PyPrep : Data Cleaning Library

PyPrep is a wrapper around Pandas and Sklearn that makes data cleaning and preprocessing easier.

To install, use:
`pip install pyprep`
or download the PyPrep file and import it into your Jupyter notebook or Python workspace.

Documentation and Examples:

The data being used is the Califonia Housing Prices dataset.

In [51]:
```
from main import PyPrep
import pandas as pd

df = pd.read_csv('housing.csv')

df.head()
```

Out[51]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

Initialization

Parameters:

- **data** : Pandas.DataFrame
- **copy** : Boolean, default True
- **y** : String, default None

Returns:

- **PyPrep object**

Initialize with only data:

In [52]:
```
pyp = PyPrep(data = df)
```

Initialize with a target variable:

In [53]:
```
pyp = PyPrep(data = df, y = 'median_house_value')
```

Functions

PyPrep.head()

> Wrapper around Pandas head method

In [54]:
```
pyp.head()
```

Out[54]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

PyPrep.get_numeric()

> Returns the numeric columns in the dataframe in list form

In [55]:
```
numerics = pyp.get_numeric()
numerics
```

Out[55]:
```
array(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'], dtype=object)
```

> Since the return value is a list of columns, it can be used directly on the dataframe, or on the PyPrep object

In [56]:
```
df[numerics].head()
```

Out[56]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

In [57]:
```
pyp.data[numerics].head()
```

Out[57]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

PyPrep.get_categorical()

> Similar to *get_numeric()*, but returns a list of categorical columns

In [58]:
```
categorical = pyp.get_categorical()
categorical
```

Out[58]:
```
array(['ocean_proximity'], dtype=object)
```

In [59]:
```
df[categorical].head()
```

Out[59]:

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	NEAR BAY
3	NEAR BAY
4	NEAR BAY

In [60]:
```
pyp.data[categorical].head()
```

Out[60]:

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	NEAR BAY
3	NEAR BAY
4	NEAR BAY

PyPrep.get_missing(*type* = '*list*')

Parameters:

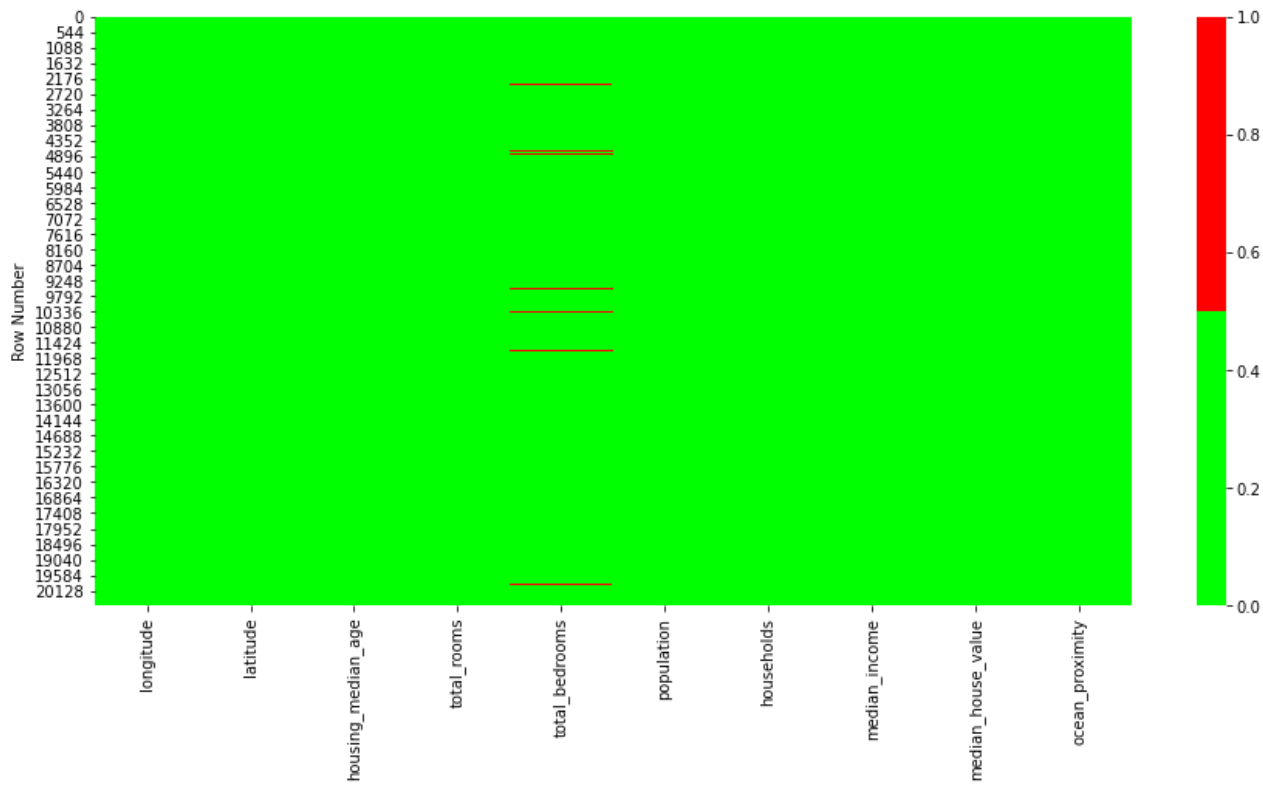
- **type** : String: *list, heatmap, percentage*

> *If type is "list", the method returns a list of null values for each column*
> *If type is "heatmap", the method returns a heatmap of null values for each column*
> *If type is "percentage", the method returns a bar plot of null values in each column*

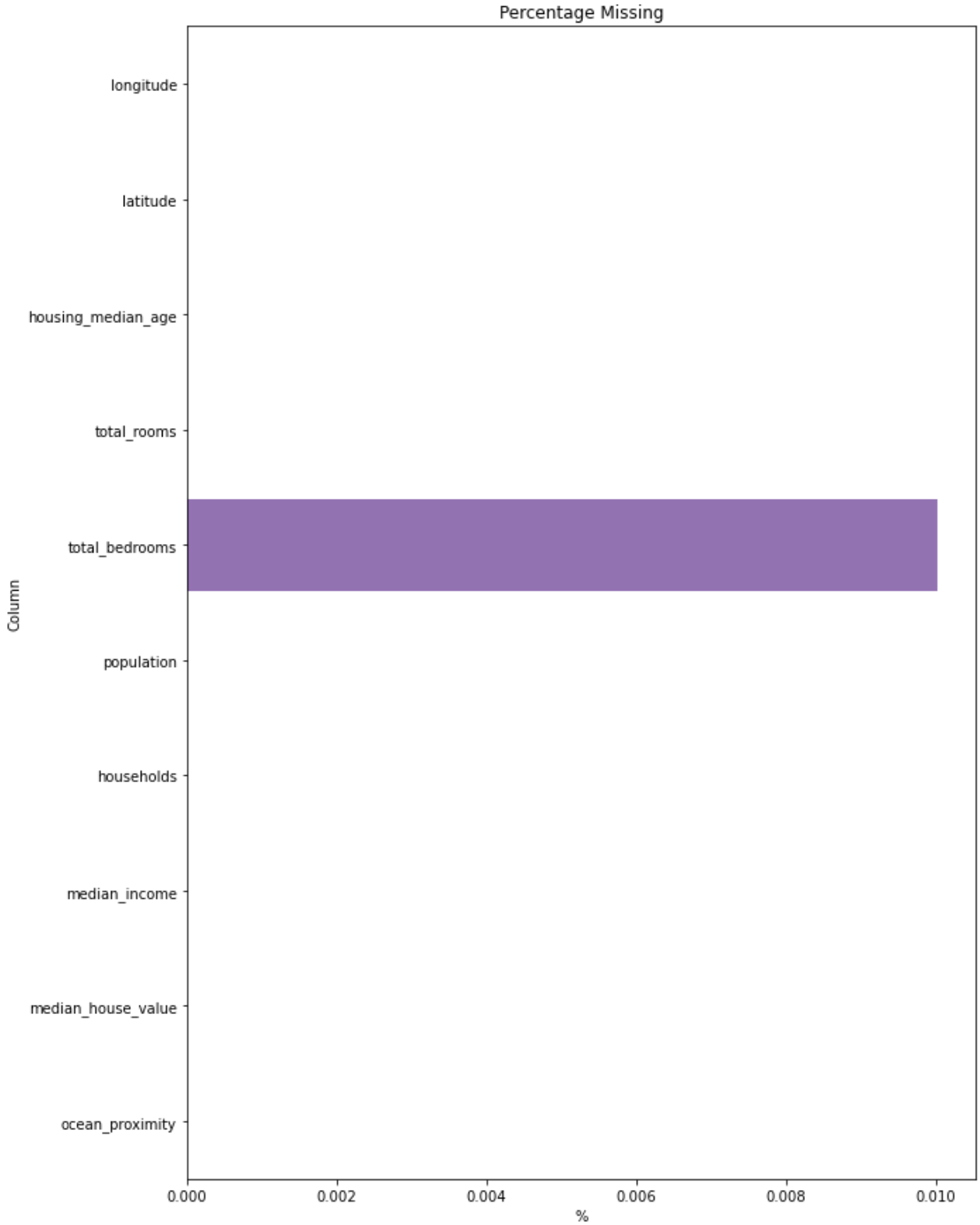
In [61]:
```
pyp.get_missing('list')
```

Out[61]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

In [62]:
```
pyp.get_missing('heatmap')
```



In [63]:
```
pyp.get_missing('percentage')
```



PyPrep.get_duplicates(*plot = False*)

Parameters:

- **plot** : Boolean

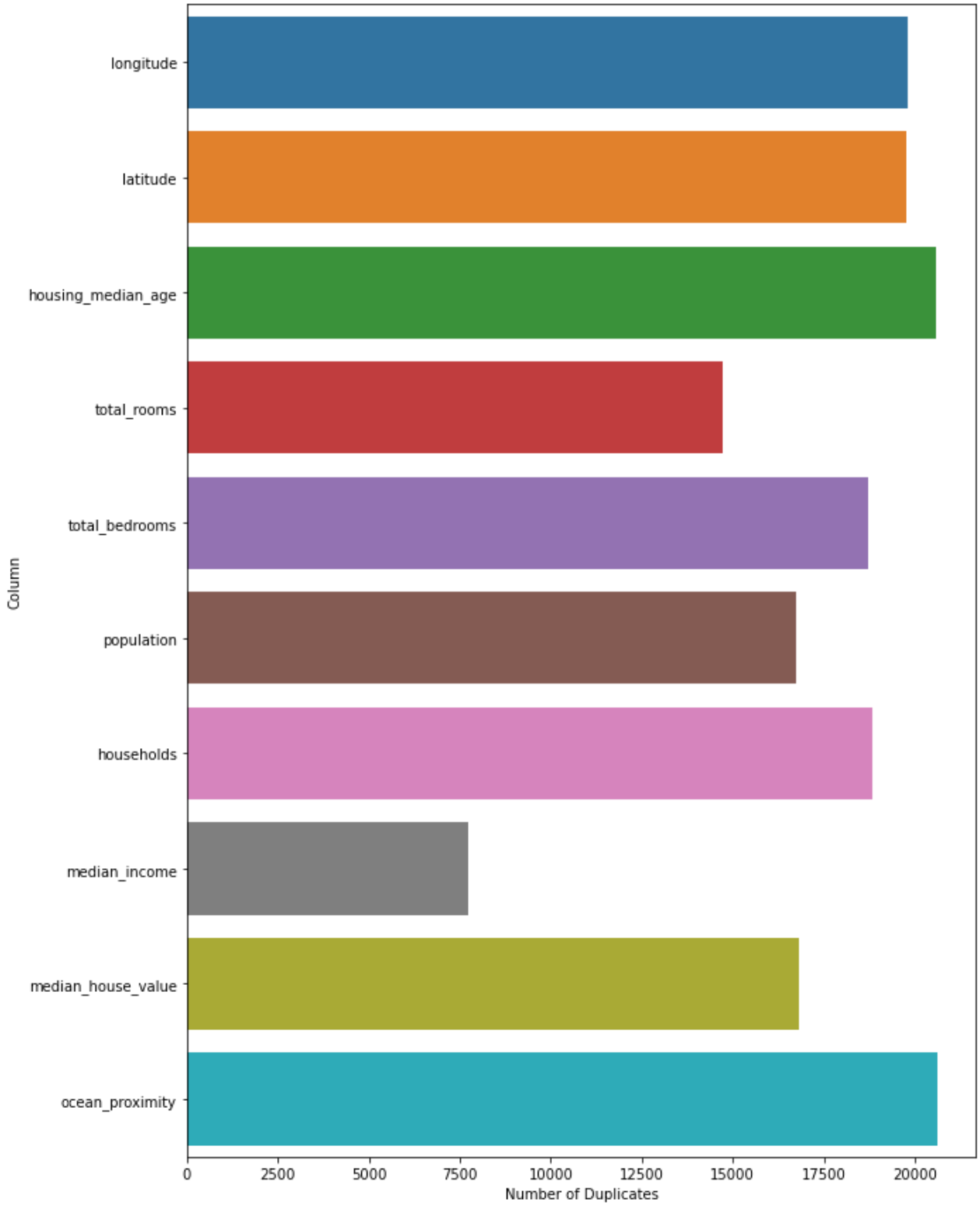
> *Returns the number of duplicate values in each column*
> *If **plot** is True, the method returns a barplot of the number of duplicate values in each*

> *column*

In [64]:
```
pyp.get_duplicates()
```

Out[64]:
```
longitude             19796
latitude              19778
housing_median_age    20588
total_rooms           14714
total_bedrooms        18716
population            16752
households            18825
median_income          7712
median_house_value    16798
ocean_proximity       20635
dtype: int64
```

In [65]:
```
pyp.get_duplicates(plot=True)
```



PyPrep.get_zscore(*columns = None, threshold = 3*)

Parameters:

- **columns** : List of Columns
- **threshold** : Integer

> *Calculates the z-score of each numeric column in the DataFrame*
> *Returns rows where the z-score is greater than or equal to **threshold** argument*

In [66]:
```
pyp.get_zscore()
```

Out[66]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
95	-122.26	37.80	36.0	5329.0	2477.0	3469.0	2323.0
104	-122.26	37.81	34.0	5871.0	1914.0	2689.0	1789.0
116	-122.25	37.83	28.0	5022.0	1750.0	2558.0	1661.0
131	-122.19	37.84	18.0	1617.0	210.0	533.0	194.0
283	-122.16	37.79	22.0	12842.0	2048.0	4985.0	1967.0
...	...	...	...	...	...	...	...
20530	-121.76	38.57	11.0	15018.0	3008.0	7984.0	2962.0
20543	-121.74	38.55	33.0	6861.0	1820.0	3717.0	1767.0
20544	-121.76	38.55	23.0	8800.0	1857.0	6330.0	1832.0
20563	-121.75	38.67	9.0	12139.0	2640.0	6837.0	2358.0
20629	-121.39	39.12	28.0	10035.0	1856.0	6912.0	1818.0

892 rows × 9 columns

In [67]:
```
pyp.get_zscore(columns = ['housing_median_age'], threshold = 1)
```

Out[67]:

	housing_median_age
2	52.0
3	52.0
4	52.0
5	52.0
6	52.0
...	...
20557	43.0
20561	42.0
20562	45.0
20589	48.0
20592	52.0

3582 rows × 1 columns

PyPrep.get_repetitive(*threshold = .95*)

Parameters:

- **threshold** : Float <= .99

> *Returns columns that have reptitive values greater than or equal to **threshold** argument*

In [68]:
```
pyp.get_repetitive(threshold = .4)
```

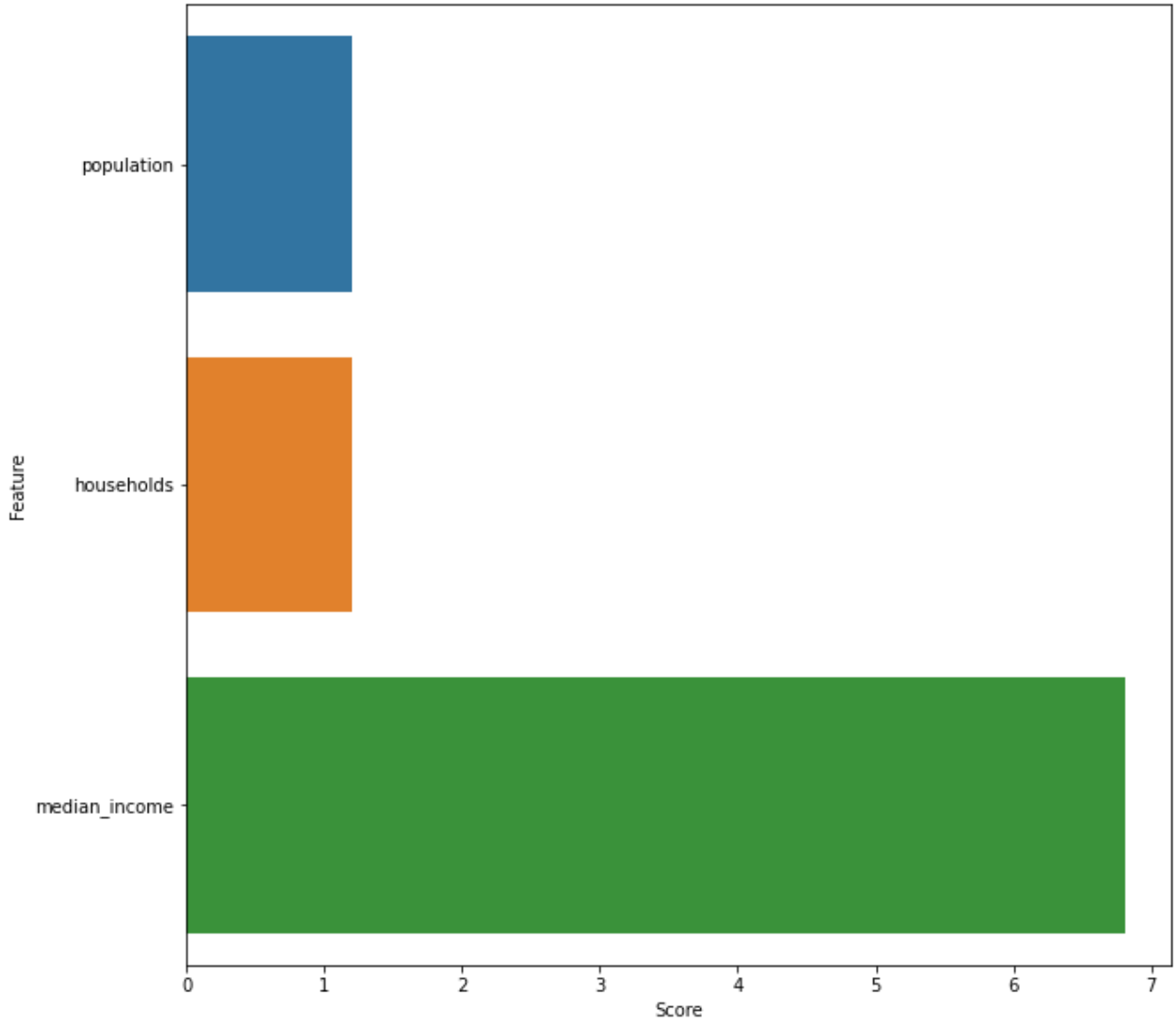
PyPrep.select_best(*columns, k = 5, dtype = 'numeric', plot = False*)

Parameters:

- **columns** : List of Columns
- **k** : Number of variables to select
- **dtype** : String : 'numeric' or 'categorical'
- **plot** : Boolean

> *Wrapper around Sklearn SelectKBest*
> *Selects **k** most statistically significant rows*

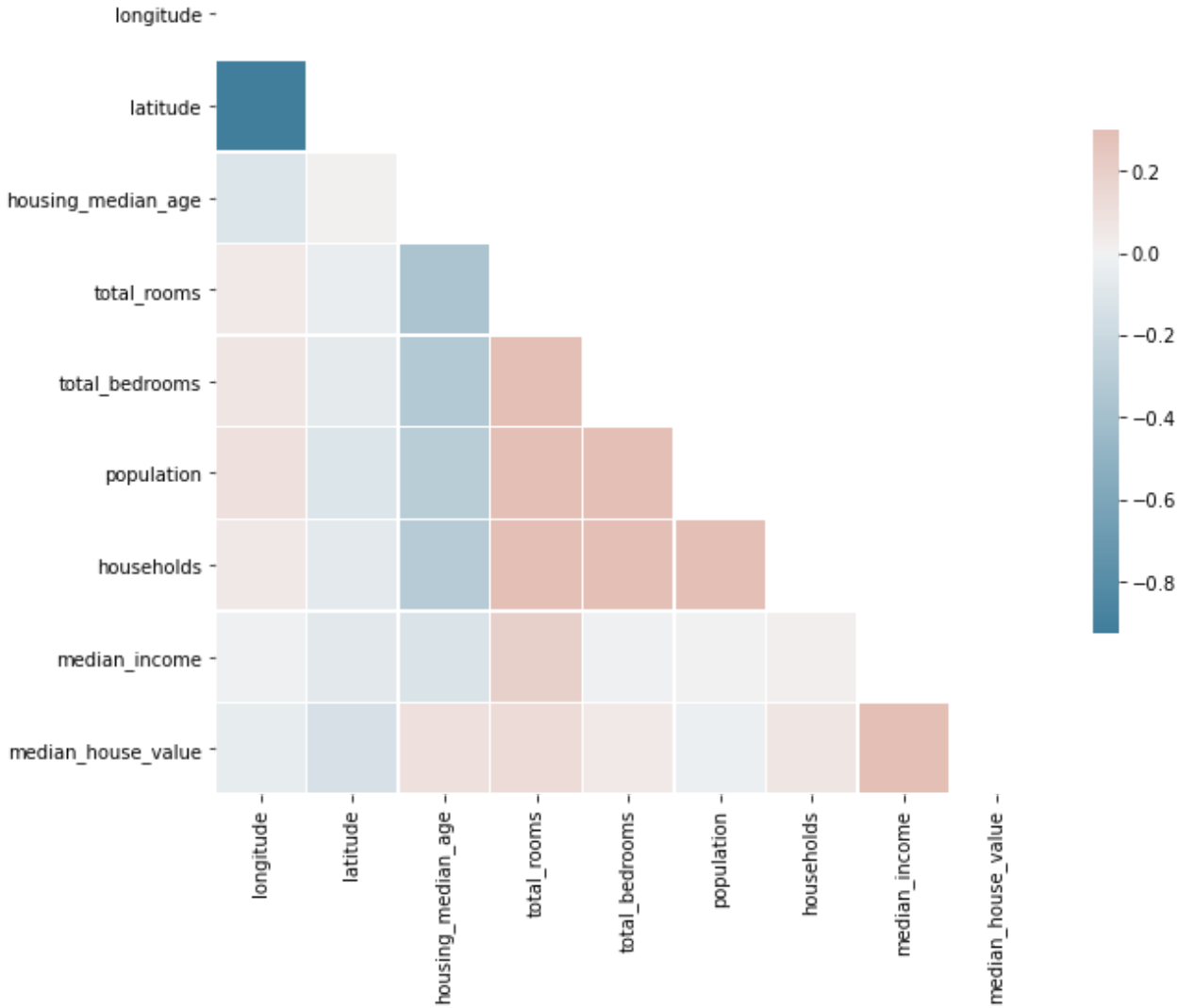
In [69]:
```
cols = ['population', 'households', 'median_income']
pyp.select_best(k=3, columns = cols, plot = True)
```



PyPrep.corr_plot()

> *Returns a correlation plot between numeric values and target variable*

In [70]:
```
pyp.corr_plot()
```



PyPrep.encode(*columns = None, method = 'onehot', drop = False)

Parameters:

- **columns** : List of Columns
- **method** : String : 'onehot' or 'label'
- **drop** : Boolean

> *Wrapper around OneHotEncoding or LabelEncoder*