# Prediction and Analysis of Visa Application Acceptance using Deep Learning

*Pranav Bhandari, Dushyant Singh Udawat, Krzysztof Dutka, Leo Franco Soto*

Team Name: Bond (Group Number 2)

December 8th, 2022

# Problem Statement

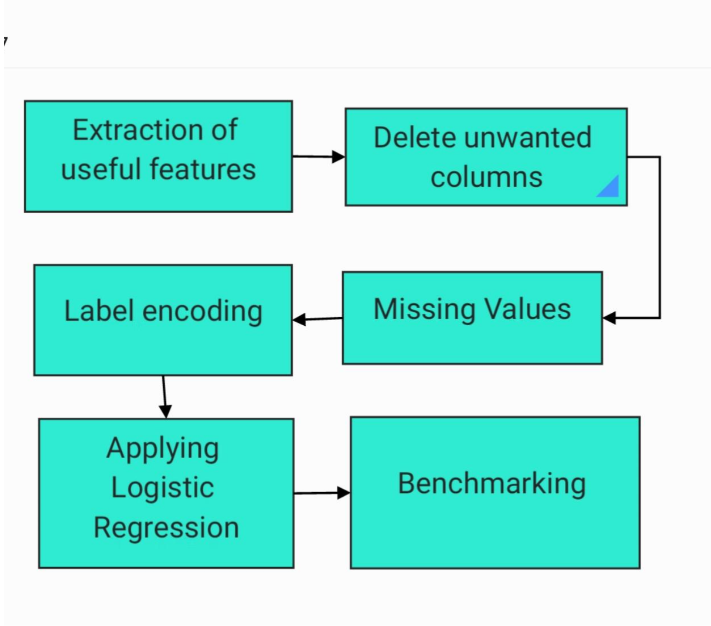


1. Predict the result of an application (Approved / Denied / Withdrawn)
2. Find out if the dataset is self-exciting?
    a. Self-exciting describe random sequences of events where the occurrence of an event increases the likelihood that subsequent events occur nearby in time and space
3. Make various models to predict the percentage of applications getting Approved / Denied / Withdrawn
    a. Logistic Regression
    b. **C**onvolutional **N**eural **N**etwork (CNN)
    c. **L**ong **S**hort-**T**erm **M**emory (LSTM)

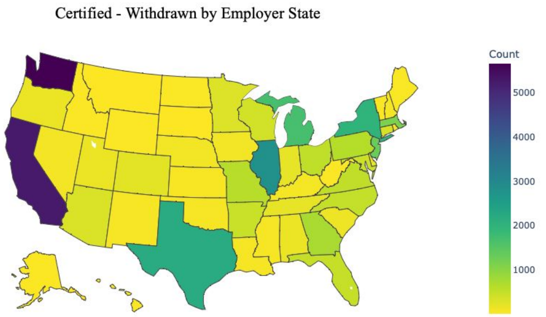
**Image Source:** https://www.istockphoto.com/photos/h1b-visa

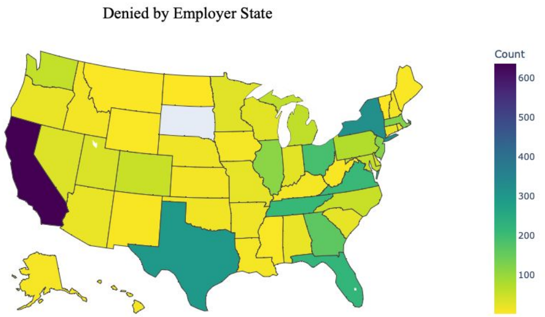
# Data preprocessing

| Inputs |
|---|
| Employer City |
| Employer State |
| Worksite City |
| Worksite State |
| No. of Employees in Worksite |
| Application Date |
| **Outputs** |
| Status of Application |

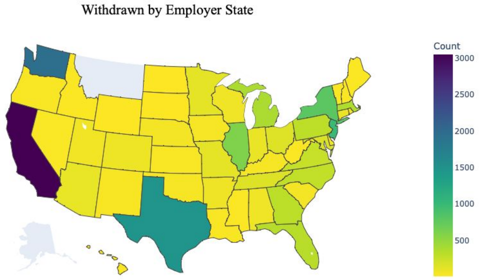


# Data Visualization Maps by Employer State

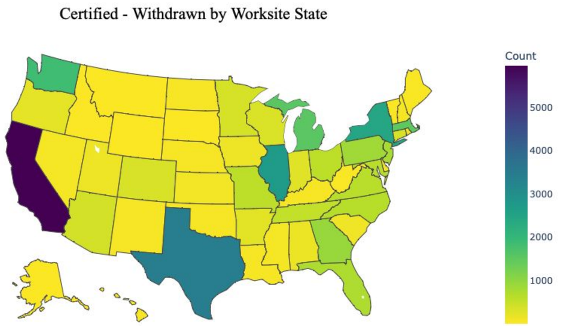


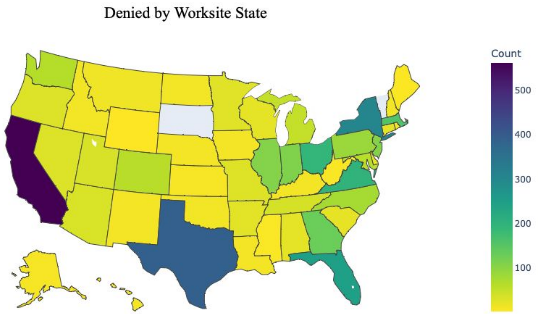



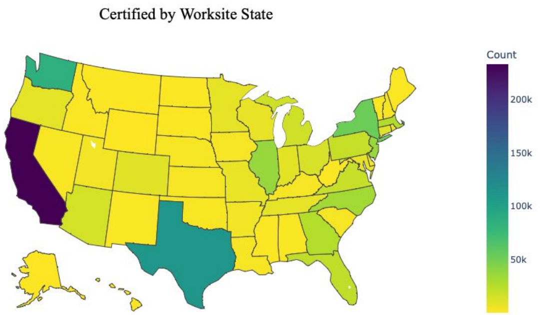


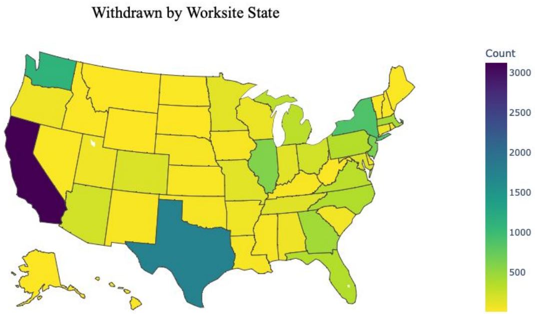


# Data Visualization Maps by Worksite

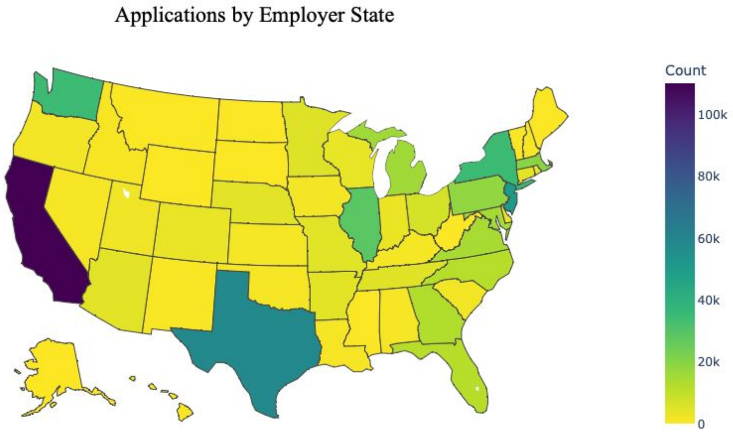


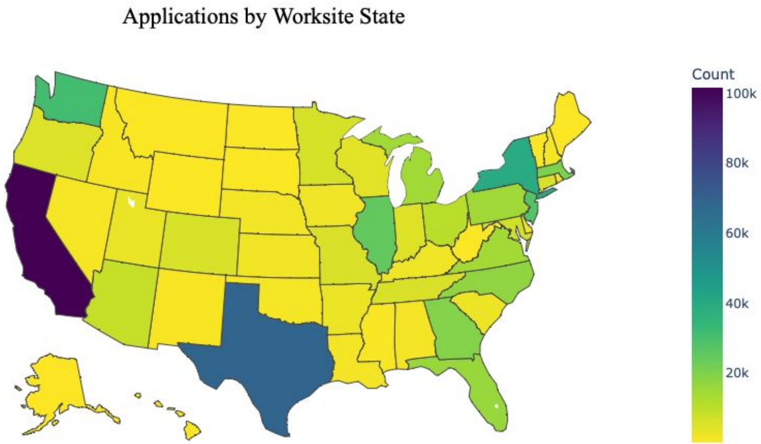



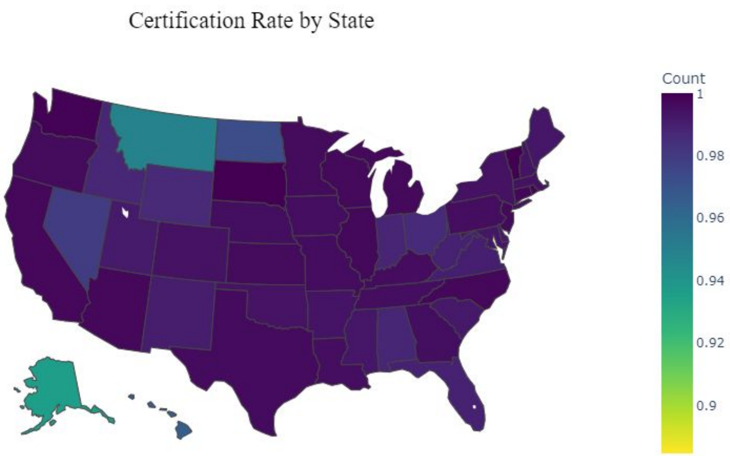



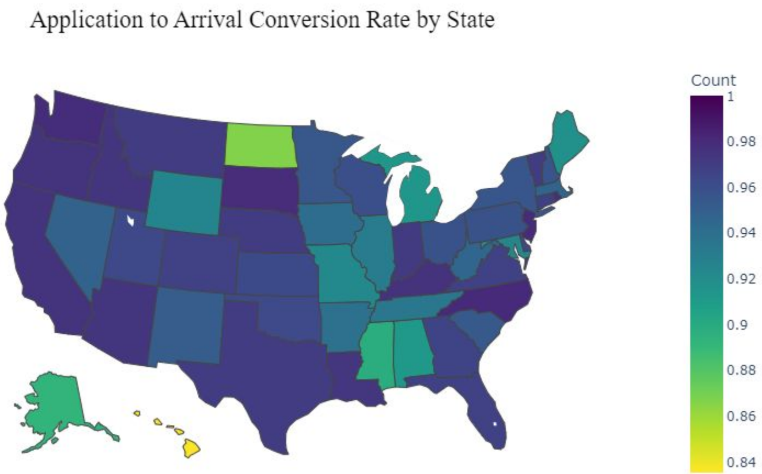
# Data Visualization by Number of Applications



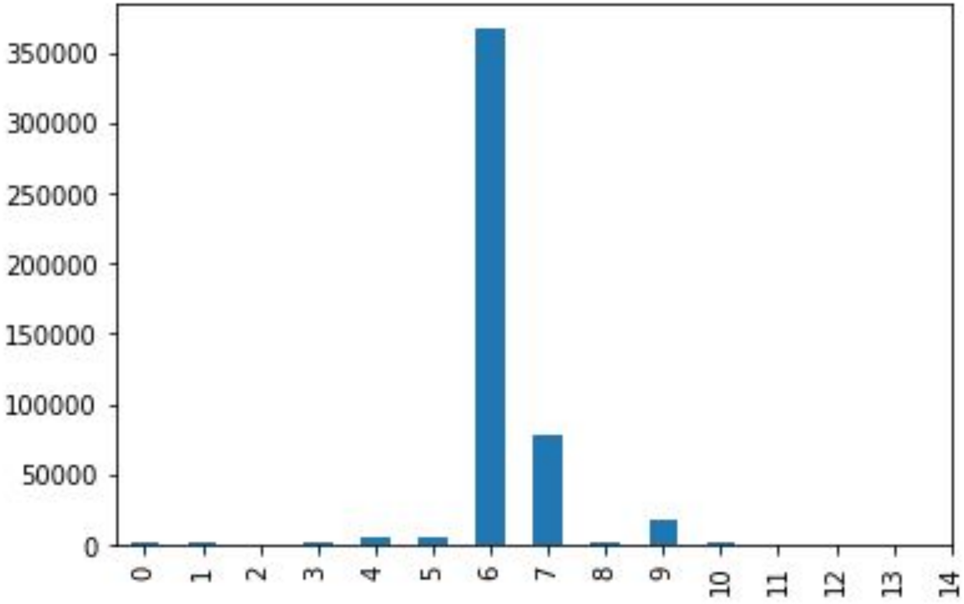



# Data Visualization by Number of Applications



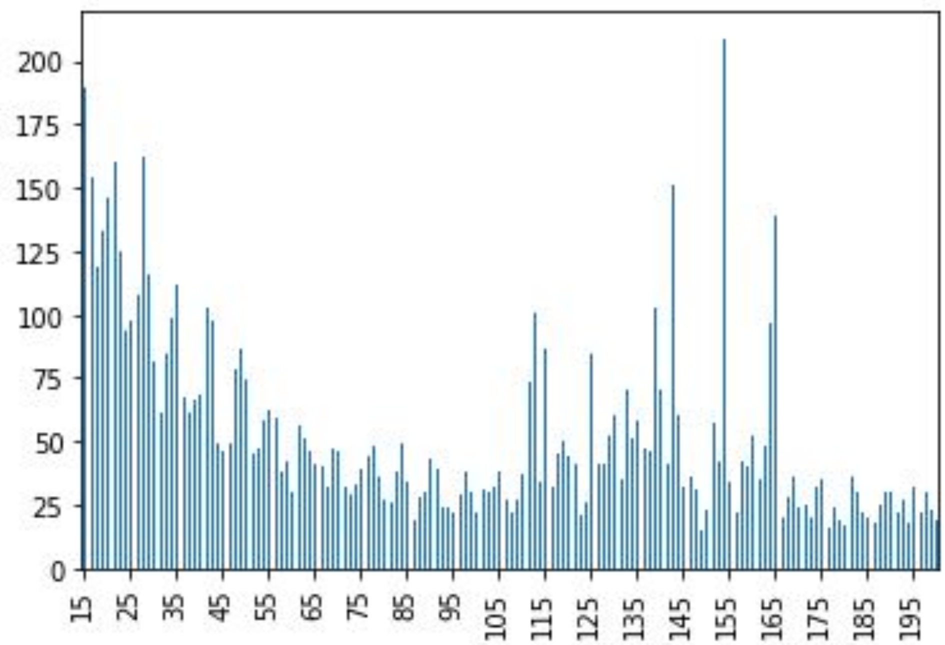



# Number of Results per day within 2 weeks




- Day 6 is the greatest in terms of replies
- With a total of 350000+ all landing on day 6

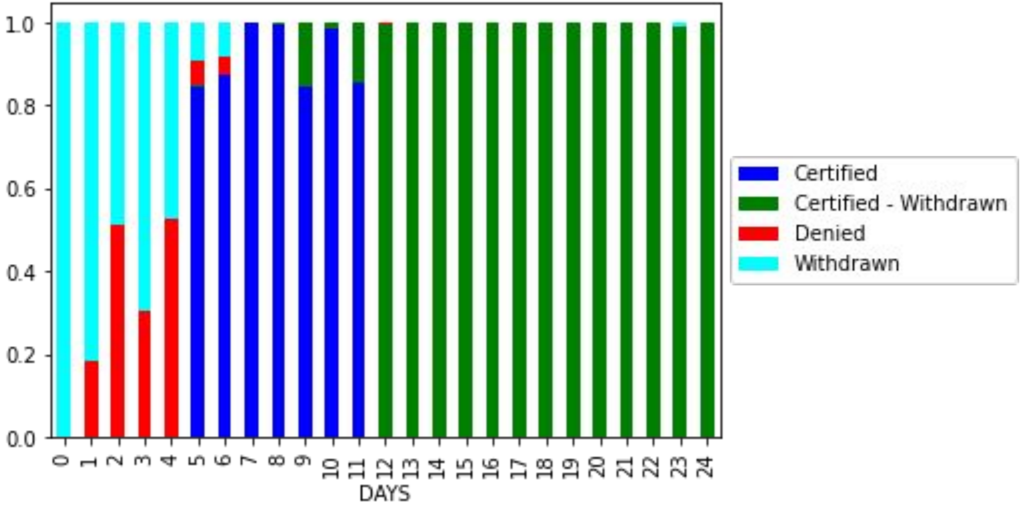
# Number of Results per day 2+ weeks




- Anything past the first two weeks usually gets very little replies.
- This is possibly due to the number of withdrawn and denied applications in the first two weeks

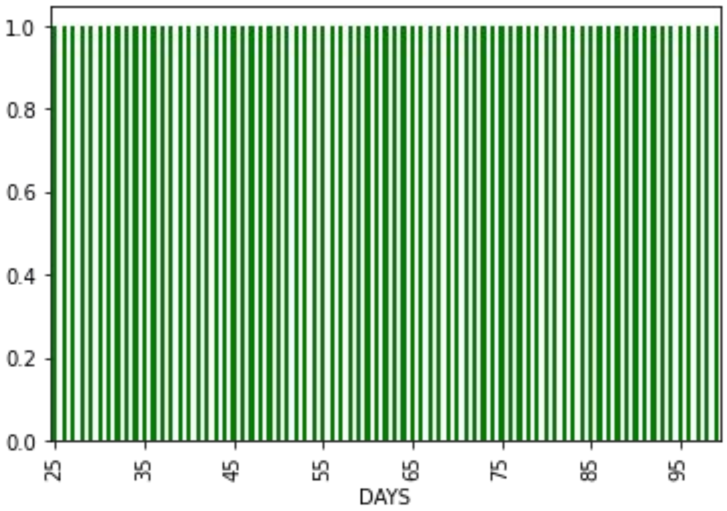
Slide and Code by: Leo Franco Soto (francst2)

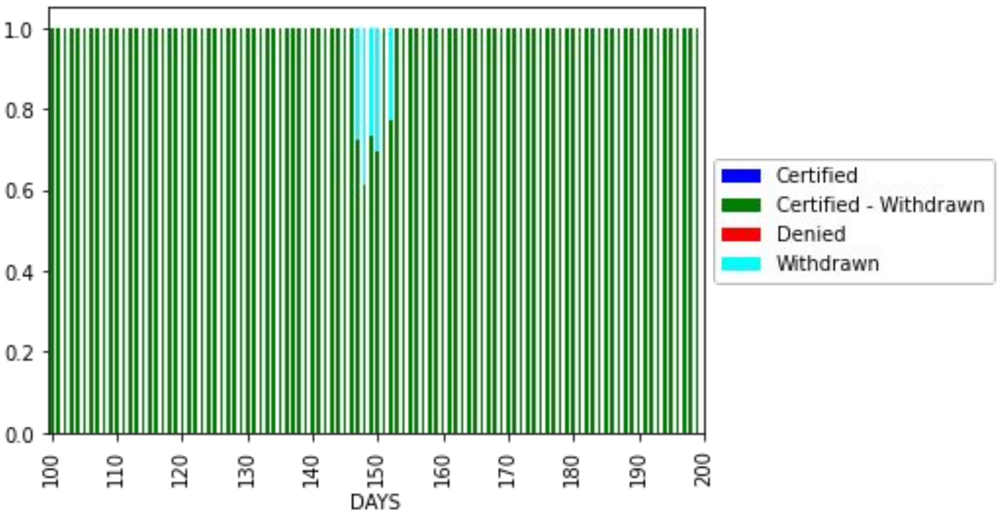
# Breakdown of results per day within 4 weeks




- Most applicants get accepted with two weeks
- Past two weeks, mostly withdrawals

# Breakdown of results per day past 4 weeks



# Benchmarking using Logistic Regression

```
PERFORMANCE_OF_LOGISTIC_REGRESSION_TRAINING =  0.9558854230093422
PERFORMANCE_OF_LOGISTIC_REGRESSION_TESTING =   0.9563949505822488
```

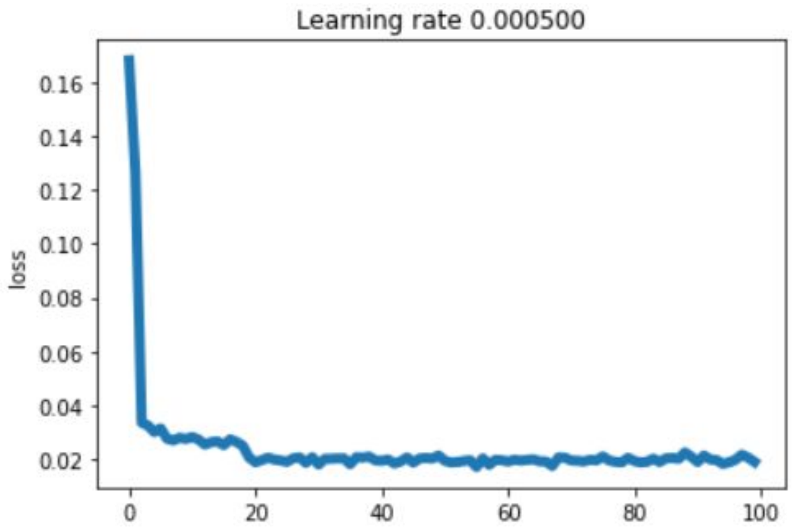
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| C | 0.96 | 1.00 | 0.98 | 47503 |
| CW | 1.00 | 0.57 | 0.72 | 2402 |
| W | 0.00 | 0.00 | 0.00 | 233 |
| D | 0.00 | 0.00 | 0.00 | 957 |
| accuracy | | | 0.96 | 51095 |
| macro avg | 0.49 | 0.39 | 0.43 | 51095 |
| weighted avg | 0.94 | 0.96 | 0.94 | 51095 |

Good at Classifying Certified and Certified Withdrawn Cases.

In our case, They comprise of 99.5% of the data. Performs horribly bad on W, and D cases.

Accuracy Overall - 95.6 %. (Pretty good)
Avg F-score - 0.43 (Not good)
Avg weighted Fscore - 0.94

# CNN




1. No. of Epochs - 100
2. Architecture -
   a. Layersize - 8(input), 16, 16, 8, 8, 8, 4 (output)
   b. ReLU on all layers and Softmax on the last layer.
3. Performance doesn’t become better after 20th epoch.

# Performance of CNN Model.

```
Accuracy of the model in MSE:  0.2030531363147079
PERFORMANCE_OF_CNN_REGRESSION_TRAINING =   0.9597627911831735
PERFORMANCE_OF_CNN_REGRESSION_TESTING =   0.9601722282023681
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| C | 0.96 | 1.00 | 0.98 | 47503 |
| CW | 0.99 | 0.65 | 0.79 | 2402 |
| W | 0.00 | 0.00 | 0.00 | 233 |
| D | 0.00 | 0.00 | 0.00 | 957 |
| accuracy | | | 0.96 | 51095 |
| macro avg | 0.49 | 0.41 | 0.44 | 51095 |
| weighted avg | 0.94 | 0.96 | 0.95 | 51095 |

Slightly Better Performance that Logistic Regression.

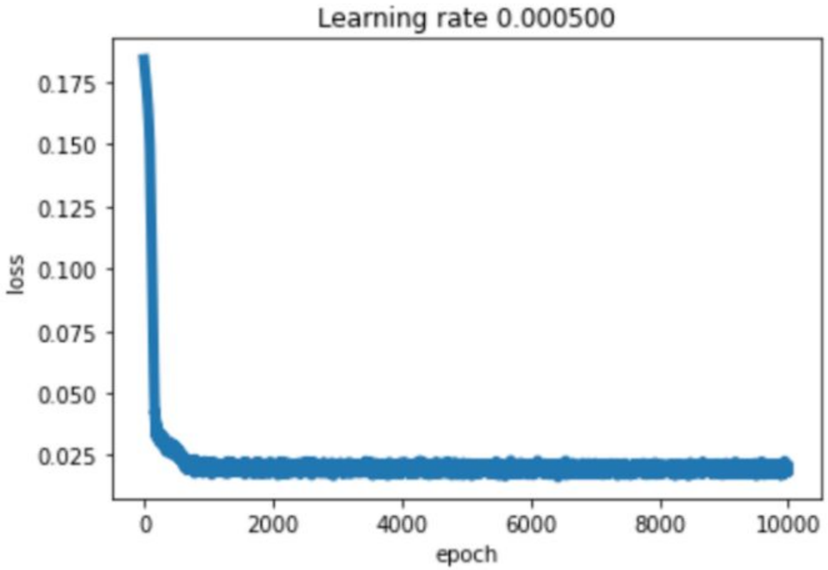
But since the model isn't interpretable, we shouldn't choose it as best model.

The data is not complex enough to require neural net to predict status.

**Accuracy Overall - 96.0 %. (Pretty good)** **Avg F-score - 0.44 (Not good)** **Avg weighted F-score - 0.95**

(All slightly better than logistic regression)

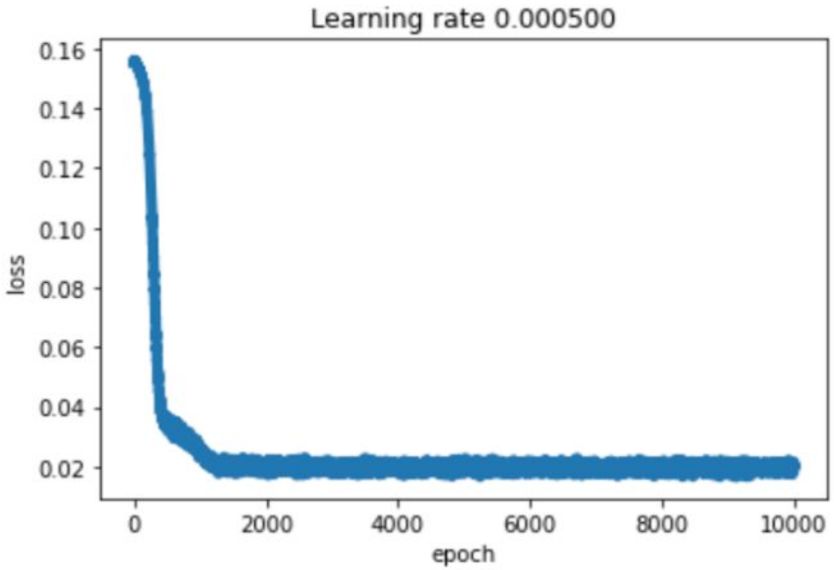
# Running CNN with Different Optimizers (AdamW)




```
Accuracy of the model in MSE:  0.20039142773265486
PERFORMANCE_OF_LINEAR_REGRESSION_TRAINING =  0.9598476018162523
PERFORMANCE_OF_LINEAR_REGRESSION_TESTING =   0.959643800763284
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| C | 0.96 | 1.00 | 0.98 | 47447 |
| CW | 0.99 | 0.65 | 0.78 | 2464 |
| W | 0.00 | 0.00 | 0.00 | 255 |
| D | 0.00 | 0.00 | 0.00 | 929 |
| accuracy | | | 0.96 | 51095 |
| macro avg | 0.49 | 0.41 | 0.44 | 51095 |
| weighted avg | 0.94 | 0.96 | 0.95 | 51095 |

# Running CNN with Different Optimizers (RAdam)



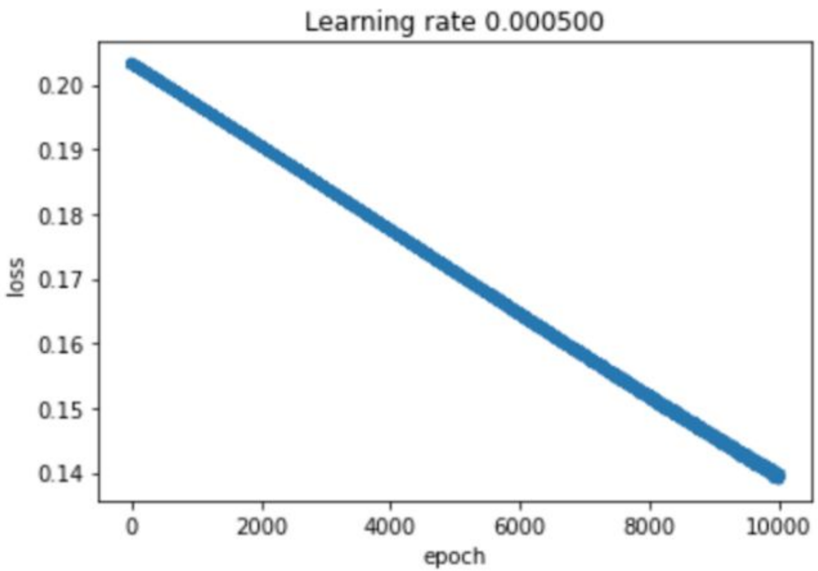

```
Accuracy of the model in MSE:  0.20046971327918583
PERFORMANCE_OF_LINEAR_REGRESSION_TRAINING =  0.9597497433934692
PERFORMANCE_OF_LINEAR_REGRESSION_TESTING =   0.9595655152167532

              precision    recall  f1-score   support

           C       0.96      1.00      0.98     47447
          CW       1.00      0.64      0.78      2464
           W       0.00      0.00      0.00       255
           D       0.00      0.00      0.00       929

    accuracy                           0.96     51095
   macro avg       0.49      0.41      0.44     51095
weighted avg       0.94      0.96      0.95     51095
```

# Running CNN with Different Optimizers (SGD)

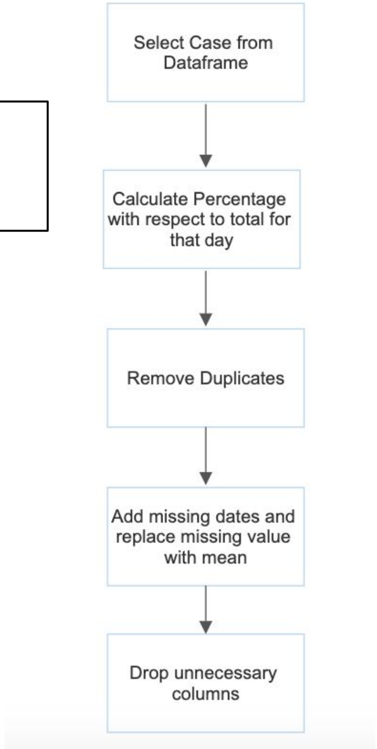



```
Accuracy of the model in MSE:  0.23182307466484
PERFORMANCE_OF_LINEAR_REGRESSION_TRAINING =  0.9287112263182616
PERFORMANCE_OF_LINEAR_REGRESSION_TESTING =   0.9286035815637538
```

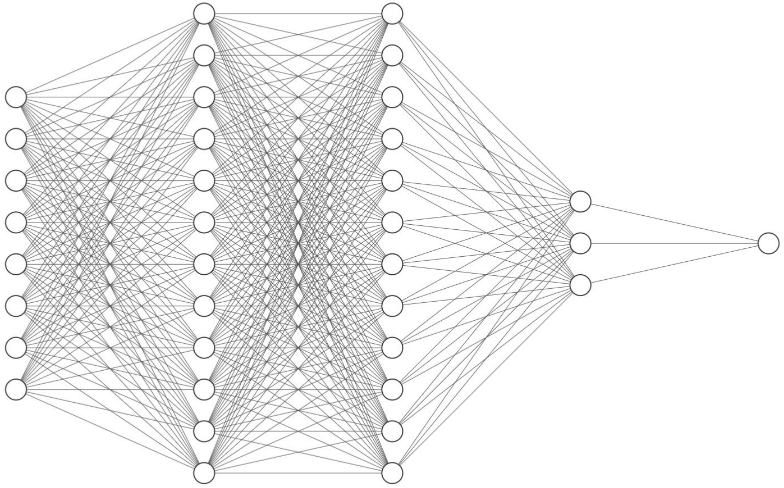
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| C | 0.93 | 1.00 | 0.96 | 47447 |
| CW | 0.00 | 0.00 | 0.00 | 2464 |
| W | 0.00 | 0.00 | 0.00 | 255 |
| D | 0.00 | 0.00 | 0.00 | 929 |
| accuracy | | | 0.93 | 51095 |
| macro avg | 0.23 | 0.25 | 0.24 | 51095 |
| weighted avg | 0.86 | 0.93 | 0.89 | 51095 |

# LSTM Model

Data Preprocessing




LSTM Architecture

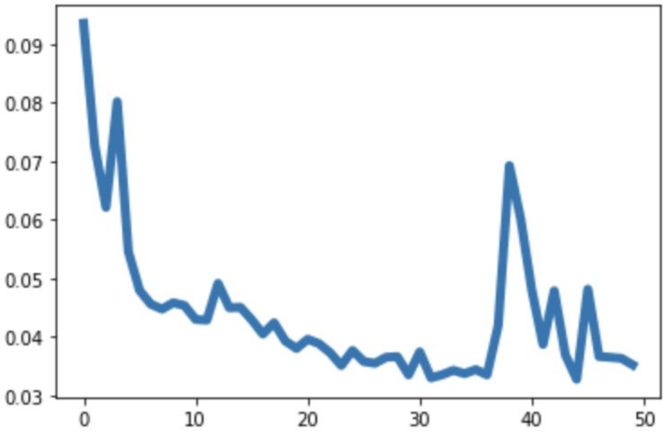


Input Layer
Hidden Layer 1: 100 nodes
Hidden Layer 2: 100 nodes
Dense Layer 1: 25 nodes
Dense Layer 2: 1 node

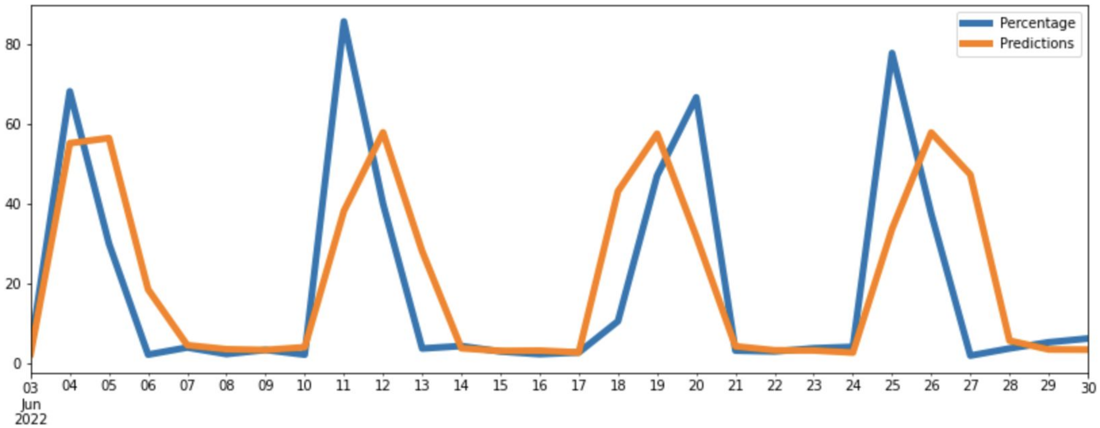
Epochs: 50, Optimizer: Adam, Loss: MSE

# LSTM Result - Certified Withdrawn

Loss vs Epoch



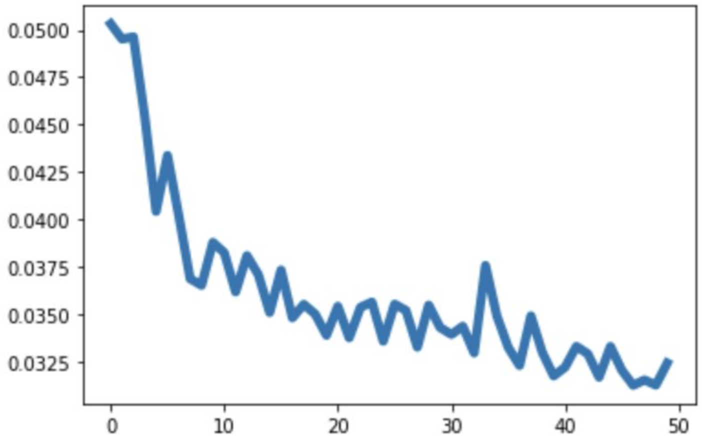

Performance on Test Set



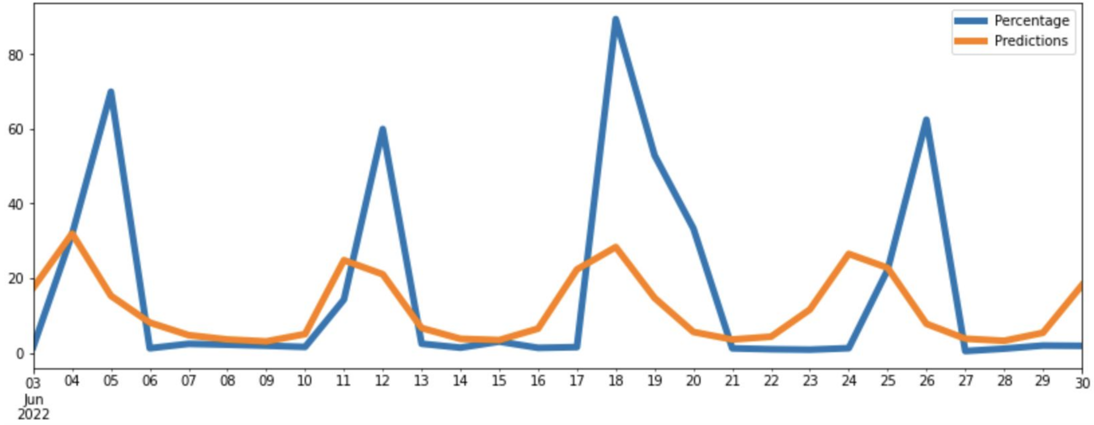

RSME: 19.95

# LSTM Result - Withdrawn

Loss vs Epoch



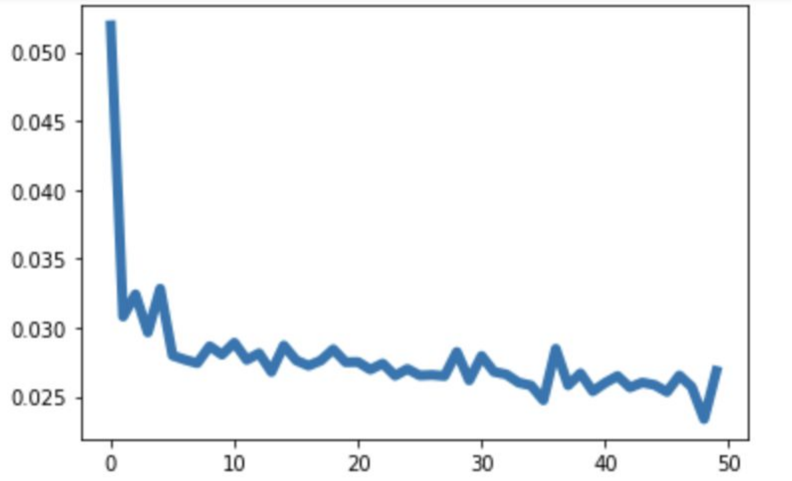

Performance on Test Set



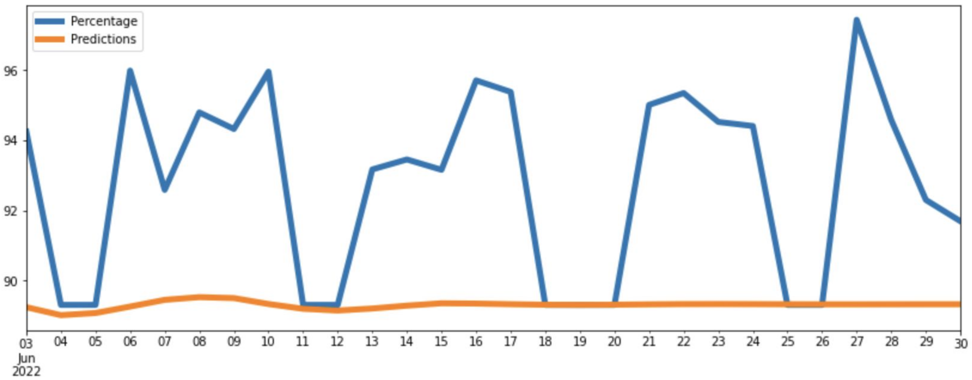

RSME: 23.49

# LSTM Result - Certified

Loss vs Epoch



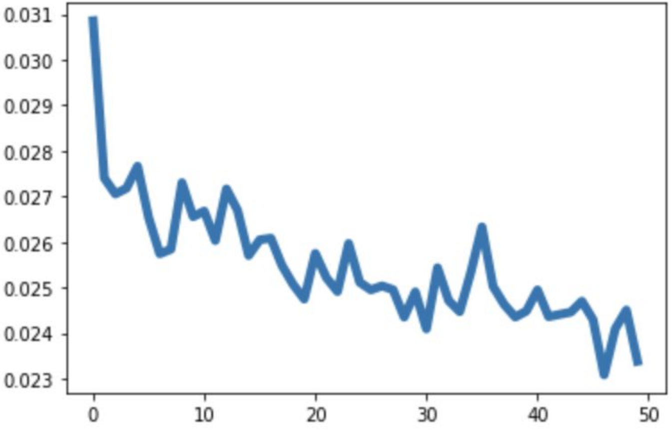

Performance on Test Set



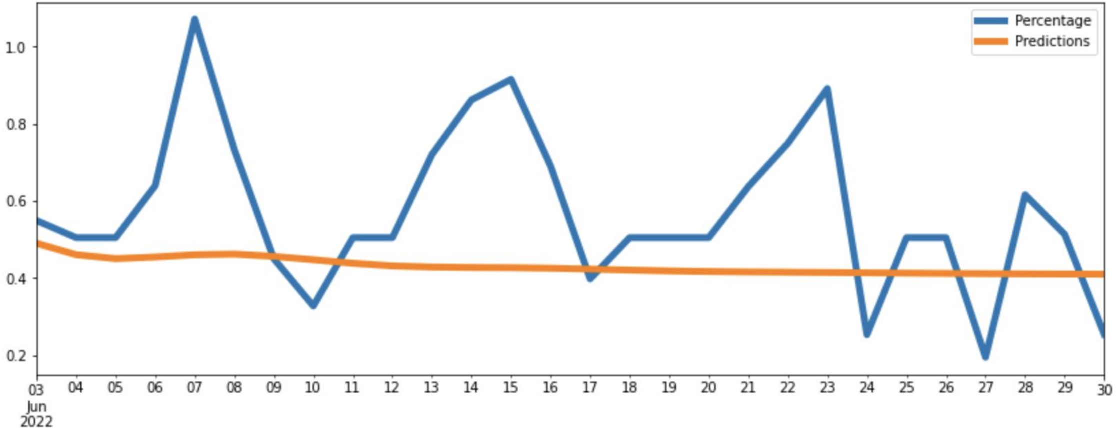

RSME: 4.35

# LSTM Result - Denied

Loss vs Epoch




Performance on Test Set




RSME: 0.24

# Code Snippets

```
approved_pred = approved_predictor(False).to(device)

batch_size, lr = 10000, 5e-4

print(approved_pred)

approved_pred, losses = train(approved_pred, x_tr_tensor, y_tr_tensor_oh, learningRate=lr)

plt.plot(losses)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.title("Learning rate %f"%(lr))
plt.show()


y_p_te = np.around( np.array(torch.argmax(approved_pred(x_te_tensor.float()) , dim=1).detach().cpu()))
y_p_tr = np.around( np.array(torch.argmax(approved_pred(x_tr_tensor.float()), dim=1).detach().cpu()))

conf_mat_tr = confusion_matrix(y_tr, y_p_tr)
conf_mat_te = confusion_matrix(y_te, y_p_te)
print("Training Confusion Matrix")
print(confusion_matrix(y_tr, y_p_tr))
print("\nTesting Confusion Matrix")
print(confusion_matrix(y_te, y_p_te))

print("\nAccuracy of the model in MSE: " , mean_squared_error(y_te, y_p_te) )
#print('r2_score_test :' ,r2_score(y_te, y_p_te))

test_models_regression['deep_le
```

```
# Finding the best strategies for the data to altered and preprocessed for Lea

strategy = {}

for i in range(len(train_data.columns)):
    print(train_data.columns[i] , train_data.iloc[:, i].nunique(dropna = True)
    if (train_data.iloc[:, i].nunique(dropna = True) != train_data.iloc[:, i].
        strategy[i] = 'most_frequent'
    if (train_data.iloc[:, i].nunique(dropna = True) != train_data.iloc[:, i].
        strategy[i] = 'median'
print(strategy)
```

```
class approved_predictor(nn.Module):
    def __init__(self, d_date=True):
        super(approved_predictor, self).__init__

        self.d_date = d_date
        inputSize = 11 if d_date else 8

        self.model = nn.Sequential(
            nn.Linear(inputSize, 16),
            nn.ReLU(),
            nn.Linear(16,16),
            nn.ReLU(),
            nn.Linear(16,8),
            nn.ReLU(),
            nn.Linear(8,8),
            nn.ReLU(),
            nn.Linear(8,8),
            nn.ReLU(),
            nn.Linear(8,8),
            nn.ReLU(),
            nn.Linear(8,4),
            nn.Softmax(dim=1)
        )

ef forward(self, x):
        if self.d_date == False:
            x = x[:, :8]
        return self.model(x)
```

```
for itr in range(MAX_iter):

    # Clear gradient buffers
    optimizer.zero_grad()

    # Create the iterations batch and labels
    rand_vec = torch.randint(0, points.size()[0
    batch = points.float()[rand_vec,:]
    batch_labels = labels.float()[rand_vec]

    # get output from the model, given the inpu
    outputs = model(batch)

    # get loss for the predicted output
    lossvalue = loss(outputs, batch_labels)
    if itr%100 == 0:
      losses.append(lossvalue.item())

    # get gradients w.r.t to parameters
    lossvalue.backward()

    optimizer.step()

    if itr % printEvery == 0:
        print("Epoch {}: loss={:.5f}".format(it

return model, losses
```

# Code Snippets

```python
def plotmap1(column,value,title):
  demo = data_viz.loc[data_viz['CASE_STATUS'] == value]
  experiment = demo.groupby(column).sum().rename(columns = {'WORKSITE_WORKERS' : 'Count'})
  fig = px.choropleth(experiment,
                      locations=experiment.index,
                      locationmode="USA-states",
                      scope="usa",
                      color='Count',
                      color_continuous_scale="Viridis_r"
                      )

  fig.update_layout(
        title_text = title,
        title_font_family="Times New Roman",
        title_font_size = 22,
        title_font_color="black",
        title_x=0.45,
        showlegend=False
          )
  fig.show()
```

```python
# Dropping rows with missing values
data_x.dropna(inplace = True)
```

```python
# Converting Decision date and received date to date time
data_x['DECISION_DATE'] = pd.to_datetime(data_x['DECISION_DATE'])
data_x['RECEIVED_DATE'] = pd.to_datetime(data_x['RECEIVED_DATE'])

# Sorting by decision date
data_x.sort_values(by = ['DECISION_DATE'],inplace = True)

#Dropping columns not needed in LSTM
data_x.drop(['RECEIVED_DATE','EMPLOYER_CITY','EMPLOYER_STATE','WORKSITE_CITY','WORKSITE_STATE','WORKSITE_WORKERS'],axis = 1,inplace = Tru
```

```python
# Adding total visas processed per data to dataframe
trial = {}
sum = 0
for i in data_x['DECISION_DATE']:
  if i not in trial:
    trial[i] = 1
  else:
    trial[i] += 1

for k,v in trial.items():
  data_x.loc[data_x['DECISION_DATE'] == k, 'TOTAL_COUNT'] = int(v)
```

```python
#Restting the index to get correct dates
data_x.reset_index(drop=True, inplace=True)
```

```python
def data_prep_lstm(case):
  # Creating a copy of the dataset
  data_x_copy = data_x.copy()

  # Filtering to particular case status
  certified = data_x_copy[data_x_copy["CASE_STATUS"] == case]

  #Finding percentage of case status with respect to total visas given that day
  certified1 = {}
  sum = 0
  for i in certified['DECISION_DATE']:
    if i not in certified1:
      certified1[i] = 1
    else:
      certified1[i] += 1
  for k,v in certified1.items():
    certified.loc[certified['DECISION_DATE'] == k, 'COUNT'] = int(v)

  #Dropping duplicate decision dates
  certified = certified.drop_duplicates(subset=['DECISION_DATE'], keep='last')

  #Finding percentage
  certified['Percentage'] = (certified['COUNT']/certified['TOTAL_COUNT'])*100

  #Dropping unrequired columns
  certified.drop(['CASE_STATUS','TOTAL_COUNT','COUNT'],axis = 1,inplace = True)

  #Setting decision date as the index
  certified.reset_index(drop=True, inplace=True)
  certified.set_index('DECISION_DATE',inplace = True)
```

```python
  #Dropping unrequired columns
  certified.drop(['CASE_STATUS','TOTAL_COUNT','COUNT'],axis = 1,inplace = True)

  #Setting decision date as the index
  certified.reset_index(drop=True, inplace=True)
  certified.set_index('DECISION_DATE',inplace = True)

  #Adding missing dates
  new_date_range = pd.date_range(start=data_x['DECISION_DATE'][0], end=data_x['DECISION_DATE'].iat[-1],
                                 freq="D")
  certified = certified.reindex(new_date_range, fill_value=np.NaN)

  #Filling missing values for newly added dates by mean
  certified = certified.fillna(certified.mean())

  return certified
```

# Thanks!