Homework #5:

1. $Y_{t+1} = (1-\omega)\mu + \omega Y_t + \epsilon_t$ for $t = 0, 1, \ldots,$

where $0 < \omega < 1$ and $\mu = \epsilon_0, \epsilon_1$ are i.i.d random varable.

mean $= 0$ and variance σ_ϵ^2. Y_0, independent, mean μ and

variance $\sigma_\epsilon^2 / (1-\omega^2)$.

a) Derive formulas for $E|Y_1|$ and $Var(Y_1)$

Given random variables $= 0$, and variance σ_ϵ^2.

Rewrite the time series:

$$Y_1 = (1-\omega)\mu + \omega Y_0 + \epsilon_0 \qquad t = \underline{\underline{0}}, 1 \ldots$$

$$
\begin{aligned}
E|Y_1| &= E|(1-\omega)\mu + \omega Y_0 + \epsilon_0| \\
&= E|(1-\omega)\mu| + E|\omega Y_0| + E|\epsilon_0| \\
&= (1-\omega)\mu + \omega\mu + 0 \\
&= (1-\omega)\mu + \omega\mu \Rightarrow (1-\omega+\omega)\mu \\
&= (1)\mu \\
&= \boxed{\mu}
\end{aligned}
$$

$$\boxed{E|Y_1| = \mu}$$

$$
\begin{aligned}
Var|Y_1| &= Var|(1-\omega)\mu + \omega Y_0 + \epsilon_0| \\
&= Var|(1-\omega)\mu + Var(\omega Y_0) + Var(\epsilon_0) \\
&= 0 + \omega^2 Var(Y_0) + \sigma_\epsilon^2
\end{aligned}
$$

plug values from given information

simplify

$$= \omega^2\left(\frac{\sigma_\epsilon^2}{1-\omega^2}\right) + \sigma_\epsilon^2 = (\omega^2)\left(\frac{\sigma_\epsilon^2}{1-\omega^2} + \frac{\sigma_\epsilon^2 (1-\omega^2)}{}\right)$$

$$= \frac{(1-\omega^2+\omega^2)\sigma_\epsilon^2}{1-\omega^2} \Rightarrow \boxed{Var(Y_1) = \frac{\sigma_\epsilon^2}{1-\omega^2}}$$

b) In statistics, there is topic by induction which would play an important role. Basically it says that Y_1 and ϵ_1 will be the same thing as Y_0 and ϵ_0 vise-versa. The reason is if you prove it once it stays the same for example if you have a value (limit) it will eventually get you the same value or close to it.

$$E|Y_t| = E\left[(1-\omega)\mu + \omega Y_0 + \epsilon_0\right]$$
$$= E\left[(1-\omega)\mu + E(\omega Y_0) + E(\epsilon_0)\right]$$
$$E(Y_t| = (1-\omega)\mu + \omega\mu + 0 = \mu$$

$$\boxed{E|Y_t| = \mu}$$

$$Var|Y_t| = Var((1-\omega)\mu + \omega Y_0 + \epsilon_0)$$
$$= Var[0] + \omega^2 Var(Y_0) + Var[\epsilon_0]$$
$$= 0 + \omega^2 Var|Y_0| + \sigma_\epsilon^2$$
$$= \frac{\omega^2 \sigma_\epsilon^2}{(1-\omega^2)} + \sigma_\epsilon^2$$
$$= \frac{\omega^2 \sigma_\epsilon^2}{(1-\omega^2)} + \frac{(1-\omega^2)\sigma_\epsilon^2}{(1-\omega^2)} = \frac{(1-\omega^2+\omega^2)\sigma_\epsilon^2}{(1-\omega^2)}$$

$$\boxed{Var(Y_t) = \frac{\sigma_\epsilon^2}{(1-\omega^2)}}$$

6 Suppose now that $w=1$ and Y_0 is deterministically equal to μ (hence Y_0 has variance zero). Derive formulas for $E|Y_1|$ and $Var(Y_1)$

$w=1,\ Y_0=\mu \Rightarrow Y_0=0$

$$E|Y_1| = E|(1-w)\mu + wY_0 + \epsilon_0|$$
$$= E|0| + E|wY_0| + E|\epsilon_0|$$
$$\Rightarrow 0 + w(\mu) + 0$$
$$\boxed{E|Y_1| = \mu}$$

$$Var(Y_1) = Var((1-w)\mu + wY_0 + \epsilon_0)$$
$$= Var|0| + w^2 Var|Y_0| + Var|\epsilon_0|$$
$$= 0 + 0 + \sigma_\epsilon^2$$
$$\boxed{Var(Y_1) = \sigma_\epsilon^2}$$

d) Again, suppose that $w=1$ and Y_0 is deterministically equal to μ (hence Y_0 has variance zero). Give conjecture for formulas for $E|Y_t|$ and $var(Y_t)$ for $t>1$ and explain the intuition for your conjecture.

$E|Y_2| = E|(1-w)\mu| + E|wY_1| + E|\epsilon_0|$
$= 0 + \mu + 0 \Rightarrow E|Y_2| = \mu$

$\boxed{E|Y_2| = \mu}$ Basically the same, so conjuring it makes sense when $t>1$.

$Var(Y_2) = var(0) + w^2 var|Y_1| + var|\epsilon_0|$
$= 0 + \sigma_\epsilon^2 + \sigma_\epsilon^2 \Rightarrow 2\sigma_\epsilon^2$

$\boxed{Var(Y_2) = 2\sigma_\epsilon^2}$

so for variance it would change by whatever t value is.

$\boxed{Var|Y_t| = t\sigma_\epsilon^2}$

HW5 Skeleton Code

Please note that this skeleton code is provided to help you with homework. Full description of each question can be found on HW5.pdf, so please read instruction of each question carefully. There might be some questions that is not presented in this code.

In [53]:
```
import os
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
```

Q. Changing HTML Text to Plain Text

The Python library **BeautifulSoup** is useful for dealing with html text. In order to use this library, you will need to install it first by running the following command: **conda install beautifulsoup4** in the terminal.
In the code, you can import it by running the following line:
from bs4 import BeautifulSoup

In [54]:
```
!pip install beautifulsoup4 ## this my computer configuration, so it makes sense for me to use pip install.
from bs4 import BeautifulSoup
```

```
ERROR: Invalid requirement: '##'
```

In [55]:
```
#Read our data file
df_train = pd.read_csv('stack_stats_2023_train.csv') #Todo
df_test = pd.read_csv('stack_stats_2023_test.csv') #Todo
```

In [56]:
```
df_train.head()
```

Out[56]:

	Id	Score	Body	Title	Tags
0	500620	0	I have the following head of data \n<pre...	Linear mixed effect model; degrees of freedom ...	<r><lme4-nlme>
1	483677	0	<p>I want to predict a multivariate time serie...	Does LSTM without delayed inputs work as a dee...	<machine-learning><time-series><neural-network...
2	464381	1	<p>Bolker (2015) talks of a research scenario ...	Mixed models: Why are deviations of each level...	<mixed-model><terminology>
3	494560	0	<p>there is a reference to the <span class="ma...	Understanding leverage and influence	<machine-learning><inference><intuition><lever...
4	466706	3	<p>We want to estimate <span class="math-conta...	Goodness of Fit ot Least Squares with known me...	<regression><chi-squared-test><least-squares><...

In [57]:
```
#Cleaning 'Body'
#Change HTML Text to Plain text using get_text() function from BeautifulSoup
#If you are not familiar with the apply method, please check discussion week 10 lecture and code.

df_train['Body'] = df_train['Body'].apply(lambda get_text: BeautifulSoup(get_text, "html.parser").get_text()) #Todo
#Manually cleaned up newline tag \n and tab tag \t.
df_train['Body'] = df_train['Body'].apply(lambda get_text: get_text.replace('\n', '')) #Todo
df_train['Body'] = df_train['Body'].apply(lambda get_text: get_text.replace('\t', '') ) #Todo

#If you need any other cleaning process, please uncomment the below.
df_train['Body'] = df_train['Body'].replace(r'\r+|\n+|\t+','', regex=True) # this is old regex from cs61b.
#Cleaning Tags
#This would be somewhat similar to the above.
#Todo: Clean Tags, please feel free to add any lines below
df_train['Tags'] = df_train['Tags'].apply(lambda text: text.replace('>', ' ').replace('<', '')) #todo


#Todo: Repeat the same process for test dataset
df_test['Body'] = df_test['Body'].apply(lambda get_text: BeautifulSoup(get_text, "html.parser").get_text()) #Todo
df_test['Body'] = df_test['Body'].apply(lambda get_text: get_text.replace('\n', '')) #Todo
df_test['Body'] = df_test['Body'].apply(lambda get_text: get_text.replace('\t', '') ) #Todo
df_test['Body'] = df_test['Body'].replace(r'\r+|\n+|\t+','', regex=True) # this is old regex from cs61b.
df_test['Tags'] = df_test['Tags'].apply(lambda text: text.replace('>', ' ').replace('<', '')) #todo
```

In [58]:
```
#df_train -- there are some weird letters but not sure if we have to get rid of them as well but the regex is working
#df_test --> looks much cleaner
```

Q. Basic Text Cleaning and Merging into a single Text data

Change to Lower Case, Remove puncuation, digits,

In [59]:
```
#Change to Lowercase
df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].applymap(str.lower) #Todo, do you see why we us
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].applymap(str.lower) #Todo
```

In [60]:
```
#df_train --> Working fine, lowered case the both cases.
#df_test  --> Working fine, lowered case the both cases.
```

In [61]:
```
#Remove Punctations
from string import punctuation

#You can get this function from our discussion session code. However, we leave it as a blank for a practice.
def remove_punctuation(document):
    no_punct = ''.join([character for character in document if character not in punctuation])  #Todo
    ## like the above said, from the lab notes.
    return no_punct

df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].applymap(remove_punctuation)#Todo
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].applymap(remove_punctuation) #Todo
```

In [62]:
```
#Remove Digits

def remove_digit(document):

    no_digit = ''.join([character for character in document if not character.isdigit()]) #Todo

    return no_digit

df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].applymap(remove_digit) #Todo
df_test[['Body','Title','Tags']] =  df_test[['Body','Title','Tags']].applymap(remove_digit) #Todo
```

Tokenization and Remove Stopwords and do stemming

In [63]:
```
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
df_train[['Body','Title','Tags']] =  df_train[['Body','Title','Tags']].applymap(word_tokenize) #Todo
df_test[['Body','Title','Tags']] =   df_test[['Body','Title','Tags']].applymap(word_tokenize) #Todo
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\jackf\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [64]:
```
#Remove Stopwords

from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def remove_stopwords(document):

    words = [word for word in document if not word in stop_words]  #Todo

    return words

df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].applymap(remove_stopwords)#Todo
df_test[['Body','Title','Tags']] =  df_test[['Body','Title','Tags']].applymap(remove_stopwords)#Todo
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jackf\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [65]:
```
#We use porter stemming

from nltk.stem import PorterStemmer

porter = PorterStemmer()

def stemmer(document):

    stemmed_document = [porter.stem(word) for word in document] #Todo

    return stemmed_document

df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].applymap(stemmer) #Todo
df_test[['Body','Title','Tags']] =  df_test[['Body','Title','Tags']].applymap(stemmer)#Todo
```

Let's Check our dataframe

In [66]:
```
df_train.head(5)
```

Out[66]:

	Id	Score	Body	Title	Tags
0	500620	0	[follow, head, data, p, p, id, type, host, ca,...	[linear, mix, effect, model, degre, freedom, p...	[r, lmenlm]
1	483677	0	[want, predict, multivari, time, seri, time, s...	[lstm, without, delay, input, work, deep, net]	[machinelearn, timeseri, neuralnetwork, lstm, ...
2	464381	1	[bolker, talk, research, scenario, site, group...	[mix, model, deviat, level, group, factor, val...	[mixedmodel, terminolog]
3	494560	0	[refer, ith, diagon, entri, h, hxxtxxt, defini...	[understand, leverag, influenc]	[machinelearn, infer, intuit, leverag]
4	466706	3	[want, estim, beta, fori, xbeta, epsilonwher, ...	[good, fit, ot, least, squar, known, measur, u...	[regress, chisquaredtest, leastsquar, goodness...

Q. Treat Three text data independently and merge into one column

In [67]:
```
#Treat Three types of data independently
#let's define functions that will help this operation

def add_body(document):

    string = "_body"
    added_document = [x + string for x in document] #Todo

    return added_document

def add_title(document):
    string = "_title"
    added_document = [x + string for x in document] #Todo

    return added_document

def add_tags(document):
    string = "tags"
    added_document = [x + string for x in document] #Todo

    return added_document
```

In [68]:
```
df_train['Body'] = df_train['Body'].apply(add_body)
df_train['Title'] = df_train['Title'].apply(add_title)
df_train['Tags'] = df_train['Tags'].apply(add_tags)

df_test['Body'] = df_test['Body'].apply(add_body)
df_test['Title'] = df_test['Title'].apply(add_title)
df_test['Tags'] = df_test['Tags'].apply(add_tags)
```

In [69]:
```
#Now we need to merge all those 3 columns into a single column. Implement this below.
df_train['text'] = df_train["Body"] + df_train["Title"] + df_train["Tags"]#Todo
df_test['text'] =  df_test["Body"] + df_test["Title"] + df_test["Tags"]  #Todo
```

Let's check our DataFrame

In [70]:
```
#df_train["text"][0] --> just merfed them together.
```

In [71]:
```
df_train.head(5)
```

Out[71]:

	Id	Score	Body	Title	Tags	text
0	500620	0	[follow_body, head_body, data_body, p_body, p_...	[linear_title, mix_title, effect_title, model_...	[rtags, lmenlmtags]	[follow_body, head_body, data_body, p_body, p_...
1	483677	0	[want_body, predict_body, multivari_body, time...	[lstm_title, without_title, delay_title, input...	[machinelearntags, timeseritags, neuralnetwork...	[want_body, predict_body, multivari_body, time...
2	464381	1	[bolker_body, talk_body, research_body, scenar...	[mix_title, model_title, deviat_title, level_t...	[mixedmodeltags, terminologtags]	[bolker_body, talk_body, research_body, scenar...
3	494560	0	[refer_body, ith_body, diagon_body, entri_body...	[understand_title, leverag_title, influenc_title]	[machinelearntags, infertags, intuittags, leve...	[refer_body, ith_body, diagon_body, entri_body...
4	466706	3	[want_body, estim_body, beta_body, fori_body, ...	[good_title, fit_title, ot_title, least_title,...	[regresstags, chisquaredtesttags, leastsquarta...	[want_body, estim_body, beta_body, fori_body, ...

Q. Detokenize and convert to document term matrices

In [72]:
```
#Merge Three text column into one column and detokenize

from nltk.tokenize.treebank import TreebankWordDetokenizer
from sklearn.feature_extraction.text import CountVectorizer

text_train = df_train['text'].apply(TreebankWordDetokenizer().detokenize) #Todo: Detokenize your tokenized text data
countvec_train = CountVectorizer(min_df = 0.001) #Todo: Define your own CountVectorizer here
 ## this to offset the memeory problem in the pandas becuase its too big
sparse_dtm_train = countvec_train.fit_transform(text_train) #Todo: Fit and Transform your Countvectorizer and return s
```

In [73]:
```
#Todo: Do same on the test set.
text_test = df_test['text'].apply(TreebankWordDetokenizer().detokenize)
#sparse_dtm_test =  CountVectorizer(min_df = 0.001) ## this breaks the code apparently on discord
sparse_dtm_test = countvec_train.transform(text_test)
```

In [74]:
```
#Convert the sprase dtm to pandas DataFrame.
dtm_train =  pd.DataFrame(sparse_dtm_train.toarray(), columns = countvec_train.get_feature_names_out(), index=df_trai
dtm_test  =  pd.DataFrame(sparse_dtm_test.toarray(), columns = countvec_train.get_feature_names_out(), index=df_test.
```

Q. Change dependent variable to binary variable

In [75]:
```
#Change 'Score' to a binary variable, which indicates whether the question is good or not.

y_train = (df_train['Score'] >= 1).astype(int) #Todo
y_test =  (df_test['Score'] >= 1).astype(int) #Todo
```

In [76]:
```
#Add y_train and y_test to your data frame if it is needed. Drop unnecessary columns
df_train['Binary_value'] = y_train
df_test['Binary_value'] = y_test
df_train.drop(columns = ['Score','Id'], inplace = True) ## these column seem useless. so I am assuming its these two
df_test.drop(columns = ['Score','Id'], inplace = True)
```

Let's check our DataFrame

In [78]:
```
df_train.head(5)
```

Out[78]:

	Body	Title	Tags	text	Binary_value
0	[follow_body, head_body, data_body, p_body, p_...	[linear_title, mix_title, effect_title, model_...	[rtags, lmenlmtags]	[follow_body, head_body, data_body, p_body, p_...	0
1	[want_body, predict_body, multivari_body, time...	[lstm_title, without_title, delay_title, input...	[machinelearntags, timeseritags, neuralnetwork...	[want_body, predict_body, multivari_body, time...	0
2	[bolker_body, talk_body, research_body, scenar...	[mix_title, model_title, deviat_title, level_t...	[mixedmodeltags, terminologtags]	[bolker_body, talk_body, research_body, scenar...	1
3	[refer_body, ith_body, diagon_body, entri_body...	[understand_title, leverag_title, influenc_title]	[machinelearntags, infertags, intuittags, leve...	[refer_body, ith_body, diagon_body, entri_body...	0
4	[want_body, estim_body, beta_body, fori_body, ...	[good_title, fit_title, ot_title, least_title,...	[regresstags, chisquaredtesttags, leastsquarta...	[want_body, estim_body, beta_body, fori_body, ...	1

(b) Please read the instruction carefully in the pdf.

In [79]:
```
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
```

BaseLine Model

Its just a simple baseline model. Where you set the values to 0, to create the baseline, and try to get models that are better than this model in terms of accuracy and TPR and FPR. So thats what I will be doing in other models to look out for that.

In [80]:
```
from sklearn.metrics import confusion_matrix
zero_label_count = np.count_nonzero(y_train == 0)
one_label_count = np.count_nonzero(y_train == 1)

label_counts = pd.Series({'Zero': zero_label_count, 'One': one_label_count})

baseline_acc = zero_label_count /(zero_label_count  + one_label_count)
print(baseline_acc)
baseline_TPR = 0
baseline_FPR = 0
baseline_PRE = 0
```

```
0.5024159609289759
```

Logistic Regression:

Logistic regression, and the split point for the p-value will be 0.5, its kinda like basic model. But will improve the p-value of the model to improve the model. Everything else should be the same. This a good model because its like the "baseline" model of machine learning for classification, on which we can build more models such as Random forest regression model.

In [81]:
```
logreg = sm.Logit(y_train,dtm_train).fit()
log_prob = logreg.predict(dtm_test)

log_pred = pd.Series([1 if x > 0.5 else 0 for x in log_prob], index = log_prob.index)
cm = confusion_matrix(y_test, log_pred)
print ("Confusion Matrix : \n", cm)
log_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
print(log_acc)
log_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
log_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
log_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Optimization terminated successfully.
         Current function value: 0.510529
         Iterations 9
Confusion Matrix :
 [[2365 1875]
 [1838 2171]]
0.549884834525397
```

Decision Tree Classifier

For this model I used the ccp_alpha values from the lab, they did fine there so will use the same values. Random_State value doesn't matter,its just there to produce the same result on different computer by using the same value. Then had used the CV at 5, one reason is because I noticed from past homeworks and labs, increasing the value after 5 doesn't really change the results by that much. The big reason was to prevent this code to be running for another 40 minutes. So 5 is a good value.

In [30]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
grid_values = {'ccp_alpha': np.linspace(0, 0.01, 10)}
dtc = DecisionTreeClassifier(random_state = 42)
dtc_cv = GridSearchCV(dtc, param_grid = grid_values, cv = 5).fit(dtm_train, y_train)

dtc_pred = dtc_cv.best_estimator_.predict(dtm_test)
cm = confusion_matrix(y_test, dtc_pred)
print ("Confusion Matrix : \n", cm)
dtc_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
dtc_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
dtc_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
dtc_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix :
 [[3476  764]
 [2978 1031]]
```

In [31]:
```
dtc_acc
```

Out[31]:
```
0.5463692568796218
```

Random Forest with CV

For max_features I just used the built in functions for random forest, gave it the values, auto, sqrt and log2. For min_samples_leaf went with the value 5, its large enough for it give good predictive value prediction. The n_estimators are 300, I went with 500, got the same result, and the code takes forever to run so 300 is good to run with. CV is 5 simply large enough and not to small to cause any problems with prediction.

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

grid_values = {
    'max_features': ['auto', 'sqrt', 'log2'],
    'min_samples_leaf': [5],
    'n_estimators': [300],
    'random_state': [42]
}
rf = RandomForestClassifier()
rf_cv = GridSearchCV(rf, param_grid = grid_values, cv=5)
rf_cv.fit(dtm_train, y_train)

y_pred = rf_cv.best_estimator_.predict(dtm_test)
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix: \n", cm)
rf_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
print(rf_acc)
rf_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
rf_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
rf_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix:
 [[2628 1612]
 [1754 2255]]
0.5919505394593284
```

Linear Discriminant Analysis

I am using the same code from the lab here, and didn't put any hyper parameters in it, I want to use the first 4 for this assignment, and just did this because to see if it was good or bad model

In [33]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(dtm_train, y_train)

y_pred = lda.predict(dtm_test)
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix: \n", cm)
lda_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
print(lda_acc)
lda_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
lda_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
lda_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix:
 [[2433 1807]
 [1863 2146]]
0.5550975875863741
```

In [34]:
```
#Create Comparison Table
#These lines are provided for you to help construct a comparison table.
#It is not requred to follow this format. + You need to find ACC, TPR, FPR, PRE for each model that you choose.
comparison_data = {'Baseline':[baseline_acc,baseline_TPR,baseline_FPR, baseline_PRE],
                   'Logistic Regression':[log_acc,log_TPR,log_FPR, log_PRE],
                   'Decision Tree Classifier':[dtc_acc,dtc_TPR,dtc_FPR,dtc_PRE],
                   'Random Forest with CV':[rf_acc,rf_TPR, rf_FPR,rf_PRE],
                  'Linear Discriminant Analysis':[lda_acc,lda_TPR, lda_FPR,lda_PRE]}

comparison_table = pd.DataFrame(data=comparison_data, index=['Accuracy', 'TPR', 'FPR','PRE']).transpose()
comparison_table.style.set_properties(**{'font-size': '12pt',}).set_table_styles([{'selector': 'th', 'props': [('font
comparison_table
```

Out[34]:

	Accuracy	TPR	FPR	PRE
Baseline	0.502416	0.000000	0.000000	0.000000
Logistic Regression	0.549885	0.541532	0.442217	0.536579
Decision Tree Classifier	0.546369	0.257171	0.180189	0.574373
Random Forest with CV	0.591951	0.562484	0.380189	0.583139
Linear Discriminant Analysis	0.555098	0.535296	0.426179	0.542879

Answer 2b: I selected all models and did them, but I want to pick the Baseline, Logistic Regression, Decision Tree Classifier and Random Forest with CV. We can see that all of their accuray is better than the Baseline model, which is a good thing, and Random Forest with CV has the highest Accuracy. For all the models used regression Classification because tahts what we are working with here. I explained all models and why I picked their parameters underneath their title. From these models its good to pick the Random Forest, which has the highest Accuracy, but also has high TPR ratio to FPR to the othe rmodels, so it would make sense to pick this model up.

Report details of your training procedures and final comparisons on the test set in this cell. Use your best judgment to choose a final model and explain your choice.

In [33]:
```python
import time

def bootstrap_validation(test_data, test_label, train_label, model, metrics_list, sample=100, random_state=42):
    tic = time.time()
    n_sample = sample
    n_metrics = len(metrics_list)
    output_array = np.zeros([n_sample, n_metrics])
    output_array[:] = np.nan
    print(output_array.shape)
    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index), replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
        bs_predicted = model.predict(bs_data)

        for metrics_iter in range(n_metrics):
            metrics = metrics_list[metrics_iter]
            # Ensure that metrics receive the correct number of arguments
            output_array[bs_iter, metrics_iter] = metrics(bs_predicted, bs_label, train_label)
    # Print or log the time if needed
    # print("Elapsed time:", time.time() - tic)

    output_df = pd.DataFrame(output_array)
    return output_df
```

In [30]:
```python
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import GridSearchCV
grid_values = {'max_features': np.linspace(20, 40, 5, dtype='int32'),
               'min_samples_leaf': [4],
               'n_estimators': [100],
               'random_state': [42]}

rf = RandomForestClassifier()
rf_cv = GridSearchCV(rf, param_grid=grid_values, cv=5, n_jobs=-1)
rf_cv.fit(dtm_train, y_train)
best_rf_model = rf_cv.best_estimator_

import joblib
joblib.dump(best_rf_model, 'best_rf_model.joblib')
```

Out[30]:
```
['best_rf_model.joblib']
```

In [46]:
```
import time
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score
import inspect

def true_positive_rate(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    tpr = cm[1, 1] / (cm[1, 0] + cm[1, 1])
    return tpr

def false_positive_rate(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    fpr = cm[0, 1] / (cm[0, 0] + cm[0, 1])
    return fpr

def bootstrap_validation(test_data, test_label, train_label, model, metrics_list, sample=100, random_state=42):
    tic = time.time()
    n_sample = sample
    n_metrics = len(metrics_list)
    output_array = np.zeros([n_sample, n_metrics])
    output_array[:] = np.nan

    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index), replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
        bs_predicted = model.predict(bs_data)

        for metrics_iter in range(n_metrics):
            metrics_func = metrics_list[metrics_iter]

            if metrics_func == accuracy_score:
                # Special handling for accuracy_score with 4 arguments
                output_array[bs_iter, metrics_iter] = metrics_func(bs_label, bs_predicted)
            elif metrics_func == recall_score:
                # Special handling for recall_score with 7 arguments
                output_array[bs_iter, metrics_iter] = recall_score(bs_label, bs_predicted, labels=None,
                                                                   pos_label=1, average='binary', sample_weight=None)
            elif metrics_func == precision_score:
                # Special handling for precision_score with 7 arguments
                output_array[bs_iter, metrics_iter] = precision_score(bs_label, bs_predicted, labels=None,
                                                                      pos_label=1, average='binary', sample_weight=No
            else:
                num_expected_args = len(inspect.signature(metrics_func).parameters)

                if num_expected_args == 2:
                    output_array[bs_iter, metrics_iter] = metrics_func(bs_predicted, bs_label)
                elif num_expected_args == 3:
                    output_array[bs_iter, metrics_iter] = metrics_func(bs_predicted, bs_label, train_label)
                else:
                    print(f"Metric function '{metrics_func.__name__}' has {num_expected_args} arguments.")
                    raise ValueError("Unsupported number of arguments for metric function.")

    output_df = pd.DataFrame(output_array)
    print("Elapsed time:", time.time() - tic)
    return output_df

# Example usage:
metrics_list_extended = [accuracy_score, recall_score, precision_score, true_positive_rate, false_positive_rate]

# Assuming dtm_test, y_test, y_train, and best_rf_model are defined
bs_output_extended = bootstrap_validation(dtm_test, y_test, y_train, best_rf_model,
                                          metrics_list=metrics_list_extended,
                                          sample = 1000)


quantiles_025_extended = bs_output_extended.quantile(0.025)
quantiles_975_extended = bs_output_extended.quantile(0.975)
mean_values_extended = bs_output_extended.mean()
std_dev_values_extended = bs_output_extended.std()

# Print or log the results for the extended metrics
print("Bootstrap Evaluation (Extended Metrics):")
print("0.025 Quantile of Performance Metrics:")
print(quantiles_025_extended)
print("\n0.975 Quantile of Performance Metrics:")
print(quantiles_975_extended)
print("\nMean Performance Metrics:")
```

```
print(mean_values_extended)
print("\nStandard Deviation of Performance Metrics:")
print(std_dev_values_extended)
```

```
Elapsed time: 905.2332525253296
Bootstrap Evaluation (Extended Metrics):
0.025 Quantile of Performance Metrics:
0    0.571221
1    0.534488
2    0.557264
3    0.557264
4    0.395804
Name: 0.025, dtype: float64

0.975 Quantile of Performance Metrics:
0    0.592560
1    0.564667
2    0.588084
3    0.588084
4    0.425927
Name: 0.975, dtype: float64

Mean Performance Metrics:
0    0.581744
1    0.549195
2    0.572769
3    0.572769
4    0.410421
dtype: float64

Standard Deviation of Performance Metrics:
0    0.005512
1    0.007894
2    0.008162
3    0.008162
4    0.007467
dtype: float64
```

Report Bootstrap Analysis in this cell

Answer Bootstrap Anaylsis: I picked random forest, as I think that was good model. And I ran Bootstrap on this model, and its a good model, because if you lookat the mean, the average was around .58, which is only 0.1 less than the model did prediction the first time I ran it. The STD is also very small value, indiciating no overfitting or underfitting. You can see from the 0.975 quanitile the perforamnce metrics value hover around 0.57 which is really good for us.

(c)

Answer 2c: I picked Logistic regression, even though its not the highest in terms of its precision, this model would be the easiest to see the difference between fpr and tpr once you fix the value, it will make different on what those values are afterwards. Also the company would rather like to know their FPR and TRP to determine their threshold value, because at the end of the day its a business, and also we get 15 values, so I believe using logistic regression is the best usage of a model here. We need to make the TPR value between fpr and tpr to be greater than 0.15 based on the accuracy curve we make.

In [ ]:
```
# original model
logreg = sm.Logit(y_train,dtm_train).fit()
log_prob = logreg.predict(dtm_test)
log_pred = pd.Series([1 if x > 0.5 else 0 for x in log_prob], index = log_prob.index)
cm = confusion_matrix(y_test, log_pred)
print ("Confusion Matrix : \n", cm)
log_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
log_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
log_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
log_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

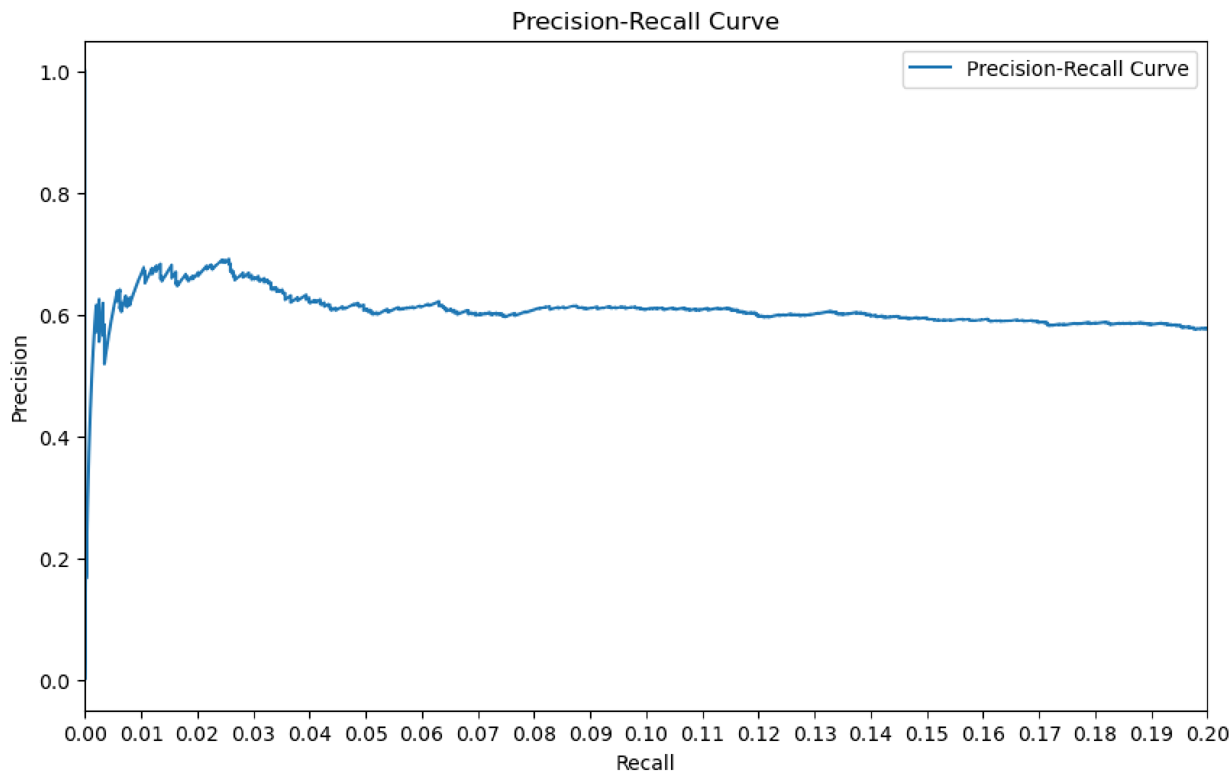
Here I am finding the new model threshold value using the best_f1 score. This doesn't really work out because the value it gives is 0.

In [51]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.metrics import f1_score, confusion_matrix


logreg = sm.Logit(y_train, dtm_train).fit()
log_prob = logreg.predict(dtm_test)

# Create a range of threshold values
thresholds = np.arange(0.01, 1.01, 0.01)

# Initialize variables to store the best threshold and corresponding F1 score
best_threshold = 0
best_f1_score = 0

for threshold in thresholds:
    log_pred = (log_prob > threshold).astype(int)
    cm = confusion_matrix(y_test, log_pred)
    precision = cm[1, 1] / (cm[1, 1] + cm[0, 1])
    recall = cm[1, 1] / (cm[1, 1] + cm[1, 0])
    f1 = 2 * (precision * recall) / (precision + recall)
    if f1 > best_f1_score:
        best_f1_score = f1
        best_threshold = threshold

best_log_pred = (log_prob > best_threshold).astype(int)

print("Best Threshold:", best_threshold)
print("Best F1 Score:", best_f1_score)
```

```
Optimization terminated successfully.
         Current function value: 0.510529
         Iterations 9
Best Threshold: 0.01
Best F1 Score: 0.6500703002233066

C:\Users\jackf\AppData\Local\Temp\ipykernel_8112\2856412304.py:25: RuntimeWarning: invalid value encountered in scal
ar divide
  precision = cm[1, 1] / (cm[1, 1] + cm[0, 1])
```

Going back to the basic precision recall curve, which shows the value that we would want to pick, if you eyeball it, should be around 0.7

In [90]:
```
# Set the size of the figure
plt.figure(figsize=(10, 6))
plt.plot(recall, precision, label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.xlim(0, 1)
plt.xticks(np.arange(0, 1, 0.01))
plt.xlim(0, 0.15)  # Set x-axis limits from 0 to 0.2
plt.xticks(np.arange(0, 0.21, 0.01))
plt.legend()
plt.show()
```



In [85]:
```
best_threshold = 0.69 ## picked the value for the threshold. And then reran the model, reworked it.
```

In [86]:
```
# New Model with threshold vlaue

import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix

logreg = sm.Logit(y_train, dtm_train).fit()
log_prob = logreg.predict(dtm_test)
threshold = best_threshold  # Use the best threshold obtained

log_pred =  pd.Series([1 if x > threshold else 0 for x in log_prob], index = log_prob.index)

cm = confusion_matrix(y_test, log_pred)

print("Confusion Matrix : \n", cm)

log_acc = (cm.ravel()[0] + cm.ravel()[3]) / sum(cm.ravel())
log_TPR = cm.ravel()[3] / (cm.ravel()[2] + cm.ravel()[3])
log_FPR = cm.ravel()[1] / (cm.ravel()[0] + cm.ravel()[1])
log_PRE = cm.ravel()[3] / (cm.ravel()[1] + cm.ravel()[3])
print("Accuracy:", log_acc)
print("True Positive Rate (Recall):", log_TPR)
print("False Positive Rate:", log_FPR)
print("Precision:", log_PRE)
```

```
Optimization terminated successfully.
         Current function value: 0.510529
         Iterations 9
Confusion Matrix :
 [[3083 1157]
 [2533 1476]]
Accuracy: 0.552673051278943
True Positive Rate (Recall): 0.3681716138687952
False Positive Rate: 0.27287735849056605
Precision: 0.5605772882643373
```

Answer 2c:

I tried to first find the pvalue by using code and f-statistics, but the value that it was giving me for the best TPR was the value of 0.00 which is basically the baseline model which isn't what we want. So I used the precision_recall_curve and eyeballed the value to be around 0.70, its little bit less at the highest peak towards the 1.0 precision. The model did improve a little but not too much, its pression went by .3 and difference between TPR and FPR is now 11, but the values drop from .54. Wouldn't be a suitable model but however, you get less FPR and little bit more TPR so it would be good with whatever dicision they make with this given information (the company). We can see thew new value after the threshold value, the PRE is 0.56, while the old model it was 0.53, so .3 improvement. THE old TPR and FPR were 0.54 and 0.44 the new values at 0.36 and FPR is .27, we decreased the fpr and the tpr values (should of increased the tpr value as thats more desirable) but lower FPR or higher TPR would depend on what the company needs to do, and they can just change the threshold value (p_value) to their liking. The accruracy of the old model is 0.54 and the new model its 0.55, so we improved its accracy by only 0.1.

I retrained the model with the new value, as that seemed to appropritate here.

In [ ]: