```
import re
import sys

import time
import datetime

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

```
%matplotlib inline
sns.set(style = 'darkgrid')
sns.set_palette('PuBuGn_d')
```

```
#Loading the dataset
df = pd.read_csv("googleplaystore.csv")
```

Data Exploration and Cleaning

```
df.head(10)
# Display the first 10 rows of the dataset as shown below
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid &	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up

```
# Checking the data type of the columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             10841 non-null  object 
 1   Category        10841 non-null  object 
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object 
 4   Size            10841 non-null  object 
 5   Installs        10841 non-null  object 
 6   Type            10840 non-null  object 
 7   Price           10841 non-null  object 
 8   Content Rating  10840 non-null  object 
 9   Genres          10841 non-null  object 
 10  Last Updated    10841 non-null  object 
 11  Current Ver     10833 non-null  object 
 12  Android Ver     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

The dataset has 10,841 records and 13 columns (labels), all of them are object types except the target column (Rating) which is float.

```
# Exploring missing data and checking if any has NaN values
plt.figure(figsize=(7, 5))
sns.heatmap(df.isnull(), cmap='viridis')
df.isnull().any()
```

```
App               False
Category          False
Rating             True
Reviews           False
Size              False
Installs          False
Type               True
Price             False
Content Rating     True
Genres            False
Last Updated      False
Current Ver        True
Android Ver        True
dtype: bool
```

0
452
904
1356
1808
2260
1.0
0.8

Looks like there are missing values in "Rating", "Type", "Content Rating" and " Android Ver". But most of these missing values in Rating column.

```
df.isnull().sum()
```

```
App                  0
Category             0
Rating            1474
Reviews              0
Size                 0
Installs             0
Type                 1
Price                0
Content Rating       1
Genres               0
Last Updated         0
Current Ver          8
Android Ver          3
dtype: int64
```

There are two strategies to handle missing data:

- either removing records with these missing values or,
- replacing missing values with a specific value like (mean, median or mode) value of the column.

The best way to fill missing values might be using median instead of mean here.

```
df['Rating'] = df['Rating'].fillna(df['Rating'].median())

# Before filling null values we have to clean all non numerical values & unicode charachters
replaces = [u'\u00AE', u'\u2013', u'\u00C3', u'\u00E3', u'\u00B3', '[', ']', "'"]
for i in replaces:
  df['Current Ver'] = df['Current Ver'].astype(str).apply(lambda x : x.replace(i, ''))

regex = [r'[-+|/:/;(_)@]', r'\s+', r'[A-Za-z]+']
for j in regex:
  df['Current Ver'] = df['Current Ver'].astype(str).apply(lambda x : re.sub(j, '0', x))
```

```
df['Current Ver'] = df['Current Ver'].astype(str).apply(lambda x : x.replace('.', ',',1).replace('.', '').replace(',', '.',1)).astype(float)
df['Current Ver'] = df['Current Ver'].fillna(df['Current Ver'].median())
```

```
# Count the number of unique values in category column
df['Category'].unique()
```

```
array(['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY',
       'BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION',
       'DATING', 'EDUCATION', 'ENTERTAINMENT', 'EVENTS', 'FINANCE',
       'FOOD_AND_DRINK', 'HEALTH_AND_FITNESS', 'HOUSE_AND_HOME',
       'LIBRARIES_AND_DEMO', 'LIFESTYLE', 'GAME', 'FAMILY', 'MEDICAL',
       'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS', 'TRAVEL_AND_LOCAL',
       'TOOLS', 'PERSONALIZATION', 'PRODUCTIVITY', 'PARENTING', 'WEATHER',
       'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_AND_NAVIGATION',
       '1.9'], dtype=object)
```

```
# Check the record  of unreasonable value which is 1.9
i = df[df['Category'] == '1.9'].index
df.loc[i]
```

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
Life Made WI-Fi										February		

It's obvious that the first value (App Name) of this record is missing and all other values are respectively propagated backward starting from "Category" towards the "Current Ver"; and the last column which is "Android Ver" is left null.
It's better to drop the entire recored instead of consider these unreasonable values while cleaning each column.

```
df = df.drop(i)
```

```
# Removing NaN values
df = df[pd.notnull(df['Last Updated'])]
df = df[pd.notnull(df['Content Rating'])]
```

Categorical Data Encoding

We need to make all data ready for the model, so we will convert categorical variables (variables that stored as text values) into numerical variables.

```
# App values (or "name") encoding
le = preprocessing.LabelEncoder()
df['App'] = le.fit_transform(df['App'])
# This encoder converts the app names into numeric values
```

```
# Category features encoding
category_list = df['Category'].unique().tolist()
category_list = ['cat_' + word for word in category_list]
df = pd.concat([df, pd.get_dummies(df['Category'], prefix='cat')], axis=1)
```

```
# Genres features encoding
le = preprocessing.LabelEncoder()
df['Genres'] = le.fit_transform(df['Genres'])
```

```
# Encode Content Rating features
le = preprocessing.LabelEncoder()
df['Content Rating'] = le.fit_transform(df['Content Rating'])
```

```
# Price cleaning
df['Price'] = df['Price'].apply(lambda x : x.strip('$'))
```

```
# Installs cleaning
df['Installs'] = df['Installs'].apply(lambda x : x.strip('+').replace(',', ''))
```

```
# Type encoding
df['Type'] = pd.get_dummies(df['Type'])
```

The above line drops the reference column and just keeps only one of the two columns as retaining this extra column does not add any new information for the modeling process, this line is exactly the same as setting drop_first parameter to True.

```
# Last Updated encoding
df['Last Updated'] = df['Last Updated'].apply(lambda x : time.mktime(datetime.datetime.strptime(x, '%B %d, %Y').timetuple()))
```

```
# Convert kbytes to Mbytes
k_indices = df['Size'].loc[df['Size'].str.contains('k')].index.tolist()
converter = pd.DataFrame(df.loc[k_indices, 'Size'].apply(lambda x: x.strip('k')).astype(float).apply(lambda x: x / 1024).apply(lambda x: round(x, 3)).astype(str))
df.loc[k_indices,'Size'] = converter
```

This can be done by selecting all k values from the "Size" column and replace those values by their corresponding M values, and since k indices belong to a list of non-consecutive numbers, a new dataframe (converter) will be created with these k indices to perform the conversion, then the final values will be assigned back to the "Size" column.

```
# Size cleaning
df['Size'] = df['Size'].apply(lambda x: x.strip('M'))
df[df['Size'] == 'Varies with device'] = 0
df['Size'] = df['Size'].astype(float)
```

Applying Prediction Models

- k-Nearest Neighbours
- Random Forest

Here we will see how k-nearest neighbors and random forests can be used to predict app ratings based on the other matrices. First, the dataset has to separate into dependent and independent variables (or features and labels). Then those variables have to split into a training and test set.

During training stage we give the model both the features and the labels so it can learn to classify points based on the features.

Splitting the data into Training Set and Test Set

[ ] ↳ 5 cells hidden

1. k-Nearest Neighbours Model

The k-nearest neighbors algorithm is based around the simple idea of predicting unknown values by matching them with the most similar known values.

Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset — its "nearest neighbors".

```
# Look at the 15 closest neighbors
model1 = KNeighborsRegressor(n_neighbors=15)
```

```
# Find the mean accuracy of knn regression using X_test and y_test
model1.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=15, p=2,
                    weights='uniform')
```

Minkowski distance used.

```
predicted = model1.predict(X_test)
print("\nPredicted Set: \n", predict)
test = y_test.to_numpy()
print("\nTest set:\n", test)
```

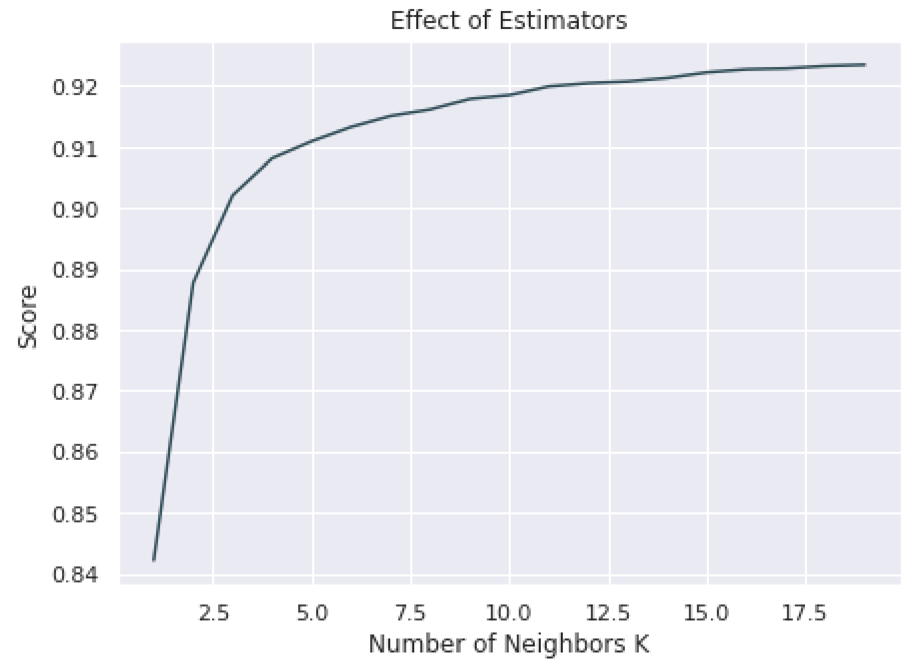
```

Predicted Set: 
 [4.25789474 4.32105263 4.12631579 ... 0.         4.31052632 4.19473684]

Test set:
 [4.1 4.4 4.5 ... 0.  4.4 4.4]
```

```
# Calculate the mean accuracy of the KNN model
accuracy = model1.score(X_test,y_test)
'Accuracy: ' + str(np.round(accuracy*100, 2)) + '%'
```

```
'Accuracy: 92.22%'
```

```
# Try different numbers of n_estimators
n_neighbors = np.arange(1, 20, 1)
scores = []
for n in n_neighbors:
    model1.set_params(n_neighbors=n)
    model1.fit(X_train, y_train)
    scores.append(model1.score(X_test, y_test))
plt.figure(figsize=(7, 5))
plt.title("Effect of Estimators")
plt.xlabel("Number of Neighbors K")
plt.ylabel("Score")
plt.plot(n_neighbors, scores)
```

```
[<matplotlib.lines.Line2D at 0x7f04665aec88>]
```



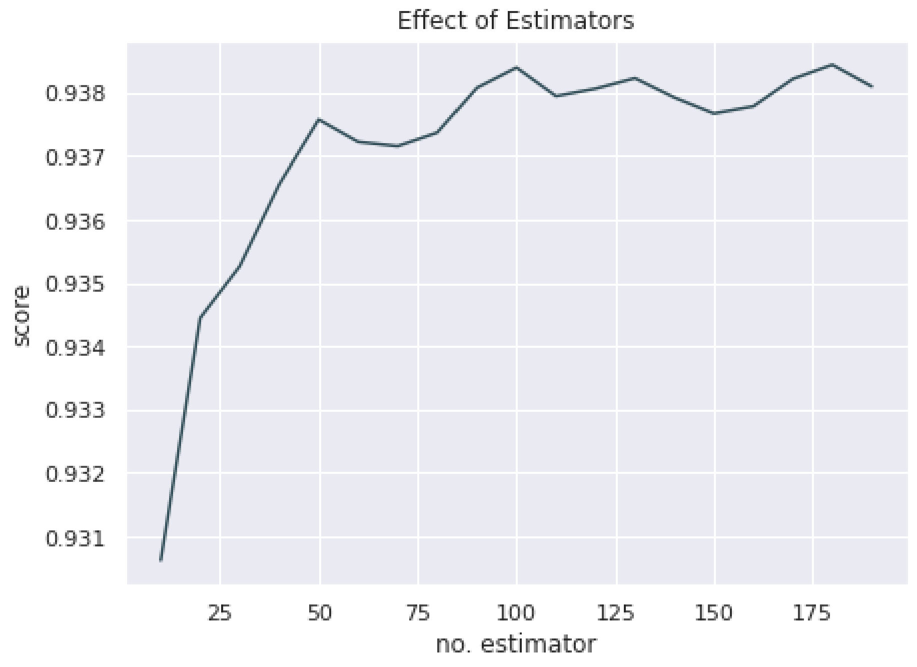
Random Forest Model

The RandomForestRegressor class of the sklearn.ensemble library is used to solve classification & regression problems via random forest. It is an ensemble of decision trees

The most important parameter of the RandomForestRegressor class is the n_estimators parameter. This parameter defines the number of trees in the random forest.

```
model = RandomForestRegressor(n_jobs=-1)
# Try different numbers of n_estimators - (takes some time)
estimators = np.arange(10, 200, 10)
scores = []
for n in estimators:
    model.set_params(n_estimators=n)
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))
plt.figure(figsize=(7, 5))
plt.title("Effect of Estimators")
plt.title("Effect of Estimators")
plt.xlabel("no. estimator")
plt.ylabel("score")
plt.plot(estimators, scores)
results = list(zip(estimators,scores))
results
```

```
[(10, 0.9306207617734934),
 (20, 0.9344443409900342),
 (30, 0.9352600832539933),
 (40, 0.936552770850468),
 (50, 0.9375737700314158),
 (60, 0.937220507829467),
 (70, 0.9371523826008019),
 (80, 0.9373652546083545),
 (90, 0.9380758441904525),
 (100, 0.9383940594549373),
 (110, 0.9379428712705331),
 (120, 0.9380608129998546),
 (130, 0.9382270177851928),
 (140, 0.937920075784905),
 (150, 0.9376687792081319),
 (160, 0.9377822793292183),
 (170, 0.9382156507199576),
 (180, 0.9384387731661884),
 (190, 0.9380963420155776)]
```



Performance Metrics:

- Mean Absolute Error
- Mean Squared Error
- Root Mean Square Error

```
predictions = model.predict(X_test)
'Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions)
```

```
('Mean Absolute Error:', 0.24211342008156939)
```

```
'Mean Squared Error:', metrics.mean_squared_error(y_test, predictions)
```

```
('Mean Squared Error:', 0.1615235430487269)
```

```
'Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predictions))
```

```
('Root Mean Squared Error:', 0.40189991670654385)
```