ДОДАТОК А

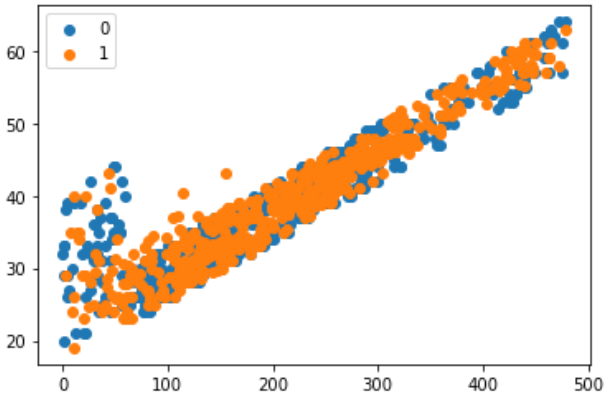
Опис змінних датасету Kaggle

Номер	Назва змінної	Опис змінної
Інформація про клієнта:		
1	months as customer	кількість місяців у якості клієнта
2	age	вік
3	insured zip	поштовий індекс
4	insured sex	стать
5	insured education level	рівень освіти
6	insured occupation	вид зайнятості
7	insured hobbies	хобі
8	insured relationship	сімейний стан
9	capital-gains	приріст капіталу
10	capital loss	втрата капіталу
Характеристика інциденту:		
11	incident date	дата інциденту
12	incident type	тип інциденту
13	collision type	тип зіткнення
14	incident severity	важкість інциденту
15	authorities contacted	звернення до органів влади
16	incident state	штат, у якому стався інцидент
17	incident city	місто інциденту
18	incident location	місце інциденту
19	incident hour of the day	година доби, у якій стався інцидент
20	number of vehicles involved	кількість задіяних транспортних засобів
21	property damage	пошкодження майна
22	bodily injuries	тілесні ушкодження
23	witnesses	свідки
24	police report available	наявність поліцейського звіту
25	total claim amount	загальна сума претензії
26	injury claim	позов про відшкодування шкоди (при травмуванні)
27	property claim	майновий позов
28	vehicle claim	претензія на транспортний засіб
29	auto make	марка автомобіля
30	auto model	модель автомобіля
31	auto year	рік автомобіля
32	fraud reported	повідомлення про шахрайство
Дані страхового полісу:		
33	policy number	номер поліса
34	policy bind date	дата прив’язки поліса
35	policy state	штат поліса
36	policy csl	єдиний комбінований ліміт поліса
37	policy deductible	франшиза поліса
38	policy annual premium	річна премія за полісом
39	umbrella limit	межа покриття.

ДОДАТОК Б

Фрагмент виконання коду з моделювання класифікаторів на даних вибірки 2 SMOTE

```
#pip install -U imbalanced-learn
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='minority',random_state=10)
X_sm, y_sm = smote.fit_resample(X, y)
from collections import Counter
counter = Counter(y_sm)
print(counter)
Counter({0: 525, 1: 525})
# scatter plot of examples by class label
from numpy import where
from matplotlib import pyplot
from sklearn.datasets import make_classification
for label, _ in counter.items():
 row_ix = where(y_sm == label)[0]
 pyplot.scatter(X_sm[row_ix, 0], X_sm[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```




```
X_train = X_sm
X_test = test.iloc[:, :-1].values
y_train = y_sm
y_test = test.iloc[:, -1].values
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler().fit(X_train)
X_train = sc_X.transform(X_train)
X_test = sc_X.transform(X_test)
pd.Series(y_train).value_counts()
```

Out[26]:

```
1    525
0    525
dtype: int64

pd.Series(y_test).value_counts()
```

Out[27]:

```
0    228
1     72
dtype: int64
```

```
#Logistic Regression
#Логістична регресія на усіх змінних
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 13).fit(X_train, y_train)
# Predicting the Test set results
y_pred = lr.predict(X_test)
lr.score(X_test,y_test)
```

Out[35]:

```
0.8666666666666667
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
[[195  33]
 [  7  65]]
Error_rate = (7+33)/(300)
Error_rate
```

Out[37]:

```
0.13333333333333333
```

```
Se = 65/(65+7)
Se
```

Out[38]:

```
0.9027777777777778
```

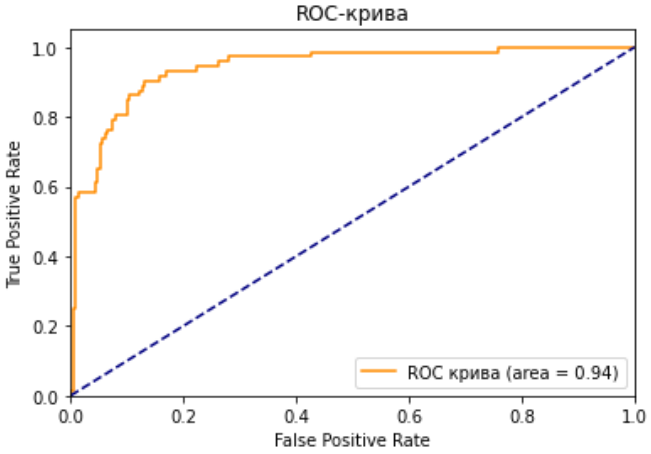
```
Sp = 195/(195+33)
Sp
```

Out[39]:

```
0.8552631578947368
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot as plt
lr_probs = lr.predict_proba(X_test)
lr_probs = lr_probs[:, 1]
lr_auc = roc_auc_score(y_test, lr_probs)
print('LogisticRegression: ROC AUC=%.3f' % (lr_auc))
fpr, tpr, treshold = roc_curve(y_test, lr_probs)
roc_auc = auc(fpr, tpr)
```

```
plt.plot(fpr, tpr, color='darkorange',
         label='ROC крива (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title( 'ROC-крива')
plt.legend(loc="lower right")
plt.show()
LogisticRegression: ROC AUC=0.941
```




```
#Logistic Regression
#Логістична регресія на відібраних змінних
# Baseline model
import statsmodels.api as sm
lr = sm.Logit(y_train, X_train).fit()
print(lr.summary2())

Optimization terminated successfully.
         Current function value: 0.355883
         Iterations 12
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.487
Dependent Variable: y                AIC:              813.3550
Date:               2021-12-11 19:37 BIC:              976.9210
No. Observations:   1050             Log-Likelihood:   -373.68
Df Model:           32               LL-Null:          -727.80
Df Residuals:       1017             LLR p-value:      2.2090e-128
Converged:          1.0000           Scale:            1.0000
No. Iterations:     12.0000
-----------------------------------------------------------------
        Coef.     Std.Err.       z       P>|z|      [0.025     0.975]
-----------------------------------------------------------------
x1       0.2924     0.2642    1.1066    0.2685     -0.2255     0.8102
x2      -0.3254     0.2646   -1.2297    0.2188     -0.8440     0.1932
x3       0.0404     0.0987    0.4087    0.6828     -0.1532     0.2339
x4       0.2440     0.1029    2.3703    0.0178      0.0422     0.4458
```

```
x5      -0.0180     0.0995   -0.1811   0.8563     -0.2130     0.1770
x6      -0.0935     0.1003   -0.9320   0.3513     -0.2902     0.1031
x7       0.0195     0.1028    0.1896   0.8496     -0.1820     0.2210
x8       0.0999     0.0980    1.0194   0.3080     -0.0922     0.2920
x9      -0.0250     0.0990   -0.2525   0.8006     -0.2190     0.1690
x10      0.3319     0.1022    3.2464   0.0012      0.1315     0.5322
x11      1.3711     0.1201   11.4196   0.0000      1.1358     1.6064
x12      0.2961     0.0982    3.0149   0.0026      0.1036     0.4886
x13     -0.1166     0.0985   -1.1841   0.2364     -0.3096     0.0764
x14     -0.0825     0.0975   -0.8455   0.3978     -0.2737     0.1087
x15     -0.7073     0.3635   -1.9458   0.0517     -1.4197     0.0051
x16      0.7612     0.3178    2.3951   0.0166      0.1383     1.3842
x17      1.5713     0.1101   14.2769   0.0000      1.3556     1.7870
x18      0.2343     0.1473    1.5906   0.1117     -0.0544     0.5231
x19      0.2240     0.1033    2.1678   0.0302      0.0215     0.4266
x20      0.1404     0.0973    1.4428   0.1491     -0.0503     0.3312
x21     -0.0918     0.1014   -0.9053   0.3653     -0.2906     0.1070
x22     -0.2603     0.1121   -2.3213   0.0203     -0.4800    -0.0405
x23     -0.0074     0.0981   -0.0753   0.9399     -0.1997     0.1849
x24      0.0551     0.0970    0.5682   0.5699     -0.1350     0.2452
x25      0.2253     0.0989    2.2776   0.0227      0.0314     0.4192
x26      0.0239     0.0985    0.2421   0.8087     -0.1693     0.2170
x27    -56.9357   278.6463   -0.2043   0.8381   -603.0724   489.2010
x28     10.1075    50.7022    0.1993   0.8420    -89.2670   109.4819
x29     11.0809    53.4107    0.2075   0.8356    -93.6021   115.7639
x30     40.7575   199.4715    0.2043   0.8381   -350.1996   431.7145
x31     -0.0829     0.1162   -0.7131   0.4758     -0.3107     0.1449
x32      0.4252     0.1226    3.4691   0.0005      0.1850     0.6655
x33     -0.1632     0.1007   -1.6203   0.1052     -0.3605     0.0342
==================================================================
```

```
# Automatic Feature Selection
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
estimator=LogisticRegression(random_state=123)
selector=RFECV(estimator, step=1)
selector=selector.fit(X_train, y_train)
print(selector.n_features_)
print(selector.support_)
print(selector.ranking_)
20
[ True True False True False False False True False True True True
True False True True True True True True False True False False True
False False True True False False True True]
[ 1  1  7  1 11  4 10  1  9  1  1  1  1  3  1  1  1  1  1  1  2  1
13  6  1  8 12  1  1 14  5  1  1]
# Selected Features
selected_columns=[]
for i in range(len(X_train[0])):
    if selector.support_[i]==1:
```

```
        selected_columns.append(i)
print(selected_columns)
print('№\tIndex\tFeature')
i=0
for column in selected_columns:
    print(str(i)+'\t'+str(column)+'\t'+str(train.columns[column]))
    i=i+1
[0, 1, 3, 7, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 21, 24, 27, 28,
31, 32]
№      Index  Feature
0      0      months_as_customer
1      1      age
2      3      policy_csl
3      7      insured_sex
4      9      insured_occupation
5      10     insured_hobbies
6      11     insured_relationship
7      12     capital-gains
8      14     incident_type
9      15     collision_type
10     16     incident_severity
11     17     authorities_contacted
12     18     incident_state
13     19     incident_city
14     21     number_of_vehicles_involved
15     24     witnesses
16     27     injury_claim
17     28     property_claim
18     31     auto_model
19     32     auto_year

# p-value < 3% Features
bas_X_train = X_train[:,[3,9,10,11,15,16,18,21,24,31]]
bas_X_test = X_test[:,[3,9,10,11,15,16,18,21,24,31]]
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
bas = LogisticRegression(random_state = 13).fit(bas_X_train,y_train)
# Predicting the Test set results
y_bas_pred = bas.predict(bas_X_test)
bas.score(bas_X_test,y_test)
                                                          Out[46]:
0.88

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_bas_pred)
print(cm)
[[197  31]
 [  5  67]]
Error_rate = (5+31)/(300)
```

```
Error_rate
```

Out[48]:

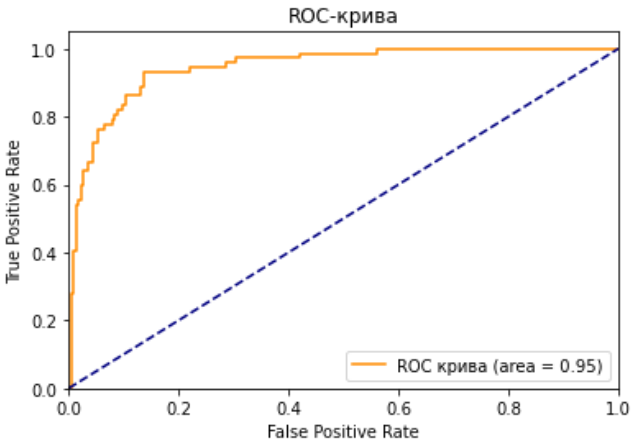
```
0.12

Se = 67/(67+5)
Se
```

Out[50]:

```
0.9305555555555556

Sp = 197/(197+31)
Sp
```

Out[51]:

```
0.8640350877192983

bas_probs = bas.predict_proba(bas_X_test)
bas_probs = bas_probs[:, 1]
bas_auc = roc_auc_score(y_test, bas_probs)
print('LogisticRegression: ROC AUC=%.3f' % (bas_auc))
fpr, tpr, treshold = roc_curve(y_test, bas_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC крива (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
LogisticRegression: ROC AUC=0.945
```




```
#SVM
#SVM на значущих змінних
# Fitting SVM to the Training set (2 variables)
from sklearn.svm import SVC
```

```
svm = SVC(kernel = 'linear', random_state = 10,
probability=True).fit(bas_X_train, y_train)
# Predicting the Test set results
svm_pred = svm.predict(bas_X_test)
svm.score(bas_X_test,y_test)
```

Out[63]:

```
0.8666666666666667

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, svm_pred)
print(cm)
[[192  36]
 [  4  68]]
Error_rate = (4+36)/(300)
Error_rate
```

Out[65]:

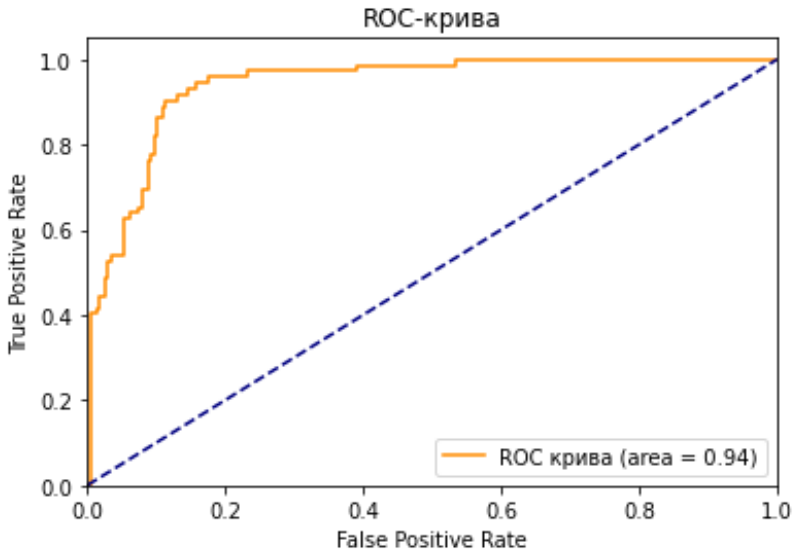
```
0.13333333333333333

Se = 68/(68+4)
Se
```

Out[66]:

```
0.9444444444444444

Sp = 192/(192+36)
Sp
```

Out[68]:

```
0.8421052631578947

svm_probs = svm.predict_proba(bas_X_test)
# сохраняем вероятности только для положительного исхода
svm_probs = svm_probs[:, 1]
# рассчитываем ROC AUC
svm_auc = roc_auc_score(y_test, svm_probs)
print('SVM: ROC AUC=%.3f' % (svm_auc))
# рассчитываем roc-кривую
fpr, tpr, treshold = roc_curve(y_test, svm_probs)
roc_auc = auc(fpr, tpr)
# строим график
plt.plot(fpr, tpr, color='darkorange',
         label='ROC крива (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
SVM: ROC AUC=0.940
```




```
#SVM на усіх змінних
# Fitting SVM to the Training set (2 variables)
from sklearn.svm import SVC
svm1 = SVC(kernel = 'linear', random_state = 10,
probability=True).fit(X_train, y_train)
# Predicting the Test set results
svm_pred = svm1.predict(X_test)
svm1.score(X_test,y_test)
```

Out[71]:

```
0.8633333333333333

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, svm_pred)
print(cm)
[[192  36]
 [  5  67]]

Error_rate = (5+36)/(300)
Error_rate
```

Out[74]:

```
0.13666666666666666

Se = 67/(67+5)
Se
```

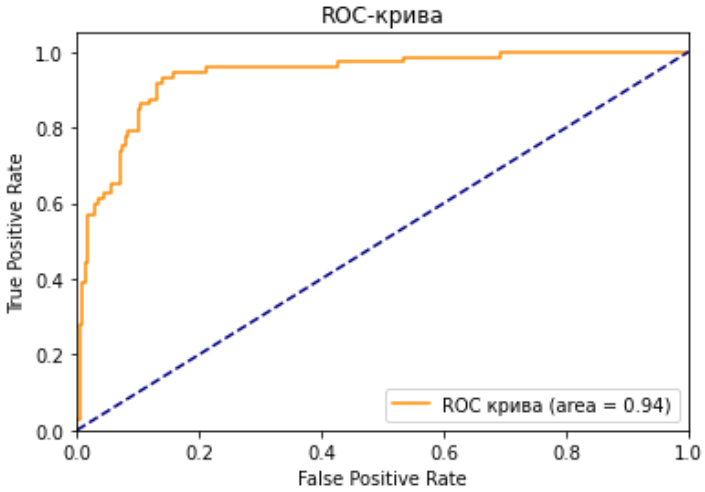
Out[75]:

```
0.9305555555555556

Sp = 192/(192+36)
Sp
```

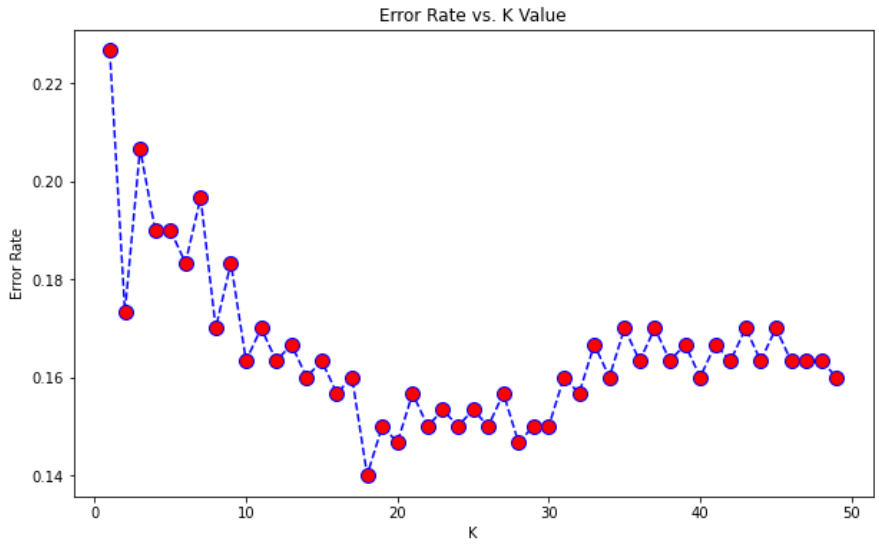
Out[76]:

```
0.8421052631578947

svm1_probs = svm1.predict_proba(X_test)
# сохраняем вероятности только для положительного исхода
```

```
svm1_probs = svm1_probs[:, 1]
# рассчитываем ROC AUC
svm1_auc = roc_auc_score(y_test, svm1_probs)
print('SVM: ROC AUC=%.3f' % (svm1_auc))
fpr, tpr, treshold = roc_curve(y_test, svm1_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='ROC крива (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
SVM: ROC AUC=0.937
```




```
#K-Nearest Neighbors
#KNN на значущих змінних
# Optimal K
from sklearn.neighbors import KNeighborsClassifier
error_rate = []
for i in range(1,50):
     knn = KNeighborsClassifier(n_neighbors=i)
     knn.fit(bas_X_train,y_train)
     pred_i = knn.predict(bas_X_test)
     error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,50),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K
=",error_rate.index(min(error_rate)))
Minimum error:- 0.14 at K = 17
```




```
# Fitting K-NN to the Training set (2 variables)
knn = KNeighborsClassifier(n_neighbors = 17, metric = 'minkowski', p
= 2).fit(bas_X_train, y_train)
# Predicting the Test set results
knn_pred = knn.predict(bas_X_test)
knn.score(bas_X_test,y_test).round(3)
```

Out[82]:

```
0.84
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, knn_pred)
print(cm)
[[185  43]
 [  5  67]]

Error_rate = (5+43)/(300)
Error_rate
```

Out[84]:

```
0.16
```

```
Se = 67/(67+5)
Se
```

Out[85]:

```
0.9305555555555556
```

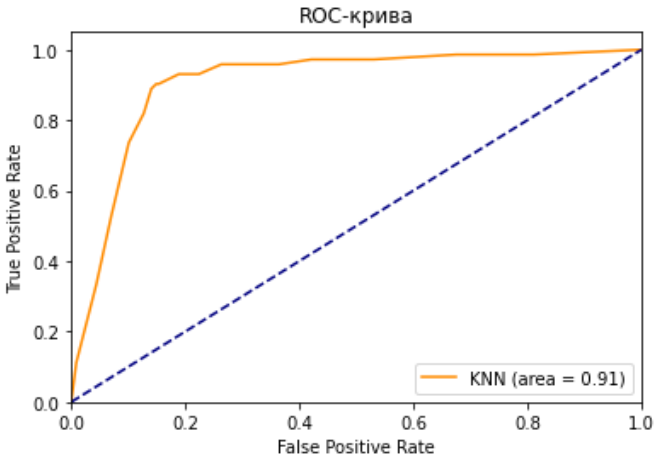
```
Sp = 185/(185+43)
Sp
```

Out[86]:

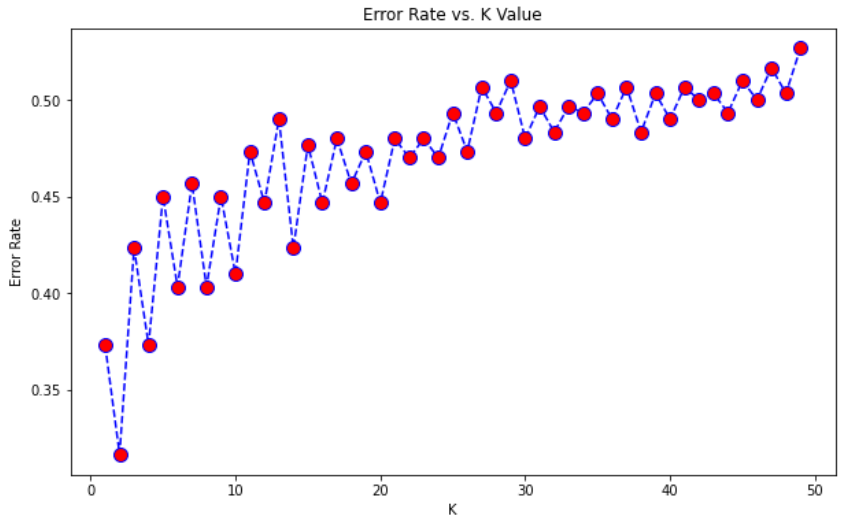
```
0.8114035087719298
```

```
knn_probs = knn.predict_proba(bas_X_test)
knn_probs = knn_probs[:, 1]
knn_auc = roc_auc_score(y_test, knn_probs)
print('KNN: ROC AUC=%.3f' % (knn_auc))
fpr, tpr, treshold = roc_curve(y_test, knn_probs)
```

```
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='KNN (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
KNN: ROC AUC=0.906
```




```
#KNN на усіх змінних
# Optimal K
from sklearn.neighbors import KNeighborsClassifier
error_rate = []
for i in range(1,50):
     knn1 = KNeighborsClassifier(n_neighbors=i)
     knn1.fit(X_train,y_train)
     pred_i = knn1.predict(X_test)
     error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,50),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K
=",error_rate.index(min(error_rate)))
Minimum error:- 0.31666666666666665 at K = 1
```




```
# Fitting K-NN to the Training set (2 variables)
knn = KNeighborsClassifier(n_neighbors = 1, metric = 'minkowski', p
= 2).fit(X_train, y_train)
# Predicting the Test set results
knn_pred = knn.predict(X_test)
knn.score(X_test,y_test).round(2)
```

Out[96]:

```
0.63
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, knn_pred)
print(cm)
[[137  91]
 [ 21  51]]

Error_rate = (21+91)/(300)
Error_rate
```

Out[99]:

```
0.37333333333333335

Se = 51/(51+21)
Se
```

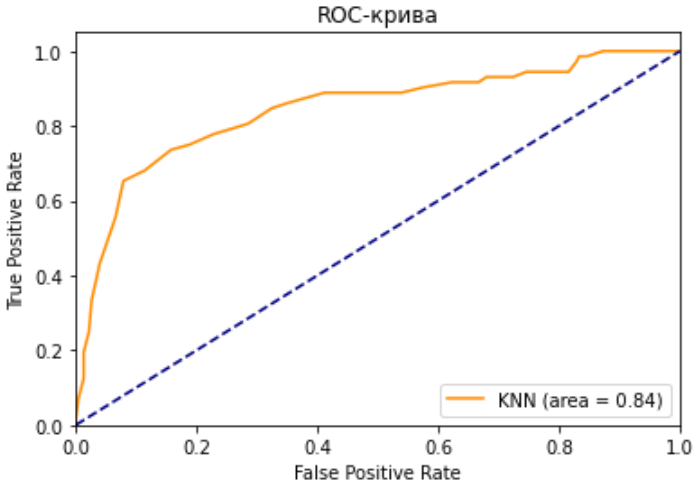
Out[100]:

```
0.7083333333333334

Sp = 137/(137+91)
Sp
```

Out[101]:

```
0.6008771929824561

knn1_probs = knn1.predict_proba(X_test)
knn1_probs = knn1_probs[:, 1]
knn1_auc = roc_auc_score(y_test, knn1_probs)
print('KNN: ROC AUC=%.3f' % (knn1_auc))
fpr, tpr, treshold = roc_curve(y_test, knn1_probs)
```

```
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='KNN (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
KNN: ROC AUC=0.843
```




```
#Naive Bayes
#Naive Bayes на значущих змінних
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB().fit(bas_X_train, y_train)
# Predicting the Test set results
nb_pred = nb.predict(bas_X_test)
nb.score(bas_X_test,y_test)
```

Out[104]:

```
0.86

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, nb_pred)
print(cm)
[[192  36]
 [  6  66]]

Error_rate = (6+36)/(300)
Error_rate
```

Out[106]:

```
0.14

Se = 66/(66+6)
Se
```

Out[107]:

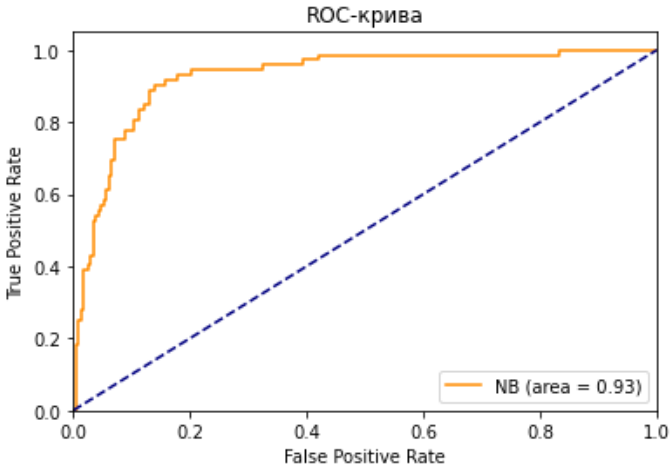
```
0.9166666666666666
```

```
Sp = 192/(192+36)
Sp
```

Out[108]:

```
0.8421052631578947
```

```
nb_probs = nb.predict_proba(bas_X_test)
nb_probs = nb_probs[:, 1]
nb_auc = roc_auc_score(y_test, nb_probs)
print('NB: ROC AUC=%.3f' % (nb_auc))
fpr, tpr, treshold = roc_curve(y_test, nb_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='NB (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
NB: ROC AUC=0.926
```




```
#Naive Bayes на усіх змінних
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
nb1 = GaussianNB().fit(X_train, y_train)
# Predicting the Test set results
nb1_pred = nb1.predict(X_test)
nb1.score(X_test,y_test)
```

Out[114]:

```
0.6466666666666666
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, nb1_pred)
```

```
print(cm)
[[131  97]
 [  9  63]]

Error_rate = (9+97)/(300)
Error_rate
```

Out[116]:

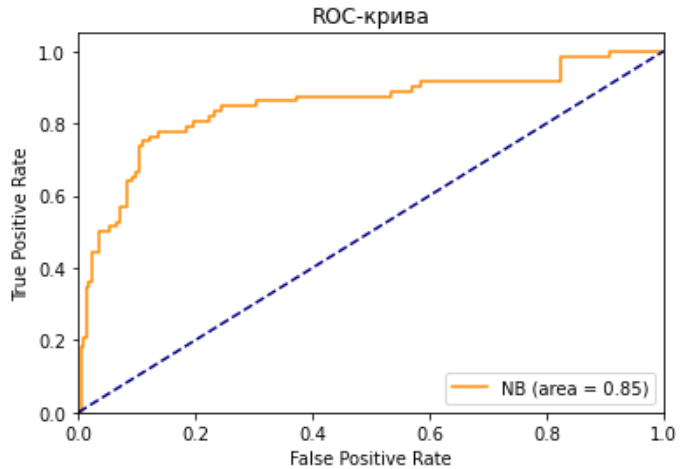
```
0.35333333333333333

Se = 63/(63+9)
Se
```

Out[117]:

```
0.875

Sp = 131/(131+97)
Sp
```

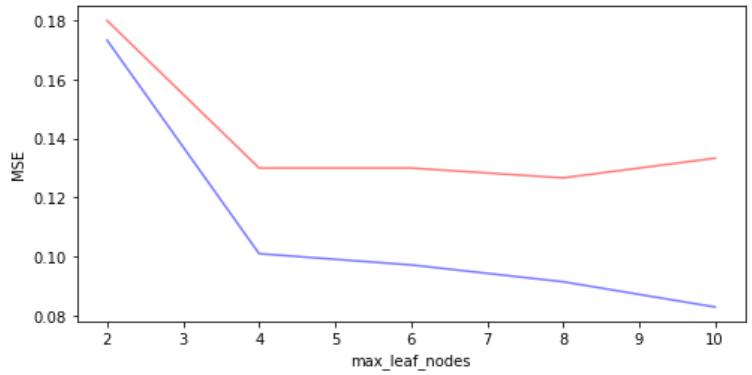
Out[118]:

```
0.5745614035087719

nb1_probs = nb1.predict_proba(X_test)
nb1_probs = nb1_probs[:, 1]
nb1_auc = roc_auc_score(y_test, nb1_probs)
print('NB: ROC AUC=%.3f' % (nb1_auc))
fpr, tpr, treshold = roc_curve(y_test, nb1_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='NB (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
NB: ROC AUC=0.850
```




```
#Classification Tree
#Classification Tree на значущих змінних
```

```
# Function Max Leaf
def max_leaf_nodes(bas_X_train, bas_X_test, y_train, y_test, n):
    mse_train = []
    mse_test = []
    for i in n:
        ct = DecisionTreeClassifier(max_leaf_nodes = i,
random_state=10).fit(bas_X_train, y_train)
        mse_train.append(mean_squared_error(y_train,
ct.predict(bas_X_train)))
        mse_test.append(mean_squared_error(y_test,
ct.predict(bas_X_test)))
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(n, mse_train, alpha=0.5, color='blue', label='train')
    ax.plot(n, mse_test, alpha=0.5, color='red', label='test')
    ax.set_ylabel("MSE")
    ax.set_xlabel("max_leaf_nodes")
# The optimal number of max_leaf_nodes
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error
max_leaf_nodes(bas_X_train, bas_X_test, y_train, y_test, [2, 4, 6,
8, 10])
```




```
# Fitting Classification Tree to the Training set (2 variables)
ct = DecisionTreeClassifier(max_leaf_nodes = 4, criterion =
'entropy', random_state = 10).fit(bas_X_train, y_train)
# Predicting the Test set results
ct_pred = ct.predict(bas_X_test)
ct.score(bas_X_test,y_test)
```

Out[138]:

```
0.8666666666666667

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, ct_pred)
print(cm)
[[192  36]
 [  4  68]]

Error_rate = (4+36)/(300)
Error_rate
```

Out[140]:

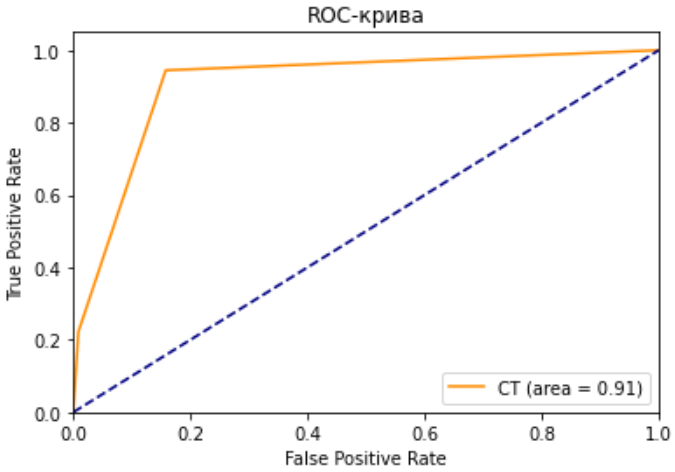
```
0.13333333333333333

Se = 68/(68+4)
Se
```

Out[141]:

```
0.9444444444444444

Sp = 192/(192+36)
Sp
```

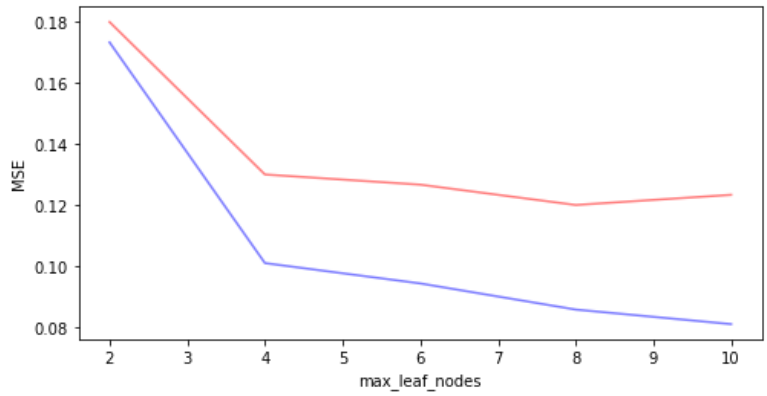
Out[142]:

```
0.8421052631578947

ct_probs = ct.predict_proba(bas_X_test)
ct_probs = ct_probs[:, 1]
ct_auc = roc_auc_score(y_test, ct_probs)
print('CT: ROC AUC=%.3f' % (ct_auc))
fpr, tpr, treshold = roc_curve(y_test, ct_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='CT (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
CT: ROC AUC=0.907
```




```
#Classification Tree на усіх змінних
# Function Max Leaf
def max_leaf_nodes(X_train, X_test, y_train, y_test, n):
    mse_train = []
    mse_test = []
    for i in n:
        ct = DecisionTreeClassifier(max_leaf_nodes = i,
random_state=10).fit(X_train, y_train)
```

```
        mse_train.append(mean_squared_error(y_train,
ct.predict(X_train)))
        mse_test.append(mean_squared_error(y_test,
ct.predict(X_test)))
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(n, mse_train, alpha=0.5, color='blue', label='train')
    ax.plot(n, mse_test, alpha=0.5, color='red', label='test')
    ax.set_ylabel("MSE")
    ax.set_xlabel("max_leaf_nodes")
# The optimal number of max_leaf_nodes
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error
max_leaf_nodes(X_train, X_test, y_train, y_test, [2, 4, 6, 8, 10])
```




```
# Fitting Classification Tree to the Training set (2 variables)
ct1 = DecisionTreeClassifier(max_leaf_nodes = 4, criterion =
'entropy', random_state = 10).fit(X_train, y_train)
# Predicting the Test set results
ct_pred = ct1.predict(X_test)
ct1.score(X_test,y_test)
```

Out[162]:

```
0.8666666666666667

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, ct_pred)
print(cm)
[[192  36]
 [  4  68]]

Error_rate = (36+4)/(300)
Error_rate
```

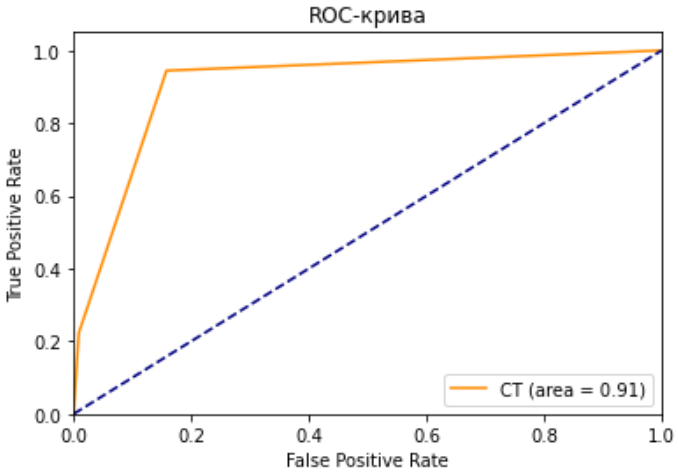
Out[164]:

```
0.13333333333333333

Se = 68/(68+4)
Se
```

Out[166]:

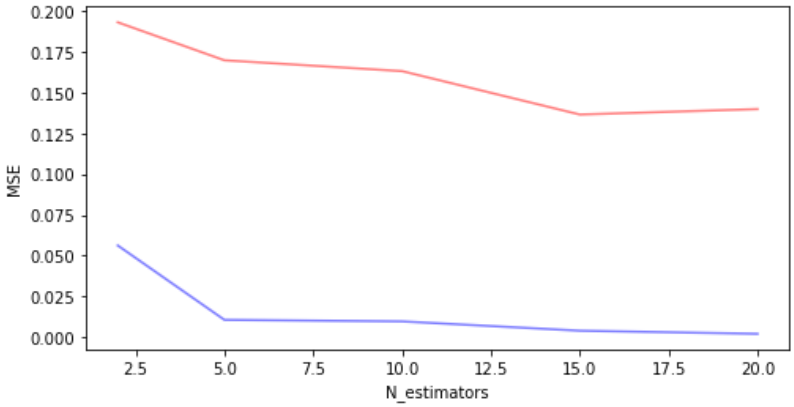
```
0.9444444444444444

Sp = 192/(192+36)
```

```
Sp
                                                                    Out[167]:
0.8421052631578947

ct1_probs = ct1.predict_proba(X_test)
ct1_probs = ct1_probs[:, 1]
ct1_auc = roc_auc_score(y_test, ct1_probs)
print('CT: ROC AUC=%.3f' % (ct1_auc))
fpr, tpr, treshold = roc_curve(y_test, ct1_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='CT (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
CT: ROC AUC=0.907
```

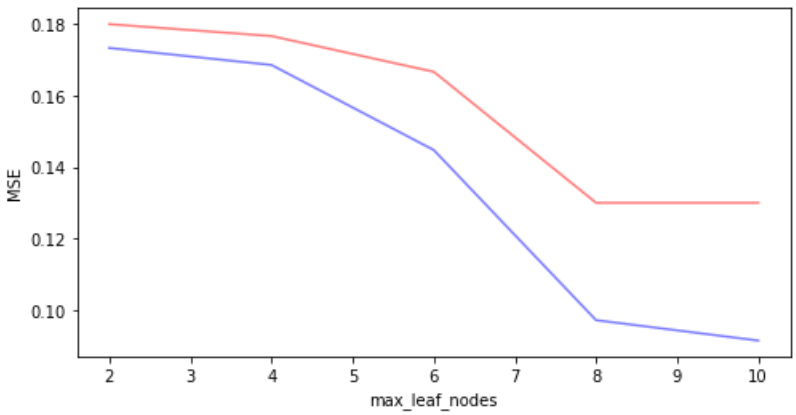



```
#Random Forest
#Random Forest на значущих змінних
def rf_best_n_estimator(bas_X_train, bas_X_test, y_train, y_test,n):
    mse_train = []
    mse_test = []
    for i in n:
        rf = RandomForestClassifier(n_estimators=i,
random_state=10).fit(bas_X_train, y_train)
        mse_train.append(mean_squared_error(y_train,
rf.predict(bas_X_train)))
        mse_test.append(mean_squared_error(y_test,
rf.predict(bas_X_test)))
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(n, mse_train, alpha=0.5, color='blue', label='train')
    ax.plot(n, mse_test, alpha=0.5, color='red', label='test')
    ax.set_ylabel("MSE")
    ax.set_xlabel("N_estimators")
```

```
# The optimal number of trees
from sklearn.ensemble import RandomForestClassifier
rf_best_n_estimator(bas_X_train, bas_X_test, y_train, y_test, [2, 5,
10, 15, 20])
```




```
def rf_max_leaf_nodes(bas_X_train, bas_X_test, y_train, y_test, n):
    mse_train = []
    mse_test = []
    for i in n:
        rf = RandomForestClassifier(n_estimators=15, max_leaf_nodes
=i, random_state=10).fit(bas_X_train, y_train)
        mse_train.append(mean_squared_error(y_train,
rf.predict(bas_X_train)))
        mse_test.append(mean_squared_error(y_test,
rf.predict(bas_X_test)))
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(n, mse_train, alpha=0.5, color='blue', label='train')
    ax.plot(n, mse_test, alpha=0.5, color='red', label='test')
    ax.set_ylabel("MSE")
    ax.set_xlabel("max_leaf_nodes")
# The optimal number of max_leaf_nodes
rf_max_leaf_nodes(bas_X_train,bas_X_test,y_train,y_test,[2,4,6,8,10])
```




```
# Fitting Random Forest to the Training set
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 15, max_leaf_nodes = 8,
random_state = 10).fit(bas_X_train, y_train)
# Predicting the Test set results
rf_pred = rf.predict(bas_X_test)
rf_pred = (rf_pred > 0.5)
```

```
rf.score(bas_X_test,y_test)
```

Out[196]:

```
0.87
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, rf_pred)
print(cm)
[[198  30]
 [  9  63]]

Error_rate = (9+30)/(300)
Error_rate
```

Out[198]:

```
0.13
```

```
Se = 63/(63+9)
Se
```

Out[199]:

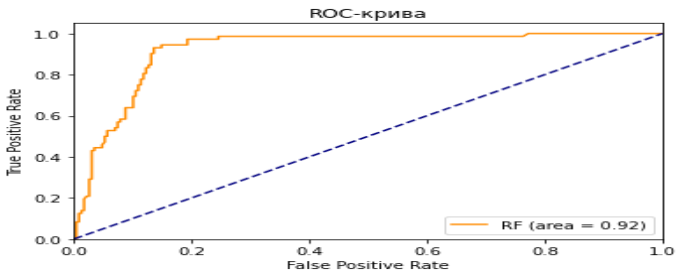
```
0.875
```

```
Sp = 198/(198+30)
Sp
```

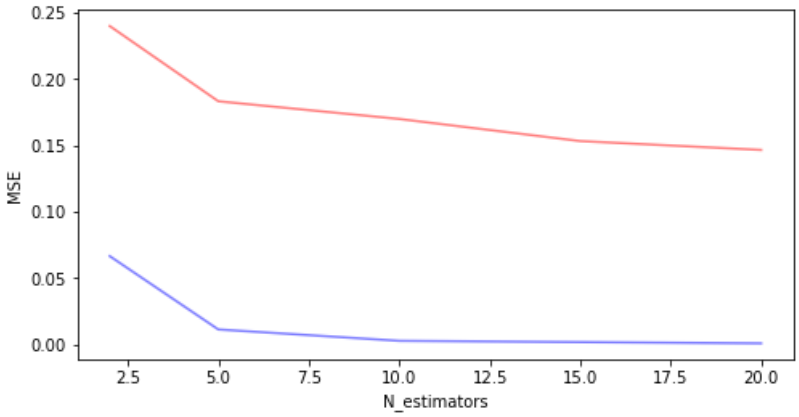
Out[200]:

```
0.868421052631579
```

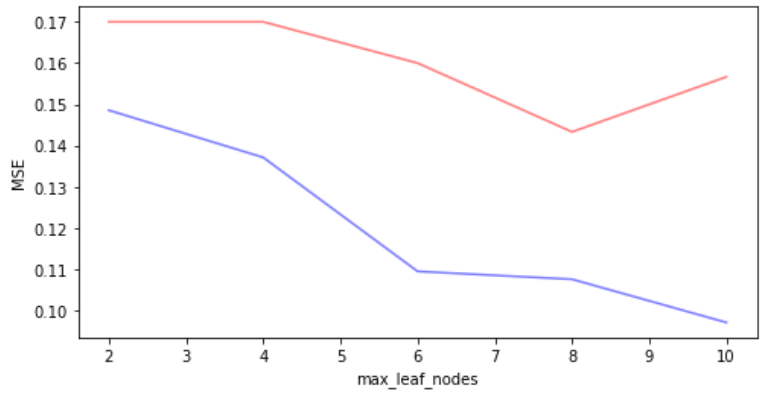
```
rf_probs = rf.predict_proba(bas_X_test)
rf_probs = rf_probs[:, 1]
rf_auc = roc_auc_score(y_test, rf_probs)
print('RF: ROC AUC=%.3f' % (rf_auc))
fpr, tpr, treshold = roc_curve(y_test, rf_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='RF (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
RF: ROC AUC=0.922
```



```
#Random Forest на усіх змінних
def rf_best_n_estimator(X_train, X_test, y_train, y_test, n):
    mse_train = []
    mse_test = []
    for i in n:
        rf = RandomForestClassifier(n_estimators=i,
random_state=10).fit(X_train, y_train)
        mse_train.append(mean_squared_error(y_train,
rf.predict(X_train)))
        mse_test.append(mean_squared_error(y_test,
rf.predict(X_test)))
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(n, mse_train, alpha=0.5, color='blue', label='train')
    ax.plot(n, mse_test, alpha=0.5, color='red', label='test')
    ax.set_ylabel("MSE")
    ax.set_xlabel("N_estimators")
# The optimal number of trees
from sklearn.ensemble import RandomForestClassifier
rf_best_n_estimator(X_train,X_test, y_train, y_test, [2,5,10,15,20])
```




```
def rf_max_leaf_nodes(bas_X_train, bas_X_test, y_train, y_test, n):
    mse_train = []
    mse_test = []
    for i in n:
        rf = RandomForestClassifier(n_estimators=17, max_leaf_nodes
=i, random_state=10).fit(X_train, y_train)
        mse_train.append(mean_squared_error(y_train,
rf.predict(X_train)))
        mse_test.append(mean_squared_error(y_test,
rf.predict(X_test)))
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(n, mse_train, alpha=0.5, color='blue', label='train')
    ax.plot(n, mse_test, alpha=0.5, color='red', label='test')
    ax.set_ylabel("MSE")
    ax.set_xlabel("max_leaf_nodes")
# The optimal number of max_leaf_nodes
rf_max_leaf_nodes(X_train, X_test, y_train, y_test, [2,4,6,8,10])
```




```
# Fitting Random Forest to the Training set
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 15, max_leaf_nodes = 8,
random_state = 10).fit(X_train, y_train)
# Predicting the Test set results
rf_pred = rf.predict(X_test)
rf_pred = (rf_pred > 0.6)
rf.score(X_test,y_test)
```

Out[241]:

```
0.8533333333333334
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, rf_pred)
print(cm)
[[205  23]
 [ 21  51]]

Error_rate = (21+23)/(300)
Error_rate
```

Out[243]:

```
0.14666666666666667

Se = 51/(51+21)
Se
```

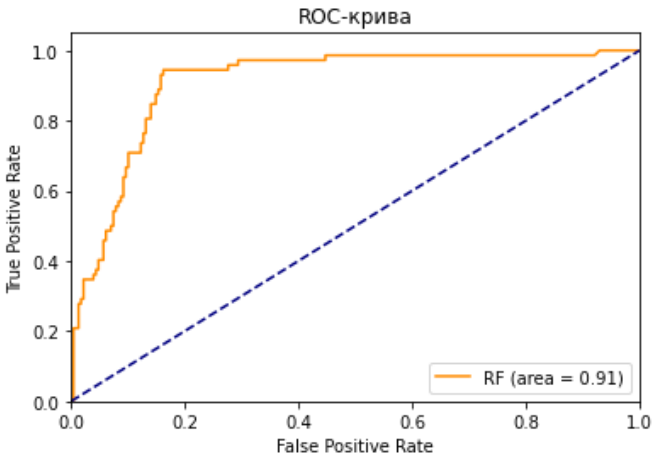
Out[244]:

```
0.7083333333333334

Sp = 205/(205+23)
Sp
```

Out[245]:

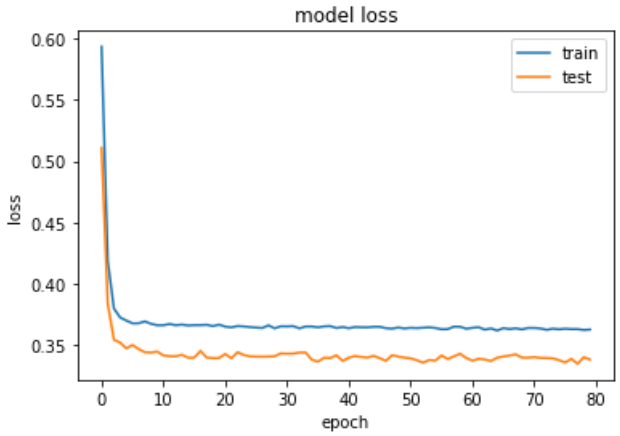
```
0.8991228070175439

rf_probs = rf.predict_proba(X_test)
rf_probs = rf_probs[:, 1]
rf_auc = roc_auc_score(y_test, rf_probs)
print('RF: ROC AUC=%.3f' % (rf_auc))
fpr, tpr, treshold = roc_curve(y_test, rf_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
```

```
         label='RF (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
RF: ROC AUC=0.909
```



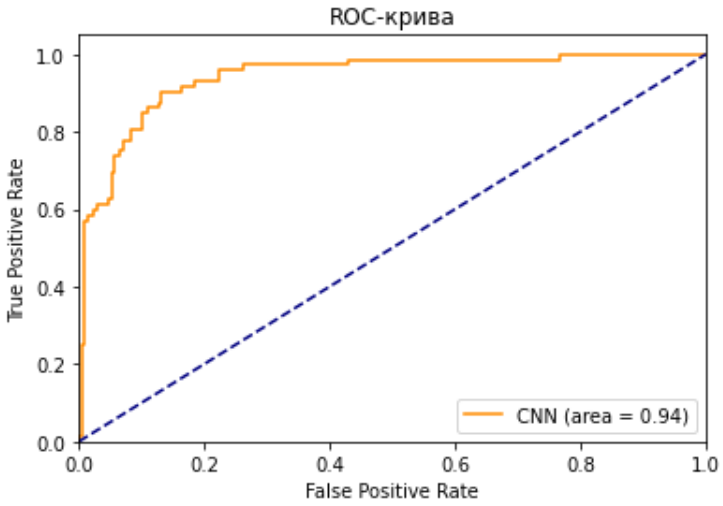

```
#Classification Neural Network
#Classification Neural Network на усіх змінних
# Install Tensorflow
# Install Keras
# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense
Using TensorFlow backend.
# Initialising the ANN
cnn = Sequential()
# Adding the input layer and the first hidden layer
cnn.add(Dense(output_dim = 33, init = 'uniform', activation =
'linear', input_dim =33))

# Adding the output layer
cnn.add(Dense(output_dim=1, init='uniform', activation = 'sigmoid'))
# Compiling the ANN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])
# Fitting the ANN to the Training set
history = cnn.fit(X_train, y_train, batch_size = 8, nb_epoch = 80,
validation_data=(X_test, y_test))
Train on 1050 samples, validate on 300 samples
# Plotting loss & accuracy
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')
plt.show()
```

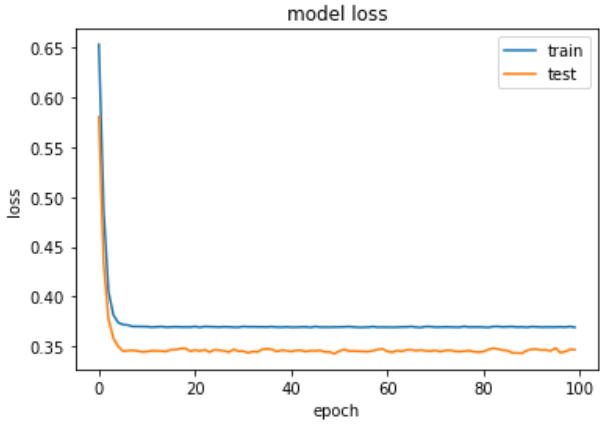



```
# Predicting the Test set results
y_pred = cnn.predict(X_test)
y_pred = (y_pred > 0.52)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
[[198  30]
 [  7  65]]

Accurary_rate = (198+65)/(300)
Accurary_rate
                                                    Out[277]:
0.8766666666666667

Error_rate = (7+30)/(300)
Error_rate
                                                    Out[278]:
0.12333333333333334

Se = 65/(65+7)
Se
                                                    Out[279]:
0.9027777777777778

Sp = 198/(198+30)
Sp
                                                    Out[280]:
0.868421052631579

cnn_probs = cnn.predict_proba(X_test)
cnn_probs = cnn_probs[:, -1]
cnn_auc = roc_auc_score(y_test, cnn_probs)
print('CNN: ROC AUC=%.3f' % (cnn_auc))
```

```
fpr, tpr, treshold = roc_curve(y_test, cnn_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='CNN (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
CNN: ROC AUC=0.941
```




```
#Classification Neural Network на значущих змінних
# Initialising the ANN
cnn1 = Sequential()
# Adding the input layer and the first hidden layer
cnn1.add(Dense(output_dim = 9, init = 'uniform', activation =
'linear', input_dim = 10))
# Adding the output layer
cnn1.add(Dense(output_dim=1, init='uniform', activation='sigmoid'))
# Compiling the ANN
cnn1.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])
# Fitting the ANN to the Training set
history = cnn1.fit(bas_X_train, y_train, batch_size = 8, nb_epoch =
100, validation_data=(bas_X_test, y_test))
Train on 1050 samples, validate on 300 samples
# Plotting loss & accuracy
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')
plt.show()
```




```
# Predicting the Test set results
y_pred = cnn1.predict(bas_X_test)
y_pred = (y_pred > 0.49)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
[[197  31]
 [  5  67]]
Accurary_rate = (197+67)/(300)
Accurary_rate
```

Out[328]:

```
0.88

Error_rate = (5+31)/(300)
Error_rate
```

Out[329]:

```
0.12

Se = 67/(67+5)
Se
```

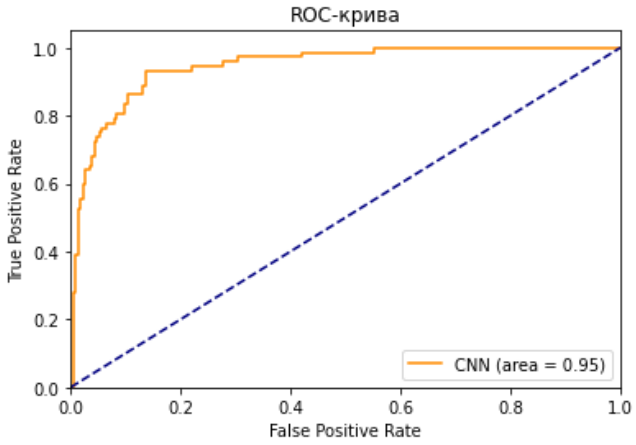
Out[330]:

```
0.9305555555555556

Sp = 197/(197+31)
Sp
```

Out[332]:

```
0.8640350877192983

cnn1_probs = cnn1.predict_proba(bas_X_test)
cnn1_probs = cnn1_probs[:, -1]
cnn1_auc = roc_auc_score(y_test, cnn1_probs)
print('CNN: ROC AUC=%.3f' % (cnn1_auc))
fpr, tpr, treshold = roc_curve(y_test, cnn1_probs)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange',
         label='CNN (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива')
plt.legend(loc="lower right")
plt.show()
CNN: ROC AUC=0.945
```



ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	119 с.
	Інші документи	
2.	Презентаційні матеріали	плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Зубова В.В.			Класифікація страхових випадків методами машинного навчання	Шифр групи	Код напр./спец.
Перевірив		Аврунін О.Г.				СШІмзд-22-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					