DS project Group-12

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_selection import f_regression,SelectKBest
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV
```

In [2]:

```
df = pd.read_csv('Crime_Incidents_in_2012.csv')
# df2 = pd.read_csv('Crime_Incidents_in_2013.csv')
# df3 = pd.read_csv('Crime_Incidents_in_2014.csv')
# df4 = pd.read_csv('Crime_Incidents_in_2015.csv')
# df5 = pd.read_csv('Crime_Incidents_in_2016.csv')
# df6 = pd.read_csv('Crime_Incidents_in_2017.csv')
# df7 = pd.read_csv('Crime_Incidents_in_2018.csv')
# df8 = pd.read_csv('Crime_Incidents_in_2019.csv')
# df9 = pd.read_csv('Crime_Incidents_in_2020.csv')
# df0 = pd.read_csv('Crime_Incidents_in_2021.csv')
```

In [3]:

```
# COLUMN_NAMES = ['X', 'Y', 'CCN', 'REPORT_DAT', 'SHIFT', 'METHOD', 'OFFENSE', 'BLOCK
#                   'YBLOCK', 'WARD', 'ANC', 'DISTRICT', 'PSA','NEIGHBORHOOD_CLUSTER',
#                   'VOTING_PRECINCT', 'LATITUDE', 'LONGITUDE', 'BID', 'START_DATE','EN
# df = pd.DataFrame(columns=COLUMN_NAMES)
# df =pd.concat([df1,df2, df3,df4,df5,df6,df7,df8,df9,df0],ignore_index =True)
```

In [4]:

```
# df.to_csv('data.csv',index=False)
```

In [5]:

```
#df=pd.read_csv('data.csv')
```

In [6]:

```
df=df.iloc[:500]
df.shape
```

Out[6]:

```
(500, 24)
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['X', 'Y', 'CCN', 'REPORT_DAT', 'SHIFT', 'METHOD', 'OFFENSE', 'BLOCK',
       'XBLOCK', 'YBLOCK', 'WARD', 'ANC', 'DISTRICT', 'PSA',
       'NEIGHBORHOOD_CLUSTER', 'BLOCK_GROUP', 'CENSUS_TRACT',
       'VOTING_PRECINCT', 'LATITUDE', 'LONGITUDE', 'BID', 'START_DATE',
       'END_DATE', 'OBJECTID'],
      dtype='object')
```

In [8]:

```
df.head()
```

Out[8]:

	X	Y	CCN	REPORT_DAT	SHIFT	METHOD	OFFENSE	BLOCK	XBL
0	-76.999518	38.901924	9074624	2012-04-25T00:00:00.000Z	MIDNIGHT	OTHERS	SEX ABUSE	900 - 999 BLOCK OF 5TH STREET NE	40
1	-76.995541	38.905032	10123633	2012-02-29T00:00:00.000Z	MIDNIGHT	OTHERS	SEX ABUSE	700 - 799 BLOCK OF FLORIDA AVENUE NE	40
2	-76.948897	38.885680	11102619	2012-05-14T00:00:00.000Z	MIDNIGHT	GUN	HOMICIDE	153 - 399 BLOCK OF RIDGE ROAD SE	40
3	-76.967571	38.855724	11141272	2012-06-25T00:00:00.000Z	MIDNIGHT	OTHERS	HOMICIDE	2800 - 2899 BLOCK OF BUENA VISTA TERRACE SE	40
4	-76.939620	38.910718	11158196	2012-01-05T00:00:00.000Z	MIDNIGHT	OTHERS	HOMICIDE	4280 - 4499 BLOCK OF DOUGLAS STREET NE	40

5 rows × 24 columns

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   X                     500 non-null    float64
 1   Y                     500 non-null    float64
 2   CCN                   500 non-null    int64  
 3   REPORT_DAT            500 non-null    object 
 4   SHIFT                 500 non-null    object 
 5   METHOD                500 non-null    object 
 6   OFFENSE               500 non-null    object 
 7   BLOCK                 500 non-null    object 
 8   XBLOCK                500 non-null    int64  
 9   YBLOCK                500 non-null    int64  
 10  WARD                  500 non-null    float64
 11  ANC                   500 non-null    object 
 12  DISTRICT              500 non-null    float64
 13  PSA                   500 non-null    float64
 14  NEIGHBORHOOD_CLUSTER  492 non-null    object 
 15  BLOCK_GROUP           499 non-null    object 
 16  CENSUS_TRACT          499 non-null    float64
 17  VOTING_PRECINCT       500 non-null    object 
 18  LATITUDE              500 non-null    float64
 19  LONGITUDE             500 non-null    float64
 20  BID                   76 non-null     object 
 21  START_DATE            500 non-null    object 
 22  END_DATE              496 non-null    object 
 23  OBJECTID              500 non-null    int64  
dtypes: float64(8), int64(4), object(12)
memory usage: 93.9+ KB
```

In [10]:
```
df.describe()
```

Out[10]:

	X	Y	CCN	XBLOCK	YBLOCK	WARD	DISTRICT	
count	500.000000	500.000000	5.000000e+02	500.000000	500.000000	500.000000	500.00000	5C
mean	-77.007607	38.905599	1.199100e+07	399341.046000	137526.588000	4.634000	3.80600	38
std	0.036525	0.034332	1.732524e+05	3167.849485	3811.243612	2.310567	1.99407	19
min	-77.091050	38.813478	9.074624e+06	392105.000000	127300.000000	1.000000	1.00000	10
25%	-77.032420	38.887592	1.200530e+07	397190.750000	135527.750000	2.000000	2.00000	20
50%	-77.008114	38.905650	1.200648e+07	399296.000000	137532.000000	5.000000	4.00000	40
75%	-76.983647	38.927541	1.200982e+07	401419.750000	139962.250000	7.000000	5.25000	60
max	-76.917789	38.986543	1.201348e+07	407132.000000	146512.000000	8.000000	7.00000	70

In [11]:
```
df.isnull().sum()
```

Out[11]:

```
X                         0
Y                         0
CCN                       0
REPORT_DAT                0
SHIFT                     0
METHOD                    0
OFFENSE                   0
BLOCK                     0
XBLOCK                    0
YBLOCK                    0
WARD                      0
ANC                       0
DISTRICT                  0
PSA                       0
NEIGHBORHOOD_CLUSTER      8
BLOCK_GROUP               1
CENSUS_TRACT              1
VOTING_PRECINCT           0
LATITUDE                  0
LONGITUDE                 0
BID                     424
START_DATE                0
END_DATE                  4
OBJECTID                  0
dtype: int64
```

In [12]:

```
#df.drop(['BID','OCTO_RECORD_ID'],axis=1,inplace=True)
df.drop(['BID'],axis=1,inplace=True)
# having more than 75% null values
```

In [13]:

```
df.shape
```

Out[13]:

```
(500, 23)
```

In [14]:

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

Out[14]:

```
X                        0
Y                        0
CCN                      0
REPORT_DAT               0
SHIFT                    0
METHOD                   0
OFFENSE                  0
BLOCK                    0
XBLOCK                   0
YBLOCK                   0
WARD                     0
ANC                      0
DISTRICT                 0
PSA                      0
NEIGHBORHOOD_CLUSTER     8
BLOCK_GROUP              1
CENSUS_TRACT             1
VOTING_PRECINCT          0
LATITUDE                 0
LONGITUDE                0
START_DATE               0
END_DATE                 4
OBJECTID                 0
dtype: int64
```

In [15]:

```
df.dropna(subset=['WARD','DISTRICT','PSA','NEIGHBORHOOD_CLUSTER','BLOCK_GROUP','CENSUS
```

In [16]:

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

Out[16]:

```
X                        0
Y                        0
CCN                      0
REPORT_DAT               0
SHIFT                    0
METHOD                   0
OFFENSE                  0
BLOCK                    0
XBLOCK                   0
YBLOCK                   0
WARD                     0
ANC                      0
DISTRICT                 0
PSA                      0
NEIGHBORHOOD_CLUSTER     0
BLOCK_GROUP              0
CENSUS_TRACT             0
VOTING_PRECINCT          0
LATITUDE                 0
LONGITUDE                0
START_DATE               0
END_DATE                 0
OBJECTID                 0
dtype: int64
```

In [17]:

```
df.shape
```

Out[17]:

```
(487, 23)
```

In [18]:

```
df['VOTING_PRECINCT'].head()
```

Out[18]:

```
0     Precinct 83
1     Precinct 83
3    Precinct 134
4     Precinct 92
5      Precinct 5
Name: VOTING_PRECINCT, dtype: object
```

In [19]:

```
df['NEIGHBORHOOD_CLUSTER'].head()
```

Out[19]:

```
0    Cluster 25
1    Cluster 25
3    Cluster 36
4    Cluster 29
5     Cluster 4
Name: NEIGHBORHOOD_CLUSTER, dtype: object
```

> ## Removing "Precinct" and "Cluster" from each record of VOTIING_PRECINCT and NEIGHBORHOOD_CLUSTER..

In [20]:

```
df['VOTING_PRECINCT']=df['VOTING_PRECINCT'].str.replace('Precinct ','').astype(int)
```

In [21]:

```
df['NEIGHBORHOOD_CLUSTER']=df['NEIGHBORHOOD_CLUSTER'].str.replace('Cluster ','').astyp
```

In [22]:

```
df['VOTING_PRECINCT'].head()
```

Out[22]:

```
0     83
1     83
3    134
4     92
5      5
Name: VOTING_PRECINCT, dtype: int32
```

In [23]:

```
df['NEIGHBORHOOD_CLUSTER'].head()
```

Out[23]:

```
0    25
1    25
3    36
4    29
5     4
Name: NEIGHBORHOOD_CLUSTER, dtype: int32
```

> ## Values in column X and Y is same as that of the column LATITUDE and LONGITUDE. So we can remove X and Y.

In [24]:

```
df.drop(['X','Y'],axis=1,inplace=True)
```

In [25]:

```
df.head()
```

Out[25]:

	CCN	REPORT_DAT	SHIFT	METHOD	OFFENSE	BLOCK	XBLOCK	YBLOCK	WAF
0	9074624	2012-04-25T00:00:00.000Z	MIDNIGHT	OTHERS	SEX ABUSE	900 - 999 BLOCK OF 5TH STREET NE	400042	137118	6
1	10123633	2012-02-29T00:00:00.000Z	MIDNIGHT	OTHERS	SEX ABUSE	700 - 799 BLOCK OF FLORIDA AVENUE NE	400387	137463	6
3	11141272	2012-06-25T00:00:00.000Z	MIDNIGHT	OTHERS	HOMICIDE	2800 - 2899 BLOCK OF BUENA VISTA TERRACE SE	402815	131990	8
4	11158196	2012-01-05T00:00:00.000Z	MIDNIGHT	OTHERS	HOMICIDE	4280 - 4499 BLOCK OF DOUGLAS STREET NE	405237	138096	7
5	12005414	2012-01-11T18:52:00.000Z	EVENING	OTHERS	THEFT/OTHER	1500 - 1599 BLOCK OF 32ND STREET NW	394480	138000	2

5 rows × 21 columns

> ### Also dates are represented as object type, it should be converted into datetime datatype.

In [26]:
```
df['START_DATE']=pd.to_datetime(df['START_DATE'])
```

In [27]:
```
df['END_DATE']=pd.to_datetime(df['END_DATE'])
```

In [28]:
```
df['REPORT_DAT']=pd.to_datetime(df['REPORT_DAT'])
```

In [29]:
```
df['OFFENSE'].unique()
```

Out[29]:
```
array(['SEX ABUSE', 'HOMICIDE', 'THEFT/OTHER',
       'ASSAULT W/DANGEROUS WEAPON', 'ROBBERY', 'THEFT F/AUTO',
       'MOTOR VEHICLE THEFT', 'BURGLARY'], dtype=object)
```

In [30]:
```
# Import label encoder
from sklearn import preprocessing as pp
```

```
offense_encoder = pp.LabelEncoder()
df['OFFENSE']=offense_encoder.fit_transform(df['OFFENSE'])
```

In [31]:

```
df['OFFENSE'].head()
```

Out[31]:

```
0    5
1    5
3    2
4    2
5    7
Name: OFFENSE, dtype: int32
```

> ## Same like that convert BLOCK,ANC column to int32

In [32]:

```
df.head()
```

Out[32]:

	CCN	REPORT_DAT	SHIFT	METHOD	OFFENSE	BLOCK	XBLOCK	YBLOCK	WARD	AI
0	9074624	2012-04-25 00:00:00+00:00	MIDNIGHT	OTHERS	5	900 - 999 BLOCK OF 5TH STREET NE	400042	137118	6.0	
1	10123633	2012-02-29 00:00:00+00:00	MIDNIGHT	OTHERS	5	700 - 799 BLOCK OF FLORIDA AVENUE NE	400387	137463	6.0	
3	11141272	2012-06-25 00:00:00+00:00	MIDNIGHT	OTHERS	2	2800 - 2899 BLOCK OF BUENA VISTA TERRACE SE	402815	131990	8.0	
4	11158196	2012-01-05 00:00:00+00:00	MIDNIGHT	OTHERS	2	4280 - 4499 BLOCK OF DOUGLAS STREET NE	405237	138096	7.0	
5	12005414	2012-01-11 18:52:00+00:00	EVENING	OTHERS	7	1500 - 1599 BLOCK OF 32ND STREET NW	394480	138000	2.0	

5 rows × 21 columns

In [33]:

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

Out[33]:

```
459
```

In [34]:

```
block_encoder = pp.LabelEncoder()
df['BLOCK']=block_encoder.fit_transform(df['BLOCK'])

anc_encoder = pp.LabelEncoder()
df['ANC']=anc_encoder.fit_transform(df['ANC'])
```

In [35]:

```
df['BLOCK'].head()
```

Out[35]:

```
0    423
1    397
3    218
4    316
5    102
Name: BLOCK, dtype: int32
```

In [36]:

```
df['ANC'].head()
```

Out[36]:

```
0    27
1    27
3    36
4    32
5     8
Name: ANC, dtype: int32
```

In [37]:

```
df.head()
```

Out[37]:

	CCN	REPORT_DAT	SHIFT	METHOD	OFFENSE	BLOCK	XBLOCK	YBLOCK	WARD	ANC
0	9074624	2012-04-25 00:00:00+00:00	MIDNIGHT	OTHERS	5	423	400042	137118	6.0	27
1	10123633	2012-02-29 00:00:00+00:00	MIDNIGHT	OTHERS	5	397	400387	137463	6.0	27
3	11141272	2012-06-25 00:00:00+00:00	MIDNIGHT	OTHERS	2	218	402815	131990	8.0	36
4	11158196	2012-01-05 00:00:00+00:00	MIDNIGHT	OTHERS	2	316	405237	138096	7.0	32
5	12005414	2012-01-11 18:52:00+00:00	EVENING	OTHERS	7	102	394480	138000	2.0	8

5 rows × 21 columns

> ## One Hot Encoding SHIFT and METHOD using get_dummies() function. It will create new columns with value 0 and 1 only.

In [38]:

```
df['SHIFT'].unique()
```

Out[38]:

```
array(['MIDNIGHT', 'EVENING', 'DAY'], dtype=object)
```

In [39]:
```
df['METHOD'].unique()
```

Out[39]:
```
array(['OTHERS', 'GUN', 'KNIFE'], dtype=object)
```

In [40]:
```
shift_encoder = pp.LabelEncoder()
df['SHIFT']=shift_encoder.fit_transform(df['SHIFT'])

met_encoder = pp.LabelEncoder()
df['METHOD']=met_encoder.fit_transform(df['METHOD'])
```

In [41]:
```
df.head()
```

Out[41]:

	CCN	REPORT_DAT	SHIFT	METHOD	OFFENSE	BLOCK	XBLOCK	YBLOCK	WARD	ANC	...
0	9074624	2012-04-25 00:00:00+00:00	2	2	5	423	400042	137118	6.0	27	...
1	10123633	2012-02-29 00:00:00+00:00	2	2	5	397	400387	137463	6.0	27	...
3	11141272	2012-06-25 00:00:00+00:00	2	2	2	218	402815	131990	8.0	36	...
4	11158196	2012-01-05 00:00:00+00:00	2	2	2	316	405237	138096	7.0	32	...
5	12005414	2012-01-11 18:52:00+00:00	1	2	7	102	394480	138000	2.0	8	...

5 rows × 21 columns

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 487 entries, 0 to 499
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype              
---  ------                --------------  -----              
 0   CCN                   487 non-null    int64              
 1   REPORT_DAT            487 non-null    datetime64[ns, UTC]
 2   SHIFT                 487 non-null    int32              
 3   METHOD                487 non-null    int32              
 4   OFFENSE               487 non-null    int32              
 5   BLOCK                 487 non-null    int32              
 6   XBLOCK                487 non-null    int64              
 7   YBLOCK                487 non-null    int64              
 8   WARD                  487 non-null    float64            
 9   ANC                   487 non-null    int32              
 10  DISTRICT              487 non-null    float64            
 11  PSA                   487 non-null    float64            
 12  NEIGHBORHOOD_CLUSTER  487 non-null    int32              
 13  BLOCK_GROUP           487 non-null    object             
 14  CENSUS_TRACT          487 non-null    float64            
 15  VOTING_PRECINCT       487 non-null    int32              
 16  LATITUDE              487 non-null    float64            
 17  LONGITUDE             487 non-null    float64            
 18  START_DATE            487 non-null    datetime64[ns, UTC]
 19  END_DATE              487 non-null    datetime64[ns, UTC]
 20  OBJECTID              487 non-null    int64              
dtypes: datetime64[ns, UTC](3), float64(6), int32(7), int64(4), object(1)
memory usage: 70.4+ KB
```

> ## Now here block group is only object left , should convert it to int.

In [43]:

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

Out[43]:

```
250
```

In [44]:

```
bg_encoder = pp.LabelEncoder()
df['BLOCK_GROUP']=bg_encoder.fit_transform(df['BLOCK_GROUP'])
```

In [45]:

```
df['BLOCK_GROUP'].head()
```

Out[45]:

```
0    242
1    241
3    139
4    207
5      2
Name: BLOCK_GROUP, dtype: int32
```

In [46]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 487 entries, 0 to 499
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype              
---  ------                --------------  -----              
 0   CCN                   487 non-null    int64              
 1   REPORT_DAT            487 non-null    datetime64[ns, UTC]
 2   SHIFT                 487 non-null    int32              
 3   METHOD                487 non-null    int32              
 4   OFFENSE               487 non-null    int32              
 5   BLOCK                 487 non-null    int32              
 6   XBLOCK                487 non-null    int64              
 7   YBLOCK                487 non-null    int64              
 8   WARD                  487 non-null    float64            
 9   ANC                   487 non-null    int32              
 10  DISTRICT              487 non-null    float64            
 11  PSA                   487 non-null    float64            
 12  NEIGHBORHOOD_CLUSTER  487 non-null    int32              
 13  BLOCK_GROUP           487 non-null    int32              
 14  CENSUS_TRACT          487 non-null    float64            
 15  VOTING_PRECINCT       487 non-null    int32              
 16  LATITUDE              487 non-null    float64            
 17  LONGITUDE             487 non-null    float64            
 18  START_DATE            487 non-null    datetime64[ns, UTC]
 19  END_DATE              487 non-null    datetime64[ns, UTC]
 20  OBJECTID              487 non-null    int64              
dtypes: datetime64[ns, UTC](3), float64(6), int32(8), int64(4)
memory usage: 68.5 KB
```

> ## Successfully all the features converted to int,float,datetime
>
> *Now should remove unwanted features like CCN, strat_date, end_date,OBJECTID.*
>
> *Because we dont want CCN,OBJECTID and also instead of two dates we have another feature report_date.*

In [47]:

```
df.drop(['CCN','START_DATE','END_DATE','OBJECTID'],axis=1,inplace=True)
```

> *Report date converted to month,day day of week.*

In [48]:

```
df['MONTH']=df['REPORT_DAT'].dt.month
df['DAY']=df['REPORT_DAT'].dt.day
df['Day_Week']=df['REPORT_DAT'].dt.dayofweek
df.drop('REPORT_DAT',axis=1,inplace=True)
```

Final DataSet

In [49]:

```
df.head()
```

Out[49]:

	SHIFT	METHOD	OFFENSE	BLOCK	XBLOCK	YBLOCK	WARD	ANC	DISTRICT	PSA	NEIGHBORI
0	2	2	5	423	400042	137118	6.0	27	1.0	104.0	
1	2	2	5	397	400387	137463	6.0	27	5.0	506.0	
3	2	2	2	218	402815	131990	8.0	36	7.0	702.0	
4	2	2	2	316	405237	138096	7.0	32	6.0	601.0	
5	1	2	7	102	394480	138000	2.0	8	2.0	206.0	

In [50]:

```
df.shape
```

Out[50]:

```
(487, 19)
```

In [51]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 487 entries, 0 to 499
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   SHIFT                 487 non-null    int32  
 1   METHOD                487 non-null    int32  
 2   OFFENSE               487 non-null    int32  
 3   BLOCK                 487 non-null    int32  
 4   XBLOCK                487 non-null    int64  
 5   YBLOCK                487 non-null    int64  
 6   WARD                  487 non-null    float64
 7   ANC                   487 non-null    int32  
 8   DISTRICT              487 non-null    float64
 9   PSA                   487 non-null    float64
 10  NEIGHBORHOOD_CLUSTER  487 non-null    int32  
 11  BLOCK_GROUP           487 non-null    int32  
 12  CENSUS_TRACT          487 non-null    float64
 13  VOTING_PRECINCT       487 non-null    int32  
 14  LATITUDE              487 non-null    float64
 15  LONGITUDE             487 non-null    float64
 16  MONTH                 487 non-null    int64  
 17  DAY                   487 non-null    int64  
 18  Day_Week              487 non-null    int64  
dtypes: float64(6), int32(8), int64(5)
memory usage: 60.9 KB
```

In [52]:

```
df.iloc[0]
```

Out[52]:

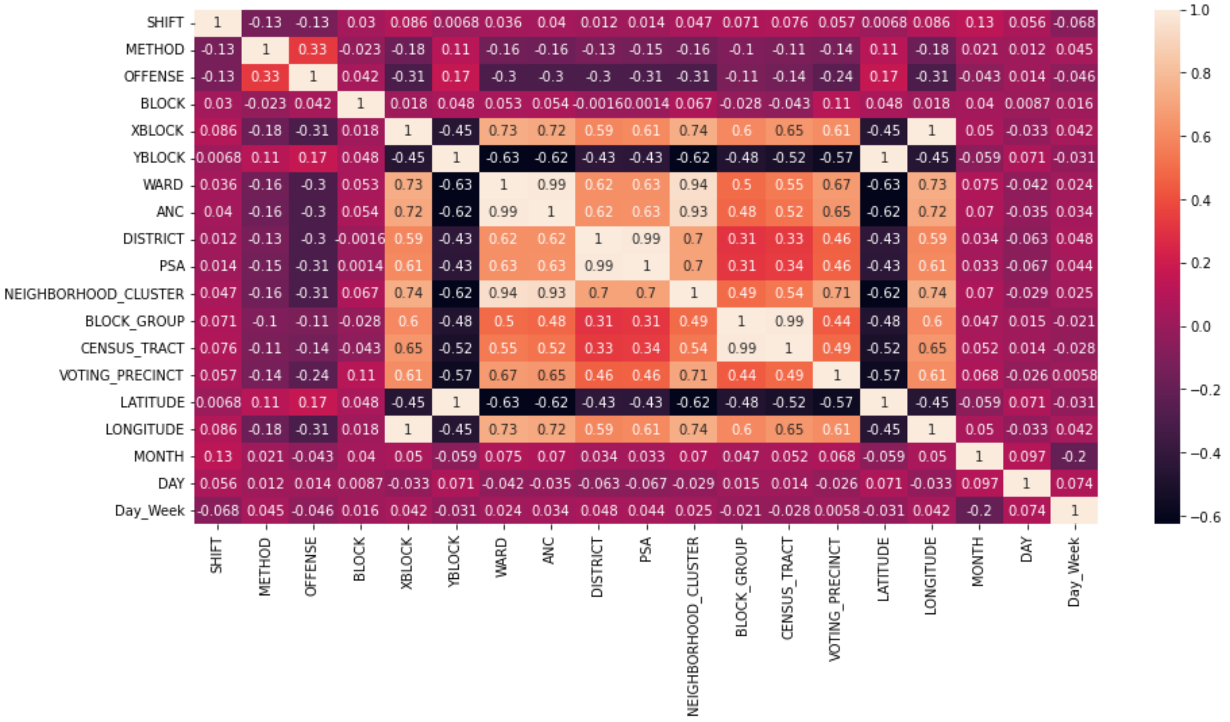
```
SHIFT                           2.000000
METHOD                          2.000000
OFFENSE                         5.000000
BLOCK                         423.000000
XBLOCK                     400042.000000
YBLOCK                     137118.000000
WARD                            6.000000
ANC                            27.000000
DISTRICT                        1.000000
PSA                           104.000000
NEIGHBORHOOD_CLUSTER           25.000000
BLOCK_GROUP                   242.000000
CENSUS_TRACT                10600.000000
VOTING_PRECINCT                83.000000
LATITUDE                       38.901916
LONGITUDE                     -76.999516
MONTH                           4.000000
DAY                            25.000000
Day_Week                        2.000000
Name: 0, dtype: float64
```

In [53]:

```
# correlation heat map
plt.figure(figsize=[15,7])
sb.heatmap(df.corr(), annot=True)
```

Out[53]:

```
<AxesSubplot:>
```



In [54]:

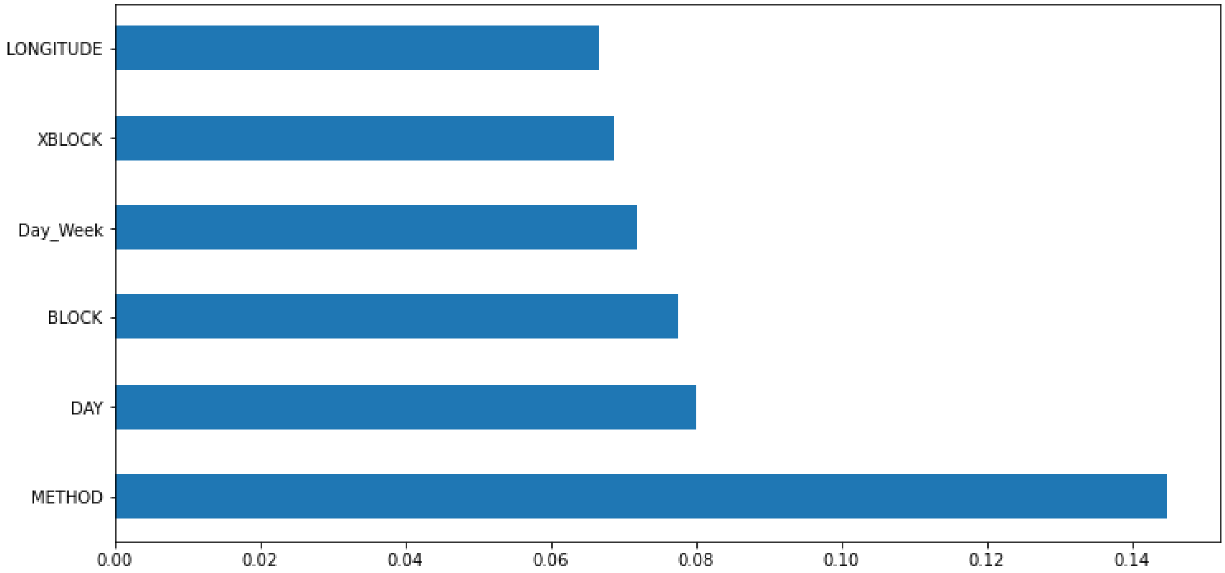
```
# Important feature using ExtraTreesRegressor
from sklearn.ensemble import ExtraTreesRegressor
X = df.drop('OFFENSE',axis=1)
y = df['OFFENSE']
model = ExtraTreesRegressor()
model.fit(X,y)
```

Out[54]:

```
ExtraTreesRegressor()
```

In [55]:

```
# plot graph of feature importances for 6 better visualization
plt.figure(figsize=[12,6])
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(6).plot(kind='barh')
plt.show()
```



In [56]:

```
print(feat_importances.sort_values(ascending=False))
```

```
METHOD                  0.144868
DAY                     0.079951
BLOCK                   0.077631
Day_Week                0.071847
XBLOCK                  0.068694
LONGITUDE               0.066604
ANC                     0.057553
PSA                     0.054855
LATITUDE                0.049692
YBLOCK                  0.048032
SHIFT                   0.045858
BLOCK_GROUP             0.043801
NEIGHBORHOOD_CLUSTER    0.042991
VOTING_PRECINCT         0.041342
DISTRICT                0.039411
CENSUS_TRACT            0.037104
WARD                    0.028830
MONTH                   0.000935
dtype: float64
```

In [57]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2, random_state=365)
```

In [58]:

```
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split,cross_val_score,KFold
#Importing Models
from sklearn.linear_model import LogisticRegression,LinearRegression,Lasso,Ridge,Bayes
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn import svm
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score,mean_squared_error
import seaborn as sns
```

In [59]:

```
models = {
    'Logistic' : LogisticRegression(),
    'random forest' : RandomForestRegressor(),
    'decision tree' : DecisionTreeRegressor(max_depth=5),
    'support vector': svm.SVC(),
    'knn' : KNeighborsRegressor(n_neighbors = 4)
}
```

In [60]:

```
for name, model in models.items():
    model.fit(X_train, y_train)
    print(f'{name} trained')
```

```
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic trained
random forest trained
decision tree trained
support vector trained
knn trained
```

In [61]:

```
results = {}
kf = KFold(n_splits= 10)

for name, model in models.items():
    result = np.mean(np.sqrt(-cross_val_score(model, X_train, y_train,scoring='neg_mea
    results[name] = result
```

```
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [62]:

```
for name, result in results.items():
    print(f"{name} : {round(result, 3)}")
```

```
Logistic : 2.91
random forest : 2.125
decision tree : 2.348
support vector : 2.761
knn : 2.322
```

In [63]:

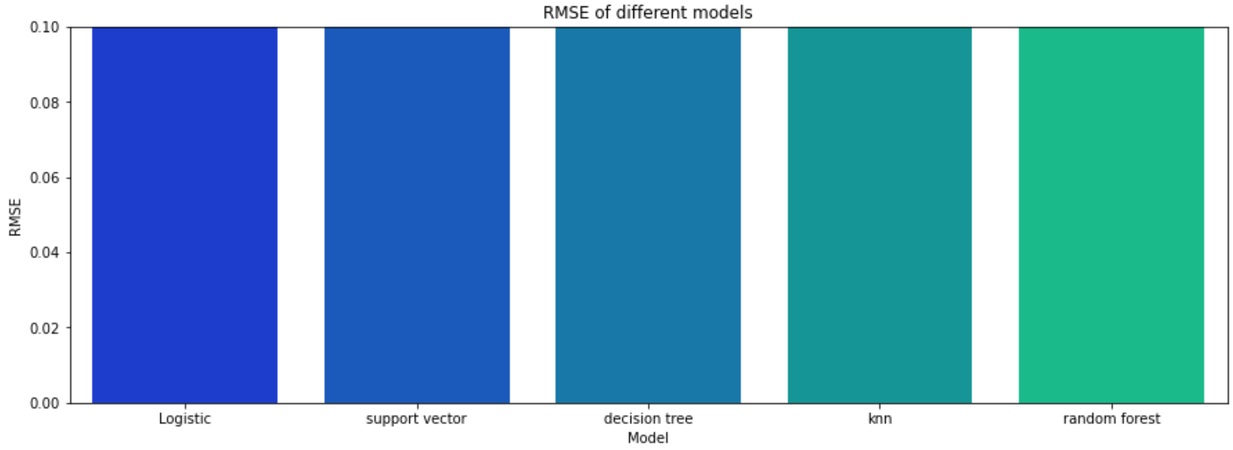
```
results_df = pd.DataFrame(results, index=range(0,1)).T.rename(columns={0: 'RMSE'}).sor
results_df.T
```

Out[63]:

	Logistic	support vector	decision tree	knn	random forest
RMSE	2.909916	2.760594	2.347988	2.322145	2.125403

In [64]:

```
plt.figure(figsize = (15, 5))
sns.barplot(x= results_df.index, y = results_df['RMSE'], palette = 'winter')
plt.ylim(0,0.1)
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('RMSE of different models');
```



In [65]:

```
def prediction(model,X_train,y_train,X_test,y_test):
    model.fit(X_train,y_train)
    pred_data=np.exp(model.predict(X_test))
    return r2_score(np.exp(y_test),pred_data)
```

In [66]:

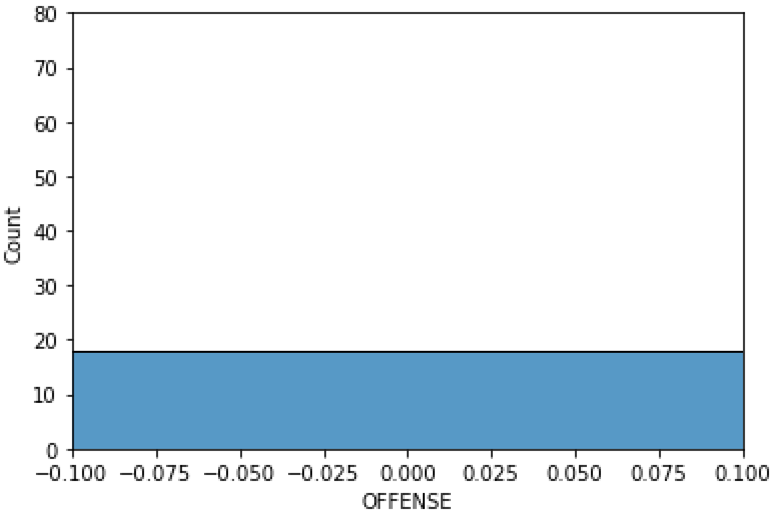
```
for name,model in models.items():
    score=prediction(model,X_train,y_train,X_test,y_test)
    print(f'{name} r2_score is {score}')
```

```
C:\Users\91938\Documents\sample_project_1\env\lib\site-packages\sklearn\linear_model
\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic r2_score is -0.37982653533356814
random forest r2_score is -0.28386472502650695
decision tree r2_score is -0.22290812828469675
support vector r2_score is -0.0427985797836654l4
knn r2_score is -0.5942471882184519
```

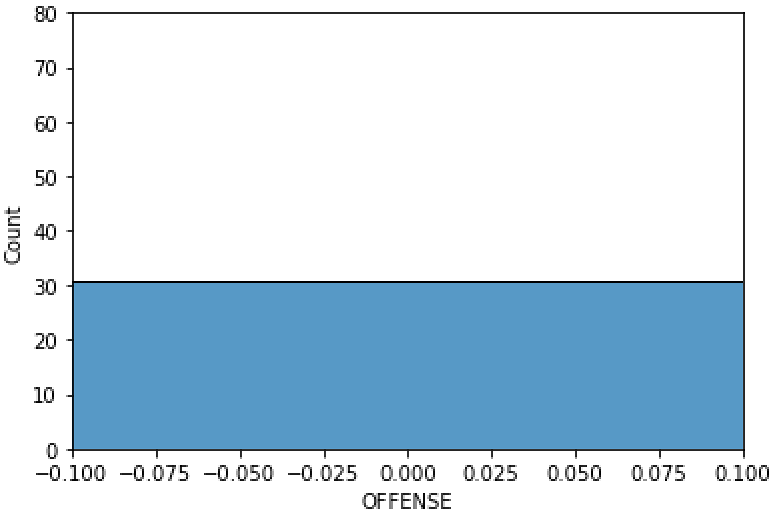
In [67]:

```
model=LinearRegression()
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-0.1,0.1)
plt.ylim(0,80)
plt.show()
```



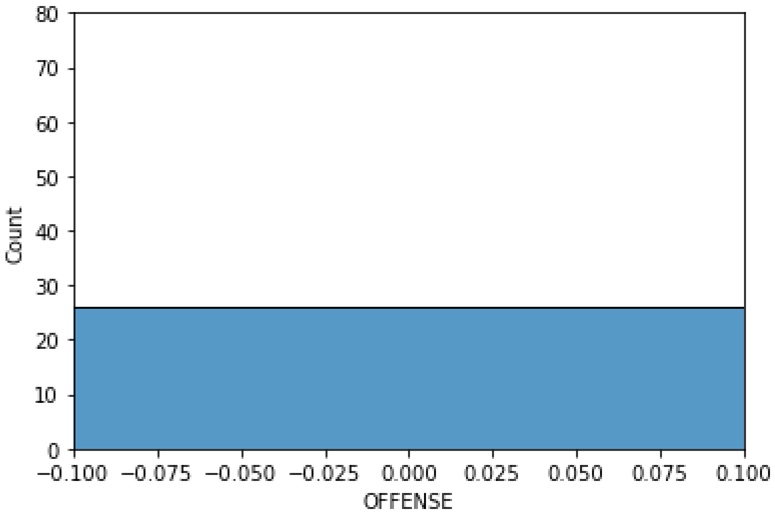
In [68]:

```
model=RandomForestRegressor()
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-0.1,0.1)
plt.ylim(0,80)
plt.show()
```



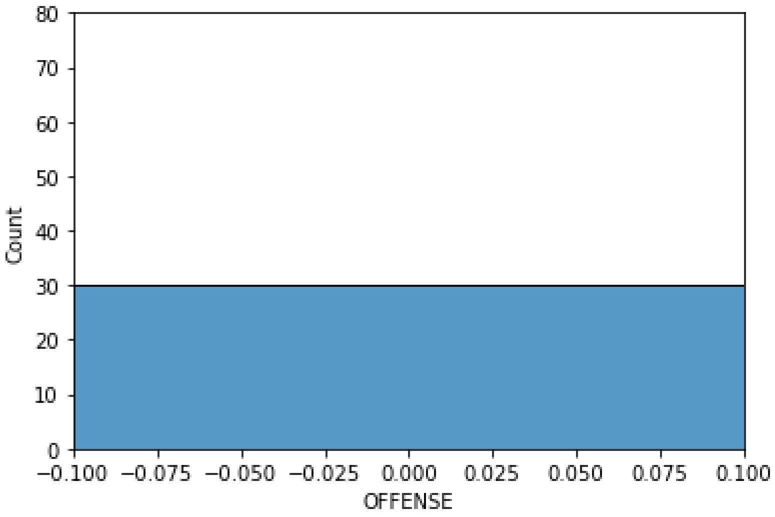
In [69]:

```
model=DecisionTreeRegressor(max_depth=5)
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-0.1,0.1)
plt.ylim(0,80)
plt.show()
```



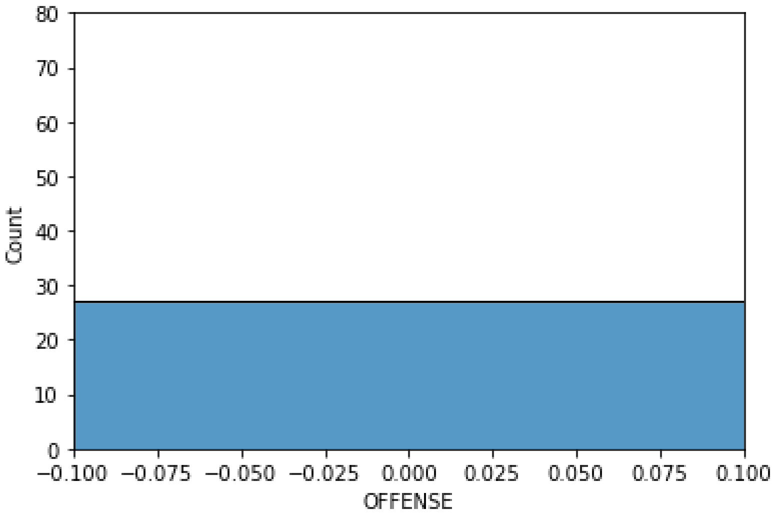
In [70]:

```
model=SVR()
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-0.1,0.1)
plt.ylim(0,80)
plt.show()
```



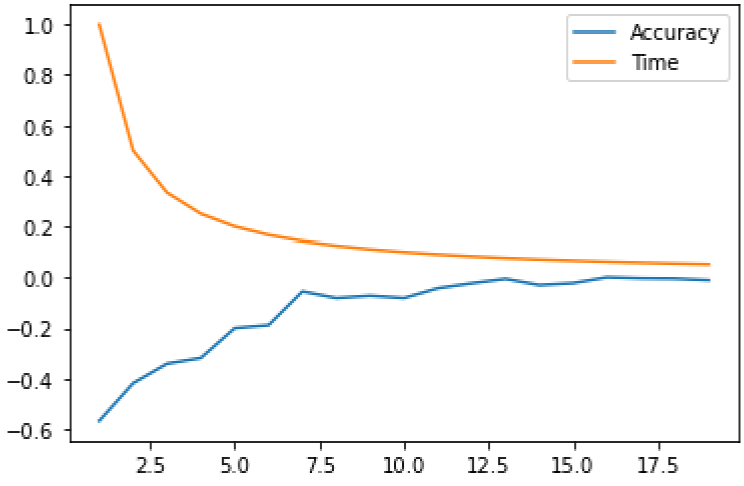
In [71]:

```
model=KNeighborsRegressor(n_neighbors = 4)
model.fit(X_train,y_train)
pred_y=np.exp(model.predict(X_test))
sns.histplot(np.exp(y_test)-pred_y)
plt.xlim(-0.1,0.1)
plt.ylim(0,80)
plt.show()
```



In [72]:

```
# •Similarly plot execution time for different values of k.
k = []
acc = []
time = []
for i in range(1, 20):
    kn = KNeighborsRegressor(n_neighbors=i)
    kn.fit(X_train, y_train)
    kn_pred = kn.predict(X_test)
    kn_acc = r2_score(y_test, kn_pred)
    k.append(i)
    acc.append(kn_acc)
    time.append(1/i)
# Plotting accuracy and execution time for different values of k
plt.plot(k, acc, label='Accuracy')
plt.plot(k, time, label='Time')
plt.legend()
plt.show()
```



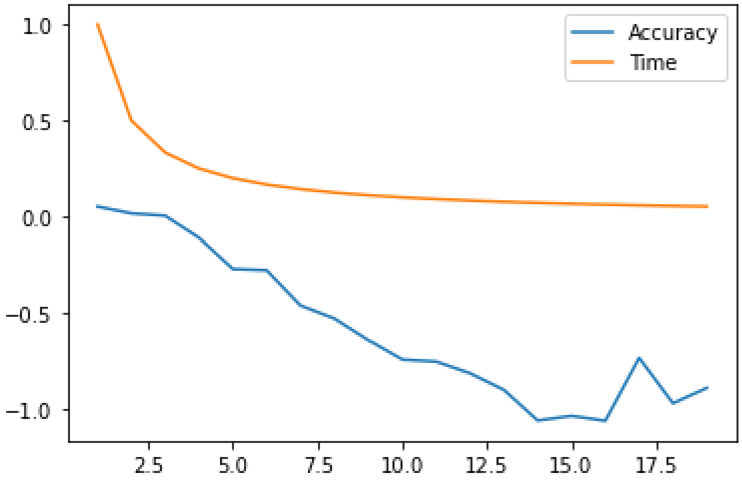
In [73]:

```
from sklearn.tree import DecisionTreeRegressor
for i in range(1, 10):
    dt = DecisionTreeRegressor(max_depth=i)
    dt.fit(X_train, y_train)
    dt_pred = dt.predict(X_test)
    dt_acc = r2_score(y_test, dt_pred)
    print(dt_acc)
```

```
0.05230631512283146
0.01714083312990855
0.004949400115422331
-0.1092329926322293
-0.2709842509575755
-0.2774021692346915
-0.4608248117722109
-0.5487967648453589
-0.6432599642018022
```

In [74]:

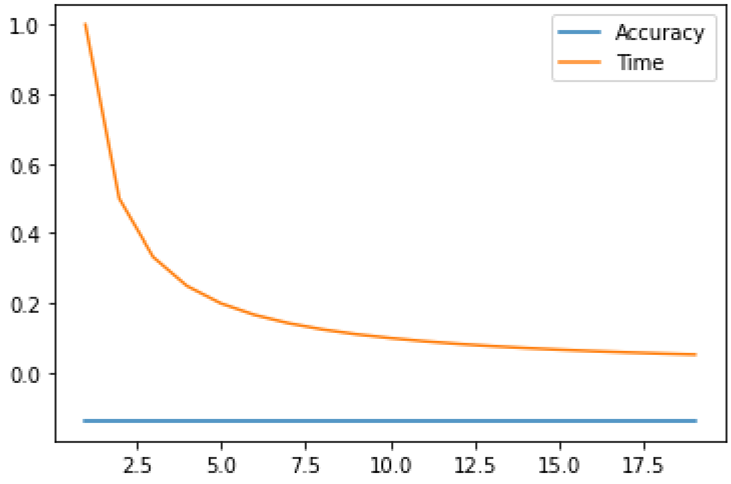
```python
# Make predictions on dataset. Plot accuracy and time for varying parameters
#decision tree model
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
# Checking accuracy of each model for various values of max_depth
acc = []
time = []
k = []
for i in range(1,20):
    dt = DecisionTreeRegressor(max_depth=i)
    dt.fit(X_train, y_train)
    dt_pred = dt.predict(X_test)
    dt_acc = r2_score(y_test, dt_pred)
    acc.append(dt_acc)
    k.append(i)
    time.append(1/i)
# Plotting accuracy of each model for various values of max_depth
plt.plot(k, acc, label='Accuracy')
plt.plot(k, time, label='Time')
plt.legend()
plt.show()
```



In [75]:

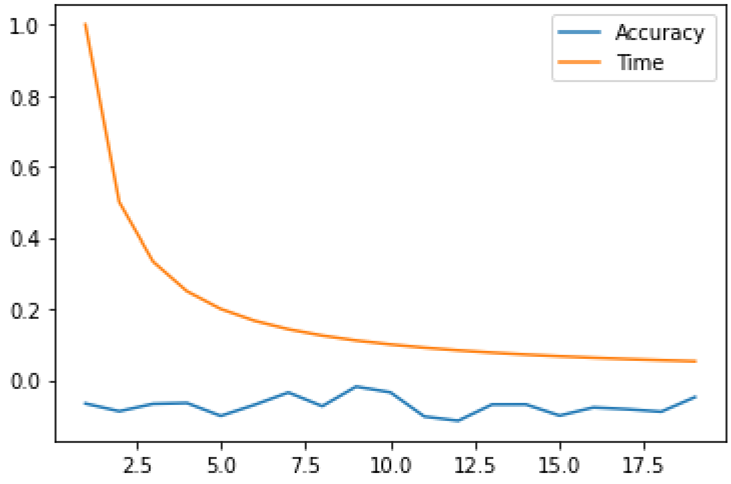
```python
k = []
acc = []
time = []
for i in range(1, 20):
    kn = SVR()
    kn.fit(X_train, y_train)
    kn_pred = kn.predict(X_test)
    kn_acc = r2_score(y_test, kn_pred)
    k.append(i)
    acc.append(kn_acc)
    time.append(1/i)
# Plotting accuracy and execution time for different values of k
```

```python
plt.plot(k, acc, label='Accuracy')
plt.plot(k, time, label='Time')
plt.legend()
plt.show()
```



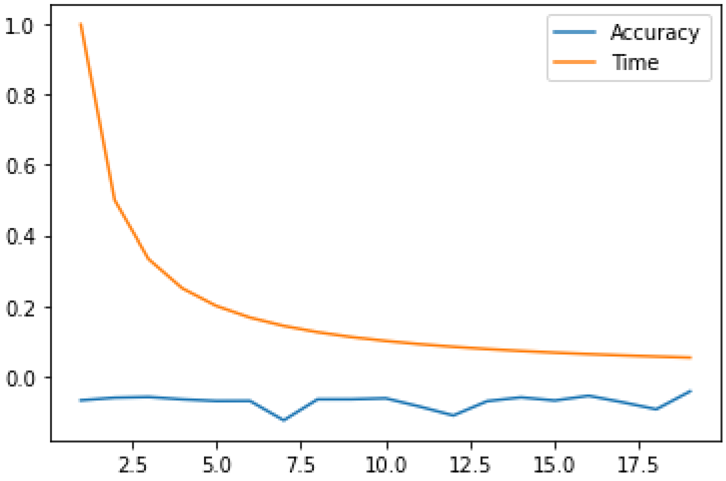
In [76]:

```python
k = []
acc = []
time = []
for i in range(1, 20):
    kn = RandomForestRegressor()
    kn.fit(X_train, y_train)
    kn_pred = kn.predict(X_test)
    kn_acc = r2_score(y_test, kn_pred)
    k.append(i)
    acc.append(kn_acc)
    time.append(1/i)
# Plotting accuracy and execution time for different values of k
plt.plot(k, acc, label='Accuracy')
plt.plot(k, time, label='Time')
plt.legend()
plt.show()
```



In [77]:

```python
k = []
acc = []
time = []
for i in range(1, 20):
    kn = RandomForestRegressor()
    kn.fit(X_train, y_train)
    kn_pred = kn.predict(X_test)
    kn_acc = r2_score(y_test, kn_pred)
    k.append(i)
    acc.append(kn_acc)
    time.append(1/i)
# Plotting accuracy and execution time for different values of k
plt.plot(k, acc, label='Accuracy')
plt.plot(k, time, label='Time')
plt.legend()
plt.show()
```



In [81]:

```
from sklearn.metrics import mean_absolute_error
xgb_model = RandomForestRegressor()
xgb_model.fit(X_train, y_train)
final_preds = xgb_model.predict(X_test)

model_results = pd.DataFrame([y_test.values, final_preds])
model_results = model_results.transpose()
model_results = model_results.rename(columns={0:'Actual',1:'Predicted'})

print(model_results.describe(),'\n')
```

```
          Actual  Predicted
count  98.000000  98.000000
mean    5.132653   4.844592
std     2.064115   1.383657
min     0.000000   0.610000
25%     4.000000   4.172500
50%     6.000000   4.960000
75%     7.000000   5.850000
max     7.000000   6.890000
```

In [82]:

```
!pip install pandoc
```

```
Requirement already satisfied: pandoc in c:\users\91938\documents\sample_project_1\en
v\lib\site-packages (2.2)
Requirement already satisfied: plumbum in c:\users\91938\documents\sample_project_1\e
nv\lib\site-packages (from pandoc) (1.8.0)
Requirement already satisfied: ply in c:\users\91938\documents\sample_project_1\env\l
ib\site-packages (from pandoc) (3.11)
Requirement already satisfied: pywin32 in c:\users\91938\documents\sample_project_1\e
nv\lib\site-packages (from plumbum->pandoc) (302)
```

```
In [ ]:
```