Comparison between Machine Learning Algoritms and Baysian Inference

Wali Ullah (ERP-09745)

In [3]:
```
#Libraries
#import TemplateMLII as template
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import pydot
from IPython.display import Image
import numpy as np
import pandas as pd
import chart_studio.plotly
import seaborn as sns
%matplotlib inline
sns.set()

import seaborn as sns

sns.set_style(
    style="darkgrid",
    rc={"axes.facecolor": ".9", "grid.color": ".8"}
)
sns.set_palette(palette="deep")
sns_c = sns.color_palette(palette="deep")
```

In [4]:
```
# Import the necessaries libraries
import plotly.offline as pyo
import plotly.graph_objs as go
# Set notebook mode to work in offline
pyo.init_notebook_mode()
```

In [2]:
```
pip install chart_studio
```

```
Collecting chart_studio
  Downloading chart_studio-1.1.0-py3-none-any.whl (64 kB)
     |████████████████████████████████| 64 kB 1.7 MB/s
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from chart_studio) (4.4.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from chart_studio) (1.15.0)
```

```
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from chart_studio) (2.23.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->chart_studio)
(3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from re
quests->chart_studio) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->chart_studio)
(2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->chart_studio) (2.1
0)
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

In [5]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [6]:
```
import sys
import os

py_file_location = "/content/drive/MyDrive/New folder/"
#/content/drive/MyDrive/TemplateML.py
sys.path.append(os.path.abspath(py_file_location))
import TemplateMLII as template
```

In [7]:
```
import numpy as np
import pandas as pd
pd.set_option('max_columns', 105)
#import matplotlib.pyplot as plt


from scipy import stats
from scipy.stats import skew
from math import sqrt

# plotly
#import plotly.plotly as py
import plotly.graph_objs as go
from plotly import tools
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

# sklearn
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, RobustScaler, QuantileTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, HuberRegressor, Lasso, ElasticNet, BayesianRidge
from sklearn.kernel_ridge import KernelRidge
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor

import xgboost as xgb
from xgboost import XGBRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor

from mlxtend.regressor import StackingRegressor
import math
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.metrics import mean_squared_error

from math import sqrt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore")

from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8"))
import chart_studio.plotly
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import pydot
from IPython.display import Image
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
import numpy as np
import pandas as pd
import seaborn as sns
#import seaborn as sb
sns.set_style(
    style="darkgrid",
    rc={"axes.facecolor": ".9", "grid.color": ".8"}
)
sns.set_palette(palette="deep")
sns_c = sns.color_palette(palette="deep")

import arviz as az
import patsy
import pymc3 as pm
from pymc3 import glm
from sklearn.model_selection import train_test_split

plt.rcParams["figure.figsize"] = [7, 6]
plt.rcParams["figure.dpi"] = 100

from sklearn.linear_model import LinearRegression
#import performance scores
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from math import sqrt
import seaborn as sns
import math
from math import *
#Stats Libraries
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.api as sm
import theano

# import scaling
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
#Plot libries
# Import the necessaries libraries
import plotly.offline as pyo
import plotly.graph_objs as go
# Set notebook mode to work in offline
pyo.init_notebook_mode()

SEED=42
```

Helping Funtions

In [8]:
```
###############################################################################
###############################################################################################
###############################################################################################
#Helping Function
###############################################################################################
# Train Test Split for ML Algorithms


def traintestsplit(df,split,random=SEED, LABEL_COL=''):
    #make a copy of the label column and store in y
    y = df[LABEL_COL].copy()

    #now delete the original
    X = df.drop(LABEL_COL,axis=1)

    #manual split
    trainX, testX, trainY, testY= train_test_split(X, y, test_size=split, random_state=random)
    return trainX, testX, trainY, testY

#####################################################
#Train Test Split for Baysian Framework
def train_test(formula,df):
    # define standard scaler
    scaler = StandardScaler()

    # Define model formula.
    formula = formula
    # Create features.
    y, X = patsy.dmatrices(formula_like=formula, data=df)
    y = np.asarray(y).flatten()
    labels = X.design_info.column_names
    x = np.asarray(X)
    x = scaler.fit_transform(X)
    #do a train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=SEED)
    return X_train, X_test, y_train, y_test, labels
###############################################################################################
#Regression Validation Score Function
def validationmetrics_reg(y_test,y_pred,X_test):
```

```
    # R-squared
    r2 = r2_score(y_test,y_pred, multioutput='variance_weighted')

    # Adjusted R-squared
    r2_adjusted = 1-(1-r2)*(X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)
    #MAE
    mae = mean_absolute_error(y_test,y_pred)
    # MSE
    mse = mean_squared_error(y_test,y_pred)
    #RMSE
    rmse = math.sqrt(mse)
    return r2, r2_adjusted,mae,mse,rmse

##########################################################################################################
# Get Best Parameters and Score in Grid Search
def get_best_score(grid):

    best_score = np.sqrt(-grid.best_score_)
    print(best_score)
    print(grid.best_params_)
    print(grid.best_estimator_)

    return best_score

##########################################################################################################
```

Part 2. Pipeline approach

sklearn.pipeline.Pipeline

Pipeline of transforms with a final estimator. Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be 'transforms', that is, they must implement fit and transform methods. The final estimator only needs to implement fit. The transformers in the pipeline can be cached using memory argument.

The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters.

2.1 Preprocessing pipeline Function

In [9]:
```
# Function to seperate Numerical and catagorical Columns

def Num_Cat_Col(df,LABEL_COL):
    data1=df.drop(LABEL_COL,axis=1)
```

```
    numerical_features=data1.select_dtypes(exclude=['object']).columns.tolist()
    print('numerical_features')
    print(numerical_features)
    categorical_features = data1.select_dtypes(include=['object']).columns.tolist()
    print('categorical_features')
    print(categorical_features)
    return numerical_features,categorical_features
#Funtion for defining steps for Numerical and Catagorical Columns

def pipline_process(df,LABEL_COL):
    data1=df.drop(LABEL_COL,axis=1)
    numerical_features   =data1.select_dtypes(exclude=['object']).columns.tolist()
    categorical_features = data1.select_dtypes(include=['object']).columns.tolist()
    #numerical_features,categorical_features=Num_Cat_Col(df,label_col)
    numerical_transformer = Pipeline(steps=[
        #('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())])

    categorical_transformer = Pipeline(steps=[
        #('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_transformer,   numerical_features),
            ('cat', categorical_transformer, categorical_features)])
    return preprocessor
```

2.2. Append regressors to pipeline

We consider Pipelines with default model parameters and run a 5 fold cross validation for each pipeline/model. The ML models are:

1. LinearRegression
2. Ridge Regression
3. Huber Regressor
4. Lasso Regression
5. Elastic Net Regression
6. Bayes Ridge Regression
7. Random Forest Regression
8. Decision Tree Regression
9. SVR Regression

10. GBR Regression
11. XGB Regression
12. LGBM Regression
13. ADA Regression

For this we create loop over list of pipeline and calculate the mean, std and min score (=error) for every model.
By this we get a first estimate for the different regression pipelines. We fit the the data (features X and target y) using the default model parameters.

In [10]:
```
##########################################################################################################
##########################################################################################
# ML Models to Predict (Using the Cross-Validation to choose)

##########################################################################################
#Append regression Models to Pipeline
#ML Models to Pipeline

def pipe_ML_models(df, LABEL_COL):

    preprocessor=pipline_process(df,LABEL_COL)
    # LinearRegression
    pipe_Linear = Pipeline(steps   = [('preprocessor', preprocessor),
                                      ('Linear', LinearRegression()) ])
# Ridge
    pipe_Ridge = Pipeline(
        steps  = [('preprocessor', preprocessor),
                  ('Ridge', Ridge(random_state=5)) ])
# Huber
    pipe_Huber = Pipeline(
        steps  = [('preprocessor', preprocessor),
                  ('Huber', HuberRegressor()) ])
# Lasso
    pipe_Lasso = Pipeline(
        steps  = [ ('preprocessor', preprocessor),
                  ('Lasso', Lasso(random_state=5)) ])
# ElasticNet
    pipe_ElaNet = Pipeline(
        steps   = [ ('preprocessor', preprocessor),
                   ('ElaNet', ElasticNet(random_state=5)) ])
# BayesianRidge
    pipe_BayesRidge = Pipeline(
        steps   = [ ('preprocessor', preprocessor),
```

```
                ('BayesRidge', BayesianRidge(n_iter=500, compute_score=True)) ])

    pipe_RandomForestReg=Pipeline(
        steps   = [ ('preprocessor', preprocessor),
                    ('RandomForestReg', RandomForestRegressor(n_estimators=100)) ])

    pipe_DecisionTreeReg=Pipeline(
        steps   = [ ('preprocessor', preprocessor),
                    ('DecisionTreeReg', DecisionTreeRegressor()) ])

    pipe_SVReg=Pipeline(
        steps   = [ ('preprocessor', preprocessor),
                    ('SVReg', SVR(kernel="rbf")) ])

# GradientBoostingRegressor
    pipe_GBR  = Pipeline(
        steps = [ ('preprocessor', preprocessor),
                  ('GBR', GradientBoostingRegressor(random_state=5 )) ])

# XGBRegressor
    pipe_XGB  = Pipeline(
        steps = [ ('preprocessor', preprocessor),
                  ('XGB', XGBRegressor(objective='reg:squarederror',eval_metric='rmse',# metric='rmse',
                                       random_state=5, nthread = -1)) ])
# LGBM
    pipe_LGBM = Pipeline(
        steps= [('preprocessor', preprocessor),
                ('LGBM', LGBMRegressor(objective='regression', metric='rmse',
                                       random_state=5)) ])
# AdaBoostRegressor
    pipe_ADA = Pipeline(
        steps= [('preprocessor', preprocessor),
                ('ADA', AdaBoostRegressor(DecisionTreeRegressor(),
                                          random_state=5, loss='exponential')) ])
    list_pipelines = [pipe_Linear, pipe_Ridge, pipe_Huber, pipe_Lasso, pipe_ElaNet,
                  pipe_BayesRidge,pipe_RandomForestReg,pipe_DecisionTreeReg,pipe_SVReg,
                  pipe_GBR, pipe_XGB, pipe_LGBM, pipe_ADA]
    return pipe_Linear,pipe_Ridge,pipe_Huber,pipe_Lasso,pipe_ElaNet,pipe_BayesRidge,pipe_RandomForestReg,pipe_DecisionTre


########################################################################
# ML model with Crossvalidation to fit and predict the outcome
#We run a 5 fold cross validation for each pipeline/model. Function to estimate
#different regression pipelines. We fit the the data (features X and target y)
#using the default model parameters.
```

```
##############################################################################

def runML_models_CV(df,split,random, LABEL_COL):
    data1 = []
    data2 = []
    data3 = []
    X_train, X_test, y_train, y_test=traintestsplit(df,split,random, LABEL_COL)
    pipe_Linear,pipe_Ridge,pipe_Huber,pipe_Lasso,pipe_ElaNet,pipe_BayesRidge,pipe_RandomForestReg,pipe_DecisionTreeReg,pi
    list_pipelines=list_pipelines
    for pipe in list_pipelines :
        pipe.fit(X_train, y_train)
        scores = cross_val_score(pipe, X_train, y_train,scoring='neg_mean_squared_error', cv=5)


        scores = np.sqrt(-scores)
        mean=np.mean(scores)
        sd=np.std(scores)
        min_sc=np.min(scores)
        data1.append(({'mean_rmse':mean,
                       'std': sd,
                       'min_rmse': min_sc
                      }))

        #fitting training

        pred_yfit = pipe.predict(X_train)
        r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_train,pred_yfit,X_train)

        data2.append(({'R-Squared':r2,
                       'Adj-R Squared': r2_adjusted,
                       'MAE': mae,
                       'MSE':mse,
                       'RMSE': rmse
                      }))

    #prediction test
        pred_y = pipe.predict(X_test)
        r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_test,pred_y,X_test)
        data3.append(({'R-Squared':r2,
                       'Adj-R Squared': r2_adjusted,
                       'MAE': mae,
                       'MSE':mse,
                       'RMSE': rmse
                      }))
```

```
    results1 = pd.DataFrame(data=data1, columns=['mean_rmse','std','min_rmse'],
                            index=['Linear','Ridge', 'Huber','Lasso','ElaNet','BayesRidge','RandomForestReg',
                                   'DecisionTreeReg','SVReg','GBR', 'XGB','LGBM','ADA'])
    results2 = pd.DataFrame(data=data2, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                            index=['Linear','Ridge', 'Huber','Lasso','ElaNet','BayesRidge','RandomForestReg',
                                  'DecisionTreeReg','SVReg','GBR', 'XGB','LGBM','ADA'])

    results3 = pd.DataFrame(data=data3, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                            index=['Linear','Ridge', 'Huber','Lasso','ElaNet','BayesRidge','RandomForestReg',
                                   'DecisionTreeReg','SVReg','GBR', 'XGB','LGBM','ADA'])

    print('CV Results')
    display(results1)
    print('Training Data Fit Results')
    display(results2)
    print('Test Data Prediction Results')
    display(results3)
    return results1,results2,results3
```

Part 3: GridSearch CV For ML Models

GridSearch and Hyper Parameters tuning with Preprocessing function. Here we consider the gridsearch approach to select the optimal hyper parameters to tune. The ML models are:

1. LinearRegression
2. Ridge Regression
3. Huber Regressor
4. Lasso Regression
5. Elastic Net Regression
6. Bayes Ridge Regression
7. Random Forest Regression
8. Decision Tree Regression
9. SVR Regression
10. GBR Regression
11. XGB Regression
12. LGBM Regression
13. ADA Regression

For this we create loops over the list of pipeline and gridsearch to select best parameters. The optimal choice is used to calculate the mean, std and min score (=error) for every model.

By this we get a first the optimal (best) estimate for the different regression pipelines. We fit the the data (features X and target y) using the optimal model parameters. The same optimal parameters are used to compute the prediction.

In [11]:
```
##########################################################################################################
##########################################################################################################
# Tuning parameters
# ML Models to Predict (Using the Grid_Search with Cross-Validation)

##########################################################################################################################
#Defiining Grid Search for all Linear Models and Ensembles Models

def gridsearch_ML_models(df,LABEL_COL):
    list_scalers = [StandardScaler(),
                    RobustScaler(),
                    QuantileTransformer(output_distribution='normal')]
    preprocessor=pipline_process(df,LABEL_COL)
    pipe_Linear,pipe_Ridge,pipe_Huber,pipe_Lasso,pipe_ElaNet,pipe_BayesRidge,pipe_RandomForestReg,pipe_DecisionTreeReg,pi
    #Linear Regression
    parameters_Linear = { 'preprocessor__num__scaler': list_scalers,
                         'Linear__fit_intercept':  [True,False],
                         'Linear__normalize':  [True,False] }

    gscv_Linear = GridSearchCV(pipe_Linear, parameters_Linear, n_jobs=-1,
                               scoring='neg_mean_squared_error', verbose=0, cv=5)

#Ridge Regression
    parameters_Ridge = { 'preprocessor__num__scaler': list_scalers,
                        'Ridge__alpha': [7,8,9],
                        'Ridge__fit_intercept':  [True,False],
                        'Ridge__normalize':  [True,False] }

    gscv_Ridge = GridSearchCV(pipe_Ridge, parameters_Ridge, n_jobs=-1,
                              scoring='neg_mean_squared_error', verbose=0, cv=5)
#Huber Regression
    parameters_Huber = { 'preprocessor__num__scaler': list_scalers,
                        'Huber__epsilon': [1.3, 1.35, 1.4],
                        'Huber__max_iter': [150, 200, 250],
                        'Huber__alpha': [0.0005, 0.001, 0.002],
                        'Huber__fit_intercept':  [True], }

    gscv_Huber = GridSearchCV(pipe_Huber, parameters_Huber, n_jobs=-1,
```

```
                              scoring='neg_mean_squared_error', verbose=1, cv=5)


# #### Lasso


    parameters_Lasso = { 'preprocessor__num__scaler': list_scalers,
                         'Lasso__alpha': [0.0005, 0.001],
                         'Lasso__fit_intercept':  [True],
                         'Lasso__normalize':  [True,False] }

    gscv_Lasso = GridSearchCV(pipe_Lasso, parameters_Lasso, n_jobs=-1,
                              scoring='neg_mean_squared_error', verbose=1, cv=5)

# **ElasticNet**

    parameters_ElaNet = { 'ElaNet__alpha': [0.0005, 0.001],
                          'ElaNet__l1_ratio':  [0.85, 0.9],
                          'ElaNet__normalize':  [True,False] }

    gscv_ElaNet = GridSearchCV(pipe_ElaNet, parameters_ElaNet, n_jobs=-1,
                               scoring='neg_mean_squared_error', verbose=1, cv=5)



# ** Random Forest Regression
    parameters_RandomForestReg={ 'RandomForestReg__n_estimators':  [200,300,500],
                      'RandomForestReg__max_depth':  [4,7,10],
                      'RandomForestReg__min_samples_leaf':  [5,6,8],
                      'RandomForestReg__max_features':  ["auto",0.5,0.7]}

    gscv_RandomForestReg = GridSearchCV(pipe_RandomForestReg, parameters_RandomForestReg, n_jobs=-1,
                           scoring='neg_mean_squared_error', verbose=1, cv=5)

 # Decision Tree
#
    parameters_DecisionTreeReg={ #'DecisionTreeReg__n_estimators':  [200,300,500],
                      'DecisionTreeReg__max_depth':  [4,7,10],
                      'DecisionTreeReg__min_samples_leaf':  [5,6,8],
                      'DecisionTreeReg__max_features':  ["auto",0.5,0.7]}

    gscv_DecisionTreeReg = GridSearchCV(pipe_DecisionTreeReg, parameters_DecisionTreeReg, n_jobs=-1,
                           scoring='neg_mean_squared_error', verbose=1, cv=5)

# SVR
```

```
    parameters_SVReg={ 'SVReg__kernel':  ['linear', 'rbf'],
                      'SVReg__C':  [8,9, 10]}

    gscv_SVReg = GridSearchCV(pipe_SVReg, parameters_SVReg, n_jobs=-1,
                              scoring='neg_mean_squared_error', verbose=1, cv=5)


# **GradientBoostingRegressor**

    parameters_GBR = { 'GBR__n_estimators':  [400],
                      'GBR__max_depth':  [3,4],
                      'GBR__min_samples_leaf':  [5,6],
                      'GBR__max_features':  ["auto",0.5,0.7]}

    gscv_GBR = GridSearchCV(pipe_GBR, parameters_GBR, n_jobs=-1,
                              scoring='neg_mean_squared_error', verbose=1, cv=5)

# **XGB**
    parameters_XGB = { 'XGB__learning_rate': [0.021,0.022],
                      'XGB__max_depth':  [2,3],
                      'XGB__n_estimators':  [2000],
                      'XGB__reg_lambda':  [1.5, 1.6],
                      'XGB__reg_alpha':  [1,1.5],
                      #colsample_bytree , subsample
                     }

    gscv_XGB = GridSearchCV(pipe_XGB, parameters_XGB, n_jobs=-1,
                              scoring='neg_mean_squared_error', verbose=1, cv=5)

# #### LGBM

#
    parameters_LGBM = { 'LGBM__learning_rate': [0.01,0.02],
                       'LGBM__n_estimators':  [1000],
                       'LGBM__num_leaves':  [8,10],
                       'LGBM__bagging_fraction':  [0.7,0.8],
                       'LGBM__bagging_freq':  [1,2]}

    gscv_LGBM = GridSearchCV(pipe_LGBM, parameters_LGBM, n_jobs=-1,
                              scoring='neg_mean_squared_error', verbose=1, cv=5)

# #### AdaBoostRegressor


    parameters_ADA = { 'ADA__learning_rate': [3.5],
```

```
                    'ADA__n_estimators':  [500],
                    'ADA__base_estimator__max_depth':  [8,9,10]}

    pipe_ADA = Pipeline(
        steps= [('preprocessor', preprocessor),
                ('ADA', AdaBoostRegressor(
                    DecisionTreeRegressor(min_samples_leaf=5,
                                          min_samples_split=5),
                    random_state=5,loss='exponential')) ])

    gscv_ADA = GridSearchCV(pipe_ADA, parameters_ADA, n_jobs=-1,
                            scoring='neg_mean_squared_error', verbose=1, cv=5)
    list_pipelines_gscv = [gscv_Linear,gscv_Ridge, gscv_Huber,gscv_Lasso,gscv_ElaNet,gscv_RandomForestReg,gscv_DecisionTr
    return gscv_Linear,gscv_Ridge, gscv_Huber,gscv_Lasso,gscv_ElaNet,gscv_RandomForestReg,gscv_DecisionTreeReg, gscv_SVRe


##########################################################################################################################
### Loop over GridSearchCV Pipelines
#Grid Search and Crossvalidation (to Choose optimal hyper parameters)

##########################################################################

def run_MLAlg_gridCV(df,split,random, LABEL_COL):
    data1 = []
    data2 = []
    data3 = []
    X_train, X_test, y_train, y_test=traintestsplit(df,split,random, LABEL_COL)
    gscv_Linear,gscv_Ridge,gscv_Huber,gscv_Lasso,gscv_ElaNet,gscv_RandomForestReg,gscv_DecisionTreeReg, gscv_SVReg,gscv_G
    list_pipelines_gscv=list_pipelines_gscv
    for gscv in list_pipelines_gscv:
        gscv.fit(X_train, y_train)
        scores = cross_val_score(gscv.best_estimator_, X_train, y_train,
                                 scoring='neg_mean_squared_error', cv=5)


        scores = np.sqrt(-scores)
        mean=np.mean(scores)
        sd=np.std(scores)
        min_sc=np.min(scores)
        data1.append(({'mean_rmse':mean,
                       'std': sd,'min_rmse': min_sc}))

        #fitting training
        #gscv.fit(X_train, y_train)
```

```
        best_scr=np.sqrt(-gscv.best_score_)
        #print(np.sqrt(-gscv.best_score_))
        pred_yfit = gscv.predict(X_train)
        r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_train,pred_yfit,X_train)
        data2.append(({'R-Squared':r2,'Adj-R Squared': r2_adjusted,'MAE': mae,
                       'MSE':mse,'RMSE': rmse,'Best-Score': best_scr}))


        gscv.best_params_
    #prediction test
        pred_y = gscv.predict(X_test)
        r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_test,pred_y,X_test)
        data3.append(({'R-Squared':r2,'Adj-R Squared': r2_adjusted,'MAE': mae,
                       'MSE':mse,'RMSE': rmse}))
    results1 = pd.DataFrame(data=data1, columns=['mean_rmse','std','min_rmse'],
                            index=['gscv_Linear','gscv_Ridge', 'gscv_Huber','gscv_Lasso','gscv_ElaNet',
                                   'gscv_RandomForestReg','Decision Tree Reg', 'gscv_SVReg','gscv_GBR', 'gscv_XGB','gscv_
    results2 = pd.DataFrame(data=data2, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                            index=['gscv_Linear','gscv_Ridge', 'gscv_Huber','gscv_Lasso','gscv_ElaNet',
                                   'gscv_RandomForestReg','Decision Tree Reg','gscv_SVReg','gscv_GBR', 'gscv_XGB','gscv_L

    results3 = pd.DataFrame(data=data3, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                            index=['gscv_Linear','gscv_Ridge', 'gscv_Huber','gscv_Lasso','gscv_ElaNet',
                                   'gscv_RandomForestReg','Decision Tree Reg','gscv_SVReg','gscv_GBR', 'gscv_XGB','gscv_L
    #gscv_Linear,gscv_Ridge, gscv_Huber,gscv_Lasso,gscv_ElaNet,gscv_GBR, gscv_XGB,gscv_LGBM,gscv_ADA
    print('Grid Search CV Results')
    display(results1)
    print('Training Data Fit Results')
    display(results2)
    print('Test Data Prediction Results')
    display(results3)

    return results1, results2, results3
```

Part 3: Baysian Regression Functions

I use the default MCMC method in PYMC3's `sample` function, which is Hamiltonian Monte Carlo (HMC). Briefly, MCMC algorithms work by defining multi-dimensional Markovian stochastic processes, that when simulated (using Monte Carlo methods), will eventually converge to a state where successive simulations will be equivalent to drawing random samples from the posterior distribution of the model we wish to estimate.

The posterior distribution has one dimension for each model parameter, so we can then use the distribution of samples for each parameter to infer the range of possible values and/or compute point estimates (e.g. by taking the mean of all samples). I have define seven main functions to model inference using:

1. NUTS Algorithm
2. Hamiltonian MC
3. Metropolis MC
4. Slice MC
5. DEMetropolis MC
6. MetropolisZ MC
7. Sequential MC (SMC)

Below are the functions for these seven types of MCMC to estimate the parameters of Linear Regression Model through Bayesian approach.

The data is split into training and test (using training-test split to compute the predectionof these seven approaches.

The trace, Posterior and Forests plot are not shown here (options are made off). These are given in Appendix-I

In [12]:
```
##########################################################################################################
#Baysian Regression Problem Setup

##########################################################################################
# Baysian Regression Algorithms

# NUTS Algorithms
def NUTS_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
    # Set priors.
        priors = prior
    # Specify model.
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)
```

```
    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal = pm.sample(draws=draws, init = 'advi+adapt_diag', chains=chains, tune=tune, step=pm.NUTS(),  random
    return linear_model, trace_normal

#Hamiltonian MC Algorithms
def HamiltonianMC_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
    # Set priors.
        priors = prior
    # Specify model.
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)


    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal = pm.sample(draws=draws, chains=chains, tune=tune, step=pm.HamiltonianMC(),  random_seed=SEED)
    return linear_model, trace_normal

# Metropolis MC Algorithm
def Metropolis_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune,verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
    # Set priors.
        priors = prior
    # Specify model.
```

```
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)


    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal = pm.sample(draws=draws, chains=chains, tune=tune, step=pm.Metropolis(),  random_seed=SEED)
    return linear_model, trace_normal
# Slice MC Algorithm

def Slice_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
    # Set priors.
        priors = prior
    # Specify model.
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)


    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal = pm.sample(draws=draws, chains=chains, tune=tune, step=pm.Slice(),  random_seed=SEED)
    return linear_model, trace_normal

def DEMetropolis_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
    # Set priors.
```

```
        priors = prior
    # Specify model.
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)


    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal = pm.sample(draws=draws, chains=chains, tune=tune, step=pm.DEMetropolis(),  random_seed=SEED)
    return linear_model, trace_normal

def DEMetropolisZ_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
    # Set priors.
        priors = prior
    # Specify model.
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)


    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal = pm.sample(draws=draws, chains=chains, tune=tune, step=pm.DEMetropolisZ(),  random_seed=SEED)
    return linear_model, trace_normal
# Seuential MC Algorithm
def SMC_Reg(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0):
    X=X_train
    y=y_train
    with pm.Model() as linear_model:
    # Set data container.
        data = pm.Data("data", X)
    # Define GLM family.
        family =family#glm.families.Normal()# pm.glm.families.Binomial()
```

```
    # Set priors.
        priors = prior
    # Specify model.
        glm.GLM(y=y, x=data,family=family,intercept=False,labels=labels,priors=priors)

    # Sample from prior distribution.
    with linear_model:
        prior_checks = pm.sample_prior_predictive(samples=prior_samples, random_seed=SEED)
    with linear_model:
    # Configure sampler.

        trace_normal =  pm.sample_smc(draws)

    return linear_model, trace_normal



#########################################################################################################################
#Get the defined function
# Helper function to provide list of supported algorithms for Baysian Regression

def get_supported_algorithms_reg():
    covered_algorithms = [NUTS_Reg,HamiltonianMC_Reg,Metropolis_Reg,Slice_Reg,DEMetropolis_Reg,DEMetropolisZ_Reg,SMC_Reg]
    return covered_algorithms

#########################################################################################################################
#Baysian Regression Call (Note:does not show the Posterior plots, Trace and Forest plots )
#####################################

# Helper function to run all algorithms provided in algo_list over given dataframe,
# By default it will run all supported algorithms
# Here we do not show the Posterior plots, Trace and Forest plots
def run_baysian_algorithms(df,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=0):
    algo_list=get_supported_algorithms_reg()
    data1=[]
    data2=[]
    # Lets make a copy of dataframe and work on that to be on safe side
    _df = df.copy()
    X_train, X_test, y_train, y_test, labels=train_test(formula,_df)

    #algo_model_map = {}
    for algo in algo_list:
        print("============ " + algo.__name__ + " ===========")
        model, trace_normal = algo(X_train, y_train,family,labels,prior,prior_samples,SEED,draws, chains, tune, verbose=0
        #algo_model_map[algo.__name__] = model.get("model_obj", None)
```

```
        print ("============================== \n")
        #(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0)
        # Plot chains.
        #az.plot_trace(data=trace_normal);
        #display(az.summary(trace_normal))
        print('===============Training Data Fit=======================')

        pm.set_data({"data": X_train}, model=model)
        # Generate posterior samples.
        ppc_train = pm.sample_posterior_predictive(trace_normal, model=model, samples=1000)
        train_pred = ppc_train["y"].mean(axis=0)
        #pm.plot_posterior(trace_normal, figsize = (12, 10));#, text_size = 20);
        #pm.forestplot(trace_normal);
        r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_train,train_pred,X_train)
        data1.append(({'R-Squared':r2,'Adj-R Squared': r2_adjusted,'MAE': mae,
                       'MSE':mse,'RMSE': rmse}))
        print('===============Test Data Fit=======================')
        pm.set_data({"data": X_test}, model=model)
        # Generate posterior samples.
        ppc_test = pm.sample_posterior_predictive(trace_normal, model=model, samples=1000)
        # Compute the point prediction by taking the mean

        p_test_pred = ppc_test["y"].mean(axis=0)
        r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_test,p_test_pred,X_test)
        data2.append(({'R-Squared':r2,'Adj-R Squared': r2_adjusted,'MAE': mae,
                       'MSE':mse,'RMSE': rmse}))
    results1 = pd.DataFrame(data=data1, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                            index=['NUTS','HamiltonianMC','Metropolis','Slice','DEMetropolis','DEMetropolisZ','SMC'])


    results2 = pd.DataFrame(data=data2, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                            index=['NUTS','HamiltonianMC','Metropolis','Slice','DEMetropolis','DEMetropolisZ','SMC'])
    print('Training Data Fit Results')
    display(results1)
    print('Test Data Prediction Results')
    display(results2)
    return results1, results2

##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
#Baysian Regression Call (to show the Posterior plots, Trace and Forest plots ) for Appendix-I
####################################
def run_baysian_algorithms_plots(df,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=0):
    algo_list=get_supported_algorithms_reg()
```

```
data1=[]
data2=[]
# Lets make a copy of dataframe and work on that to be on safe side
_df = df.copy()
X_train, X_test, y_train, y_test, labels=train_test(formula,_df)

#algo_model_map = {}
for algo in algo_list:
    print("============ " + algo.__name__ + " ==========")
    model, trace_normal = algo(X_train, y_train,family,labels,prior,prior_samples,SEED,draws, chains, tune, verbose=0
    #algo_model_map[algo.__name__] = model.get("model_obj", None)
    print ("============================== \n")
    #(X_train, y_train,family,labels, prior,prior_samples,SEED,draws, chains, tune, verbose=0)
    # Plot chains.
    az.plot_trace(data=trace_normal);
    display(az.summary(trace_normal))
    print('===============Training Data Fit======================')

    pm.set_data({"data": X_train}, model=model)
    # Generate posterior samples.
    ppc_train = pm.sample_posterior_predictive(trace_normal, model=model, samples=1000)
    train_pred = ppc_train["y"].mean(axis=0)
    pm.plot_posterior(trace_normal, figsize = (12, 10));#, text_size = 20);
    pm.forestplot(trace_normal);
    r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_train,train_pred,X_train)
    data1.append(({'R-Squared':r2,'Adj-R Squared': r2_adjusted,'MAE': mae,
                   'MSE':mse,'RMSE': rmse}))
    print('===============Test Data Fit======================')
    pm.set_data({"data": X_test}, model=model)
    # Generate posterior samples.
    ppc_test = pm.sample_posterior_predictive(trace_normal, model=model, samples=1000)
    # Compute the point prediction by taking the mean

    p_test_pred = ppc_test["y"].mean(axis=0)
    r2, r2_adjusted,mae,mse,rmse=validationmetrics_reg(y_test,p_test_pred,X_test)
    data2.append(({'R-Squared':r2,'Adj-R Squared': r2_adjusted,'MAE': mae,
                   'MSE':mse,'RMSE': rmse}))
results1 = pd.DataFrame(data=data1, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                        index=['NUTS','HamiltonianMC','Metropolis','Slice','DEMetropolis','DEMetropolisZ','SMC'])


results2 = pd.DataFrame(data=data2, columns=['R-Squared','Adj-R Squared','MAE','MSE','RMSE'],
                        index=['NUTS','HamiltonianMC','Metropolis','Slice','DEMetropolis','DEMetropolisZ','SMC'])
print('Training Data Fit Results')
display(results1)
```

```
    print('Test Data Prediction Results')
    display(results2)
    return results1, results2
```

Data Analysis

Consider two data sets:

1. House Prices Dataset-1 (20640, 9)
2. Prices Dataset-2 (2919, 81)

Here I am trying to explore the following:

- EDA (using Seaborn and interactive charts with Plotly
- Feature Engineering
- Preprocessing using sklearn Pipeline
- use GridSearchCV with Pipelines
- apply ML models like Ridge, Lasso, ElasticNet, Boosting, etc (13 ML Algorithms)
- apply the Baysian MCMC Algorithms like NUTS, Hamiltonian,Metropolis MC, MetropolisZ MC etc (7 Baysian MCMC Agorithms)
- compare the performance of the ML and Baysian Algorithms for validation and test data

Note: My both datasets are house pricing datas. The first one has more rows (less features) and second has more columns but less rows and needs lots of feature engineering. The reason for choosing these datasets is to see and compare the performance of ML and Baysian algorithms with more rows and less features and smaller number of observations and more features.

Dataset I: House Prices Dataset

House and Location features are considered to predict the Price.

The data consists of 20640 rows and 9 columns. It has 7 features and one Price - the house's sale price in million dollars, which is the target variable to be predicted. The features are:

1. MedInc
2. HouseAge
3. AveRooms
4. AveBedrms

5. Population
6. AveOccup
7. Latitude
8. Longitude

Load Data

In [13]:
```
from google.colab import files
uploaded=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving house_data.csv to house_data.csv
```

In [14]:
```
import io
df=pd.read_csv(io.BytesIO(uploaded['house_data.csv']))
```

In [15]:
```
#FILE_NAME = "house_data.csv"
LABEL_COL = "Price"
#df = template.load_data(FILE_NAME)
display(df.head())
print(df.shape)
print(df.dtypes)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
(20640, 9)
MedInc        float64
HouseAge      float64
AveRooms      float64
AveBedrms     float64
Population    float64
AveOccup      float64
Latitude      float64
```

```
Longitude     float64
Price         float64
dtype: object
```

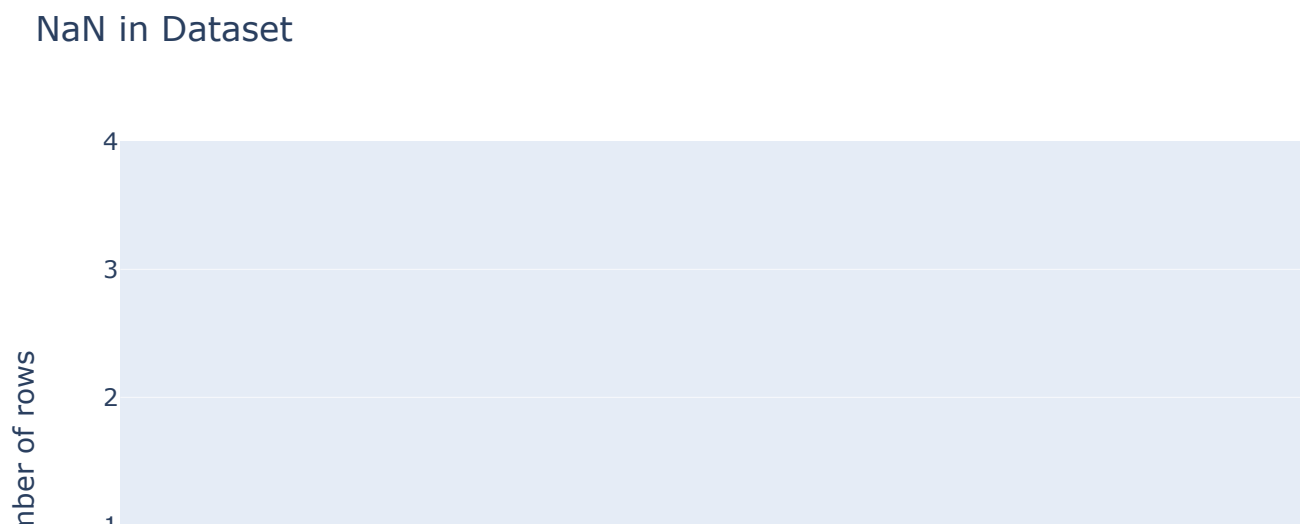
In [ ]:
```
df.isnull().sum()
```

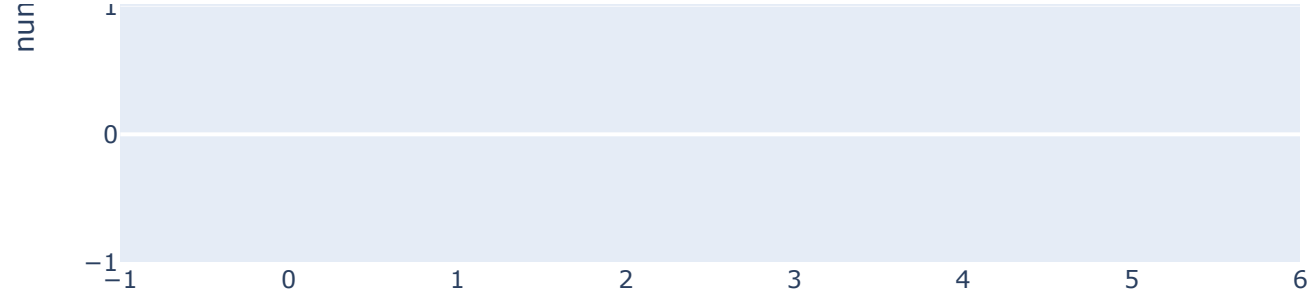
Out[ ]:
```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
Price         0
dtype: int64
```

In [19]:
```
# Import the necessaries libraries
import plotly.offline as pyo
import plotly.graph_objs as go
# Set notebook mode to work in offline
pyo.init_notebook_mode()
```

Missing Values Analysis

In [ ]:
```
template.missing_values_display(df)
```





	Total	Percent
Price	0	0.0
Longitude	0	0.0
Latitude	0	0.0
AveOccup	0	0.0
Population	0	0.0
AveBedrms	0	0.0
AveRooms	0	0.0
HouseAge	0	0.0
MedInc	0	0.0

The missing value analysis shows that there is no missing values in the dataset.

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'Price'],
      dtype='object')
```

Cleaning Data

This funtion remove the redunt features, which improves the performance of the ML models and interpolate the missing values. However, here we are not removing any column or row as there is no missing data or reduntand factors.

In [ ]:
```
df = template.cleaningup(df, to_numeric=[], cols_to_interpolate=[],
                         cols_to_delete=[], to_date=[])
```

```
df is all cleaned up..
```

In [ ]:
```
df.isnull().sum().sum()
```

Out[ ]:
```
0
```

Numerical and Categorical features

In [ ]:
```
numerical_feats, categorical_feats=template.numirical_catagorical_columns(df)
```

```
Number of Numerical features:  9
Number of Categorical features:  0
**************************************************************************************************
Numrical Columns
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'Price'],
      dtype='object')
**************************************************************************************************
Catagorical Columns
Index([], dtype='object')
```

There is no string or catagorical column in data. All nine variables are contenous.

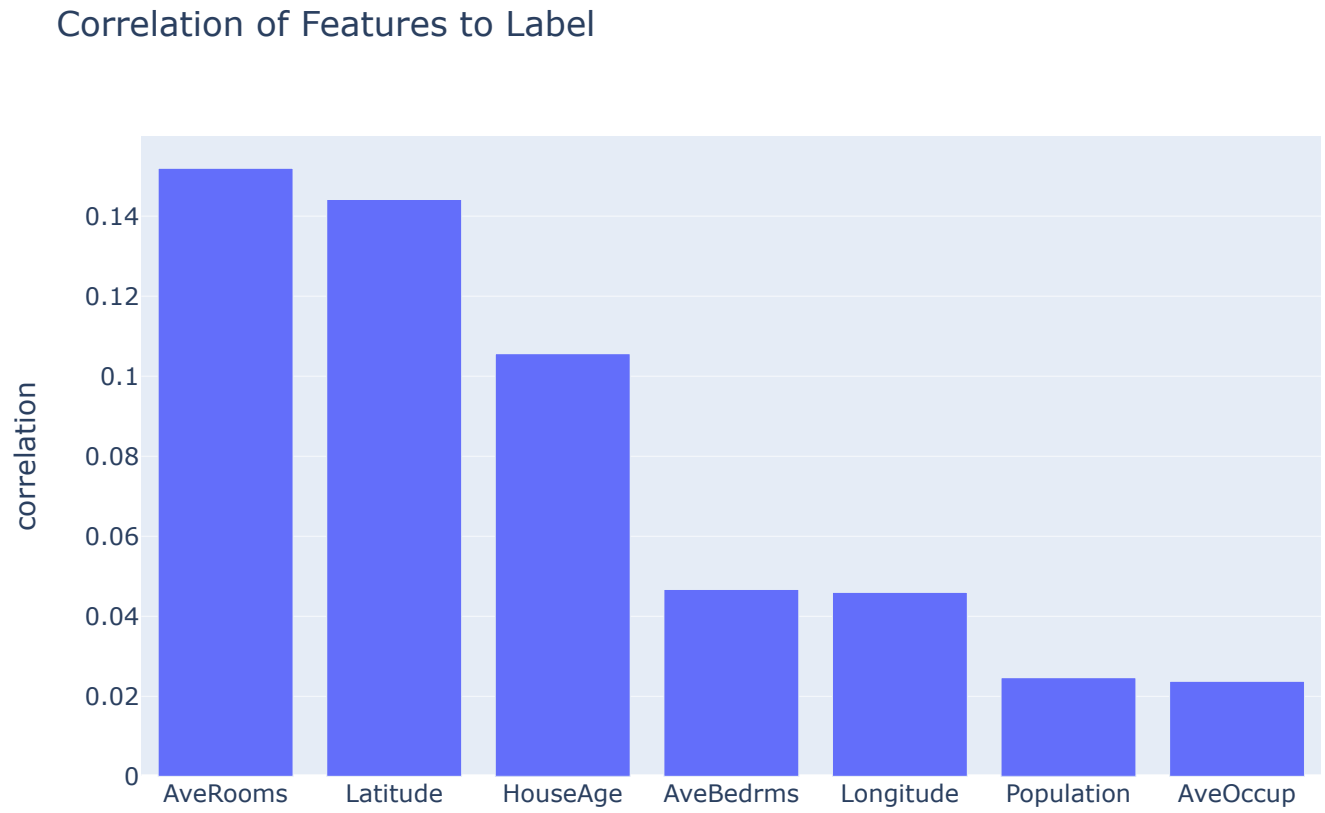
In [ ]:
```
df[numerical_feats].head()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Correlation of Features with SalePrice

In [ ]:
```
LABEL_COL = "Price"
df2=df
template.corr_x_y(df2,LABEL_COL)
```



Correlation of numerical features to SalePrice

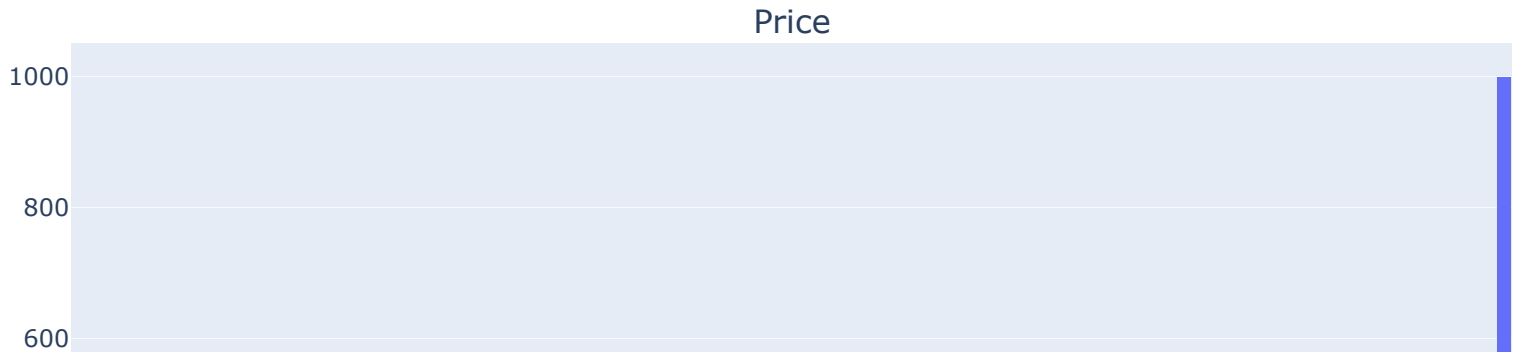
Keeping other parameters constant, we expect the value of a House to increase with MedInc and AveRooms.

Also for this dataset, large correlations to Price are observed for many of the Area features, such as MedInc, Latitude, etc. In the numerical features section, these features are explored in more detail and see how the results can be used for outlier detection and feature engineering.

Distribution of the target variable

In [ ]:
```
LABEL_COL = "Price"
df2=df
template.dist_label(df2,LABEL_COL)
```

```
mean: 2.0685581690891843
standard deviation: 1.1539561587441387
skewness: 0.9777632739098348
kurtosis: 0.3278702429465876
```



The distribution of Price seems to be normal with mean of 2.069 and standard deviation of 1.153. The skewness is close to 1 and there is a very big spike at the price between 4.97 and 5. However,the distribution is a bit skewed to the right. The excess Kurtosis are close to zero.

In [ ]:
```
df.corr()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.015176	0.688075
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.108197	0.105623
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.027540	0.151948
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013344	-0.046701
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.099773	-0.024650
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002476	-0.023737
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924664	-0.144160
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000000	-0.045967
Price	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.144160	-0.045967	1.000000

The MedInc variable has the highest uncoditional correlation with the price which is about 0.69, followed by AveRooms which is 0.15 and Latitude (-0.144) which is indicator of location. Since we analyse these features in more detail through scatterplot which will be helpful to pinpoint the possible outliers.

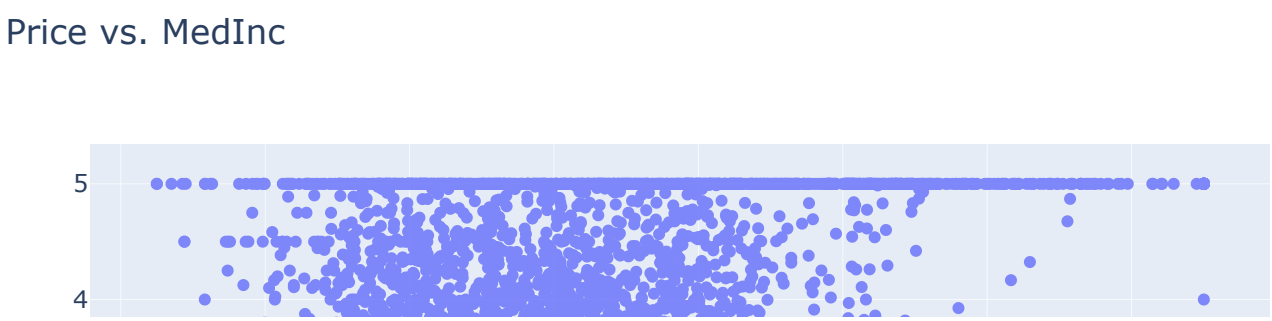
Ffeatures Analysis

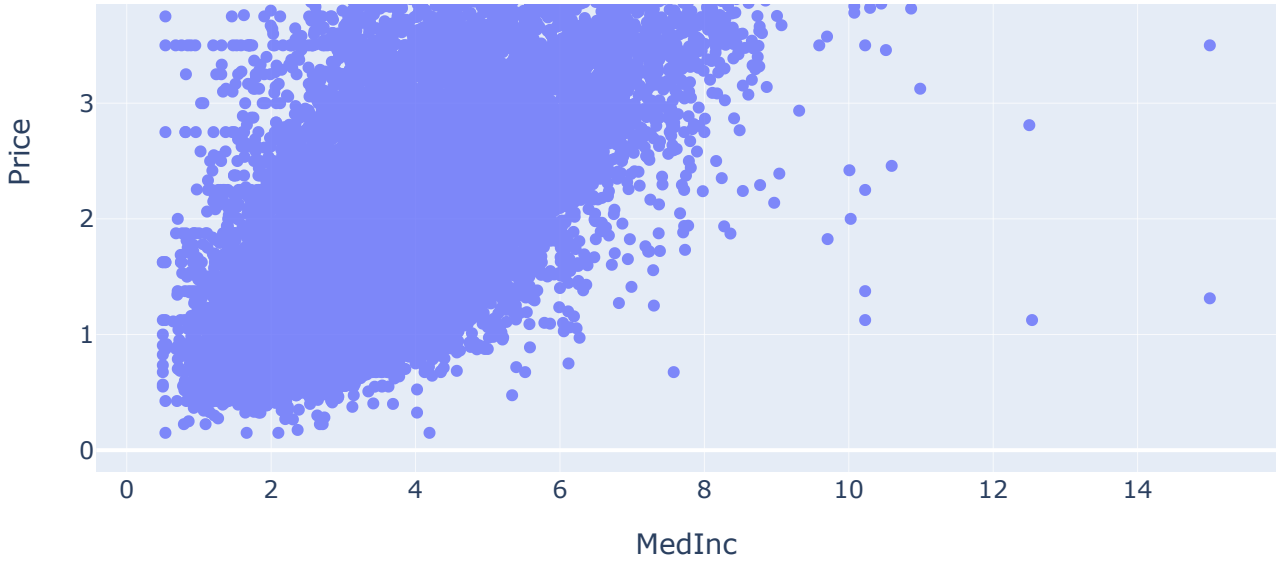
Scatterplot: Price vs MedInc

Of the features, 'MedInc' has the largest correlation to Price.

In [ ]:
```
df=df2
```

In [ ]:
```
template.plotly_scatter_x_y(df, 'MedInc', 'Price')
```





There seems to be positive relation between MedInc and Price, however there are certain outliers. ***Note on Outlier Detection***
We store the index of the these data points to the lower right,
with Price < 4.3 and MedInc > 12

In [ ]:
```
# outliers GrLivArea
outliers_MedInc = df.loc[(df['MedInc']>12.0) & (df['Price']<4.3)]
outliers_MedInc[['MedInc' , 'Price']]
```

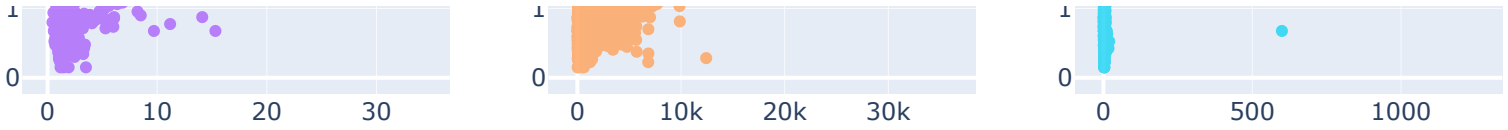
Out[ ]:

	MedInc	Price
1566	15.0001	3.500
11912	12.5381	1.125
16828	12.5000	2.810
17798	12.3292	4.167
18501	15.0001	1.313
18504	15.0001	4.000

In [ ]:
```
df = df.drop(outliers_MedInc.index)
```

Scatterplots: SalePrice vs features

In [ ]:
```
df=df
y_col_vals = 'Price'
features = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup','Latitude', 'Longitude']
                # 'ScreenPorch'
x_col_vals = features
title='Price' + ' vs. features'
fig=template.scatter_multi_col(df, x_col_vals, y_col_vals,title)
```



The results of scatter plot show that there are some outliers in AveRoom, AveBedromm, Population and AveOccup. It seems that due to these outliers there is negetive correlation between price and two features (AveBedromm, and AveOccup), which were expected to be positively correlated with price.

Like for AveRoom, AveBedromm, Population and AveOccup, there are certain outliers at the lower right also for all_SF
We are going to drop these now.

In [ ]:
```
outliers_AveRooms = df.loc[(df['AveRooms']>50.0) & (df['Price']<5.0)]
print(outliers_AveRooms[['AveRooms' , 'Price']])
df = df.drop(outliers_AveRooms.index)
```

```
         AveRooms  Price
1912    56.269231  2.063
1913    61.812500  4.375
1979   132.533333  1.625
2395    50.837838  1.250
9676    52.848214  2.214
11707   52.690476  1.700
11862   59.875000  0.675
12447   62.422222  0.875
```

Dropping Outliers

In [ ]:
```
outliers_AveBedrms = df.loc[(df['AveBedrms']>10.0) & (df['Price']<5.0)]
print(outliers_AveBedrms[['AveBedrms' , 'Price']])
df = df.drop(outliers_AveBedrms.index)
```

```
      AveBedrms  Price
1240  11.181818  0.775
```

In [ ]:
```
outliers_Population = df.loc[(df['Population']>12000.0) & (df['Price']<5.0)]
print(outliers_Population[['Population' , 'Price']])
df = df.drop(outliers_Population.index)
```

```
       Population  Price
922       12203.0  4.511
```

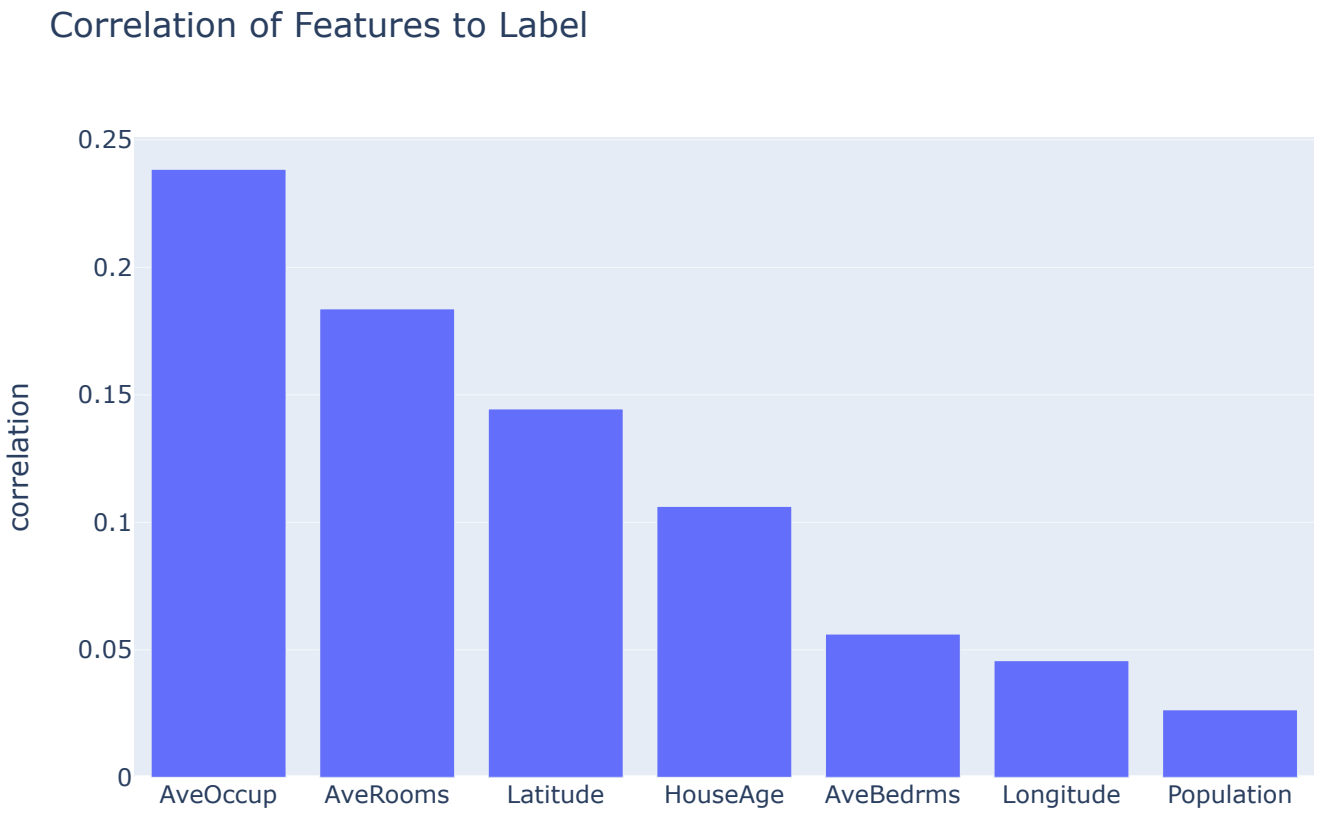
```
6057      15507.0  2.539
6066      15037.0  3.397
9019      12873.0  3.992
9744      12153.0  1.528
9880      28566.0  1.188
10309     16122.0  3.663
12215     13251.0  2.123
13139     16305.0  1.537
15360     35682.0  1.344
17413     12427.0  0.283
```

In [ ]:
```
outliers_AveOccup = df.loc[(df['AveOccup']>10.0) & (df['Price']<5.0)]
print(outliers_AveOccup[['AveOccup' , 'Price']])
df = df.drop(outliers_AveOccup.index)
```

```
          AveOccup  Price
91       10.272727  1.375
270      12.234043  2.250
1039     17.177700  1.219
1067     11.295918  1.375
2511     14.000000  1.625
2723     12.843750  1.425
2899     11.634146  0.475
3364    599.714286  0.675
4479     18.500000  1.357
5985     12.130081  1.250
5986     21.333333  2.125
7164     15.812155  1.238
9172     83.171429  1.546
12104    63.750000  1.625
12443    12.098940  0.535
13034   230.172414  2.250
13366    33.952941  1.833
14756    18.821818  1.533
14804    10.980000  0.675
15790    15.602941  2.250
16420    51.400000  1.625
16528    12.895397  1.118
16594    12.296089  0.630
16643    13.693487  3.000
16669   502.461538  3.500
16672    16.937500  0.425
17891    10.153846  0.675
18520    13.594828  1.313
19006  1243.333333  1.375
19435    18.444444  1.625
19524    16.048780  1.625
20121    13.212987  1.154
20352    19.312500  0.525
```

After dropping these outliers, the correlation with these variables has increased

In [ ]:
```
LABEL_COL = "Price"
df=df
template.corr_x_y(df,LABEL_COL)
```



In [ ]:
```
df.corr()
```

Out[ ]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
MedInc	1.000000	-0.119342	0.389322	-0.080391	0.002725	-0.049794	-0.081478	-0.013700	0.690500
HouseAge	-0.119342	1.000000	-0.175808	-0.096217	-0.308045	0.001478	0.010999	-0.108092	0.106246

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
AveRooms	0.389322	-0.175808	1.000000	0.784547	-0.079197	-0.048957	0.111463	-0.033171	0.183725
AveBedrms	-0.080391	-0.096217	0.784547	1.000000	-0.073009	-0.073512	0.072970	0.017019	-0.056187
Population	0.002725	-0.308045	-0.079197	-0.073009	1.000000	0.168922	-0.114995	0.106702	-0.026477
AveOccup	-0.049794	0.001478	-0.048957	-0.073512	0.168922	1.000000	-0.153856	0.159203	-0.238442
Latitude	-0.081478	0.010999	0.111463	0.072970	-0.114995	-0.153856	1.000000	-0.925107	-0.144475
Longitude	-0.013700	-0.108092	-0.033171	0.017019	0.106702	0.159203	-0.925107	1.000000	-0.045748
Price	0.690500	0.106246	0.183725	-0.056187	-0.026477	-0.238442	-0.144475	-0.045748	1.000000

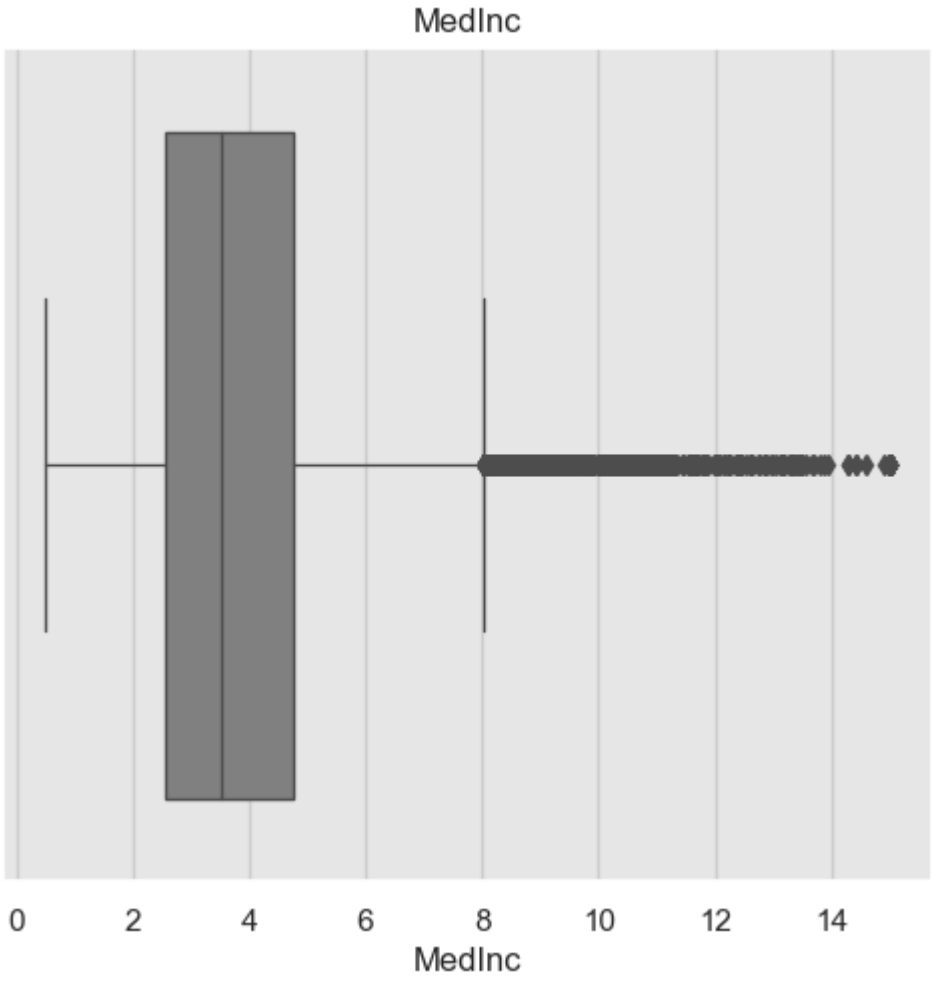
As seen before in the simple scatter plot, there is a strong tendency for increasing Price with a smaller number of Bedrooms. But this color plot also shows a correlation of Bedrooms and AveRooms. So, the probability that a house has a more rooms increases with its number of bathrooms. Another option to highlight the correlation of Price to AveBedrms and AveRooms as well as the correlation between AveOccup and AveBedrms is a 3d scatter plot as below. However, we cannot show it here because of hanging the notebook.

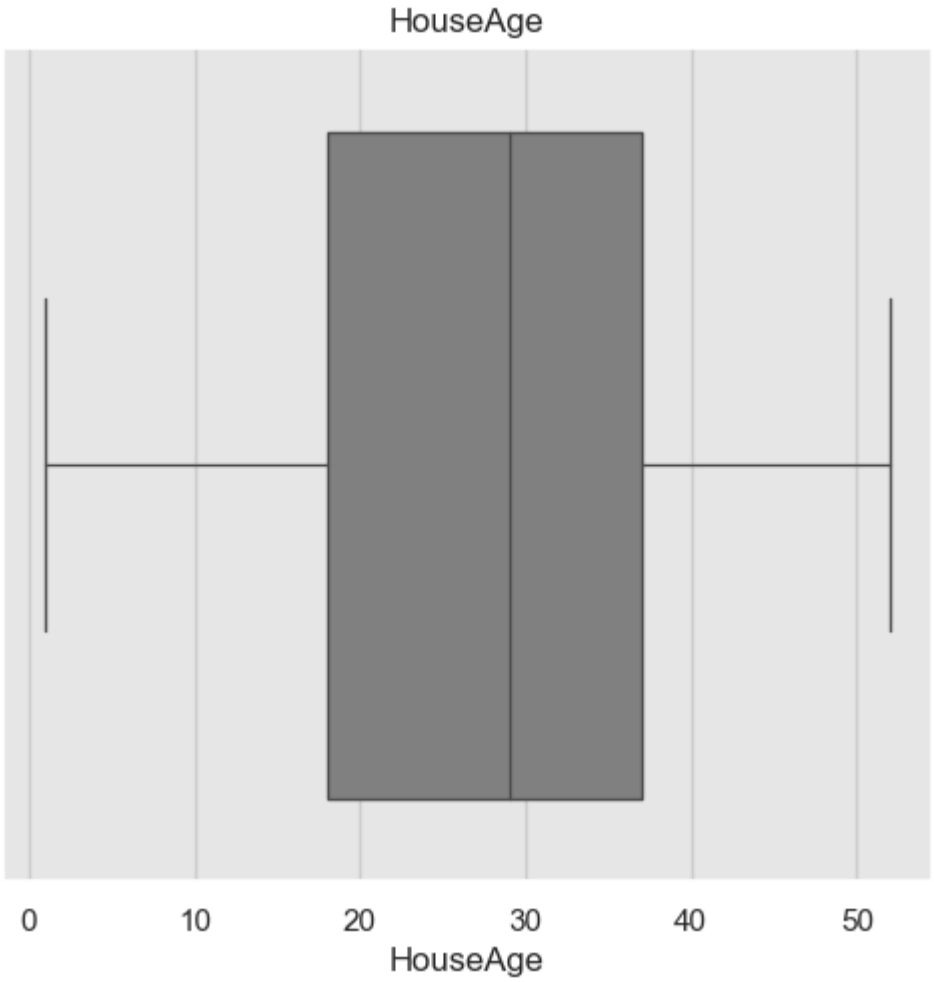
Numerical Columns discriptive stats and plots

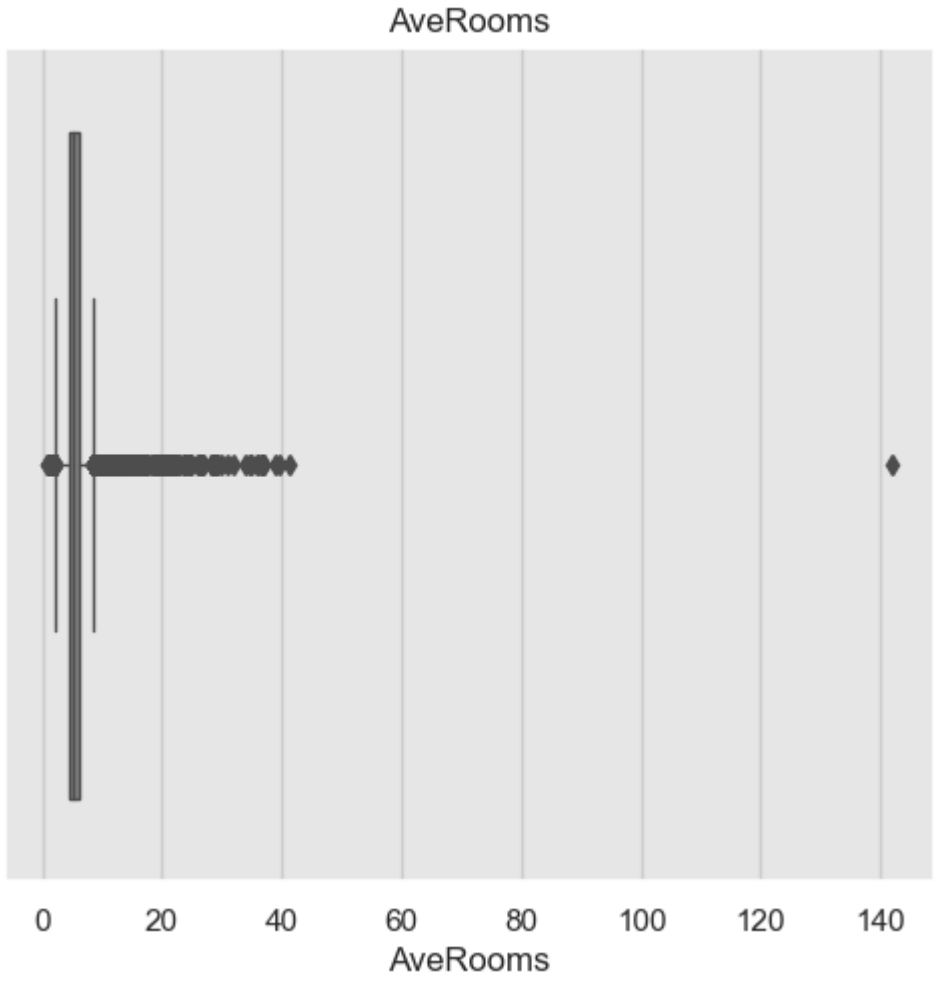
In [ ]:
```
template.basicanalysis(df)

template.numcolanalysis(df)
```

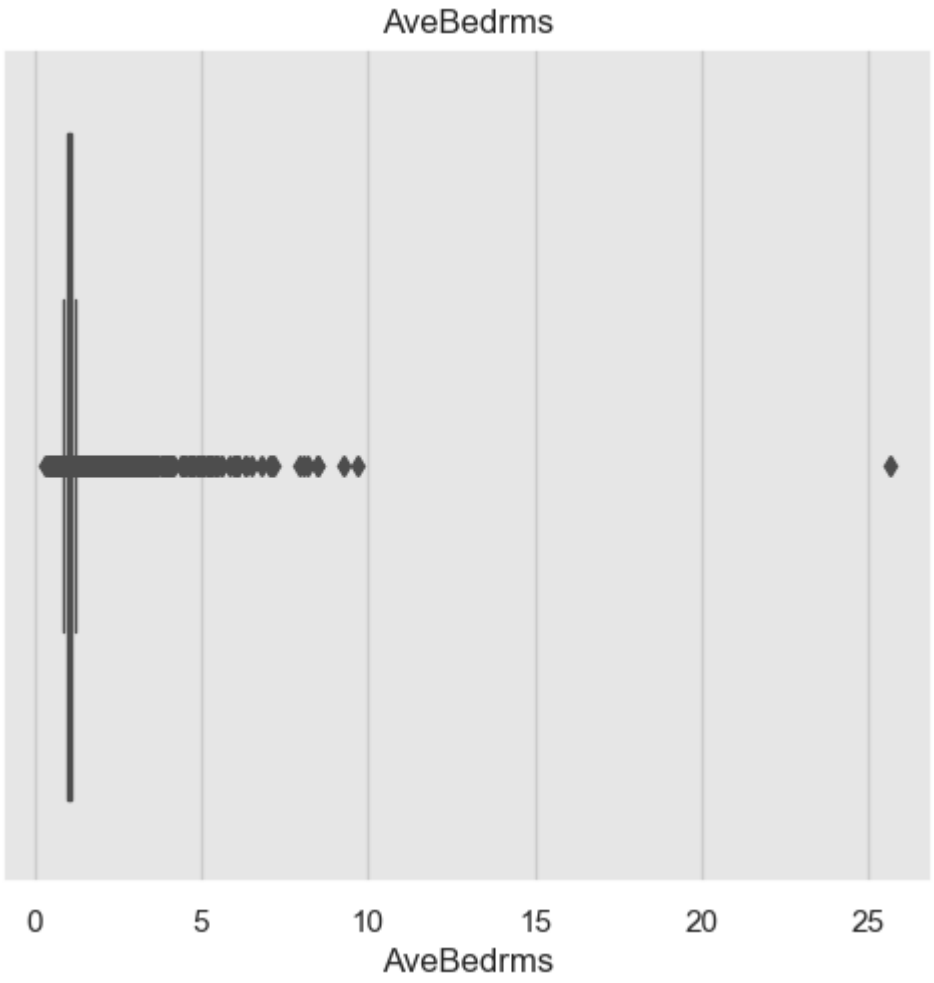
```
Shape is:
 (20581, 9)

 Columns are:
 Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'Price'],
      dtype='object')

 Types are:
 MedInc        float64
HouseAge      float64
AveRooms      float64
AveBedrms     float64
Population    float64
AveOccup      float64
Latitude      float64
Longitude     float64
Price         float64
```

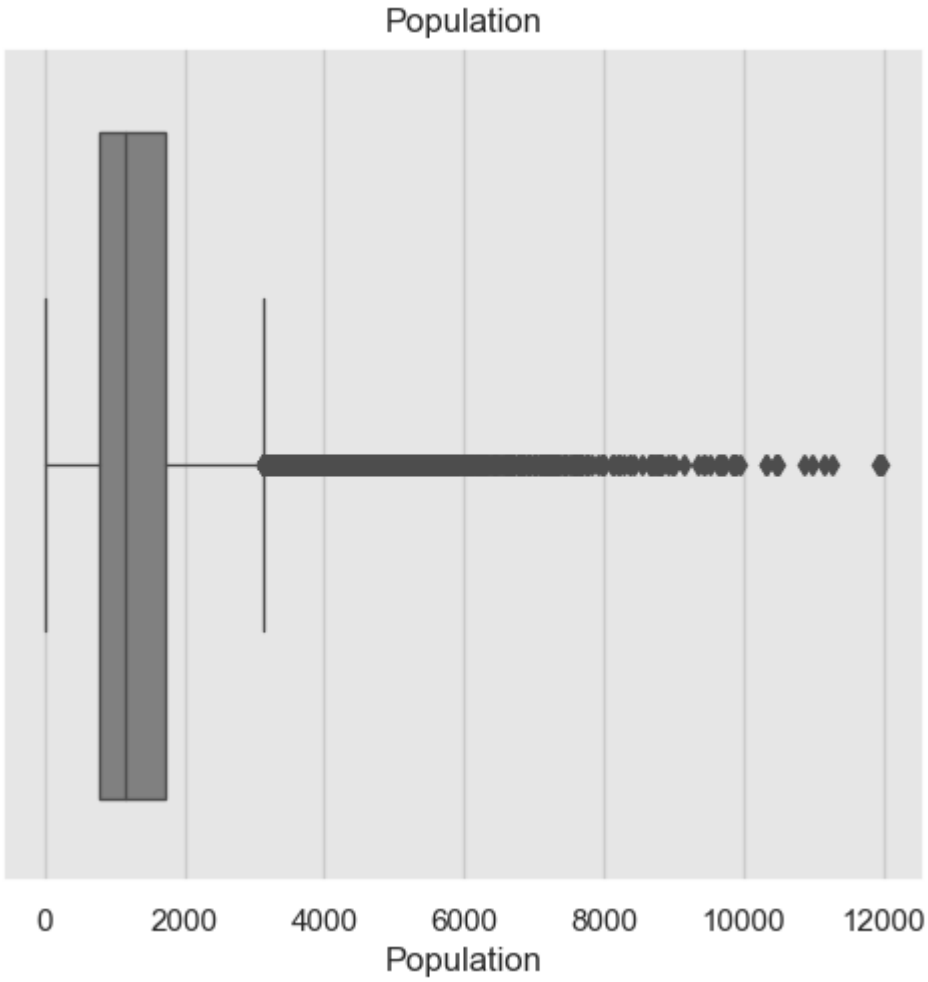
```
dtype: object

 Statistical Analysis of Numerical Columns:
              MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20581.000000  20581.000000  20581.000000  20581.000000  20581.000000
mean       3.868265     28.653904      5.401116      1.090840   1415.832613
std        1.892212     12.576015      2.079524      0.357684   1053.605253
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.564000     18.000000      4.440000      1.006061    788.000000
50%        3.534500     29.000000      5.227848      1.048689   1166.000000
75%        4.742600     37.000000      6.048843      1.099222   1723.000000
max       15.000100     52.000000    141.909091     25.636364  11973.000000

           AveOccup      Latitude     Longitude         Price
count  20581.000000  20581.000000  20581.000000  20581.000000
mean       2.921659     35.629853   -119.568985      2.069348
std        0.818022      2.135623      2.003484      1.154092
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429194     33.930000   -121.800000      1.196000
50%        2.817391     34.250000   -118.490000      1.798000
75%        3.279359     37.710000   -118.010000      2.649000
max       41.214286     41.950000   -114.310000      5.000010
```

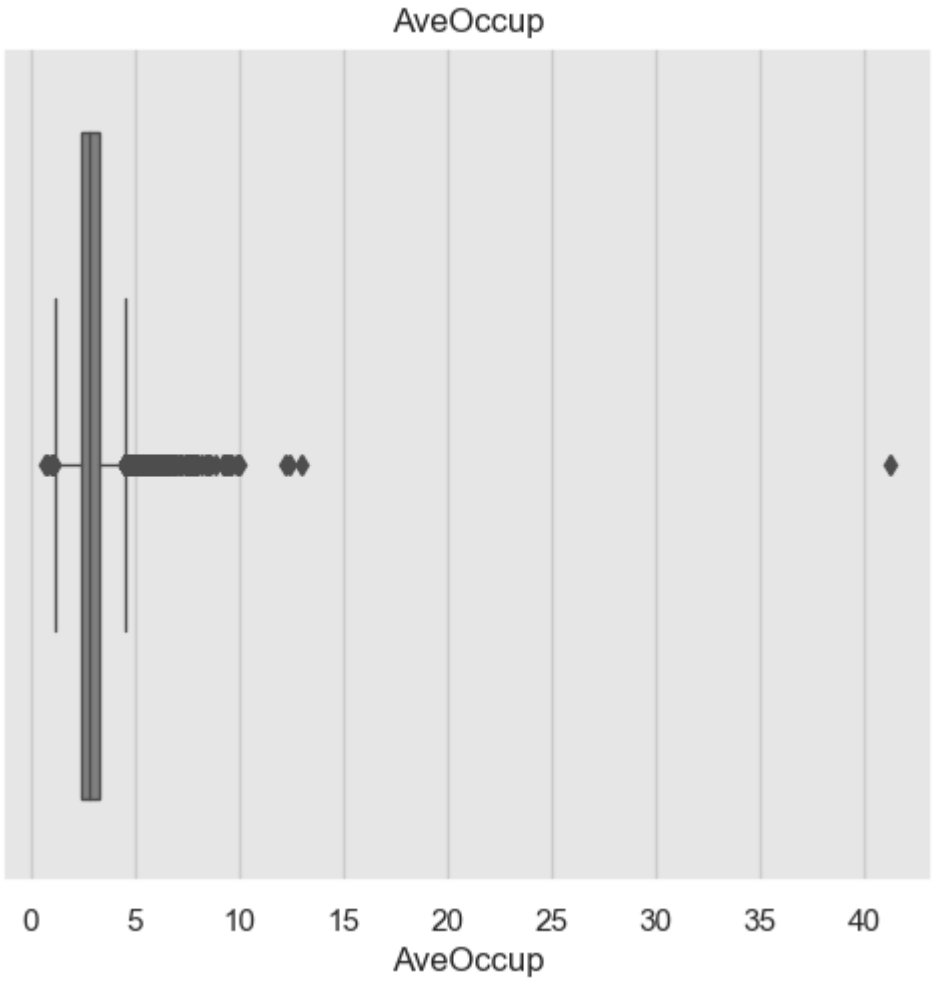

MedInc
0
2
4
6
8
10
12
14
MedInc

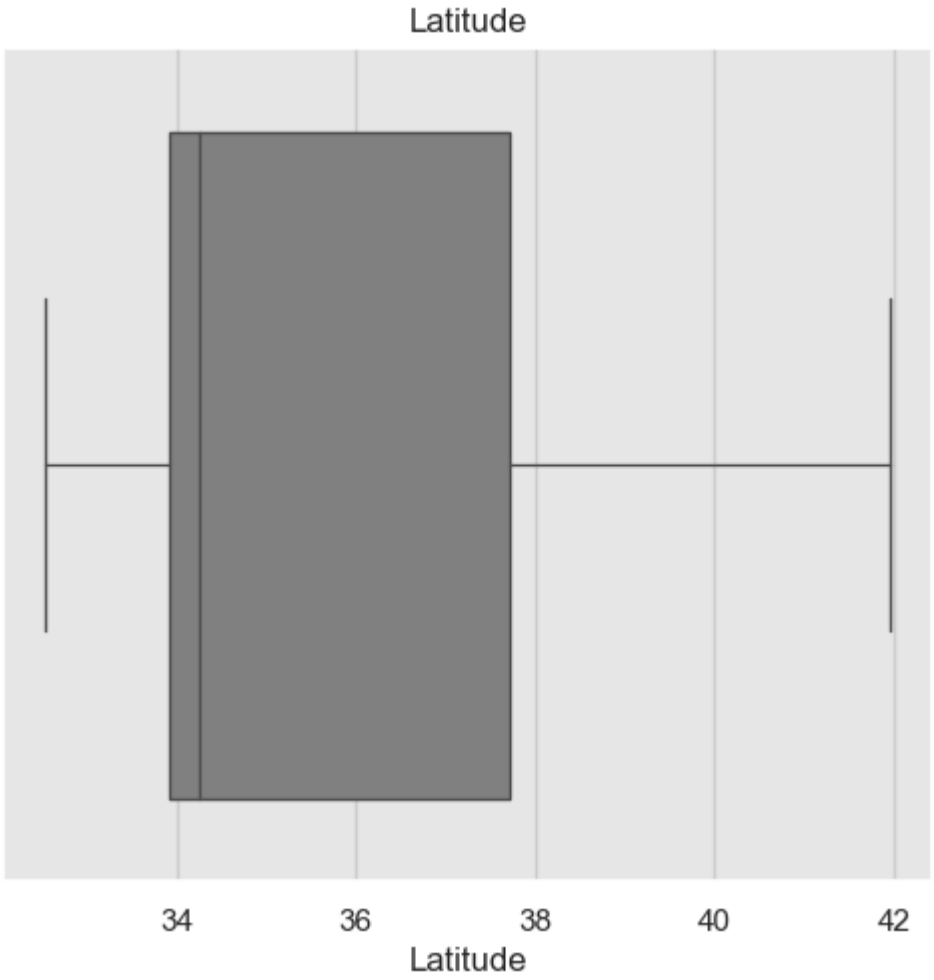

HouseAge
0
10
20
30
40
50
HouseAge

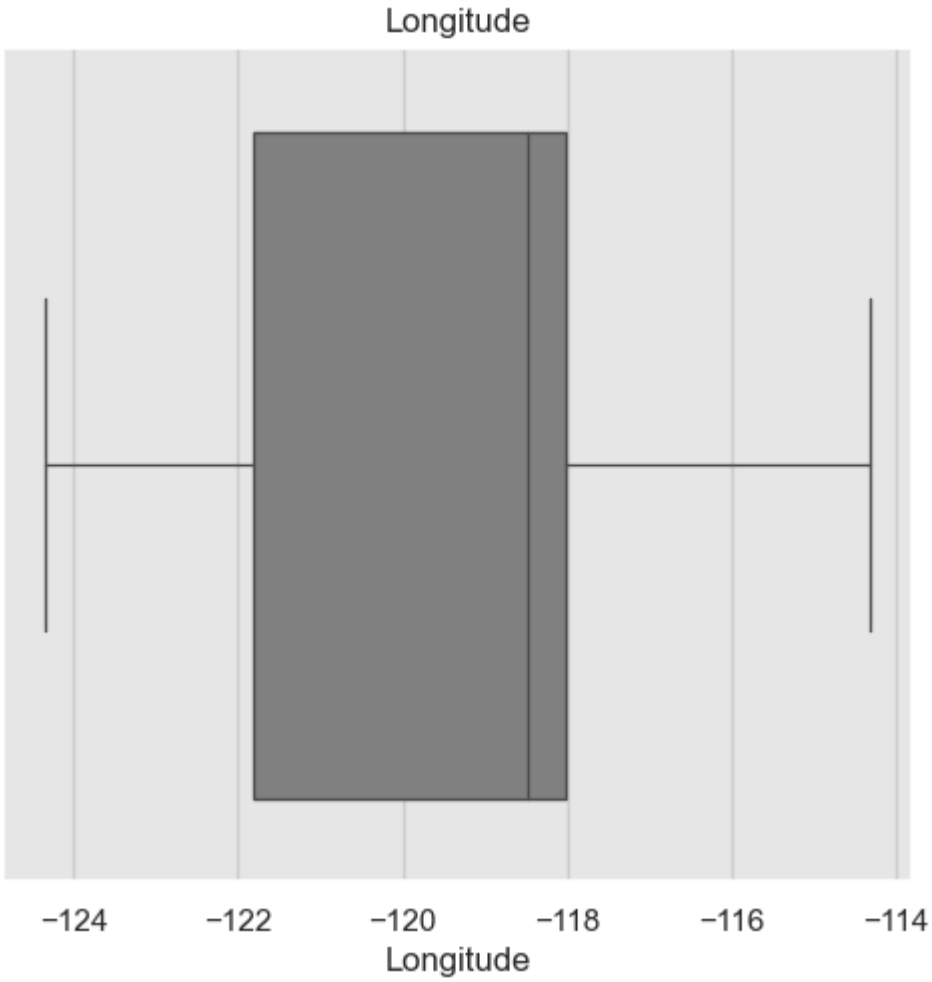

AveRooms
0
20
40
60
80
100
120
140
AveRooms

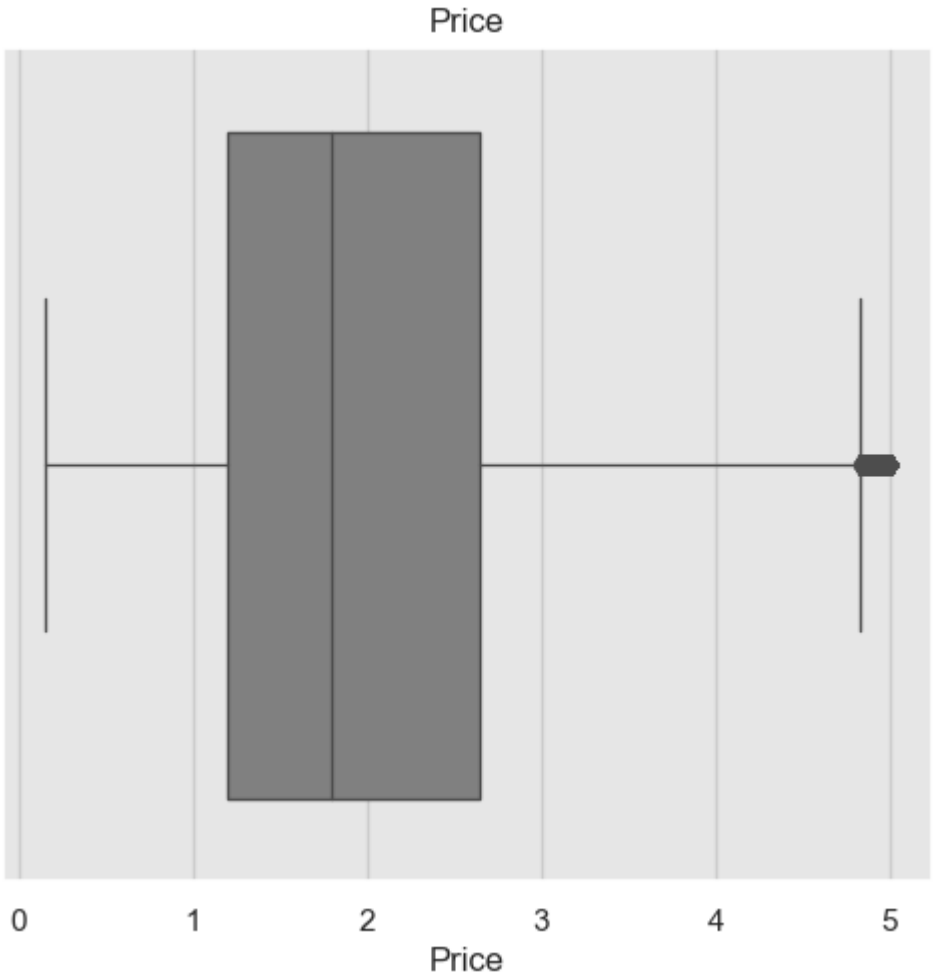

AveBedrms
0
5
10
15
20
25
AveBedrms

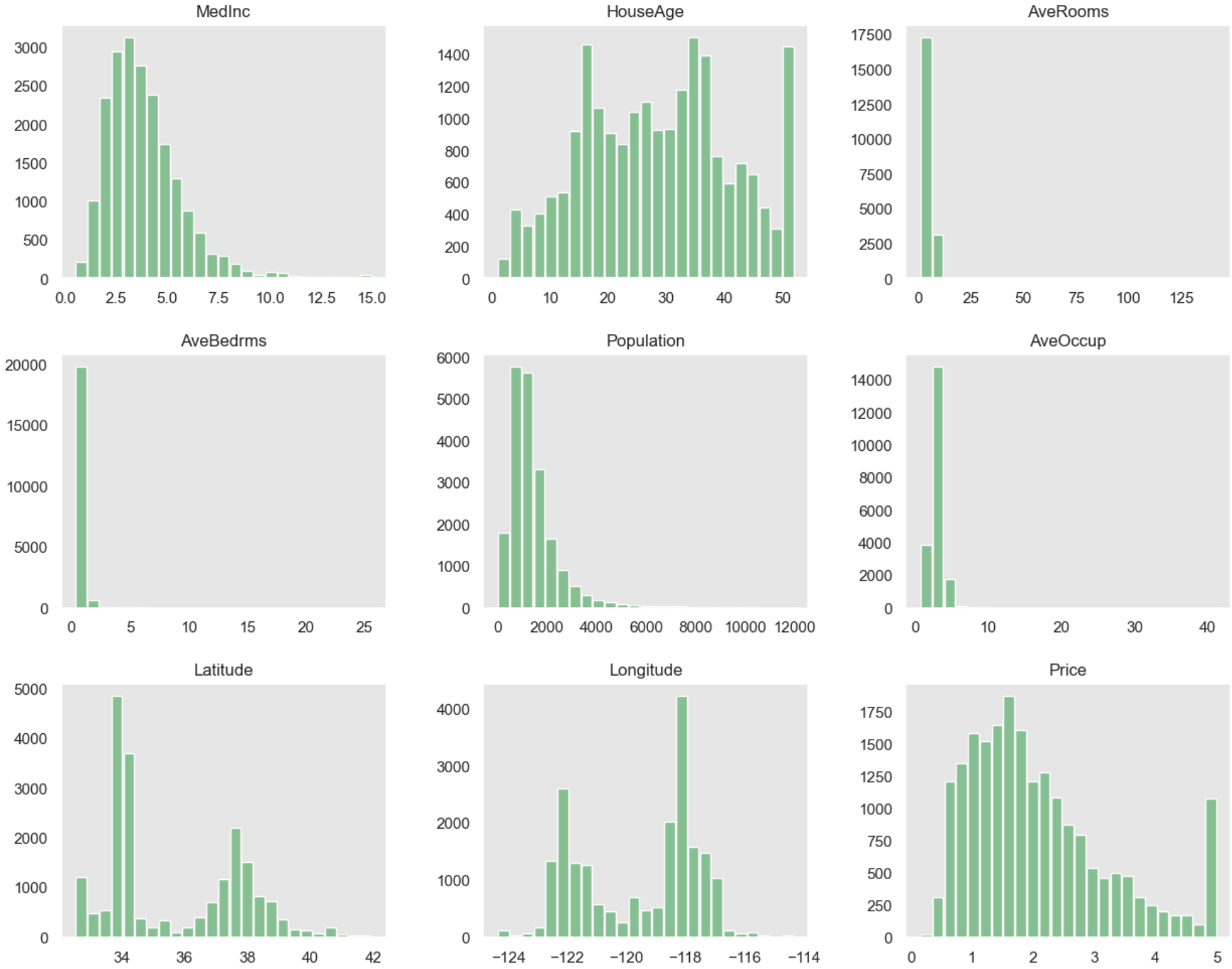

Population
0
2000
4000
6000
8000
10000
12000
Population


AveOccup
0
5
10
15
20
25
30
35
40
AveOccup


Latitude
34
36
38
40
42
Latitude


Longitude
−124
−122
−120
−118
−116
−114
Longitude


Price
0
1
2
3
4
5
Price



t-test for mean equlity

In [ ]:
```
template.t_test(df)
```

```
t-test for equality of mean between all numric columns
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude', 'Price']
(MedInc,HouseAge) => t-value=-279.5948542458597, p-value=0.0
(MedInc,AveRooms) => t-value=-78.21416853964867, p-value=0.0
(MedInc,AveBedrms) => t-value=206.9102313281982, p-value=0.0
(MedInc,Population) => t-value=-192.25530815223107, p-value=0.0
(MedInc,AveOccup) => t-value=65.87596102010772, p-value=0.0
(MedInc,Latitude) => t-value=-1596.9347510407445, p-value=0.0
(MedInc,Longitude) => t-value=6425.875069211692, p-value=0.0
(MedInc,Price) => t-value=116.43880086922015, p-value=0.0
(HouseAge,AveRooms) => t-value=261.70230236111394, p-value=0.0
(HouseAge,AveBedrms) => t-value=314.2983389564446, p-value=0.0
(HouseAge,Population) => t-value=-188.86730703886778, p-value=0.0
(HouseAge,AveOccup) => t-value=292.92136324963724, p-value=0.0
(HouseAge,Latitude) => t-value=-78.45488482045317, p-value=0.0
(HouseAge,Longitude) => t-value=1669.7948111394871, p-value=0.0
(HouseAge,Price) => t-value=301.99414938383603, p-value=0.0
(AveRooms,AveBedrms) => t-value=293.05104601502404, p-value=0.0
(AveRooms,Population) => t-value=-192.04652834686283, p-value=0.0
(AveRooms,AveOccup) => t-value=159.17827524387445, p-value=0.0
(AveRooms,Latitude) => t-value=-1454.846100032601, p-value=0.0
(AveRooms,Longitude) => t-value=6208.67307697947, p-value=0.0
(AveRooms,Price) => t-value=200.97402856776444, p-value=0.0
(AveBedrms,Population) => t-value=-192.63378623625033, p-value=0.0
(AveBedrms,AveOccup) => t-value=-294.18643290311377, p-value=0.0
(AveBedrms,Latitude) => t-value=-2288.2907052989253, p-value=0.0
(AveBedrms,Longitude) => t-value=8505.439690398644, p-value=0.0
(AveBedrms,Price) => t-value=-116.18248777914768, p-value=0.0
(Population,AveOccup) => t-value=192.38445176308065, p-value=0.0
(Population,Latitude) => t-value=187.93051688407377, p-value=0.0
(Population,Longitude) => t-value=209.0626793635986, p-value=0.0
(Population,Price) => t-value=192.5004465223632, p-value=0.0
(AveOccup,Latitude) => t-value=-2051.809436533238, p-value=0.0
(AveOccup,Longitude) => t-value=8120.241710297278, p-value=0.0
(AveOccup,Price) => t-value=86.43671298050494, p-value=0.0
(Latitude,Longitude) => t-value=7603.4167115246955, p-value=0.0
(Latitude,Price) => t-value=1983.3543482951457, p-value=0.0
(Longitude,Price) => t-value=-7547.343682910688, p-value=0.0
```

Normality test of Numerical features

In [ ]:
```
template.Normality_test(df)
```

```
Normality Test for all numric columns
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude', 'Price']
```

```
Normaility Test for Column: MedInc
Statistics=0.895, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: HouseAge
Statistics=0.978, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: AveRooms
Statistics=0.541, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: AveBedrms
Statistics=0.229, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: Population
Statistics=0.776, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: AveOccup
Statistics=0.838, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: Latitude
Statistics=0.877, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: Longitude
Statistics=0.897, p_value=0.000
Sample does not look Gaussian (reject H0)
Normaility Test for Column: Price
Statistics=0.912, p_value=0.000
Sample does not look Gaussian (reject H0)
```

The results show that all variables are not Normally distributed

In [ ]:
```
template.chisquare_test(df)
```

```
(MedInc,AveOccup) => chisqr-value=244382377.67538816, p-value=0.0
Dependent (reject H0)
(MedInc,Latitude) => chisqr-value=10611706.219135297, p-value=1.0
Independent (H0 holds true)
(MedInc,Longitude) => chisqr-value=10220205.263067888, p-value=1.0
Independent (H0 holds true)
(MedInc,Price) => chisqr-value=53081044.06720008, p-value=0.0
Dependent (reject H0)
(HouseAge,AveRooms) => chisqr-value=981250.6528468417, p-value=0.9999198958265979
Independent (H0 holds true)
(HouseAge,AveBedrms) => chisqr-value=730495.6808569825, p-value=9.11006073407887e-08
Dependent (reject H0)
(HouseAge,Population) => chisqr-value=215597.48372112517, p-value=9.11437869129925e-185
Dependent (reject H0)
(HouseAge,AveOccup) => chisqr-value=953367.1276815209, p-value=0.9998156789347916
Independent (H0 holds true)
(HouseAge,Latitude) => chisqr-value=57456.98224336913, p-value=0.0
```

```
Dependent (reject H0)
(HouseAge,Longitude) => chisqr-value=54751.6102095066, p-value=2.3783358288975364e-304
Dependent (reject H0)
(HouseAge,Price) => chisqr-value=194350.62867331685, p-value=0.986843965157714
Independent (H0 holds true)
(AveRooms,AveBedrms) => chisqr-value=277784817.53173494, p-value=0.0
Dependent (reject H0)
(AveRooms,Population) => chisqr-value=75097637.86127418, p-value=2.968449695201019e-270
Dependent (reject H0)
(AveRooms,AveOccup) => chisqr-value=365242247.734166, p-value=0.0
Dependent (reject H0)
(AveRooms,Latitude) => chisqr-value=16591085.080512887, p-value=0.999999999999996
Independent (H0 holds true)
(AveRooms,Longitude) => chisqr-value=16167186.0088699, p-value=1.0
Independent (H0 holds true)
(AveRooms,Price) => chisqr-value=74845786.5569255, p-value=0.0
Dependent (reject H0)
(AveBedrms,Population) => chisqr-value=59278364.21813671, p-value=0.0
Dependent (reject H0)
(AveBedrms,AveOccup) => chisqr-value=270068736.5797934, p-value=0.0
Dependent (reject H0)
(AveBedrms,Latitude) => chisqr-value=12884700.073810939, p-value=0.0
Dependent (reject H0)
(AveBedrms,Longitude) => chisqr-value=12636273.58462036, p-value=0.0
Dependent (reject H0)
(AveBedrms,Price) => chisqr-value=55099505.89283504, p-value=0.0
Dependent (reject H0)
(Population,AveOccup) => chisqr-value=74205110.17817134, p-value=0.0
Dependent (reject H0)
(Population,Latitude) => chisqr-value=3179072.454392943, p-value=1.0
Independent (H0 holds true)
(Population,Longitude) => chisqr-value=3289595.808874925, p-value=1.2922430049846182e-64
Dependent (reject H0)
(Population,Price) => chisqr-value=14341245.59794925, p-value=1.0
Independent (H0 holds true)
(AveOccup,Latitude) => chisqr-value=15806931.133772364, p-value=1.0
Independent (H0 holds true)
(AveOccup,Longitude) => chisqr-value=15530703.122333555, p-value=1.0
Independent (H0 holds true)
(AveOccup,Price) => chisqr-value=72530760.17245917, p-value=1.2125768757457033e-260
Dependent (reject H0)
(Latitude,Longitude) => chisqr-value=1779012.7917422317, p-value=0.0
Dependent (reject H0)
(Latitude,Price) => chisqr-value=3257496.4985568863, p-value=1.0
Independent (H0 holds true)
(Longitude,Price) => chisqr-value=3020377.716576377, p-value=1.0
Independent (H0 holds true)
```

The results of Chi-square test show that the predictors in most cases are linearly independent. There is no tendency of strong depency

among the regressors (features), however they are correlated with the price.

Catagorical Columns plots and analysis

In [ ]:
```
template.stringcolanalysis(df)
```

```
<Figure size 800x1500 with 0 Axes>
```

There are no catagorical columns. Since there is no display.

Label encoding for Catagorical Columns

There are no catagorical columns. Since we do not need label encoding here.

In [ ]:
```
df1=df
df2 = template.onehotencoding(df1)
print(df2.columns)
print(df2.shape)
```

```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'Price'],
      dtype='object')
(20581, 9)
```

Feature Selection throgh Random Forest

We have 8 features and there is no need for features selection for this dataset.

In [ ]:
```
#df3=df2

#LABEL_COL="SalePrice"
#split=0.3
#random=SEED
#threshold=2#.999
#tree=500


#df4=template.RFfeatureimportance_reg(df3, LABEL_COL, split,random, tree,threshold)
```

In [16]:
```
df4=df3
```

ML Regressors Algorithm with Default model parameters (different regression pipelines)

We run the function of ML models with default model parameters and run a 5 fold cross validation for each pipeline/model.For this we calculate the mean, std and min score (=error) for every model. We fit the the data (features X and target y) using the default model parameters and apply to X_test to predict the SalePrice.

In [ ]:
```
#function Call

LABEL_COL="Price"
random=SEED
split=0.3

r1,r2,r3=runML_models_CV(df4,split,random, LABEL_COL)
```

CV Results

	mean_rmse	std	min_rmse
Linear	0.725712	0.012918	0.707697
Ridge	0.725714	0.012910	0.707706
Huber	1.392242	1.305235	0.713357
Lasso	1.157438	0.018388	1.124622
ElaNet	1.031410	0.017047	1.001338
BayesRidge	0.725715	0.012905	0.707713
RandomForestReg	0.518189	0.006387	0.508777
DecisionTreeReg	0.746432	0.016114	0.727960
SVReg	0.599072	0.013869	0.584322
GBR	0.535959	0.010557	0.522227
XGB	0.537417	0.009396	0.528607
LGBM	0.477754	0.005766	0.472440
ADA	0.511238	0.006556	0.504171

Training Data Fit Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
Linear	0.609346	0.609130	5.309688e-01	5.233576e-01	7.234346e-01
Ridge	0.609346	0.609129	5.309640e-01	5.233577e-01	7.234347e-01

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
Huber	0.587265	0.587036	5.149853e-01	5.529397e-01	7.435992e-01
Lasso	0.000000	-0.000554	9.142458e-01	1.339696e+00	1.157452e+00
ElaNet	0.205958	0.205518	8.134591e-01	1.063775e+00	1.031395e+00
BayesRidge	0.609346	0.609129	5.309609e-01	5.233580e-01	7.234348e-01
RandomForestReg	0.972354	0.972339	1.250525e-01	3.703738e-02	1.924510e-01
DecisionTreeReg	1.000000	1.000000	4.059599e-17	8.343137e-32	2.888449e-16
SVReg	0.742716	0.742573	3.856957e-01	3.446826e-01	5.870968e-01
GBR	0.806222	0.806114	3.557724e-01	2.596040e-01	5.095135e-01
XGB	0.805852	0.805745	3.562553e-01	2.600989e-01	5.099990e-01
LGBM	0.885268	0.885205	2.701006e-01	1.537058e-01	3.920533e-01
ADA	0.999732	0.999732	5.052327e-03	3.584293e-04	1.893223e-02

Test Data Prediction Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
Linear	0.595770	0.595247	0.527247	0.530568	0.728401
Ridge	0.595789	0.595266	0.527243	0.530544	0.728384
Huber	0.579848	0.579304	0.511959	0.551466	0.742608
Lasso	-0.000004	-0.001298	0.906201	1.312545	1.145664
ElaNet	0.206009	0.204981	0.805537	1.042145	1.020855
BayesRidge	0.595801	0.595278	0.527240	0.530528	0.728373
RandomForestReg	0.804337	0.804084	0.332573	0.256815	0.506770
DecisionTreeReg	0.598631	0.598111	0.469978	0.526813	0.725819
SVReg	0.733683	0.733338	0.397206	0.349552	0.591229
GBR	0.780356	0.780072	0.371436	0.288291	0.536928
XGB	0.781123	0.780839	0.371110	0.287285	0.535990
LGBM	0.840684	0.840477	0.307697	0.209109	0.457285

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
ADA	0.815673	0.815434	0.306024	0.241937	0.491871

GridSearch CV For ML Models

Here I apply the GridSearch and Hyper Parameters tuning with Preprocessing function. Using the gridsearch approach to select the optimal hyper parameters with the ML models. For this I calculate the mean, std and min score (=error) for every model. By this I get a first the optimal (best) estimate for the different regression pipelines on the training data. We fit the the data (features X_train and target y_train) using the optimal model parameters. The same optimal parameters are used to compute the prediction of SalePrice using the X_test data.

In [ ]:

```
SEED=42
```

In [ ]:

```
#function Call Grid Search
### call for Loop over GridSearchCV Pipelines
LABEL_COL="Price"

random=SEED
split=0.3


r4,r5,r6=run_MLAlg_gridCV(df4,split,random, LABEL_COL)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=0 will be ignored. Current value: bagging_freq=2
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=0 will be ignored. Current value: bagging_freq=2
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=0 will be ignored. Current value: bagging_freq=2
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=0 will be ignored. Current value: bagging_freq=2
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored. Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=2, subsample_freq=0 will be ignored. Current value: bagging_freq=2
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Grid Search CV Results
```

	mean_rmse	std	min_rmse
gscv_Linear	0.725456	0.007979	0.715727
gscv_Ridge	0.725455	0.007989	0.715707
gscv_Huber	0.730169	0.008779	0.717131
gscv_Lasso	0.725457	0.008000	0.715692
gscv_ElaNet	0.725659	0.012812	0.707959
gscv_RandomForestReg	0.539633	0.007129	0.529472
Decision Tree Reg	0.633762	0.017308	0.610154
gscv_SVReg	0.571103	0.012707	0.558982
gscv_GBR	0.475775	0.005284	0.469012
gscv_XGB	0.486805	0.009328	0.476951
gscv_LGBM	0.483168	0.007203	0.476242
gscv_ADA	0.527001	0.003914	0.524479

```
Training Data Fit Results
```

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
gscv_Linear	0.608031	0.607814	0.542702	0.525119	0.724651
gscv_Ridge	0.608030	0.607813	0.542690	0.525120	0.724652
gscv_Huber	0.602768	0.602548	0.535378	0.532171	0.729500
gscv_Lasso	0.608028	0.607810	0.542694	0.525124	0.724654
gscv_ElaNet	0.609308	0.609091	0.531020	0.523409	0.723470
gscv_RandomForestReg	0.847180	0.847096	0.311601	0.204732	0.452473
Decision Tree Reg	0.782212	0.782092	0.375818	0.291769	0.540157
gscv_SVReg	0.775595	0.775471	0.356341	0.300634	0.548301
gscv_GBR	0.896685	0.896628	0.257468	0.138411	0.372036
gscv_XGB	0.864043	0.863968	0.294192	0.182141	0.426780

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
gscv_LGBM	0.860473	0.860396	0.298518	0.186923	0.432346
gscv_ADA	0.889484	0.889422	0.339730	0.148058	0.384784

```
Test Data Prediction Results
```

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
gscv_Linear	0.597037	0.596515	0.546242	0.528906	0.727259
gscv_Ridge	0.597056	0.596534	0.546205	0.528881	0.727242
gscv_Huber	0.591763	0.591235	0.539157	0.535827	0.732002
gscv_Lasso	0.597103	0.596582	0.546167	0.528818	0.727199
gscv_ElaNet	0.596359	0.595837	0.527295	0.529795	0.727870
gscv_RandomForestReg	0.780584	0.780300	0.365805	0.287992	0.536649
Decision Tree Reg	0.686223	0.685817	0.443453	0.411845	0.641752
gscv_SVReg	0.757469	0.757156	0.377841	0.318331	0.564208
gscv_GBR	0.836882	0.836671	0.310462	0.214099	0.462708
gscv_XGB	0.826994	0.826770	0.322708	0.227077	0.476526
gscv_LGBM	0.830712	0.830493	0.322104	0.222197	0.471378
gscv_ADA	0.794205	0.793939	0.409885	0.270114	0.519725

Analysis with Baysian Regression Functions

I use seven different MCMC method in PYMC3's `sample` function to fit the models to X_taining data. Seven main functions are applied. The Baysian algorithms are applied to training data and the fitting features such as R-Suared, MAE and RMSE are reported for the training data. In the next predictions are computed using the optimal weights estimated during the training session and X_test data. The Baysian Algorithms employed to inference are:

1. NUTS Algorithm
2. Hamiltonian MC
3. Metropolis MC

4. Slice MC
5. DEMetropolis MC
6. MetropolisZ MC
7. Sequential MC (SMC)

The data is split into training and test (using training-test split to compute the predectionof these seven approaches.

Note: The trace, Posterior and Forests plot are not shown here. These are given in Appendix-I

In [17]:
```
### call for Min Max transformation
LABEL_COL="Price"
df5=template.MinMax_Transformation(df4, LABEL_COL)
```

In [18]:
```
df5.columns
```

Out[18]:
```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude', 'Price'],
      dtype='object')
```

In [ ]:
```
formula="Price ~ MedInc + HouseAge + AveRooms + AveBedrms + Population + AveOccup + Latitude + Longitude"
family=pm.glm.families.Normal()
prior = {"Intercept": pm.Normal.dist(mu=0, sd=10),
         "x1": pm.Normal.dist(mu=0, sd=10),
         "x2": pm.Normal.dist(mu=0, sd=10),
         "x3": pm.Normal.dist(mu=0, sd=10),
         "x4": pm.Normal.dist(mu=0, sd=10),
         "x5": pm.Normal.dist(mu=0, sd=10),
         "x6": pm.Normal.dist(mu=0, sd=10),
         "x7": pm.Normal.dist(mu=0, sd=10),
         "x8": pm.Normal.dist(mu=0, sd=10),
         }
prior_samples=1000
draws=5000
chains=5
tune=1000
target_accept=0.87
SEED=42

results1, results2=run_baysian_algorithms(df5,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=0)
```

```
============ NUTS_Reg ===========
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of the library we use for blas. If you use ATLAS, make sure to compile it with dynamics library.
```

```
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of the library we use for blas. If you
use ATLAS, make sure to compile it with dynamics library.
Sequential sampling (5 chains in 1 job)
NUTS: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
                                          100.00% [6000/6000 04:05<00:00 Sampling chain 0, 0 divergences]
                                          100.00% [6000/6000 02:03<00:00 Sampling chain 1, 4,472 divergences]
                                          100.00% [6000/6000 03:58<00:00 Sampling chain 2, 0 divergences]
                                          100.00% [6000/6000 03:48<00:00 Sampling chain 3, 0 divergences]
                                          100.00% [6000/6000 04:30<00:00 Sampling chain 4, 0 divergences]
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 1107 seconds.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 0.5776614842855722, but should be close to 0.8. Try to increa
se the number of tuning steps.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
The number of effective samples is smaller than 10% for some parameters.
==============================

===============Training Data Fit=======================
                                          100.00% [1000/1000 00:05<00:00]
===============Test Data Fit=======================
                                          100.00% [1000/1000 00:07<00:00]
============ HamiltonianMC_Reg ===========
Sequential sampling (5 chains in 1 job)
HamiltonianMC: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
                                          100.00% [6000/6000 02:45<00:00 Sampling chain 0, 0 divergences]
                                          100.00% [6000/6000 02:46<00:00 Sampling chain 1, 0 divergences]
                                          100.00% [6000/6000 03:09<00:00 Sampling chain 2, 0 divergences]
                                          100.00% [6000/6000 05:22<00:00 Sampling chain 3, 283 divergences]
                                          100.00% [6000/6000 06:37<00:00 Sampling chain 4, 18 divergences]
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 1241 seconds.
The acceptance probability does not match the target. It is 0.7555975368908672, but should be close to 0.65. Try to incre
ase the number of tuning steps.
The acceptance probability does not match the target. It is 0.7790599900377474, but should be close to 0.65. Try to incre
ase the number of tuning steps.
```

```
The acceptance probability does not match the target. It is 0.7901675992864043, but should be close to 0.65. Try to incre
ase the number of tuning steps.
There were 283 divergences after tuning. Increase `target_accept` or reparameterize.
There were 301 divergences after tuning. Increase `target_accept` or reparameterize.
The number of effective samples is smaller than 25% for some parameters.
==============================

===============Training Data Fit=======================
                                        100.00% [1000/1000 00:09<00:00]

===============Test Data Fit=======================
                                        100.00% [1000/1000 00:14<00:00]

============ Metropolis_Reg ===========
Sequential sampling (5 chains in 1 job)
CompoundStep
>Metropolis: [sd]
>Metropolis: [Longitude]
>Metropolis: [Latitude]
>Metropolis: [AveOccup]
>Metropolis: [Population]
>Metropolis: [AveBedrms]
>Metropolis: [AveRooms]
>Metropolis: [HouseAge]
>Metropolis: [MedInc]
>Metropolis: [Intercept]
                                        100.00% [6000/6000 01:56<00:00 Sampling chain 0, 0 divergences]

                                        100.00% [6000/6000 01:49<00:00 Sampling chain 1, 0 divergences]

                                        100.00% [6000/6000 01:30<00:00 Sampling chain 2, 0 divergences]

                                        100.00% [6000/6000 00:48<00:00 Sampling chain 3, 0 divergences]

                                        100.00% [6000/6000 00:48<00:00 Sampling chain 4, 0 divergences]

Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 412 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
==============================

===============Training Data Fit=======================
                                        100.00% [1000/1000 00:04<00:00]

===============Test Data Fit=======================
                                        100.00% [1000/1000 00:07<00:00]

============ Slice_Reg ===========
```

```
Sequential sampling (5 chains in 1 job)
CompoundStep
>Slice: [sd]
>Slice: [Longitude]
>Slice: [Latitude]
>Slice: [AveOccup]
>Slice: [Population]
>Slice: [AveBedrms]
>Slice: [AveRooms]
>Slice: [HouseAge]
>Slice: [MedInc]
>Slice: [Intercept]
```

100.00% [6000/6000 03:20<00:00 Sampling chain 0, 0 divergences]

100.00% [6000/6000 03:18<00:00 Sampling chain 1, 0 divergences]

100.00% [6000/6000 03:17<00:00 Sampling chain 2, 0 divergences]

100.00% [6000/6000 03:17<00:00 Sampling chain 3, 0 divergences]

100.00% [6000/6000 03:17<00:00 Sampling chain 4, 0 divergences]

```
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 991 seconds.
The estimated number of effective samples is smaller than 200 for some parameters.
==============================

===============Training Data Fit=======================
```

100.00% [1000/1000 00:04<00:00]

```
===============Test Data Fit=======================
```

100.00% [1000/1000 00:07<00:00]

```
============ DEMetropolis_Reg ===========
Population sampling (5 chains)
DEMetropolis: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
Chains are not parallelized. You can enable this by passing `pm.sample(cores=n)`, where n > 1.
```

100.00% [6000/6000 00:34<00:00]

```
Sampling 5 chains for 0 tune and 6_000 draw iterations (0 + 30_000 draws total) took 35 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
==============================

===============Training Data Fit=======================
```

100.00% [1000/1000 00:04<00:00]

```
===============Test Data Fit=======================
```

100.00% [1000/1000 00:08<00:00]

```
============ DEMetropolisZ_Reg ===========
Sequential sampling (5 chains in 1 job)
DEMetropolisZ: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
                                          100.00% [6000/6000 00:06<00:00 Sampling chain 0, 0 divergences]
                                          100.00% [6000/6000 00:06<00:00 Sampling chain 1, 0 divergences]
                                          100.00% [6000/6000 00:06<00:00 Sampling chain 2, 0 divergences]
                                          100.00% [6000/6000 00:06<00:00 Sampling chain 3, 0 divergences]
                                          100.00% [6000/6000 00:06<00:00 Sampling chain 4, 0 divergences]
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 33 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
==============================

===============Training Data Fit=======================
                                          100.00% [1000/1000 00:04<00:00]
===============Test Data Fit=======================
                                          100.00% [1000/1000 00:08<00:00]
============ SMC_Reg ===========
Initializing SMC sampler...
Sampling 2 chains in 1 job
Stage:   0 Beta: 0.000
Stage:   1 Beta: 0.000
Stage:   2 Beta: 0.000
Stage:   3 Beta: 0.000
Stage:   4 Beta: 0.000
Stage:   5 Beta: 0.000
Stage:   6 Beta: 0.000
Stage:   7 Beta: 0.001
Stage:   8 Beta: 0.001
Stage:   9 Beta: 0.001
Stage:  10 Beta: 0.002
Stage:  11 Beta: 0.002
Stage:  12 Beta: 0.002
Stage:  13 Beta: 0.003
Stage:  14 Beta: 0.004
Stage:  15 Beta: 0.005
Stage:  16 Beta: 0.006
Stage:  17 Beta: 0.007
Stage:  18 Beta: 0.009
Stage:  19 Beta: 0.011
Stage:  20 Beta: 0.013
```

```
Stage:  21 Beta: 0.016
Stage:  22 Beta: 0.020
Stage:  23 Beta: 0.024
Stage:  24 Beta: 0.030
Stage:  25 Beta: 0.037
Stage:  26 Beta: 0.046
Stage:  27 Beta: 0.058
Stage:  28 Beta: 0.073
Stage:  29 Beta: 0.091
Stage:  30 Beta: 0.113
Stage:  31 Beta: 0.138
Stage:  32 Beta: 0.167
Stage:  33 Beta: 0.201
Stage:  34 Beta: 0.239
Stage:  35 Beta: 0.279
Stage:  36 Beta: 0.322
Stage:  37 Beta: 0.366
Stage:  38 Beta: 0.411
Stage:  39 Beta: 0.459
Stage:  40 Beta: 0.511
Stage:  41 Beta: 0.563
Stage:  42 Beta: 0.615
Stage:  43 Beta: 0.670
Stage:  44 Beta: 0.725
Stage:  45 Beta: 0.780
Stage:  46 Beta: 0.836
Stage:  47 Beta: 0.887
Stage:  48 Beta: 0.938
Stage:  49 Beta: 0.990
Stage:  50 Beta: 1.000
Stage:   0 Beta: 0.000
Stage:   1 Beta: 0.000
Stage:   2 Beta: 0.000
Stage:   3 Beta: 0.000
Stage:   4 Beta: 0.000
Stage:   5 Beta: 0.000
Stage:   6 Beta: 0.000
Stage:   7 Beta: 0.001
Stage:   8 Beta: 0.001
Stage:   9 Beta: 0.001
Stage:  10 Beta: 0.001
Stage:  11 Beta: 0.002
Stage:  12 Beta: 0.002
Stage:  13 Beta: 0.003
Stage:  14 Beta: 0.003
Stage:  15 Beta: 0.004
Stage:  16 Beta: 0.005
Stage:  17 Beta: 0.006
Stage:  18 Beta: 0.007
```

```
Stage:  19 Beta: 0.009
Stage:  20 Beta: 0.011
Stage:  21 Beta: 0.014
Stage:  22 Beta: 0.017
Stage:  23 Beta: 0.021
Stage:  24 Beta: 0.026
Stage:  25 Beta: 0.031
Stage:  26 Beta: 0.038
Stage:  27 Beta: 0.046
Stage:  28 Beta: 0.057
Stage:  29 Beta: 0.070
Stage:  30 Beta: 0.085
Stage:  31 Beta: 0.103
Stage:  32 Beta: 0.125
Stage:  33 Beta: 0.150
Stage:  34 Beta: 0.180
Stage:  35 Beta: 0.216
Stage:  36 Beta: 0.255
Stage:  37 Beta: 0.293
Stage:  38 Beta: 0.332
Stage:  39 Beta: 0.374
Stage:  40 Beta: 0.419
Stage:  41 Beta: 0.468
Stage:  42 Beta: 0.518
Stage:  43 Beta: 0.568
Stage:  44 Beta: 0.623
Stage:  45 Beta: 0.675
Stage:  46 Beta: 0.728
Stage:  47 Beta: 0.781
Stage:  48 Beta: 0.835
Stage:  49 Beta: 0.890
Stage:  50 Beta: 0.944
Stage:  51 Beta: 1.000
Stage:  52 Beta: 1.000
==============================

===============Training Data Fit=======================
                                                100.00% [1000/1000 00:04<00:00]

===============Test Data Fit=======================
                                                100.00% [1000/1000 00:08<00:00]
```

Training Data Fit Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
NUTS	0.608756	0.608512	0.531562	0.524184	0.724006
HamiltonianMC	0.609093	0.608850	0.531033	0.523732	0.723693

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
Metropolis	0.599025	0.598775	0.532306	0.537221	0.732954
Slice	0.609158	0.608914	0.531011	0.523646	0.723634
DEMetropolis	-0.041078	-0.041727	0.868232	1.394823	1.181026
DEMetropolisZ	0.173783	0.173268	0.841502	1.106955	1.052119
SMC	-7.330868	-7.336062	2.464570	11.161593	3.340897

Test Data Prediction Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
NUTS	0.595663	0.595075	0.527410	0.530623	0.728439
HamiltonianMC	0.594907	0.594317	0.527849	0.531616	0.729120
Metropolis	0.582317	0.581709	0.529839	0.548138	0.740364
Slice	0.595236	0.594647	0.527780	0.531184	0.728824
DEMetropolis	-0.037268	-0.038778	0.861448	1.361238	1.166721
DEMetropolisZ	0.174650	0.173449	0.832114	1.083132	1.040736
SMC	-8.020284	-8.033414	2.446828	11.837597	3.440581

Results Discussion

The above three results suggest:

- In case of default parameter setup for best parameter selection the decision tree works well on the training data with highest R squared and lowest RMSE (1 and 2.8E -16 respectively). Random forest model performed equally well in case of model however its RMSE is higher than that of Decision tree. In case of default parameter setup on test data ***LGBM*** model is found to be the best among all models with lowest RMSE and highest R squared (0.4 and 0.8 respectively).
- In the case of Grid search cross validation Gradient boost regression seem to perform best in terms of lowest RMSE among all model parameters. The Grid search for best model parameters on training data find GBR to be the best performing model with the lowest root mean squared error and the best fit model among all others. Grid search on test data, however, finds ***GBR, XGB, and LGBM*** models to be performing equally well with almost same RMSE and R squared values.

- Bayesian method of model selection finds no difference among performance of NUTS, Hamiltonian MC , Metropolis and Slice models when using training data. The SMC, however, is the poorest performing model in this case. These results also stand true for the test data.

- The ***ML based models outperform the Baysian type models*** as Baysian models has higer RMSE and lower R-squared as compared to the ML based models. However, the results of ***Baysian are very close to the ML based to Linear, Ridge etc but wost than the tree ensamble algorithams***.

Appendix

In [19]:
```
#algo_list=get_supported_algorithms_reg()
formula="Price ~ MedInc + HouseAge + AveRooms + AveBedrms + Population + AveOccup + Latitude + Longitude"
family=pm.glm.families.Normal()
prior = {"Intercept": pm.Normal.dist(mu=0, sd=10),
        "x1": pm.Normal.dist(mu=0, sd=10),
        "x2": pm.Normal.dist(mu=0, sd=10),
        "x3": pm.Normal.dist(mu=0, sd=10),
        "x4": pm.Normal.dist(mu=0, sd=10),
        "x5": pm.Normal.dist(mu=0, sd=10),
        "x6": pm.Normal.dist(mu=0, sd=10),
        "x7": pm.Normal.dist(mu=0, sd=10),
        "x8": pm.Normal.dist(mu=0, sd=10),
        }
prior_samples=1000
draws=5000
chains=5
tune=1000
target_accept=0.87
SEED=42
#run_baysian_algorithms_plots
results3, results4=run_baysian_algorithms_plots(df5,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=
```

```
============ NUTS_Reg ===========
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of the library we use for blas. If you use ATLAS, make sure to compile it with dynamics library.
WARNING (theano.tensor.blas): We did not find a dynamic library in the library_dir of the library we use for blas. If you use ATLAS, make sure to compile it with dynamics library.
Sequential sampling (5 chains in 1 job)
NUTS: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
                                                    100.00% [6000/6000 04:02<00:00 Sampling chain 0, 0 divergences]

                                                    100.00% [6000/6000 02:04<00:00 Sampling chain 1, 4,472 divergences]
```

100.00% [6000/6000 03:55<00:00 Sampling chain 2, 0 divergences]

100.00% [6000/6000 03:47<00:00 Sampling chain 3, 0 divergences]

100.00% [6000/6000 03:45<00:00 Sampling chain 4, 0 divergences]

```
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 1056 seconds.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 0.5776614842855722, but should be close to 0.8. Try to increa
se the number of tuning steps.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
There were 4473 divergences after tuning. Increase `target_accept` or reparameterize.
The number of effective samples is smaller than 10% for some parameters.
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	3.640	0.079	3.488	3.784	0.003	0.002	781.0	1657.0	1.00
MedInc	6.463	0.073	6.331	6.605	0.003	0.002	802.0	1530.0	1.00
HouseAge	0.494	0.027	0.444	0.545	0.001	0.000	1517.0	2158.0	1.00
AveRooms	-17.187	0.982	-19.060	-15.383	0.033	0.024	879.0	2099.0	1.00
AveBedrms	26.226	1.193	23.957	28.394	0.043	0.030	783.0	1997.0	1.00
Population	-0.020	0.200	-0.380	0.363	0.007	0.005	883.0	3114.0	1.02
AveOccup	-4.187	0.618	-5.304	-3.018	0.022	0.016	797.0	2732.0	1.00
Latitude	-3.938	0.079	-4.083	-3.787	0.003	0.002	1005.0	1677.0	1.00
Longitude	-4.354	0.089	-4.518	-4.184	0.003	0.002	906.0	1347.0	1.01
sd	0.724	0.004	0.716	0.732	0.000	0.000	1021.0	2079.0	1.01

```
==============Training Data Fit=======================
```

100.00% [1000/1000 00:05<00:00]

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
==============Test Data Fit=======================
```

100.00% [1000/1000 00:07<00:00]

```
============ HamiltonianMC_Reg ===========
Sequential sampling (5 chains in 1 job)
HamiltonianMC: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
```

100.00% [6000/6000 02:58<00:00 Sampling chain 0, 0 divergences]

100.00% [6000/6000 02:53<00:00 Sampling chain 1, 0 divergences]

100.00% [6000/6000 02:50<00:00 Sampling chain 2, 0 divergences]

100.00% [6000/6000 02:40<00:00 Sampling chain 3, 283 divergences]

100.00% [6000/6000 02:52<00:00 Sampling chain 4, 18 divergences]

```
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 856 seconds.
The acceptance probability does not match the target. It is 0.7555975368908672, but should be close to 0.65. Try to incre
ase the number of tuning steps.
The acceptance probability does not match the target. It is 0.7790599900377474, but should be close to 0.65. Try to incre
ase the number of tuning steps.
The acceptance probability does not match the target. It is 0.7901675992864043, but should be close to 0.65. Try to incre
ase the number of tuning steps.
There were 283 divergences after tuning. Increase `target_accept` or reparameterize.
There were 301 divergences after tuning. Increase `target_accept` or reparameterize.
The number of effective samples is smaller than 25% for some parameters.
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	3.640	0.081	3.486	3.789	0.001	0.001	5348.0	9195.0	1.00
MedInc	6.464	0.074	6.324	6.602	0.001	0.001	7674.0	9472.0	1.00
HouseAge	0.494	0.031	0.436	0.553	0.000	0.000	19223.0	196.0	1.04
AveRooms	-17.210	0.991	-19.096	-15.389	0.014	0.010	4939.0	10030.0	1.00
AveBedrms	26.264	1.191	24.131	28.569	0.016	0.011	5488.0	11014.0	1.00
Population	-0.026	0.221	-0.432	0.396	0.001	0.014	25433.0	591.0	1.02
AveOccup	-4.193	0.611	-5.368	-3.062	0.004	0.003	20723.0	13155.0	1.00
Latitude	-3.939	0.082	-4.094	-3.789	0.001	0.001	6017.0	10316.0	1.00

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Longitude	-4.354	0.091	-4.527	-4.186	0.001	0.001	6315.0	10246.0	1.00
sd	0.724	0.004	0.716	0.732	0.000	0.000	21930.0	12835.0	1.00

```
===============Training Data Fit=======================
```
100.00% [1000/1000 00:04<00:00]

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
===============Test Data Fit=======================
```
100.00% [1000/1000 00:07<00:00]

```
============ Metropolis_Reg ===========
Sequential sampling (5 chains in 1 job)
CompoundStep
>Metropolis: [sd]
>Metropolis: [Longitude]
>Metropolis: [Latitude]
>Metropolis: [AveOccup]
>Metropolis: [Population]
>Metropolis: [AveBedrms]
>Metropolis: [AveRooms]
>Metropolis: [HouseAge]
>Metropolis: [MedInc]
>Metropolis: [Intercept]
```
100.00% [6000/6000 00:48<00:00 Sampling chain 0, 0 divergences]

100.00% [6000/6000 00:51<00:00 Sampling chain 1, 0 divergences]

100.00% [6000/6000 00:49<00:00 Sampling chain 2, 0 divergences]

100.00% [6000/6000 00:49<00:00 Sampling chain 3, 0 divergences]

100.00% [6000/6000 00:49<00:00 Sampling chain 4, 0 divergences]

```
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 248 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	1.593	0.860	0.405	3.541	0.376	0.283	5.0	10.0	4.15
MedInc	7.356	0.374	6.571	7.945	0.161	0.120	6.0	11.0	3.18
HouseAge	0.809	0.137	0.490	0.988	0.059	0.044	6.0	11.0	3.28
AveRooms	-24.708	3.172	-29.911	-18.220	1.311	0.978	6.0	18.0	2.25
AveBedrms	34.088	3.378	26.978	39.607	1.363	1.014	7.0	20.0	2.06
Population	1.119	0.531	-0.065	1.969	0.215	0.160	6.0	13.0	2.20
AveOccup	-4.943	0.697	-6.232	-3.614	0.136	0.097	26.0	135.0	1.12
Latitude	-2.001	0.815	-3.835	-0.810	0.356	0.268	5.0	11.0	4.13
Longitude	-2.159	0.922	-4.223	-0.791	0.403	0.302	5.0	11.0	4.06
sd	0.744	0.014	0.719	0.768	0.006	0.004	6.0	17.0	2.63

```
==========Training Data Fit=======================
```

100.00% [1000/1000 00:04<00:00]

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
==========Test Data Fit=======================
```

100.00% [1000/1000 00:07<00:00]

```
============ Slice_Reg ===========
Sequential sampling (5 chains in 1 job)
CompoundStep
>Slice: [sd]
>Slice: [Longitude]
>Slice: [Latitude]
>Slice: [AveOccup]
>Slice: [Population]
>Slice: [AveBedrms]
>Slice: [AveRooms]
>Slice: [HouseAge]
>Slice: [MedInc]
>Slice: [Intercept]
```

100.00% [6000/6000 03:13<00:00 Sampling chain 0, 0 divergences]

100.00% [6000/6000 03:16<00:00 Sampling chain 1, 0 divergences]

100.00% [6000/6000 03:09<00:00 Sampling chain 2, 0 divergences]

100.00% [6000/6000 03:11<00:00 Sampling chain 3, 0 divergences]

100.00% [6000/6000 03:16<00:00 Sampling chain 4, 0 divergences]

```
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 967 seconds.
The estimated number of effective samples is smaller than 200 for some parameters.
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	3.637	0.075	3.498	3.774	0.007	0.005	100.0	160.0	1.05
MedInc	6.464	0.075	6.328	6.607	0.005	0.004	225.0	665.0	1.01
HouseAge	0.495	0.027	0.443	0.545	0.001	0.001	526.0	2202.0	1.01
AveRooms	-17.197	1.043	-19.152	-15.272	0.070	0.050	222.0	594.0	1.01
AveBedrms	26.234	1.263	23.791	28.532	0.083	0.059	233.0	682.0	1.01
Population	-0.027	0.201	-0.410	0.350	0.004	0.003	2123.0	6153.0	1.00
AveOccup	-4.195	0.604	-5.314	-3.042	0.004	0.003	22078.0	17153.0	1.00
Latitude	-3.936	0.075	-4.076	-3.795	0.007	0.005	113.0	225.0	1.04
Longitude	-4.351	0.084	-4.503	-4.192	0.008	0.006	109.0	212.0	1.04
sd	0.724	0.004	0.716	0.731	0.000	0.000	23501.0	18366.0	1.00

```
===============Training Data Fit=======================
```

100.00% [1000/1000 00:04<00:00]

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
===============Test Data Fit=======================
```

100.00% [1000/1000 00:07<00:00]

```
============ DEMetropolis_Reg ===========
Population sampling (5 chains)
DEMetropolis: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
Chains are not parallelized. You can enable this by passing `pm.sample(cores=n)`, where n > 1.
```

100.00% [6000/6000 00:33<00:00]

```
Sampling 5 chains for 0 tune and 6_000 draw iterations (0 + 30_000 draws total) took 33 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	3.714	1.392	1.703	5.208	0.465	0.392	9.0	84.0	1.67
MedInc	3.672	2.107	-0.375	5.679	0.615	0.488	11.0	195.0	1.38
HouseAge	0.511	0.550	-0.029	1.508	0.143	0.103	13.0	147.0	1.33
AveRooms	-2.300	2.252	-5.207	1.162	0.826	0.625	8.0	130.0	1.71
AveBedrms	11.040	6.277	-1.747	17.075	1.410	1.012	33.0	107.0	1.43
Population	-2.070	3.924	-5.536	5.892	1.071	0.774	13.0	106.0	1.39
AveOccup	5.250	2.904	-0.331	8.499	0.624	0.447	30.0	81.0	1.22
Latitude	-3.454	1.528	-5.206	-0.698	0.570	0.458	8.0	71.0	1.70
Longitude	-3.831	1.736	-5.797	-0.741	0.598	0.513	9.0	76.0	1.69
sd	0.849	0.199	0.710	1.103	0.043	0.031	16.0	95.0	1.28

```
==============Training Data Fit=======================
                                        100.00% [1000/1000 00:04<00:00]

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
==============Test Data Fit=======================
                                        100.00% [1000/1000 00:07<00:00]

============ DEMetropolisZ_Reg ===========
Sequential sampling (5 chains in 1 job)
DEMetropolisZ: [sd, Longitude, Latitude, AveOccup, Population, AveBedrms, AveRooms, HouseAge, MedInc, Intercept]
                                        100.00% [6000/6000 00:06<00:00 Sampling chain 0, 0 divergences]

                                        100.00% [6000/6000 00:06<00:00 Sampling chain 1, 0 divergences]

                                        100.00% [6000/6000 00:06<00:00 Sampling chain 2, 0 divergences]

                                        100.00% [6000/6000 00:06<00:00 Sampling chain 3, 0 divergences]
```

100.00% [6000/6000 00:06<00:00 Sampling chain 4, 0 divergences]

```
Sampling 5 chains for 1_000 tune and 5_000 draw iterations (5_000 + 25_000 draws total) took 32 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	1.299	1.509	-1.372	3.756	0.538	0.395	8.0	11.0	1.64
MedInc	5.087	1.390	2.291	6.501	0.414	0.328	10.0	21.0	1.47
HouseAge	0.805	0.594	0.104	2.252	0.212	0.163	8.0	12.0	1.62
AveRooms	1.446	3.069	-3.489	6.244	1.337	1.003	6.0	13.0	2.92
AveBedrms	0.841	3.148	-3.773	5.173	1.311	0.978	7.0	27.0	2.00
Population	-1.704	1.551	-4.552	0.905	0.539	0.424	8.0	20.0	1.60
AveOccup	1.294	3.617	-1.908	9.252	1.589	1.194	6.0	13.0	2.79
Latitude	-1.105	1.594	-4.152	1.776	0.558	0.410	8.0	11.0	1.68
Longitude	-1.032	1.942	-4.530	2.670	0.690	0.507	8.0	12.0	1.61
sd	0.853	0.115	0.724	1.084	0.036	0.026	9.0	22.0	1.49

```
===============Training Data Fit=======================
```

100.00% [1000/1000 00:04<00:00]

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
===============Test Data Fit=======================
```

100.00% [1000/1000 00:07<00:00]

```
============ SMC_Reg ===========
Initializing SMC sampler...
Sampling 2 chains in 1 job
Stage:   0 Beta: 0.000
Stage:   1 Beta: 0.000
Stage:   2 Beta: 0.000
Stage:   3 Beta: 0.000
```

```
Stage:   4 Beta: 0.000
Stage:   5 Beta: 0.000
Stage:   6 Beta: 0.000
Stage:   7 Beta: 0.001
Stage:   8 Beta: 0.001
Stage:   9 Beta: 0.001
Stage:  10 Beta: 0.001
Stage:  11 Beta: 0.002
Stage:  12 Beta: 0.002
Stage:  13 Beta: 0.003
Stage:  14 Beta: 0.004
Stage:  15 Beta: 0.005
Stage:  16 Beta: 0.006
Stage:  17 Beta: 0.008
Stage:  18 Beta: 0.009
Stage:  19 Beta: 0.012
Stage:  20 Beta: 0.014
Stage:  21 Beta: 0.018
Stage:  22 Beta: 0.023
Stage:  23 Beta: 0.029
Stage:  24 Beta: 0.036
Stage:  25 Beta: 0.044
Stage:  26 Beta: 0.054
Stage:  27 Beta: 0.068
Stage:  28 Beta: 0.085
Stage:  29 Beta: 0.104
Stage:  30 Beta: 0.127
Stage:  31 Beta: 0.152
Stage:  32 Beta: 0.182
Stage:  33 Beta: 0.214
Stage:  34 Beta: 0.250
Stage:  35 Beta: 0.288
Stage:  36 Beta: 0.328
Stage:  37 Beta: 0.370
Stage:  38 Beta: 0.413
Stage:  39 Beta: 0.460
Stage:  40 Beta: 0.506
Stage:  41 Beta: 0.554
Stage:  42 Beta: 0.604
Stage:  43 Beta: 0.650
Stage:  44 Beta: 0.697
Stage:  45 Beta: 0.745
Stage:  46 Beta: 0.797
Stage:  47 Beta: 0.846
Stage:  48 Beta: 0.894
Stage:  49 Beta: 0.943
Stage:  50 Beta: 0.993
Stage:  51 Beta: 1.000
Stage:   0 Beta: 0.000
```

```
Stage:   1 Beta: 0.000
Stage:   2 Beta: 0.000
Stage:   3 Beta: 0.000
Stage:   4 Beta: 0.000
Stage:   5 Beta: 0.000
Stage:   6 Beta: 0.000
Stage:   7 Beta: 0.001
Stage:   8 Beta: 0.001
Stage:   9 Beta: 0.001
Stage:  10 Beta: 0.001
Stage:  11 Beta: 0.002
Stage:  12 Beta: 0.002
Stage:  13 Beta: 0.003
Stage:  14 Beta: 0.004
Stage:  15 Beta: 0.005
Stage:  16 Beta: 0.006
Stage:  17 Beta: 0.007
Stage:  18 Beta: 0.009
Stage:  19 Beta: 0.011
Stage:  20 Beta: 0.014
Stage:  21 Beta: 0.018
Stage:  22 Beta: 0.022
Stage:  23 Beta: 0.027
Stage:  24 Beta: 0.034
Stage:  25 Beta: 0.042
Stage:  26 Beta: 0.052
Stage:  27 Beta: 0.065
Stage:  28 Beta: 0.081
Stage:  29 Beta: 0.101
Stage:  30 Beta: 0.123
Stage:  31 Beta: 0.151
Stage:  32 Beta: 0.182
Stage:  33 Beta: 0.216
Stage:  34 Beta: 0.254
Stage:  35 Beta: 0.293
Stage:  36 Beta: 0.337
Stage:  37 Beta: 0.383
Stage:  38 Beta: 0.431
Stage:  39 Beta: 0.482
Stage:  40 Beta: 0.536
Stage:  41 Beta: 0.592
Stage:  42 Beta: 0.647
Stage:  43 Beta: 0.702
Stage:  44 Beta: 0.755
Stage:  45 Beta: 0.810
Stage:  46 Beta: 0.868
Stage:  47 Beta: 0.926
Stage:  48 Beta: 0.984
```

```
Stage:  49 Beta: 1.000
Got error No model on context stack. trying to find log_likelihood in translation.
==============================

Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	25.022	8.537	15.491	34.578	6.018	5.093	3.0	45.0	1.83
MedInc	-2.576	8.706	-12.296	7.138	6.137	5.194	3.0	42.0	1.83
HouseAge	-3.843	0.759	-4.946	-2.697	0.509	0.422	3.0	46.0	1.83
AveRooms	80.009	123.827	-58.707	216.722	87.295	73.881	3.0	46.0	1.83
AveBedrms	-115.867	158.237	-291.179	58.949	111.589	94.456	3.0	44.0	1.83
Population	-15.742	11.695	-30.159	-1.060	8.174	6.891	3.0	43.0	1.83
AveOccup	-2.361	26.251	-37.184	32.537	18.168	15.251	3.0	42.0	1.83
Latitude	-22.666	7.152	-30.864	-14.406	5.034	4.257	3.0	40.0	1.83
Longitude	-25.487	7.541	-34.160	-16.724	5.306	4.488	3.0	42.0	1.83
sd	7.321	0.491	6.802	7.832	0.347	0.294	3.0	45.0	1.83

```
===============Training Data Fit=======================
```

100.00% [1000/1000 00:05<00:00]

```
Got error No model on context stack. trying to find log_likelihood in translation.
Got error No model on context stack. trying to find log_likelihood in translation.
===============Test Data Fit=======================
```

100.00% [1000/1000 00:07<00:00]

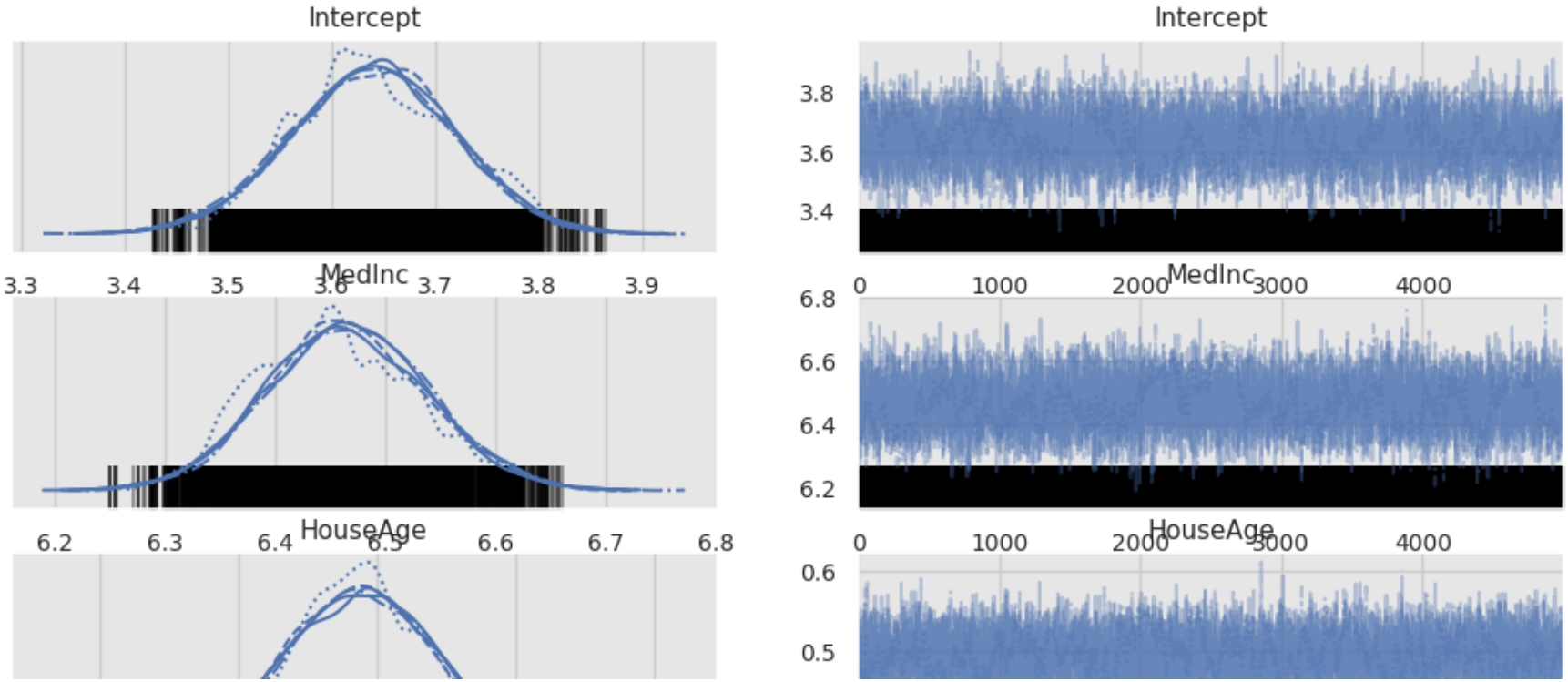
```
Training Data Fit Results
```

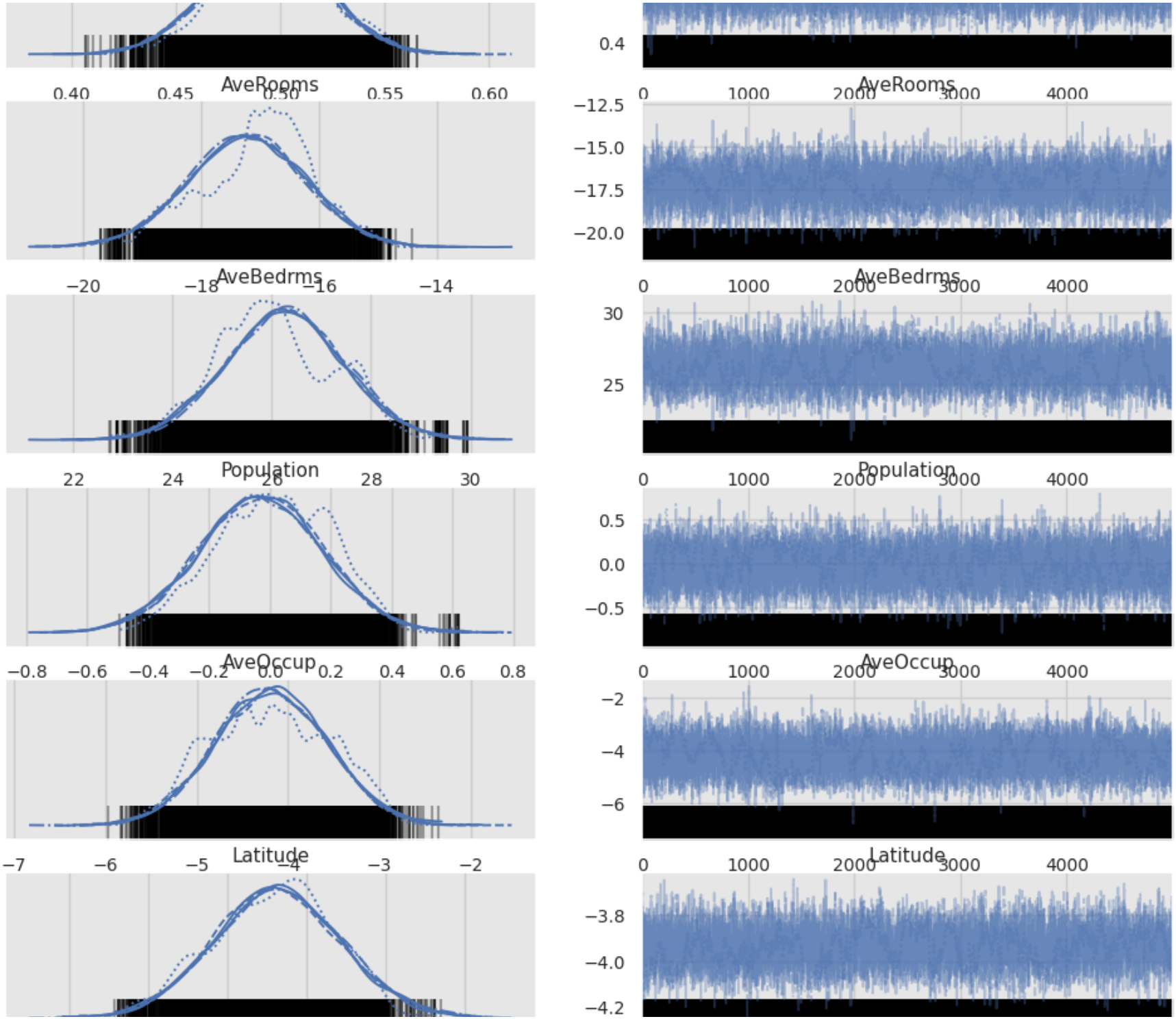
	R-Squared	Adj-R Squared	MAE	MSE	RMSE
NUTS	0.608756	0.608512	0.531562	0.524184	0.724006
HamiltonianMC	0.609093	0.608850	0.531033	0.523732	0.723693
Metropolis	0.599025	0.598775	0.532306	0.537221	0.732954
Slice	0.609158	0.608914	0.531011	0.523646	0.723634
DEMetropolis	-0.041078	-0.041727	0.868232	1.394823	1.181026

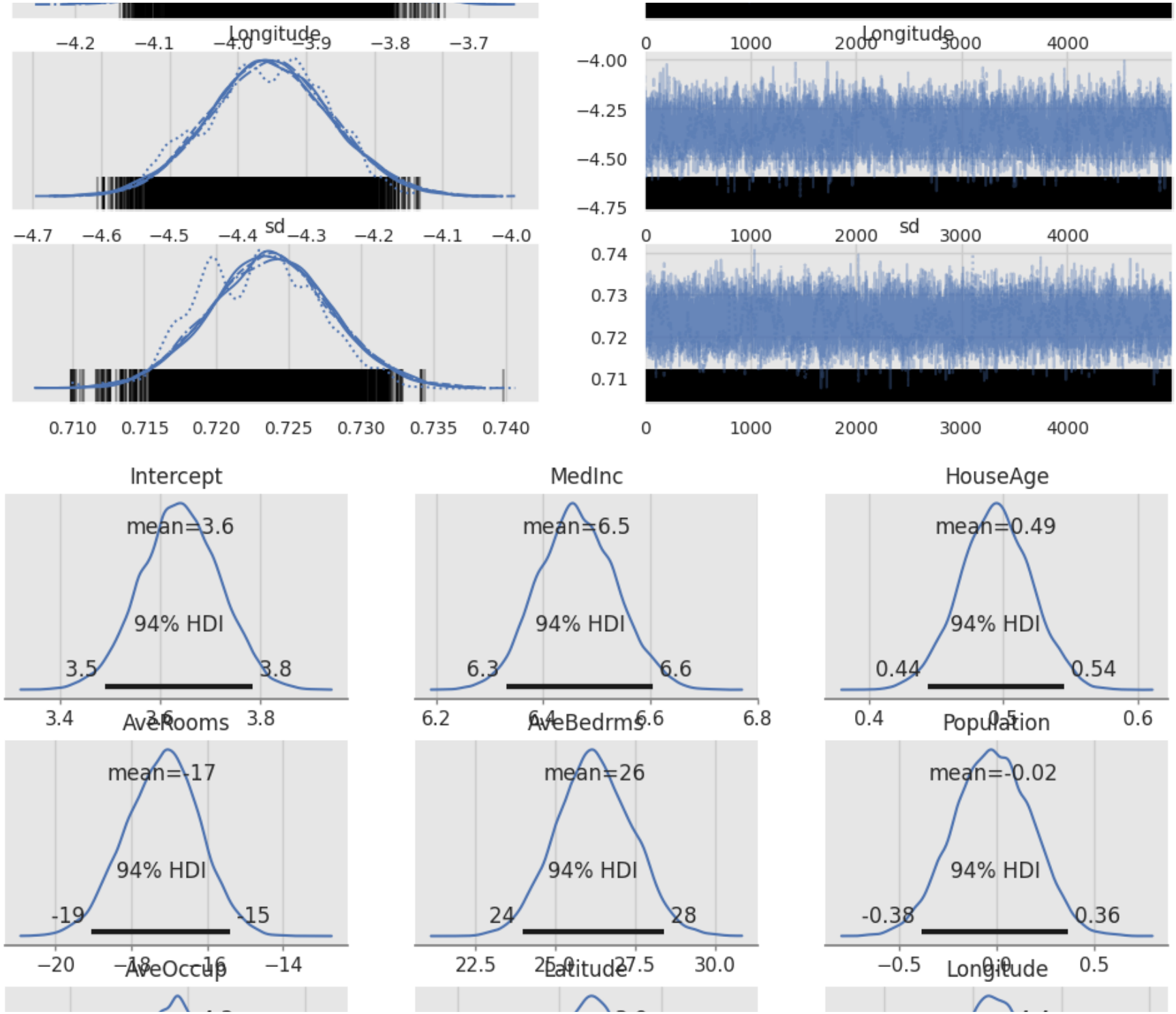
	R-Squared	Adj-R Squared	MAE	MSE	RMSE
DEMetropolisZ	0.173783	0.173268	0.841502	1.106955	1.052119
SMC	-0.562602	-0.563576	1.073220	2.093555	1.446912

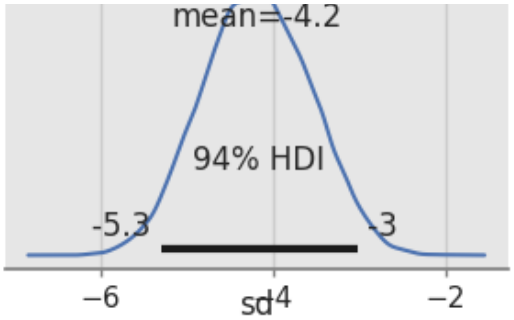
Test Data Prediction Results

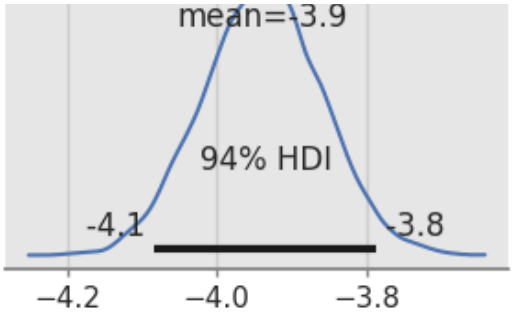
	R-Squared	Adj-R Squared	MAE	MSE	RMSE
NUTS	0.595663	0.595075	0.527410	0.530623	0.728439
HamiltonianMC	0.594907	0.594317	0.527849	0.531616	0.729120
Metropolis	0.582317	0.581709	0.529839	0.548138	0.740364
Slice	0.595236	0.594647	0.527780	0.531184	0.728824
DEMetropolis	-0.037268	-0.038778	0.861448	1.361238	1.166721
DEMetropolisZ	0.174650	0.173449	0.832114	1.083132	1.040736
SMC	-0.467537	-0.469673	1.046555	1.925894	1.387766

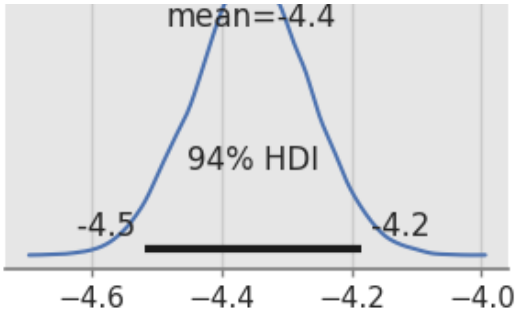


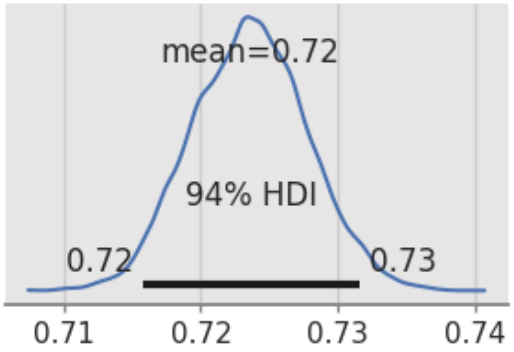

AveRooms
AveBedrms
Population
AveOccup
Latitude
0.40
0.45
0.50
0.55
0.60
−20
−18
−16
−14
22
24
26
28
30
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
−7
−6
−5
−4
−3
−2
0
1000
2000
3000
4000
0.4
−12.5
−15.0
−17.5
−20.0
30
25
0.5
0.0
−0.5
−2
−4
−6
−3.8
−4.0
−4.2

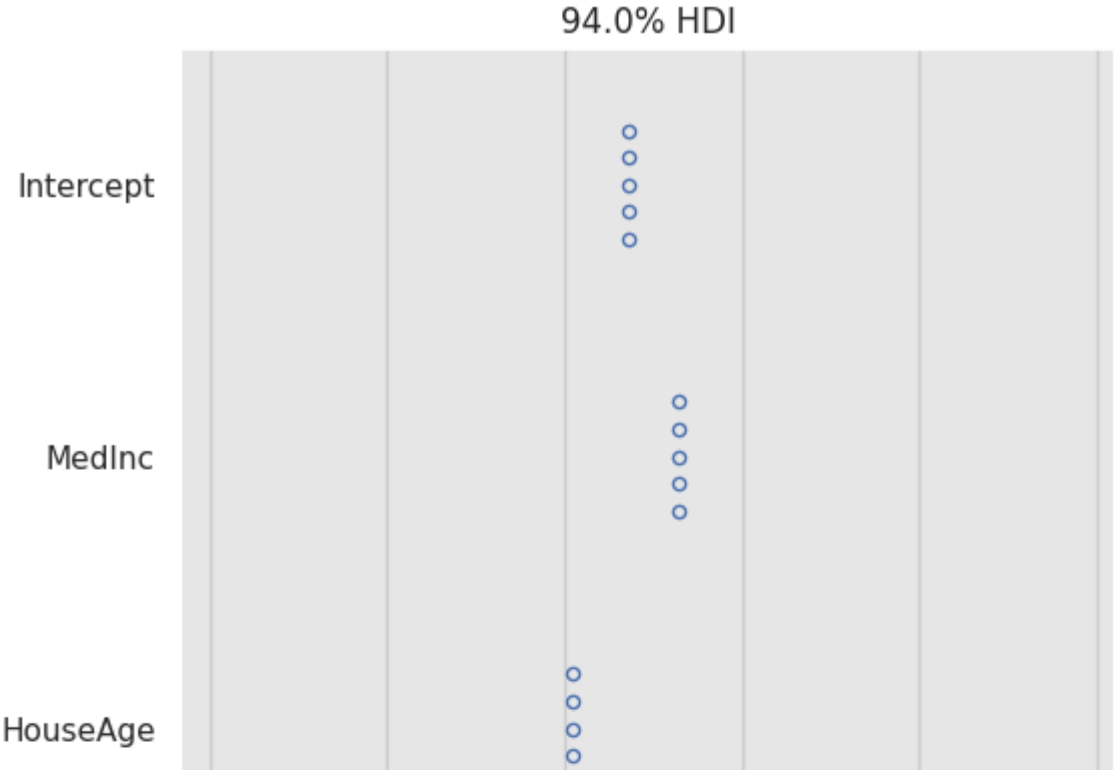

Longitude
−4.2 −4.1 −4.0 −3.9 −3.8 −3.7
sd
−4.7 −4.6 −4.5 −4.4 −4.3 −4.2 −4.1 −4.0
0.710 0.715 0.720 0.725 0.730 0.735 0.740
Longitude
0 1000 2000 3000 4000
−4.00 −4.25 −4.50 −4.75
sd
0 1000 2000 3000 4000
0.74 0.73 0.72 0.71
0 1000 2000 3000 4000
Intercept
mean=3.6
94% HDI
3.5 3.8
3.4 3.6 3.8
MedInc
mean=6.5
94% HDI
6.3 6.6
6.2 6.4 6.6 6.8
HouseAge
mean=0.49
94% HDI
0.44 0.54
0.4 0.5 0.6
AveRooms
mean=-17
94% HDI
-19 -15
−20 −18 −16 −14
AveBedrms
mean=26
94% HDI
24 28
22.5 25.0 27.5 30.0
Population
mean=-0.02
94% HDI
-0.38 0.36
−0.5 0.0 0.5
AveOccup
Latitude
Longitude

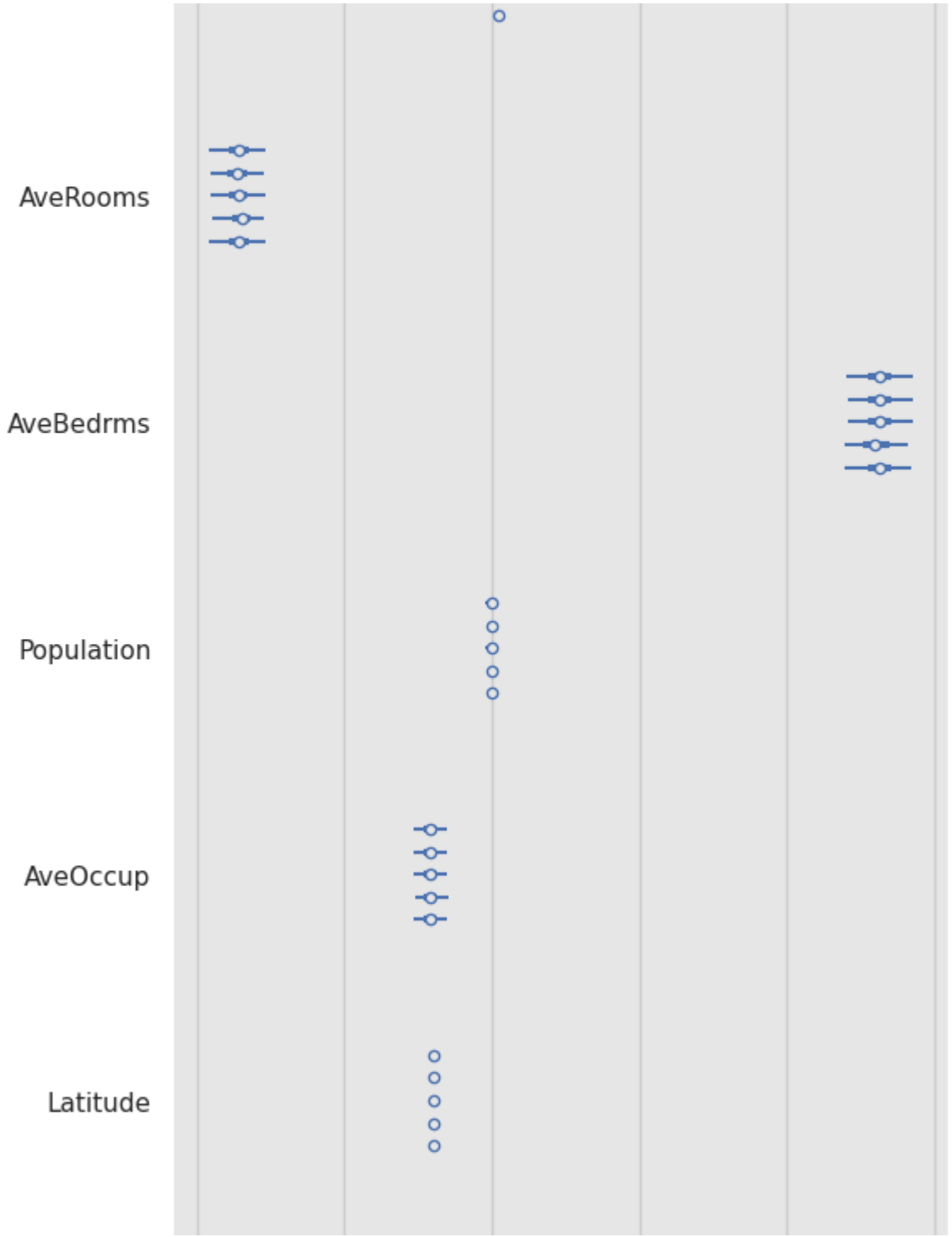

mean=-4.2
94% HDI
-5.3
-3
−6
−4
−2
sd

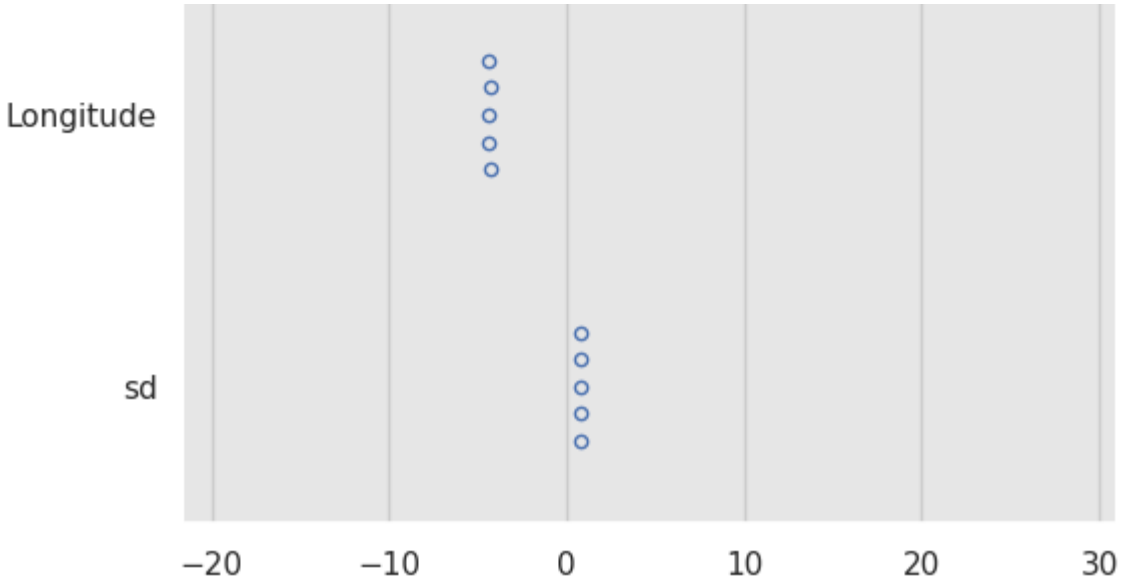

mean=-3.9
94% HDI
-4.1
-3.8
−4.2
−4.0
−3.8

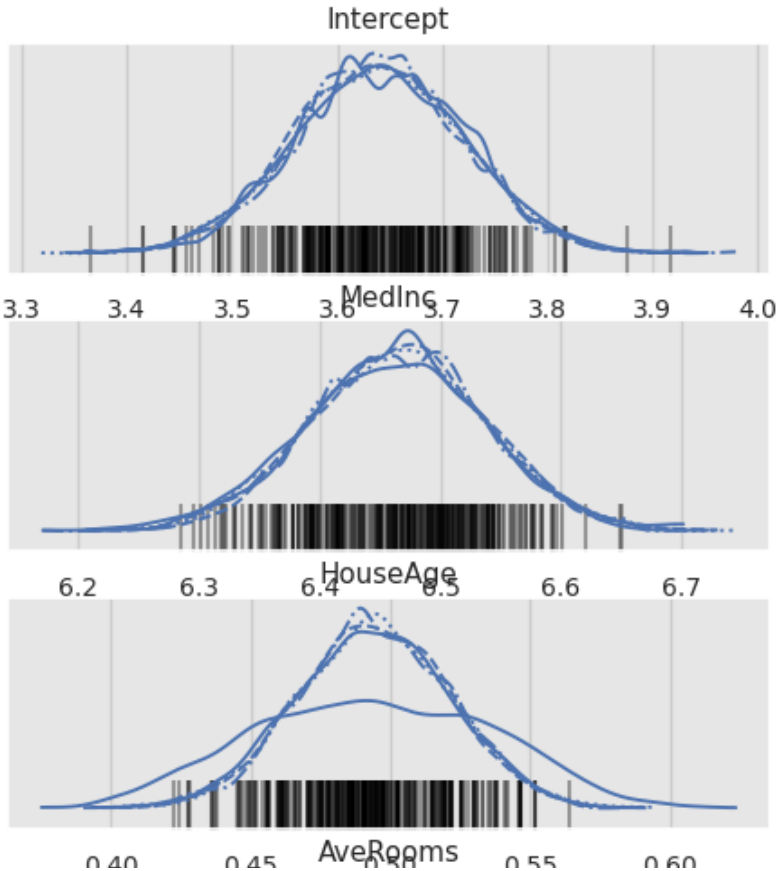

mean=-4.4
94% HDI
-4.5
-4.2
−4.6
−4.4
−4.2
−4.0

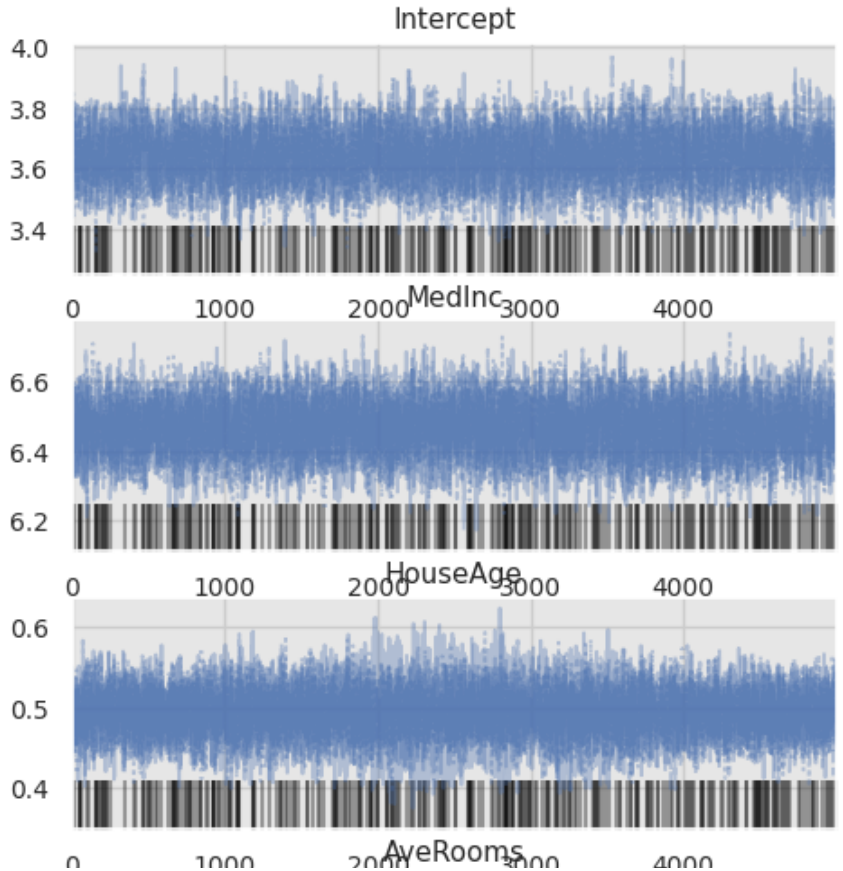

mean=0.72
94% HDI
0.72
0.73
0.71
0.72
0.73
0.74

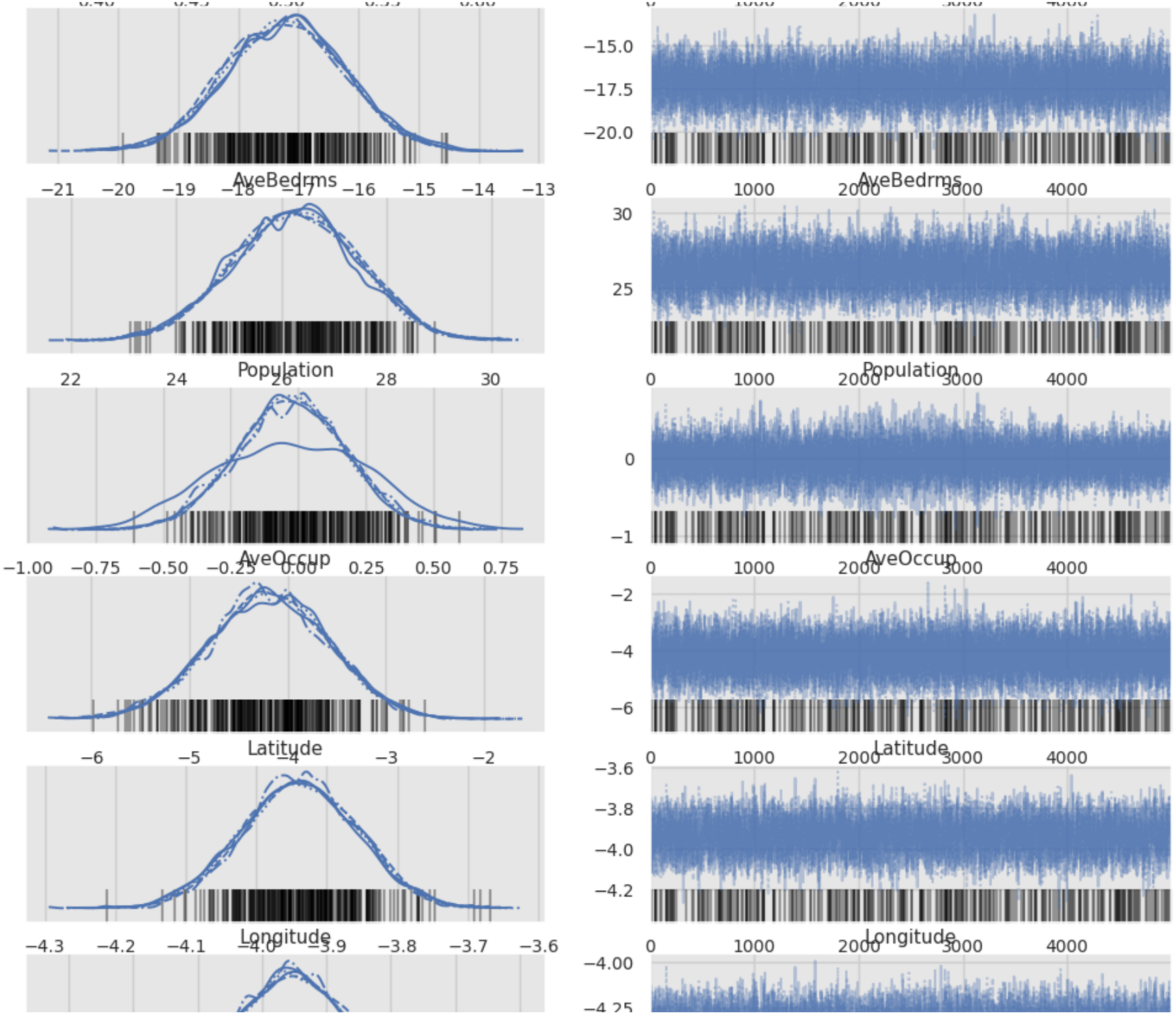

94.0% HDI
Intercept
MedInc
HouseAge

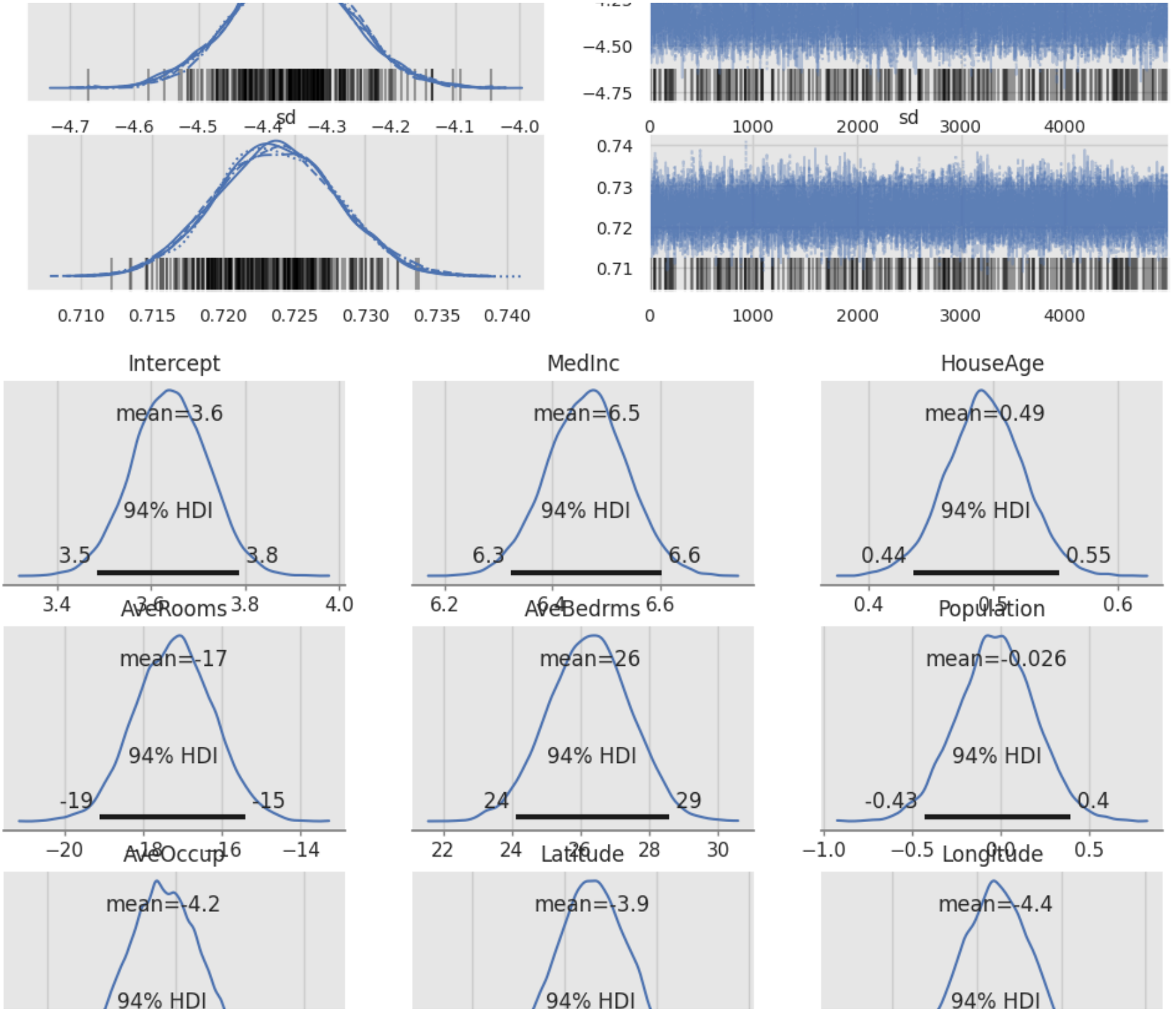

AveRooms
AveBedrms
Population
AveOccup
Latitude

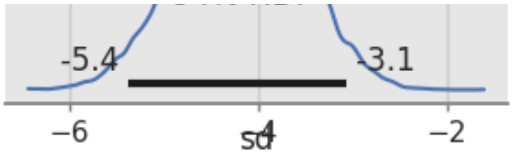

Longitude
sd
−20
−10
0
10
20
30

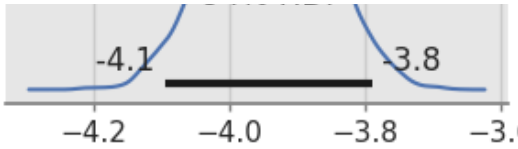

Intercept
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4.0
MedInc
6.2
6.3
6.4
6.5
6.6
6.7
HouseAge
0.40
0.45
0.55
0.60
AveRooms

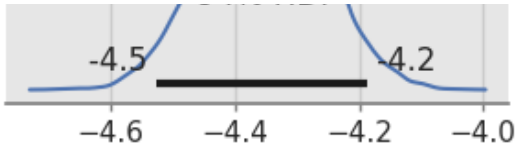

Intercept
4.0
3.8
3.6
3.4
0
1000
2000
3000
4000
MedInc
6.6
6.4
6.2
HouseAge
0.6
0.5
0.4
AveRooms

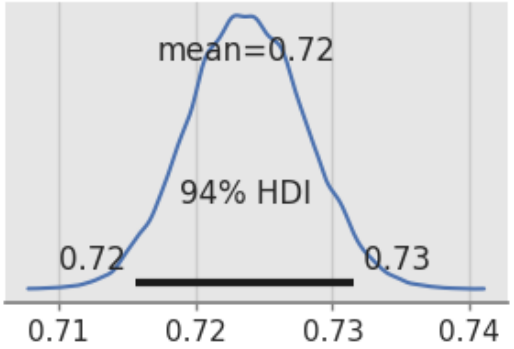

AveBedrms
Population
AveOccup
Latitude
Longitude

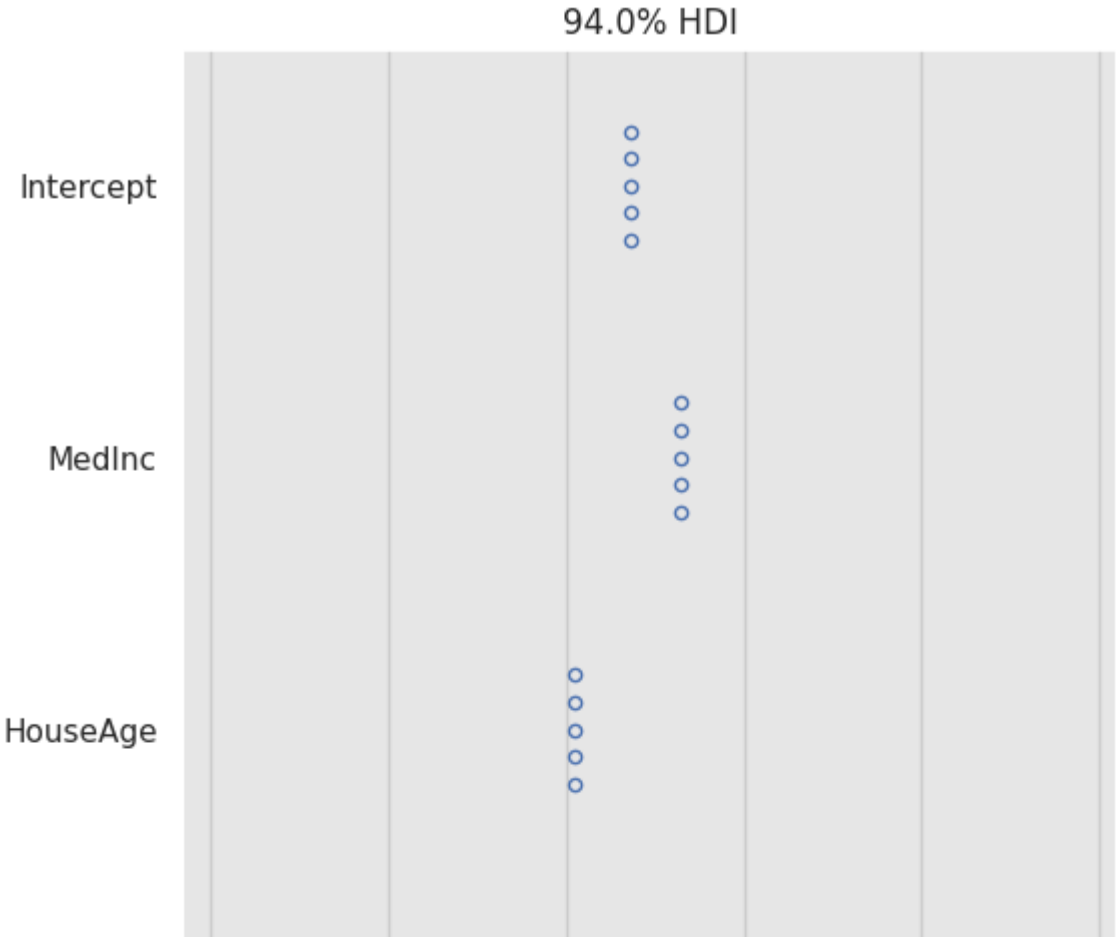

sd
−4.7 −4.6 −4.5 −4.4 −4.3 −4.2 −4.1 −4.0
0.710 0.715 0.720 0.725 0.730 0.735 0.740
−4.50
−4.75
0 1000 2000 3000 4000
sd
0.74
0.73
0.72
0.71
0 1000 2000 3000 4000
Intercept
mean=3.6
94% HDI
3.5
3.8
3.4 3.6 3.8 4.0
MedInc
mean=6.5
94% HDI
6.3
6.6
6.2 6.4 6.6
HouseAge
mean=0.49
94% HDI
0.44
0.55
0.4 0.5 0.6
AveRooms
mean=-17
94% HDI
-19
-15
−20 −18 −16 −14
AveBedrms
mean=26
94% HDI
24
29
22 24 26 28 30
Population
mean=-0.026
94% HDI
-0.43
0.4
−1.0 −0.5 0.0 0.5
AveOccup
mean=-4.2
94% HDI
Latitude
mean=-3.9
94% HDI
Longitude
mean=-4.4
94% HDI

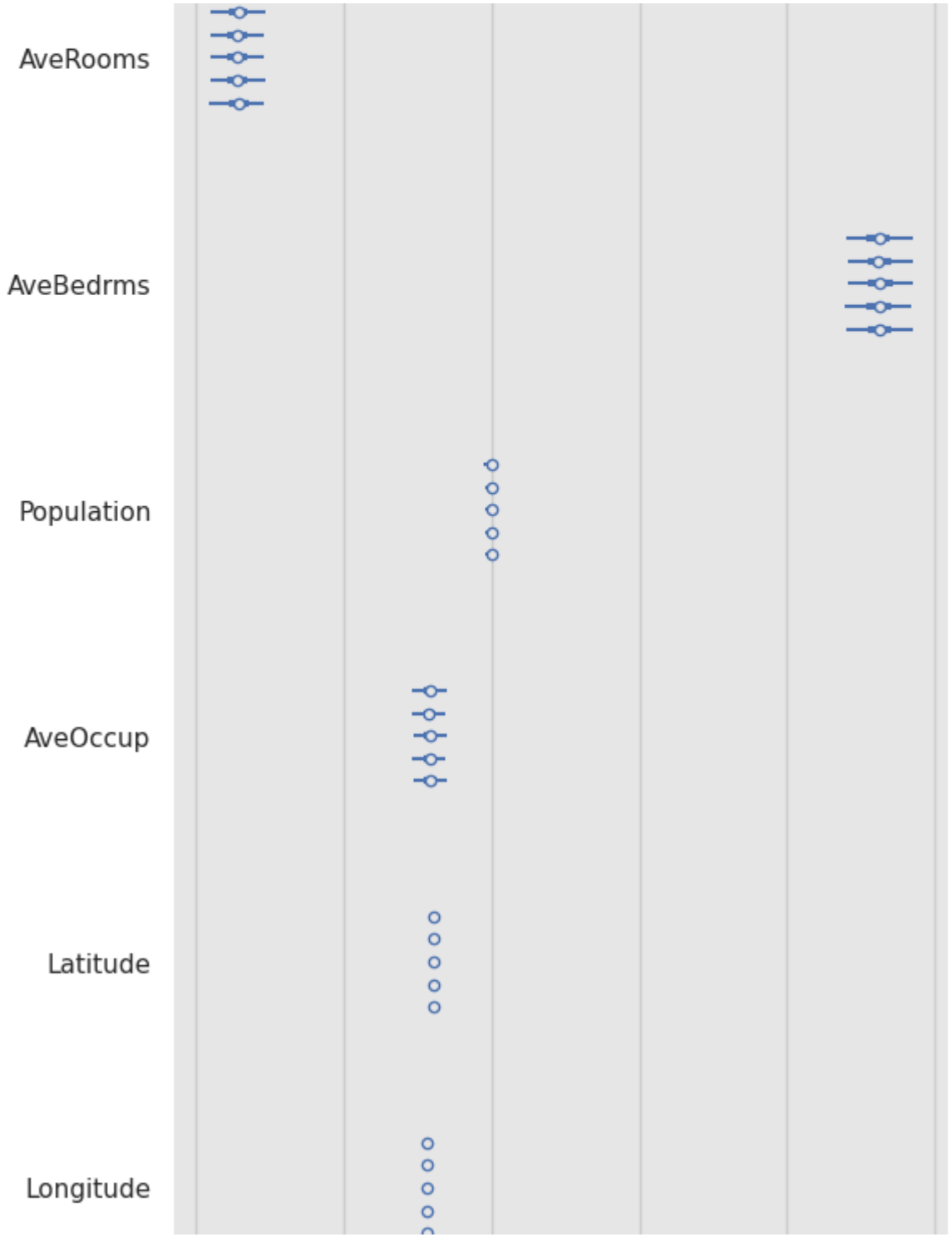

-5.4
-3.1
−6
−4
−2
sd

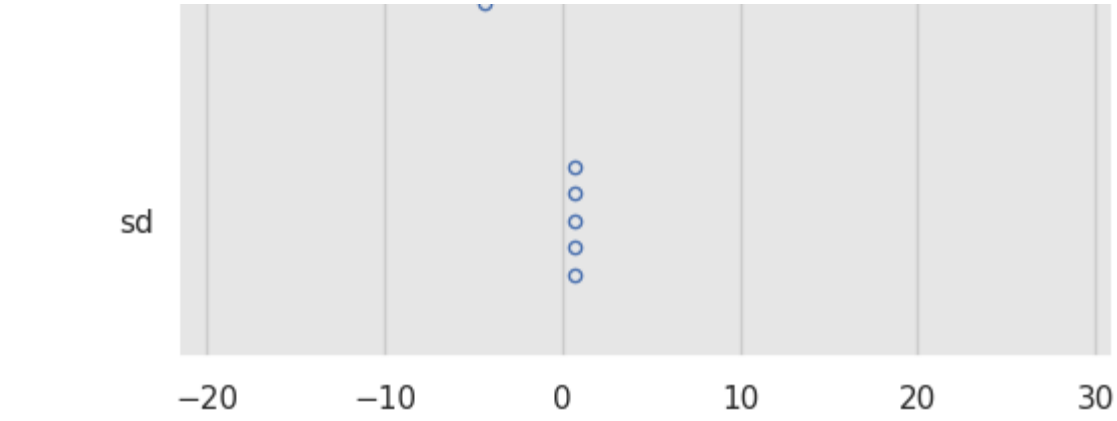

-4.1
-3.8
−4.2
−4.0
−3.8
−3.6

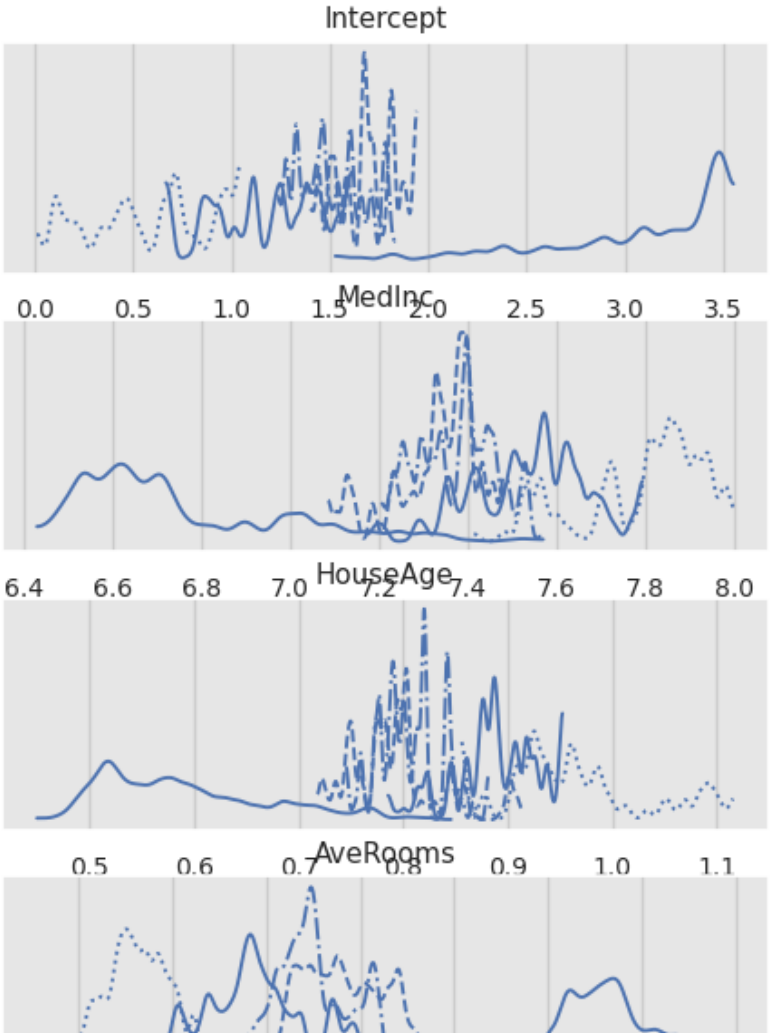

-4.5
-4.2
−4.6
−4.4
−4.2
−4.0

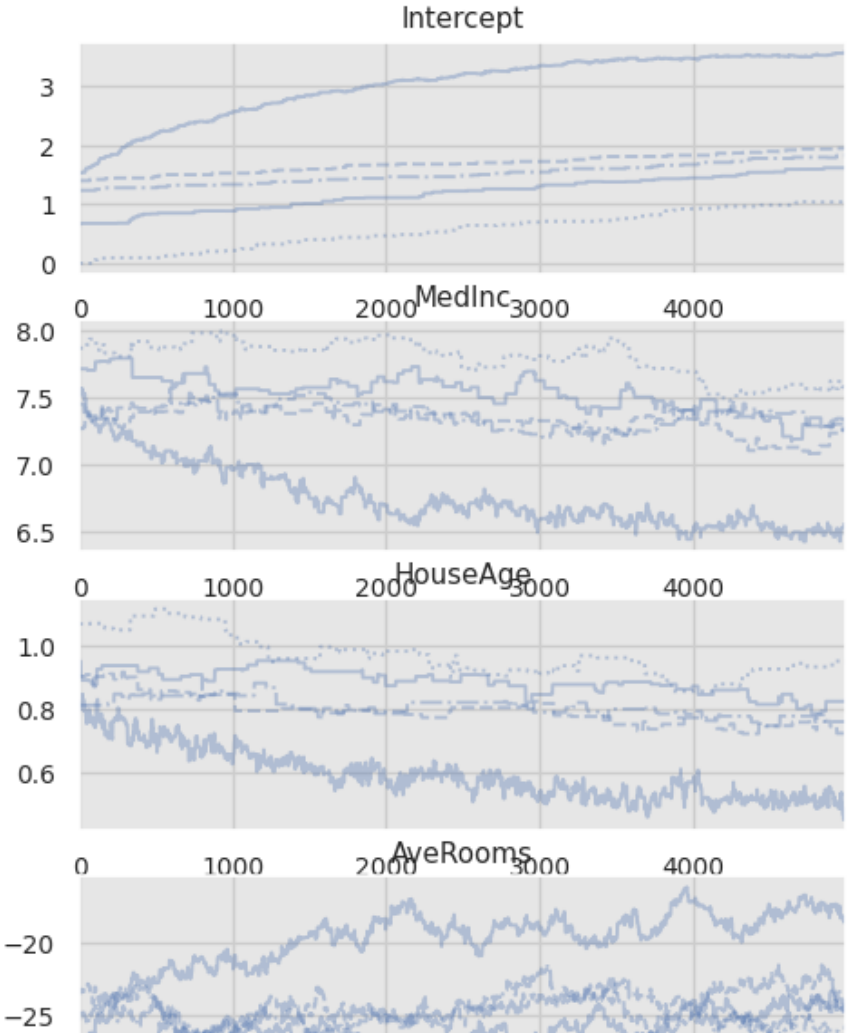

mean=0.72
94% HDI
0.72
0.73
0.71
0.72
0.73
0.74

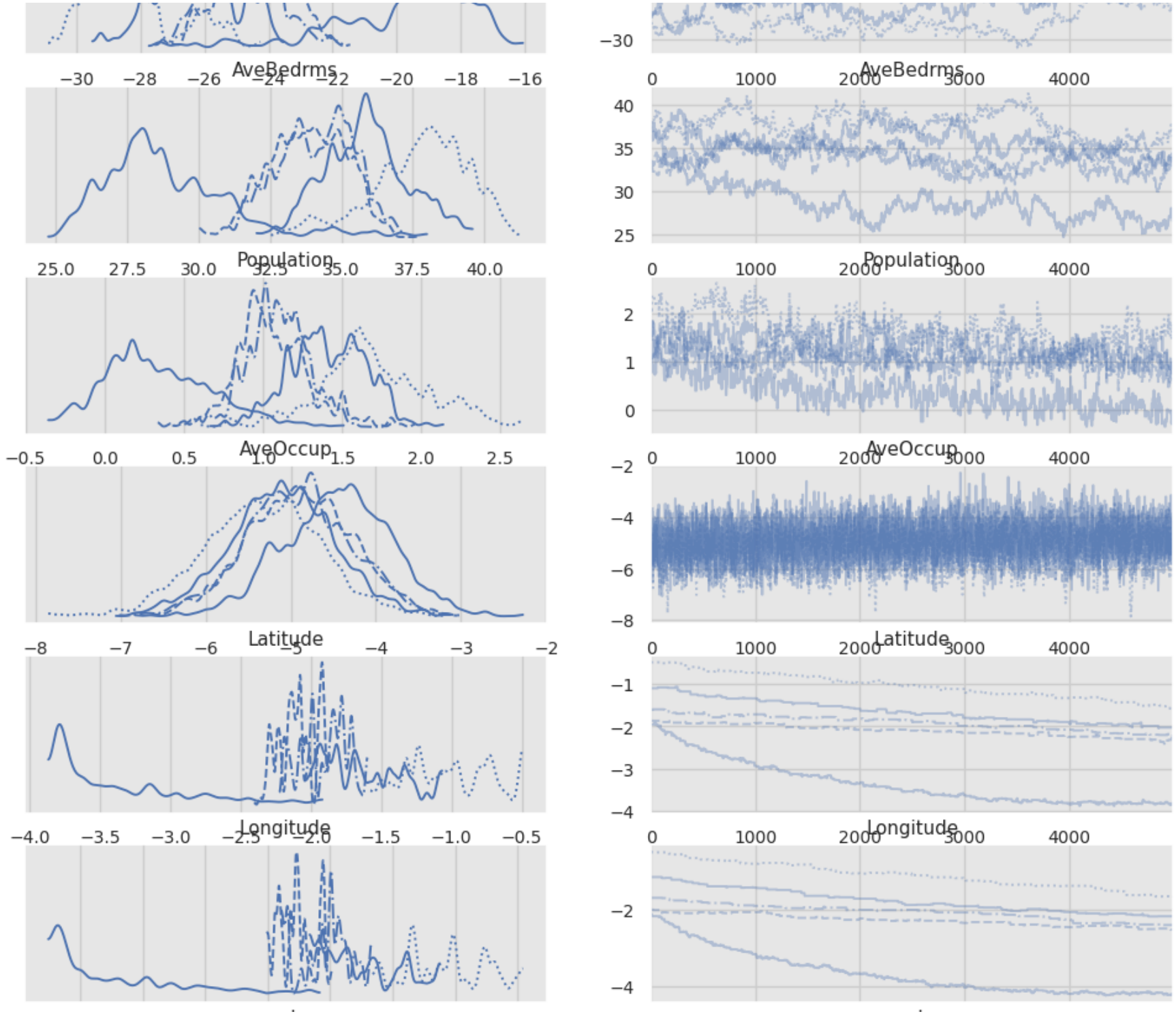

94.0% HDI
Intercept
MedInc
HouseAge

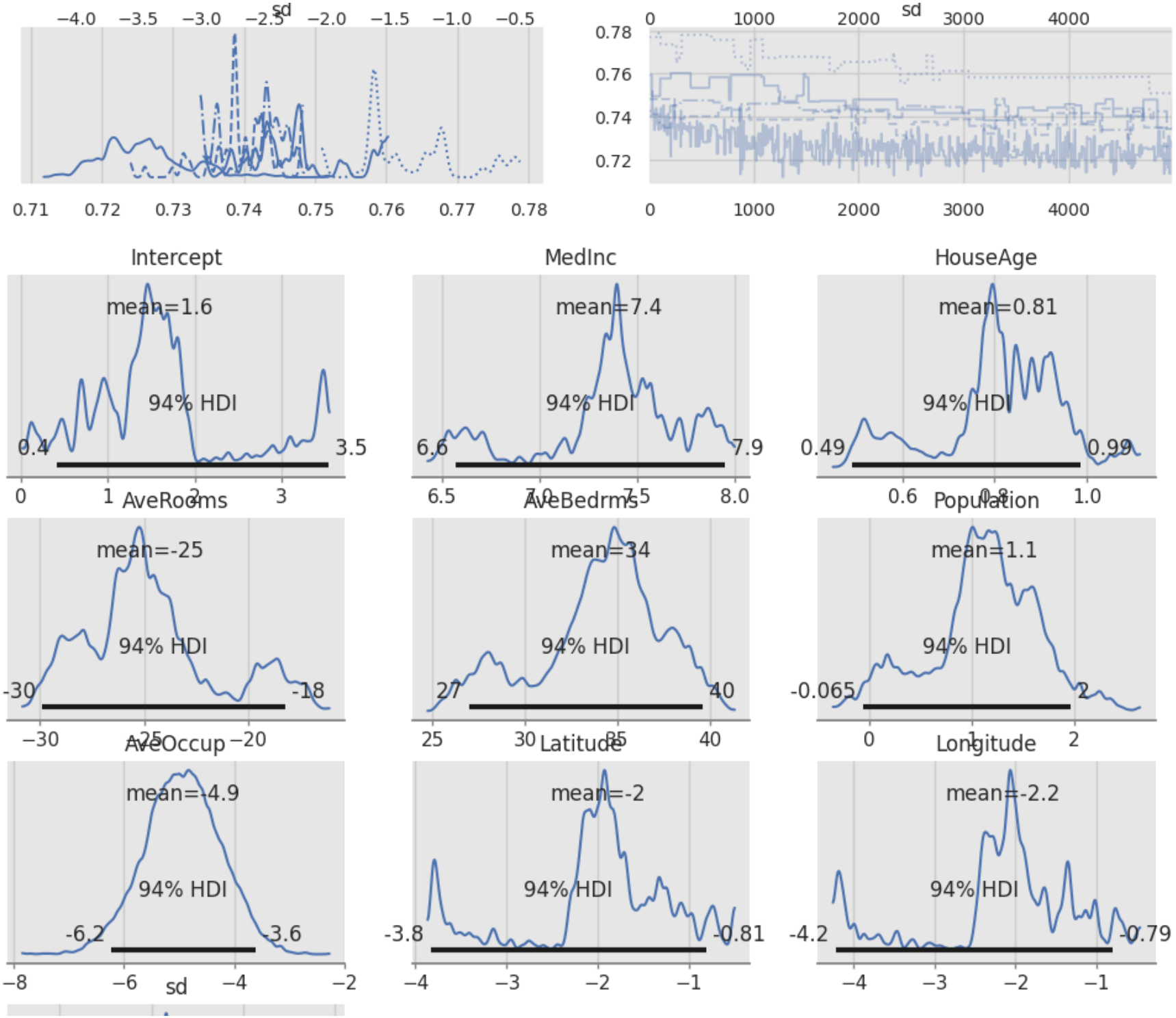

AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude

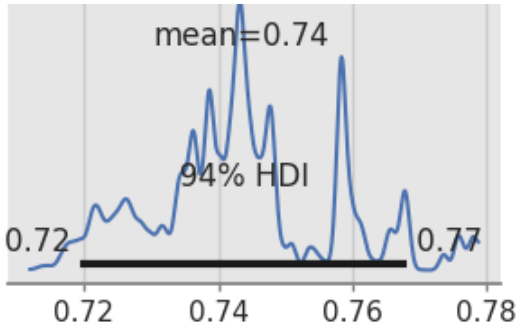

sd
−20
−10
0
10
20
30

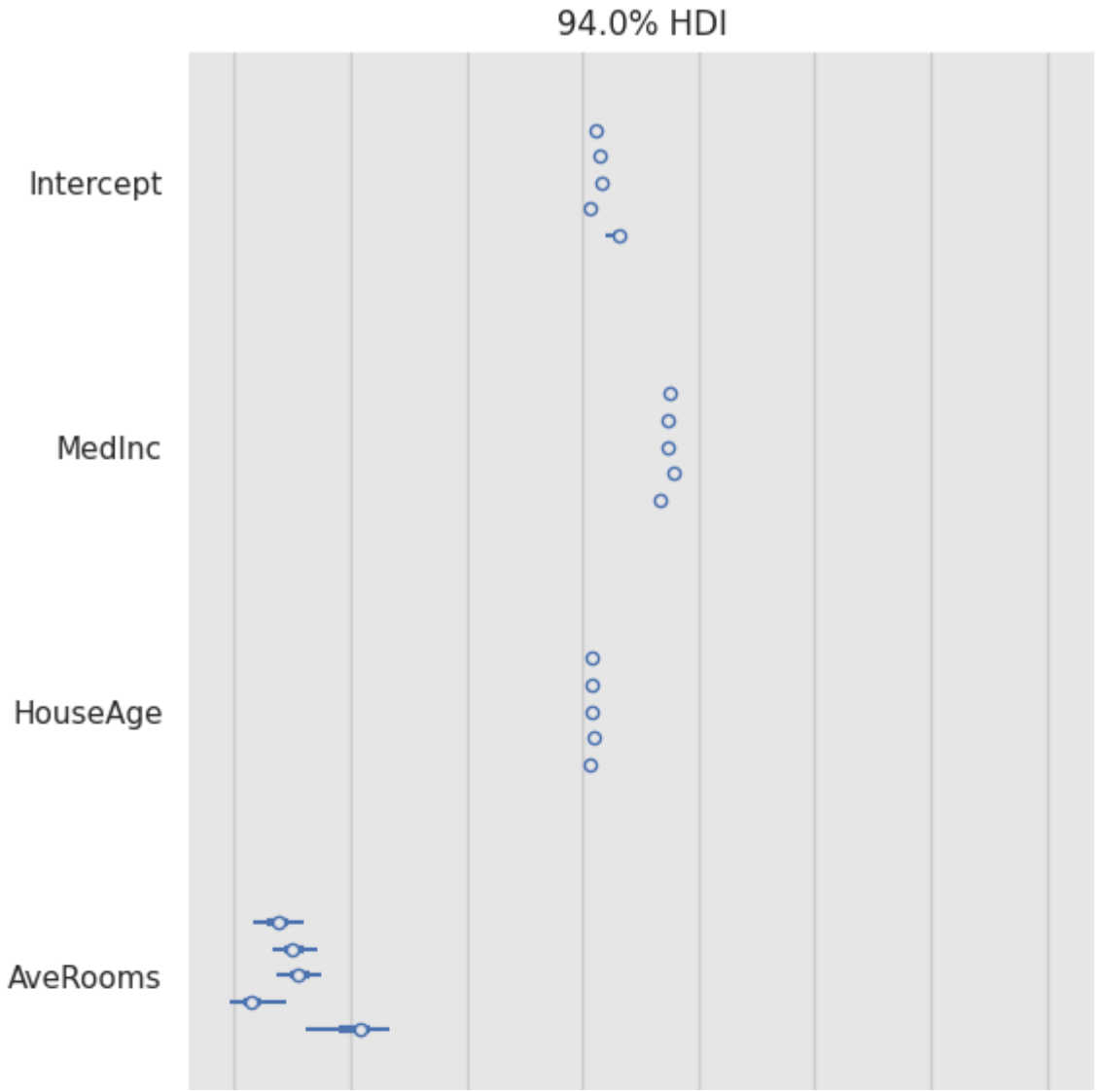

Intercept
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
MedInc
6.4
6.6
6.8
7.0
7.2
7.4
7.6
7.8
8.0
HouseAge
0.5
0.6
0.7
0.8
0.9
1.0
1.1
AveRooms


Intercept
3
2
1
0
0
1000
2000
3000
4000
MedInc
8.0
7.5
7.0
6.5
HouseAge
1.0
0.8
0.6
AveRooms
−20
−25

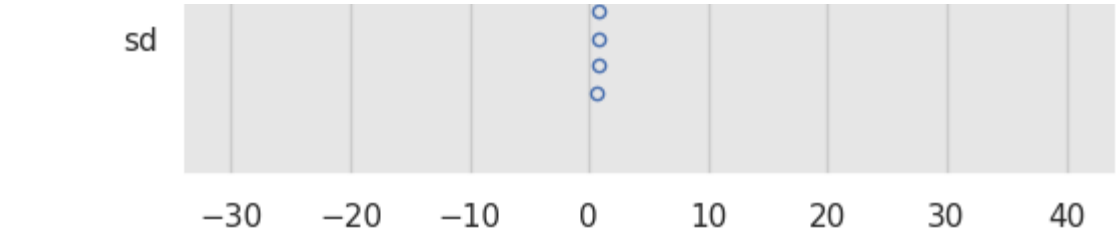

AveBedrms
Population
AveOccup
Latitude
Longitude

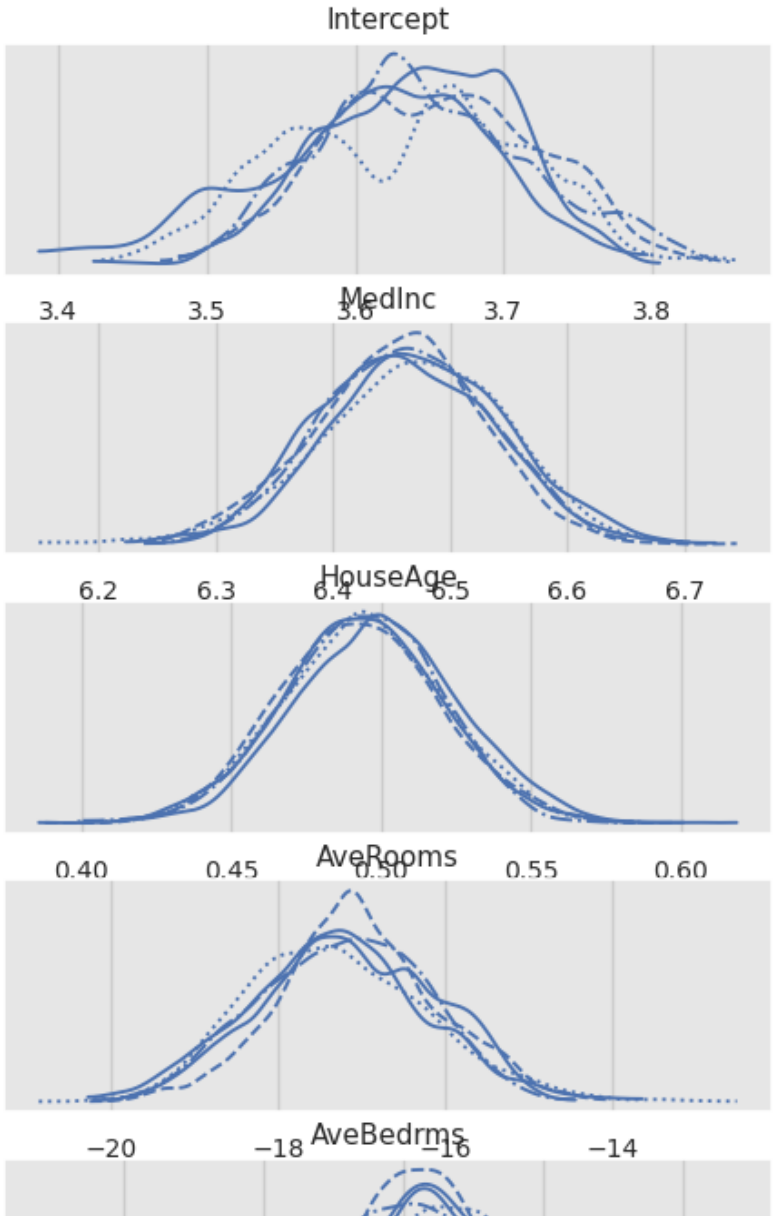

sd
sd
Intercept
mean=1.6
94% HDI
0.4
3.5
MedInc
mean=7.4
94% HDI
6.6
7.9
HouseAge
mean=0.81
94% HDI
0.49
0.99
AveRooms
mean=-25
94% HDI
-30
-18
AveBedrms
mean=34
94% HDI
27
40
Population
mean=1.1
94% HDI
-0.065
2
AveOccup
mean=-4.9
94% HDI
-6.2
-3.6
Latitude
mean=-2
94% HDI
-3.8
-0.81
Longitude
mean=-2.2
94% HDI
-4.2
-0.79
sd

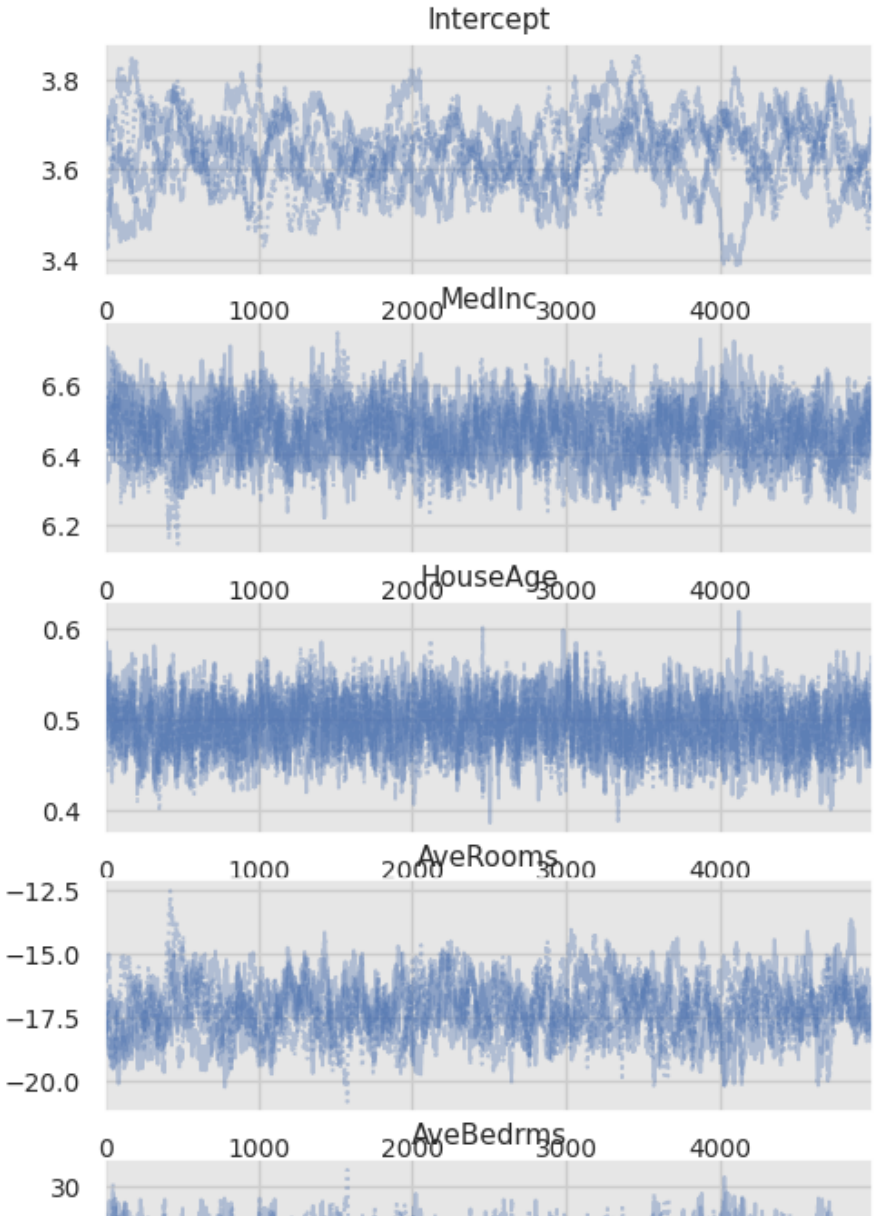

mean=0.74
94% HDI
0.72
0.77
0.72
0.74
0.76
0.78

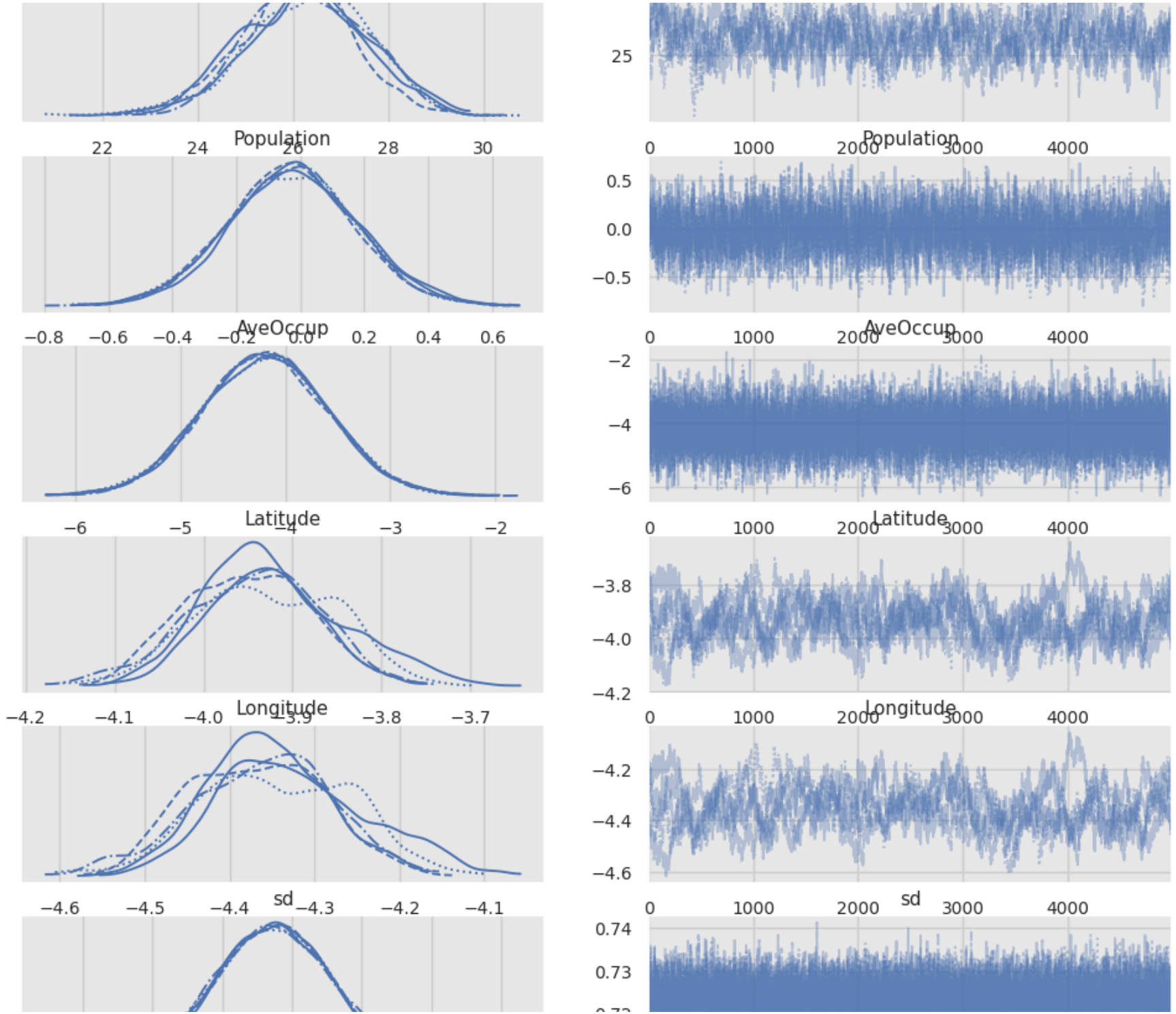

94.0% HDI
Intercept
MedInc
HouseAge
AveRooms

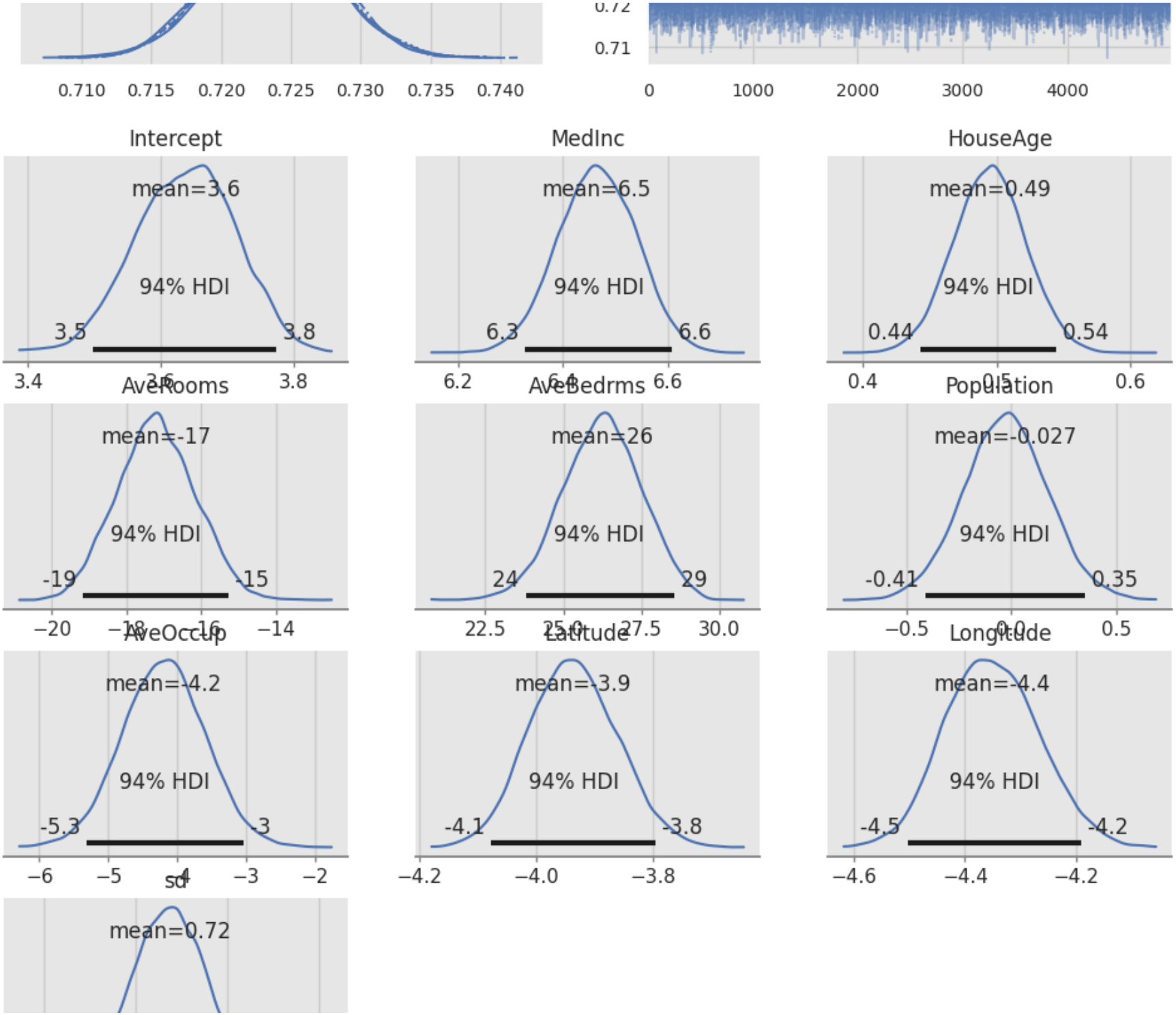

AveBedrms
Population
AveOccup
Latitude
Longitude

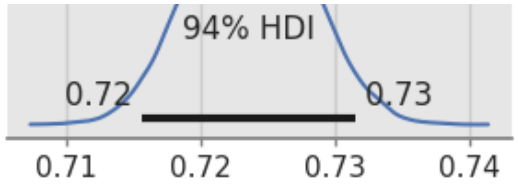

sd
−30
−20
−10
0
10
20
30
40

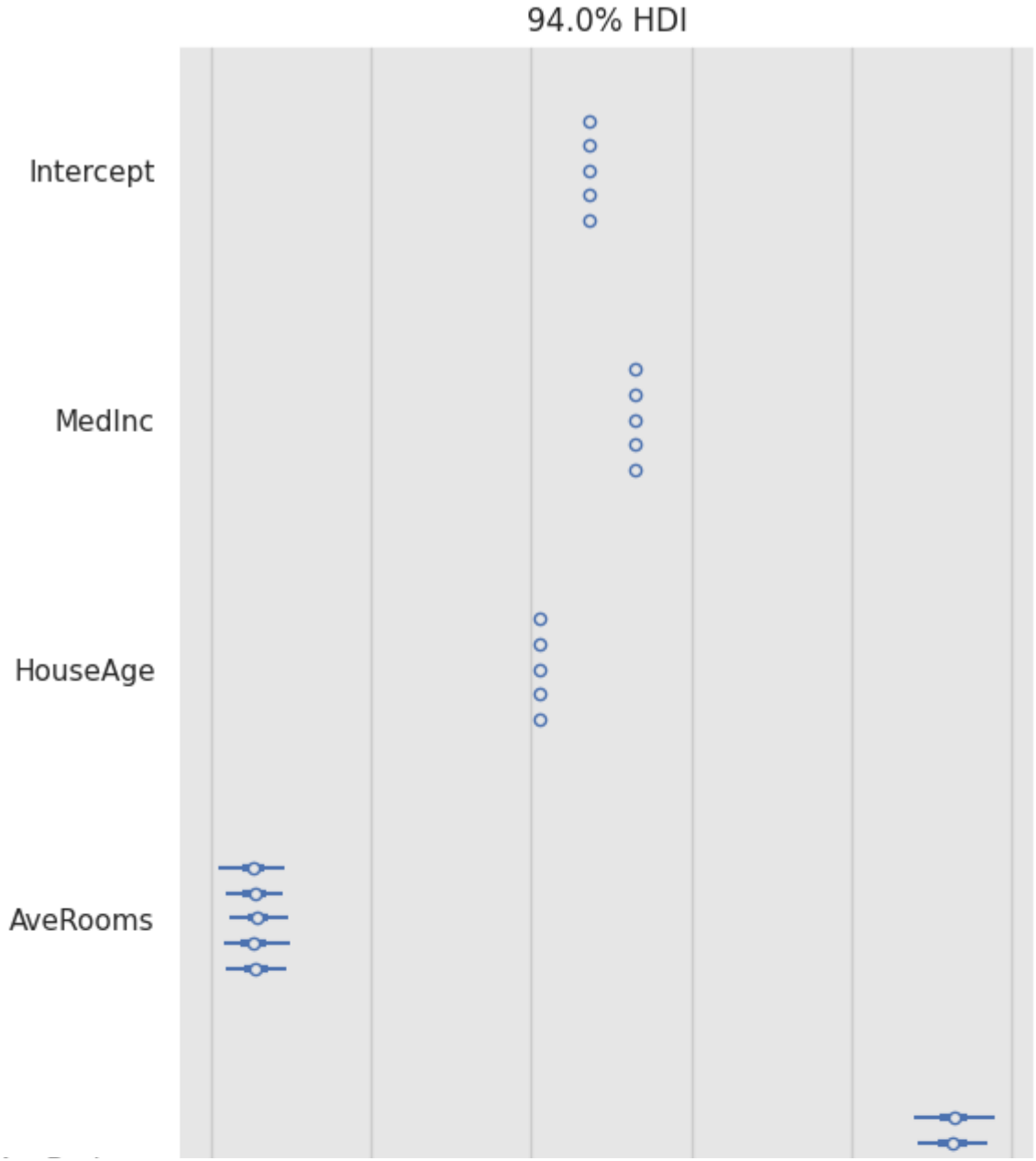

Intercept
3.4
3.5
3.6
3.7
3.8
MedInc
6.2
6.3
6.4
6.5
6.6
6.7
HouseAge
0.40
0.45
0.50
0.55
0.60
AveRooms
−20
−18
−16
−14
AveBedrms

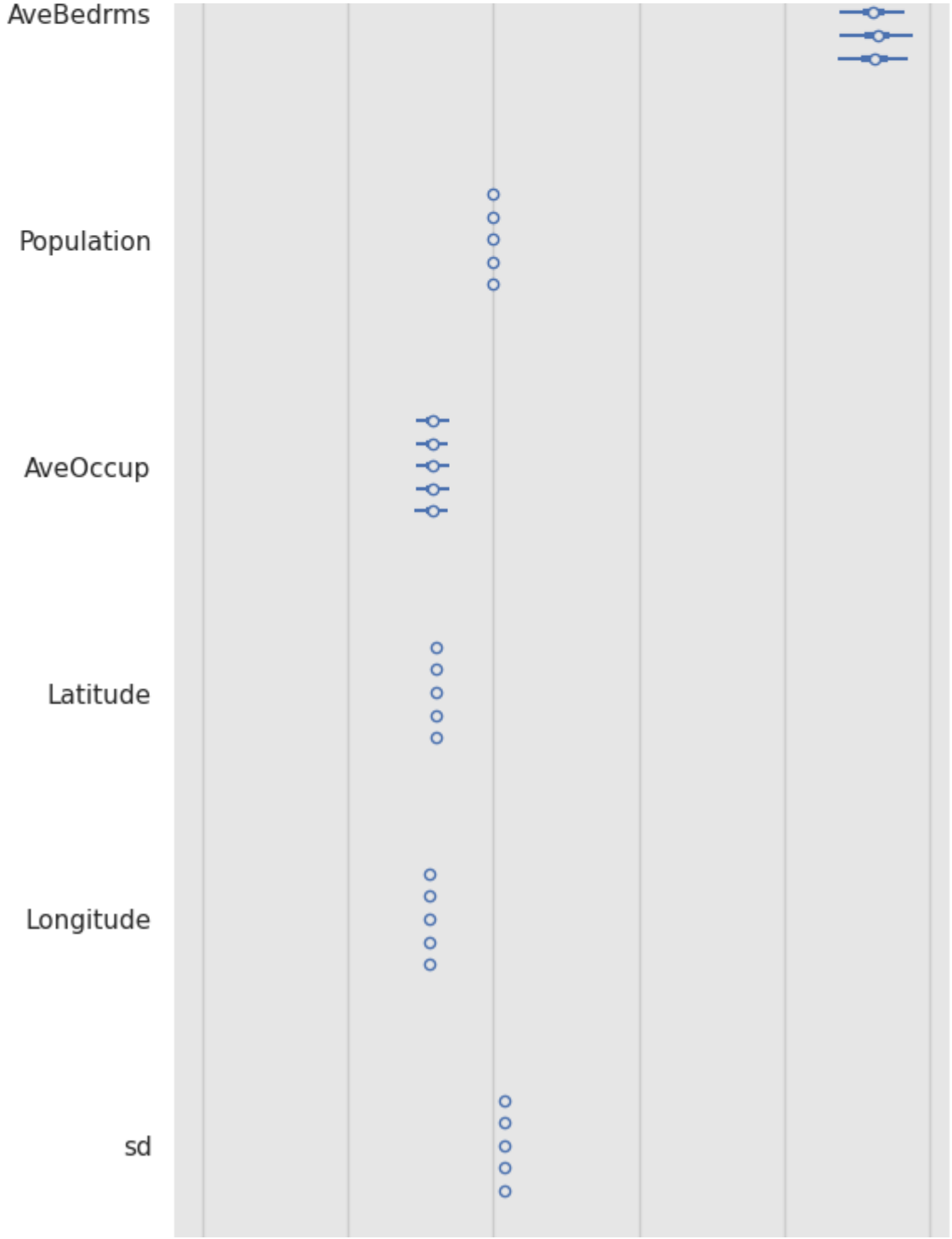

Intercept
3.8
3.6
3.4
0
1000
2000
3000
4000
MedInc
6.6
6.4
6.2
HouseAge
0.6
0.5
0.4
AveRooms
−12.5
−15.0
−17.5
−20.0
AveBedrms
30

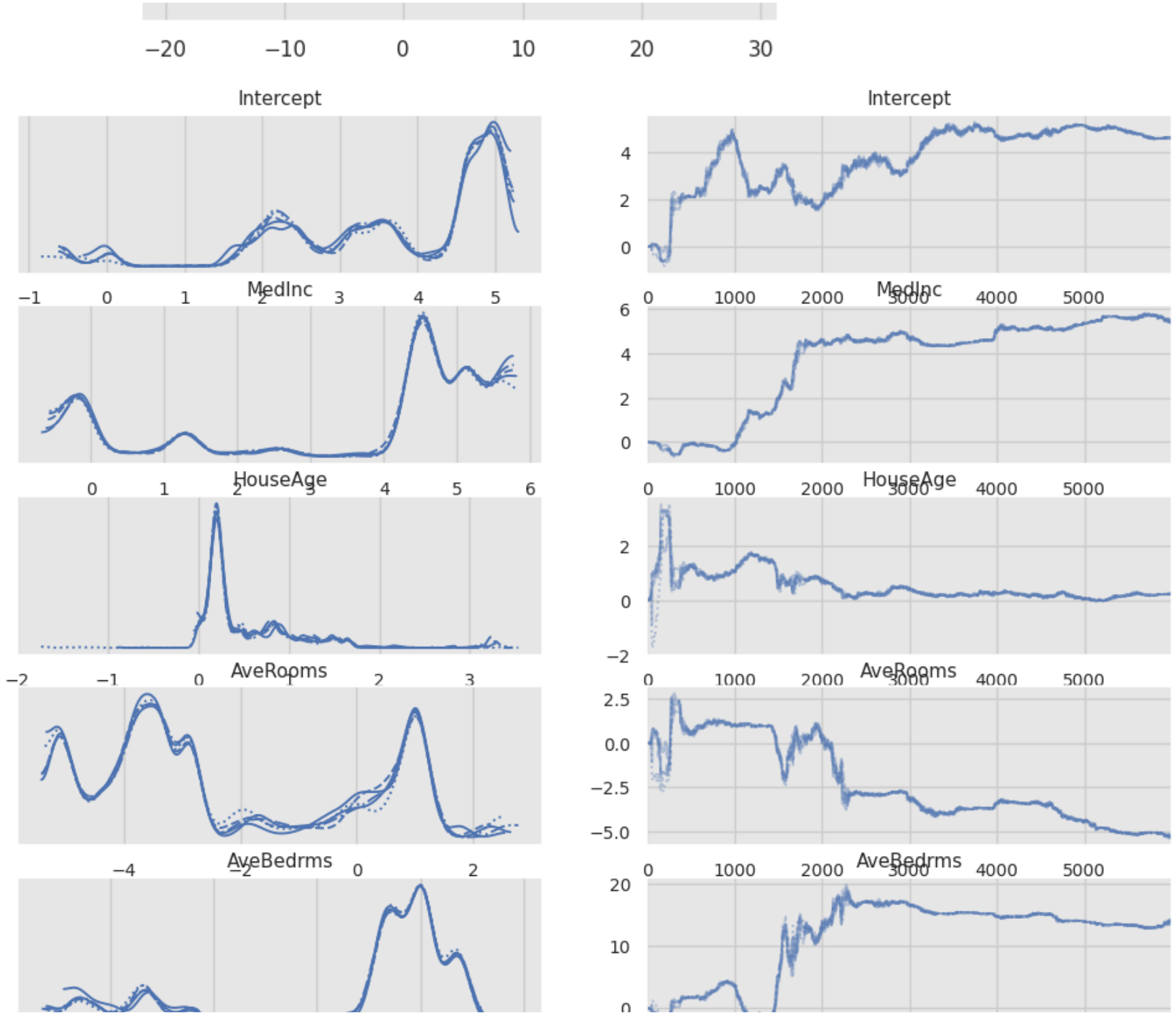

Population
AveOccup
Latitude
Longitude
sd
22
24
26
28
30
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
−6
−5
−4
−3
−2
−4.2
−4.1
−4.0
−3.9
−3.8
−3.7
−4.6
−4.5
−4.4
−4.3
−4.2
−4.1
0
1000
2000
3000
4000
25
0.5
0.0
−0.5
−2
−4
−6
−3.8
−4.0
−4.2
−4.4
−4.6
0.74
0.73

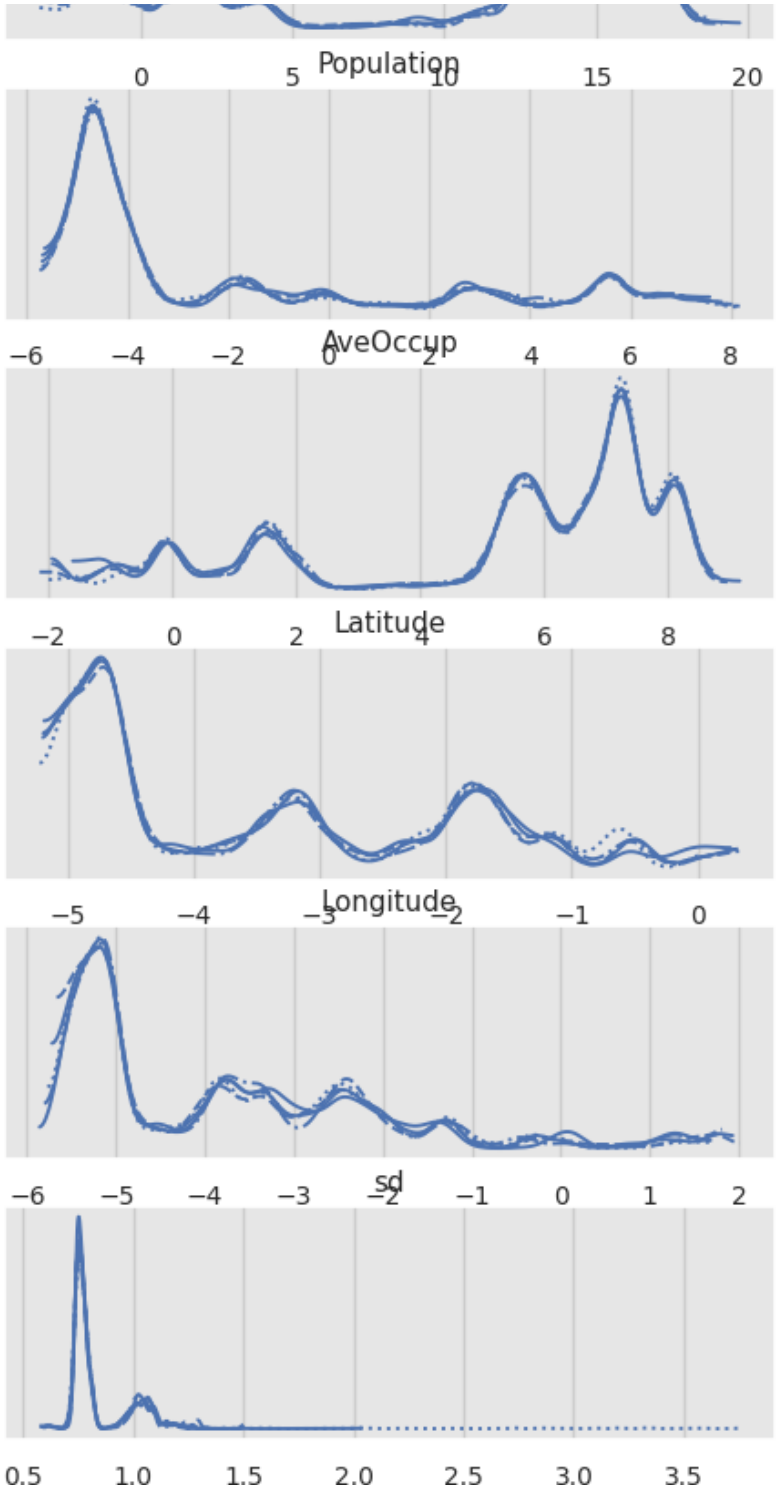

0.710
0.715
0.720
0.725
0.730
0.735
0.740
0.72
0.71
0
1000
2000
3000
4000
Intercept
mean=3.6
94% HDI
3.5
3.8
3.4
3.6
3.8
MedInc
mean=6.5
94% HDI
6.3
6.6
6.2
6.4
6.6
HouseAge
mean=0.49
94% HDI
0.44
0.54
0.4
0.5
0.6
AveRooms
mean=-17
94% HDI
-19
-15
−20
−18
−16
−14
AveBedrms
mean=26
94% HDI
24
29
22.5
25.0
27.5
30.0
Population
mean=-0.027
94% HDI
-0.41
0.35
−0.5
0.0
0.5
AveOccup
mean=-4.2
94% HDI
-5.3
-3
−6
−5
−4
−3
−2
Latitude
mean=-3.9
94% HDI
-4.1
-3.8
−4.2
−4.0
−3.8
Longitude
mean=-4.4
94% HDI
-4.5
-4.2
−4.6
−4.4
−4.2
sd
mean=0.72

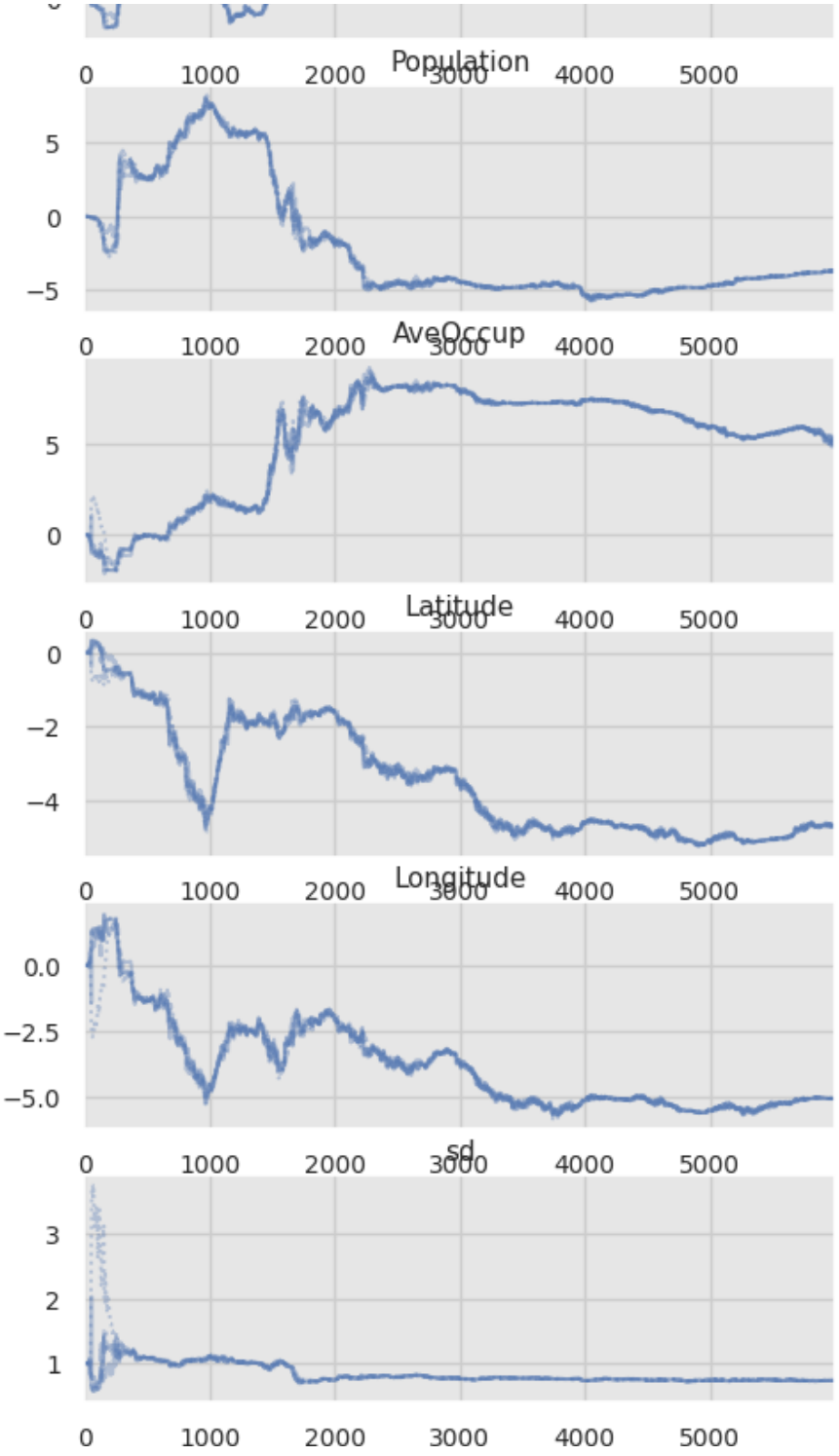

94% HDI
0.72
0.73
0.71
0.72
0.73
0.74

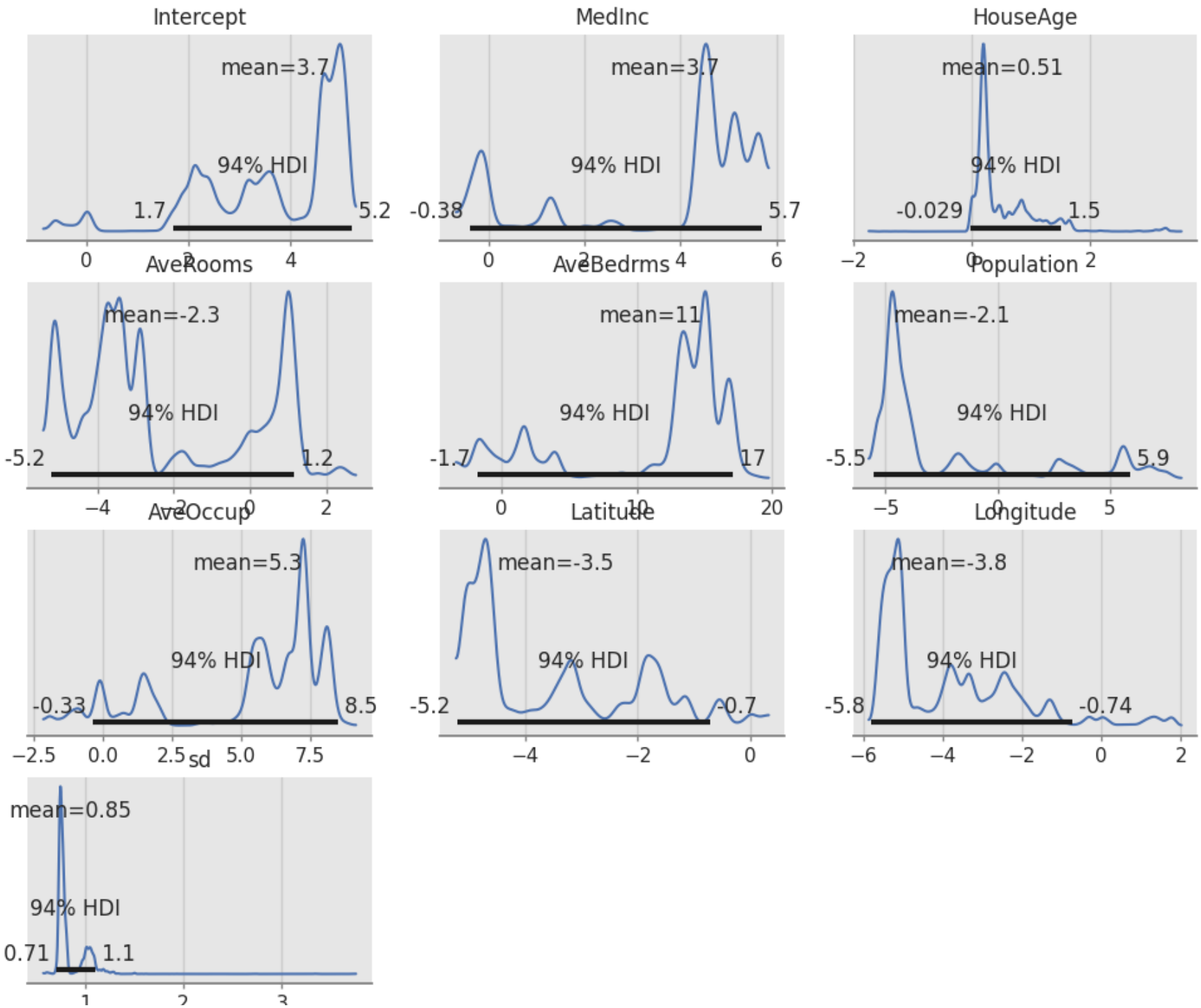

94.0% HDI
Intercept
MedInc
HouseAge
AveRooms

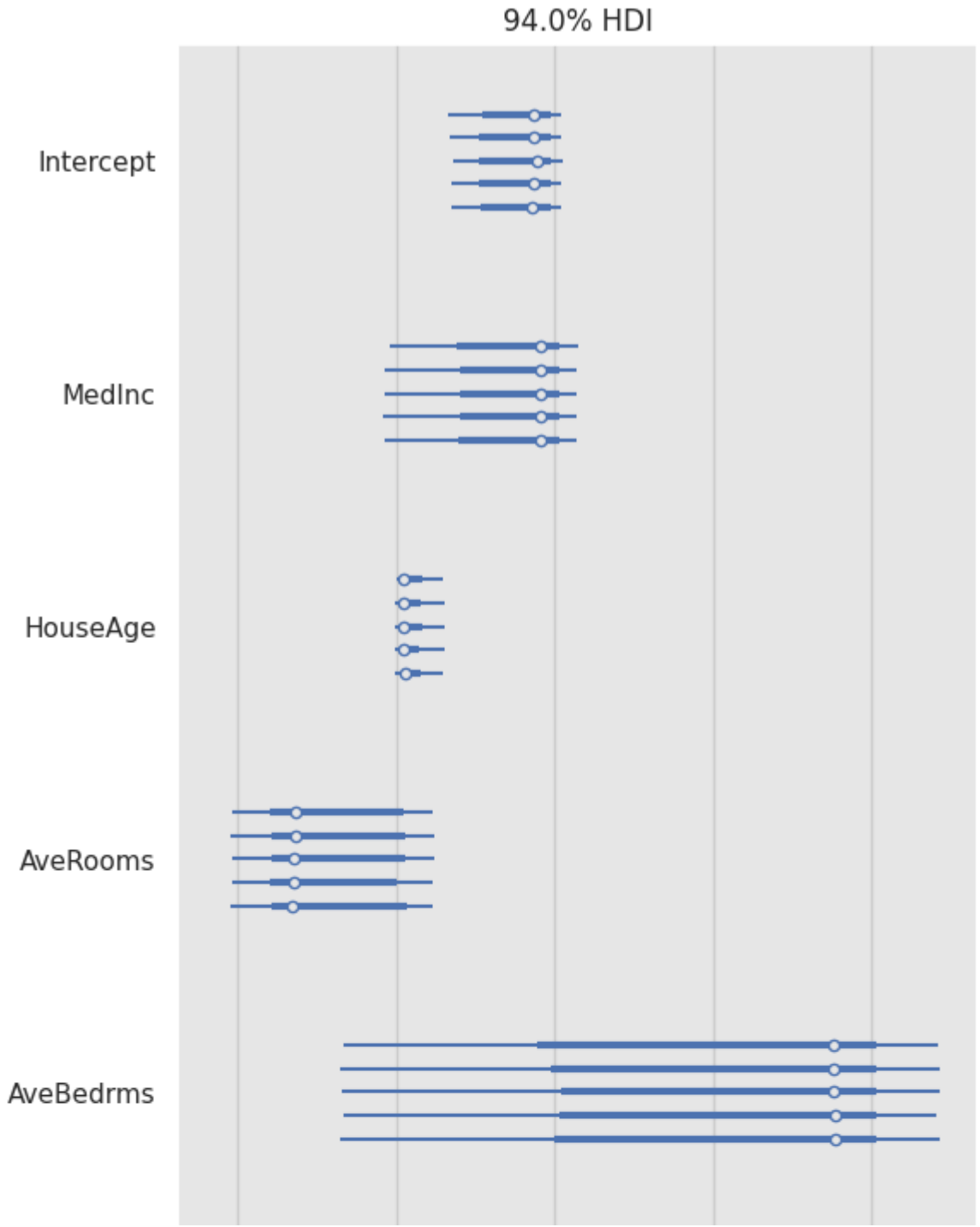

AveBedrms
Population
AveOccup
Latitude
Longitude
sd

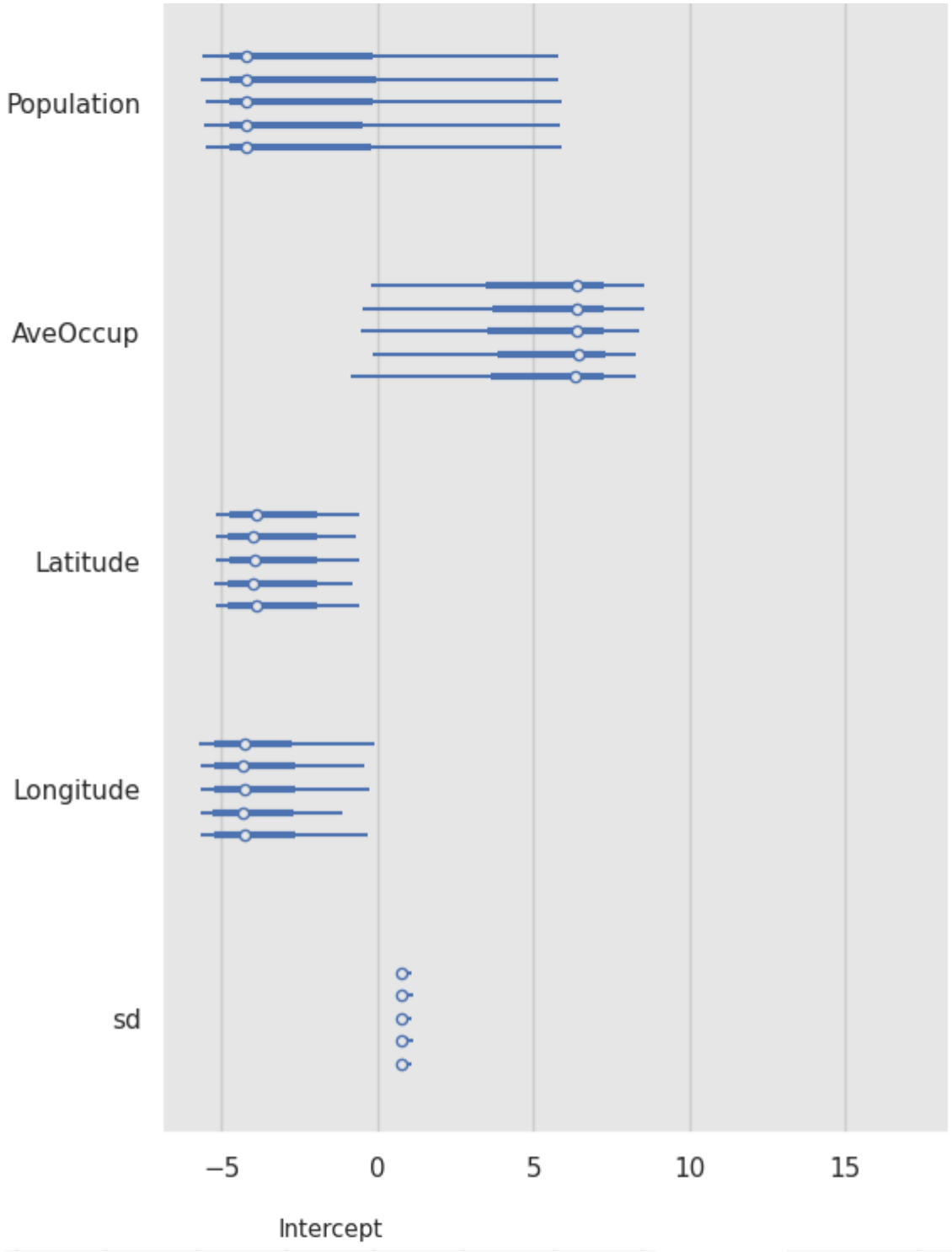

−20
−10
0
10
20
30
Intercept
Intercept
MedInc
MedInc
HouseAge
HouseAge
AveRooms
AveRooms
AveBedrms
AveBedrms


Population
AveOccup
Latitude
Longitude
sd

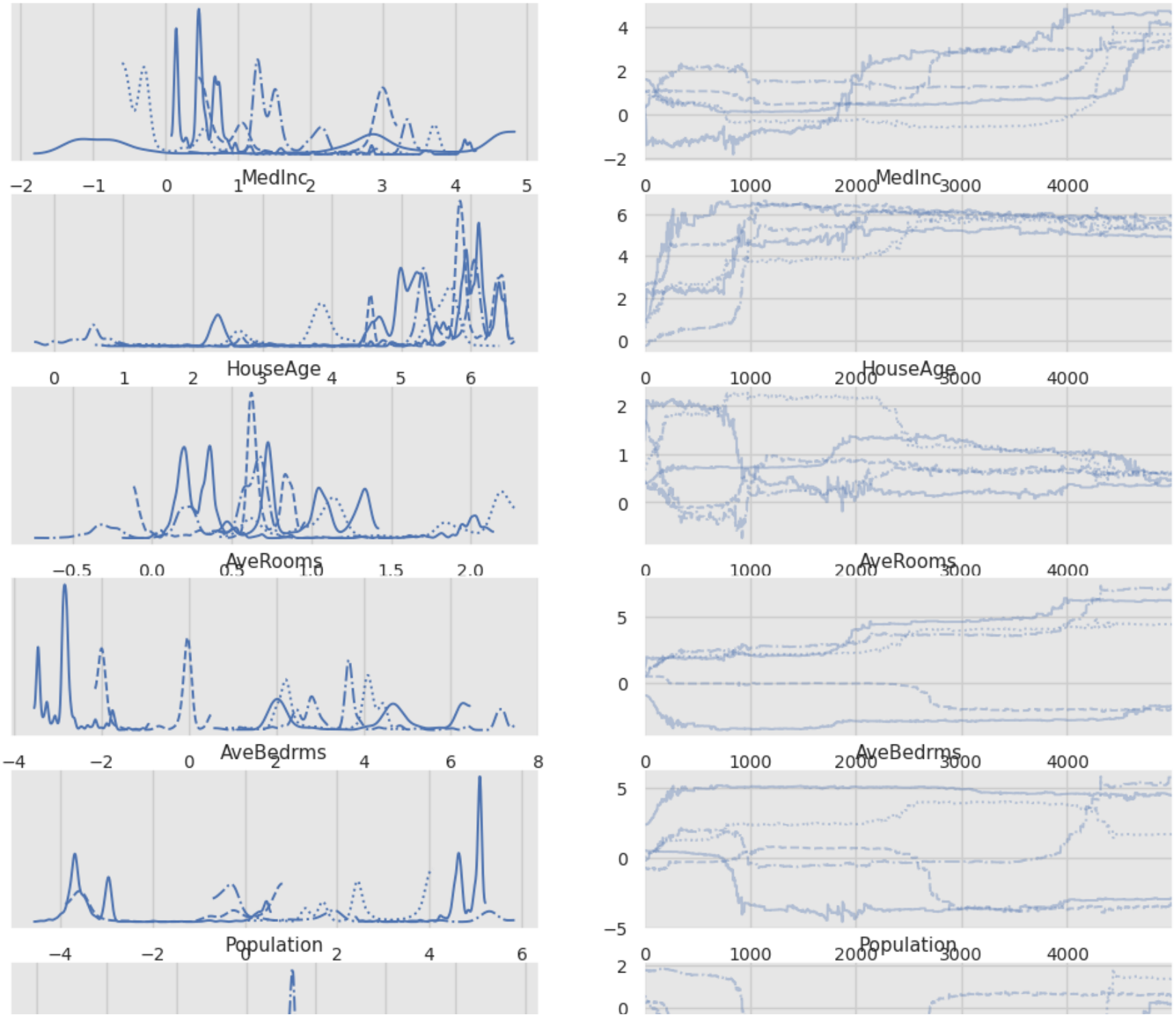

Population
AveOccup
Latitude
Longitude
sd

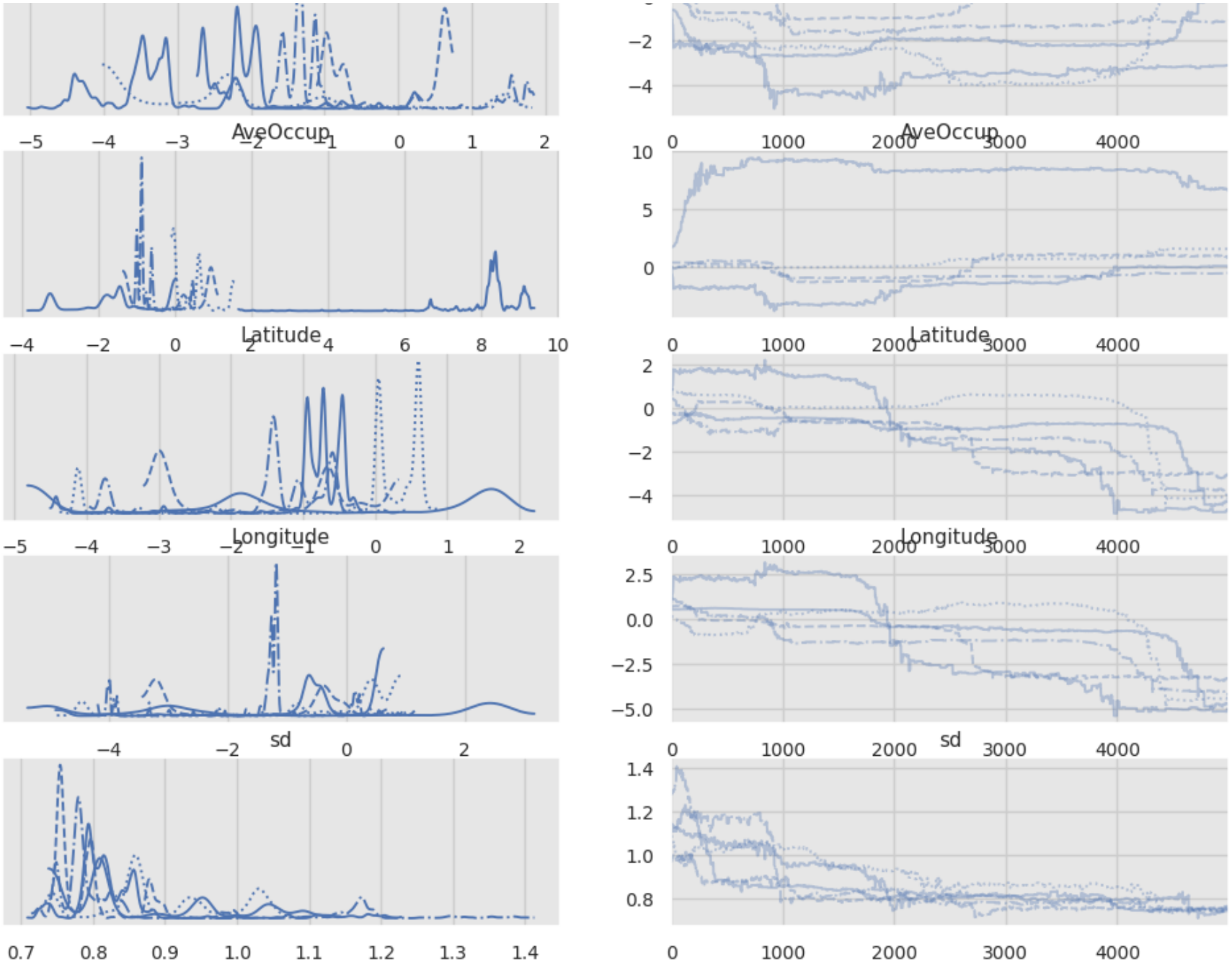

Intercept
mean=3.7
94% HDI
1.7
5.2
MedInc
mean=3.7
94% HDI
-0.38
5.7
HouseAge
mean=0.51
94% HDI
-0.029
1.5
AveRooms
mean=-2.3
94% HDI
-5.2
1.2
AveBedrms
mean=11
94% HDI
-1.7
17
Population
mean=-2.1
94% HDI
-5.5
5.9
AveOccup
mean=5.3
94% HDI
-0.33
8.5
Latitude
mean=-3.5
94% HDI
-5.2
-0.7
Longitude
mean=-3.8
94% HDI
-5.8
-0.74
sd
mean=0.85
94% HDI
0.71
1.1

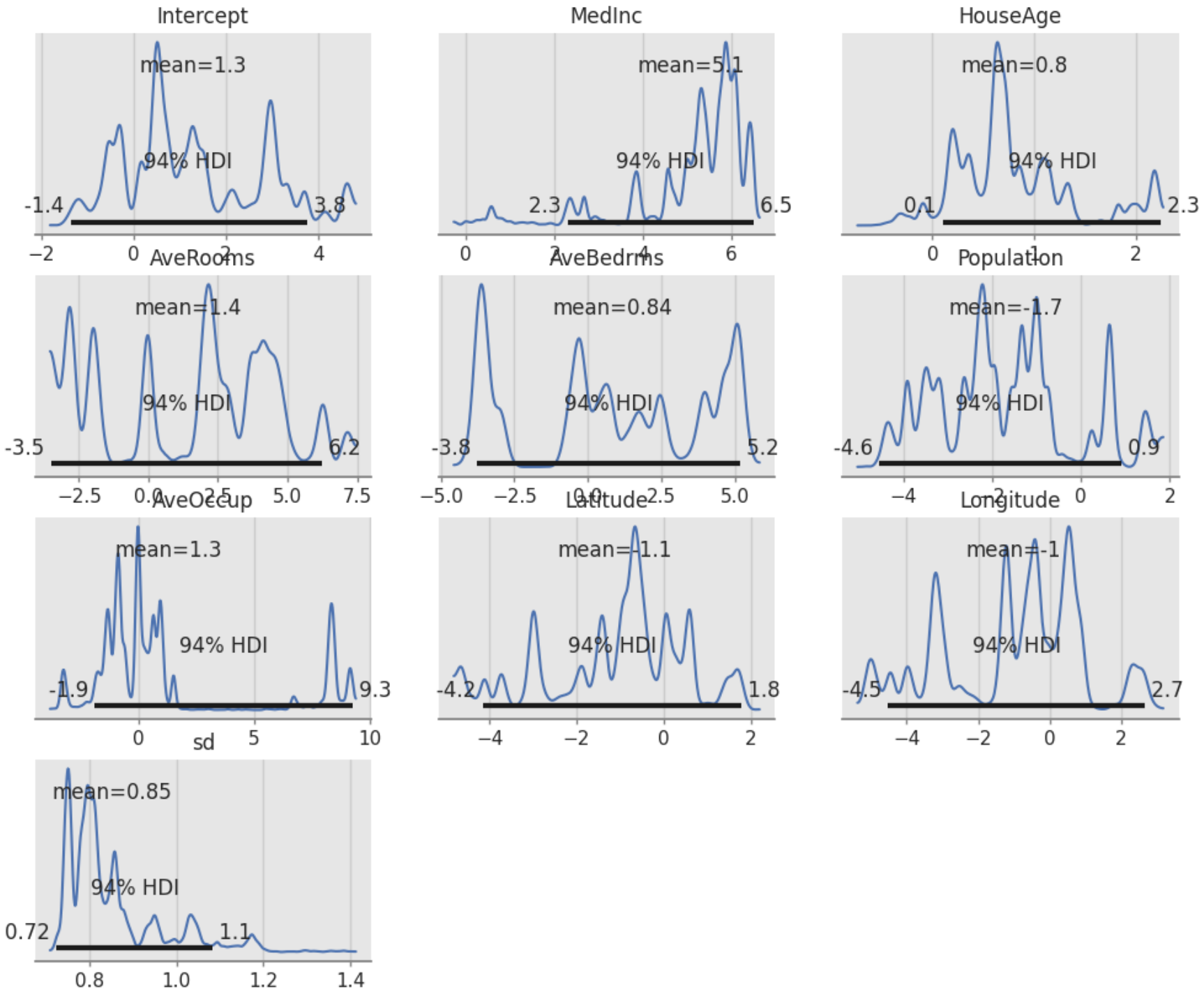

94.0% HDI
Intercept
MedInc
HouseAge
AveRooms
AveBedrms

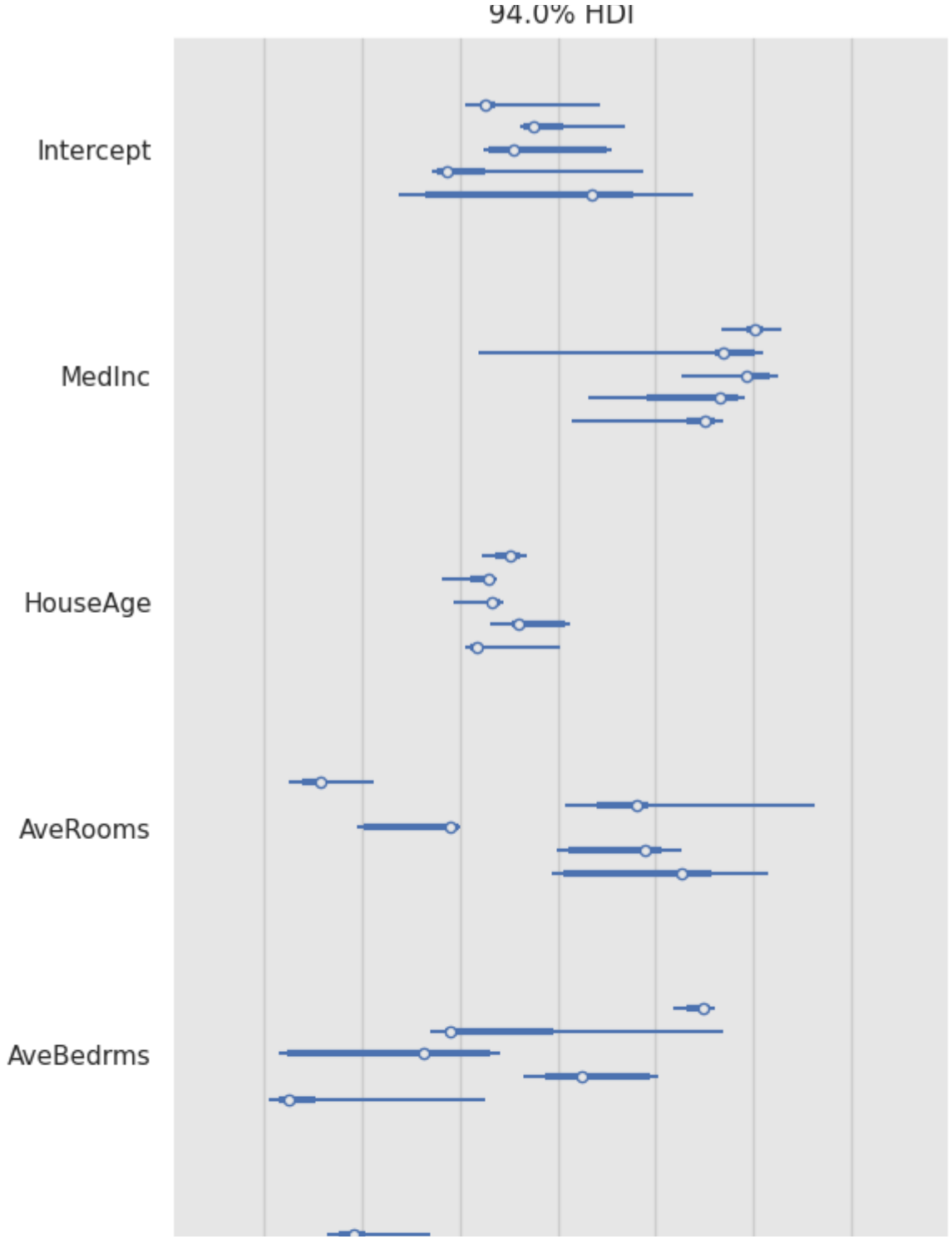

Population
AveOccup
Latitude
Longitude
sd
−5
0
5
10
15
Intercept

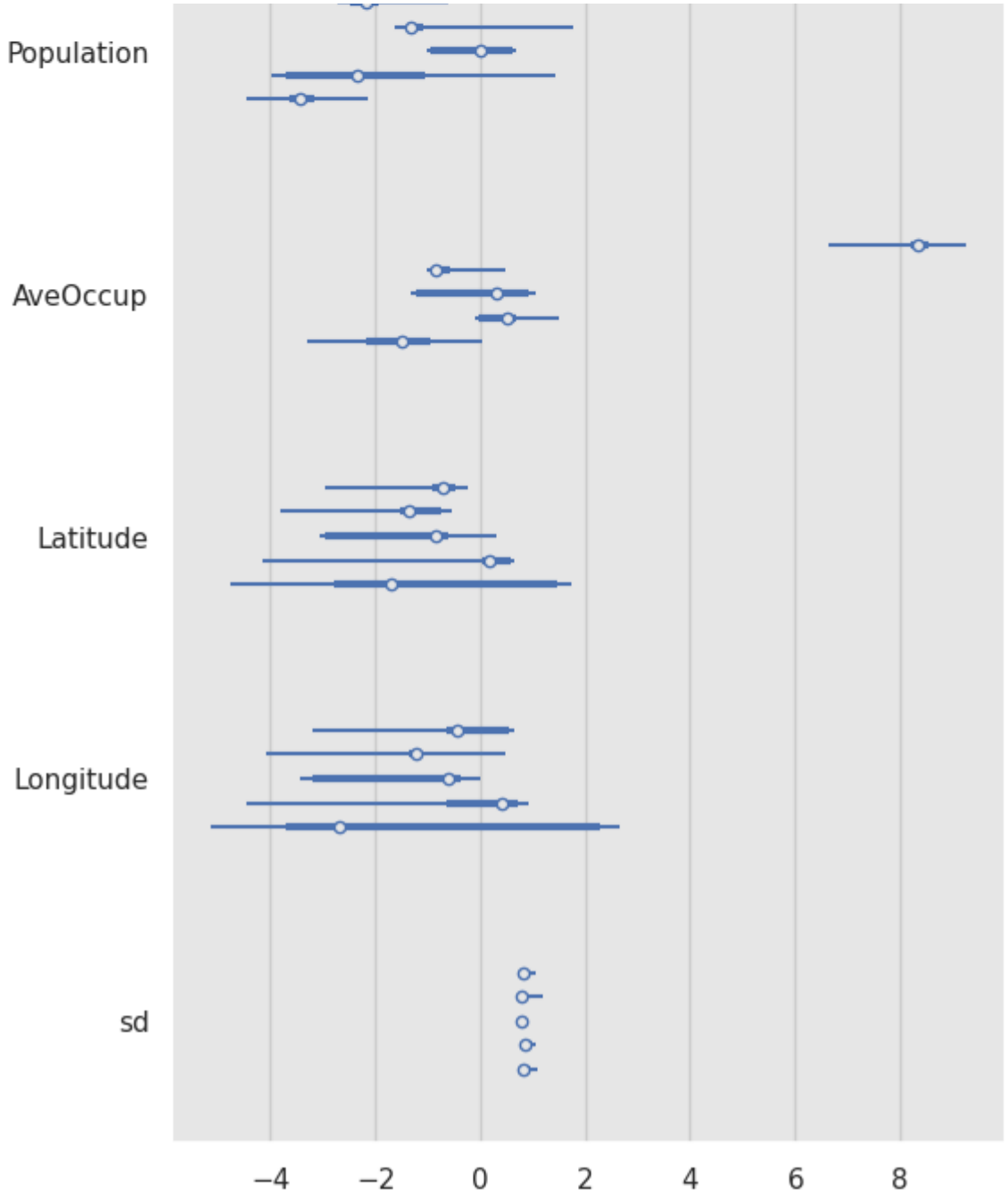
Intercept

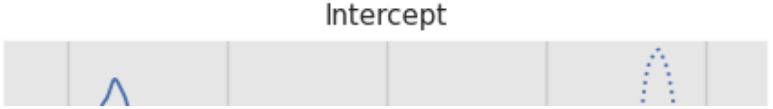

MedInc
HouseAge
AveRooms
AveBedrms
Population
MedInc
HouseAge
AveRooms
AveBedrms
Population


AveOccup
Latitude
Longitude
sd
−5 −4 −3 −2 −1 0 1 2
−4 −2 0 2 4 6 8 10
−4 −2 0 2
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
0 1000 2000 3000 4000
−2 −4
10 5 0
2 0 −2 −4
2.5 0.0 −2.5 −5.0
1.4 1.2 1.0 0.8

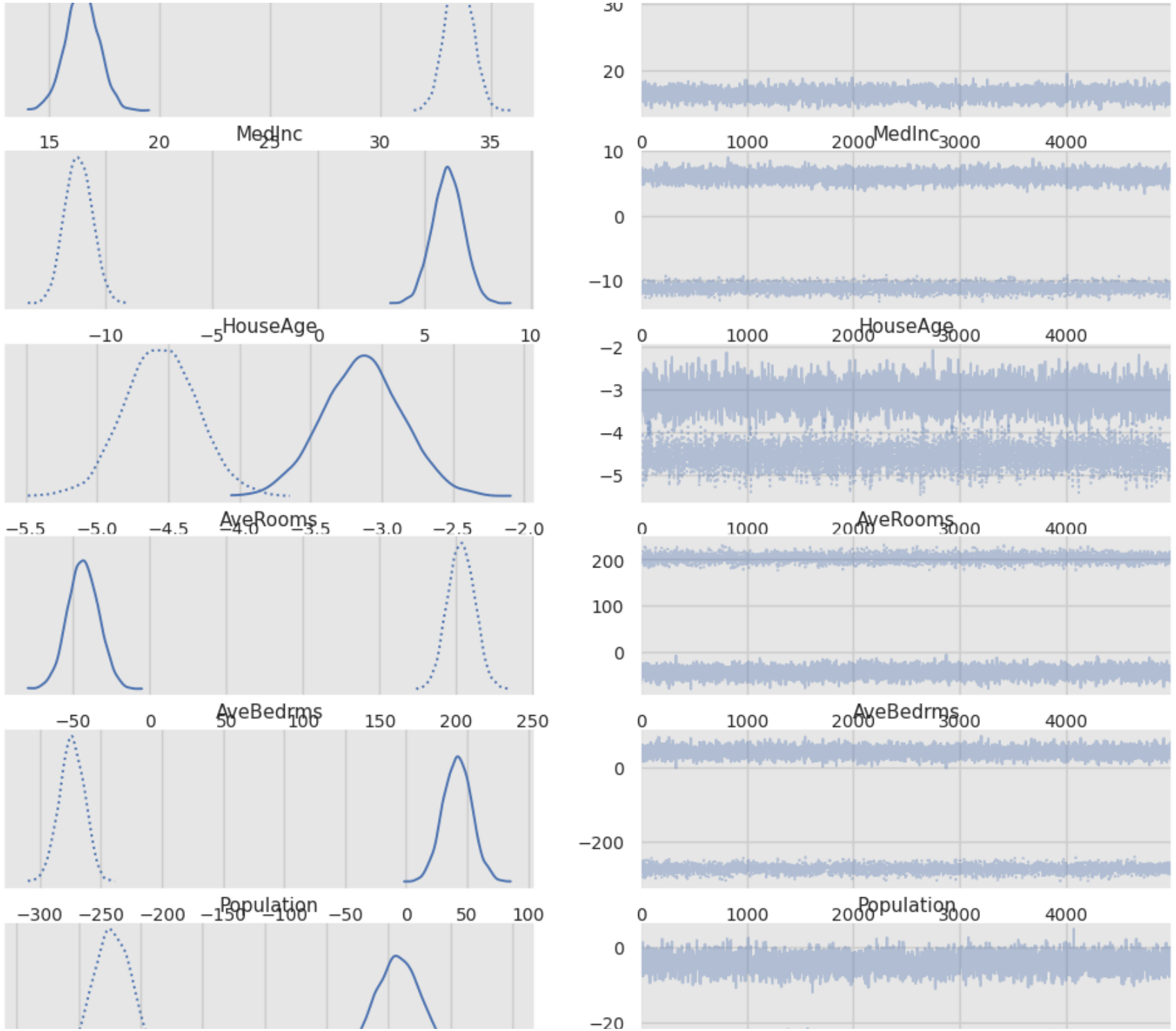

Intercept
mean=1.3
94% HDI
-1.4
3.8
MedInc
mean=5.1
94% HDI
2.3
6.5
HouseAge
mean=0.8
94% HDI
0.1
2.3
AveRooms
mean=1.4
94% HDI
-3.5
6.2
AveBedrms
mean=0.84
94% HDI
-3.8
5.2
Population
mean=-1.7
94% HDI
-4.6
0.9
AveOccup
mean=1.3
94% HDI
-1.9
9.3
Latitude
mean=-1.1
94% HDI
-4.2
1.8
Longitude
mean=-1
94% HDI
-4.5
2.7
sd
mean=0.85
94% HDI
0.72
1.1

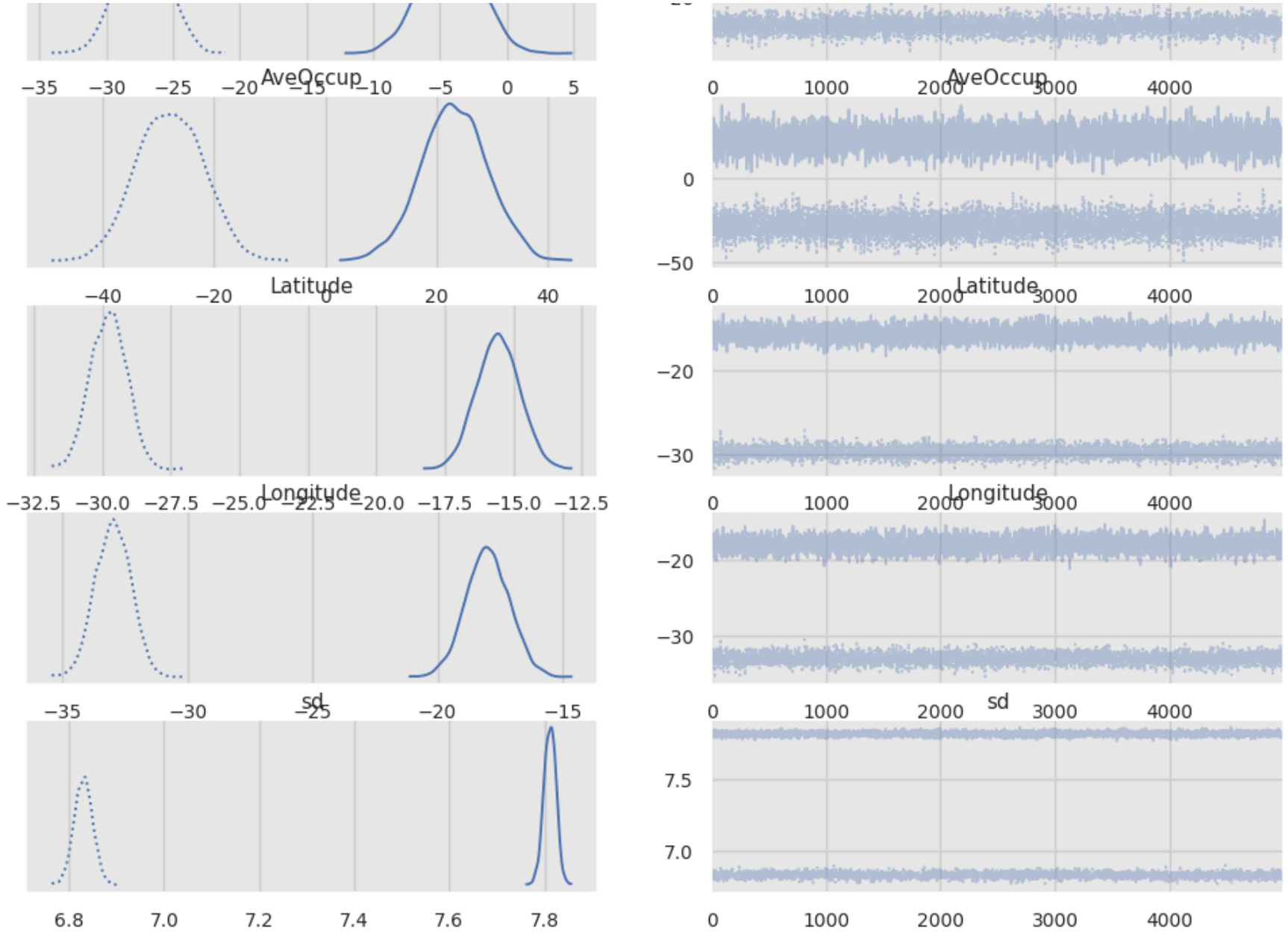

94.0% HDI
Intercept
MedInc
HouseAge
AveRooms
AveBedrms

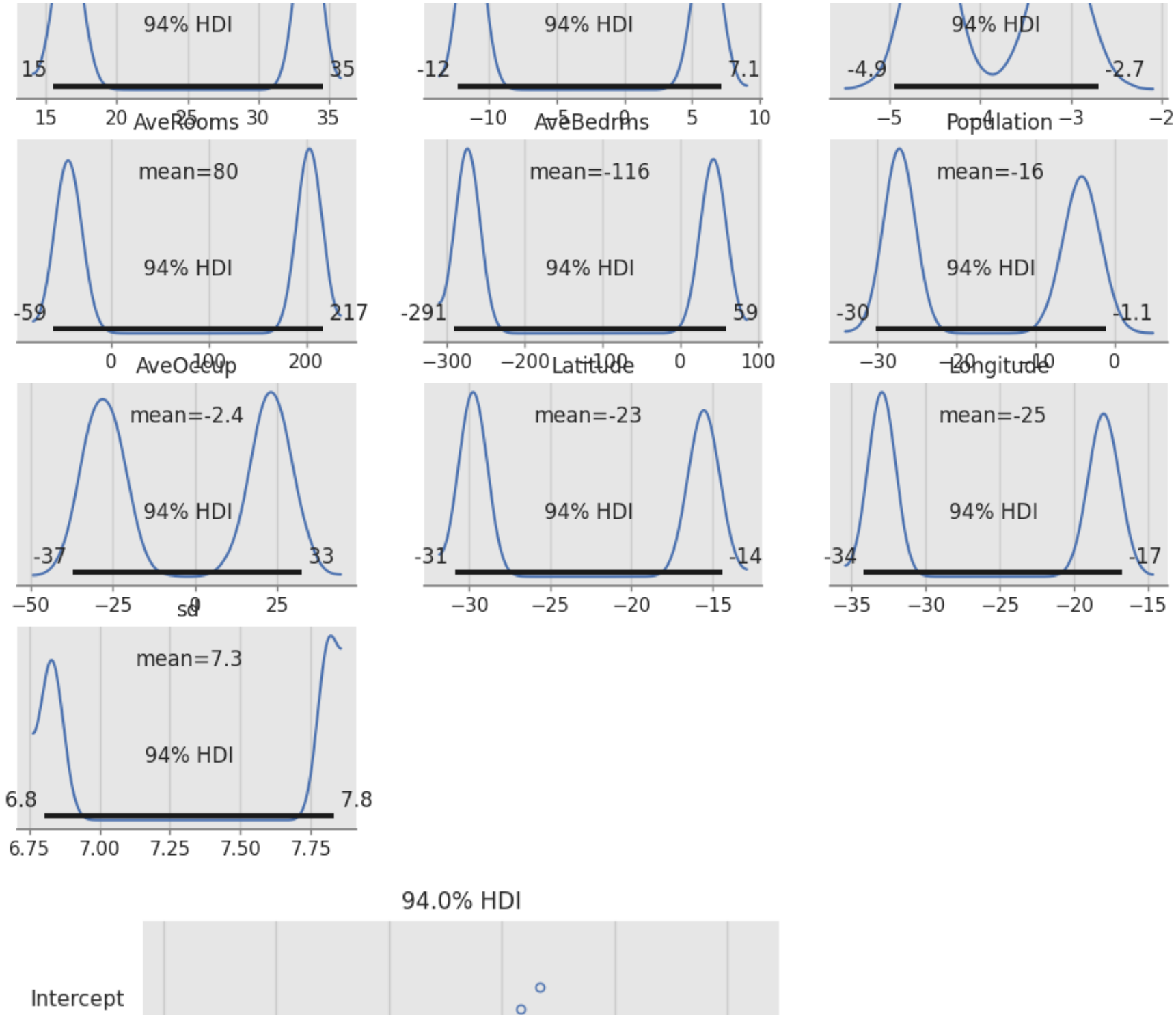

Population
AveOccup
Latitude
Longitude
sd
−4
−2
0
2
4
6
8

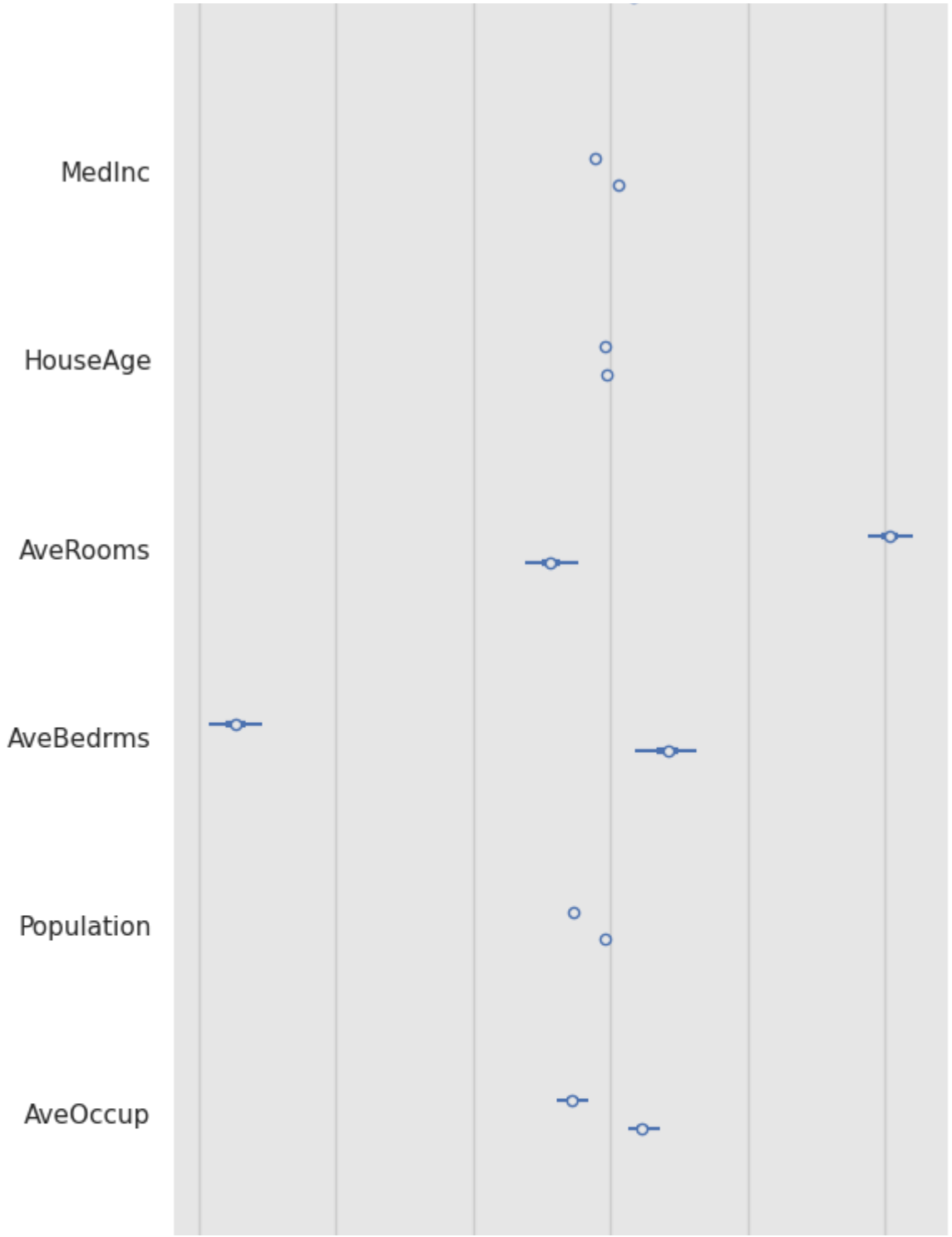

Intercept

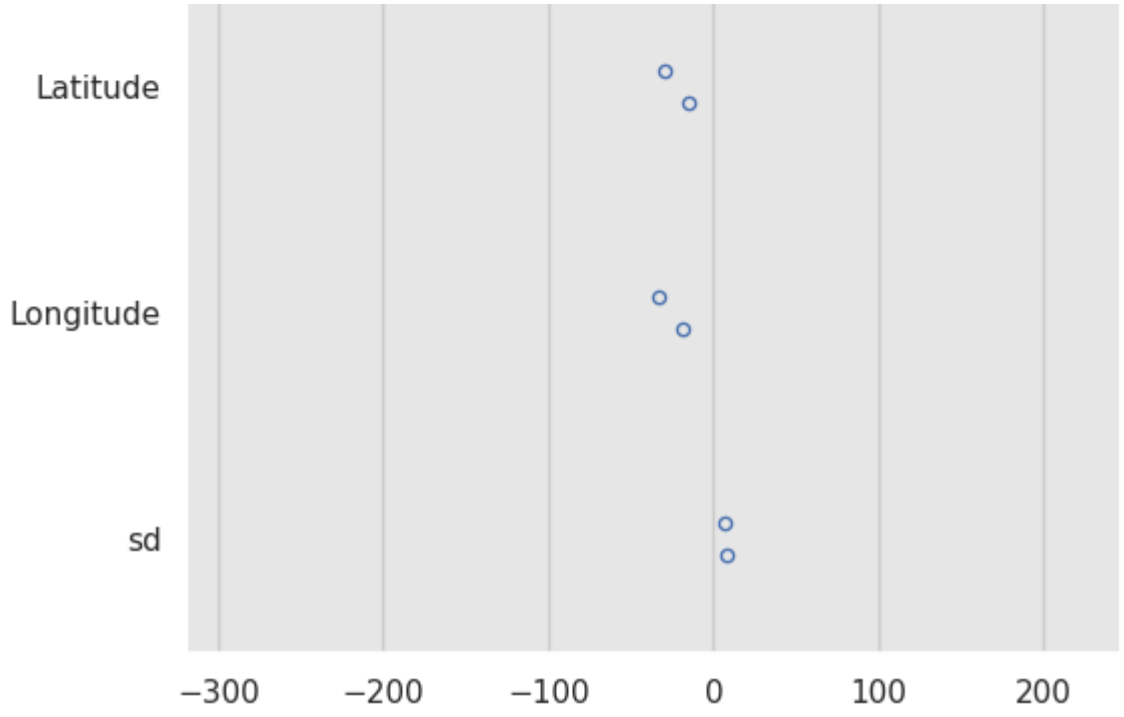

Intercept


MedInc
HouseAge
AveRooms
AveBedrms
Population
MedInc
HouseAge
AveRooms
AveBedrms
Population


AveOccup
Latitude
Longitude
sd
AveOccup
Latitude
Longitude
sd
Intercept
mean=25
MedInc
mean=-2.6
HouseAge
mean=-3.8


94% HDI
15
35
15 20 25 30 35
AveRooms
94% HDI
-12
7.1
−10 −5 0 5 10
AveBedrms
94% HDI
-4.9
-2.7
−5 −4 −3 −2
Population
mean=80
94% HDI
-59
217
0 100 200
mean=-116
94% HDI
-291
59
−300 −200 −100 0 100
mean=-16
94% HDI
-30
-1.1
−30 −20 −10 0
AveOccup
mean=-2.4
94% HDI
-37
33
−50 −25 0 25
Latitude
mean=-23
94% HDI
-31
-14
−30 −25 −20 −15
Longitude
mean=-25
94% HDI
-34
-17
−35 −30 −25 −20 −15
sd
mean=7.3
94% HDI
6.8
7.8
6.75 7.00 7.25 7.50 7.75
94.0% HDI
Intercept


MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup



Dataset II: Prices Dataset

House and Land features are considered to predict the Sale Price.

The data consists of 2919 rows and 81 columns. It has 79 features and one SalePrice - the property's sale price in dollars, which is the target variable to be predicted. (ID column is reduntant). The features can be broadly catagorized in

1. Areas Covered, such as 1stFlrSF, 2ndFlrSF, GrLivArea -16 variables
2. Class, Condition, Quality- 25 variables
3. Rooms, numbers etc- 6 variables
4. Lot, Street, Alley- 11 variables
5. BASEMENT- 6 variables
6. Garage- 6 variables
7. Years- 4 variables
8. MiscFeature: Miscellaneous feature such as MiscVal, Type of sale, Condition of sale etc- 5 vaariables

Load Data

In [ ]:
```
#from google.colab import files
#uploaded=files.upload()
```

In [ ]:
```
#import io
#df6=pd.read_csv(io.BytesIO(uploaded['house_data.csv']))
```

In [ ]:
```
FILE_NAME = "houseprice_Data.csv"
LABEL_COL = "SalePrice"
df6 = template.load_data(FILE_NAME)
display(df.head())
print(df6.shape)
print(df6.dtypes)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
(2919, 81)
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
                 ...
MoSold            int64
YrSold            int64
SaleType         object
SaleCondition    object
SalePrice       float64
Length: 81, dtype: object
```

In [ ]:
```
df6.isnull().sum()
```

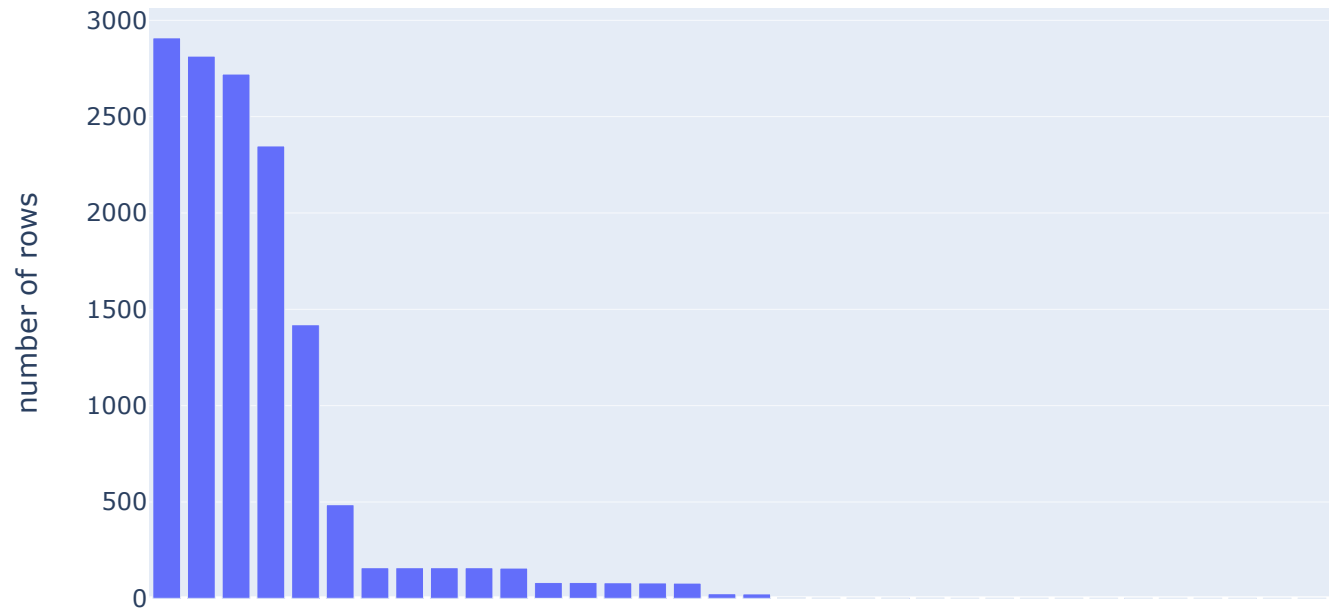
Out[ ]:
```
Id                0
MSSubClass        0
MSZoning          4
```

```
LotFrontage      486
LotArea            0
                 ...
MoSold             0
YrSold             0
SaleType           1
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

In [ ]:
```
# Import the necessaries libraries
import plotly.offline as pyo
import plotly.graph_objs as go
# Set notebook mode to work in offline
pyo.init_notebook_mode()
```

Missing Values Analysis

In [ ]:
```
template.missing_values_display(df6)
```



PoolQC
MiscFeature
Alley
Fence
FireplaceQu
LotFrontage
GarageFinish
GarageYrBlt
GarageQual
GarageCond
GarageType
BsmtExposure
BsmtCond
BsmtQual
BsmtFinType2
BsmtFinType1
MasVnrType
MasVnrArea
MSZoning
BsmtFullBath
BsmtHalfBath
Utilities
Functional
Exterior2nd
Exterior1st
SaleType
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
Electrical
KitchenQual
GarageCars
GarageArea
TotalBsmtSF

	Total	Percent
PoolQC	2909	0.996574
MiscFeature	2814	0.964029
Alley	2721	0.932169
Fence	2348	0.804385
FireplaceQu	1420	0.486468
...	...	...
CentralAir	0	0.000000
SaleCondition	0	0.000000
Heating	0	0.000000
Foundation	0	0.000000
Id	0	0.000000

81 rows × 2 columns

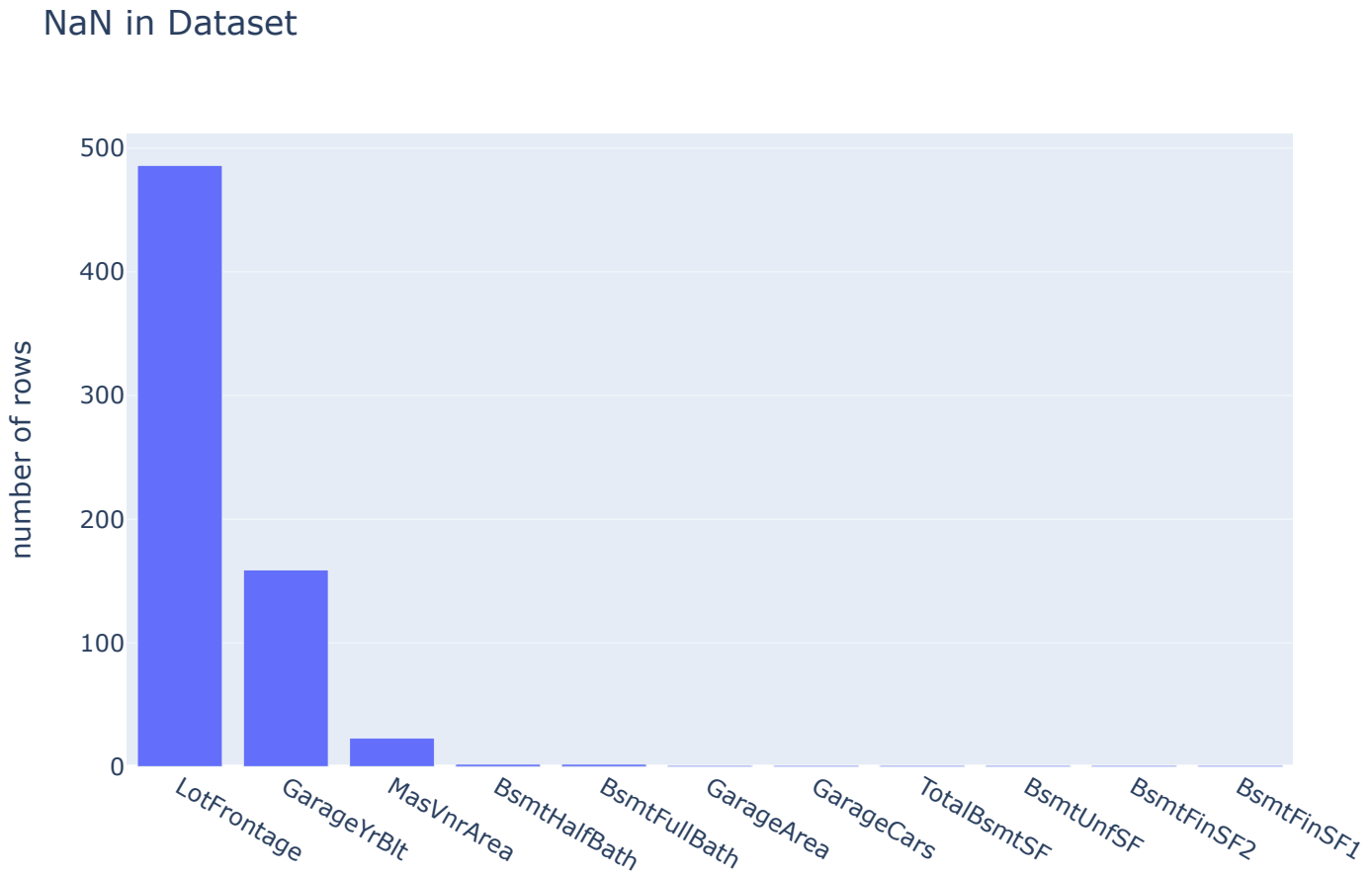
Filling missing values

For a few columns there is lots of NaN entries. However, from the data description, it is clear that this is not missing data. For PoolQC, NaN is not missing data but means no pool, likewise for Fence, FireplaceQu etc.

In [ ]:
```
# columns where NaN values have meaning e.g. no pool etc.
cols_fillna = ['PoolQC','MiscFeature','Alley','Fence','MasVnrType','FireplaceQu',
               'GarageQual','GarageCond','GarageFinish','GarageType', 'Electrical',
               'KitchenQual', 'SaleType', 'Functional', 'Exterior2nd', 'Exterior1st',
               'BsmtExposure','BsmtCond','BsmtQual','BsmtFinType1','BsmtFinType2',
               'MSZoning', 'Utilities']

# replace 'NaN' with 'None' in these columns
for col in cols_fillna:
```

```
    df6[col].fillna('None',inplace=True)
```

In [ ]:
```
template.missing_values_display(df6)
```



	Total	Percent
LotFrontage	486	0.166495
GarageYrBlt	159	0.054471
MasVnrArea	23	0.007879
BsmtFullBath	2	0.000685

	Total	Percent
BsmtHalfBath	2	0.000685
...	...	...
KitchenQual	0	0.000000
TotRmsAbvGrd	0	0.000000
Functional	0	0.000000
Fireplaces	0	0.000000
Id	0	0.000000

81 rows × 2 columns

Drop columns with lots of missing data

Dropping these features improves the performance of the ML models. For the remaining missing values we use linear interpolation in our cleaning function.

In [ ]:
```
df6 = template.cleaningup(df6, to_numeric=[], cols_to_interpolate=['LotFrontage','GarageYrBlt','MasVnrArea', 'BsmtFullBat
                                                                   'GarageCars','GarageArea','TotalBsmtSF','BsmtUnfSF','Bsm
                          cols_to_delete=['Id', 'PoolQC', 'FireplaceQu', 'Fence', 'Alley', 'MiscFeature'], to_date=[])
```

```
df is all cleaned up..
```

In [ ]:
```
df6.isnull().sum().sum()
```

Out[ ]:
```
0
```

Numerical and Categorical features

In [ ]:
```
numerical_feats, categorical_feats=template.numirical_catagorical_columns(df6)
```

```
Number of Numerical features:  37
Number of Categorical features:  38
*****************************************************************************************
Numrical Columns
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
```

```
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
**********************************************************************************************
Catagorical Columns
Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
       'PavedDrive', 'SaleType', 'SaleCondition'],
      dtype='object')
```

In [ ]:
```
df6[numerical_feats].head()
```

Out[ ]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF
0	60	65.0	8450	7	5	2003	2003	196.0	706.0	0.0	150.0
1	20	80.0	9600	6	8	1976	1976	0.0	978.0	0.0	284.0
2	60	68.0	11250	7	5	2001	2002	162.0	486.0	0.0	434.0
3	70	60.0	9550	7	5	1915	1970	0.0	216.0	0.0	540.0
4	60	84.0	14260	8	5	2000	2000	350.0	655.0	0.0	490.0

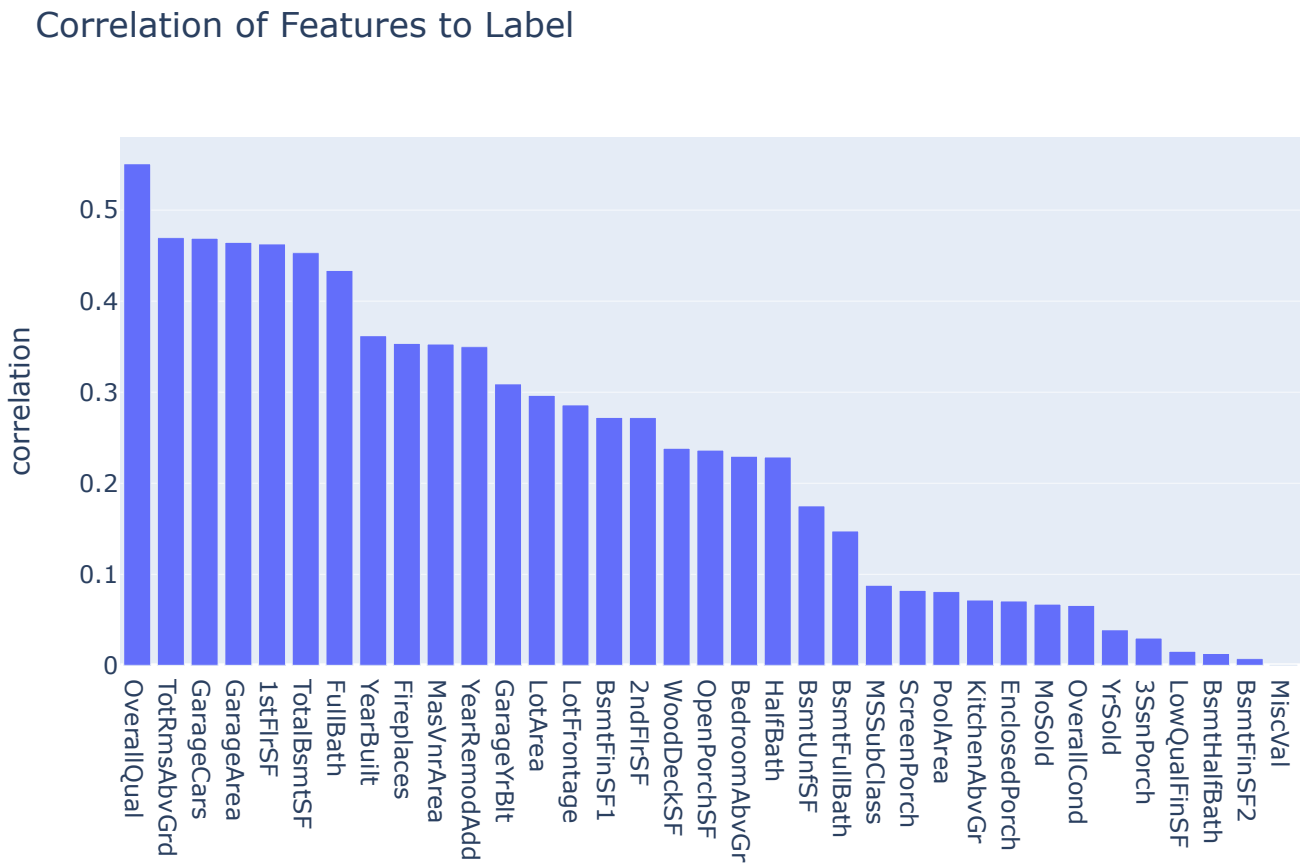
In [ ]:
```
df6[categorical_feats].head()
```

Out[ ]:

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle
0	RL	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story
1	RL	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	1Fam	1Story
2	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story
3	RL	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	1Fam	2Story
4	RL	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	1Fam	2Story

Correlation of Features with SalePrice

In [ ]:
```
LABEL_COL = "SalePrice"
df7=df6
template.corr_x_y(df7,LABEL_COL)
```



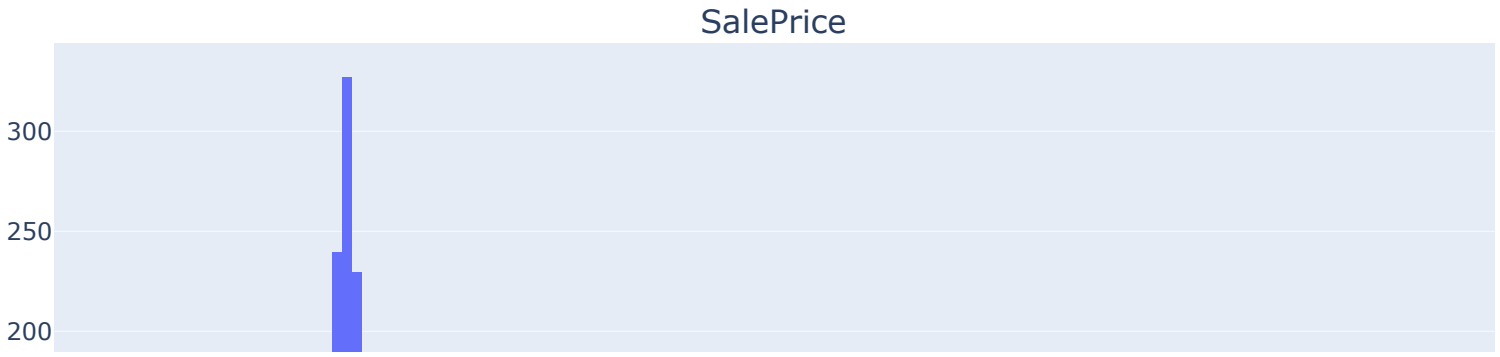
Correlation of numerical features to SalePrice

Keeping other parameters constant, we expect the value of a House to increase with its size and area. Also for this dataset, large correlations to SalePrice are observed for many of the Area features, such as GrLivArea, GarageArea, TotalBsmtSF, 1stFlrSF, etc. In the numerical features section, these features are explored in more detail and see how the results can be used for outlier detection and feature engineering.

Distribution of the target variable

In [ ]:
```
LABEL_COL = "SalePrice"

template.dist_label(df7,LABEL_COL)
```

```
mean: 180052.85464775655
standard deviation: 57381.565720790655
skewness: 2.5492483896291027
kurtosis: 14.66503546575219
```



The distribution of SalePrice seems to be normal with mean of 180052.85464775655 and standard deviation of 57381.565720790655.

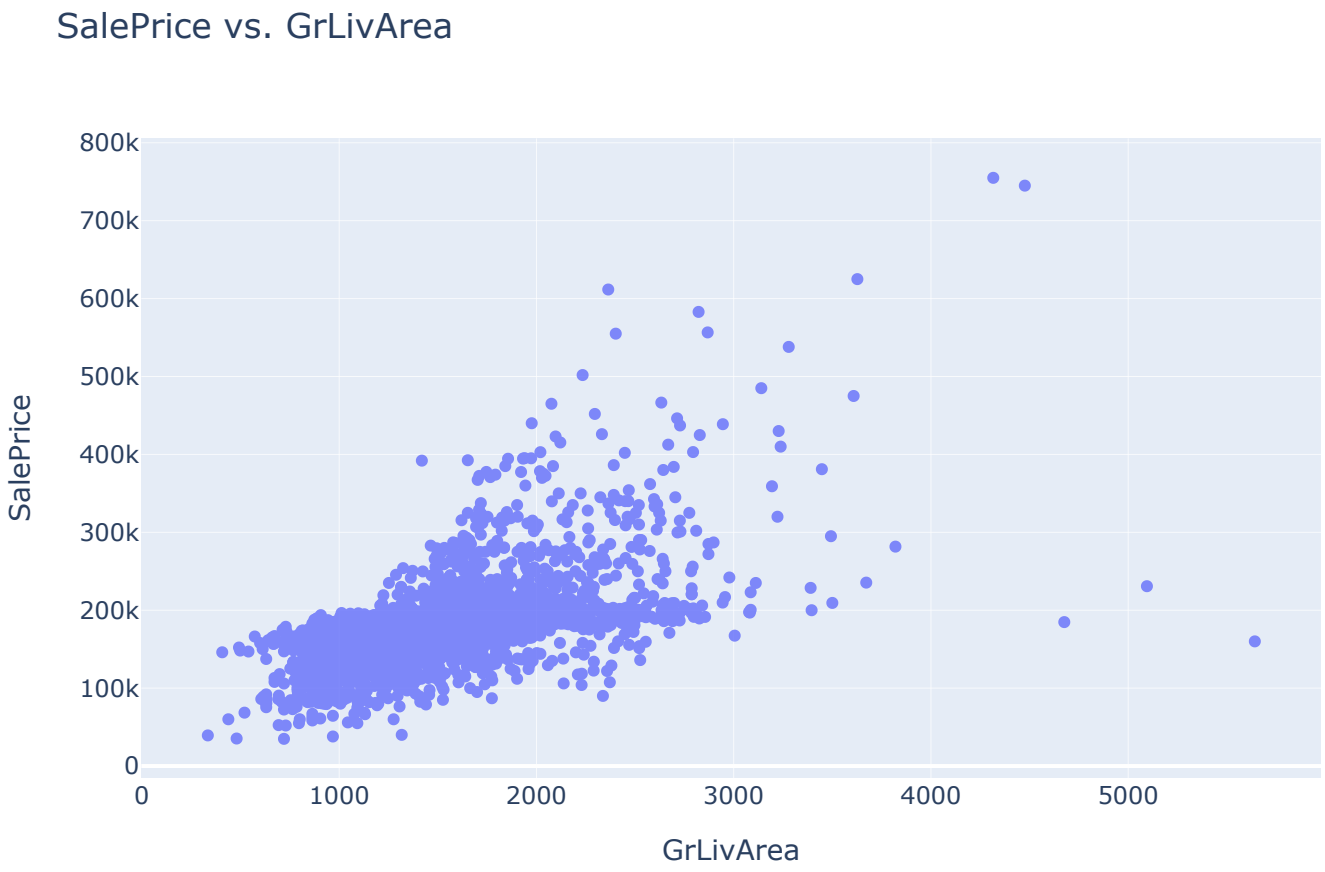
However,the distribution is skewed to the right.

Area features Analysis

Scatterplot: SalePrice vs GrLivArea

Of the Area features, 'GrLivArea' has the largest correlation to SalePrice.

In [ ]:
```
template.plotly_scatter_x_y(df7, 'GrLivArea', 'SalePrice')
```



Note on Outlier Detection

We store the index of the two data points to the lower right,

with SalePrice < 200 k and GrLivArea > 4000

In [ ]:
```
# outliers GrLivArea
outliers_GrLivArea = df7.loc[(df7['GrLivArea']>4000.0) & (df7['SalePrice']<300000.0)]
outliers_GrLivArea[['GrLivArea' , 'SalePrice']]
```

Out[ ]:

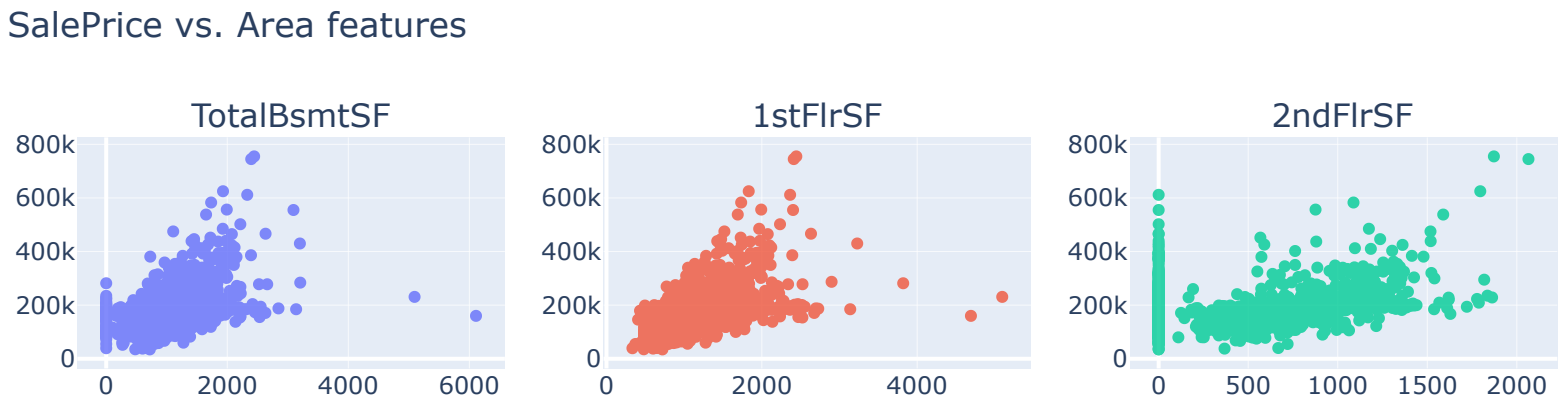
	GrLivArea	SalePrice
523	4676	184750.0000
1298	5642	160000.0000
2549	5095	230841.3386

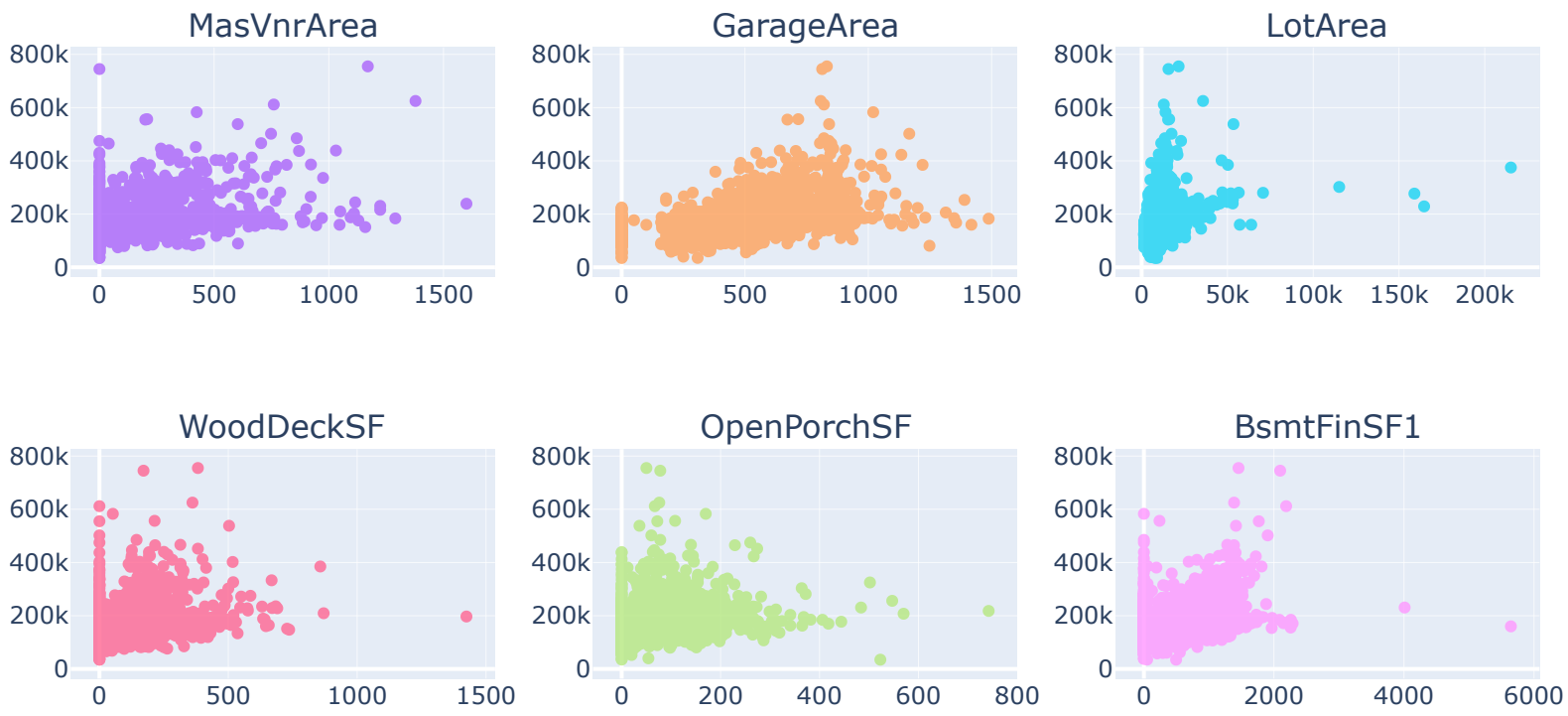
Note on Feature Engineering

Lets look at the Area features and see if we can derive a feature that has even larger correlation to SalePrice

Scatterplots: SalePrice vs Area features

In [ ]:
```
df7=df7
y_col_vals = 'SalePrice'
area_features = ['TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
                 'MasVnrArea', 'GarageArea', 'LotArea',
                 'WoodDeckSF', 'OpenPorchSF', 'BsmtFinSF1']
                # 'ScreenPorch'
x_col_vals = area_features
title='SalePrice' + ' vs. Area features'
fig=template.scatter_multi_col(df7, x_col_vals, y_col_vals,title)
```





new feature : sum of all Living SF areas

all_Liv_SF = 'TotalBsmtSF' + '1stFlrSF' + '2ndFlrSF'

In [ ]:
```
df7['all_Liv_SF'] = df7['TotalBsmtSF'] + df7['1stFlrSF'] + df7['2ndFlrSF']

print(df7['all_Liv_SF'].corr(df7['SalePrice']))
```

```
0.6188919783286284
```

By summming up square feet for Basement, 1st and 2nd floor. we derive a feature 'all_Liv_SF' that has a correlation to SalePrice of 0.62

New feature: Sum of many area features

all_SF = 'all_Liv_SF' + 'GarageArea' + 'MasVnrArea' + 'WoodDeckSF' + 'OpenPorchSF' + 'ScreenPorch'

For 'all_SF' we further add some of the outside area values.

This results in a correlation to SalePrice and also SalePriceLog of around 0.82

In [ ]:
```
df7['all_SF'] = ( df7['all_Liv_SF'] + df7['GarageArea'] + df7['MasVnrArea']
                        + df7['WoodDeckSF'] + df7['OpenPorchSF'] + df7['ScreenPorch'] )

print(df7['all_SF'].corr(df7['SalePrice']))
```

```
0.6363538382426783
```

The two new features are highly correlated. This multicorrelation may be a problem for some linear models. Since, we drop the 'all_Liv_SF' feature.

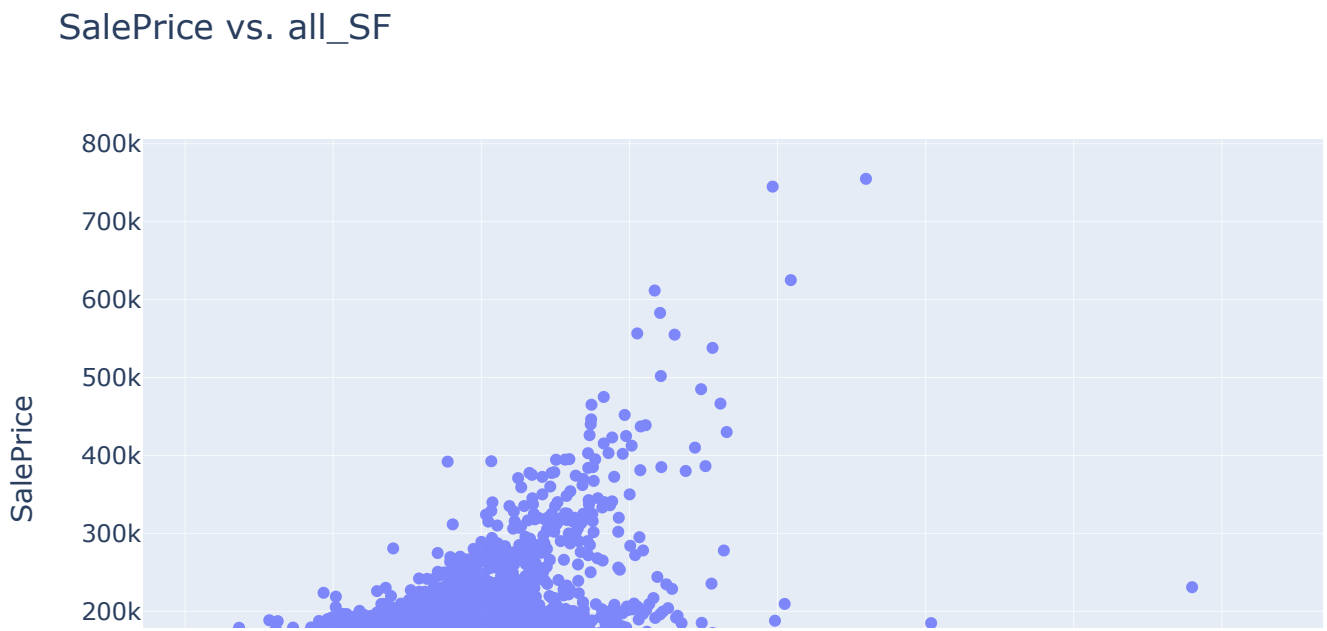
In [ ]:
```
df7['all_SF'].corr(df7['all_Liv_SF'])
```

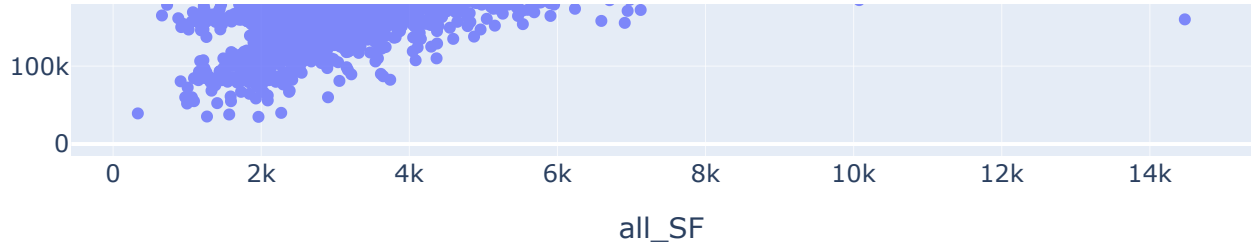
Out[ ]:
```
0.9633782404205463
```

In [ ]:
```
df7=df7.drop('all_Liv_SF',axis=1)
```

Scatterplot: SalePrice vs all_SF

In [ ]:
```
template.plotly_scatter_x_y(df7, 'all_SF', 'SalePrice')
```





Like for GrlivArea, there are two outliers at the lower right also for all_SF

We are going to drop these now.

In [ ]:
```
outliers_allSF = df7.loc[(df7['all_SF']>8000.0) & (df7['SalePrice']<200000.0)]
outliers_allSF[['all_SF' , 'SalePrice']]
```

Out[ ]:

	all_SF	SalePrice
523	10074.0	184750.0
1298	14472.0	160000.0

Indexes for the outliers are the same like for GrLivArea

In [ ]:
```
df7 = df7.drop(outliers_allSF.index)
```

In [ ]:
```
df7.corr().abs()[['SalePrice']].sort_values(by='SalePrice', ascending=False)[2:16]
```

Out[ ]:

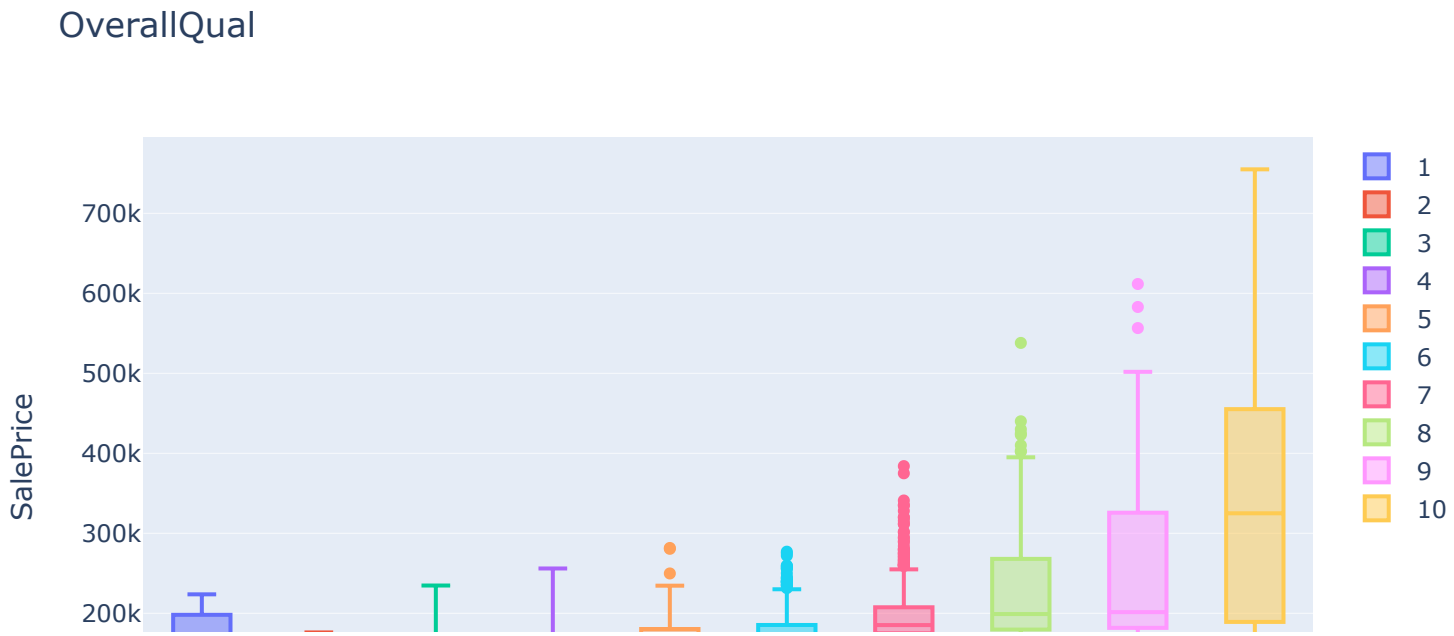
	SalePrice
GrLivArea	0.599867
OverallQual	0.552638
1stFlrSF	0.472519
TotRmsAbvGrd	0.471847
GarageCars	0.469279
TotalBsmtSF	0.467029
GarageArea	0.466857
FullBath	0.434286

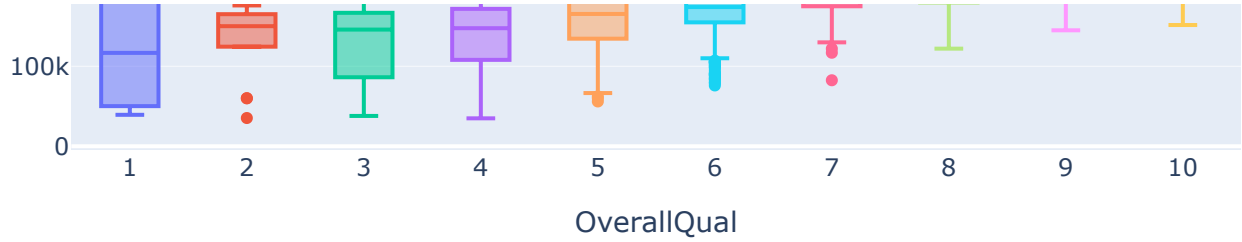
	SalePrice
YearBuilt	0.362364
MasVnrArea	0.355034
Fireplaces	0.354869
YearRemodAdd	0.350299
GarageYrBlt	0.309443
LotArea	0.300360

After dropping these two outliers all_SF has a correlation to SalePrice of 0.86

Boxplot: SalePrice vs. OverallQual

In [ ]:
```
df7=df7
num_feature="OverallQual"
LABEL_COL="SalePrice"
r1=template.plotly_boxplots_num_yvals(df7, num_feature, LABEL_COL)
```





As can be expected from the large correlation coefficient of 0.796, there is an almost perfect linear increase of SalePrice with the OverallQual.

We notice that this feature is in fact categorical (ordinal), only the discrete values 1,2..10 occur. Also there are a few outliers for some of the OverallQual values. We are dropping those that are very far from the upper fences:

In [ ]:
```
outliers_OverallQual_4 = df7.loc[(df7['OverallQual']==4) & (df7['SalePrice']>200000.0)]
outliers_OverallQual_8 = df7.loc[(df7['OverallQual']==8) & (df7['SalePrice']>500000.0)]
outliers_OverallQual_9 = df7.loc[(df7['OverallQual']==9) & (df7['SalePrice']>500000.0)]
outliers_OverallQual_10 = df7.loc[(df7['OverallQual']==10) & (df7['SalePrice']>700000.0)]

outliers_OverallQual = pd.concat([outliers_OverallQual_4, outliers_OverallQual_8,
                                  outliers_OverallQual_9, outliers_OverallQual_10])
```

In [ ]:
```
outliers_OverallQual[['OverallQual' , 'SalePrice']]
```

Out[ ]:

	OverallQual	SalePrice
457	4	256000.0000
1593	4	209258.3369
2570	4	210820.1151
2610	4	222406.6427
769	8	538000.0000
178	9	501837.0000
803	9	582933.0000
898	9	611657.0000
1046	9	556581.0000

	OverallQual	SalePrice
691	10	755000.0000
1182	10	745000.0000

In [ ]:
```
df7 = df7.drop(outliers_OverallQual.index)
```

In [ ]:
```
df7.corr().abs()[['SalePrice']].sort_values(by='SalePrice', ascending=False)[2:16]
```

Out[ ]:

	SalePrice
GrLivArea	0.585532
OverallQual	0.559040
GarageCars	0.478030
GarageArea	0.470927
1stFlrSF	0.462663
TotRmsAbvGrd	0.461699
TotalBsmtSF	0.458444
FullBath	0.430801
YearBuilt	0.372433
YearRemodAdd	0.361750
Fireplaces	0.356486
MasVnrArea	0.337748
GarageYrBlt	0.316689
LotArea	0.297942

Scatterplot colors: SalePrice vs. all_SF and OverallQual

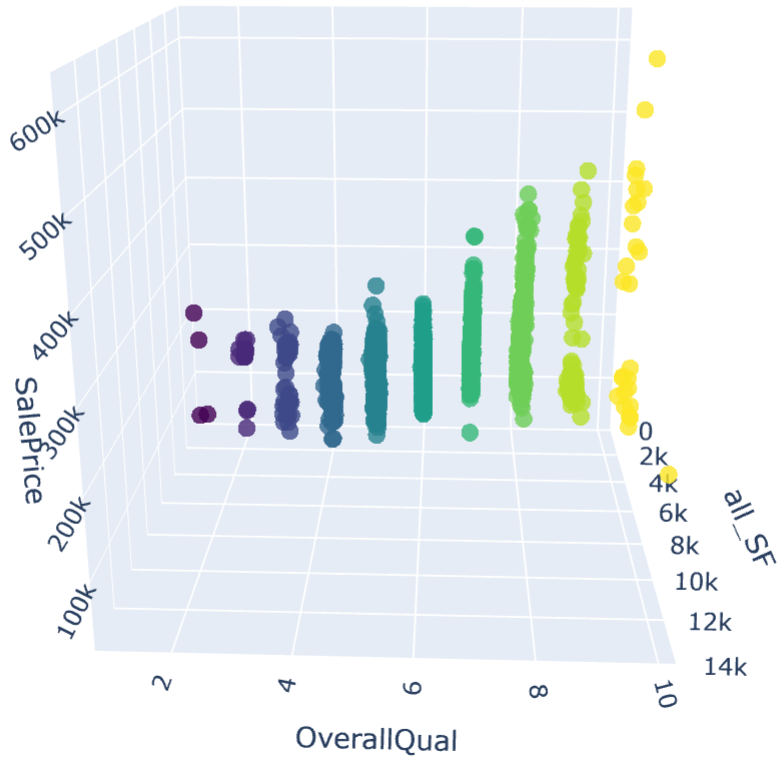
In [ ]:
```
template.plotly_scatter_x_y_catg_color(df7, 'all_SF', 'SalePrice', 'OverallQual')
```

As seen before in the simple scatter plot, there is a strong tendency for increasing SalePrice with a higher value for OverallQual. But this color plot also shows a correlation of all_SF and OverallQual. So, the probability that a house has a large area increases with its Overall Quality. And vice versa: Quality increases with House size.
This corrrelation is not necessary, one would expect that there are also small houses with high quality and big houses with low quality. It would be nice to know how the rating for OverallQual is calculated or estimated, but that info is not included in the data description.

Another option to highlight the correlation of SalePrice to all_SF and OverallQual as well as the correlation between all_SF and OverallQual is

a 3d scatter plot as below.

In [ ]:
```
template.plotly_scatter3d(df7, 'all_SF', 'OverallQual', 'SalePrice')
```



Rotating the 3d view reveals that:

- SalePrice increases almost linearly with all_SF and OverallQual

- all_SF increases almost linearly with OverallQual and vice versa

In fact, the bulk of the data follows the 45 degree line in 3 dim space. This also results in the high correlation coefficient for OverallQual and all_SF.

In [ ]:
```
print(df7['OverallQual'].corr(df7['all_SF']))
```

```
0.7069010268929039
```

other numerical features

In [ ]:
```
print(df7['OverallCond'].corr(df7['SalePrice']))
print(df7['MSSubClass'].corr(df7['SalePrice']))
```

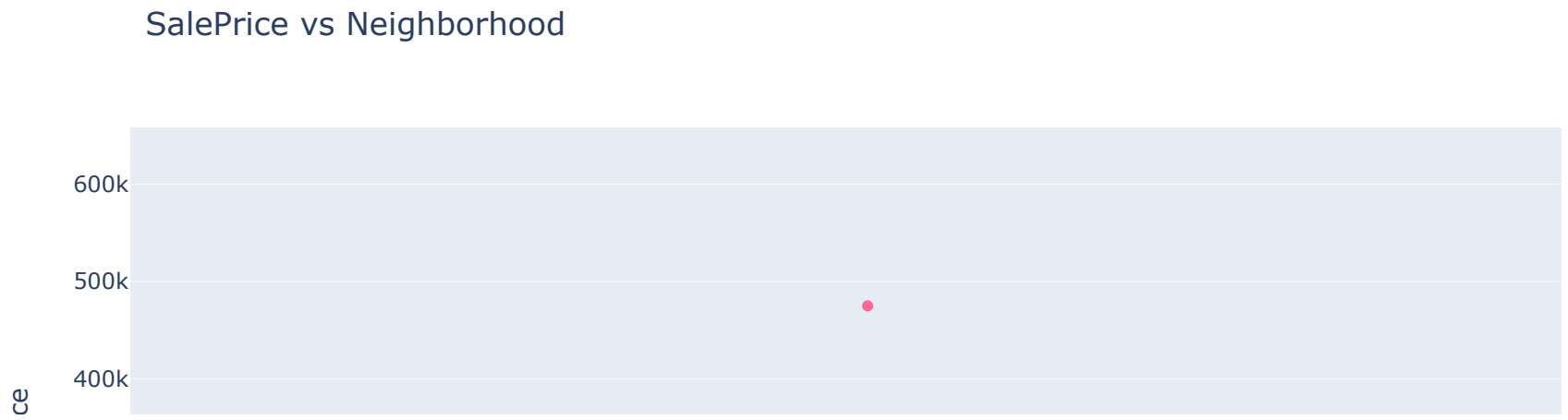
```
-0.06415467388618624
-0.09111261974550716
```

Visualizations for Categorical features

In [ ]:
```
categorical_columns = df7.select_dtypes(include=['object']).columns.tolist()
```

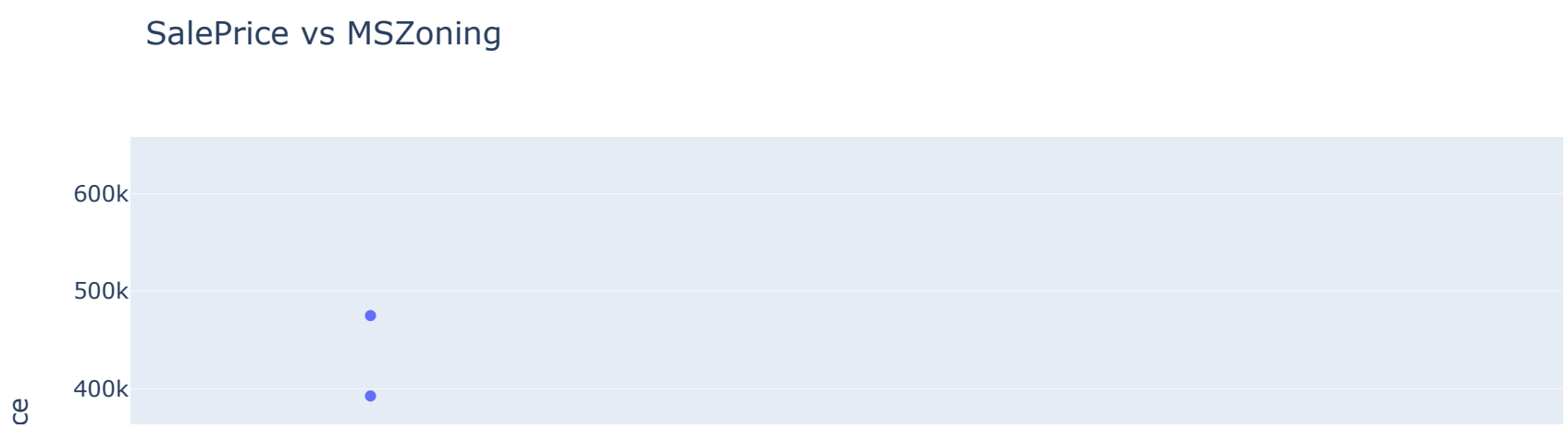
Boxplot: SalePrice for Neighborhood

In [ ]:
```
fig = template.plotly_boxplots_sorted_by_yvals(df7, 'Neighborhood', 'SalePrice')
iplot(fig)
```



Boxplot: SalePrice for MSZoning

In [ ]:
```
fig = template.plotly_boxplots_sorted_by_yvals(df7, 'MSZoning', 'SalePrice')
iplot(fig)
```



Numerical Columns disriptive stats and plots

In [ ]:
```
template.basicanalysis(df7)

template.numcolanalysis(df7)
```

```
Shape is:
 (2906, 76)

 Columns are:
 Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
       'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating', 'HeatingQC',
       'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
       'Functional', 'Fireplaces', 'GarageType', 'GarageYrBlt', 'GarageFinish',
       'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice', 'all_SF'],
      dtype='object')

 Types are:
 MSSubClass         int64
MSZoning           object
LotFrontage       float64
LotArea             int64
Street             object
                   ...
YrSold              int64
SaleType           object
SaleCondition      object
```

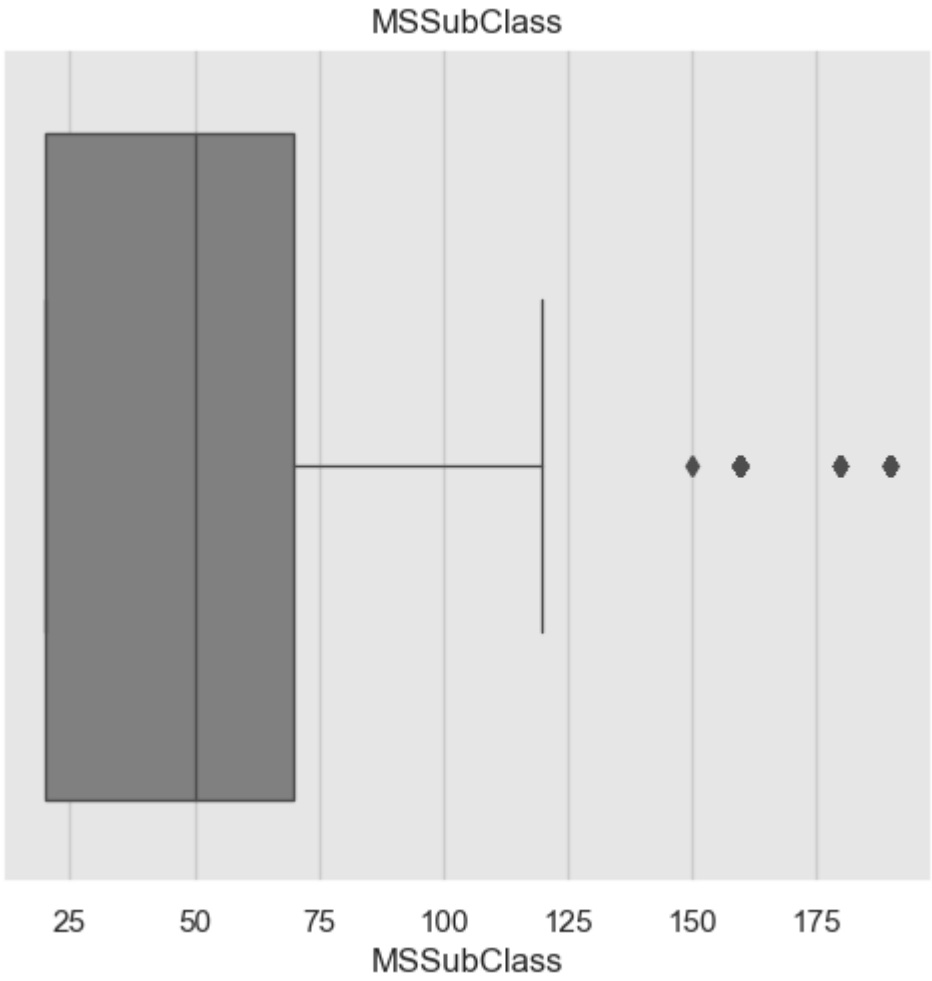
```
SalePrice        float64
all_SF           float64
Length: 76, dtype: object

 Statistical Analysis of Numerical Columns:
         MSSubClass  LotFrontage        LotArea  OverallQual  OverallCond  \
count  2906.000000  2906.000000    2906.000000  2906.000000  2906.000000
mean     57.183414    69.384635   10089.400551     6.081900     5.566758
std      42.580171    21.820506    7724.581404     1.398857     1.114070
min      20.000000    21.000000    1300.000000     1.000000     1.000000
25%      20.000000    59.000000    7455.500000     5.000000     5.000000
50%      50.000000    68.833333    9432.000000     6.000000     5.000000
75%      70.000000    80.000000   11500.000000     7.000000     6.000000
max     190.000000   313.000000  215245.000000    10.000000     9.000000

         YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1   BsmtFinSF2  \
count  2906.000000   2906.000000  2906.000000  2906.000000  2906.000000
mean   1971.213696   1984.207158   100.767034   436.688575    49.786992
std      30.296665     20.898503   176.109776   440.657099   169.524656
min    1872.000000   1950.000000     0.000000     0.000000     0.000000
25%    1953.000000   1965.000000     0.000000     0.000000     0.000000
50%    1973.000000   1993.000000     0.000000   368.000000     0.000000
75%    2001.000000   2004.000000   163.750000   732.000000     0.000000
max    2010.000000   2010.000000  1600.000000  4010.000000  1526.000000

         BsmtUnfSF  TotalBsmtSF     1stFlrSF     2ndFlrSF  LowQualFinSF  \
count  2906.000000  2906.000000  2906.000000  2906.000000   2906.000000
mean    560.073469  1046.549036  1154.957674   333.940124      4.715416
std     438.426583   425.802042   382.413352   424.711046     46.499457
min       0.000000     0.000000   334.000000     0.000000      0.000000
25%     220.000000   792.000000   876.000000     0.000000      0.000000
50%     467.500000   988.000000  1080.500000     0.000000      0.000000
75%     802.750000  1296.750000  1382.000000   702.750000      0.000000
max    2336.000000  5095.000000  5095.000000  1862.000000   1064.000000

         GrLivArea  BsmtFullBath  BsmtHalfBath     FullBath     HalfBath  \
count  2906.000000   2906.000000   2906.000000  2906.000000  2906.000000
mean   1493.613214      0.428940      0.061425     1.564350     0.379215
std     488.650900      0.523978      0.245641     0.549281     0.502697
min     334.000000      0.000000      0.000000     0.000000     0.000000
25%    1124.250000      0.000000      0.000000     1.000000     0.000000
50%    1441.500000      0.000000      0.000000     2.000000     0.000000
75%    1740.000000      1.000000      0.000000     2.000000     1.000000
max    5095.000000      3.000000      2.000000     4.000000     2.000000

       BedroomAbvGr  KitchenAbvGr  TotRmsAbvGrd   Fireplaces  GarageYrBlt  \
count   2906.000000   2906.000000   2906.000000  2906.000000  2906.000000
mean       2.857880      1.044391      6.436683     0.594288  1977.510438
std        0.819786      0.214190      1.550842     0.643427    25.505530
```

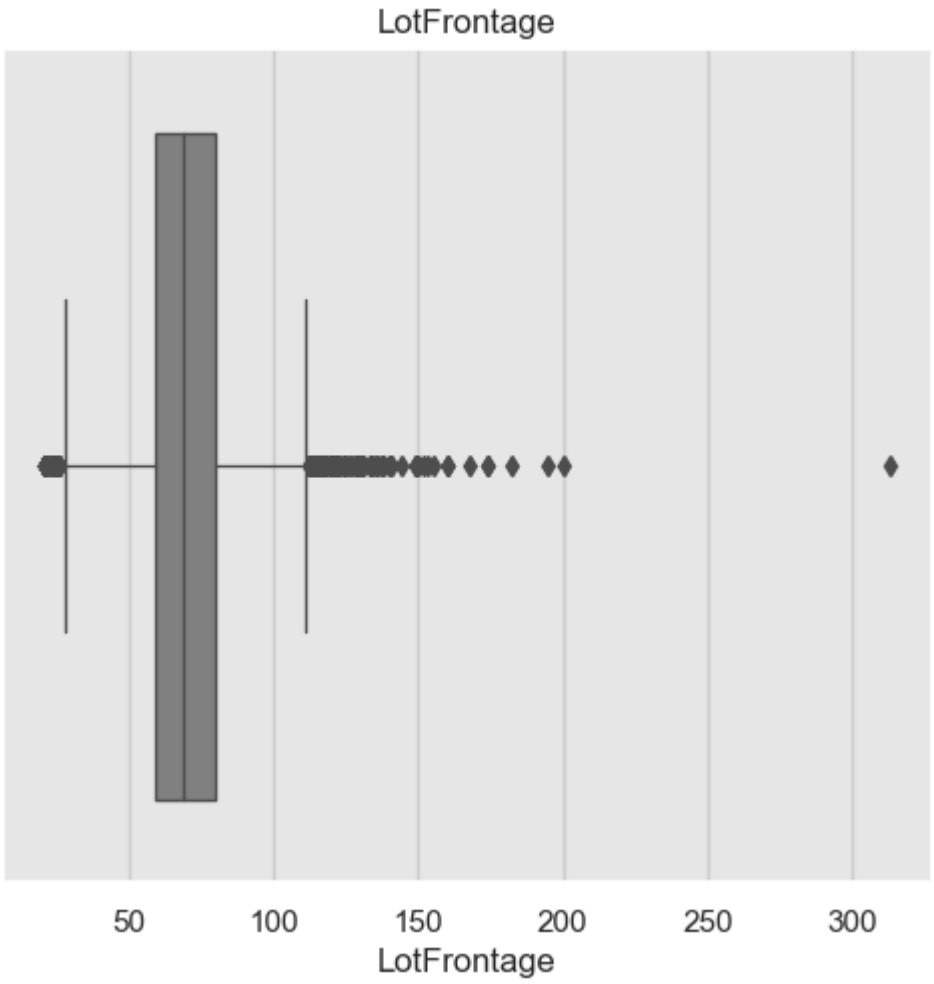
min	0.000000	0.000000	2.000000	0.000000	1895.000000
25%	2.000000	1.000000	5.000000	0.000000	1960.000000
50%	3.000000	1.000000	6.000000	1.000000	1978.000000
75%	3.000000	1.000000	7.000000	1.000000	2001.000000
max	8.000000	3.000000	15.000000	4.000000	2207.000000

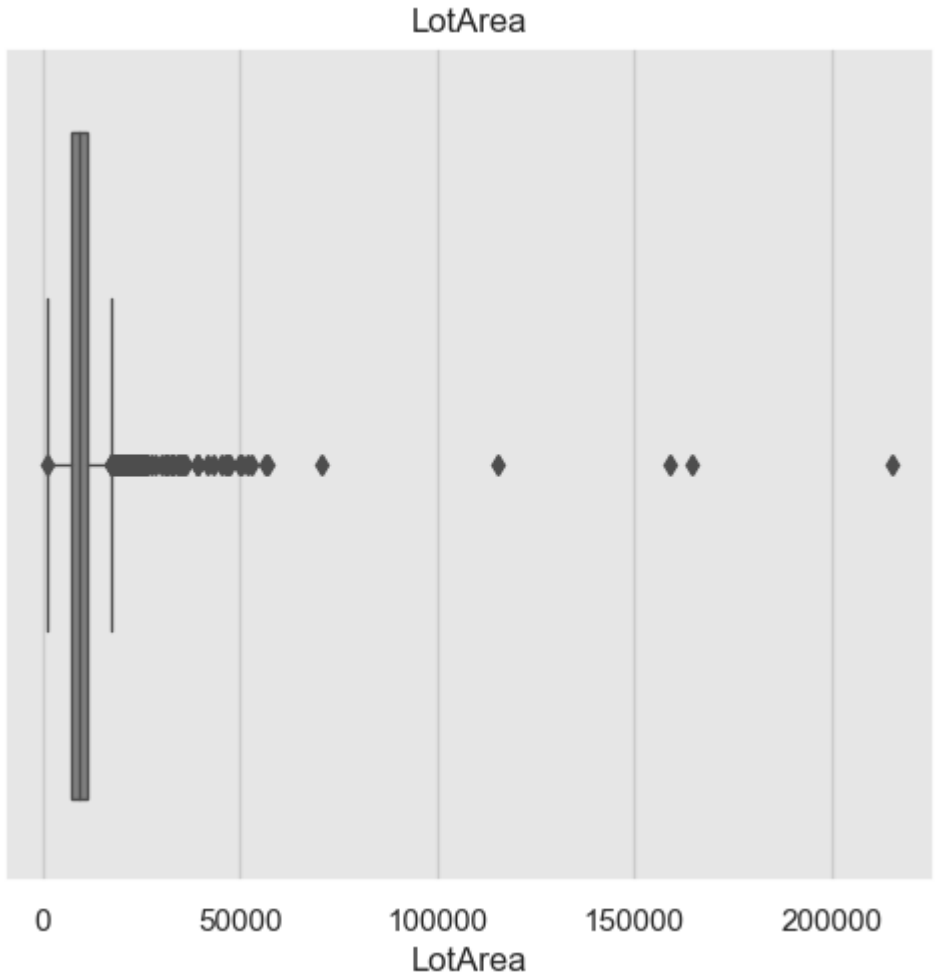
	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	\
count	2906.000000	2906.000000	2906.000000	2906.000000	2906.000000	
mean	1.763421	471.365107	93.347213	47.203028	23.201652	
std	0.759926	213.851421	126.332287	67.161403	64.369207	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	320.000000	0.000000	0.000000	0.000000	
50%	2.000000	478.000000	0.000000	26.000000	0.000000	
75%	2.000000	576.000000	168.000000	70.000000	0.000000	
max	5.000000	1488.000000	1424.000000	742.000000	1012.000000	

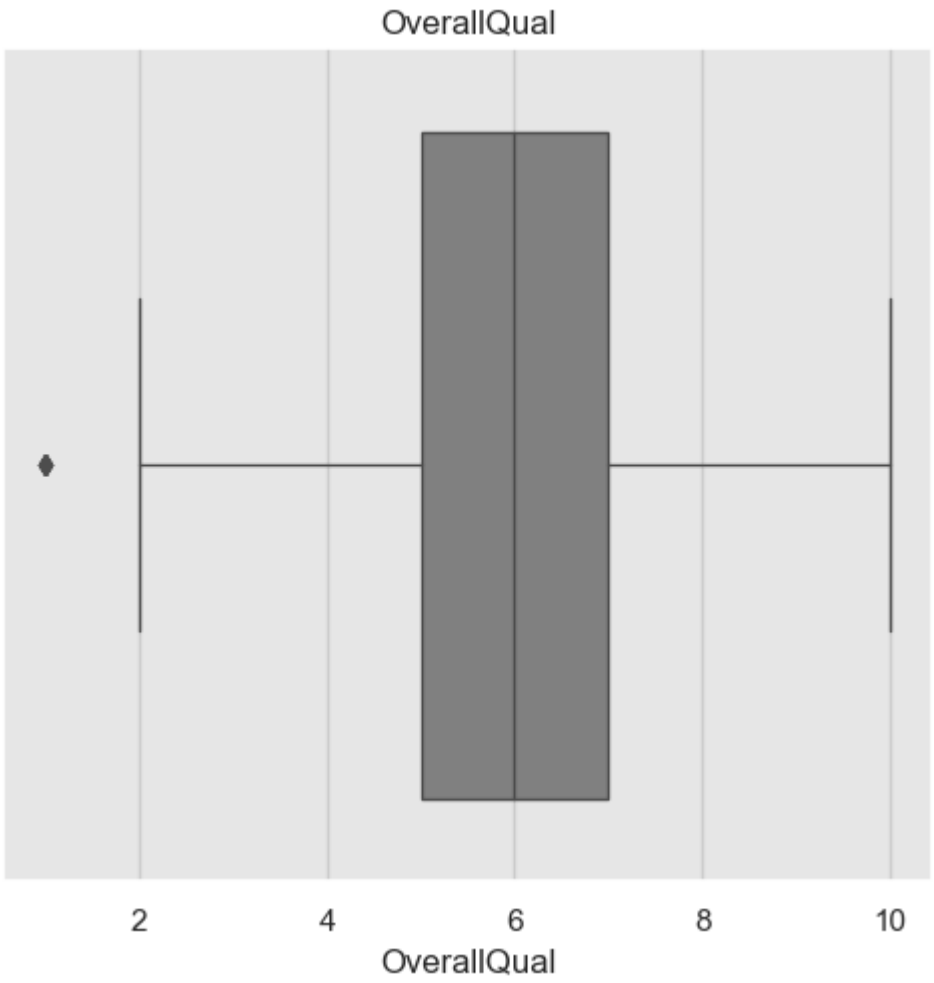
	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	\
count	2906.000000	2906.000000	2906.000000	2906.000000	2906.000000	
mean	2.613902	15.995871	1.905712	50.881280	6.218513	
std	25.243863	56.091221	33.071540	568.599914	2.710739	
min	0.000000	0.000000	0.000000	0.000000	1.000000	
25%	0.000000	0.000000	0.000000	0.000000	4.000000	
50%	0.000000	0.000000	0.000000	0.000000	6.000000	
75%	0.000000	0.000000	0.000000	0.000000	8.000000	
max	508.000000	576.000000	800.000000	17000.000000	12.000000	

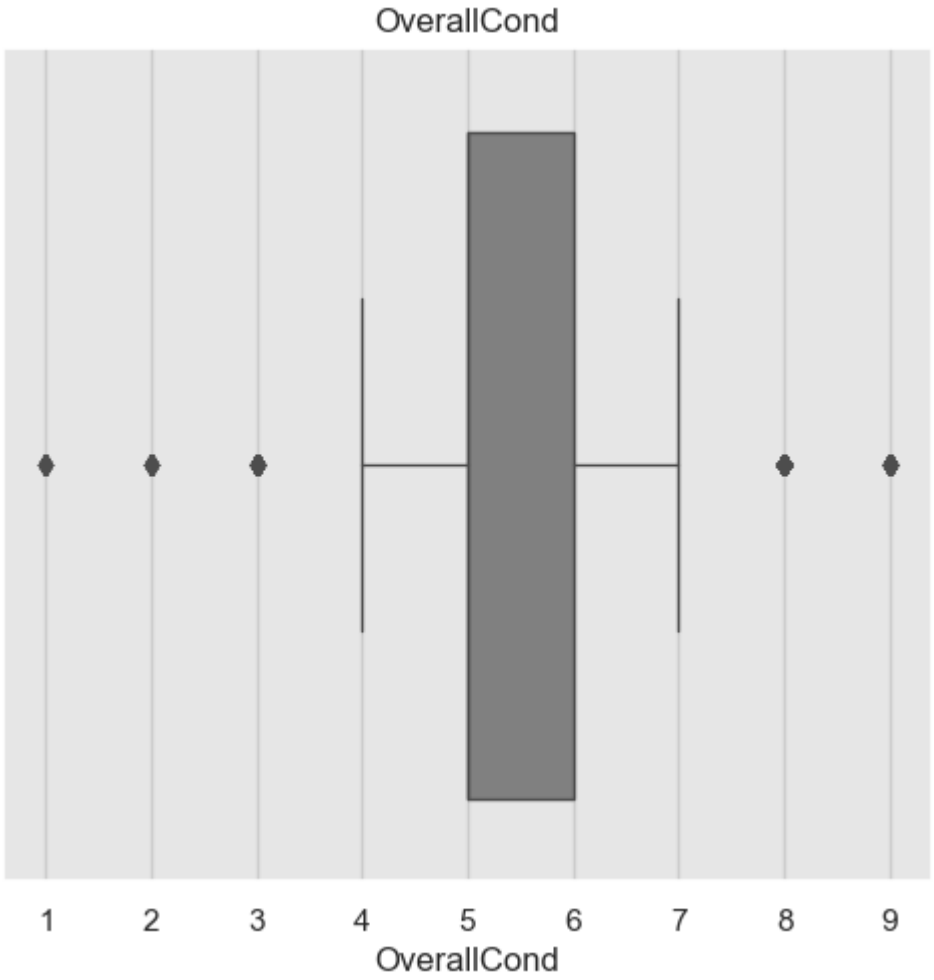
	YrSold	SalePrice	all_SF
count	2906.000000	2906.000000	2906.000000
mean	2007.791466	178953.902141	3264.125086
std	1.314768	53203.314151	1077.273766
min	2006.000000	34900.000000	334.000000
25%	2007.000000	154537.549750	2489.250000
50%	2008.000000	176520.198650	3099.500000
75%	2009.000000	191687.740325	3862.000000
max	2010.000000	625000.000000	13598.000000

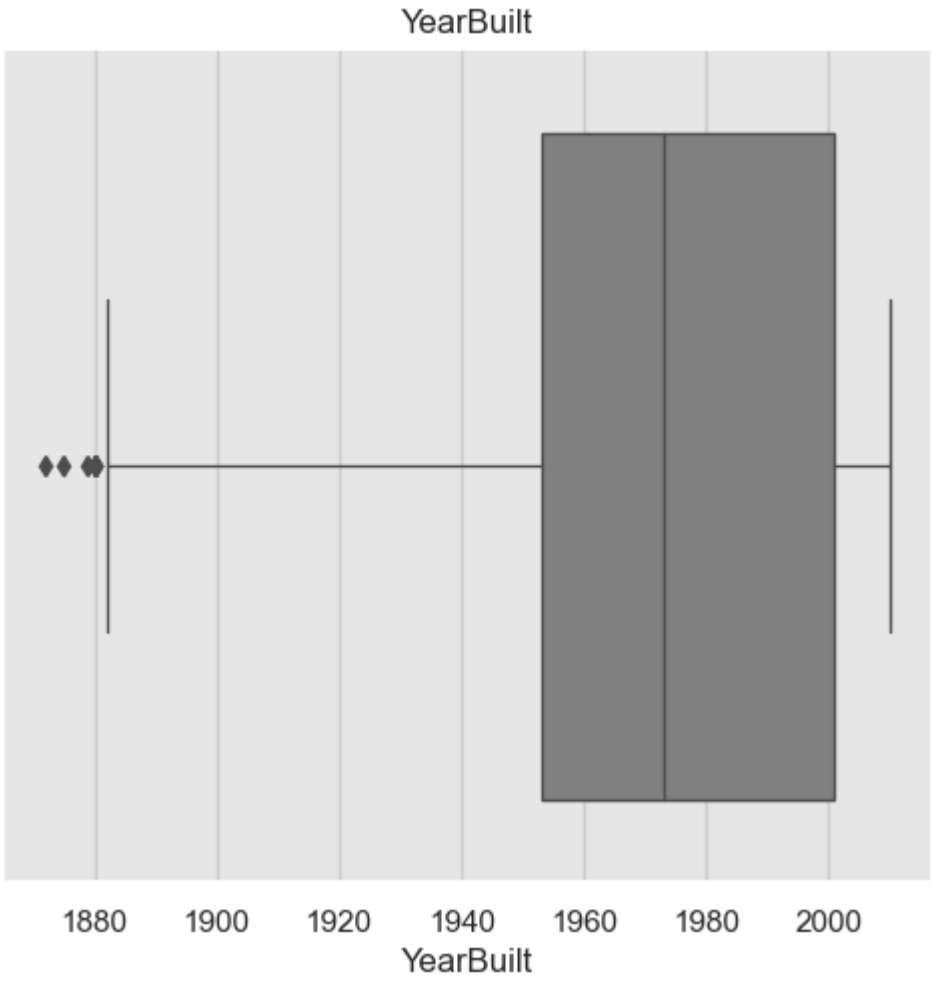

MSSubClass
25
50
75
100
125
150
175
MSSubClass

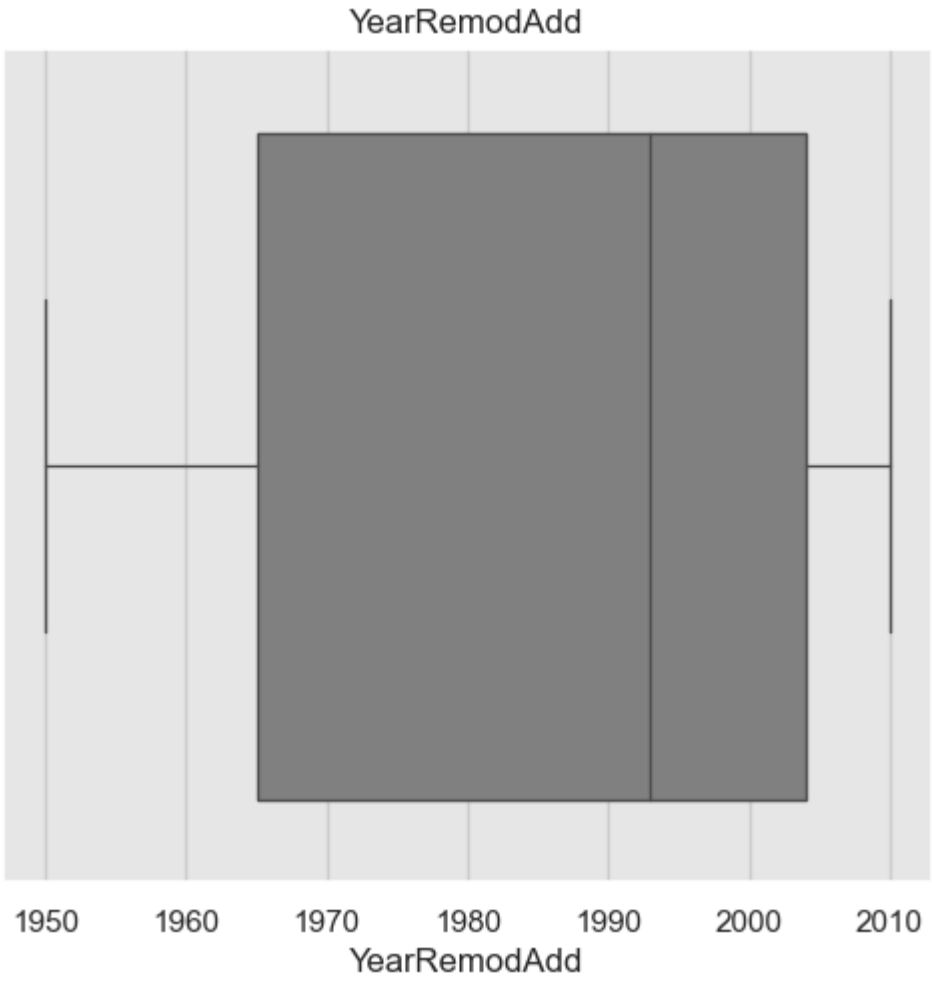

LotFrontage
50
100
150
200
250
300
LotFrontage

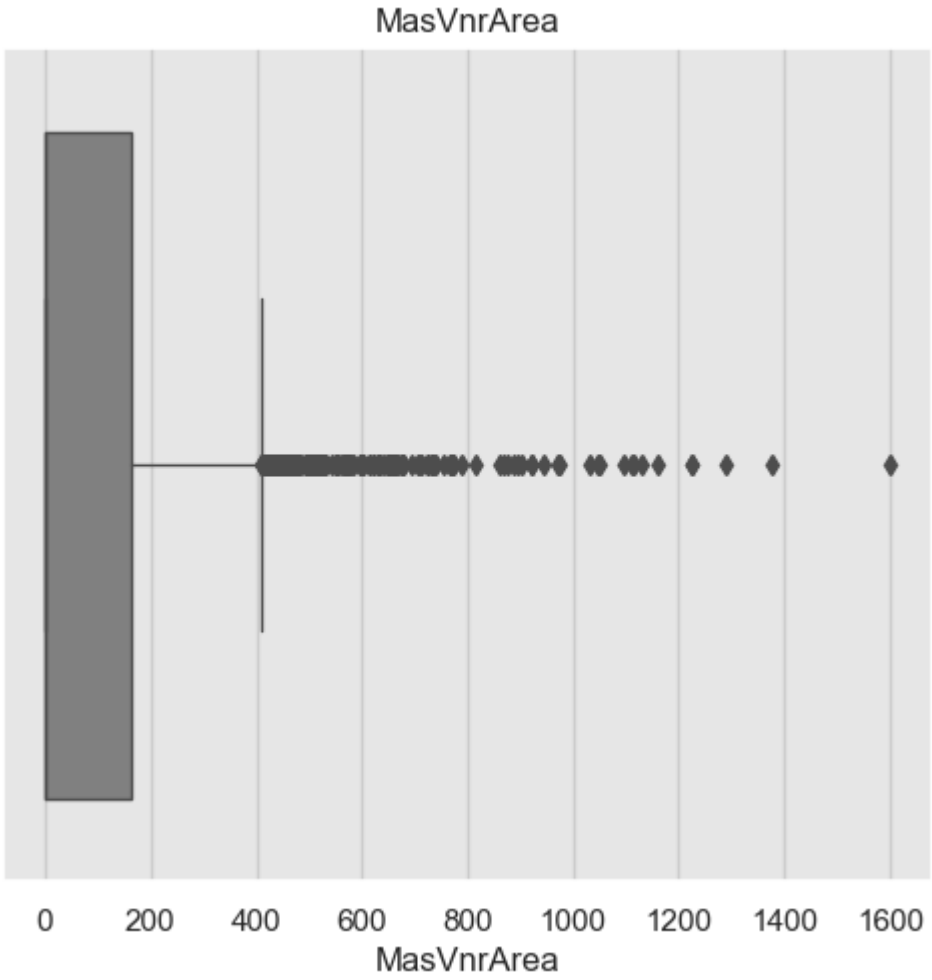

LotArea
0
50000
100000
150000
200000
LotArea

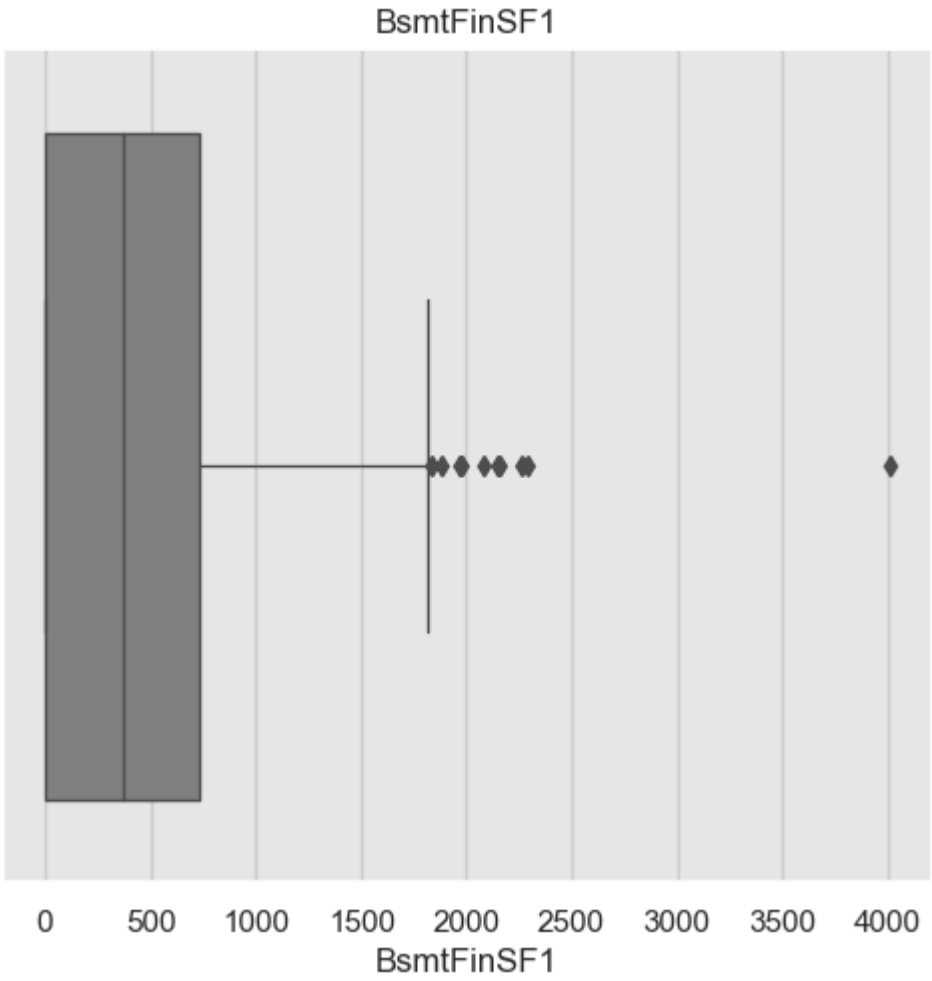

OverallQual
2
4
6
8
10
OverallQual

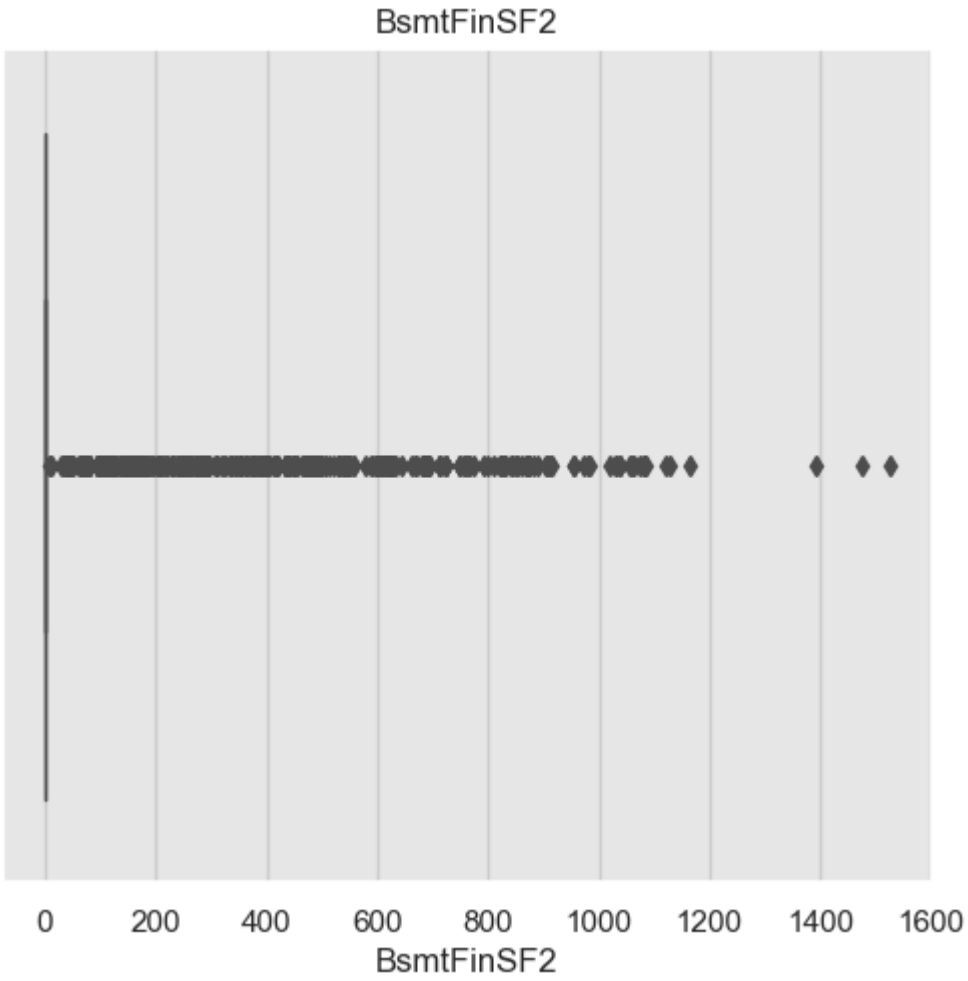

OverallCond
1
2
3
4
5
6
7
8
9
OverallCond

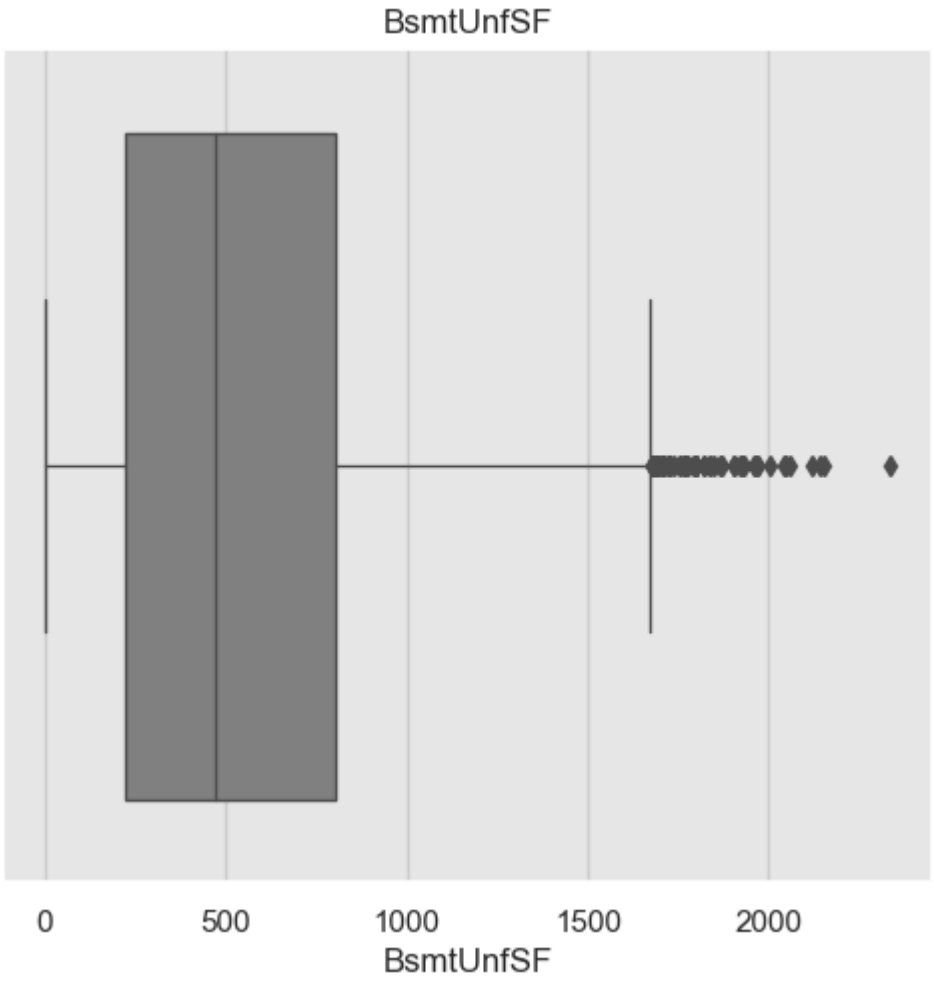

YearBuilt
1880
1900
1920
1940
1960
1980
2000
YearBuilt

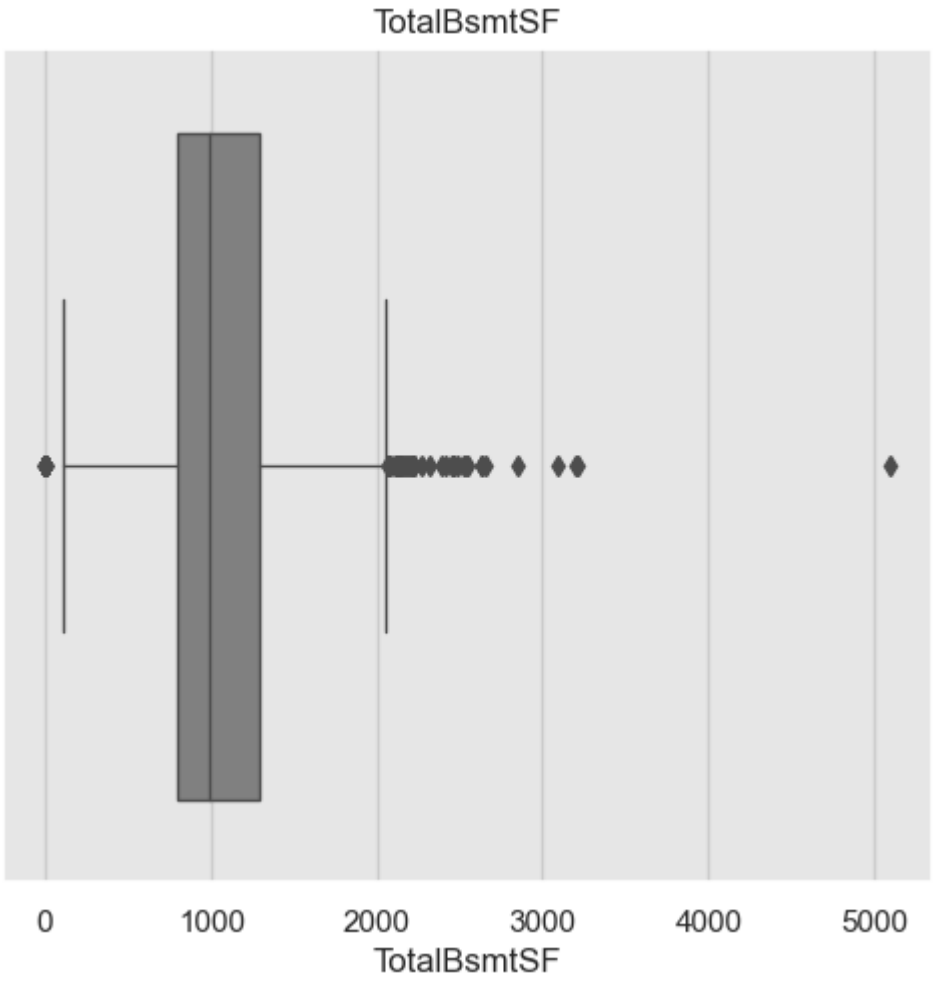

YearRemodAdd
1950
1960
1970
1980
1990
2000
2010
YearRemodAdd

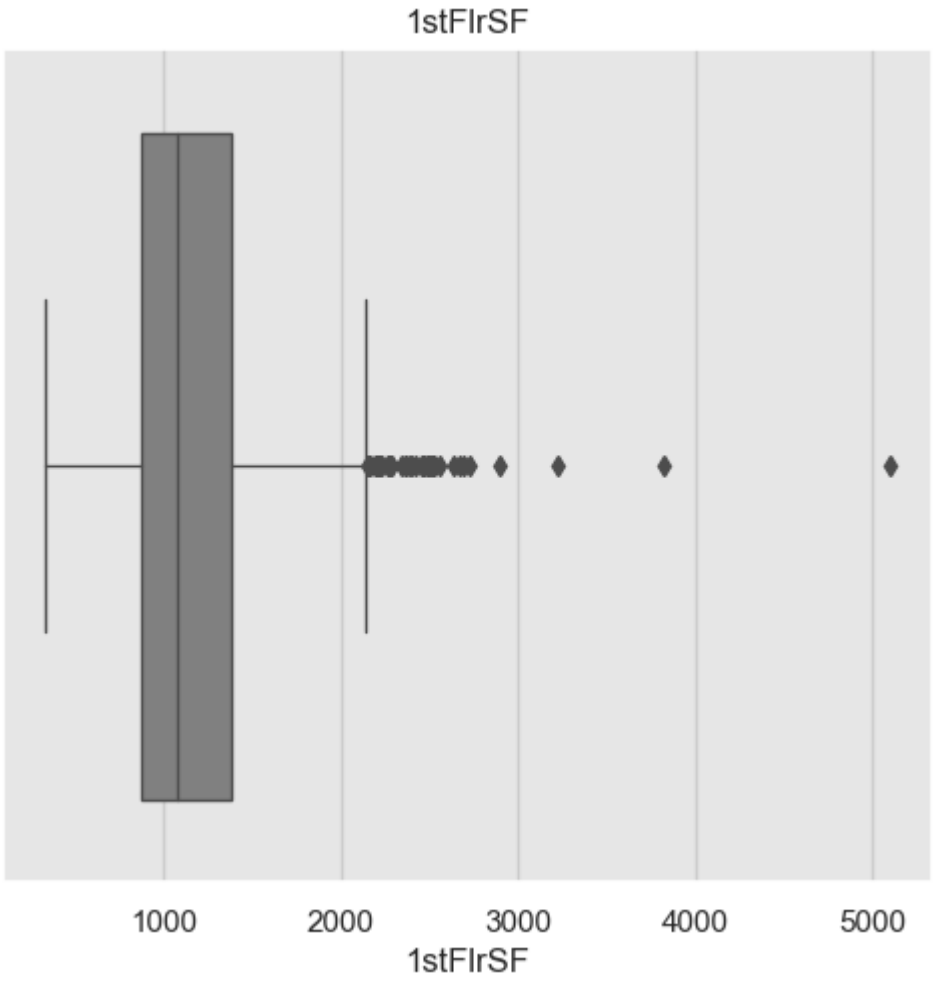

MasVnrArea
0
200
400
600
800
1000
1200
1400
1600
MasVnrArea

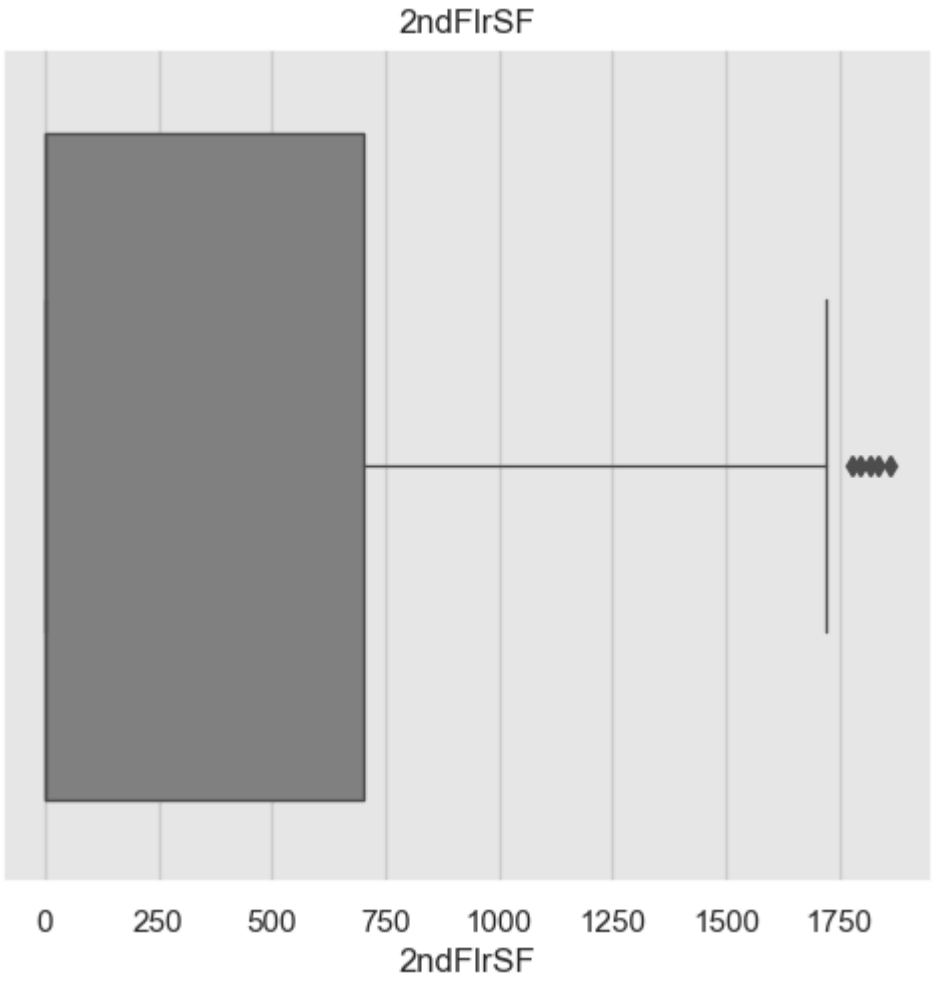

BsmtFinSF1
0
500
1000
1500
2000
2500
3000
3500
4000
BsmtFinSF1

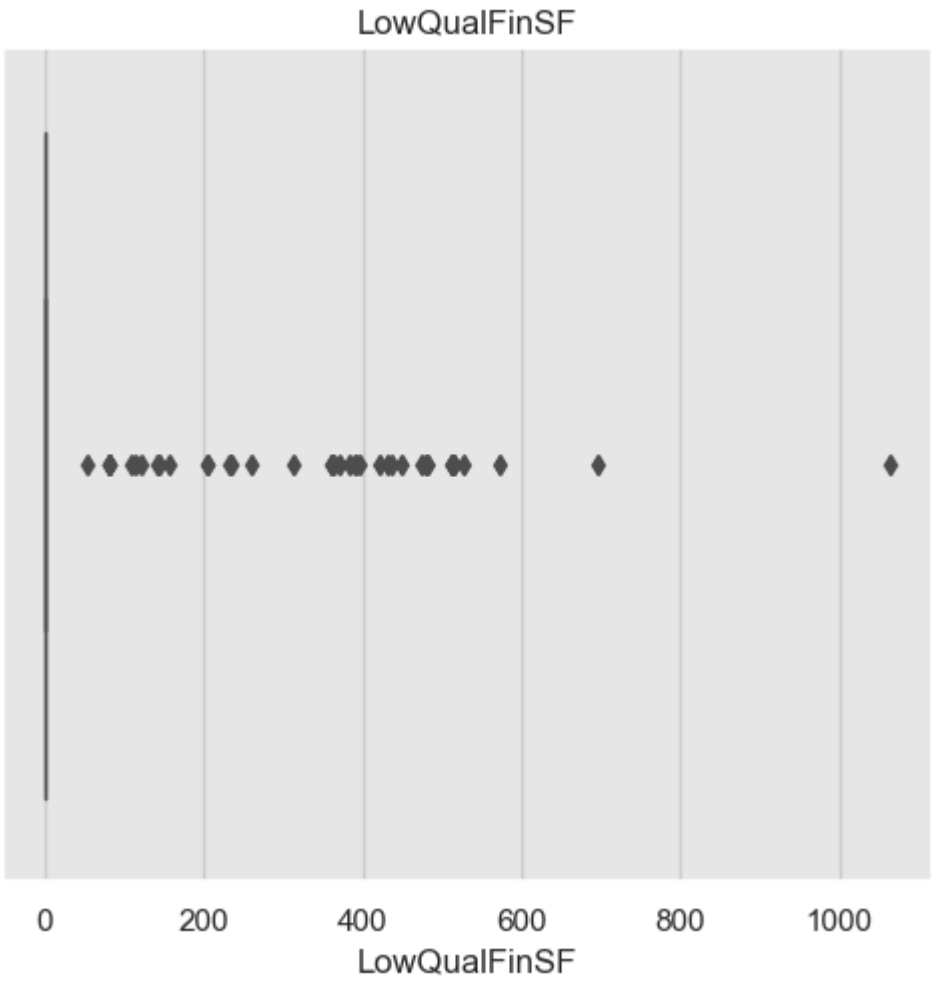

BsmtFinSF2
0
200
400
600
800
1000
1200
1400
1600
BsmtFinSF2

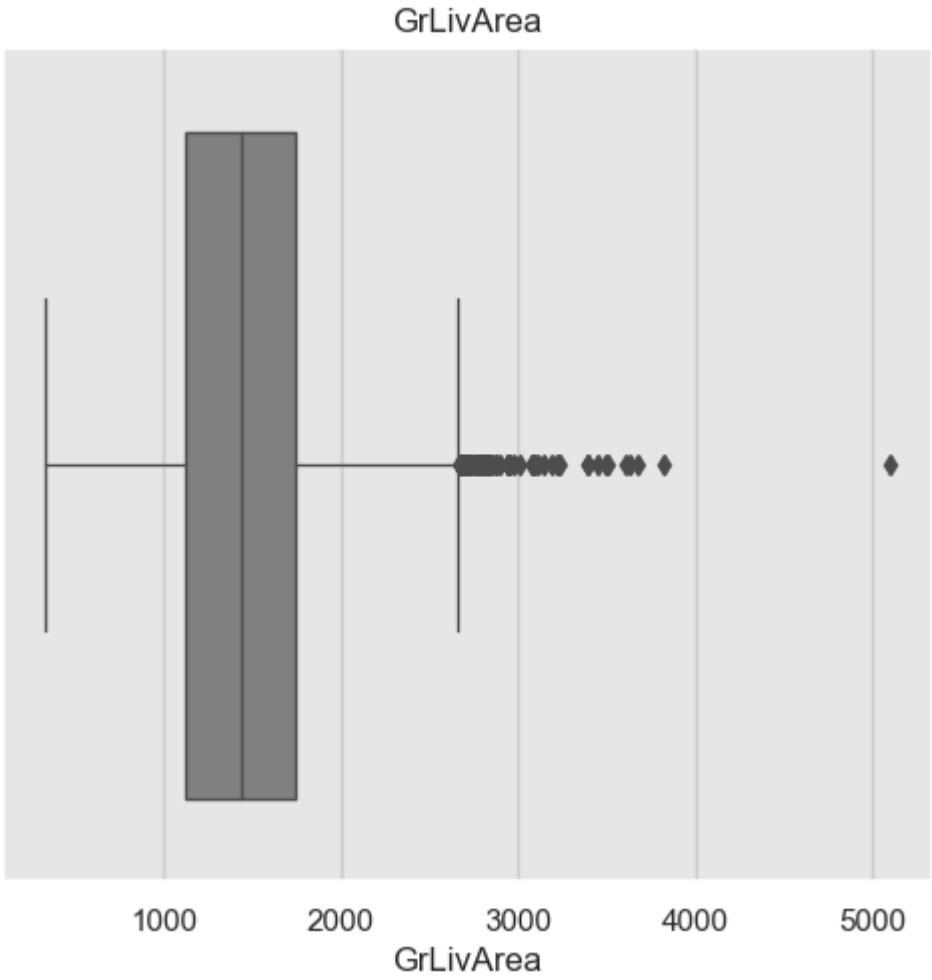

BsmtUnfSF
0
500
1000
1500
2000
BsmtUnfSF

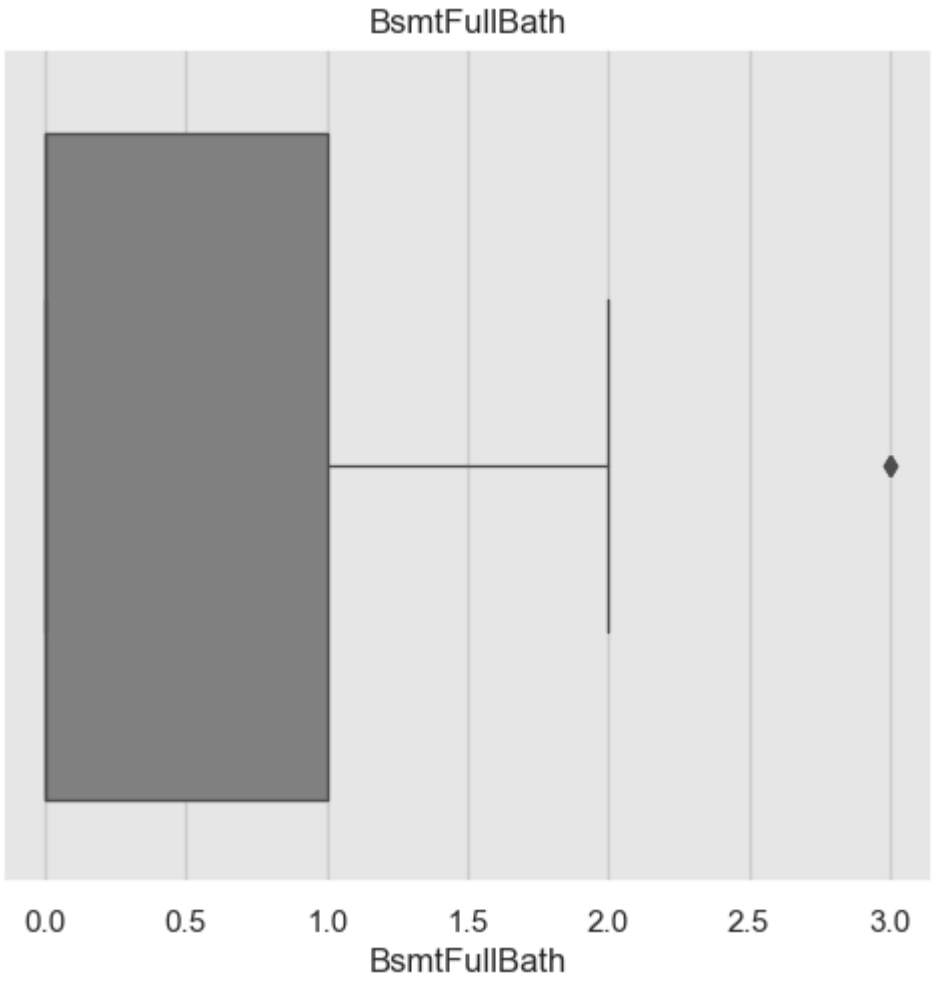

TotalBsmtSF
0
1000
2000
3000
4000
5000
TotalBsmtSF

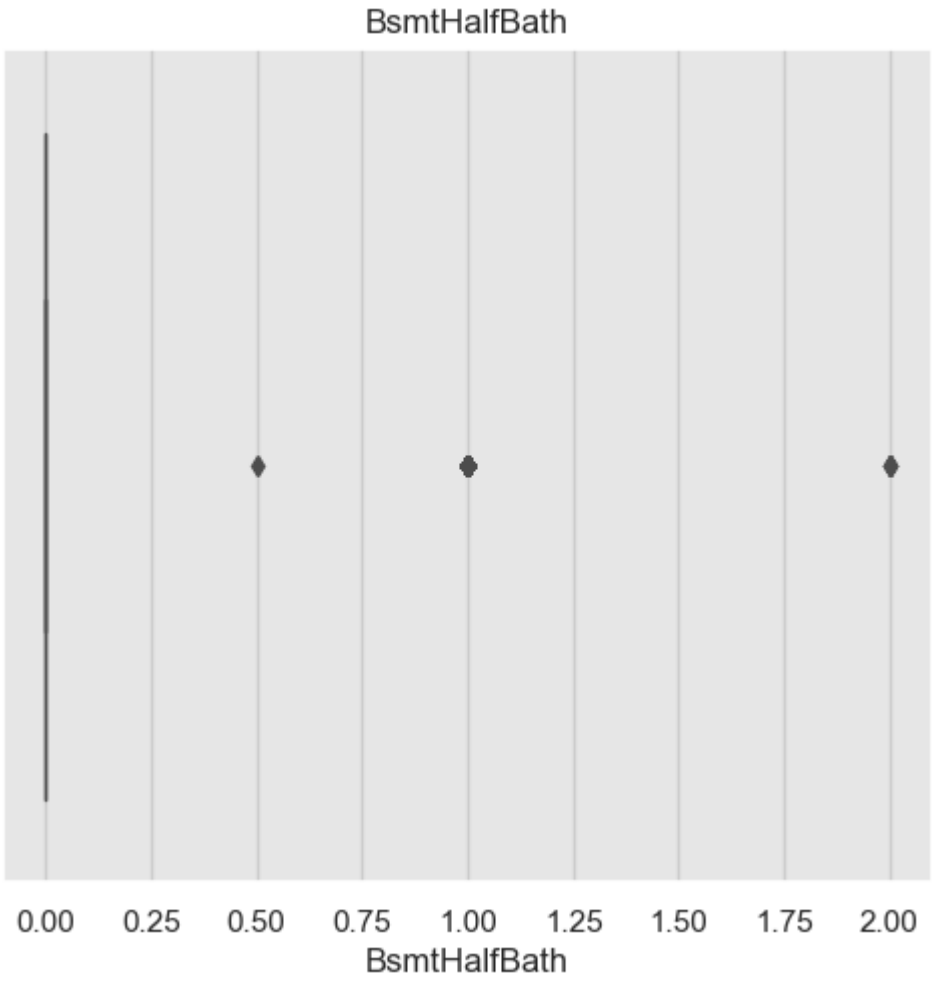

1stFlrSF
1000
2000
3000
4000
5000
1stFlrSF

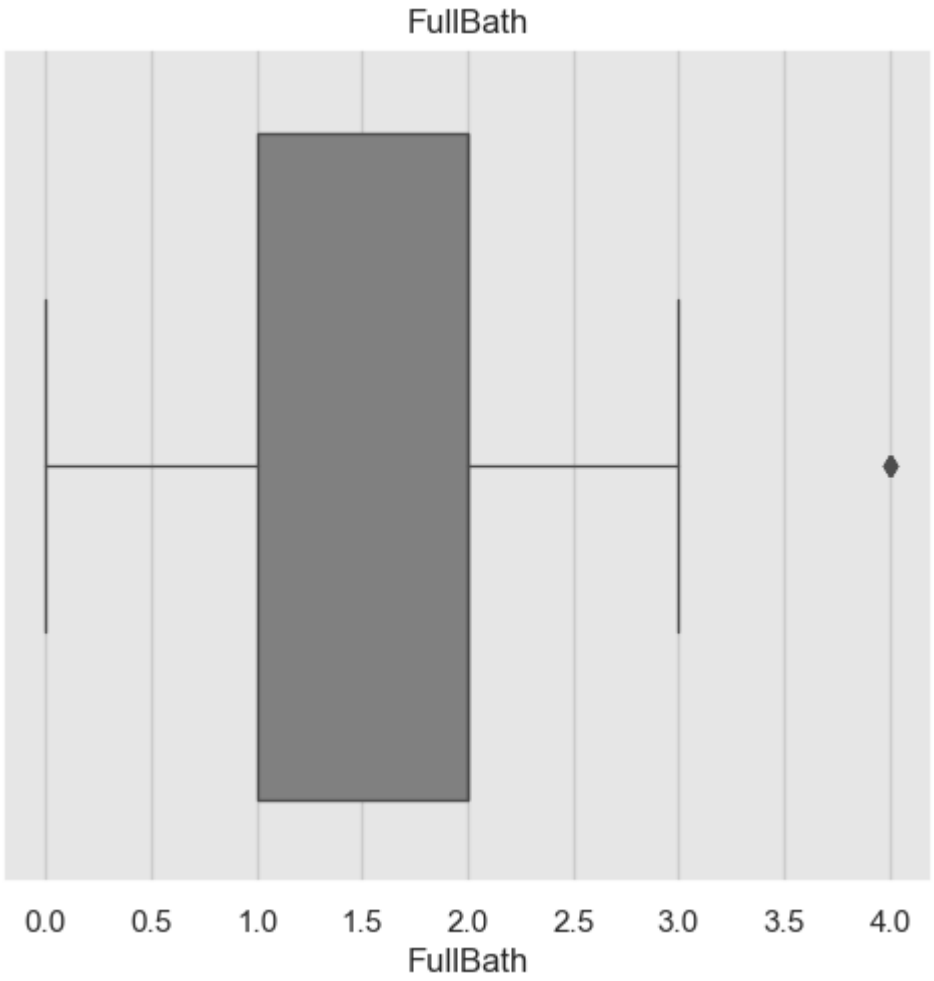

2ndFlrSF
0
250
500
750
1000
1250
1500
1750
2ndFlrSF

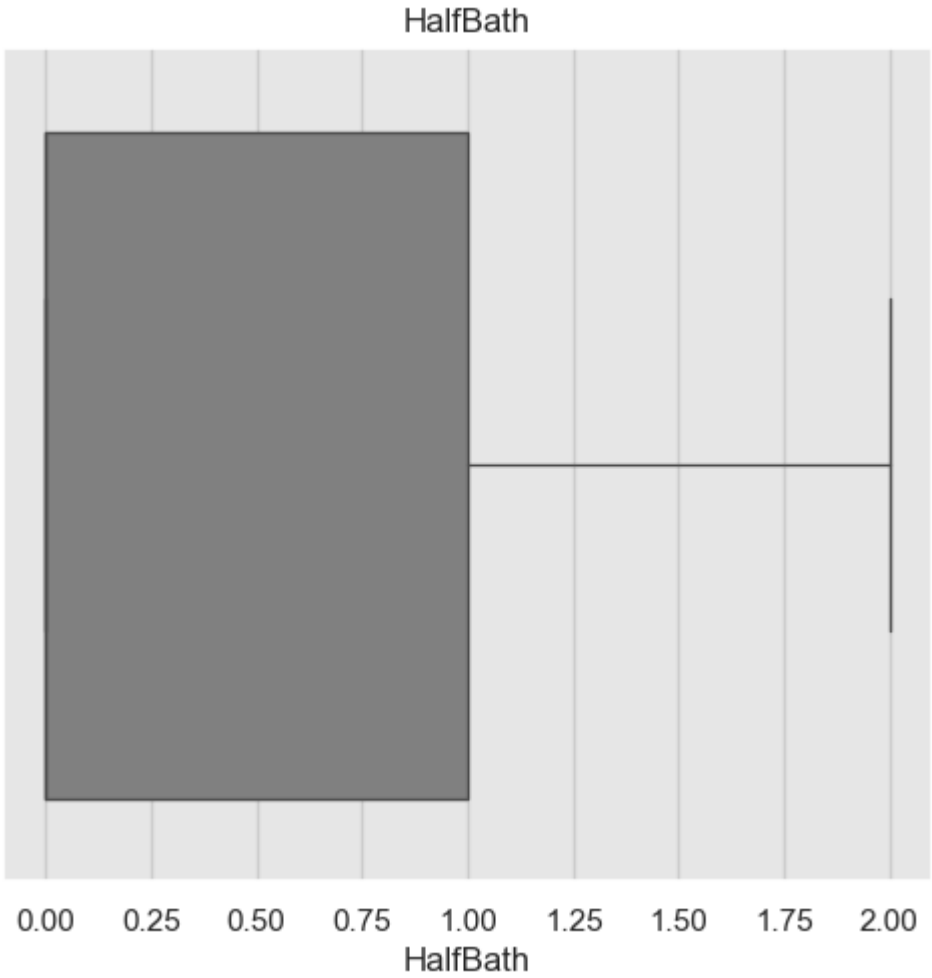

LowQualFinSF
0
200
400
600
800
1000
LowQualFinSF

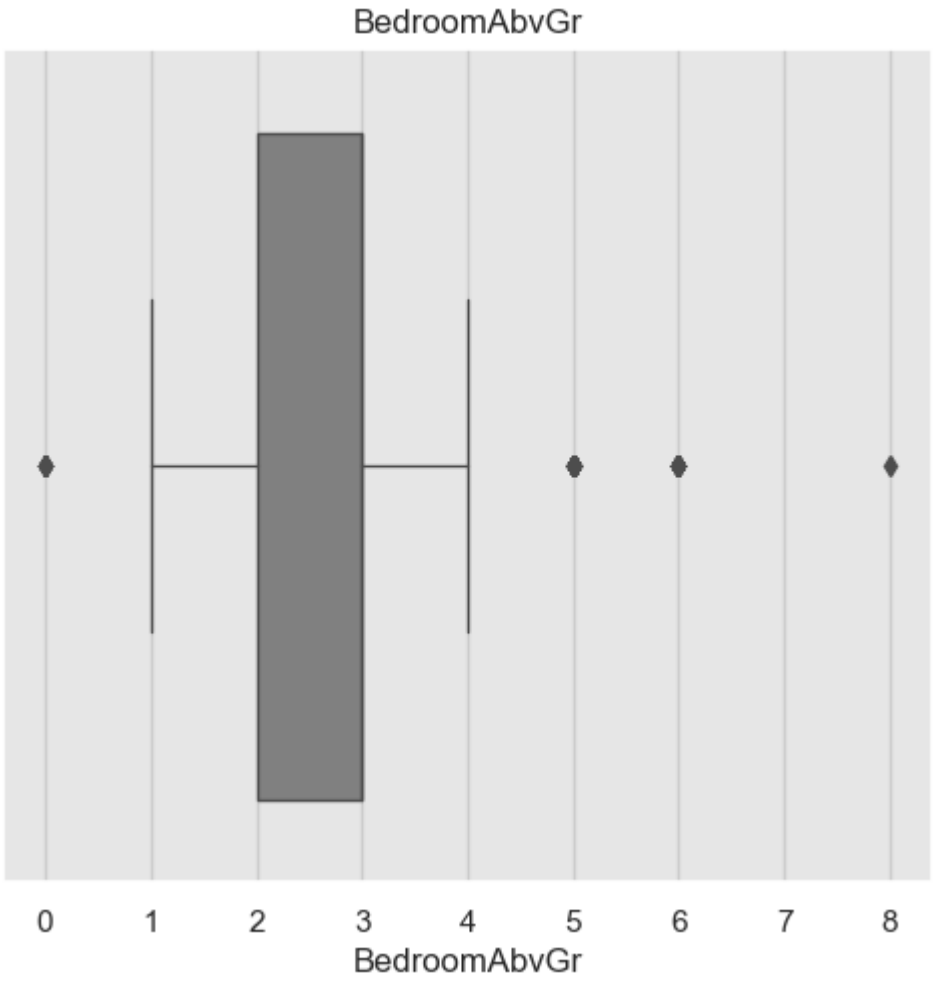

GrLivArea
1000
2000
3000
4000
5000
GrLivArea

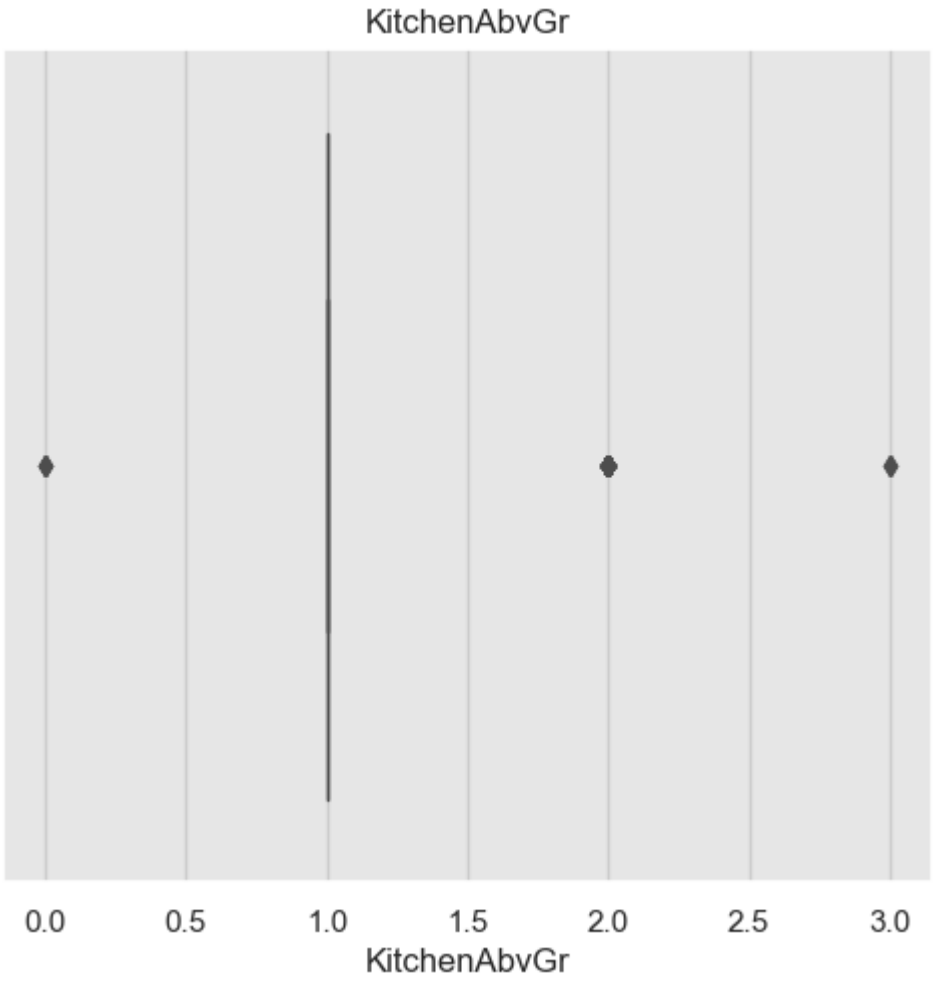

BsmtFullBath
0.0
0.5
1.0
1.5
2.0
2.5
3.0
BsmtFullBath

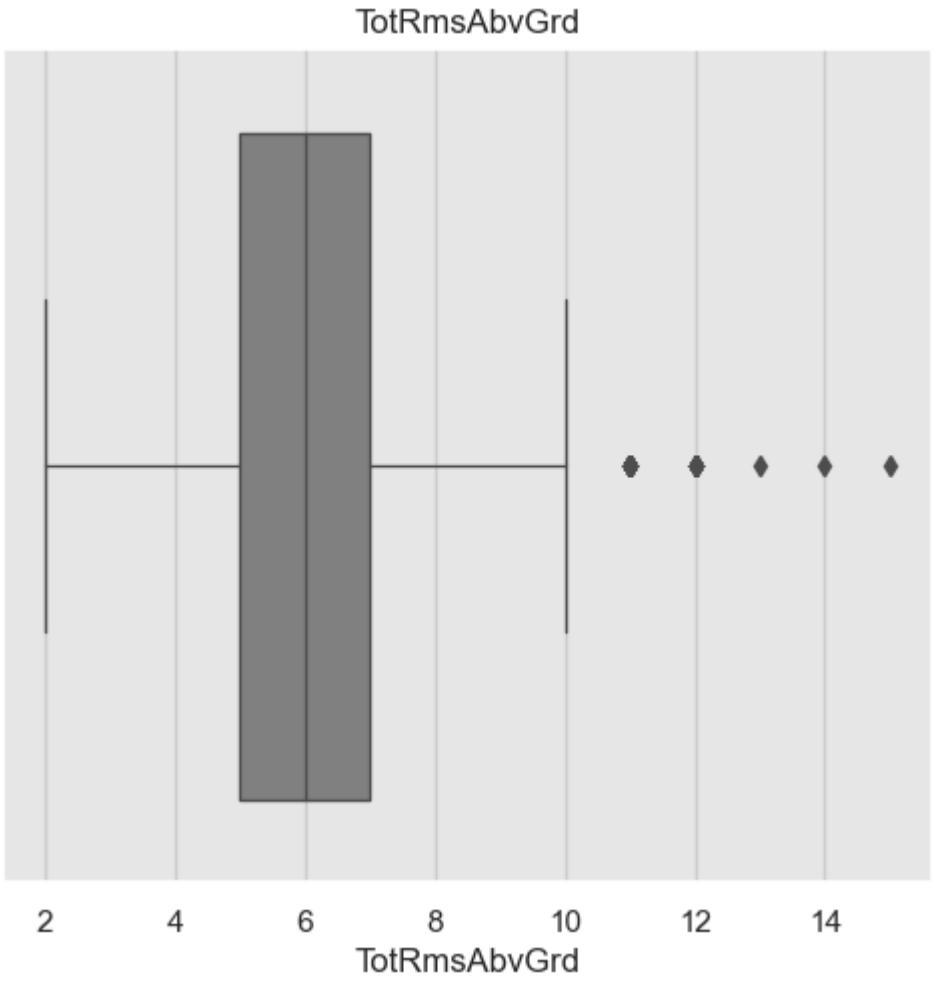

BsmtHalfBath
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
BsmtHalfBath

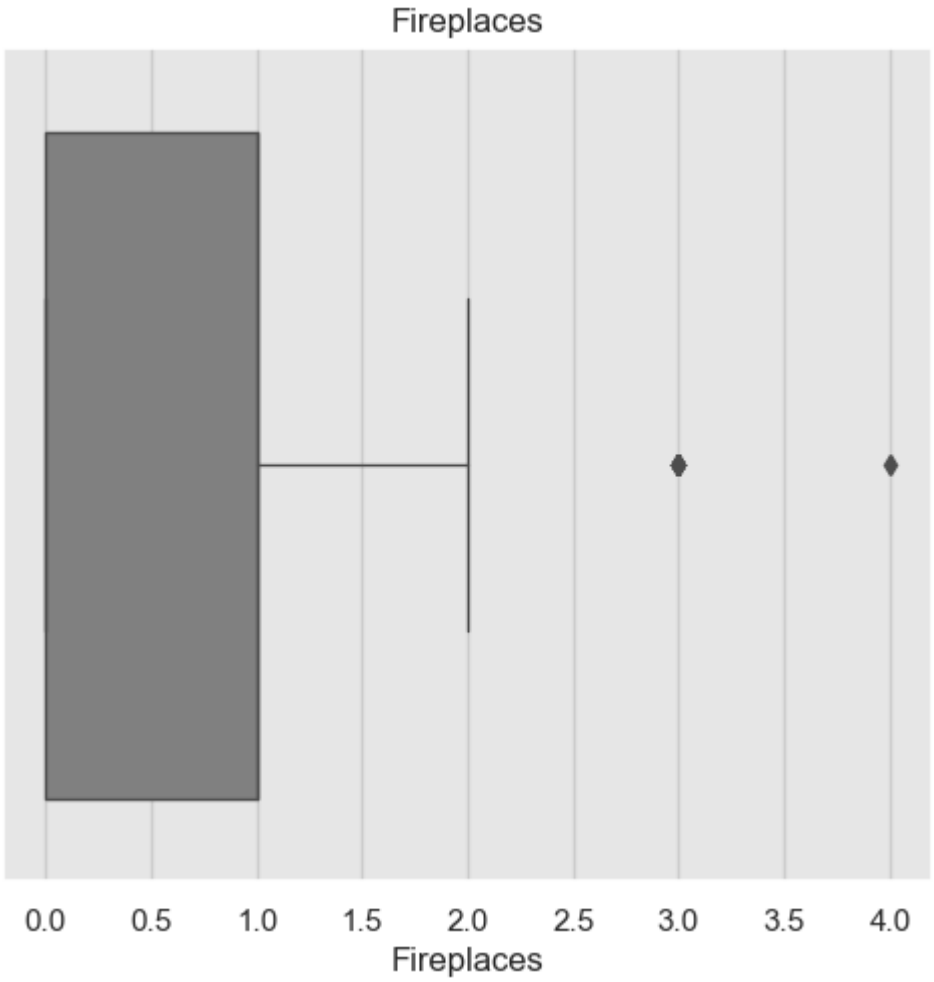

FullBath
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
FullBath

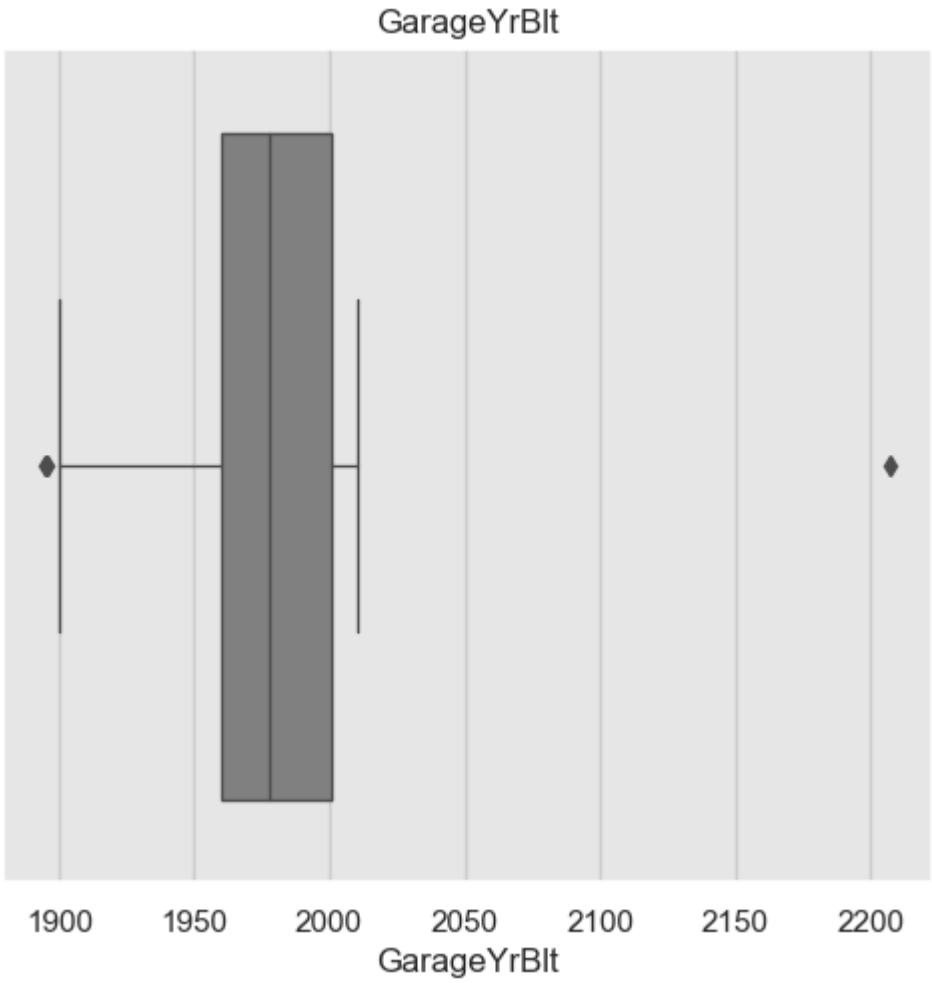

HalfBath
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
HalfBath

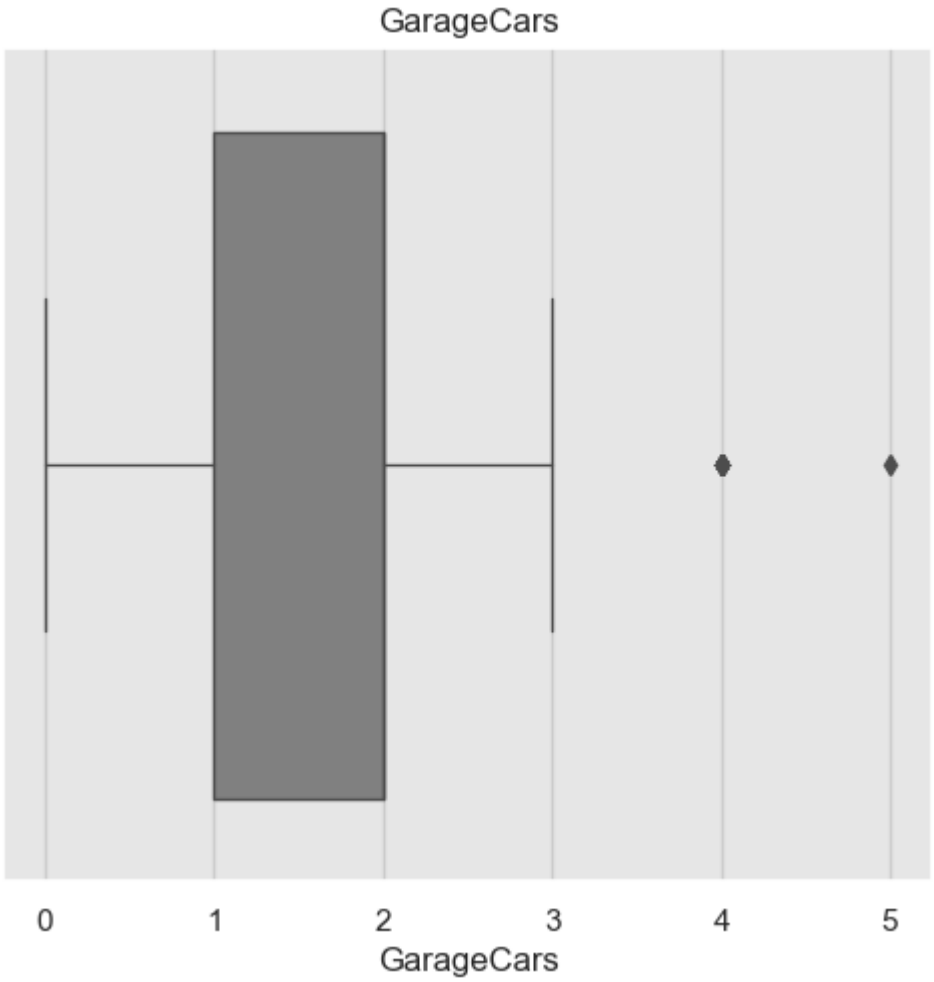

BedroomAbvGr
0
1
2
3
4
5
6
7
8
BedroomAbvGr

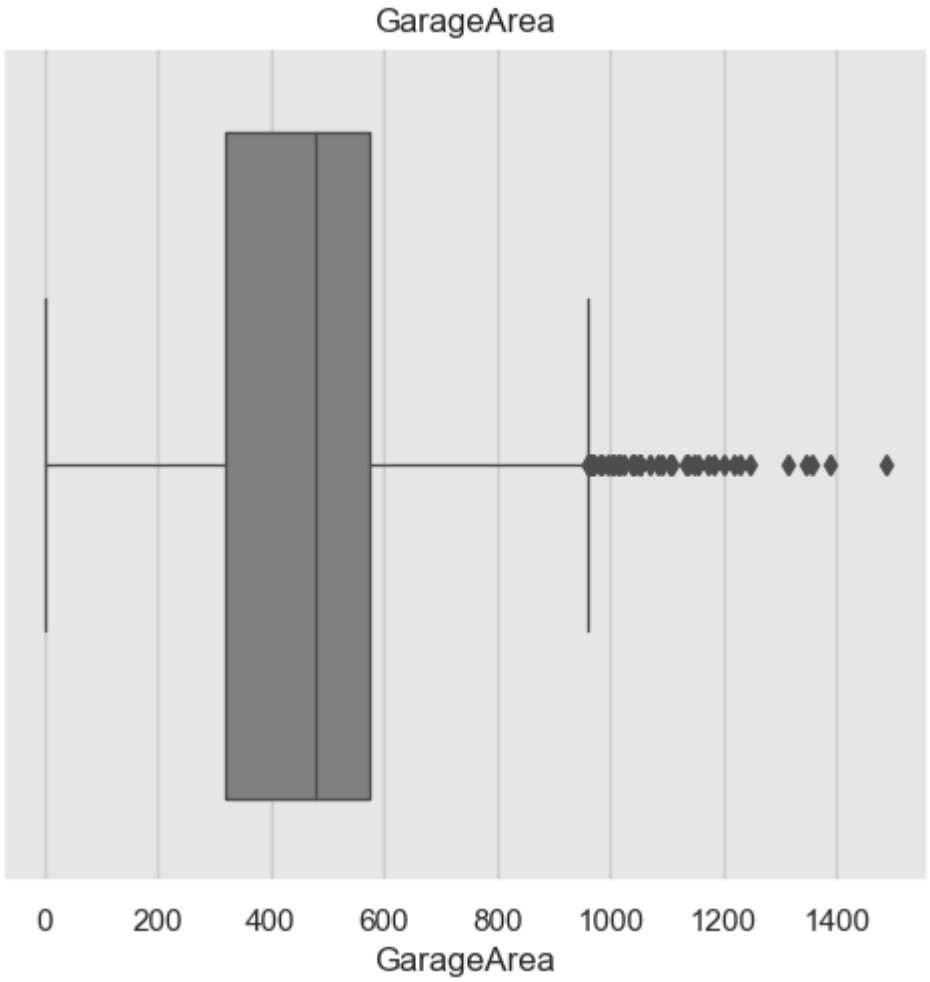

KitchenAbvGr
0.0
0.5
1.0
1.5
2.0
2.5
3.0
KitchenAbvGr

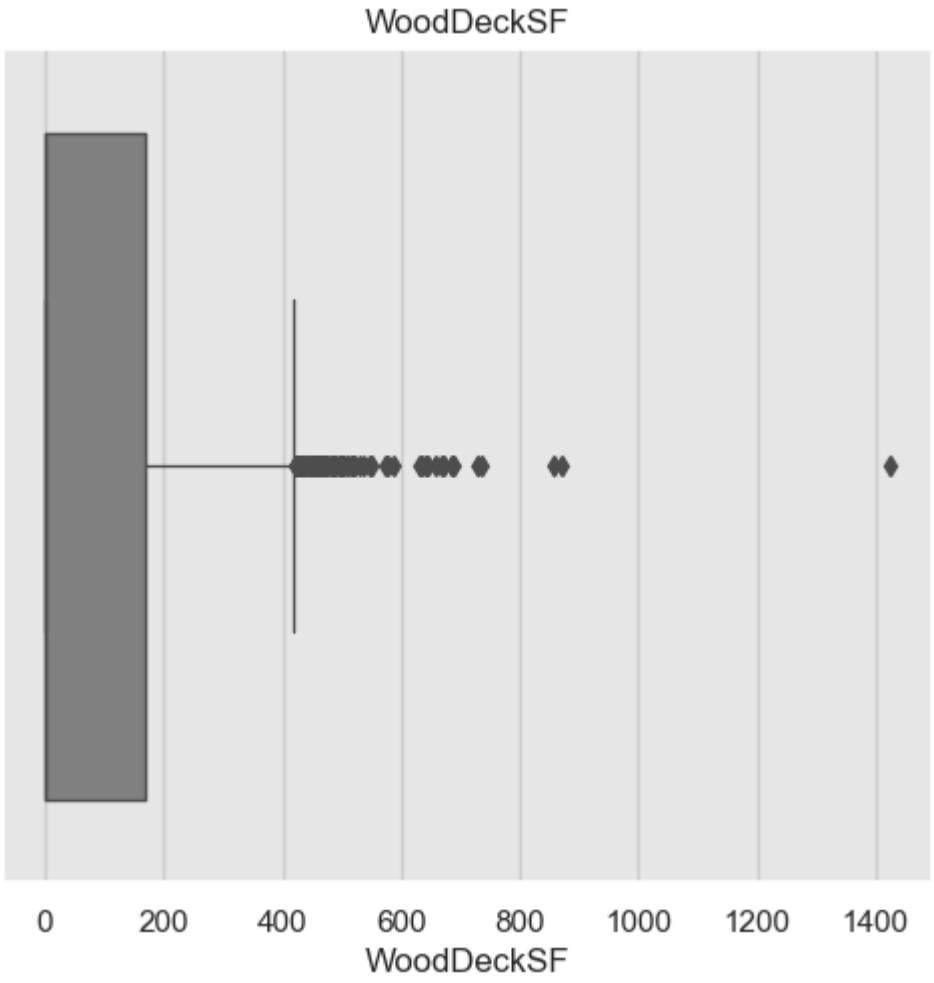

TotRmsAbvGrd
2
4
6
8
10
12
14
TotRmsAbvGrd

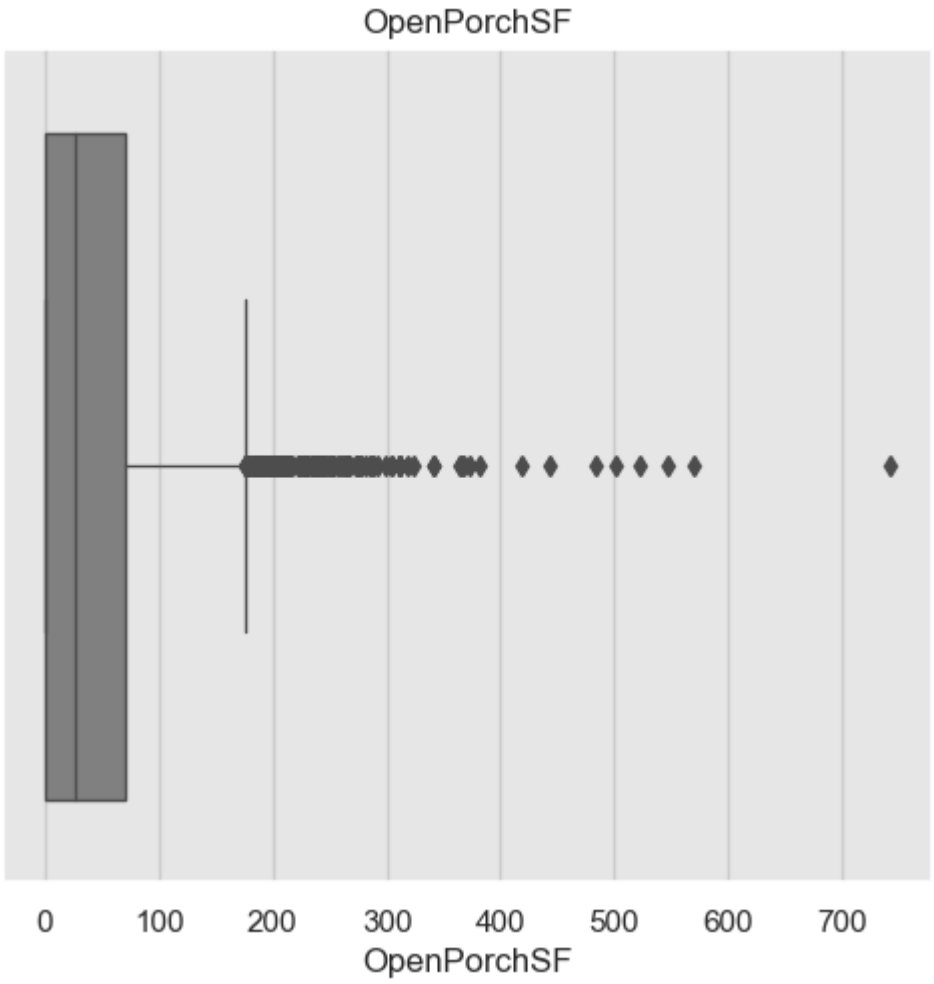

Fireplaces
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Fireplaces

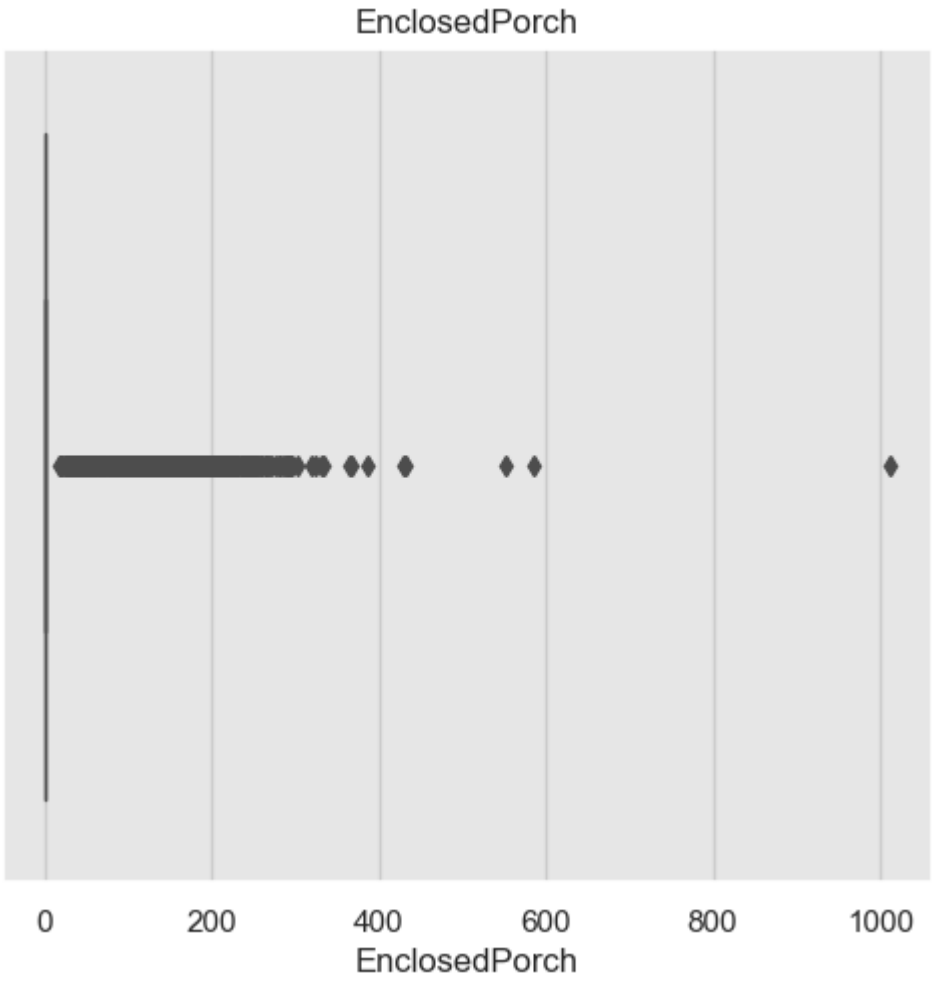

GarageYrBlt
1900
1950
2000
2050
2100
2150
2200
GarageYrBlt

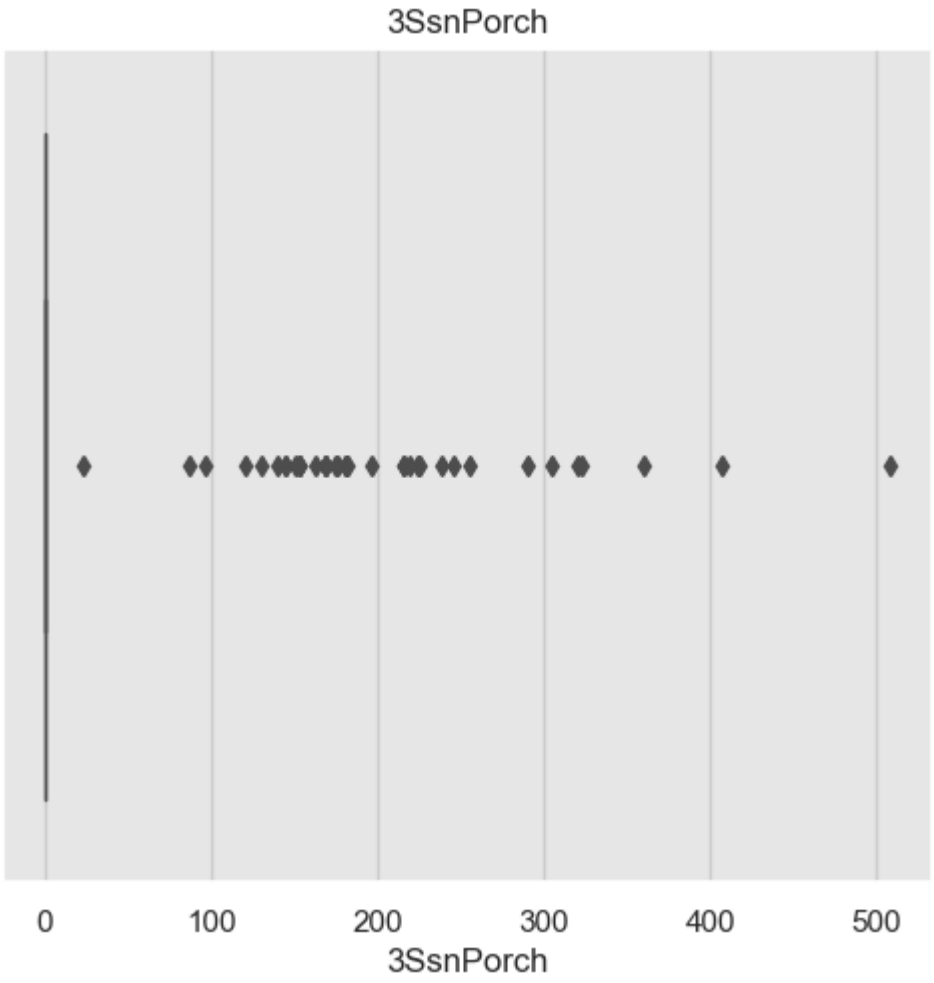

GarageCars
0
1
2
3
4
5
GarageCars

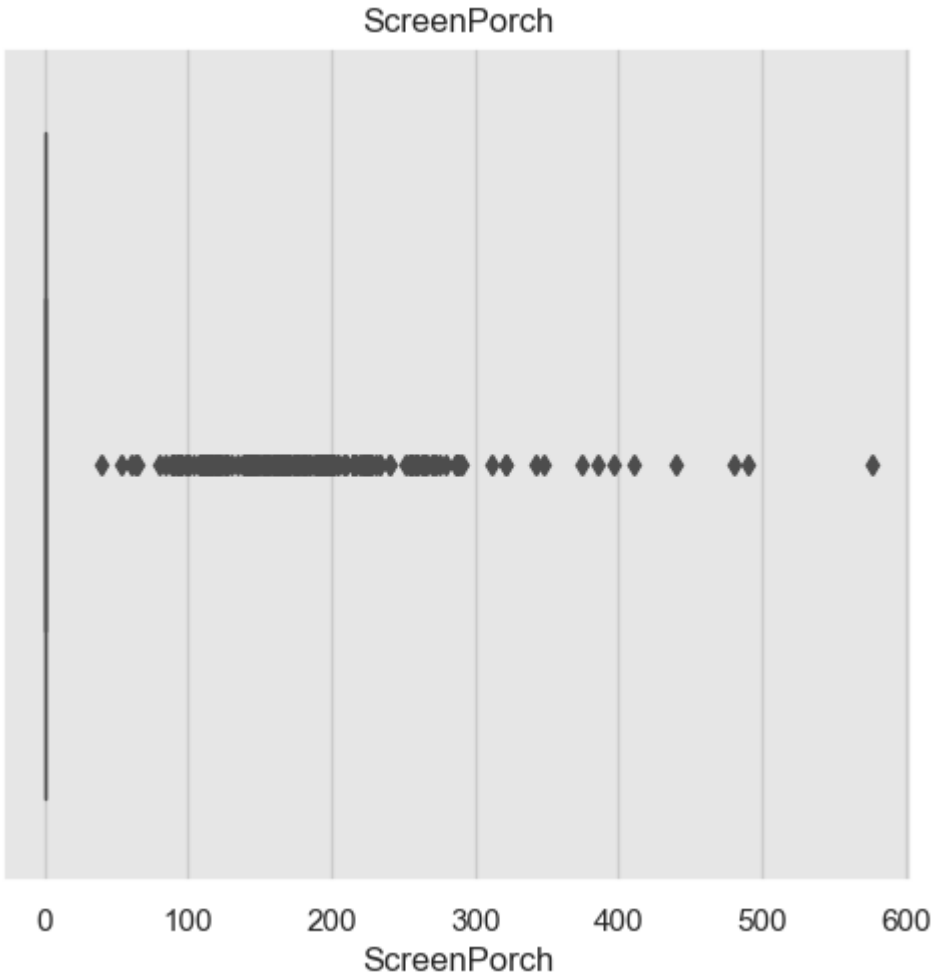

GarageArea
0
200
400
600
800
1000
1200
1400
GarageArea

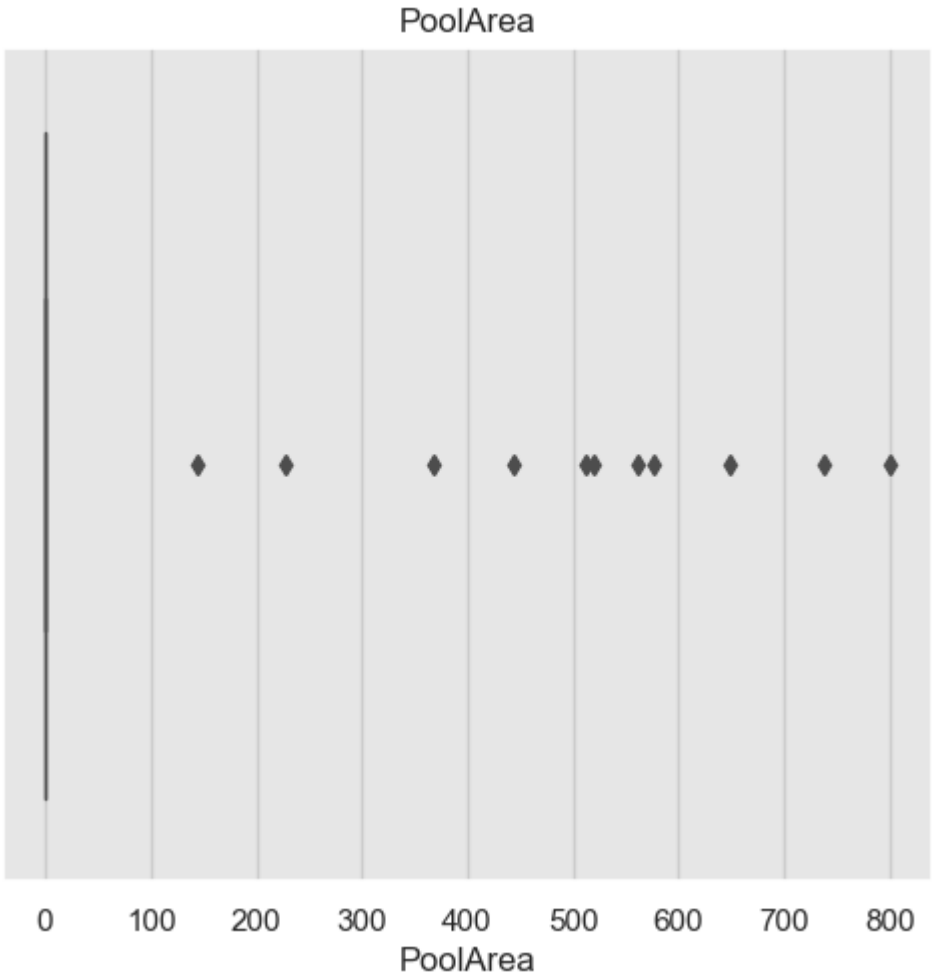

WoodDeckSF
0
200
400
600
800
1000
1200
1400
WoodDeckSF

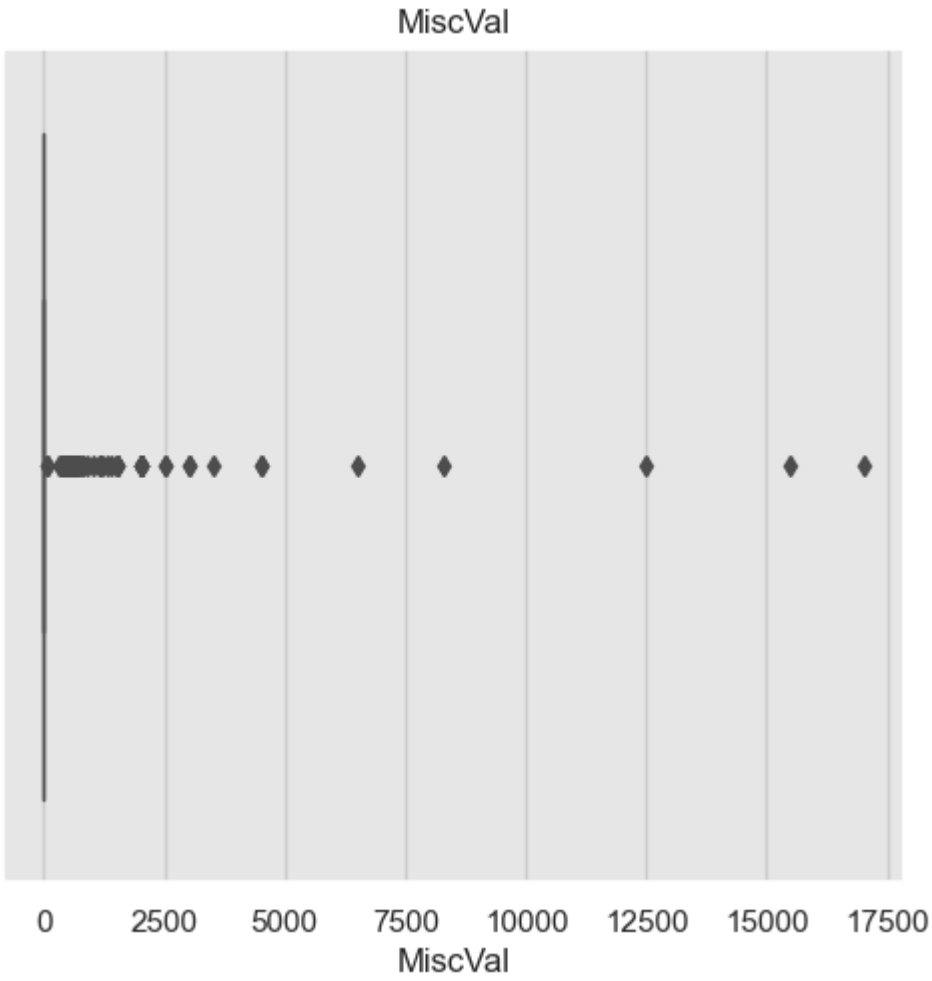

OpenPorchSF
0
100
200
300
400
500
600
700
OpenPorchSF

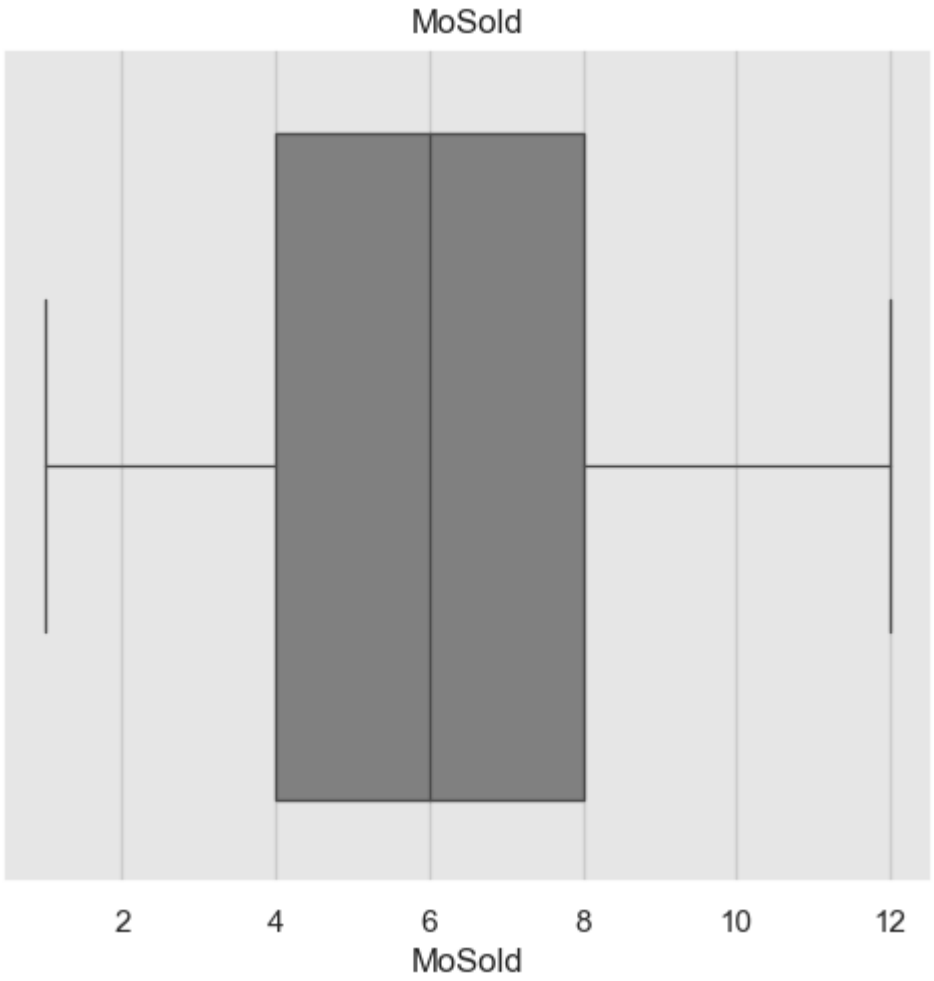

EnclosedPorch
0
200
400
600
800
1000
EnclosedPorch

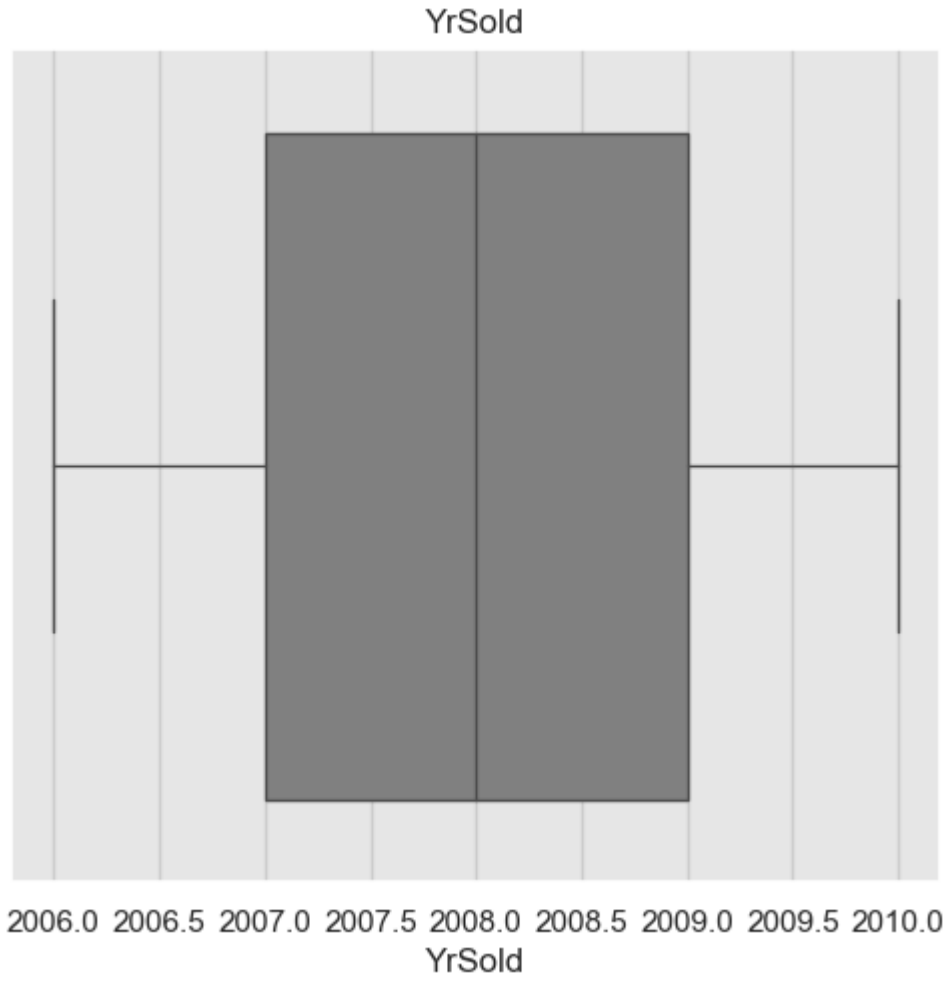

3SsnPorch
0
100
200
300
400
500
3SsnPorch

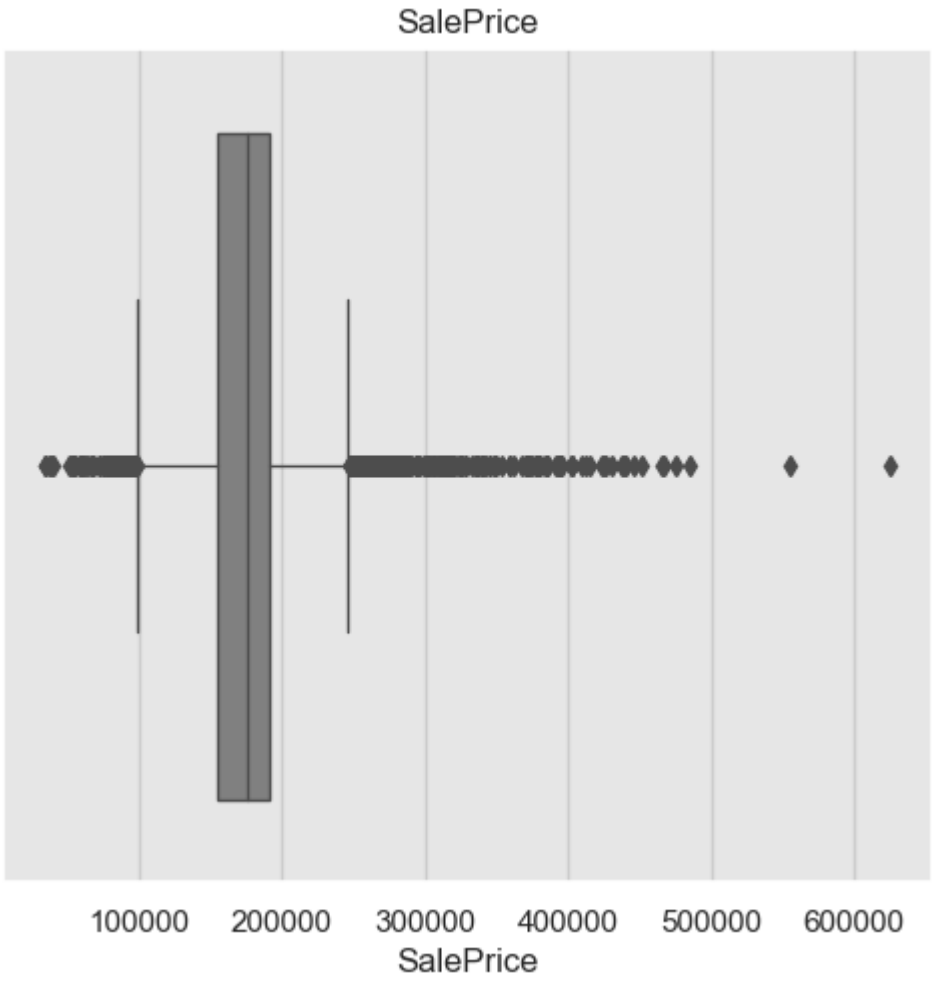

ScreenPorch
0
100
200
300
400
500
600
ScreenPorch

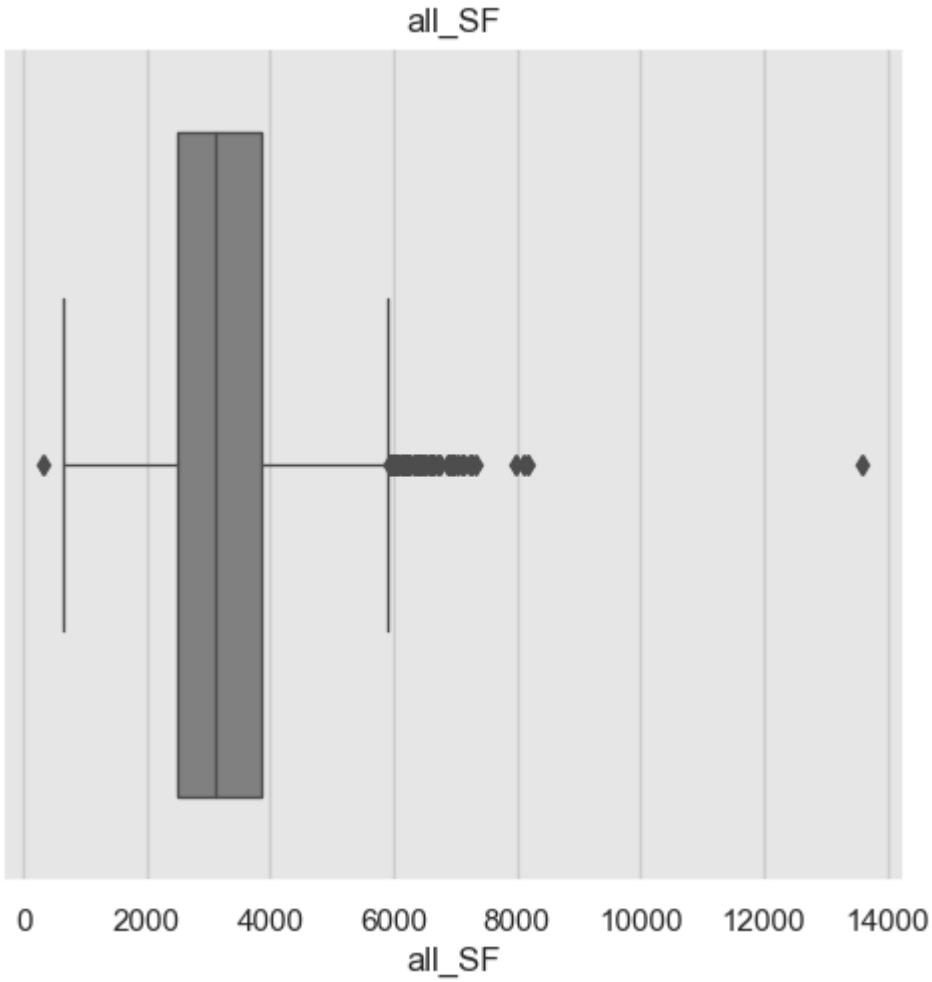

PoolArea
0
100
200
300
400
500
600
700
800
PoolArea

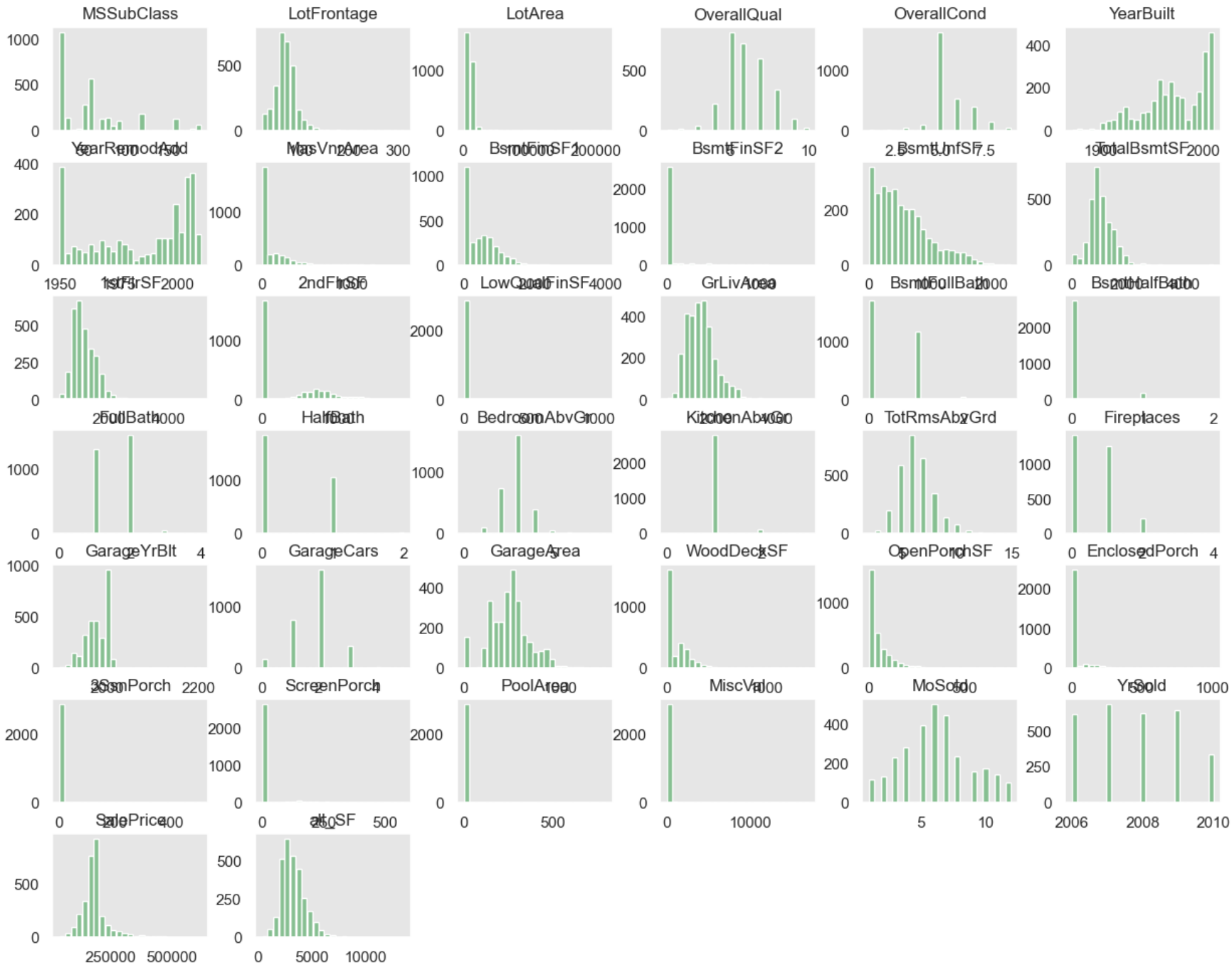

MiscVal
0
2500
5000
7500
10000
12500
15000
17500
MiscVal


MoSold
2
4
6
8
10
12
MoSold


YrSold
2006.0
2006.5
2007.0
2007.5
2008.0
2008.5
2009.0
2009.5
2010.0
YrSold


SalePrice
100000
200000
300000
400000
500000
600000
SalePrice


all_SF
0
2000
4000
6000
8000
10000
12000
14000
all_SF



Label encoding for Catagorical Columns

In [ ]:
```
df8=df7
df8 = template.onehotencoding(df8)
print(df8.columns)
print(df8.shape)
```

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       ...
       'SaleType_New', 'SaleType_None', 'SaleType_Oth', 'SaleType_WD',
       'SaleCondition_Abnorml', 'SaleCondition_AdjLand',
       'SaleCondition_Alloca', 'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=288)
(2906, 288)
```

Feature Selection throgh Random Forest

We have 288 features and seems that many of them will be redundunt, which increase the variance of estimates. Therefor, we consider 8 to 10 most important features to carry out the ML and Baysian inferential analysis and predict the sale price.

In [ ]:
```
df8=df8

LABEL_COL="SalePrice"
split=0.3
random=SEED
threshold=4#.999
tree=500


df9=template.RFfeatureimportance_reg(df8, LABEL_COL, split,random, tree,threshold)
```

```
Random Forest FS Analysis
Selected Features
0       all_SF
1    GrLivArea
2      LotArea
dtype: object
```

In [ ]:
```
df9
```

Out[ ]:

	all_SF	GrLivArea	LotArea	SalePrice
0	3371.0	1710	8450	208500.0000

	all_SF	GrLivArea	LotArea	SalePrice
1	3282.0	1262	9600	181500.0000
2	3518.0	1786	11250	223500.0000
3	3150.0	1717	9550	140000.0000
4	4805.0	2198	14260	250000.0000
...	...	...	...	...
2914	1638.0	1092	1936	167081.2209
2915	1948.0	1092	1894	164788.7782
2916	3498.0	1224	20000	219222.4234
2917	1994.0	970	10441	184924.2797
2918	3978.0	2000	9627	187741.8667

2906 rows × 4 columns

In [ ]:
```
### Call for Min_max tranformation of features
LABEL_COL="SalePrice"


df10=template.MinMax_Transformation(df9, LABEL_COL)
```

In [ ]:
```
#saving final as csv
df10.to_csv('fdata.csv')
```

ML Regressors Algorithm with Default model parameters (different regression pipelines)

We run the function of ML models with default model parameters and run a 5 fold cross validation for each pipeline/model.For this we calculate the mean, std and min score (=error) for every model. We fit the the data (features X and target y) using the default model parameters and apply to X_test to predict the SalePrice.

In [ ]:
```
#function Call

LABEL_COL="SalePrice"
random=SEED
split=0.3
```

```
r1,r2,r3=runML_models_CV(df9,split,random, LABEL_COL)
```

CV Results

	mean_rmse	std	min_rmse
Linear	40342.991465	2600.730106	36240.985519
Ridge	40342.458782	2600.599543	36240.105120
Huber	41397.323095	2995.898890	37372.630405
Lasso	40343.000068	2600.851314	36241.031891
ElaNet	40913.649449	3007.726116	36728.496777
BayesRidge	40341.129750	2600.624476	36235.422320
RandomForestReg	42215.657112	1744.747041	40325.011598
DecisionTreeReg	57128.692495	2865.780873	54246.002612
SVReg	53503.959515	3903.455965	49788.882354
GBR	41561.467887	1971.857825	39277.969339
XGB	45196.882485	1577.854814	43531.570064
LGBM	43245.066028	2160.425492	40600.293307
ADA	48219.774075	2543.452671	45627.716812

Training Data Fit Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
Linear	0.443287	0.442464	28871.210620	1.605833e+09	40072.840271
Ridge	0.443286	0.442464	28869.516359	1.605833e+09	40072.843052
Huber	0.409539	0.408667	27997.328953	1.703176e+09	41269.546914
Lasso	0.443287	0.442464	28871.320728	1.605833e+09	40072.840344
ElaNet	0.421860	0.421006	28196.493907	1.667637e+09	40836.714274
BayesRidge	0.443283	0.442461	28860.328460	1.605842e+09	40072.954477
RandomForestReg	0.913670	0.913543	10969.255673	2.490165e+08	15780.255220
DecisionTreeReg	0.999832	0.999831	36.209486	4.858570e+05	697.034425

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
SVReg	0.003200	0.001726	33535.298134	2.875258e+09	53621.429685
GBR	0.652740	0.652227	24310.132955	1.001666e+09	31649.112333
XGB	0.940825	0.940737	9921.191506	1.706908e+08	13064.867473
LGBM	0.708237	0.707806	21153.747889	8.415868e+08	29010.116112
ADA	0.999033	0.999031	381.992279	2.790518e+06	1670.484404

Test Data Prediction Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
Linear	0.417875	0.415863	28704.086319	1.569249e+09	39613.752988
Ridge	0.417857	0.415845	28702.752486	1.569298e+09	39614.369665
Huber	0.378844	0.376698	27915.626551	1.674466e+09	40920.242496
Lasso	0.417879	0.415867	28704.132073	1.569240e+09	39613.636677
ElaNet	0.394335	0.392242	27991.578014	1.632707e+09	40406.775277
BayesRidge	0.417757	0.415745	28695.581012	1.569567e+09	39617.764380
RandomForestReg	0.339521	0.337238	29375.860175	1.780472e+09	42195.637125
DecisionTreeReg	-0.219635	-0.223850	37429.399920	3.287804e+09	57339.376485
SVReg	-0.003770	-0.007240	32878.181357	2.705891e+09	52018.181804
GBR	0.348681	0.346430	29022.036457	1.755780e+09	41902.024259
XGB	0.230976	0.228318	31094.562987	2.073079e+09	45531.072428
LGBM	0.368191	0.366007	29257.587051	1.703186e+09	41269.674282
ADA	0.179494	0.176658	30856.359422	2.211862e+09	47030.433342

GridSearch CV For ML Models

Here I apply the GridSearch and Hyper Parameters tuning with Preprocessing function. Using the gridsearch approach to select the optimal hyper parameters with the ML models. For this I calculate the mean, std and min score (=error) for every model. By this I get a first the

optimal (best) estimate for the different regression pipelines on the training data. We fit the the data (features X_train and target y_train) using the optimal model parameters. The same optimal parameters are used to compute the prediction of SalePrice using the X_test data.

In [ ]:
```
#function Call Grid Search
### call for Loop over GridSearchCV Pipelines
LABEL_COL="SalePrice"

random=SEED
split=0.3


r4,r5,r6=run_MLAlg_gridCV(df9,split,random, LABEL_COL)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=0 will be ignored. Current value: bagging_freq=1
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=0 will be ignored. Current value: bagging_freq=1
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=0 will be ignored. Current value: bagging_freq=1
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=0 will be ignored. Current value: bagging_freq=1
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] bagging_freq is set=1, subsample_freq=0 will be ignored. Current value: bagging_freq=1
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Grid Search CV Results
```

	mean_rmse	std	min_rmse
gscv_Linear	40342.991465	2600.730106	36240.985519
gscv_Ridge	40335.813107	2597.717912	36231.667407
gscv_Huber	41273.615742	2973.386283	37237.494559
gscv_Lasso	40342.991139	2600.732644	36240.986464
gscv_ElaNet	40335.115820	2627.967542	36198.303974
gscv_RandomForestReg	40096.313603	2612.393796	36726.342044

	mean_rmse	std	min_rmse
Decision Tree Reg	41609.868739	2994.648229	38130.930443
gscv_SVReg	47877.919249	3826.630055	44217.850677
gscv_GBR	43402.931624	2052.044202	41559.175387
gscv_XGB	42579.798463	1800.775184	41180.353938
gscv_LGBM	41463.599987	2318.007741	39108.946184
gscv_ADA	43888.641207	2191.810693	41465.019140

Training Data Fit Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
gscv_Linear	0.443287	0.442464	28871.210620	1.605833e+09	40072.840271
gscv_Ridge	0.443273	0.442450	28848.605325	1.605872e+09	40073.337257
gscv_Huber	0.413003	0.412135	28039.913455	1.693186e+09	41148.343651
gscv_Lasso	0.443287	0.442464	28871.215192	1.605833e+09	40072.840271
gscv_ElaNet	0.441376	0.440551	28620.553258	1.611343e+09	40141.536490
gscv_RandomForestReg	0.497877	0.497135	27289.100949	1.448366e+09	38057.399738
Decision Tree Reg	0.459661	0.458862	28161.063796	1.558601e+09	39479.126388
gscv_SVReg	0.218260	0.217105	29217.169991	2.254917e+09	47485.970554
gscv_GBR	0.722835	0.722425	21642.442257	7.994801e+08	28275.079696
gscv_XGB	0.665100	0.664605	23736.327842	9.660157e+08	31080.793434
gscv_LGBM	0.582796	0.582179	25595.472751	1.203420e+09	34690.339644
gscv_ADA	0.718431	0.718015	24588.339108	8.121813e+08	28498.794145

Test Data Prediction Results

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
gscv_Linear	0.417875	0.415863	28704.086319	1.569249e+09	39613.752988
gscv_Ridge	0.417670	0.415657	28684.956987	1.569803e+09	39620.739475
gscv_Huber	0.382317	0.380182	27955.250271	1.665104e+09	40805.691132

	R-Squared	Adj-R Squared	MAE	MSE	RMSE
gscv_Lasso	0.417875	0.415864	28704.088788	1.569249e+09	39613.749220
gscv_ElaNet	0.413883	0.411858	28476.540169	1.580011e+09	39749.353766
gscv_RandomForestReg	0.408623	0.406579	28366.439407	1.594190e+09	39927.311331
Decision Tree Reg	0.354489	0.352258	29581.988031	1.740122e+09	41714.763863
gscv_SVReg	0.197997	0.195225	28936.736058	2.161981e+09	46497.110853
gscv_GBR	0.311101	0.308720	29313.672819	1.857085e+09	43093.911327
gscv_XGB	0.321214	0.318868	29363.110345	1.829823e+09	42776.424677
gscv_LGBM	0.393406	0.391309	28760.322136	1.635213e+09	40437.760988
gscv_ADA	0.311007	0.308626	31045.726112	1.857337e+09	43096.831424

Analysis with Baysian Regression Functions

I use seven different MCMC method in PYMC3's `sample` function to fit the models to X_taining data. Seven main functions are applied. The Baysian algorithms are applied to training data and the fitting features such as R-Suared, MAE and RMSE are reported for the training data. In the next predictions are computed using the optimal weights estimated during the training session and X_test data. The Baysian Algorithms employed to inference are:

1. NUTS Algorithm
2. Hamiltonian MC
3. Metropolis MC
4. Slice MC
5. DEMetropolis MC
6. MetropolisZ MC
7. Sequential MC (SMC)

The data is split into training and test (using training-test split to compute the predectionof these seven approaches.

Note: The trace, Posterior and Forests plot are not shown here. These are given in Appendix-II

In [ ]:
```
df10.columns
```

Out[ ]:
```
Index(['all_SF', 'GrLivArea', 'LotArea', 'SalePrice'], dtype='object')
```

In [ ]:
```
formula="SalePrice ~ all_SF + GrLivArea + LotArea"
family=pm.glm.families.Normal()
prior ={"Intercept": pm.Normal.dist(mu=0, sd=2.5),
        "x1": pm.Normal.dist(mu=0, sd=10),
        "x2": pm.Normal.dist(mu=0, sd=5),
        "x3": pm.Normal.dist(mu=0, sd=4.1),
        #"x4": pm.Normal.dist(mu=0, sd=1),
        #"x5": pm.Normal.dist(mu=0, sd=10),
        #"x6": pm.Normal.dist(mu=0, sd=1.9),
        #"x7": pm.Normal.dist(mu=0, sd=2.45),
        }
prior_samples=1000
draws=5000
chains=5
tune=400
target_accept=0.87
SEED=42

b_r1, b_r2=run_baysian_algorithms(df10,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=0)
```

```
============ NUTS_Reg ===========
Multiprocess sampling (5 chains in 4 jobs)
NUTS: [sd, LotArea, GrLivArea, all_SF, Intercept]
                                                100.00% [27000/27000 1:05:19<00:00 Sampling 5 chains, 12,521 divergences]

Sampling 5 chains for 400 tune and 5_000 draw iterations (2_000 + 25_000 draws total) took 3939 seconds.
There were 2511 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 1.0, but should be close to 0.8. Try to increase the number o
f tuning steps.
There were 2509 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 1.0, but should be close to 0.8. Try to increase the number o
f tuning steps.
There were 2484 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 1.0, but should be close to 0.8. Try to increase the number o
f tuning steps.
There were 2475 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 1.0, but should be close to 0.8. Try to increase the number o
f tuning steps.
There were 2542 divergences after tuning. Increase `target_accept` or reparameterize.
The acceptance probability does not match the target. It is 1.0, but should be close to 0.8. Try to increase the number o
f tuning steps.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
```

```
==============================

===============Training Data Fit=======================
                                                100.00% [1000/1000 00:01<00:00]

===============Test Data Fit=======================
                                                100.00% [1000/1000 00:02<00:00]

============ HamiltonianMC_Reg ===========
Multiprocess sampling (5 chains in 4 jobs)
HamiltonianMC: [sd, LotArea, GrLivArea, all_SF, Intercept]
                                                0.46% [124/27000 1:03:55<230:55:16 Sampling 5 chains, 0 divergences]
```

In [ ]:

Conclusion

In case of default parameter setup for best parameter selection the decision tree works well on the training data with highest R squared and lowest RMSE (1 and 2.8E -16 respectively). Random forest model performed equally well in case of model however its RMSE is higher than that of Decision tree. In case of default parameter setup on test data ***LGBM*** model is found to be the best among all models with lowest RMSE and highest R squared (0.4 and 0.8 respectively).

In the case of Grid search cross validation Gradient boost regression seem to perform best in terms of lowest RMSE among all model parameters. The Grid search for best model parameters on training data find GBR to be the best performing model with the lowest root mean squared error and the best fit model among all others. Grid search on test data, however, finds ***GBR, XGB, and LGBM*** models to be performing equally well with almost same RMSE and R squared values.

Bayesian method of model selection finds no difference among performance of NUTS, Hamiltonian MC , Metropolis and Slice models when using training data. The SMC, however, is the poorest performing model in this case. These results also stand true for the test data.

As compared to the overall performance the ***ML models outperform the Bayesian approach*** in accuracy as well as computational ease.

More importantly, the results of **Baysian are very close to the ML based to Linear, Ridge etc but wost than the tree ensamble algorithams**.

The reason could be computational difficulty in updating current prediction on the basis of previous history which requires too many re sampling and thus made this approach computationally complex and slow. The grid search method is therefore better choice for model selection especially in case of large samples.

Appendix-I

Trace and Posteriror Plots for Baysian Models (Dataset-1)*

In [ ]:
```
#algo_list=get_supported_algorithms_reg()
formula="Price ~ MedInc + HouseAge + AveRooms + AveBedrms + Population + AveOccup + Latitude + Longitude"
family=pm.glm.families.Normal()
prior = {"Intercept": pm.Normal.dist(mu=0, sd=10),
        "x1": pm.Normal.dist(mu=0, sd=10),
        "x2": pm.Normal.dist(mu=0, sd=10),
        "x3": pm.Normal.dist(mu=0, sd=10),
        "x4": pm.Normal.dist(mu=0, sd=10),
        "x5": pm.Normal.dist(mu=0, sd=10),
        "x6": pm.Normal.dist(mu=0, sd=10),
        "x7": pm.Normal.dist(mu=0, sd=10),
        "x8": pm.Normal.dist(mu=0, sd=10),
        }
prior_samples=1000
draws=5000
chains=5
tune=1000
target_accept=0.87
SEED=42
#run_baysian_algorithms_plots
results3, results4=run_baysian_algorithms_plots(df4,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=
```

Appendix-II

Trace and Posteriror Plots for Baysian Models (Dataset-2)*

In [ ]:
```
formula="SalePrice ~ all_SF + GrLivArea + LotArea"
family=pm.glm.families.Normal()
prior ={"Intercept": pm.Normal.dist(mu=0, sd=2.5),
        "x1": pm.Normal.dist(mu=0, sd=10),
        "x2": pm.Normal.dist(mu=0, sd=5),
        "x3": pm.Normal.dist(mu=0, sd=4.1),
        }
prior_samples=1000
draws=5000
chains=5
tune=400
target_accept=0.87
```

```
SEED=42

b_r1, b_r2=run_baysian_algorithms_plots(df10,formula, family,prior,prior_samples,SEED,draws, chains, tune,verbose=0)
```