1. Credit card applications

```python
# Import pandas
import pandas as pd

# Load dataset
cc_apps = pd.read_csv("datasets/cc_approvals.data", header=None)

# Inspect data
print(cc_apps.head())
```

```
  0      1      2  3  4  5  6     7  8  9  10 11 12     13   14 15
0  b  30.83  0.000  u  g  w  v  1.25  t  t   1  f  g  00202    0  +
1  a  58.67  4.460  u  g  q  h  3.04  t  t   6  f  g  00043  560  +
2  a  24.50  0.500  u  g  q  h  1.50  t  f   0  f  g  00280  824  +
3  b  27.83  1.540  u  g  w  v  3.75  t  t   5  t  g  00100    3  +
4  b  20.17  5.625  u  g  w  v  1.71  t  f   0  f  s  00120    0  +
```

2. Inspecting the applications

```python
# Print summary statistics
cc_apps_description = cc_apps.describe()
print(cc_apps_description)

print("\n")

# Print DataFrame information
cc_apps_info = cc_apps.info()
print(cc_apps_info)

print("\n")

# Inspect missing values in the dataset
print(cc_apps.tail(17))
```

```
                2           7          10             14
count  690.000000  690.000000  690.00000     690.000000
mean     4.758725    2.223406    2.40000    1017.385507
std      4.978163    3.346513    4.86294    5210.102598
min      0.000000    0.000000    0.00000       0.000000
25%      1.000000    0.165000    0.00000       0.000000
50%      2.750000    1.000000    0.00000       5.000000
75%      7.207500    2.625000    3.00000     395.500000
max     28.000000   28.500000   67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
0     690 non-null object
1     690 non-null object
```

```
2     690 non-null float64
3     690 non-null object
4     690 non-null object
5     690 non-null object
6     690 non-null object
7     690 non-null float64
8     690 non-null object
9     690 non-null object
10    690 non-null int64
11    690 non-null object
12    690 non-null object
13    690 non-null object
14    690 non-null int64
15    690 non-null object
dtypes: float64(2), int64(2), object(12)
memory usage: 86.3+ KB
None


    0      1       2  3  4   5   6      7  8  9   10 11 12     13   14
15
673  ?  29.50   2.000  y  p   e   h  2.000  f  f   0  f  g  00256   17
-
674  a  37.33   2.500  u  g   i   h  0.210  f  f   0  f  g  00260  246
-
675  a  41.58   1.040  u  g  aa   v  0.665  f  f   0  f  g  00240  237
-
676  a  30.58  10.665  u  g   q   h  0.085  f  t  12  t  g  00129    3
-
677  b  19.42   7.250  u  g   m   v  0.040  f  t   1  f  g  00100    1
-
678  a  17.92  10.210  u  g  ff  ff  0.000  f  f   0  f  g  00000   50
-
679  a  20.08   1.250  u  g   c   v  0.000  f  f   0  f  g  00000    0
-
680  b  19.50   0.290  u  g   k   v  0.290  f  f   0  f  g  00280  364
-
681  b  27.83   1.000  y  p   d   h  3.000  f  f   0  f  g  00176  537
-
682  b  17.08   3.290  u  g   i   v  0.335  f  f   0  t  g  00140    2
-
683  b  36.42   0.750  y  p   d   v  0.585  f  f   0  f  g  00240    3
-
684  b  40.58   3.290  u  g   m   v  3.500  f  f   0  t  s  00400    0
-
685  b  21.08  10.085  y  p   e   h  1.250  f  f   0  f  g  00260    0
-
686  a  22.67   0.750  u  g   c   v  2.000  f  t   2  t  g  00200  394
-
687  a  25.25  13.500  y  p  ff  ff  2.000  f  t   1  t  g  00200    1
```

```
-
688  b  17.92   0.205  u  g  aa   v  0.040  f  f   0  f  g  00280  750
-
689  b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000    0
-
```

3. Handling the missing values (part i)

```python
# Import numpy
import numpy as np

# Inspect missing values in the dataset
print(cc_apps.tail(50))

# Replace the '?'s with NaN
cc_apps = cc_apps.replace('?', np.NaN)

# Inspect the missing values again
print(cc_apps.tail(50))
```

```
    0      1       2  3  4   5   6       7  8  9   10 11 12     13   14
15
640  b  34.17   2.750  u  g   i  bb  2.500  f  f   0  t  g  00232  200
-
641  ?  33.17   2.250  y  p  cc   v  3.500  f  f   0  t  g  00200  141
-
642  b  31.58   0.750  y  p  aa   v  3.500  f  f   0  t  g  00320    0
-
643  a  52.50   7.000  u  g  aa   h  3.000  f  f   0  f  g  00000    0
-
644  b  36.17   0.420  y  p   w   v  0.290  f  f   0  t  g  00309    2
-
645  b  37.33   2.665  u  g  cc   v  0.165  f  f   0  t  g  00000  501
-
646  a  20.83   8.500  u  g   c   v  0.165  f  f   0  f  g  00000  351
-
647  b  24.08   9.000  u  g  aa   v  0.250  f  f   0  t  g  00000    0
-
648  b  25.58   0.335  u  g   k   h  3.500  f  f   0  t  g  00340    0
-
649  a  35.17   3.750  u  g  ff  ff  0.000  f  t   6  f  g  00000  200
-
650  b  48.08   3.750  u  g   i  bb  1.000  f  f   0  f  g  00100    2
-
651  a  15.83   7.625  u  g   q   v  0.125  f  t   1  t  g  00000  160
-
652  a  22.50   0.415  u  g   i   v  0.335  f  f   0  t  s  00144    0
-
653  b  21.50  11.500  u  g   i   v  0.500  t  f   0  t  g  00100   68
-
654  a  23.58   0.830  u  g   q   v  0.415  f  t   1  t  g  00200   11
```

-	655	a	21.08	5.000	y	p	ff	ff	0.000	f	f	0	f	g	00000	0
-	656	b	25.67	3.250	u	g	c	h	2.290	f	t	1	t	g	00416	21
-	657	a	38.92	1.665	u	g	aa	v	0.250	f	f	0	f	g	00000	390
-	658	a	15.75	0.375	u	g	c	v	1.000	f	f	0	f	g	00120	18
-	659	a	28.58	3.750	u	g	c	v	0.250	f	t	1	t	g	00040	154
-	660	b	22.25	9.000	u	g	aa	v	0.085	f	f	0	f	g	00000	0
-	661	b	29.83	3.500	u	g	c	v	0.165	f	f	0	f	g	00216	0
-	662	a	23.50	1.500	u	g	w	v	0.875	f	f	0	t	g	00160	0
-	663	b	32.08	4.000	y	p	cc	v	1.500	f	f	0	t	g	00120	0
-	664	b	31.08	1.500	y	p	w	v	0.040	f	f	0	f	s	00160	0
-	665	b	31.83	0.040	y	p	m	v	0.040	f	f	0	f	g	00000	0
-	666	a	21.75	11.750	u	g	c	v	0.250	f	f	0	t	g	00180	0
-	667	a	17.92	0.540	u	g	c	v	1.750	f	t	1	t	g	00080	5
-	668	b	30.33	0.500	u	g	d	h	0.085	f	f	0	t	s	00252	0
-	669	b	51.83	2.040	y	p	ff	ff	1.500	f	f	0	f	g	00120	1
-	670	b	47.17	5.835	u	g	w	v	5.500	f	f	0	f	g	00465	150
-	671	b	25.83	12.835	u	g	cc	v	0.500	f	f	0	f	g	00000	2
-	672	a	50.25	0.835	u	g	aa	v	0.500	f	f	0	t	g	00240	117
-	673	?	29.50	2.000	y	p	e	h	2.000	f	f	0	f	g	00256	17
-	674	a	37.33	2.500	u	g	i	h	0.210	f	f	0	f	g	00260	246
-	675	a	41.58	1.040	u	g	aa	v	0.665	f	f	0	f	g	00240	237
-	676	a	30.58	10.665	u	g	q	h	0.085	f	t	12	t	g	00129	3
-	677	b	19.42	7.250	u	g	m	v	0.040	f	t	1	f	g	00100	1
-	678	a	17.92	10.210	u	g	ff	ff	0.000	f	f	0	f	g	00000	50
-	679	a	20.08	1.250	u	g	c	v	0.000	f	f	0	f	g	00000	0

-

680	b	19.50	0.290	u	g	k	v	0.290	f	f	0	f	g	00280	364	-
681	b	27.83	1.000	y	p	d	h	3.000	f	f	0	f	g	00176	537	-
682	b	17.08	3.290	u	g	i	v	0.335	f	f	0	t	g	00140	2	-
683	b	36.42	0.750	y	p	d	v	0.585	f	f	0	f	g	00240	3	-
684	b	40.58	3.290	u	g	m	v	3.500	f	f	0	t	s	00400	0	-
685	b	21.08	10.085	y	p	e	h	1.250	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.000	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.000	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.040	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.290	f	f	0	t	g	00000	0	-

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
640	b	34.17	2.750	u	g	i	bb	2.500	f	f	0	t	g	00232	200	-
641	NaN	33.17	2.250	y	p	cc	v	3.500	f	f	0	t	g	00200	141	-
642	b	31.58	0.750	y	p	aa	v	3.500	f	f	0	t	g	00320	0	-
643	a	52.50	7.000	u	g	aa	h	3.000	f	f	0	f	g	00000	0	-
644	b	36.17	0.420	y	p	w	v	0.290	f	f	0	t	g	00309	2	-
645	b	37.33	2.665	u	g	cc	v	0.165	f	f	0	t	g	00000	501	-
646	a	20.83	8.500	u	g	c	v	0.165	f	f	0	f	g	00000	351	-
647	b	24.08	9.000	u	g	aa	v	0.250	f	f	0	t	g	00000	0	-
648	b	25.58	0.335	u	g	k	h	3.500	f	f	0	t	g	00340	0	-
649	a	35.17	3.750	u	g	ff	ff	0.000	f	t	6	f	g	00000	200	-
650	b	48.08	3.750	u	g	i	bb	1.000	f	f	0	f	g	00100	2	-
651	a	15.83	7.625	u	g	q	v	0.125	f	t	1	t	g	00000	160	-
652	a	22.50	0.415	u	g	i	v	0.335	f	f	0	t	s	00144	0	-
653	b	21.50	11.500	u	g	i	v	0.500	t	f	0	t	g	00100		

```
68  -
654    a  23.58   0.830  u  g   q   v  0.415  f  t   1  t  g  00200
11  -
655    a  21.08   5.000  y  p  ff  ff  0.000  f  f   0  f  g  00000
0  -
656    b  25.67   3.250  u  g   c   h  2.290  f  t   1  t  g  00416
21  -
657    a  38.92   1.665  u  g  aa   v  0.250  f  f   0  f  g  00000
390  -
658    a  15.75   0.375  u  g   c   v  1.000  f  f   0  f  g  00120
18  -
659    a  28.58   3.750  u  g   c   v  0.250  f  t   1  t  g  00040
154  -
660    b  22.25   9.000  u  g  aa   v  0.085  f  f   0  f  g  00000
0  -
661    b  29.83   3.500  u  g   c   v  0.165  f  f   0  f  g  00216
0  -
662    a  23.50   1.500  u  g   w   v  0.875  f  f   0  t  g  00160
0  -
663    b  32.08   4.000  y  p  cc   v  1.500  f  f   0  t  g  00120
0  -
664    b  31.08   1.500  y  p   w   v  0.040  f  f   0  f  s  00160
0  -
665    b  31.83   0.040  y  p   m   v  0.040  f  f   0  f  g  00000
0  -
666    a  21.75  11.750  u  g   c   v  0.250  f  f   0  t  g  00180
0  -
667    a  17.92   0.540  u  g   c   v  1.750  f  t   1  t  g  00080
5  -
668    b  30.33   0.500  u  g   d   h  0.085  f  f   0  t  s  00252
0  -
669    b  51.83   2.040  y  p  ff  ff  1.500  f  f   0  f  g  00120
1  -
670    b  47.17   5.835  u  g   w   v  5.500  f  f   0  f  g  00465
150  -
671    b  25.83  12.835  u  g  cc   v  0.500  f  f   0  f  g  00000
2  -
672    a  50.25   0.835  u  g  aa   v  0.500  f  f   0  t  g  00240
117  -
673  NaN  29.50   2.000  y  p   e   h  2.000  f  f   0  f  g  00256
17  -
674    a  37.33   2.500  u  g   i   h  0.210  f  f   0  f  g  00260
246  -
675    a  41.58   1.040  u  g  aa   v  0.665  f  f   0  f  g  00240
237  -
676    a  30.58  10.665  u  g   q   h  0.085  f  t  12  t  g  00129
3  -
677    b  19.42   7.250  u  g   m   v  0.040  f  t   1  f  g  00100
1  -
678    a  17.92  10.210  u  g  ff  ff  0.000  f  f   0  f  g  00000
```

```
50  -
679    a  20.08   1.250  u  g   c   v  0.000  f  f   0  f  g  00000
0  -
680    b  19.50   0.290  u  g   k   v  0.290  f  f   0  f  g  00280
364  -
681    b  27.83   1.000  y  p   d   h  3.000  f  f   0  f  g  00176
537  -
682    b  17.08   3.290  u  g   i   v  0.335  f  f   0  t  g  00140
2  -
683    b  36.42   0.750  y  p   d   v  0.585  f  f   0  f  g  00240
3  -
684    b  40.58   3.290  u  g   m   v  3.500  f  f   0  t  s  00400
0  -
685    b  21.08  10.085  y  p   e   h  1.250  f  f   0  f  g  00260
0  -
686    a  22.67   0.750  u  g   c   v  2.000  f  t   2  t  g  00200
394  -
687    a  25.25  13.500  y  p  ff  ff  2.000  f  t   1  t  g  00200
1  -
688    b  17.92   0.205  u  g  aa   v  0.040  f  f   0  f  g  00280
750  -
689    b  35.00   3.375  u  g   c   h  8.290  f  f   0  t  g  00000
0  -
```

4. Handling the missing values (part ii)

```
# Impute the missing values with mean imputation
cc_apps.fillna(cc_apps[[2,7,10,14]].mean(),  inplace=True)

# Count the number of NaNs in the dataset to verify
print(cc_apps.isnull().sum())
```

```
0     12
1     12
2      0
3      6
4      6
5      9
6      9
7      0
8      0
9      0
10     0
11     0
12     0
13    13
14     0
15     0
dtype: int64
```

5. Handling the missing values (part iii)

```
# Iterate over each column of cc_apps
for col in cc_apps.columns:
    # Check if the column is of object type
    if cc_apps[col].dtypes == 'object':
        # Impute with the most frequent value
        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])

# Count the number of NaNs in the dataset and print the counts to
verify
print(cc_apps.isnull().sum())
```

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
dtype: int64
```

6. Preprocessing the data (part i)

```
# Import LabelEncoder
from sklearn.preprocessing import LabelEncoder

# Instantiate LabelEncoder
le=LabelEncoder()

# Iterate over all the values of each column and extract their dtypes
for col in cc_apps.columns:
    # Compare if the dtype is object
    if cc_apps[col].dtypes=='object':
    # Use LabelEncoder to do the numeric transformation

cc_apps[col]=le.fit(cc_apps[col].values).transform(cc_apps[col].values
)
```

7. Splitting the dataset into train and test sets

```
# Import train_test_split
from sklearn.model_selection import train_test_split
```

```
# Drop the features 11 and 13 and convert the DataFrame to a NumPy
array
cc_apps = cc_apps.drop([11, 13], axis=1)
cc_apps = cc_apps.values

# Segregate features and labels into separate variables
X,y = cc_apps[:,0:13] , cc_apps[:,13]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                test_size=0.33,
                                random_state=42)
```

8. Preprocessing the data (part ii)

```
# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

# Instantiate MinMaxScaler and use it to rescale X_train and X_test
scaler = MinMaxScaler(feature_range=(0,1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.fit_transform(X_test)
```

9. Fitting a logistic regression model to the train set

```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate a LogisticRegression classifier with default parameter
values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train, y_train)

LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear',
tol=0.0001,
          verbose=0, warm_start=False)
```

10. Making predictions and evaluating performance

```
# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Use logreg to predict instances from the test set and store it
y_pred = logreg.predict(rescaledX_test)
```

```python
# Get the accuracy score of logreg model and print it
print("Accuracy of logistic regression classifier: ",
logreg.score(rescaledX_test, y_test))

# Print the confusion matrix of the logreg model
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy of logistic regression classifier:  0.8377192982456141
[[92 11]
 [26 99]]
```

11. Grid searching and making the model perform better

```python
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Define the grid of values for tol and max_iter
tol = [0.01, 0.001, 0.0001]
max_iter = [100, 150, 200]

# Create a dictionary where tol and max_iter are keys and the lists of
their values are corresponding values
param_grid = dict(tol=tol, max_iter=max_iter)
```

12. Finding the best performing model

```python
# Instantiate GridSearchCV with the required parameters
grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid,
cv=5)

# Use scaler to rescale X and assign it to rescaledX
rescaledX = scaler.fit_transform(X)

# Fit data to grid_model
grid_model_result = grid_model.fit(rescaledX, y)

# Summarize results
best_score, best_params = grid_model_result.best_score_,
grid_model_result.best_params_
print("Best: %f using %s" % (best_score, best_params))
```

```
Best: 0.853623 using {'tol': 0.01, 'max_iter': 100}
```