Project :

Predicting Hotel Rating

Name :

Jamila Zaab Al shahrani

Dataset:

https://www.kaggle.com/datafiniti/hotel-reviews

Information about dataset:

- This is a list of 1,000 hotels and their reviews provided by Datafiniti's Business Database. The dataset includes hotel location, name, rating, review data, title, username, and more.
- 19 columns.

Importing needed libraries:

In [1]:
```
#importing needed libraries
import pandas as pd
import plotly.graph_objs as go
import pandas as pd
import numpy as np
import plotly.offline as py
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import category_encoders as ce
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression
```

Importing datasets:

```
In [2]: # uploading data
        data = pd.read_csv("Downloads/7282_1.csv")
```

Checking shape of data

```
In [3]: # Cheching shape of data
        data.shape
```

```
Out[3]: (35912, 19)
```

Printing the head of data

Out[4]:

ateAdded	reviews.doRecommend	reviews.id	reviews.rating	reviews.text	reviews.title	reviews.userCity	reviews.username	reviews.userProvince
2016-10-00:00:25Z	NaN	NaN	4.0	Pleasant 10 min walk along the sea front to th...	Good location away from the crouds	NaN	Russ (kent)	NaN
2016-10-00:00:25Z	NaN	NaN	5.0	Really lovely hotel. Stayed on the very top fl...	Great hotel with Jacuzzi bath!	NaN	A Traveler	NaN
2016-10-00:00:25Z	NaN	NaN	5.0	Ett mycket bra hotell. Det som drog ner betyge...	Lugnt l��ge	NaN	Maud	NaN
2016-10-00:00:25Z	NaN	NaN	5.0	We stayed here for four nights in October. The...	Good location on the Lido.	NaN	Julie	NaN
2016-10-00:00:25Z	NaN	NaN	5.0	We stayed here for four nights in October. The...	������ ���������������	NaN	sungchul	NaN

Checking on null values

In [5]:
```
# checking for null values
data.isnull().sum()
```

Out[5]:
```
address                 0
categories              0
city                    0
country                 0
latitude               86
longitude              86
name                    0
postalCode             55
province                0
reviews.date          259
reviews.dateAdded       0
reviews.doRecommend   35912
reviews.id            35912
reviews.rating          862
reviews.text             22
reviews.title          1622
reviews.userCity      19649
reviews.username         43
reviews.userProvince  18394
dtype: int64
```

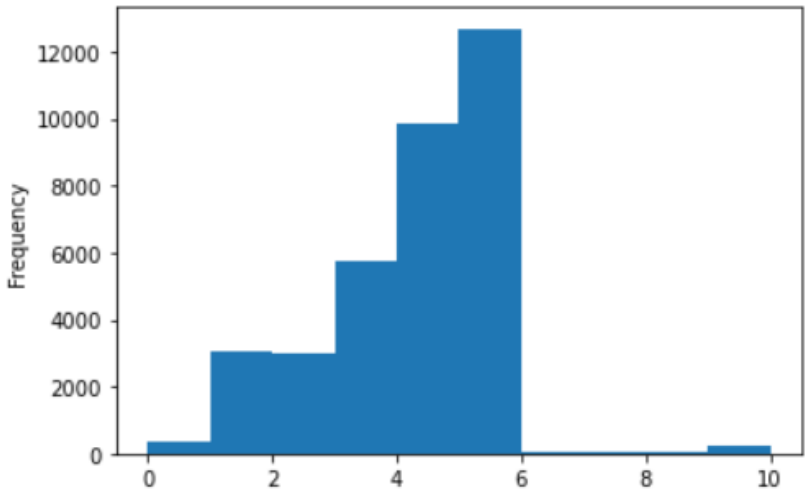
Statistical summary for column 'reviews.rating'

In [6]:
```
# getting stats summary for cloumn ('reviews.rating')
data['reviews.rating'].describe()
```

Out[6]:
```
count    35050.000000
mean         3.776431
std          1.416195
min          0.000000
25%          3.000000
50%          4.000000
75%          5.000000
max         10.000000
Name: reviews.rating, dtype: float64
```

Frequency distribution for column 'reviews.rating'

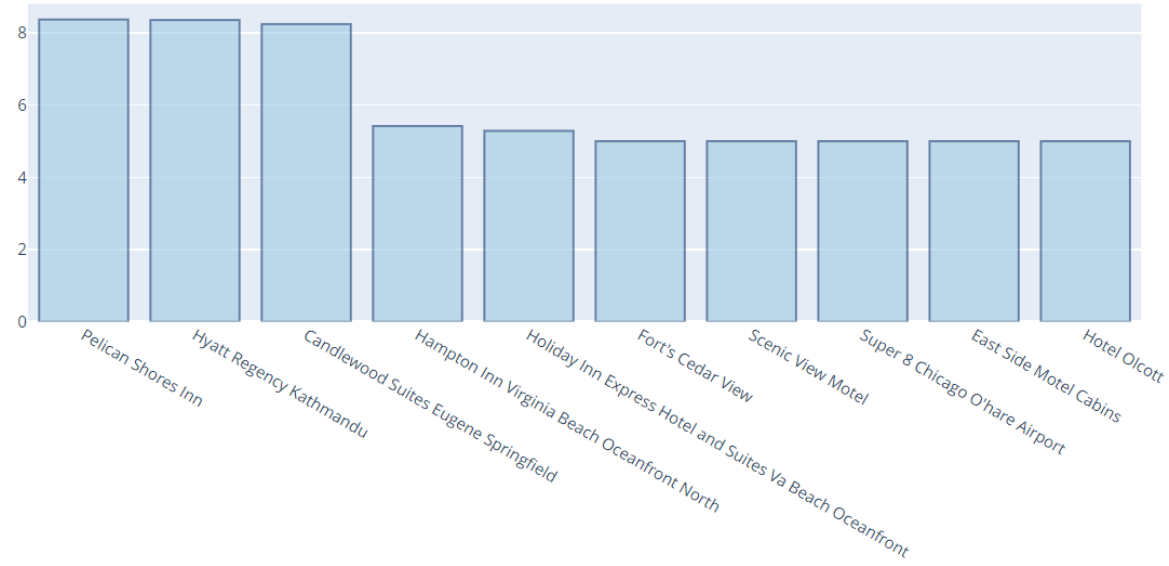
In [7]:
```
# lpoting the frequency distribution for cloumn ('reviews.rating')
data['reviews.rating'].plot(kind='hist', bins=10);
```



Plotting the top 10 hotels with highest average ratings

In [10]:
```
# ploting the top 10 hotels with highiest avarage ratings
q2 = data.groupby('name')['reviews.rating'].mean().reset_index().sort_values(by='reviews.rating', ascending=False)[:10]
trace = go.Bar(
    x=q2['name'],
    y=q2['reviews.rating'],
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data1 = [trace]

layout = go.Layout(
    title='Bar Chat Showing Top 10 Hotels With Highest Average Ratings.',
)

fig = go.Figure(data=data1, layout=layout)
py.iplot(fig, filename='hotel-reviews-highest-rating')
```

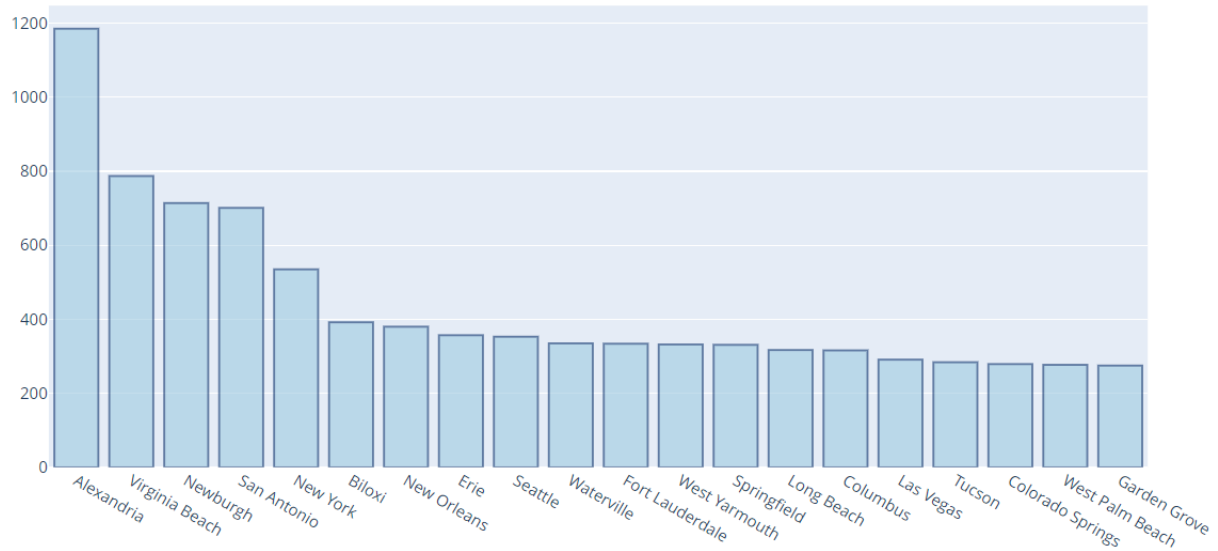
Bar Chat Showing Top 10 Hotels With Highest Average Ratings.



Plotting the top 20 cities with highest reviews

In [11]:
```
#ploting the top 20 cities with highest reviews
q3 = data['city'].value_counts()[:20]
trace = go.Bar(
    x=q3.index,
    y=q3.values,
    marker=dict(
        color='rgb(158,202,225)',
        line=dict(
            color='rgb(8,48,107)',
            width=1.5,
        )
    ),
    opacity=0.6
)

data1 = [trace]

layout = go.Layout(
    title='Bar Chart Showing Top 20 Cities With Highest Reviews.',
)

fig = go.Figure(data=data1, layout=layout)
py.iplot(fig, filename='hotel-reviews-highest-cities')
```

Bar Chart Showing Top 20 Cities With Highest Reviews.



Designing Binary classification target:

In [12]:
```
# Derive binary classification target:
# We define a 'Best' hotels
# overall rating of 5 or higher, on a 10 point scale.
# Drop unrated hotels.

data = data.dropna(subset=['reviews.rating'])
data['Best'] = data['reviews.rating'] >= 5
```

In [13]:
```
# choose a target
y = data['Best']
```

In [14]:
```
# There are 2 classes
# this is a binary classification problem.

y.nunique()
```

Out[14]:
```
2
```

In [15]:
```
# The majority class occurs with 62% frequency,
# so this is not too imbalanced.

y.value_counts(normalize=True)
```

Out[15]:
```
False    0.628302
True     0.371698
Name: Best, dtype: float64
```

Drop unwanted columns

In [20]:
```
# Drop some high cardinality categoricals

data = data.drop(columns=['address', 'latitude', 'longitude', 'reviews.dateAdded', 'reviews.doRecommend', 'reviews.title', 'revie
```

Dealing with null values on

:
```
data['reviews.text'] = data['reviews.text'].fillna('')
```

Converging data record to datetime

In [24]:
```
# Convert date_recorded to datetime

data['reviews.date'] = pd.to_datetime(data['reviews.date'], infer_datetime_format=True)
```

In [25]:
```
data['reviews.text'] = data['reviews.text'].apply(str.lower)
```

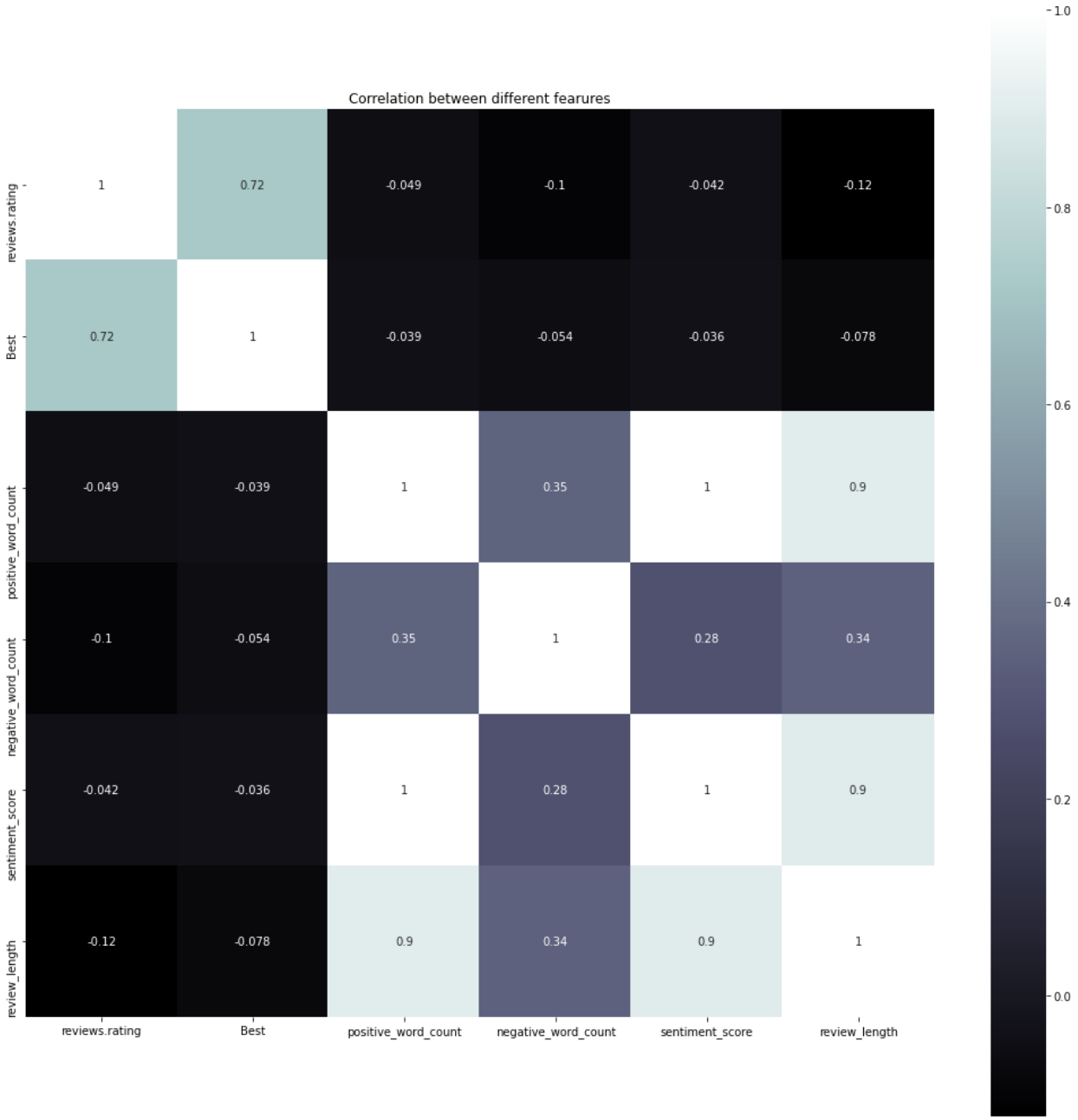
Sentiment analysis

In [26]:
```
# sentiment analysis

positive_words = ['great', 'good', 'pleasant', 'Only' , 'park','No', 'real', 'complaints', 'hotel', 'great',  'location', 'surrou

negative_words = ['bad', 'horrible', 'bad location', 'not clean', 'no','staff', 'bad behaviour','awful', 'sad','lost','failure','



def count_positives(review):
    positive_count = 0
    for word in review.split(' '):
        if word in positive_words:
            positive_count += 1
    return positive_count

def count_negatives(review):
    negative_count = 0
    for word in review.split(' '):
        if word in negative_words:
            negative_count += 1
    return negative_count
```

In [27]:
```
#positive and negative word counts from the review
data['positive_word_count'] = data['reviews.text'].apply(count_positives)
data['negative_word_count'] = data['reviews.text'].apply(count_negatives)

#sentiment score (number of positives words minus the number of negative words)
data['sentiment_score'] = data['positive_word_count'] - data['negative_word_count']

#feature that counts the length of the review- maybe there is a relationship
#between the length of a review and whether or not its a high rating
data['review_length'] = data['reviews.text'].apply(len)
```

Correlation matrix between different features

In [29]:
```
correlation = data.corr()
plt.figure(figsize=(18, 18))
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='bone')

plt.title('Correlation between different fearures')
```


Correlation between different fearures
reviews.rating
Best
positive_word_count
negative_word_count
sentiment_score
review_length
1
0.72
-0.049
-0.1
-0.042
-0.12
0.72
1
-0.039
-0.054
-0.036
-0.078
-0.049
-0.039
1
0.35
1
0.9
-0.1
-0.054
0.35
1
0.28
0.34
-0.042
-0.036
1
0.28
1
0.9
-0.12
-0.078
0.9
0.34
0.9
1
1.0
0.8
0.6
0.4
0.2
0.0

Splitting the data and define a target

In [30]:
```
# split train , test , val
train = data[data['reviews.date'].dt.year <= 2013]
val = data[data['reviews.date'].dt.year == 2014]
test = data[data['reviews.date'].dt.year >= 2015]
```

In [31]:
```
train.shape, val.shape , test.shape
```

Out[31]:
```
((4158, 17), (3342, 17), (27291, 17))
```

In [32]:
```
# set up the features

target = 'Best'

features = data.columns.drop([target, 'reviews.date','reviews.text','reviews.rating'])



# Arrange data into X features matrix and y target vector

X_train = train[features]
y_train = train[target]
X_val = val[features]
y_val = val[target]
X_test = test[features]
```

Modeling

Random Forest Classifier with one hot encoder

In [33]:
```
pipeline = make_pipeline(
    ce.OneHotEncoder(use_cat_names=True),
    SimpleImputer(strategy='median'),
    RandomForestClassifier(n_estimators=100, n_jobs=-1, random_state=0)
)

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_val)

print('Train Accuracy', pipeline.score(X_train, y_train))

print('Validation Accuracy', pipeline.score(X_val, y_val))
```

```
Train Accuracy 0.976911976911977
Validation Accuracy 0.6867145421903053
```

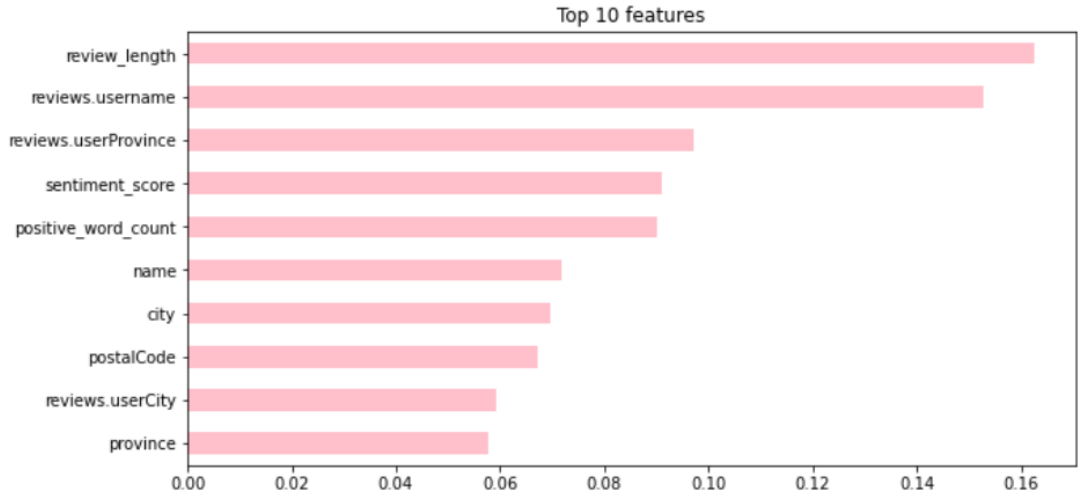
Random Forest model with one hot encoder

In [34]:
```
pipeline = make_pipeline(
    ce.OrdinalEncoder(),
    RandomForestClassifier(n_estimators = 100, n_jobs=-1)
)

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_val)

print('Train Accuracy', pipeline.score(X_train, y_train))

print('Validation Accuracy', pipeline.score(X_val, y_val))
```

```
Train Accuracy 0.976911976911977
Validation Accuracy 0.6642728904847397
```

Visualizing the top 10 important features

In [35]:
```
rf = pipeline.named_steps['randomforestclassifier']
importances = pd.Series(rf.feature_importances_, X_train.columns)

%matplotlib inline
import matplotlib.pyplot as plt

n=10

plt.figure(figsize=(10,n/2))
plt.title(f'Top {n} features')
importances.sort_values()[-n:].plot.barh(color='pink');
```



Decision Tree model

In [36]:
```
pipeline = make_pipeline(
    ce.OrdinalEncoder(),

    DecisionTreeClassifier(random_state=42)
)

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_val)

print('Train Accuracy', pipeline.score(X_train, y_train))

print('Validation Accuracy', pipeline.score(X_val, y_val))
```

```
Train Accuracy 0.976911976911977
Validation Accuracy 0.5864751645721125
```

Linear Regression model

In [37]:
```
pipeline = make_pipeline(
    ce.OrdinalEncoder(),
    LinearRegression()
)

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_val)

print('Train Accuracy', pipeline.score(X_train, y_train))

print('Validation Accuracy', pipeline.score(X_val, y_val))
```

```
Train Accuracy 0.03645228588383287
Validation Accuracy 0.0017811921616731619
```