ENCODING

```
#mapping in sequence to get an order in data (label encoding)
area={'T Nagar':0,'Adyar':1,
      'Anna Nagar':2,'Velachery':3,
      'KK Nagar':4,'Karapakkam':5,
      'Chrompet':6}
df['AREA']=df['AREA'].map(area)

buildtype={'House':0,'Others':1,'Commercial':3}
df['BUILDTYPE']=df['BUILDTYPE'].map(buildtype)

street={'NoAccess':0,'Paved':1,'Gravel':2}
df['STREET']=df['STREET'].map(street)

salecondition={'AbNormal':2, 'Family':1, 'Partial':0, 'AdjLand':4, 'NormalSale':3}
df['SALE_COND']=df['SALE_COND'].map(salecondition)

park_cond = {'Yes': 1, 'No': 0}
df['PARK_FACIL']=df['PARK_FACIL'].map(park_cond)
```

Encoded for categorical columns with order.

```
#one hot encoding using dummies
df=pd.get_dummies(df,columns=['MZZONE','UTILITY_AVAIL'])
df
```

	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_

```
df.columns
```

```
Index(['AREA', 'INT_SQFT', 'DATE_SALE', 'DIST_MAINROAD', 'N_BEDROOM',
       'N_BATHROOM', 'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'DATE_BUILD',
       'BUILDTYPE', 'STREET', 'QS_ROOMS', 'QS_BATHROOM', 'QS_BEDROOM',
       'QS_OVERALL', 'SALES_PRICE', 'AGE', 'MZZONE_A', 'MZZONE_C', 'MZZONE_I',
       'MZZONE_RH', 'MZZONE_RL', 'MZZONE_RM', 'UTILITY_AVAIL_All Pub',
       'UTILITY_AVAIL_ELO', 'UTILITY_AVAIL_NoSeWa', 'UTILITY_AVAIL_NoSewr '],
      dtype='object')
```

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

```
AREA                     0
INT_SQFT                 0
DATE_SALE                0
DIST_MAINROAD            0
N_BEDROOM                0
N_BATHROOM               0
N_ROOM                   0
SALE_COND                0
PARK_FACIL               0
DATE_BUILD               0
BUILDTYPE                0
STREET                   0
QS_ROOMS                 0
QS_BATHROOM              0
QS_BEDROOM               0
QS_OVERALL               0
SALES_PRICE              0
AGE                      0
MZZONE_A                 0
MZZONE_C                 0
MZZONE_I                 0
MZZONE_RH                0
MZZONE_RL                0
MZZONE_RM                0
UTILITY_AVAIL_All Pub    0
UTILITY_AVAIL_ELO        0
UTILITY_AVAIL_NoSeWa     0
UTILITY_AVAIL_NoSewr     0
dtype: int64
```

```
X=df[['AREA', 'INT_SQFT', 'N_BEDROOM',
       'N_BATHROOM', 'N_ROOM', 'SALE_COND', 'PARK_FACIL',
       'BUILDTYPE', 'STREET', 'QS_BATHROOM', 'AGE',
          'MZZONE_A', 'MZZONE_C',
       'MZZONE_I', 'MZZONE_RH', 'MZZONE_RL', 'MZZONE_RM',
       'UTILITY_AVAIL_All Pub', 'UTILITY_AVAIL_ELO', 'UTILITY_AVAIL_NoSeWa',
       'UTILITY_AVAIL_NoSewr ']]
```

```
X
```

	AREA	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYP
0	5	1004	1	1	3	2	1	[illegible]
1	2	1986	2	1	5	2	0	[illegible]
2	1	909	1	1	3	2	1	[illegible]
3	3	1855	3	2	5	1	0	[illegible]
4	5	1226	1	1	3	2	1	[illegible]
...	...	...	...	...	...	...	...	..
7104	5	598	1	1	2	4	0	[illegible]
7105	3	1897	3	2	5	1	1	[illegible]
7106	3	1614	2	1	4	3	0	[illegible]
7107	5	787	1	1	2	0	1	[illegible]
7108	3	1896	3	2	5	0	1	[illegible]

7109 rows × 21 columns

```
y=df['SALES_PRICE']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 4
```

```
df.AREA.unique()
```

```
array([5, 2, 1, 3, 6, 4, 0], dtype=int64)
```

```
# Import library for Linear Regression
from sklearn import metrics
from sklearn import linear_model
from sklearn.linear_model import LinearRegression

# Create a Linear regressor
lm = LinearRegression()

# Train the model using the training sets
lm.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

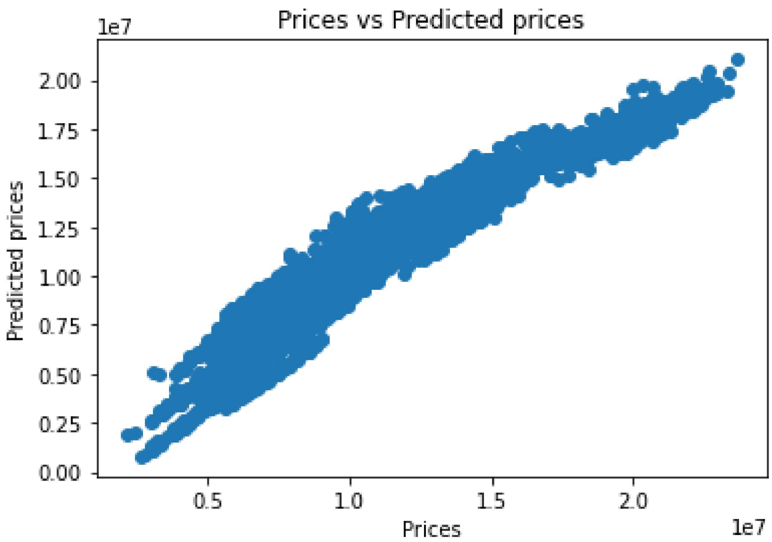
```
# Model prediction on train data
y_pred = lm.predict(X_train)
y_pred
```

```
array([ 8255392.56827569, 18499107.9790409 , 11318452.04276718, ...,
       13128586.59435115, 16959875.52963616, 11457371.77192385])
```

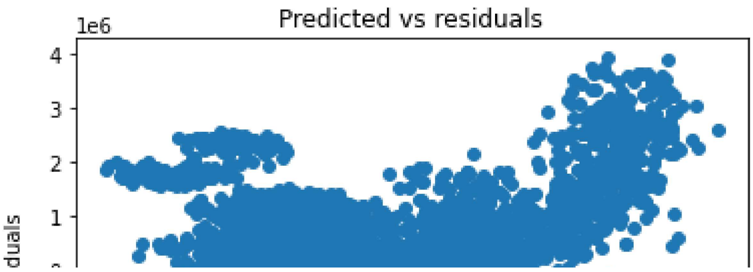
```
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_tr
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9100140148945116
Adjusted R^2: 0.9096325644126353
MAE: 879374.7599016309
MSE: 1262184514547.6724
RMSE: 1123469.8547569811
```

```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```




```
# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```




```
y_test_pred = lm.predict(X_test)
y_test_pred
```

```
array([11742059.51615519, 10091515.85546189,  9462677.52811251, ...,
        7210851.22840429,  4500084.70981717, 12180310.87965248])
```

0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00

```
# Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.9087153539549042
Adjusted R^2: 0.9078072641553083
MAE: 890981.6147025187
MSE: 1332531460339.6929
RMSE: 1154353.2649668788
```

RANDOMN FOREST

```
# Import Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regressor
reg = RandomForestRegressor()

# Train the model using the training sets
reg.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
y_pred = reg.predict(X_train)
y_pred
```

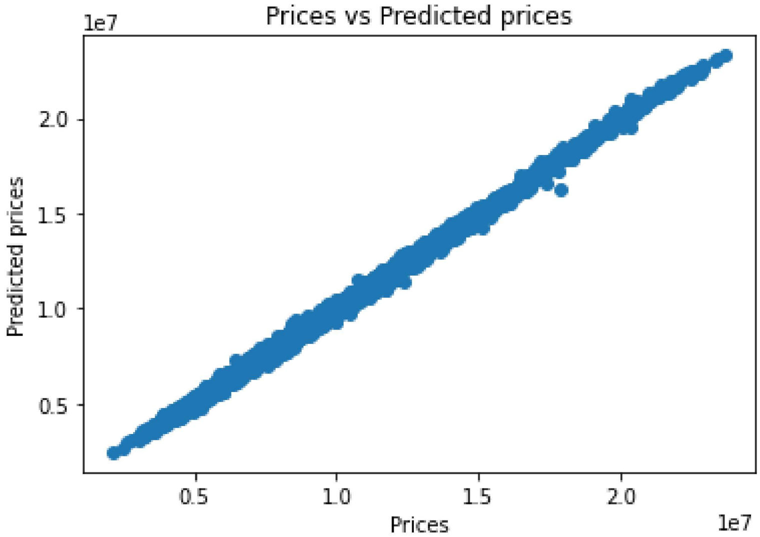
```
array([ 8088816.65, 19550568.4 , 10936636.65, ..., 12071177.4 ,
       18632139.5 , 10016315.25])
```

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
```

```
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_tr
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9977033551158098
Adjusted R^2: 0.9976936196409273
MAE: 135543.93639469452
MSE: 32213789790.063965
RMSE: 179482.00408415313
```

```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```




```
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```


```
# Predicting Test data with the model
y_test_pred = reg.predict(X_test)
y_test_pred
```

```
array([11698865.  , 10203239.15, 10317692.95, ...,  8147056.75,
        5686229.95, 12300640.6 ])
```

```
# Model Evaluation
acc_rf = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_rf)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.9849801182432028
Adjusted R^2: 0.9848307020817187
MAE: 360092.52369901544
MSE: 219253355724.54465
RMSE: 468244.97405155847
```

```
reg.score(X_train,y_train)
```

```
0.9977033551158098
```

```
reg.score(X_test,y_test)
```

```
0.9849801182432028
```

XG BOOST

```
# Import XGBoost Regressor
from xgboost import XGBRegressor

#Create a XGBoost Regressor
reg2 = XGBRegressor()

# Train the model using the training sets
reg2.fit(X_train, y_train)
```

XGBRegressor

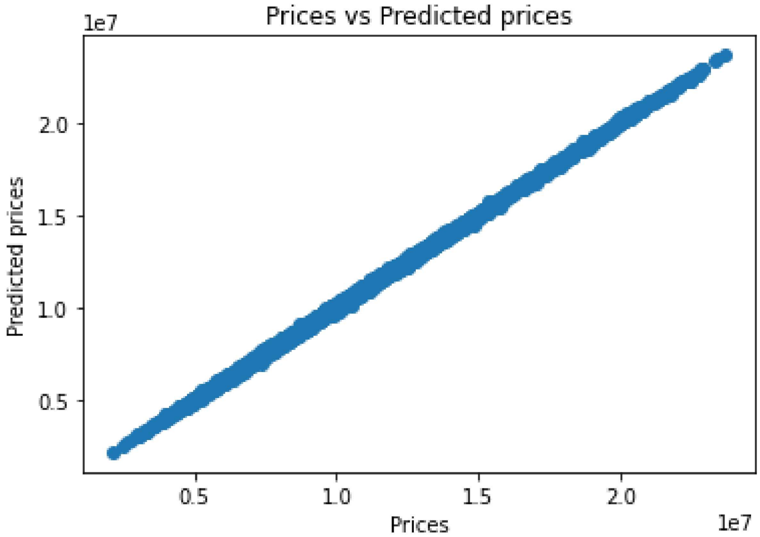
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
```

```
y_pred = reg2.predict(X_train)
```

```
 # Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_tr
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9994222352758405
Adjusted R^2: 0.9994197861318745
MAE: 68633.61238946945
MSE: 8103991827.5174465
RMSE: 90022.17408792928
```

```
 # Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```




```
#Predicting Test data with the model
y_test_pred = reg2.predict(X_test)
y_test_pred
```

```
array([11862186., 10262795., 10272243., ...,  8444848.,  4906911.,
       11992434.], dtype=float32)
```

```
# Model Evaluation
acc_xgb = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_xgb)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y
```

```
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.9955885395369309
Adjusted R^2: 0.9955446548047071
MAE: 190374.91303328646
MSE: 64396479668.44568
RMSE: 253764.61468937248
```

```
reg2.score(X_train,y_train)
```

```
0.9994222352758405
```

```
reg2.score(X_test,y_test)
```

```
0.9955885395369309
```

```
models = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost'],
    'R-squared Score': [acc_linreg*100, acc_rf*100, acc_xgb*100]})
models.sort_values(by='R-squared Score', ascending=False)
```

	Model	R-squared Score
2	XGBoost	99.558854
1	Random Forest	98.498012
0	Linear Regression	90.871535

```
feature_cols = X.columns.tolist()
```

```
# Plot feature importance of selected model - Random Forest
fea_df = pd.DataFrame({'Feature':feature_cols, 'Feature importance':reg.feature_importance
fea_df = fea_df.sort_values(by='Feature importance')

fig, ax = plt.subplots(figsize=(10,8))
fea_df.plot.barh(x='Feature', y='Feature importance', ax=ax)
plt.title('Feature Importances', fontsize=14);
```