

***Faculty of Science and Technology***

**Assignment Coversheet**

|  |  |
| --- | --- |
| **Student ID number &**  **Student Name** | u3261091  Oliver Davey |
| **Unit name** | Software Technology 1 |
| **Unit number** | 4483 |
| **Unit Tutor** | Ms. Linda Ma |
| **Assignment name** | ST1 Capstone Project – Semester 1 2023 |
| **Due date** | 12/5/2023 |
| **Date submitted** | 12/5/2023 |

**You must keep a photocopy or electronic copy of your assignment.**

**Student declaration**

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

**Signature of student: Oliver Davey Date: 12/5/2023**

Table of Contents

[Introduction 1](#_Toc131590041)

[Methodology 2](#_Toc131590042)

[Stage 1: Algorithm Design Stage 2](#_Toc131590043)

[Dataset Description 3](#_Toc131590044)

[Exploratory Data Analysis 3](#_Toc131590045)

[Predictive Data Analytics Stage 11](#_Toc131590046)

[Model Preparation and Development 14](#_Toc131590047)

[Stage 2: Algorithm Implementation Stage 22](#_Toc131590048)

[Stage 3: Software Deployment Stage 22](#_Toc131590049)

[Conclusions 23](#_Toc131590050)

[References 23](#_Toc131590051)

# Introduction

This report was produced for the Software Technology 1 Unit Capstone Project. The capstone project requires the completion of an exploratory data analysis (EDA) and predictive data analysis (PDA) on a dataset of our choosing. It then requires that we implement our predictive model into a application that predicts a target value. I decided to work with a New York Rental Property Pricing dataset from the Kaggle [2] the online dataset repository.

New York rental properties consist of four main types: entire homes/apartments, private rooms, shared rooms, and hotel rooms. These four property types are widely distributed throughout New York City throughout different neighbourhoods with varying prices. Often a renter is paying for privacy and location. Each property comes with specific requirements for property renting, such as the length of stay the renter will have. The property price’s, location, whether it’s shared or not and how highly regarded it is, normally defines what type of property you are staying at.

A renter often has a limited amount of money they would be willing to pay for a rental property, they may also have an ideal location and property type in mind. Knowing this, the renter may want to know what chance they have at getting the ideal property type with their budget and location preference. Therefore, it would be useful for potential renters to have access to an application that can determine what kind of property they may be looking at given certain variables including: price, location, how highly regarded by other renters the property is and the number of people who have rented the property. This service would aim to provide an accurate estimation at property types in the New York area based on user values who are looking to rent a property in any of New York’s neighbourhoods.

This report demonstrates the details of building a software application and deploying it. It also aims to show the use of the Python language for data analysis of a chosen dataset, through exploratory data analysis, predictive data analysis, Tkinter for algorithm development and the micro web service tool Flask for deployment of the tool. The methodology details are shown in the following section.

# Methodology

To produce this software application there were multiple stages to go through, these included:

1. The algorithm design stage of predictive models for the dataset, this stage involved performing an EDA and PDA on the NY Rental Property Price data to identify and validate the best choice of predictive model.
2. Implementation followed designing the algorithm, this was in the form of a TKinter GUI that users could input their data into and receive a prediction.
3. Deployment of the implemented solution as a web-based platform

## Stage 1: Algorithm Design Stage

The algorithm design stage is vital to being able to produce an accurate user application. This is due to the algorithm being the driving factor behind prediction accuracy. To get the most accurate algorithm for the dataset an exploratory and predictive data analysis was required. To perform the analysis the dataset had to go through the following process:

1. Loading of the dataset
2. Checking that the loading was successful with basic EDA questions such as what do the first and last five rows look like, what’s the overall dataset shape and how is the data distributed.
3. Removing outliers from the dataset with the help of graphs for visual reference
4. Splitting cleaned data set into a training set and test set for predictive models
5. Formatting all attributes to numerical values to pre-process the data.
6. Building the models from SciKit-Learn
7. Testing the models with our previously split dataset

### Dataset Description

The dataset used in this capstone project is available from Kaggle an online data repository it was, authored by Ivan Chavez. It has 17614 unique observations, 11 attributes and a singular target class/attribute for prediction. The 11 unique attributes include F1 (ID), id (ID), the neighbourhood of the property, the location (latitude and longitude), price, days occupied in 2019, the minimum nights to stay, number of reviews, reviews per month and availability in 2020. The target attribute for the dataset is the room type, it depends upon all the unique attributes. The room type can be one of four types, entire homes/apartments, private rooms, shared rooms, and hotels. Using this data, the goal is to create an algorithm that can with high success guess a room type of any rental property within the New York area.

### Exploratory Data Analysis

As described above in the method section, stage 1 of producing the software involves an exploratory data analysis. An EDA allows us to understand and visualise the dataset that’s being worked with. To do this Google Collab an online notebook software was chosen. This is due to its ability to function independently of the user’s computer abilities and offer its own digital services. The language used in the notebook was Python, it was used to efficiently create graphs and data for the EDA with the use of several imported modules. These modules are included in the script below:

from google.colab import drive

drive.mount("/content/drive")

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import plotly.graph\_objects as go

import plotly.express as px

import seaborn as sns

import warnings

# ignores warnings

warnings.filterwarnings("ignore")

# keeps all plots in the notebook screen

%matplotlib inline

# read the dataset

dataset = pd.read\_csv('/content/drive/MyDrive/MyCapstoneProject/NY Realstate Pricing.csv')

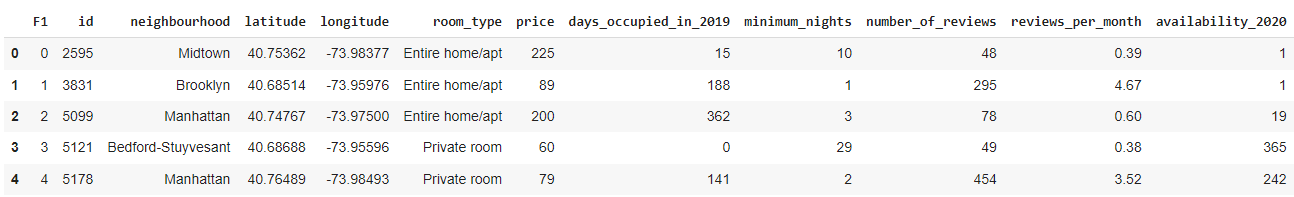
1. The first steps of the EDA explore the dataset at a basic level, the steps are shown below:

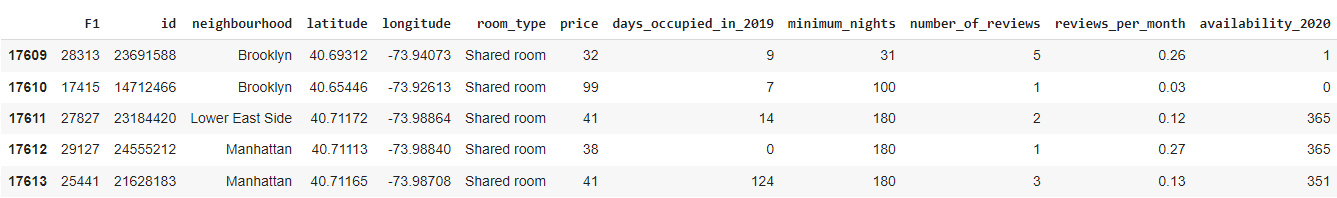
#checking shape of dataset, makes sure it was imported properly

dataset.shape

(17614, 12)

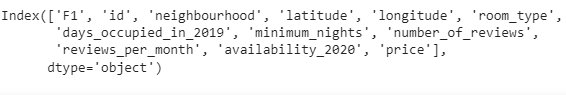
# look at first 5 observations

dataset.head()

dataset.tail() # looks at last 5 observations

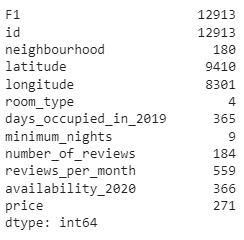
# get attribute names

dataset.columns



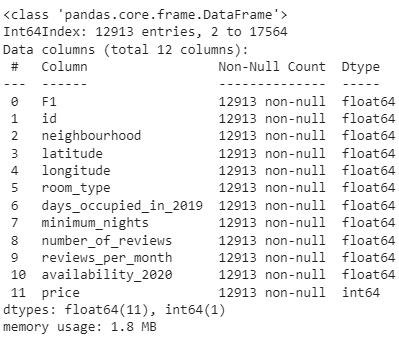
# find how many unique values there are in the data for each attribute

dataset.nunique()



# info about attributes and types in dataset

dataset.info()



# Visualisation of data distribution

# creating subplots and setting plot sizes

figure = plt.figure(figsize = (30,30))

ax = figure.gca()

# accessing database and seperating values into 100 bins

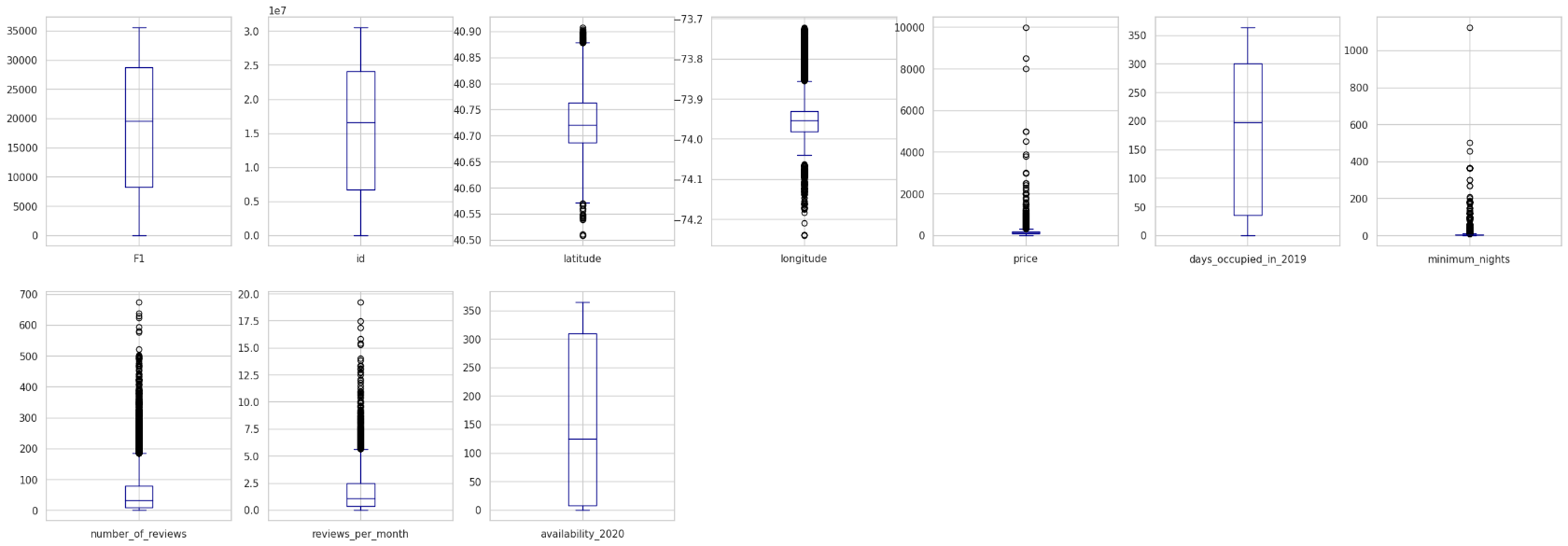
dataset.hist(ax=ax, bins = 100, color = "darkblue")

# display the plots

plt.show()



# showing outliers through boxplots

dataset.plot(kind='box', subplots=True, layout=(2,7), figsize=(30, 10), color='darkblue');

# define outliers

outlier\_list = ['price','minimum\_nights','number\_of\_reviews','reviews\_per\_month']  # list of outliers

# detects and optionally removes outliers

def outliers(dataset\_outliers, drop = False):

    for attribute in dataset\_outliers.columns:

        # getting the outlier attribute data

        outlier\_data = dataset\_outliers[attribute]

        # getting first and third quartile of the outlier data

        Q1 = np.percentile(outlier\_data, 25.)

        Q3 = np.percentile(outlier\_data, 75.)

        interquartile\_range = Q3-Q1

        # determining unacceptable values that are outside of range using constant

        out\_acceptable = interquartile\_range \* 1.5 # 1.5 is a constant for finding outliers

        # subtracting the IQR\*1.5 from Q1 to find bottom outliers, adding IQR\*1.5 to Q2 to find top outliers

        attribute\_outliers = outlier\_data[~((outlier\_data >= Q1 - out\_acceptable) & (outlier\_data <= Q3 + out\_acceptable))].index.tolist()

        if not drop: # show outliers

            print(f"For the feature {attribute}, No of Outliers is {len(attribute\_outliers)}")

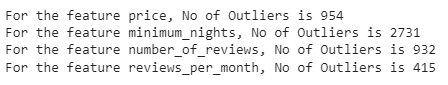
        if drop: # remove the outliers from the attributes

            dataset.drop(attribute\_outliers, inplace = True, errors = 'ignore')

            print(f"{attribute} outliers removed from dataset")

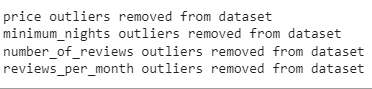
# show the outliers and the amount

outliers(dataset[outlier\_list])



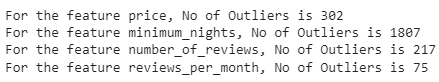
# remove outliers

outliers(dataset[outlier\_list], drop = True)

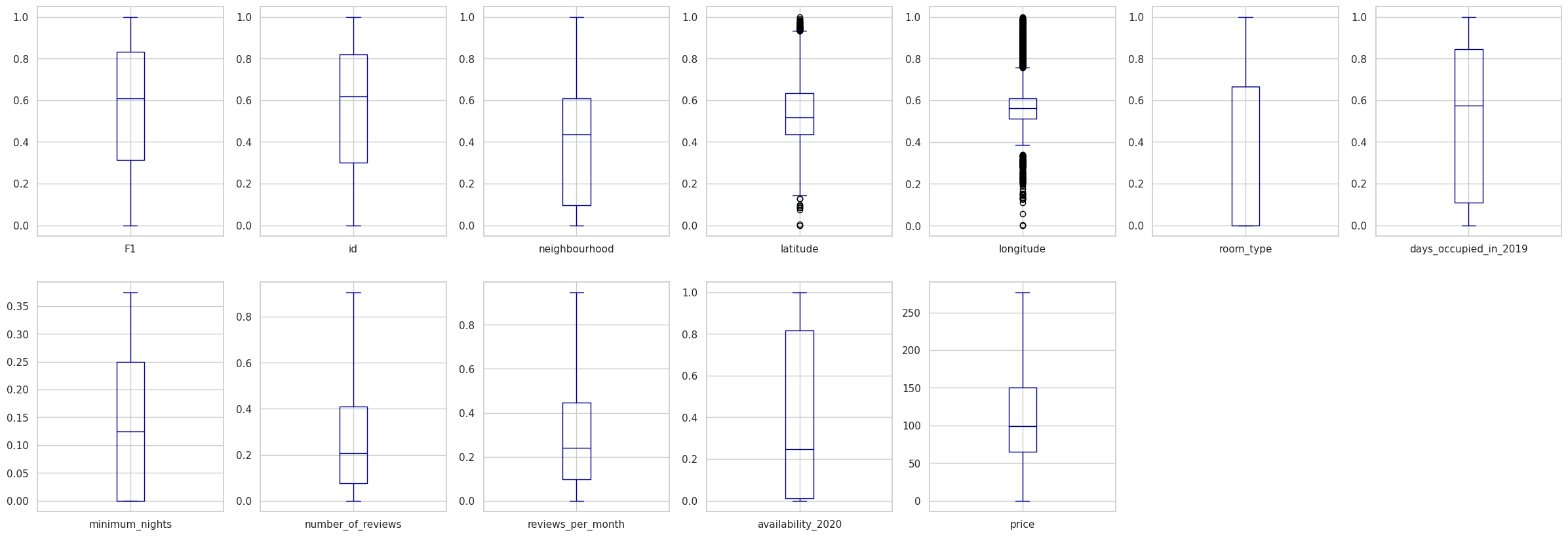


# checking the number of remaining outliers

outliers(dataset[outlier\_list])



# visual check for outlier removal if outliers

dataset.plot(kind='box', subplots=True, layout=(2,7), figsize=(30, 10), color='darkblue');

# checking what data looks like after outlier removal

dataset.shape

(12913, 12)

# room type (target variable) distribution

sns.set(style = "whitegrid") # found style method in seaborn documentation https://seaborn.pydata.org/tutorial/aesthetics.html

# list of potential room types

room\_types = ["Home/apt", "Private room", "Shared room", "Hotel room"]

# get data for graph

data = dataset.room\_type.value\_counts()

# create horizontal bar graph with matplotlib

ax = data.plot.barh(figsize=(10,6))

ax.set\_title("Room Types For Rent")

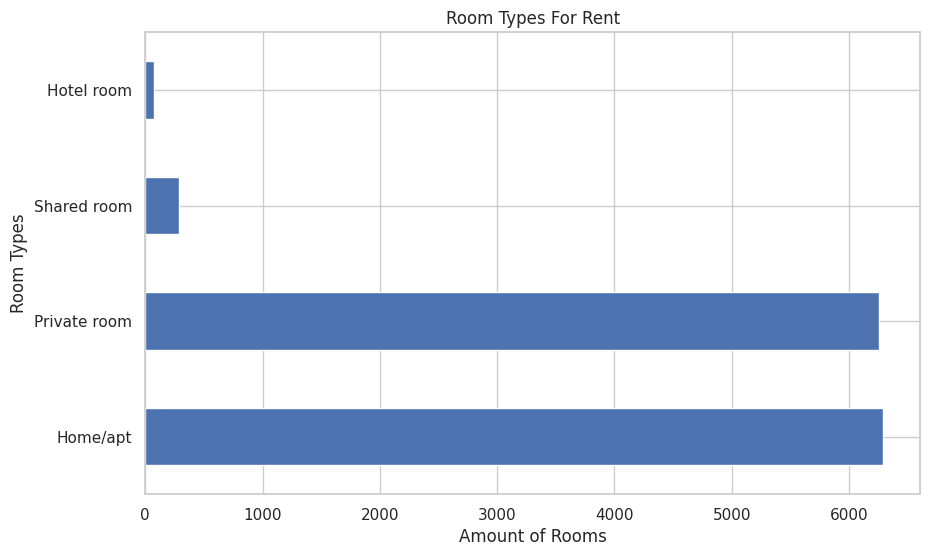
#labels

ax.set\_xlabel("Amount of Rooms")

ax.set\_ylabel("Room Types")

ax.set\_yticklabels(room\_types)

plt.show()



# visualisation of variable relationships with heatmap

# setting figure width, height

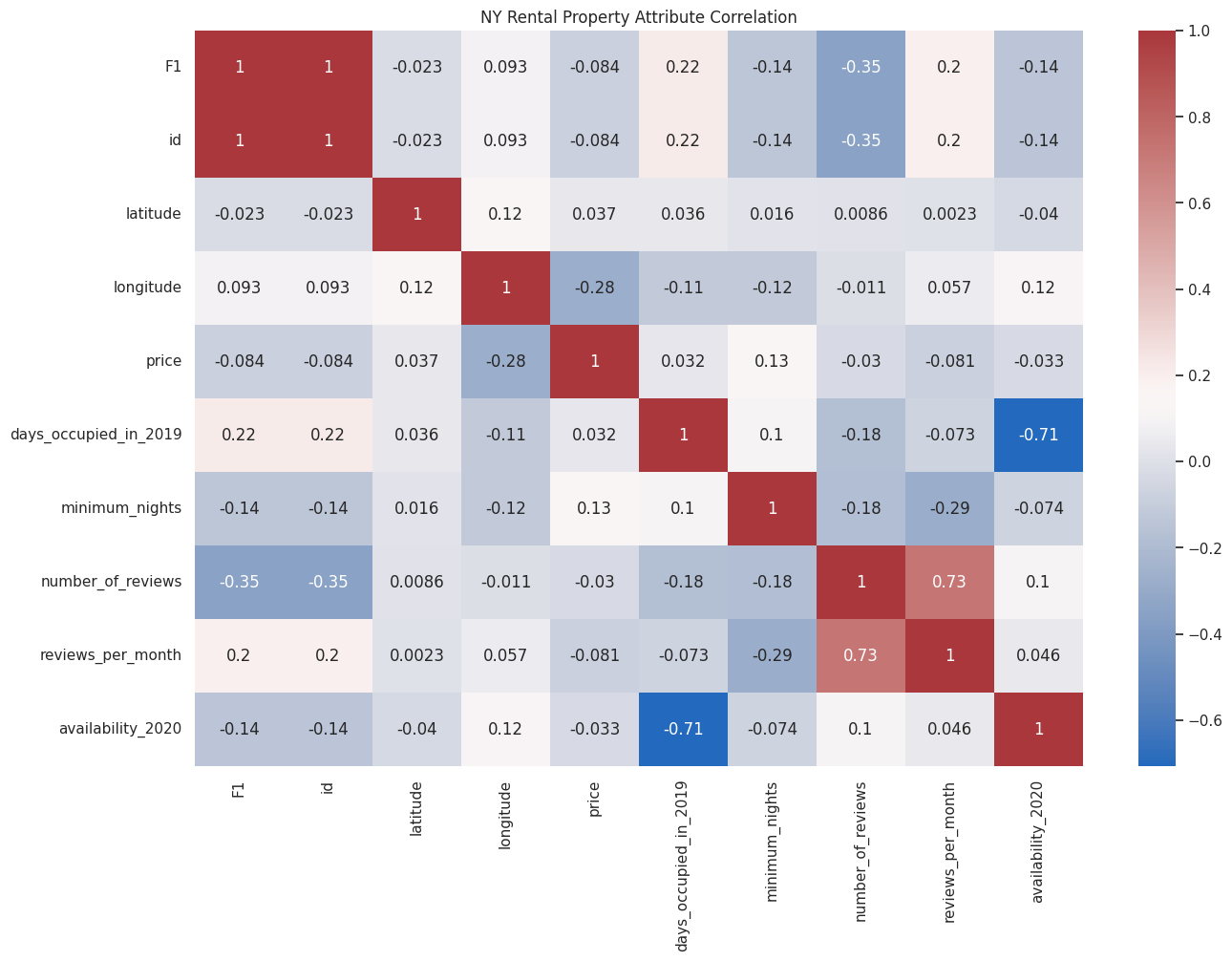
plt.figsize = (15, 10)

# create seaborn heatmap with values in squares

sns.heatmap(dataset.corr(), annot = True, cmap ="vlag") # colour found in seaborn documentation https://seaborn.pydata.org/tutorial/color\_palettes.html

plt.title('NY Rental Property Attribute Correlation')

plt.show()



!pip install <https://github.com/pandas-profiling/pandas-profiling/archive/master.zip>

# profiler report to chack against manual EDA

import pandas as pd

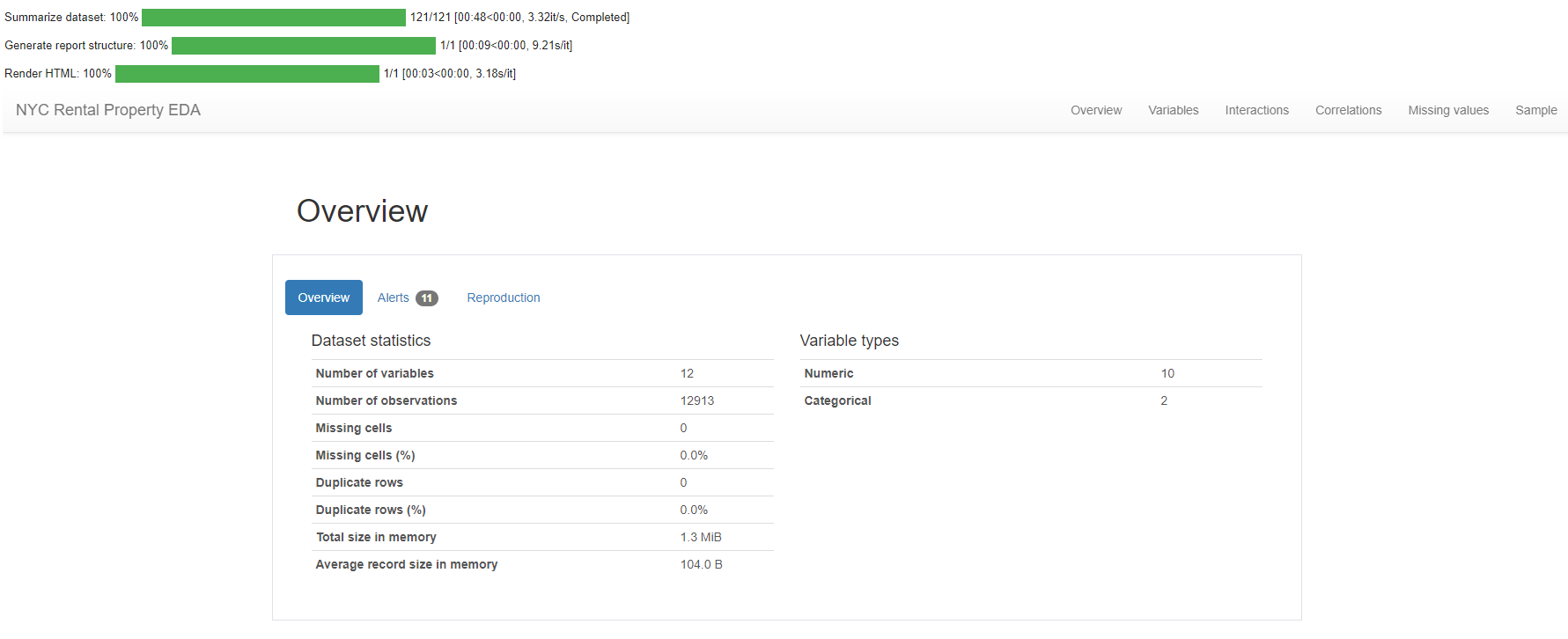
import numpy as np

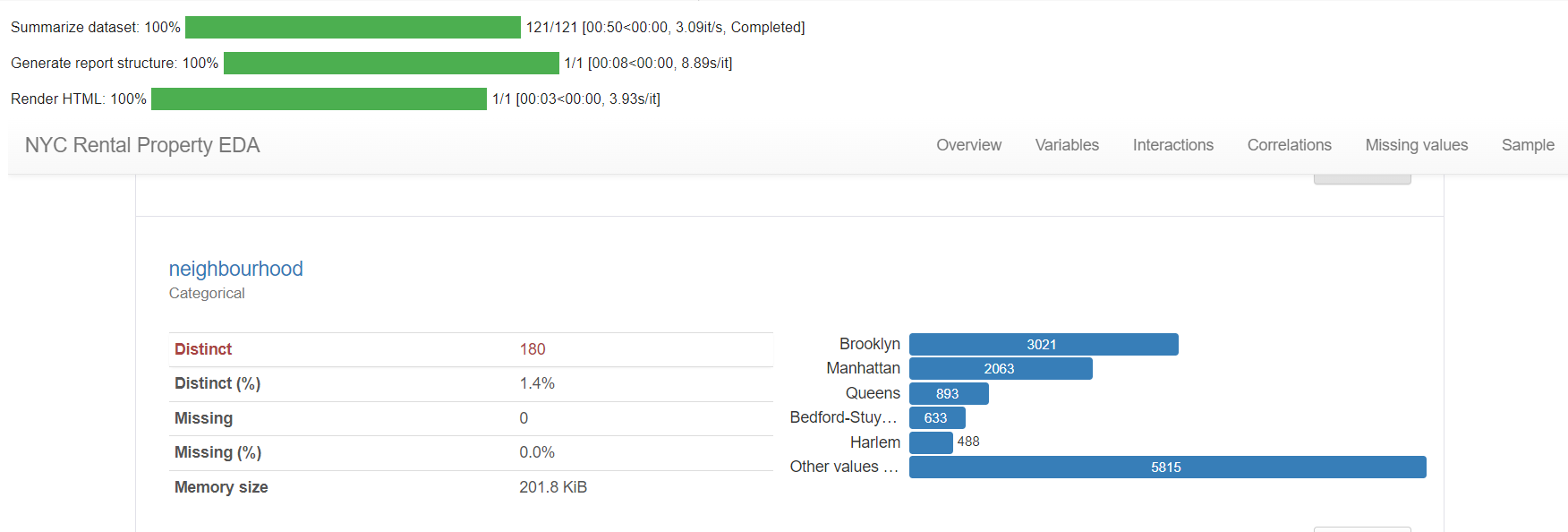
from pandas\_profiling import ProfileReport

# create profile

profile = ProfileReport(dataset,title="NYC Rental Property EDA", html={'style':{'full\_width':True}})

profile.to\_notebook\_iframe()





# Predictive Data Analytics Stage

The Predictive analysis of the NY Rental Properties involved numerus steps that aimed to nominalise and process data before building the predictive algorithm. The first stage of the PDA was to pre process the data which was followed by remodelling. After the data was processed, I could begin testing different predictive models. Once the predictive models had been tested the best model could be identified and tested using confusion matrix’s, prediction, and classification reports.

The pre-processing stage consist of identifying object types in the database then using scikits-learn Ordinal Encoder to assign numeric values to all non int value in the database making them usable by scikit-learns predictive models.

The normalisation of values in the database revolves around normalising the target attribute through its separation from the rest of the database, transformation, and reattachment.

#pre-processing data for prediction

from sklearn.exceptions import DataDimensionalityWarning

#encode object columns to integer values

from sklearn import preprocessing

from sklearn.preprocessing import OrdinalEncoder

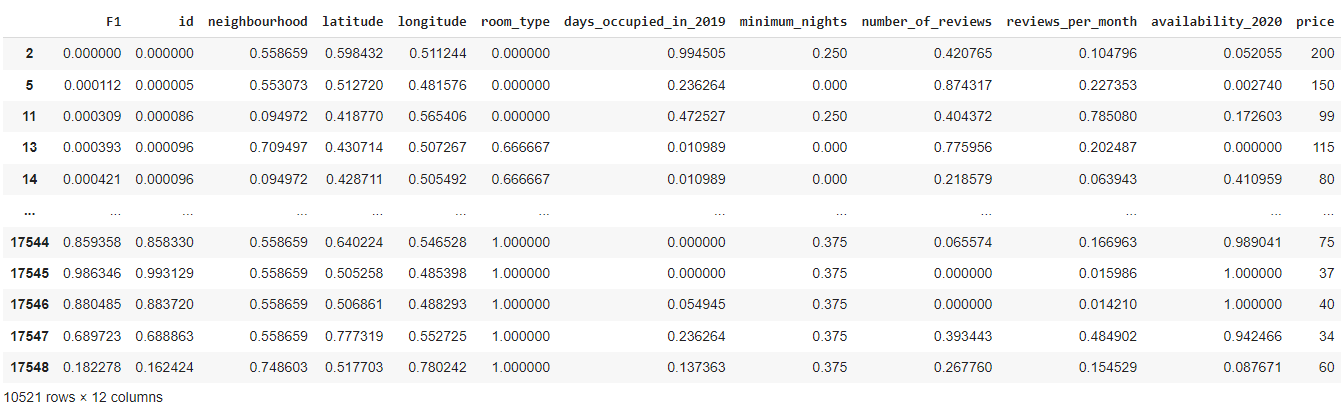
for col in dataset:

    if dataset[col].dtype == "object":

        # reshaping attributes that are objects

        dataset[col]=OrdinalEncoder().fit\_transform(dataset[col].values.reshape(-1,1))

dataset



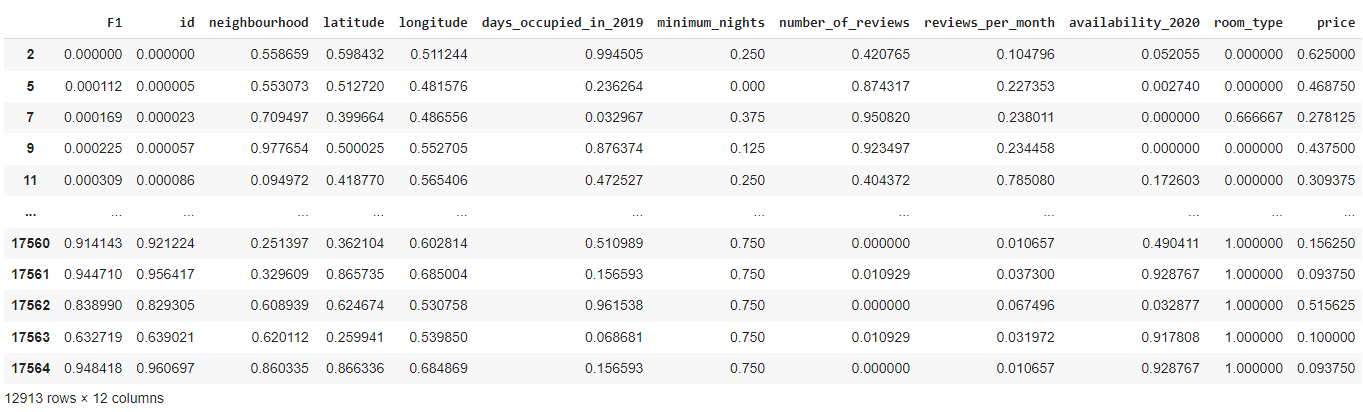
class\_label = dataset['price']

dataset = dataset.drop(['price'], axis =1)

dataset = (dataset-dataset.min())/(dataset.max()-dataset.min())

dataset['price']=class\_label

dataset



# pre-processing dataset for PDA

nyc\_rental\_data = dataset.copy()

le = preprocessing.LabelEncoder()

F1 = le.fit\_transform(list(nyc\_rental\_data["F1"])) # identifier

id = le.fit\_transform(list(nyc\_rental\_data["id"])) # identifier

neighbourhood = le.fit\_transform(list(nyc\_rental\_data["neighbourhood"])) # location

latitude = le.fit\_transform(list(nyc\_rental\_data["latitude"])) # location

longitude = le.fit\_transform(list(nyc\_rental\_data["longitude"])) # location

days\_occupied\_in\_2019 = le.fit\_transform(list(nyc\_rental\_data["days\_occupied\_in\_2019"]))

minimum\_nights = le.fit\_transform(list(nyc\_rental\_data["minimum\_nights"]))

number\_of\_reviews = le.fit\_transform(list(nyc\_rental\_data["number\_of\_reviews"]))

reviews\_per\_month = le.fit\_transform(list(nyc\_rental\_data["reviews\_per\_month"]))

availability\_2020 = le.fit\_transform(list(nyc\_rental\_data["availability\_2020"]))

price = le.fit\_transform(list(nyc\_rental\_data["price"])) # price in USD

room\_type = le.fit\_transform(list(nyc\_rental\_data["room\_type"])) # 4 types

### Model Preparation and Development

## Steps used for machine learning model preparation are described below:

* + Split the data into two categories, testing and training. The training set takes 80% of the entire data leaving 20% for testing after the algorithm is trained on the training set.
  + Separating the target attribute from the rest of the attributes in the training dataset
  + Using the target class create the y\_train
  + Using the remaining attributes form the x\_train
  + Repeat the y\_train and x\_train creation for the test data as well

# attributes used to predict target value

x = list(zip(neighbourhood, latitude, longitude, days\_occupied\_in\_2019, minimum\_nights, number\_of\_reviews, reviews\_per\_month, availability\_2020, price))

# target value, room type

y = list(room\_type)

num\_folds = 5

seed = 7

# what the prediction is scored on

scoring = 'accuracy'

import sklearn.model\_selection

# seperating dataset into test set (80%) and train set (20%)

x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(x, y, test\_size = 0.20, random\_state=seed)

#size of train and test subsets after splitting

np.shape(x\_train), np.shape(x\_test)

((10330, 9), (2583, 9))

# modules from scikit learn to create a PDA, including predictive models

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

from sklearn.metrics import accuracy\_score

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import ExtraTreesClassifier

# list of models used

predictive\_models = []

# four models that will be tested for predictive accuracy

predictive\_models.append(("NB", GaussianNB()))

predictive\_models.append(("SVM", SVC()))

predictive\_models.append(("RF", RandomForestClassifier()))

predictive\_models.append(("GBM", GradientBoostingClassifier()))

model\_results = []

model\_names = []

print("Performance on Training set")

# go through model list and use each one

for name, model in predictive\_models:

    kfold = KFold(n\_splits=num\_folds,shuffle=True,random\_state=seed)

    cv\_results = cross\_val\_score(model, x\_train, y\_train, cv=kfold, scoring="accuracy")

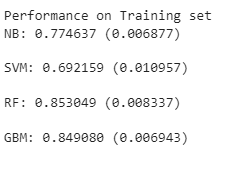
    # adding model name and accuracy to lists

    model\_results.append(cv\_results)

    model\_names.append(name)

    # outputing results of model

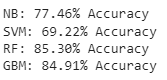
    print(f"{name}: {cv\_results.mean():,.6f} ({cv\_results.std():,.6f})\n")



for i in range(len(model\_names)):

  score = model\_results[i] \* 100

  print(f"{model\_names[i]}: {score.mean():,.2f}% Accuracy")



# comparing the predictive sklearn algorithms performance on the training set

fig = plt.figure()

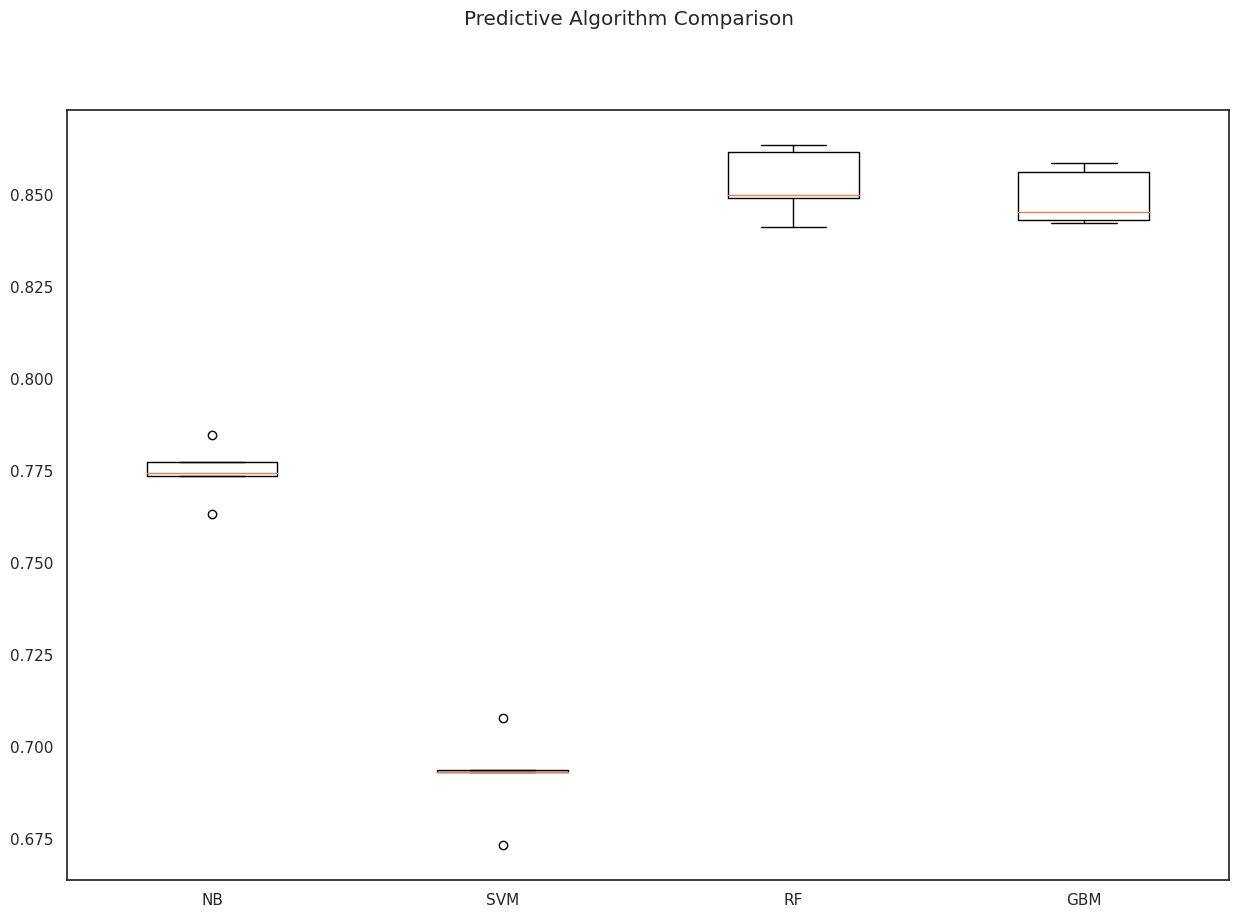
fig.suptitle("Predictive Algorithm Comparison")

ax = fig.add\_subplot(111)

plt.boxplot(model\_results)

ax.set\_xticklabels(model\_names)

plt.show()



# Evaluating the best model, random forest classfier using the test dataset

models.append(('RF', RandomForestClassifier()))

randf = RandomForestClassifier()

best\_model = randf

best\_model.fit(x\_train, y\_train)

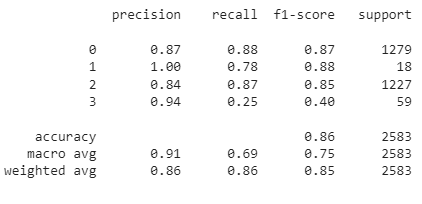
y\_pred = best\_model.predict(x\_test)

print("Best Model Accuracy Score on Test Set:", accuracy\_score(y\_test, y\_pred))



# classification report evaluation of best model

print(classification\_report(y\_test, y\_pred))



# confusion matrix evaluation report for best model

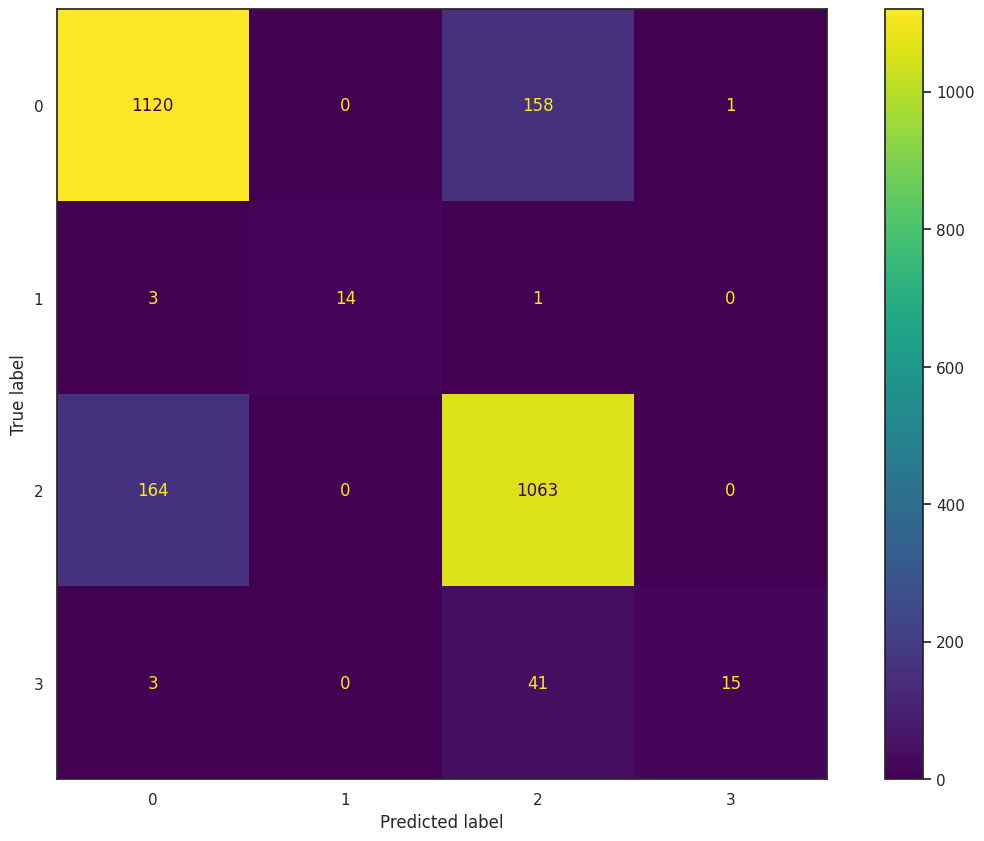
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

cm = confusion\_matrix(y\_test, y\_pred)

display = ConfusionMatrixDisplay(confusion\_matrix=cm)

display.plot()

plt.show()



# prediction report for best model

for x in range(len(y\_pred)): # too many values to look at so first 100 results are produced

  print("Predicted: ", y\_pred[x], "Actual: ", y\_test[x], "Data: ", x\_test[x],)

(First 100 values)

|  |
| --- |
| Predicted: 2 Actual: 2 Data: (154, 8853, 7697, 207, 2, 51, 166, 154, 32)  Predicted: 0 Actual: 0 Data: (100, 4767, 1379, 361, 1, 18, 29, 1, 207)  Predicted: 2 Actual: 2 Data: (86, 6443, 7064, 85, 0, 19, 38, 328, 31)  Predicted: 2 Actual: 0 Data: (152, 5984, 6068, 52, 2, 0, 0, 330, 66)  Predicted: 2 Actual: 2 Data: (111, 382, 2911, 274, 1, 20, 108, 63, 36)  Predicted: 0 Actual: 0 Data: (93, 1216, 3426, 306, 2, 114, 279, 125, 82)  Predicted: 0 Actual: 2 Data: (65, 1905, 1631, 301, 2, 7, 42, 0, 86)  Predicted: 0 Actual: 0 Data: (17, 2820, 5579, 220, 1, 74, 501, 0, 181)  Predicted: 0 Actual: 2 Data: (105, 5731, 573, 74, 3, 179, 168, 296, 135)  Predicted: 2 Actual: 2 Data: (134, 7185, 7799, 290, 1, 23, 96, 66, 31)  Predicted: 2 Actual: 2 Data: (78, 8527, 3417, 335, 4, 2, 12, 0, 61)  Predicted: 0 Actual: 0 Data: (17, 1054, 2975, 282, 2, 1, 4, 83, 174)  Predicted: 2 Actual: 2 Data: (17, 5382, 3253, 26, 0, 23, 39, 308, 41)  Predicted: 2 Actual: 2 Data: (80, 8819, 5377, 8, 0, 172, 171, 0, 36)  Predicted: 0 Actual: 0 Data: (100, 8439, 3720, 357, 3, 70, 129, 0, 101)  Predicted: 0 Actual: 0 Data: (99, 4697, 1280, 356, 1, 0, 8, 0, 179)  Predicted: 0 Actual: 0 Data: (100, 4690, 1539, 288, 2, 163, 272, 33, 217)  Predicted: 2 Actual: 2 Data: (17, 2422, 3860, 16, 1, 32, 276, 321, 51)  Predicted: 0 Actual: 0 Data: (100, 8236, 3708, 357, 0, 2, 25, 0, 184)  Predicted: 2 Actual: 2 Data: (17, 910, 4886, 17, 2, 44, 255, 338, 41)  Predicted: 2 Actual: 2 Data: (10, 3075, 4036, 286, 0, 8, 58, 90, 26)  Predicted: 2 Actual: 2 Data: (55, 1058, 3248, 287, 0, 83, 216, 57, 38)  Predicted: 0 Actual: 0 Data: (38, 1950, 2979, 202, 2, 123, 261, 144, 135)  Predicted: 2 Actual: 2 Data: (17, 2242, 6741, 56, 5, 14, 85, 155, 28)  Predicted: 0 Actual: 0 Data: (151, 459, 9, 280, 2, 21, 52, 146, 126)  Predicted: 2 Actual: 2 Data: (151, 381, 48, 247, 0, 44, 337, 0, 66)  Predicted: 0 Actual: 2 Data: (10, 3272, 5078, 289, 0, 80, 390, 74, 86)  Predicted: 0 Actual: 0 Data: (17, 2596, 1874, 15, 0, 6, 32, 278, 242)  Predicted: 2 Actual: 0 Data: (134, 3316, 7556, 214, 6, 2, 37, 0, 37)  Predicted: 2 Actual: 2 Data: (17, 3707, 6198, 11, 1, 43, 137, 346, 15)  Predicted: 0 Actual: 0 Data: (151, 536, 121, 214, 3, 135, 422, 124, 61)  Predicted: 0 Actual: 0 Data: (175, 4557, 4485, 297, 1, 49, 398, 67, 159)  Predicted: 0 Actual: 0 Data: (45, 825, 5298, 267, 2, 29, 178, 91, 111)  Predicted: 0 Actual: 2 Data: (3, 6769, 5726, 349, 3, 20, 160, 23, 66)  Predicted: 0 Actual: 0 Data: (17, 3160, 5854, 312, 4, 24, 207, 0, 160)  Predicted: 2 Actual: 2 Data: (17, 1255, 7039, 185, 2, 17, 97, 180, 37)  Predicted: 2 Actual: 2 Data: (78, 7863, 4674, 31, 1, 108, 377, 340, 68)  Predicted: 2 Actual: 2 Data: (154, 9038, 6869, 278, 2, 25, 62, 0, 27)  Predicted: 0 Actual: 2 Data: (99, 4589, 1594, 144, 4, 24, 38, 193, 155)  Predicted: 2 Actual: 2 Data: (163, 9169, 6523, 9, 2, 29, 98, 355, 26)  Predicted: 0 Actual: 0 Data: (134, 2374, 7860, 18, 1, 40, 293, 351, 150)  Predicted: 0 Actual: 0 Data: (21, 2745, 6267, 91, 2, 93, 164, 297, 207)  Predicted: 2 Actual: 2 Data: (17, 1621, 3329, 312, 0, 72, 455, 64, 56)  Predicted: 2 Actual: 2 Data: (100, 9004, 4602, 84, 2, 36, 130, 286, 11)  Predicted: 0 Actual: 0 Data: (139, 9393, 6848, 170, 5, 8, 29, 213, 140)  Predicted: 2 Actual: 2 Data: (22, 604, 6569, 190, 0, 89, 292, 79, 36)  Predicted: 2 Actual: 2 Data: (17, 2998, 4985, 226, 3, 6, 45, 126, 61)  Predicted: 0 Actual: 0 Data: (3, 6955, 6181, 360, 2, 5, 40, 0, 111)  Predicted: 1 Actual: 1 Data: (17, 3091, 3442, 6, 0, 15, 99, 364, 23)  Predicted: 0 Actual: 0 Data: (17, 2013, 3674, 306, 2, 5, 17, 59, 86)  Predicted: 2 Actual: 2 Data: (17, 1296, 1269, 22, 1, 63, 370, 348, 41)  Predicted: 2 Actual: 2 Data: (155, 171, 8209, 149, 5, 5, 11, 287, 24)  Predicted: 2 Actual: 2 Data: (100, 5842, 1984, 349, 2, 6, 55, 0, 85)  Predicted: 0 Actual: 0 Data: (17, 4459, 4658, 270, 2, 35, 189, 141, 207)  Predicted: 0 Actual: 0 Data: (100, 3819, 429, 353, 4, 51, 256, 229, 140)  Predicted: 2 Actual: 2 Data: (100, 6037, 1842, 358, 0, 13, 91, 0, 75)  Predicted: 0 Actual: 0 Data: (43, 383, 365, 43, 1, 67, 279, 344, 86)  Predicted: 2 Actual: 2 Data: (55, 1325, 3810, 112, 1, 91, 381, 214, 46)  Predicted: 2 Actual: 2 Data: (48, 4751, 2146, 217, 3, 152, 229, 142, 71)  Predicted: 2 Actual: 2 Data: (134, 3804, 6693, 215, 0, 37, 96, 2, 31)  Predicted: 0 Actual: 2 Data: (110, 6665, 2660, 176, 2, 82, 180, 334, 179)  Predicted: 0 Actual: 0 Data: (100, 4742, 1348, 324, 3, 105, 241, 144, 191)  Predicted: 2 Actual: 2 Data: (175, 4117, 2782, 19, 1, 68, 409, 349, 96)  Predicted: 2 Actual: 2 Data: (61, 2983, 2363, 15, 1, 171, 165, 335, 81)  Predicted: 2 Actual: 0 Data: (38, 1679, 4470, 239, 0, 27, 243, 55, 66)  Predicted: 2 Actual: 0 Data: (17, 1948, 4941, 122, 6, 116, 135, 330, 68)  Predicted: 2 Actual: 2 Data: (78, 8551, 3507, 146, 2, 16, 25, 279, 46)  Predicted: 0 Actual: 2 Data: (138, 3607, 6491, 6, 6, 13, 21, 328, 86)  Predicted: 0 Actual: 0 Data: (21, 3396, 5318, 332, 2, 6, 31, 66, 96)  Predicted: 2 Actual: 2 Data: (134, 5967, 6033, 42, 0, 106, 481, 316, 46)  Predicted: 0 Actual: 0 Data: (100, 5585, 656, 357, 0, 49, 205, 35, 207)  Predicted: 0 Actual: 0 Data: (17, 1265, 6856, 245, 3, 177, 329, 63, 55)  Predicted: 0 Actual: 0 Data: (175, 4175, 2652, 321, 2, 19, 73, 32, 165)  Predicted: 0 Actual: 2 Data: (100, 7880, 3445, 278, 1, 14, 22, 0, 130)  Predicted: 2 Actual: 2 Data: (17, 2602, 6112, 28, 1, 70, 162, 339, 46)  Predicted: 0 Actual: 0 Data: (26, 4269, 1214, 359, 2, 52, 103, 112, 155)  Predicted: 0 Actual: 0 Data: (45, 1039, 5864, 11, 3, 84, 164, 354, 125)  Predicted: 2 Actual: 2 Data: (140, 6682, 4029, 360, 2, 33, 128, 326, 71)  Predicted: 2 Actual: 2 Data: (61, 2735, 2273, 25, 4, 3, 13, 218, 85)  Predicted: 0 Actual: 0 Data: (10, 2907, 3653, 205, 2, 158, 444, 0, 96)  Predicted: 2 Actual: 0 Data: (100, 8398, 3718, 69, 0, 43, 165, 0, 116)  Predicted: 2 Actual: 2 Data: (133, 2124, 2455, 270, 0, 58, 434, 0, 41)  Predicted: 2 Actual: 2 Data: (17, 420, 5212, 57, 0, 8, 21, 338, 31)  Predicted: 2 Actual: 2 Data: (79, 6606, 1170, 195, 0, 82, 514, 254, 96)  Predicted: 2 Actual: 2 Data: (10, 2849, 3698, 210, 1, 36, 140, 133, 76)  Predicted: 2 Actual: 2 Data: (17, 3782, 4894, 10, 1, 44, 272, 355, 66)  Predicted: 0 Actual: 2 Data: (17, 2287, 7185, 204, 1, 65, 353, 339, 85)  Predicted: 2 Actual: 2 Data: (17, 1562, 6395, 0, 2, 3, 25, 365, 106)  Predicted: 2 Actual: 2 Data: (100, 9088, 5394, 190, 3, 42, 285, 147, 56)  Predicted: 2 Actual: 2 Data: (100, 7278, 4024, 295, 0, 135, 410, 166, 112)  Predicted: 2 Actual: 0 Data: (38, 2043, 4286, 231, 2, 1, 3, 222, 56)  Predicted: 0 Actual: 0 Data: (134, 6489, 6153, 96, 2, 87, 141, 258, 165)  Predicted: 0 Actual: 0 Data: (17, 2634, 4624, 197, 1, 43, 61, 41, 136)  Predicted: 2 Actual: 0 Data: (55, 1001, 3466, 330, 1, 0, 5, 5, 51)  Predicted: 2 Actual: 2 Data: (134, 6436, 7920, 192, 1, 27, 131, 177, 32)  Predicted: 0 Actual: 2 Data: (17, 2987, 1095, 343, 3, 7, 58, 0, 84)  Predicted: 2 Actual: 3 Data: (55, 971, 3799, 327, 1, 71, 258, 54, 34)  Predicted: 0 Actual: 0 Data: (17, 1971, 2660, 197, 3, 13, 17, 158, 179)  Predicted: 2 Actual: 2 Data: (17, 918, 399, 360, 6, 12, 22, 0, 26)  Predicted: 2 Actual: 2 Data: (175, 4335, 4780, 0, 3, 11, 15, 365, 121) |

## Stage 2: Algorithm Implementation Stage

1. Using my best performing algorithm for the NY Rental Property Prices dataset, which was the Random Forest, implement it using the TKinter Python module.
2. Use Flask a micro web framework to then deploy the implemented solution.

After performing the EDA and PDA to identify the best room type predictive algorithm and model in the first stage of the capstone project, Tkinter is used to implement a graphical user interface to access the model and algorithm.

The EDA, PDA and Tkinter Pycharm Project can be found at this google drive link:

## <https://drive.google.com/drive/folders/1E528TxyitIBrgy2YdTPbyYGWpppT3Iql?usp=sharing>

## Stage 3: Software Deployment Stage

The last stage is the Deployment stage, and it involves using the flask micro web service to deploy the algorithm. This is done after stages 1, with the EDA and PDA, and stage 2 which implements the algorithm designed in stage 1. This is required for ease of access as the Tkinter application is stored locally and unavailable to people who might be interested in the tool.

# Conclusions

This report is wholistic picture of the design, implementation, and deployment of predictive algorithms. It covers the process of performing data analysis. Implementing using Tkinter and deploying using Flask. The predictive tool I developed allows people who are interested in renting a new York property to see what room type they may be getting.

## References

1. <https://uclearn.canberra.edu.au/courses/13571/assignments/105232>
2. <https://www.kaggle.com/datasets/ivanchvez/ny-rental-properties-pricing>
3. <https://seaborn.pydata.org/tutorial/color_palettes.html>