Preparing Data for Machine Learning

Concept Session

Demo 1.1: Data Preparation

Loading Library

In [37]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split

# enable inline plots in the notebook
%matplotlib inline
```

Loading Data

In [ ]:
```
loan_df = pd.read_csv('/content/drive/MyDrive/ML/DS2_C5_S1_Loan_Data_Concept.csv')
loan_df.head()
```

In [ ]:
```
# Datatypes of all columns
print(loan_df.dtypes)
```

Data Preparation

Understanding data structure

In [ ]:
```
# All columns
loan_df_c=loan_df
columns=loan_df_c.columns
columns
```

In [ ]:
```
# structure of dataset
print(loan_df_c.shape)
```

In [ ]:
```
# Summary of dataset
loan_df_c.describe()
```

In [ ]:
```
# Count of each label in categorical column
loan_df_c.purpose.value_counts()
```

In [ ]:
```
loan_df_c['credit.policy'].value_counts()
```

In [ ]:
```
loan_df_c['not.fully.paid'].value_counts()
```

Missing Value Treatment

In [ ]:
```
# Checking for null value in each column
print(loan_df_c.isnull().sum())
```

We find that there is no missing value exist in this dataset, so we skip the missing value treatment

In [ ]:
```
"""
# Use this following procedure when you have any missing value
#For numerical column
loan_df_c.fillna(loan_df_c.mean(), inplace=True) # you can use median()
#For categorical column
income_df_c = income_df_c.apply(lambda x: x.fillna(x.value_counts().index[0]))
"""
```

Feature Engineering

In [ ]:
```
# Converting numeric labeled column into categorical column
loan_df_c['credit.policy']=loan_df_c['credit.policy'].astype('category')
loan_df_c['purpose']=loan_df_c['purpose'].astype('category')
print(loan_df.head())
print(loan_df.dtypes)
```

In [ ]:
```
# Using Label Encoder technique to convert categorical column into numerical type

#label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
loan_df_c['purpose_encode']= label_encoder.fit_transform(loan_df_c['purpose'])

loan_df_c['purpose_encode'].unique()
#print(loan_df.head())
```

In [ ]:
```
#label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
loan_df_c['not.fully.paid_encode']= label_encoder.fit_transform(loan_df_c['not.fully.paid'])

loan_df_c['not.fully.paid_encode'].unique()
loan_df_c['not.fully.paid_encode']=loan_df_c['not.fully.paid_encode'].astype('category')
#print(loan_df.head())
```

Feature Scaling

In [ ]:
```
# Defining method to perform data scaling operation based on the type of scaling
def feature_scale(scale):
  numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
  new_loan_df = loan_df_c.select_dtypes(include=numerics)
  if scale == 'minmax':
    scaler = MinMaxScaler()
  else:
    scaler = StandardScaler()
  df_scaled = pd.DataFrame(scaler.fit_transform(new_loan_df.to_numpy()),columns=new_loan_df.columns)
  return df_scaled
```

In [ ]:
```
# scaling the data using MinMax Scaling process
scale = 'minmax' #standard
loan_df_scaled=feature_scale(scale)
loan_df_scaled
```

Feature Selection

In [ ]:
```
# Finding correlation among numerical features, based on their strong relation we can choose the import
atnt features
corr=loan_df_scaled.corr()
corr.style.background_gradient(cmap='coolwarm')
```

In [ ]:
```
new_df_scaled=loan_df_scaled[['int.rate','installment', 'dti','fico', 'days.with.cr.line', 'revol.bal',
'revol.util']]
hm = sns.heatmap(new_df_scaled.corr(), annot = True)
hm.set(xlabel='\nLoan Details', ylabel='Loan Details', title = "Correlation matrix of Loan Data\n")
plt.show()
```

Splitting Data

In [ ]:
```
# splitting dataframe by row index
loan_df_c=loan_df
train_num=int(9578*0.7)
loan_df_train = loan_df_c.iloc[:train_num,:]
loan_df_test = loan_df_c.iloc[(train_num+1):,:]
print("Shape of new dataframes - {} , {}".format(loan_df_train.shape, loan_df_test.shape))
```

In [ ]:
```
# splitting dataframe using train_test_split() built in method
y = loan_df_c['credit.policy']
X = loan_df_c
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=20)
print("Shape of new dataframes - {} , {}".format(X_train.shape, X_test.shape))
```

Data Visualization

In [ ]:
```
# Histogram of all numerical features

numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']

new_loan_df = loan_df_c.select_dtypes(include=numerics)
numeric_cols=new_loan_df.columns
num_cols=4
n_bins = 50

# compute number of rows for plot
num_rows= int(len(numeric_cols)/num_cols)+1

# setting canvas for plotting
fig, axs = plt.subplots(num_rows, num_cols, tight_layout=True,figsize=(20,10))

# plotting the numerical columns
for col, ax in zip(numeric_cols,axs.flatten()[:len(numeric_cols)]):
    ax.hist(new_loan_df[col],bins=n_bins,density=True)
    ax.set_title(col)

plt.show()
```

In [ ]:
```
# Histogram of all categorical features
num_cols = 3
cat_loan_df = loan_df_c.select_dtypes('category')
categorical_cols=cat_loan_df.columns
# compute number of rows for plot
num_rows= int(len(categorical_cols)/num_cols)

# setting canvas for plotting
fig, axs = plt.subplots(num_rows, num_cols, tight_layout=True,figsize=(20,5))

# plotting the numerical columns
for col, ax in zip(categorical_cols,axs.flatten()[:len(categorical_cols)]):
    stats = cat_loan_df[col].value_counts()
    values = list(stats)
    names = list(map(lambda x : ''.join(('value_',str(x))),list(stats.index)))
    ax.bar(names,values)
    ax.set_title(col)

plt.show()
```

In [ ]:
```
# Scatter Matrix plot of all columns
pd.plotting.scatter_matrix(loan_df_c[numeric_cols].sample(4000),figsize=(20,20))
plt.show()
```

In [ ]:
```
# Pie charts of categorical features
labels=loan_df_c['purpose'].unique()
print(labels)
def label_function(val):
    return f'{val / 100 * len(loan_df_c):.0f}\n{val:.0f}%'
fig, ax = plt.subplots(ncols=1, figsize=(10, 5))
#plt.pie(loan_df_c['purpose_encode'])
loan_df_c.groupby(loan_df_c['purpose']).size().plot(kind='pie', autopct=label_function, textprops={'fon
tsize': 10}, ax=ax)
ax.set_ylabel('Per Purpose', size=15)
# show plot
plt.show()
```

In [ ]:
```
def label_function(val):
    return f'{val / 100 * len(loan_df_c):.0f}\n{val:.0f}%'
fig, ax = plt.subplots(ncols=1, figsize=(10, 5))
#plt.pie(loan_df_c['purpose_encode'])
loan_df_c.groupby(loan_df_c['credit.policy']).size().plot(kind='pie', autopct=label_function, textprops
={'fontsize': 10}, ax=ax)
ax.set_ylabel('Per credit policy', size=22)
# show plot
plt.show()
```

In [ ]:
```
def label_function(val):
    return f'{val / 100 * len(loan_df_c):.0f}\n{val:.0f}%'
fig, ax = plt.subplots(ncols=1, figsize=(10, 5))

loan_df_c.groupby(loan_df_c['not.fully.paid_encode']).size().plot(kind='pie', autopct=label_function, t
extprops={'fontsize': 10}, ax=ax)
ax.set_ylabel('Per fully encoded', size=22)
# show plot
plt.show()
```