House Price Prediction

Team Name : CODE CRAFTERS

Team Member 1 : Nithin Rosarieo

Team Member 2 : Naveenkumar S K

Team Member 3 : Aswanth Kumar R

Team Member 4 : Aneesh Balaji

Team Member 5 : Selvam M R

Importing required libraries

In [1]:

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

from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix,mean_absolute_error, mean_squared_error,

from sklearn.preprocessing import normalize,StandardScaler

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

In [2]:
```python
data=pd.read_csv("Chennai houseing sale.csv")
data.head()
```

Out[2]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes

5 rows × 22 columns

In [3]:
```python
data.shape
```

Out[3]: (7109, 22)

In [4]:
```python
data['DATE_SALE'].nunique()
```

Out[4]: 2798

Data Preprocessing

In [5]:
```python
data[['day_s', 'month_s', 'year_s']] = data['DATE_SALE'].str.split('-', expand=True)
data.head()
```

Out[5]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No

2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes

5 rows × 25 columns

In [6]:
```
data[['day_b', 'month_b', 'year_b']] = data['DATE_BUILD'].str.split('-', expand=True)
data.head()
```

Out[6]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes

5 rows × 28 columns

In [7]:
```
data.drop(columns = ['DATE_SALE','day_s','month_s','day_b','month_b','DATE_BUILD'],inplace=True)
data.head()
```

Out[7]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE
0	P03210	Karapakkam	1004	131	1.0	1.0	3	AbNormal	Yes	Commercial
1	P09411	Anna Nagar	1986	26	2.0	1.0	5	AbNormal	No	Commercial
2	P01812	Adyar	909	70	1.0	1.0	3	AbNormal	Yes	Commercial
3	P05346	Velachery	1855	14	3.0	2.0	5	Family	No	Others

4	P06210	Karapakkam	1226	84	1.0	1.0	3	AbNormal	Yes	Others

5 rows × 22 columns

In [8]:

```
data.AREA.replace(["Ann Nagar","Ana Nagar"],"Anna Nagar",inplace = True)
data.AREA.replace('Karapakam','Karapakkam',inplace=True)
data.AREA.replace(['Chrompt','Chrmpet','Chormpet'],'Chrompet',inplace=True)
data.AREA.replace('KKNagar','KK Nagar',inplace=True)
data.AREA.replace('TNagar','T Nagar',inplace=True)
data.AREA.replace('Adyr','Adyar',inplace=True)
data.AREA.replace('Velchery','Velachery',inplace=True)
data.SALE_COND.replace('Ab Normal','AbNormal',inplace=True)
data.SALE_COND.replace(['PartiaLl','Partiall'],'Partial',inplace=True)
data.SALE_COND.replace('Adj Land','AdjLand',inplace=True)
data.PARK_FACIL.replace('Noo','No',inplace=True)
data.BUILDTYPE.replace('Comercial','Commercial',inplace=True)
data.BUILDTYPE.replace('Other','Others',inplace=True)
data.UTILITY_AVAIL.replace('AllPub','All Pub',inplace=True)
data.UTILITY_AVAIL.replace('NoSewr ','NoSeWa',inplace=True)
data.STREET.replace('Pavd','Paved',inplace=True)
data.STREET.replace('NoAccess','No Access',inplace=True)
```

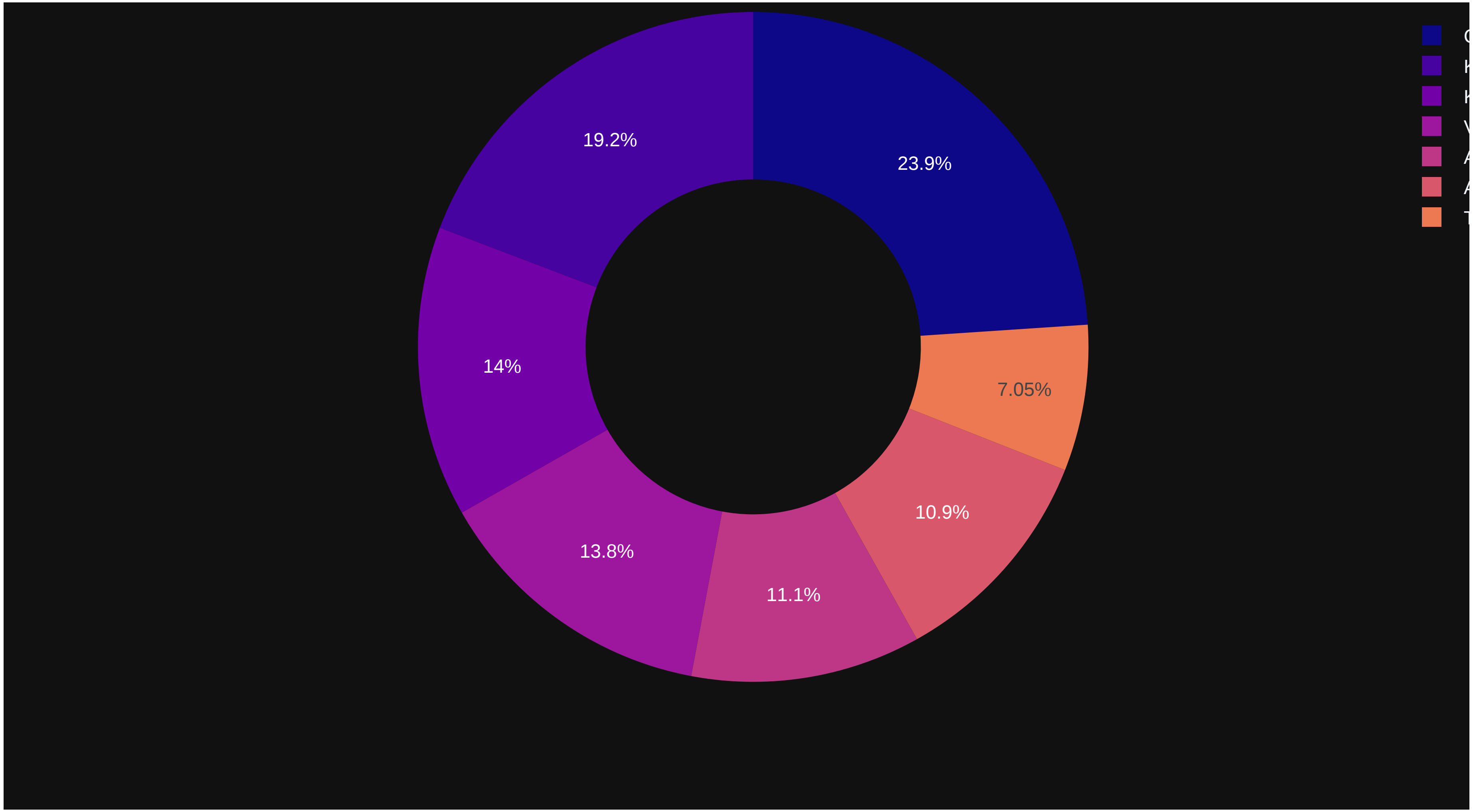
In [9]:

```
data.info()
```

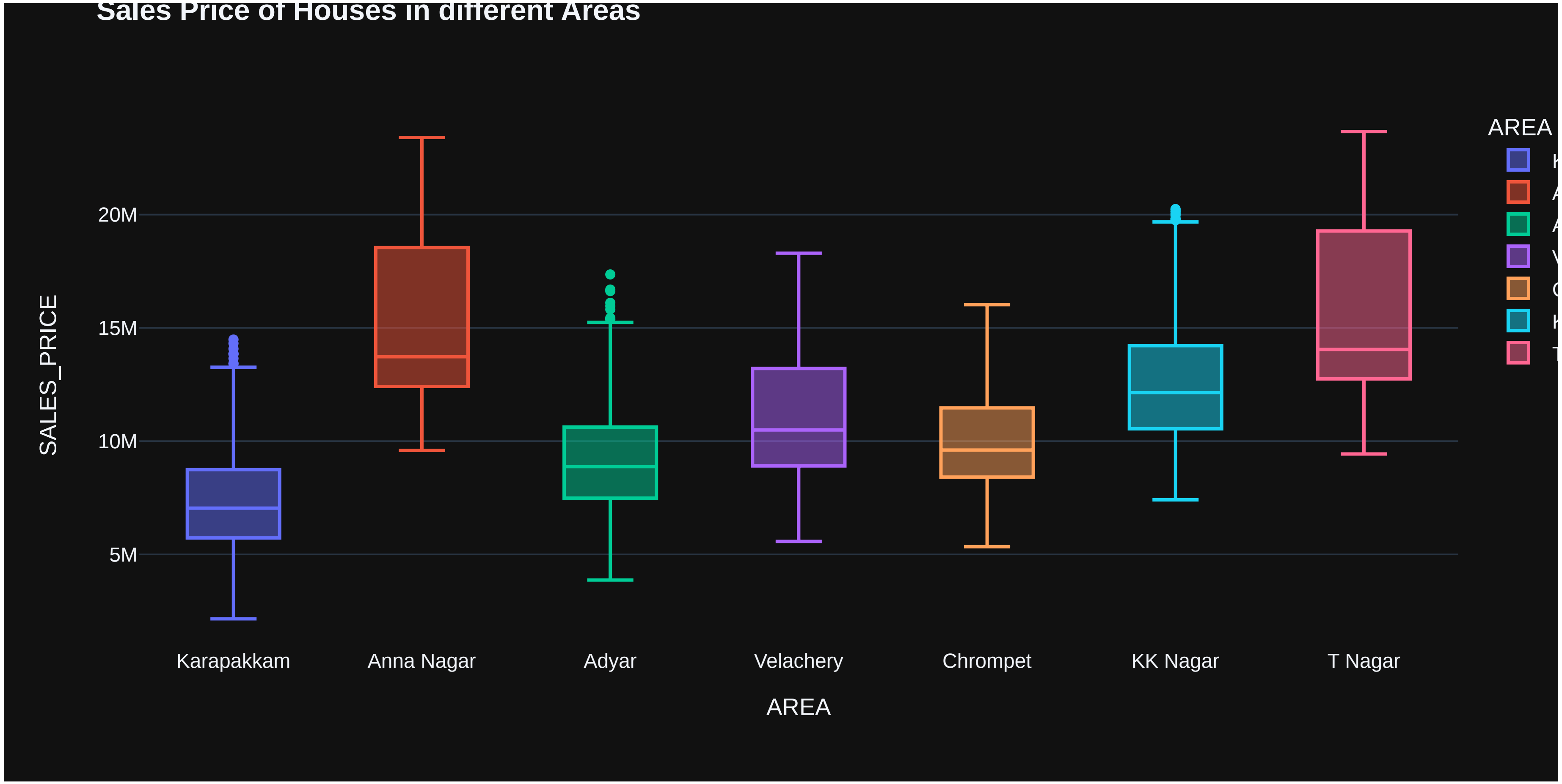
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7109 entries, 0 to 7108
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PRT_ID         7109 non-null   object
 1   AREA           7109 non-null   object
 2   INT_SQFT       7109 non-null   int64
 3   DIST_MAINROAD  7109 non-null   int64
 4   N_BEDROOM      7108 non-null   float64
 5   N_BATHROOM     7104 non-null   float64
 6   N_ROOM         7109 non-null   int64
 7   SALE_COND      7109 non-null   object
 8   PARK_FACIL     7109 non-null   object
 9   BUILDTYPE      7109 non-null   object
```

```
 10  UTILITY_AVAIL  7109 non-null   object
 11  STREET         7109 non-null   object
 12  MZZONE         7109 non-null   object
 13  QS_ROOMS       7109 non-null   float64
 14  QS_BATHROOM    7109 non-null   float64
 15  QS_BEDROOM     7109 non-null   float64
 16  QS_OVERALL     7061 non-null   float64
 17  REG_FEE        7109 non-null   int64
 18  COMMIS         7109 non-null   int64
 19  SALES_PRICE    7109 non-null   int64
 20  year_s         7109 non-null   object
 21  year_b         7109 non-null   object
dtypes: float64(6), int64(6), object(10)
memory usage: 1.2+ MB
```

In [10]:
```
import plotly.express as px

# Assuming 'data' is your DataFrame
fig = px.pie(
    data.groupby('AREA', as_index=False)['PRT_ID'].count(),
    values='PRT_ID',
    names='AREA',
    labels={'PRT_ID': 'Count'},
    template='plotly_dark',
    color_discrete_sequence=px.colors.sequential.Plasma,
    hole=0.5,
    title='<b> Houses Count in different Areas of Chennai</b>'
)

fig.update_layout(
    width=1000,  # Specify the width of the figure
    height=600,  # Specify the height of the figure
)

fig.show()
```



In [11]:
```
fig = px.box(data,x='AREA',y='SALES_PRICE',color='AREA',template='plotly_dark',title='<b> Sales Price of Houses in
fig.update_layout(
    width=1000,  # Specify the width of the figure
    height=500,  # Specify the height of the figure
)

fig.show()
```



Converting categorical columns to numerical columns

In [12]:
```
cat_cols=['PRT_ID','AREA','SALE_COND','PARK_FACIL','BUILDTYPE','UTILITY_AVAIL','STREET','MZZONE']

for col in cat_cols:
    le=LabelEncoder()
    data[col]=le.fit_transform(data[col])
```

In [13]:
```
data.head()
```

Out[13]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	...	MZ
0	2266	4	1004	131	1.0	1.0	3	0	1	0	...	
1	6664	1	1986	26	2.0	1.0	5	0	0	0	...	
2	1270	0	909	70	1.0	1.0	3	0	1	0	...	
3	3755	6	1855	14	3.0	2.0	5	2	0	2	...	
4	4393	4	1226	84	1.0	1.0	3	0	1	2	...	

5 rows × 22 columns

In [14]:

```
data.isnull().sum()
```

Out[14]:
```
PRT_ID            0
AREA              0
INT_SQFT          0
DIST_MAINROAD     0
N_BEDROOM         1
N_BATHROOM        5
N_ROOM            0
SALE_COND         0
PARK_FACIL        0
BUILDTYPE         0
UTILITY_AVAIL     0
STREET            0
MZZONE            0
QS_ROOMS          0
QS_BATHROOM       0
QS_BEDROOM        0
QS_OVERALL       48
REG_FEE           0
COMMIS            0
SALES_PRICE       0
year_s            0
year_b            0
dtype: int64
```

Imputing null values

In [15]:
```
data['N_BEDROOM']=data['N_BEDROOM'].fillna(data['N_BEDROOM'].mode()[0])
data['N_BATHROOM']=data['N_BATHROOM'].fillna(data['N_BATHROOM'].mode()[0])
data['QS_OVERALL'] = data['QS_OVERALL'].fillna(data['QS_OVERALL'].mean())
```

In [16]:
```
data.isnull().sum()
```

Out[16]:
```
PRT_ID           0
AREA             0
INT_SQFT         0
DIST_MAINROAD    0
N_BEDROOM        0
N_BATHROOM       0
N_ROOM           0
```

```
SALE_COND        0
PARK_FACIL       0
BUILDTYPE        0
UTILITY_AVAIL    0
STREET           0
MZZONE           0
QS_ROOMS         0
QS_BATHROOM      0
QS_BEDROOM       0
QS_OVERALL       0
REG_FEE          0
COMMIS           0
SALES_PRICE      0
year_s           0
year_b           0
dtype: int64
```

In [17]:
```
data.describe()
```

Out[17]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL
count	7109.000000	7109.000000	7109.000000	7109.000000	7109.000000	7109.000000	7109.000000	7109.000000	7109.000000
mean	3554.000000	2.959347	1382.073006	99.603179	1.636939	1.213110	3.688704	2.003939	0.504572
std	2052.335864	1.837797	457.410902	57.403110	0.802881	0.409534	1.019099	1.415302	0.500014
min	0.000000	0.000000	500.000000	0.000000	1.000000	1.000000	2.000000	0.000000	0.000000
25%	1777.000000	2.000000	993.000000	50.000000	1.000000	1.000000	3.000000	1.000000	0.000000
50%	3554.000000	3.000000	1373.000000	99.000000	1.000000	1.000000	4.000000	2.000000	1.000000
75%	5331.000000	4.000000	1744.000000	148.000000	2.000000	1.000000	4.000000	3.000000	1.000000
max	7108.000000	6.000000	2500.000000	200.000000	4.000000	2.000000	6.000000	4.000000	1.000000

In [18]:
```
data.head()
```

Out[18]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	...	MZ
0	2266	4	1004	131	1.0	1.0	3	0	1	0	...	
1	6664	1	1986	26	2.0	1.0	5	0	0	0	...	
2	1270	0	909	70	1.0	1.0	3	0	1	0	...	
3	3755	6	1855	14	3.0	2.0	5	2	0	2	...	
4	4393	4	1226	84	1.0	1.0	3	0	1	2	...	

5 rows × 22 columns

In [19]:
```
data['SALES_PRICE'] = np.log(data['SALES_PRICE'])
```

Separating Features and Targets

In [20]:
```
X = data.drop(columns=['PRT_ID','SALES_PRICE'], axis=1)
y = data['SALES_PRICE']
```

Splitting into training and testing data

In [21]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

Normalizing the data

In [22]:
```
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler to your data and transform it
```

```
X_train_scaled = scaler.fit_transform(x_train)

# Apply the same transformation to your test data
X_test_scaled = scaler.transform(x_test)
```

Linear Regression

In [23]:
```
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)

# 6. Evaluate the model
y_pred = lr.predict(X_test_scaled)  # Predi

# Calculate evaluation metrics on the original scale
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.10230502470065968
Mean Squared Error: 0.016233263477323828
R-squared: 0.8602312381474337
```

Decision Tree Regressor

In [24]:
```
from sklearn.tree import DecisionTreeRegressor

# Create a DecisionTreeRegressor object
dt= DecisionTreeRegressor()

# Fit the model to your scaled training data
dt.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
pred = dt.predict(X_test_scaled)
```

```
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, pred)

# Calculate R-squared (R2)
r2 = r2_score(y_test, pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.07208493521702575
Mean Squared Error: 0.009547641056492978
R-squared: 0.9177945968201123
```

Random Forest Regressor

In [25]:

```
from sklearn.ensemble import RandomForestRegressor

# Create a DecisionTreeRegressor object
rf= RandomForestRegressor()

# Fit the model to your scaled training data
rf.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
pred = rf.predict(X_test_scaled)


from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, pred)
```

```
# Calculate R-squared (R2)
r2 = r2_score(y_test, pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.05385719163257443
Mean Squared Error: 0.004898969168076341
R-squared: 0.9578197658201992
```

Gradient Boosting Regressor

In [26]:

```
from sklearn.ensemble import GradientBoostingRegressor

gbr= GradientBoostingRegressor()

# Fit the model to your scaled training data
gbr.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
pred = gbr.predict(X_test_scaled)


from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, pred)

# Calculate R-squared (R2)
r2 = r2_score(y_test, pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.05086750276316893
Mean Squared Error: 0.004074659156418248
R-squared: 0.9649170934692617
```

Extra Trees Regressor

In [27]:

```
from sklearn.ensemble import ExtraTreesRegressor

etr = ExtraTreesRegressor()

# Fit the model to your scaled training data
etr.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
pred = etr.predict(X_test_scaled)


from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, pred)

# Calculate R-squared (R2)
r2 = r2_score(y_test, pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.036687817783372476
Mean Squared Error: 0.002428708169339726
R-squared: 0.9790887683056452
```

XGB Regressor

In [28]:

```python
from xgboost import XGBRegressor

xgb= XGBRegressor()

# Fit the model to your scaled training data
xgb.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
pred = xgb.predict(X_test_scaled)


from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, pred)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, pred)

# Calculate R-squared (R2)
r2 = r2_score(y_test, pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

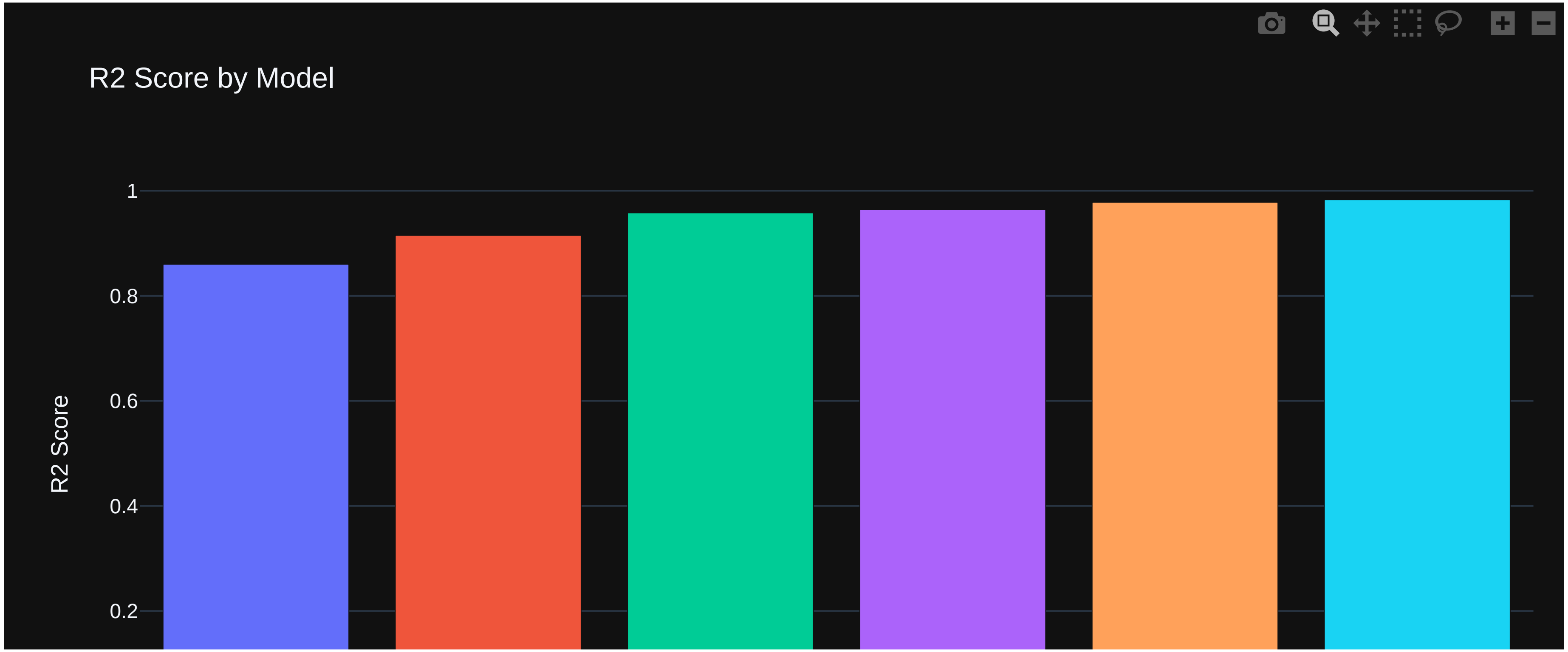
```
Mean Absolute Error: 0.03238330941141542
Mean Squared Error: 0.0019038846869966523
R-squared: 0.9836075102345686
```

In [29]:

```python
data.AREA.replace(0 ,'Adyar',inplace=True)
data.AREA.replace(1 ,'Anna Nagar',inplace=True)
data.AREA.replace(2 ,'Chrompet',inplace=True)
data.AREA.replace(3 ,'KK Nagar',inplace=True)
data.AREA.replace(4 ,'Karapakkam',inplace=True)
data.AREA.replace(5 ,'T Nagar',inplace=True)
data.AREA.replace(6 ,'Velachery',inplace=True)
```

Performance of all models

In [30]:
```
import plotly.express as px

# Define the data
models = ['LR', 'DTR', 'RFR', 'GBR', 'ETR', 'XGB']
r2_scores = [0.86, 0.915, 0.958, 0.964, 0.978, 0.983]

# Create a DataFrame from the data
df = {'Model': models, 'R2 Score': r2_scores}
df = pd.DataFrame(df)

# Plot the bar chart
fig = px.bar(df, x='Model', y='R2 Score', color='Model', title='R2 Score by Model',template='plotly_dark')
fig.update_layout(width=1000, height=500)
fig.show()
```



In [31]:

```
data['SALES_PRICE'] = np.exp(data['SALES_PRICE'])
```

In [32]:

```
X = data.drop(columns=['SALES_PRICE'], axis=1)
y = data['SALES_PRICE']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

In [33]:

```
xgb= XGBRegressor()

# Fit the model to your scaled training data
xgb.fit(X_train_scaled, y_train)

# Make predictions on the scaled test data
pred = xgb.predict(X_test_scaled)
out = pd.DataFrame({"Actual Price": y_test,"Predicted Price":pred})

Result = data.merge(out, left_index =True, right_index = True)
Result[['AREA','Actual Price','Predicted Price']].sample(20)
```

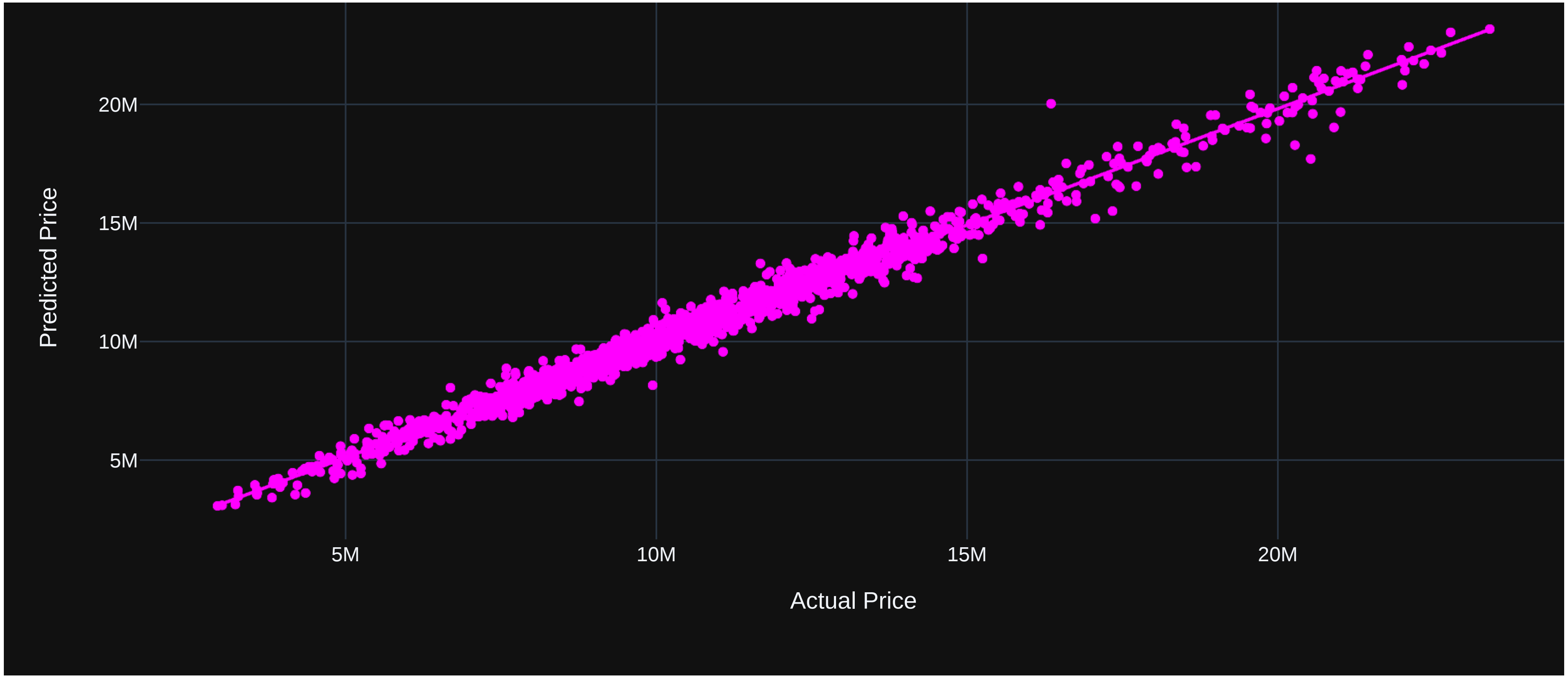
Out[33]:

	AREA	Actual Price	Predicted Price
1480	Chrompet	8720250.0	8637664.0
351	KK Nagar	12126880.0	12530122.0
2471	T Nagar	11202460.0	11039454.0
2299	Chrompet	12980150.0	12916391.0
5337	Anna Nagar	13876620.0	14139018.0

4824	T Nagar	20559960.0	19594830.0
4248	Velachery	7145230.0	7309117.0
2519	Chrompet	7728500.0	8117522.5
3573	Karapakkam	7142250.0	6831076.5
4502	Chrompet	9750500.0	9641069.0
1786	Karapakkam	4918500.0	4433794.0
247	Anna Nagar	12300050.0	12952053.0
4529	Chrompet	12827050.0	12935952.0
6703	Adyar	9410965.0	9522386.0
4553	Anna Nagar	14000930.0	14070038.0
683	T Nagar	21988560.0	21876064.0
4402	KK Nagar	11177540.0	11216770.0
4669	Velachery	11531350.0	12105379.0
319	Karapakkam	10147125.0	11360664.0
5517	Karapakkam	9886000.0	10117961.0

In [37]:

```
fig = px.scatter(Result,x='Actual Price',y='Predicted Price',trendline='ols',color_discrete_sequence=['magenta'],ter
fig.update_layout(width=1000, height=500)
fig.show()
```



Thus a maximum of 98.3% R2_score is obtained from XGB Regressor