Menganalisa Datasheet Siswa Depresi

Dosen : Theopilus Bayu Sasongko, S.Kom., M.Eng.

Disusun oleh

Haidar Farhan Zaelani	22.11.5198
Mirza Hafiz	22.11.5213
Ivan Rizky Maulana Saputra	22.11.5229
Irhab Muqsiht Arrasyid	22.11.5243

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS AMIKOM YOGYAKARTA

2025

Alasan memilih bidang

Kita memilih di bidang Kesehatan dan Pendidikan yaitu tentang siswa yang depresi. Alasan kita memilih bidang ini karena tema yang menarik dan cukup bermanfaat karena dua bidang tersebut lumayan penting di kehidupan sehari-hari.

Tentang Datasheet

Kita menemukan datasheet tersebut menggunakan Kaggle, lalu menggunakan filter datasheet yang memiliki ukuran sedang dan terbaru.

Datasheet tersebut terupdate 2 bulan yang lalu dari laporan ini dibuat.

NURRIZKY ARUM JATMIKO AND 4 COLLABORATORS · UPDATED 2 MONTHS AGO

Penjelasan setiap kolom :

Gender : Kolom untuk mengidentifikasi alat kelamin.

Age : Kolom untuk mengetahui umur setiap siswa.

Academic Pressure : Kolom yang berisi tingkatan tekanan secara akademis

Study Satisification : Kolom untuk mengetahui kenyamanan setiap siswa saat belajar

Sleep Duration : Kolom berisi durasi siswa tidur

Dietary Habits : Kolom untuk mengetahui kebiasaan siswa saat menjaga pola makan

Have you ever have suicide thought : Kolom tentang siswa yang mempunyai pemikiran untuk bunuh diri atau tidak.

Study Hours : Kolom yang berisi durasi siswa belajar

Financial stress : Tingkatan stress yang dimiliki siswa terhadap finansial keluarga atau dirinya.

Family History Of Mental Illnes : Kolom tentang Riwayat keluarganya memiliki penyakit mental atau tidak.

Depression : Kolom yang menjadi label dan klasifikasi berisi ya atau tidak siswa memiliki depresi.

Sumber datasheet :

https://www.kaggle.com/datasets/ikynahidwin/depression-student-dataset

Preprocessing

```
# Menghitung jumlah nilai null di setiap kolom
df.select([count(when(isnull(c), c)).alias(c) for c in df.columns]).show()
```

```
+------+---+---------------+------------------+--------------+---------------+----------------+-----------+---------------+--------------+----------+
|Gender|Age|Academic Pressure|Study Satisfaction|Sleep Duration|Dietary Habits|Suicidal Thoughts|Study Hours|Financial Stress|Family History|Depression|
+------+---+---------------+------------------+--------------+---------------+----------------+-----------+---------------+--------------+----------+
|     0|  0|              0|                 0|             0|              0|               0|          0|              0|             0|         0|
+------+---+---------------+------------------+--------------+---------------+----------------+-----------+---------------+--------------+----------+
```

Melihat apakah terdapat nilai null atau tidak didalam datasheet,dan hasilnya datasheet yang kita pakai bersih dari nilai null.

```
[ ]  # nama kolom saya ganti akrena kepanjangan

     df = df.withColumnRenamed("Have you ever had suicidal thoughts ?", "Suicidal Thoughts") \
            .withColumnRenamed("Family History of Mental Illness", "Family History")
```

Mengganti Nama Kolom yang kepanjangan menggunakan df.withcolumnrenamed

```
[ ]  df.printSchema()

root
 |-- Age: integer (nullable = true)
 |-- Academic Pressure: double (nullable = true)
 |-- Study Satisfaction: double (nullable = true)
 |-- Study Hours: integer (nullable = true)
 |-- Financial Stress: integer (nullable = true)
 |-- Gender: double (nullable = false)
 |-- Sleep Duration: double (nullable = false)
 |-- Dietary Habits: double (nullable = false)
 |-- Suicidal Thoughts: double (nullable = false)
 |-- Family History: double (nullable = false)
 |-- Depression: double (nullable = false)
```

Diatas merupakan summary dari tipe setiap kolom, dan ternyata masih terdapat beberapa kolom yang kategorial.

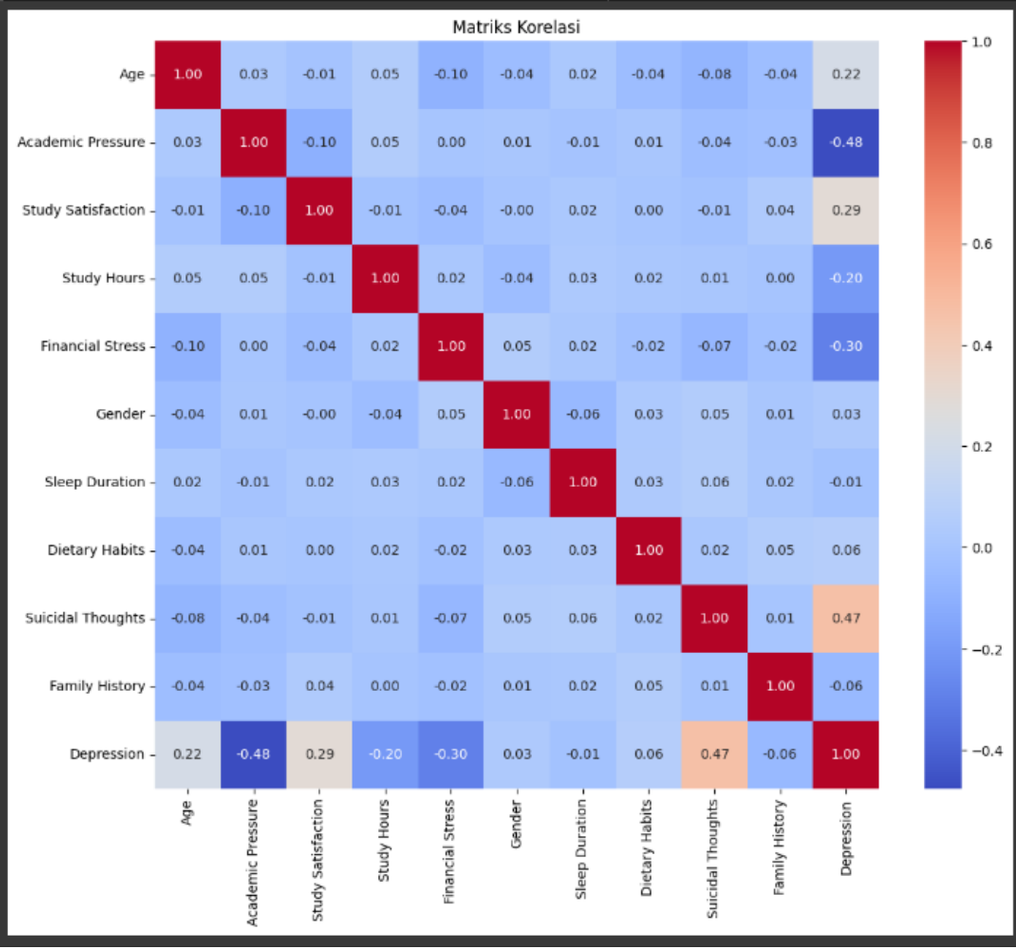
```python
# melakukan konversi karena masih terdapat kolom string
# Mendapatkan daftar kolom non-numerik
categorical_cols = [f.name for f in df.schema.fields if isinstance(f.dataType, StringType) and not isinstance(f.dataType, NumericType)]

# Loop melalui kolom non-numerik dan ubah menggunakan StringIndexer, lalu hapus kolom lama
for col in categorical_cols:
  indexer = StringIndexer(inputCol=col, outputCol=col + "_index")
  df = indexer.fit(df).transform(df)

  # Menghapus kolom lama dan mengganti nama kolom baru
  df = df.drop(col).withColumnRenamed(col + "_index", col)

# Menampilkan skema DataFrame (opsional)
df.printSchema()
```

```python
# mengecek apakah data sudah di konversi ke numerik semua atau belum
df.show()
```

```
+---+----------------+-----------------+----------+---------------+------+-------------+-------------+---------------+-------------+----------+
|Age|Academic Pressure|Study Satisfaction|Study Hours|Financial Stress|Gender|Sleep Duration|Dietary Habits|Suicidal Thoughts|Family History|Depression|
+---+----------------+-----------------+----------+---------------+------+-------------+-------------+---------------+-------------+----------+
| 28|             2.0|              4.0|         9|              2|   0.0|          0.0|          0.0|            0.0|          1.0|       1.0|
| 28|             4.0|              5.0|         7|              1|   0.0|          2.0|          2.0|            0.0|          1.0|       1.0|
| 25|             1.0|              3.0|        10|              4|   0.0|          2.0|          1.0|            0.0|          0.0|       0.0|
| 23|             1.0|              4.0|         7|              2|   0.0|          1.0|          1.0|            0.0|          1.0|       1.0|
| 31|             1.0|              5.0|         4|              2|   1.0|          1.0|          2.0|            0.0|          1.0|       1.0|
| 19|             4.0|              4.0|         1|              4|   0.0|          2.0|          1.0|            0.0|          1.0|       0.0|
| 34|             4.0|              2.0|         6|              2|   1.0|          1.0|          0.0|            0.0|          0.0|       0.0|
| 20|             4.0|              1.0|         3|              4|   1.0|          1.0|          2.0|            0.0|          1.0|       0.0|
| 33|             1.0|              4.0|        10|              3|   1.0|          1.0|          0.0|            1.0|          0.0|       1.0|
| 33|             4.0|              3.0|        10|              1|   0.0|          3.0|          1.0|            0.0|          0.0|       0.0|
| 31|             5.0|              4.0|         6|              4|   1.0|          2.0|          2.0|            0.0|          0.0|       0.0|
| 24|             2.0|              1.0|        11|              5|   0.0|          0.0|          1.0|            0.0|          0.0|       0.0|
| 23|             5.0|              5.0|         2|              1|   1.0|          3.0|          1.0|            0.0|          1.0|       0.0|
| 25|             1.0|              1.0|        12|              3|   0.0|          2.0|          0.0|            0.0|          1.0|       0.0|
| 21|             5.0|              1.0|         3|              5|   0.0|          1.0|          1.0|            0.0|          1.0|       0.0|
| 28|             5.0|              3.0|         8|              3|   0.0|          2.0|          2.0|            0.0|          1.0|       0.0|
| 23|             5.0|              2.0|        10|              4|   0.0|          1.0|          0.0|            1.0|          0.0|       0.0|
| 23|             1.0|              3.0|         0|              3|   1.0|          3.0|          2.0|            0.0|          0.0|       1.0|
| 20|             5.0|              5.0|         2|              5|   1.0|          1.0|          1.0|            0.0|          0.0|       0.0|
| 29|             4.0|              3.0|         1|              3|   0.0|          1.0|          1.0|            0.0|          0.0|       0.0|
+---+----------------+-----------------+----------+---------------+------+-------------+-------------+---------------+-------------+----------+
only showing top 20 rows
```

Mengubah kolom yang masih kategorial menjadi numerik menggunakan kodingan diatas, metode encoder yang ada tidak menggunakan labelencoder karena kita menggunakan pyspark dimana labelencoder tidak dapat digunakan. Untuk code kedua memverikasi bahwa semua kolom sudah numerik dalam bentuk show.

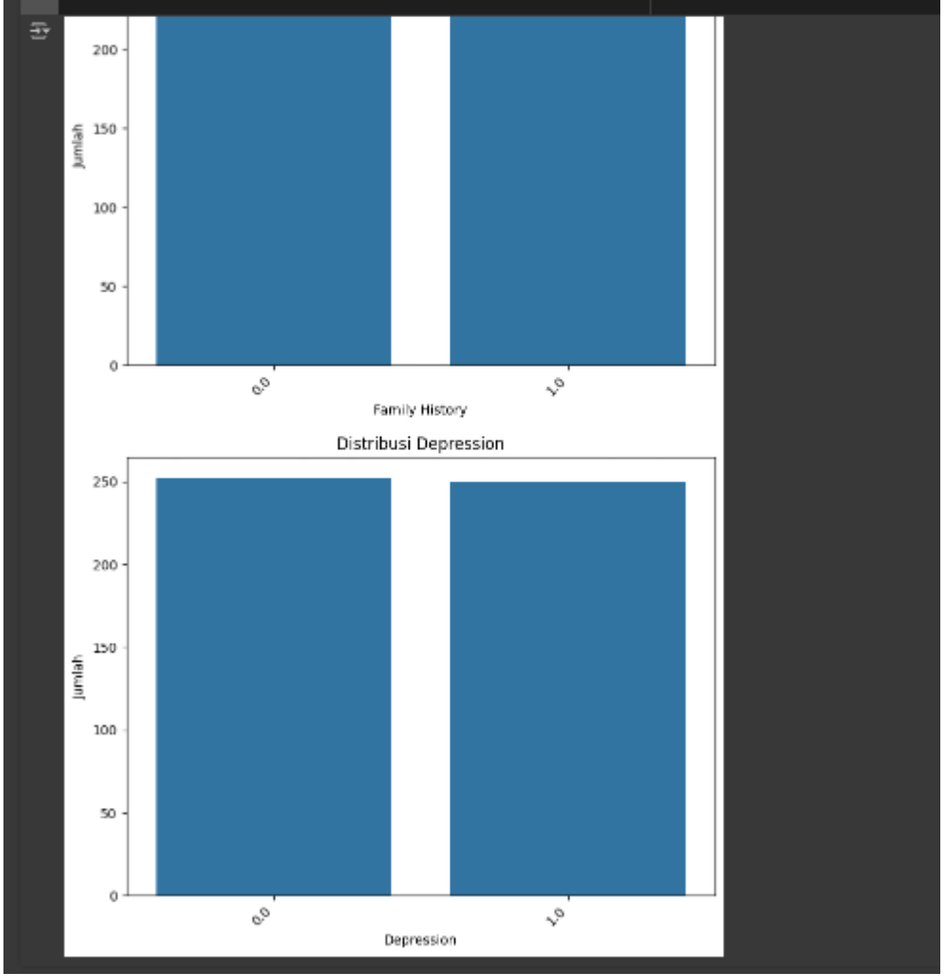
```
# visualisasi data menggunakan heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", xticklabels=numeric_cols, yticklabels=numeric_cols)
plt.title('Matriks Korelasi')
plt.show()
```



Lalu kita melakukan visualisasi korelasi antar kolom menggunakan heatmap seperti diatas.

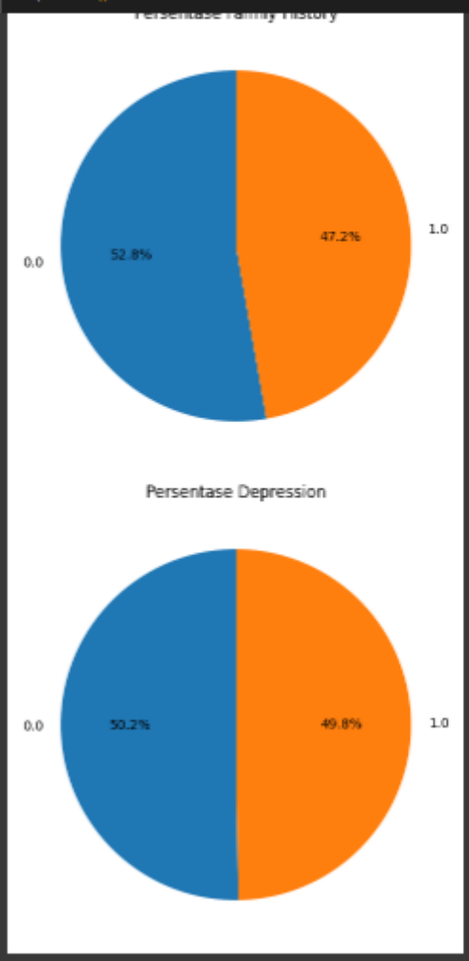
EDA

```python
# Loop melalui semua kolom dalam DataFrame
for column in df.columns:
    # Mengambil data untuk kolom saat ini
    column_data = df.select(column).toPandas()

    # Membuat bar chart menggunakan seaborn dan matplotlib
    plt.figure(figsize=(8, 6))
    sns.countplot(x=column, data=column_data)
    plt.title(f'Distribusi {column}')
    plt.xlabel(column)
    plt.ylabel('Jumlah')
    plt.xticks(rotation=45, ha='right')  # Rotasi label sumbu x jika diperlukan
    plt.show()
```



Melakukan visualisasi menggunakan barchart untuk mengetahui distribusi tiap kolom

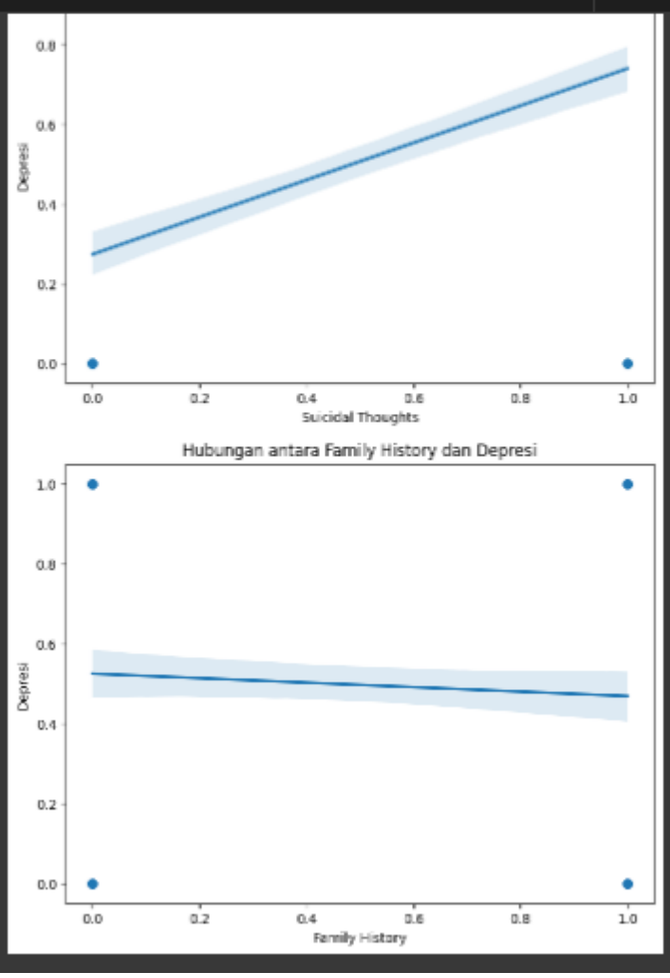
```python
# Loop melalui semua kolom dalam DataFrame
for column in df.columns:
    # Menghitung frekuensi nilai dalam kolom
    value_counts = df.groupby(cSolumn).count().toPandas()

    # Membuat pie chart menggunakan matplotlib
    plt.figure(figsize=(6, 6))
    plt.pie(value_counts["count"], labels=value_counts[column], autopct='%1.1f%%', startangle=90)
    plt.title(f'Persentase {column}')
    plt.show()
```



Persentase Depression

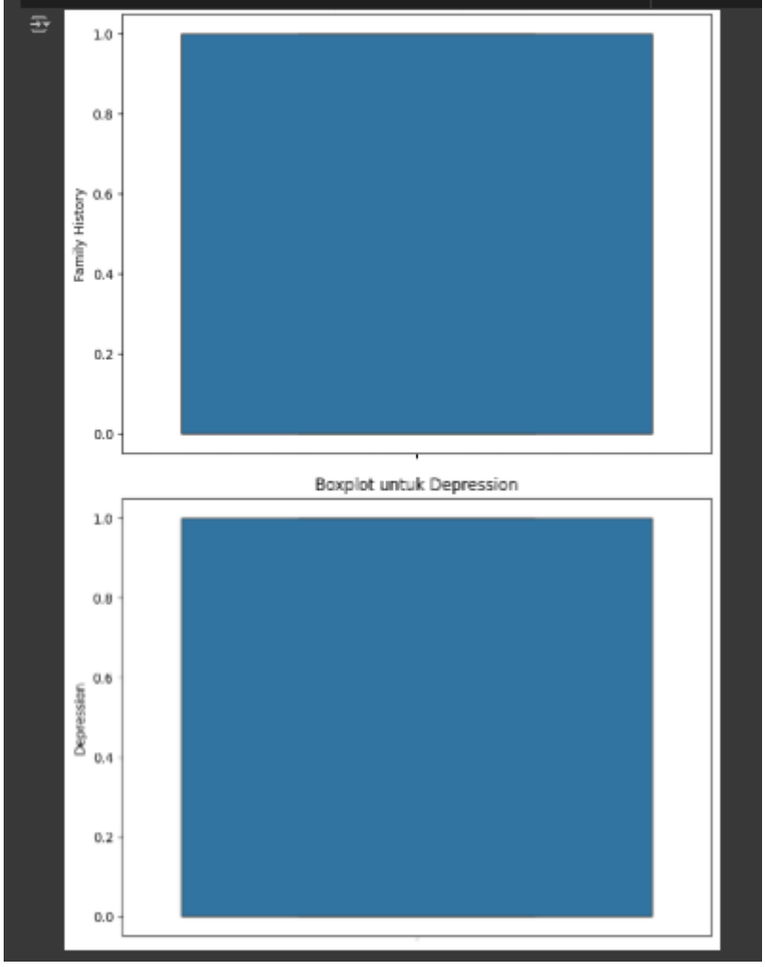
Melakukan visualisasi data setiap kolom menggunakan piechart.

```python
# Mendapatkan daftar kolom numerik
numeric_cols = [f.name for f in df.schema.fields if isinstance(f.dataType, NumericType)]

# Loop melalui kolom numerik (kecuali 'Depression')
for column in numeric_cols:
    if column != 'Depression':
        # Mengambil data untuk kolom saat ini dan 'Depression'
        data = df.select(column, 'Depression').toPandas()

        # Membuat scatter plot dengan garis regresi menggunakan seaborn dan matplotlib
        plt.figure(figsize=(8, 6))
        sns.regplot(x=column, y='Depression', data=data, scatter_kws={'alpha':0.5})
        plt.title(f'Hubungan antara {column} dan Depresi')
        plt.xlabel(column)
        plt.ylabel('Depresi')
        plt.show()

#untuk melihat korelasi
```



Visualisasi korelasi setiap kolom dengan kolom depresi, sehingga dapat mengetahui lebih mudah korelasi negative/positifnya dan seberapa tajam.

```
[145] # Mendapatkan daftar kolom numerik
      numeric_cols = [f.name for f in df.schema.fields if isinstance(f.dataType, NumericType)]

      # Loop melalui setiap kolom numerik dan buat boxplot terpisah
      for column in numeric_cols:
          # Mengambil data untuk kolom saat ini
          column_data = df.select(column).toPandas()

          # Membuat boxplot menggunakan seaborn dan matplotlib
          plt.figure(figsize=(8, 6))  # Mengatur ukuran figure
          sns.boxplot(y=column, data=column_data)  # Membuat boxplot untuk kolom saat ini
          plt.title(f'Boxplot untuk {column}')  # Memberi judul plot dengan nama kolom
          plt.show()  # Menampilkan plot
```



Visualisasi dalam bentuk boxplot untuk mengetahui apakah terdapat kolom yang memiliki noise atau tidak

```
categoricalColumns = ['Gender', 'Sleep Duration', 'Dietary Habits', 'Have you ever had suicidal thoughts ?', 'Family History of Mental Illness']
stages = []

for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + 'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]

label_stringIdx = StringIndexer(inputCol='Depression', outputCol='label')
stages += [label_stringIdx]

numericCols = ['Age', 'Academic Pressure', 'Study Satisfaction', 'Study Hours', 'Financial Stress']
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

```
[152] from pyspark.ml import Pipeline

pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(df_machinelearning)
df_transformed = pipelineModel.transform(df_machinelearning)

selectedCols = ['label', 'features'] + df_machinelearning.columns
df_transformed = df_transformed.select(selectedCols)
df_machinelearning = df_transformed
df_machinelearning.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Academic Pressure: double (nullable = true)
 |-- Study Satisfaction: double (nullable = true)
 |-- Sleep Duration: string (nullable = true)
 |-- Dietary Habits: string (nullable = true)
 |-- Have you ever had suicidal thoughts ?: string (nullable = true)
 |-- Study Hours: integer (nullable = true)
 |-- Financial Stress: integer (nullable = true)
 |-- Family History of Mental Illness: string (nullable = true)
 |-- Depression: string (nullable = true)
```

Kita memilih semua kolom yang ada kecuali depression menjadi fitur. Kenapa? Karena Ketika kita mencoba membuang beberapa kolom yang memiliki korelasi kecil atau kurang dari 0.1 terhadap kolom depresi, accuracy yang didapat menurun jauh. Sehingga lebih aman menggunakan semua kolom yang ada.

Pengembangan model machine learning.

```
train, test = df_machinelearning.randomSplit([0.8, 0.2], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 400
Test Dataset Count: 102
```

Melakukan Split dan train data seperti biasa disini saya menggunakan perbandinga 80:20 karena Ketika saya menggunakan 90:10 tidak terdapat peningkatan maka dari itu menggunakan perbandingan yang lebih optimal yaitu 80:20.

Kita menggunakan 4 model yaitu Naïve bayes, LinearSVC, Gradient Boost Tree,dan Random Forest

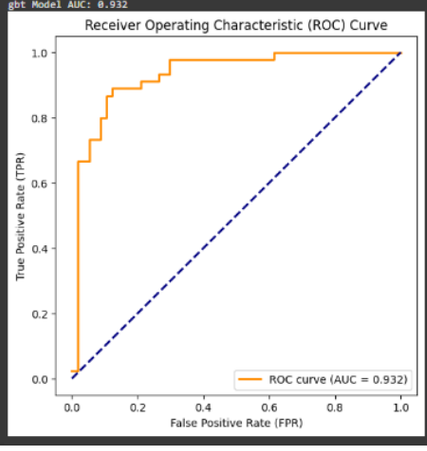
```python
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# Gradient Boosting Model
gbt = GBTClassifier(featuresCol='features', labelCol='label', maxIter=10)  # Initialize GBT model
model_gbt = gbt.fit(train)  # Train GBT model

# Make predictions on the test data
predictions_gbt = model_gbt.transform(test)
# Evaluate the Model (Accuracy)
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy_gbt = evaluator.evaluate(predictions_gbt)
print("Accuracy (Gradient Boosting) = %g" % accuracy_gbt)

# Evaluate the Model (F1 Score)
evaluator.setMetricName("f1")
f1_score_gbt = evaluator.evaluate(predictions_gbt)
print("F1 Score (Gradient Boosting) = %g" % f1_score_gbt)

evaluator.setMetricName("weightedPrecision")
precision_rf = evaluator.evaluate(predictions_gbt)
print("Precision (Gradient Boosting = %g" % precision_rf)

evaluator.setMetricName("weightedRecall")
recall_rf = evaluator.evaluate(predictions_gbt)
print("Recall (Gradient Boosting) = %g" % recall_rf)
```

```
Accuracy (Gradient Boosting) = 0.872549
F1 Score (Gradient Boosting) = 0.872882
Precision (Gradient Boosting = 0.874728
Recall (Gradient Boosting) = 0.872549
```

Kode diatas melakukan modeling menggunakan gradient boosting dan model tersebut mendapat accuracy di 0.87, accuracy tersebut cukup bagus namun bukan paling optimal untuk datasheet yang kita miliki.

```python
from sklearn.metrics import auc
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.functions import vector_to_array
import pandas as pd
import matplotlib.pyplot as plt

# Ekstrak probabilitas prediksi
predictions_gbt = predictions_gbt.withColumn("probability_array", vector_to_array("probability"))

# Evaluator untuk menghitung AUC
evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="probability", metricName="areaUnderROC")
auc_score = evaluator.evaluate(predictions_gbt)
print(f"gbt Model AUC: {auc_score:.3f}")

# Hitung ROC secara manual
roc_data = predictions_gbt.select("label", "probability_array").rdd \
    .map(lambda row: (float(row["probability_array"][1]), float(row["label"]))) \
    .toDF(["probability", "label"])

roc_pd = pd.DataFrame(roc_data.sort("probability", ascending=False).toPandas())
roc_pd = roc_pd.sort_values(by="probability", ascending=False)  # Pastikan sorting benar
roc_pd['TPR'] = roc_pd['label'].cumsum() / roc_pd['label'].sum()  # TPR
roc_pd['FPR'] = ((1 - roc_pd['label']).cumsum()) / (1 - roc_pd['label']).sum()  # FPR

# Plot ROC Curve
plt.figure(figsize=(6, 6))
plt.plot(roc_pd['FPR'], roc_pd['TPR'], color='darkorange', lw=2, label=f'ROC curve (AUC = {auc_score:.3f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
gbt Model AUC: 0.932
```



Kode diatas menampilkan AUC/ROC yang dimiliki oleh model gradient boosting

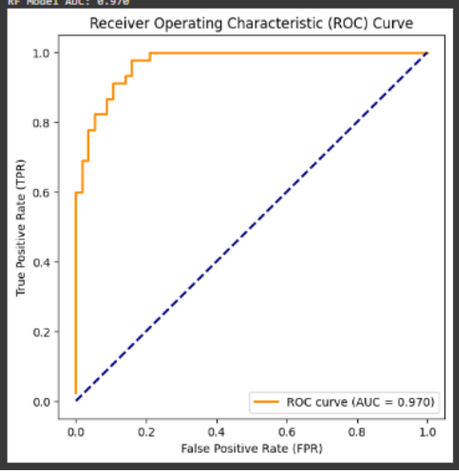
```python
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

rf = RandomForestClassifier(featuresCol='features', labelCol='label', numTrees=10)  # Initialize RF model
model_rf = rf.fit(train)
# Make predictions on the test data
predictions_rf = model_rf.transform(test)

# Evaluate the Model (Accuracy)
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy_rf = evaluator.evaluate(predictions_rf)
print("Accuracy (Random Forest) = %g" % accuracy_rf)

# Evaluate the Model (F1 Score)
evaluator.setMetricName("f1")
f1_score_rf = evaluator.evaluate(predictions_rf)
print("F1 Score (Random Forest) = %g" % f1_score_rf)

evaluator.setMetricName("weightedPrecision")
precision_rf = evaluator.evaluate(predictions_rf)
print("Precision (Random Forest) = %g" % precision_rf)

evaluator.setMetricName("weightedRecall")
recall_rf = evaluator.evaluate(predictions_rf)
print("Recall (Random Forest) = %g" % recall_rf)
```

```
Accuracy (Random Forest) = 0.882353
F1 Score (Random Forest) = 0.882353
Precision (Random Forest) = 0.882353
Recall (Random Forest) = 0.882353
```

Melakukan modeling menggunakan random forest, seperti gambar diatas accuracy yang didapat yaitu 0.88 lebih tinggi sedikit daripada gradient boosting.

```python
from sklearn.metrics import auc
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.functions import vector_to_array
import pandas as pd
import matplotlib.pyplot as plt

# Ekstrak probabilitas prediksi
predictions_rf = predictions_rf.withColumn("probability_array", vector_to_array("probability"))

# Evaluator untuk menghitung AUC
evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="probability", metricName="areaUnderROC")
auc_score = evaluator.evaluate(predictions_rf)
print(f"RF Model AUC: {auc_score:.3f}")

# Hitung ROC secara manual
roc_data = predictions_rf.select("label", "probability_array").rdd \
    .map(lambda row: (float(row["probability_array"][1]), float(row["label"]))) \
    .toDF(["probability", "label"])

roc_pd = pd.DataFrame(roc_data.sort("probability", ascending=False).toPandas())
roc_pd = roc_pd.sort_values(by="probability", ascending=False)  # Pastikan sorting benar
roc_pd['TPR'] = roc_pd['label'].cumsum() / roc_pd['label'].sum()  # TPR
roc_pd['FPR'] = ((1 - roc_pd['label']).cumsum()) / (1 - roc_pd['label']).sum()  # FPR

# Plot ROC Curve
plt.figure(figsize=(6, 6))
plt.plot(roc_pd['FPR'], roc_pd['TPR'], color='darkorange', lw=2, label=f'ROC curve (AUC = {auc_score:.3f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
RF Model AUC: 0.970
```

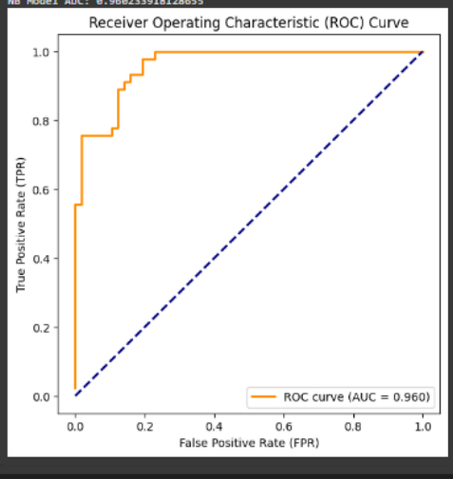


Kode diatas menampilkan ROC/AUC yang dimiliki oleh model random forest

```
[193] # Inisialisasi Naive Bayes
      nb = NaiveBayes(featuresCol='features', labelCol='label', smoothing=30)

      # Latih model dengan data training
      model_nb = nb.fit(train)

      # Prediksi data testing
      predictions_nb = model_nb.transform(test)

      # Evaluasi Akurasi
      evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
      accuracy_nb = evaluator.evaluate(predictions_nb)
      print(f"Accuracy (Naive Bayes) = {accuracy_nb}")

      # Evaluasi F1-Score
      evaluator.setMetricName("f1")
      f1_score_nb = evaluator.evaluate(predictions_nb)
      print(f"F1 Score (Naive Bayes) = {f1_score_nb}")

      evaluator.setMetricName("weightedPrecision")
      precision_nb = evaluator.evaluate(predictions_nb)
      print("Precision (Random Forest) = %g" % precision_nb)

      evaluator.setMetricName("weightedRecall")
      recall_nb = evaluator.evaluate(predictions_nb)
      print("Recall (Random Forest) = %g" % recall_nb)

      Accuracy (Naive Bayes) = 0.8627450980392157
      F1 Score (Naive Bayes) = 0.8631679086729032
      Precision (Random Forest) = 0.874221
      Recall (Random Forest) = 0.862745
```

Kode diatas melakukan Modeling terhadap naïve bayes, dimana accuracy yang didapat menurun dibandingkan menggunakan model random forest

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.functions import vector_to_array
import pandas as pd
import matplotlib.pyplot as plt

# Ekstrak probabilitas prediksi
predictions_nb = predictions_nb.withColumn("probability_array", vector_to_array("probability"))

# Evaluator untuk menghitung AUC
evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="probability", metricName="areaUnderROC")
auc_score = evaluator.evaluate(predictions_nb)
print(f"NB Model AUC: {auc_score}")

# Hitung ROC secara manual
roc_data = predictions_nb.select("label", "probability_array").rdd \
    .map(lambda row: (float(row["probability_array"][1]), float(row["label"]))) \
    .toDF(["probability", "label"])

roc_pd = pd.DataFrame(roc_data.sort("probability", ascending=False).toPandas())
roc_pd = roc_pd.sort_values(by="probability", ascending=False)  # Pastikan sorting benar
roc_pd['TPR'] = roc_pd['label'].cumsum() / roc_pd['label'].sum()  # TPR
roc_pd['FPR'] = ((1 - roc_pd['label']).cumsum()) / (1 - roc_pd['label']).sum()  # FPR


# Plot ROC Curve
plt.figure(figsize=(6, 6))
plt.plot(roc_pd['FPR'], roc_pd['TPR'], color='darkorange', lw=2, label=f'ROC curve (AUC = {auc_score:.3f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

NB Model AUC: 0.960233918128655
```



Kode diatas menampilkan ROC/AUC yang dimiliki oleh random forest.

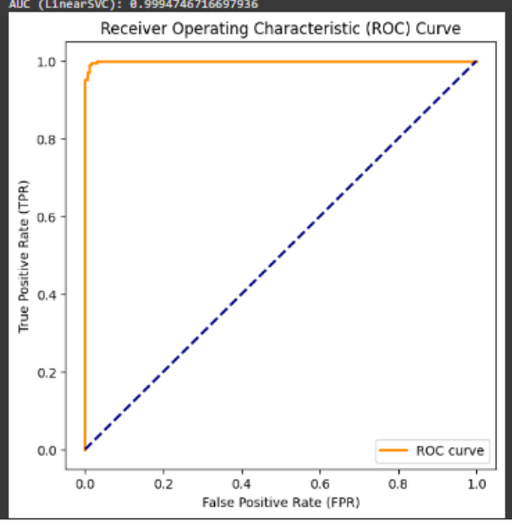
```python
# Inisialisasi model LinearSVC (SVM)
svm = LinearSVC(featuresCol='features', labelCol='label', maxIter=10)

# Latih model
svm_model = svm.fit(train)

# Prediksi
predictions_svm = svm_model.transform(test)

# Evaluasi Akurasi
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy_svm = evaluator.evaluate(predictions_svm)
print(f"Accuracy (SVM) = {accuracy_svm}")

# Evaluasi F1-Score
evaluator.setMetricName("f1")
f1_score_svm = evaluator.evaluate(predictions_svm)
print(f"F1 Score (SVM) = {f1_score_svm}")

# Calculate Precision and Recall
evaluator.setMetricName("weightedPrecision")  # For precision
precision_svm = evaluator.evaluate(predictions_svm)
print(f"Precision (SVM) = {precision_svm}")

evaluator.setMetricName("weightedRecall")  # For recall
recall_svm = evaluator.evaluate(predictions_svm)
print(f"Recall (SVM) = {recall_svm}")
```

```
Accuracy (SVM) = 0.9607843137254902
F1 Score (SVM) = 0.9608604311289732
Precision (SVM) = 0.9616793719422003
Recall (SVM) = 0.9607843137254902
```

Melakukan modeling menggunakan linear svc/SVM, model ini merupakan model terbaik dan paling cocok dengan datasheet yang kita memiliki dapat dilihat model ini mendapatkan accuracy dengan nilai 0.96.

```python
from sklearn.metrics import auc
trainingSummary = svm_model.summary
summary_object = trainingSummary()  # Call to get the summary object
roc = summary_object.roc.toPandas()

# Calculate AUC (using imported 'auc' function)
roc_auc_score = auc(roc['FPR'], roc['TPR'])
print(f"AUC (LinearSVC): {roc_auc_score}")
# Rest of your code for plotting the ROC curve remains the same
plt.figure(figsize=(6, 6))
plt.plot(roc['FPR'], roc['TPR'], color='darkorange', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
AUC (LinearSVC): 0.9994746716697936
```



Bahkan ROC/AUC yang dimiliki oleh Linear SVC/SVM memiliki nilai yang hampir sempurna.

HyperTuning

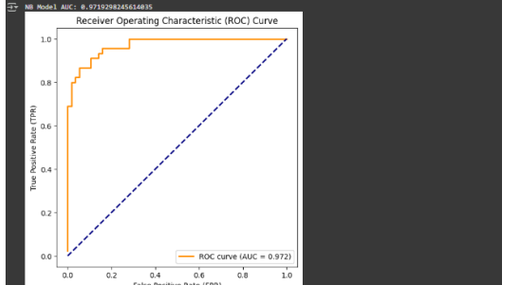
Kita melakukan HyperTuning terhadap model 2 terbaik yaitu Linear SVC/SVM dan random forest, karena kedua model tersebut memiliki accuracy paling tinggi.

```python
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Initialize Random Forest model
rf = RandomForestClassifier(featuresCol='features', labelCol='label')

# Create parameter grid for tuning
paramGrid = (ParamGridBuilder()
             .addGrid(rf.numTrees, [10, 20, 50])  # Number of trees
             .addGrid(rf.maxDepth, [5, 10, 15])   # Maximum depth of trees
             .addGrid(rf.maxBins, [32, 64])       # Maximum bins for splitting
             .build())

# Define evaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

# Setup CrossValidator
crossval = CrossValidator(estimator=rf,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)  # 5-fold cross-validation

# Fit the model using cross-validation
cvModel = crossval.fit(train)

# Best model from cross-validation
bestModel = cvModel.bestModel

# Make predictions on test data
predictions_rf = bestModel.transform(test)

# Evaluate final model
accuracy_rf = evaluator.evaluate(predictions_rf)
print("Best Model Accuracy (Random Forest) = %g" % accuracy_rf)

# Evaluate F1 Score
evaluator.setMetricName("f1")
f1_score_rf = evaluator.evaluate(predictions_rf)
print("Best Model F1 Score (Random Forest) = %g" % f1_score_rf)

# Calculate Precision and Recall
evaluator.setMetricName("weightedPrecision")  # For precision
precision_rf = evaluator.evaluate(predictions_rf)
print(f"Precision (Random Forest)  = {precision_svm}")

evaluator.setMetricName("weightedRecall")  # For recall
recall_rf = evaluator.evaluate(predictions_rf)
print(f"Recall (Random Forest)  = {recall_svm}")
```

```
Best Model Accuracy (Random Forest) = 0.892157
Best Model F1 Score (Random Forest) = 0.89202
Precision (Random Forest)  = 0.9004092071611252
Recall (Random Forest)  = 0.9001960784313726
```

```python
from sklearn.metrics import auc
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.functions import vector_to_array
import pandas as pd
import matplotlib.pyplot as plt

# Ekstrak probabilitas prediksi
predictions_rf = predictions_rf.withColumn("probability_array", vector_to_array("probability"))

# Evaluator untuk menghitung AUC
evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="probability", metricName="areaUnderROC")
auc_score = evaluator.evaluate(predictions_rf)
print(f"NB Model AUC: {auc_score}")

# Hitung ROC secara manual
roc_data = predictions_rf.select("label", "probability_array").rdd \
    .map(lambda row: (float(row["probability_array"][1]), float(row["label"]))) \
    .toDF(["probability", "label"])

roc_pd = pd.DataFrame(roc_data.sort("probability", ascending=False).toPandas())
roc_pd = roc_pd.sort_values(by='probability', ascending=False)  # Pastikan sorting benar
roc_pd['TPR'] = roc_pd['label'].cumsum() / roc_pd['label'].sum()  # TPR
roc_pd['FPR'] = ((1 - roc_pd['label']).cumsum()) / (1 - roc_pd['label']).sum()  # FPR


# Plot ROC Curve
plt.figure(figsize=(6, 6))
plt.plot(roc_pd['FPR'], roc_pd['TPR'], color='darkorange', lw=2, label=f'ROC curve (AUC = {auc_score:.3f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
NB Model AUC: 0.9719298245614035
```



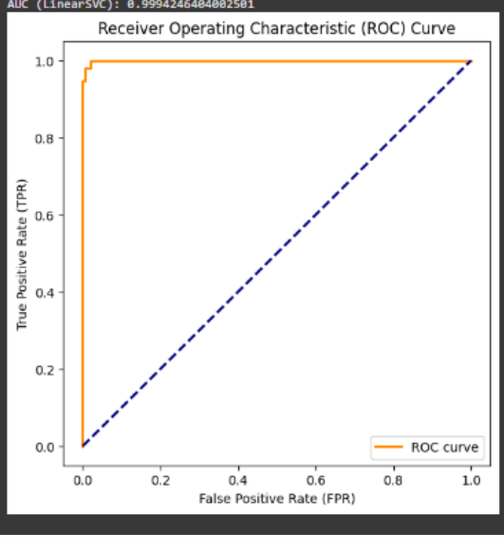
Kode diatas melakukan hypertuning terhadap random forest beserta hasilnya, mendapatkan hasil accuracy 0.89 terdapat peningkatan sebanyak 0.01 dari 0.88 sebelum hypertuning. Untuk ROC/AUC nya juga memiliki sedikit peningkatan.

```python
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator


# Inisialisasi model LinearSVC (SVM)
svm = LinearSVC(featuresCol='features', labelCol='label')

# Define the parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder() \
    .addGrid(svm.maxIter, [10, 15]) \
    .addGrid(svm.regParam, [0.01, 0.05]) \
    .build()

# Define the evaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

# Create a CrossValidator
cv = CrossValidator(estimator=svm, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=3)  # Use 3 folds for cross-validation

# Fit the model with hyperparameter tuning
cvModel = cv.fit(train)

# Get the best model from cross-validation
bestModel = cvModel.bestModel

# Make predictions on the test data using the best model
predictions_svm1 = bestModel.transform(test)

# Evaluate the best model
accuracy_svm = evaluator.evaluate(predictions_svm1)
print(f"Accuracy (SVM) = {accuracy_svm}")

evaluator.setMetricName("f1")
f1_score_svm = evaluator.evaluate(predictions_svm1)
print(f"F1 Score (SVM) = {f1_score_svm}")

# Calculate Precision and Recall
evaluator.setMetricName("weightedPrecision")  # For precision
precision_svm = evaluator.evaluate(predictions_svm1)
print(f"Precision (SVM) = {precision_svm}")

evaluator.setMetricName("weightedRecall")  # For recall
recall_svm = evaluator.evaluate(predictions_svm1)
print(f"Recall (SVM) = {recall_svm}")
# Prediksi pada data pelatihan
predictions_train = bestModel.transform(train)
```

```
Accuracy (SVM) = 0.9901960784313726
F1 Score (SVM) = 0.9902865659483672
Precision (SVM) = 0.9904092871611252
Recall (SVM) = 0.9901960784313726
```

```python
from sklearn.metrics import auc
trainingSummary = bestModel.summary
summary_object = trainingSummary()  # Call to get the summary object
roc = summary_object.roc.toPandas()

# Calculate AUC (using imported 'auc' function)
roc_auc_score = auc(roc['FPR'], roc['TPR'])
print(f"AUC (LinearSVC): {roc_auc_score}")
# Rest of your code for plotting the ROC curve remains the same
plt.figure(figsize=(6, 6))
plt.plot(roc['FPR'], roc['TPR'], color='darkorange', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
AUC (LinearSVC): 0.9994246404002501
```



Kode diatas melakukan hypertuning terhadap model linear SVC/SVM dan hasil yang didapatkan cukup memuaskan dengan accuracy mendekati 1 yaitu 0.99 namun terdapat penuruna di roc/auc nya dengan selisih yang sangat sedikit sehingga hampir tidak berubah.

Sifat Linear SVC yang cocok dengan datasheet kita :

Hubungan Linier Antara Fitur dan Label

- Jika fitur-fitur dalam data memiliki hubungan yang cukup jelas dan linier dengan hasil (label 0 atau 1), Linear SVM akan bekerja sangat baik.
- Misalnya, semakin tinggi **Financial Stress**, semakin besar kemungkinan labelnya adalah **Yes (1)**. Hubungan sederhana seperti ini cocok untuk dipisahkan dengan garis lurus.

Data yang Rapi dan Tidak Berisik

- Jika data tidak memiliki banyak outlier (data yang menyimpang jauh dari pola umum), Linear SVM lebih mudah membangun garis pemisah yang stabil.

Distribusi Data yang Seimbang

- Jika jumlah data di kelas 0 dan 1 tidak terlalu timpang, Linear SVM dapat bekerja lebih optimal karena tidak akan cenderung memihak salah satu kelas.

Jumlah Fitur yang Tidak Berlebihan

- Linear SVM lebih efisien jika jumlah fitur tidak terlalu banyak.