#

数据预处理

#

In [ ]:
```
import pandas as pd
# 加载数据
df = pd.read_csv('boston_housing_data.csv')
# 显示原始数据中的缺失值情况
print("原始数据的缺失值情况：")
print(df.isnull().sum())
# 使用线性插值方法填充缺失值
df_interpolated = df.interpolate(method='linear')
# 再次检查数据中的缺失值情况
print("插值后数据的缺失值情况：")
print(df_interpolated.isnull().sum())
# 如果数据集中的首行或尾行存在缺失值，插值无法解决，可以选择填充：
df_filled = df_interpolated.fillna(method='bfill').fillna(method='ffill')
# 保存处理后的数据
#df_filled.to_csv("/home/sunrong/homework1/after_data.csv", index=False)
# 打印一部分处理后的数据查看
print(df_filled.head())
```

```
原始数据的缺失值情况：
CRIM        0
ZN          0
INDUS       0
CHAS        0
NOX         0
RM          0
AGE         0
DIS         0
RAD         0
TAX         0
PIRATIO     0
B           0
LSTAT       0
MEDV       54
dtype: int64
插值后数据的缺失值情况：
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PIRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222.0

   PIRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

#

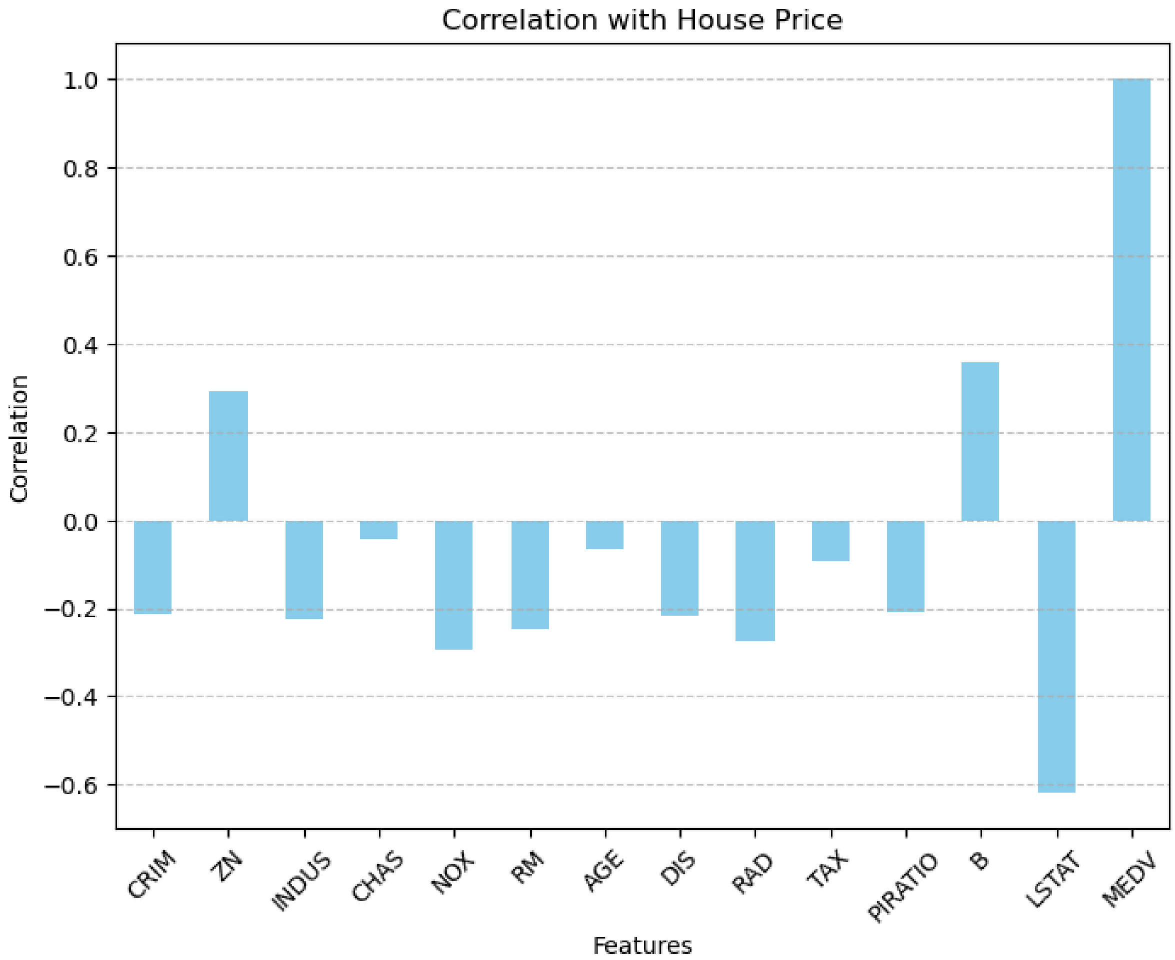
相关性分析

#

In [ ]:
```
import matplotlib.pyplot as plt
# 计算相关系数
correlation = df_filled .corr()
# 提取最后一列数据
last_column = correlation.iloc[:, -1]

# 可视化
plt.figure(figsize=(8, 6))
last_column.plot(kind='bar', color='skyblue')
plt.title('Correlation with House Price')
```

```
plt.xlabel('Features')
plt.ylabel('Correlation')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



#

主成分分析

#

In [ ]:
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np
# 加载数据
df = df_filled
# 分离特征和目标变量
X = df.iloc[:, :-1]  # 除了最后一列，其他都是特征
y = df.iloc[:, -1]   # 最后一列是房价，即目标变量
feature_names = X.columns  # 保存特征名称
# 标准化特征
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# 创建PCA实例并指定保留90%的方差
pca = PCA(n_components=0.90)
X_pca = pca.fit_transform(X_scaled)

# 分析PCA加载量，以选取原始特征
components = pca.components_
```

```python
important_features = set()  # 使用集合避免重复添加特征
for component in components:
    # 对于每个主成分，找到绝对值最大的加载量对应的特征索引
    max_index = np.argmax(np.abs(component))
    important_features.add(feature_names[max_index])

# 从原始数据中选择这些重要的特征
selected_features = df[list(important_features)]

# 将这些特征及目标变量一起保存到新的CSV文件中
selected_features['MEDV'] = y  # 添加目标变量

print("选取的特征包括：", list(important_features))
```

```
选取的特征包括： ['CHAS', 'ZN', 'DIS', 'LSTAT']
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_500\1246984030.py:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  selected_features['MEDV'] = y  # 添加目标变量
```

#

神经网络建模

#

In [ ]:

```python
import pandas as pd
import torch
from torch.utils.data import DataLoader, Dataset, random_split
import torch.nn as nn
import torch.optim as optim
#  定义自定义 Dataset 类
class BostonDataset(Dataset):
    def __init__(self, features, targets):
        self.features = features
        self.targets = targets

    def __len__(self):
        return len(self.targets)

    def __getitem__(self, idx):
        return self.features[idx], self.targets[idx]

# 加载数据
df = selected_features
features = df.iloc[:, :-1].values
targets = df.iloc[:, -1].values

# 将数据转换为tensor
features = torch.tensor(features, dtype=torch.float32)
targets  = torch.tensor(targets, dtype=torch.float32).view(-1, 1)

# # 数据标准化（手动）
# mean = features.mean(0, keepdim=True)
# std = features.std(0, keepdim=True, unbiased=False)
# features = (features - mean) / std

# 创建数据集
```

```python
dataset = BostonDataset(features, targets)

# 数据集划分
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# 创建数据加载器
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)


class Net(nn.Module):
    def __init__(self, input_size):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, 16)
        self.relu1 = nn.GELU()
        self.fc2 = nn.Linear(16, 32)
        self.relu2 = nn.GELU()
        self.fc3 = nn.Linear(32, 1)

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        x = self.relu2(self.fc2(x))
        x = self.fc3(x)
        return x
# model = torch.nn.Sequential(
#     torch.nn.Linear(input_neuron,hidden_neuron),
#     torch.nn.Sigmoid(),
#     torch.nn.Linear(hidden_neuron , output_neuron)
# )
# 初始化网络和优化器
model = Net(input_size=features.shape[1])
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# 训练模型
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    for inputs, targets in train_loader:
        #print(inputs)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

# 测试模型
model.eval()
total_loss = 0
with torch.no_grad():
    for inputs, targets in test_loader:
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        total_loss += loss.item()

print(f'Test Loss: {total_loss / len(test_loader)}')
```

```
Test Loss: 8.6738355954488116
```