```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
%matplotlib inline
```

This code block is written in Python and imports several libraries:

numpy is imported with the alias np. It's a popular library for numerical computing in Python, providing efficient arrays and mathematical operations.
pandas is imported with the alias pd. It's a library used for data manipulation and analysis, offering data structures and functions for working with structured data, such as CSV files.
matplotlib.pyplot is imported as plt. This library provides a wide range of functions for creating various types of plots and visualizations.
%matplotlib inline is a magic command used in Jupyter Notebook or JupyterLab environments to display matplotlib plots directly in the notebook output without the need for additional commands.
By importing these libraries, the code sets up the necessary tools for data processing, analysis, and visualization, which will be used later in the code.

```
train = pd.read_csv('../input/act_train.csv', parse_dates=['date'])
test = pd.read_csv('../input/act_test.csv', parse_dates=['date'])
ppl = pd.read_csv('../input/people.csv', parse_dates=['date'])

df_train = pd.merge(train, ppl, on='people_id')
df_test = pd.merge(test, ppl, on='people_id')
del train, test, ppl
```

This code block reads and merges several CSV files using the pandas library:

train = pd.read_csv('../input/act_train.csv', parse_dates=['date']): This line reads the CSV file named 'act_train.csv' located in the '../input/' directory. The parse_dates=['date'] argument instructs pandas to parse the 'date' column as datetime objects. The data is then stored in the train DataFrame.

test = pd.read_csv('../input/act_test.csv', parse_dates=['date']): Similarly, this line reads the CSV file named 'act_test.csv' located in the '../input/' directory. The 'date' column is parsed as datetime objects, and the data is stored in the test DataFrame.

ppl = pd.read_csv('../input/people.csv', parse_dates=['date']): This line reads the CSV file named 'people.csv' located in the '../input/' directory. Again, the 'date' column is parsed as datetime objects, and the data is stored in the ppl DataFrame.

df_train = pd.merge(train, ppl, on='people_id'): The merge() function is used to combine the train and ppl DataFrames based on a common column 'people_id'. This merge operation combines the columns and rows from both DataFrames into a new DataFrame called df_train.

df_test = pd.merge(test, ppl, on='people_id'): Similarly, the merge() function is used to merge the test and ppl DataFrames based on the 'people_id' column. The resulting merged DataFrame is assigned to df_test.

del train, test, ppl: This line deletes the train, test, and ppl DataFrames to free up memory. Since the merged DataFrames (df_train and df_test) contain all the necessary data, there is no need to keep the individual DataFrames used for merging.

```
for d in ['date_x', 'date_y']:
    print('Start of ' + d + ': ' + str(df_train[d].min().date()))
    print('  End of ' + d + ': ' + str(df_train[d].max().date()))
    print('Range of ' + d + ': ' + str(df_train[d].max() - df_train[d].min()) + '\n')
```

This code block iterates over a list of column names (['date_x', 'date_y']) and prints information about each column in the df_train DataFrame:

for d in ['date_x', 'date_y']:: This line initiates a loop where d takes on the values 'date_x' and 'date_y', iterating over each item in the list.

print('Start of ' + d + ': ' + str(df_train[d].min().date())): This line prints the minimum value of the column specified by d in the df_train DataFrame. The .min() function returns the minimum value, .date() converts it to a date object, and the result is concatenated with a string for display.

print(' End of ' + d + ': ' + str(df_train[d].max().date())): Similarly, this line prints the maximum value of the column specified by d in the df_train DataFrame. The .max() function returns the maximum value, .date() converts it to a date object, and the result is concatenated with a string for display.

print('Range of ' + d + ': ' + str(df_train[d].max() - df_train[d].min()) + '\n'): This line calculates and prints the range of values in the column specified by d in the df_train DataFrame. The range is obtained by subtracting the minimum value from the maximum value. The result is concatenated with a string for display, followed by a new line character (\n) for spacing between iterations.

Overall, this code provides insights into the start date, end date, and range of values for the specified columns in the df_train DataFrame.

```
date_x = pd.DataFrame()
date_x['Class probability'] = df_train.groupby('date_x')['outcome'].mean()
date_x['Frequency'] = df_train.groupby('date_x')['outcome'].size()
date_x.plot(secondary_y='Frequency', figsize=(20, 10))
```

This code block performs some operations on the df_train DataFrame related to the 'date_x' column and generates a plot using matplotlib.pyplot:

date_x = pd.DataFrame(): This line creates an empty DataFrame called date_x.

date_x['Class probability'] = df_train.groupby('date_x')['outcome'].mean(): Here, the 'outcome' column of the df_train DataFrame is grouped by the 'date_x' column. The .mean() function calculates the mean value of the 'outcome' column for each unique date in 'date_x'. The resulting values are assigned to the 'Class probability' column in the date_x DataFrame.

date_x['Frequency'] = df_train.groupby('date_x')['outcome'].size(): Similarly, this line groups the 'outcome' column by the 'date_x' column and uses the .size() function to count the number of occurrences of each unique date. The resulting counts are assigned to the 'Frequency' column in the date_x DataFrame.

date_x.plot(secondary_y='Frequency', figsize=(20, 10)): This line generates a plot using the plot() function of the date_x DataFrame. The 'Class probability' column is plotted on the primary y-axis, and the 'Frequency' column is plotted on the secondary y-axis. The secondary_y='Frequency' argument specifies the column to be plotted on the secondary y-axis. The figsize=(20, 10) argument sets the size of the figure to 20 inches in width and 10 inches in height.

The resulting plot visualizes the relationship between the 'Class probability' and 'Frequency' columns over the unique dates in the 'date_x' column of the df_train DataFrame.

```
date_y = pd.DataFrame()
date_y['Class probability'] = df_train.groupby('date_y')['outcome'].mean()
date_y['Frequency'] = df_train.groupby('date_y')['outcome'].size()
# We need to split it into multiple graphs since the time-scale is too long to show well on one graph
i = int(len(date_y) / 3)
date_y[:i].plot(secondary_y='Frequency', figsize=(20, 5), title='date_y Year 1')
date_y[i:2*i].plot(secondary_y='Frequency', figsize=(20, 5), title='date_y Year 2')
date_y[2*i:].plot(secondary_y='Frequency', figsize=(20, 5), title='date_y Year 3')
```

This code block further analyzes the 'date_y' column from the df_train DataFrame and generates multiple plots to accommodate the long time-scale:

date_y = pd.DataFrame(): This line creates an empty DataFrame called date_y.

date_y['Class probability'] = df_train.groupby('date_y')['outcome'].mean(): The 'outcome' column of the df_train DataFrame is grouped by the 'date_y' column. The .mean() function calculates the mean value of the 'outcome' column for each unique date in 'date_y'. The resulting mean values are assigned to the 'Class probability' column in the date_y DataFrame. date_y['Frequency'] = df_train.groupby('date_y')['outcome'].size(): Similarly, this line groups the 'outcome' column by the 'date_y' column and uses the .size() function to count the number of occurrences of each unique date. The resulting counts are assigned to the 'Frequency' column in the date_y DataFrame. i = int(len(date_y) / 3): This line calculates the value of i by dividing the length of the date_y DataFrame by 3 and converting it to an integer. It determines the splitting point for the subsequent plots.

date_y[:i].plot(secondary_y='Frequency', figsize=(20, 5), title='date_y Year 1'): The first plot displays the portion of date_y from the beginning to the index i. The 'Class probability' column is plotted on the primary y-axis, and the 'Frequency' column is plotted on the secondary y-axis. The figsize=(20, 5) argument sets the size of the figure to 20 inches in width and 5 inches in height. The plot is given the title 'date_y Year 1'.

date_y[i:2*i].plot(secondary_y='Frequency', figsize=(20, 5), title='date_y Year 2'): The second plot displays the portion of date_y from index i to index 2*i. It has the same configuration as the first plot but covers a different time period. The plot is given the title 'date_y Year 2'.

date_y[2*i:].plot(secondary_y='Frequency', figsize=(20, 5), title='date_y Year 3'): The third plot displays the portion of date_y from index 2*i to the end. It follows the same configuration as the previous plots but represents a different time period. The plot is given the title 'date_y Year 3'.

By splitting the data into three plots and assigning appropriate titles, this code block visualizes the relationship between 'Class probability' and 'Frequency' over time for the 'date_y' column in the df_train DataFrame. Each plot represents a different year, allowing for a more detailed examination of the data.

```
date_x_freq = pd.DataFrame()
date_x_freq['Training set'] = df_train.groupby('date_x')['activity_id'].count()
date_x_freq['Testing set'] = df_test.groupby('date_x')['activity_id'].count()
date_x_freq.plot(secondary_y='Testing set', figsize=(20, 8),
                 title='Comparison of date_x distribution between training/testing set')
date_y_freq = pd.DataFrame()
date_y_freq['Training set'] = df_train.groupby('date_y')['activity_id'].count()
date_y_freq['Testing set'] = df_test.groupby('date_y')['activity_id'].count()
date_y_freq[:i].plot(secondary_y='Testing set', figsize=(20, 8),
                 title='Comparison of date_y distribution between training/testing set (first year)')
date_y_freq[2*i:].plot(secondary_y='Testing set', figsize=(20, 8),
                 title='Comparison of date_y distribution between training/testing set (last year)')
```

This code block compares the distribution of the 'date_x' and 'date_y' columns between the training and testing datasets by creating plots:

date_x_freq = pd.DataFrame(): This line initializes an empty DataFrame called date_x_freq to store the frequency information.

date_x_freq['Training set'] = df_train.groupby('date_x')['activity_id'].count(): The 'activity_id' column of the df_train DataFrame is grouped by the 'date_x' column. The .count() function calculates the number of occurrences of each unique date in 'date_x' for the training set. The resulting counts are assigned to the 'Training set' column in the date_x_freq DataFrame.

date_x_freq['Testing set'] = df_test.groupby('date_x')['activity_id'].count(): Similarly, this line groups the 'activity_id' column by the 'date_x' column in the df_test DataFrame. It calculates the counts of occurrences for each unique date in 'date_x' for the testing set and assigns them to the 'Testing set' column in the date_x_freq DataFrame.

date_x_freq.plot(secondary_y='Testing set', figsize=(20, 8), title='Comparison of date_x distribution between training/testing set'): This line generates a plot using the plot() function of the date_x_freq DataFrame. The 'Training set' column is plotted on the primary y-axis, and the 'Testing set' column is plotted on the secondary y-axis. The secondary_y='Testing set' argument specifies the column to be plotted on the secondary y-axis. The figsize=(20, 8) argument sets the size of the figure to 20 inches in width and 8 inches in height. The plot is given the title 'Comparison of date_x distribution between training/testing set'.

date_y_freq = pd.DataFrame(): This line initializes an empty DataFrame called date_y_freq to store the frequency information.

date_y_freq['Training set'] = df_train.groupby('date_y')['activity_id'].count(): The 'activity_id' column of the df_train DataFrame is grouped by the 'date_y' column. The .count() function calculates the number of occurrences of each unique date in 'date_y' for the training set. The resulting counts are assigned to the 'Training set' column in the date_y_freq DataFrame.

date_y_freq['Testing set'] = df_test.groupby('date_y')['activity_id'].count(): Similarly, this line groups the 'activity_id' column by the 'date_y' column in the df_test DataFrame. It calculates the counts of occurrences for each unique date in 'date_y' for the testing set and assigns them to the 'Testing set' column in the date_y_freq DataFrame.

date_y_freq[:i].plot(secondary_y='Testing set', figsize=(20, 8), title='Comparison of date_y distribution between training/testing set (first year)'): This line generates a plot for the first year using the portion of the date_y_freq DataFrame from the beginning to index i. The configuration of the plot is similar to the previous one but focuses on the first year of data. It is given the title 'Comparison of date_y distribution between training/testing set (first year)'.

date_y_freq[2*i:].plot(secondary_y='Testing set', figsize=(20, 8), title='Comparison of date_y distribution between training/testing set (last year)'): Similarly, this line generates a plot for the last year using the portion of the `date

```
print('Correlation of date_x distribution in training/testing sets: ' +
str(np.corrcoef(date_x_freq.T)[0,1]))
print('Correlation of date_y distribution in training/testing sets: ' +
str(np.corrcoef(date_y_freq.fillna(0).T)[0,1]))
```

This code block calculates and prints the correlation coefficients between the date distributions in the training and testing sets for the 'date_x' and 'date_y' columns:

np.corrcoef(date_x_freq.T): The np.corrcoef() function calculates the correlation coefficients between the columns of the date_x_freq DataFrame. The .T attribute is used to transpose the DataFrame, allowing the function to compute the correlation between the 'Training set' and 'Testing set' columns.

np.corrcoef(date_y_freq.fillna(0).T): Similarly, this line calculates the correlation coefficients between the columns of the date_y_freq DataFrame. The .fillna(0) method is used to replace any missing values in the DataFrame with zeros before calculating the correlation coefficients.

str(np.corrcoef(date_x_freq.T)[0,1]): This line converts the correlation coefficient between the 'Training set' and 'Testing set' columns of date_x_freq into a string.

str(np.corrcoef(date_y_freq.fillna(0).T)[0,1]): Likewise, this line converts the correlation coefficient between the 'Training set' and 'Testing set' columns of date_y_freq into a string.

Printing the correlation coefficients: The calculated correlation coefficients are concatenated with the respective strings and printed as output.

In relation to the previous code, these lines provide additional analysis by quantifying the correlation between the date distributions in the training and testing sets for both 'date_x' and 'date_y' columns. The correlation coefficient helps assess the similarity or relationship between the two sets, providing insights into potential discrepancies or consistency in the data distribution across the sets.

This code block calculates and prints the correlation coefficients between the 'date_y' distributions in different years of the training and testing sets:

np.corrcoef(date_y_freq[:i].fillna(0).T): This line calculates the correlation coefficients between the columns of the date_y_freq DataFrame for the first year. The [:i] indexing selects the portion

```
print('date_y correlation in year 1: ' + str(np.corrcoef(date_y_freq[:i].fillna(0).T)[0,1]))
print('date_y correlation in year 2: ' + str(np.corrcoef(date_y_freq[i:2*i].fillna(0).T)[0,1]))
print('date_y correlation in year 3: ' + str(np.corrcoef(date_y_freq[2*i:].fillna(0).T)[0,1]))
```

of date_y_freq corresponding to the first year, the .fillna(0) method replaces any missing values with zeros, and the .T attribute transposes the DataFrame for correlation coefficient calculation.

np.corrcoef(date_y_freq[i:2*i].fillna(0).T): Similarly, this line calculates the correlation coefficients between the columns of date_y_freq for the second year. The [i:2*i] indexing selects the portion corresponding to the second year.

np.corrcoef(date_y_freq[2*i:].fillna(0).T): Likewise, this line calculates the correlation coefficients between the columns of date_y_freq for the third year. The [2*i:] indexing selects the portion corresponding to the third year.

str(np.corrcoef(date_y_freq[:i].fillna(0).T)[0,1]): This line converts the correlation coefficient between the 'Training set' and 'Testing set' columns of date_y_freq for the first year into a string.

str(np.corrcoef(date_y_freq[i:2*i].fillna(0).T)[0,1]): Similarly, this line converts the correlation coefficient between the 'Training set' and 'Testing set' columns of date_y_freq for the second year into a string.

str(np.corrcoef(date_y_freq[2*i:].fillna(0).T)[0,1]): Likewise, this line converts the correlation coefficient between the 'Training set' and 'Testing set' columns of date_y_freq for the third year into a string.

Printing the correlation coefficients: The calculated correlation coefficients for each year are concatenated with the respective strings and printed as output.

This code block provides further analysis by calculating and printing the correlation coefficients for each year separately, allowing for a more detailed understanding of the relationship between the 'date_y' distributions in the training and testing sets.

```
from sklearn.metrics import roc_auc_score
features = pd.DataFrame()
features['date_x_prob'] = df_train.groupby('date_x')['outcome'].transform('mean')
features['date_y_prob'] = df_train.groupby('date_y')['outcome'].transform('mean')
features['date_x_count'] = df_train.groupby('date_x')['outcome'].transform('count')
features['date_y_count'] = df_train.groupby('date_y')['outcome'].transform('count')
_=[print(f.ljust(12) + ' AUC: ' + str(round(roc_auc_score(df_train['outcome'], features[f]), 6))) for f
in features.columns]
```

This code block performs the following tasks:

Importing the necessary libraries: The code imports the roc_auc_score function from the sklearn.metrics module, which is used to calculate the Area Under the ROC Curve (AUC).

Creating a DataFrame called features: This DataFrame is initialized to store the extracted features.

Calculating the 'date_x' probability: The code groups the 'outcome' column of the df_train DataFrame by the 'date_x' column and calculates the mean of 'outcome' for each unique date. The resulting probabilities are assigned to the 'date_x_prob' column in the features DataFrame using the transform('mean') method.

Calculating the 'date_y' probability: Similarly, the code groups the 'outcome' column by the 'date_y' column and calculates the mean of 'outcome' for each unique date. The resulting probabilities are assigned to the 'date_y_prob' column in the features DataFrame using the transform('mean') method.

Calculating the 'date_x' count: The code groups the 'outcome' column by the 'date_x' column and counts the occurrences of 'outcome' for each unique date. The resulting counts are assigned to the 'date_x_count' column in the features DataFrame using the transform('count') method.

Calculating the 'date_y' count: Similarly, the code groups the 'outcome' column by the 'date_y' column and counts the occurrences of 'outcome' for each unique date. The resulting counts are assigned to the 'date_y_count' column in the features DataFrame using the transform('count') method.

Printing the AUC scores: For each column in the features DataFrame, the code calculates the AUC score by comparing the 'outcome' column of the df_train DataFrame with the corresponding feature column. The AUC score is rounded to six decimal places and printed, along with the column name.

In summary, this code block creates a DataFrame (features) that contains extracted features related to the 'date_x' and 'date_y' columns. It calculates the probabilities and counts of 'outcome' for each unique date in 'date_x' and 'date_y'. Then, it uses the roc_auc_score function to calculate the AUC scores for each feature column and prints the scores.