```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import scipy.stats as stats
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

```
df=pd.read_csv("/content/sample_data/california_housing_test.csv")
df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
0	-122.05	37.37	27.0	3885.0	661.0	1537.
1	-118.30	34.26	43.0	1510.0	310.0	809.
2	-117.81	33.78	27.0	3589.0	507.0	1484.
3	-118.36	33.82	28.0	67.0	15.0	49.
4	-119.67	36.33	19.0	1241.0	244.0	850.
...	...	...	...	...	...	.
2995	-119.86	34.42	23.0	1450.0	642.0	1258.
2996	-118.14	34.06	27.0	5257.0	1082.0	3496.
2997	-119.70	36.30	10.0	956.0	201.0	693.
2998	-117.12	34.10	40.0	96.0	14.0	46.
2999	-119.63	34.42	42.0	1765.0	263.0	753.

3000 rows × 9 columns

```
df.info
```

```
pandas.core.frame.DataFrame.info
def info(verbose: bool | None=None, buf: WriteBuffer[str] | None=None, max_cols:
int | None=None, memory_usage: bool | str | None=None, show_counts: bool |
None=None) -> None
```

```
Print a concise summary of a DataFrame.

This method prints information about a DataFrame including
the index dtype and columns, non-null values and memory usage.
```

```
#check for missing values
missing_values=df.isnull().sum()
#calculate the percentage of missing data in each column
missing_percentage=(missing_values/len(df))*100
#display the missing values in column
print("missing_values\n",missing_values)
print("missing_percentage\n",missing_percentage)
```

```
missing_values
 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
dtype: int64
missing_percentage
 longitude             0.0
latitude              0.0
housing_median_age    0.0
total_rooms           0.0
total_bedrooms        0.0
population            0.0
```

```
households            0.0
median_income         0.0
median_house_value    0.0
dtype: float64
```

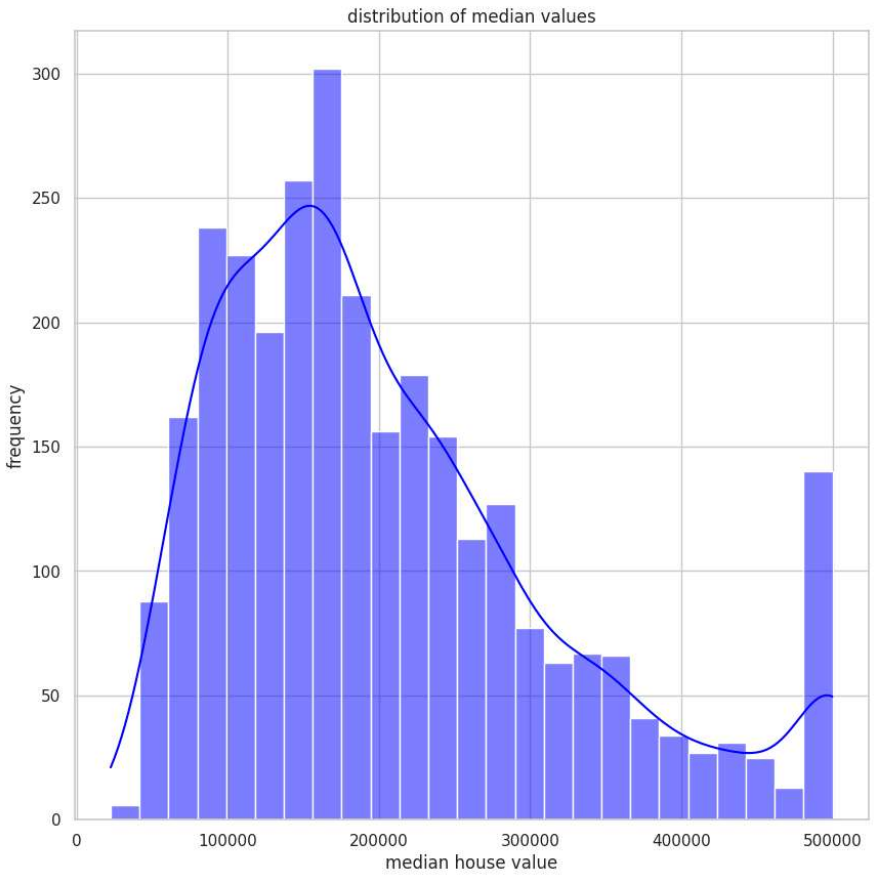
```
#remove rows with missing values
data_cleaned = df.dropna()
#verify that missing values have been removed
print("\n missing value in each rows and column:")
print(data_cleaned.isnull().sum())
```

```
 missing value in each rows and column:
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
dtype: int64
```

```
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	pop
count	3000.000000	3000.00000	3000.000000	3000.000000	3000.000000	300
mean	-119.589200	35.63539	28.845333	2599.578667	529.950667	140
std	1.994936	2.12967	12.555396	2155.593332	415.654368	103
min	-124.180000	32.56000	1.000000	6.000000	2.000000	
25%	-121.810000	33.93000	18.000000	1401.000000	291.000000	78
50%	-118.485000	34.27000	29.000000	2106.000000	437.000000	115
75%	-118.020000	37.69000	37.000000	3129.000000	636.000000	174
max	-114.490000	41.92000	52.000000	30450.000000	5419.000000	1193

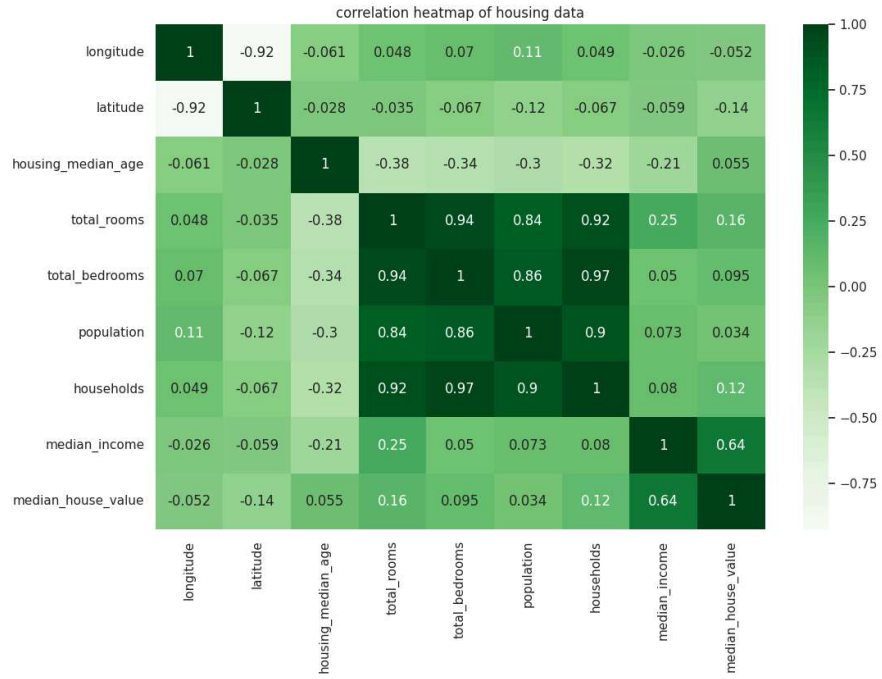
```
sns.set(style="whitegrid")#discrite figures
plt.figure(figsize=(10,10))#figure size
sns.histplot(data_cleaned['median_house_value'],color='blue',kde=True)
plt.title('distribution of median values')
plt.xlabel('median house value')
plt.ylabel('frequency')
plt.show()
```



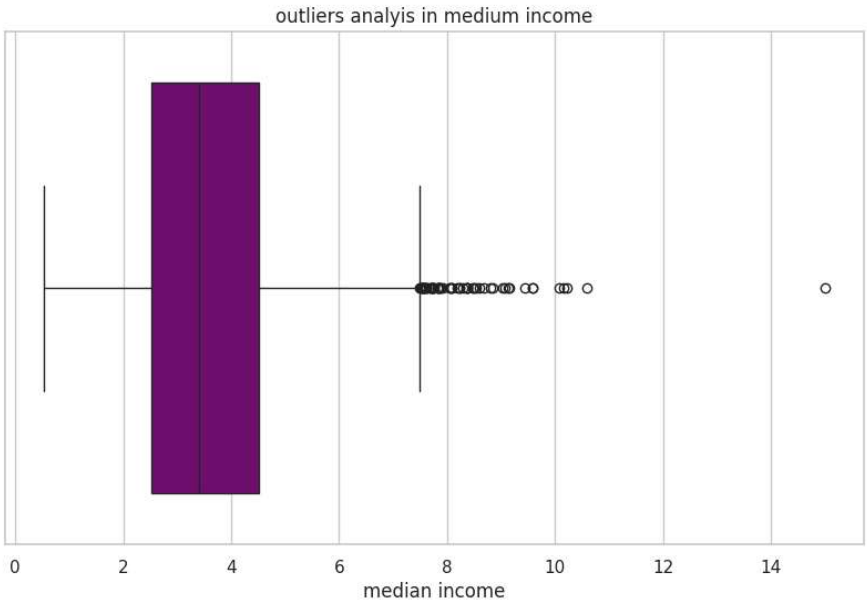

```
#assuming 'data' is your Datframe and 'mediumhouse of teh column of the interset
Q1=data_cleaned['median_house_value'].quantile(0.25)
print(Q1)
Q3=data_cleaned['median_house_value'].quantile(0.75)
print(Q3)
IQR=Q3-Q1

#definethe bounds for the outliers
lower_bound =Q1-1.5*IQR
upper_bound=Q3+1.5*IQR
#remove outliers
data_no_outliers=data_cleaned[(data_cleaned['median_house_value']>=lower_bound)&(dat
#check the shape of the data before and after removal of outliers
print("original data shape:",data_cleaned.shape)
print("new data shape without outliers",data_no_outliers.shape)
```

```
121200.0
263975.0
original data shape: (3000, 9)
new data shape without outliers (2859, 9)
```

```
data=data_no_outliers
plt.figure(figsize=(12,8))
sns.heatmap(data.corr(),annot=True,cmap='Greens')
plt.title('correlation heatmap of housing data')
plt.show()
```




```
#outliers of the mediam income
plt.figure(figsize=(10,6))
sns.boxplot(x=data_no_outliers['median_income'],color='purple')
plt.title("outliers analyis in medium income")
plt.xlabel("median income")
plt.show()
```




```
#unique value count for categorical data
for column in ['ocean_proximity']:
  # Check if the column exists before trying to access it
  if column in data.columns:
      print("Unique values in {column}:",data[column].unique())
  else:
      print(f"Column '{column}' not found in the DataFrame.")
```

```
Column 'ocean_proximity' not found in the DataFrame.
```

```
ocean_proximity_dummies = pd.get_dummies(data['ocean_proximity'],prefix='ocean_proxi
data=pd.concat([data.drop("ocean_proximity",axis=1),ocean_proximity_dummies],axis=1)
ocean_proximity_dummies
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key)
   3652         try:
-> 3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:

 ---------------------------------------- 4 frames ----------------------------------------
pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'ocean_proximity'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key)
   3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:
-> 3655             raise KeyError(key) from err
   3656         except TypeError:
   3657             # If we have a listlike key, _check_indexing_error will raise
```

```
ocean_proximity_dummies = pd.get_dummies(data_no_outliers['ocean_proximity'],prefix=
data=pd.concat([data.drop("ocean_proximity",axis=1),ocean_proximity_dummies],axis=1)
ocean_proximity_dummies
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key)
   3652         try:
-> 3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:

 ---------------------------------- 4 frames ----------------------------------
pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'ocean_proximity'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key)
   3653             return self._engine.get_loc(casted_key)
   3654         except KeyError as err:
-> 3655             raise KeyError(key) from err
   3656         except TypeError:
   3657             # If we have a listlike key, _check_indexing_error will raise
```

```
data.columns
```

```
data = data.drop("ocean_proximity_ISLAND",axis = 1)
data.columns
```

```
data.head()
```

```
#define your features(independent variables)and target(dependent variable)
features = {'longitude','latitude','housing_median_age','total_rooms',
            'total_bedrooms','population','households','median_income','ocean_proxim
            'ocean_proximity_NEAR BAY','ocean_proximity_NEAR OCEAN'}
target={'median_house_value'}
X = data[features]
y = data[target]

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=111)
#check the size of the splits
print('training set size:{x_train.shape[0]} samples')
print('test set size:{x_test.shape[0]} samples')
```

```
X_test_const=sm.add_constant(X_train)
#fit OLS modal ordinary least square
modal_fitted=sm.OLS(y_train,X_test_const).fit()
#printing summary
print(modal_fitted.summary())
```

```
#adding a constant to the test predictors
X_test_const=sm.add_constant(X_test)
#making predictions on the test set
test_predictions=modal_fitted.predict(X_test_const)
```

```
#checking ols assumptions
#scatter plot for observed vs predicated values on test data
plt.scatter(X_test_const,test_predictions,color="forestgreen")
plt.xlabel("Observed Values")
plt.ylabel("Predicated Values")
plt.title("Observed vs Predicted Values on Test Data")
plt.plot(y_test,y_test,color="darkred")
plt.show
```

Random samples

```
#calculate the mean of the residuals
mean_residuals = np.mean(modal_fitted.resid)
```

P Karthikeyan
Mar 27, 2024

Assumption3:Exogeneity

```
#plotting residuals
plt.scatter(modal_fitted.fittedvalues,modal_fitted.resid,color="navyblue")
plt.axhline(y=0,color='red',linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Fitted Values')
plt.show()
```

```
#exogeneity
#calculate the residuals
residuals = modal_fitted.resid
#check the correlation between residuals and each predictor
for column in X_train.columns:
  corr_coefficient = np.corrcoef(X_train[column],residuals)[0,1]
  print(f'Correlation between residuals and {column}:{np.round(corr_coefficient,2)}'
```

```
#homoskedasticity
#plotting the residuals
plt.scatter(modal_fitted.fittedvalues,modal_fitted.resid,color="forestgreen")
plt.axhline(y=0,color='red',linestyle='--')
```