

BE Final Project: House Price Prediction

1. Data Preprocessing

1.1 Importing Required Libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
from sklearn.metrics import r2_score, confusion_matrix
from sklearn import metrics
```

1.2 Data Acquisition and DataFrame Creation

In [2]:

```
df = pd.read_csv('../CSV/India.csv', index_col=False)
```

1.3 Data Evaluation

In [3]:

```
df.head()
```

Out[3]:

	Price	Area	City	Location	No. of Bedrooms	Resale	MaintenanceStaff	Gymnas
0	30000000	3340	Banglore	JP Nagar Phase 1	4	0	1	
1	7888000	1045	Banglore	Dasarahalli on Tumkur Road	2	0	0	
2	4866000	1179	Banglore	Kannur on Thanisandra Main Road	2	0	0	
3	8358000	1675	Banglore	Doddanekundi	3	0	0	
4	6845000	1670	Banglore	Kengeri	3	0	1	

5 rows × 41 columns



In [4]:

```
df.size
```

Out[4]:

413813

In [5]:

```
df.shape
```

Out[5]:

(10093, 41)

In [6]:



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10093 entries, 0 to 10092
Data columns (total 41 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Price            10093 non-null   int64  
 1   Area             10093 non-null   int64  
 2   City              10093 non-null   object  
 3   Location          10093 non-null   object  
 4   No. of Bedrooms  10093 non-null   int64  
 5   Resale            10093 non-null   int64  
 6   MaintenanceStaff 10093 non-null   int64  
 7   Gymnasium         10093 non-null   int64  
 8   SwimmingPool      10093 non-null   int64  
 9   LandscapedGardens 10093 non-null   int64  
 10  JoggingTrack     10093 non-null   int64  
 11  RainWaterHarvesting 10093 non-null   int64  
 12  IndoorGames      10093 non-null   int64  
 13  ShoppingMall     10093 non-null   int64  
 14  Intercom          10093 non-null   int64  
 15  SportsFacility   10093 non-null   int64  
 16  ATM               10093 non-null   int64  
 17  ClubHouse         10093 non-null   int64  
 18  School            10093 non-null   int64  
 19  24X7Security     10093 non-null   int64  
 20  PowerBackup       10093 non-null   int64  
 21  CarParking         10093 non-null   int64  
 22  StaffQuarter      10093 non-null   int64  
 23  Cafeteria         10093 non-null   int64  
 24  MultipurposeRoom  10093 non-null   int64  
 25  Hospital           10093 non-null   int64  
 26  WashingMachine    10093 non-null   int64  
 27  Gasconnection     10093 non-null   int64  
 28  AC                10093 non-null   int64  
 29  Wifi               10093 non-null   int64  
 30  Children'splayarea 10093 non-null   int64  
 31  LiftAvailable     10093 non-null   int64  
 32  BED               10093 non-null   int64  
 33  VaastuCompliant   10093 non-null   int64  
 34  Microwave          10093 non-null   int64  
 35  GolfCourse         10093 non-null   int64  
 36  TV                10093 non-null   int64  
 37  DiningTable        10093 non-null   int64  
 38  Sofa               10093 non-null   int64  
 39  Wardrobe           10093 non-null   int64  
 40  Refrigerator       10093 non-null   int64  
dtypes: int64(39), object(2)
memory usage: 3.2+ MB
```

In [7]:



```
df.columns
```

Out[7]:

```
Index(['Price', 'Area', 'City', 'Location', 'No. of Bedrooms', 'Resale',  
       'MaintenanceStaff', 'Gymnasium', 'SwimmingPool', 'LandscapedGardens',  
       'JoggingTrack', 'RainWaterHarvesting', 'IndoorGames', 'ShoppingMall',  
       'Intercom', 'SportsFacility', 'ATM', 'ClubHouse', 'School',  
       '24X7Security', 'PowerBackup', 'CarParking', 'StaffQuarter',  
       'Cafeteria', 'MultipurposeRoom', 'Hospital', 'WashingMachine',  
       'Gasconnection', 'AC', 'Wifi', 'Childrensplayarea', 'LiftAvailable',  
       'BED', 'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV',  
       'DiningTable', 'Sofa', 'Wardrobe', 'Refrigerator'],  
      dtype='object')
```

1.4 Data Insights

In [8]:

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



Out[8]:

```
Price          0
Area           0
City           0
Location       0
No. of Bedrooms 0
Resale          0
MaintenanceStaff 0
Gymnasium      0
SwimmingPool   0
LandscapedGardens 0
JoggingTrack   0
RainWaterHarvesting 0
IndoorGames    0
ShoppingMall   0
Intercom        0
SportsFacility 0
ATM             0
ClubHouse       0
School          0
24X7Security   0
PowerBackup     0
CarParking      0
StaffQuarter    0
Cafeteria      0
MultipurposeRoom 0
Hospital        0
WashingMachine 0
Gasconnection   0
AC              0
Wifi            0
Children'splayarea 0
LiftAvailable   0
BED             0
VaastuCompliant 0
Microwave       0
GolfCourse      0
TV              0
DiningTable     0
Sofa            0
Wardrobe         0
Refrigerator    0
dtype: int64
```

In [9]:



```
df.describe()['Wardrobe']
```

Out[9]:

```
count    10093.000000
mean      0.020906
std       0.143075
min      0.000000
25%     0.000000
50%     0.000000
75%     0.000000
max      1.000000
Name: Wardrobe, dtype: float64
```

In [10]:



```
#Corelation without considering LOCATION.  
df.corr()
```

Out[10]:

	Price	Area	No. of Bedrooms	Resale	MaintenanceStaff	Gymnasium
Price	1.000000	0.652585	0.439123	0.128312	0.088580	0.140000
Area	0.652585	1.000000	0.729552	-0.003146	-0.011789	0.270000
No. of Bedrooms	0.439123	0.729552	1.000000	0.003044	-0.096969	0.130000
Resale	0.128312	-0.003146	0.003044	1.000000	0.263689	-0.150000
MaintenanceStaff	0.088580	-0.011789	-0.096969	0.263689	1.000000	0.130000
Gymnasium	0.140537	0.276395	0.133460	-0.154323	0.137828	1.000000
SwimmingPool	0.156452	0.315745	0.181079	-0.226471	0.068617	0.750000
LandscapedGardens	0.123177	0.234055	0.121685	-0.137831	0.191619	0.570000
JoggingTrack	0.101692	0.255538	0.132476	-0.173717	0.165869	0.630000
RainWaterHarvesting	0.134042	0.200987	0.069604	-0.016485	0.323851	0.480000
IndoorGames	0.095156	0.244171	0.153838	-0.217318	0.102242	0.570000
ShoppingMall	0.043710	0.052815	0.009616	0.000341	0.260753	0.220000
Intercom	0.097880	0.233039	0.203128	-0.083848	0.115378	0.480000
SportsFacility	0.130813	0.259235	0.178083	-0.022974	0.064229	0.490000
ATM	0.037061	0.062419	-0.001850	-0.040649	0.206241	0.260000
ClubHouse	0.156836	0.299993	0.167430	-0.119821	0.148941	0.710000
School	0.021775	0.019527	-0.014630	0.056319	0.173322	0.190000
24X7Security	0.148066	0.184674	0.015184	0.031876	0.448519	0.520000
PowerBackup	0.156585	0.242213	0.074998	-0.054921	0.230572	0.540000
CarParking	0.143605	0.140859	0.016771	0.082128	0.429793	0.380000
StaffQuarter	0.088917	0.139186	0.061825	0.060067	0.239825	0.320000
Cafeteria	0.087466	0.145529	0.072895	-0.100546	0.202881	0.340000
MultipurposeRoom	0.062019	0.214171	0.118662	-0.186358	0.110524	0.500000
Hospital	0.040926	0.016409	-0.020175	0.028304	0.209561	0.160000
WashingMachine	0.104595	0.125486	0.094115	-0.009938	-0.004495	0.140000
Gasconnection	0.176744	0.166430	0.155670	0.232408	-0.001943	0.080000
AC	0.168448	0.154301	0.104396	0.083754	0.043612	0.090000
Wifi	0.056363	0.006440	-0.027855	0.083799	0.141772	0.060000
Children'splayarea	0.152377	0.316423	0.190020	-0.155772	0.030003	0.720000
LiftAvailable	0.073478	0.122343	0.105624	0.012941	0.127789	0.370000
BED	0.112406	0.039565	-0.016500	0.185955	0.229347	0.120000
VaastuCompliant	0.060695	0.152859	0.081532	0.008700	0.220419	0.360000
Microwave	0.098230	0.129399	0.093558	-0.019580	-0.013074	0.100000
GolfCourse	0.065167	0.061354	0.017765	-0.049791	0.144741	0.130000
TV	0.103201	0.091985	0.070671	0.014443	0.023459	0.060000
DiningTable	0.089542	0.105422	0.080242	-0.014219	-0.022006	0.120000

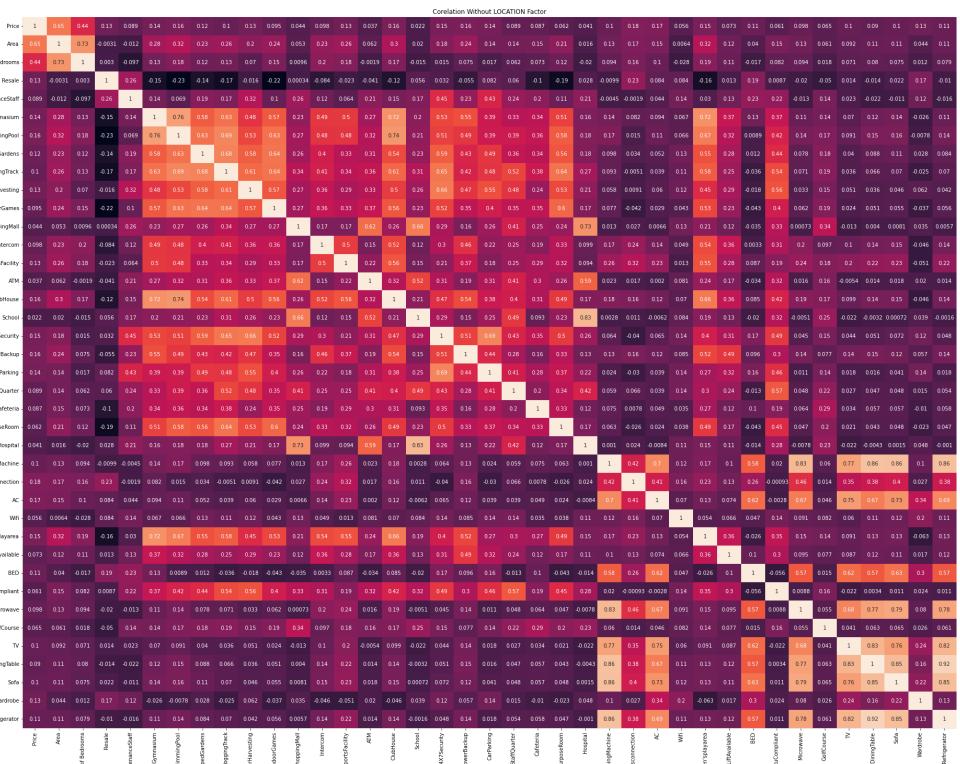
	Price	Area	No. of Bedrooms	Resale	MaintenanceStaff	Gymnas
Sofa	0.099936	0.113188	0.074601	0.021793	-0.011499	0.138
Wardrobe	0.125293	0.044020	0.011868	0.165969	0.116114	-0.026
Refrigerator	0.107972	0.107304	0.079399	-0.010177	-0.016042	0.105

In [11]:
39 rows × 39 columns

```
plt.figure(figsize=(40,25))
sns.heatmap(df.corr(), annot=True)
plt.title("Corelation Without LOCATION Factor")
# plt.savefig('../Results/Corelation-heatmap-Corelation-Without-LOCATION-Factor', dpi=
```

Out[11]:

Text(0.5, 1.0, 'Corelation Without LOCATION Factor')



2. Data Transformation and Scaling

2.1 Encoding Data: Label Encoding

In [12]:

```
# Checking that, is it feasible to use Label Encoding for 'Location column'?<br>
# ---> Yes, Because the no.of Total Rows are 7719 and No. of unique values for Location
df['Location'].unique()
df['Location'].unique().size
```

Out[12]:

849

In [13]:

```
df['City'].unique()
df['City'].unique().size
```

Out[13]:

6

In [14]:

```
# from sklearn.preprocessing ---> LabelEncoder
label_encoder = LabelEncoder()
print(label_encoder)
```

LabelEncoder()

In [15]:

```
df_encoded = df.copy()
df_encoded['Location'] = label_encoder.fit_transform(df['Location'])
df_encoded['City'] = label_encoder.fit_transform(df['City'])
```

In [16]:

```
df_encoded.head()
```

Out[16]:

	Price	Area	City	Location	No. of Bedrooms	Resale	MaintenanceStaff	Gymnasium	Swi
0	30000000	3340	0	254	4	0	1	1	
1	7888000	1045	0	155	2	0	0	1	
2	4866000	1179	0	302	2	0	0	1	
3	8358000	1675	0	163	3	0	0	0	
4	6845000	1670	0	319	3	0	1	1	

5 rows × 41 columns



In [17]:

```
df_encoded['Location'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 10093 entries, 0 to 10092
Series name: Location
Non-Null Count Dtype
-----
10093 non-null int32
dtypes: int32(1)
memory usage: 39.6 KB
```

In [18]:

```
df_encoded['City'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 10093 entries, 0 to 10092
Series name: City
Non-Null Count Dtype
-----
10093 non-null int32
dtypes: int32(1)
memory usage: 39.6 KB
```

In [19]:

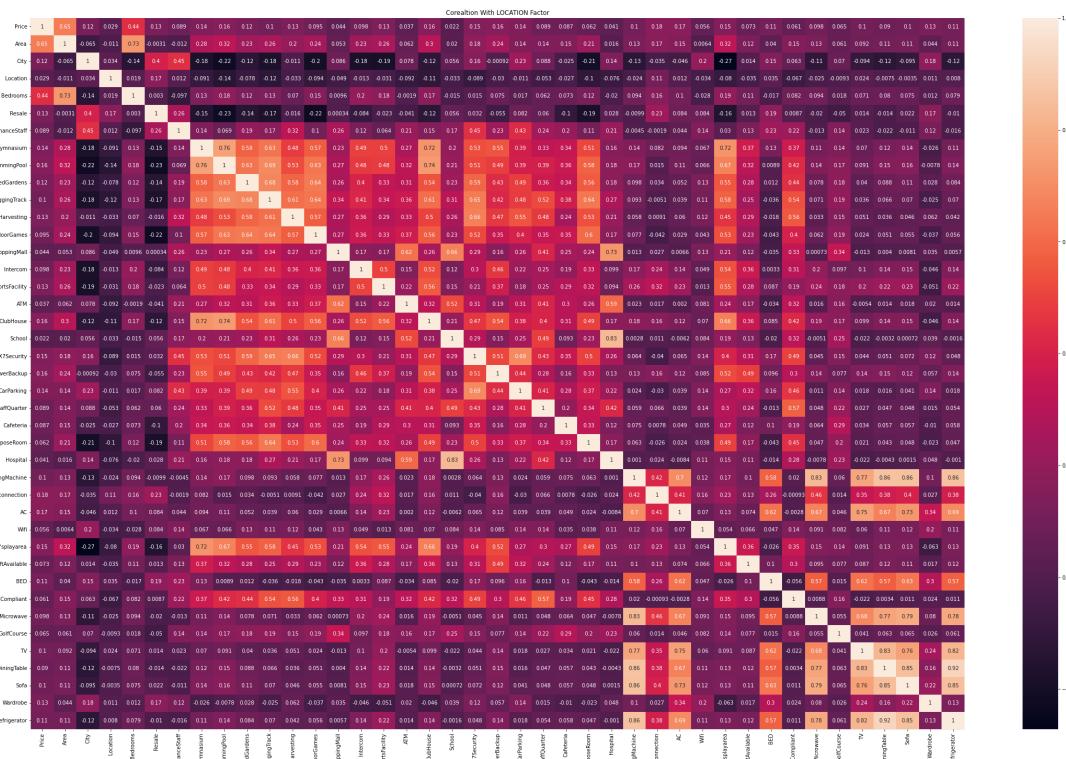
```
# sns.scatterplot(df_encoded['Price'])  
# plt.show()
```

In [20]:

```
plt.figure(figsize= (40,25))
sns.heatmap(df_encoded.corr(), annot=True)
plt.title("Corealtion With LOCATION Factor")
# plt.show()
# plt.savefig('../Results/Corelation-heatmap-Corelation-With-LOCATION-Factor', dpi=500)
```

Out[20]:

Text(0.5, 1.0, 'Corealtion With LOCATION Factor')



In [21]:



```
df_encoded.corr().iloc[0].sort_values(ascending=False)
```

Out[21]:

Price	1.000000
Area	0.652585
No. of Bedrooms	0.439123
Gasconnection	0.176744
AC	0.168448
ClubHouse	0.156836
PowerBackup	0.156585
SwimmingPool	0.156452
Children'splayarea	0.152377
24X7Security	0.148066
CarParking	0.143605
Gymnasium	0.140537
RainWaterHarvesting	0.134042
SportsFacility	0.130813
Resale	0.128312
Wardrobe	0.125293
LandscapedGardens	0.123177
City	0.117877
BED	0.112406
Refrigerator	0.107972
WashingMachine	0.104595
TV	0.103201
JoggingTrack	0.101692
Sofa	0.099936
Microwave	0.098230
Intercom	0.097880
IndoorGames	0.095156
DiningTable	0.089542
StaffQuarter	0.088917
MaintenanceStaff	0.088580
Cafeteria	0.087466
LiftAvailable	0.073478
GolfCourse	0.065167
MultipurposeRoom	0.062019
VaastuCompliant	0.060695
Wifi	0.056363
ShoppingMall	0.043710
Hospital	0.040926
ATM	0.037061
Location	0.028993
School	0.021775

Name: Price, dtype: float64

2.2 Encoding Data: One Hot Encoding

In [22]:

```
# -- from sklearn.preprocessing --> OneHotEncoder
# df_ohe = df1.copy()
# ohe = OneHotEncoder()
# print(ohe)
# feature_array = ohe.fit_transform(df_ohe[['Location']]).toarray()
# feature_array = feature_array[:,1:]
# column_names = df_ohe['Location'].tolist()
# df_ohe.drop('Location', axis = 1)
# pd.DataFrame(feature_array, columns = column_names)
```

2.3 Data Scaling

In [23]:

```
# Scaling AREA and PRICE Data as it contains high deviation of values
df_scaled = df_encoded.copy()
scaler = MinMaxScaler()
df_scaled[['Area']] = scaler.fit_transform(df_scaled[['Area']])
df_scaled[['Price']] = scaler.fit_transform(df_scaled[['Price']])
```

In [24]:

```
df_scaled.head()
```

Out[24]:

	Price	Area	City	Location	No. of Bedrooms	Resale	MaintenanceStaff	Gymnasium
0	0.070352	0.323711	0	254	4	0	1	1
1	0.014794	0.087113	0	155	2	0	0	1
2	0.007201	0.100928	0	302	2	0	0	1
3	0.015975	0.152062	0	163	3	0	0	0
4	0.012173	0.151546	0	319	3	0	1	1

5 rows × 41 columns

2.4 Splitting Data: Dependent and Independent Variable

In [25]:

```
df_temp = df_scaled.copy()  
data = df_temp.drop('Price', axis=1)  
price = df_temp['Price']
```

In [26]:

```
data.head()
```

Out[26]:

	Area	City	Location	No. of Bedrooms	Resale	MaintenanceStaff	Gymnasium	Swimmingl
0	0.323711	0	254	4	0	1	1	
1	0.087113	0	155	2	0	0	1	
2	0.100928	0	302	2	0	0	1	
3	0.152062	0	163	3	0	0	0	
4	0.151546	0	319	3	0	1	1	

5 rows × 40 columns

In [27]:

```
price.head()
```

Out[27]:

```
0    0.070352  
1    0.014794  
2    0.007201  
3    0.015975  
4    0.012173  
Name: Price, dtype: float64
```

2.5 Train - Test Split

In [28]:

```
X_train, X_test, y_train, y_test = train_test_split(data, price, test_size=0.2)  
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

8074 2019 8074 2019

3 Model Evaluation

3.1 Linear Regression

3.1.1 Linear Regression: Training Model

In [29]:

```
from sklearn import linear_model  
from sklearn.linear_model import LinearRegression
```

In [30]:

```
LR = LinearRegression()  
LR.fit(X_train, y_train)
```

Out[30]:

```
LinearRegression  
| LinearRegression()
```

3.1.2 Linear Regression: Testing

In [31]:

```
y_predict_LR = LR.predict(X_test)  
print(y_predict_LR)
```

```
[1.60685252e-02 1.93074870e-05 1.39669582e-02 ... 1.22896940e-01  
 1.74909667e-03 6.61787046e-06]
```

3.1.3 Linear Regression: Score / Results

In [32]:

```
LR_score = LR.score(X_test, y_test)  
print(LR_score*100)
```

```
45.173945051556665
```

In [33]:

```
R2_Score_LR = r2_score(y_test, y_predict_LR)*100
MAE_LR = metrics.mean_absolute_error(y_test, y_predict_LR)*100
MSE_LR = metrics.mean_squared_error(y_test, y_predict_LR)*100
RSME_LR = np.sqrt(metrics.mean_squared_error(y_test, y_predict_LR))*100
print('R2 Score Linear Regression', R2_Score_LR)
print('MAE Linear Regression:', MAE_LR)
print('MSE Linear Regression:', MSE_LR)
print('RMSE Linear Regression:', RSME_LR)
```

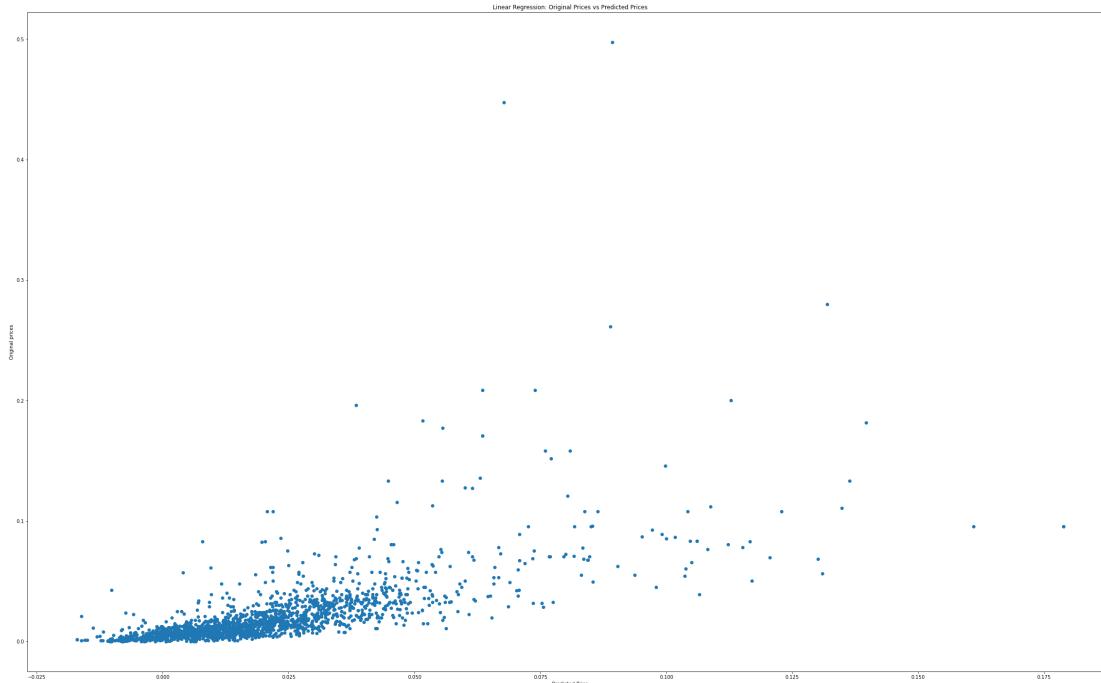
```
R2 Score Linear Regression 45.173945051556665
MAE Linear Regression: 1.0384391931038575
MSE Linear Regression: 0.04425537792839045
RMSE Linear Regression: 2.1036962216154325
```

In [34]:

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_LR, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("Linear Regression: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/Linear-Regression-Predicted-vs-Original-Scattered', dpi=500,
```

Out[34]:

```
Text(0.5, 1.0, 'Linear Regression: Original Prices vs Predicted Price
s')
```

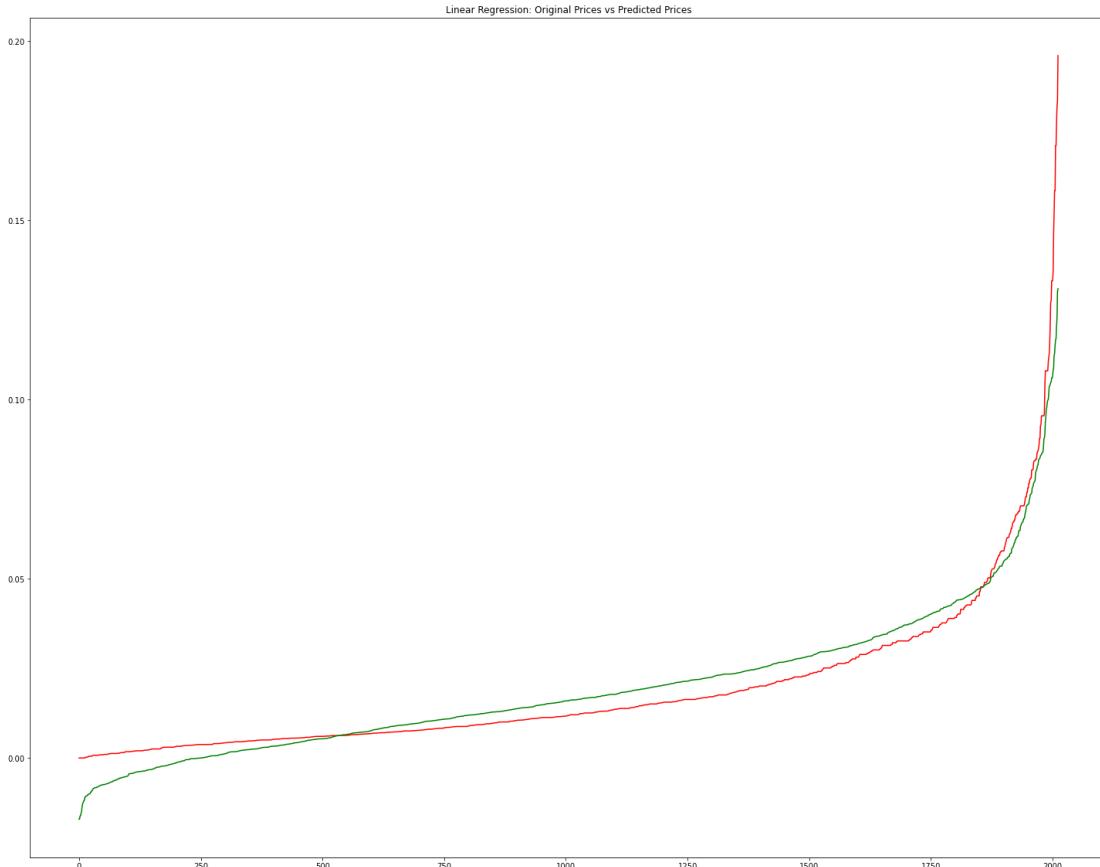


In [35]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_LR)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Linear Regression: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Linear-Regression-Predicted-vs-Original-Line', dpi=500, bbox
```

Out[35]:

```
Text(0.5, 1.0, 'Linear Regression: Original Prices vs Predicted Price  
s')
```

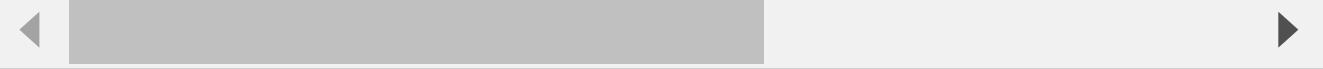


In [36]:

```
# Cross Validation
CVS_LR = cross_val_score(LR, data, price, cv=10).mean()
CVP_LR = cross_val_predict(LR, X_test, y_test, cv=10)
```

In [120]:

```
x = np.sort(y_test)[-7]
y = np.sort(CVP_LR)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - Linear Regression: Original Prices vs Predicted Prices"
# plt.savefig('../Results/Cross-Validation/Cross-Validation-Linear-Regression-Origin
```



3.2 Ridge Regression

3.2.1 Ridge Regression: Model Training

In [38]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

In [39]:

```
RR = Ridge()  
RR.fit(X_train,y_train)
```

Out[39]:

```
▼ Ridge  
Ridge()
```

3.2.2 Ridge Regression: Testing

In [40]:

```
y_predict_RR = RR.predict(X_test)  
print(y_predict_RR)
```

```
[ 0.01650011 -0.00043466  0.01433262 ...  0.11958268  0.00268245  
 0.00103198]
```

3.2.3 Ridge Regression: Score / Results

In [41]:

```
RR_score = RR.score(X_test, y_test)  
print(RR_score*100)
```

```
45.68868243262797
```

In [42]:

```
R2_Score_RR = r2_score(y_test, y_predict_RR)*100  
MAE_RR = metrics.mean_absolute_error(y_test, y_predict_RR)*100  
MSE_RR = metrics.mean_squared_error(y_test, y_predict_RR)*100  
RSME_RR = np.sqrt(metrics.mean_squared_error(y_test, y_predict_RR))*100  
print('R2 Score Linear Regression', R2_Score_RR)  
print('MAE Ridge Regression:', MAE_RR)  
print('MSE Ridge Regression:', MSE_RR)  
print('RMSE Ridge Regression:', RSME_RR)
```

```
R2 Score Linear Regression 45.68868243262797  
MAE Ridge Regression: 1.0246972972636066  
MSE Ridge Regression: 0.04383988392002888  
RMSE Ridge Regression: 2.0937976005342276
```

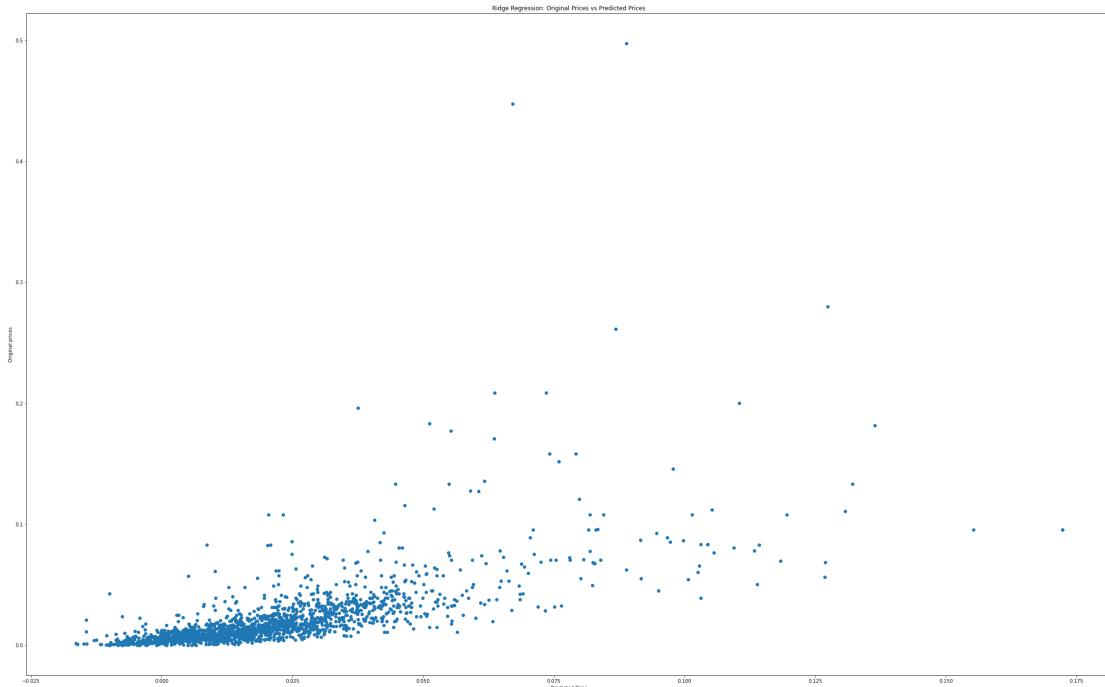
In [43]:



```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_RR, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("Ridge Regression: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/Ridge-Regression-Predicted-vs-Original-Scattered', dpi=500,
```

Out[43]:

Text(0.5, 1.0, 'Ridge Regression: Original Prices vs Predicted Prices')

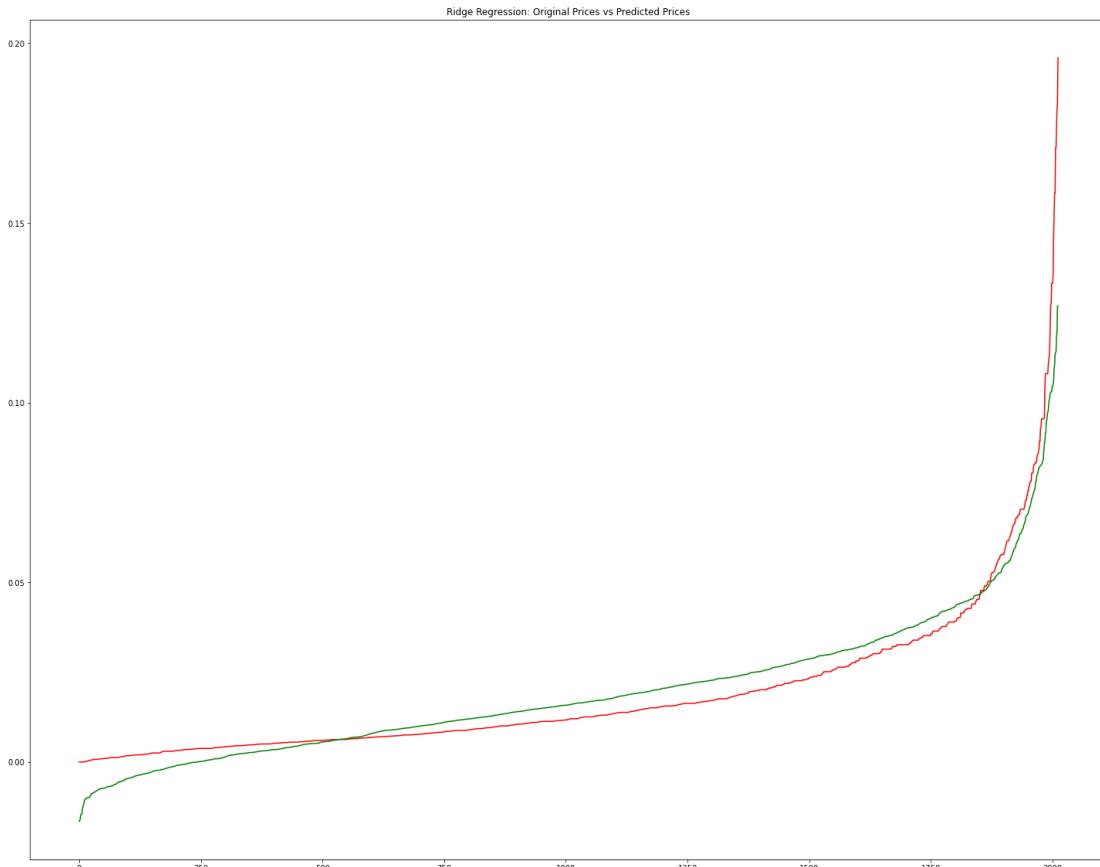


In [44]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_RR)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Ridge Regression: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Ridge-Regression-Predicted-vs-Original-Line', dpi=500, bbox_
```

Out[44]:

Text(0.5, 1.0, 'Ridge Regression: Original Prices vs Predicted Prices')

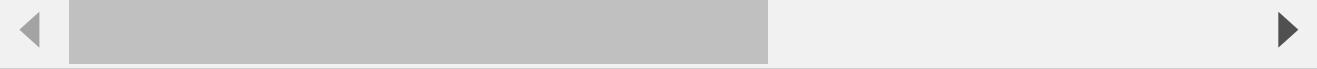


In [45]:

```
# Cross Validation
CVS_RR = cross_val_score(RR, data, price, cv=10).mean()
CVP_RR = cross_val_predict(RR, X_test, y_test, cv=10)
```

In [121]:

```
x = np.sort(y_test)[:7]
y = np.sort(CVP_RR)[:7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - Ridge Regression: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Cross-Validation/Cross-Validation-Ridge-Regression-Original
```



In []:

3.3 Lasso Regression

3.3.1 Lasso Regression: Model Training

In [47]:

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [48]:

```
Lasso_R = Lasso()  
Lasso_R.fit(X_train,y_train)
```

Out[48]:

```
▼ Lasso  
Lasso()
```

3.3.2 Lasso Regression: Testing

In [49]:

```
y_predict_Lasso_R = Lasso_R.predict(X_test)  
print(y_predict_Lasso_R)
```

```
[0.02125218 0.02125218 0.02125218 ... 0.02125218 0.02125218 0.02125218]
```

3.3.3 Lasso Regression: Score / Results

In [50]:

```
Lasso_R_score = Lasso_R.score(X_test, y_test)  
print(Lasso_R_score*100)
```

```
-0.17722422508095192
```

In [51]:

```
R2_Score_Lasso_R = r2_score(y_test, y_predict_Lasso_R)*100  
MAE_Lasso_R = metrics.mean_absolute_error(y_test, y_predict_Lasso_R)*100  
MSE_Lasso_R = metrics.mean_squared_error(y_test, y_predict_Lasso_R)*100  
RSME_Lasso_R = np.sqrt(metrics.mean_squared_error(y_test, y_predict_Lasso_R))*100  
print('R2 Score Linear Regression', R2_Score_Lasso_R)  
print('MAE Ridge Regression:', MAE_Lasso_R)  
print('MSE Ridge Regression:', MSE_Lasso_R)  
print('RMSE Ridge Regression:', RSME_Lasso_R)
```

```
R2 Score Linear Regression -0.17722422508095192  
MAE Ridge Regression: 1.6205463949979682  
MSE Ridge Regression: 0.08086266506074674  
RMSE Ridge Regression: 2.8436361416458813
```

In [52]:

▶

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_Lasso_R, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("Lasso Regression: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/Lasso-Regression-Predicted-vs-Original-Scattered', dpi=500,
```

Out[52]:

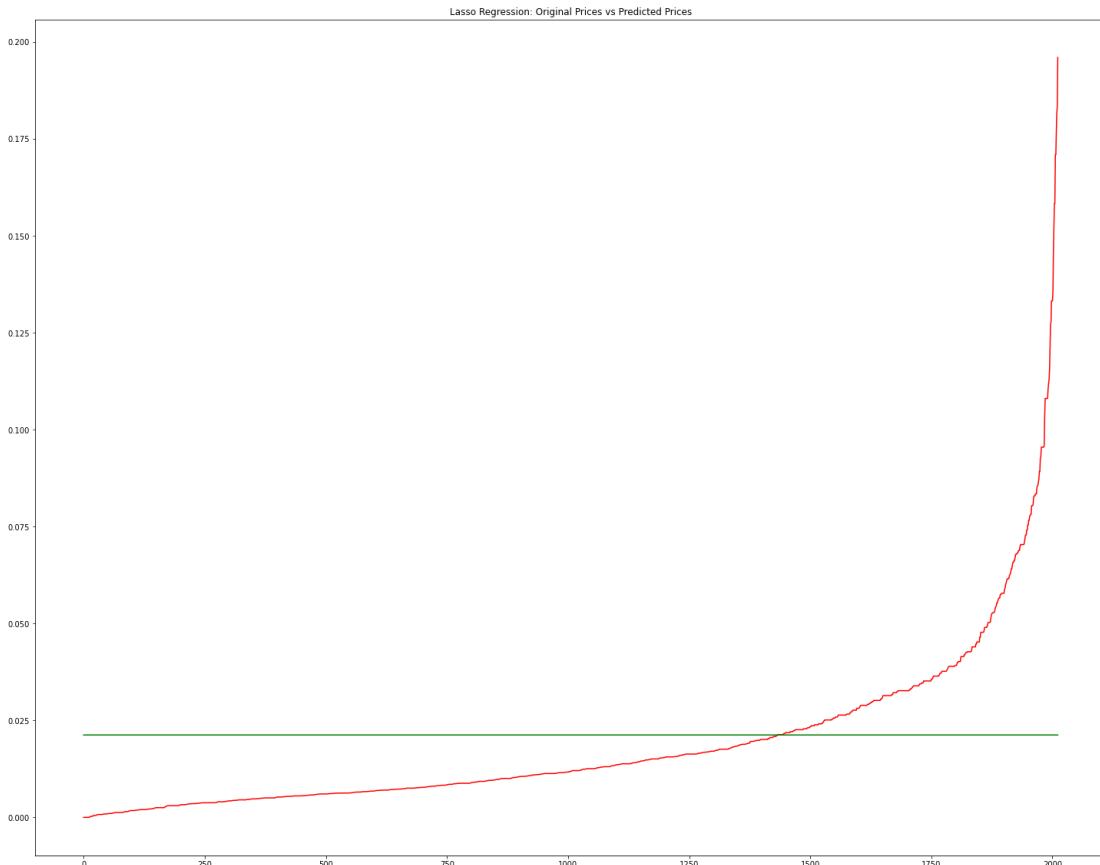
Text(0.5, 1.0, 'Lasso Regression: Original Prices vs Predicted Prices')

In [53]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_Lasso_R)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Lasso Regression: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Lasso-Regression-Predicted-vs-Original-Line', dpi=500, bbox_
```

Out[53]:

Text(0.5, 1.0, 'Lasso Regression: Original Prices vs Predicted Prices')

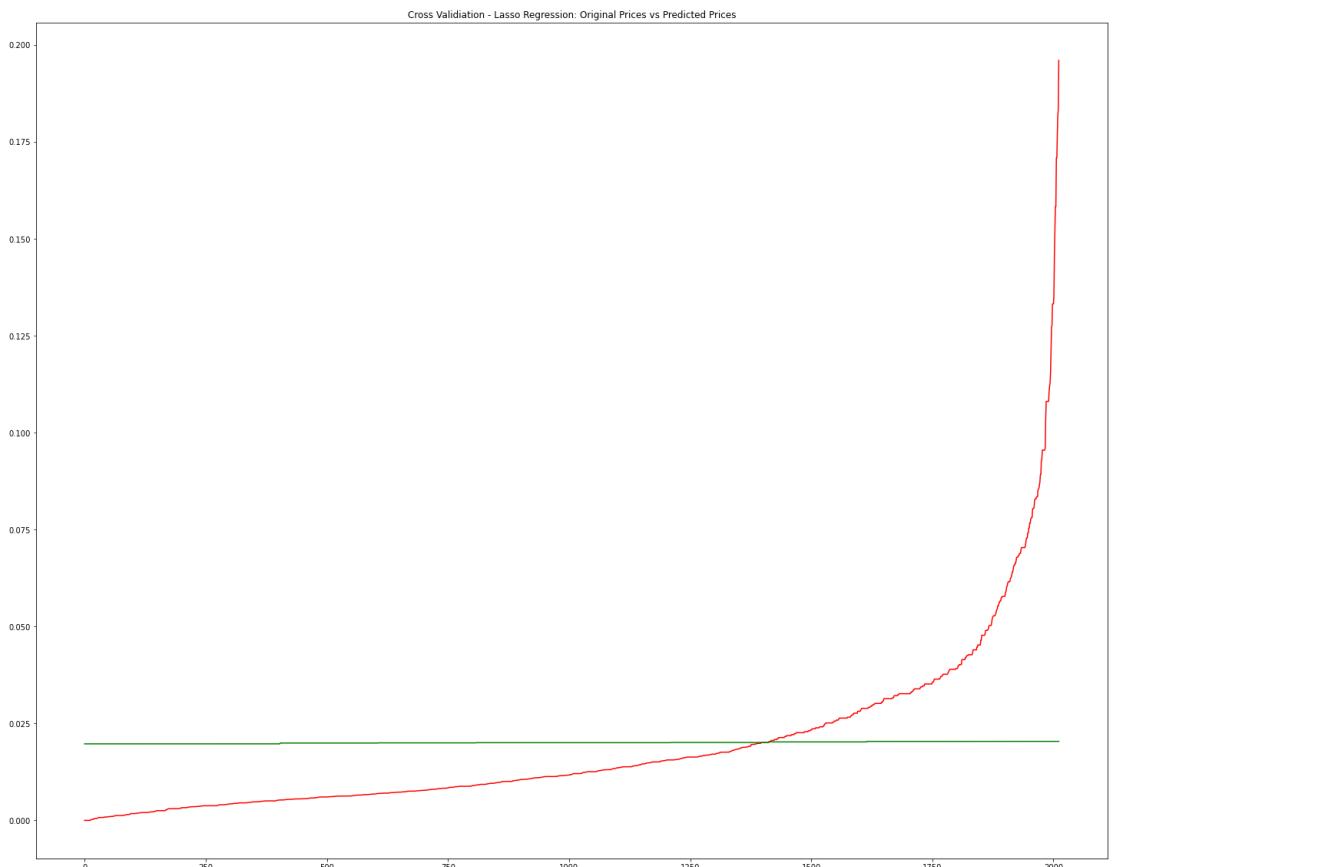


In [54]:

```
# Cross Validation
CVS_Lasso_R = cross_val_score(Lasso_R, data, price, cv=10).mean()
CVP_Lasso_R = cross_val_predict(Lasso_R, X_test, y_test, cv=10)
```

In [122]:

```
x = np.sort(y_test)[-7]
y = np.sort(CVP_Lasso_R)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - Lasso Regression: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Cross-Validation/Cross-Validation-Lasso-Regression-Original
```



In []:

3.4 Decision Tree

3.4.1 Decision Tree: Model Training

In [56]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [57]:

```
DT = DecisionTreeRegressor(max_depth=5)
DT.fit(X_train,y_train)
```

Out[57]:

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5)
```

3.4.2 Decision Tree: Testing

In [58]:

```
y_predict_DT = DT.predict(X_test)
print(y_predict_DT)
```

```
[0.01913228 0.00329474 0.01913228 ... 0.10048018 0.00727937 0.00727937]
```

3.4.3 Decision Tree: Score / Results

In [59]:

```
DT_score = DT.score(X_test, y_test)
print(DT_score*100)
```

```
47.42236053865733
```

In [60]:

```
R2_Score_DT = r2_score(y_test, y_predict_DT)*100
MAE_DT = metrics.mean_absolute_error(y_test, y_predict_DT)*100
MSE_DT = metrics.mean_squared_error(y_test, y_predict_DT)*100
RSME_DT = np.sqrt(metrics.mean_squared_error(y_test, y_predict_DT))*100
print('R2 Score Linear Regression', R2_Score_DT)
print('MAE Ridge Regression:', MAE_DT)
print('MSE Ridge Regression:', MSE_DT)
print('RMSE Ridge Regression:', RSME_DT)
```

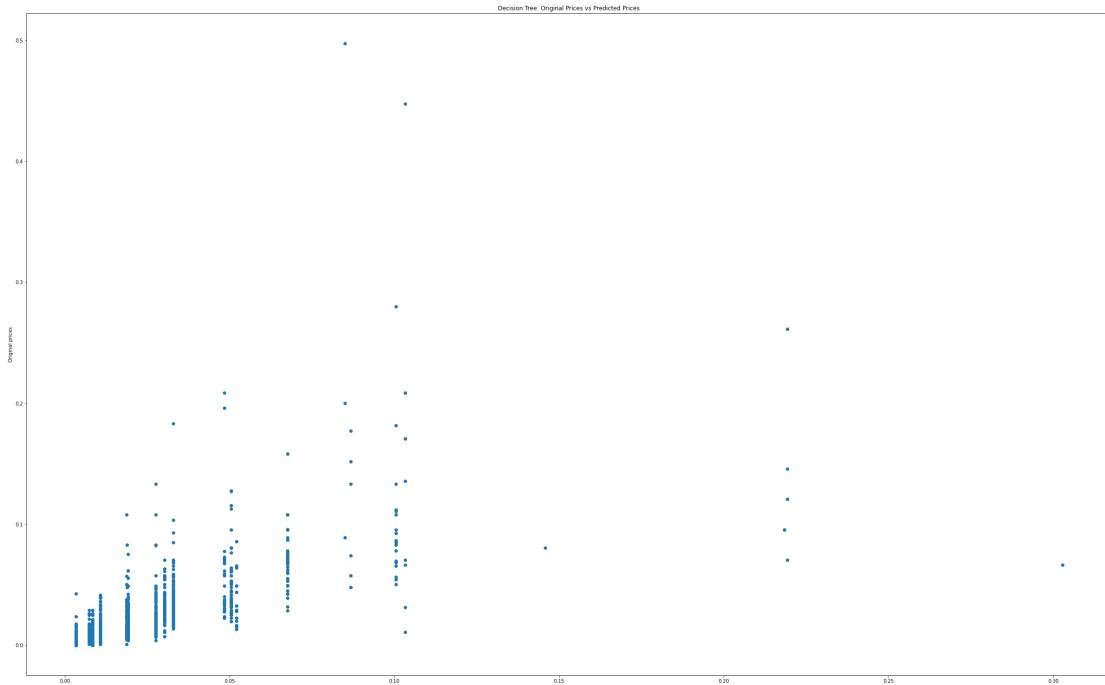
```
R2 Score Linear Regression 47.42236053865733
MAE Ridge Regression: 0.830300741721753
MSE Ridge Regression: 0.04244046570800078
RMSE Ridge Regression: 2.06010838811944
```

In [61]:

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_DT, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("Decision Tree: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/Decision-Tree-Predicted-vs-Original-Scattered', dpi=500, bbox
```

Out[61]:

```
Text(0.5, 1.0, 'Decision Tree: Original Prices vs Predicted Prices')
```



In [62]:

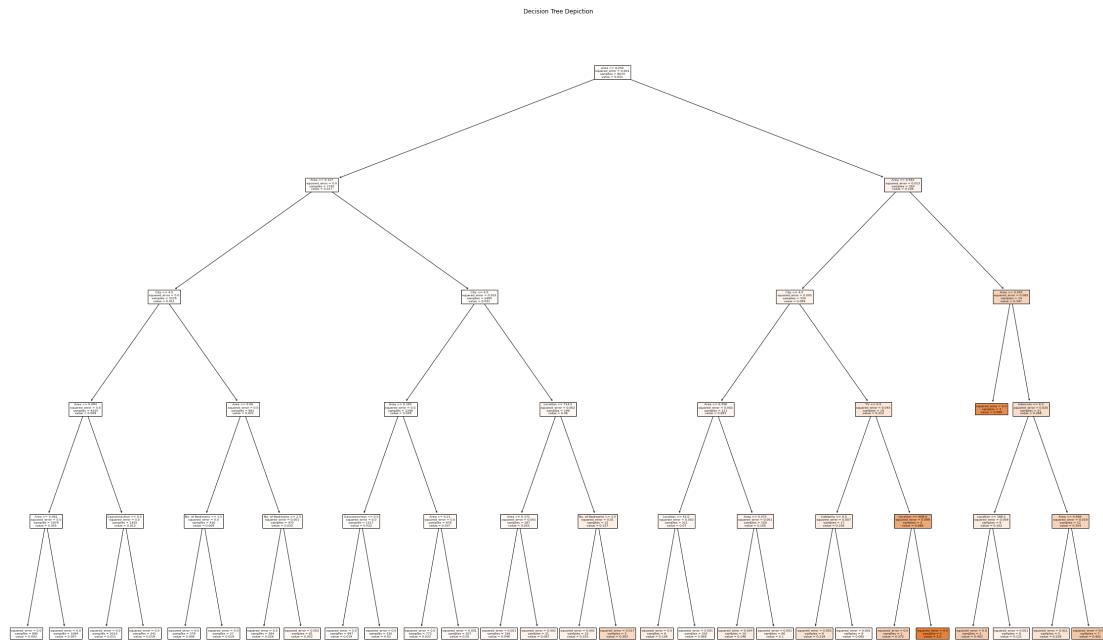
```
from sklearn import tree
```

In [63]:

```
plt.figure(figsize=(40,25))
tree.plot_tree(DT, feature_names=data.columns, class_names=price, filled = True)
plt.title("Decision Tree Depiction")
# plt.savefig('../Results/Decision-Tree-Depiction', dpi=500, bbox_inches='tight')
```

Out[63]:

Text(0.5, 1.0, 'Decision Tree Depiction')



In [64]:

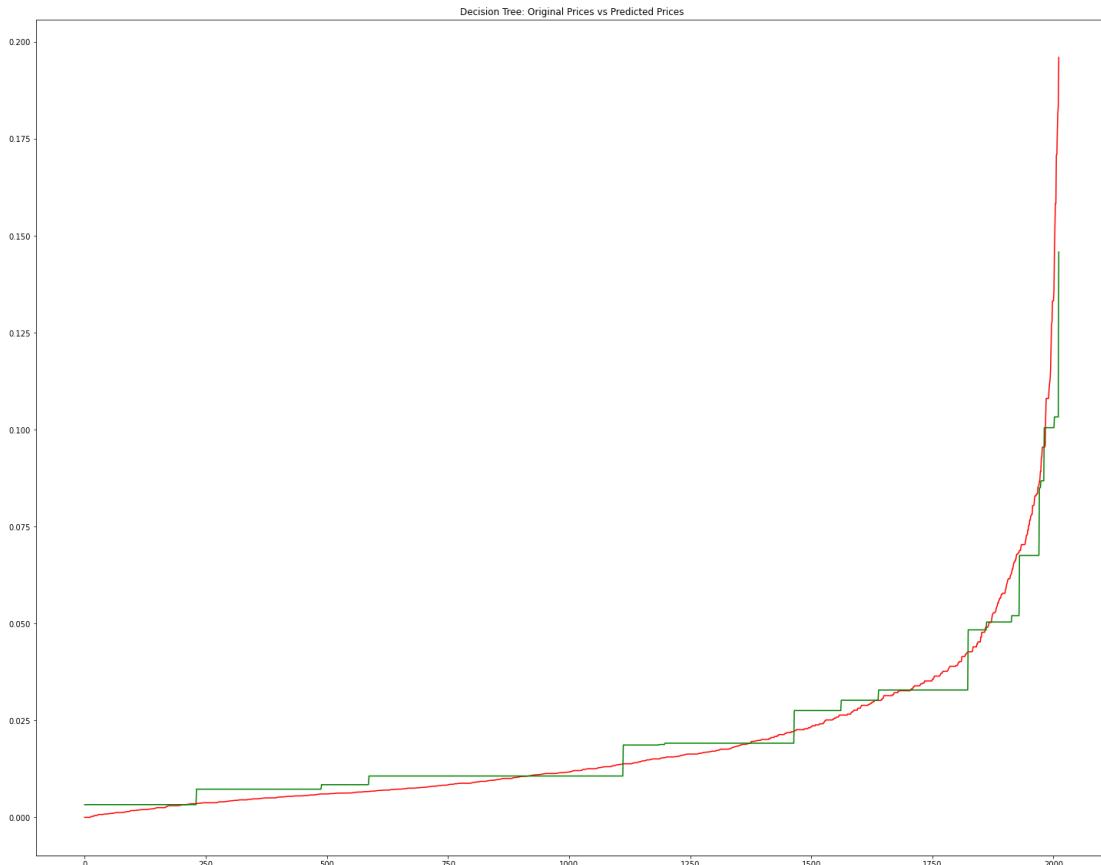
```
# !pip install -q dtreeviz -> for visualization not working currently
```

In [65]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_DT)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Decision Tree: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Decision-Tree-Predicted-vs-Original-Line', dpi=500, bbox_in
```

Out[65]:

Text(0.5, 1.0, 'Decision Tree: Original Prices vs Predicted Prices')

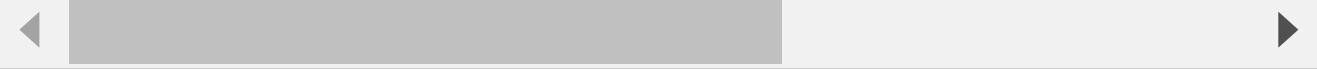


In [66]:

```
# Cross Validation
CVS_DT = cross_val_score(DT, data, price, cv=10).mean()
CVP_DT= cross_val_predict(DT, X_test, y_test, cv=10)
```

In [123]:

```
x = np.sort(y_test)[:7]
y = np.sort(CVP_DT)[:7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - Decision Tree: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Cross-Validation/Cross-Validation-Decision-Tree-Original-Pr
```



In []:

3.5 Random Forest

3.5.1 Random Forest: Model Training

In [68]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [69]:

```
RF = RandomForestRegressor(n_estimators=280, random_state=0)
RF.fit(X_train, y_train)
```

Out[69]:

```
RandomForestRegressor
RandomForestRegressor(n_estimators=280, random_state=0)
```

3.5.2 Random Forest: Testing

In [70]:

```
y_predict_RF = RF.predict(X_test)
print(y_predict_RF)
```

```
[0.01087331 0.0016187 0.01917732 ... 0.10886299 0.00687182 0.00510691]
```

3.5.3 Random Forest: Score / Results

In [71]:

```
RF_score = RF.score(X_test, y_test)
print(RF_score*100)
```

```
76.59534577701803
```

In [72]:

```
R2_Score_RF = r2_score(y_test, y_predict_RF)*100
MAE_RF = metrics.mean_absolute_error(y_test, y_predict_RF)*100
MSE_RF = metrics.mean_squared_error(y_test, y_predict_RF)*100
RSME_RF = np.sqrt(metrics.mean_squared_error(y_test, y_predict_RF))*100
print('R2 Score Linear Regression', R2_Score_RF)
print('MAE Ridge Regression:', MAE_RF)
print('MSE Ridge Regression:', MSE_RF)
print('RMSE Ridge Regression:', RSME_RF)
```

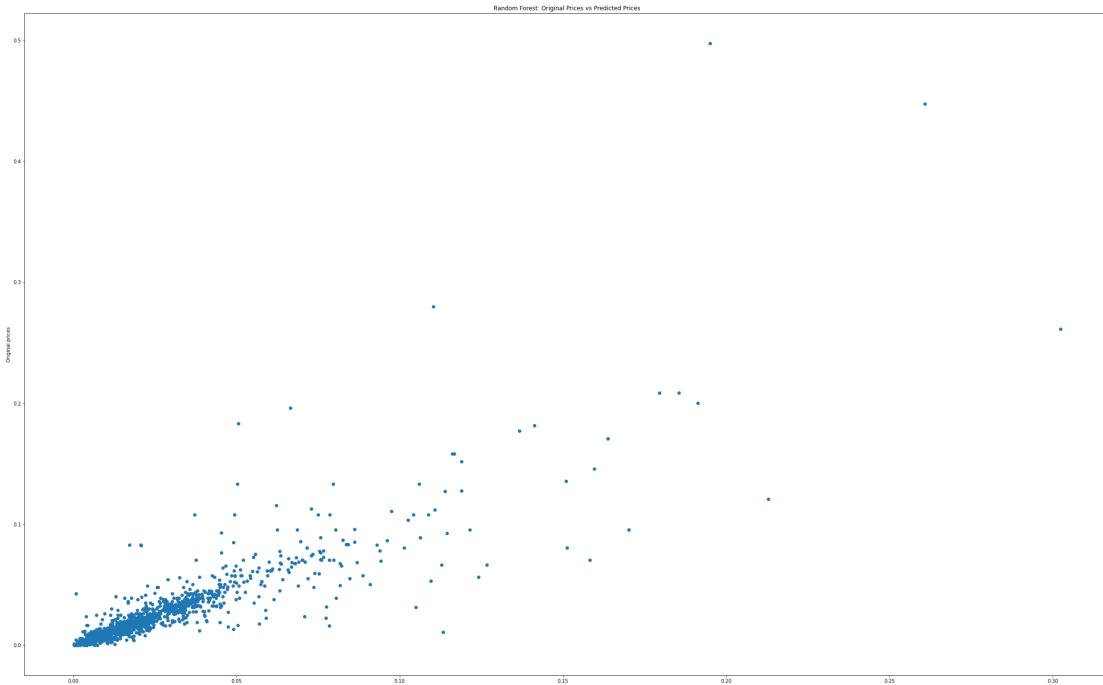
```
R2 Score Linear Regression 76.59534577701803
MAE Ridge Regression: 0.4428914597381189
MSE Ridge Regression: 0.018892145694148203
RMSE Ridge Regression: 1.3744870204606592
```

In [73]:

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_RF, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("Random Forest: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/Random-Forest-Predicted-vs-Original-Scattered', dpi=500, bbox
```

Out[73]:

Text(0.5, 1.0, 'Random Forest: Original Prices vs Predicted Prices')

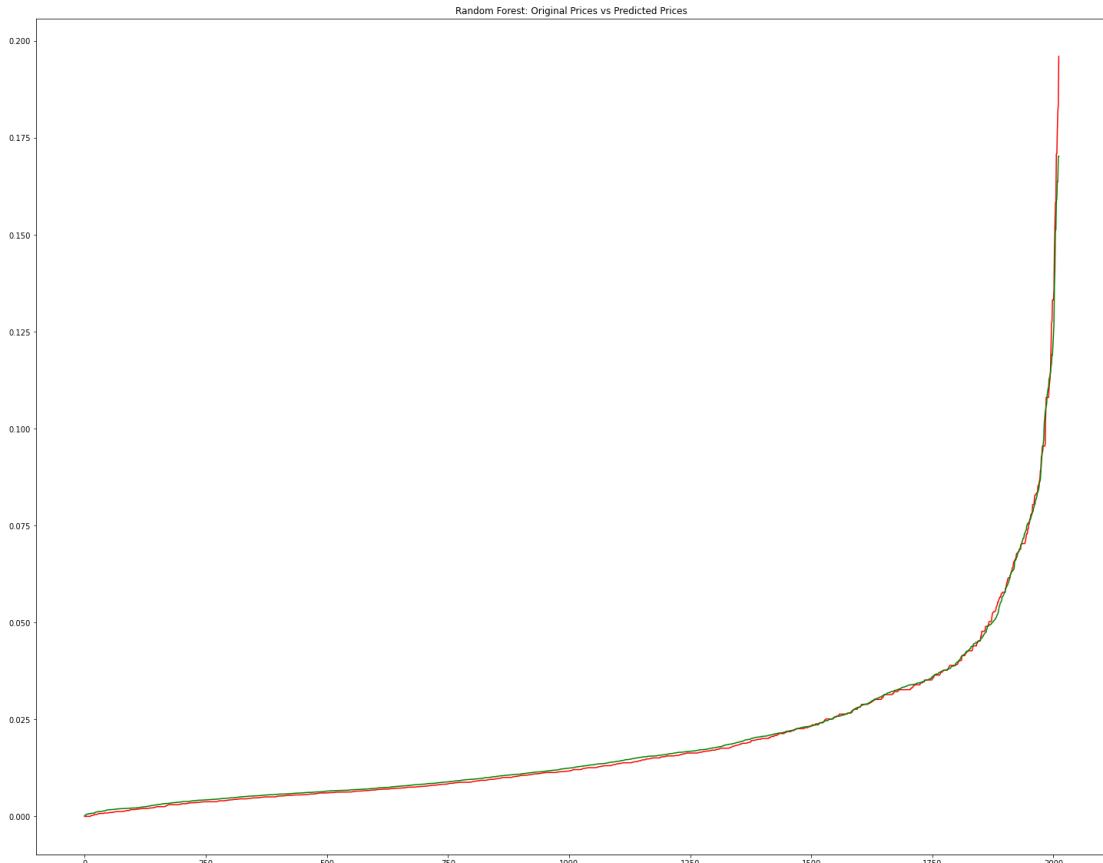


In [74]:

```
# Confusion Matrix<br>
# cm_RF = confusion_matrix(y_test, y_predict_RF)<br>
# cm_RF
```

In [75]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_RF)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Random Forest: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Random-Forest-Predicted-vs-Original-Line', dpi=500, bbox_in
```

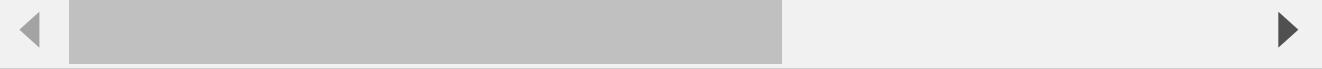


In [76]:

```
# Cross Validation
CVS_RF = cross_val_score(RF, data, price, cv=10).mean()
CVP_RF = cross_val_predict(RF, X_test, y_test, cv=10)
```

In [124]:

```
x = np.sort(y_test)[-7]
y = np.sort(CVP_RF)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - Random Forest: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Cross-Validation/Cross-Validation-Random-Forest-Original-Pr
```



In []:

3.6 K Nearest Neighbour: KNN

3.6.1 KNN: Model Training

In [78]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [79]:

```
KNR = KNeighborsRegressor(100)
KNR.fit(X_train, y_train)
```

Out[79]:

```
▼       KNeighborsRegressor
KNeighborsRegressor(n_neighbors=100)
```

3.6.2 KNN: Testing

In [80]:

```
y_predict_KNR = KNR.predict(X_test)
print(y_predict_KNR)
```

```
[0.00564967 0.00339726 0.0214409 ... 0.02243304 0.00515342 0.0050655 ]
```

3.6.3 KNN: Score / Results

In [81]:

```
KNR_score = KNR.score(X_test, y_test)
print(KNR_score*100)
```

```
12.99759718134954
```

In [82]:

```
R2_Score_KNR = r2_score(y_test, y_predict_KNR)*100
MAE_KNR = metrics.mean_absolute_error(y_test, y_predict_KNR)*100
MSE_KNR = metrics.mean_squared_error(y_test, y_predict_KNR)*100
RSME_KNR = np.sqrt(metrics.mean_squared_error(y_test, y_predict_KNR))*100
print('R2 Score Linear Regression', R2_Score_KNR)
print('MAE Ridge Regression:', MAE_KNR)
print('MSE Ridge Regression:', MSE_KNR)
print('RMSE Ridge Regression:', RSME_KNR)
```

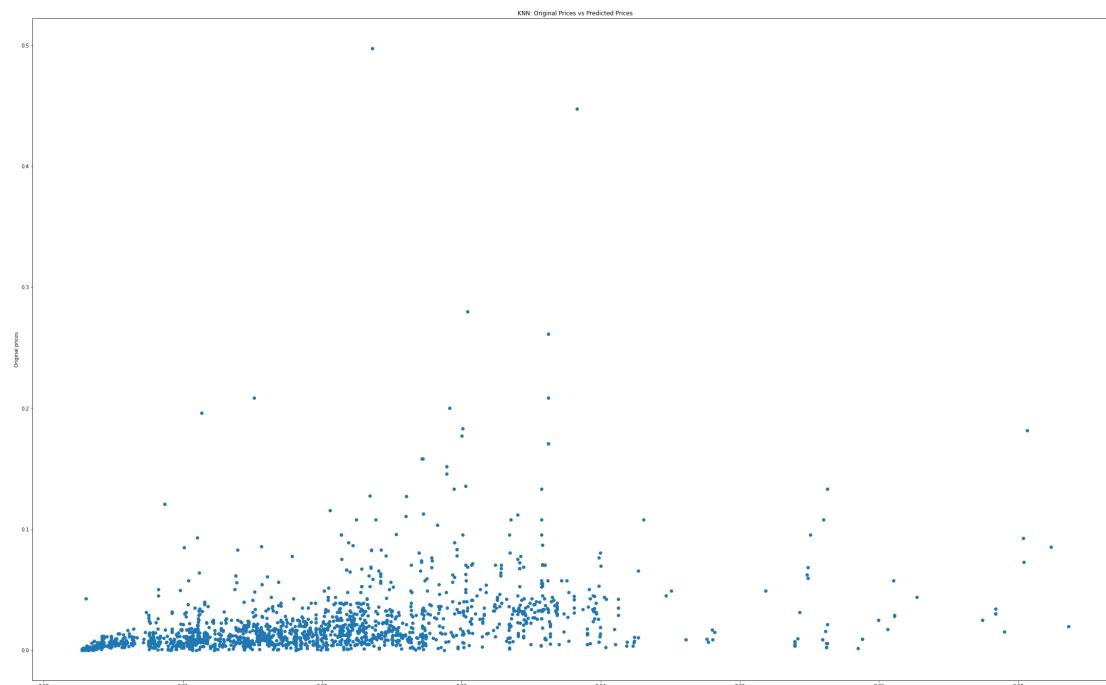
```
R2 Score Linear Regression 12.99759718134954
MAE Ridge Regression: 1.2766304684081626
MSE Ridge Regression: 0.0702280005562713
RMSE Ridge Regression: 2.6500566136645327
```

In [83]:

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_KNR, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("KNN: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/KNN-Predicted-vs-Original-Scattered', dpi=500, bbox_inches='
```

Out[83]:

Text(0.5, 1.0, 'KNN: Original Prices vs Predicted Prices')

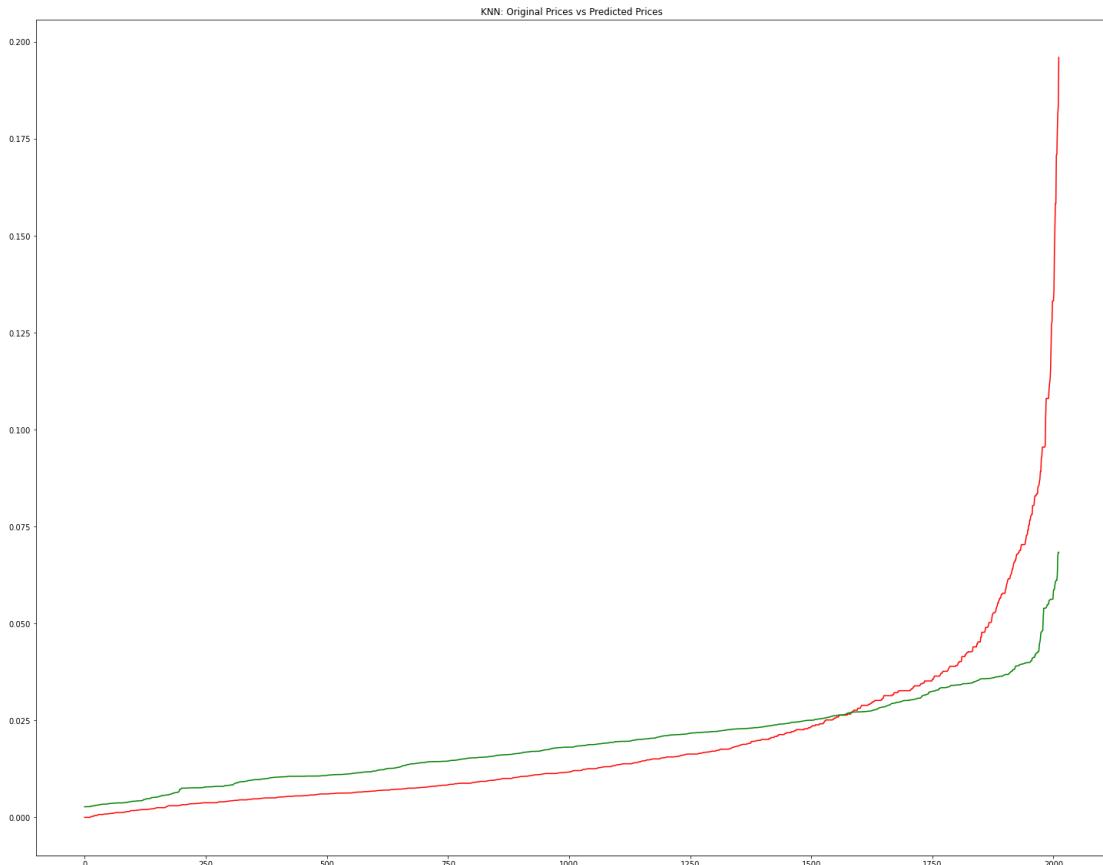


In [84]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_KNR)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("KNN: Original Prices vs Predicted Prices")
# plt.savefig('../Results/KNN-Predicted-vs-Original-Line', dpi=500, bbox_inches='tight')
```

Out[84]:

Text(0.5, 1.0, 'KNN: Original Prices vs Predicted Prices')



In [85]:

```
# Cross Validation
CVS_KNR = cross_val_score(KNR, data, price, cv=10).mean()
CVP_KNR = cross_val_predict(KNR, X_test, y_test, cv=10)
```

In [125]:

```
x = np.sort(y_test)[-7]
y = np.sort(CVP_KNR)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - KNR: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Cross-Validation/Cross-Validation-KNR-Original-Prices-vs-Predicted-Prices')
```



In []:

3.7 Support Vector Machine: SVM

3.7.1 SVM: Model Training

In [87]:

```
from sklearn.svm import SVR
```

In [88]:

```
SVM = SVR()
SVM.fit(X_train,y_train)
```

Out[88]:

```
SVR
SVR()
```

3.7.2 SVM: Testing

In [89]:

```
y_predict_SVM = SVM.predict(X_test)
print(y_predict_SVM)
```

[0.10100313 0.10000524 0.10183418 ... 0.10289083 0.1007649 0.10076457]

3.7.3 SVM: Score / Results

In [90]:

```
SVM_score = SVM.score(X_test, y_test)
print(SVM_score*100)
```

-810.8921150682211

In [91]:

```
R2_Score_SVM = r2_score(y_test, y_predict_SVM)*100
MAE_SVM = metrics.mean_absolute_error(y_test, y_predict_SVM)*100
MSE_SVM = metrics.mean_squared_error(y_test, y_predict_SVM)*100
RSME_SVM = np.sqrt(metrics.mean_squared_error(y_test, y_predict_SVM))*100
print('R2 Score Linear Regression', R2_Score_SVM)
print('MAE Ridge Regression:', MAE_SVM)
print('MSE Ridge Regression:', MSE_SVM)
print('RMSE Ridge Regression:', RSME_SVM)
```

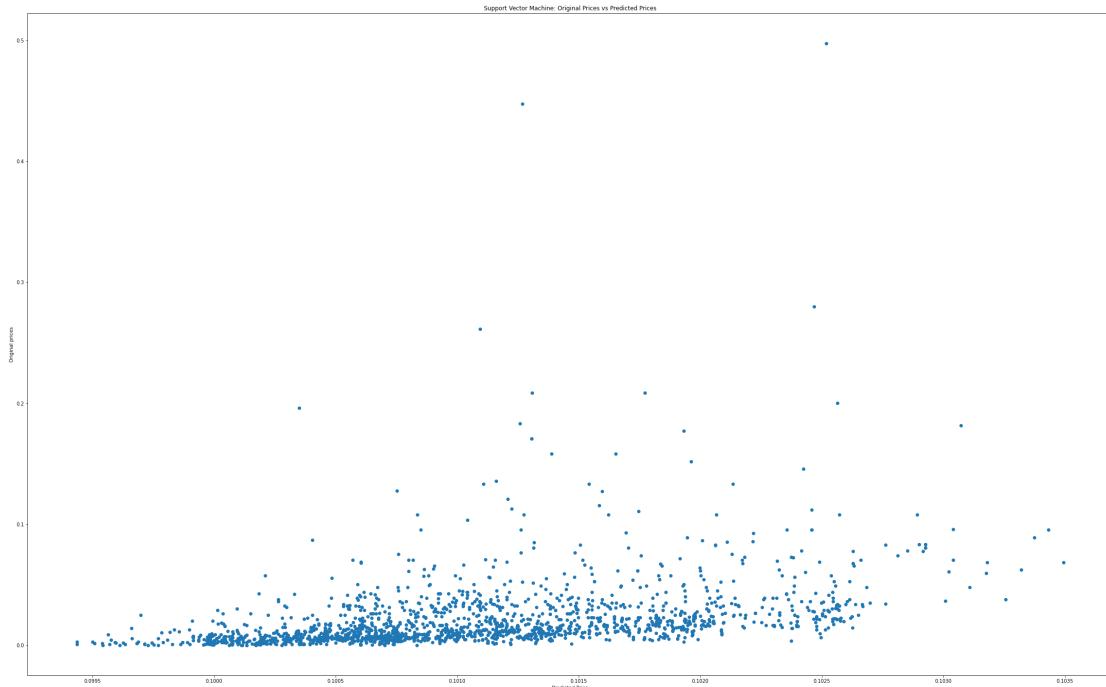
R2 Score Linear Regression -810.8921150682211
MAE Ridge Regression: 8.333134591601446
MSE Ridge Regression: 0.7352685660539143
RMSE Ridge Regression: 8.5747802657206

In [92]:

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_SVM, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("Support Vector Machine: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/Support-Vector-Machine-Predicted-vs-Original-Scattered', dpi
```

Out[92]:

```
Text(0.5, 1.0, 'Support Vector Machine: Original Prices vs Predicted Prices')
```

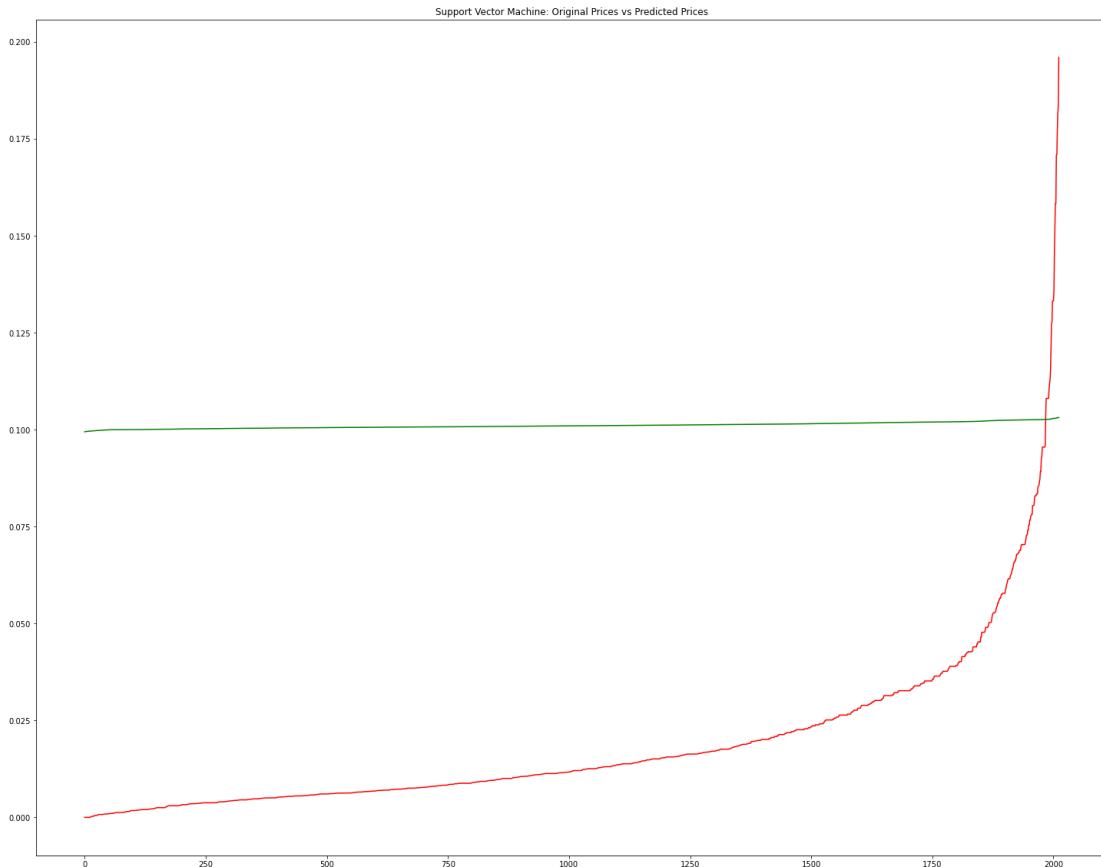


In [93]:

```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_SVM)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Support Vector Machine: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Support-Vector-Machine-Predicted-vs-Original-Line', dpi=500,
```

Out[93]:

Text(0.5, 1.0, 'Support Vector Machine: Original Prices vs Predicted Prices')

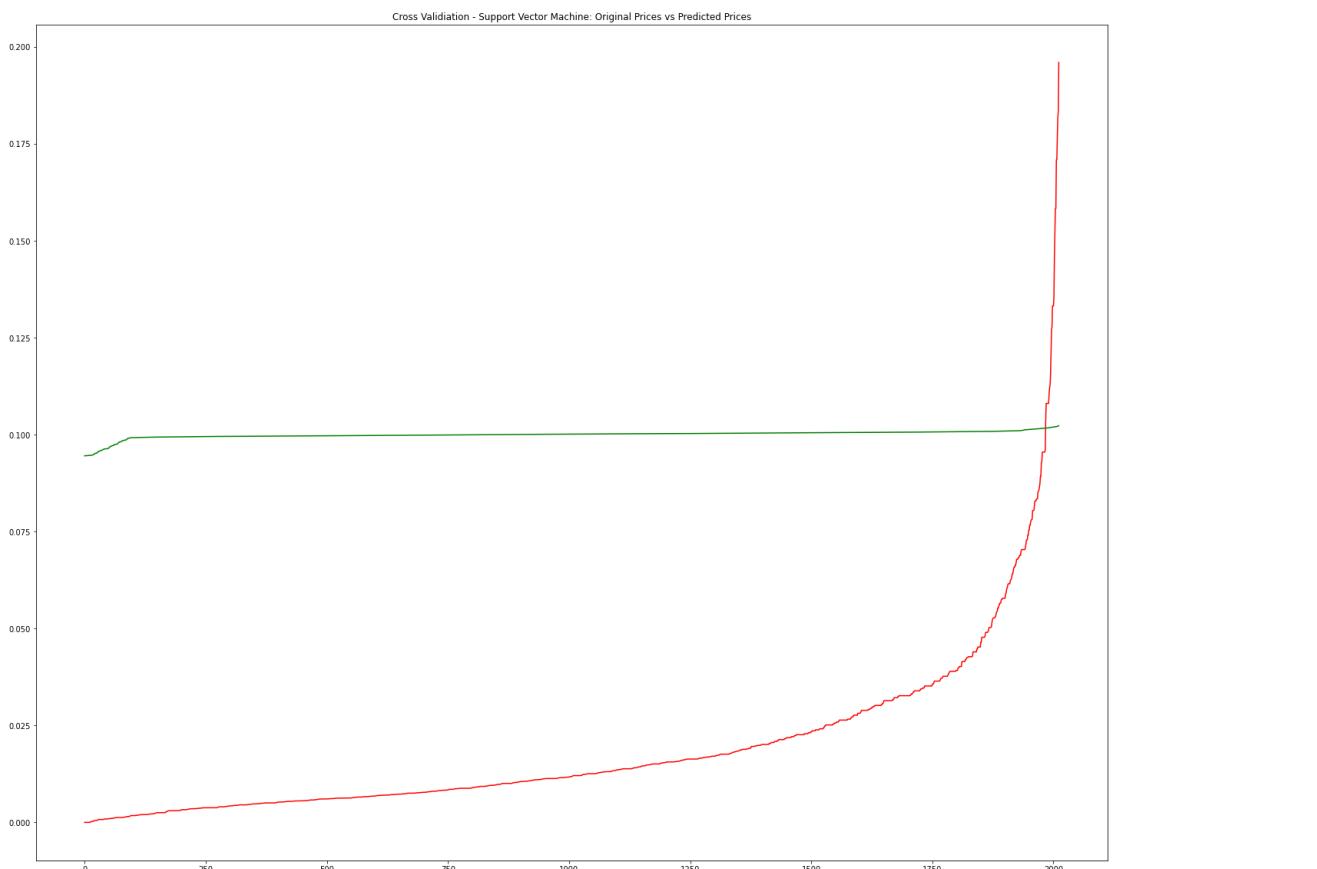


In [94]:

```
# Cross Validation
CVS_SVM = cross_val_score(SVM, data, price, cv=10).mean()
CVP_SVM = cross_val_predict(SVM, X_test, y_test, cv=10)
```

In [126]:

```
x = np.sort(y_test)[-7]
y = np.sort(CVP_SVM)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - Support Vector Machine: Original Prices vs Predicted Pr
# plt.savefig('../Results/Cross-Validation/Cross-Validation-Support-Vector-Machine-Or
```



In []:

3.8 Extreme Gradient Boost: XGBoost

3.8.1 XGBoost: Model Training

In [96]:

```
from xgboost import XGBRegressor
```

In [97]:

```
XGB = XGBRegressor()
XGB.fit(X_train, y_train)
```

Out[97]:

```
▼          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_by_
tree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='de
pthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_on
ehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_chil
```

3.8.2 XGBoost: Testing

In [98]:

```
y_predict_XGB = XGB.predict(X_test)
print(y_predict_XGB)
```

```
[0.01302374 0.00124807 0.01904653 ... 0.09009115 0.00688615 0.0061185 ]
```

3.8.3 XGBoost: Score / Results

In [99]:

```
XGB_score = XGB.score(X_test, y_test)
print(XGB_score*100)
```

```
78.15178470420672
```

In [100]:

```
R2_Score_XGB = r2_score(y_test, y_predict_XGB)*100
MAE_XGB = metrics.mean_absolute_error(y_test, y_predict_XGB)*100
MSE_XGB = metrics.mean_squared_error(y_test, y_predict_XGB)*100
RSME_XGB = np.sqrt(metrics.mean_squared_error(y_test, y_predict_XGB))*100
print('R2 Score Linear Regression', R2_Score_XGB)
print('MAE Ridge Regression:', MAE_XGB)
print('MSE Ridge Regression:', MSE_XGB)
print('RMSE Ridge Regression:', RSME_XGB)
```

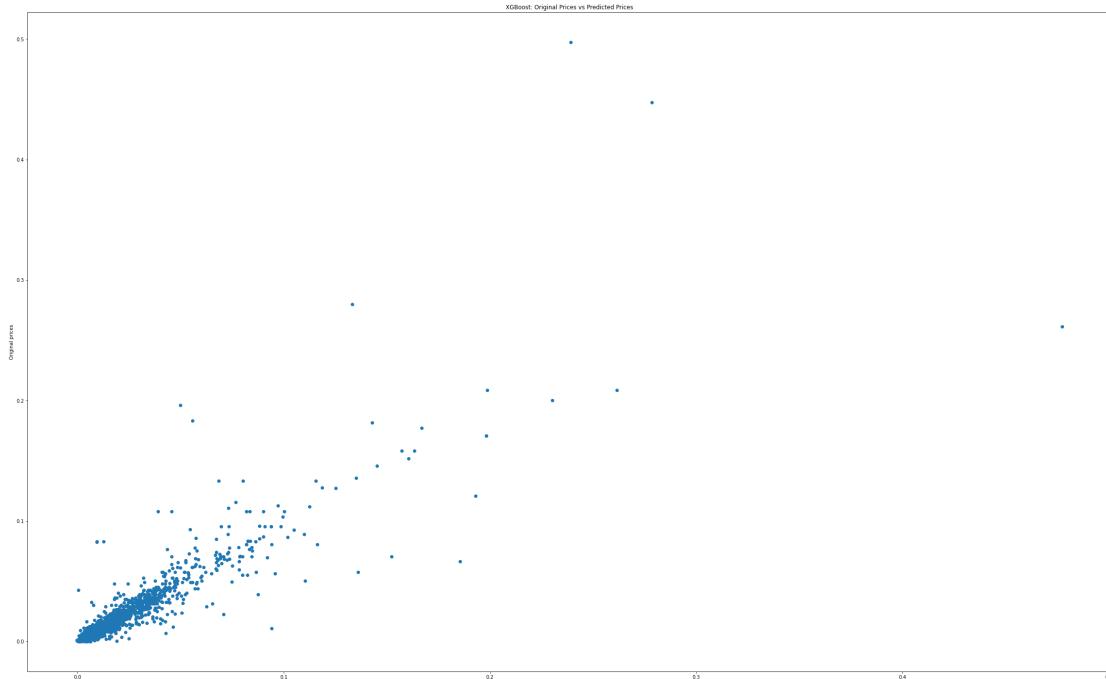
R2 Score Linear Regression 78.15178470420672
MAE Ridge Regression: 0.4448213764720972
MSE Ridge Regression: 0.017635794256679885
RMSE Ridge Regression: 1.3279982777353248

In [101]:

```
plt.figure(figsize= (40,25))
plt.scatter(y_predict_XGB, y_test)
# plt.plot(y_test, LR.predict(X_test), color='red', linestyle='--')
plt.xlabel("Predicted Price")
plt.ylabel("Original prices")
plt.title("XGBoost: Original Prices vs Predicted Prices")
# plt.show()
# plt.savefig('../Results/XGBoost-Predicted-vs-Original-Scattered', dpi=500, bbox_inches
```

Out[101]:

Text(0.5, 1.0, 'XGBoost: Original Prices vs Predicted Prices')



In []:



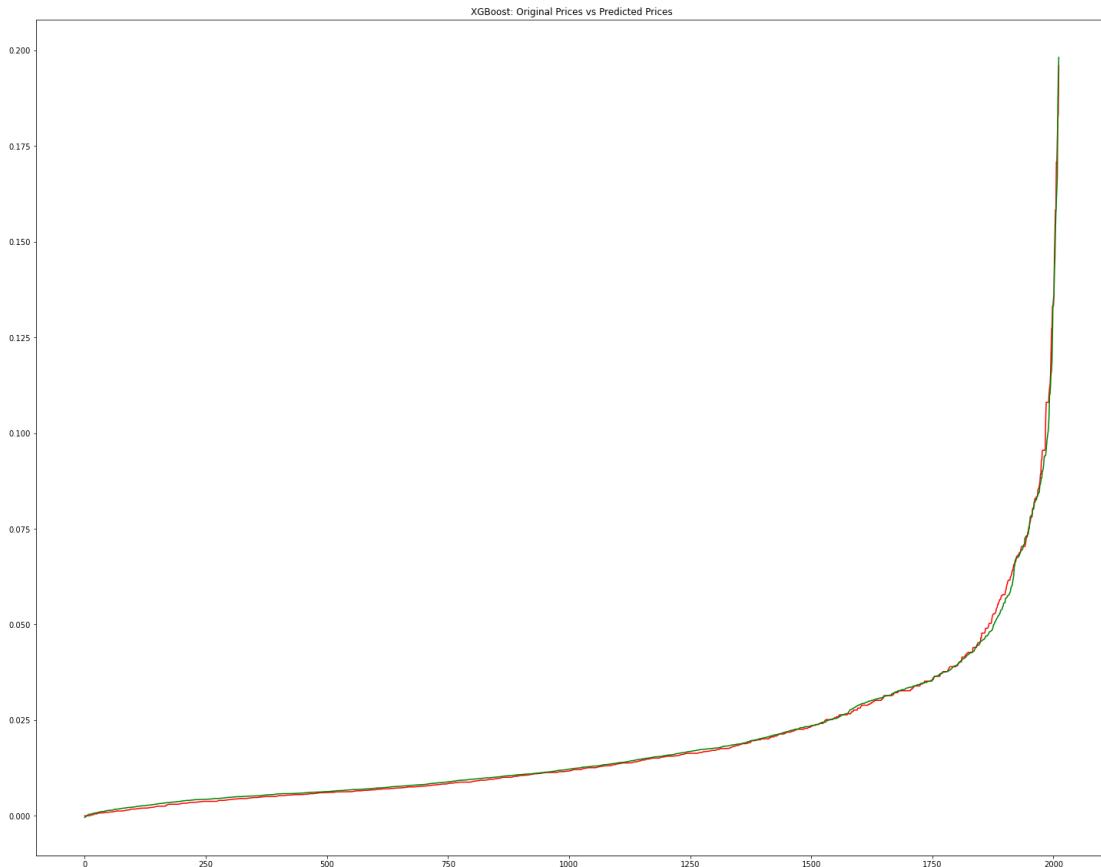
In [102]:



```
x = np.sort(y_test)[-7]
y = np.sort(y_predict_XGB)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("XGBoost: Original Prices vs Predicted Prices")
# plt.savefig('../Results/XGBoost-Predicted-vs-Original-Line', dpi=500, bbox_inches='tight')
```

Out[102]:

Text(0.5, 1.0, 'XGBoost: Original Prices vs Predicted Prices')



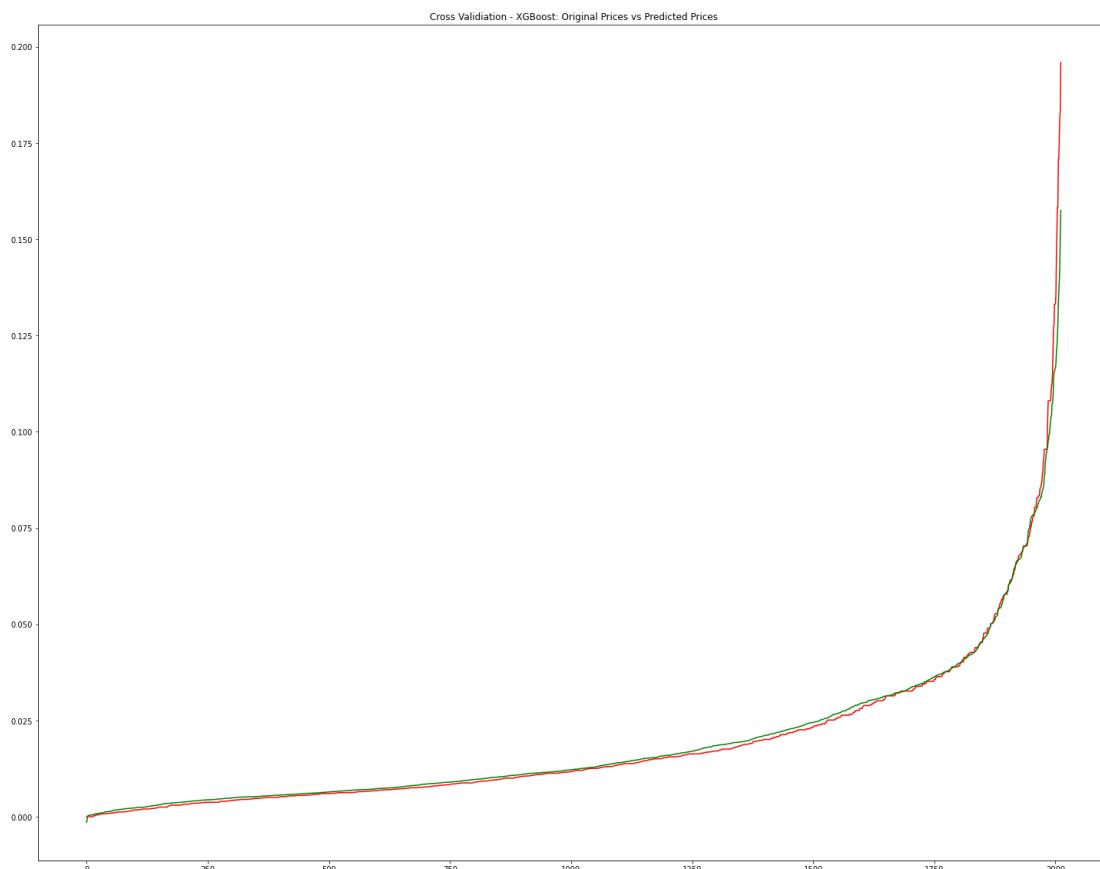
In [103]:



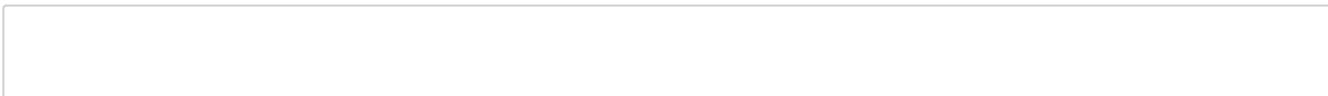
```
# Cross Validation
CVS_XGB = cross_val_score(XGB, data, price, cv=10).mean()
CVP_XGB = cross_val_predict(XGB, X_test, y_test, cv=10)
```

In [127]:

```
x = np.sort(y_test)[-7]
y = np.sort(CVP_XGB)[-7]
plt.figure(figsize=(25,20))
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.title("Cross Validation - XGBoost: Original Prices vs Predicted Prices")
# plt.savefig('../Results/Cross-Validation/Cross-Validation-XGBoost-Original-Prices-v
```



In []:



4 Results

4.1 Score Comparision

In [115]:

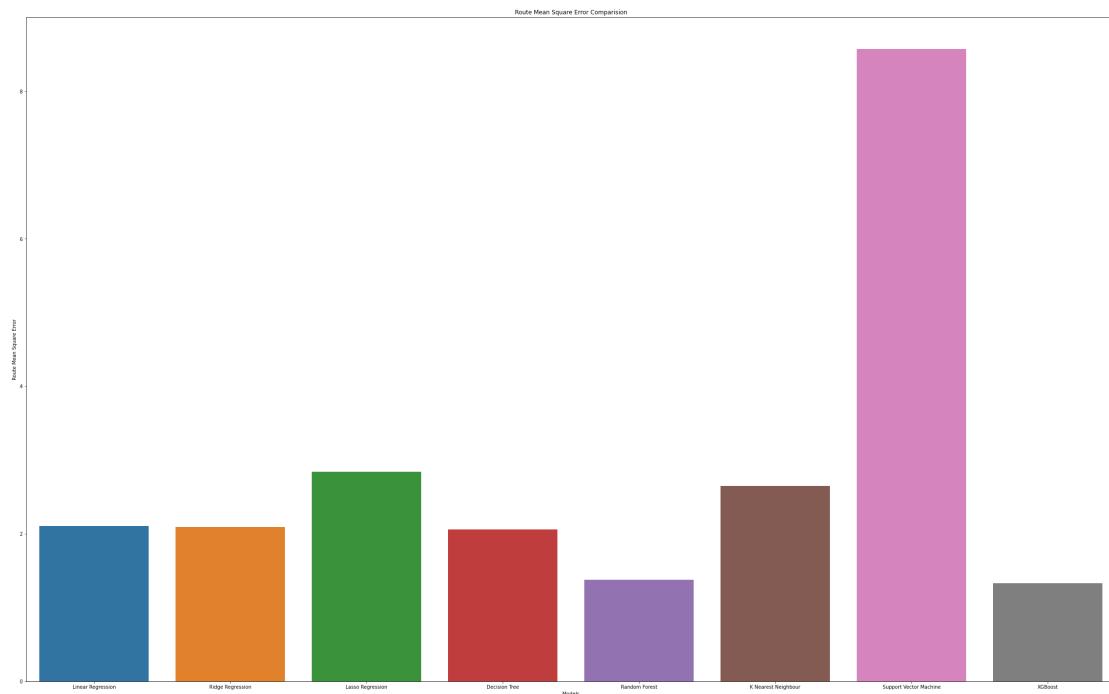
```
All_result = pd.DataFrame.from_dict({"Models" : ["Linear Regression", "Ridge Regression", "Lasso Regression"],  
                                      "Mean Absolute Error" : [MAE_LR, MAE_RR, MAE_Lasso],  
                                      "Mean Square Error" : [MSE_LR, MSE_RR, MSE_Lasso_R],  
                                      "Route Mean Square Error" : [RSME_LR, RSME_RR, RSM_Lasso_R],  
                                      "Score": [LR_score*100, RR_score*100, Lasso_R_score*100],  
                                      "Cross Validation Score" : [CVS_LR*100, CVS_RR*100, CVS_Lasso_R*100]})  
All_result.head(8)
```

Out[115]:

	Models	Mean Absolute Error	Mean Square Error	Route Mean Square Error	Score
0	Linear Regression	1.038439	0.044255	2.103696	45.173945
1	Ridge Regression	1.024697	0.043840	2.093798	45.688682
2	Lasso Regression	1.620546	0.080863	2.843636	-0.177224
3	Decision Tree	0.830301	0.042440	2.060108	47.422361
4	Random Forest	0.442891	0.018892	1.374487	76.595346
5	K Nearest Neighbour	1.276630	0.070228	2.650057	12.997597
6	Support Vector Machine	8.333135	0.735269	8.574780	-810.892115
7	XGBoost	0.444821	0.017636	1.327998	78.151785

In [106]:

```
plt.figure(figsize= (40,25))
sns.barplot(x='Models',y='Route Mean Square Error',data=All_result)
plt.title('Route Mean Square Error Comparision')
# plt.savefig('../Results/Route-Mean-Square-Error-Comparision', dpi=500, bbox_inches='
```



In []:

In [128]:



```
plt.figure(figsize=(60,25))

# Linear Regression
plt.subplot(2,4,1)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_LR)[-7]
plt.title("Linear Regression")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# Ridge Regression
plt.subplot(2,4,2)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_RR)[-7]
plt.title("Ridge Regression")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# Lasso Regression
plt.subplot(2,4,3)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_Lasso_R)[-7]
plt.title("Lasso Regression")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# Decision Tree
plt.subplot(2,4,4)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_DT)[-7]
plt.title("Decision Tree")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# Random Forest
plt.subplot(2,4,5)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_RF)[-7]
plt.title("Random Forest")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# K Nearest Neighbour
plt.subplot(2,4,6)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_KNR)[-7]
plt.title("K Nearest Neighbour")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# Support Vector Machine
plt.subplot(2,4,7)
x = np.sort(y_test)[-7]
```

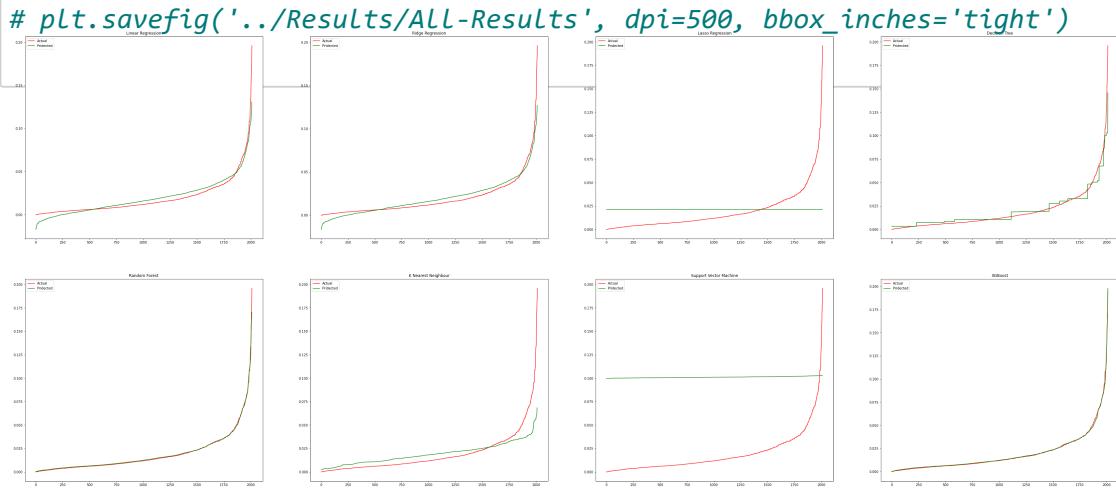
```

y = np.sort(y_predict_SVM)[-7:]
plt.title("Support Vector Machine")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# XGBoost
plt.subplot(2,4,8)
x = np.sort(y_test)[-7:]
y = np.sort(y_predict_XGB)[-7:]
plt.title("XGBoost")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.legend()

# plt.savefig('../Results/All-Results', dpi=500, bbox_inches='tight')

```



In [130]:



```
plt.figure(figsize=(60,25))

# Linear Regression
plt.subplot(2,4,1)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_LR)[-7]
z = np.sort(CVP_LR)[-7]
plt.title("Linear Regression")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Normal Pridected", color='green')
plt.plot(z, label = "Cross Validation Pridected", color='blue')
# plt.legend()

# Ridge Regression
plt.subplot(2,4,2)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_RR)[-7]
z = np.sort(CVP_RR)[-7]
plt.title("Ridge Regression")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Pridected", color='green')
plt.plot(z, label = "Cross Validation Pridected", color='blue')
# plt.legend()

# Lasso Regression
plt.subplot(2,4,3)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_Lasso_R)[-7]
z = np.sort(CVP_Lasso_R)[-7]
plt.title("Lasso Regression")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Pridected", color='green')
plt.plot(z, label = "Cross Validation Pridected", color='blue')
# plt.legend()

# Decision Tree
plt.subplot(2,4,4)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_DT)[-7]
z = np.sort(CVP_DT)[-7]
plt.title("Decision Tree")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Pridected", color='green')
plt.plot(z, label = "Cross Validation Pridected", color='blue')
# plt.legend()

# Random Forest
plt.subplot(2,4,5)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_RF)[-7]
z = np.sort(CVP_RF)[-7]
plt.title("Random Forest")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Pridected", color='green')
plt.plot(z, label = "Cross Validation Pridected", color='blue')
# plt.legend()

# K Nearest Neighbour
plt.subplot(2,4,6)
```

```

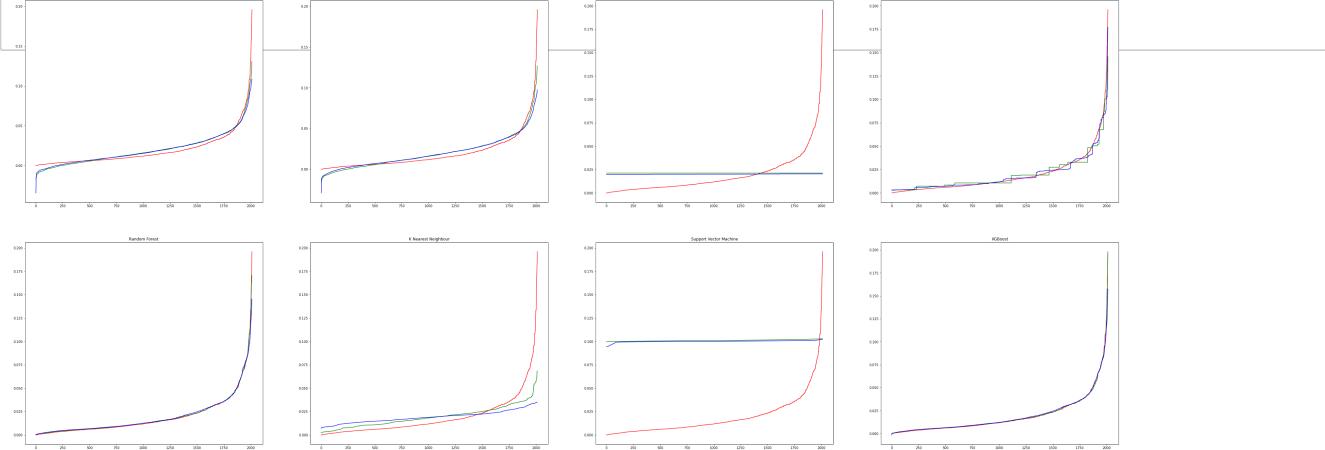
x = np.sort(y_test)[-7]
y = np.sort(y_predict_KNR)[-7]
z = np.sort(CVP_KNR)[-7]
plt.title("K Nearest Neighbour")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.plot(z, label = "Cross Validation Predicted", color='blue')
# plt.legend()

# Support Vector Machine
plt.subplot(2,4,7)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_SVM)[-7]
z = np.sort(CVP_SVM)[-7]
plt.title("Support Vector Machine")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.plot(z, label = "Cross Validation Predicted", color='blue')
# plt.legend()

# XGBoost
plt.subplot(2,4,8)
x = np.sort(y_test)[-7]
y = np.sort(y_predict_XGB)[-7]
z = np.sort(CVP_XGB)[-7]
plt.title("XGBoost")
plt.plot(x, label = "Actual", color="red")
plt.plot(y, label = "Predicted", color='green')
plt.plot(z, label = "Cross Validation Predicted", color='blue')
# plt.legend()

```

```
plt.savefig('../Results/Cross-Validation/All-Results-Cross-Validation2', dpi=500, bbox
```



In []:

