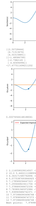
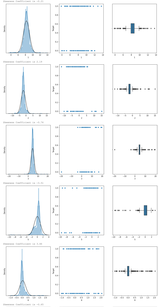
part1

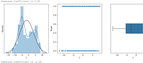
```
Expected Cost:0.0
```

```
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.pyplot plotting
import seaborn as sns # statistical data visualization
%matplotlib inline # show plots in notebook
from sklearn.model_selection import train_test_split # splitting the dataset between features
from sklearn.preprocessing import StandardScaler # standardization of features
from sklearn.metrics import confusion_matrix # to plot confusion matrix
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout

def plot_hist(hist, yscale='linear'):
    '''Utility function to plot the history of a keras model
    yscale can be 'linear' or 'log'.
    '''
    pd.DataFrame(hist.history).plot(figsize=(8,5))
    plt.gca().set_ylim(1,0)
    plt.gca().set_yscale(yscale)
    plt.gca().set_xlabel('epoch')
    return plt.gca().set_ylabel('')

# Parameters
n_features = 30 # number of features
n_dense = 10 # number of hidden units
model = Sequential()
model.add(Dense(n_dense, activation='relu', # activation function: rectified linear unit
                input_shape=(n_features,)))
model.add(Dense(1, activation='sigmoid', # activation function: sigmoid
                kernel_initializer='uniform', # random initialization of weights
                bias_initializer='zeros', # initialize biases to zero
                kernel_regularizer=keras.regulizers.l2(0.01), # L2 regularization
                bias_regularizer=keras.regulizers.l2(0.01)))
model.compile(optimizer='adam', # optimizer: Adam
              loss='binary_crossentropy', # loss function: binary cross-entropy
              metrics=['accuracy', # report accuracy
                       'precision', # report precision
                       'recall', # report recall
                       'fbeta_score', # report F-beta score
                       'binary_crossentropy', # report binary cross-entropy loss
                       'hinge', # report hinge loss
                       'mean_squared_error', # report mean squared error
                       'mean_absolute_error', # report mean absolute error
                       'cosine_proximity', # report cosine proximity
                       'mean_squared_logarithmic_error', # report mean squared logarithmic error
                       'mean_absolute_percentage_error', # report mean absolute percentage error
                       'mean_squared_logarithmic_error', # report mean squared logarithmic error
                       'mean_absolute_percentage_error', # report mean absolute percentage error
```



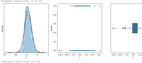
```
1.1.1.0.0.0
```


DIAGNOSTIC CHARTS FOR CATEGORICAL VARIABLES




```
ggsave("Rplots.pdf", plot = ggarrange(plotlist = ggplot_objects(plot), ncol = 3), width = 9, height = 12)
pdf_path = "Rplots.pdf"
pdf_base64 = base64encode(pdf_path)
print(paste0("data:application/pdf; base64,", pdf_base64))
```

```
[1] "data:application/pdf; base64,
```



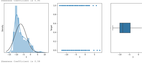

```
rf_model = ranger(Heart.disease ~ ., data = Heart_Disease_train, num.trees = 500, importance = "permutation")
print(rf_model)
```

```
Ranger result

Call:
 ranger(Heart.disease ~ ., data = Heart_Disease_train, num.trees = 500,
 importance = "permutation")

Type:                             Classification
Number of trees:                  500
Sample size:                      270
Number of independent variables:  13
Mahalanobis distance:             FALSE
Variable importance mode:         permutation
Splitting rule:                   gini
OOB prediction error:             19.26 %
```

```
# Calculating feature importance
importance_values = rf_model$variable.importance
importance_df = data.frac(Feature = names(importance_values), Importance = importance_values)
importance_df = importance_df[order(-importance_df$Importance), ]
print(importance_df)
```

```
                   Feature Importance
Thal               Thal 0.   0.1 0.   0.1 0.
Chest pain type    Chest pain type   0.1 0.
Oldpeak            Oldpeak   0.1 0.   0.1 0.
Slope              Slope   0.1 0.   0.1 0.
Number of major ves Number of major ves   0.1 0.
Maximum heart rate ach Maximum heart rate ach   0.1 0.
Age                Age   0.1 0.   0.1 0.
Serum cholesterol  Serum cholesterol   0.1 0.
Resting blood pressure Resting blood pressure   0.1 0.
Exercise induced angina Exercise induced angina   0.1 0.
Sex                Sex   0.1 0.   0.1 0.
Resting electrocardiog Resting electrocardiog   0.1 0.
Fasting blood sugar Fasting blood sugar   0.1 0.
```

```
# Plot feature importance
ggplot(importance_df, aes(x = Feature, y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  theme_minimal() +
  ggtitle("Feature Importance")
ggsave("Rplots.pdf", plot = ggplot_last(), width = 9, height = 12)
pdf_path = "Rplots.pdf"
pdf_base64 = base64encode(pdf_path)
print(paste0("data:application/pdf; base64,", pdf_base64))
```



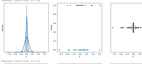

```
# Predicting probabilities
rf_pred = predict(rf_model, data = Heart_Disease_test)$predictions
print(head(rf_pred))
```

```
[1] 1 1 1 1 1 1
Levels: 0 1
```

```
# Calculating ROC curve
roc_obj = roc(Heart_Disease_test$Heart.disease, rf_pred)
print(roc_obj)
```

```
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Warning message:
In roc.default(Heart_Disease_test$Heart.disease, rf_pred) :
  Ignoring extra argument(s) 'ncomponents, ncomponents, dimension'

Call:
roc.default(response = Heart_Disease_test$Heart.disease, predictor = rf_pred)

Data: rf_pred in 54 controls (0 0) 54 cases (1 1).
Multivariate AUC = 1.000
```

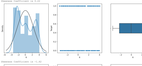
```
# Plot ROC curve
plot(roc_obj, main = "ROC Curve", col = "blue", lwd = 2)
ggsave("Rplots.pdf", plot = ggplot_last(), width = 9, height = 12)
pdf_path = "Rplots.pdf"
pdf_base64 = base64encode(pdf_path)
print(paste0("data:application/pdf; base64,", pdf_base64))
```




```
# Calculating learning curve
library(caret)
learning_curve = learning.curve(rf_model, Heart_Disease_train, method = "rf")
print(learning_curve)
```

```
Error: unexpected '>' in "learning_curve = learning.curve(rf_model, Heart_Disease_train, method = "rf")
>"
```

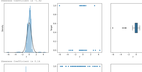
```
# Plot learning curve
ggplot(learning_curve, aes(x = Training.Size, y = Accuracy, color = "Training")) +
  geom_line() +
  geom_ribbon(aes(ymin = Accuracy - SD, ymax = Accuracy + SD, fill = "Training"), alpha = 0.2) +
  geom_line(aes(y = Cross.Validation.Accuracy, color = "Cross-Validation")) +
  geom_ribbon(aes(ymin = Cross.Validation.Accuracy - Cross.Validation.SD, ymax = Cross.Validation.Accuracy + Cross.Validation.SD, fill = "Cross-Validation"), alpha = 0.2) +
  scale_color_manual(values = c("Training" = "blue", "Cross-Validation" = "red")) +
  scale_fill_manual(values = c("Training" = "blue", "Cross-Validation" = "red")) +
  ggtitle("Learning Curve") +
  xlab("Training Size") +
  ylab("Accuracy") +
  theme_minimal() +
  theme(legend.position = "bottom")
ggsave("Rplots.pdf", plot = ggplot_last(), width = 9, height = 12)
pdf_path = "Rplots.pdf"
pdf_base64 = base64encode(pdf_path)
print(paste0("data:application/pdf; base64,", pdf_base64))
```




```
# Calculating learning curve
library(caret)
learning_curve = learning.curve(rf_model, Heart_Disease_train, method = "rf")
print(learning_curve)
```

```
Error: unexpected '>' in "learning_curve = learning.curve(rf_model, Heart_Disease_train, method = "rf")
>"
```

```
# Plot learning curve
ggplot(learning_curve, aes(x = Training.Size, y = Accuracy, color = "Training")) +
  geom_line() +
  geom_ribbon(aes(ymin = Accuracy - SD, ymax = Accuracy + SD, fill = "Training"), alpha = 0.2) +
  geom_line(aes(y = Cross.Validation.Accuracy, color = "Cross-Validation")) +
  geom_ribbon(aes(ymin = Cross.Validation.Accuracy - Cross.Validation.SD, ymax = Cross.Validation.Accuracy + Cross.Validation.SD, fill = "Cross-Validation"), alpha = 0.2) +
  scale_color_manual(values = c("Training" = "blue", "Cross-Validation" = "red")) +
  scale_fill_manual(values = c("Training" = "blue", "Cross-Validation" = "red")) +
  ggtitle("Learning Curve") +
  xlab("Training Size") +
  ylab("Accuracy") +
  theme_minimal() +
  theme(legend.position = "bottom")
ggsave("Rplots.pdf", plot = ggplot_last(), width = 9, height = 12)
pdf_path = "Rplots.pdf"
pdf_base64 = base64encode(pdf_path)
print(paste0("data:application/pdf; base64,", pdf_base64))
```



dataset 3

```
# Load the dataset
Heart_Disease_data = read.csv("/home/kali/Downloads/Heart_disease.csv")
# Convert target variable to factor
Heart_Disease_data$Heart.disease = factor(Heart_Disease_data$Heart.disease)
```

```
Error: cannot open file '/home/kali/Downloads/Heart_disease.csv': No such file or directory
```

```
# Calculating feature scores
library(Boruta)
boruta_output = Boruta(Heart.disease ~ ., data = Heart_Disease_data, doTrace = 2)
print(boruta_output)
```

```
Error: object 'Heart_Disease_data' not found
```

```
# Calculating feature importance
importance_df = data.frame(Feature = rownames(attr(boruta_output, "ImpHistory")),
                           Importance = colMeans(attr(boruta_output, "ImpHistory")))
importance_df = importance_df[order(-importance_df$Importance), ]
print(importance_df)
```

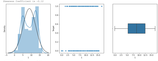
```
Error: object 'boruta_output' not found
```

```
# Plot feature importance
ggplot(importance_df, aligned = TRUE, aes(x = Feature, y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  theme_minimal() +
  ggtitle("Feature Importance")
ggsave("Rplots.pdf", plot = ggplot_last(), width = 9, height = 12)
pdf_path = "Rplots.pdf"
pdf_base64 = base64encode(pdf_path)
print(paste0("data:application/pdf; base64,", pdf_base64))
```

```
Error: unexpected '>' in "ggplot(importance_df, aligned = TRUE, aes(x = Feature, y = Importance)) +
>"
```

```
# Calculating feature importance
importance_values = rf_model$variable.importance
importance_df = data.frac(Feature = names(importance_values), Importance = importance_values)
importance_df = importance_df[order(-importance_df$Importance), ]
print(importance_df)
```

```
Error: object 'rf_model
```


Variable: TwoFriendsAway
Variable: CoupleAtHome
Variable: CoupleAtHome_Lt_10_20
Variable: CoupleAtHome_Lt_20_30
Variable: CoupleAtHome_Lt_30_40
Variable: CoupleAtHome_Lt_40_50
Variable: CoupleAtHome_Lt_50_60
Variable: CoupleAtHome_Lt_60_70
Variable: CoupleAtHome_Lt_70_80
Variable: CoupleAtHome_Lt_80_90
Variable: CoupleAtHome_Lt_90_100
Variable: Fare
Variable: Fare_Lt_0_10
Variable: Fare_Lt_10_20
Variable: Fare_Lt_20_30
Variable: Fare_Lt_30_40
Variable: Fare_Lt_40_50
Variable: Fare_Lt_50_60
Variable: Fare_Lt_60_70
Variable: Fare_Lt_70_80
Variable: Fare_Lt_80_90
Variable: Fare_Lt_90_100
Variable: Fare_Lt_100_110
Variable: Fare_Lt_110_120
Variable: Fare_Lt_120_130
Variable: Fare_Lt_130_140
Variable: Fare_Lt_140_150
Variable: Fare_Lt_150_160
Variable: Fare_Lt_160_170
Variable: Fare_Lt_170_180
Variable: Fare_Lt_180_190
Variable: Fare_Lt_190_200
Variable:

Original column values of Amount

```
                 0         1         2         3         4         5         6         7         8         9        10        11
count  1.000000e+03  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
mean   1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
std    0.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
min    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
25%    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
50%    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
75%    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
max    1.000000e+00  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000

                 13        14        15        16        17        18        19        20        21        22        23        24
count  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
mean   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
std    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
min    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
25%    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
50%    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
75%    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
max    1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
```

Shape = (1000,25)

Pre-Processing

```
numerical_data.hist(figsize=(4,4),edgecolor='')  # diagnostic visual testing data data
```

```
sns.set(rc={'figure.figsize':(4,4)})
for i in numerical_data.columns:
    sns.distplot(numerical_data[i],kde=False,bins=20,color='b').set(title='Original')
    plt.show()
```

```
data_cleaned = data.dropna()  # checking for null values
```

```
isnull
False    1000
dtype: int64
False    1000
dtype: int64
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
False    1000
dtype: int64
```

```
# Data Normalization - # standard deviation of dataset
```

```
                 0         1         2         3         4         5         6         7         8         9        10        11
count  1.000000e+03  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
mean   1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
std    0.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
min    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
25%    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
50%    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
75%    1.000000e+00  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
max    1.000000e+00  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000

                 13        14        15        16        17        18        19        20        21        22        23        24
count  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
mean   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
std    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
min    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
25%    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
50%    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
75%    0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
max    1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
```

```
sns.pairplot(numerical_data,kind='scatter')  # visualizing the data distribution
```

```
# Data Scaling
numerical_data = data.select_dtypes(exclude=['object'])
numerical_data.shape,numerical_data.dtypes
numerical_data.describe()
numerical_data.isnull().sum()
numerical_data.dropna()
numerical_data.cov()
numerical_data.corr()
sns.pairplot(numerical_data)
```