About Dataset:

Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. The attributes are defined as follows (taken from the UCI Machine Learning Repository1): CRIM: per capita crime rate by town

X: Predictors

- CRIM: per capita crime rate by town
- ZN: proportion of residential land zoned for lots over 25,000sq. Ft.
- INDUS: proportion of non-retail business acres per town
- CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX: nitric oxides concentration (parts per 10 million)
- RM: average number of rooms per dwelling
- AGE: proportion of owner-occupied units built prior to 1940
- DIS: weighted distances to five Boston employment centre's
- RAD: index of accessibility to radial highways
- TAX: full-value property-tax rate per 10k
- PTRATIO: pupil-teacher ratio by town 12.
- B: 1000(Bk−0.63)2 where Bk is the proportion of blacks by town 13.
- LSTAT: Percentage lower status of the population

Y: Outcome

- MEDV: Median value of owner-occupied homes in $1000s

Objective:

Our Objective is to most accurately predict the prices of houses in Boston using Linear Regression models.

Data Cleaning:

1. Checking for missing values

```
# check for missing values in all the columns
print("[INFO] df isnull():\n {}".format(df.isnull().sum()))
```

```
[INFO] df isnull():
 CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

   Hence no missing values found in our dataset

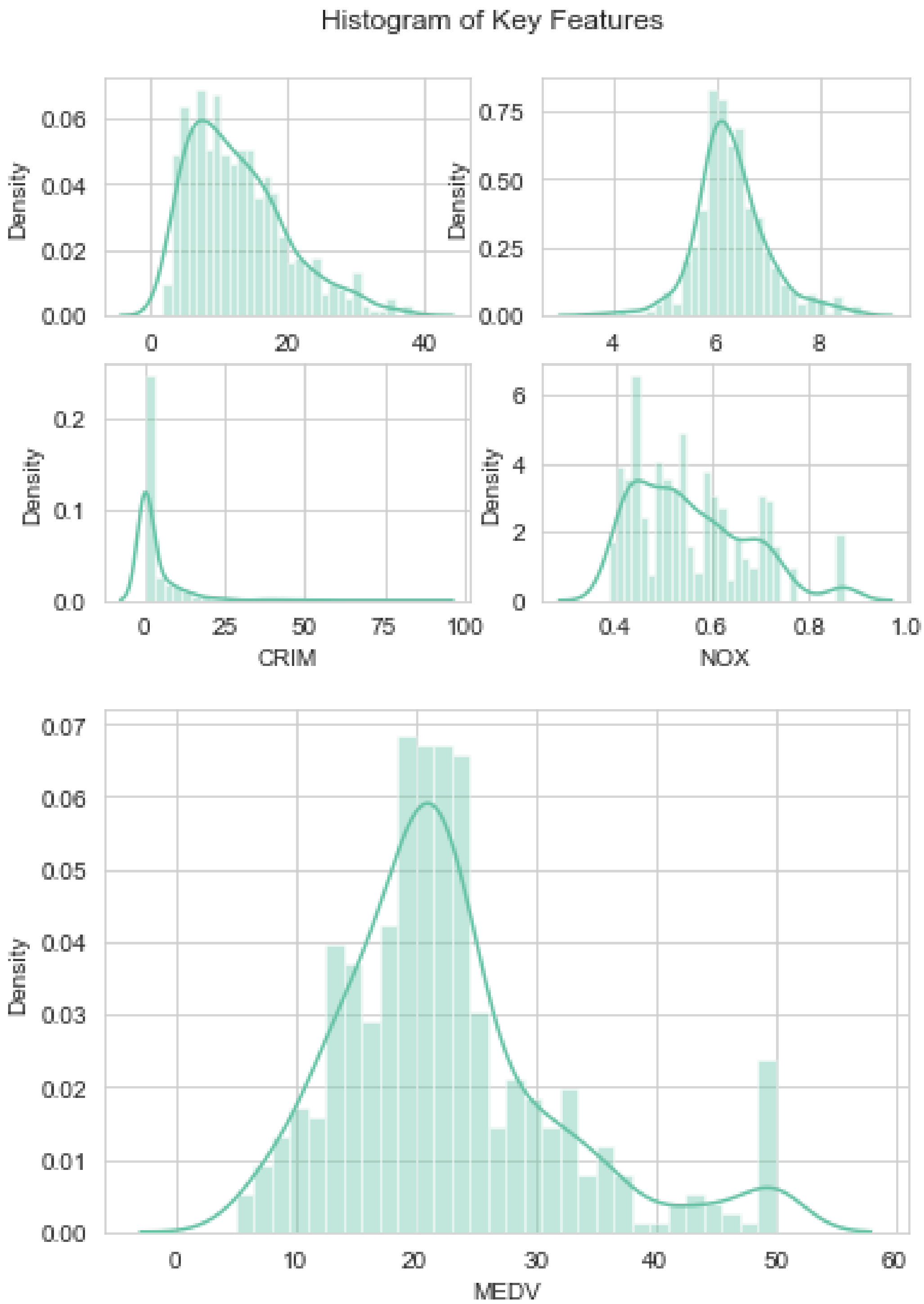
2. Removing Inconsistent text and types

   Our dataset does not contain any inconsistent text and types.

3. Removing Duplicate or Unnecessary data

   Our dataset does not contain any duplicate or unnecessary data.

4. Outliers



Hence none of the key features contain any outliers.

EDA (Expletory Data Analysis)

Data Types of the Features:

```
df.dtypes
```

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

Column Description:

Describe important statistics of columns

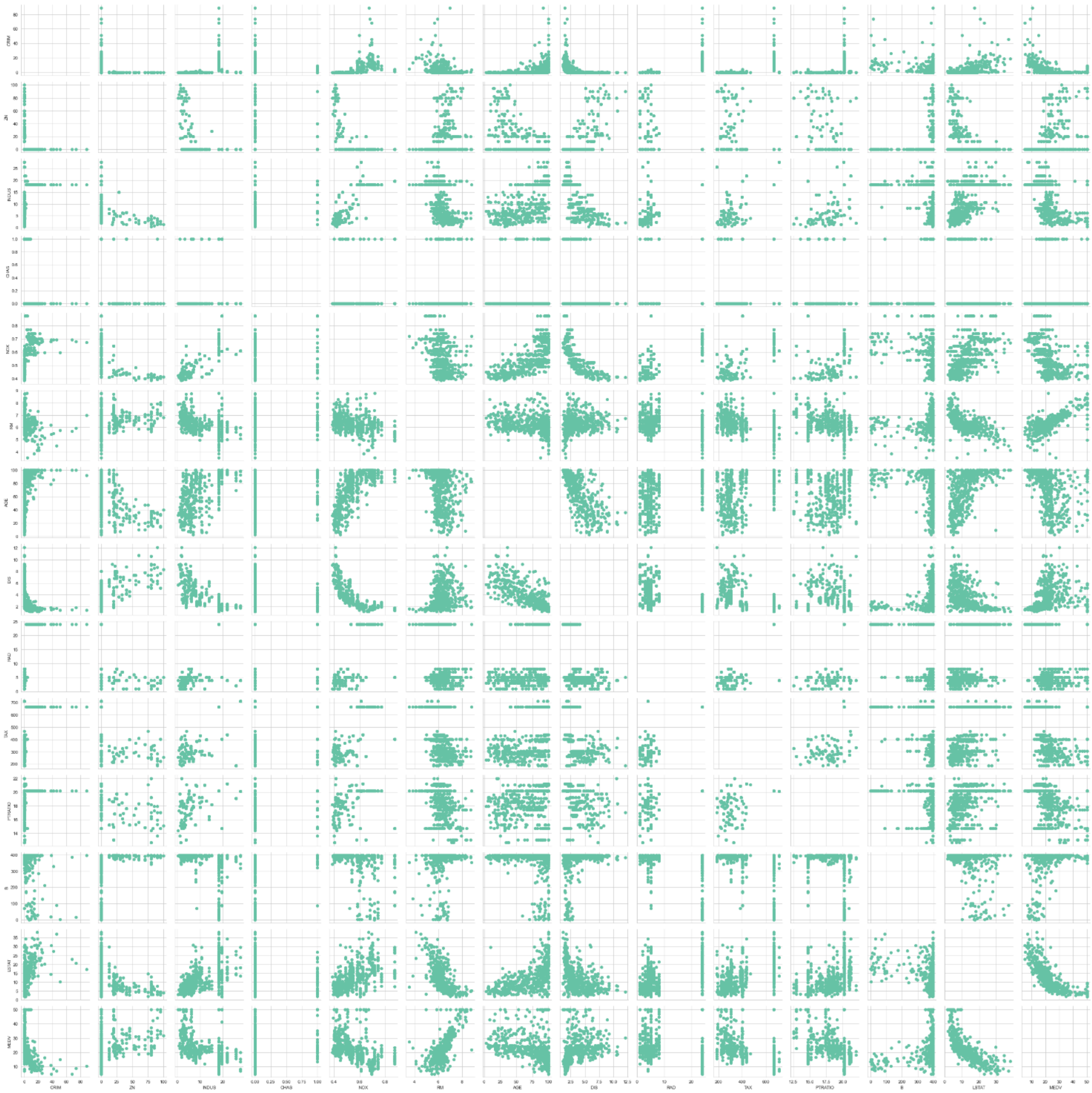
```
df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.6
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.1
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.7
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.9
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.9
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.9

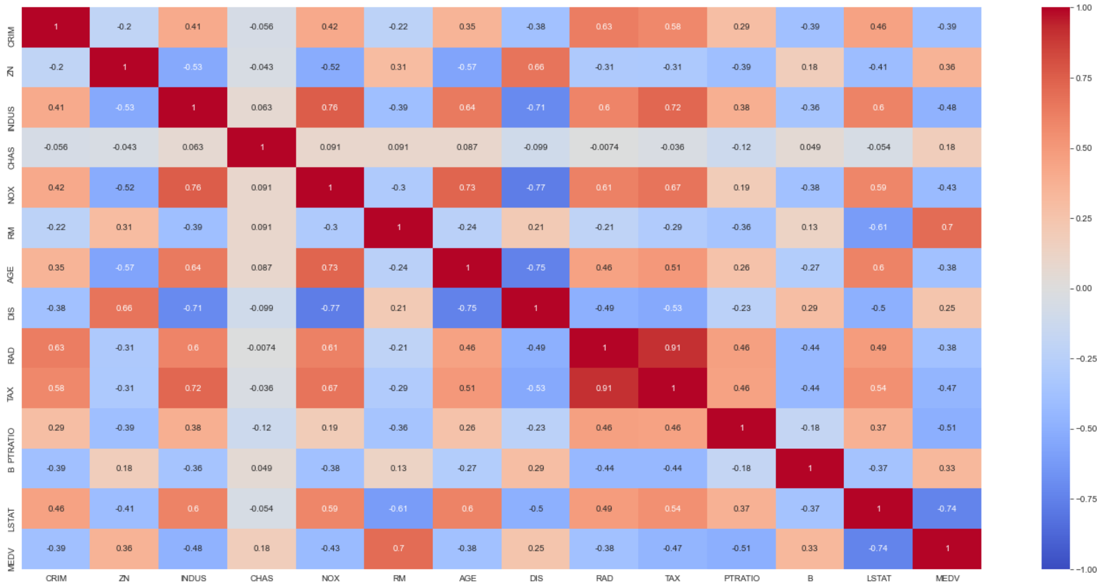
For more please refer the Jupyter Notebook provided

Scatter plot

According to the plots on the last row, we can observe moderate to strong relationship between each predictor and median house price, suggesting these predictors could explain the house prices to some extent.



Corelation between various features…



Now let’s perform Linear Regression on our Dataset…

We will make following models

- Simple Linear Regression
- Polynomial Linear Regression
- Linear Regression with Ridge Regularization (L-2 Regularization)

Simple Linear Regression

- Y = aX + b
- Y = target, X = features
- a,b= parameters of model
- best line of fit: minimize the error function (SSE) --> best a,b

```
#split the data into predictors X and Y
X=df.iloc[:,:12]
y=df.iloc[:,13]
```

```
#Splitting to training and testing data
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=1)
```

```
#Applying Linear Regression
lr_all=LinearRegression()
lr_all.fit(X_train, y_train)
y_pred1=lr_all.predict(X_test)
# coefficient of intercept
lr_all.intercept_
```

```
30.8578330890054
```

```
#accuracy score
print('R^2:',metrics.r2_score(y_test, y_pred1))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_pred1))*(len(y_test)-1)/(len(y_test)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_pred1))
print('MSE:',metrics.mean_squared_error(y_test, y_pred1))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred1)))
```

```
R^2: 0.7585173392839138
Adjusted R^2: 0.7376699153372014
MAE: 3.452004055697846
MSE: 22.132969411911
RMSE: 4.7045689932140435
```

Using above code, we make a simple Linear Regression model with an accuracy of....


```
#accuracy score
print('R^2:',metrics.r2_score(y_test, y_pred1))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_pred1))*(len(y_test)-1)/(len(y_test)-X_train.sh
print('MAE:',metrics.mean_absolute_error(y_test, y_pred1))
print('MSE:',metrics.mean_squared_error(y_test, y_pred1))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_pred1)))
```

```
R^2: 0.7585173392839138
Adjusted R^2: 0.7376699153372014
MAE: 3.452004055697846
MSE: 22.132969411911
RMSE: 4.7045689932140435
```

Model Evaluation :

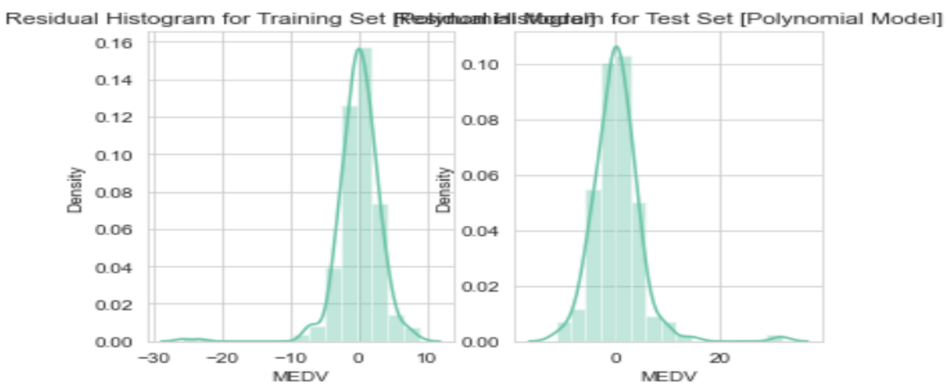
- R^2: It is a measure of the linear relationship between X and Y. It is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.
- Adjusted R^2 : The adjusted R-squared compares the explanatory power of regression models that contain different numbers of predictors.
- MAE : It is the mean of the absolute value of the errors. It measures the difference between two continuous variables, here actual and predicted values of y.
- MSE: The mean square error (MSE) is just like the MAE, but squares the difference before summing them all instead of using the absolute value.
- RMSE: The mean square error (MSE) is just like the MAE, but squares the difference before summing them all instead of using the absolute value.

Polynomial Linear Regression

We can conclude that the straight regression line is unable to capture the patterns in the data. This is an example of *underfitting*. To overcome underfitting, we need to increase the complexity of the model. This could be done by converting the original features into their higher order polynomial terms by using the Polynomial Features class provided by scikit-learn. Next, we train the model using Polynomial Linear Regression.

```
"Creates a polynomial regression model for the given degree"
poly_features = PolynomialFeatures(degree=2)

# transform the features to higher degree features.
X_train_poly = poly_features.fit_transform(X_train)

# fit the transformed features to Linear Regression
poly_model = LinearRegression()

poly_model.fit(X_train_poly, y_train)

# predicting on training data-set
y_train_predicted = poly_model.predict(X_train_poly)

# predicting on test data-set
y_test_predicted = poly_model.predict(poly_features.fit_transform(X_test))
```

```
y_train_residual = y_train_predicted - y_train
y_test_residual = y_test_predicted - y_test

plt.subplot(1, 2, 1)
sns.distplot(y_train_residual, bins=15)
plt.title('Residual Histogram for Training Set [Polynomial Model]')

plt.subplot(1, 2, 2)
sns.distplot(y_test_residual, bins=15)
plt.title('Residual Histogram for Test Set [Polynomial Model]')

plt.show()
```



Using above code, we make a Polynomial Linear Regression model with an accuracy of….

```
print('R^2:',metrics.r2_score(y_test, y_test_predicted))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_predicted))*(len(y_test)-1)/(len(y_test)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_predicted))
print('MSE:',metrics.mean_squared_error(y_test, y_test_predicted))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_predicted)))
```

```
R^2: 0.7685114086533842
Adjusted R^2: 0.7485267820623094
MAE: 3.065270977833707
MSE: 21.216968109792376
RMSE: 4.60618802371249
```

So Here we can see that using polynomial regression our prediction become more accurate as value of R^2 increases from 0.75 to 0.76.

Linear Regression with Regularization

- Default Performance Metrics: accuracy=correct prediction/ total # of prediction
- The loss function: OLS: minimize sum of squares of residuals
- the smaller the loss function, the better the model
- Regularization: Penalizing large coefficients

Ridge Regression

- one of the simple techniques to reduce model complexity and prevent over-fitting which may result from linear regression
- The loss function is altered by adding a penalty equivalent to square of the magnitude of the coefficients
- One parameter: Alpha (also called 'lambda')
- higher the alpha value --> more restriction on the coeffs
- lower alpha --> more generalization

```
#Ridge Regression
ridge=Ridge(alpha=100)
ridge.fit(X_train, y_train)
y_pred2=ridge.predict(X_test)
ridge.score(X_test, y_test)
#Low alpha
rr1=Ridge(alpha=0.01)
rr1.fit(X_train,y_train)
#High alpha
rr2=Ridge(alpha=100)
rr2.fit(X_train,y_train)
#Just Right
rr3=Ridge(alpha=1)
rr3.fit(X_train,y_train)
#Ridge regression test score with low alpha(0.1):
print('Linear regression test score:',lr_all.score(X_test,y_test))
print('Ridge regression test score with low alpha(0.1):',rr1.score(X_test,y_test))
print('Ridge regression test score with high alpha(100):',rr2.score(X_test,y_test))
print('Ridge regression test score with low alpha(1):',rr3.score(X_test,y_test))
```

```
Linear regression test score: 0.7585173392839138
Ridge regression test score with low alpha(0.1): 0.7587193637552938
Ridge regression test score with high alpha(100): 0.6757283037427892
Ridge regression test score with low alpha(1): 0.7645516158373611
```

In terms of test score: Ridge regression with high alpha has lowest test score, followed by vale of alpha very low and we can see that at alpha = 1 highest test score is achieved. That is just right value of alpha.

Note: The LASSO Regression is also performed in the Jupyter Notebook attached.

Recommend Model

For the given dataset on basis of my observations I will recommend polynomial regression as it is most accurate model .

Key Findings

We find out that for our dataset polynomial Regression and Linear Regression with Ridge Regularization both shows promising results as they bot are most accurate in the above models

Linear Regression with LASSO regularization shows most inaccurate model with R^2 value of 0.68

Next Steps…

So we can use Elastic Net Regularization on the model to get more accurate predictions also we can perform some more feature engineering for better understanding of underlying features affecting the price which in turn will provide us a more accurate models