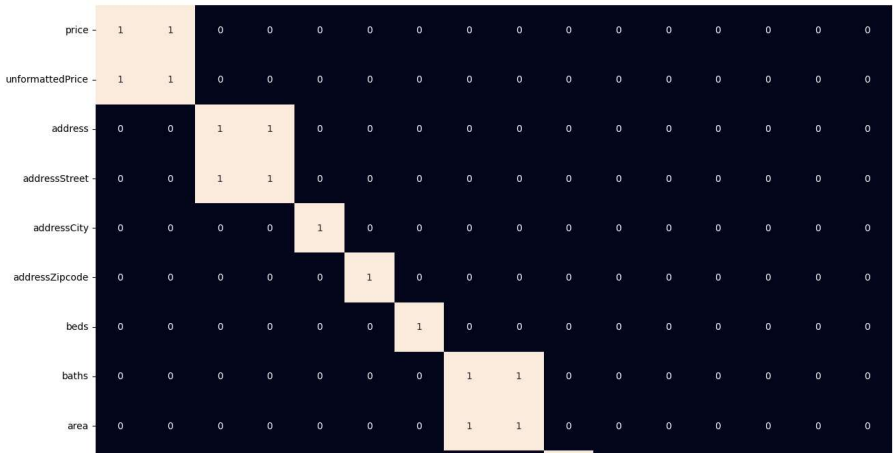

```
"""print('Shape of the dataframe before removal of outliers', df.shape)
df = df[(df['price'] > -1) & (df['price'] < 1)]
print('Shape of the dataframe after removal of outliers ', df.shape) """
#outliers where removing entire datasets so prefer to go by IQR method to remove outliers
```

```
'print('Shape of the dataframe before removal of outliers', df.shape) \ndf = df[(df['pr
ice'] > -1) & (df['price'] < 1)] \nprint('Shape of the dataframe after removal of outli
```

```
for col in objects:
  le = LabelEncoder()
  df[col] = le.fit_transform(df[col])
```

+ Code + Text

```
from ssl import ALERT_DESCRIPTION_BAD_CERTIFICATE_STATUS_RESPONSE
plt.figure(figsize=(15, 15))
sb.heatmap(df.corr() > 0.8,
    annot=True,
    cbar=False)
plt.show()
```




```
to_remove = ['unformattedPrice', 'addressStreet', 'baths']
df.drop(to_remove, axis=1, inplace=True)
```

```
df
```

	price	address	addressCity	addressZipcode	beds	area	latitude	longitude	variableData	sgapt	zestimate	info3String	bro
0	445000	746	0	78735	3.0	1802.0	30.269207	-97.862060	183	0	1189900	1	
1	1995000	242	0	78757	4.0	3443.0	30.333755	-97.734140	18	1	2154600	1	
2	929900	257	0	78704	2.0	1318.0	30.252070	-97.768600	78	0	1048700	1	
3	6495000	331	0	78703	5.0	5000.0	30.290514	-97.753610	325	0	6553400	1	
4	365000	98	0	78754	4.0	2127.0	30.355240	-97.612920	0	0	421000	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
795	9990000	118	0	78733	5.0	8357.0	30.326190	-97.844025	58	0	0	1	
796	11250800	415	0	78746	5.0	7947.0	30.293275	-97.764621	55	1	0	1	
797	13000000	155	0	78703	5.0	6528.0	30.299100	-97.749725	37	0	0	1	
798	13500000	693	0	78735	5.0	8549.0	30.271202	-97.867645	95	0	0	1	
799	18000000	710	0	78701	3.0	14025.0	30.270054	-97.741615	2	0	0	1	

800 rows × 13 columns

```
features = df.drop(['price'], axis=1)
target = df['price'].values

X_train, X_val,Y_train, Y_val = train_test_split(features, target,
                                  test_size=0.1,
                                  random_state=22)
X_train.shape, X_val.shape
```

```
((720, 12), (80, 12))
```

```
# Normalizing the features for stable and fast training.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

```
from sklearn.metrics import mean_absolute_error as mae
models = [LinearRegression(), XGBRegressor(),
    Lasso(), RandomForestRegressor(), Ridge()]

for i in range(5):
  models[i].fit(X_train, Y_train)

  print(f'{models[i]} : ')

  train_preds = models[i].predict(X_train)
  print('Training Error : ', mae(Y_train, train_preds))
```

```
val_preds = models[i].predict(X_val)
print('Validation Error : ', mae(Y_val, val_preds))
print()
```

```
LinearRegression() :
Training Error :  504373.5411832306
Validation Error :  399103.62977603

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...) :
Training Error :  8811.670128038195
Validation Error :  201277.8509033203

Lasso() :
Training Error :  504372.84216277895
Validation Error :  399102.74087378226

RandomForestRegressor() :
Training Error :  122632.76930555556
Validation Error :  189605.45512499998

Ridge() :
Training Error :  503799.7421691335
Validation Error :  398476.37660499534
```