MachineLearningProject

July 19, 2023

0.1 Non-Neural Machine Learning Home Assignment

0.1.1 Ashley Honeycutt | Group B - Linear Regression

In this notebook, I use machine learning to analyze a cursor flow dataset of user information in order to predict the binned income range of a user. I train a linear regression and random forest regression model to predict income from the independent variables, and then I evaluate and compare the output of the two models.

First, import the necessary libraries and packages

[1246]:
```
import pandas as pd #Python data analysis library
import numpy as np #Library for working with arrays
from sklearn.linear_model import LinearRegression #scikit-learn (sklearn) is a␣
↪machine learning library
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.model_selection import train_test_split #this package helps divide␣
↪the data for training and testing our models
import matplotlib.pyplot as plt #for plotting/data visualization
```

Read in the dataset csv files and assign them to pandas dataframes

[1247]:
```
participants_df = pd.read_csv('participants.tsv',sep='\t')

groundtruth_df = pd.read_csv('groundtruth.tsv', sep='\t')
```

0.2 Data Preparation

We want to merge the two dataframes, so we need to make sure they have a feature in common. In this case, we check if the dataframes share the same user ids. First, we check the length of the list of unique values in the user id column of the groundtruth_df dataframe. The output tells us there are 2909 unique user ids.

[1248]:
```
len(groundtruth_df.user_id.unique())
```

[1248]: 2909

Then we compare the length of unique user ids in both dataframes. "True" tells us the lengths are equal.

[1249]:
```
len(groundtruth_df.user_id.unique()) == len(participants_df.user_id.unique())
```

[1249]: True

Then we check if the user ids in both dataframes match. We use the isin() function which returns a boolean and value_counts() which gives us a count of values corresponding to the output of isin(). The output tells us that all 2909 user ids are the same in both dataframes.

[1250]:
```
groundtruth_df['user_id'].isin(participants_df['user_id']).value_counts()
```

[1250]:
```
True    2909
Name: user_id, dtype: int64
```

Next, we merge the participants dataframe with the user_id, ad_clicked, and attention columns of the groundtruth dataframe. We merge them on the user id column.

[1251]:
```
df = pd.merge(participants_df, groundtruth_df[['user_id', 'ad_clicked',␣
↪'attention']], how='right', on='user_id')
df.head()
```

[1251]:
```
                       user_id country education age income gender ad_position  \
0  5npsk114ba8hfbj4jr3lt8jhf5     PHL         3   3      1   male    top-left
1  5o9js8slc8rg2a8mo5p3r93qm0     VEN         3   1      1   male   top-right
2  pi17qjfqmnhpsiahbumcsdq0r6     VEN         2   3      1   male    top-left
3  3rptg9g7183imkbdsu2miignv7     VEN         3   2      1   male   top-right
4  049onniafv6fe4e6q42k6nq1n2     VEN         3   5      1   male    top-left

  ad_type              ad_category        serp_id          query  \
0      dd  Computers & Electronics        tablets        tablets
1      dd      Shop - Luxury Goods  casio-watches  casio watches
2  native      Shop - Luxury Goods   chivas-regal   chivas regal
3      dd      Shop - Luxury Goods   chivas-regal   chivas regal
4  native         Autos & Vehicles   audi-r8-used    audi r8 used

           log_id  ad_clicked  attention
0  20181002033126           0          4
1  20181001211223           1          5
2  20181001170952           0          4
3  20181001140754           0          1
4  20181001132434           0          1
```

Next, we handle the cells with undefined values and make sure numeric columns have numeric datatypes.

We use the replace() function to replace all string 'na' values with a numeric nan value. Then we check the sum of na values for each feature of our dataframe.

[1252]:
```
df.replace({'na':np.nan}, inplace = True)
df.isna().sum()
```

[1252]:
```
user_id          0
country          2
education       47
age             20
income         202
gender          14
ad_position      0
ad_type          0
ad_category      0
serp_id          0
query            0
log_id           0
ad_clicked       0
attention        0
dtype: int64
```

We can see in the output above that the country column contains two na values. To decide how to handle them, we explore the value counts of the country column and find there are more USA values than non-USA values. Based on this, we will assign the na values to the more prevalent value, USA.

[1253]:
```
df['country'].value_counts()
```

[1253]:
```
USA    1768
VEN     368
GBR     209
CAN      77
EGY      38
       …
BOL       1
DNK       1
MYS       1
KWT       1
HUN       1
Name: country, Length: 68, dtype: int64
```

We use the fillna function to fill na values with 'USA'.

[1254]:
```
df.fillna({'country':'USA'})
```

[1254]:
```
                           user_id country  education  age  income   gender  \
0     5npsk114ba8hfbj4jr3lt8jhf5     PHL          3    3       1    male
1     5o9js8slc8rg2a8mo5p3r93qm0     VEN          3    1       1    male
2     pi17qjfqmnhpsiahbumcsdq0r6     VEN          2    3       1    male
3     3rptg9g7183imkbdsu2miignv7     VEN          3    2       1    male
4     049onniafv6fe4e6q42k6nq1n2     VEN          3    5       1    male
…                              …       …          …   ..     …       …
2904  2jbfmshmhsji4smrgph018k410     USA          2    6       2    male
```

```
2905  p1tt6ehhpcihelra9j558acgv7    USA        1   5       5  female
2906  tl1hfafsot8s5qud19bkij68f7    USA        4   2       4  female
2907  lvmrfennsqgn49ndepfnl68ok4    USA        1   8       3  female
2908  bsqgffkob06hgsb6r0csjk4gv6    USA        5   2       4    male

     ad_position ad_type                ad_category         serp_id  \
0       top-left      dd        Computers & Electronics         tablets
1      top-right      dd           Shop - Luxury Goods  casio-watches
2       top-left  native           Shop - Luxury Goods   chivas-regal
3      top-right      dd           Shop - Luxury Goods   chivas-regal
4       top-left  native              Autos & Vehicles   audi-r8-used
…            …       …                            …             …
2904    top-left      dd        Computers & Electronics         macbook
2905    top-left  native      Shop - Event Ticket Sales   cubs-tickets
2906   top-right      dd  Shop - Gifts & Special Event      flowers_2
2907   top-right      dd        Computers & Electronics      laptop-bag
2908   top-right      dd           Shop - Luxury Goods  famous-grouse

               query          log_id  ad_clicked  attention
0            tablets  20181002033126           0          4
1      casio watches  20181001211223           1          5
2       chivas regal  20181001170952           0          4
3       chivas regal  20181001140754           0          1
4       audi r8 used  20181001132434           0          1
…                …               …         …          …
2904         macbook  20170203232414           0          2
2905    cubs tickets  20170131193748           0          4
2906         flowers  20170106152837           0          2
2907      laptop bag  20170102171535           0          4
2908   famous grouse  20161227191740           0          4

[2909 rows x 14 columns]
```

We want to reassign all the countries which are not the USA to "non-USA". We use the loc method to access the values not equal to USA and replace them with 'non-USA'

[1255]:
```
df.loc[df["country"] != "USA", "country"] = "non-USA"
df['country'].value_counts()
```

[1255]:
```
USA        1768
non-USA    1141
Name: country, dtype: int64
```

[1256]:
```
df.dtypes
```

[1256]:
```
user_id          object
country          object
education        object
```

```
age              object
income           object
gender           object
ad_position      object
ad_type          object
ad_category      object
serp_id          object
query            object
log_id            int64
ad_clicked        int64
attention         int64
dtype: object
```

Now we need to handle the na values in the education, age, and income columns.

Viewing our data, we see that education, age, and income variables have been binned in different categories represented by ordinal numbers. We first change the datatype of those columns from object to Int64 (numeric) so that we can calculate a median value for each column. Then, we assign the resulting median values to the na values.

[1257]:
```
numcols = ['education', 'age', 'income']
df[numcols] = df[numcols].astype('Int64')
df[numcols]= df[numcols].fillna(df[numcols].median())
df[numcols].dtypes
```

[1257]:
```
education    Int64
age          Int64
income       Int64
dtype: object
```

Next we handle the na values for the gender column. In this case, there should be two nominal categories: male and female. We find the mode (most frequently occuring gender) and use fillna() to replace the na values with the mode.

[1258]:
```
replace_mode = df['gender'].mode()[0]
df.fillna({'gender': replace_mode}, inplace=True)
df['gender'].value_counts()
```

[1258]:
```
male      1727
female    1182
Name: gender, dtype: int64
```

Double-checking for na values in our dataframe. All sums are zero so all have been handled.

[1259]:
```
df.isna().sum()
```

[1259]:
```
user_id        0
country        0
education      0
```

```
age            0
income         0
gender         0
ad_position    0
ad_type        0
ad_category    0
serp_id        0
query          0
log_id         0
ad_clicked     0
attention      0
dtype: int64
```

[1260]:
```
df.columns
```

[1260]:
```
Index(['user_id', 'country', 'education', 'age', 'income', 'gender',
       'ad_position', 'ad_type', 'ad_category', 'serp_id', 'query', 'log_id',
       'ad_clicked', 'attention'],
      dtype='object')
```

Drop the columns we don't want to input to our models.

[1261]:
```
df = df.drop(['user_id','serp_id','query','log_id'], axis=1)
df.head()
```

[1261]:
```
   country  education  age  income gender ad_position ad_type  \
0  non-USA          3    3       1   male    top-left      dd
1  non-USA          3    1       1   male   top-right      dd
2  non-USA          2    3       1   male    top-left  native
3  non-USA          3    2       1   male   top-right      dd
4  non-USA          3    5       1   male    top-left  native

              ad_category  ad_clicked  attention
0  Computers & Electronics           0          4
1      Shop - Luxury Goods           1          5
2      Shop - Luxury Goods           0          4
3      Shop - Luxury Goods           0          1
4         Autos & Vehicles           0          1
```

0.2.1 One-hot encoding

Next, we need to handle the categorical features in our dataset. Our categorical features are: country, gender, ad_position, ad_type, and ad_category.

Categorical variables cannot be input to a regression model, so we must convert them into numerical variables. We use the pandas 'get_dummies' method which performs the function of one-hot encoding. This method creates a new column for each sub-group of a categorical feature and assigns a 1 or 0 to represent the absence or presence of that sub-category for each record in the dataframe.

In the process of one-hot encoding, we need to account for the possibility of multicollinearity, which arises when multiple independent variables are perfectly correlated. Ideally, the independent variables should be uncorrelated since the purpose of the model is to estimate the independent effects of the X variables on Y (income). In this code, we break perfect multicollinearity by dropping the first column of each dummy variable.

[1262]:
```python
df = pd.get_dummies(df, drop_first=True)
df
```

[1262]:
```
      education  age  income  ad_clicked  attention  country_non-USA  \
0             3    3       1           0          4                1
1             3    1       1           1          5                1
2             2    3       1           0          4                1
3             3    2       1           0          1                1
4             3    5       1           0          1                1
…           …   …    …          …         …                …
2904          2    6       2           0          2                0
2905          1    5       5           0          4                0
2906          4    2       4           0          2                0
2907          1    8       3           0          4                0
2908          5    2       4           0          4                0

      gender_male  ad_position_top-right  ad_type_native  \
0               1                      0               0
1               1                      1               0
2               1                      0               1
3               1                      1               0
4               1                      0               1
…             …                    …               …
2904            1                      0               0
2905            0                      0               1
2906            0                      1               0
2907            0                      1               0
2908            1                      1               0

      ad_category_Computers & Electronics  …  ad_category_Real Estate  \
0                                       1  …                        0
1                                       0  …                        0
2                                       0  …                        0
3                                       0  …                        0
4                                       0  …                        0
…                                     …  …                      …
2904                                    1  …                        0
2905                                    0  …                        0
2906                                    0  …                        0
2907                                    1  …                        0
2908                                    0  …                        0
```

```
      ad_category_Shop - Apparel  ad_category_Shop - Event Ticket Sales  \
0                              0                                      0
1                              0                                      0
2                              0                                      0
3                              0                                      0
4                              0                                      0
…                            …                                    …
2904                           0                                      0
2905                           0                                      1
2906                           0                                      0
2907                           0                                      0
2908                           0                                      0

      ad_category_Shop - Gifts & Special Event  \
0                                            0
1                                            0
2                                            0
3                                            0
4                                            0
…                                          …
2904                                         0
2905                                         0
2906                                         1
2907                                         0
2908                                         0

      ad_category_Shop - Luxury Goods  \
0                                   0
1                                   1
2                                   1
3                                   1
4                                   0
…                                 …
2904                                0
2905                                0
2906                                0
2907                                0
2908                                1

      ad_category_Shop - Photo & Video Services  \
0                                             0
1                                             0
2                                             0
3                                             0
4                                             0
…                                           …
```

```
2904                                        0
2905                                        0
2906                                        0
2907                                        0
2908                                        0

      ad_category_Shop - Sporting Goods  ad_category_Shop - Toys  \
0                                     0                        0
1                                     0                        0
2                                     0                        0
3                                     0                        0
4                                     0                        0
…                                   …                      …
2904                                  0                        0
2905                                  0                        0
2906                                  0                        0
2907                                  0                        0
2908                                  0                        0

      ad_category_Shop - Wholesalers & Liquidatr  ad_category_Travel
0                                              0                   0
1                                              0                   0
2                                              0                   0
3                                              0                   0
4                                              0                   0
…                                            …                 …
2904                                           0                   0
2905                                           0                   0
2906                                           0                   0
2907                                           0                   0
2908                                           0                   0

[2909 rows x 22 columns]
```

0.3 Modeling

0.3.1 Linear Regression Model

The first step of modeling is assigning our X and Y variables and creating a train-test-split. We assign all features except income to X, and assign income to y, our dependent variable. Then we create the train-test split, assigning 33% of the data to the test set.

[1263]:
```
X=df.drop('income',axis=1)

y=df['income']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33)
```

Here, we use the shape function to return tuples of our train and test sets. We can see that our training set contains ~66% of the data, and the test set contains ~33%. The X sets reflect 21 columns and the y sets reflect only one (income).

[1264]:
```python
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1949, 21)
(960, 21)
(1949,)
(960,)
```

Next, we instantiate an object of a linear regression model and we fit our training data to the model.

[1265]:
```python
LR_model = LinearRegression()

LR_model.fit(X_train,y_train)
```

[1265]:
```
LinearRegression()
```

Next, we create a dataframe displaying the coefficients and intercepts of each independent variable in our model. The closer a coefficient is to 1 (positive or negative), the greater its correlation to the dependent variable. The coefficient is interpreted as the change in the dependent variable for every unit change in the independent variable.
Examining this dataframe, the coefficient closest to |1| is country_non-USA, and it is negative, indicating a negative correlation with income.

[1266]:
```python
coef_df = pd.DataFrame(LR_model.coef_,X.columns, columns = ['Coeff'])
intercept_df = pd.DataFrame(LR_model.intercept_,X.columns,columns=['Intercept'])
C_I_df= pd.concat([coef_df, intercept_df],axis=1)
C_I_df
```

[1266]:

	Coeff	Intercept
education	0.279352	1.54368
age	0.108516	1.54368
ad_clicked	-0.185664	1.54368
attention	0.045216	1.54368
country_non-USA	-0.879050	1.54368
gender_male	0.075416	1.54368
ad_position_top-right	-0.066407	1.54368
ad_type_native	-0.032521	1.54368
ad_category_Computers & Electronics	0.114509	1.54368
ad_category_Food & Drink	1.027083	1.54368
ad_category_Games	-0.056505	1.54368
ad_category_Real Estate	-0.790946	1.54368
ad_category_Shop - Apparel	0.297008	1.54368
ad_category_Shop - Event Ticket Sales	0.090600	1.54368

```
ad_category_Shop - Gifts & Special Event   -0.309335    1.54368
ad_category_Shop - Luxury Goods            -0.096777    1.54368
ad_category_Shop - Photo & Video Services   0.144838    1.54368
ad_category_Shop - Sporting Goods           0.099413    1.54368
ad_category_Shop - Toys                     0.033867    1.54368
ad_category_Shop - Wholesalers & Liquidatr -0.001874    1.54368
ad_category_Travel                          0.204766    1.54368
```

Next, we use the predict function to generate predicted income values from our X_test array.

[1267]:
```
predictions = LR_model.predict(X_test)
```

Linear Regression Visualizations

Below we have visualized the real income (y_test values) vs the predicted income values in a scatterplot. If the test data matches the predicted values, the scatterplot should show the points closely fitted around a diagonal line. In this case, it does not appear that the values match well. This indicates that our linear regression model may not be a good predictor of income.

[1268]:
```
predictions = LR_model.predict(X_test)
plt.scatter(y_test,predictions,color='red')
plt.xlabel('real income (test values)')
plt.ylabel('predicted income')
plt.title('Linear Regression: Real Income vs. Predicted Income')
```

[1268]:
```
Text(0.5, 1.0, 'Linear Regression: Real Income vs. Predicted Income')
```

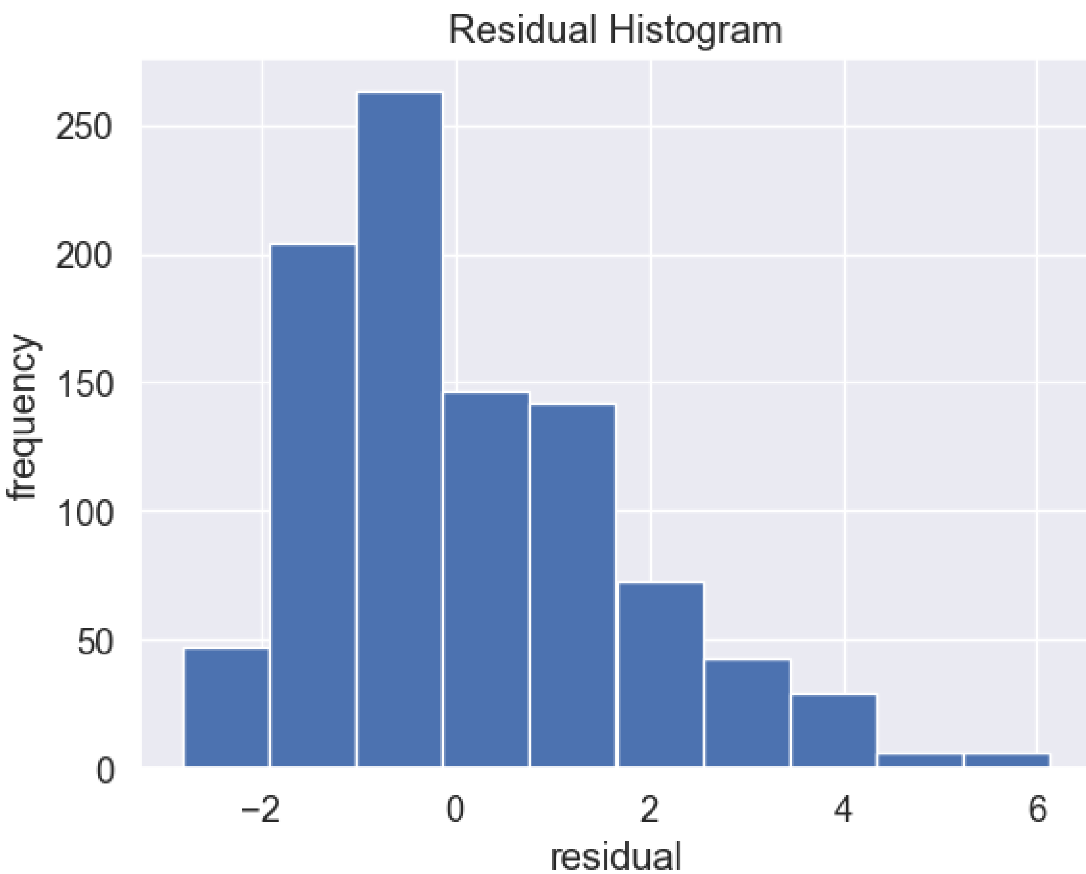
Next, we plot the residuals in a histogram. To calculate the residuals we subtract the predicted values from the actual/test values.

We look for a normal distribution of residuals to indicate a reliable linear regression model. Residuals should be centered on 0 and evenly spread on either side to indicate normal random error.

In our histogram we can see that the residuals are skewed right. This means that our model is likely systematically incorrect.

[1269]:
```
plt.hist(y_test-predictions)
plt.title('Residual Histogram')
plt.ylabel('frequency')
plt.xlabel('residual')
```

[1269]:
```
Text(0.5, 0, 'residual')
```



Linear Regression Metrics

[1270]:
```python
from sklearn import metrics
```

We use four different metrics to evaluate our linear regression model:

1. **Mean Absolute Error (MAE)** is the average of the absolute value of the distances between the actual datapoints and the predicted datapoints (or residuals). The lower the MAE, the better our data is fitted to the model and the more reliable its predictions. We can compare the MAE directly to the units of the y variable. In this case, the y variable is binned income and it is measured by unit of 1 from range 0 to 8.

2. **Mean Squared Error (MSE)** is the average of the squared residuals. By squaring instead of taking absolute value, MSE imposes a greater penalty on larger residuals than MAE. In our case, MSE is a larger number than MAE.

3. **Root Mean Squared Error (RMSE)** is the square root of MSE. Like MAE, we can compare it directly to the units of the y variable.

4. **Coefficient of Determination (R2)** is equal to the sum of squared residuals divided by the total sum of squares, all subtracted from 1. An R2 closer to $|1|$ is an indicator of stronger correlation. Our R2 is closer to 0 than to 1, indicating no-to-very weak positive correlation

between our actual and predicted values.

[1271]:
```
LR_MAE = metrics.mean_absolute_error(y_test,predictions)
LR_MSE = metrics.mean_squared_error(y_test,predictions)
LR_RMSE = np.sqrt(metrics.mean_squared_error(y_test,predictions))
LR_RSquared = metrics.r2_score(y_test,predictions)
print(f'Mean Absolute Error: {LR_MAE}')
print(f'Mean Squared Error: {LR_MSE}')
print(f'Root Mean Squared Error: {LR_RMSE}')
print(f'R Squared: {LR_RSquared}')
```

```
Mean Absolute Error: 1.2680396813168133
Mean Squared Error: 2.554294154462197
Root Mean Squared Error: 1.598215928609835
R Squared: 0.1638579658301258
```

0.3.2 Random Forest Regression Model

Random forest regression works by generating multiple decision trees on multiple random samples of features, and then averaging the output of all trees to help correct for errors that arise in individual trees.

We first instantiate an object of a random forest regression model and we fit our training data to the model.

[1272]:
```
RF_model = RFR(n_estimators=100) #n_estimators is the number of decision trees␣
 ↪run by the model
RF_model.fit(X_train,y_train)
```

[1272]:
```
RandomForestRegressor()
```

Next we use the predict function to generate predicted income values using our X_test data.

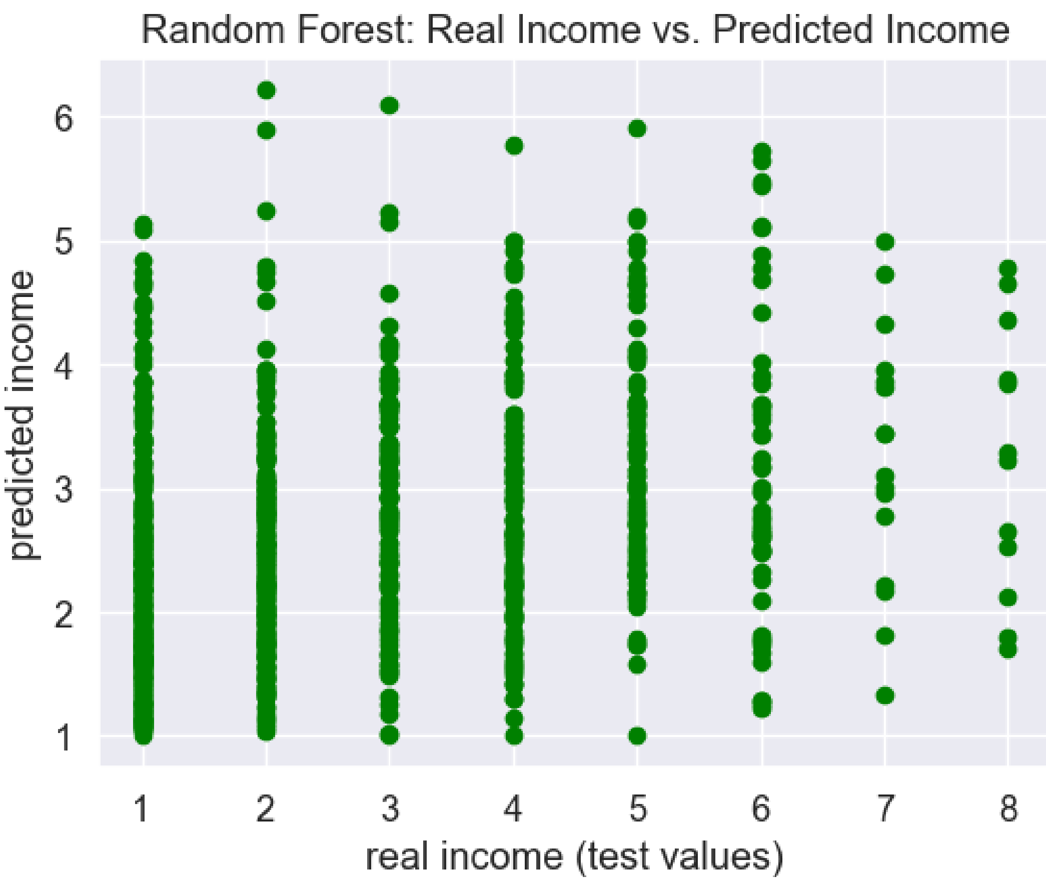
[1273]:
```
RF_predictions=RF_model.predict(X_test)
```

Random Forest Visualizations

Then we visualize the real income (y_test values) vs the predicted income values in a scatterplot. For this random forest regression model, it does not appear that the test and prediction values closely align, which can indicate that our model may not be a good predictor of income.

[1274]:
```
plt.scatter(y_test,RF_predictions, color='green')
plt.xlabel('real income (test values)')
plt.ylabel('predicted income')
plt.title('Random Forest: Real Income vs. Predicted Income')
```

[1274]:
```
Text(0.5, 1.0, 'Random Forest: Real Income vs. Predicted Income')
```

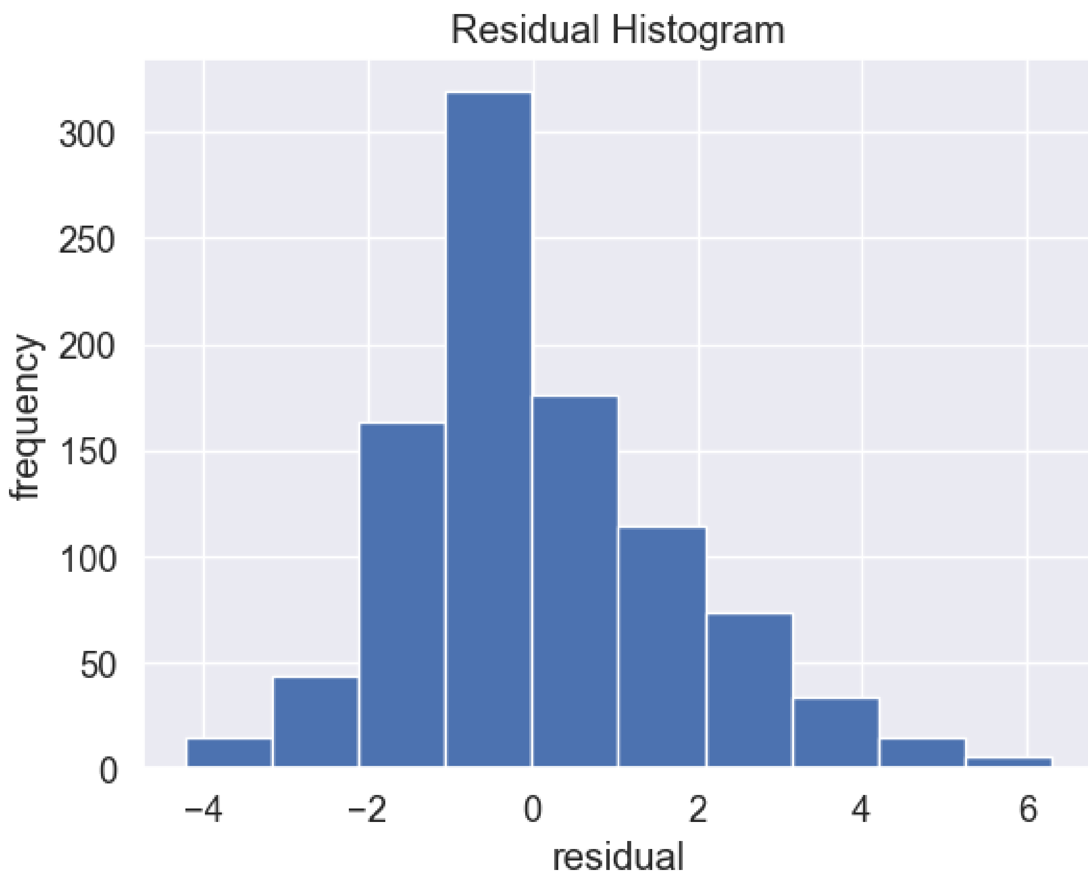


Next, we plot the residuals in a histogram. To calculate the residuals we subtract the predicted values from the actual/test values.

In our histogram we can see that the residuals are skewed slightly right, but overall it appears closer to a normal distribution than the histogram of linear regression residuals. This might indicate that our random forest model is a better predictor of income than our linear regression model.

[1275]:
```
plt.hist(y_test-RF_predictions)
plt.title('Residual Histogram')
plt.ylabel('frequency')
plt.xlabel('residual')
```

[1275]:
```
Text(0.5, 0, 'residual')
```



Random Forest Metrics

The output of our random forest metrics is very similar to the output of the linear regression metrics. Increasing the n_estimators parameter does not seem to decrease the level of error significantly, or increase the R2.

[1276]:
```
RF_MAE = metrics.mean_absolute_error(y_test,RF_predictions)
RF_MSE = metrics.mean_squared_error(y_test,RF_predictions)
RF_RMSE = np.sqrt(metrics.mean_squared_error(y_test,RF_predictions))
RF_RSquared = metrics.r2_score(y_test,predictions)

print(f'Mean Absolute Error: {RF_MAE}')
print(f'Mean Squared Error: {RF_MSE}')
print(f'Root Mean Squared Error: {RF_RMSE}')
print(f'R Squared: {RF_RSquared}')
```

```
Mean Absolute Error: 1.3096294887115199
Mean Squared Error: 2.8915734285247123
Root Mean Squared Error: 1.7004627101247214
R Squared: 0.1638579658301258
```

Feature Importances

Since we have many features, we can rank their importance to better understand their contribution to our random forest model. Importance is calculated by averaging the variance reduction for each feature in each decision tree. The features with highest average variance reduction are of the greatest importance. We use the feature_importances method which is part of sklearn, and visualize it in a dataframe and a bar graph. According to this function, age, education, and attention are of the highest importance to our model.

[1285]:
```
RF_model.feature_importances_

importance_df = pd.DataFrame({"Feature": X_train.columns,"Importance": RF_model.
 ↪feature_importances_})
importance_df.sort_values(by="Importance", ascending=False)
```

[1285]:

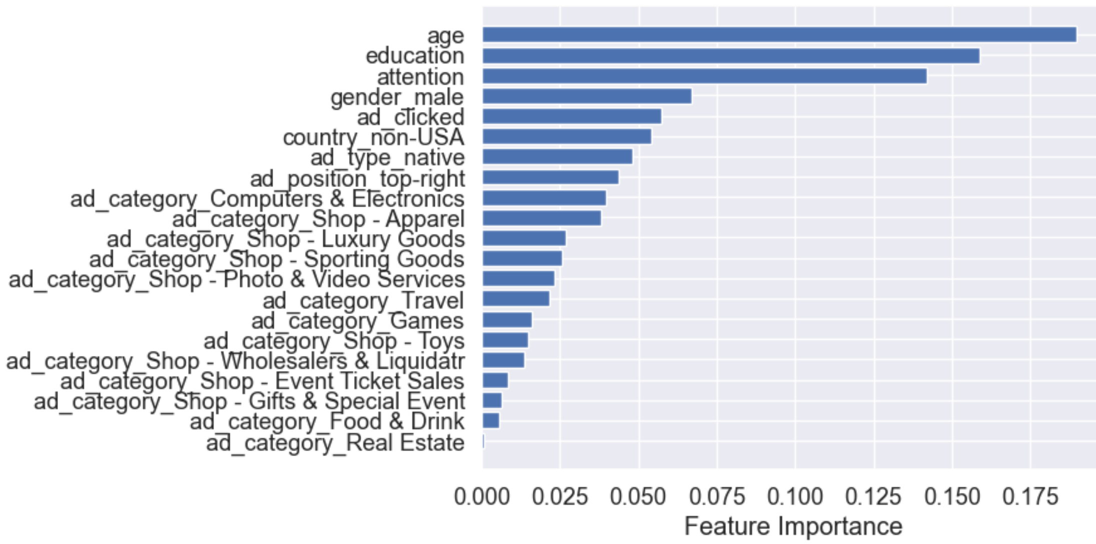
	Feature	Importance
1	age	0.189590
0	education	0.159023
3	attention	0.142107
5	gender_male	0.066834
2	ad_clicked	0.057357
4	country_non-USA	0.054138
7	ad_type_native	0.047955
6	ad_position_top-right	0.043516
8	ad_category_Computers & Electronics	0.039514
12	ad_category_Shop - Apparel	0.037986
15	ad_category_Shop - Luxury Goods	0.026820
17	ad_category_Shop - Sporting Goods	0.025682
16	ad_category_Shop - Photo & Video Services	0.023090
20	ad_category_Travel	0.021728
10	ad_category_Games	0.015919
18	ad_category_Shop - Toys	0.014610
19	ad_category_Shop - Wholesalers & Liquidatr	0.013529
13	ad_category_Shop - Event Ticket Sales	0.008278
14	ad_category_Shop - Gifts & Special Event	0.006167
9	ad_category_Food & Drink	0.005539
11	ad_category_Real Estate	0.000619

Visualize Feature Importances

[1282]:
```
sort = RF_model.feature_importances_.argsort()

plt.barh(X_train.columns[sort], RF_model.feature_importances_[sort])
plt.xlabel('Feature Importance')
```

[1282]: Text(0.5, 0, 'Feature Importance')



Permutation Importance

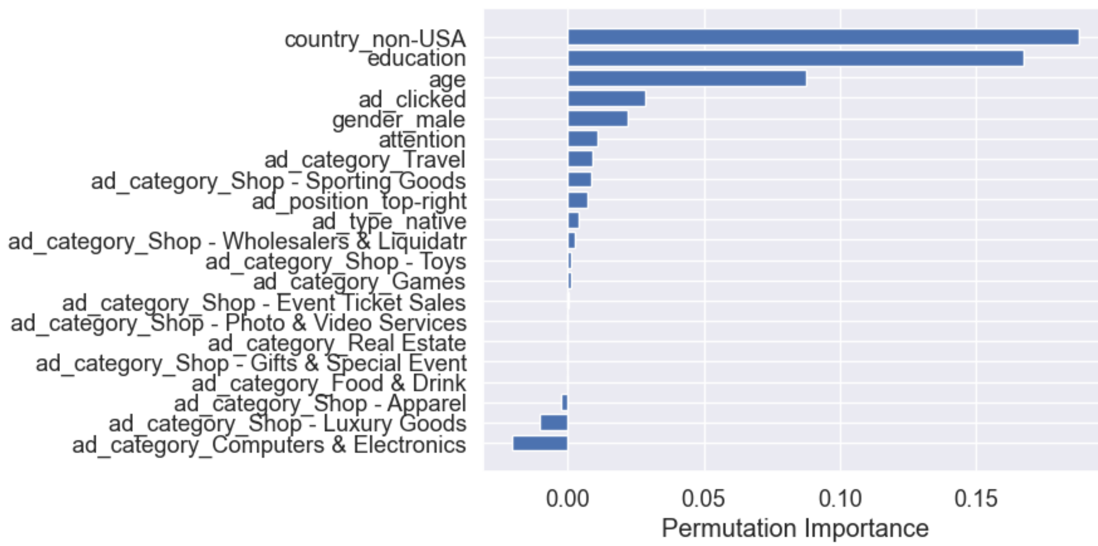
We can also calculate permutation importance, which accounts for a possible error in the regular feature importance function, which is that importance may automatically be assigned to the features with highest cardinality. In our case above, the top ranked features do have higher cardinality than the others. The permutation function finds country_non-USA to be the highest ranked feature. This was also the feature with the highest coefficient in our linear regression model.

[1284]:
```
from sklearn.inspection import permutation_importance

perm_importance = permutation_importance(RF_model, X_test, y_test)

sort2 = perm_importance.importances_mean.argsort()
plt.barh(X_train.columns[sort2], perm_importance.importances_mean[sort2])
plt.xlabel("Permutation Importance")
```

[1284]:
```
Text(0.5, 0, 'Permutation Importance')
```



0.4 Conclusions

Based on our metrics, we can conclude that the performance of the linear regression and random forest models is roughly the same, with the linear model producing *slightly* lower error.

Both models are able to predict income between 1 to 2 units from the actual income. This isn't terrible, but it also isn't great. Perhaps a different type of regression model would fit the data better. It is also possible that the features in our dataset just aren't particularly correlated with income.

Country_non-USA was found by both models to show strongest association with income - in the linear regression model we determined this by finding its coefficient, and in random forest by ranking permutation importances.

Ultimately, our models do not provide overwhelming insights to our data. Perhaps for the future we could try a different model, reduce the number of features (since most do not appear to be associated much with income), and possibly gather different data to find new variables which are better predictors of income.