REGRESSION MODEL FOR HOUSE PRICE PREDICTION

Dataset was obtain from Kaggle platform where the price of a house was predicted based on the following variables:
number of bedrooms,number of bathrooms sqft_living size,,sqft_lot size, number of floors,number of waterfront,view,house condition and grade amongst others.
R_squared score was used to measure model accuracy.
Linear Regression model R2_squared score was (77%),Extra Tree Regressor model (89%),CatBoost Regressor(91%) and Gradient Boosting Regressor(88%).
Hence CatBoost Regressor model was the best in predicting house prices for this dataset.

STEPS

1.Loading the dataset
2.Explonatory data analysis
3.Data pre_processing
4.Building and Training the models
6.Fitting the models
7.Model Evaluation

In [1]:

```python
#Importing relevant packages
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
sns.set()
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```python
#loading dataset
data=pd.read_csv('C:/Users/User/Desktop/DATA/house_data.csv')
```

In [3]:

```python
#Head of data
data.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	0	98178	47.
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	1991	98125	47.
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	0	98028	47.
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	0	98136	47.
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	0	98074	47.

5 rows × 21 columns

In [4]:

```python
#shape of data
data.shape
```

Out[4]:

```
(21613, 21)
```

In [5]:

```python
#columns in data
data.columns
```

Out[5]:

```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [6]:

```python
#descriptive statistics
data.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
id	21613.0	4.580302e+09	2.876566e+09	1.000102e+06	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	21613.0	5.401822e+05	3.673622e+05	7.500000e+04	3.219500e+05	4.500000e+05	6.450000e+05	7.700000e+06
bedrooms	21613.0	3.370842e+00	9.300618e-01	0.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
bathrooms	21613.0	2.114757e+00	7.701632e-01	0.000000e+00	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
sqft_living	21613.0	2.079900e+03	9.184409e+02	2.900000e+02	1.427000e+03	1.910000e+03	2.550000e+03	1.354000e+04
sqft_lot	21613.0	1.510697e+04	4.142051e+04	5.200000e+02	5.040000e+03	7.618000e+03	1.068800e+04	1.651359e+06
floors	21613.0	1.494309e+00	5.399889e-01	1.000000e+00	1.000000e+00	1.500000e+00	2.000000e+00	3.500000e+00
waterfront	21613.0	7.541757e-03	8.651720e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
view	21613.0	2.343034e-01	7.663176e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
condition	21613.0	3.409430e+00	6.507430e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	5.000000e+00
grade	21613.0	7.656873e+00	1.175459e+00	1.000000e+00	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
sqft_above	21613.0	1.788391e+03	8.280910e+02	2.900000e+02	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03
sqft_basement	21613.0	2.915090e+02	4.425750e+02	0.000000e+00	0.000000e+00	0.000000e+00	5.600000e+02	4.820000e+03
yr_built	21613.0	1.971005e+03	2.937341e+01	1.900000e+03	1.951000e+03	1.975000e+03	1.997000e+03	2.015000e+03
yr_renovated	21613.0	8.440226e+01	4.016792e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.015000e+03
zipcode	21613.0	9.807794e+04	5.350503e+01	9.800100e+04	9.803300e+04	9.806500e+04	9.811800e+04	9.819900e+04
lat	21613.0	4.756005e+01	1.385637e-01	4.715590e+01	4.747100e+01	4.757180e+01	4.767800e+01	4.777760e+01
long	21613.0	-1.222139e+02	1.408283e-01	-1.225190e+02	-1.223280e+02	-1.222300e+02	-1.221250e+02	-1.213150e+02
sqft_living15	21613.0	1.986552e+03	6.853913e+02	3.990000e+02	1.490000e+03	1.840000e+03	2.360000e+03	6.210000e+03
sqft_lot15	21613.0	1.276846e+04	2.730418e+04	6.510000e+02	5.100000e+03	7.620000e+03	1.008300e+04	8.712000e+05

In [7]:

```python
#information on data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [8]:

```
# Convert 'zipcode' to string type
data['zipcode'] = data['zipcode'].astype(str)
```

In [9]:

```
#dtypes on data
data.dtypes
```

Out[9]:

```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode           object
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [10]:

```
#checking null values
data.isnull().sum().sum()
```

Out[10]:

```
0
```

In [11]:

```
data.duplicated().sum()
```

Out[11]:

```
0
```

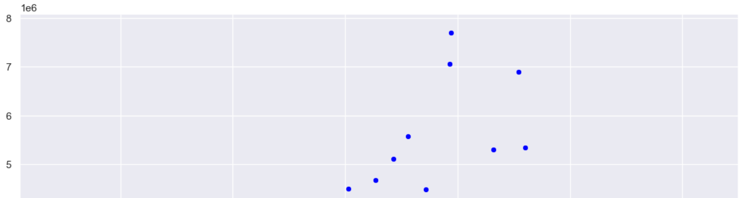
EXPLONATORY DATA ANALYSIS

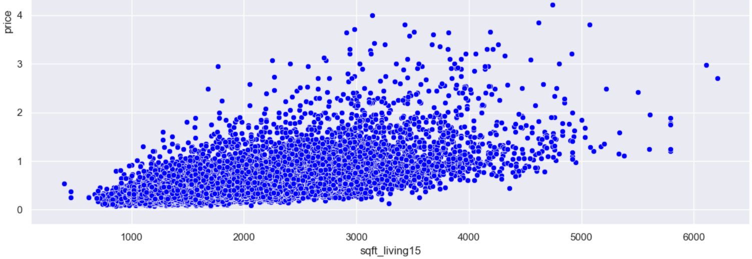
In [12]:

```
plt.figure(figsize=(14,8))
sns.scatterplot(x='sqft_living15',y='price',data=data,color='blue')
```

Out[12]:

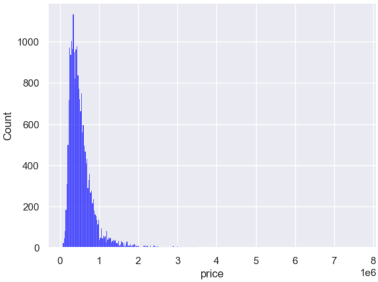
```
<Axes: xlabel='sqft_living15', ylabel='price'>
```





```
In [13]: sns.histplot(x='price',data=data,color='blue')
```

```
Out[13]: <Axes: xlabel='price', ylabel='Count'>
```



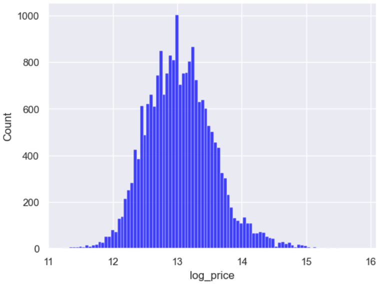
```
In [14]: #tranforming price using np.log
```

```
In [15]: log_price=np.log(data['price'])
```

```
In [16]: data['log_price']=log_price
```

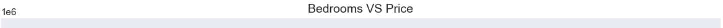
```
In [17]: sns.histplot(x='log_price',data=data,color='blue')
```

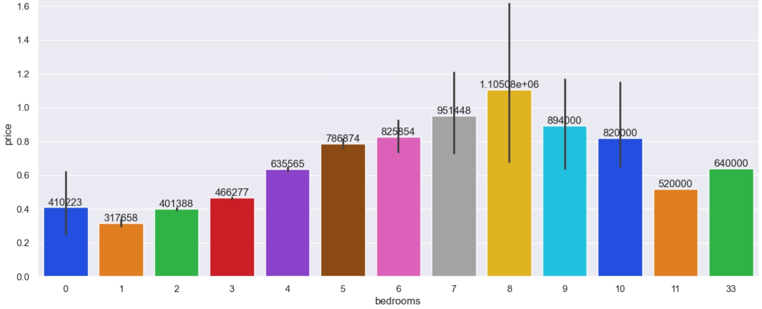
```
Out[17]: <Axes: xlabel='log_price', ylabel='Count'>
```



```
In [18]: #barplot on bedrooms against price
         plt.figure(figsize=(15,6))
         ax=sns.barplot(x=data['bedrooms'], y=data['price'],palette='bright')
         for i in ax.containers:
             ax.bar_label(i,)
         plt.title('Bedrooms VS Price', fontsize=14)
```

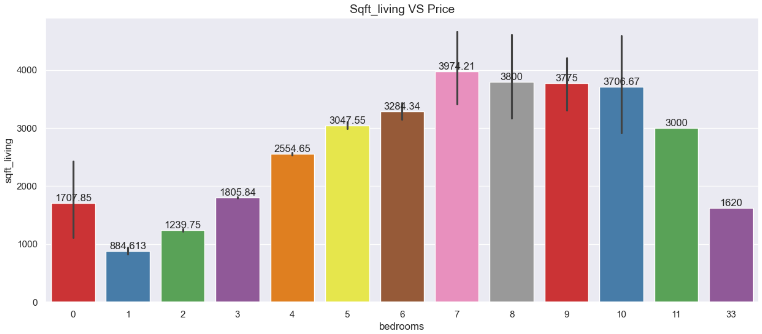
```
Out[18]: Text(0.5, 1.0, 'Bedrooms VS Price')
```





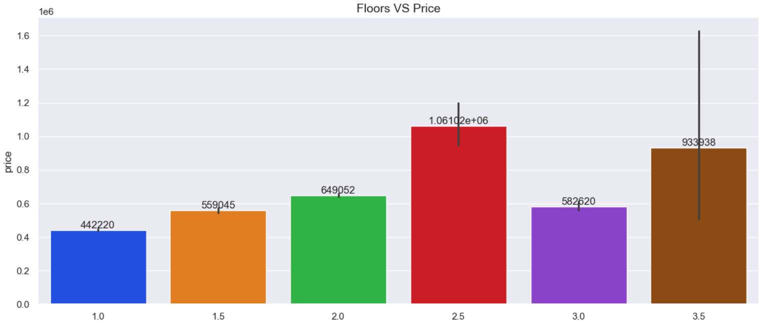
In [19]:
```
#barplot on bedrooms against sqt_living
plt.figure(figsize=(15,6))
ax=sns.barplot(x=data['bedrooms'], y=data['sqft_living'],palette='Set1')
for i in ax.containers:
     ax.bar_label(i,)
plt.title('Sqft_living VS Price', fontsize=14)
```

Out[19]:
```
Text(0.5, 1.0, 'Sqft_living VS Price')
```



In [20]:
```
#barplot on floors against price
plt.figure(figsize=(15,6))
ax=sns.barplot(x=data['floors'], y=data['price'],palette='bright')
for i in ax.containers:
     ax.bar_label(i,)
plt.title('Floors VS Price', fontsize=14)
```

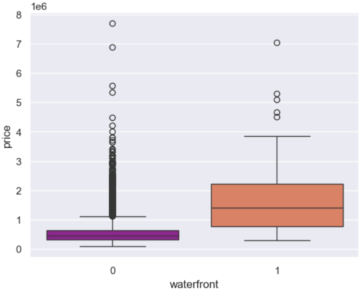
Out[20]:
```
Text(0.5, 1.0, 'Floors VS Price')
```



floors

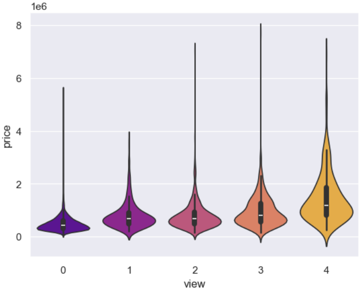
In [21]:
```
#boxplot on waterfrot against price
sns.boxplot(x='waterfront',y='price',data=data,palette='plasma')
```

Out[21]:
```
<Axes: xlabel='waterfront', ylabel='price'>
```



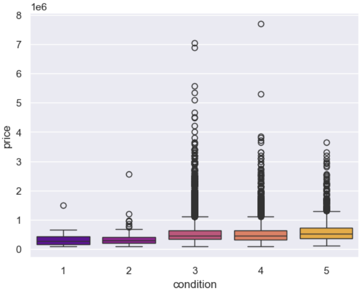
In [22]:
```
#violin plot on view against price
sns.violinplot(x='view',y='price',data=data,palette='plasma')
```

Out[22]:
```
<Axes: xlabel='view', ylabel='price'>
```



In [23]:
```
# boxplot on condition against price
sns.boxplot(x='condition',y='price',data=data,palette='plasma')
```

Out[23]:
```
<Axes: xlabel='condition', ylabel='price'>
```



DATA CLEANING

In [24]:
```
#dropping irrelevant columns
data.drop(columns=['date','id'],axis=1,inplace=True)
```

```
In [25]: data.shape
```

```
Out[25]: (21613, 20)
```

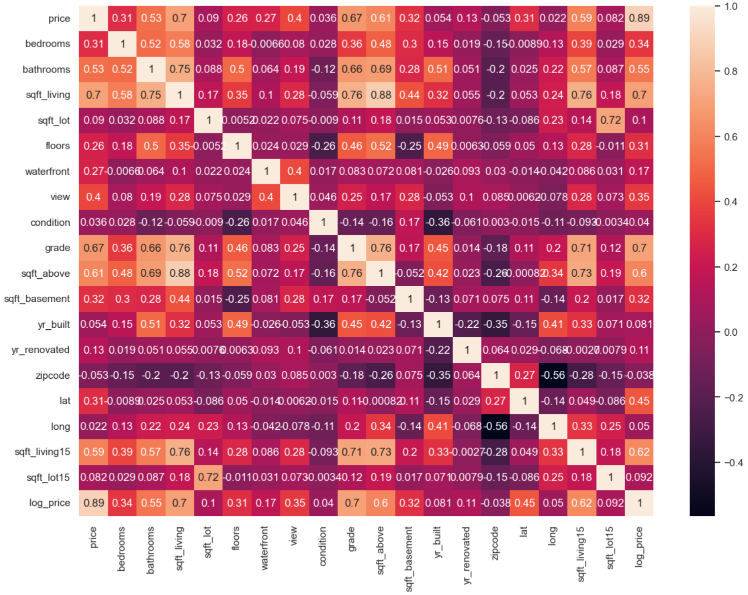
```
In [26]: #correlation
         data.corr()
```

Out[26]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcod
price	1.000000	0.308338	0.525134	0.702044	0.089655	0.256786	0.266331	0.397346	0.036392	0.667463	0.605566	0.323837	0.053982	0.126442	-0.05316
bedrooms	0.308338	1.000000	0.515884	0.576671	0.031703	0.175429	-0.006582	0.079532	0.028472	0.356967	0.477600	0.303093	0.154178	0.018841	-0.15266
bathrooms	0.525134	0.515884	1.000000	0.754665	0.087740	0.500653	0.063744	0.187737	-0.124982	0.664983	0.685342	0.283770	0.506019	0.050739	-0.20386
sqft_living	0.702044	0.576671	0.754665	1.000000	0.172826	0.353949	0.103818	0.284611	-0.058753	0.762704	0.876597	0.435043	0.318049	0.055363	-0.19943
sqft_lot	0.089655	0.031703	0.087740	0.172826	1.000000	-0.005201	0.021604	0.074710	-0.008958	0.113621	0.183512	0.015286	0.053080	0.007644	-0.12957
floors	0.256786	0.175429	0.500653	0.353949	-0.005201	1.000000	0.023698	0.029444	-0.263768	0.458183	0.523885	-0.245705	0.489319	0.006338	-0.05912
waterfront	0.266331	-0.006582	0.063744	0.103818	0.021604	0.023698	1.000000	0.401857	0.016653	0.082775	0.072075	0.080588	-0.026161	0.092885	0.03028
view	0.397346	0.079532	0.187737	0.284611	0.074710	0.029444	0.401857	1.000000	0.045990	0.251321	0.167649	0.276947	-0.053440	0.103917	0.08482
condition	0.036392	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.016653	0.045990	1.000000	-0.144674	-0.158214	0.174105	-0.361417	-0.060618	0.00302
grade	0.667463	0.356967	0.664983	0.762704	0.113621	0.458183	0.082775	0.251321	-0.144674	1.000000	0.755923	0.168392	0.446963	0.014414	-0.18486
sqft_above	0.605566	0.477600	0.685342	0.876597	0.183512	0.523885	0.072075	0.167649	-0.158214	0.755923	1.000000	-0.051943	0.423898	0.023285	-0.26119
sqft_basement	0.323837	0.303093	0.283770	0.435043	0.015286	-0.245705	0.080588	0.276947	0.174105	0.168392	-0.051943	1.000000	-0.133124	0.071323	0.07484
yr_built	0.053982	0.154178	0.506019	0.318049	0.053080	0.489319	-0.026161	-0.053440	-0.361417	0.446963	0.423898	-0.133124	1.000000	-0.224874	-0.34686
yr_renovated	0.126442	0.018841	0.050739	0.055363	0.007644	0.006338	0.092885	0.103917	-0.060618	0.014414	0.023285	0.071323	-0.224874	1.000000	0.06435
zipcode	-0.053168	-0.152668	-0.203866	-0.199430	-0.129574	-0.059121	0.030285	0.084827	0.003026	-0.184862	-0.261190	0.074845	-0.346869	0.064357	1.00000
lat	0.306919	-0.008931	0.024573	0.052529	-0.085683	0.049614	-0.014274	0.006157	-0.014941	0.114084	-0.000816	0.110538	-0.148122	0.029398	0.26704
long	0.021571	0.129473	0.223042	0.240223	0.229521	0.125419	-0.041910	-0.078400	-0.106500	0.198372	0.343803	-0.144765	0.409356	-0.068372	-0.56407
sqft_living15	0.585374	0.391638	0.568634	0.756420	0.144608	0.279885	0.086463	0.280439	-0.092824	0.713202	0.731870	0.200355	0.326229	-0.002673	-0.27903
sqft_lot15	0.082456	0.029244	0.087175	0.183286	0.718557	-0.011269	0.030703	0.072575	-0.003406	0.119248	0.194050	0.017276	0.070958	0.007854	-0.14722
log_price	0.891680	0.343563	0.550819	0.695380	0.099625	0.310556	0.174597	0.346576	0.039574	0.703675	0.601830	0.316998	0.080638	0.114512	-0.03829

```
In [27]: #heatmap on correlation
         plt.figure(figsize=(14,10))
         sns.heatmap(data.corr(),annot=True)
```

```
Out[27]: <Axes: >
```



```
In [28]: #dropping some of the columns having high correlation with others
         data.drop(columns=['price','sqft_above'],axis=1,inplace=True)
```

```
In [29]: data.shape
```

Out[29]:
```
(21613, 18)
```

SPLITTING THE DATA

In [30]:
```python
from sklearn.model_selection import train_test_split
```

In [31]:
```python
#declaring variables
X=data.drop('log_price',axis=1)
y=data['log_price']
```

In [32]:
```python
#train test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [33]:
```python
#standardization
```

In [34]:
```python
from sklearn.preprocessing import StandardScaler
```

In [35]:
```python
scaler=StandardScaler()
```

In [36]:
```python
scaler.fit(X_train)
```

Out[36]:
```
▾ StandardScaler
StandardScaler()
```

In [37]:
```python
scaled_data=scaler.transform(X_train)
```

In [38]:
```python
scaled_test=scaler.transform(X_test)
```

MODEL TRAINING AND TESTING

LINEAR REGRESSION

In [39]:
```python
from sklearn.linear_model import LinearRegression
```

In [40]:
```python
reg=LinearRegression()
```

In [41]:
```python
reg.fit(X_train,y_train)
```

Out[41]:
```
▾ LinearRegression
LinearRegression()
```

In [42]:
```python
reg.score(X_test,y_test)
```

Out[42]:
```
0.7710015538574033
```

In [43]:
```python
y_pred1=reg.predict(X_test)
```

In [44]:
```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred1)
```

Out[44]:
```
0.06530804159283013
```

EXTRA TREE REGRESSOR MODEL

In [45]:
```python
from sklearn.ensemble import ExtraTreesRegressor
```

In [46]:
```python
etr=ExtraTreesRegressor(n_estimators=100,random_state=42)
```

In [47]:
```python
etr.fit(X_train,y_train)
```

Out[47]:
```
▾         ExtraTreesRegressor
ExtraTreesRegressor(random_state=42)
```

In [48]:
```python
y_pred2=etr.predict(X_test)
```

In [49]:
```python
from sklearn.metrics import r2_score
```

In [50]:
```python
print('R2 score: ',r2_score(y_test,y_pred2)*100)
```

```
R2 score:  88.91947863675324
```

In [51]:
```python
mean_squared_error(y_test,y_pred2)
```

Out[51]:
```
0.031600526652069 48
```

CAT BOOST REGRESSOR

In [52]:
```python
from catboost import CatBoostRegressor
```

In [53]:
```python
cbr = CatBoostRegressor(verbose=0)
```

In [54]:
```python
cbr.fit(X_train,y_train)
```

Out[54]:
```
<catboost.core.CatBoostRegressor at 0x20b83e21990>
```

In [55]:
```python
y_pred3=cbr.predict(X_test)
```

In [56]:
```python
print('R2 score: ',r2_score(y_test,y_pred3)*100)
```

```
R2 score:  91.13995151161583
```

In [57]:
```python
mean_squared_error(y_test,y_pred3)
```

Out[57]:

```
0.025267962509822 95
```

GRADIENT BOOSTING REGRESSOR

In [58]:

```
from sklearn.ensemble import GradientBoostingRegressor
```

In [59]:

```
gbr =GradientBoostingRegressor()
```

In [60]:

```
gbr.fit(X_train, y_train)
```

Out[60]:

```
▾ GradientBoostingRegressor
GradientBoostingRegressor()
```

In [61]:

```
y_pred4=gbr.predict(X_test)
```

In [62]:

```
r2_score(y_test,y_pred4)
```

Out[62]:

```
0.8807240437310799
```

In [63]:

```
mse = mean_squared_error(y_test,y_pred4)
mse
```

Out[63]:

```
0.034016296810086694
```

MODELS EVALUATION

In [64]:

```
results_df=pd.DataFrame({'Models':['LR','ETR','CBC','GBR'],
                        'R2_score':[r2_score(y_test,y_pred1),
                       r2_score(y_test,y_pred2),
                       r2_score(y_test,y_pred3),
                       r2_score(y_test,y_pred4)]})
```

In [65]:

```
results_df.reset_index(drop=True, inplace=True)
results_df
```

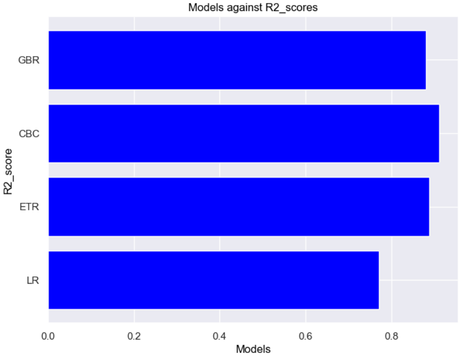
Out[65]:

	Models	R2_score
0	LR	0.771002
1	ETR	0.889195
2	CBC	0.911400
3	GBR	0.880724

In [66]:

```
#barh plot on Models against R2_scores
plt.figure(figsize=(8, 6))
plt.barh(results_df.index, results_df['R2_score'], tick_label=results_df['Models'],color='blue')
plt.xlabel('Models',color='black')
plt.ylabel('R2_score',color='black')
plt.title('Models against R2_scores',color='black')

plt.show()
```



CONCLUSION

Catboost Classifier was best in prediction wiyh R_2 score of 91%.

In [ ]: