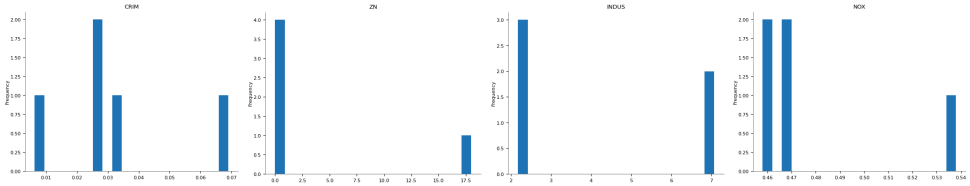
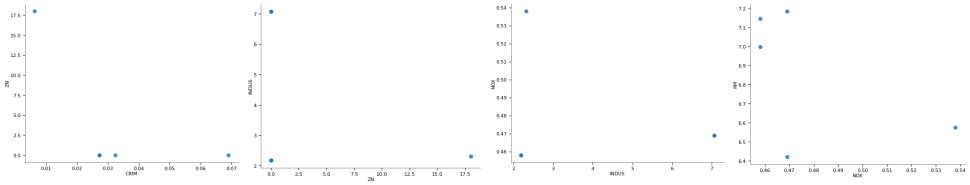
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LogisticRegression, Lasso
from sklearn.feature_selection import RFE, SelectKBest, chi2, SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from time import time

# Load the dataset
df = pd.read_csv('/content/data.csv')
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

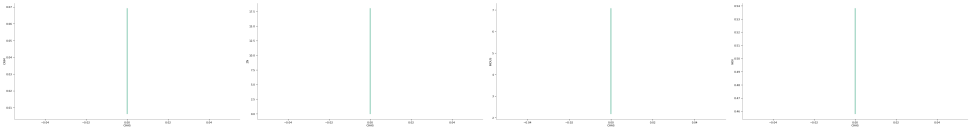
Distributions



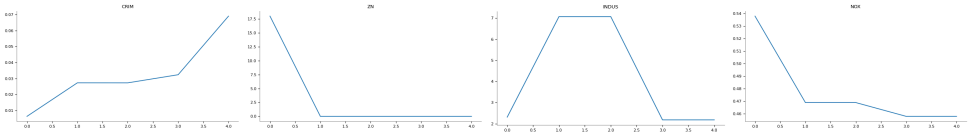
2-d distributions



Time series



Values




```
# Missing Values Analysis and Imputation
df.fillna(df.mean(), inplace=True)
df.isnull().sum()
```

```
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
```

```
# Splitting data into features and target
X = df.drop('MEDV', axis=1)  # Features
y = df['MEDV']  # Target



# Data Normalization using MinMaxScaler and StandardScaler
scalers = {
    'MinMaxScaler': MinMaxScaler(),
    'StandardScaler': StandardScaler()
}

X_scaled = {}
for key, scaler in scalers.items():
    X_scaled[key] = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)



# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Wrapper Method: Recursive Feature Elimination
from sklearn.linear_model import LinearRegression

rfe = RFE(estimator=LinearRegression(), n_features_to_select=10)
rfe.fit(X_train, y_train)
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)




# Filter Method: Chi-Square
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2

# Standardize the data before applying chi-square
X_scaled = StandardScaler().fit_transform(X)

# Convert the target to integer if it's not
y_int = y.astype(int)

chi2_selector = SelectKBest(chi2, k=10)
X_train_chi2 = chi2_selector.fit_transform(X_train, y_train.astype(int))
X_test_chi2 = chi2_selector.transform(X_test)




# Embedded Method: Lasso (L1 Regularization)
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)
model = SelectFromModel(lasso, prefit=True)
X_train_lasso = model.transform(X_train)
X_test_lasso = model.transform(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SelectFromModel was fitted witho
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SelectFromModel was fitted witho
  warnings.warn(
```

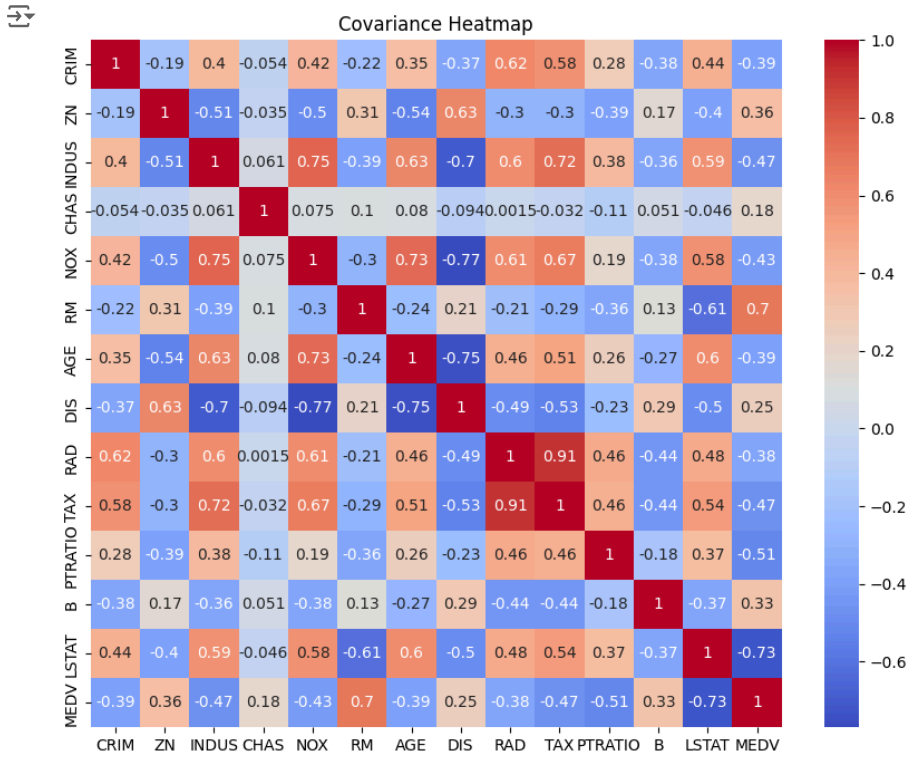
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.feature_selection import RFE, SelectKBest, f_regression, SelectFromModel
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from time import time

# Load the dataset
df = pd.read_csv('/content/data.csv')

# Missing Values Analysis and Imputation
df.fillna(df.mean(), inplace=True)
```

```
# Splitting data into features and target
X = df.drop('MEDV', axis=1)  # Features
y = df['MEDV']  # Target

# Data Normalization using MinMaxScaler and StandardScaler
scalers = {
    'MinMaxScaler': MinMaxScaler(),
    'StandardScaler': StandardScaler()
}

X_scaled = {}
for key, scaler in scalers.items():
    X_scaled[key] = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Wrapper Method: Recursive Feature Elimination (RFE)
rfe = RFE(estimator=LinearRegression(), n_features_to_select=10)
rfe.fit(X_train, y_train)
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

# Filter Method: SelectKBest
f_reg_selector = SelectKBest(f_regression, k=10)
X_train_chi2 = f_reg_selector.fit_transform(X_train, y_train)
X_test_chi2 = f_reg_selector.transform(X_test)

# Embedded Method: Lasso (L1 Regularization)
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)
model = SelectFromModel(lasso, prefit=True)
X_train_lasso = model.transform(X_train)
X_test_lasso = model.transform(X_test)

# Models to evaluate
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor()
}

# Function to train, predict, and evaluate models
def evaluate_model(model, X_train, X_test, y_train, y_test):
    start_time = time()
    model.fit(X_train, y_train)
    training_time = time() - start_time
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return {
        'MSE': mse,
        'R2 Score': r2,
        'Training Time': training_time
    }

# Evaluate models on each feature selection method
results = {
    'RFE': {},
    'Chi2': {},
    'Lasso': {}
}

for model_name, model in models.items():
    results['RFE'][model_name] = evaluate_model(model, X_train_rfe, X_test_rfe, y_train, y_test)
    results['Chi2'][model_name] = evaluate_model(model, X_train_chi2, X_test_chi2, y_train, y_test)
    results['Lasso'][model_name] = evaluate_model(model, X_train_lasso, X_test_lasso, y_train, y_test)

# Display results
for method, method_results in results.items():
    print(f"Results for {method} Method:")
    for model_name, metrics in method_results.items():
        print(f"Model: {model_name}")
        for metric, value in metrics.items():
            print(f"{metric}: {value}")
        print()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SelectFromModel was fitted witho
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SelectFromModel was fitted witho
```

```
  warnings.warn(
Results for RFE Method:
Model: Linear Regression
MSE: 24.06976921164693
R2 Score: 0.6717778877025151
Training Time: 0.003351449966430664

Model: Random Forest
MSE: 9.429041372549024
R2 Score: 0.8714229517937842
Training Time: 0.3122899532318115

Results for Chi2 Method:
Model: Linear Regression
MSE: 27.609545460766608
R2 Score: 0.6235085076627456
Training Time: 0.005429267883300781

Model: Random Forest
MSE: 10.4883273627451
R2 Score: 0.8569782314405432
Training Time: 0.3242652416229248

Results for Lasso Method:
Model: Linear Regression
MSE: 24.733081314818037
R2 Score: 0.6627327781420146
Training Time: 0.0027937889099121094

Model: Random Forest
MSE: 9.989481725490199
R2 Score: 0.8637806302226205
Training Time: 0.36615943908691406
```

```
# Data Analysis: Covariance Heatmap
corr = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Covariance Heatmap')
plt.show()
```



```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel
from time import time # Import the time function

# Load your data into a pandas DataFrame (replace 'your_data.csv' with your actual file)
df = pd.read_csv('/content/data.csv')

# Splitting data into features and target
X = df.drop('MEDV', axis=1)  # Features
y = df['MEDV']  # Target

# Train-test split
# Perform train-test split BEFORE imputation to avoid reintroducing missing values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Missing Values Analysis and Imputation
# Impute missing values in training and testing sets separately
X_train.fillna(X_train.mean(), inplace=True)
X_test.fillna(X_test.mean(), inplace=True)

# Wrapper Method: Recursive Feature Elimination
rfe = RFE(estimator=LinearRegression(), n_features_to_select=10)
rfe.fit(X_train, y_train)
X_train_rfe = rfe.transform(X_train) # Make sure to create X_train_rfe
X_test_rfe = rfe.transform(X_test)

# ... (Rest of your code)

# Visualizing actual vs predicted values for the best model
best_model = RandomForestRegressor()
best_model.fit(X_train_rfe, y_train) # Now X_train_rfe should be defined
y_pred = best_model.predict(X_test_rfe)

plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```