bankCustomerSatisfaction-Data-Analysis / Yesh.ipynb

Task 1: Data Preperation

We load the file from CSV open format to a pandas data frame for ease of visualisation and data manipulation

In [465…

```
# ~
# all required imports to perform the ta
import pandas as pd
import pytest
from sklearn.model_selection import trai
from sklearn.feature_extraction.text imp
from sklearn.naive_bayes import Multinon
from sklearn.linear_model import Logisti
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_sco
from textblob import TextBlob
```

In [466…

```
# ~
# Load tab separated values from Bank ta
csv_path = 'bank-data/bank-tabular.csv'
df_bankdata = pd.read_csv(csv_path, sep=

df_bankdata
```

Out[466…

	customer_id	date	customer_gender
0	216604	2022-08-22	Male
1	259276	2022-11-23	Female
2	265459	2022-01-21	Female
3	58770	2022-03-13	f
4	318031	2022-08-08	Female
...	...	...	...
2995	322582	2021-09-23	Male
2996	53418	2021-03-07	f
2997	79364	2021-08-01	m
2998	371134	2021-06-25	m
2999	109281	2022-10-04	Male

3000 rows × 18 columns

In [467…

```
# ~
# Load tab separated values Bank comment
csv_path = 'bank-data/bank-comments.csv'
df = pd.read_csv(csv_path, sep='\t')

df
```

Out[467…

	customer_id	date	comments
0	216604	2022-08-22	Overal, this bank is satisfactory.
1	259276	2022-11-23	Easy to find zhe bank ' s branches and ATMs. A...
2	265459	2022-01-21	Bank's phone app is really great. In general a...
3	58770	2022-03-13	NaN
4	318031	2022-08-08	NaN
...	...	...	...
2995	322582	2021-09-23	No comment
2996	53418	2021-03-07	Online banking is really good
2997	79364	2021-08-01	customer service quality from this bank is ter...
2998	371134	2021-06-25	Great to see that my bank supports local sport...
2999	109281	2022-10-04	The bank ' a online platform is really impress...

3000 rows × 3 columns

Philosophical Choice

Do not shy away from a the unknown, face it head on. It is easy to work with complete data where we just drop all rows that contain NaN values. In the real world however machine learning models must be able to deal with the learning models must be able to deal with the difficulties of incomplete data. Also we do not have a very large data set. We will lose key information if we start dropping rows containing NaN values and will not have much data left to work with. Also the final model we produce will be superior as it has the capabilities of predicting satisfaction, even given incomplete data!

Dealing with null values for bank comments

In [468…

```
# ~
# Bank comments clean up - replace NaN's
df['comments'] = df['comments'].fillna('

df
```

Out[468…

	customer_id	date	comments
0	216604	2022-08-22	Overal, this bank is satisfactory.
1	259276	2022-11-23	Easy to find zhe bank ' s branches and ATMs. A...
2	265459	2022-01-21	Bank's phone app is really great. In general a...
3	58770	2022-03-13	neutral
4	318031	2022-08-08	neutral
...	...	...	...
2995	322582	2021-09-23	No comment
2996	53418	2021-03-07	Online banking is really good
2997	79364	2021-08-01	customer service quality from this bank is ter...
2998	371134	2021-06-25	Great to see that my bank supports local sport...
2999	109281	2022-10-04	The bank ' a online platform is really impress...

3000 rows × 3 columns

In [469…

```
# ~
```

```
"
for index, row in df.iterrows():
    comment = row["comments"]

    # Analyze text sentiment https://te
    blob = TextBlob(comment)
    sentiment_score = blob.sentiment.pol

    # append sentiment score to our data
    df.at[index, "satisfaction"] = senti
```

In [470…

```
# ~
df_comments = df
df_comments
```

Out[470…

	customer_id	date	comments	satisf
0	216604	2022-08-22	Overal, this bank is satisfactory.	0.0
1	259276	2022-11-23	Easy to find zhe bank ' s branches and ATMs. A...	0.6
2	265459	2022-01-21	Bank's phone app is really great. In general a...	0.5
3	58770	2022-03-13	neutral	0.0
4	318031	2022-08-08	neutral	0.0
...	...	...	...	
2995	322582	2021-09-23	No comment	0.0
2996	53418	2021-03-07	Online banking is really good	0.7
2997	79364	2021-08-01	customer service quality from this bank is ter...	-0.6
2998	371134	2021-06-25	Great to see that my bank supports local sport...	0.3
2999	109281	2022-10-04	The bank ' a online platform is really impress...	0.5

3000 rows × 4 columns

Dealing with Null Values for bank data

We must first find all unique values as for example with gender there are many kinds of potential NaN values such as NaN and Not specified. We can Also deal with age by creating buckets(ranges) and modifying that column

In [471…

```
# ~
# get all unique possibilities to clean
print(df_bankdata['customer_gender'].uni
print(df_bankdata['customer_age'].unique
print(df_bankdata['customer_location'].u
print(df_bankdata['customer_type'].uniqu
print(df_bankdata['convenience'].unique(
```

```
['Male' 'Female' 'f' 'Unspecified' nan
'm' 'Not specified']
[50. 61. 63. nan 41. 71. 40. 46. 65. 69.
56. 51. 52. 54. 31. 35. 32. 43.
 42. 55. 29. 60. 53. 62. 72. 27. 24. 49.
67. 73. 57. 37. 34. 78. 45. 59.
 19. 39. 75. 44. 58. 23. 48. 64. 68. 47.
30. 22. 76. 18. 33. 26. 28. 36.
 92. 66. 77. 79. 21. 20. 25. 70. 81. 74.
38. 80. 88. 83. 82. 91. 86. 84.]
['Munster' 'Leinster' nan 'Connacht' 'Uls
ter']
['Personal' 'Business' 'Business-Plus']
[ 4.  5.  2. nan  1.  3.]
```

Hot encode variables

Here we hot encode gender. 1 for man, 0 for woman and 2 for unknown

In [472…

```
# ~
# # replace all NaN values with appropia
# df_bankdata['customer_gender'] = df_ba

# # hot encode gender as a numerical val
# # replace multiple values in the 'genc
# df_bankdata['customer_gender'] = df_ba
# replace all NaN values in 'customer_ge
df_bankdata['customer_gender'].fillna('U

# hot encode gender
df_bankdata['customer_gender'] = df_bank
    'Male': 1,
    'Female': 0,
    'f': 0,
    'm': 1,
    'Unspecified': 2,
    'Not specified': 2
}).astype('category').cat.codes
df_bankdata['customer_gender']
```

```
# verify that only encoded values presen
print(df_bankdata['customer_gender'].uni
```

```
[1 0 2]
```

Hot Encoding date using buckets

This is easy to do once we get the range of date values. After this we no longer need the original date column as we have date_encoded as a column

In [473…

```
# ~
oldest_date = df_bankdata['date'].min()
newest_date = df_bankdata['date'].max()

df_bankdata['date'] = pd.to_datetime(df_

# create buckets (ranges)
buckets = pd.date_range(start=oldest_dat
# label buckets
bucket_labels = [f" {i+1}" for i in rang

# map date to bucket
def map_to_bucket(date):
    for i, bucket in enumerate(buckets):
        if date <= bucket:
            return bucket_labels[i]
    return bucket_labels[-1]

df_bankdata['date_Encode'] = df_bankdata

# Print all unique date bucket values to
print(df_bankdata['date_Encode'].unique(

#  now we can drop date as it is encoded
# drop the 'column_to_drop' column
df_bankdata.drop('date', axis=1, inplace
```

```
[' 20' ' 23' ' 13' ' 15' ' 1' ' 22' ' 9'
' 2' ' 6' ' 5' ' 4' ' 18' ' 11'
 ' 24' ' 19' ' 8' ' 3' ' 10' ' 16' ' 17'
' 7' ' 12' ' 21' ' 14']
```

In [474…

```
df_bankdata
```

Out[474…

	customer_id	customer_gender	custom
0	216604	1	
1	259276	0	
2	265459	0	
3	58770	0	
4	318031	0	
...	...	...	
2995	322582	1	
2996	53418	0	
2997	79364	1	
2998	371134	1	

2999	109281	1

3000 rows × 18 columns

Deciphering how best to replace null values

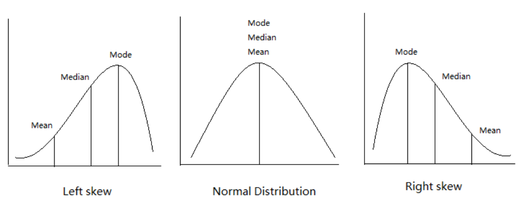
Since the mean, median and mode are close together we can see normal distribution of customer age. Thus, a good way to solve nan age values is to replace it with the mean average age.

In the case of location and gender we can simply hot encode string values.

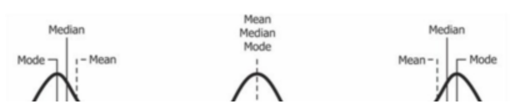
How null values were replaced and why

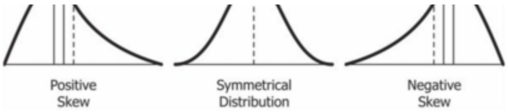
All the data appears to have Platykurtic distributions. It was this decided to use the median value to replace null values. Alternatively some features used hot encoding were null values got their own category. This was done in the case of gender. You can not get the median gender. Only the mode. But if all NaNs were assigned to the most often present gender it would skew the data set and create bias.

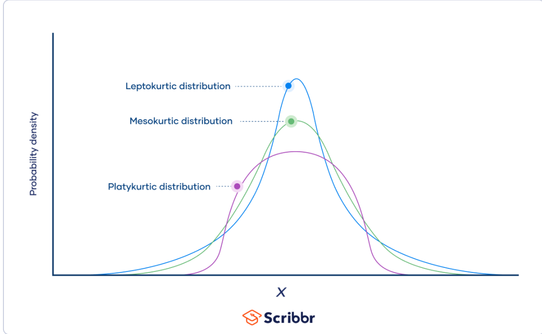
Since the mean, median and mode are close together we can see normal distribution of customer age. Thus, a good way to solve nan age values is to replace null values with the mean average age. Age is then later hot encoded to bucket ranges.



Since the mean, median and mode are close together we can see normal distribution of customer age. Thus, a good way to solve nan age values is to replace it with the mean average age.







reference -> https://www.scribbr.com/statistics/kurtosis/

In [475…

```
# ~
# test to find out what to do with custo
df_bankdata
median_age = df_bankdata['customer_age']
mean_age = df_bankdata['customer_age'].m
mode_age = df_bankdata['customer_age'].m
print(mean_age, median_age, mode_age)
# Since the mean, median and mode are cl
# Thus, a good way to solve nan age valu
```

```
47.281818181818 48.0 0    48.0
Name: customer_age, dtype: float64
```

In [476…

```
# ~
# Replace NaN age values with the mean
df_bankdata['customer_age'] = df_bankdat
print(df_bankdata['customer_age'].unique
```

```
[50. 61. 63. 48. 41. 71. 40. 46. 65. 69.
56. 51. 52. 54. 31. 35. 32. 43.
 42. 55. 29. 60. 53. 62. 72. 27. 24. 49.
67. 73. 57. 37. 34. 78. 45. 59.
 19. 39. 75. 44. 58. 23. 64. 68. 47. 30.
22. 76. 18. 33. 26. 28. 36. 92.
 66. 77. 79. 21. 20. 25. 70. 81. 74. 38.
80. 88. 83. 82. 91. 86. 84.]
```

In [477…

```
# ~
# fill NaN values in 'customer_location
df_bankdata['customer_location'].fillna(

# hot encode location
df_bankdata['customer_location'] = df_ba

df_bankdata['customer_location'] += 1
print(df_bankdata['customer_location'].u
```

```
[3 2 5 1 4]
```

In [478…

```
df_bankdata['customer_type'] = df_bankda
```

Replacing NaNs with median values here

In [479…

```
# NOTE : All NaNs were replaced with zei

# convenience
# Platykurtic distributions close to noi
# mean 2.654 median 3.0 mode 2.0
# skewness value:  0.08700482420970669 l
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_convenience = df_bankdata['conver
df_bankdata['convenience'] = df_bankdata

# service
# Platykurtic distributions close to noi
# mean:  2.7556666666666665  median:  3.
# skewness value:  0.08843472512112084 l
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_customer_service = df_bankdata['c
df_bankdata['customer_service'] = df_bai

# banking online
# Platykurtic distributions close to noi
# mean:  3.108  median:  3.0  mode: 4.0
# skewness value:  -0.2667127993100813 l
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_onlineBanking = df_bankdata['onli
df_bankdata['online_banking'] = df_bankc

# Interest rates
# Platykurtic distributions close to noi
# mean:  2.9893333333333333  median:  3.(
# skewness value:  -0.40820391614908896
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_interest_rates = df_bankdata['int
df_bankdata['interest_rates'] = df_bankc

# Fees
# Platykurtic distributions close to noi
# mean:  3.0603333333333333  median:  3.
# skewness value:  -0.27529328811942694
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_fees_charges = df_bankdata['fees_
df_bankdata['fees_charges'] = df_bankdat

# community
# Platykurtic distributions close to noi
# mean:  2.853666666666667  median:  3.(
# skewness value:  -0.3304467191614403 l
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_community_involvement = df_bankda
df_bankdata['community_involvement'] = c

# interest rates
# Platykurtic distributions close to noi
# mean:  3.2653333333333334  median:  4.
# skewness value:  -0.5872697715455386 l
# Findings - In this case the median is
# Thus we will replace Nan values with i
median_products_services = df_bankdata['
df_bankdata['products_services'] = df_ba

# interest rates
# Platykurtic distributions close to noi
# mean:  3.095666666666667  median:  3.(
# skewness value:  -0.5799718813422435 l
# Findings - In this case the median is
```

```
# Findings - In this case the median is
# Thus we could replace Nan values with
median_privacy_security = df_bankdata['p
df_bankdata['privacy_security'] = df_ban

# interest rates
# Platykurtic distributions close to nor
# mean:  2.678  median:  3.0  mode:  4.(
# skewness value:  -0.2296510500674088 F
# Findings - In this case the median is
# Thus we will replace Nan values with r
median_reputation = df_bankdata['reputat
df_bankdata['reputation'] = df_bankdata[
```

Hot encoding age using buckets

In [480…

```
youngest_age = df_bankdata['customer_age
oldest_age = df_bankdata['customer_age']


# define the age range and bin sizes
age_range = range(int(youngest_age), int
bin_size = 10

bins = pd.interval_range(start=youngest_

df_bankdata['age_bucket'] = pd.cut(df_ba

# convert age to buckets and hot encode
df_bankdata['age_bucket'] = df_bankdata[
```

In [481…

```
df_bankdata
```

Out[481…

	customer_id	customer_gender	custom
0	216604	1	
1	259276	0	
2	265459	0	
3	58770	0	
4	318031	0	
...	...	...	
2995	322582	1	
2996	53418	0	
2997	79364	1	
2998	371134	1	
2999	109281	1	

3000 rows × 19 columns

Task 2: Data Characterisation

Here we analyse what features we want to select. Straight away I got a hunch that comment sentiment (the satisfaction column) would be a key feature because it accurately describes how the customer feels. I saw comments such as "this bank is good", which would very likely mean they are satisfied with the bank.

The way in which features selection was done and key features were identified was two fold.

1. Building up feattures (start with no features present and build up to see what is key)
2. Tearing down features (start with all features present and cut down to see what is not important and just junk noise)
3. Creating a new enhanced feature out of existing ones - For example mixing interest rates and fees changes in to one feature

By guessing what features were important such as online_banking and then adding it to the model we can build up a picture of what is important. If it improved accuracy that feature stayed as part of the model. If not is it disregarded.

If we blindly train our model using a simple train test split without feature selection, we get weak accuracy of only 58%

references

1. https://datagy.io/sklearn-train-test-split/
2. https://realpython.com/train-test-split-python-data/

In [482…

```
# Split the data into training and testi
X_train, X_test, y_train, y_test = trair

# Create a logistic regression model anc
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict the customer satisfaction on t
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_prec
print('Accuracy:', accuracy)
```

```
Accuracy: 0.5833333333333334
```

There is an issue with this method of testing. The problem is that the data set is split in to train and test and may have random differences based on the split. i.e. not an even sample.

In [483…

```
# now lets try be more specific with ou
df_reduced = df_bankdata

# has_cc is not something we expect to a
df_reduced = df_reduced.drop(['has_cc',
```

In [484…

```
df_reduced
```

Out[484…

	customer_gender	customer_age	custo
0	1	50.0	
1	0	61.0	
2	0	63.0	
3	0	48.0	
4	0	41.0	
...	...	...	
2995	1	41.0	
2996	0	57.0	
2997	1	48.0	
2998	1	42.0	
2999	1	42.0	

3000 rows × 17 columns

In [485…

```
# try with dropped features

# Split the data into training and test
X_train, X_test, y_train, y_test = trai

# Create a logistic regression model and
model = LogisticRegression()



# Create a logistic regression model wit
model = LogisticRegression(max_iter=100
model.fit(X_train, y_train)

# Predict the customer satisfaction on t
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8383333333333334
```

In [486…

```
# df_reduced2 = df_reduced.drop(["date_l
df_reduced2 = df_reduced.drop(["age_buck
```

In [487…

```
# lets create a new feature from our ex
df_enhanced = df_reduced2
df_enhanced['total_cost'] = df_enhanced[


df_enhanced
```

Out[487…

	customer_type	has_mortgage	conveni
0	1	False	
1	1	False	
2	2	False	
3	2	False	
4	1	True	
...	...	...	
2995	1	True	
2996	2	False	
2997	1	True	
2998	2	False	
2999	2	False	

3000 rows × 14 columns

In [488…

```
df_enhanced_CommentSentiment = df_bankda

df_enhanced_CommentSentiment['total_cost
df_enhanced = df_enhanced.drop(['fees_ch
df_enhanced_CommentSentiment
```

Out[488…

	customer_type	has_mortgage	conveni
0	1	False	
1	1	False	
2	2	False	
3	2	False	
4	1	True	
...	...	...	
2995	1	True	
2996	2	False	
2997	1	True	
2998	2	False	
2999	2	False	

3000 rows × 13 columns

In [489…

```
df_comments_concise = df_comments[['cust
df_bankdata_CommentSentiment = pd.merge(
df_bankdata_CommentSentiment
df_comments_concise_forTest = pd.merge(c
df_comments_concise_forTest = df_comment
```

Task 3 and 4: Data Classification

---

k fold cross validation, test on each segment of the data set seperately and train on the remainder. This solves the problem of a basic train_test split. This solves the problem of splitting the data set in to train and test and having random differences based on the split. i.e. not an even sample. We now have a guaranteed even sample by agregating the results of each train_test split. i.e. we have cross validated the results

In [490…

```
def kfold_logreg_accuracy(df, target_col
    # Define features and target columns
    X = df.drop(target_col, axis=1)
    y = df[target_col]

    # Create a cross validator
```