Filter Based Methods - Mutual Information - 0.578273098930927

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression, Lasso
```

```
dataset = fetch_california_housing()
n_features = dataset.data.shape[1]
print(n_features)
feature_names = dataset.feature_names
print(feature_names)
#print(dataset.DESCR)
```

```
8
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']
```

```
x=dataset['data']
y=dataset['target']
feature_names = dataset['feature_names']
```

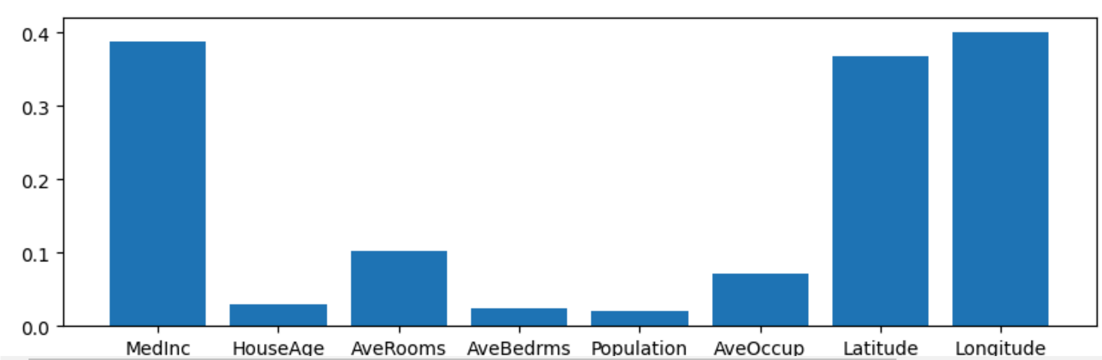
```
from sklearn.feature_selection import mutual_info_regression, SelectPercentile,SelectKBest
```

```
mi = mutual_info_regression(x,y)
#print(mi)
# Create a dictionary of feature names and their importance scores
feature_importance = dict(zip(dataset.feature_names, mi))
# Print in a formatted way
for feature, importance in feature_importance.items():
    print(f"{feature}: {importance:.4f}")
```

```
MedInc: 0.3875
HouseAge: 0.0294
AveRooms: 0.1032
AveBedrms: 0.0242
Population: 0.0213
AveOccup: 0.0722
Latitude: 0.3688
Longitude: 0.4003
```

Visualise feature Selection

```
plt.figure(figsize=(10,3))
plt.bar(feature_names,mi)
plt.show()
```



How to pick the feature

Option 1

```
x_new = SelectKBest(mutual_info_regression,k=5).fit_transform(x,y)
print(x_new.shape)
# Get scores for all features
selector = SelectKBest(mutual_info_regression,k=5).fit(x,y)
scores = selector.scores_
# Create a dictionary of feature names and their scores
feature_scores = dict(zip(dataset.feature_names, scores))
# Sort features by score
sorted_features = sorted(feature_scores.items(), key=lambda x: x[1], reverse=True)[:5]

print("Top 5 features with their scores:")
for feature, score in sorted_features:
    print(f"{feature}: {score:.4f}")
```

```
(20640, 5)
Top 5 features with their scores:
Longitude: 0.4024
MedInc: 0.3880
Latitude: 0.3678
AveRooms: 0.1032
AveOccup: 0.0728
```

Option 2

```
x_new = SelectPercentile(mutual_info_regression,percentile=50).fit_transform(x,y)
print(x_new.shape)
# Get scores for all features
selector = SelectKBest(mutual_info_regression,k=5).fit(x,y)
scores = selector.scores_
# Create a dictionary of feature names and their scores
feature_scores = dict(zip(dataset.feature_names, scores))
# Sort features by score
sorted_features = sorted(feature_scores.items(), key=lambda x: x[1], reverse=True)[:5]

print("Top 5 features with their scores:")
for feature, score in sorted_features:
    print(f"{feature}: {score:.4f}")
```

```
(20640, 4)
Top 5 features with their scores:
Longitude: 0.4009
MedInc: 0.3874
Latitude: 0.3713
AveRooms: 0.1036
AveOccup: 0.0720
```

```
x_train,x_test,y_train,y_test=train_test_split(x_new,y, test_size=0.2)
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print(r2_score(y_test,y_pred))
```

```
0.578273098930927
```

Filter Based Methods - Chi Squared Cannot be used for Continuous dataset

Double-click (or enter) to edit

**Filter Based Methods - Pearson Correlation* *- 0.5173065395604509

f_regression: A scoring function that computes F-statistics between each feature and target

SelectKBest: A feature selector that selects top k features based on a scoring function

```
from sklearn.feature_selection import f_regression, SelectKBest
```

```
#x_new = SelectKBest(f_regression,k=8).fit_transform(x,y)
#print(x_new.shape)

```

```
#Create the selector
selector = SelectKBest(f_regression, k=6)

# Fit the selector (without transforming)
selector.fit(x, y)

# Get the selected feature mask (True for selected features)
selected_mask = selector.get_support()

# Get the names of selected features using the mask
#selected_features = dataset.feature_names[selected_mask]
selected_features = [feature for feature, is_selected in zip(dataset.feature_names, selected_mask) if is_selected]
"""
zip(dataset.feature_names, selected_mask) pairs each feature name with its corresponding boolean value (True/False) from the selected_mas
for feature, is_selected in ... loops through each pair, where:

feature gets the feature name from dataset.feature_names
is_selected gets the corresponding True/False value from selected_mask


if is_selected only keeps features where the mask value is True
[feature for ...] creates a list containing only the feature names that passed the condition
"""
# Print selected features and their scores
for feature, score in zip(selected_features, selector.scores_):
    print(f"{feature}: {score:.4f}")
```

```
MedInc: 18556.5716
HouseAge: 232.8415
AveRooms: 487.7575
AveBedrms: 45.1086
Latitude: 12.5474
Longitude: 11.6353
```

```
x_train,x_test,y_train,y_test  = train_test_split(x_new,y,test_size=0.2)
```

```
model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(r2_score(y_test,y_pred))
```

```
0.5979581682306951
```

To drop feature manually

```
import pandas as pd
x_pd = pd.DataFrame(x,columns=feature_names)
x_pd.head(10)
x_pd.corr()
```

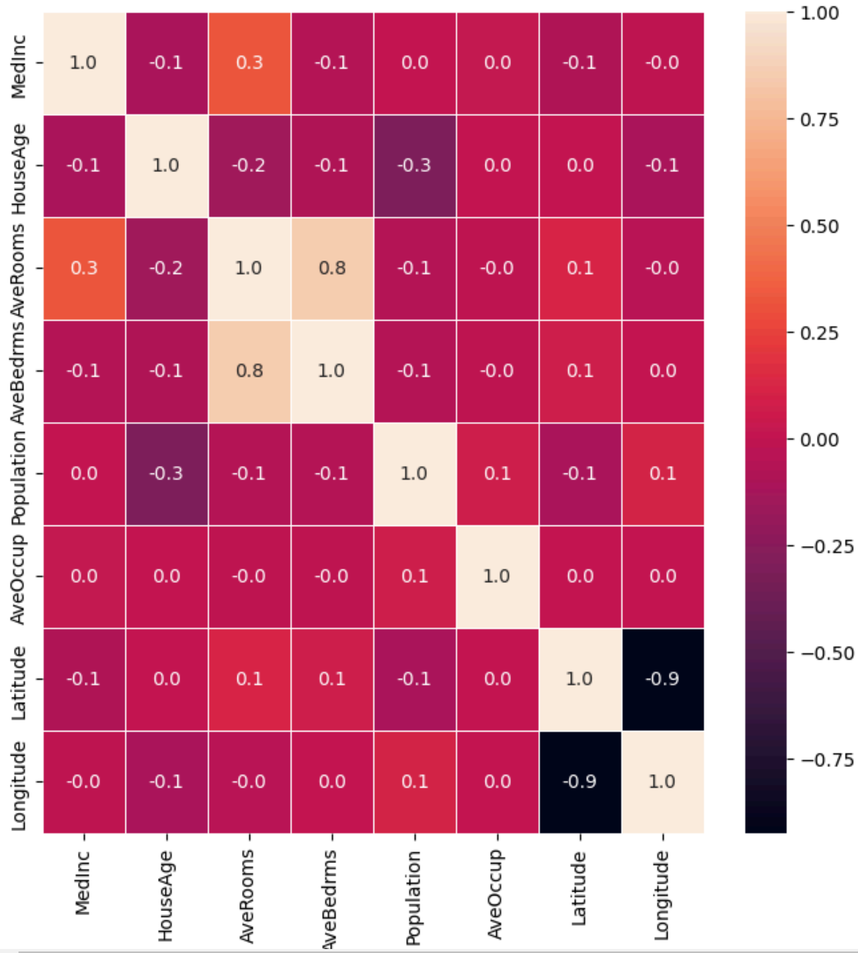
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.015176
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.108197
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.027540
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013344
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.099773
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002476
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924664
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000000

```
y_series = pd.Series(y)
x_pd.corrwith(y_series).sort_values(ascending=False)
```

	0
MedInc	0.688075
AveRooms	0.151948
HouseAge	0.105623
AveOccup	-0.023737
Population	-0.024650
Longitude	-0.045967
AveBedrms	-0.046701
Latitude	-0.144160

```
import seaborn as sns
```

```
f,ax=plt.subplots(figsize=(8,8))
sns.heatmap(x_pd.corr(),annot=True,linewidths=0.5,fmt='.1f',ax=ax)
plt.show()
```



Latitude and Longitude have the strongest negative correlation (-0.9), meaning as latitude increases, longitude tends to decrease significantly. This makes sense geographically. AveRooms and AveBedrms have a strong positive correlation (0.8), which is logical since houses with more total rooms tend to have more bedrooms.

Before dropping these features, you should consider: Check their importance scores (using mutual_info_regression or feature_importance_ from your model) to see which features are most predictive of your target variable

```
mi = mutual_info_regression(x, y)
feature_importance = dict(zip(dataset.feature_names, mi))
sorted_importance = dict(sorted(feature_importance.items(), key=lambda x: x[1], reverse=True))
```

```

for feature, importance in sorted_importance.items():
    print(f"{feature}: {importance:.4f}")
```

```
Longitude: 0.4013
MedInc: 0.3868
Latitude: 0.3718
AveRooms: 0.1032
AveOccup: 0.0727
HouseAge: 0.0331
AveBedrms: 0.0244
Population: 0.0212
```

```
x_new = x_pd.drop(['Longitude','AveRooms'],axis=1)
x_new.head()
```

	MedInc	HouseAge	AveBedrms	Population	AveOccup	Latitude
0	8.3252	41.0	1.023810	322.0	2.555556	37.88
1	8.3014	21.0	0.971880	2401.0	2.109842	37.86
2	7.2574	52.0	1.073446	496.0	2.802260	37.85
3	5.6431	52.0	1.073059	558.0	2.547945	37.85
4	3.8462	52.0	1.081081	565.0	2.181467	37.85

Next steps: Generate code with x_new View recommended plots New interactive sheet

```
x_train,x_test,y_train,y_test = train_test_split(x_new,y,test_size=0.2)
model = LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(r2_score(y_test,y_pred))
```

```
0.5173065395604509
```

Wrapper Based Methods - Recursive Feature Elimination (RFE) - 0.5896765399567889

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import Lasso
```

```
estimator = Lasso()
#Creates a Lasso (Least Absolute Shrinkage and Selection Operator) regression model which will be used as the base estimator.
#Lasso is a type of linear regression that includes L1 regularization
selector = RFE(estimator,n_features_to_select=5,step=1).fit(x,y)
print(selector.ranking_)

#Print features with their rankings
for feature, rank in zip(dataset.feature_names, selector.ranking_):
    print(f"{feature}: {rank}")

# Or just print selected features
selected_features = [feature for feature, is_selected in zip(dataset.feature_names, selector.support_) if is_selected]
print("Selected features:", selected_features)
```

```
[1 1 4 3 1 2 1 1]
MedInc: 1
HouseAge: 1
AveRooms: 4
AveBedrms: 3
Population: 1
AveOccup: 2
Latitude: 1
Longitude: 1
Selected features: ['MedInc', 'HouseAge', 'Population', 'Latitude', 'Longitude']
```

```
x_new = selector.transform(x)
print(x_new.shape)
```

```
(20640, 5)
```

```
x_train,x_test,y_train,y_test=train_test_split(x_new,y, test_size=0.2)
```

```
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print(r2_score(y_test,y_pred))
```

```
0.5896765399567889
```