dav

October 17, 2024

[ ]:
```
# TODO:
# 1.        Import
# 2.        Read csv
# 3.        Head
# 4.        Shape
# 5.        Describe
# 6.        Isnull().sum()
# 7.        Dropna(inplace=True)
# 8.        Isnull().sum()
# 9.        Separate numerical and categorical cols using select_dtypes
# 10.        Calculate mean, median, std dev, quartiles
# 11.        Correlation matrix (heatmap)
# 12.        Find top 5 features from heatmap
# 13.        Plot Histogram (all numerical cols)
# 14.        Box Plot (all numerical cols)
# 15.        Scatter Plot (all numerical cols, upon a target)
# 16.        Feature Engineering
# 17.        Define function to find outliers (IQR) Note: if df.quantile(float␣
↪val); np.percentile(whole val)
# 18.         Find number of outliers
# 19.        Remove outliers or fill with median
# 20.        Display bar chart for categorical variable
# 21.        Display mean, median, mode, std dev, IQR
# 22.        Scale the numerical data - MinMax or StandardScaler
# 23.        Select one column for KDE histogram, making it normal distribution
# 24.        Plot regular histogram and KDE histogram
# 25.        Plot QQ plot (stats.probplot)
# 26.        Do Shapiro and KSTest (stats.shapiro and stats.kstest)
# 27.        If p values are > 0.05, normal or else not normal dist
# 28.        Transformation using np.log1p or boxcox (stats.boxcox)
# 29.        Histogram and QQ Plot for new transformed value
# 30.        Do those tests again and check if p > 0.05
# 31.        Select independent and dependent variable (for linear regression)␣
↪(X - [[]
# 32.        Split the data for train and test (0.2 for test size)
# 33.        Create model and fit
```

```
# 34.        Find intercept and slope (intercept -> model.intercept_, slope ->␣
↪model.coef_[0]) and print regression eqn
# 35.        Predict using model (X_test)
# 36.        Plot regression line
# 37.        Display metrics
# 38.        Display residual graph
```

[47]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

[48]:
```
url = "https://raw.githubusercontent.com/ageron/handson-ml/refs/heads/master/
↪datasets/housing/housing.csv"
df = pd.read_csv(url)
```

[49]:
```
df.head()
```

[49]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

[50]:
```
df.shape
```

[50]:
```
(20640, 10)
```

[51]:
```
df.describe()
```

[51]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
```

```
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

[52]:
```python
df.isnull().sum()
```

[52]:
```
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
```

[53]:
```python
df.dropna(inplace=True)
```

[54]:
```python
df.isnull().sum()
```

[54]:
```
longitude             0
latitude              0
housing_median_age    0
```

```
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

[55]:
```
numerical_cols = df.select_dtypes(include=["float64", "int64", "number"])
categorical_cols = df.select_dtypes(include=["object"])
```

[56]:
```
numerical_cols_names = numerical_cols.columns
categorical_cols_names = categorical_cols.columns
```

[57]:
```
mean_values = np.mean(numerical_cols, axis=0)
median_values = np.median(numerical_cols, axis=0)
std_dev = np.std(numerical_cols, axis=0)
quartiles = np.percentile(numerical_cols, [25, 50, 75], axis=0)
```

[58]:
```
print(f"{mean_values=}\n\n\n{median_values=}\n\n\n{std_dev=}\n\n\n{quartiles=}")
```

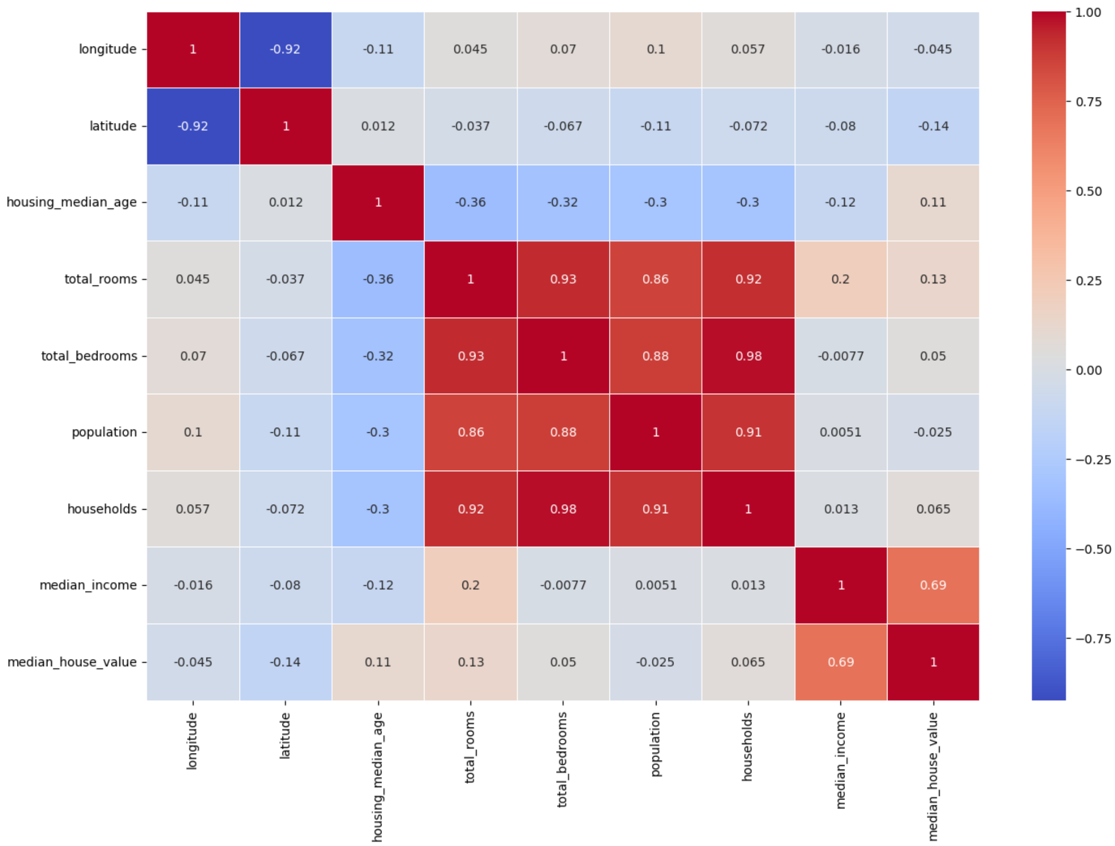
```
mean_values=longitude              -119.570689
latitude                35.633221
housing_median_age      28.633094
total_rooms           2636.504233
total_bedrooms         537.870553
population            1424.946949
households             499.433465
median_income            3.871162
median_house_value  206864.413155
dtype: float64


median_values=array([-1.1849e+02,  3.4260e+01,  2.9000e+01,  2.1270e+03,
4.3500e+02,
        1.1660e+03,  4.0900e+02,  3.5365e+00,  1.7970e+05])


std_dev=longitude                 2.003529
latitude                 2.136295
housing_median_age      12.591497
total_rooms           2185.216092
total_bedrooms         421.374759
population            1133.180760
households             382.289871
median_income            1.899245
```

```
median_house_value    115432.842328
dtype: float64
```

```
quartiles=array([[-1.2180e+02,  3.3930e+01,  1.8000e+01,  1.4500e+03,
2.9600e+02,
         7.8700e+02,  2.8000e+02,  2.5637e+00,  1.1950e+05],
       [-1.1849e+02,  3.4260e+01,  2.9000e+01,  2.1270e+03,  4.3500e+02,
         1.1660e+03,  4.0900e+02,  3.5365e+00,  1.7970e+05],
       [-1.1801e+02,  3.7720e+01,  3.7000e+01,  3.1430e+03,  6.4700e+02,
         1.7220e+03,  6.0400e+02,  4.7440e+00,  2.6470e+05]])
```

[59]:
```python
corr_matrix = numerical_cols.corr()

plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, cmap="coolwarm", annot=True, linewidths=0.5)
```

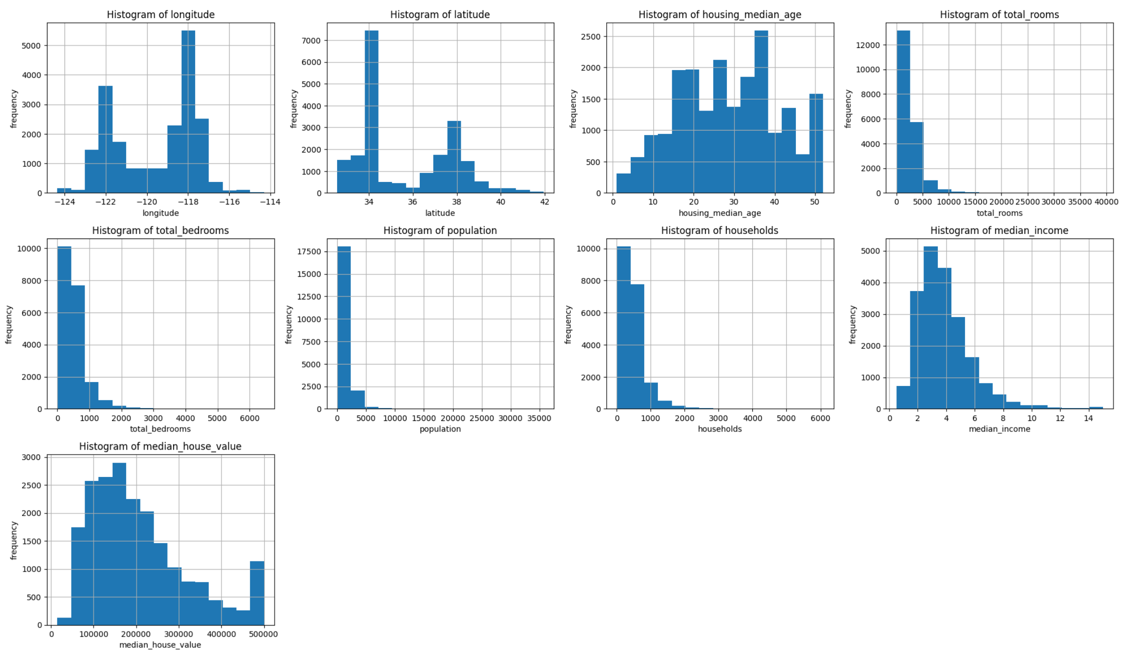
[59]: <Axes: >



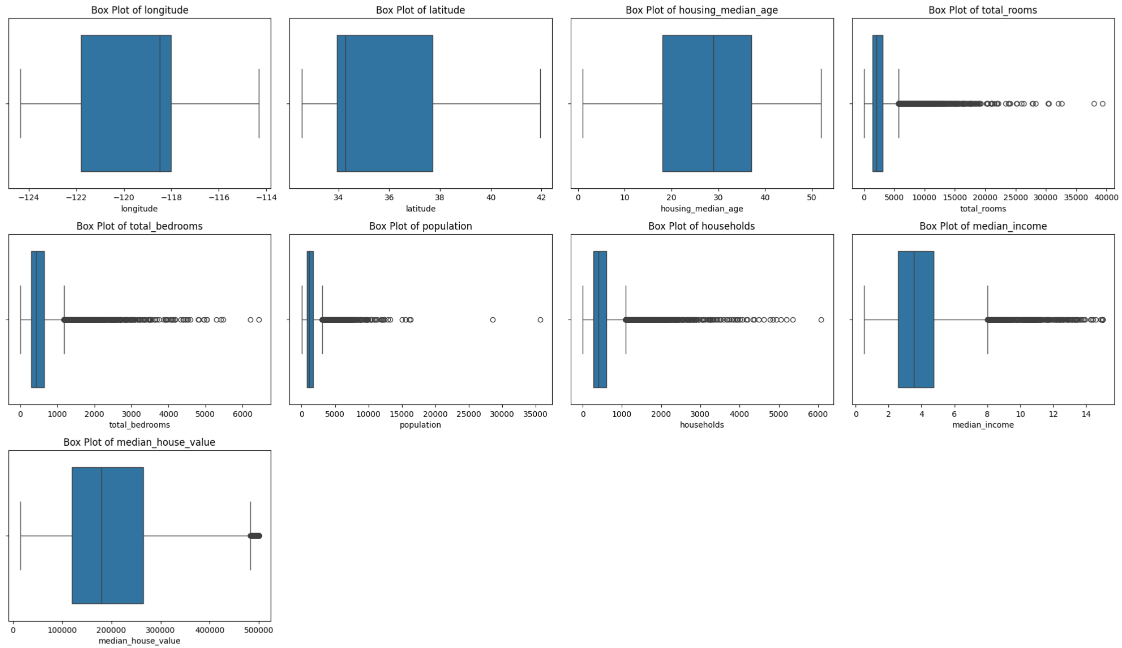
[60]:
```python
target_corr = corr_matrix["median_income"].sort_values(ascending=False)
target_corr[1:6]
```

```
[60]: median_house_value    0.688355
      total_rooms           0.197882
      households            0.013434
      population            0.005087
      total_bedrooms       -0.007723
      Name: median_income, dtype: float64
```

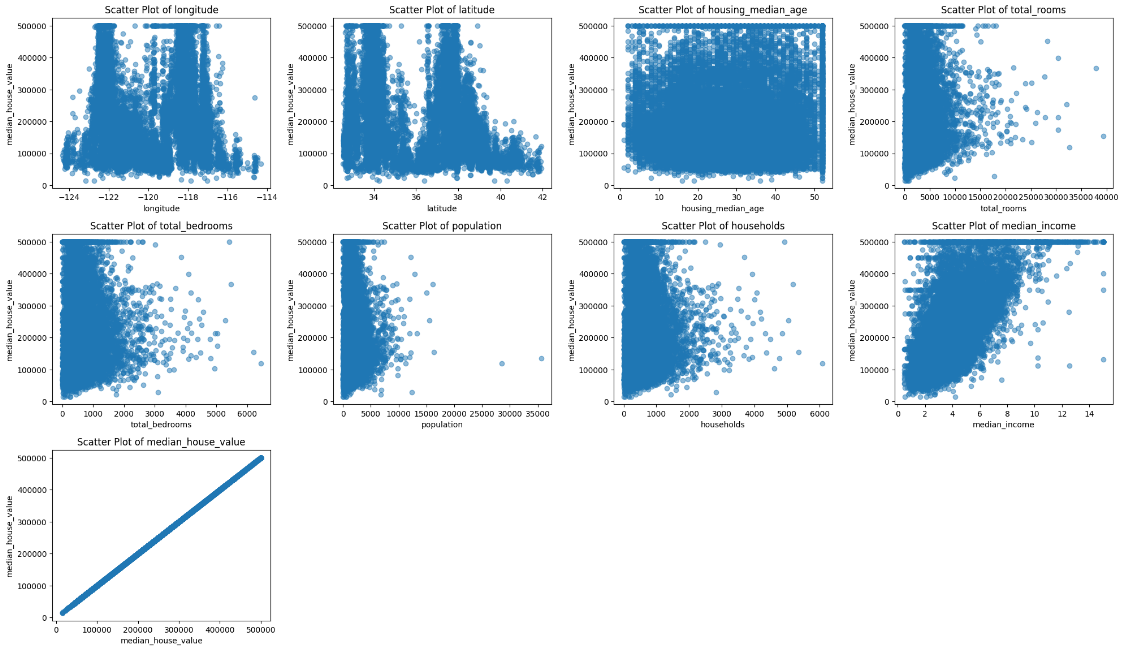
```
[61]: plt.figure(figsize=(20, 15))
      for i, col in enumerate(numerical_cols_names):
          plt.subplot(4, 4, i+1)
          df[col].hist(bins=15)
          plt.xlabel(col)
          plt.ylabel("frequency")
          plt.title(f"Histogram of {col}")
      plt.tight_layout()
      plt.show()
```




```
[62]: plt.figure(figsize=(20, 15))
      for i, col in enumerate(numerical_cols_names):
          plt.subplot(4, 4, i+1)
          sns.boxplot(data=df, x=col)
          plt.xlabel(col)
          plt.title(f"Box Plot of {col}")
      plt.tight_layout()
      plt.show()
```



[63]:
```
plt.figure(figsize=(20, 15))
target = "median_house_value"

for i, col in enumerate(numerical_cols_names):
    plt.subplot(4, 4, i+1)
    plt.scatter(df[col], df[target], alpha=0.5)
    plt.xlabel(col)
    plt.ylabel(target)
    plt.title(f"Scatter Plot of {col}")
plt.tight_layout()
plt.show()
```



[64]:
```
df['income_per_room'] = df['median_income'] / df['total_rooms']
df['average_household_per_population'] = df['households'] / df['population']
print(df['income_per_room'])
print(df['average_household_per_population'])
```

```
0        0.009460
1        0.001169
2        0.004947
3        0.004429
4        0.002364
           …
20635    0.000937
20636    0.003668
20637    0.000754
20638    0.001004
20639    0.000858
Name: income_per_room, Length: 20433, dtype: float64
0        0.391304
1        0.473969
2        0.356855
3        0.392473
4        0.458407
           …
20635    0.390533
20636    0.320225
20637    0.429990
```

```
20638    0.470985
20639    0.382120
Name: average_household_per_population, Length: 20433, dtype: float64
```

[65]:
```
def find_outliers(df, column):
    Q1 = np.percentile(df[column], 25)
    Q3 = np.percentile(df[column], 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    return (df[column] < lower_bound) | (df[column] > upper_bound)
```

[66]:
```
df_outliers_removed = df.copy()
for col in numerical_cols_names:
    outliers = find_outliers(df, col)
    print(col, outliers.sum())
    df_outliers_removed = df_outliers_removed[~outliers]
    # df_outliers_removed.loc[outliers, col] = df_outliers_removed[col].median()
```

```
longitude 0
latitude 0
housing_median_age 0
total_rooms 1290
total_bedrooms 1271
population 1190
households 1210
median_income 670
median_house_value 1064
```

```
C:\Users\Pranesh\AppData\Local\Temp\ipykernel_1776\76152672.py:5: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  df_outliers_removed = df_outliers_removed[~outliers]
C:\Users\Pranesh\AppData\Local\Temp\ipykernel_1776\76152672.py:5: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  df_outliers_removed = df_outliers_removed[~outliers]
C:\Users\Pranesh\AppData\Local\Temp\ipykernel_1776\76152672.py:5: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  df_outliers_removed = df_outliers_removed[~outliers]
C:\Users\Pranesh\AppData\Local\Temp\ipykernel_1776\76152672.py:5: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  df_outliers_removed = df_outliers_removed[~outliers]
C:\Users\Pranesh\AppData\Local\Temp\ipykernel_1776\76152672.py:5: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  df_outliers_removed = df_outliers_removed[~outliers]
```

[67]:
```
for col in numerical_cols_names:
    outliers = find_outliers(df_outliers_removed, col)
```

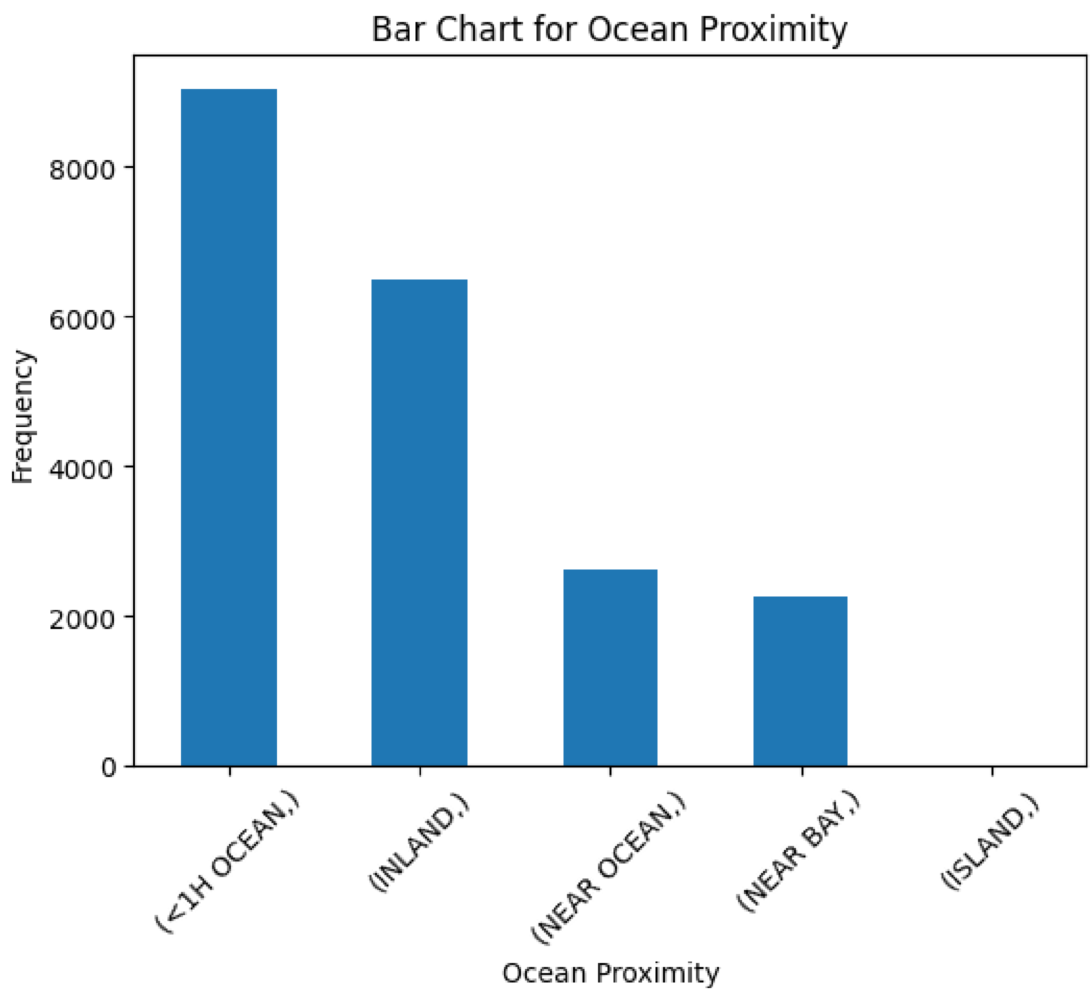
```
    print(col, outliers.sum())
```

```
longitude 0
latitude 0
housing_median_age 0
total_rooms 322
total_bedrooms 273
population 288
households 272
median_income 175
median_house_value 261
```

[68]:
```
df = df_outliers_removed
```

[69]:
```
categorical_cols.value_counts().plot(kind='bar')
plt.xlabel("Ocean Proximity")
plt.ylabel("Frequency")
plt.title("Bar Chart for Ocean Proximity")
plt.xticks(rotation=45)
```

[69]:
```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, '(<1H OCEAN,)'),
  Text(1, 0, '(INLAND,)'),
  Text(2, 0, '(NEAR OCEAN,)'),
  Text(3, 0, '(NEAR BAY,)'),
  Text(4, 0, '(ISLAND,)')])
```



[70]:
```python
for col in numerical_cols_names:
    print(f"Mean of {col}: ", np.mean(df[col], axis=0))
    print(f"Median of {col}: ", np.median(df[col], axis=0))
    print(f"Mode of {col}: ", df[col].mode()[0])
    print("\n")
```

```
Mean of longitude:  -119.60577262819778
Median of longitude:  -118.61
Mode of longitude:  -118.31


Mean of latitude:  35.69786164965011
Median of latitude:  34.31
Mode of latitude:  34.08
```

```
Mean of housing_median_age:  29.489216473557416
Median of housing_median_age:  30.0
Mode of housing_median_age:  52.0


Mean of total_rooms:  2144.865205919468
Median of total_rooms:  1979.0
Mode of total_rooms:  1527.0


Mean of total_bedrooms:  445.4726970287943
Median of total_bedrooms:  411.0
Mode of total_bedrooms:  280.0


Mean of population:  1197.1082941378916
Median of population:  1111.0
Mode of population:  1227.0


Mean of households:  416.80612596076634
Median of households:  387.0
Mode of households:  306.0


Mean of median_income:  3.575592640816795
Median of median_income:  3.3906
Mode of median_income:  3.125


Mean of median_house_value:  187056.1888264311
Median of median_house_value:  170100.0
Mode of median_house_value:  137500.0
```

[71]:
```python
for col in numerical_cols_names:
    df[col] = (df[col] - df[col].min()) / (df[col].max() - df[col].min())

# or
# scaler = StandardScaler()

# df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# df.head()
```

[72]:
```python
df
```

[72]:

```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
2       0.213996  0.564293            1.000000     0.258241        0.160547
3       0.212982  0.564293            1.000000     0.224220        0.198975
4       0.212982  0.564293            1.000000     0.286445        0.237404
5       0.212982  0.564293            1.000000     0.161643        0.180188
6       0.212982  0.563231            1.000000     0.446501        0.415884
…            …         …                   …            …             …
20635   0.330629  0.737513            0.470588     0.293143        0.317677
20636   0.318458  0.738576            0.333333     0.122510        0.126388
20637   0.317444  0.732200            0.313725     0.396968        0.412468
20638   0.307302  0.732200            0.333333     0.327516        0.347566
20639   0.315416  0.725824            0.294118     0.490569        0.524338

       population  households  median_income  median_house_value  \
2        0.157962    0.160846       0.899633            0.721533
3        0.177828    0.199449       0.684719            0.698417
4        0.180070    0.236213       0.445496            0.700343
5        0.131368    0.175551       0.470871            0.545164
6        0.349567    0.470588       0.420587            0.608306
…             …           …             …                  …
20635    0.269785    0.301471       0.141172            0.135062
20636    0.113105    0.102941       0.273837            0.132921
20637    0.321692    0.396140       0.159770            0.165456
20638    0.236463    0.318934       0.182030            0.149188
20639    0.443448    0.485294       0.251444            0.159248

      ocean_proximity  income_per_room  average_household_per_population
2            NEAR BAY         0.004947                          0.356855
3            NEAR BAY         0.004429                          0.392473
4            NEAR BAY         0.002364                          0.458407
5            NEAR BAY         0.004393                          0.467312
6            NEAR BAY         0.001443                          0.469835
…                 …              …                                 …
20635          INLAND         0.000937                          0.390533
20636          INLAND         0.003668                          0.320225
20637          INLAND         0.000754                          0.429990
20638          INLAND         0.001004                          0.470985
20639          INLAND         0.000858                          0.382120

[17434 rows x 12 columns]
```

[73]:

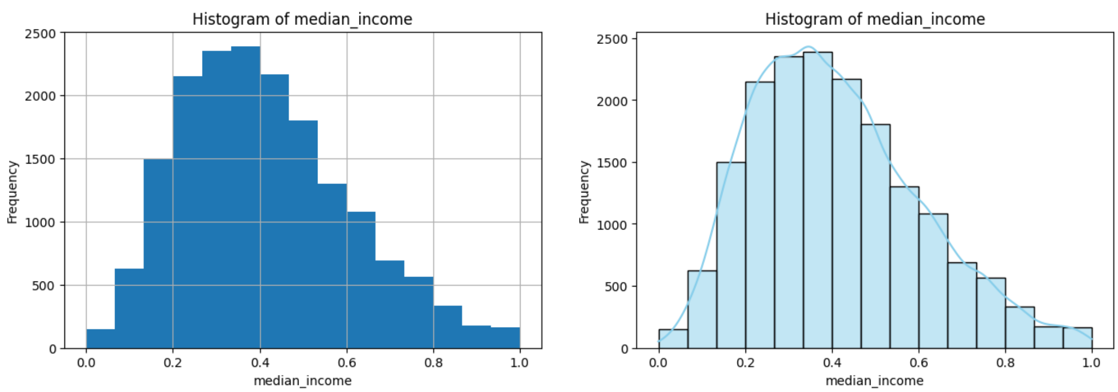
```
target_col = 'median_income'
```

[74]:

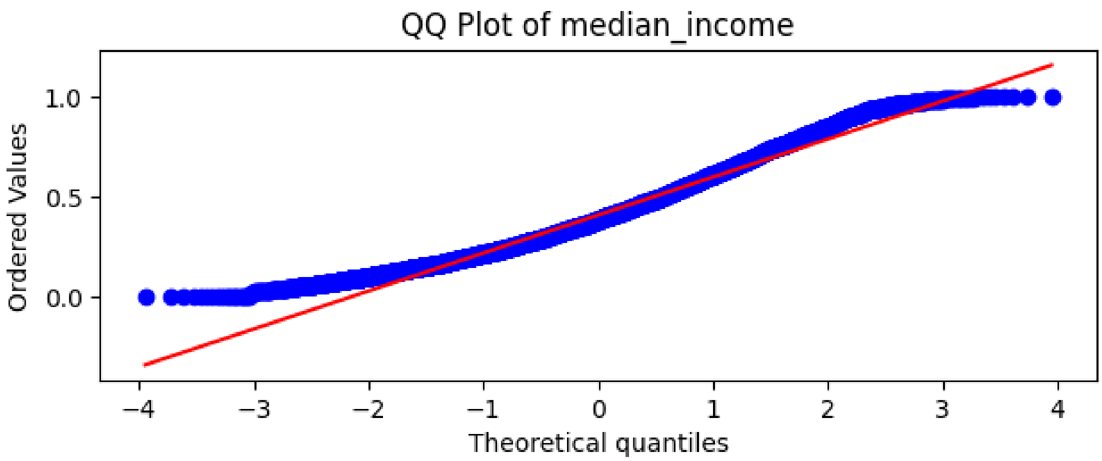
```
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
df[target_col].hist(bins=15)
```

```
plt.xlabel(target_col)
plt.ylabel("Frequency")
plt.title(f'Histogram of {target_col}')

plt.subplot(2, 2, 2)
sns.histplot(df[target_col], bins=15, kde=True, color='skyblue')
plt.xlabel(target_col)
plt.ylabel("Frequency")
plt.title(f'Histogram of {target_col}')
```

[74]: Text(0.5, 1.0, 'Histogram of median_income')



[75]:
```
plt.subplot(2, 1, 1)

stats.probplot(df[target_col], dist="norm", plot=plt)
plt.title(f'QQ Plot of {target_col}')
plt.tight_layout()
plt.show()
```



[76]:
```
shapiro_stat, shapiro_p = stats.shapiro(df[target_col])
```

```
C:\Users\Pranesh\AppData\Local\Programs\Python\Python39\lib\site-
packages\scipy\stats\_morestats.py:1882: UserWarning: p-value may not be
accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

[77]:
```
kstest_stat, kstest_p = stats.kstest(df[target_col], "norm",␣
↪args=(df[target_col].mean(), df[target_col].std()))
```

[78]:
```
print(shapiro_p, kstest_p)
```

```
0.0 5.49876265719009e-48
```

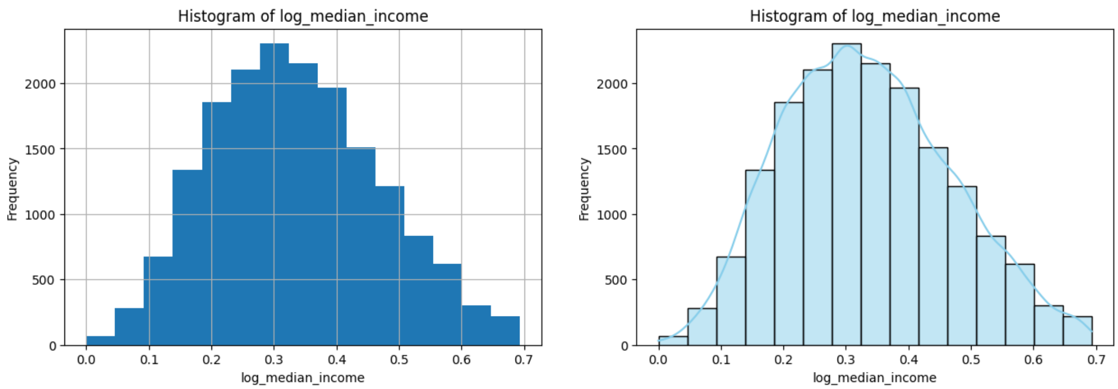
[79]:
```
if shapiro_p > 0.05 and kstest_p > 0.05:
    print("Data appears to be normally distributed")
else:
    print("Data does not appear to be normally distributed")
```

```
Data does not appear to be normally distributed
```

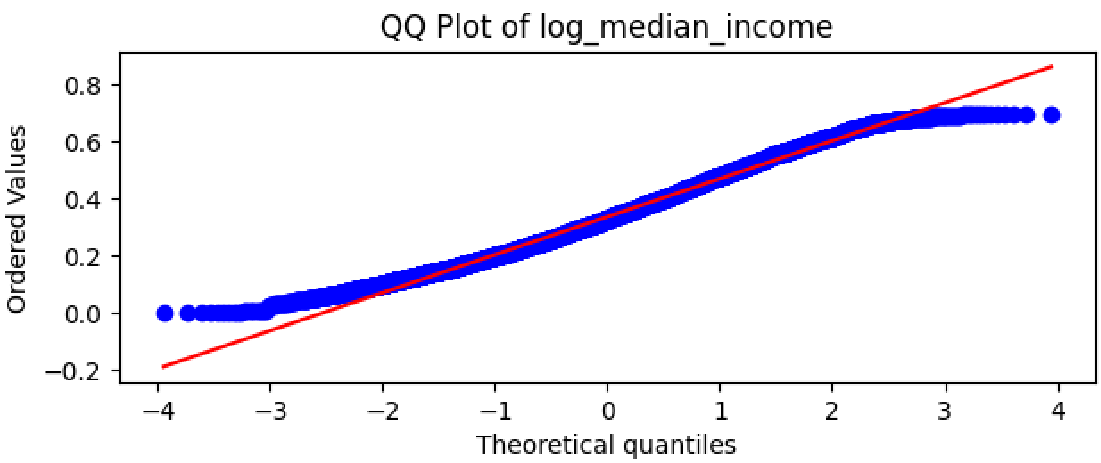
[80]:
```
df[f'log_{target_col}'] = np.log1p(df[target_col])
```

[81]:
```
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
df[f'log_{target_col}'].hist(bins=15)
plt.xlabel(f'log_{target_col}')
plt.ylabel("Frequency")
plt.title(f'Histogram of log_{target_col}')

plt.subplot(2, 2, 2)
sns.histplot(df[f'log_{target_col}'], bins=15, kde=True, color='skyblue')
plt.xlabel(f'log_{target_col}')
plt.ylabel("Frequency")
plt.title(f'Histogram of log_{target_col}')
```

[81]:
```
Text(0.5, 1.0, 'Histogram of log_median_income')
```



[82]:
```
plt.subplot(2, 1, 1)

stats.probplot(df[f'log_{target_col}'], dist="norm", plot=plt)
plt.title(f'QQ Plot of log_{target_col}')
plt.tight_layout()
plt.show()
```



[83]:
```
help(stats.boxcox)
```

```
Help on function boxcox in module scipy.stats._morestats:

boxcox(x, lmbda=None, alpha=None, optimizer=None)
    Return a dataset transformed by a Box-Cox power transformation.

    Parameters
    ----------
    x : ndarray
```

```
    Input array to be transformed.

    If `lmbda` is not None, this is an alias of
    `scipy.special.boxcox`.
    Returns nan if ``x < 0``; returns -inf if ``x == 0 and lmbda < 0``.

    If `lmbda` is None, array must be positive, 1-dimensional, and
    non-constant.

lmbda : scalar, optional
    If `lmbda` is None (default), find the value of `lmbda` that maximizes
    the log-likelihood function and return it as the second output
    argument.

    If `lmbda` is not None, do the transformation for that value.

alpha : float, optional
    If `lmbda` is None and `alpha` is not None (default), return the
    ``100 * (1-alpha)%`` confidence  interval for `lmbda` as the third
    output argument. Must be between 0.0 and 1.0.

    If `lmbda` is not None, `alpha` is ignored.
optimizer : callable, optional
    If `lmbda` is None, `optimizer` is the scalar optimizer used to find
    the value of `lmbda` that minimizes the negative log-likelihood
    function. `optimizer` is a callable that accepts one argument:

    fun : callable
        The objective function, which evaluates the negative
        log-likelihood function at a provided value of `lmbda`

    and returns an object, such as an instance of
    `scipy.optimize.OptimizeResult`, which holds the optimal value of
    `lmbda` in an attribute `x`.

    See the example in `boxcox_normmax` or the documentation of
    `scipy.optimize.minimize_scalar` for more information.

    If `lmbda` is not None, `optimizer` is ignored.

Returns
-------
boxcox : ndarray
    Box-Cox power transformed array.
maxlog : float, optional
    If the `lmbda` parameter is None, the second returned argument is
    the `lmbda` that maximizes the log-likelihood function.
(min_ci, max_ci) : tuple of float, optional
```

```
    If `lmbda` parameter is None and `alpha` is not None, this returned
    tuple of floats represents the minimum and maximum confidence limits
    given `alpha`.

See Also
--------
probplot, boxcox_normplot, boxcox_normmax, boxcox_llf

Notes
-----
The Box-Cox transform is given by::

    y = (x**lmbda - 1) / lmbda,  for lmbda != 0
        log(x),                  for lmbda = 0

`boxcox` requires the input data to be positive.  Sometimes a Box-Cox
transformation provides a shift parameter to achieve this; `boxcox` does
not.  Such a shift parameter is equivalent to adding a positive constant to
`x` before calling `boxcox`.

The confidence limits returned when `alpha` is provided give the interval
where:

.. math::

    llf(\hat{\lambda}) - llf(\lambda) < \frac{1}{2}\chi^2(1 - \alpha, 1),

with ``llf`` the log-likelihood function and :math:`\chi^2` the chi-squared
function.

References
----------
G.E.P. Box and D.R. Cox, "An Analysis of Transformations", Journal of the
Royal Statistical Society B, 26, 211-252 (1964).

Examples
--------
>>> from scipy import stats
>>> import matplotlib.pyplot as plt

We generate some random variates from a non-normal distribution and make a
probability plot for it, to show it is non-normal in the tails:

>>> fig = plt.figure()
>>> ax1 = fig.add_subplot(211)
>>> x = stats.loggamma.rvs(5, size=500) + 5
>>> prob = stats.probplot(x, dist=stats.norm, plot=ax1)
>>> ax1.set_xlabel('')
```

```
>>> ax1.set_title('Probplot against normal distribution')

We now use `boxcox` to transform the data so it's closest to normal:

>>> ax2 = fig.add_subplot(212)
>>> xt, _ = stats.boxcox(x)
>>> prob = stats.probplot(xt, dist=stats.norm, plot=ax2)
>>> ax2.set_title('Probplot after Box-Cox transformation')

>>> plt.show()
```

[84]:
```
df.columns
```

[84]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'income_per_room',
       'average_household_per_population', 'log_median_income'],
      dtype='object')
```

[85]:
```
X = df[['median_income']] # independent (input); double brackets is to convert␣
↪it into a data frame (needs to be 2D in fitting)
Y = df['median_house_value'] # dependent (output)
```

[86]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,␣
↪random_state = 42)
```

[ ]:
```

```

[87]:
```
model = LinearRegression()
model.fit(X_train, Y_train)
```

[87]:
```
LinearRegression()
```

[88]:
```
intercept = model.intercept_
slope = model.coef_[0]
print(f"Regression Eqn: median_house_value = {slope:4f} * median_income +␣
↪{intercept:.4f}")
```

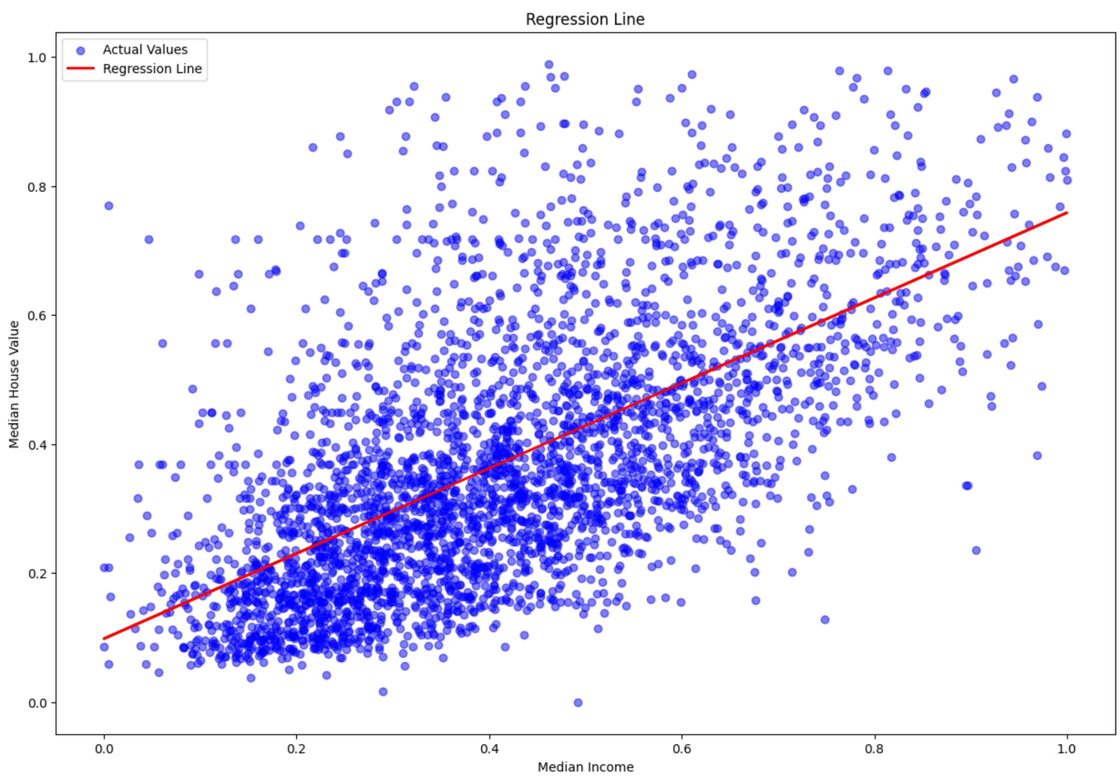
```
Regression Eqn: median_house_value = 0.660493 * median_income + 0.0983
```

[89]:
```
y_pred = model.predict(X_test)
```

[90]:
```
plt.figure(figsize=(15, 10))
plt.scatter(X_test, Y_test, color='blue', label='Actual Values', alpha=0.5) #␣
↪for give x, what is the actual y
```

```
plt.plot(X_test, y_pred, color="red", label="Regression Line", linewidth=2) #␣
↪for given x, what is the predicted y
plt.title('Regression Line')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.legend()
```

[90]: <matplotlib.legend.Legend at 0x246faca68e0>



[91]:
```
mse = mean_squared_error(Y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(Y_test, y_pred)
r_squared = r2_score(Y_test, y_pred)

# Print the metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R-squared: {r_squared:.2f}")
```

```
Mean Squared Error (MSE): 0.02
Root Mean Squared Error (RMSE): 0.16
Mean Absolute Error (MAE): 0.12
```

```
R-squared: 0.38
```

[92]:

```python
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
residuals = Y_test - y_pred
plt.scatter(y_pred, residuals, color="orange", alpha=0.5)
plt.axhline(0, color="red", linestyle="--")
plt.title('Predicted Values vs Residuals')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')

plt.tight_layout()
plt.show()
```