DS4400 Final Project: Airbnb Modeling

December 7, 2018

1 Collect Data

Here, all necessary data is retrieved and aggregated. The resulting dataframes are then engineered as necessary and exported as csv files for later use.

1.1 Load Packages and define utility functions

```
In [13]: import numpy as np
         import pandas as pd
         from pathlib import Path
         from sklearn.metrics import accuracy_score
         from sklearn.preprocessing import LabelEncoder
         import datetime
         import requests
         pd.options.display.max_columns = 100
         pd.options.display.max_rows = 1000
```

```
In [14]: # utility functions

         # returns the time difference between now and a date given as a string
         def host_dur(t):
             if str(t) == "nan":
                 return np.nan
             t = np.datetime64(str(t))
             delt = np.datetime64(datetime.datetime.now()) - t
             delt_int = np.timedelta64(delt, 'D').astype(int)
             return delt_int
```

1.2 Generate urls and read in all data

Note: All data is collected from insideairbnb.com. Only data from Boston is used for this project.

```
In [15]: # generate urls
         start = "http://data.insideairbnb.com/united-states/ma/boston/"
         list_end = "/data/listings.csv.gz"
         cal_end = "/data/calendar.csv.gz"
         dates = ['2015-10-03', '2016-09-07', '2017-10-06', '2018-04-14', '2018-05-17', '2018-07
```

```
list_urls = list(map(lambda d: start + d + list_end, dates))
cal_urls = list(map(lambda d: start + d + cal_end, dates))
```

In [16]:
```
# read in, append data, and write full tables to csv

all_listings = []
all_calendar = []

print("Reading listing urls...")
for l in list_urls:
    data = pd.read_csv(l)
    all_listings.append(data)

print("Reading calendar urls...")
for c in cal_urls:
    data = pd.read_csv(c)
    all_calendar.append(data)

print("Concatenating data...")
list_df = pd.concat(all_listings, axis=0, ignore_index=True, sort=True)
cal_df = pd.concat(all_calendar, axis=0, ignore_index=True, sort=True)
print("Done")
```

```
Reading listing urls...
Reading calendar urls...
Concatenating data...
Done
```

1.3 Clean and engineer data

Generated features include: * is_professional -- whether or not a host is a professional host * n_amenities -- the number of amenities a listing has * occ_rate -- the occupancy rate of a listing during the year that it was scraped from * host_dur -- the length of time that a listings host has been a host * n_pchange -- the number of price changes of a listing during the year that it was scraped from

In [19]:
```
print("Cleaning and engineering listing features...")

# engineer feature 'is_professional'
list_df['is_professional'] = 0
list_df.is_professional.loc[list_df["host_total_listings_count"] > 1] = 1

# engineer feature 'number of amenities'
list_df['n_amenities'] = list_df.amenities.map(lambda x: len(x))

# clean columns by removing '$'
list_df['cleaning_fee'] = list_df['cleaning_fee'].str.replace('$', '').apply(pd.to_nume
list_df['extra_people'] = list_df['extra_people'].str.replace('$', '').apply(pd.to_nume
```

```
list_df['monthly_price'] = list_df['monthly_price'].str.replace('$', '').str.replace(',
list_df['price'] = list_df['price'].str.replace('$', '').str.replace(',', '').apply(pd.
list_df['weekly_price'] = list_df['weekly_price'].str.replace('$', '').str.replace(',',
list_df['security_deposit'] = list_df['security_deposit'].str.replace('$', '').str.repl

# convert percentage to decimal
list_df['host_acceptance_rate'] = list_df['host_acceptance_rate'].str.replace('%', '').
list_df['host_response_rate'] = list_df['host_response_rate'].str.replace('%', '').appl

# engineer feature 'host_length'
list_df['host_dur'] = list_df.host_since.map(lambda x: host_dur(x))

# change zipcode feature
list_df['zipcode'] = list_df['zipcode'].apply(str)

# add year feature
list_df['year'] = list_df.last_scraped.map(lambda x: x[0:4]).astype(int)

# rename listing id feature
list_df['listing_id'] = list_df['id']
list_df = list_df.drop(['id'], axis=1)

# drop unwanted columns
bad_features = ['license', 'security_deposit', 'square_feet', 'last_scraped', 'weekly_p
list_df = list_df.drop(bad_features, axis=1)
list_df = list_df.dropna()

print("Done")
```

```
Cleaning and engineering listing features...
Done
```

In [9]:
```
# NOTE: takes a while

print("Cleaning calendar data...")
# remove $ from price column
cal_df['price'] = cal_df['price'].str.replace('$', '').str.replace(',','').apply(pd.to_n
cal_df['available'] = cal_df['available'].str.replace('f', '0').str.replace('t', '1').as

# make a copy before engineering
cal_raw = cal_df.copy(deep=True)

print("Engineering calendar features...")
# generate occupancy rate
cal_df['year'] = cal_df.date.map(lambda x: x[0:4])
cal_count = cal_df.groupby(['listing_id', 'year']).count()
cal_count['occ_rate'] = cal_count.price / cal_count.available
```

```
cal_count = cal_count.reset_index()

print("Generating price changes...")
# generate price changes
cal_nunique = cal_df.groupby(['listing_id', 'year']).nunique()

print("Reformatting calendar dataframe...")
# reformat dataframe
cal_df = pd.DataFrame(data={'listing_id': cal_count['listing_id'].values,
                        'year': cal_count['year'].values,
                        'occ_rate': cal_count['occ_rate'].values,
                        'n_pchange':cal_nunique['price'].values})
cal_df['year'] = cal_df['year'].astype(int)

print("Done")
```

```
Cleaning calendar data...
Engineering calendar features...
Generating price changes...
Reformatting calendar dataframe...
Done
```

1.4 Merge dataframes and write to .csv

```
In [10]: print("Merging dataframes...")
         # merge dataframes
         df = list_df.merge(cal_df, on=['listing_id', 'year'], how='left')

         print("Changing year column...")
         # change year column to year since
         df['year_since'] = df['year'] - df['year'].min()
         df = df.drop('year', axis=1)

         print("Done")
```

```
Merging dataframes...
Changing year column...
Done
```

```
In [11]: dir = str(Path().resolve())
         df.to_csv(dir + "/../data/listings.csv", index=False)
         list_df.to_csv(dir + "/../data/list_df.csv", index=False)
         cal_raw.to_csv(dir + "/../data/cal_df.csv", index=False)
```

[T1]fontenc mathpazo
graphicx caption nolabel

adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem

visualizations

verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

2 Explore Data

In this section, data is explored by viewing feature distributions and relationships between different variables

2.1 Import packages and load data

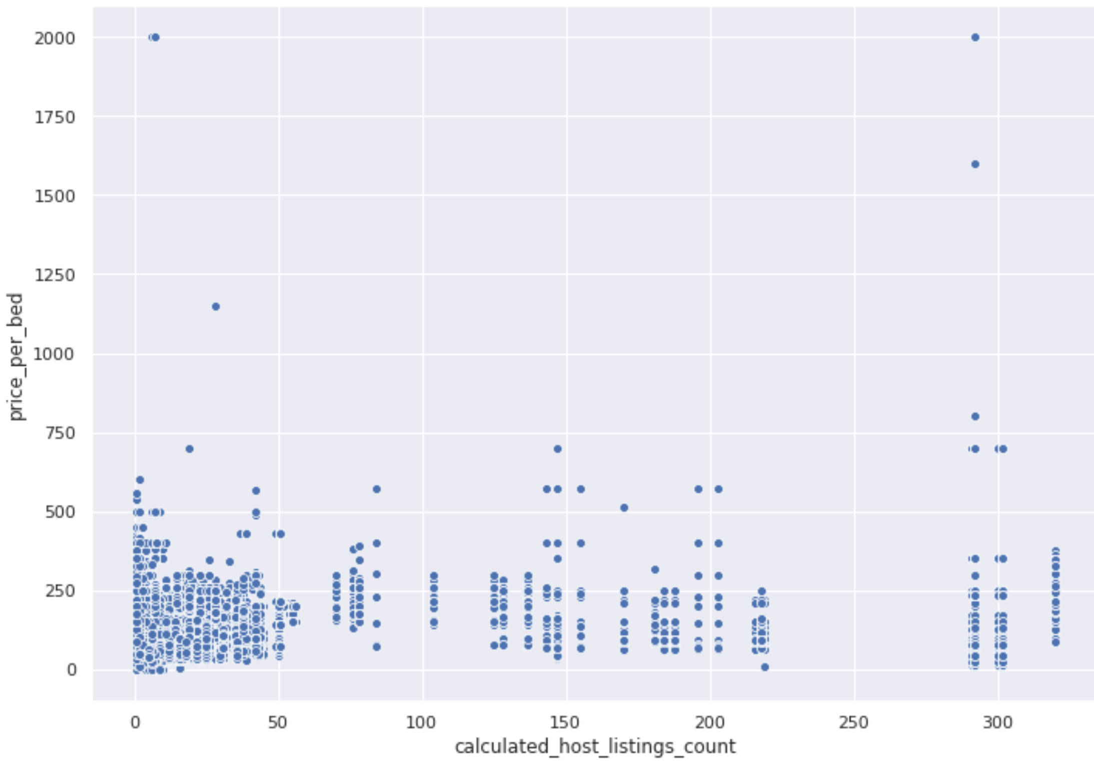
```
In [1]: import numpy as np
        import pandas as pd
        from pathlib import Path
        from sklearn.metrics import accuracy_score
        from sklearn.preprocessing import LabelEncoder
        import datetime
        import seaborn as sns
        pd.options.display.max_columns = 100
        pd.options.display.max_rows = 1000
```

```
In [11]: # import data
         dir = str(Path().resolve())
         df = pd.read_csv("/Users/cccdenhart/Documents/ds4400/project/data/listings.csv")
```

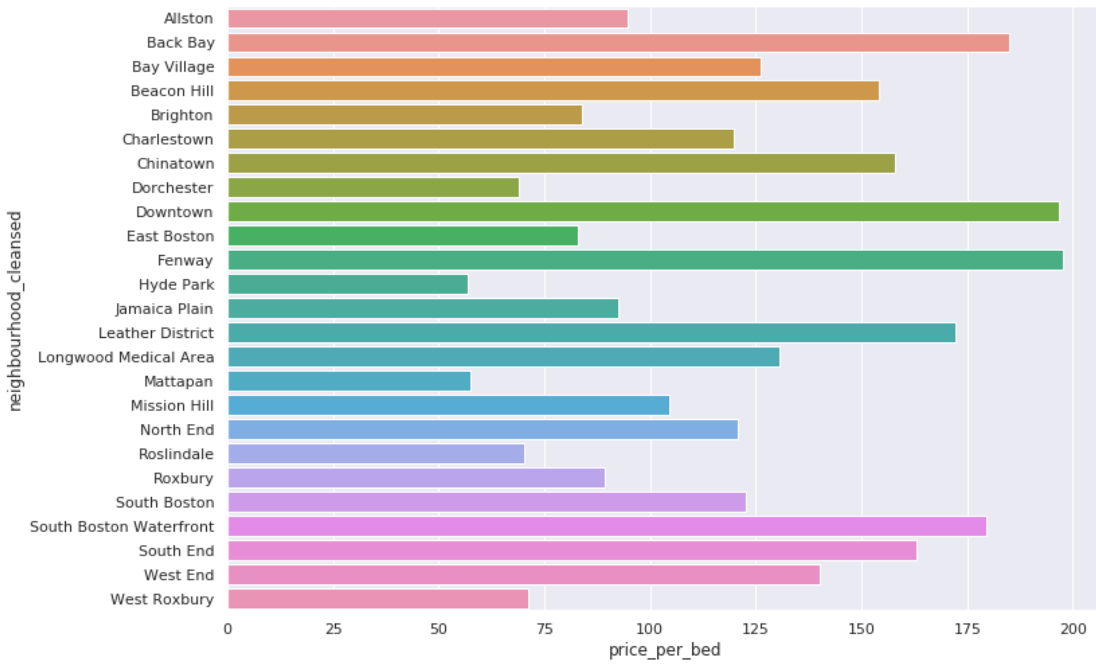
```
In [12]: # impute bedroom values of 0.0 with 1.0
         df.loc[df.bedrooms==0.0,['bedrooms']] = 1.0
```

```
In [13]: # engineer feature price_per_bed
         df['price_per_bed'] = df.price / df.bedrooms
```

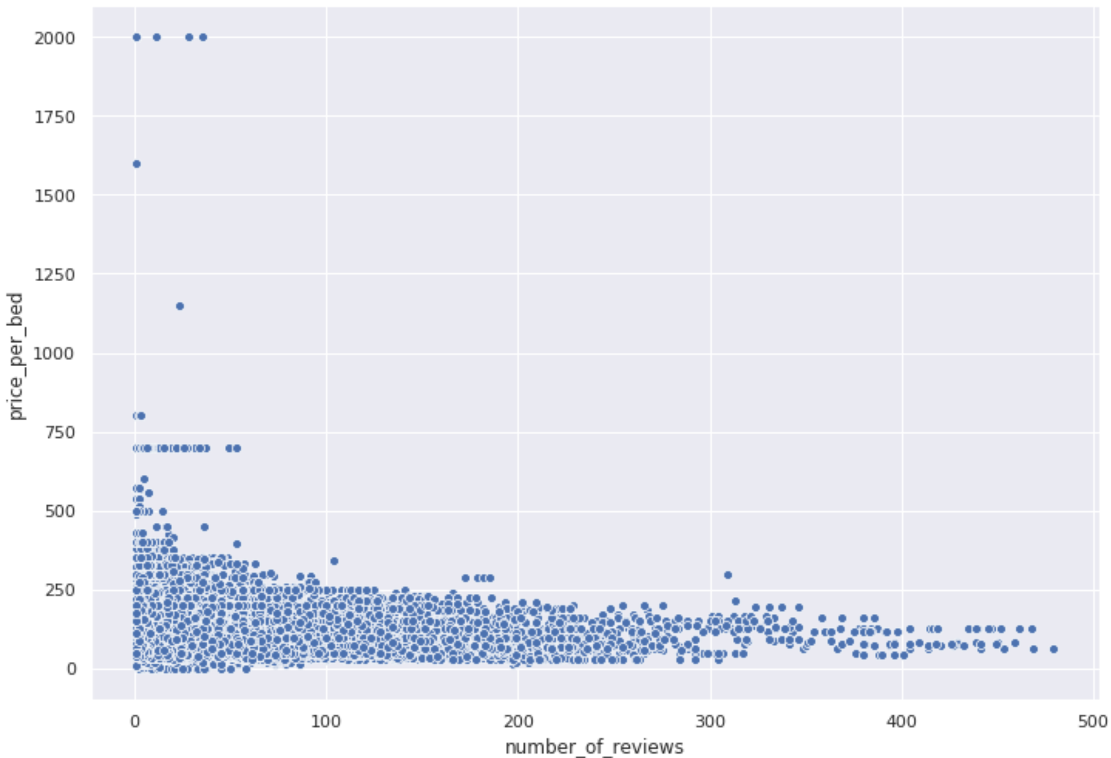
```
In [14]: # scatterplot price_per_bed with host_listing_count
         plt = sns.scatterplot(x=df.calculated_host_listings_count, y=df.price_per_bed)
         fig = plt.get_figure()
         fig.savefig("/Users/cccdenhart/Documents/ds4400/project/plots/list_vs_price.png")
```



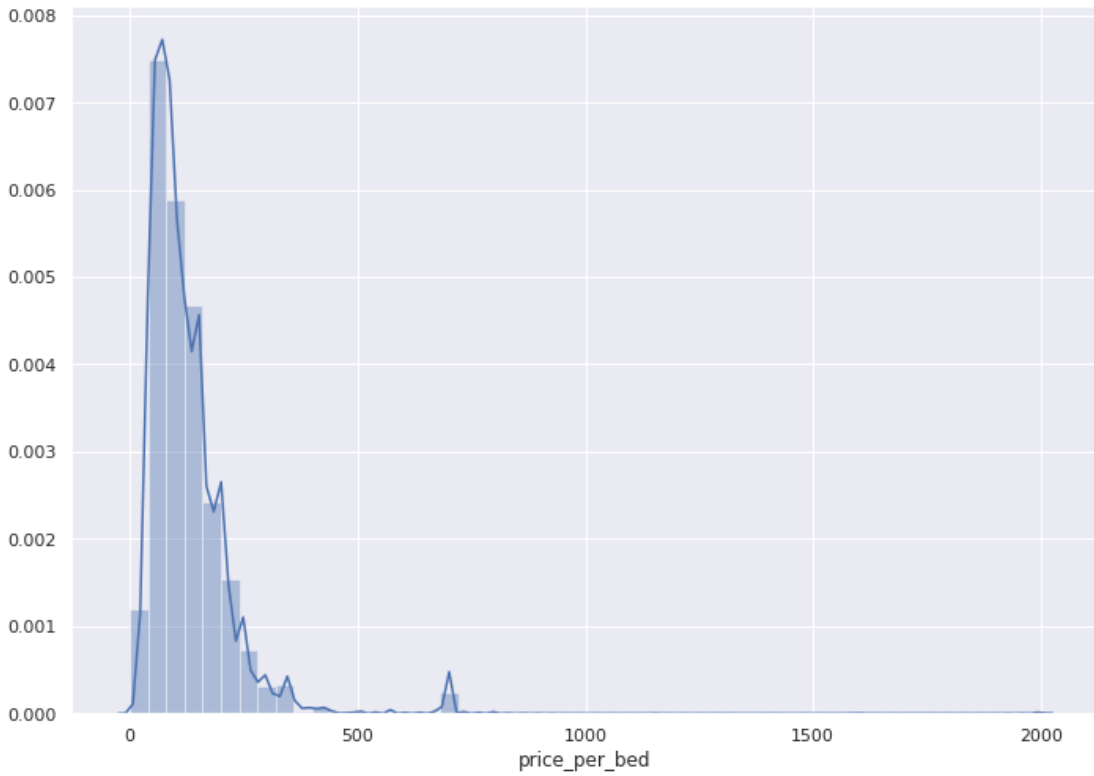

```
In [15]: # barplot price_per_bed by neighborhood
         by_neigh = df.groupby(['neighbourhood_cleansed']).mean().reset_index()
         sns.set(rc={'figure.figsize':(11.7,8.27)})
         plt = sns.barplot(y=by_neigh.neighbourhood_cleansed, x=by_neigh.price_per_bed)
         fig = plt.get_figure()
         fig.savefig("/Users/cccdenhart/Documents/ds4400/project/plots/neigh_vs_price.png")
```



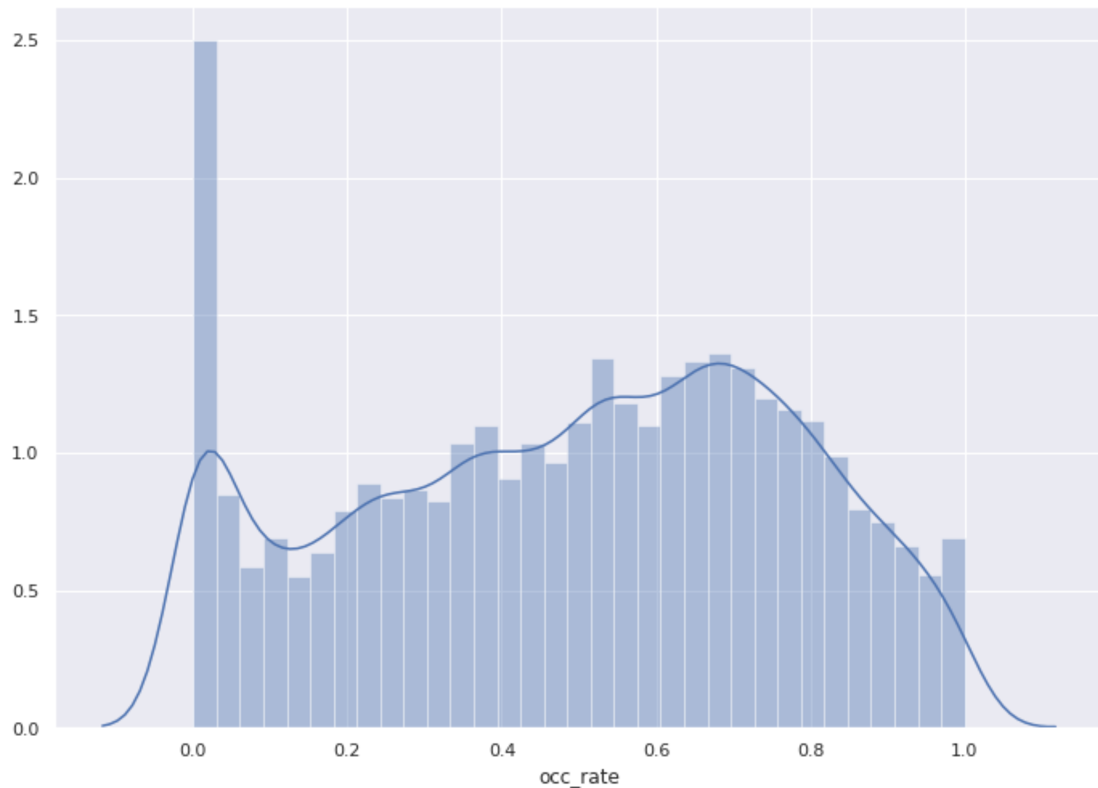

```
In [16]: # plot occupancy rate vs host duration
    plt = sns.scatterplot(x=df.number_of_reviews, y=df.price_per_bed)
    fig = plt.get_figure()
    fig.savefig("/Users/cccdenhart/Documents/ds4400/project/plots/numrev_vs_price.png")
```



In [17]:
```
# plot occupancy rate vs host duration
plt = sns.distplot(df.price_per_bed)
fig = plt.get_figure()
fig.savefig("/Users/cccdenhart/Documents/ds4400/project/plots/price_hist.png")
```




```
In [18]: # plot occupancy rate vs host duration
         plt = sns.distplot(df.occ_rate)
         fig = plt.get_figure()
         fig.savefig("/Users/cccdenhart/Documents/ds4400/project/plots/occ_hist.png")
```



[T1]fontenc mathpazo
graphicx caption nolabel
adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem
review_scores
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

3 Predict Review Scores

Here, I try to predict the average review score of a listing using linear regressions and a radial SVR

3.1 Import packages and load in data

```
In [1]: import numpy as np
        import pandas as pd
        from pathlib import Path
        from sklearn.metrics import accuracy_score
        from sklearn.model_selection import train_test_split
        from sklearn.decomposition import PCA
        from sklearn.preprocessing import normalize
        from sklearn.linear_model import LinearRegression, Ridge
```

```
from sklearn.svm import SVR
from sklearn.metrics import explained_variance_score, mean_squared_error, r2_score
import seaborn as sns
import math
pd.options.display.max_columns = 100
pd.options.display.max_rows = 1000
```

In [2]:
```
# import data
dir = str(Path().resolve())
df = pd.read_csv(dir + "/../data/listings.csv")
```

3.2 Prepare data

In [3]:
```
# average all reviews to generate average_score feature and drop excess review features
df['average_score'] = (df.review_scores_accuracy + df.review_scores_checkin + df.review_
df = df.drop(['review_scores_accuracy', 'review_scores_checkin', 'review_scores_cleanlin
```

In [4]:
```
# split into X and Y
X = df.drop('average_score', axis=1)
y = df['average_score']

# convert objects to dummy variables
X = pd.get_dummies(X)

# split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

3.3 Train models

In [5]:
```
# NOTE: takes a long time
# train models
print("Training linear regression...")
lm = LinearRegression().fit(X_train, y_train)
print("Training linear reg with ridge regularization")
reg = Ridge(alpha=.5).fit(X_train, y_train)
print("Training radial kernel SVR...")
rad = SVR(gamma="auto", kernel="rbf").fit(X_train, y_train)
print("Done")
```

```
Training linear regression...
Training linear reg with ridge regularization
Training radial kernel SVR...


/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: LinA
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number3.765920e-19
  overwrite_a=True).T
```

```
Done
```

```
In [6]: # get predictions
        print("Generating predictions...")
        lm_pred = lm.predict(X_test)
        reg_pred = reg.predict(X_test)
        rad_pred = rad.predict(X_test)
        print("Done")
```

```
Generating predictions...
Done
```

3.4 View results

```
In [7]: # display evaluations
        print("Linear Model Results:")
        print("Explained Variance: " + str(explained_variance_score(y_test, lm_pred)))
        print("MSE: " + str(mean_squared_error(y_test, lm_pred)))
        print("R2: " + str(r2_score(y_test, lm_pred)))
        print()
        # display evaluations
        print("Ridge Regularization Results:")
        print("Explained Variance: " + str(explained_variance_score(y_test, reg_pred)))
        print("MSE: " + str(mean_squared_error(y_test, reg_pred)))
        print("R2: " + str(r2_score(y_test, reg_pred)))
        print()
        print("Linear Model with Regularization:")
        print("Radial Kernel SVR Results:")
        print("Explained Variance: " + str(explained_variance_score(y_test, rad_pred)))
        print("MSE: " + str(mean_squared_error(y_test, rad_pred)))
        print("R2: " + str(r2_score(y_test, rad_pred)))
```

```
Linear Model Results:
Explained Variance: 0.1434287737138703
MSE: 0.34507206786518557
R2: 0.1430113908638756

Ridge Regularization Results:
Explained Variance: 0.14379407926425725
MSE: 0.3449246444374958
R2: 0.14337751785593122

Linear Model with Regularization:
Radial Kernel SVR Results:
Explained Variance: 0.2965080758246498
MSE: 0.28368482487039837
```

R2: 0.29546698751146394

3.5 Discussion

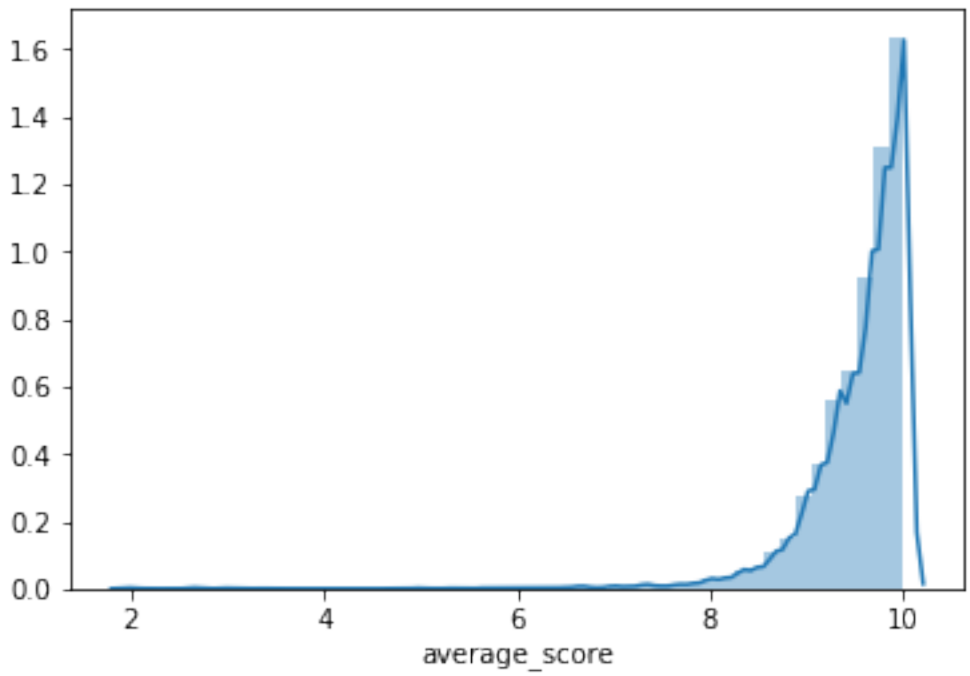
3.5.1 Why are results so bad?

- Large left skew (histogram 1)
- Tried applying log transform to data --> still large skew (histogram 2)

```
In [10]: sns.distplot(df.average_score)
```

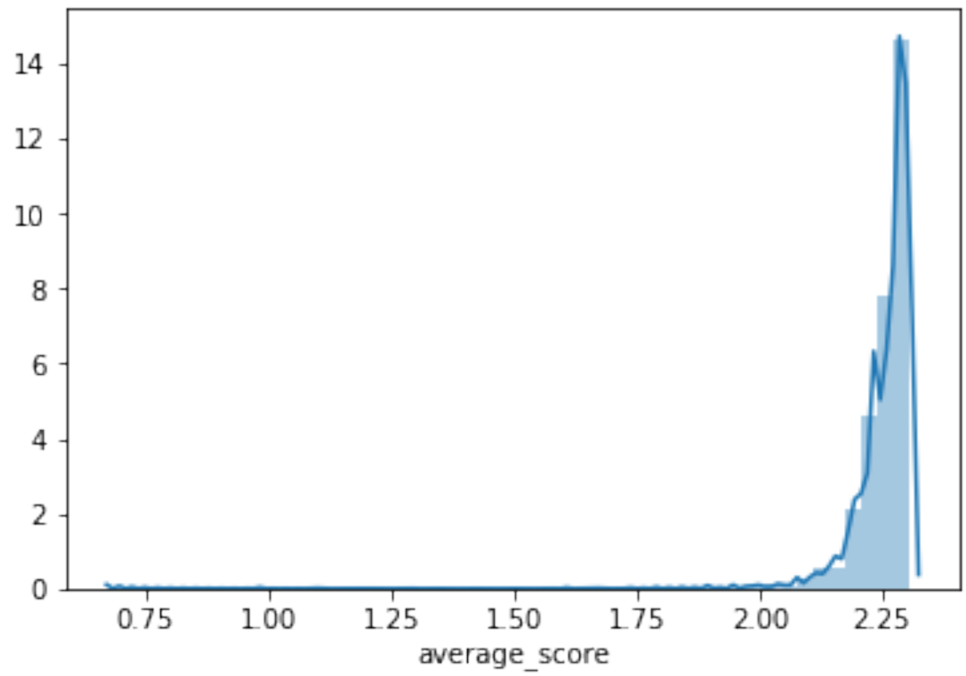
```
/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarnin
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1c22b58470>
```




```
In [12]: sns.distplot(df.average_score.apply(math.log))
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1c26871978>
```



[T1]fontenc mathpazo
graphicx caption nolabel
adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem
prof_status
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

4 Predict Host Professional Status

In this section, I attempt to predict whether a host is a professional or not. Note that a host is considered professional if they list more than one unit on Airbnb, as justified by previous literature.
Logistic regression, a decision tree, and AdaBoost are all used to classfiy host status.

4.1 Import packages and load data

```
In [1]: import numpy as np
        import pandas as pd
        from pathlib import Path
        from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import AdaBoostClassifier
        from sklearn.model_selection import train_test_split
        import seaborn as sns
```

```
        pd.options.display.max_columns = 100
        pd.options.display.max_rows = 1000
```

```
In [2]: # import data
        dir = str(Path().resolve())
        df = pd.read_csv(dir + "/../data/listings.csv")
```

4.2 Prepare data

```
In [3]: # split into X and y
        X = df.drop('is_professional', axis=1)
        y = df['is_professional']

        # drop host listing count
        X = X.drop(['calculated_host_listings_count', 'host_total_listings_count', 'host_listing

        # get dummies
        X = pd.get_dummies(X)

        # split into train and test
        X_train, X_test, y_train, y_test = train_test_split(X, y)
```

4.3 Train models

```
In [4]: # train models
        print("Training logistic regression...")
        log = LogisticRegression().fit(X_train, y_train)
        print("Training decision tree...")
        dec = DecisionTreeClassifier().fit(X_train, y_train)
        print("Training AdaBoost...")
        ada = AdaBoostClassifier(n_estimators=50).fit(X_train, y_train)
        print("Done")
```

```
Training logistic regression...


/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: F
  FutureWarning)


Training decision tree...
Training AdaBoost...
Done
```

```
In [5]: # get predictions
        print("Getting predictions...")
        log_pred = log.predict(X_test)
        dec_pred = dec.predict(X_test)
```

```
        ada_pred = ada.predict(X_test)
        # add in adaboost
        print("Done")
```

```
Getting predictions...
Done
```

4.4 View results

```
In [6]: # display performance metrics
        print("Logistic Regression Results:")
        print("accuracy: " + str(accuracy_score(y_test, log_pred)))
        print("precision: " + str(precision_score(y_test, log_pred)))
        print("recall: " + str(recall_score(y_test, log_pred)))
        print("auc: " + str(roc_auc_score(y_test, log_pred)))
        print()
        print("Decision Tree Results:")
        print("accuracy: " + str(accuracy_score(y_test, dec_pred)))
        print("precision: " + str(precision_score(y_test, dec_pred)))
        print("recall: " + str(recall_score(y_test, dec_pred)))
        print("auc: " + str(roc_auc_score(y_test, dec_pred)))
        print()
        print("AdaBoost Results:")
        print("accuracy: " + str(accuracy_score(y_test, ada_pred)))
        print("precision: " + str(precision_score(y_test, ada_pred)))
        print("recall: " + str(recall_score(y_test, ada_pred)))
        print("auc: " + str(roc_auc_score(y_test, ada_pred)))
```

```
Logistic Regression Results:
accuracy: 0.7376577728071577
precision: 0.7376577728071577
recall: 1.0
auc: 0.5

Decision Tree Results:
accuracy: 0.9012621824572615
precision: 0.9277005347593583
recall: 0.9393545592376001
auc: 0.8667540153069851

AdaBoost Results:
accuracy: 0.8041220642275124
precision: 0.8370105346849533
recall: 0.9120641108945202
auc: 0.7063365621464075
```

4.5 Discussion

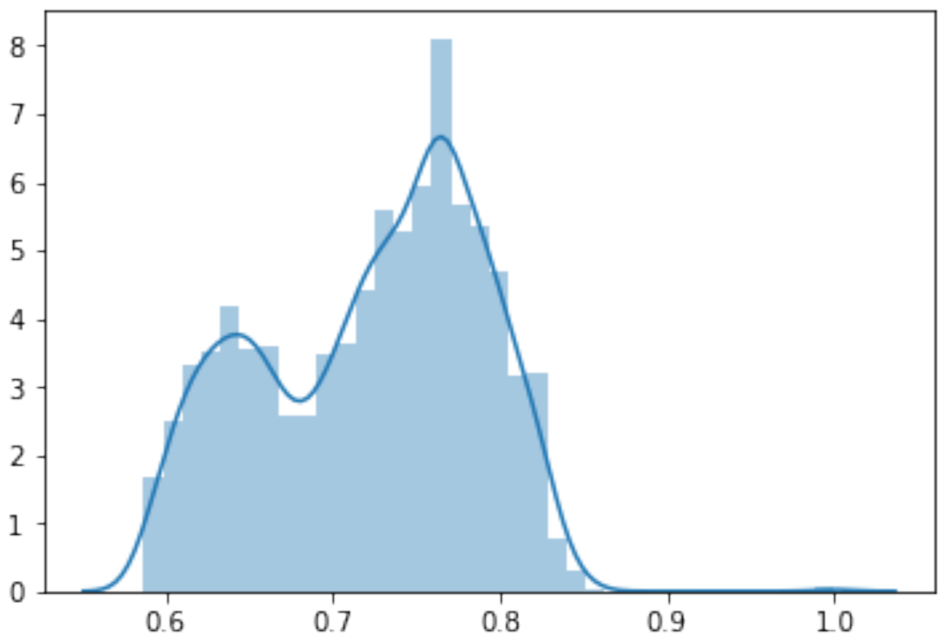
4.5.1 Why are Logistic Reg results odd?

- All prediction probabilities are > .5

```
In [41]: probs = list(map(lambda x: x[1], log.predict_proba(X_test)))
         sns.distplot(probs)
```

```
/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarnin
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2b05be10>
```



[T1]fontenc mathpazo
graphicx caption nolabel
adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem
neighborhood
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

5 Neighborhood prediction

In this section, the neighborhood of a given listing is predicted using KNN, Naive-Bayes, and a random forest.

5.1 Import packages and read in data

In [2]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import math
import seaborn as sns
pd.options.display.max_columns = 100
pd.options.display.max_rows = 1000
```

In [3]:
```
# import data
dir = str(Path().resolve())
df = pd.read_csv(dir + "/../data/listings.csv")
```

5.2 Prepare data

In [4]:
```
# split into X and y
X = df.drop('neighbourhood_cleansed', axis=1)
y = LabelEncoder().fit_transform(df['neighbourhood_cleansed'])

# get dummies
X = pd.get_dummies(X)

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

5.3 Train models

In [52]:
```
# train models
print("Training KNN...")
knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)
print("Training decision tree...")
nb = GaussianNB().fit(X_train, y_train)
print("Training random forest...")
rf = RandomForestClassifier(n_estimators=int(math.sqrt(X.shape[1])), max_depth=10).fit(
print("Done")
```

```
Training KNN...
Training decision tree...
Training random forest...
Done
```

```
In [53]: # get predictions
         print("Generating predictions...")
         knn_pred = knn.predict(X_test)
         nb_pred = nb.predict(X_test)
         rf_pred = rf.predict(X_test)
         print("Done")
```

```
Generating predictions...
Done
```

5.4 View results

```
In [55]: # display performance metrics
         print("KNN Results:")
         print("accuracy: " + str(accuracy_score(y_test, knn_pred)))
         print("precision: " + str(precision_score(y_test, knn_pred, average="macro")))
         print("recall: " + str(recall_score(y_test, knn_pred, average="macro")))
         # print("auc: " + str(roc_auc_score(y_test, knn_pred)))
         print()
         print("Naive Bayes Results:")
         print("accuracy: " + str(accuracy_score(y_test, nb_pred)))
         print("precision: " + str(precision_score(y_test, nb_pred, average="macro")))
         print("recall: " + str(recall_score(y_test, nb_pred, average="macro")))
         #print("auc: " + str(roc_auc_score(y_test, nb_pred)))
         print()
         print("Random Forest Results:")
         print("accuracy: " + str(accuracy_score(y_test, rf_pred)))
         print("precision: " + str(precision_score(y_test, rf_pred, average="macro")))
         print("recall: " + str(recall_score(y_test, rf_pred, average="macro")))
         #print("auc: " + str(roc_auc_score(y_test, rf_pred)))
```

```
KNN Results:
accuracy: 0.777760025563189
precision: 0.7721725205515614
recall: 0.7237850559072051

Naive Bayes Results:
accuracy: 0.11199872184054961
precision: 0.05855255442334741
recall: 0.07022498947156558

Random Forest Results:
accuracy: 0.9129253874420834
precision: 0.8929981744782387
recall: 0.7782661572565764
```

```
/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143:
  'precision', 'predicted', average, warn_for)
/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143:
  'precision', 'predicted', average, warn_for)
```

[T1]fontenc mathpazo
graphicx caption nolabel
adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem
price_changes
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

6 Generate Price Change Data

In this section, daily price data about listings is reshaped in order to convey more information about how price changes of a given listing impact property performance. Price changes for each listing are extracted and for each price change, the change in average daily revenue resulting from the price change is calculated.

6.1 Import packages and load in data

```
In [6]: import numpy as np
        import pandas as pd
        from pathlib import Path
        from sklearn.model_selection import train_test_split
        import datetime
        import seaborn as sns
        pd.options.display.max_columns = 100
        pd.options.display.max_rows = 1000
```

```
In [2]: # import data
        dir = str(Path().resolve())
        cal_df = pd.read_csv(dir + "/../data/cal_df.csv")
        cal_df['date'] = df['date'].apply(lambda x: datetime.date(int(x[0:4]), int(x[5:7]), int(
```

6.2 Define functions to be used to extract average daily revenues

```
In [123]: # find the difference between each sequential value in the given list
          def get_diff(arr):
              if len(arr) > 0:
                  diff = [arr.pop(0)]
                  for index in range(len(arr)):
                      diff.append(arr[index] - arr[index - 1])
                  return diff
              else:
                  return arr
```

```
In [94]: # Array[float] -> Array[float]
         # given an array of prices, finds the average daily revenue (adr) changes between every
         def get_adr(prices):
             adr = []
             pc = []
             ind = []
             cur_p = prices[0]
             start_index = 0
             revenue = 0
             for index, p in enumerate(prices):
                 if not np.isnan(p):
                     if p == cur_p:
                         revenue += p
                     else:
                         adr.append(revenue / (index - start_index))
                         pc.append(p - cur_p)
                         ind.append(index)
                         cur_p = p
                         start_index = index
                         revenue = 0
             return get_diff(adr), pc, ind
```

6.3 Implement functions to find average daily revenue changes per price change per listing

```
In [124]: # find adr changes for each listing
          ids = cal_df.listing_id.unique()
          all_dfs = []
          for i in ids:
              print(i)
              l = cal_df.loc[cal_df.listing_id==i,:].sort_values(by='date', ascending=True)
              adr, pc, ind = get_adr(l.price.values)
              dates = l.date.iloc[ind]
              id = [l.listing_id.values[0]] * len(adr)
              all_dfs.append(pd.DataFrame(data={'id': id, 'date' : dates, 'revenue_change': adr,
```

```
1810172
6976
3075044
4283698
4085362
225834
7252607
1936861
225979
2583074
6933545
5434353
```

```
29186224
29189657
29122035
```

6.4 Engineer and export resulting dataset

```
In [148]: # group all dataframes from adr change results
          df = pd.concat(all_dfs, axis=0, ignore_index=True, sort=True)

In [149]: # engineer dataframe before writing
          df = df.dropna()
          df['year'] = df.date.apply(lambda x: x.year)
          year_start = df.date.apply(lambda x: datetime.date(int(x.year), 1, 1))
          df['days_since'] = (df.date - year_start).apply(lambda x: int(x.days))
          df = df.drop('date', axis=1)

In [150]: df.to_csv(dir + "/../data/price_changes.csv", index=False)
```

[T1]fontenc mathpazo
graphicx caption nolabel
adjustbox xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs [utf8x]inputenc fancyvrb grffile hyperref longtable booktabs [inline]enumitem [normalem]ulem
optimal_price
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

7 Predict Price Change Effects

Using the reshaped listing price data, the average daily revenue change resulting from a price change is predicted using KNN, a random forest, and a feed-forward neural network.

7.1 Import packages and load in data

```
In [11]: import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.feature_selection import SelectKBest, f_classif
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
     from keras.models import Sequential
     from keras.layers.core import Dense, Dropout, Activation
     from keras.layers import Conv2D, MaxPooling2D, Flatten
     from keras.losses import categorical_crossentropy
     from keras.optimizers import SGD
     from keras.utils import np_utils
     import keras.callbacks as cb
```

```
 from pathlib import Path
 pd.options.display.max_columns = 100
 pd.options.display.max_rows = 1000
```

```
In [2]: # import data
    dir = str(Path().resolve())
    pc_df = pd.read_csv(dir + "/../data/price_changes.csv")
    list_df = pd.read_csv(dir + "/../data/list_df.csv")
```

7.2 Prepare data

```
In [3]: # rename listing id column
    list_df['id'] = list_df['listing_id']
    list_df = list_df.drop('listing_id', axis=1)
```

```
In [4]: print("Merging dataframes...")
    # merge dataframes
    df = pc_df.merge(list_df, on=['id', 'year'], how='left')
    print("Done")
```

```
Merging dataframes...
Done
```

```
In [5]: # remove rows with missing values
    df = df.dropna()

    # drop listing id
    df = df.drop('id', axis=1)
```

```
In [21]: print("df shape: ", df.shape)
```

```
df shape:  (5230391, 43)
```

```
In [6]: def bin_adr(x):
        if x > 0.0:
            return 0
        elif x < 0.0:
            return 1
        else:
            return 2
```

```
In [7]: # bin revenue
    df['revenue_change'] = df.revenue_change.apply(bin_adr)
```

```
In [8]: print("Splitting into X and y...")
    # split into X and y
    y = df['revenue_change']
```

```python
    X = df.drop('revenue_change', axis=1)

    print("Getting dummy values...")
    # get dummy values
    X = pd.get_dummies(X)

    print("Selecting features...")
    # select 20 best features
    X = pd.DataFrame(SelectKBest(score_func=f_classif, k=20).fit_transform(X, y.values))

    print("Splitting into train and test...")
    # split into training and testing data
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    print("Done")
```

```
Splitting into X and y...
Getting dummy values...
Selecting features...
```

```
/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/sklearn/feature_selection/univariate_se
UserWarning)
/Users/cccdenhart/miniconda3/lib/python3.7/site-packages/sklearn/feature_selection/univariate_se
f = msb / msw
```

```
Splitting into train and test...
Done
```

7.3 Train models

In [12]:
```python
print("Training KNN...")
knn = KNeighborsClassifier(n_neighbors=10).fit(X_train, y_train)
print("Predicting KNN...")
knn_pred = knn.predict(X_test)
```

```
Training KNN...
Predicting KNN...
```

In [13]:
```python
print("Training random forest...")
rf = RandomForestClassifier(n_estimators=50).fit(X_train, y_train)
print("Predicting random forest...")
rf_pred = rf.predict(X_test)
```

```
Training random forest...
Predicting random forest...
```

```
In [15]: # convert y values
         nn_train = np_utils.to_categorical(y_train, 3)
         nn_test = np_utils.to_categorical(y_test, 3)

         # initialize feed-forward neural network
         print("Initializing network...")
         ffnn = Sequential()
         ffnn.add(Dense(units=12, activation='relu', input_dim=20))
         ffnn.add(Dropout(0.2))
         ffnn.add(Dense(units=10, activation='exponential'))
         ffnn.add(Dropout(0.2))
         ffnn.add(Dense(units=8, activation='sigmoid'))
         ffnn.add(Dropout(0.2))
         ffnn.add(Dense(units=3, activation='relu'))

         # compile model
         print("Compiling model...")
         ffnn.compile(loss=categorical_crossentropy, optimizer=SGD(lr=0.01, momentum=0.9, nesterov=T

         # fit model
         print("Fitting model...")
         ffnn.fit(X_train, nn_train, epochs=10, batch_size=128, verbose=2)
```

```
Initializing network...
Compiling model...
Fitting model...
Epoch 1/10
- 59s - loss: nan
Epoch 2/10
- 63s - loss: nan
Epoch 3/10
- 59s - loss: nan
Epoch 4/10
- 60s - loss: nan
Epoch 5/10
- 68s - loss: nan
Epoch 6/10
- 68s - loss: nan
Epoch 7/10
- 58s - loss: nan
Epoch 8/10
- 57s - loss: nan
Epoch 9/10
- 58s - loss: nan
Epoch 10/10
- 63s - loss: nan
```

```
Out[15]: <keras.callbacks.History at 0x1a1d88ce10>
```

```
In [16]: # get ffnn predictions
         ffnn_preds = ffnn.predict_classes(X_test)
```

7.4 View results

```
In [19]: print("KNN Results:")
         print("accuracy: " + str(accuracy_score(y_test, knn_pred)))
         print("precision: " + str(precision_score(y_test, knn_pred, average=None)))
         print("recall: " + str(recall_score(y_test, knn_pred, average=None)))
         print()
         print("RF Results:")
         print("accuracy: " + str(accuracy_score(y_test, rf_pred)))
         print("precision: " + str(precision_score(y_test, rf_pred, average=None)))
         print("recall: " + str(recall_score(y_test, rf_pred, average=None)))
         print()
         print("FFNN Results:")
         print("accuracy: " + str(accuracy_score(y_test, ffnn_preds)))
         print("precision: " + str(precision_score(y_test, ffnn_preds, average=None)))
         print("recall: " + str(recall_score(y_test, ffnn_preds, average=None)))
```

```
KNN Results:
accuracy: 0.6353007575722814
precision: [0.50947136 0.50497735 0.70702975]
recall: [0.46252035 0.39264124 0.78957387]

RF Results:
accuracy: 0.6187023840660508
precision: [0.49549426 0.47101407 0.69487956]
recall: [0.43826491 0.38092479 0.77402013]

FFNN Results:
accuracy: 0.2137705931027732
precision: [0.21376999 1.         0.        ]
recall: [1.0000000e+00 3.5984167e-06 0.0000000e+00]
```

7.5 Discussion

One contribution to the mediocre results could be the fact that there are significantly more "no change" observations than the other two.

```
In [18]: np.bincount(y_train.values)

Out[18]: array([ 839331,  833825, 2249637])
```