Final Case Study

US Permanent Visa Analysis & Prediction

Case Study: Perform analysis to identify different important factors that could impact the US permanent visa application
I will be using US permanent visa dataset and run various graph analysis on the selected features to see how each of them impact the outcome of the US permanent visa application.

Dataset:

- Original dataset has been taken from https://www.kaggle.com/jboysen/us-perm-visas (https://www.kaggle.com/jboysen/us-perm-visas).
- In DSC 540, I have applied several data modification on this dataset by combining few columns and by normalizing the data in few columns.
- For this exercise I am going to use the final csv created out of my previous exercise.

In [1]:
```
# Import necessary packages
import pandas as pd
import yellowbrick
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
from yellowbrick.features import Rank2D
from yellowbrick.style import set_palette
from yellowbrick.features import ParallelCoordinates
import numpy as np
```

Step 1: Load data into a dataframe

In [2]:
```
# Load data into article dataframe
addr1 = "case_study_data/us_perm_visas_final.csv"
raw_df = pd.read_csv(addr1, low_memory=False)
print("Shape of raw input:", raw_df.shape)
```

```
Shape of raw input: (374362, 17)
```

In [3]:
```
# Get an idea about original dataset
print("The dimension of the table is: ", raw_df.shape)
# Display the data
print("\nTop 5 rows", raw_df.head(5))
# Print columns with None data
print("\nRows with missing data by column:\n", raw_df.isna().sum())
```

```
The dimension of the table is:  (374362, 17)

Top 5 rows      case_number case_status class_of_admission country_of_c
itizenship  \
0  A-07323-97014   Certified                J-1              ARMENIA
1  A-07332-99439      Denied                B-2               POLAND
2  A-07333-99643   Certified               H-1B                INDIA
3  A-07339-01930   Certified                B-2          SOUTH KOREA
4  A-07345-03565   Certified                L-1               CANADA

  decision_date                employer_name  employer_num_employees  \
0    2012-02-01              NETSOFT USA INC.                    NaN
1    2011-12-21  PINNACLE ENVIRONEMNTAL CORP                    NaN
2    2011-12-01   SCHNABEL ENGINEERING, INC.                    NaN
3    2011-12-01      EBENEZER MISSION CHURCH                    NaN
4    2012-01-26   ALBANY INTERNATIONAL CORP.                    NaN

               employer_name.1 employer_state  \
0             NETSOFT USA INC.             NY
1  PINNACLE ENVIRONEMNTAL CORP             NY
2   SCHNABEL ENGINEERING, INC.             VA
3      EBENEZER MISSION CHURCH             NY
4   ALBANY INTERNATIONAL CORP.             NY

  foreign_worker_info_birth_country job_info_work_city job_info_work_st
ate  \
0                               NaN           New York
NY
1                               NaN           New York
NY
2                               NaN        Lutherville
MD
3                               NaN           Flushing
NY
4                               NaN             Albany
NY

                          pw_job_title_9089 pw_level_9089  \
0  Computer Software Engineers, Applications      Level II
1                           ASBESTOS HANDLER       Level I
2                             Civil Engineer       Level I
3                                 File Clerk      Level II
4                   Sales & Service Engineer      Level IV

                                pw_soc_title pw_amount_9089  \
0  Computer Software Engineers, Applications        75629.0
1        Hazardous Materials Removal Workers        37024.0
2                            Civil Engineers        47923.0
3                                File Clerks          10.97
4                            Sales Engineers        94890.0

  pw_unit_of_pay_9089
0                  yr
1                  yr
2                  yr
3                  hr
4                  yr
```

```
Rows with missing data by column:
 case_number                             0
case_status                             0
class_of_admission                  22845
country_of_citizenship                 59
decision_date                           0
employer_name                          12
employer_num_employees             135349
employer_name.1                        12
employer_state                         42
foreign_worker_info_birth_country  135300
job_info_work_city                    102
job_info_work_state                   103
pw_job_title_9089                     392
pw_level_9089                       27627
pw_soc_title                         2336
pw_amount_9089                       2216
pw_unit_of_pay_9089                  1572
dtype: int64
```

Step 2: Feature & data selection - Select the features and rows meaningful for the goal

In [4]:
```
# Drop the rows with missing data for columns 'class_of_admission', 'cou
ntry_of_citizenship',
# 'employer_state', 'pw_unit_of_pay_9089'
raw_df.dropna(axis=0, how = 'any', subset=['class_of_admission',
                                           'country_of_citizenship',
                                           'employer_state',
                                           'pw_unit_of_pay_9089'], inpla
ce=True)
# Select few interesting columns
# .copy() would create a new DF
data=raw_df[['case_status',
         'class_of_admission',
         'country_of_citizenship',
         'employer_num_employees',
         'employer_state',
         'pw_level_9089',
         'decision_date']].copy()

# Modify the column names
data.columns = ['case_status', 'entry_visa', 'citizenship',
                'no_of_employees', 'state', 'job_level', 'year']

def getsalary(x):
    """This method will calculate the salary based on rate in column pw_
unit_of_pay_9089"""
    if x[15].__class__ == str:
        salary=float(x[15].replace(',', ''))
    else:
        salary = x[15]

    # Get correct wage_unit value due to bad input data
    wage_unit = x[16]

    if salary > 15000:
        wage_unit='Year'
    elif x[16] == 'Year':
        if salary < 200:
            wage_unit='Hour'
        elif salary < 2500:
            wage_unit='Week'
        elif salary < 10000:
            wage_unit = 'Month'
    elif x[16] == 'Bi-Weekly':
        if salary < 100:
            wage_unit = 'Hour'
    elif x[16] == 'Week':
        if salary < 100:
            wage_unit='Hour'
    elif x[16] == 'Hour':
        if salary > 15000:
            wage_unit='Year'
        elif salary > 2000:
            wage_unit='Month'
        elif salary > 1000:
            wage_unit='Bi-Weekly'
        elif salary > 200:
```

```
            wage_unit='Week'

    # Get salary based on wage unit
    if wage_unit == 'Hour':
        return salary*52*40
    elif wage_unit == 'Week' or x[16] == 'wk':
        return salary*52
    elif wage_unit == 'Bi-Weekly' or x[16] == 'bi':
        return salary*26
    elif wage_unit == 'Month' or x[16] == 'mth':
        return salary*12
    else:
        if salary > 1000000:
            return salary/100
        return salary

# Get yearly salary on all rows
data['salary'] = raw_df.apply(getsalary, axis=1)
```

In [5]:

```
# Select only data from 2014, 2015 & 2016 years for this analysis
data = data[data['year'] > '2013-12-31']

# Reset the index as we eliminated some rows
data = data.reset_index(drop=True)
```

In [6]:
```python
print("Dataset state after step 2 - feature and data selection\n")
# Get an idea about original dataset
print("The dimension of the table is: ", data.shape)
# Display the data
print("\nTop 5 rows", data.head(5))
# Print columns with None data
print("\nRows with missing data by column:\n", data.isna().sum())
```

```
Dataset state after step 2 - feature and data selection

The dimension of the table is:  (279365, 8)

Top 5 rows          case_status entry_visa  citizenship  no_of_employee
s          state  \
0  Certified-Expired       H-1B        INDIA              NaN  MASSACHU
SETTS
1  Certified-Expired       H-1B        INDIA              NaN       ARK
ANSAS
2          Certified       H-1B        INDIA              NaN       NEW
YORK
3  Certified-Expired       H-1B  SOUTH KOREA              NaN     CALIF
ORNIA
4          Certified       H-1B        INDIA              NaN      WISC
ONSIN

   job_level        year    salary
0   Level IV  2014-02-21  116542.4
1    Level I  2014-01-08   42973.0
2  Level III  2014-05-22  101629.0
3   Level II  2014-03-28   60445.0
4   Level IV  2014-05-28   92414.0

Rows with missing data by column:
 case_status           0
entry_visa            0
citizenship           0
no_of_employees   57167
state                 0
job_level         19805
year                  0
salary                0
dtype: int64
```

Step 3: Modify feature values to make it more suitable for analysis

In [7]:
```
# Modify data to get better analysis results

# Derive the year column with substring function of str
data['year'] = data['year'].apply(lambda x: int(x[0:4]))

# Convert job level to a numeric field as needed for different analysis
 algorithms
data = data.replace({'job_level': {'Level I': int(1),
                                   'Level II': int(2),
                                   'Level III': int(3),
                                   'Level IV': int(4)}})
```

In [8]:
```
# Fill missing values for no_of_employees & salary using median of the c
olumn
def fill_na_median(data, inplace=True):
    """This method fills missing rows with median for that column"""
    return data.fillna(data.median(), inplace=inplace)

fill_na_median(data['salary'])
fill_na_median(data['no_of_employees'])

# Fill missing values for job_level using the most common value (Level I
I)
def fill_na_top_value(data, value, inplace=True):
    """This method fills missing rows with median for that column"""
    return data.fillna(value, inplace=inplace)

fill_na_top_value(data['job_level'], 2)
```

In [9]:
```
# Derive the log transformed values for salary & no_of_employees
data['salary_log'] = data['salary'].apply(np.log1p)
data['no_of_employees_log'] = data['no_of_employees'].apply(np.log1p)
print("\nTop 5 rows after log transformation for salary & no_of_employee
s along with the original values\n",
      data[['salary', 'salary_log', 'no_of_employees', 'no_of_employees_
log']].head(5))
```

```
Top 5 rows after log transformation for salary & no_of_employees along
with the original values
      salary  salary_log  no_of_employees  no_of_employees_log
0  116542.4   11.666019           1634.0             7.399398
1   42973.0   10.668351           1634.0             7.399398
2  101629.0   11.529094           1634.0             7.399398
3   60445.0   11.009506           1634.0             7.399398
4   92414.0   11.434045           1634.0             7.399398
```

In [10]:
```
print("Dataset state after step 3 - modifying some features and filling
 missing values\n")
# Get an idea about original dataset
print("The dimension of the table is: ", data.shape)
# Display the data
print("\nTop 5 rows", data.head(5))
# Print columns with None data
print("\nRows with missing data by column:\n", data.isna().sum())
```

```
Dataset state after step 3 - modifying some features and filling missin
g values

The dimension of the table is:  (279365, 10)

Top 5 rows          case_status entry_visa  citizenship  no_of_employee
s          state  \
0  Certified-Expired       H-1B        INDIA           1634.0  MASSACHU
SETTS
1  Certified-Expired       H-1B        INDIA           1634.0       ARK
ANSAS
2          Certified       H-1B        INDIA           1634.0       NEW
YORK
3  Certified-Expired       H-1B  SOUTH KOREA           1634.0     CALIF
ORNIA
4          Certified       H-1B        INDIA           1634.0      WISC
ONSIN

   job_level  year    salary  salary_log  no_of_employees_log
0        4.0  2014  116542.4   11.666019             7.399398
1        1.0  2014   42973.0   10.668351             7.399398
2        3.0  2014  101629.0   11.529094             7.399398
3        2.0  2014   60445.0   11.009506             7.399398
4        4.0  2014   92414.0   11.434045             7.399398

Rows with missing data by column:
 case_status            0
entry_visa             0
citizenship            0
no_of_employees        0
state                  0
job_level              0
year                   0
salary                 0
salary_log             0
no_of_employees_log    0
dtype: int64
```

Step 4: Understand the type of variables in the dataset after all modifications

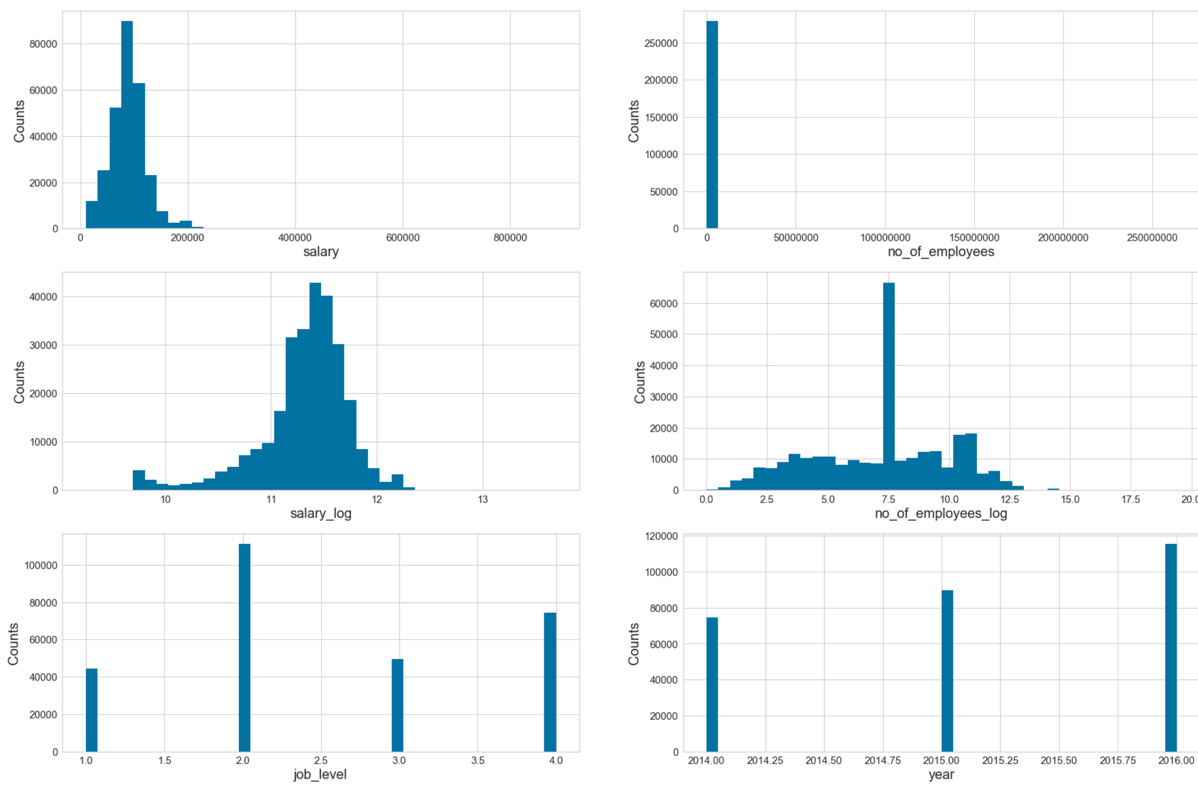
In [11]:
```
#what type of variables are in the table
print("\nDescribe Data")
print(data.describe())
print("\nSummarized Data")
print(data.describe(include=['O']))
```

```
Describe Data
       no_of_employees      job_level           year         salary  \
count     2.793650e+05  279365.000000  279365.000000  279365.000000
mean      1.992288e+04       2.551426    2015.146586   88646.609885
std       5.044350e+05       1.047190       0.811143   31965.935855
min       0.000000e+00       1.000000    2014.000000   10400.000000
25%       1.700000e+02       2.000000    2014.000000   71074.000000
50%       1.634000e+03       2.000000    2015.000000   88254.000000
75%       1.080000e+04       4.000000    2016.000000  106288.000000
max       2.635506e+08       4.000000    2016.000000  885666.000000

          salary_log  no_of_employees_log
count  279365.000000        279365.000000
mean       11.313593             7.199814
std         0.431814             2.754739
min         9.249657             0.000000
25%        11.171491             5.141664
50%        11.387986             7.399398
75%        11.573917             9.287394
max        13.694096            19.389756

Summarized Data
       case_status entry_visa citizenship   state
count       279365     279365      279365  279365
unique           4         54         197     112
top      Certified      H-1B       INDIA      CA
freq        147213     222234      159643   36454
```

Step 5: Plot the histogram for numeric columns to understand the data

In [12]:
```
# Specify the features of interest
num_features = ['salary', 'no_of_employees', 'salary_log', 'no_of_employ
ees_log', 'job_level', 'year']
xaxes = num_features
yaxes = ['Counts', 'Counts', 'Counts', 'Counts', 'Counts', 'Counts']
```

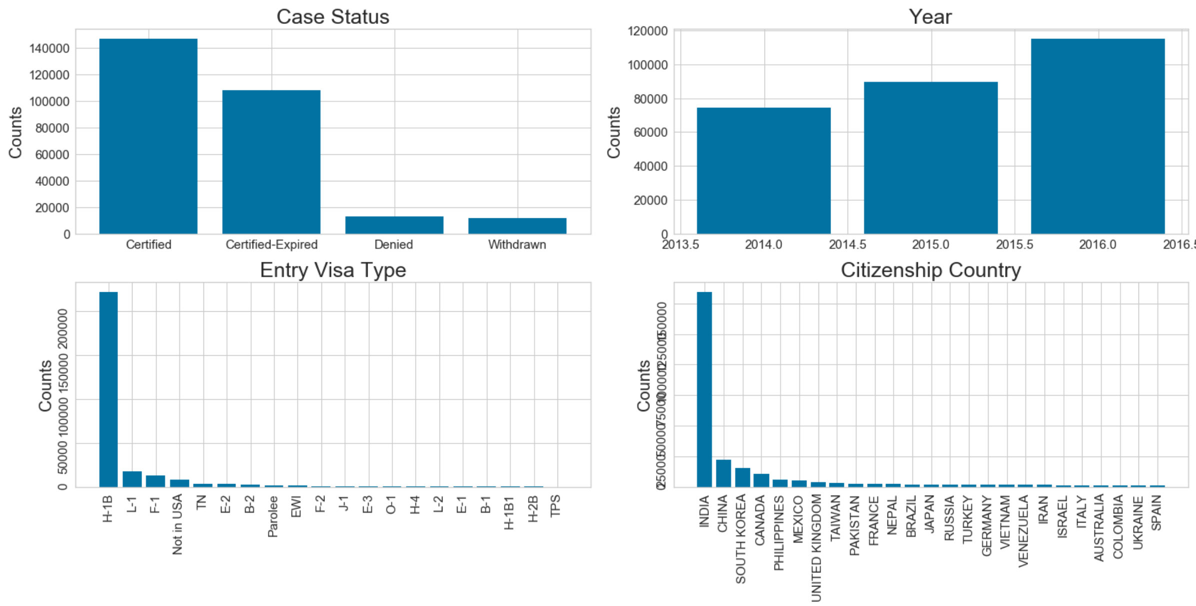
In [13]:
```
#%matplotlib inline
# set up the figure size
plt.rcParams['figure.figsize'] = (30, 20)

# make subplots
fig, axes = plt.subplots(nrows = 3, ncols = 2)

# draw histograms
axes = axes.ravel()
for idx, ax in enumerate(axes):
    ax.hist(data[num_features[idx]].dropna(), bins=40)
    ax.set_xlabel(xaxes[idx], fontsize=20)
    ax.set_ylabel(yaxes[idx], fontsize=20)
    ax.tick_params(axis='both', labelsize=15)
    ax.ticklabel_format(useOffset=False, style='plain')

# Display the plot
plt.show()
```



Step 6: Setup and plot bar charts for different features

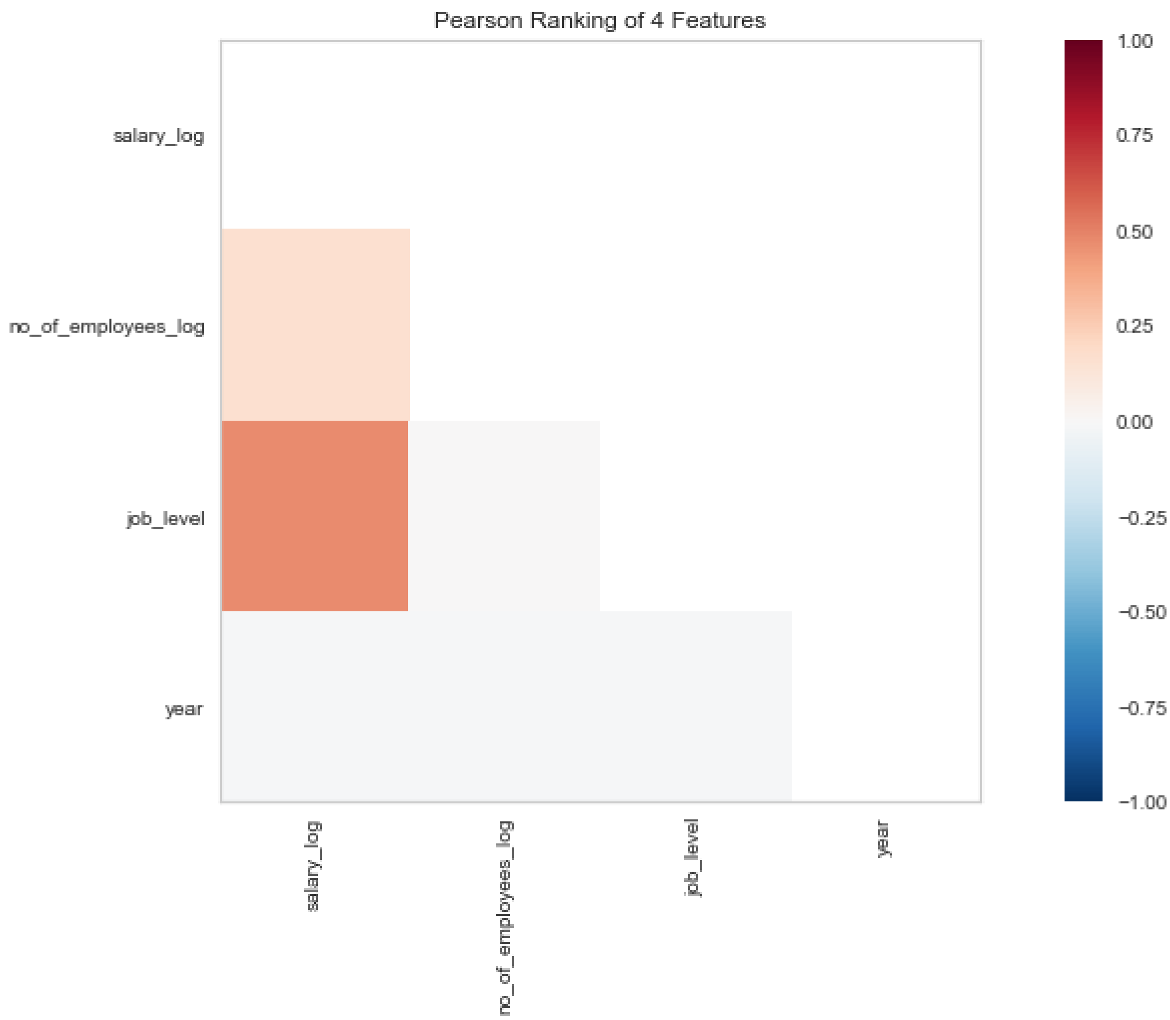
In [14]:
```
#%matplotlib inline
# set up the figure size
plt.rcParams['figure.figsize'] = (20, 10)

# make subplots
fig = plt.figure(constrained_layout=True)
# Create grid of plots
gs = GridSpec(2,2, figure=fig)

# Derive different plot axes objects for bar chart plots
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])
ax3 = fig.add_subplot(gs[-1, 0])
ax4 = fig.add_subplot(gs[-1, 1])

# make the data ready to feed into the visulizer
X_case_status = data.groupby('case_status').size().reset_index(name='Cou
nts')['case_status']
Y_case_status = data.groupby('case_status').size().reset_index(name='Cou
nts')['Counts']
# make the bar plot
ax1.bar(X_case_status, Y_case_status)
ax1.set_title('Case Status', fontsize=25)
ax1.set_ylabel('Counts', fontsize=20)
ax1.tick_params(axis='both', labelsize=15)

# make the data read to feed into the visulizer
X_year = data.groupby('year').size().reset_index(name='Counts')['year']
Y_year = data.groupby('year').size().reset_index(name='Counts')['Counts'
]
# make the bar plot
ax2.bar(X_year, Y_year)
ax2.set_title('Year', fontsize=25)
ax2.set_ylabel('Counts', fontsize=20)
ax2.tick_params(axis='both', labelsize=15)

# make the data read to feed into the visulizer
X_entry_visa = data.groupby('entry_visa').size().reset_index(name='Count
s').nlargest(20, columns=['Counts'])['entry_visa']
Y_entry_visa = data.groupby('entry_visa').size().reset_index(name='Count
s').nlargest(20, columns=['Counts'])['Counts']
# make the bar plot
ax3.bar(X_entry_visa, Y_entry_visa)
ax3.set_title('Entry Visa Type', fontsize=25)
ax3.set_ylabel('Counts', fontsize=20)
ax3.tick_params(axis='both', labelsize=15, labelrotation=90)

# make the data read to feed into the visulizer
X_citizenship = data.groupby('citizenship').size().reset_index(name='Cou
nts').nlargest(25, columns=['Counts'])['citizenship']
Y_citizenship = data.groupby('citizenship').size().reset_index(name='Cou
nts').nlargest(25, columns=['Counts'])['Counts']
# make the bar plot
ax4.bar(X_citizenship, Y_citizenship)
ax4.set_title('Citizenship Country', fontsize=25)
ax4.set_ylabel('Counts', fontsize=20)
```

```
ax4.tick_params(axis='both', labelsize=15, labelrotation=90)

plt.show()
```



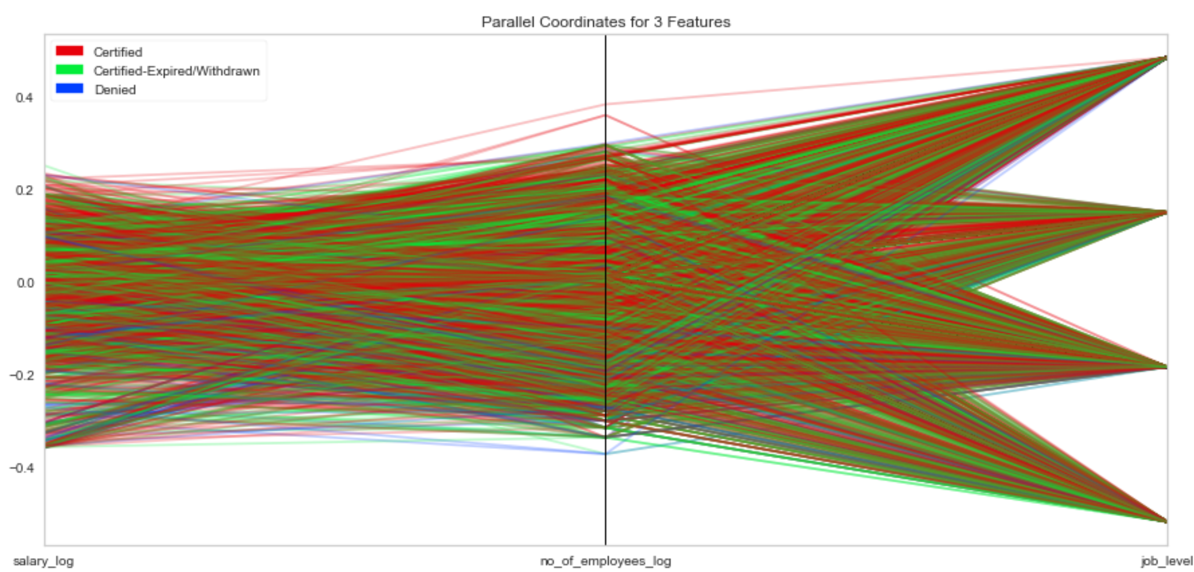
Step 7: Compare features using Pearson Ranking

In [15]:
```
#set up the figure size
#%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 7)

# extract the numpy arrays from the data frame
num_features = ['salary_log', 'no_of_employees_log', 'job_level', 'year'
]
X = data[num_features].to_numpy()

# instantiate the visualizer with the Covariance ranking algorithm
visualizer = Rank2D(features=num_features, algorithm='pearson')
visualizer.fit(X)                    # Fit the data to the visualizer
visualizer.transform(X)              # Transform the data
visualizer.poof(outpath="case_study_data/pearson1.png") # Draw/show/poof
the data
plt.show()
```



Step 8: Compare features againt case status using parallel coordinates

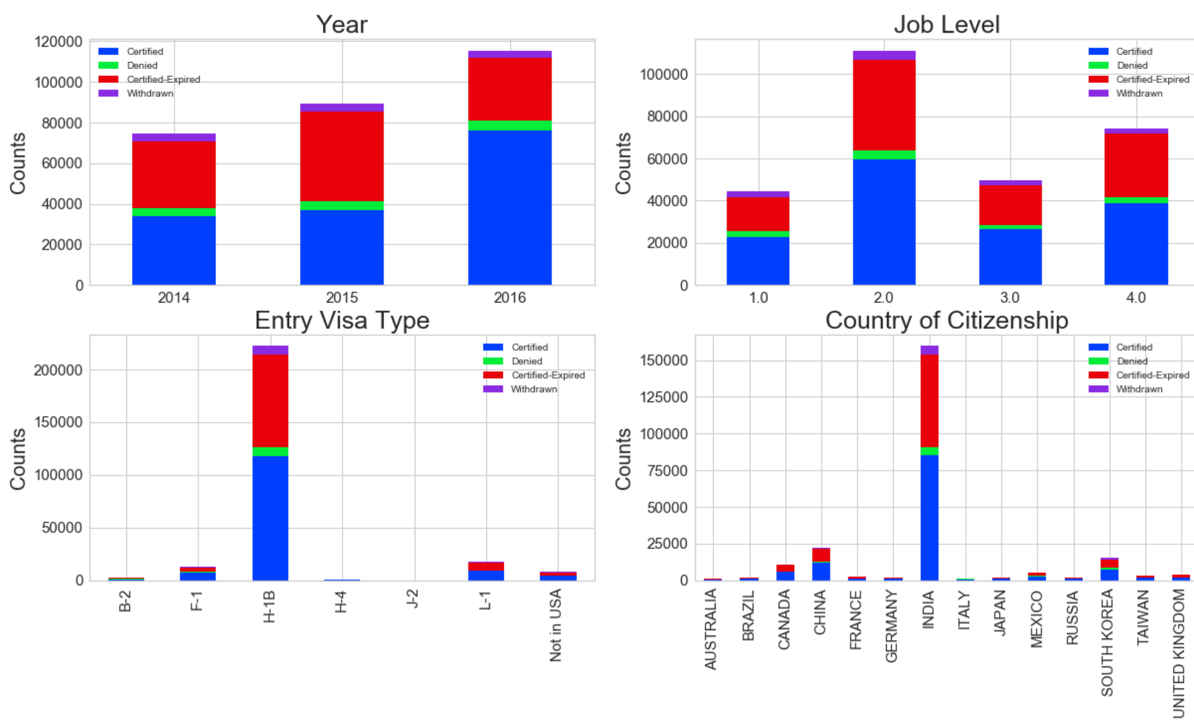
In [16]:
```
#%matplotlib inline
#set up the figure size
plt.rcParams['figure.figsize'] = (15, 7)
plt.rcParams['font.size'] = 50

# Set palette color
set_palette('sns_bright')

# Specify the features of interest and the classes of the target
classes = ['Denied', 'Certified-Expired/Withdrawn', 'Certified']
num_features = ['salary_log', 'no_of_employees_log', 'job_level']

# copy data to a new dataframe and transform case_status to numeric
data_norm = data.copy().replace({'case_status':{'Certified': float(1),
                                    'Certified-Expired': float(0.5),
                                    'Withdrawn': float(0.5),
                                    'Denied': float(0)}})
# normalize data to 0-1 range
for feature in num_features:
    data_norm[feature] = (data[feature] - data[feature].mean()) / (
                          data[feature].max() - data[feature].min())

# Extract the numpy arrays from the data frame
X = data_norm[num_features].to_numpy()
y = data_norm.case_status.to_numpy()

# Instantiate the visualizer
# Used options sample & shuffle to get speader plot, otherwise fitting w
ould take more time.
visualizer = ParallelCoordinates(classes=classes, features=num_features,
                                 sample=0.02, shuffle=True)

visualizer.fit(X, y)                                 # Fit the data to the
 visualizer
visualizer.transform(X)                              # Transform the data
visualizer.poof(outpath="case_study_data/pcords2.png") # Draw/show/poof
 the data
plt.show()
```



Step 9: Compare visa counts using stacked bar charts for several features

In [17]:
```
#%matplotlib inline

def plotbarchart(df, field, ax):
    """This method prepares the temporary dataframe using the field prov
ided
        and plots the stacked bar chart using the axes object provide
d."""
    certified = df[df['case_status']=='Certified'][field].value_counts()

    denied = df[df['case_status']=='Denied'][field].value_counts()
    denied = denied.reindex(index = denied.index)

    expired = df[df['case_status']=='Certified-Expired'][field].value_co
unts()
    expired = expired.reindex(index = expired.index)

    withdrawn = df[df['case_status']=='Withdrawn'][field].value_counts()
    withdrawn = withdrawn.reindex(index = withdrawn.index)

    df = pd.concat([certified, denied, expired, withdrawn], axis=1, sort
=True)
    df.columns = ['Certified', 'Denied', 'Certified-Expired', 'Withdraw
n']

    df.plot.bar(stacked=True, ax=ax)

#set up the figure size
plt.rcParams['figure.figsize'] = (20, 10)

# make subplots
fig, axes = plt.subplots(nrows = 2, ncols = 2)
ax1 = axes[0, 0]
ax2 = axes[0, 1]
ax3 = axes[1, 0]
ax4 = axes[1, 1]

# Get stacked bar chart for counts by year
plotbarchart(data, 'year', ax1)

ax1.set_title('Year', fontsize=25)
ax1.set_ylabel('Counts', fontsize=20)
ax1.tick_params(axis='both', labelsize=15, rotation=0)

# Get stacked bar chart for counts by job level
plotbarchart(data, 'job_level', ax2)

ax2.set_title('Job Level', fontsize=25)
ax2.set_ylabel('Counts', fontsize=20)
ax2.tick_params(axis='both', labelsize=15, rotation=0)


# Get stacked bar chart for counts by entry visa type
# Filtered few popular visa types to get a better view in bar chart
visa_types = ['H-1B', 'Not in USA', 'F-1', 'J-2', 'B-2', 'H-4', 'L-1']
data_temp = data[data['entry_visa'].isin(visa_types)]
```

```
plotbarchart(data_temp, 'entry_visa', ax3)

ax3.set_title('Entry Visa Type', fontsize=25)
ax3.set_ylabel('Counts', fontsize=20)
ax3.tick_params(axis='both', labelsize=15)

# Get stacked bar chart for counts by citizenship country
# Filtered few countries out of top 25 to get a better view in bar chart
countries = ['INDIA', 'CHINA', 'CANADA', 'UNITED KINGDOM', 'FRANCE',
             'SOUTH KOREA', 'MEXICO', 'BRAZIL', 'TAIWAN', 'AUSTRALIA',
             'GERMANY', 'RUSSIA', 'ITALY', 'JAPAN']
data_temp = data[data['citizenship'].isin(countries)]

plotbarchart(data_temp, 'citizenship', ax4)

ax4.set_title('Country of Citizenship', fontsize=25)
ax4.set_ylabel('Counts', fontsize=20)
ax4.tick_params(axis='both', labelsize=15)

plt.show()
```



Step 10: Convert categorical data to numbers

In [18]:
```
# Get temporary dataset with only certified and denied rows
data_temp = data[(data['case_status'] == 'Certified') | (data['case_stat
us'] == 'Denied')]
# Prepare a list of categorical data
cat_features = ['entry_visa', 'citizenship', 'state']
# One Hot Encoding
data_cat_dummies = pd.get_dummies(data_temp[cat_features])
# Check data
print("\nData before conversion:\n", data_temp[cat_features].head(8))
print("\nData after conversion:\n", data_cat_dummies.head(8))
```

```
Data before conversion:
    entry_visa citizenship           state
2        H-1B       INDIA        NEW YORK
4        H-1B       INDIA       WISCONSIN
7        H-1B       INDIA        NEW YORK
23       H-1B       INDIA        MICHIGAN
24       H-1B       INDIA      CALIFORNIA
26        E-3   AUSTRALIA  NORTH CAROLINA
34       H-1B       INDIA         GEORGIA
35       H-1B       INDIA        NEW YORK

Data after conversion:
     entry_visa_A-3  entry_visa_A1/A2  entry_visa_B-1  entry_visa_B-2
\
2                 0                 0               0               0
4                 0                 0               0               0
7                 0                 0               0               0
23                0                 0               0               0
24                0                 0               0               0
26                0                 0               0               0
34                0                 0               0               0
35                0                 0               0               0

    entry_visa_C-1  entry_visa_C-3  entry_visa_D-1  entry_visa_E-1  \
2                0               0               0               0
4                0               0               0               0
7                0               0               0               0
23               0               0               0               0
24               0               0               0               0
26               0               0               0               0
34               0               0               0               0
35               0               0               0               0

    entry_visa_E-2  entry_visa_E-3  ...  state_VIRGINIA  state_VT  stat
e_WA  \
2                0               0  ...               0         0
0
4                0               0  ...               0         0
0
7                0               0  ...               0         0
0
23               0               0  ...               0         0
0
24               0               0  ...               0         0
0
26               0               1  ...               0         0
0
34               0               0  ...               0         0
0
35               0               0  ...               0         0
0

    state_WASHINGTON  state_WEST VIRGINIA  state_WI  state_WISCONSIN  \
2                  0                    0         0                0
4                  0                    0         0                1
7                  0                    0         0                0
23                 0                    0         0                0
```

```
24                  0                     0         0                0
26                  0                     0         0                0
34                  0                     0         0                0
35                  0                     0         0                0

    state_WV  state_WY  state_WYOMING
2          0         0              0
4          0         0              0
7          0         0              0
23         0         0              0
24         0         0              0
26         0         0              0
34         0         0              0
35         0         0              0

[8 rows x 348 columns]
```

Step 11: Create final feature datasets that can be used for train and validation

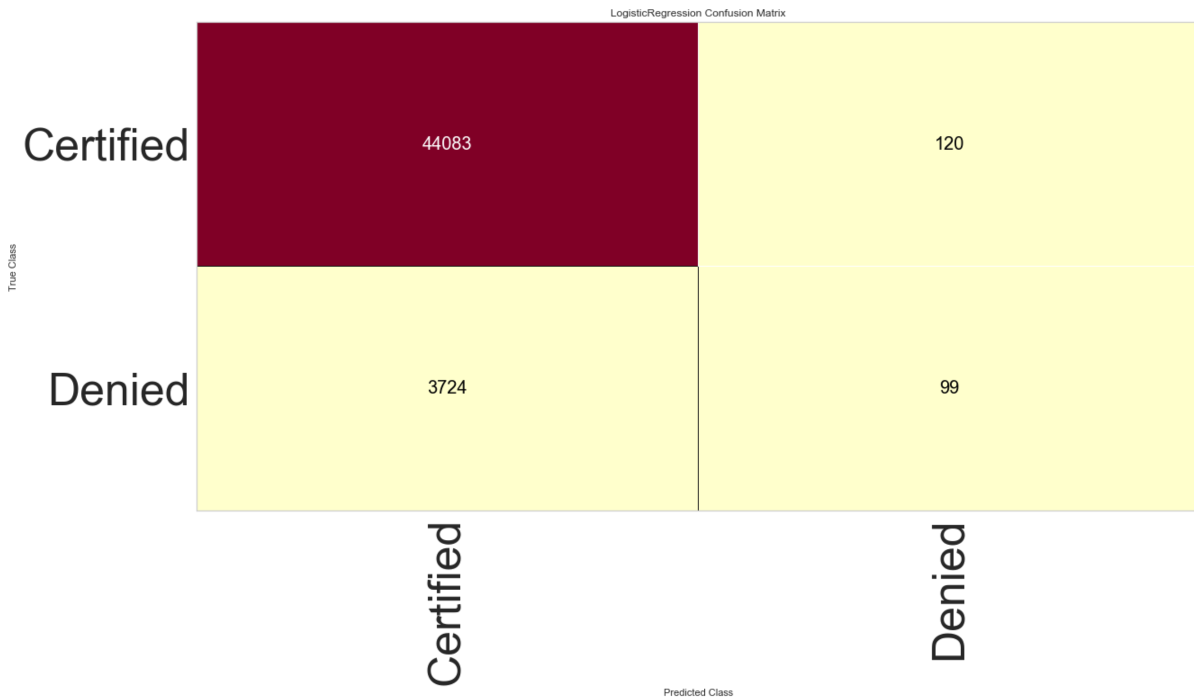
In [19]:
```
# Here we will combine the numerical features and the dummie features to
gether
# Excluded state column alone, considered all other features for model
features_model = ['no_of_employees_log', 'job_level', 'year', 'salary_lo
g']
data_model_X = pd.concat([data_temp[features_model], data_cat_dummies],
axis=1)
data_model_y = data_temp['case_status']

# separate data into training and validation and check the details of th
e datasets
# import packages
from sklearn.model_selection import train_test_split
# split the data
X_train, X_val, y_train, y_val = train_test_split(data_model_X, data_mod
el_y, test_size =0.3, random_state=11)

# number of samples in each set
print("No. of samples in training set: ", X_train.shape[0])
print("No. of samples in validation set:", X_val.shape[0])

# Survived and not-survived
print('\n')
print('Look at different case_status values in the training set:')
print(y_train.value_counts())

print('\n')
print('Look at different case_status values in the validation set:')
print(y_val.value_counts())
```

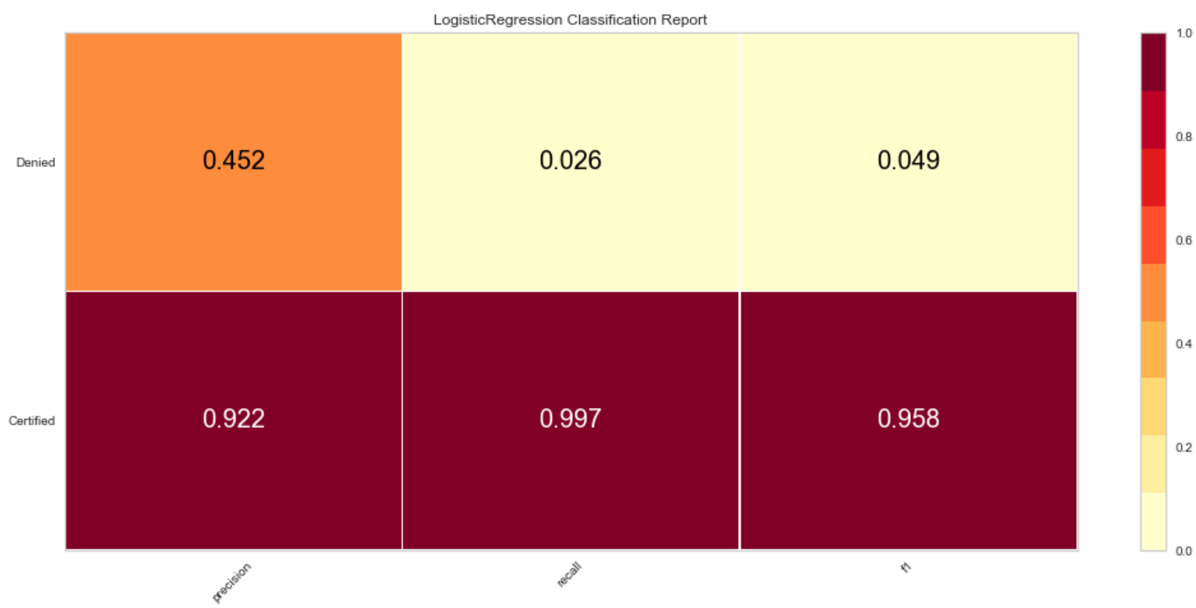
```
No. of samples in training set:  112060
No. of samples in validation set: 48026


Look at different case_status values in the training set:
Certified    103010
Denied         9050
Name: case_status, dtype: int64


Look at different case_status values in the validation set:
Certified    44203
Denied        3823
Name: case_status, dtype: int64
```

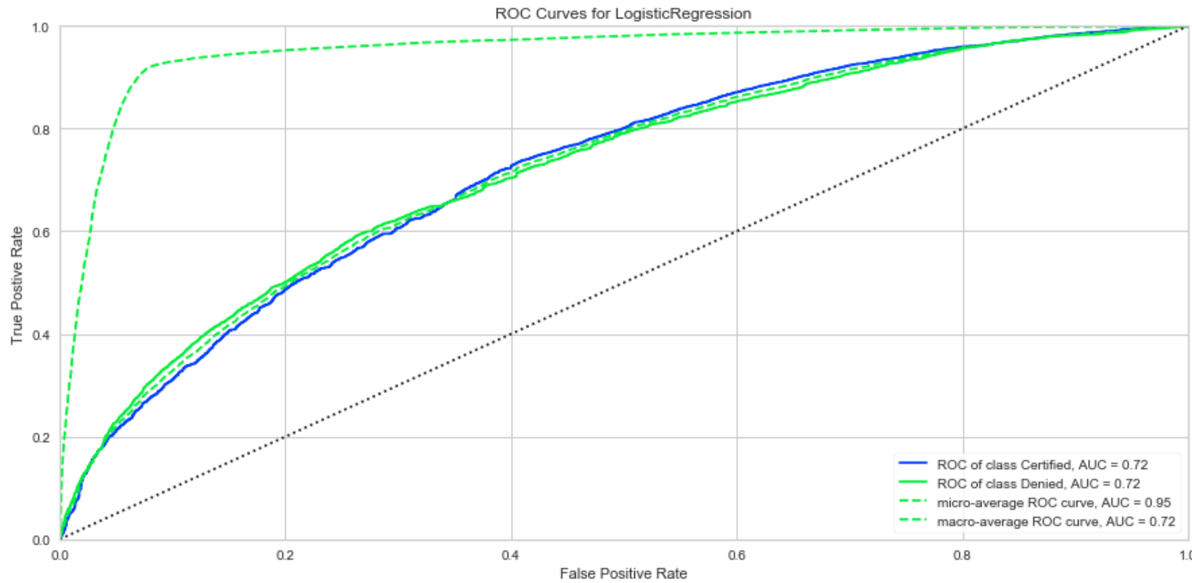
Step 12: Create logistic regression model and evaluate the same

In [20]:
```
from sklearn.linear_model import LogisticRegression

from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import ROCAUC

# Instantiate the classification model
model = LogisticRegression()

#The ConfusionMatrix visualizer taxes a model
classes = ['Certified','Denied']
cm = ConfusionMatrix(model, classes=classes, percent=False)

#Fit fits the passed model. This is unnecessary if you pass the visualiz
er a pre-fitted model
cm.fit(X_train, y_train)

#To create the ConfusionMatrix, we need some test data. Score runs predi
ct() on the data
#and then creates the confusion_matrix from scikit learn.
cm.score(X_val, y_val)

# change fontsize of the labels in the figure
for label in cm.ax.texts:
    label.set_size(20)

#How did we do?
cm.poof()

# Precision, Recall, and F1 Score
# set the size of the figure and the font size
#%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 7)
plt.rcParams['font.size'] = 20

# Instantiate the visualizer
visualizer = ClassificationReport(model, classes=classes)

visualizer.fit(X_train, y_train)  # Fit the training data to the visuali
zer
visualizer.score(X_val, y_val)  # Evaluate the model on the test data
g = visualizer.poof()

# ROC and AUC
#Instantiate the visualizer
visualizer = ROCAUC(model)

visualizer.fit(X_train, y_train)  # Fit the training data to the visuali
zer
visualizer.score(X_val, y_val)  # Evaluate the model on the test data
g = visualizer.poof()
```

```
/Users/chandramouliyalamanchili/anaconda3/lib/python3.7/site-packages/s
klearn/linear_model/logistic.py:432: FutureWarning: Default solver will
be changed to 'lbfgs' in 0.22. Specify a solver to silence this warnin
g.
  FutureWarning)
```






ROC Curves for LogisticRegression
True Postive Rate
False Positive Rate
ROC of class Certified, AUC = 0.72
ROC of class Denied, AUC = 0.72
micro-average ROC curve, AUC = 0.95
macro-average ROC curve, AUC = 0.72
0.0
0.2
0.4
0.6
0.8
1.0