California House Price prediction

Show command palette (⌘/Ctrl+Shift+P)

- Criterion (target) - median_house_value

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('housing.csv')
```

```
data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

Next steps: Generate code with data | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
data.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_ir
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.0
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.8
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.8
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.4
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.5
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.5
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.7
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.0

```
df = data.copy()
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

Show command palette (⌘/Ctrl+Shift+P)

Next steps: Generate code with df | New interactive sheet

- Data Preprocessing

```
df.isna().sum()
```

	0
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

```
df.dropna(inplace=True)
```

```
df.isna().sum()
```

	0
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

```
df.shape
```

```
(20433, 10)
```

- Dropped Missing values

```
df.duplicated().sum()
```

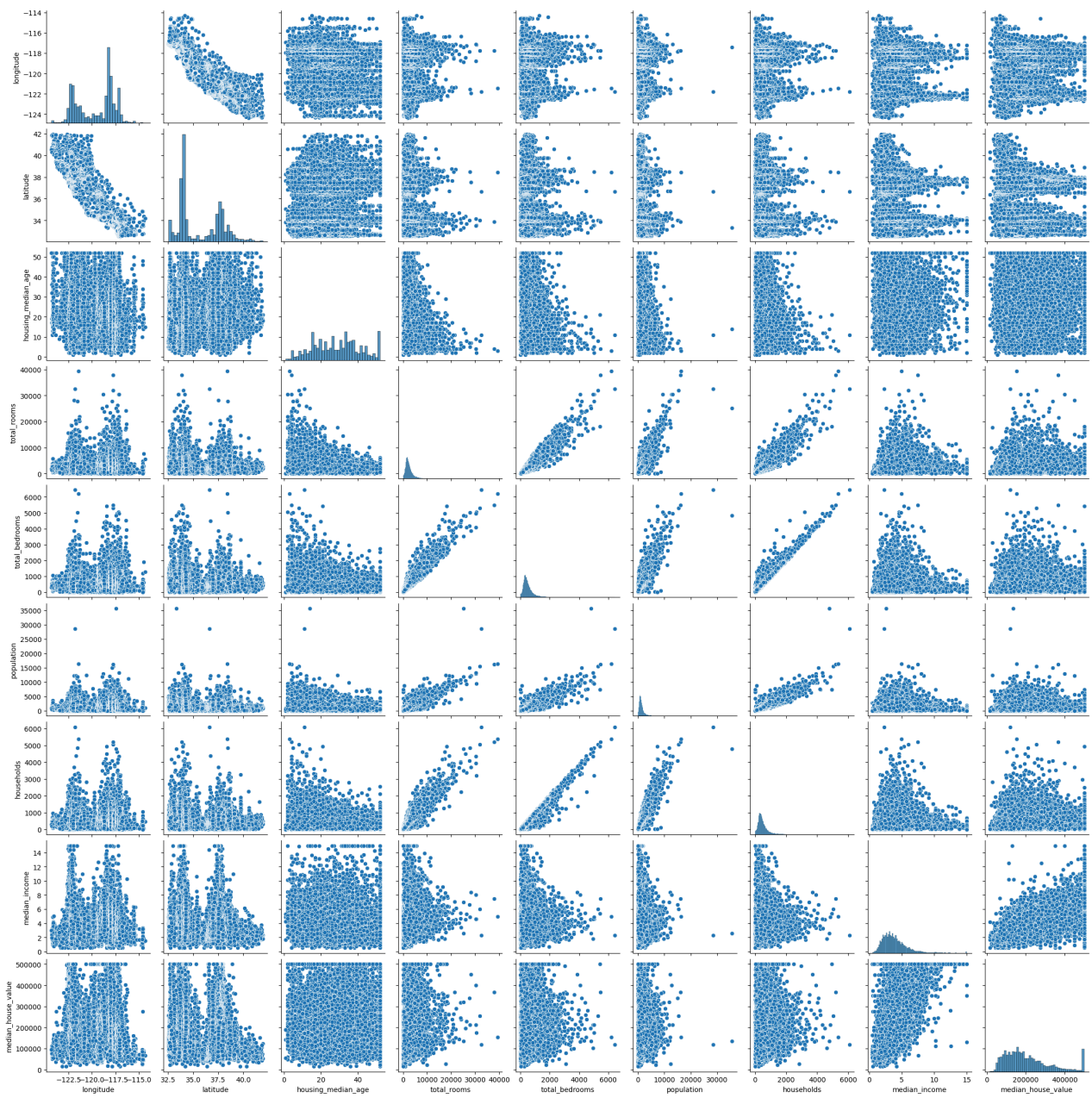
```
np.int64(0)
```

- No Duplicates

```
df_num = df.select_dtypes(include='number')
df_cat = df.select_dtypes(exclude='number')
```

```
sns.pairplot(df_num)
plt.show()
```

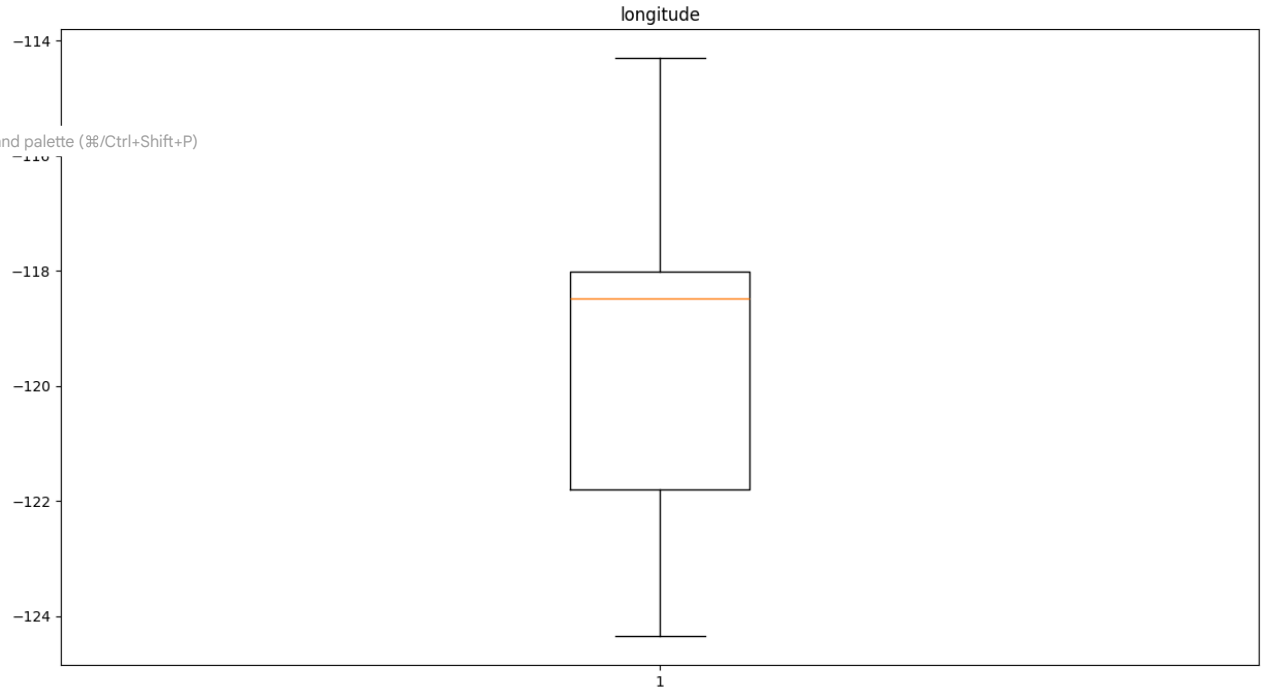
Show command palette (⌘/Ctrl+Shift+P)

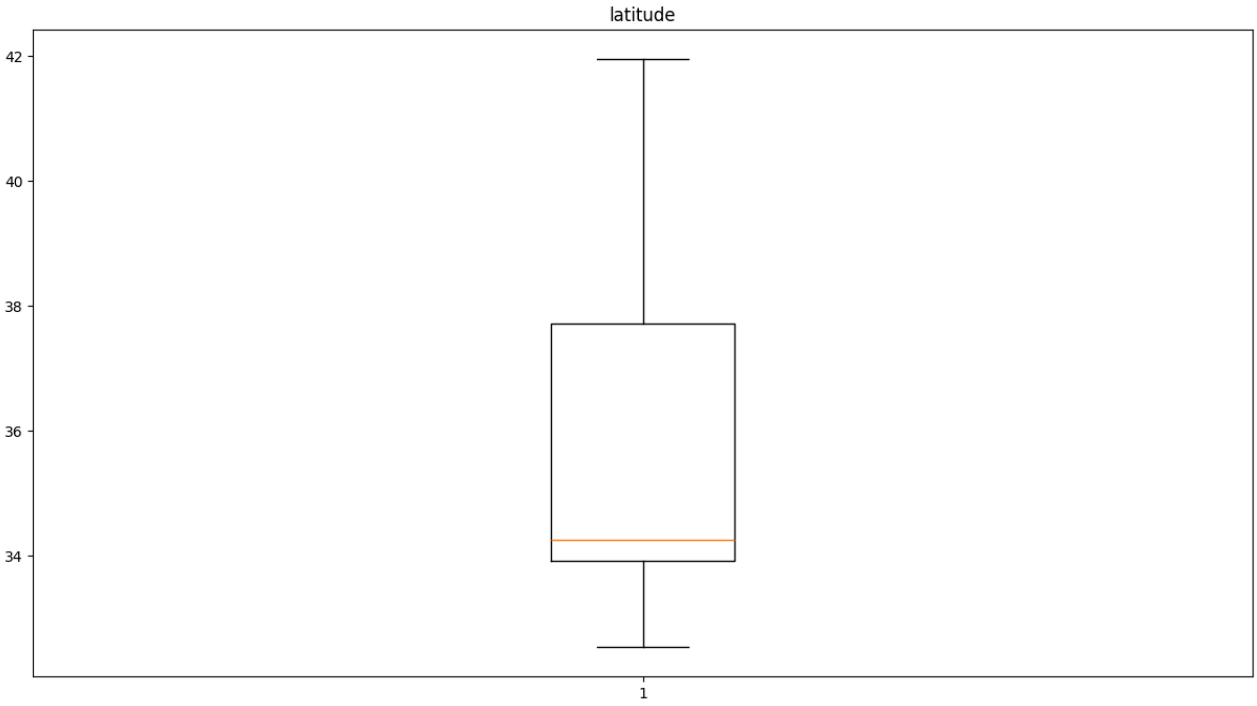


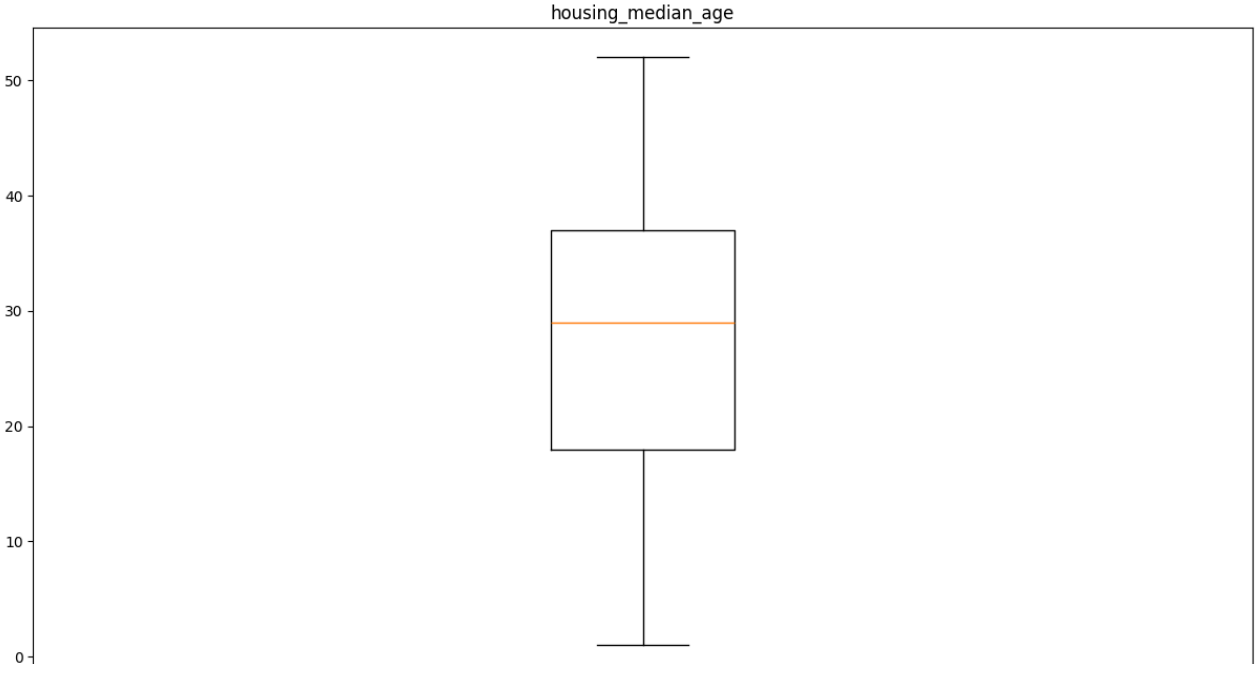

```
for col in df_num.columns:
    plt.figure(figsize=(15, 8))
    plt.boxplot(df_num[col])
    plt.title(col)
    plt.show()
```

Show command palette (⌘/Ctrl+Shift+P)

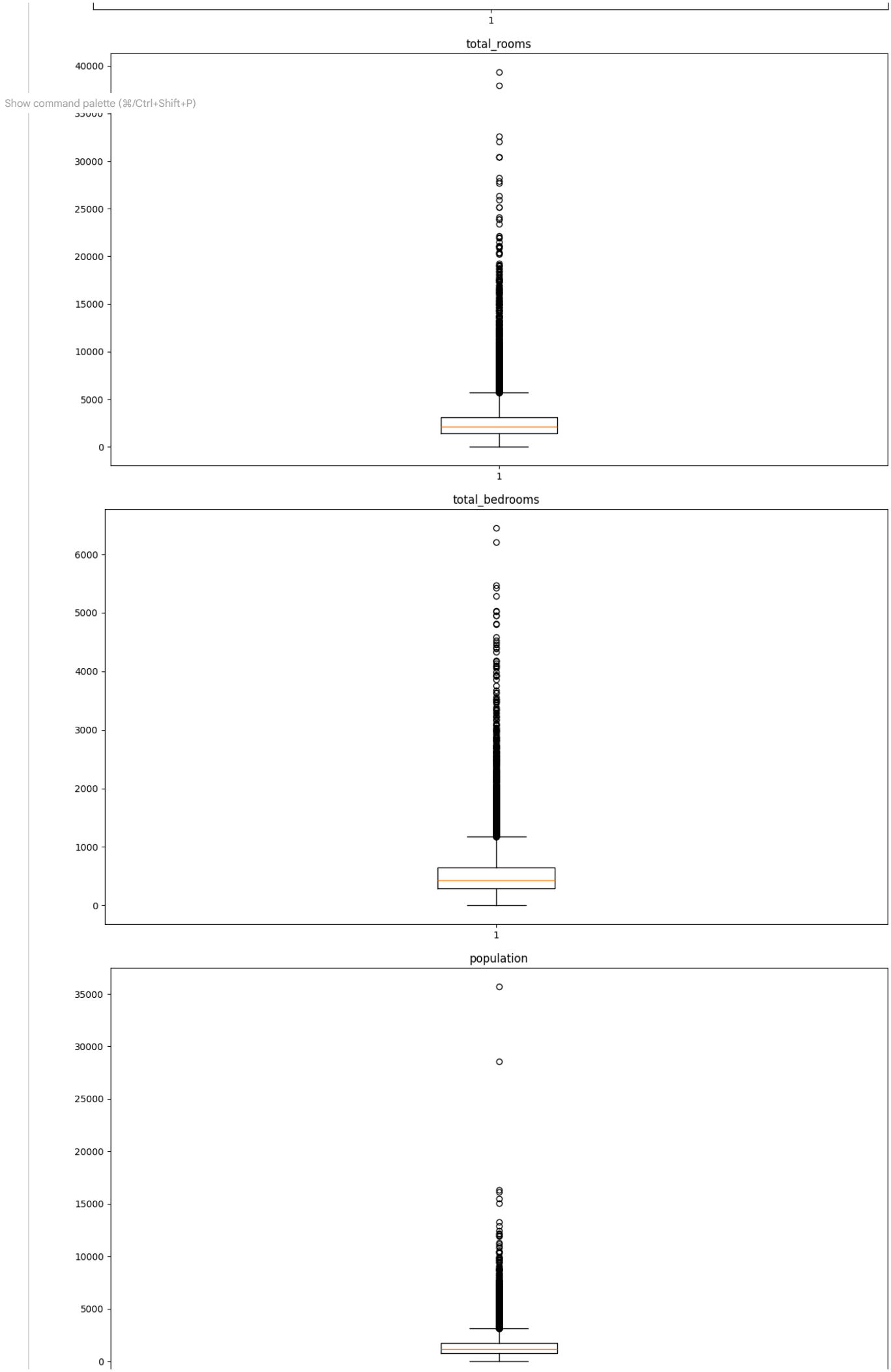
Show command palette (⌘/Ctrl+Shift+P)

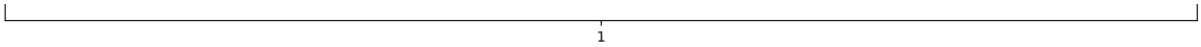


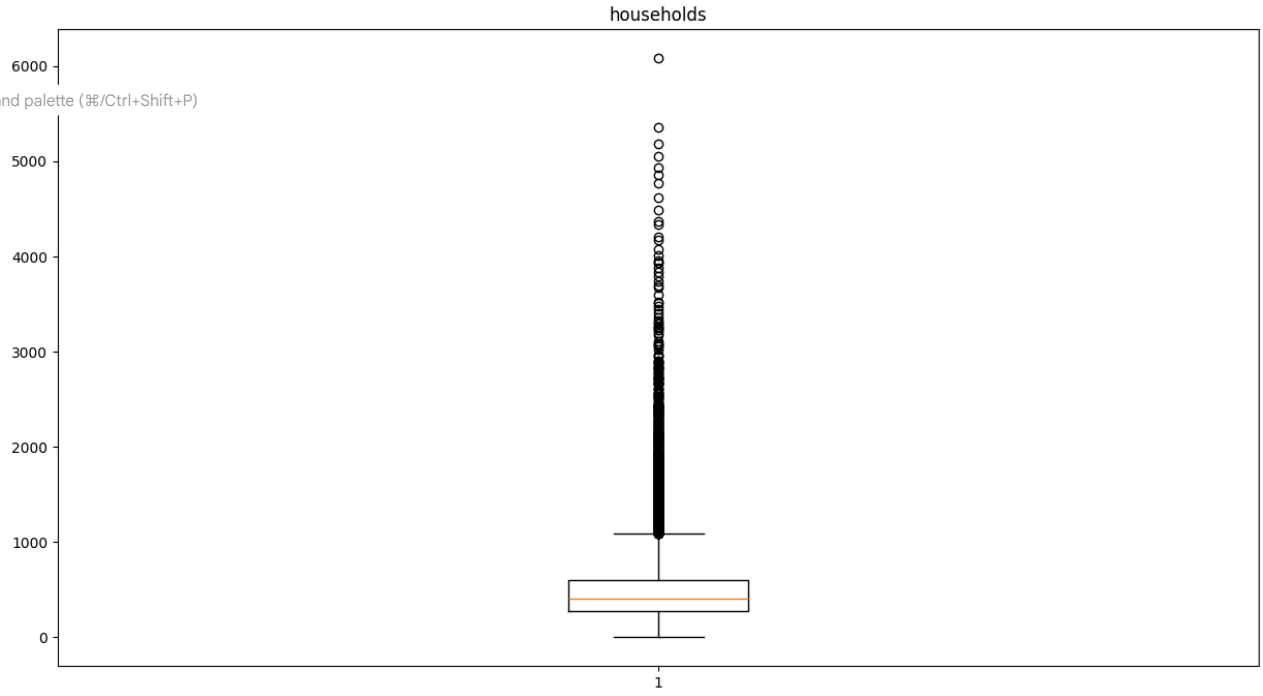




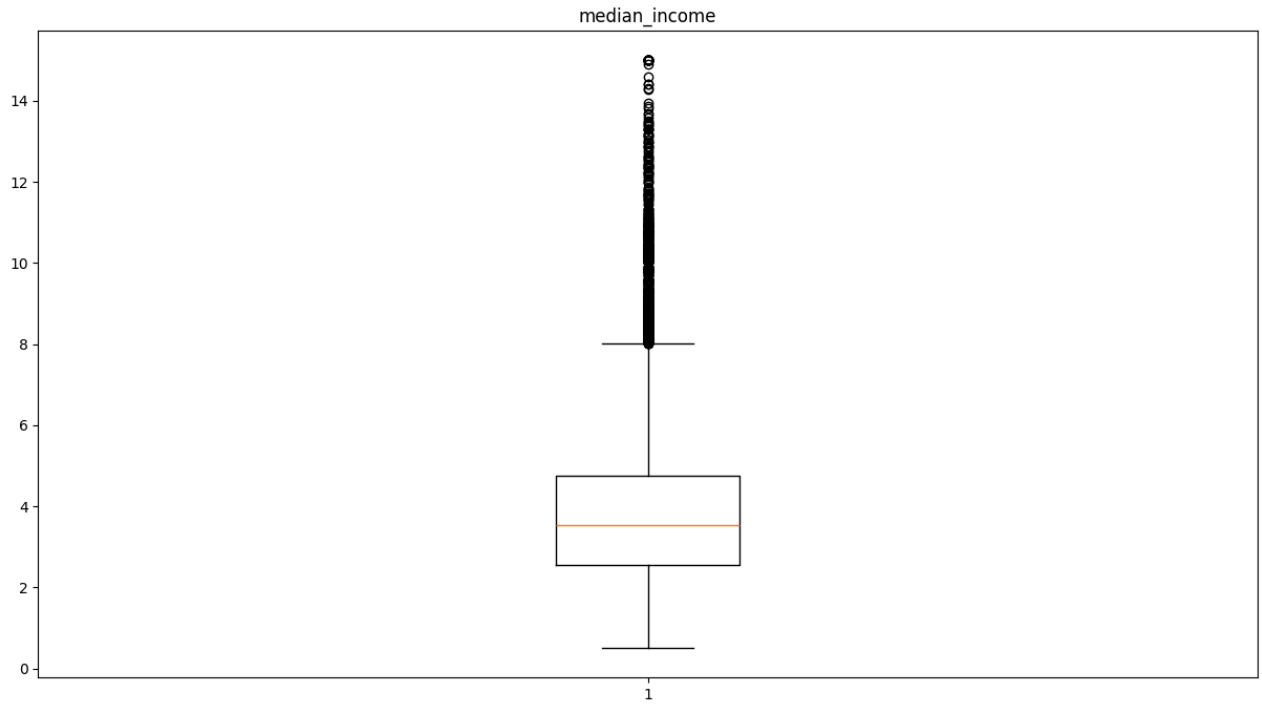
Show command palette (⌘/Ctrl+Shift+P)

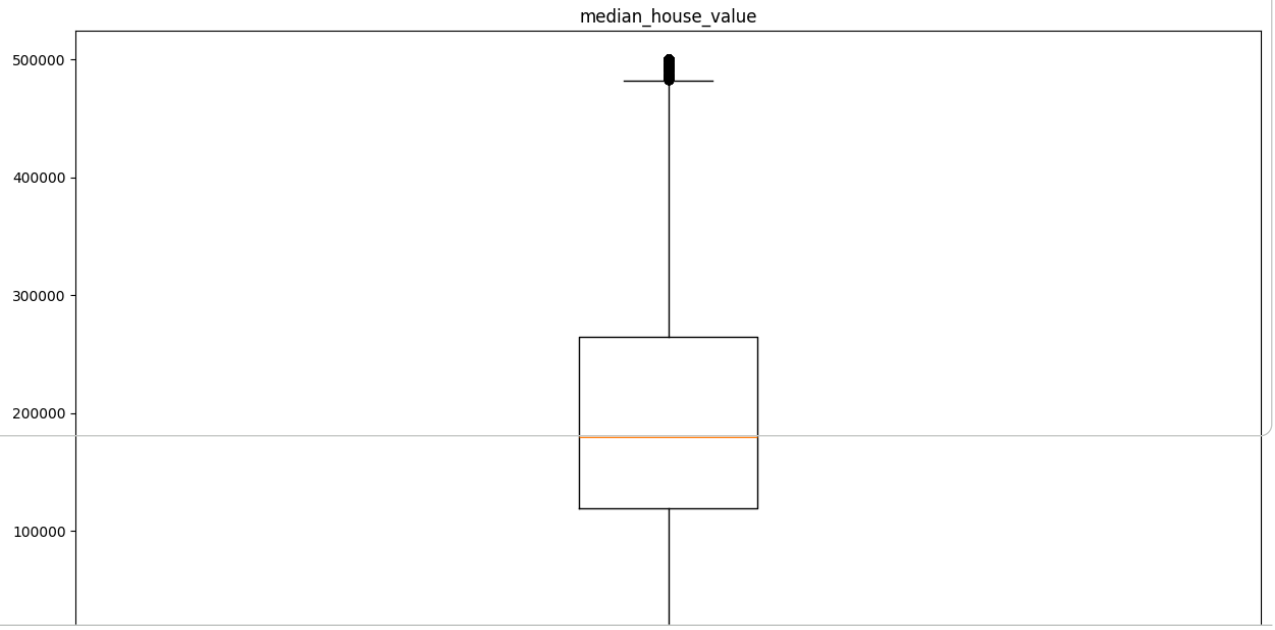






Show command palette (⌘/Ctrl+Shift+P)





- Found Outliers in ['total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_value']

```
for col in df_num.columns:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)

    iqr = q3 - q1

    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr

    df_num[col] = df_num[col].clip(lower=lower, upper=upper)
```

Show command palette (⌘/Ctrl+Shift+P)

- Handled outliers, Capping, Flooring

```
from sklearn.preprocessing import MinMaxScaler

normalizer = MinMaxScaler()
df_normalized = pd.DataFrame(normalizer.fit_transform(df_num), columns=df_num.columns, index=df_num.index)
```

- Normalized all values

```
df_encoded = pd.get_dummies(df_cat['ocean_proximity'], drop_first=True, dtype='int')
```

```
df = pd.concat([df_num, df_encoded], axis=1)
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	med
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.01445	
1	-122.22	37.86	21.0	5682.5	1106.0	2401.0	1090.0	8.01445	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.25740	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.64310	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.84620	

Next steps: Generate code with df | New interactive sheet

- Encoded Categorical Values

Splitting the data

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop('median_house_value', axis=1)
y = df['median_house_value']

print(X.shape)
print(y.shape)
```

```
(20433, 12)
(20433,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()

model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Show command palette (⌘/Ctrl+Shift+P)

```
dict(X_test)
```

Evaluating Linear Regression

```
from sklearn.metrics import r2_score, mean_squared_error,root_mean_squared_error
```

```
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)

print(r2)
print(mse)
print(rmse)
```

```
0.6661146285983737
4396983017.644153
66309.75054729245
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly = PolynomialFeatures(degree=3)

X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

```
model.fit(X_train_poly, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred_poly = model.predict(X_test_poly)
```

Evaluating Polynomial Regression

```
r2 = r2_score(y_test, y_pred_poly)
mse = mean_squared_error(y_test, y_pred_poly)
rmse = root_mean_squared_error(y_test, y_pred)

print(r2)
print(mse)
print(rmse)
```

```
0.7513382724433936
3274660967.0696397
66309.75054729245
```

Ridge Regression

Standardization

- Standardization is an important step before L2 Regularization

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

L2 Regularization

```
from sklearn.linear_model import Ridge
```

```
ridge = Ridge(alpha=1)
ridge.fit (X_train_scaled, y_train)
```

```
Ridge(alpha=1)
```

Show command palette (⌘/Ctrl+Shift+P)

```
y_pred_ridge = ridge.predict(X_test_scaled)
```

Evaluating Ridge Regression

```
r2 = r2_score(y_test, y_pred_ridge)
mse = mean_squared_error(y_test, y_pred_ridge)
rmse = root_mean_squared_error(y_test, y_pred_ridge)

print(r2)
print(mse)
print(rmse)
```

```
0.6666280944643839
4390221114.049788
66258.74367998376
```

- After adjusting alpha value we are getting 66258 at alpha = 1
- Our aim is to get model good performance by adjusting alpha value

Lasso Regression

- Applied Standardization (important)

L1 Regularization

```
from sklearn.linear_model import Lasso
```

```
lasso = Lasso(alpha=3.2)

lasso.fit(X_train_scaled, y_train)
```

```
Lasso
Lasso(alpha=3.2)
```

```
y_pred_lasso = lasso.predict(X_test_scaled)
```

Evaluating Lasso Regression

```
r2 = r2_score(y_test, y_pred_lasso)
mse = mean_squared_error(y_test, y_pred_lasso)
rmse = root_mean_squared_error(y_test, y_pred_lasso)
```

```
print(r2)
print(mse)
print(rmse)
```

```
0.6666259142804316
4390249825.196291
66258.96033893296
```

- After adjusing the alpha value we are getting 66258 at alpha=3.2

Insights

After analysing using different linear regression techniques, we can see that polynomial regression is giving