```
## Comcast Telecom Complaints Dataset
```

Context

Estimation of Comcast Customer Top Complaints.

Acknowledgements

Kaggle Datasets

```
import warnings
warnings.filterwarnings(action='ignore')

import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go

import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
```

```
####! python -m pip install spacy==3.0.0
```

```
###! pip uninstall numpy -y
```

```
###! pip install numpy==1.18.5
```

```
# Install libraries
#!pip install boostaroota==1.3
#!pip install h2o==3.36.1.3
#!pip install ppscore==3.9.0
#!pip install imblearn
```

```
#!pip install optuna
```

```
###! pip install shap
###! pip install catboost
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from tqdm import tqdm
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from sklearn import model_selection as sk_model_selection
from xgboost.sklearn import XGBRegressor
from sklearn.metrics import mean_squared_error,roc_auc_score,precision_score
from sklearn import metrics
import optuna
from boostaroota import BoostARoota
from sklearn.metrics import log_loss
from optuna.samplers import TPESampler
import functools
from functools import partial
```

```
import xgboost as xgb
import joblib
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
import statsmodels.api as sm
import pylab
from xgboost import plot_tree
import shap
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import mean_squared_error,roc_auc_score,precision_score
from sklearn import metrics
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix, recall_score, precision_score, precision_recall_curve, auc, f1_score, \
    average_precision_score, accuracy_score, roc_curve
from sklearn.preprocessing import LabelEncoder
import h2o
from h2o.automl import H2OAutoML
from catboost import Pool, CatBoostRegressor, cv
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.layers import Concatenate, LSTM, GRU
from tensorflow.keras.layers import Bidirectional, Multiply
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC


SEED = 42
```

```
import numpy as np
import pandas as pd
```

```
###!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
####!chmod 600 ~/.kaggle/kaggle.json
```

```
###!pip install keras-tuner
```

```
###!pip install kaggle
```

```
###! kaggle datasets download -d yasserh/comcast-telecom-complaints
```

```
####! unzip /content/comcast-telecom-complaints.zip
```

```
comcast = pd.read_csv("/content/Comcast.csv")
```

```
training_data = comcast.sample(frac=0.7, random_state=25)
testing_data = comcast.drop(training_data.index)
```

```
print(training_data.shape, testing_data.shape)
```

```
(1557, 11) (667, 11)
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
####! python -m textblob.download_corpora
```

```
###! pip install tensorflow==2.8.0
```

```
import tensorflow as tf
```

```
import pandas, numpy, string, textblob
import pickle
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm, decomposition, ensemble
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import xgboost
from keras import layers, models, optimizers
from keras.preprocessing import text, sequence
import matplotlib.pyplot as plt
```

```
import pandas as pd
from xgboost import XGBRegressor as xgb
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import mean_squared_error
import sklearn
import seaborn as sns
from sklearn.model_selection import KFold
```

```
print(training_data.shape, testing_data.shape)
```

```
(1557, 11) (667, 11)
```

```
training_data.columns
```

```
Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone'],
      dtype='object')
```

```
testing_data.columns
```

```
Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone'],
      dtype='object')
```

```
training_data.to_csv("train.csv")
```

```
testing_data.to_csv("test.csv")
```

```
###!pip install -q -U watermark
```

```
###!pip install -qq transformers
```

Setup & Config

```
#@title Setup & Config
import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup
import torch

import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from collections import defaultdict
from textwrap import wrap

from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

%matplotlib inline
%config InlineBackend.figure_format='retina'

sns.set(style='whitegrid', palette='muted', font_scale=1.2)

HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "#8F00FF"]

sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))

rcParams['figure.figsize'] = 12, 8

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

```
     device(type='cuda', index=0)
```

```
####!gdown --id 1S6qMioqPJjyBLpLVz4gmRTnJHnjitnuV
####!gdown --id 1zdmewp7ayS4js4VtrJEHzAheSW-5NBZv
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
print(testing_data.columns, training_data.columns)
```

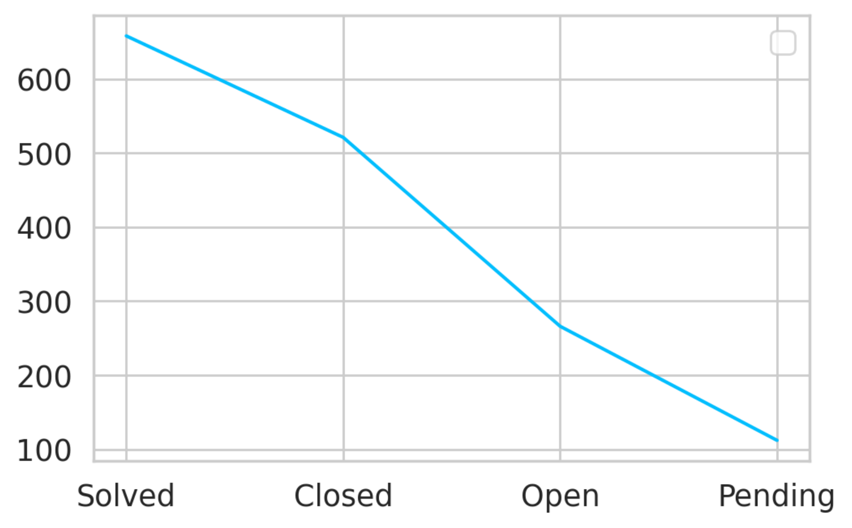
```
Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone'],
      dtype='object') Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone'],
      dtype='object')
```

```
training_data.Status.value_counts()
```

```
Solved     658
Closed     521
Open       266
Pending    112
Name: Status, dtype: int64
```

```
plt.rcParams["figure.figsize"] = [5.50, 3.50]
plt.rcParams["figure.autolayout"] = True

training= training_data.Status.value_counts()
labels = ['Solved', 'Closed', 'Open', 'Pending']
plt.legend(labels, loc="best")
plt.plot(training)
plt.show()
```



```
print(training_data.columns)
```

```
Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone'],
      dtype='object')
```

```
training_data["length"]=training_data['Customer Complaint'].apply(len)
```

```
testing_data["length"]=testing_data['Customer Complaint'].apply(len)
```

```
import string
import re
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers.'''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

```
training_data['Customer Complaint'] = training_data['Customer Complaint'].apply(lambda x: clean_text(x))
```

```
testing_data['Customer Complaint'] = testing_data['Customer Complaint'].apply(lambda x: clean_text(x))
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
training_data['Clean Customer Complaint'] = training_data['Customer Complaint'].apply(lambda x: " ".join(x for x in x.split() if
training_data['Clean Customer Complaint'].head()
```

```
1236                            comcast uverse
541     billingcontract issue related data cap
539                         inaccurate billing
1566        comcast fraudulent billing charges
632          comcast east windsor nj complaint
Name: Clean Customer Complaint, dtype: object
```

```
testing_data['Clean Customer Complaint'] = testing_data['Customer Complaint'].apply(lambda x: " ".join(x for x in x.split() if x
testing_data['Clean Customer Complaint'].head()
```

```
2                                       speed service
4                        comcast working service boot
5     isp charging arbitrary data limits overage fees
10                 billing service asked disconnected
11          yahoo failure restore email search feature
Name: Clean Customer Complaint, dtype: object
```

```
###!pip install nltk
```

```
import nltk
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
from nltk.stem.snowball import SnowballStemmer
```

```
stemmer = nltk.stem.SnowballStemmer('english')
```

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
import re
```

```
def preprocessing(text):
   tokens = [word for word in nltk.word_tokenize(text) if (len(word) > 3 and len(word.strip('Xx/')) > 2 and len(re.sub('\d+', '
   tokens = map(str.lower, tokens)
   stems = [stemmer.stem(item) for item in tokens if (item not in stop_words)]
   return stems
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
vectorizer_tf = TfidfVectorizer(tokenizer=preprocessing, stop_words=None, max_df=0.75, max_features=1000, lowercase=False, ngram
```

```
print(testing_data.columns, training_data.columns)
```

```
Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone', 'length', 'Clean Customer Complaint'],
      dtype='object') Index(['Ticket #', 'Customer Complaint', 'Date', 'Date_month_year', 'Time',
       'Received Via', 'City', 'State', 'Zip code', 'Status',
       'Filing on Behalf of Someone', 'length', 'Clean Customer Complaint'],
      dtype='object')
```

```
training_data['encoded_Status'] = training_data['Status'].astype('category').cat.codes
testing_data['encoded_Status'] = testing_data['Status'].astype('category').cat.codes
```

```
training_data['encoded_Status'].value_counts()
```

```
3    658
0    521
1    266
2    112
Name: encoded_Status, dtype: int64
```

```
###! pip install transformers
```

```
from transformers import DistilBertTokenizerFast
from transformers import TFDistilBertForSequenceClassification
import tensorflow as tf
import pandas as pd
import json
import gc
```

```
traindata_texts = training_data["Clean Customer Complaint"].to_list() # Features (not-tokenized yet)
traindata_labels = training_data["encoded_Status"].to_list() # Lables
```

```
testdata_texts = testing_data["Clean Customer Complaint"].to_list() # Features (not-tokenized yet)
testdata_labels = testing_data["encoded_Status"].to_list() # Lables
```

```
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
train_encodings = tokenizer.batch_encode_plus(traindata_texts, truncation=True, padding=True)
test_encodings = tokenizer.batch_encode_plus(testdata_texts, truncation=True, padding=True)
```

```
traindata_labels = tf.keras.utils.to_categorical(traindata_labels, dtype ="uint8")
testdata_labels = tf.keras.utils.to_categorical(testdata_labels, dtype ="uint8")
```

```
testdata_texts
```

```
 'unfair billing practices',
 'comcast complaint',
 'comcast bars hbogo streaming purchased device choosing',
 'complaint comcast',
 'comcast inaccurately measuring bandwidth consumption',
 'comcast unlimited internet access recently limited per month',
 'data cap',
 'comcast service billing',
 'increased monthly fee year contract',
 'shitty comcast service',
 'terrible pricing one viable option',
 'pay cable line',
 'access service',
 'comcast xfinity',
 'comcastxfinity communication',
 'comcastnon existent customer service terrible internet connection',
 'internet service loses signal',
 'comcast bill',
 'inexplicable disconnection subsequent mishandling',
 'false sales offers',
 'paying getting',
 'throttling comcast internet services',
 'comcast excessive charge cancelling service',
 'comcast refusal service',
 'deceptive trade practices false advertising bait switch',
 'sold billed short term services',
 'comcast transfer service complaint',
 'unauthorized charges account comcast',
 'comcast charge router unreturned equipment',
 'egregious fees',
 'cramming false internet promotion speed complaint',
 'comcast business internet',
 'comcast',
 'comcast xfinity',
 'comcast sales people reflect said bill',
 'slow internet',
 'comcast cramming',
 'billing continues months terminating service',
 'data usage caps',
 'comcast charges',
 'monopolistic billing practices',
 'comcast outage',
 'fed comcast',
 'comcast charge',
 'comcast xfintity internet data caps',
 'comcst data cap',
 'inconsistent intermittent internet connectivity',
 'comcast internet',
 'comcast cap',
 'false advertisingbait switch',
 'comcast customer service billing issues',
 'sent check payment comcast',
 'comcast unfair pricing',
 'comcast monthly billing returned modem',
 'extremely unsatisfied comcast customer']
```

```
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    traindata_labels
))
test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    testdata_labels
))
```

```
#metrics = [tf.keras.metrics.SparseCategoricalAccuracy('accuracy', dtype=tf.float32)]
metrics = tf.metrics.CategoricalAccuracy()
```

```
loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
```

```
model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=4)

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFDistilBertForSequenceClassification:
- This IS expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model trained on another t
- This IS NOT expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model that you expect
Some weights or buffers of the TF 2.0 model TFDistilBertForSequenceClassification were not initialized from the PyTorch mo
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
model.fit(train_dataset.shuffle(800).batch(24), epochs=30, batch_size=24,
          validation_data=test_dataset.shuffle(800).batch(24))
```

```
Epoch 3/30
65/65 [==============================] - 7s 109ms/step - loss: 1.1151 - accuracy: 0.4611 - val_loss: 1.1690 - val_accura
Epoch 4/30
65/65 [==============================] - 6s 90ms/step - loss: 1.0133 - accuracy: 0.5427 - val_loss: 1.1847 - val_accurac
Epoch 5/30
65/65 [==============================] - 6s 95ms/step - loss: 0.8730 - accuracy: 0.6191 - val_loss: 1.3289 - val_accurac
Epoch 6/30
65/65 [==============================] - 5s 80ms/step - loss: 0.7103 - accuracy: 0.7103 - val_loss: 1.5809 - val_accurac
Epoch 7/30
65/65 [==============================] - 6s 91ms/step - loss: 0.6274 - accuracy: 0.7360 - val_loss: 1.6180 - val_accurac
Epoch 8/30
65/65 [==============================] - 8s 127ms/step - loss: 0.5601 - accuracy: 0.7662 - val_loss: 1.6508 - val_accura
Epoch 9/30
65/65 [==============================] - 7s 102ms/step - loss: 0.4852 - accuracy: 0.8125 - val_loss: 1.8595 - val_accura
Epoch 10/30
65/65 [==============================] - 6s 84ms/step - loss: 0.4421 - accuracy: 0.8157 - val_loss: 2.0077 - val_accurac
Epoch 11/30
65/65 [==============================] - 6s 96ms/step - loss: 0.4161 - accuracy: 0.8170 - val_loss: 2.0964 - val_accurac
Epoch 12/30
65/65 [==============================] - 6s 97ms/step - loss: 0.3916 - accuracy: 0.8343 - val_loss: 2.1156 - val_accurac
Epoch 13/30
65/65 [==============================] - 6s 91ms/step - loss: 0.3709 - accuracy: 0.8343 - val_loss: 2.2491 - val_accurac
Epoch 14/30
65/65 [==============================] - 5s 85ms/step - loss: 0.3591 - accuracy: 0.8330 - val_loss: 2.2467 - val_accurac
Epoch 15/30
65/65 [==============================] - 6s 89ms/step - loss: 0.3335 - accuracy: 0.8478 - val_loss: 2.3809 - val_accurac
Epoch 16/30
65/65 [==============================] - 8s 123ms/step - loss: 0.3277 - accuracy: 0.8497 - val_loss: 2.5650 - val_accura
Epoch 17/30
65/65 [==============================] - 5s 82ms/step - loss: 0.3328 - accuracy: 0.8388 - val_loss: 2.3242 - val_accurac
Epoch 18/30
65/65 [==============================] - 6s 95ms/step - loss: 0.3448 - accuracy: 0.8465 - val_loss: 2.6136 - val_accurac
Epoch 19/30
65/65 [==============================] - 5s 84ms/step - loss: 0.3193 - accuracy: 0.8516 - val_loss: 2.6151 - val_accurac
Epoch 20/30
65/65 [==============================] - 6s 95ms/step - loss: 0.3087 - accuracy: 0.8536 - val_loss: 2.7352 - val_accurac
Epoch 21/30
65/65 [==============================] - 5s 81ms/step - loss: 0.3128 - accuracy: 0.8478 - val_loss: 2.7212 - val_accurac
Epoch 22/30
65/65 [==============================] - 5s 84ms/step - loss: 0.3034 - accuracy: 0.8600 - val_loss: 2.7213 - val_accurac
Epoch 23/30
65/65 [==============================] - 5s 84ms/step - loss: 0.3107 - accuracy: 0.8504 - val_loss: 2.6868 - val_accurac
Epoch 24/30
65/65 [==============================] - 6s 88ms/step - loss: 0.2979 - accuracy: 0.8516 - val_loss: 2.8850 - val_accurac
Epoch 25/30
65/65 [==============================] - 6s 86ms/step - loss: 0.3185 - accuracy: 0.8548 - val_loss: 2.7074 - val_accurac
Epoch 26/30
65/65 [==============================] - 5s 81ms/step - loss: 0.3073 - accuracy: 0.8459 - val_loss: 2.7514 - val_accurac
Epoch 27/30
65/65 [==============================] - 6s 95ms/step - loss: 0.2944 - accuracy: 0.8542 - val_loss: 2.8170 - val_accurac
```

```
Epoch 30/30
65/65 [==============================] - 5s 81ms/step - loss: 0.2890 - accuracy: 0.8529 - val_loss: 2.6823 - val_accurac
<keras.callbacks.History at 0x7aad58307cd0>
```

```
 # the instantiated 🤗 Transformers model - Distilled Bert

from transformers import TFDistilBertForSequenceClassification, TFTrainer, TFTrainingArguments

model.evaluate(test_dataset.shuffle(20).batch(12),
               return_dict=True,
               batch_size=12)
```

```
56/56 [==============================] - 1s 19ms/step - loss: 2.6823 - accuracy: 0.4273
{'loss': 2.6823196411132812, 'accuracy': 0.4272863566875458}
```

```
y_pred = model.predict(test_dataset)
```

```
667/667 [==============================] - 5s 6ms/step
```

```
predictions = tf.nn.softmax(y_pred.logits)
# Extracting the indices with the highest probabilities
predictions = tf.argmax(predictions, axis=1).numpy()
```

```
predictions = pd.DataFrame(predictions)
```

```
predictions.value_counts()
```

```
3    347
0    211
1     94
2     15
dtype: int64
```

```
predictions.rename(columns = {0:"Predict"}, inplace=True)
```

```
train2  = pd.read_csv("/content/train.csv")
```

```
test2  = pd.read_csv("/content/test.csv")
```

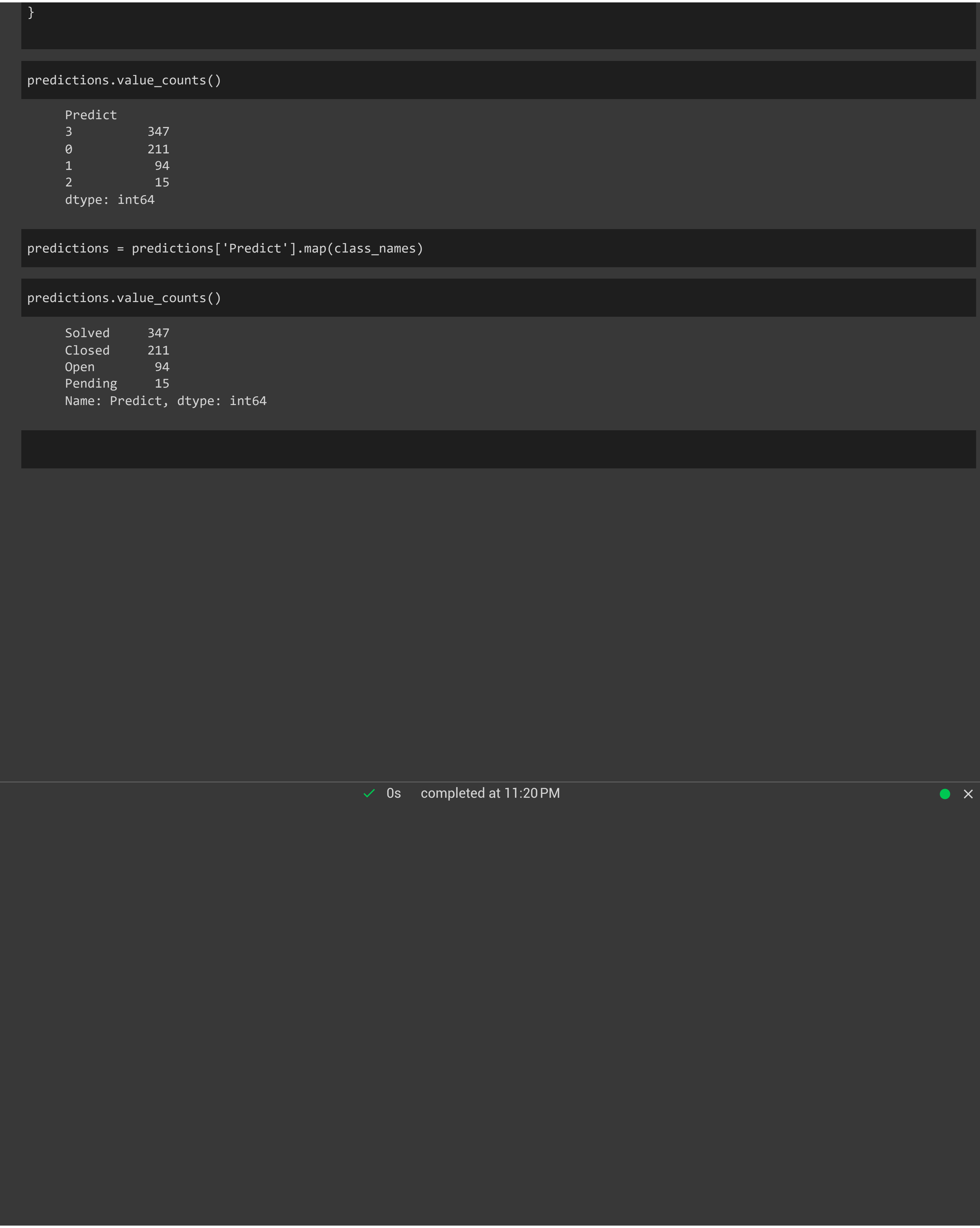
```
test2["Status"].value_counts()
```

```
Solved     315
Closed     213
Open        97
Pending     42
Name: Status, dtype: int64
```

```
test2["Status"].value_counts()
```

```
Solved     315
Closed     213
Open        97
Pending     42
Name: Status, dtype: int64
```

```
class_names = {
    3 : "Solved",
    0 : "Closed",
    1 : "Open",
    2 : "Pending"
```



```
}
```

```
predictions.value_counts()
```

```
Predict
3          347
0          211
1           94
2           15
dtype: int64
```

```
predictions = predictions['Predict'].map(class_names)
```

```
predictions.value_counts()
```

```
Solved     347
Closed     211
Open        94
Pending     15
Name: Predict, dtype: int64
```