```
#### Airbnb Cleaned Europe Dataset
```

About Dataset

This is a merged dataset of 9 famous cities in Europe.

Amsterdam, Athens, Barcelona, Berlin, Budapest, Lisbon, Paris, Rome and Vienna.

The original Dataset was really messy and lacked describing appropriate information.

Perform analysis and tell a story you'd like to tell with this dataset.

Column names are self-explanatory.

Have fun exploring.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
###! pip install kaggle
```

```
###!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
##!chmod 600 ~/.kaggle/kaggle.json
```

```
###!pip install keras-tuner
```

```
###!kaggle datasets download -d dipeshkhemani/airbnb-cleaned-europe-dataset
```

```
##! unzip /content/airbnb-cleaned-europe-dataset.zip
```

```
###!pip --inspect
```

```
# Importing all datasets
airbnb = pd.read_csv('/content/Aemf1.csv')
airbnb.head(4)
```

	City	Price	Day	Room Type	Shared Room	Private Room	Person Capacity	Superhost	Multiple Rooms	Business
0	Amsterdam	194.033698	Weekday	Private room	False	True	2.0	False	1	0
1	Amsterdam	344.245776	Weekday	Private room	False	True	4.0	False	0	0
2	Amsterdam	264.101422	Weekday	Private room	False	True	2.0	False	0	1
3	Amsterdam	433.529398	Weekday	Private room	False	True	4.0	False	0	1

```
airbnb.columns
```

```
Index(['City', 'Price', 'Day', 'Room Type', 'Shared Room', 'Private Room',
       'Person Capacity', 'Superhost', 'Multiple Rooms', 'Business',
       'Cleanliness Rating', 'Guest Satisfaction', 'Bedrooms',
       'City Center (km)', 'Metro Distance (km)', 'Attraction Index',
       'Normalised Attraction Index', 'Restraunt Index',
       'Normalised Restraunt Index'],
      dtype='object')
```

```
airbnb.isnull().sum()
```

```
City                          0
Price                         0
Day                           0
Room Type                     0
Shared Room                   0
Private Room                  0
Person Capacity               0
Superhost                     0
Multiple Rooms                0
Business                      0
Cleanliness Rating            0
Guest Satisfaction            0
Bedrooms                      0
City Center (km)              0
Metro Distance (km)           0
Attraction Index              0
Normalised Attraction Index   0
Restraunt Index               0
Normalised Restraunt Index    0
dtype: int64
```

```
###! pip install klib
```

```
import klib
```

```
airbnb = klib.data_cleaning(airbnb)
```

```
Long column names detected (>25 characters). Consider renaming the following columns ['normalised_at
Shape of cleaned data: (41714, 19) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 2.98 MB (-57.2%)
```

```
airbnb = klib.convert_datatypes(airbnb)
```

```
airbnb.info()
```

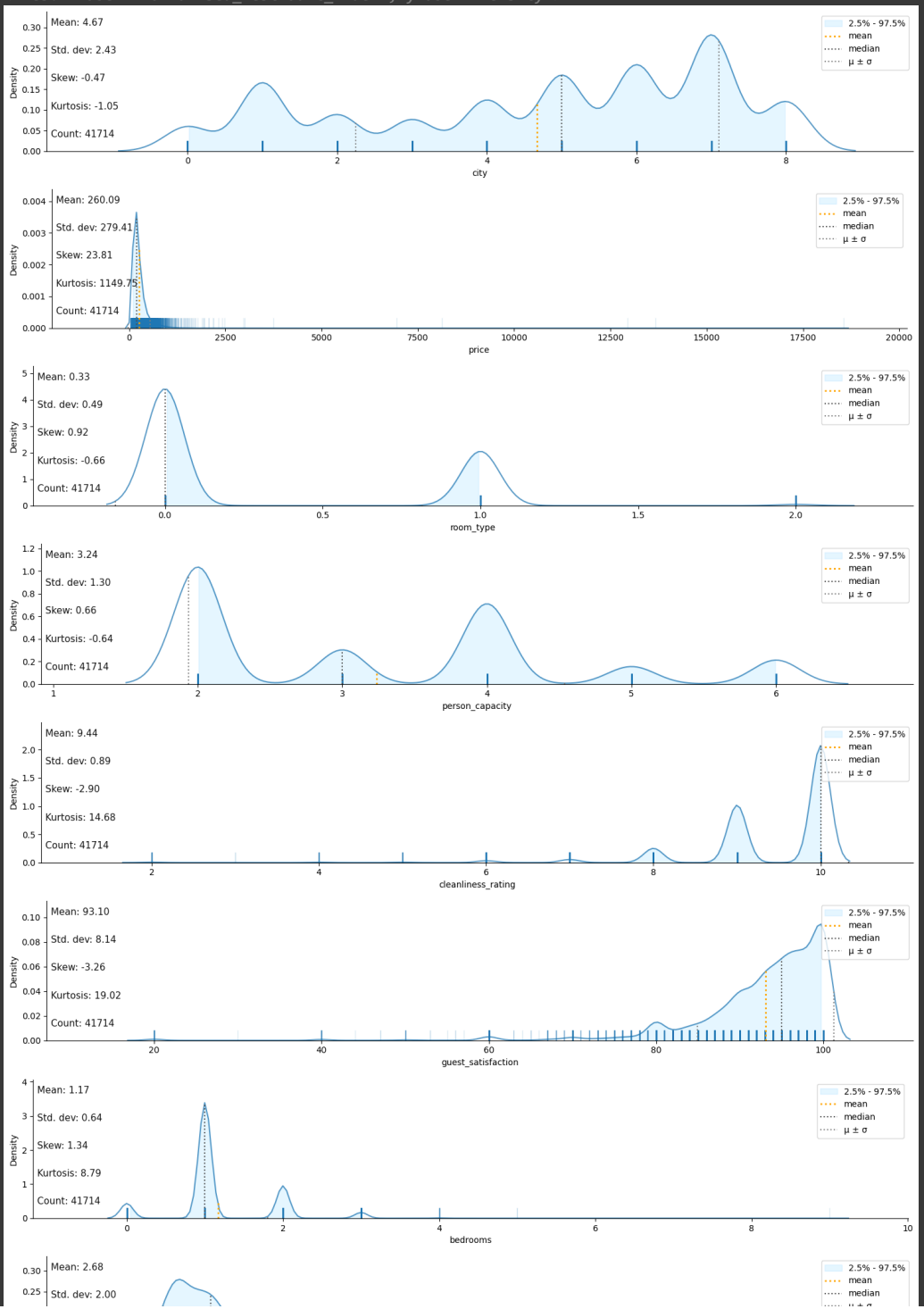
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41714 entries, 0 to 41713
Data columns (total 19 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   city                         41714 non-null  category
 1   price                        41714 non-null  float64
 2   day                          41714 non-null  category
 3   room_type                    41714 non-null  category
 4   shared_room                  41714 non-null  boolean
 5   private_room                 41714 non-null  boolean
 6   person_capacity              41714 non-null  float32
 7   superhost                    41714 non-null  boolean
 8   multiple_rooms               41714 non-null  int8
 9   business                     41714 non-null  int8
 10  cleanliness_rating           41714 non-null  float32
 11  guest_satisfaction           41714 non-null  float32
 12  bedrooms                     41714 non-null  int8
 13  city_center_km               41714 non-null  float32
 14  metro_distance_km            41714 non-null  float32
 15  attraction_index             41714 non-null  float32
 16  normalised_attraction_index  41714 non-null  float32
 17  restraunt_index              41714 non-null  float32
 18  normalised_restraunt_index   41714 non-null  float32
dtypes: boolean(3), category(3), float32(9), float64(1), int8(3)
memory usage: 2.2 MB
```

```
airbnb.city = airbnb.city.astype('category').cat.codes
```

```
airbnb.day = airbnb.day.astype('category').cat.codes
airbnb.room_type = airbnb.room_type.astype('category').cat.codes
###patients.room_type = patients.room_type.astype('category').cat.codes
```

```
klib.dist_plot(airbnb)
```

```
Large dataset detected, using 10000 random samples for the plots. Summary statistics are still based
<Axes: xlabel='normalised_restraunt_index', ylabel='Density'>
```



```
import seaborn as sns
```

```
###! pip install evalml
```

```
##!pip install featuretools==0.6.1
```

```
###!pip install tornado==4.5.3
```

```
airbnb.shape
```

```
(41714, 19)
```

```
airbnb.columns
```

```
Index(['city', 'price', 'day', 'room_type', 'shared_room', 'private_room',
       'person_capacity', 'superhost', 'multiple_rooms', 'business',
       'cleanliness_rating', 'guest_satisfaction', 'bedrooms',
       'city_center_km', 'metro_distance_km', 'attraction_index',
       'normalised_attraction_index', 'restraunt_index',
       'normalised_restraunt_index'],
      dtype='object')
```

```
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41714 entries, 0 to 41713
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   city                          41714 non-null  int8
 1   price                         41714 non-null  float64
 2   day                           41714 non-null  int8
 3   room_type                     41714 non-null  int8
 4   shared_room                   41714 non-null  boolean
 5   private_room                  41714 non-null  boolean
 6   person_capacity               41714 non-null  float32
 7   superhost                     41714 non-null  boolean
 8   multiple_rooms                41714 non-null  int8
 9   business                      41714 non-null  int8
 10  cleanliness_rating            41714 non-null  float32
 11  guest_satisfaction            41714 non-null  float32
 12  bedrooms                      41714 non-null  int8
 13  city_center_km                41714 non-null  float32
 14  metro_distance_km             41714 non-null  float32
 15  attraction_index              41714 non-null  float32
 16  normalised_attraction_index   41714 non-null  float32
 17  restraunt_index               41714 non-null  float32
 18  normalised_restraunt_index    41714 non-null  float32
dtypes: boolean(3), float32(9), float64(1), int8(6)
memory usage: 2.2 MB
```

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

```
X = airbnb.drop("price", axis=1)
y = airbnb["price"]
```

```
  # Splitting the data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(29199, 18) (29199,)
(12515, 18) (12515,)
```

```
X_train.columns
```

```
Index(['city', 'day', 'room_type', 'shared_room', 'private_room',
       'person_capacity', 'superhost', 'multiple_rooms', 'business',
       'cleanliness_rating', 'guest_satisfaction', 'bedrooms',
       'city_center_km', 'metro_distance_km', 'attraction_index',
       'normalised_attraction_index', 'restraunt_index',
       'normalised_restraunt_index'],
      dtype='object')
```

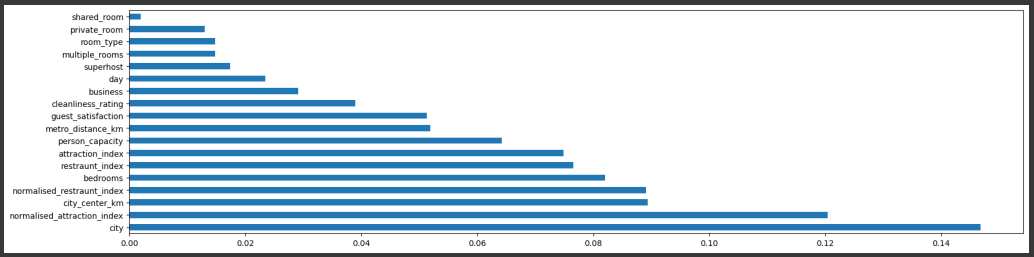
```
from sklearn.ensemble import ExtraTreesRegressor
import matplotlib.pyplot as plt
model=ExtraTreesRegressor()
model.fit(X_train,y_train)
```

```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

```
print(model.feature_importances_)
```

```
[0.14679735 0.02349276 0.01477341 0.00194364 0.01300953 0.06420916
 0.01743935 0.01479748 0.02912046 0.03897119 0.0512538  0.08201733
 0.08943083 0.05187905 0.07484113 0.12038103 0.07655399 0.08908851]
```

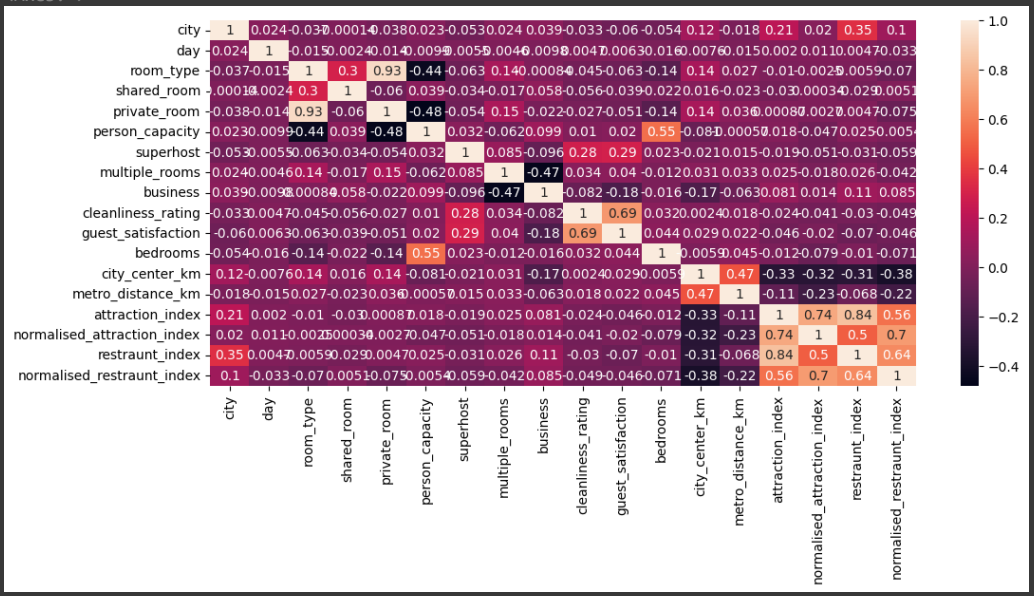
```
plt.figure(figsize=(20,5))
ranked_features=pd.Series(model.feature_importances_,index=X.columns)
ranked_features.nlargest(20).plot(kind='barh')
plt.show()
```




```
import seaborn as sns
corr=X_train.corr()
top_features=corr.index
```

```
plt.figure(figsize=(12,5))
sns.heatmap(X_train[top_features].corr(),annot=True)
```

```
<Axes: >
```




```
threshold=0.8
```

```
# find and remove correlated features
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

```
correlation(X_train,threshold)
```

```
{'private_room', 'restraunt_index'}
```

```
X_train = X_train.drop(['private_room'], axis=1)
X_train = X_train.drop(['restraunt_index'], axis=1)
```

```
X_test = X_test.drop(['private_room'], axis=1)
X_test = X_test.drop(['restraunt_index'], axis=1)
```

```
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(29199, 16) (12515, 16)
(29199,) (12515,)
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

```
from sklearn.preprocessing import RobustScaler
```

```
from sklearn.preprocessing import RobustScaler
scaler=RobustScaler()
X_train=pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
X_train.head(3)
```

	city	day	room_type	shared_room	person_capacity	superhost	multiple_rooms	business	cleanline
0	-1.0	0.0	0.0	0.0	-0.5	0.0	0.0	0.0	
1	-1.0	1.0	0.0	0.0	0.5	1.0	1.0	0.0	
2	0.0	0.0	0.0	0.0	1.5	0.0	1.0	0.0	

```
X_test=pd.DataFrame(scaler.fit_transform(X_test),columns=X_test.columns)
X_test.head(3)
```

	city	day	room_type	shared_room	person_capacity	superhost	multiple_rooms	business	cleanline
0	-0.2	-1.0	0.0	0.0	0.5	1.0	0.0	0.0	
1	0.4	0.0	1.0	0.0	-0.5	0.0	0.0	1.0	
2	-0.8	0.0	0.0	0.0	1.5	0.0	0.0	0.0	

```
pipeline_lr=Pipeline([('scalar1',RobustScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('lr_regressor',LinearRegression())])
```

```
pipeline_dt=Pipeline([('scalar2',RobustScaler()),
                      ('pca2',PCA(n_components=2)),
                      ('dt_regressorr',DecisionTreeRegressor())])
```

```
pipeline_randomforest=Pipeline([('scalar3',RobustScaler()),
                      ('pca3',PCA(n_components=2)),
                      ('rf_regressor',RandomForestRegressor())])
```

```
pipeline_gradient_boost=Pipeline([('scalar4',RobustScaler()),
                      ('pca4',PCA(n_components=2)),
                      ('gb_regressor',GradientBoostingRegressor())])
```

```
pipeline_XGboost=Pipeline([('scalar5',RobustScaler()),
                           ('pca5',PCA(n_components=2)),
                           ('xgb_regressor',XGBRegressor())])
```

```
## LEts make the list of pipelines
pipelines = [pipeline_dt, pipeline_randomforest,pipeline_gradient_boost,pipeline_XGboost]
```

```
best_accuracy=0.0
best_regressor=0
best_pipeline=""
```

```
# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {1: 'Decision Tree', 2: 'RandomForest', 3: 'Gradient Boost', 4:'XGBoost Regressor'}

# Fit the pipelines
for pipe in pipelines:
    pipe.fit(X_train, y_train)
```

```
for i,model in enumerate(pipelines):
    if model.score(X_test,y_test)>best_accuracy:
        best_accuracy=model.score(X_test,y_test)
        best_pipeline=model
        best_regressor=i
print('Regressor with best accuracy:{}'.format(pipe_dict[best_regressor]))
```

```
Regressor with best accuracy:RandomForest
```

```
random_model = RandomForestRegressor(n_estimators=300, random_state = 42, n_jobs = -1)
```

```
###! pip install optuna
```

```
import optuna
import pandas as pd
from sklearn import linear_model
from sklearn import ensemble
from sklearn import datasets
from sklearn import model_selection
```

```
import sklearn.datasets
import sklearn.ensemble
import sklearn.model_selection
```

```
def objective(trial):

    # Invoke suggest methods of a Trial object to generate hyperparameters.
    regressor_name = trial.suggest_categorical('regressor', ['SVR', 'RandomForest'])
    if regressor_name == 'SVR':
        svr_c = trial.suggest_float('svr_c', 1e-10, 1e10, log=True)
        regressor_obj = sklearn.svm.SVR(C=svr_c)
    else:
        rf_max_depth = trial.suggest_int('rf_max_depth', 2, 32)
        regressor_obj = sklearn.ensemble.RandomForestRegressor(max_depth=rf_max_depth)

    X, y = sklearn.datasets.fetch_california_housing(return_X_y=True)
    X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y, random_state=0)

    regressor_obj.fit(X_train, y_train)
    y_pred = regressor_obj.predict(X_test)

    error = sklearn.metrics.mean_squared_error(y_test, y_pred)

    return error  # An objective value linked with the Trial object.
```

```
study = optuna.create_study()
```

```
[I 2023-05-17 07:09:12,388] A new study created in memory with name: no-name-eb1ee617-49e4-4745-a89e
```

```
study.optimize(objective, n_trials=4)
```

```
[I 2023-05-17 07:09:29,974] Trial 0 finished with value: 1.38341169175308 and parameters: {'regresso
[I 2023-05-17 07:09:35,505] Trial 1 finished with value: 0.3287383005408244 and parameters: {'regres
[I 2023-05-17 07:09:49,797] Trial 2 finished with value: 1.3603451403342481 and parameters: {'regres
[I 2023-05-17 07:10:02,591] Trial 3 finished with value: 0.2709019267235761 and parameters: {'regres
```

```
study.best_params
```

```
{'regressor': 'RandomForest', 'rf_max_depth': 28}
```

```
random_forest_rgr = RandomForestRegressor(n_estimators=523, max_depth=31)
```

```
random_forest_rgr.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=31, n_estimators=523)
```

```
y_pred = random_forest_rgr.predict(X_test)
```

```
#from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
```

```
print(mean_absolute_error(y_test,y_pred))
#print(accuracy_score(y_test,y_pred))
```

```
135.35665429498428
```

```
y_pred = pd.DataFrame(y_pred)
```

```
y_pred.rename(columns = {0:"Predict"}, inplace=True)
```

```
y_pred.value_counts()
```

```
Predict
167.483344     3
193.097407     3
342.603106     3
271.784709     2
192.710006     2
              ..
194.620319     1
194.655295     1
194.661358     1
194.674391     1
2342.837771    1
Length: 12389, dtype: int64
```