experiment-1-dl

April 23, 2024

[ ]:

[3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

[5]:
```
df = pd.read_csv("Boston Dataset.csv")
df.drop(columns=['Unnamed: 0'], axis=0, inplace=True)
df.head()
```

[5]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	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	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

[6]:
```
# statistical info
df.describe()
```

[6]:

	crim	zn	indus	chas	nox	rm
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500

```
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              age         dis         rad         tax     ptratio       black  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            lstat        medv
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[7]:
```
# datatype info
df.info()
```

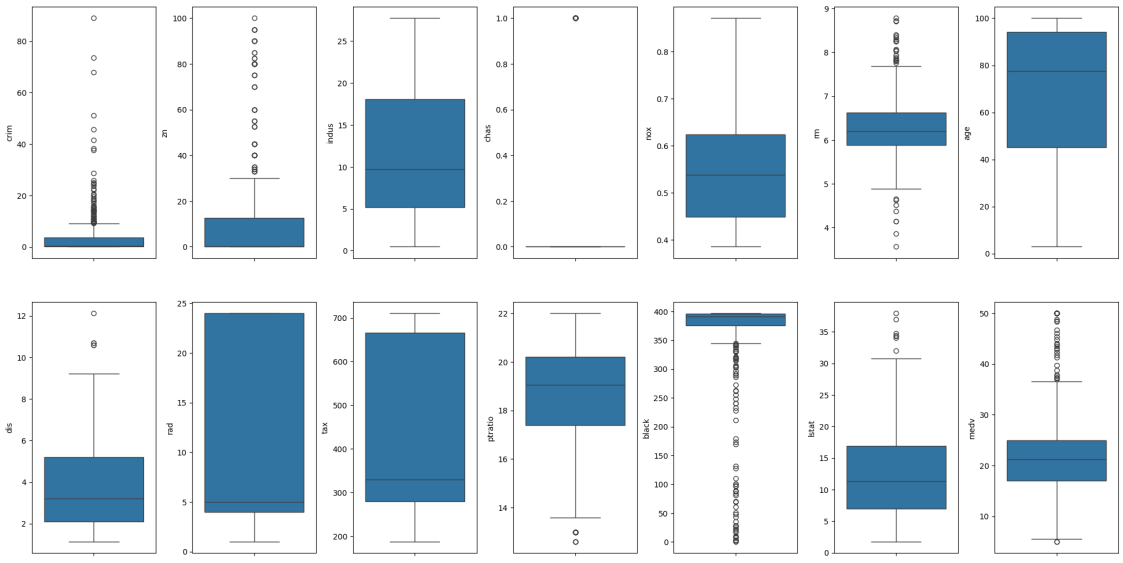
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
```

```
memory usage: 55.5 KB
```

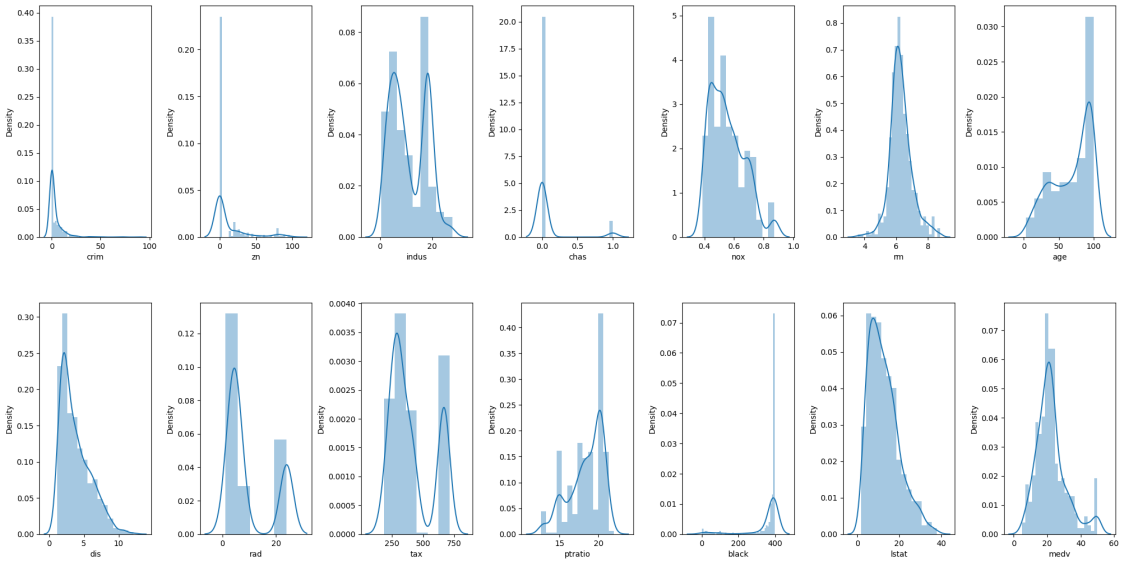
[8]:
```
# check for null values
df.isnull().sum()
```

[8]:
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

[9]:
```
# create box plots
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    sns.boxplot(y=col, data=df, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

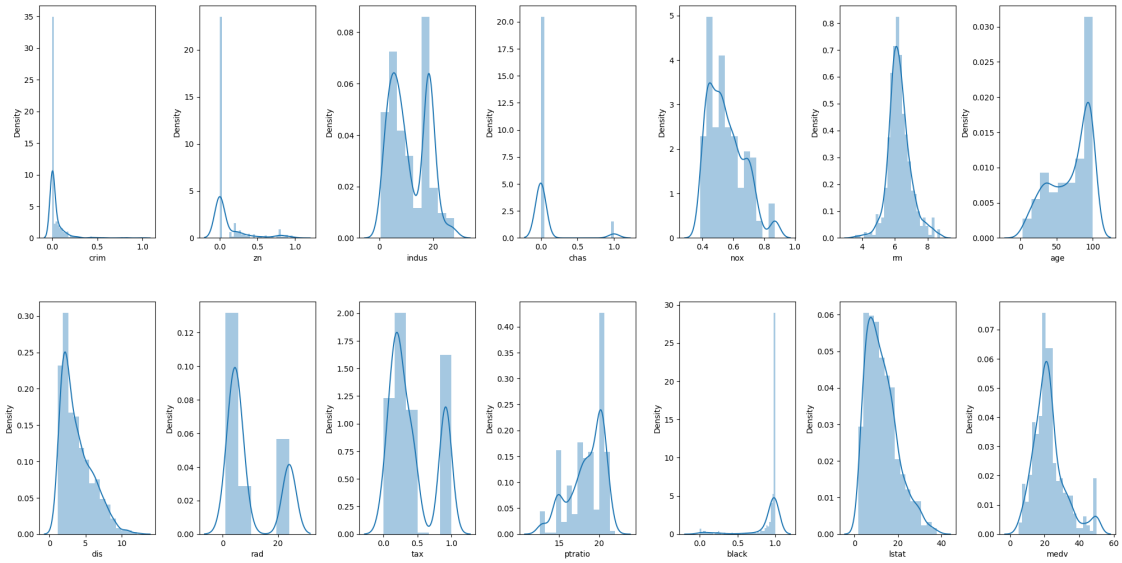


[10]:
```
# create dist plot
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    sns.distplot(value, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



[11]:
```
cols = ['crim', 'zn', 'tax', 'black']
for col in cols:
    # find minimum and maximum of that column
    minimum = min(df[col])
    maximum = max(df[col])
    df[col] = (df[col] - minimum) / (maximum - minimum)
```

[12]:
```
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    sns.distplot(value, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



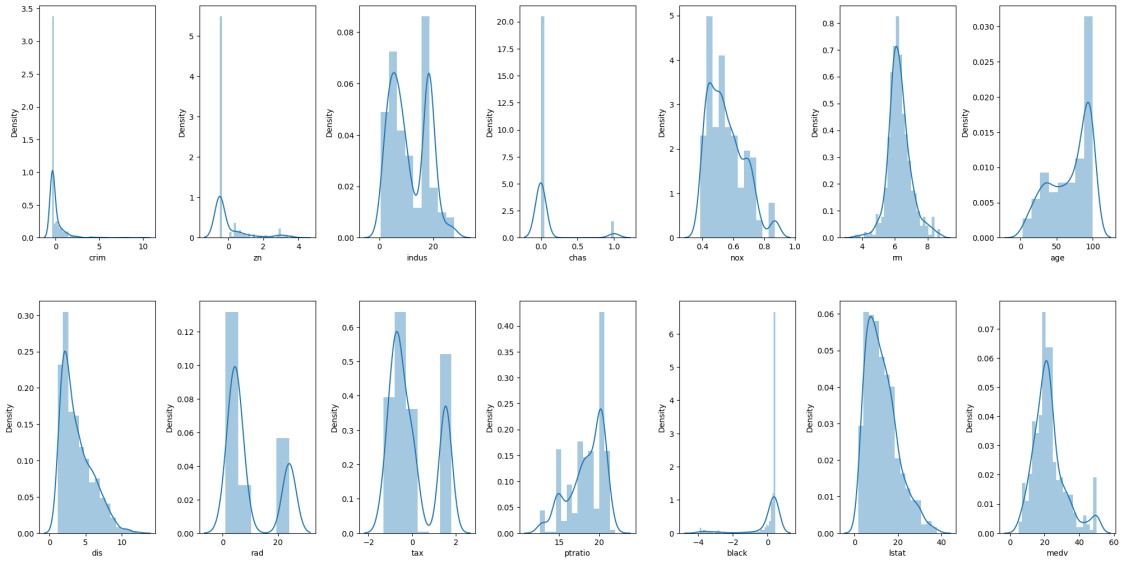
[13]:
```
# standardization
from sklearn import preprocessing
scalar = preprocessing.StandardScaler()

# fit our data
scaled_cols = scalar.fit_transform(df[cols])
scaled_cols = pd.DataFrame(scaled_cols, columns=cols)
scaled_cols.head()
```

[13]:

	crim	zn	tax	black
0	-0.419782	0.284830	-0.666608	0.441052
1	-0.417339	-0.487722	-0.987329	0.441052
2	-0.417342	-0.487722	-0.987329	0.396427
3	-0.416750	-0.487722	-1.106115	0.416163
4	-0.412482	-0.487722	-1.106115	0.441052

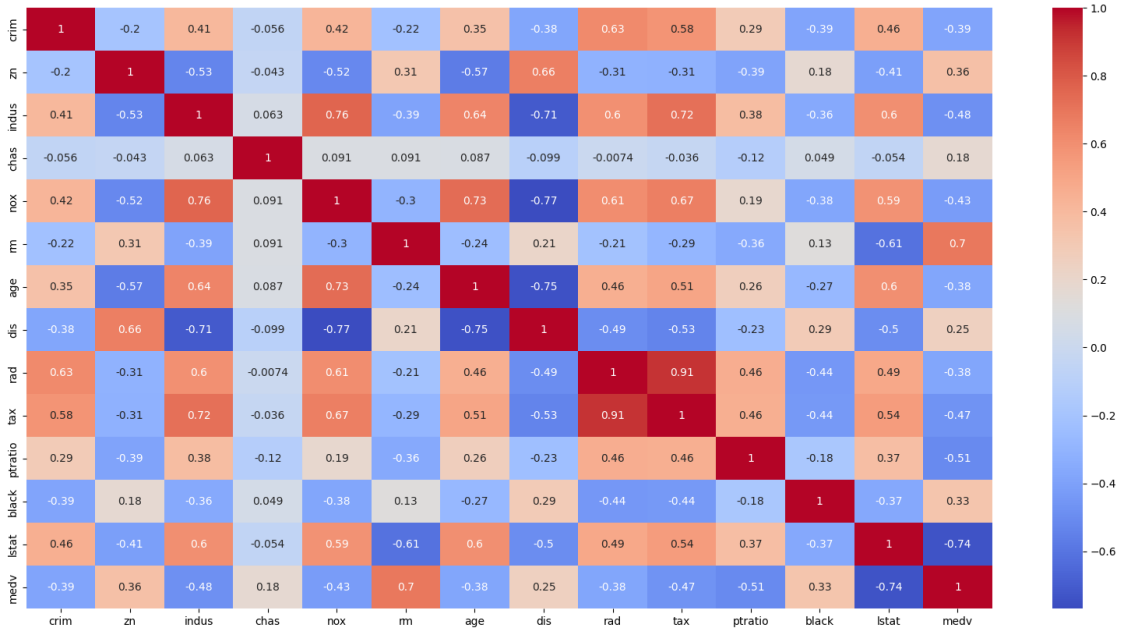
[14]:
```
for col in cols:
    df[col] = scaled_cols[col]
```

[15]:
```
fig, ax = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    sns.distplot(value, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



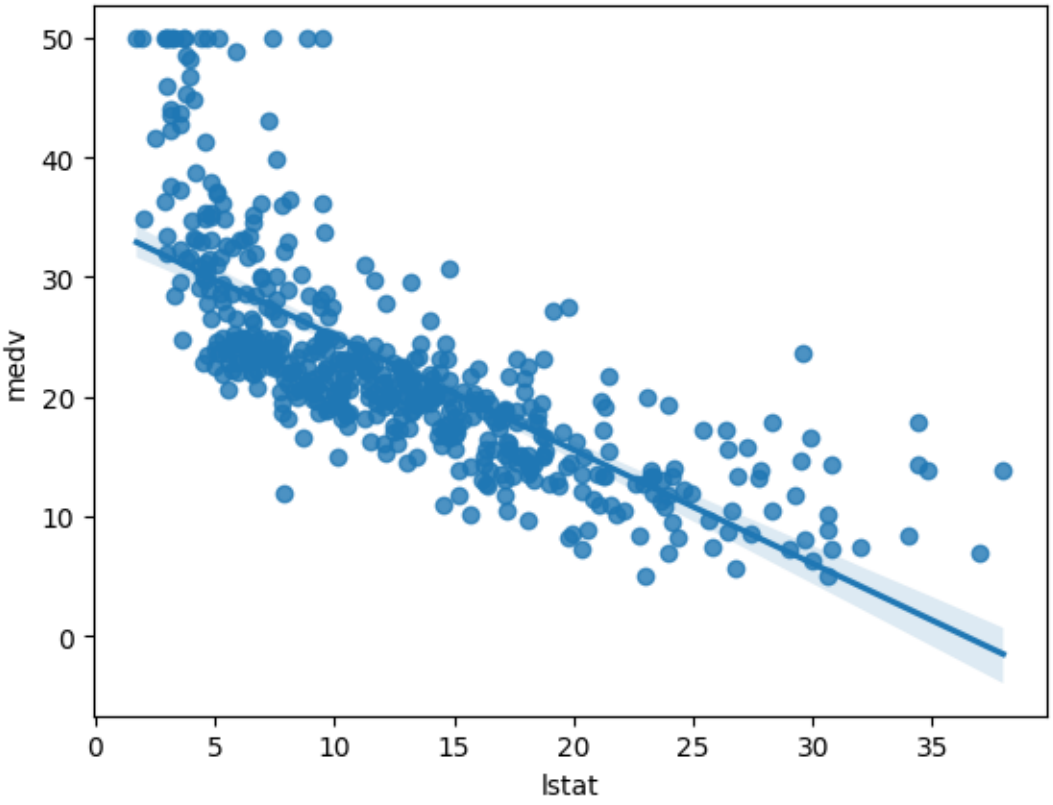
[16]:
```
corr = df.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

[16]: <Axes: >



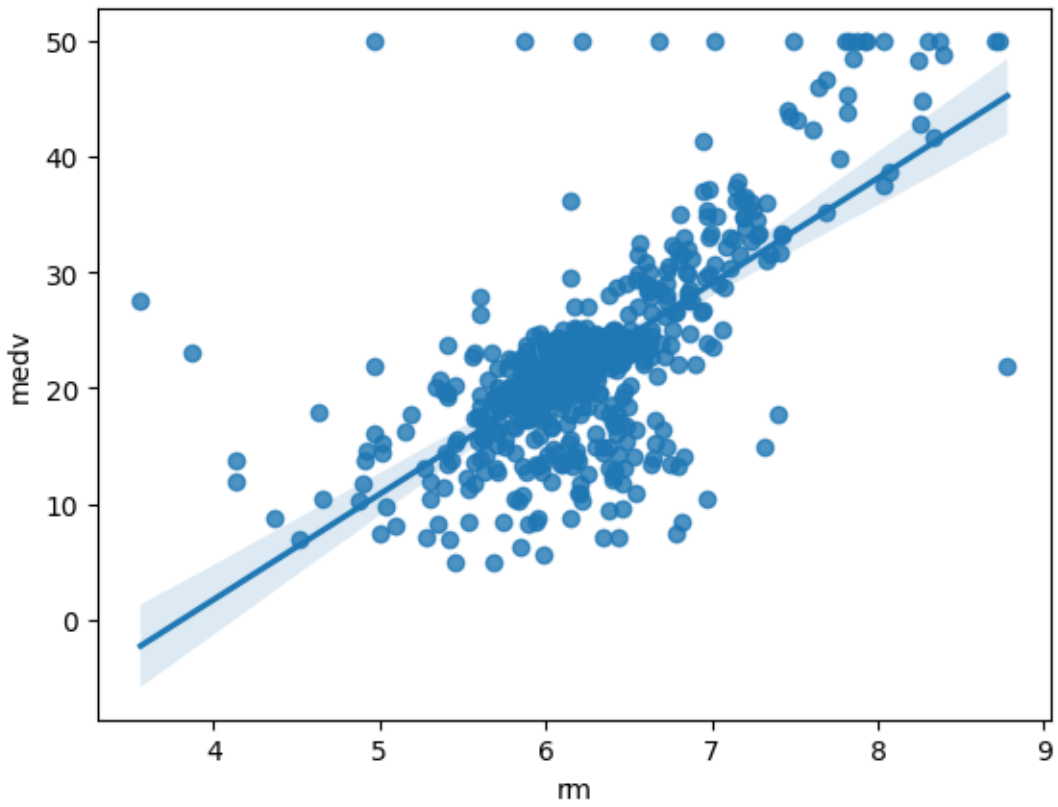
[17]:
```
sns.regplot(y=df['medv'], x=df['lstat'])
```

[17]: <Axes: xlabel='lstat', ylabel='medv'>



[18]:
```
sns.regplot(y=df['medv'], x=df['rm'])
```

[18]: <Axes: xlabel='rm', ylabel='medv'>



[19]:
```
X = df.drop(columns=['medv', 'rad'], axis=1)
y = df['medv']
```

[20]:
```
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
def train(model, X, y):
    # train the model
    x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=42)
    model.fit(x_train, y_train)

    # predict the training set
    pred = model.predict(x_test)

    # perform cross-validation
    cv_score = cross_val_score(model, X, y, scoring='neg_mean_squared_error',␣
 ↪cv=5)
    cv_score = np.abs(np.mean(cv_score))

    print("Model Report")
    print("MSE:",mean_squared_error(y_test, pred))
    print('CV Score:', cv_score)
```

[21]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression(normalize=True)
train(model, X, y)
coef = pd.Series(model.coef_, X.columns).sort_values()
coef.plot(kind='bar', title='Model Coefficients')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-35f65edbe23f> in <cell line: 2>()
      1 from sklearn.linear_model import LinearRegression
----> 2 model = LinearRegression(normalize=True)
      3 train(model, X, y)
      4 coef = pd.Series(model.coef_, X.columns).sort_values()
      5 coef.plot(kind='bar', title='Model Coefficients')

TypeError: LinearRegression.__init__() got an unexpected keyword argument␣
 ↪'normalize'
```

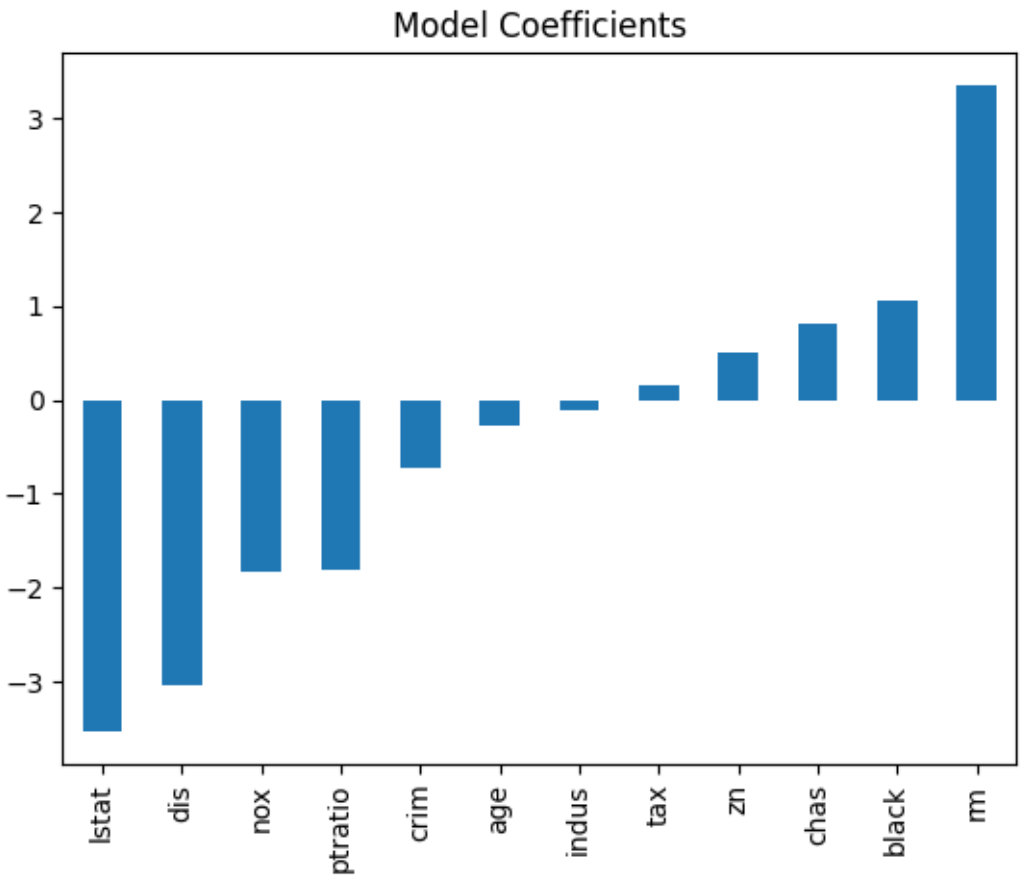
[22]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Assuming X and y are already defined
# X should be your feature matrix, and y should be your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the Linear Regression model
model = LinearRegression(normalize=True)

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

# Get coefficients and sort them
coef = pd.Series(model.coef_, X.columns).sort_values()

# Plot the coefficients
coef.plot(kind='bar', title='Model Coefficients')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-b8c0f61175c9> in <cell line: 12>()
     10
     11 # Initialize the Linear Regression model
```

```
---> 12 model = LinearRegression(normalize=True)
     13
     14 # Train the model using the training sets

TypeError: LinearRegression.__init__() got an unexpected keyword argument␣
 ↪'normalize'
```

[23]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Assuming X and y are already defined
# X should be your feature matrix, and y should be your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data
X_test_scaled = scaler.transform(X_test)

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model using the scaled training sets
model.fit(X_train_scaled, y_train)

# Get coefficients and sort them
coef = pd.Series(model.coef_, index=X.columns).sort_values()

# Plot the coefficients
coef.plot(kind='bar', title='Model Coefficients')
```

[23]:
```
<Axes: title={'center': 'Model Coefficients'}>
```


Model Coefficients
3
2
1
0
−1
−2
−3
lstat
dis
nox
ptratio
crim
age
indus
tax
zn
chas
black
rm