FLIP ROBO

NAME OF THE PROJECT

Ratings Prediction Project

Submitted by:

Ram Kumar

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Khushboo Garg .

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thanks and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Ram kumar

.

INTRODUCTION

- We have a client who has a website where people write different reviews for technical products. Now they are adding a new feature to their website i.e. The reviewer will have to add stars(rating) as well with the review

- The rating is out 5 stars and it only has 5 options available 1 star, 2 stars, 3 stars, 4 stars, 5 stars. Now they want to predict ratings for the reviews which were written in the past and they don't have a rating. So, we have to build an application which can predict the rating by seeing the review.

Analytical Problem Framing

Import library and load the dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split, cross_val_score

import random
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('googleplaystore.csv')
```

```
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
```

```
df.head().style.background_gradient()
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.100000	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.900000	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide Apps	ART_AND_DESIGN	4.700000	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.500000	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.300000	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

```python
# Cleaning Categories into integers
CategoryString = df["Category"]
categoryVal = df["Category"].unique()
categoryValCount = len(categoryVal)
category_dict = {}
for i in range(0,categoryValCount):
    category_dict[categoryVal[i]] = i
df["Category_c"] = df["Category"].map(category_dict).astype(int)
```

```python
#scaling and cleaning size of installation
def change_size(size):
    if 'M' in size:
        x = size[:-1]
        x = float(x)*1000000
        return(x)
    elif 'k' == size[-1:]:
        x = size[:-1]
        x = float(x)*1000
        return(x)
    else:
        return None

df["Size"] = df["Size"].map(change_size)

#filling Size which had NA
df.Size.fillna(method = 'ffill', inplace = True)
```

```python
#Cleaning no of installs classification
df['Installs'] = [int(i[:-1].replace(',','')) for i in df['Installs']]
```

- Display all column name of dataset.

```python
#Cleaning no of installs classification
df['Installs'] = [int(i[:-1].replace(',','')) for i in df['Installs']]
```

```python
#Converting Type classification into binary
def type_cat(types):
    if types == 'Free':
        return 0
    else:
        return 1

df['Type'] = df['Type'].map(type_cat)
```

```python
#Cleaning of content rating classification
RatingL = df['Content Rating'].unique()
RatingDict = {}
for i in range(len(RatingL)):
    RatingDict[RatingL[i]] = i
df['Content Rating'] = df['Content Rating'].map(RatingDict).astype(int)
```

```python
#dropping of unrelated and unnecessary items
df.drop(labels = ['Last Updated','Current Ver','Android Ver','App'], axis = 1, inplace = True)
```

```python
#Cleaning of genres
GenresL = df.Genres.unique()
GenresDict = {}
for i in range(len(GenresL)):
    GenresDict[GenresL[i]] = i
df['Genres_c'] = df['Genres'].map(GenresDict).astype(int)
```

```
#Cleaning prices
def price_clean(price):
    if price == '0':
        return 0
    else:
        price = price[1:]
        price = float(price)
        return price

df['Price'] = df['Price'].map(price_clean).astype(float)
```

```
# convert reviews to numeric
df['Reviews'] = df['Reviews'].astype(int)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9360 entries, 0 to 10840
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Category        9360 non-null   object 
 1   Rating          9360 non-null   float64
 2   Reviews         9360 non-null   int32  
 3   Size            9360 non-null   float64
 4   Installs        9360 non-null   int64  
 5   Type            9360 non-null   int64  
 6   Price           9360 non-null   float64
 7   Content Rating  9360 non-null   int32  
 8   Genres          9360 non-null   object 
 9   Category_c      9360 non-null   int32  
 10  Genres_c        9360 non-null   int32  
dtypes: float64(3), int32(4), int64(2), object(2)
memory usage: 731.2+ KB
```

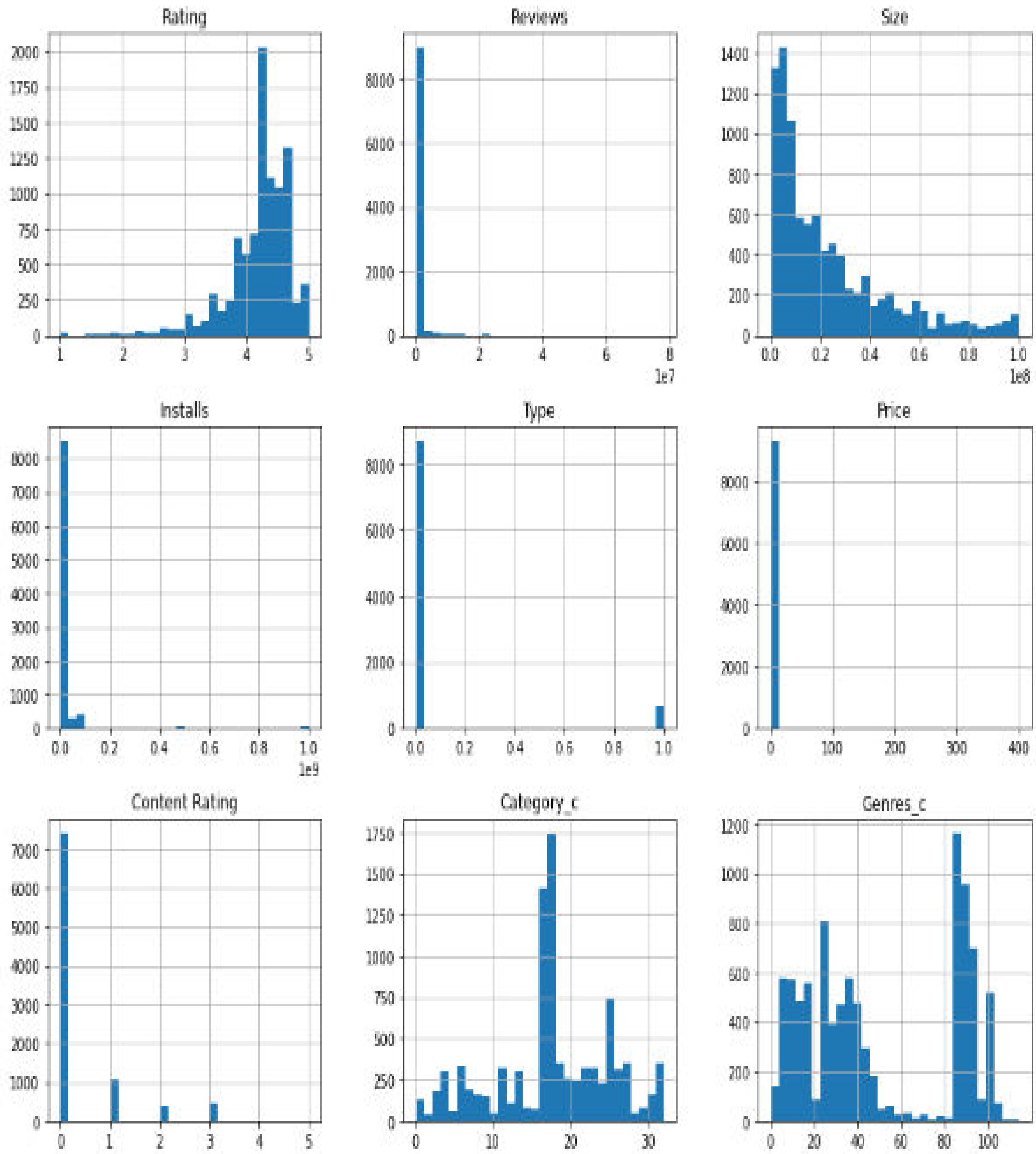
- Display statistical summary.

```
df.head().style.background_gradient()
```

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Category_c	Genres_c
0	ART_AND_DESIGN	4.100000	159	19000000.000000	10000	0	0.000000	0	Art & Design	0	0
1	ART_AND_DESIGN	3.900000	967	14000000.000000	500000	0	0.000000	0	Art & Design;Pretend Play	0	1
2	ART_AND_DESIGN	4.700000	87510	8700000.000000	5000000	0	0.000000	0	Art & Design	0	0
3	ART_AND_DESIGN	4.500000	215644	25000000.000000	50000000	0	0.000000	1	Art & Design	0	0
4	ART_AND_DESIGN	4.300000	967	2800000.000000	100000	0	0.000000	0	Art & Design;Creativity	0	2

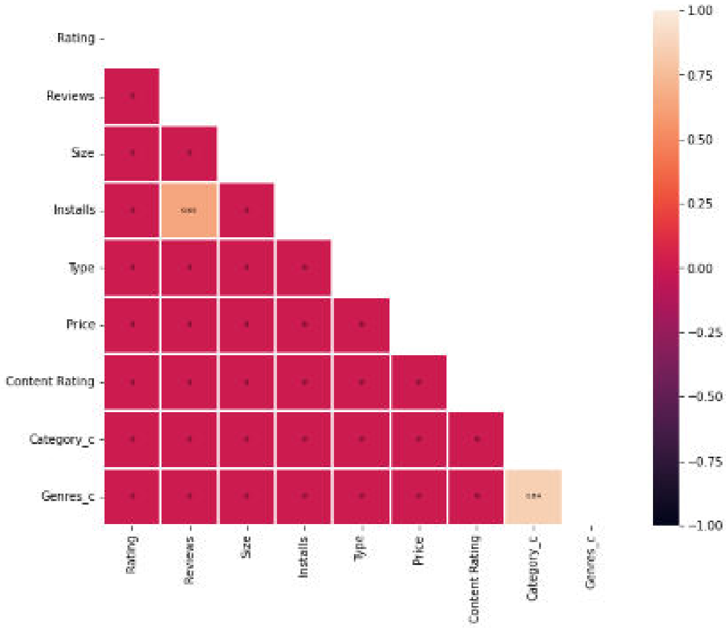
- Display histplot of all columns.

```
df.hist(figsize=(15, 12), bins=30);
```



- Display correlation of columns using heatmap.

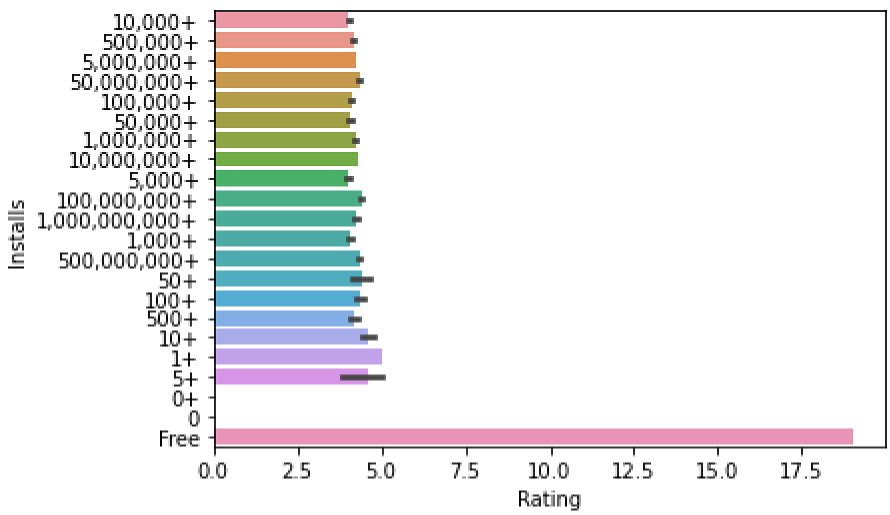
```python
pd.options.display.float_format = "{:,.2f}".format
corr_mat = df.corr('pearson')
# replace very weak correlation
corr_mat[(corr_mat < 0.3) & (corr_mat > -0.3)] = 0
# define triangular mask for better visibility
mask = np.triu(np.ones_like(corr_mat, dtype=bool))
plt.figure(figsize=(10, 8))
sns.heatmap(corr_mat, mask=mask, vmax=1.0, vmin=-1.0, square=True, annot=True, annot_kws={"size": 6, "color": "black"}, linewidt
```



- Display barplot of all columns.

```python
# Checking the months and price average by barplot:
sns.barplot(x='Rating',y='Installs',data=df)
```

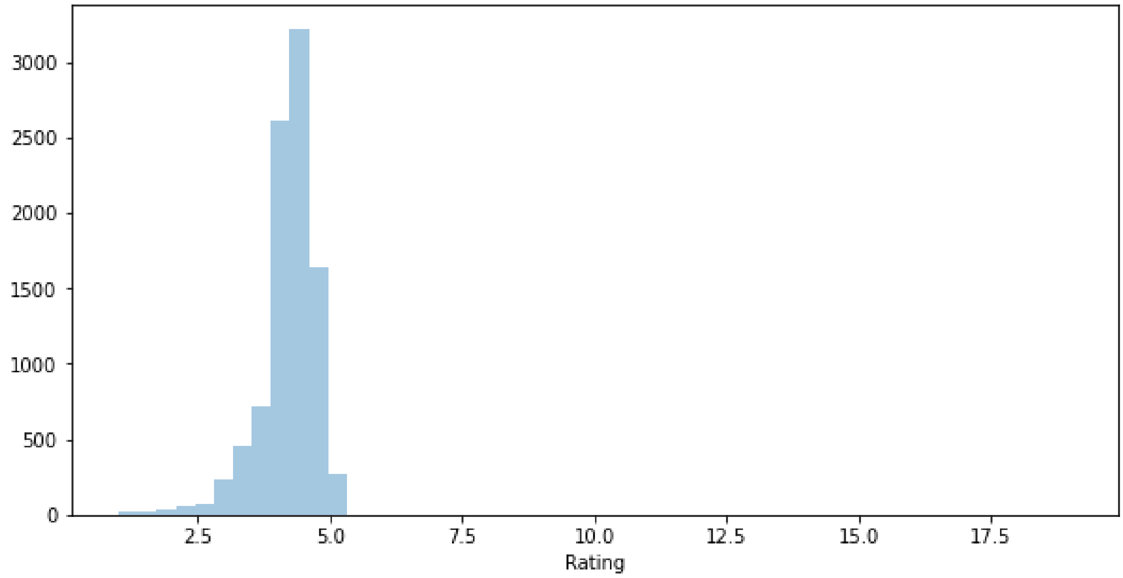
```
<AxesSubplot:xlabel='Rating', ylabel='Installs'>
```



- Display barplot of all columns

```
: # Checking the AveragePrice average by subplot:
plt.subplots(figsize=(10, 5))
sns.distplot(a=df.Rating, kde=False)
plt.xlabel('Rating')
plt.show
```

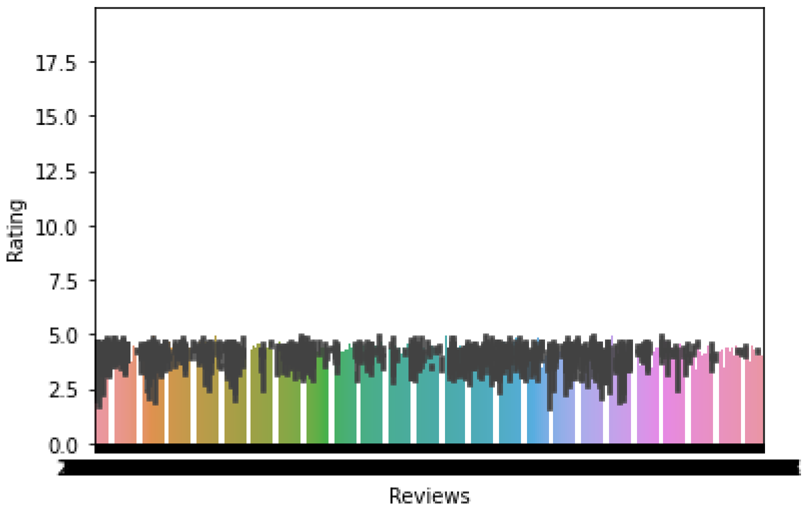
```
: <function matplotlib.pyplot.show(close=None, block=None)>
```



- Display barplot of all columns :

```
# Checking the year and price average by barplot :
sns.barplot(x='Reviews',y='Rating',data=df)
plt.subplots(figsize=(10, 5))
```

```
(<Figure size 720x360 with 1 Axes>, <AxesSubplot:>)
```



Model/s Development and Evaluation

- **Feature engeenering:**

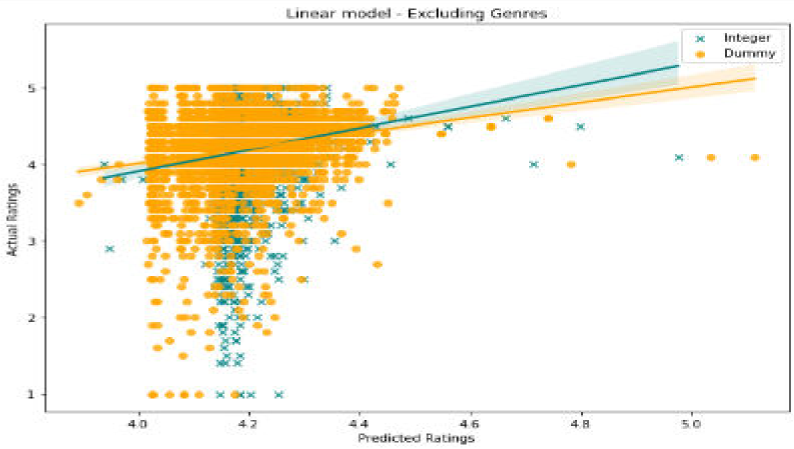
```
# Let's use 3 different regression models with two different techniques on treating the categorical variable
#for evaluation of error term and
def Evaluationmatrix(y_true, y_predict):
    print ('Mean Squared Error: '+ str(metrics.mean_squared_error(y_true,y_predict)))
    print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(y_true,y_predict)))
    print ('Mean squared Log Error: '+ str(metrics.mean_squared_log_error(y_true,y_predict)))
```

```
# regression model (without the genre label)
# to add into results_index for evaluation of error term
def Evaluationmatrix_dict(y_true, y_predict, name = 'Linear - Integer'):
    dict_matrix = {}
    dict_matrix['Series Name'] = name
    dict_matrix['Mean Squared Error'] = metrics.mean_squared_error(y_true,y_predict)
    dict_matrix['Mean Absolute Error'] = metrics.mean_absolute_error(y_true,y_predict)
    dict_matrix['Mean Squared Log Error'] = metrics.mean_squared_log_error(y_true,y_predict)
    return dict_matrix
```

```
#excluding Genre label
from sklearn.linear_model import LinearRegression

#Integer encoding
X = df.drop(labels = ['Category','Rating','Genres','Genres_c'],axis = 1)
y = df.Rating
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
model = LinearRegression()
model.fit(X_train,y_train)
Results = model.predict(X_test)

#Creation of results dataframe and addition of first entry
resultsdf = pd.DataFrame()
resultsdf = resultsdf.from_dict(Evaluationmatrix_dict(y_test,Results),orient = 'index')
resultsdf = resultsdf.transpose()

#dummy encoding
X_d = df2.drop(labels = ['Rating','Genres','Category_c','Genres_c'],axis = 1)
y_d = df2.Rating
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_d, y_d, test_size=0.30)
model_d = LinearRegression()
model_d.fit(X_train_d,y_train_d)
Results_d = model_d.predict(X_test_d)

#adding results into results dataframe
resultsdf = resultsdf.append(Evaluationmatrix_dict(y_test_d,Results_d, name = 'Linear - Dummy'),ignore_index = True)
```

- **Testing of Identified Approaches (Algorithms):**

```
plt.figure(figsize=(10,7))
sns.regplot(Results,y_test,color='teal', label = 'Integer', marker = 'x')
sns.regplot(Results_d,y_test_d,color='orange',label = 'Dummy')
plt.legend()
plt.title('Linear model - Excluding Genres')
plt.xlabel('Predicted Ratings')
plt.ylabel('Actual Ratings')
plt.show()
```

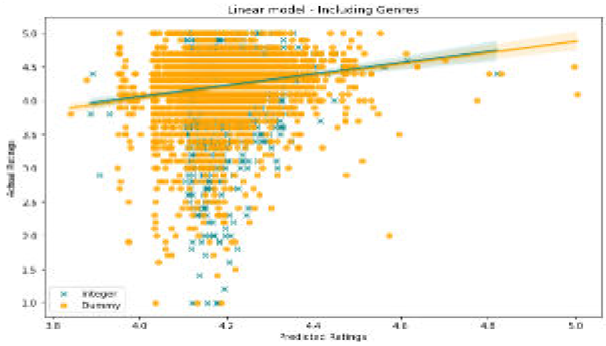



```
print ('Actual mean of population:' + str(y.mean()))
print ('Integer encoding(mean) :' + str(Results.mean()))
print ('Dummy encoding(mean) :'+ str(Results_d.mean()))
print ('Integer encoding(std) :' + str(Results.std()))
print ('Dummy encoding(std) :'+ str(Results_d.std()))
```

```
Actual mean of population:4.191837606837612
Integer encoding(mean) :4.196384106959842
Dummy encoding(mean) :4.18780111385666
Integer encoding(std) :0.05327215820341826
Dummy encoding(std) :0.10103014978518421
```

```
#Including genre label

#Integer encoding
X = df.drop(labels = ['Category','Rating','Genres'],axis = 1)
y = df.Rating
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
model = LinearRegression()
model.fit(X_train,y_train)
Results = model.predict(X_test)

resultsdf = resultsdf.append(Evaluationmatrix_dict(y_test,Results, name = 'Linear(inc Genre) - Integer'),ignore_index = True)

#dummy encoding

X_d = df2.drop(labels = ['Rating','Genres','Category_c'],axis = 1)
y_d = df2.Rating
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_d, y_d, test_size=0.30)
model_d = LinearRegression()
model_d.fit(X_train_d,y_train_d)
Results_d = model_d.predict(X_test_d)

resultsdf = resultsdf.append(Evaluationmatrix_dict(y_test_d,Results_d, name = 'Linear(inc Genre) - Dummy'),ignore_index = True)
```

```
plt.figure(figsize=(10,7))
sns.regplot(Results,y_test,color='teal', label = 'Integer', marker = 'x')
sns.regplot(Results_d,y_test_d,color='orange',label = 'Dummy')
plt.legend()
plt.title('Linear model - Including Genres')
plt.xlabel('Predicted Ratings')
plt.ylabel('Actual Ratings')
plt.show()
```



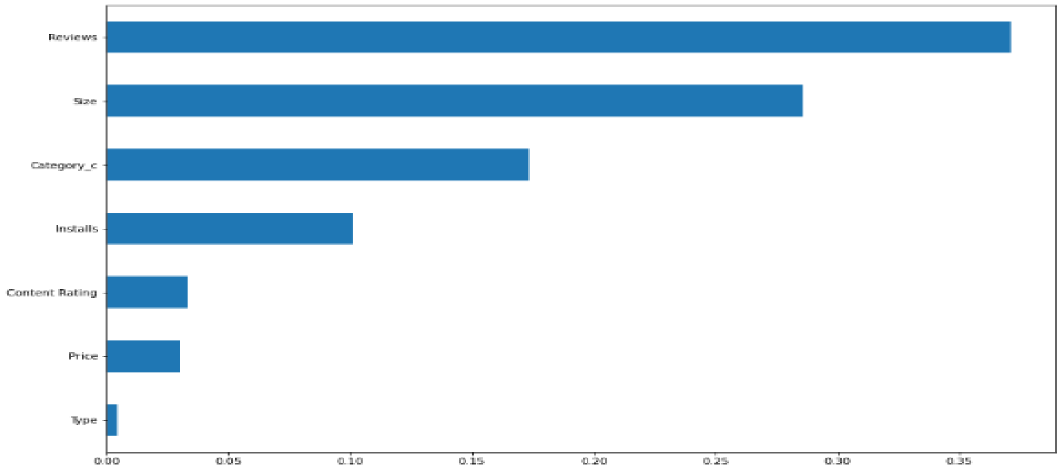

```
print ('Integer encoding(mean) :' + str(Results.mean()))
print ('Dummy encoding(mean) :'+ str(Results_d.mean()))
print ('Integer encoding(std) :' + str(Results.std()))
print ('Dummy encoding(std) :'+ str(Results_d.std()))
```

```
Integer encoding(mean) :4.1841453376485305
Dummy encoding(mean) :4.195244702338427
Integer encoding(std) :0.0616385516634759
```

- **Run and Evaluate selected models**

```
#for integer
Feat_impt = {}
for col,feat in zip(X.columns,model3.feature_importances_):
    Feat_impt[col] = feat

Feat_impt_df = pd.DataFrame.from_dict(Feat_impt,orient = 'index')
Feat_impt_df.sort_values(by = 0, inplace = True)
Feat_impt_df.rename(index = str, columns = {0:'Pct'},inplace = True)

plt.figure(figsize= (10,8))
Feat_impt_df.plot(kind = 'barh',figsize= (14,10),legend = False)
plt.show()
```

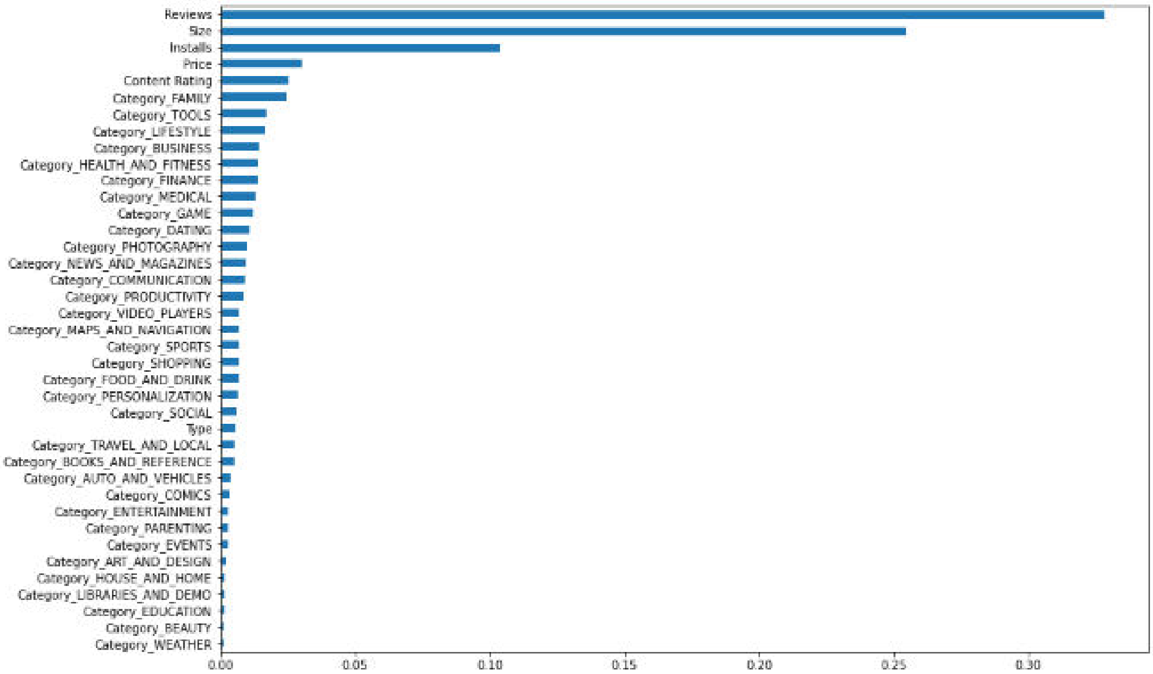
```
<Figure size 720x576 with 0 Axes>
```



- **higher overall predicted mean：**

```
#for dummy
Feat_impt_d = {}
for col,feat in zip(X_d.columns,model3_d.feature_importances_):
    Feat_impt_d[col] = feat

Feat_impt_df_d = pd.DataFrame.from_dict(Feat_impt_d,orient = 'index')
Feat_impt_df_d.sort_values(by = 0, inplace = True)
Feat_impt_df_d.rename(index = str, columns = {0:'Pct'},inplace = True)

plt.figure(figsize= (12,10))
Feat_impt_df_d.plot(kind = 'barh',figsize= (14,10),legend = False)
plt.show()
```

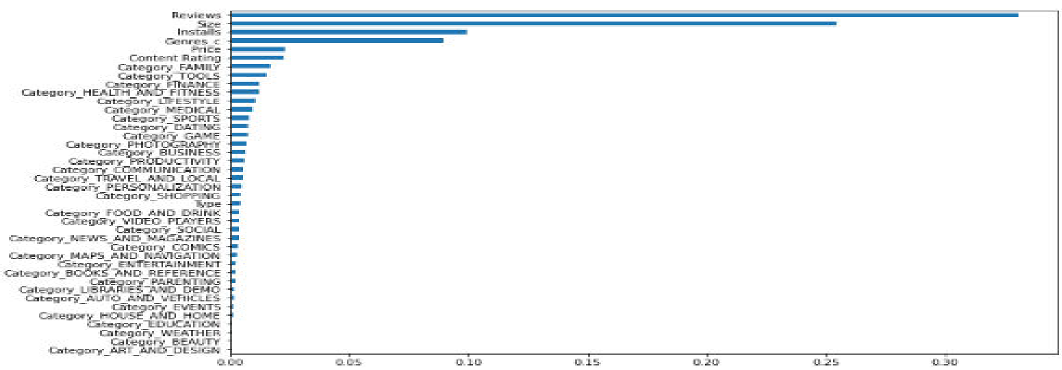
```
<Figure size 864x720 with 0 Axes>
```



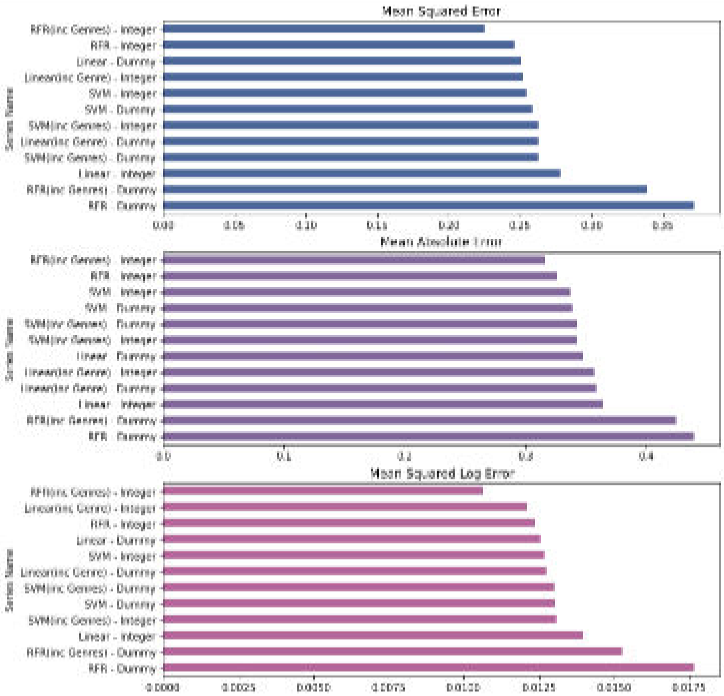
- the ratings, the top 4 being reviews, size, category, and number of installs**:**

```
#for dummy
Feat_impt_d = {}
for col,feat in zip(X_d.columns,model3a_d.feature_importances_):
    Feat_impt_d[col] = feat

Feat_impt_df_d = pd.DataFrame.from_dict(Feat_impt_d,orient = 'index')
Feat_impt_df_d.sort_values(by = 0, inplace = True)
Feat_impt_df_d.rename(index = str, columns = {0:'Pct'},inplace = True)

plt.figure(figsize= (14,10))
Feat_impt_df_d.plot(kind = 'barh',figsize= (12,8),legend = False)
plt.show()
```

```
<Figure size 1008x720 with 0 Axes>
```



```
resultsdf.set_index('Series Name', inplace = True)

plt.figure(figsize = (10,12))
plt.subplot(3,1,1)
resultsdf['Mean Squared Error'].sort_values(ascending = False).plot(kind = 'barh',color=(0.3, 0.4, 0.6, 1), title = 'Mean Square
plt.subplot(3,1,2)
resultsdf['Mean Absolute Error'].sort_values(ascending = False).plot(kind = 'barh',color=(0.5, 0.4, 0.6, 1), title = 'Mean Absol
plt.subplot(3,1,3)
resultsdf['Mean Squared Log Error'].sort_values(ascending = False).plot(kind = 'barh',color=(0.7, 0.4, 0.6, 1), title = 'Mean Sq
plt.show()
```



- Hardware and Software Requirements and Tools Used

➢ **Language :-** Python

➢ **Tool:-** Jupyter

➢ **OS:-** Windows 10

➢ **RAM:-** 8gb

CONCLUSION

It is not easy to conclude which model has the best predictive accuracy and lowest error term. Using this round of data as a basis, the dummy encoded SVM model including genres has the lowest overall error rates, followed by the integer encoded RFR model including genes. Yet, all models seem to be very close in terms of it's error term, so this result is likely to change.

What is very surprising to me is how the RFR dummy model has such a significantly more error term compared to all the other models, even though on the surface it seemed to perform very similarly to the RFR integer model.