Introduction to Data Analytics in Business

Bank Marketing Data Analytics and Prediction

Team 3: Jiawei Li, Strahinja Trenkic, Qiqi Zhou, Fan Jia

Frankfurt School of Finance & Management
German Excellence. Global Relevance.

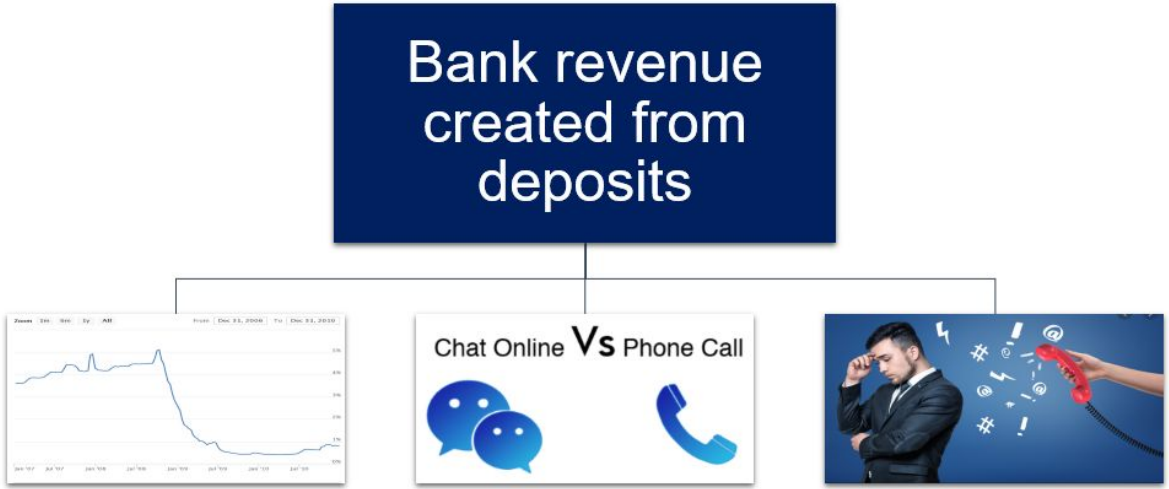
Table of Content


Chapter - 1
Introduction

1. Introduction:

Understanding the Project

- Problem statement
- Problem motivation
- Economic context
- Machine learning applications as the future of telemarketing?



1. Introduction:

Features

- Feature overview and interpretation
- Initial grouping of features
- Data types

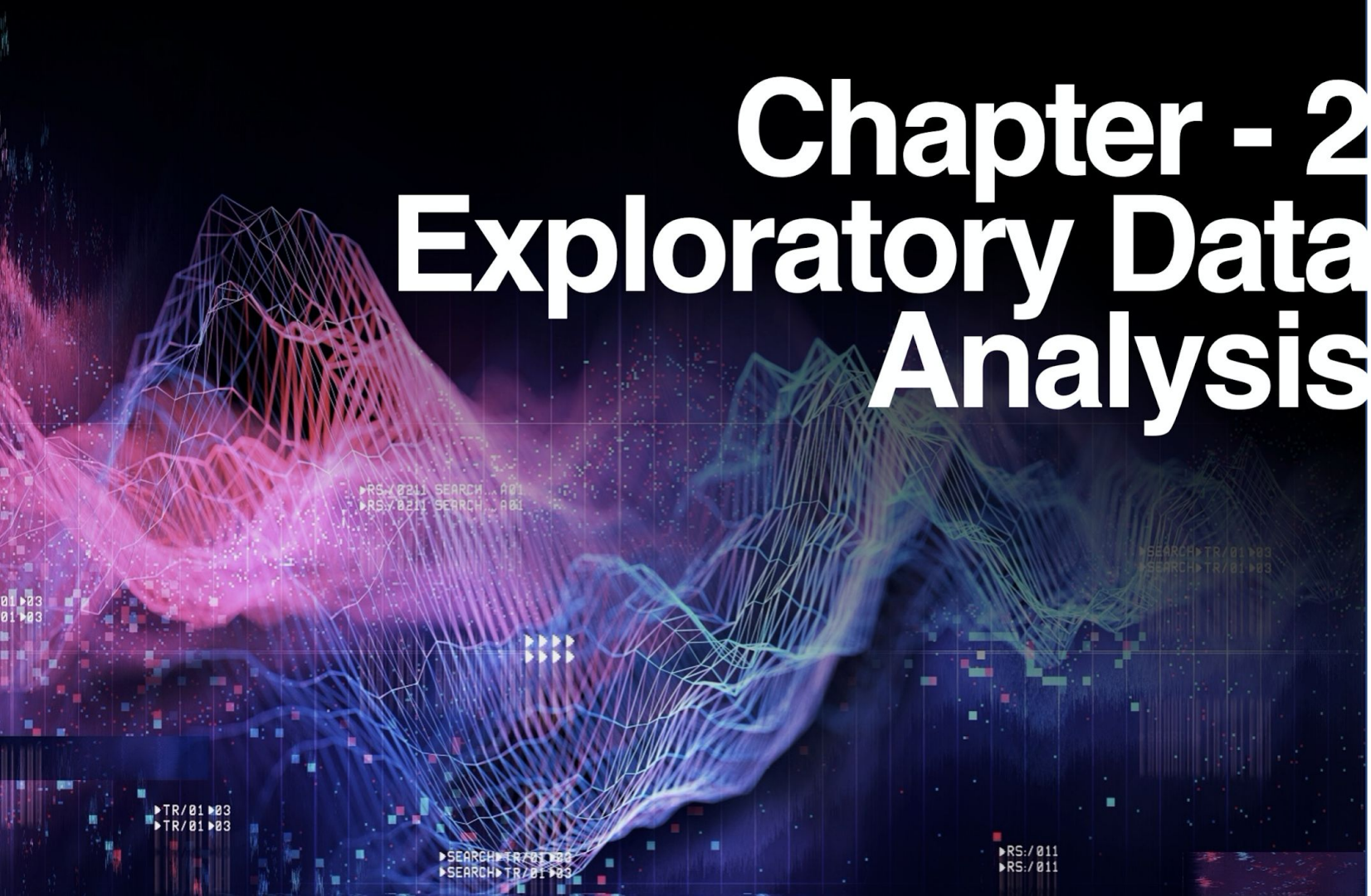
```
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  Int64
 1   job             40858 non-null  category
 2   marital         41108 non-null  category
 3   education       39457 non-null  category
 4   default         32591 non-null  boolean
 5   housing         40198 non-null  boolean
 6   loan            40198 non-null  boolean
 7   contact         41188 non-null  category
 8   month           41188 non-null  category
 9   day_of_week     41188 non-null  category
 10  duration        41188 non-null  Int64
 11  campaign        41188 non-null  Int64
 12  pdays           1515 non-null   Int64
 13  previous        41188 non-null  Int64
 14  poutcome        5625 non-null   boolean
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  boolean
dtypes: Int64(5), boolean(5), category(6), float64(5)
```

Client Data

Last Contact

Other Attributes

Social & Economic


Chapter - 2
Exploratory Data
Analysis

2. Exploratory Data Analysis:

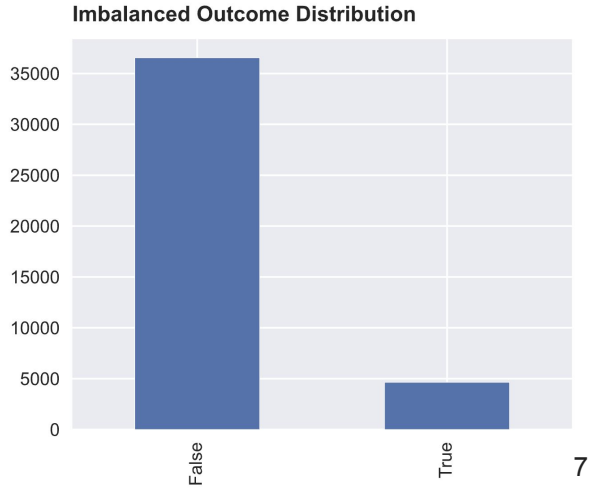
Getting Started

- Import visualization libraries
- Set cosmetic & format options
- Target variable Y: unbalanced

```
import matplotlib.pyplot as plt
import altair as alt
import seaborn as sns

# cosmetic options for matplotlib
plt.style.use("seaborn")  #use seaborn style
plt.rcParams["figure.figsize"] = (5,5)  #adjust graph size
plt.rcParams["figure.dpi"] = 300  #adjust graph resolution
plt.rcParams["axes.titleweight"] = "bold"  #adjust font weight of title
plt.rcParams["axes.titlepad"] = 10.0 #adjust pad between axes and title in points
plt.rcParams["axes.titlelocation"] = "left" #ajust titile position

#set format of graph created by matplotlist to scalable vector graphic file
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("svg")
```

```
#visualise Y distribution
y_count = bank_mkt["y"].value_counts().plot(
     kind = "bar",
     title="Imbalanced Outcome Distribution")

#count total number of y values/rows
print(bank_mkt["y"].count())

#check % of 'yes'
print(bank_mkt["y"].sum()/bank_mkt["y"].count())
```

```
41188
0.11265417111780131
```

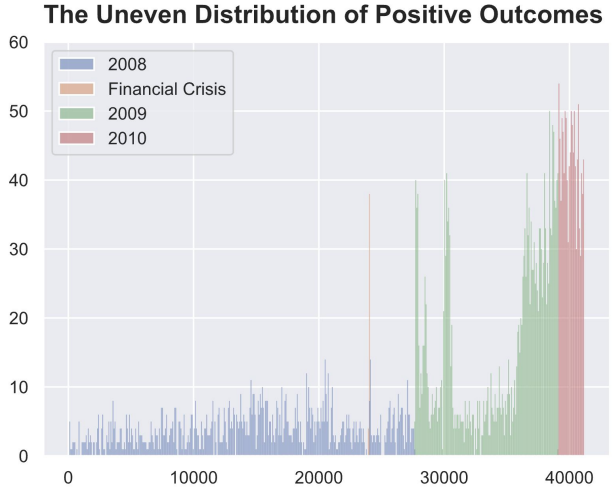


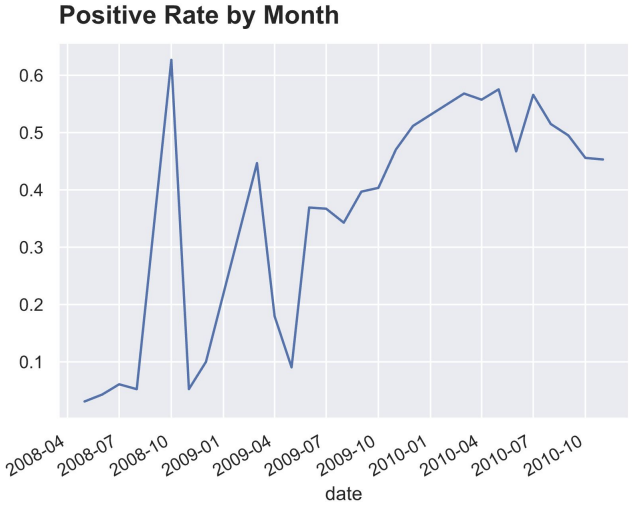
2. Exploratory Data Analysis:

Positive Outcome

- Data collected during 2008 financial crisis
- Visualize the uneven distribution of positive Y values
- Surge in campaign success since the financial crisis
- Steady growth of success rate since 2009-07

```
#visualize positive ourcome across years
bank_mkt["year"]=2008
bank_mkt.loc[27682:, "year"] = 2009
bank_mkt.loc[39118:, "year"] = 2010
bank_mkt["date"] = pd.to_datetime(bank_mkt[["month", "year"]].assign(day=1))
p = bank_mkt[bank_mkt.y == True].reset_index()
p.loc[(p.month == 10) & (p.year==2008), "year"] = "Financial Crisis"
ax = sns.histplot(data=p, x="index", stat="count", hue="year", bins=500, palette="deep", legend=True)
ax.get_legend().set_title("")
ax.set_ylim(0,60)
ax.set(title="The Uneven Distribution of Positive Outcomes", xlabel="", ylabel="");

#visualize positive outcome rate across months
bank_mkt[["date", "y"]].groupby("date").mean().plot.line(ylabel="", title="Positive Rate by Month", legend=False);
```

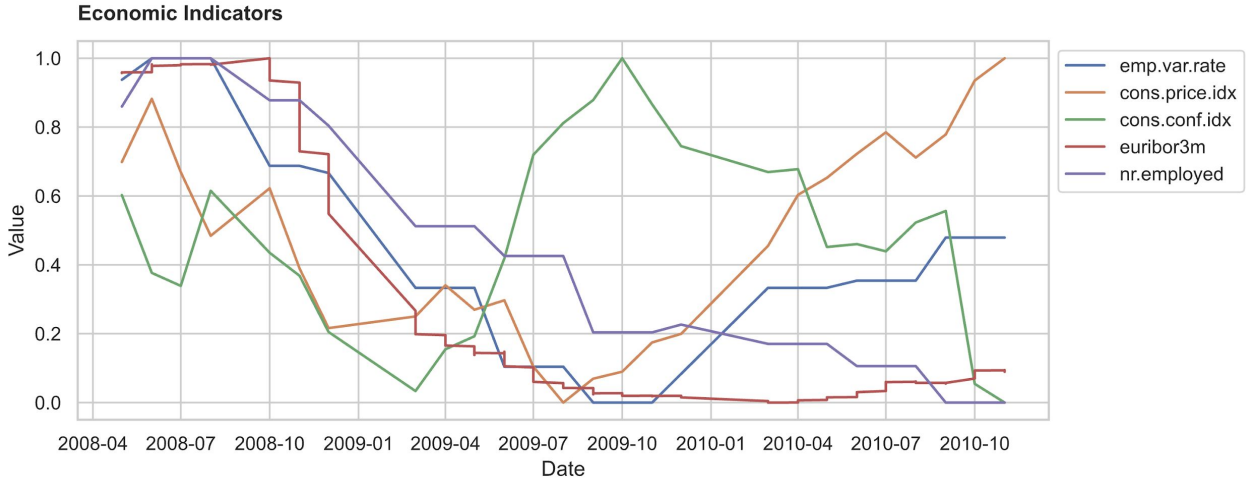


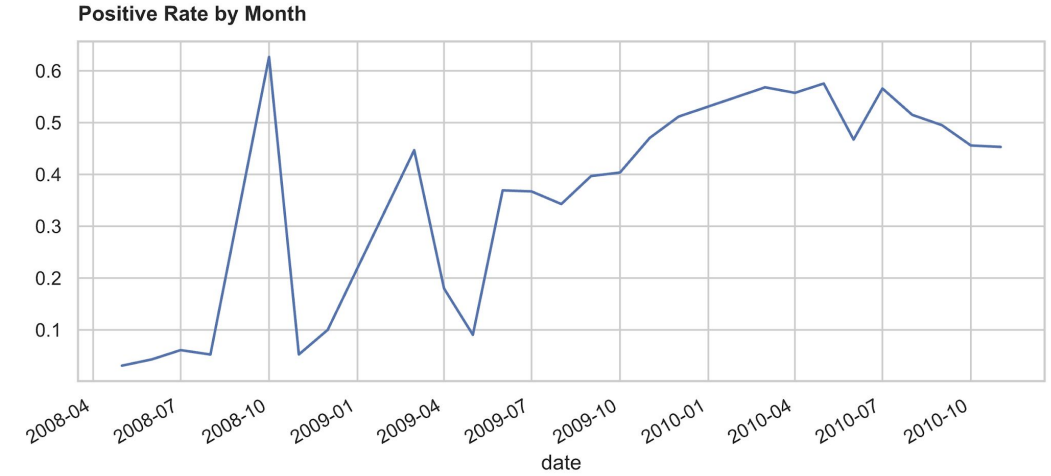


2. Exploratory Data Analysis:

Economic Indicators

- Highly relevant to the crisis
- Significant predicting power
- Relationships with positive rate
- Interest rate





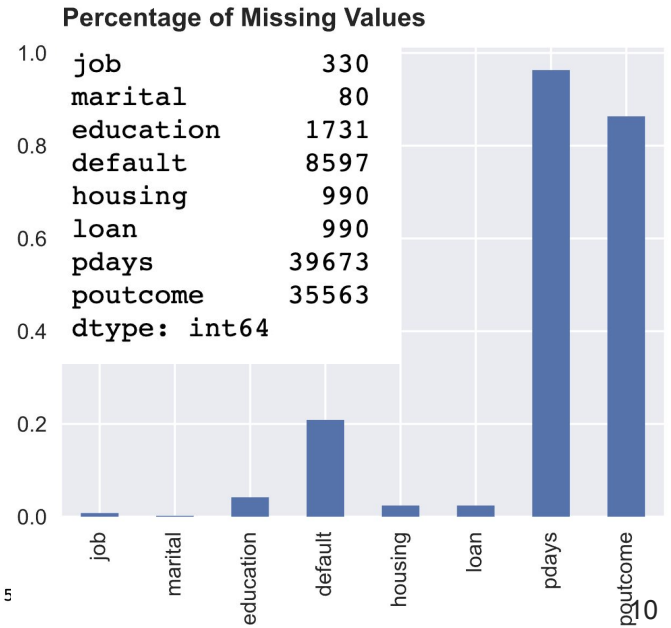
2. Exploratory Data Analysis:

Missing Values

- A lot of missing values!
- Especially Pdays & Poutcome
- Challenge for feature engineering

```
#use info() get info about a df including the index dtype and columns, non-null values and memory usage.
bank_mkt.info()

#show all features with Na value
na = bank_mkt.isna().sum()  #sum the number of na value for each feature
na_nonzero = na[na != 0]  #kick out potential na_value==0
na_nonzero

#visualize the percentage of Na values in each feature
na_perc = na_nonzero/bank_mkt.y.count()  #'y' values holds entire row number
na_bar = na_perc.plot.bar(title="Percentage of Missing Values")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  Int64
 1   job             40858 non-null  category
 2   marital         41108 non-null  category
 3   education       39457 non-null  category
 4   default         32591 non-null  boolean
 5   housing         40198 non-null  boolean
 6   loan            40198 non-null  boolean
 7   contact         41188 non-null  category
 8   month           41188 non-null  category
 9   day_of_week     41188 non-null  category
 10  duration        41188 non-null  Int64
 11  campaign        41188 non-null  Int64
 12  pdays           1515 non-null   Int64
 13  previous        41188 non-null  Int64
 14  poutcome        5625 non-null   boolean
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  boolean
dtypes: Int64(5), boolean(5), category(6), float64(5
memory usage: 4.0 MB
```

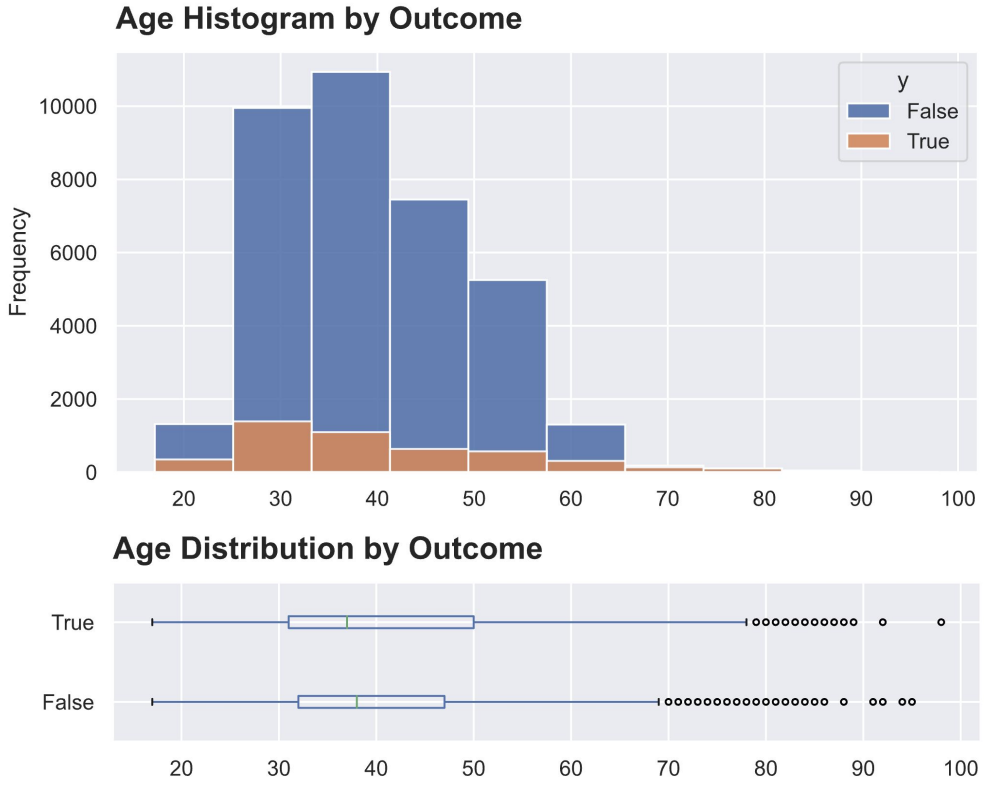


2. Exploratory Data Analysis:

Age

- Majority: 30 to 50 years old
- Higher positive rate: 30 & 60+
- No outrageous outliers

```
#visualize age distribution relative to 'y' outcome
age_y = bank_mkt[["age", "y"]].pivot(columns="y", values="age")
age_hist_outcome = age_y.plot.hist(alpha=0.6, legend=True, title="Age Histogram by Outcome")

#visualize 'age' outliers relative to 'y' outcomes
age_box_outcome = age_y.plot.box(vert=False, sym=".", title="Age Distribution by Outcome")
```

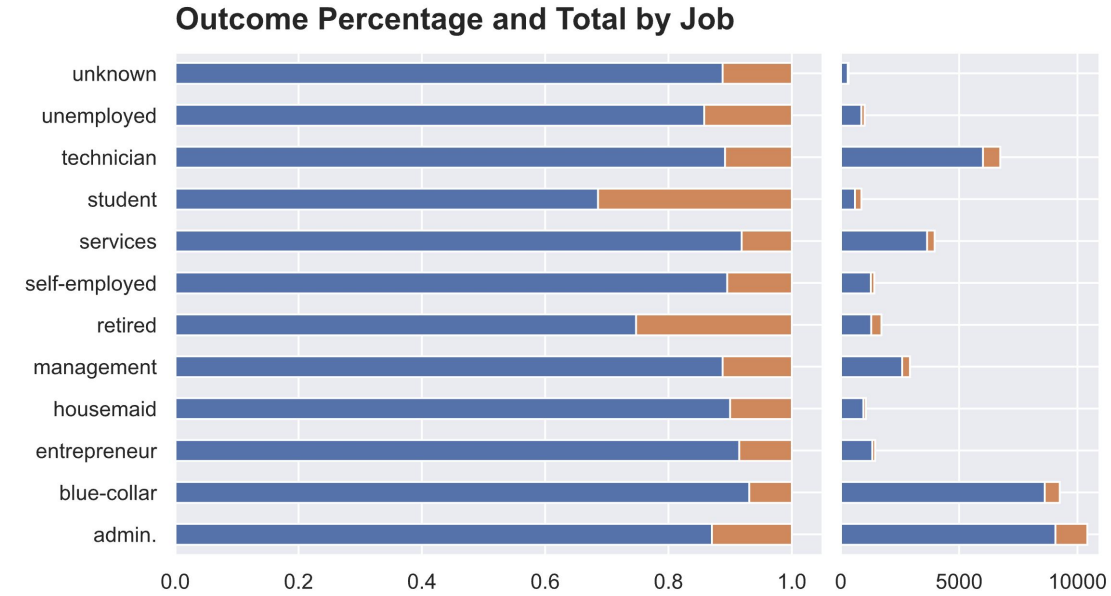


2. Exploratory Data Analysis:

Job

- Top three jobs: technician, blue-collar & admin.
- Highest positive rate: student and retired

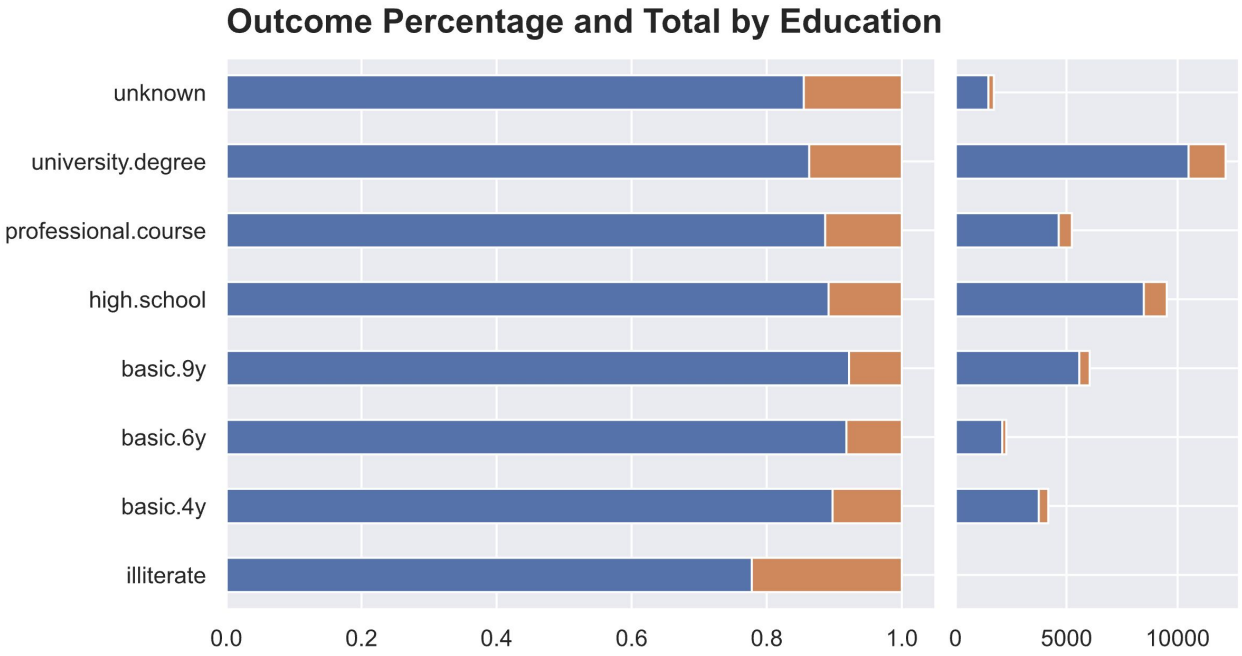
```python
def cat_outcome(df, feature):
    df = df.copy()
    if pd.api.types.is_categorical_dtype(df[feature]) and df[feature].isna().sum() > 0:
        df[feature] = df[feature].cat.add_categories("unknown")
        df[feature] = df[feature].fillna("unknown")
    title = feature.title().replace("_", " ").replace("Of", "of")
    f, axs = plt.subplots(1, 2, figsize=(8.6, 4.8), sharey=True, gridspec_kw=dict(wspace=0.04, width_ratios=[5, 2]))
    ax0 = df["y"].groupby(df[feature],
                          dropna=False).value_counts(normalize=True).unstack().plot.barh(xlabel="",
                                       legend=False, stacked=True, ax=axs[0],
                                       title=f"Outcome Percentage and Total by {title}")
    ax1 = df["y"].groupby(df[feature],
                          dropna=False).value_counts().unstack().plot.barh(xlabel="", legend=False,
                                                                           stacked=True, ax=axs[1])

job_outcome = cat_outcome(bank_mkt, "job")
```



2. Exploratory Data Analysis:

Education

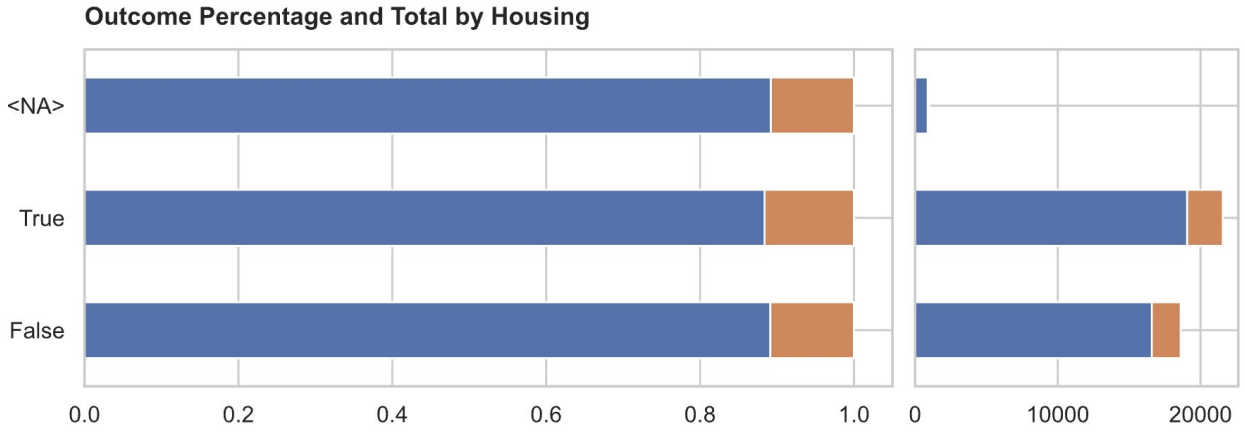
- Majority: at least high-school
- Highest positive rate: illiterate and university degree

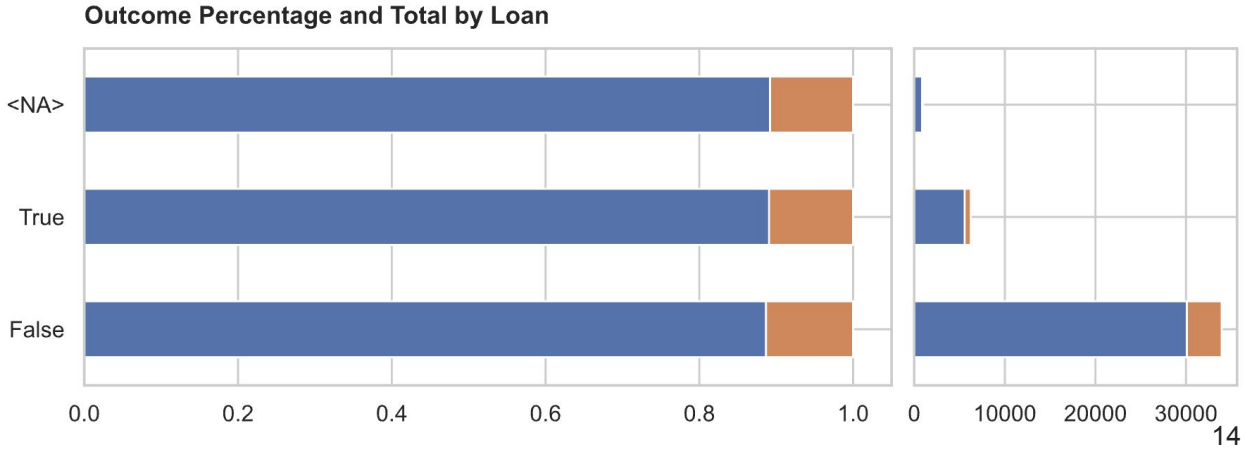


2. Exploratory Data Analysis:

Housing & Loan

- Housing: both ‘Yes’ and ‘No’ answers have large numbers
- Loan: ‘No’ dominates the numbers
- Similar outcome percentage in both features

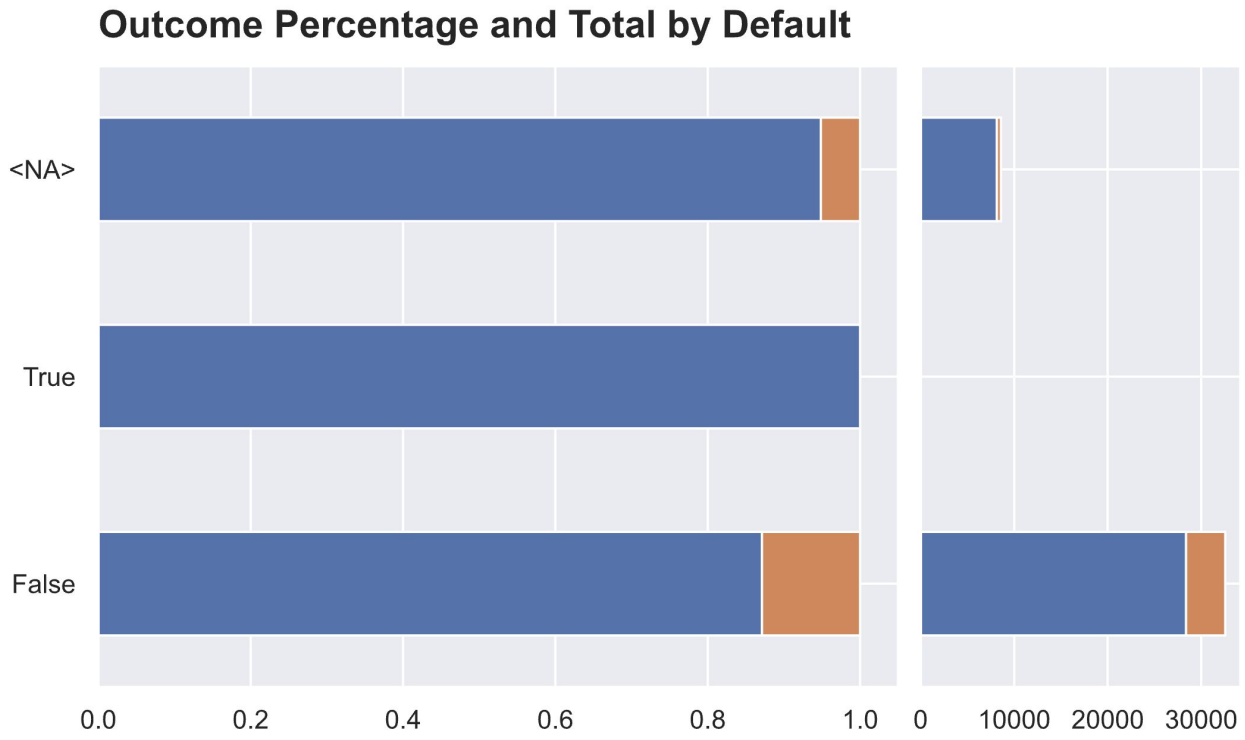




2. Exploratory Data Analysis:

Default

- Sensitive question!
- Unknown: x8600, treated as a category

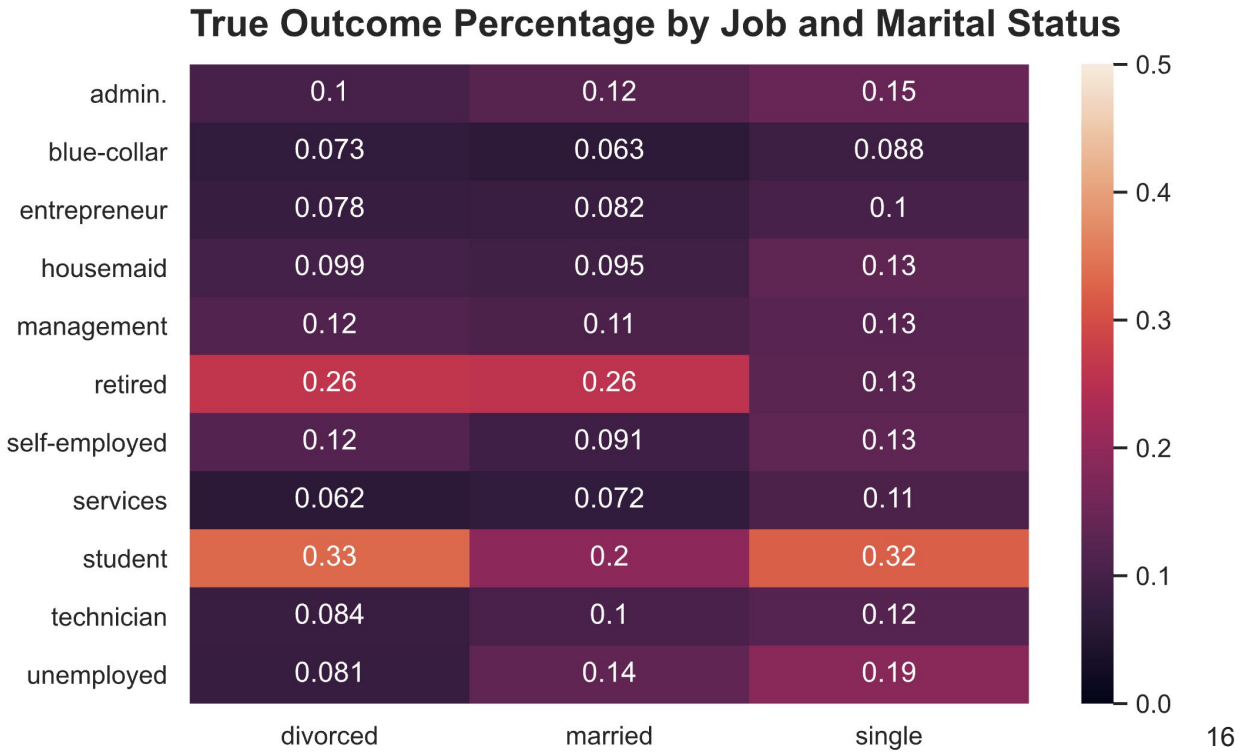


2. Exploratory Data Analysis:

Job & Marital

- Higher positive outcome rate: married and divorced retired people, and single and divorced students

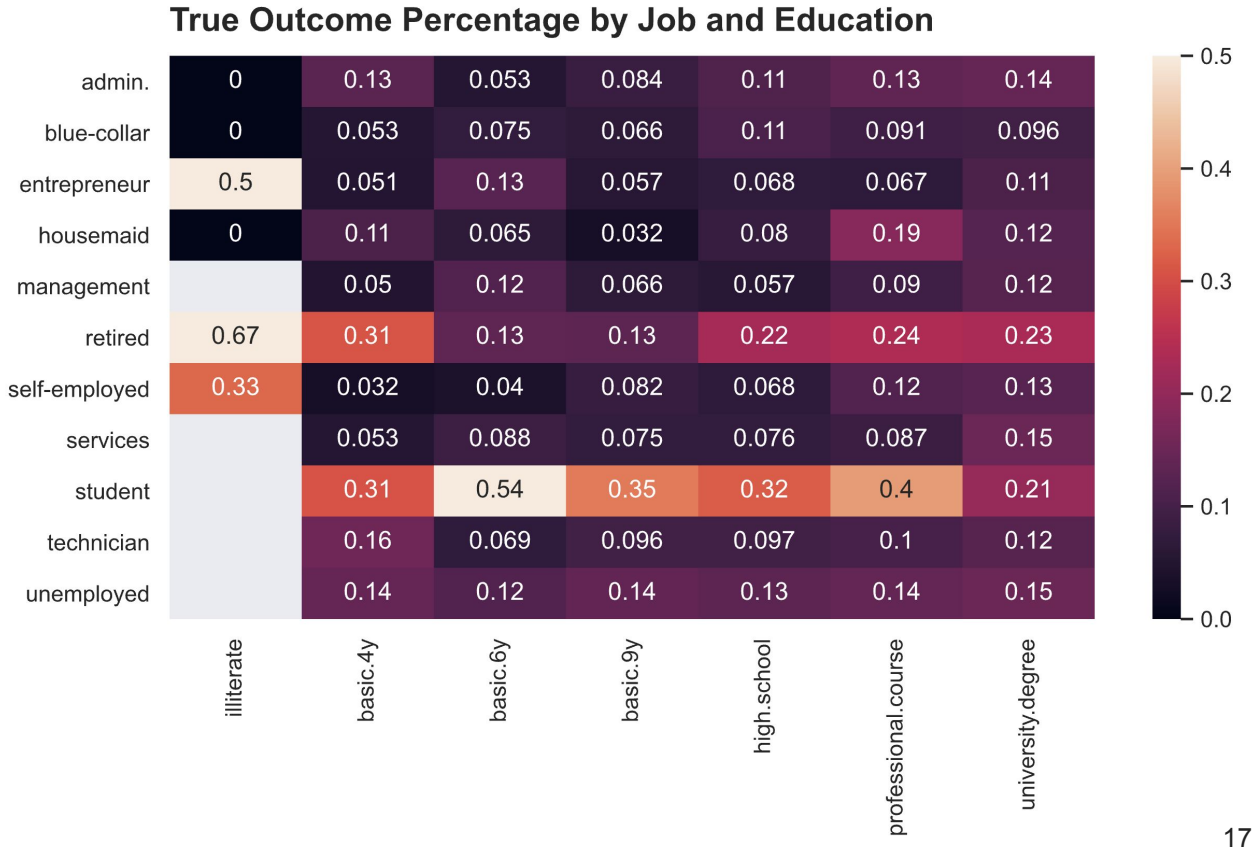
```
plt.rcParams["figure.figsize"] = (8,5)  #adjust graph size
job_marital_total = bank_mkt[["job", "marital", "y"]].groupby(["job", "marital"]).count().y.unstack()
job_marital_true = bank_mkt[["job", "marital", "y"]].groupby(["job", "marital"]).sum().y.unstack()
job_marital_rate = job_marital_true / job_marital_total
job_marital_rate = job_marital_rate.rename_axis(None, axis=0).rename_axis(None, axis=1)
job_marital_heatmap = sns.heatmap(data=job_marital_rate,
                                  vmin=0, vmax=0.5,
                                  annot=True).set_title("True Outcome Percentage by Job and Marital Status")
```



2. Exploratory Data Analysis:

Job & Education

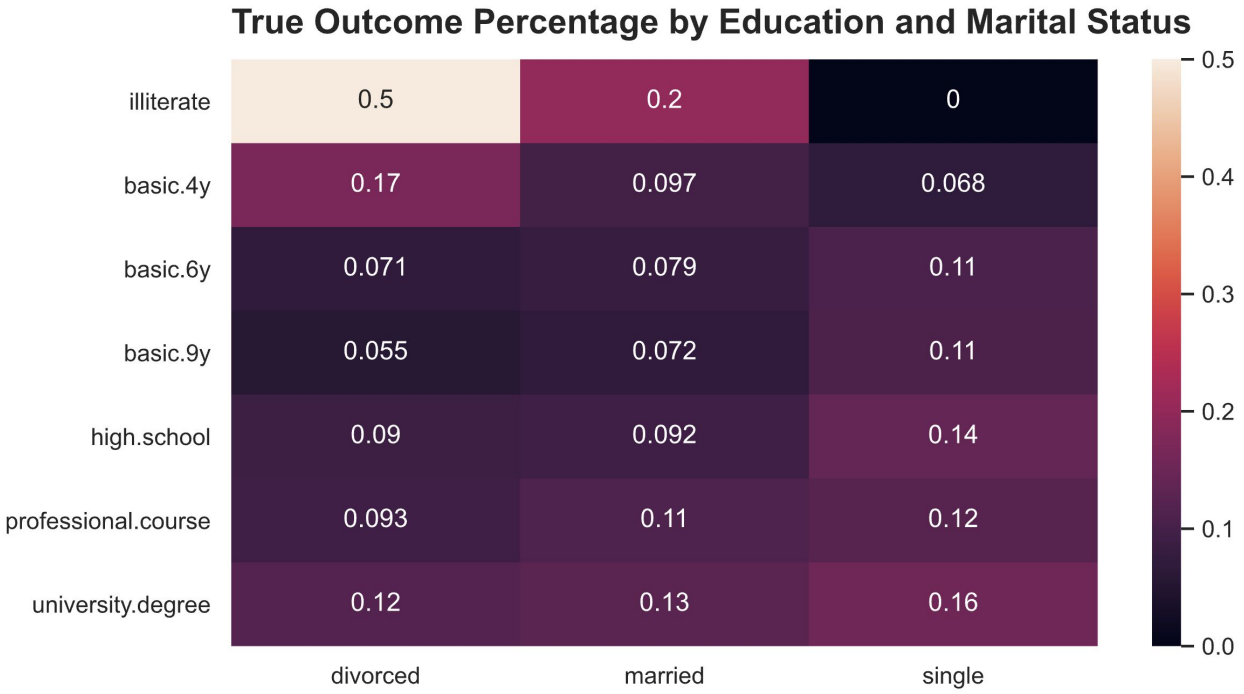
- Higher positive outcome rate: students, retired and illiterate



2. Exploratory Data Analysis:

Education & Marital

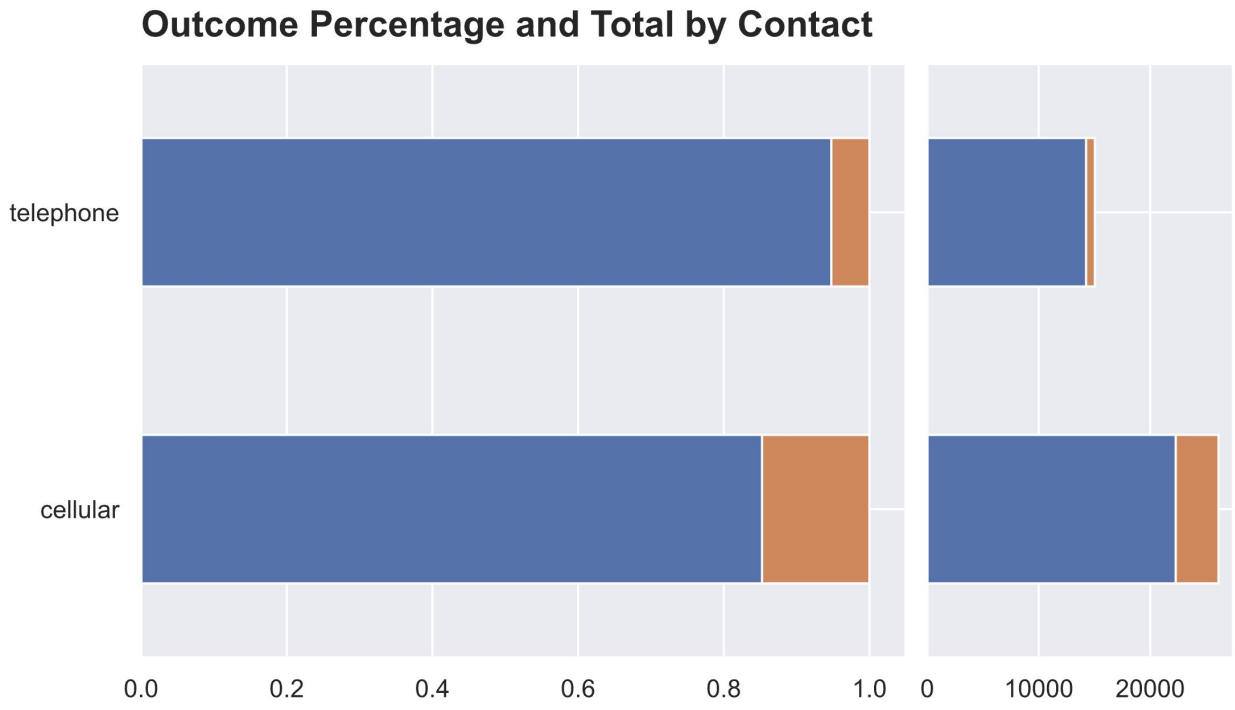
- Higher positive outcome rate: lower education, divorced



2. Exploratory Data Analysis:

Contact

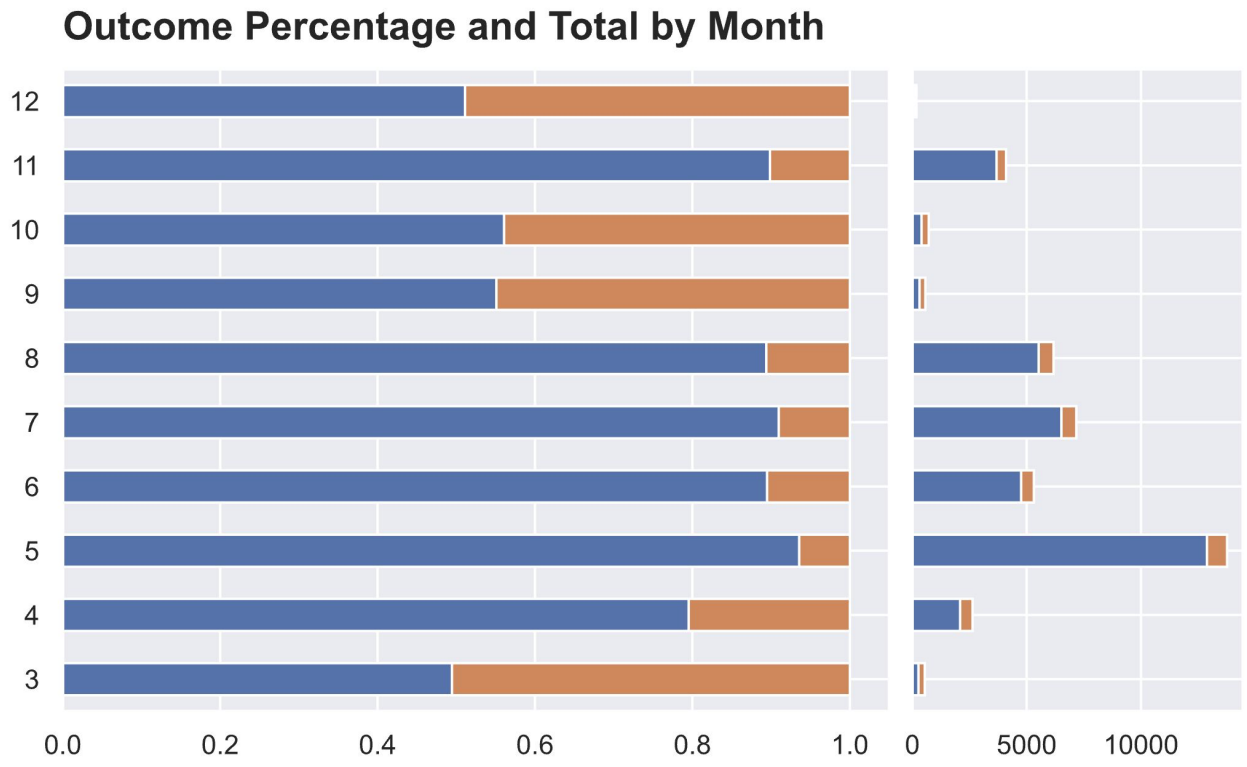
- Higher positive outcome rate: cellular



2. Exploratory Data Analysis:

Month

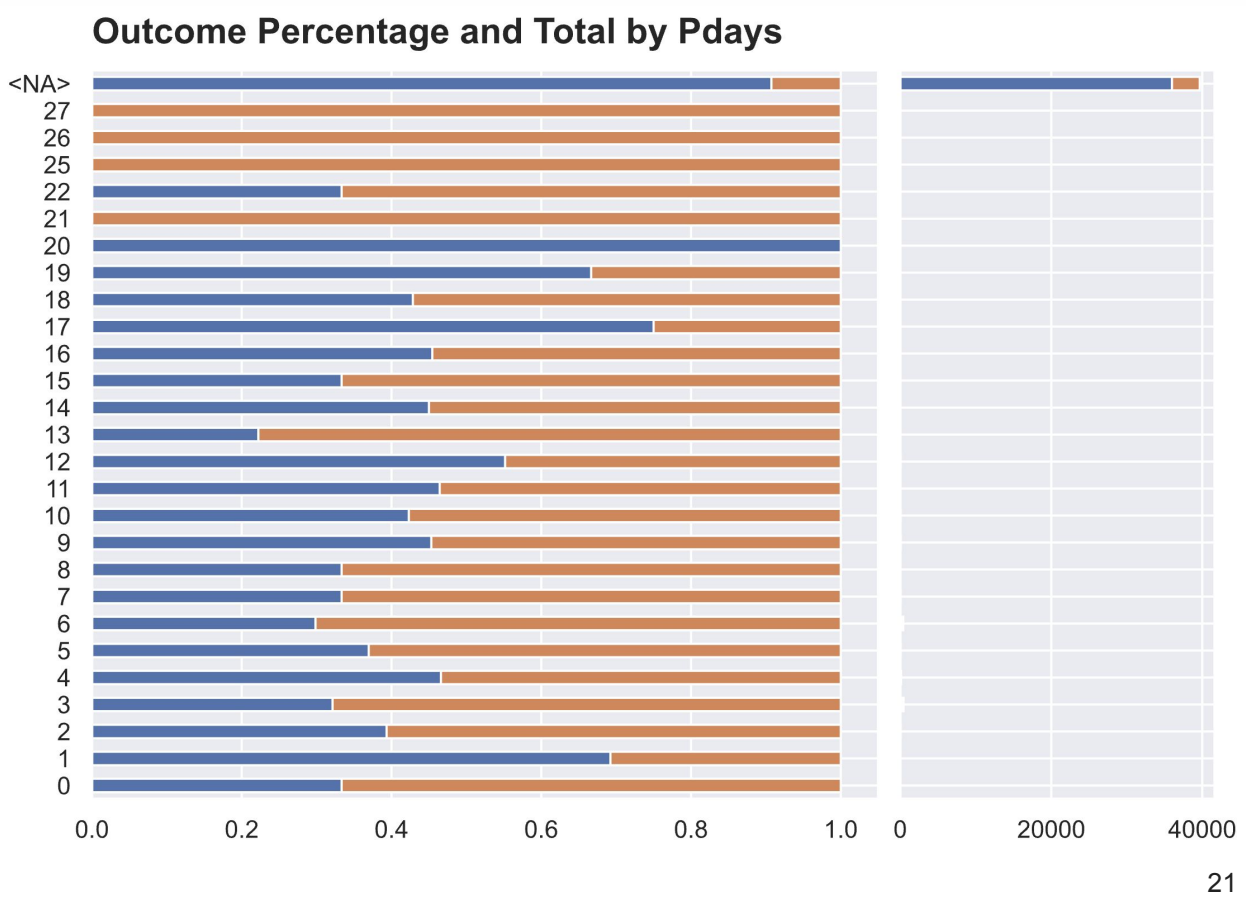
- Instances concentrate around summer time
- Higher positive outcome rate: Mar, Sep, Oct, Dec



2. Exploratory Data Analysis:

Pdays

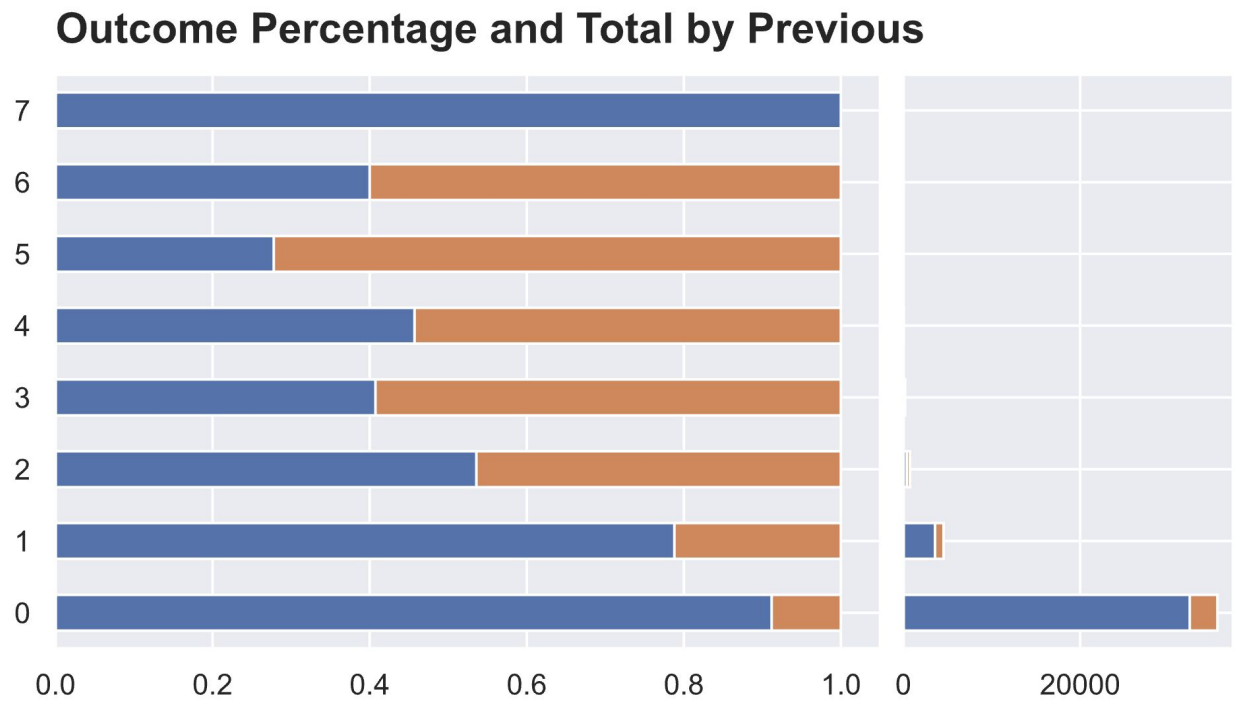
- Almost 40,000 missing values
- 1500 instances show potential predicting power
- Requires special care!



2. Exploratory Data Analysis:

Previous

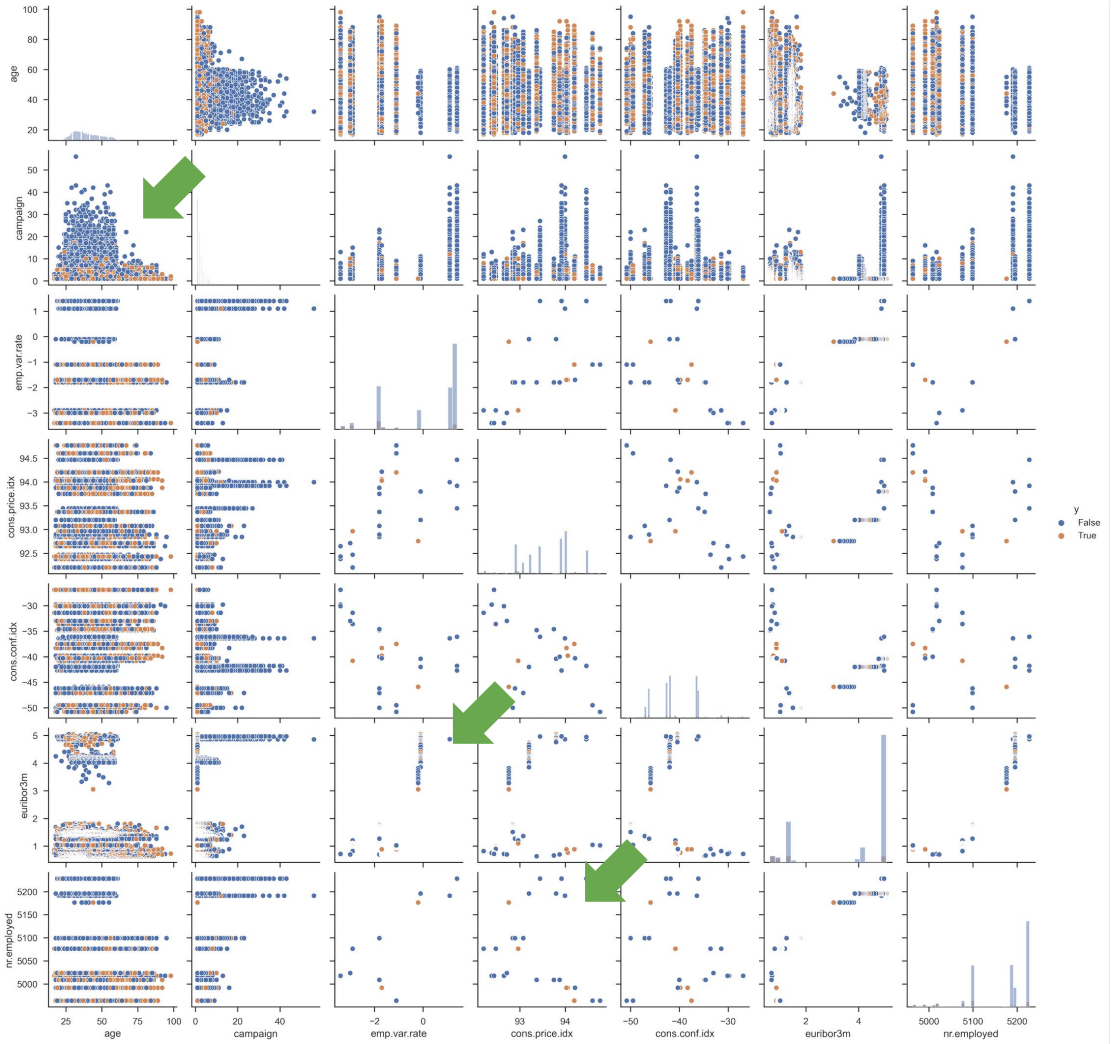
- Almost 36,000 missing values
- Remaining instances show potential predicting power
- Requires special care!



2. Exploratory Data Analysis:

Key Numerical Features

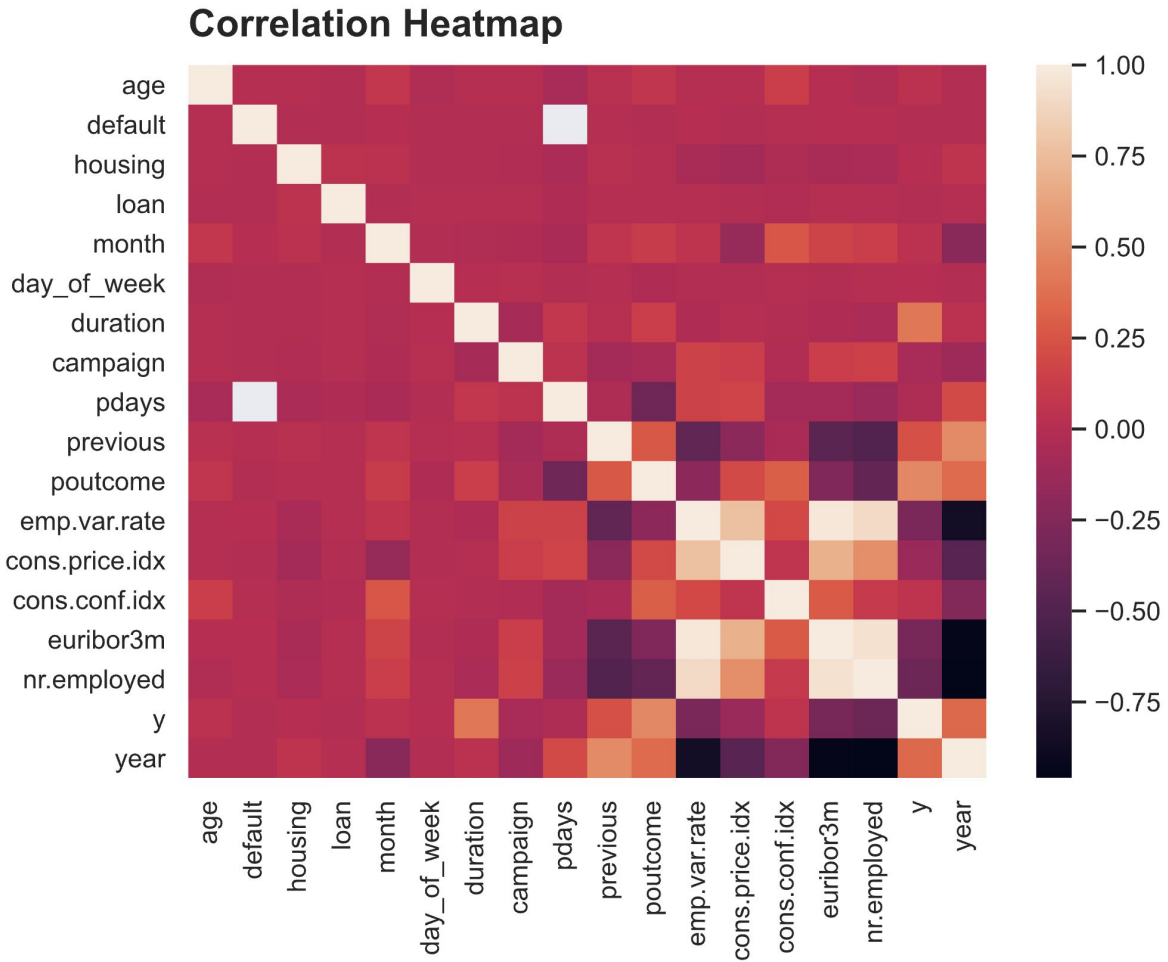
- Scatterplot with color denoting the outcome
- A few interesting findings

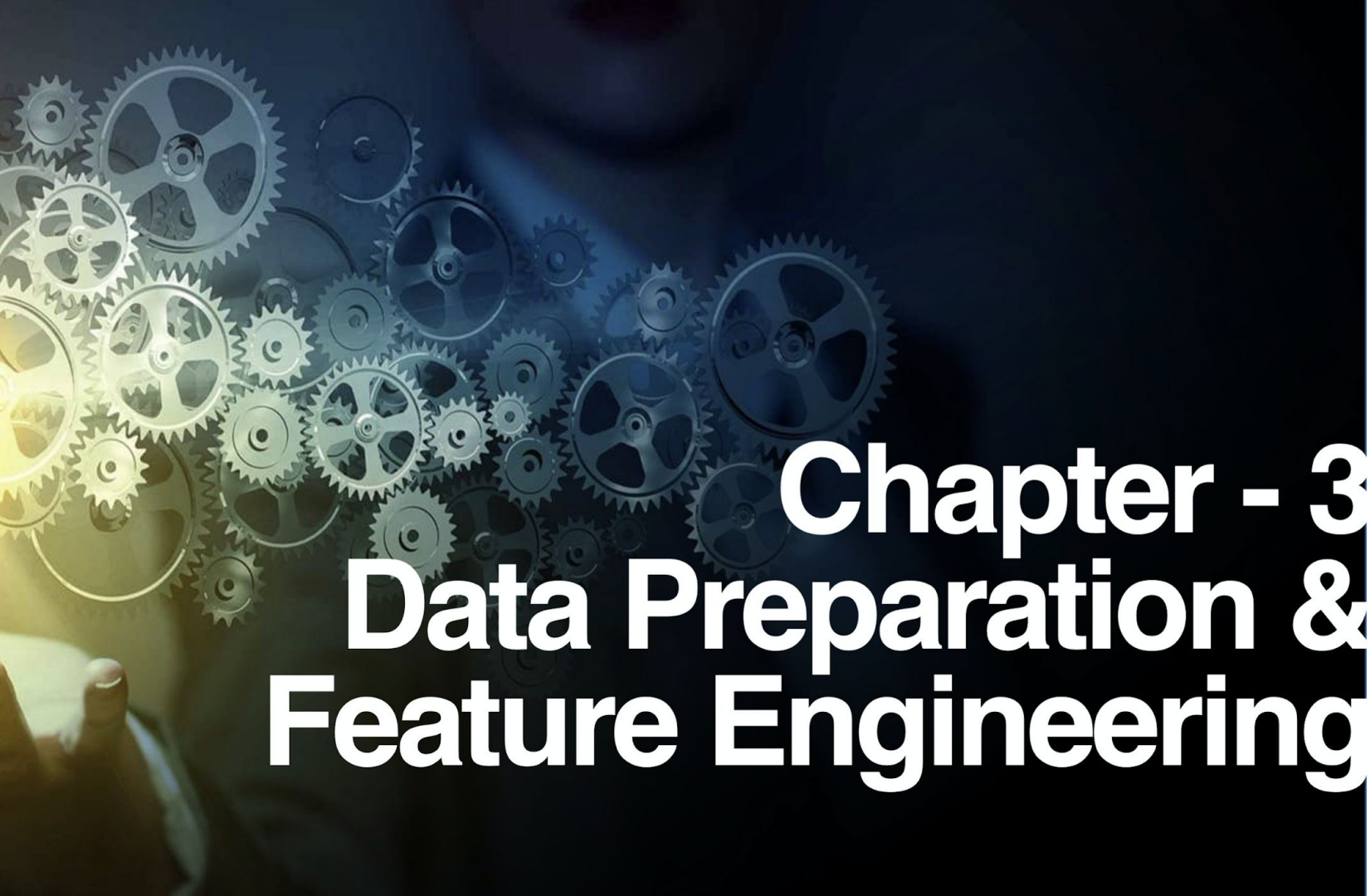


2. Exploratory Data Analysis:

Correlation

- Strong correlation among: emp.var.rate, cons.price.idx, euribor3m, nr.employed
- Important features, keep them
- Strong correlation with Y: economic indicators, Previous, Poutcome,
- Tried PCA: poor model results



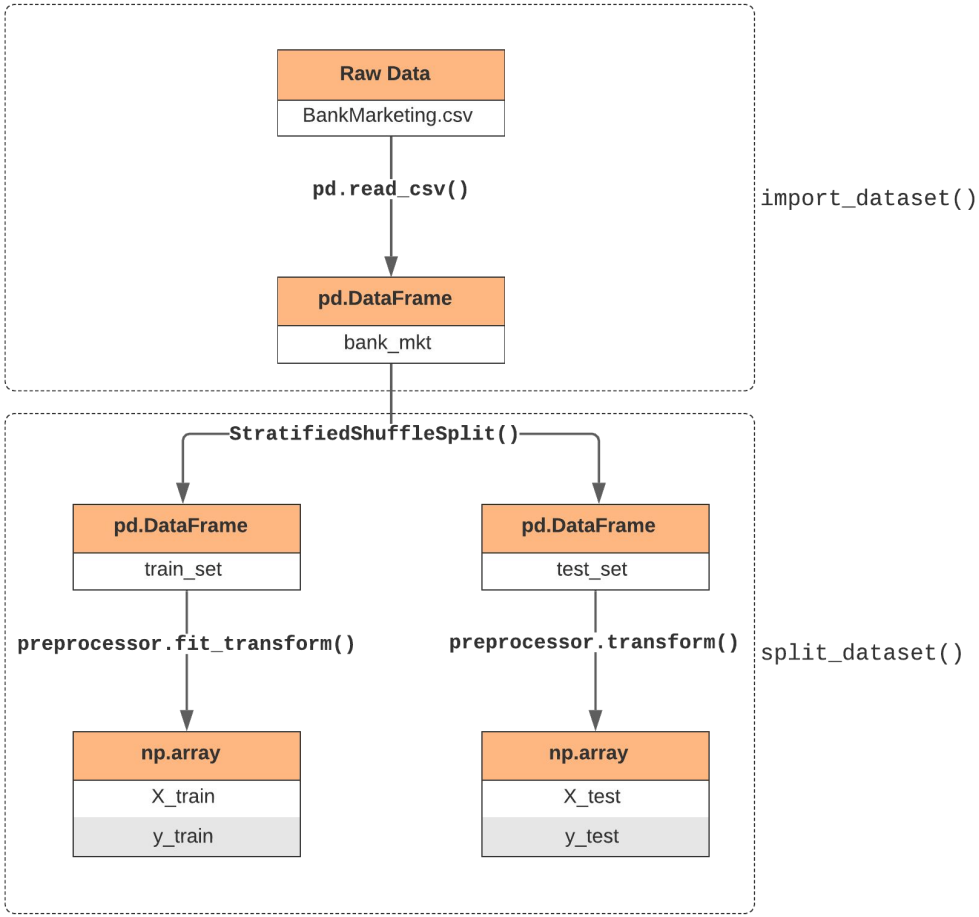


Chapter - 3
Data Preparation &
Feature Engineering

3. Data Preparation & Feature Engineering:

Data Lifecycle

- A function `import_dataset` to handle the importing process.
- A function `split_dataset` to split the dataset into training set and test set.
- `preprocessor` is a sklearn pipeline to wrap all functions and transformers needed to preprocess data.



3. Data Preparation & Feature Engineering:

Import Dataset

- Make sure duplicated rows, missing values, categorical and boolean data are properly processed by `pandas`.

```
bank_mkt = pd.read_csv(filename,
                       na_values=["unknown", "nonexistent"],
                       true_values=["yes", "success"],
                       false_values=["no", "failure"])
# Convert types, "Int64" is nullable integer data type in pandas
bank_mkt = bank_mkt.astype(dtype={"age": "Int64",...})
# Drop 12 duplicated rows
bank_mkt = bank_mkt.drop_duplicates().reset_index(drop=True)
```

Partition

- `scikit-learn` provides a useful function `StratifiedShuffleSplit` to select representative data and split them into training and test set.

```
train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2)
for train_index, test_index in train_test_split.split(data.drop("y", axis=1),
data["y"]):
    train_set = data.iloc[train_index]
    test_set = data.iloc[test_index]
```

3. Data Preparation & Feature Engineering:

Preprocessor

- A function `dftransform` is needed to handle all necessary transformation in `pandas`.
- `dftransform` then can be bundled with other transformers, e.g. `OneHotEncoder`.

```
def dftransform(X, drop=None, cut=None, gen=None, cyclic=None, target=None,
                fillna=True):
    …
    return X

dftransformer = FunctionTransformer(dftransform, kw_args={"fillna":True})
X_train = dftransformer.fit_transform(train_set)
```

Encoder & Scaler

- `OneHotEncoder` encodes categorical values into multiple binary values.
- `StandardScaler` scale numerical values into same range.

```
cat_features = ["job",...]
num_features =  ["age",...]
hot_scaler = ColumnTransformer([
    ("one_hot_encoder", OneHotEncoder(drop="first"), cat_features),
    ("scaler", StandardScaler(), num_features)], remainder="passthrough")
preprocessor = make_pipeline(dftransformer, hot_scaler)
X_train = preprocessor.fit_transform(train_set)
X_test = preprocessor.transform(test_set)
```

3. Data Preparation & Feature Engineering:

Missing Values

- XGBoost and CatBoost can handle NaN.
- For other classifiers, missing value can be filled by constant values (-1 or -999) or most frequent values.
- Estimate missing values using `IterativeImputer` but this method brings overfitting in our experience.
- Sometimes missing value itself has meanings. For example, clients who have default history may not want to declare their default status. We can create another feature to accommodate this information.

job	job_freq	job_fill	has_job
3	3	3	1
7	7	7	1
7	7	7	1
NaN	**7**	**-1**	**0**
7	7	7	1
7	7	7	1
0	0	0	1

3. Data Preparation & Feature Engineering:

Feature Generation

- Some clients do not have `pdays` but have `poutcome`, which implies that they may have been contacted before. We can put `pdays=999` for this kind of clients and construct a new feature `has_previous`.
- `pdays` can be cut into different categories which is known as the discretization process.

pdays	pdays_encode	pdays_cat	previous
999	**NaN**	**NaN**	**0**
999	**999**	**6**	**1**
3	3	**1**	1
10	10	**2**	3
13	13	**3**	1

3. Data Preparation & Feature Engineering:

Feature Generation

- Date can be important feature but is very difficult for algorithms to detect. In our case, `month` only brings seasonal information for the algorithms.
- As discussed before, financial crisis is a key event in our dataset. We can construct a new feature to bring this information into the modelling.

month	year	date	lehman	**days**
5	2008	2008-05-01	2008-09-15	**-137**
7	2008	2008-07-01	2008-09-15	**-76**
8	2008	2008-08-01	2008-09-15	**-45**
10	2008	2008-10-01	2008-09-15	**16**
11	2008	2008-11-01	2008-09-15	**47**
4	2009	2009-04-01	2008-09-15	**198**

3. Data Preparation & Feature Engineering:

Drop Features

- Surprisingly, the feature engineering processes discussed above and many more techniques we did not discuss (e.g. mean encoding, cyclic encoding, adding economic indicators, etc.) bring little improvements to our modelling performance.
- The biggest and most stable improvements are actually brought by deleting client features.
- On the other hand, One Hot Encoding and Standardization already put the dataset in good forms.

	CatBoost Baseline			CatBoost Drop		
	Train	Validate	Test	Train	Validate	Test
TNR	0.863	0.857	0.860	0.863	0.851	0.859
TPR	0.663	0.658	0.647	0.654	0.616	0.644
bACC	0.763	0.758	0.753	0.759	0.733	0.752
ROC	**0.850**	**0.808**	**0.802**	**0.826**	**0.801**	**0.804**
REC	0.663	0.658	0.647	0.654	0.616	0.644
PRE	0.380	0.369	0.370	0.377	0.344	0.367
AP	**0.536**	**0.483**	**0.447**	**0.515**	**0.463**	**0.452**

Chapter - 4
Model Evaluation

4. Model Evaluation:

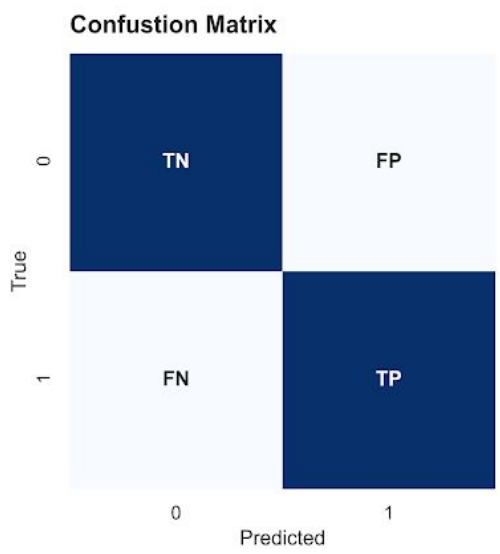
Confusion Matrix

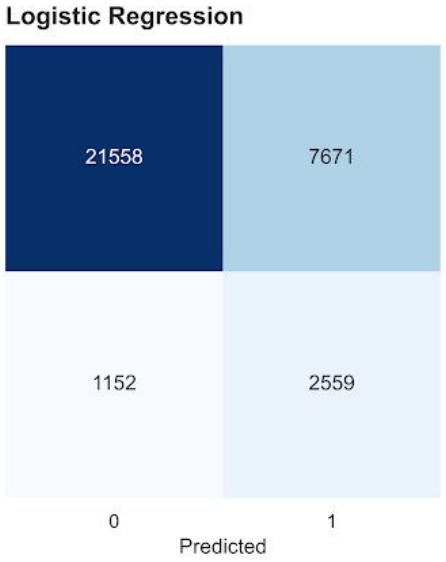
- Confusion Matrix shows the where the model prediction succeeds and fails.
- The four areas in Confusion Matrix are defined as TN, FP, FN, TP.
- The false positive rate (FPR) measures the error rate of the negative outcomes.

$$FPR = \frac{FP}{N} = \frac{FP}{FP+TN}$$

- The true negative rate (TNR) measures the accuracy rate for the negative outcomes.

$$TNR = \frac{TP}{N} = \frac{TN}{TN+FP} = 1 - FPR$$





4. Model Evaluation:

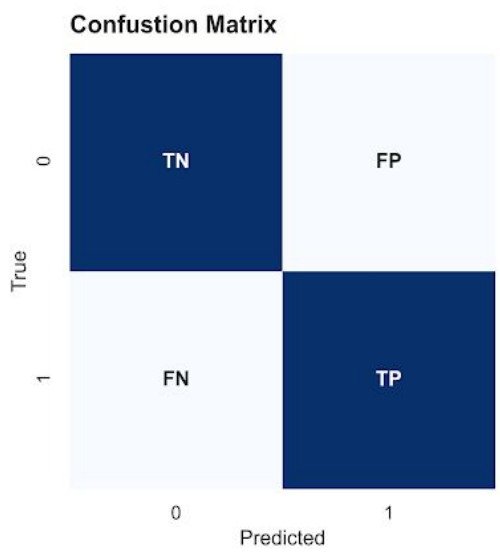
Metrics

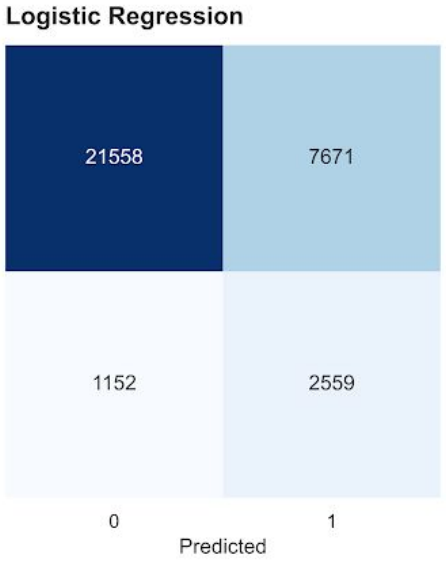
- The true positive rate (TPR) measures the accuracy rate for the positive outcomes. TPR is also known as recall.

$$REC = TPR = \frac{TP}{TP+FN}$$

- Balanced accuracy is the average of true positive rate and true negative rate.

$$bACC = \frac{TPR+TNR}{2}$$





4. Model Evaluation:

Metrics

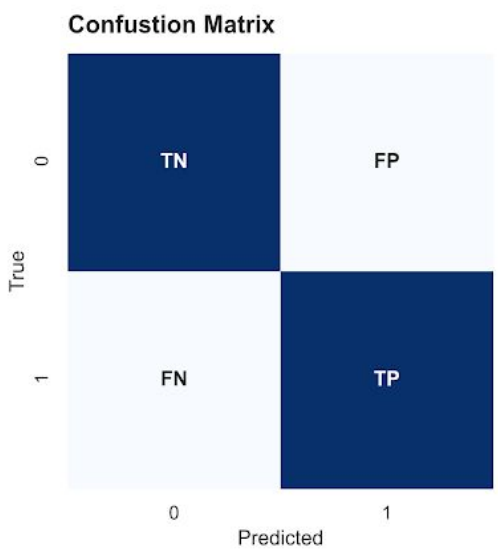
- Precision (PRE) measures the accuracy of the predicted positive outcomes.

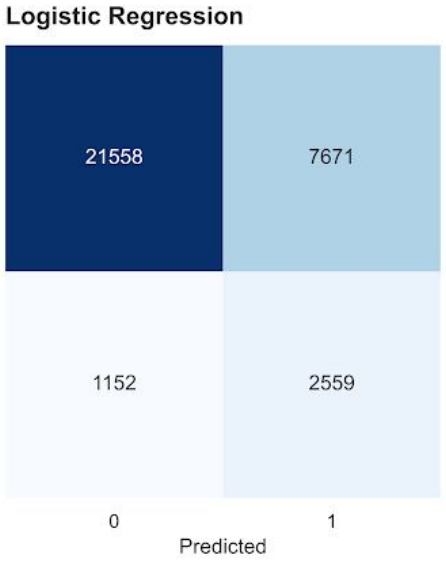
$$PRE = \frac{TP}{TP+FP}$$

- To balance the up- and down-sides of optimizing PRE and REC, the harmonic mean of precision and recall is used for F1.

$$F_1 = 2 \cdot \frac{PRE \times REC}{PRE+REC}$$

- F1 may not agree with bACC. We argue that F1 is not a good measure. Instead, Average Precision and ROC AUC are the metrics we should optimize.

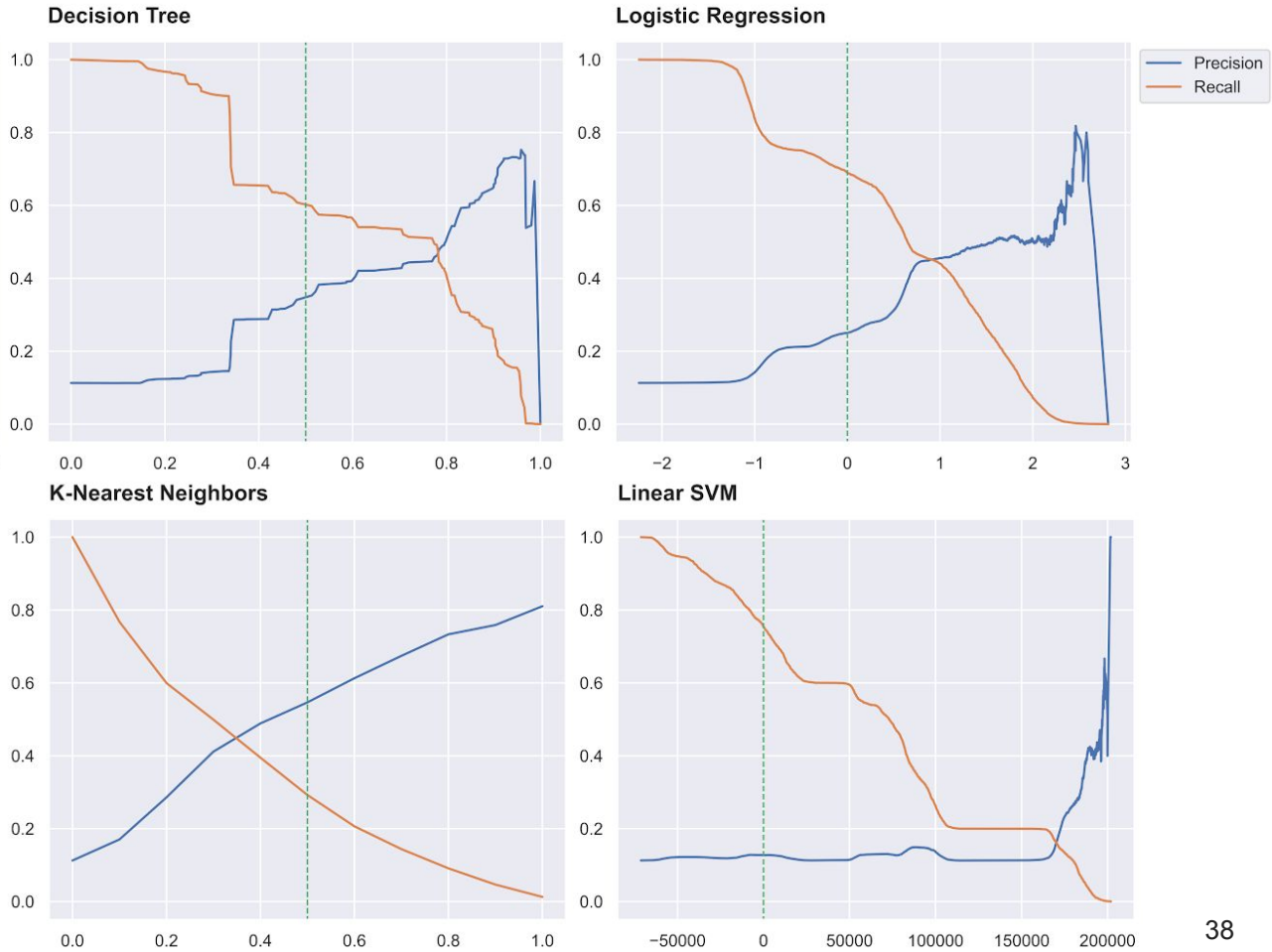




4. Model Evaluation:

Metrics

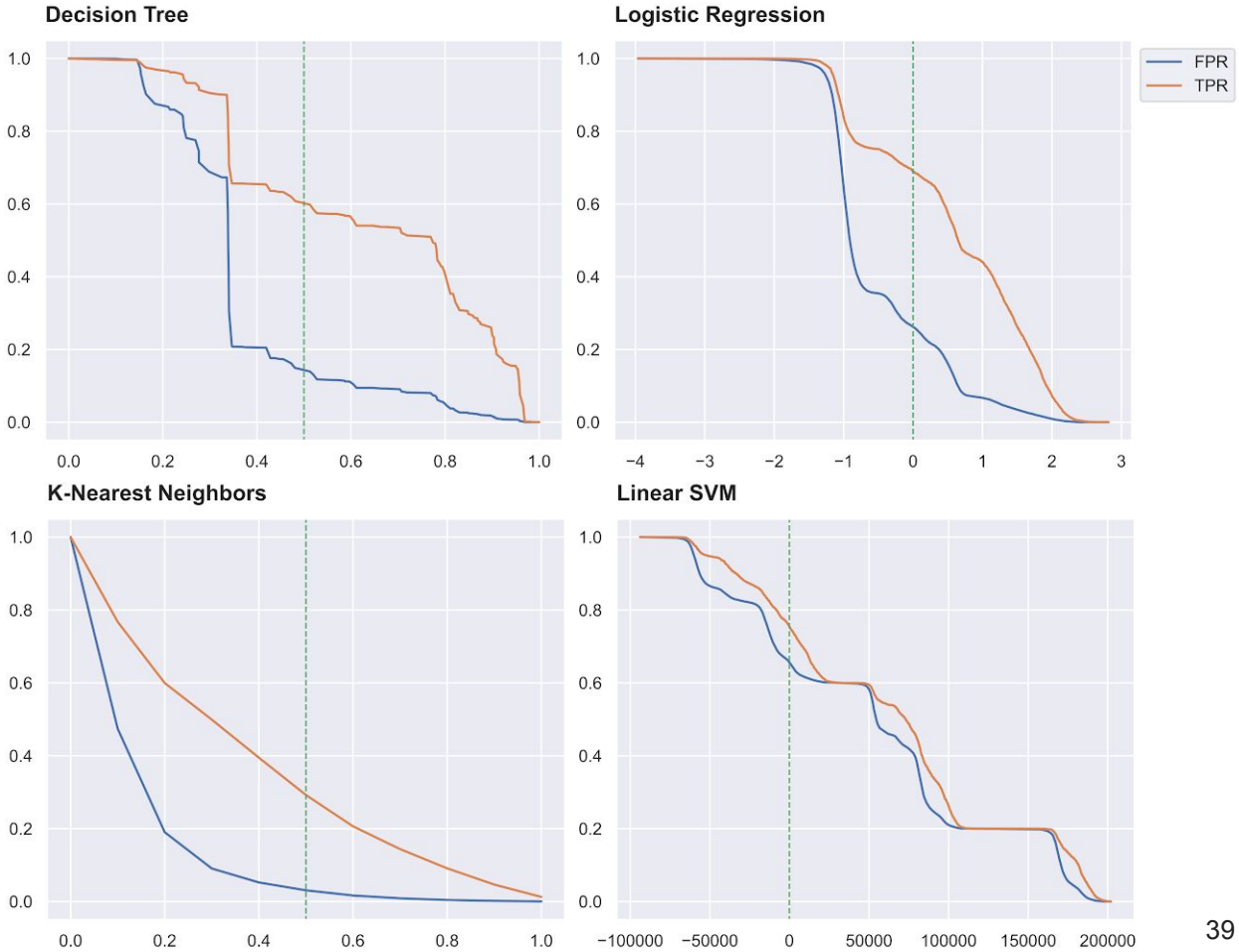
- All these metrics we discussed only show a snapshot of the model performance at a certain threshold.
- If we turn up or down the threshold, we may get a totally different trade-off which may be prefered.
- TNR and TPR is a trade off. REC and PRE is also a trade off.
- These trade offs can be plotted against each other, like utility curves in Economics.



4. Model Evaluation:

Metrics

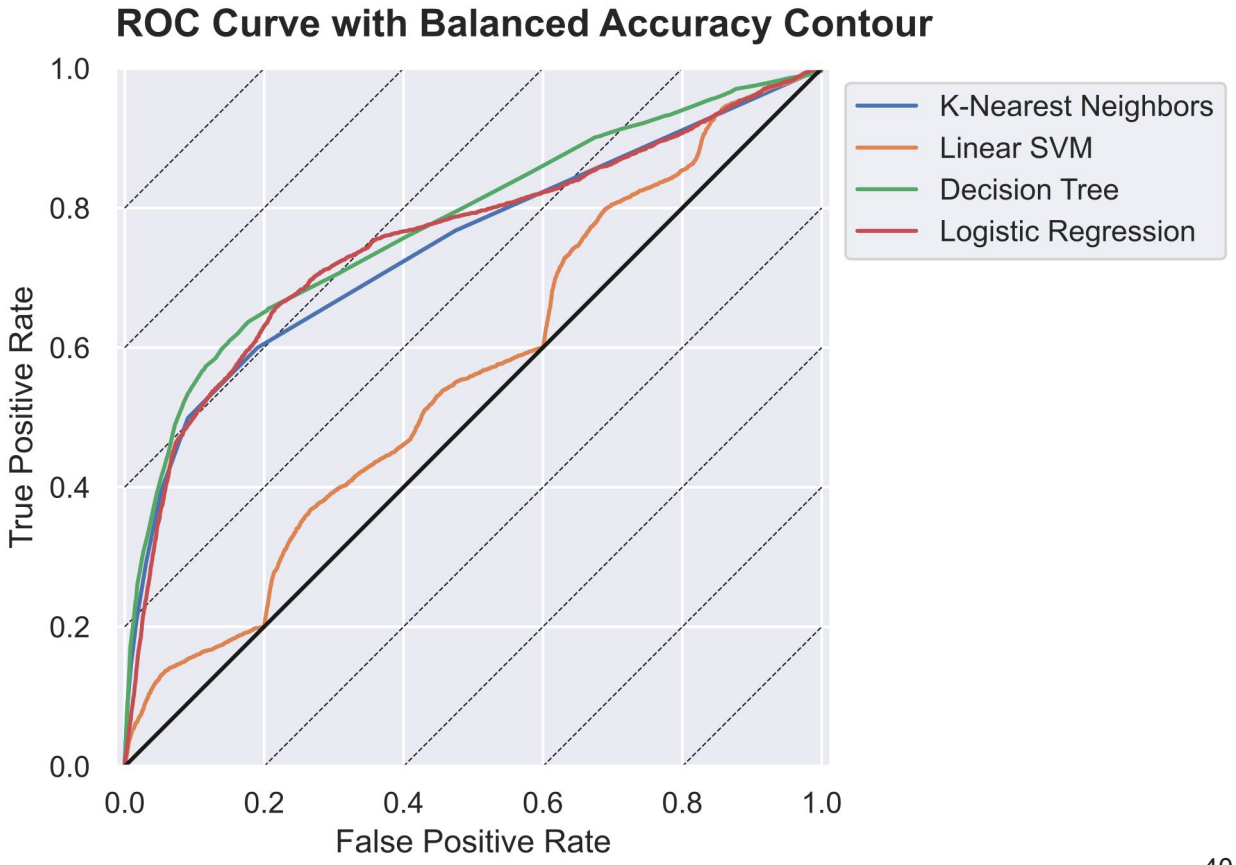
- All these metrics we discussed only show a snapshot of the model performance at a certain threshold.
- If we turn up or down the threshold, we may get a totally different trade-off which may be prefered.
- TNR and TPR is a trade off. REC and PRE is also a trade off.
- These trade offs can be plotted against each other, like utility curves in Economics.



4. Model Evaluation:

Metrics

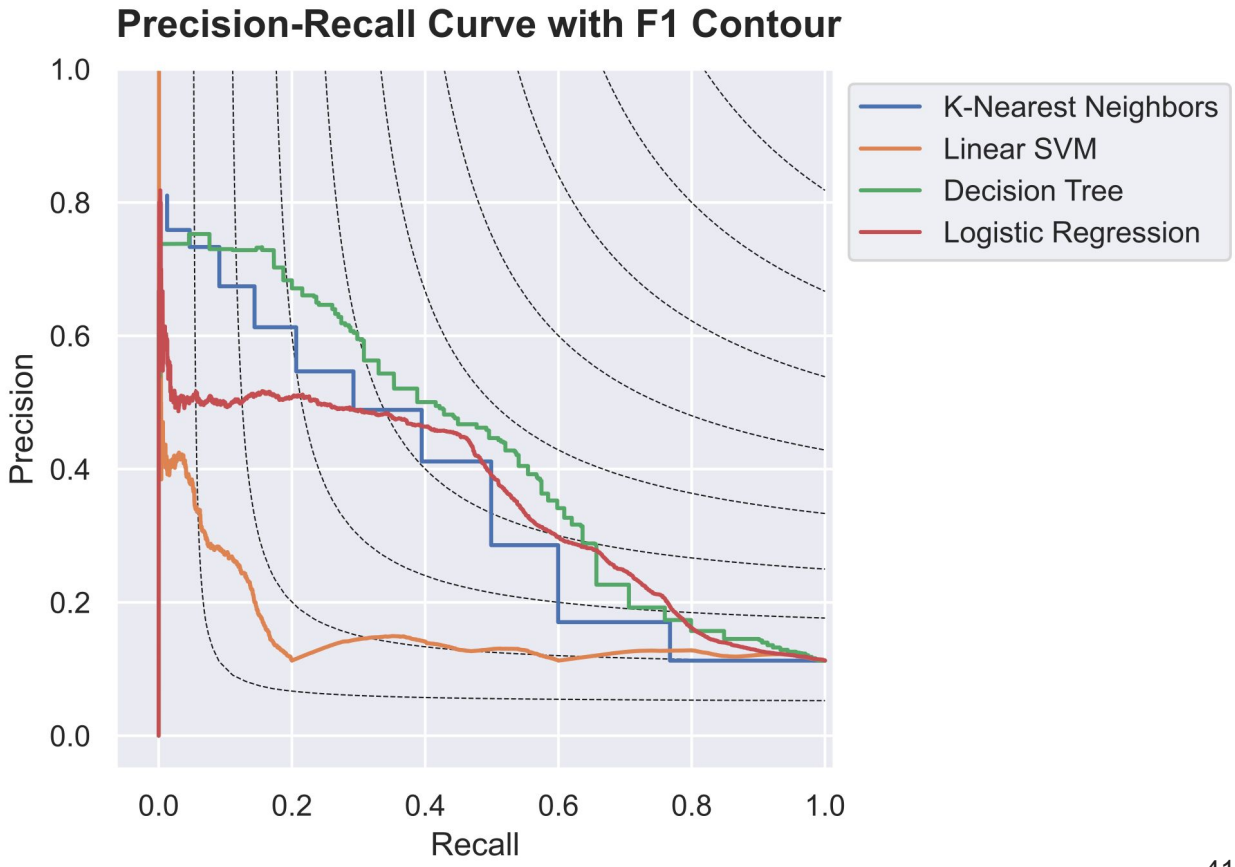
- A receiver operating characteristic (ROC) is created by plotting TPR against FPR at various threshold settings.
- We can then calculate the area under the ROC curve and call it ROC AUC.



4. Model Evaluation:

Metrics

- A Precision-Recall curve shows PRE against REC.
- For a marketing campaign, we assume that the bank wants to capture as many potential clients as possible given certain budget constraints. Therefore, the Precision-Recall curve should be pushed as far as possible.
- F1 favors models with equal PRE and REC rather than those optimize the Precision-Recall curve.



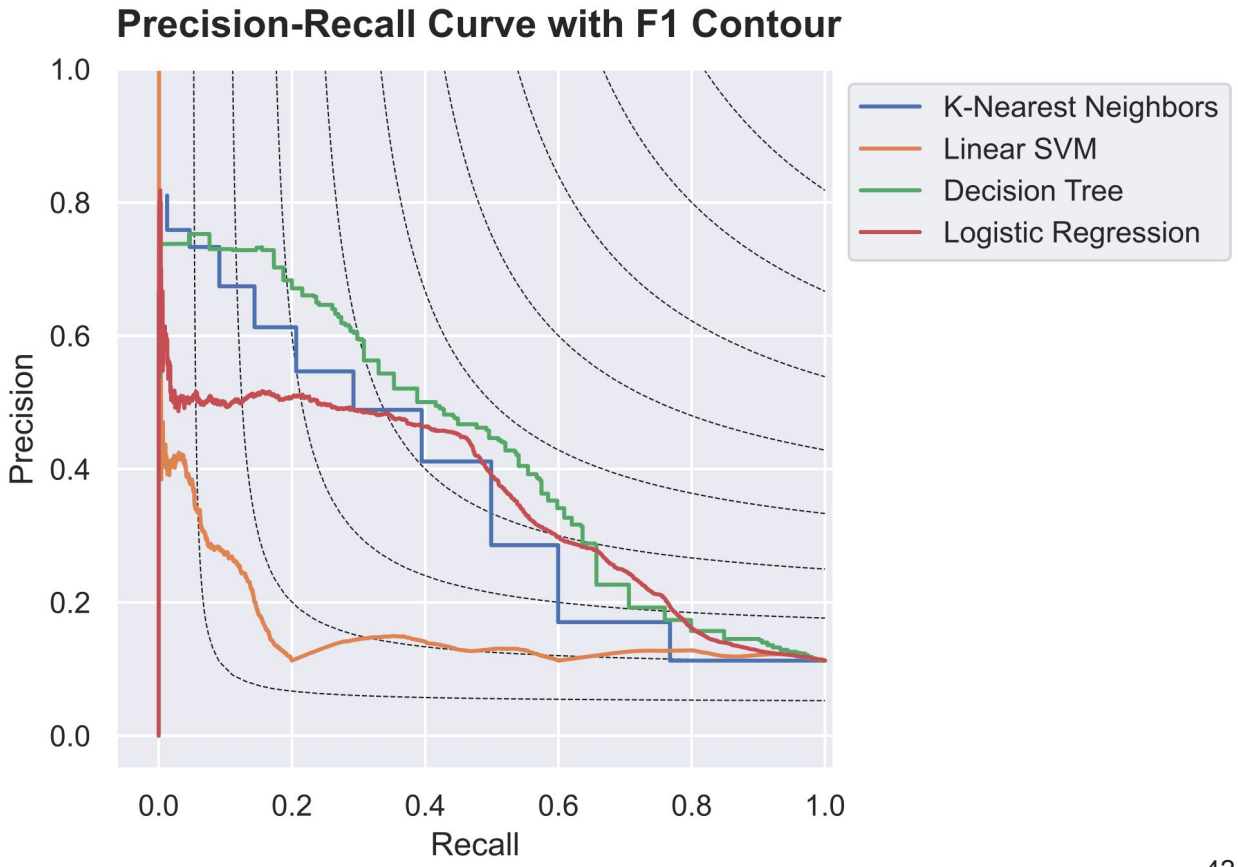
4. Model Evaluation:

Metrics

- Average Precision (AP) summarizes the Precision-Recall curve as the weighted mean of precisions achieved at each threshold.

$$AP = \sum_n (REC_n - REC_{n-1}) PRE_n$$

- This implementation is different from computing the area under the Precision-Recall curve with the trapezoidal rule, which uses linear interpolation and can be too optimistic.



4. Model Evaluation:

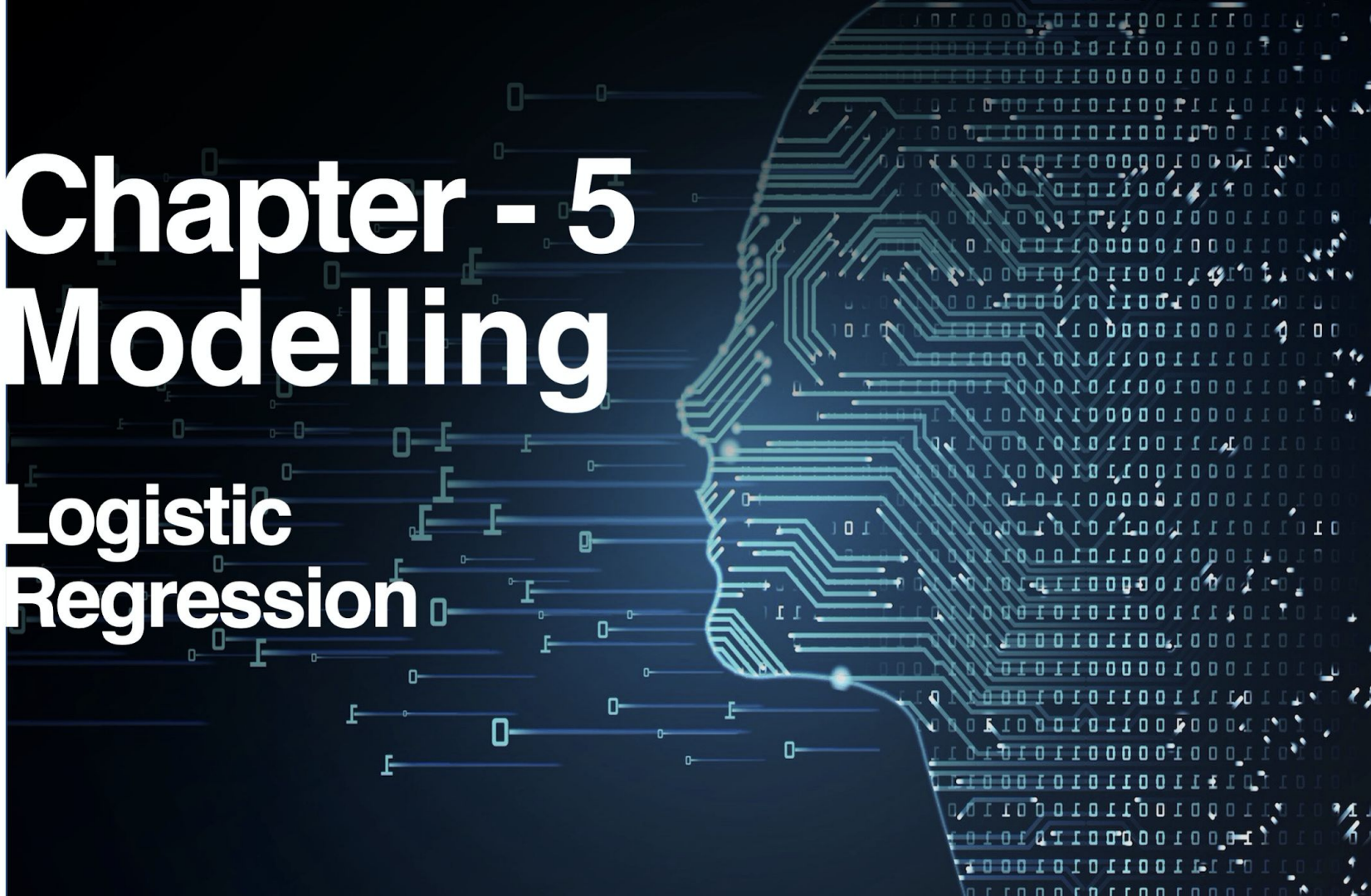
Metrics

- One metric does not rule them all. In practice, we use multiple metrics to evaluate and optimize our models to cover the whole picture.
- AP has a range of (P%, 1), i.e. (0.11, 1) in our case. AP gives a much more straightforward picture of prediction improvements in an imbalanced dataset.
- ROC AUC puts more weight on TN which might not be prefered.

	Constant Prediction	Random Prediction	K-Nearest Neighbors	Linear SVM	Decision Tree	Logistic Regression
FPR	1.000	0.500	0.017	0.681	0.139	0.262
TNR	0.000	0.500	0.983	0.319	0.861	0.738
TPR	1.000	0.495	0.206	0.806	0.598	0.690
bACC	0.500	0.497	0.595	0.563	0.729	0.714
ROC	0.500	0.500	**0.744**	0.562	0.776	**0.754**
REC	1.000	0.495	0.206	0.806	0.598	0.690
PRE	0.113	0.112	0.613	0.131	0.353	0.250
F1	0.203	0.182	0.309	0.225	0.444	0.367
AP	0.113	0.113	**0.366**	0.158	0.426	**0.353**

Chapter - 5
Modelling

Logistic Regression

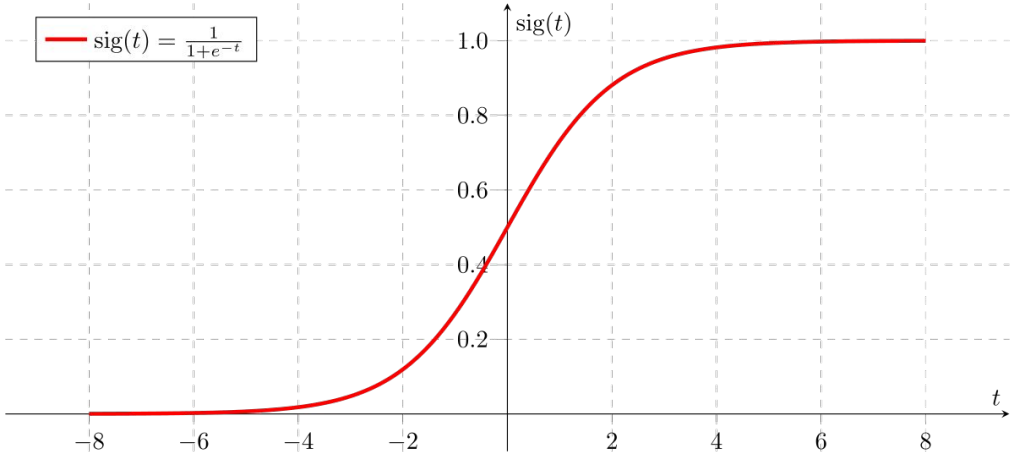


5.1. Logistic Regression:

Logistic Regression

- Probability-based binary classifier
- Logistic sigmoid function
- Perform well on linearly separable classes

$$h(x) = 1/(1 + e^{-(theta_0 + theta_1 * x_1 + theta_2 * x_2 + theta_3 * x_3 + .. + theta_n * x_n)})$$



5.1. Logistic Regression:

Fitting and Testing

- Data preparation: frequent imputer
- Import model from sklearn
- “Class_weight = ‘balanced’”
- “Max_iter = 10000”
- Result: 78% accuracy

```
#data preparation with freq_imputer, and split train & test set
cat_encoder = FunctionTransformer(cat_encode)

freq_features = ["job", "marital", "education", "default", "housing", "loan"]

freq_imputer = ColumnTransformer([
    ("freq_imputer", SimpleImputer(missing_values=-1, strategy="most_frequent"), freq_features)],
      remainder="passthrough")

freq_encoder = make_pipeline(cat_encoder, freq_imputer)

X_train, y_train, X_test, y_test, *other_sets = split_dataset(bank_mkt, freq_encoder)
```

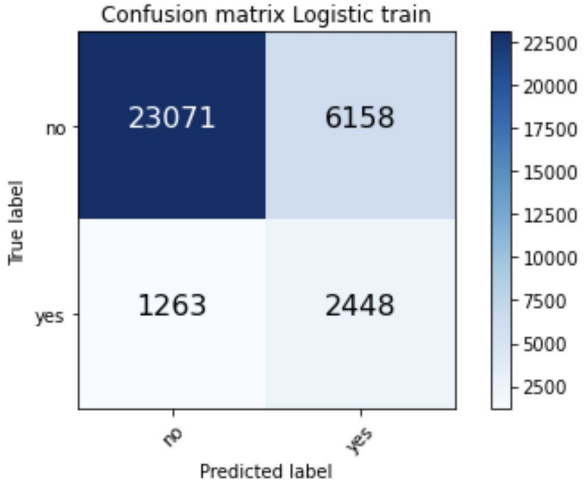
```
#### import LogisticRegression and fit train set data
from sklearn.linear_model import LogisticRegression
lrmodel = LogisticRegression(class_weight='balanced',max_iter=10000)
lrmodel.fit(X_train, y_train)
y_train_pred = lrmodel.predict(X_train)
#### model measures for training data
cmtr = confusion_matrix(y_train, y_train_pred)
acctr = accuracy_score(y_train, y_train_pred)
aps_train = average_precision_score(y_train, y_train_pred)

#### fit test set data
lrmodel.fit(X_test, y_test)
y_test_pred = lrmodel.predict(X_test)
#### model measures for testing data
cmte = confusion_matrix(y_test, y_test_pred)
accte = accuracy_score(y_test, y_test_pred)
aps_test = average_precision_score(y_test, y_test_pred)

print('Accuracy Score:',acctr, '    APS:',aps_train)
print('Accuracy Score:',accte, '    APS:',aps_test)
```

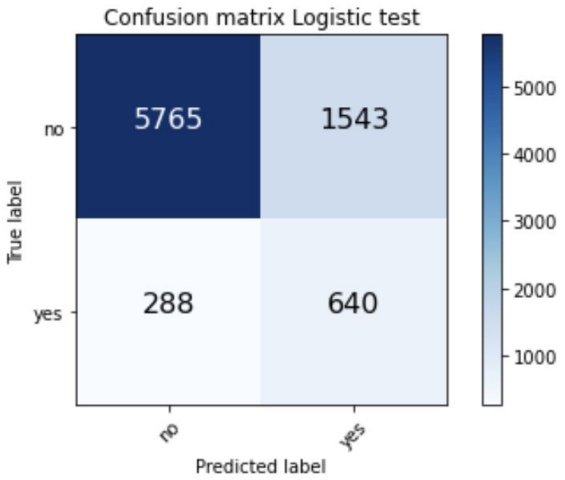
```
Accuracy Score: 0.7747115968427444
Accuracy Score: 0.7776833414278777
```

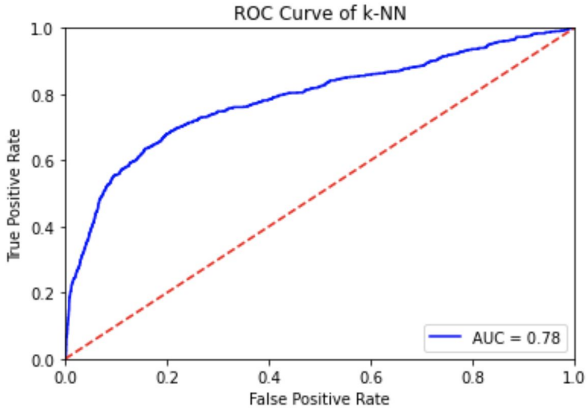
5.1. Logistic Regression:

Model Result

- Confusion matrices
- AUC = 0.78
- Average precision score: 0.428







	Train	Validate	Test
TNR	0.789976	0.790797	0.793240
TPR	0.658471	0.652291	0.683190
bACC	0.724223	0.721544	0.738215
ROC	0.778001	0.768561	0.783782
REC	0.658471	0.652291	0.683190
PRE	0.284736	0.283538	0.295571
AP	0.425629	0.411025	0.428069

5.1. Logistic Regression:

GridSearch-1

- Parameters: Penalty L2, C=[0.001,.009,0.01,.09,1,5,10,25,50,100]
- Best Parameters: L2, C=0.001

GridSearch-2

- Parameters:
  Penalty L1 + L2
  C: np.logspace(-3,3,7)
  solver: ["saga"]
- Best Parameters: L1, C=21.5

Original:

	Train	Validate	Test
TNR	0.789976	0.790797	0.793240
TPR	0.658471	0.652291	0.683190
bACC	0.724223	0.721544	0.738215
ROC	0.778001	0.768561	0.783782
REC	0.658471	0.652291	0.683190
PRE	0.284736	0.283538	0.295571
AP	0.425629	0.411025	0.428069

GridSearch-1:

	Train	Validate	Test
TNR	0.753710	0.752822	0.765326
TPR	0.681037	0.699461	0.711207
bACC	0.717374	0.726142	0.738266
ROC	0.765923	0.780018	0.780575
REC	0.681037	0.699461	0.711207
PRE	0.259864	0.264257	0.277895
AP	0.410849	0.462047	0.430201

GridSearch-2:

	Train	Validate	Test
TNR	0.730787	0.731611	0.746853
TPR	0.692826	0.700809	0.719828
bACC	0.711807	0.716210	0.733340
ROC	0.761033	0.775730	0.779457
REC	0.692826	0.700809	0.719828
PRE	0.246288	0.248923	0.265290
AP	0.409927	0.435187	0.431490

5.1. Logistic Regression:

Statistic Results

- “import statsmodels.api as sm”
- Strong predictors: economic indicators, date features
- Overall negative R-square

Logit Regression Results

Dep. Variable:	y	No. Observations:	32950
Model:	Logit	Df Residuals:	32917
Method:	MLE	Df Model:	32
Date:	Tue, 10 Nov 2020	Pseudo R-squ.:	-0.8171
Time:	15:56:19	Log-Likelihood:	-21077.
converged:	True	LL-Null:	-11599.
Covariance Type:	nonrobust	LLR p-value:	1.000

	coef	std err	z	P>|z|	[0.025	0.975]
x1	-0.0292	0.016	-1.774	0.076	-0.062	0.003
x2	-0.0075	0.012	-0.615	0.539	-0.031	0.016
x3	-0.0021	0.012	-0.172	0.864	-0.027	0.022
x4	-0.0043	0.013	-0.341	0.733	-0.029	0.021
x5	0.0453	0.014	3.203	0.001	0.018	0.073
x6	-0.0023	0.012	-0.193	0.847	-0.026	0.021
x7	-0.0230	0.013	-1.737	0.082	-0.049	0.003
x8	0.0244	0.013	1.855	0.064	-0.001	0.050
x9	-0.0018	0.014	-0.136	0.892	-0.028	0.025
x10	-0.0032	0.012	-0.261	0.794	-0.027	0.021
x11	0.0078	0.019	0.420	0.675	-0.029	0.044
x12	0.0257	0.020	1.305	0.192	-0.013	0.064
x13	-0.0011	0.012	-0.090	0.928	-0.024	0.022
x14	-0.0060	0.012	-0.514	0.607	-0.029	0.017
x15	0.0026	0.012	0.224	0.823	-0.020	0.025
x16	0.0218	0.015	1.480	0.139	-0.007	0.051
x17	-0.2523	nan	nan	nan	nan	nan
x18	1.4036	nan	nan	nan	nan	nan
x19	0.0208	nan	nan	nan	nan	nan
x20	0.0030	nan	nan	nan	nan	nan
x21	-0.0458	nan	nan	nan	nan	nan
x22	-0.0533	nan	nan	nan	nan	nan
x23	0.7774	nan	nan	nan	nan	nan
x24	0.0072	0.015	0.489	0.625	-0.022	0.036
x25	-0.2618	0.017	-15.093	0.000	-0.296	-0.228
x26	-0.1095	0.018	-6.178	0.000	-0.144	-0.075
x27	0.0230	0.012	1.987	0.047	0.000	0.046
x28	-0.0214	0.012	-1.811	0.070	-0.044	0.002
x29	0.0351	0.044	0.795	0.427	-0.052	0.122
x30	-0.7339	0.077	-9.569	0.000	-0.884	-0.584
x31	0.3205	0.041	7.752	0.000	0.239	0.402
x32	0.1379	0.021	6.465	0.000	0.096	0.180
x33	0.6045	0.118	5.126	0.000	0.373	0.836
x34	-0.4117	0.080	-5.138	0.000	-0.569	-0.255

Month & DOW

Economic Indicators

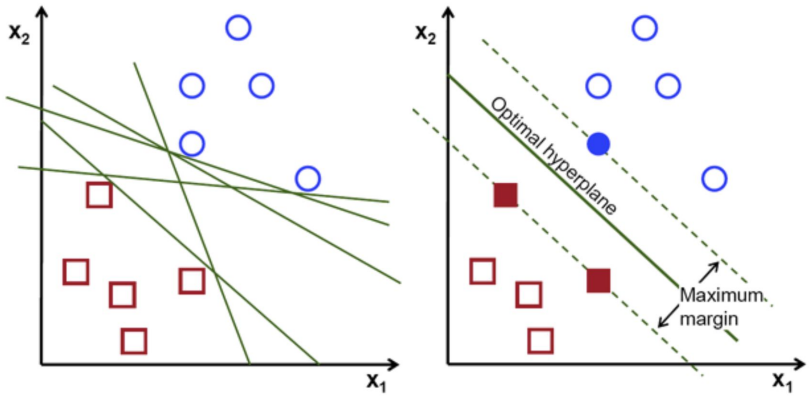
Chapter - 5 Modelling

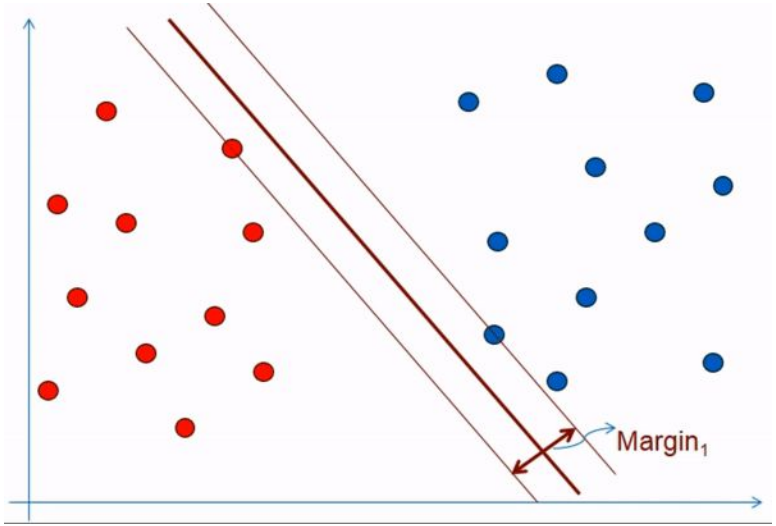
Support Vector Machine

5.2. SVM --- Large Margin Classifier

Introduction

- **Objective of SVM**
- Find a hyperplane in an N-dimensional space that distinctly classifies the data points
- The distance of an observation from the hyperplane can be seen as a measure of our confidence that the observation is correctly classified
- Support Vectors:
  Data points closer to the hyperplane and influence the position and orientation of the hyperplane





5.2. SVM --- Large Margin Classifier

Introduction

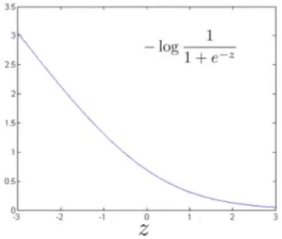
- **Important Features:**
  - **Cost Function**
  - Regularization
  - Gamma
  - Kernel

Logistic Regression

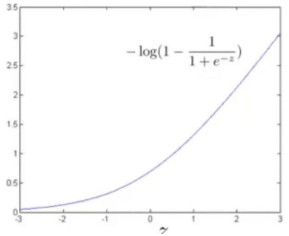
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log(1 - \frac{1}{1+e^{-\theta^T x}})$$

If $y = 1$ (want $\theta^T x \gg 0$):

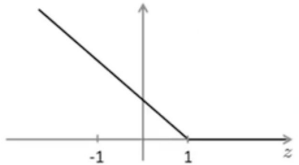


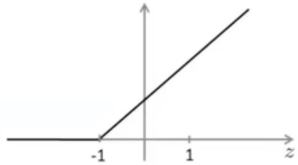
If $y = 0$ (want $\theta^T x \ll 0$):



Support Vector Machine

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$





If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

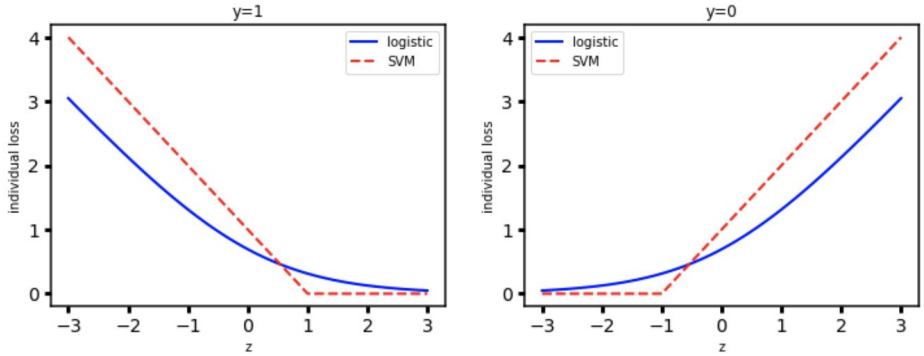
If $y = 0$, we want $\theta^T x \leq -1$ (not just < 0)

5.2. SVM --- Large Margin Classifier

Introduction

- **Important Features:**
  - **Cost Function**
  - Regularization
  - Gamma
  - Kernel

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



Correct Classification:
z < -1 for y=0 z > 1 for y=1 Individual Loss = 0

Wrong Classification:
z>-1 for y=0 z<1 for y=1

SVM tends to penalize more than logistic regression

Red curve is above the **Blue Curve** most of the time

5.2. SVM --- Large Margin Classifier

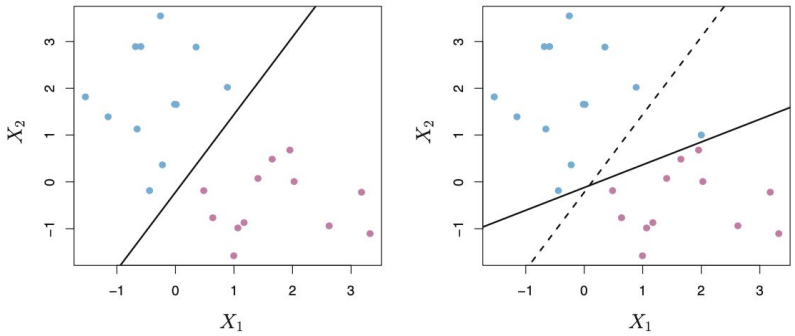
Introduction

- **Important Features:**
  - Cost Function
  - **Regularization**
  - Gamma
  - Kernel

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Cost Function = Hinge Loss + L2 Regularization

Hinge Loss: Penalize the margin violations
Regularization: Widen the distance between the two margins



Parameter C: Achieve the Balance

Large C -- small-margin separating hyperplane if the hyperplane does better job of getting all the points classified

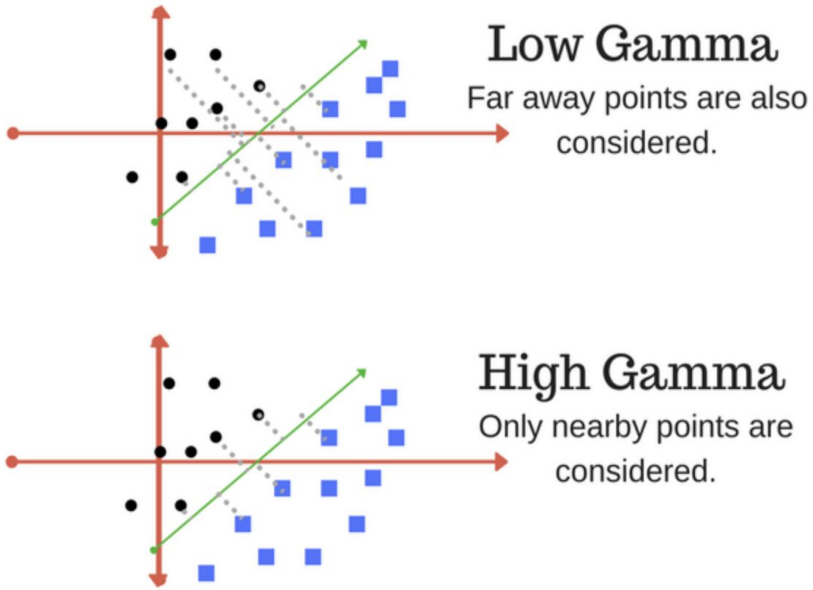
Small C -- large-margin separating hyperplane, even if the hyperplane misclassified more points

5.2. SVM --- Large Margin Classifier:

Introduction

- **Important Features:**
  - Cost Function
  - Regularization
  - **Gamma**
  - Kernel

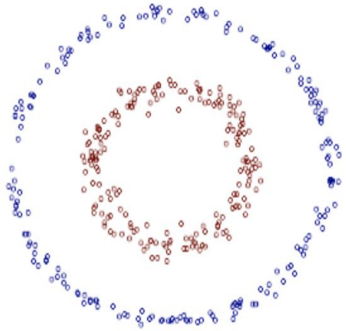
Parameter Gamma:
How far the influence of a single training example reaches:
Low values meaning ‘far’
High values meaning ‘close’

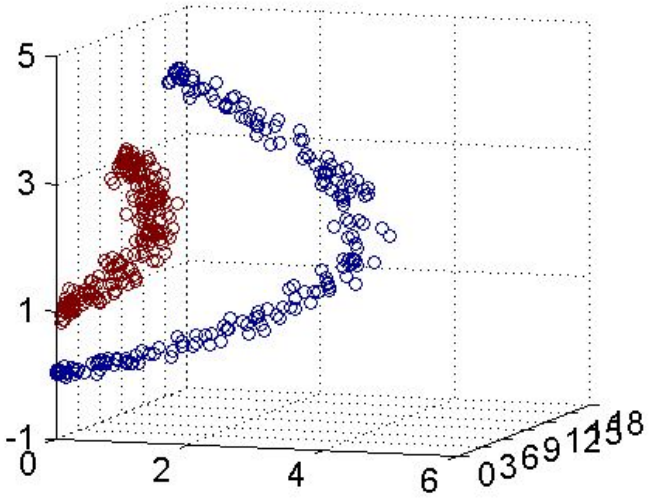


5.2. SVM --- Large Margin Classifier:

Introduction

- **Important Features:**
  - Cost Function
  - Regularization
  - Gamma
  - **Kernel**





5.2. SVM --- Large Margin Classifier

Results

- GridSearchCV

Linear SVM

```
param_distributions = {"loss": ["squared_hinge", "hinge"],
                       "C": loguniform(1e0, 1e3)}

random_search = RandomizedSearchCV(linear_svm,
                                   param_distributions,
                                   scoring="average_precision",
                                   cv=5,
                                   n_jobs=-1,
                                   n_iter=100)

grid_fit = random_search.fit(X_train, y_train)
grid_results = random_search.cv_results_
grid_best_params = random_search.best_params_
grid_best_score = random_search.best_score_
```

	Train	Validation	Test
ACC	0.775440	0.775015	0.774769
bACC	0.728312	0.727135	0.728209
ROC	0.783740	0.781762	0.781874
REC	0.667475	0.665327	0.668103
PRE	0.286718	0.285818	0.286110
F1	0.401117	0.399838	0.400646
AP	0.434161	0.432987	0.425736

Non-Linear SVM

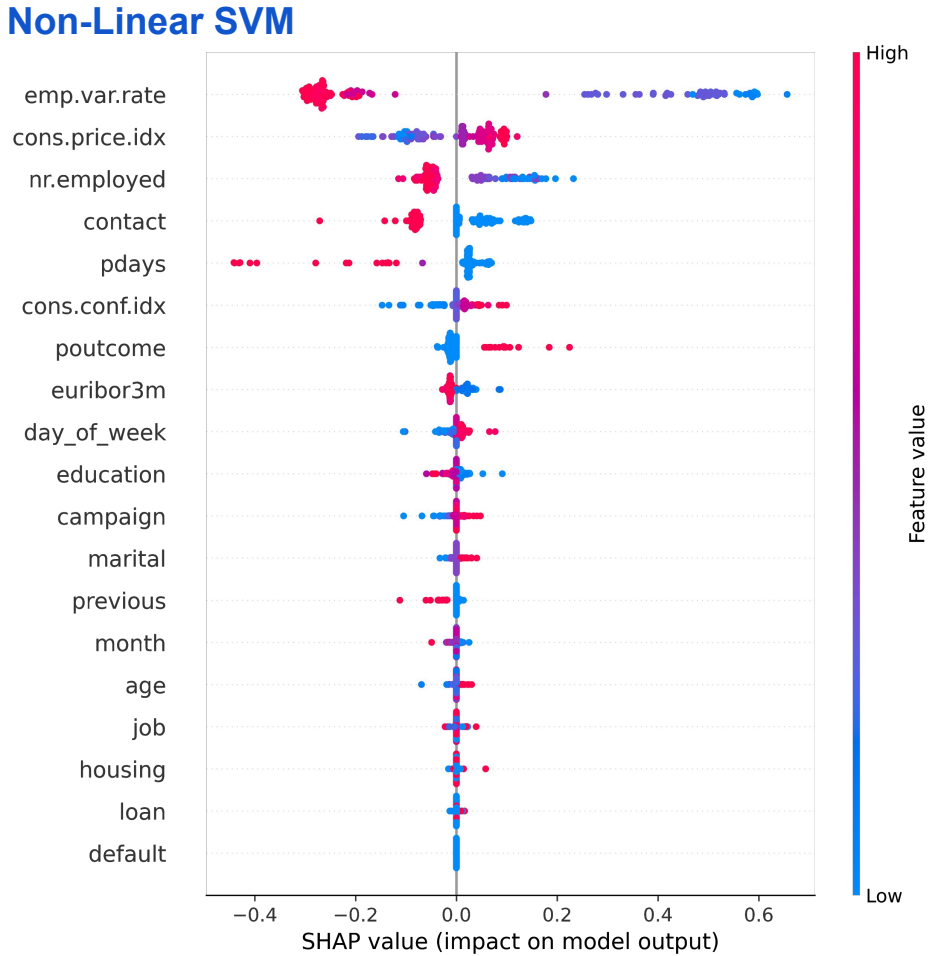
```
param_grid = {
    "rbf__gamma": [0.0001, 0.001, 0.01],
    "svm__C": [1, 10, 20]}

grid_search = GridSearchCV(rbf_clf,
                           param_grid,
                           scoring="average_precision",
                           cv=5,
                           n_jobs=-1)

grid_fit = grid_search.fit(X_train, y_train)
grid_results = grid_search.cv_results_
grid_best_params = grid_search.best_params_
grid_best_score = grid_search.best_score_
```

	Train	Validation	Test
ACC	0.765634	0.765786	0.771734
bACC	0.728932	0.728639	0.733084
ROC	0.784913	0.783210	0.783101
REC	0.681555	0.680685	0.683190
PRE	0.278976	0.278960	0.285586
F1	0.395884	0.395717	0.402795
AP	0.433833	0.431180	0.433372

5.2. SVM --- Large Margin Classifier

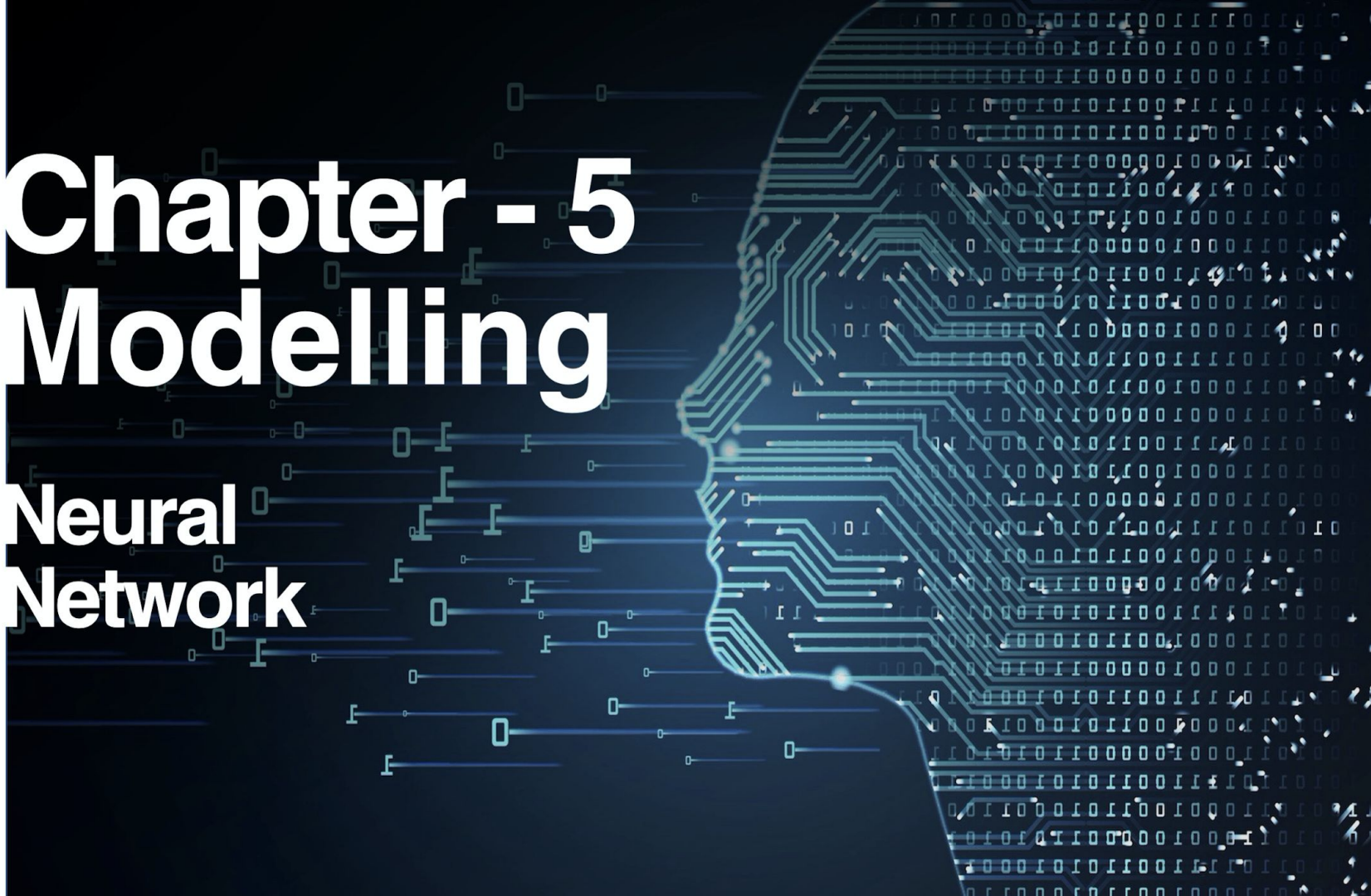
Open the Black Box

- **Shapley values**
  - Descending order of importance
  - Right cluster : Positive influence
  - Left cluster : Negative influence
  - Blue patches : Small feature values
  - Red patches: Large feature values



Chapter - 5
Modelling

Neural Network



5.3. Neural Network:

MLPClassifier

- GridSearchCV

```
mlp=MLPClassifier(random_state=42,max_iter=1000)

param_grid ={
    'solver':['lbfgs', 'sgd', 'adam'],
    'learning_rate':["constant","invscaling","adaptive"],
    'hidden_layer_sizes':[(100,),(200,),(20,5,),(10,5,),(100,50,25,)],
    'alpha':[0.0,0.001,0.01],
    'activation' :["logistic","relu","tanh"] }

grid_search = GridSearchCV(estimator=mlp,
                           param_grid=param_grid,
                           scoring = "average_precision",
                           return_train_score=True,
                           cv =2,
                           n_jobs=-1)

grid_fit = grid_search.fit(X_train, y_train)
grid_results = grid_search.cv_results_
grid_best_params = grid_search.best_params_
grid_best_score = grid_search.best_score_

print(f"best parameters found: {grid_best_params}, with mean test score: {grid_best_score}")
```

```
mlp_trained=MLPClassifier(solver ="lbfgs",
                          random_state=42,
                          max_iter=1000,
                          activation = 'relu',
                          alpha = 0.01,
                          hidden_layer_sizes = (100,),
                          learning_rate = 'constant')

benchmark(bank_mkt, freq_encoder, mlp_trained)
```

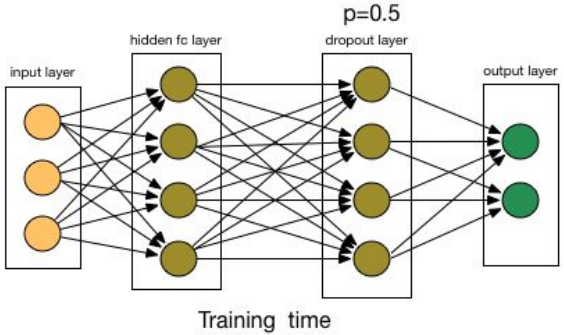
	Train	Validate	Test
TNR	0.989864	0.988026	0.986590
TPR	0.195352	0.207547	0.193966
bACC	0.592608	0.597787	0.590278
ROC	0.763937	0.762834	0.771836
REC	0.195352	0.207547	0.193966
PRE	0.709914	0.687500	0.647482
AP	0.421345	0.438027	0.415162

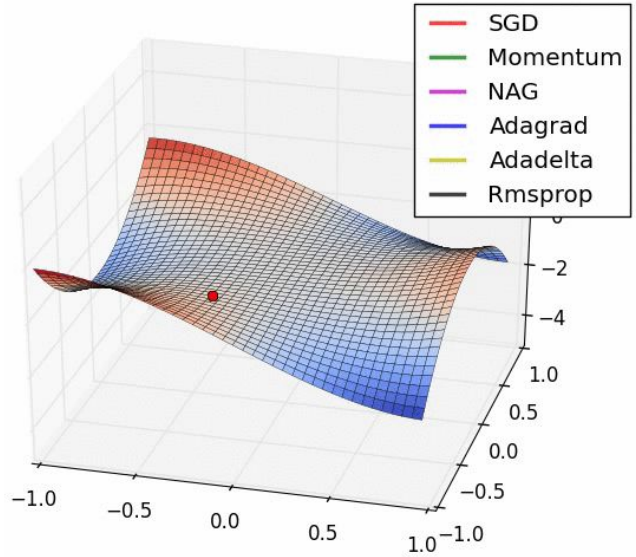
5.3. Keras

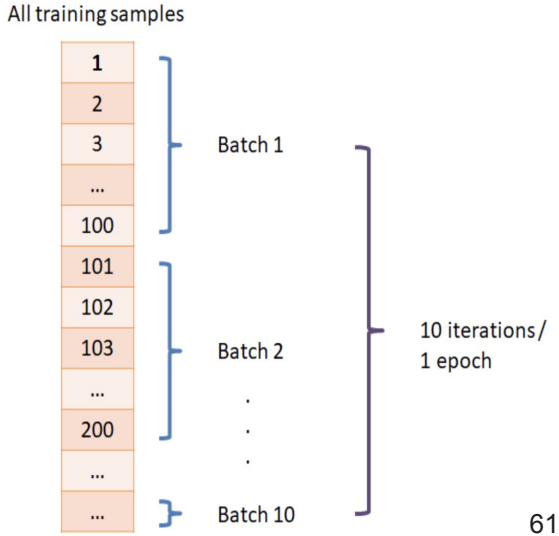
Parameters

- **Sequential**: Creating model sequentially --- Output of each layer is the input of the next layer specified
- **Dense**: Specify dimensions and activations
- **Dropout:** Prevent Overfitting
- **Compile:**
  - Loss Function
  - Optimizer
  - Metrics
- **Epoch & Batch Size**

```
model = Sequential()
model.add(Dropout(0.5, input_shape=(19,)))
model.add(Dense(5, activation='relu'))
model.add(Dropout(0.8))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['binary_accuracy'])
```







5.3 Keras:

Train & Validate & Test

```
history = model.fit(X_ttrain,
                    y_ttrain,
                    validation_data = (X_validate,y_validate),
                    epochs=25,
                    batch_size=500)
```

```
uracy: 0.8874 - val_loss: 0.4102 - val_binary_accuracy: 0.8874
Epoch 20/25
53/53 [==============================] - 0s 1ms/step - loss: 0.4071 - binary_acc
uracy: 0.8873 - val_loss: 0.4066 - val_binary_accuracy: 0.8874
Epoch 21/25
53/53 [==============================] - 0s 1ms/step - loss: 0.4018 - binary_acc
uracy: 0.8873 - val_loss: 0.3984 - val_binary_accuracy: 0.8874
Epoch 22/25
53/53 [==============================] - 0s 2ms/step - loss: 0.3954 - binary_acc
uracy: 0.8873 - val_loss: 0.3969 - val_binary_accuracy: 0.8874
Epoch 23/25
```

```
_, accuracy = model.evaluate(x=X_test, y=y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
258/258 [==============================] - 0s 624us/step - loss: 0.3774 - binary_a
ccuracy: 0.8873
Accuracy: 88.73
```

Chapter - 5
Modelling

Decision Tree
& Random Forest

5.4. Random Forest:

Hyperparameter tuning and fitting

- The forest and it's trees
- Discussion of hyperparameters
- Train vs Test accuracy measures
- Tree vs Forest

```
X_train = tree_preprocessor.fit_transform(bank_train_set.drop(["duration", "y"], axis=1))
y_train = bank_train_set["y"].astype("int").to_numpy()

from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(random_state=42,class_weight="balanced",criterion ="gini",max_features="auto",
                            min_samples_split=2)
from sklearn.model_selection import GridSearchCV
param_grid = {
    'max_depth':[6,8,10],
    'n_estimators':[1000,1500,1750,2000]
}
CV_RFmodel = GridSearchCV(estimator=RF,param_grid=param_grid,scoring="average_precision",n_jobs=-1,cv=2)
CV_RFmodel.fit(X_train,y_train)
grid_results = CV_RFmodel.cv_results_
grid_best_params = CV_RFmodel.best_params_
grid_best_score = CV_RFmodel.best_score_
grid_best_estimator = CV_RFmodel.best_estimator_
print(f"best mean test score: {grid_best_score}, for {grid_best_estimator}")
```

```
best mean test score: 0.4610938403465603, for RandomForestClassifier(class_weight='balanced', max_depth=6, n_estimators=1500,
                      random_state=42)
```

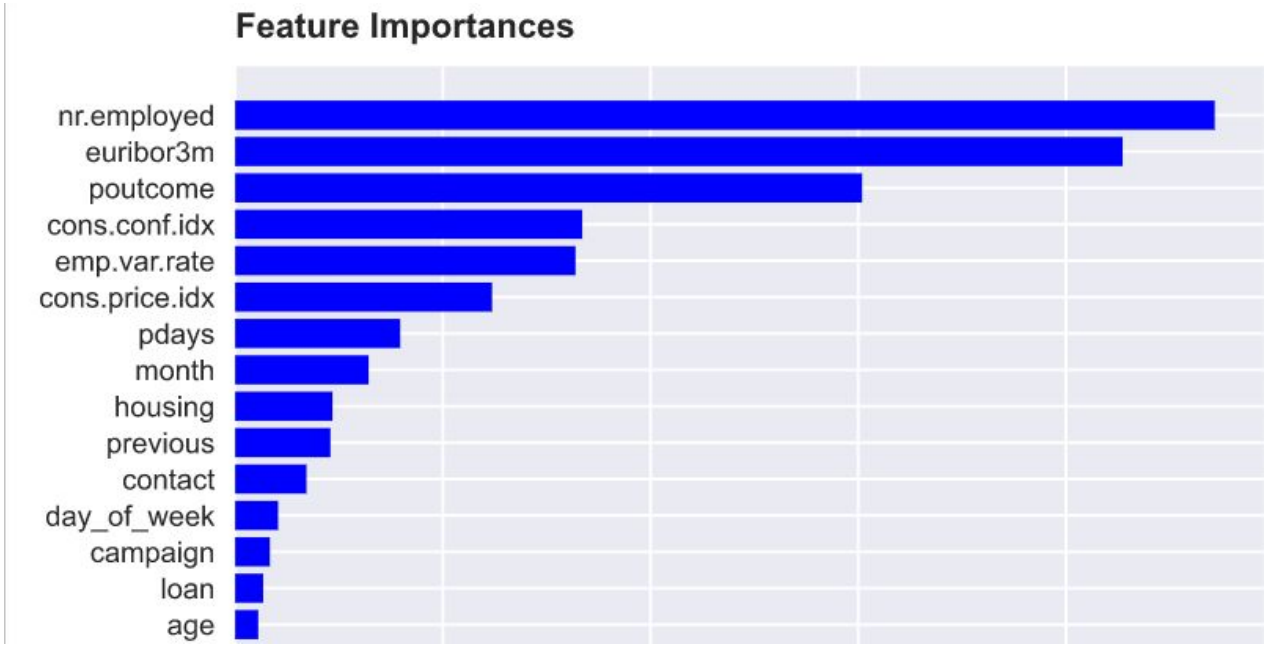
	Train	Validation	Test
TNR	0.866849	0.861875	0.868040
TPR	0.629246	0.619507	0.608791
bACC	0.748047	0.740691	0.738416
ROC	0.816748	0.796147	0.791453
REC	0.629246	0.619507	0.608791
PRE	0.376229	0.364209	0.364234
F1	0.470903	0.458730	0.455780
AP	0.507512	0.471628	0.455635

Mean test score	Max depth.	Max_ Leaf Nodes
0,414	1000	100
0,385	1000	10
0,414	100	100
0,385	100	10
0,408	10	100

5.4. Random Forest:

Feature importance

- Discuss importance of different features
- Forests and feature selection based on importance



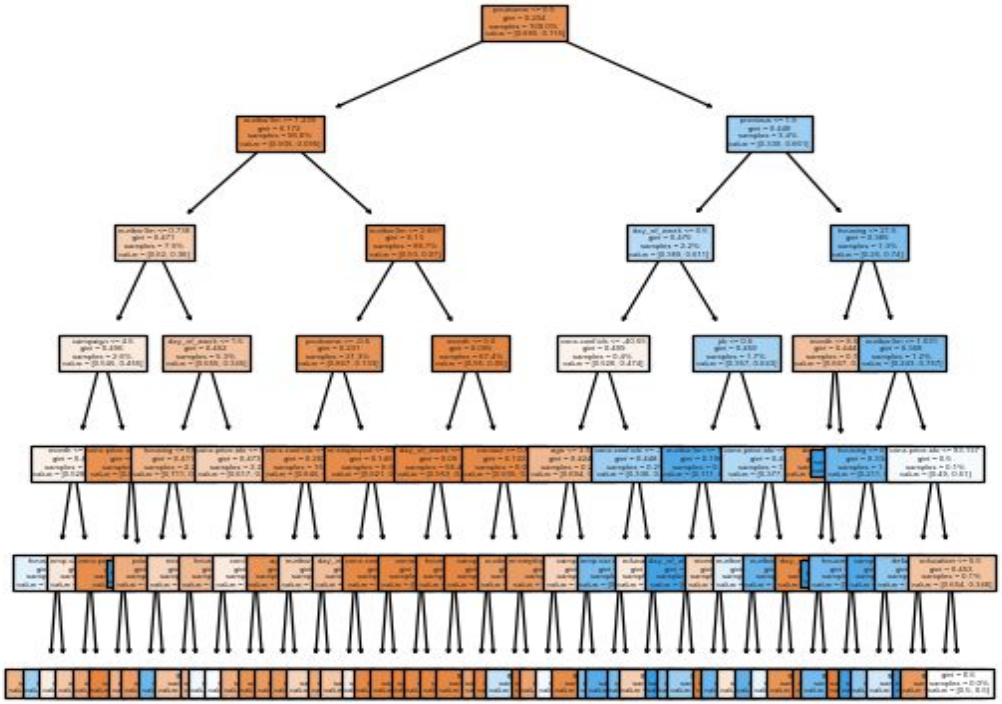

```
nr.employed = 0.23581867160908354
euribor3m = 0.213662798284482
poutcome = 0.15093401974315632
cons.conf.idx = 0.08356367745032177
emp.var.rate = 0.0819466885063493
cons.price.idx = 0.06185600062913342
```

5.4. Random Forest:

The magical forest

- The difficulty of graphical representation actually makes the forest beautiful.

"He who cannot see the forest from the trees, is doomed!"



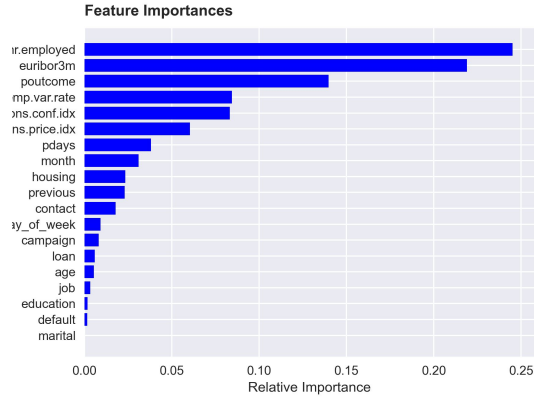
5.4. AdaBoost:

Overview

- What is AdaBoost?
- Differences between AdaBoost and random forest
- Application on our model

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'learning_rate':[0.8],
    'n_estimators':[800],
    'base_estimator':[DecisionTreeClassifier(max_depth=1),DecisionTreeClassifier(max_depth=4)]
}
CV_RFmodel = GridSearchCV(estimator=AB,param_grid=param_grid,scoring="average_precision",n_jobs=-1,cv=2)
CV_RFmodel.fit(X_train,y_train)
grid_results = CV_RFmodel.cv_results_
grid_best_params = CV_RFmodel.best_params_
grid_best_score = CV_RFmodel.best_score_
grid_best_estimator = CV_RFmodel.best_estimator_
print(f"best mean test score: {grid_best_score}, for {grid_best_estimator}")
```

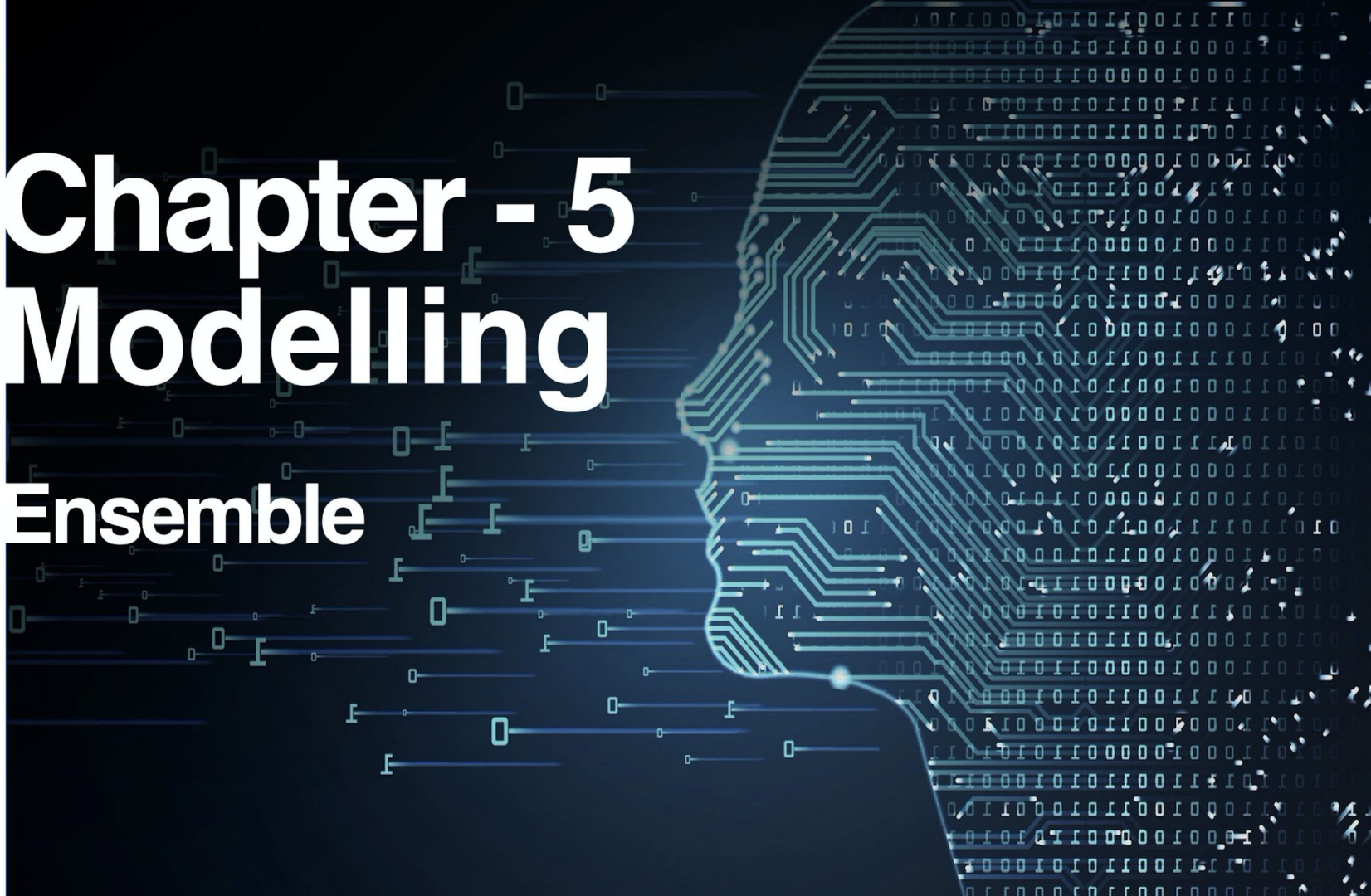
	Train	Validation	Test
TNR	0.985982	0.972650	0.979312
TPR	0.393448	0.262162	0.272630
bACC	0.689715	0.617406	0.625971
ROC	0.878366	0.743267	0.757640
REC	0.393448	0.262162	0.272630
PRE	0.780308	0.548023	0.628993
F1	0.523125	0.354662	0.380386
AP	0.632423	0.374018	0.423666




```
nr.employed = 0.24512965121119204
euribor3m = 0.2190410622256826
poutcome = 0.1397408073745672
emp.var.rate = 0.08445578032649204
```

Chapter - 5 Modelling

Ensemble



5.5. Ensemble Methods:

Gradient Boosting

- There several libraries,e.g. XGBoost and CatBoost, that implement Gradient Tree Boosting which gives us the best result.

```
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
xgb_clf = XGBClassifier(max_depth=3, scale_pos_weight=8)
cat_clf = CatBoostClassifier(eval_metric="AUC", class_weights=[1, 8])
xgb_clf.fit(X_train, y_train)
cat_clf.fit(X_train, y_train)
```

Voting Classifier

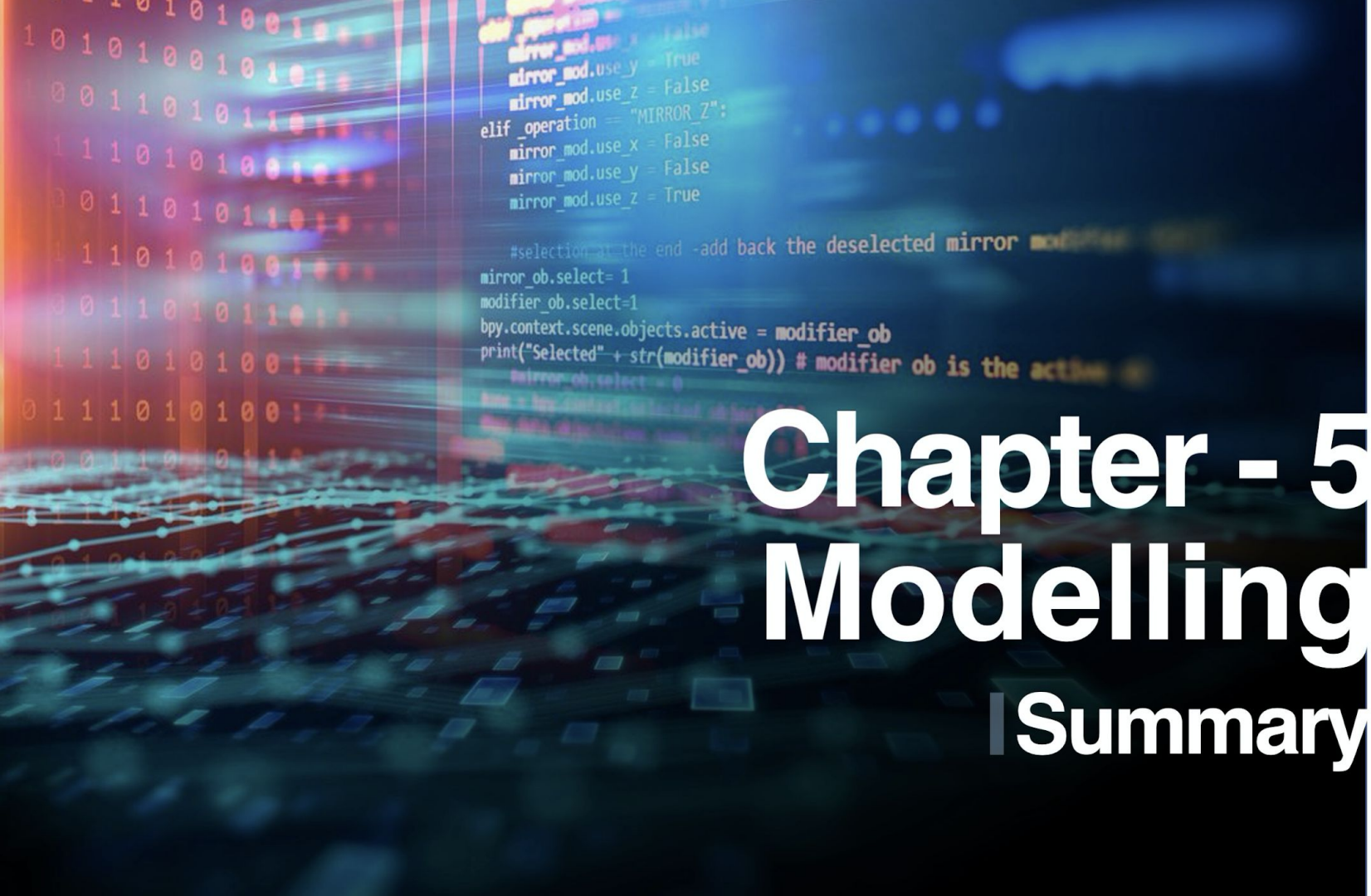
- A voting classifier aggregates the predictions of each classifier and predict the class that gets the most votes.

```
voting_clf = VotingClassifier(
    estimators=[("cat", cat_clf), ("xgb", xgb_clf)],
    voting="soft")
voting_clf.fit(X_train, y_train)
```

Stacking Classifier

- A stacking classifier combines each model's prediction and try to find the best weights.

```
stacking_clf = StackingClassifier(
    estimators=[("cat", cat_clf),("xgb", xgb_clf)])
stacking_clf.fit(X_train, y_train)
```



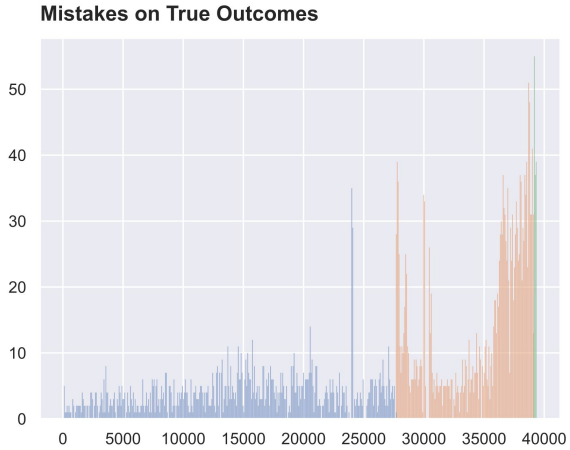
Chapter - 5
Modelling

Summary

5.6. Modelling Summary:

Best Model

- By comparing the models' performance using AP and ROC AUC, XGBoost gives us the best result.
- If we plot the index of FNs, we can see a large quantity of wrong predictions during and after financial crisis.
- There are several reasons that we can speculate.



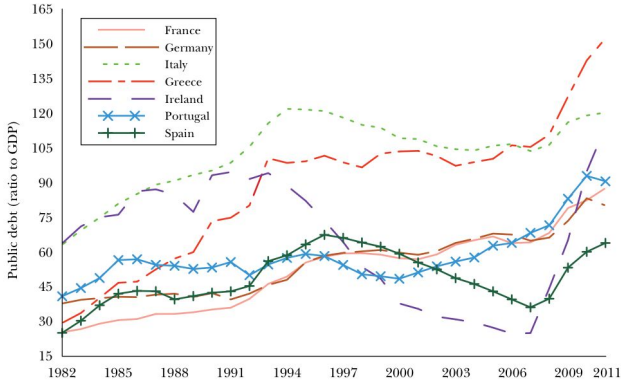
	Train	Validate	Test
TNR	0.866	0.862	0.860
TPR	0.660	0.624	0.658
bACC	0.763	0.743	0.759
ROC	0.846	0.796	0.810
REC	0.660	0.624	0.658
PRE	0.386	0.364	0.374
AP	0.532	0.455	0.478

5.6. Modelling Summary:

Learn From Mistakes

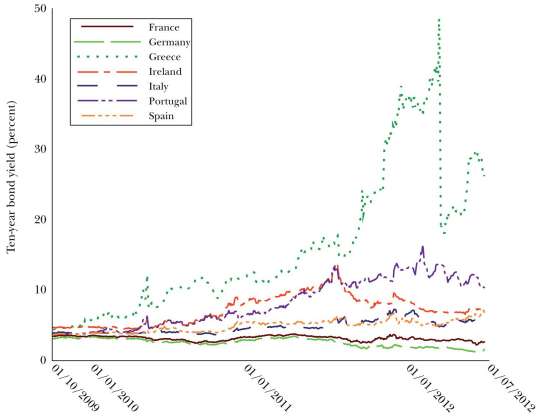
- Financial crisis in 2008 lowered people's expectation of their future income and altered their investment choice. Thus, people reacted differently to term deposit before and after the financial crisis.
- Portugal was one of the European countries that suffered the fiscal crisis most. In May 2010, Portugal 10-year bonds yields started to diverge from German bonds.
- We do not have enough data in 2009 and 2010 to learn this abnormal shift of attitude towards term deposit after the financial crisis.

Figure 1
The Evolution of Public Debt, 1982–2011



Source: Data from IMF Public Debt Database.

Figure 2
Yields on Ten-Year Sovereign Bonds, October 2009 to June 2012
(percent)



Source: Author's calculations based on data from Datastream.

Our exploratory and modelling process using the Bank Marketing dataset shows that:

- Average Precision and ROC AUC are the prefered metrics due to their flexibility on the models' thresholds and interpretability for imbalanced datasets.
- Ensemble methods show the best performance at 0.81 ROC AUC and at 0.48 Average Precision.
- Financial crisis and Portugal's debt crisis may alter people's deposit choices. The lack of data in 2009 and 2010 makes it difficult for the algorithms to further improve.

References:

Davis, J., & Goadrich, M. (2006). The Relationship Between Precision-Recall and ROC curves. Proceedings of the 23rd International Conference on Machine Learning - ICML ’06, 233–240.

Géron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O’Reilly Media, Inc.

Lane, P. R. (2012). The European Sovereign Debt Crisis. Journal of Economic Perspectives, 26(3), 49–68. https://doi.org/10.1257/jep.26.3.49

Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. O’Reilly Media, Inc.

Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. PLOS ONE, 10(3), e0118432.

Bank Marketing Data Analysis and Prediction

Team 3: Jiawei Li, Strahinja Trenkic, Qiqi Zhou, Fan Jia

Frankfurt School
of Finance & Management
German Excellence. Global Relevance.