ADVANCED MACHINE LEARNING TECHNIQUES FOR DISEASE PREDICTION: LEVERAGING SUPPORT VECTOR MACHINES (SVM) AND K-NEAREST NEIGHBORS (KNN)

MACHINE LEARNING (DSCI-6003-03)

Team Members:

- **GIRI CHANDARAGIRI (00870321)**
- **SHAAGUN SURESH (00859167)**
- **SRAVANTHI KADARI (00858480)**

Department of Data Science

University Of New Haven

26 April 2024

1. Introduction:

The Comprehensive Disease Symptom and Patient Profile Dataset serve as a foundational resource for medical research and healthcare analytics. By aggregating a diverse array of symptoms, demographic variables, and health indicators across various diseases, this dataset offers a unique opportunity to unravel the complex interplay of factors influencing disease presentation, progression, and outcomes. Moreover, this report explores the application of Support Vector Classification (SVC) and k-Nearest Neighbors (kNN) algorithms in predictive modeling tasks using the dataset.

Support Vector Classification (SVC):

SVC is a powerful machine learning algorithm that excels in binary classification tasks by finding the optimal hyperplane that separates different classes with the maximum margin. It is particularly well-suited for datasets with high dimensionality and complex decision boundaries. In the context of disease diagnosis, SVC can be utilized to develop predictive models that classify patients into positive or negative outcomes based on their symptom profiles, demographic characteristics, and health indicators. By leveraging the dataset's rich feature set, SVC can identify intricate patterns and relationships among variables, enabling accurate and reliable disease classification and risk stratification.

k-Nearest Neighbors (kNN):

kNN is a simple yet effective algorithm for classification and regression tasks, relying on the principle of proximity-based reasoning. It classifies a new data point by assigning it the majority class label among its k nearest neighbors in the feature space. kNN is non-parametric and instance-based, making it robust to noisy data and complex decision boundaries. In the context of disease prediction, kNN can be applied to the dataset to develop classification models that predict disease outcomes based on the similarity of patient profiles. By measuring the similarity between patients in terms of their symptom presentations, demographic attributes, and health status, kNN can facilitate personalized risk assessment and treatment planning, thereby enhancing clinical decision-making and patient care.

2. Dataset Overview:

The dataset encompasses a wide spectrum of attributes, each meticulously selected to capture essential aspects of disease manifestation and patient characteristics. Here's an in-depth overview of the key columns and their significance:

Disease: The primary focus of analysis, this column specifies the name of the disease or medical condition under consideration. Disease names serve as categorical variables, enabling stratified analyses and comparisons across different health conditions.

Symptoms (Fever, Cough, Fatigue, Difficulty Breathing): These binary variables indicate the presence or absence of specific symptoms commonly associated with various diseases. Fever, cough, fatigue, and difficulty breathing represent cardinal symptoms that can offer valuable diagnostic clues and insights into disease severity and progression.

Demographic Variables (Age, Gender): Demographic factors such as age and gender play pivotal roles in shaping disease epidemiology, susceptibility, and outcomes. Age serves as a continuous variable, facilitating age-stratified analyses to discern age-related patterns and vulnerabilities. Gender, a categorical variable, enables gender-specific investigations to address sex-based disparities in disease prevalence and prognosis.

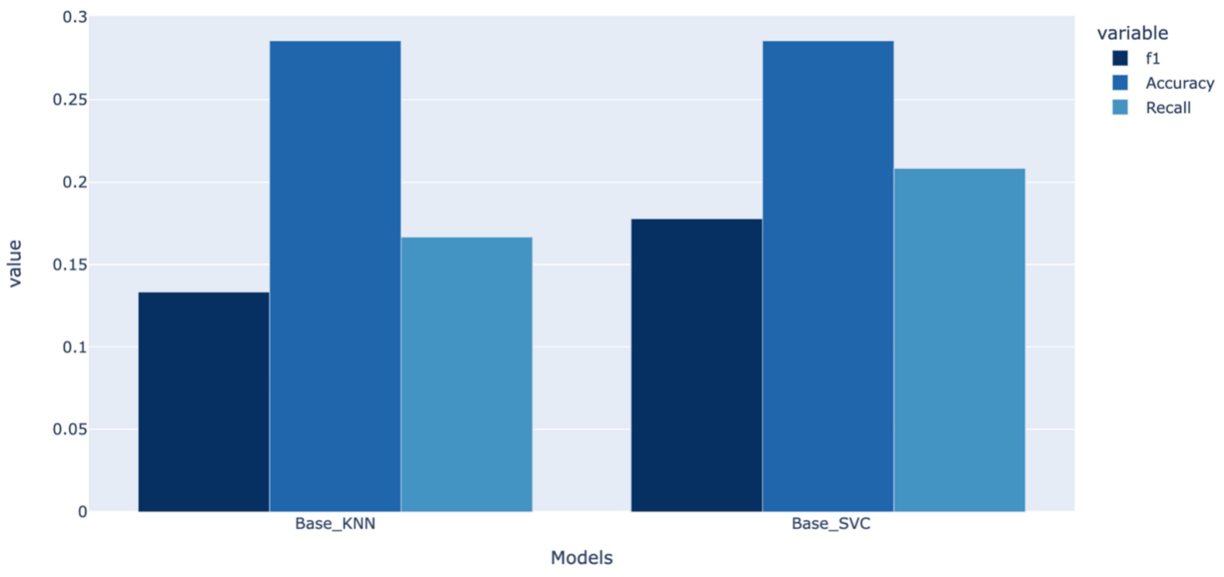
Health Indicators (Blood Pressure, Cholesterol Level): These variables provide crucial insights into the cardiovascular health and metabolic status of patients. Blood pressure levels (normal or high) reflect vascular integrity and hypertension risk, while cholesterol levels (normal or high) offer indicators of lipid metabolism and cardiovascular disease risk.

Outcome Variable: This binary variable indicates the result of the diagnostic assessment for the specific disease, categorizing patients into positive or negative outcomes. Outcome variables serve as the target variable in predictive modeling tasks, facilitating the development of diagnostic algorithms and decision support systems.

3. Model Selection:

Selecting the most appropriate machine learning models is crucial for achieving accurate predictions and meaningful insights from the Comprehensive Disease Symptom and Patient Profile Dataset. Model selection involves evaluating various algorithms based on their suitability for the task at hand, considering factors such as the dataset's size, dimensionality, and the nature of the target variable. In the context of disease prediction, models must be capable of capturing complex relationships between symptoms, demographic variables, and health indicators to yield clinically relevant results.

Support Vector Machines (SVM) and k-Nearest Neighbors (kNN) are two prominent algorithms frequently considered for disease prediction tasks. SVM is particularly effective in high-dimensional spaces and is well-suited for cases where the decision boundary is nonlinear. Its ability to handle both linear and nonlinear classification makes it versatile for capturing complex relationships between symptoms and disease outcomes. On the other hand, kNN is a non-parametric algorithm that relies on similarity measures to classify new instances based on the majority class of their k-nearest neighbors. kNN is intuitive, easy to implement, and robust to noisy data, making it suitable for exploratory analysis and initial model prototyping.



After training our K-Nearest Neighbors (K-NN) and Support Vector Machines (SVM) models, we observe that SVM significantly outperforms K-NN in terms of the macro-averaged F1 score.

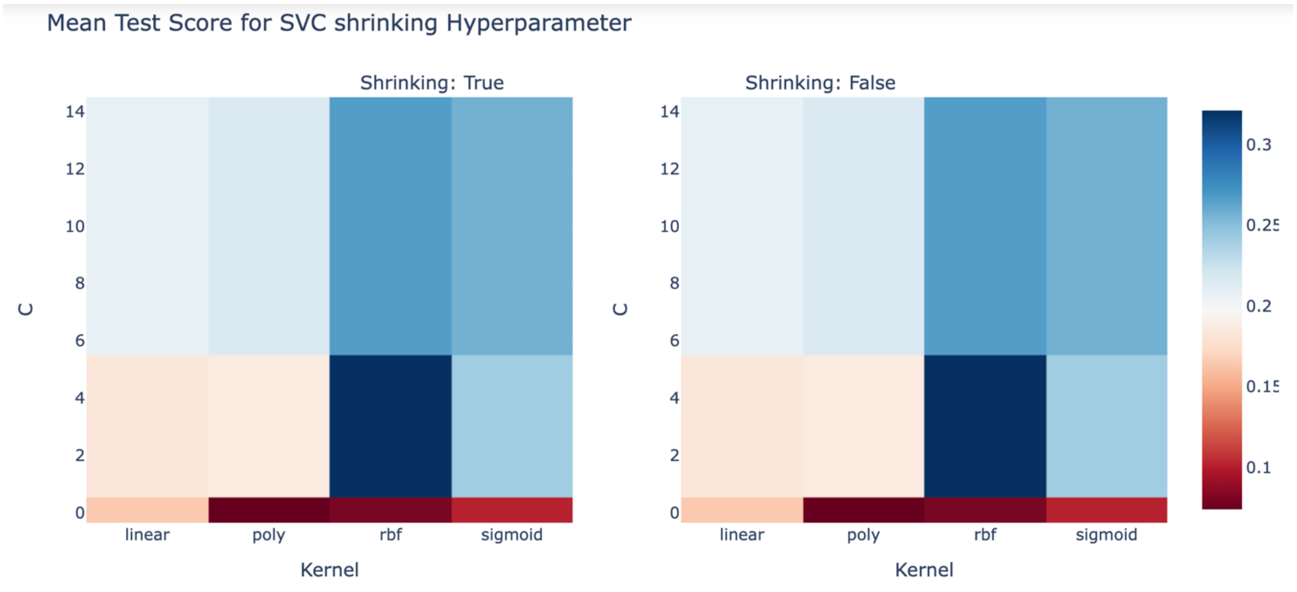
Given this performance difference, it makes sense to focus our efforts on the SVM model. We will proceed with fine-tuning this model to see if we can further improve the F1 score

4. Model Evaluation:

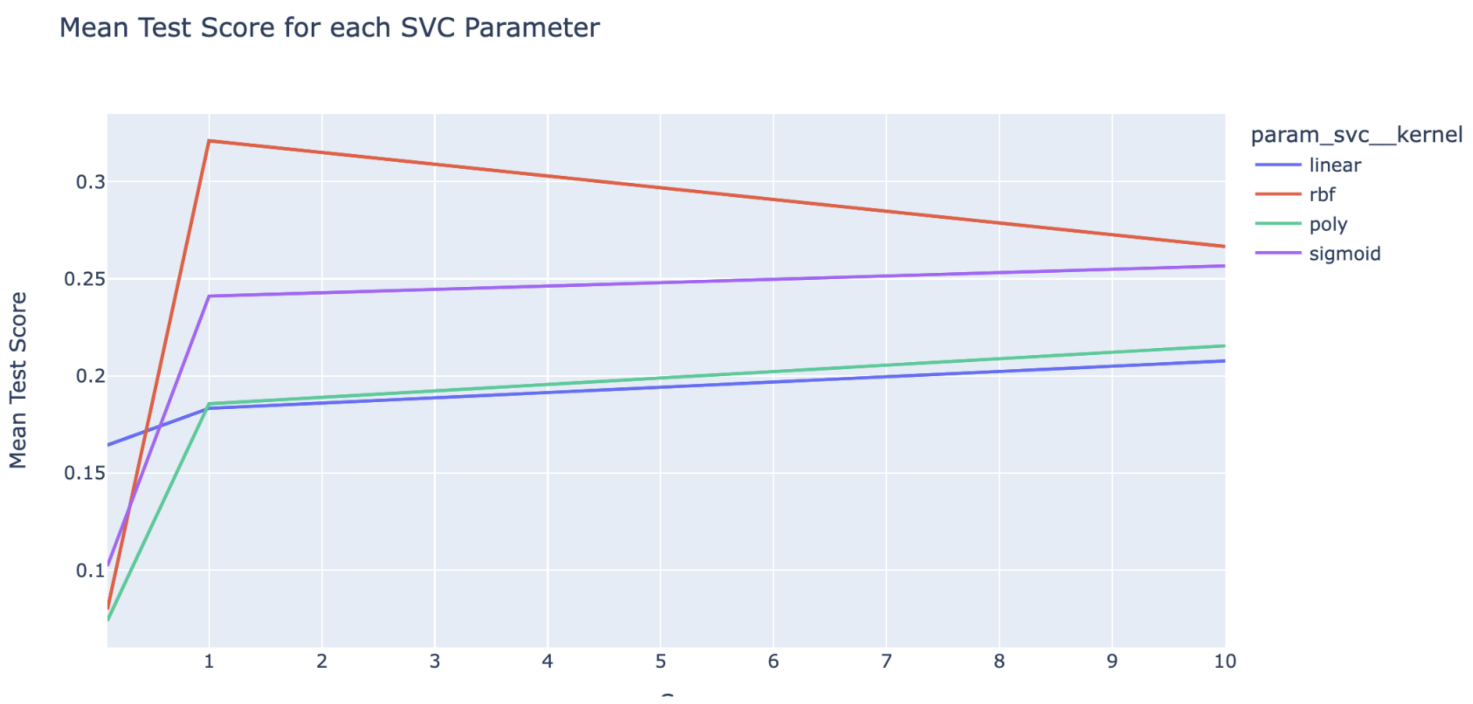
Model evaluation is a critical step in assessing the performance of machine learning algorithms and determining their suitability for real-world applications. Evaluation metrics provide quantitative measures of a model's predictive accuracy, generalization ability, and robustness across different datasets. Commonly used evaluation metrics for classification tasks include accuracy, precision, recall, F1-score, area under the receiver operating characteristic curve (AUC-ROC), and confusion matrix analysis. In the context of disease prediction, it is essential to prioritize metrics that balance sensitivity (true positive rate) and specificity (true negative rate) to minimize false positives and false negatives, ensuring reliable diagnostic accuracy and patient care. Cross-validation techniques such as k-fold cross-validation and stratified cross-validation help estimate a model's performance on unseen data, reducing the risk of overfitting and improving generalization ability.

5. Fine-Tuning:

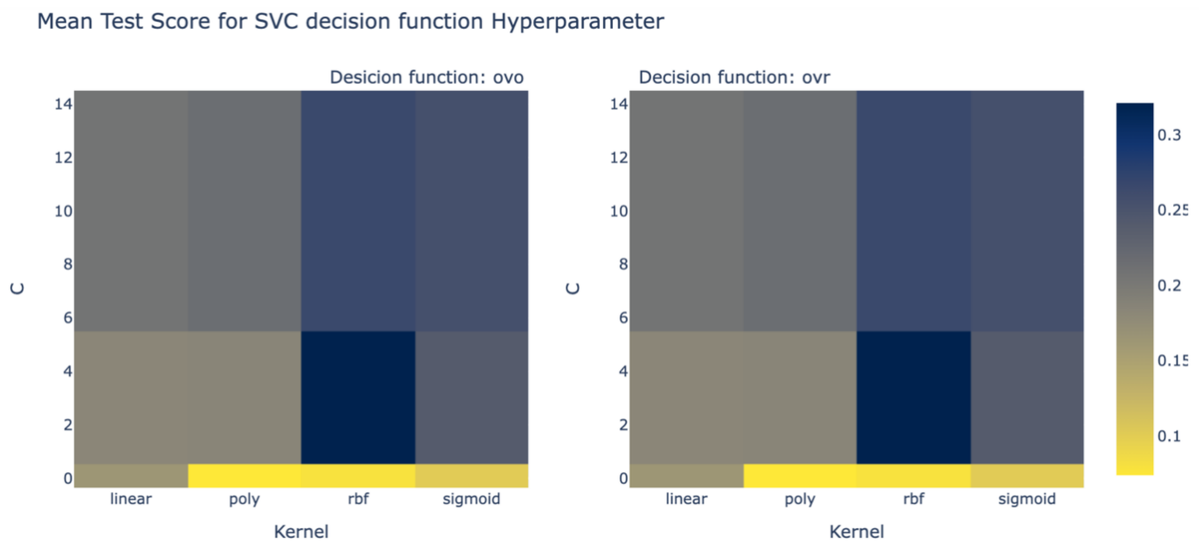
Fine-tuning machine learning models involves optimizing their hyperparameters to enhance performance and maximize predictive accuracy. Hyperparameters control the behavior of algorithms and influence their ability to capture underlying patterns and relationships within the data. Techniques such as grid search, random search, and Bayesian optimization can be employed to systematically search the hyperparameter space and identify optimal configurations that yield the best performance. In the context of disease prediction, fine-tuning models involves optimizing parameters such as regularization strength, kernel type, distance metric, tree depth, and ensemble size to improve classification accuracy, reduce bias and variance, and enhance model robustness. Fine-tuning is an iterative process that requires careful experimentation and validation to ensure the selected hyperparameters generalize well to unseen data and yield clinically actionable insights.



Here we can see that there is no meaningful difference between using the shrinking hyperparameter or not. Here, the kernel and C variants determine the mean test score with no differences, whether the parameter is true or false. Now let's see if the decision function shape makes a difference or not.



Here we can observe a similar result as with the shrinking hyperparameter - there is no effect on the score when we use OVO (One-vs-One) or OVR (One-vs-Rest) for the decision function. Now, let's plot how the score changes when we modify the kernel and the C parameter.



We can observe a significant effect of the kernel and C parameter on the mean test scores. The range of scores varies widely, ranging from 0.06 to 0.255. This emphasizes the importance of these parameters in the SVC (Support Vector Classifier).

Lastly, we will use the best model to predict our test data and report the final results. Additionally, we will examine the samples that were classified incorrectly. Let's proceed with these tasks.

6. Model Testing:

Model testing is the final stage in the machine learning pipeline, where the performance of the selected model is evaluated on an independent test dataset to assess its real-world applicability and generalization ability. The test dataset should be representative of the population of interest and contain samples that were not used during model training or validation to provide an unbiased estimate of performance. Evaluation metrics computed on the test set offer insights into the model's predictive accuracy, robustness, and reliability in clinical settings. Additionally, visualizing model predictions using calibration plots, ROC curves, and precision-recall curves helps assess the model's calibration, discrimination, and overall goodness-of-fit. Model testing serves as a critical quality assurance step, providing confidence in the model's ability to make accurate predictions and inform clinical decision-making in real-world healthcare scenarios.

Incorporating the strengths and characteristics of SVC and kNN into these topics provides stakeholders with a deeper understanding of the specific advantages and considerations associated with these algorithms in the context of disease prediction and healthcare applications.

4. Stakeholder Applications:

Healthcare Professionals: Medical practitioners, researchers, and epidemiologists can leverage the dataset for clinical analysis, epidemiological studies, and public health surveillance efforts. The dataset serves as a valuable tool for understanding disease prevalence, symptomatology, and demographic determinants of health outcomes, thereby informing evidence-based clinical practice guidelines and public health interventions.

Medical Researchers: Researchers focused on specific diseases or conditions mentioned in the dataset can utilize it to generate hypotheses, test scientific theories, and identify novel therapeutic targets. The dataset facilitates hypothesis-driven research and hypothesis-generating exploratory analyses, fostering innovation and discovery in medical science, leading to the development of targeted therapies and precision medicine approaches.

Healthcare Technology Companies: Companies engaged in healthcare innovation, including pharmaceutical firms, medical device manufacturers, and digital health startups, can harness the dataset for model training, validation, and algorithm development. The dataset fuels the development of predictive analytics tools, decision support systems, and precision medicine solutions aimed at improving patient outcomes and healthcare delivery efficiency, thereby driving innovation and competitiveness in the healthcare industry.

5. Conclusion:

The Comprehensive Disease Symptom and Patient Profile Dataset represent a cornerstone resource for unraveling the complexities of disease pathophysiology, symptomatology, and patient heterogeneity. Its multifaceted attributes and broad applicability make it indispensable for various stakeholders seeking to advance medical knowledge, improve patient care, and drive innovation in healthcare delivery. By harnessing the power of data-driven insights and predictive analytics, stakeholders can transform healthcare delivery, enhance patient outcomes, and mitigate the burden of disease on individuals and society.

6. CODE:

```
import pandas as pd
import plotly.express as px
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import f1_score, accuracy_score, recall_score
from sklearn.metrics import make_scorer
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

Data Exploration

```
df = pd.read_csv('Disease_symptom_and_patient_profile_dataset.csv')
df.head()
```

	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
0	Influenza	Yes	No	Yes	Yes	19	Female	Low	Normal	Positive
1	Common Cold	No	Yes	Yes	No	25	Female	Normal	Normal	Negative
2	Eczema	No	Yes	Yes	No	25	Female	Normal	Normal	Negative
3	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
4	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive

```
df.shape
```

```
(349, 10)
```

```
print(sum(df.Disease.value_counts() >= 1))
print(sum(df.Disease.value_counts() == 1))
```

```
116
61
```

```
print(sum(df.Disease.value_counts() > 9))
print(sum(df.Disease.value_counts() <= 9))
```

```
6
110
```

Upon examining the 'Disease' column, we notice a large number of unique diseases, many of which have only 1 to 5 samples. For a reliable disease prediction model, this sample size is insufficient.

Predicting diseases with such limited information could lead to inaccurate results and misdiagnosis, which we want to avoid. Therefore, to ensure the robustness of our model, we will focus only on the diseases that have 10 or more samples.

This decision will reduce the number of classes we are predicting down to 6, making our model more manageable and potentially more accurate

```
df = df[df.groupby('Disease')['Disease'].transform('size') >= 10]
```

```
df
```

	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
3	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
4	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
10	Asthma	Yes	No	No	Yes	28	Male	High	Normal	Positive
14	Diabetes	No	No	No	No	29	Male	Low	Normal	Negative
20	Stroke	Yes	Yes	Yes	Yes	29	Female	Normal	Normal	Positive
...	...	...	...	...	...	...	...	...	...	...
344	Stroke	Yes	No	Yes	No	80	Female	High	High	Positive
345	Stroke	Yes	No	Yes	No	85	Male	High	High	Positive
346	Stroke	Yes	No	Yes	No	85	Male	High	High	Positive
347	Stroke	Yes	No	Yes	No	90	Female	High	High	Positive
348	Stroke	Yes	No	Yes	No	90	Female	High	High	Positive

83 rows × 10 columns

```
df.shape
```

```
(83, 10)
```

Before we proceed with further analysis or model building, it's crucial to ensure the quality of our data. This involves checking for and handling missing values (NaNs) and duplicate entries.

Missing Values: Missing data can lead to misleading results and reduce the statistical power of the model. Therefore, we need to check if our dataset contains any NaN values.

Duplicate Values: Duplicate entries can bias the analysis by over-representing certain observations. Hence, it's important to identify and remove any duplicates in our dataset

```
df.isna().sum()
```

```
Disease                0
Fever                  0
Cough                  0
Fatigue                0
Difficulty Breathing   0
Age                    0
Gender                 0
Blood Pressure         0
Cholesterol Level      0
Outcome Variable       0
dtype: int64
```

```
df.loc[df.duplicated()]
```

	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
4	Asthma	Yes	Yes	No	Yes	25	Male	Normal	Normal	Positive
35	Asthma	Yes	Yes	No	Yes	30	Female	Normal	Normal	Positive
59	Asthma	No	Yes	Yes	Yes	35	Female	High	Normal	Negative
76	Asthma	Yes	Yes	No	Yes	35	Male	Normal	Normal	Positive
123	Asthma	Yes	Yes	No	Yes	40	Female	Normal	Normal	Positive
126	Asthma	Yes	No	Yes	Yes	40	Male	Normal	High	Positive
182	Asthma	Yes	Yes	No	Yes	45	Male	Normal	Normal	Positive
267	Osteoporosis	Yes	No	Yes	No	55	Female	Normal	Normal	Positive
284	Osteoporosis	No	Yes	No	No	60	Male	High	High	Negative
308	Stroke	Yes	No	Yes	No	65	Female	High	Low	Negative
339	Stroke	No	Yes	No	No	70	Male	Normal	High	Positive
344	Stroke	Yes	No	Yes	No	80	Female	High	High	Positive
346	Stroke	Yes	No	Yes	No	85	Male	High	High	Positive
348	Stroke	Yes	No	Yes	No	90	Female	High	High	Positive

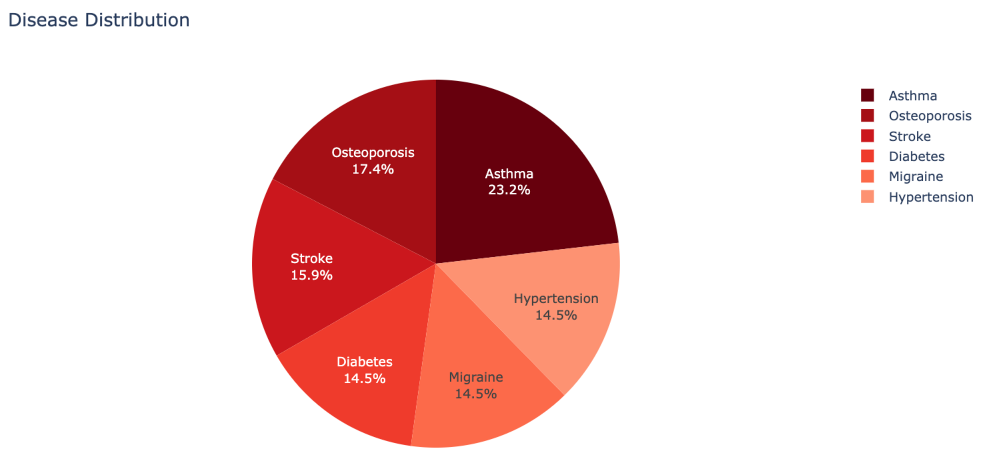
```
df = df.drop_duplicates().reset_index(drop= True)
df.shape
```

```
(69, 10)
```

Now that our data is cleaned and we've narrowed down our focus to diseases with 10 or more samples, let's visualize the distribution of these classes. Understanding the balance of classes is important as it can influence the performance of our machine learning model.

We'll use a pie chart for this purpose. Let's plot this chart and see how balanced our classes are.

```
disease_counts = df['Disease'].value_counts().reset_index()
disease_counts.columns = ['Disease', 'Count']

fig = px.pie(disease_counts,
             values= 'Count',
             names= 'Disease',
             color_discrete_sequence= px.colors.sequential.Reds_r,
             title= 'Disease Distribution')

fig.update_traces(textinfo='percent+label')

fig.show()
```



From the pie chart, it's evident that our classes are imbalanced. Diseases like Hypertension, Diabetes, and Migraine have approximately 1.7 times fewer samples than Asthma. We have to handle class imbalance

But before we proceed with that, let's first process our categorical variables. This will allow us to perform a univariate analysis, which involves the examination of one variable at a time. This analysis can provide valuable insights into the distribution and characteristics of our variables.

```
dicc = {'Yes':1, 'No':0, 'Low':1, 'Normal':2, 'High':3, 'Positive':1, 'Negative':0, 'Male':0, 'Female': 1}
def replace(x, dicc= dicc):
    if x in dicc:
        x = dicc[x]
    return x
df = df.applymap(replace)
df.head()
```

	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
0	Asthma	1	1	0	1	25	0	2	2	1
1	Asthma	1	0	0	1	28	0	3	2	1
2	Diabetes	0	0	0	0	29	0	1	2	0
3	Stroke	1	1	1	1	29	1	2	2	1
4	Migraine	1	0	0	0	30	1	2	2	0

```
df.dtypes
```

```
Disease                   object
Fever                      int64
Cough                      int64
Fatigue                    int64
Difficulty Breathing       int64
Age                        int64
Gender                     int64
Blood Pressure             int64
Cholesterol Level          int64
Outcome Variable           int64
dtype: object
```

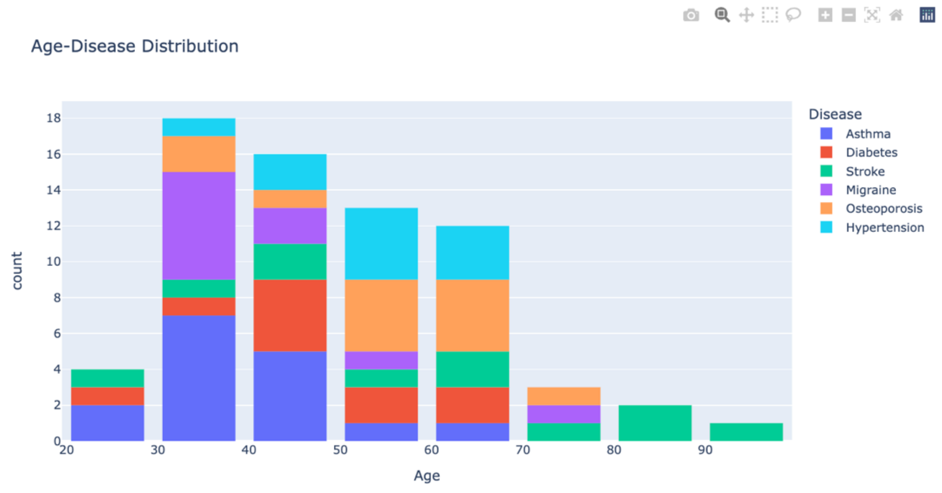
Next, let's examine how these variables correlate with each other. Understanding the relationships between different variables can help us identify patterns and potential multicollinearity, which could influence our model's performance.

We'll use the LabelEncoder from sklearn to convert our categorical variables into numerical format for this correlation analysis.

```
le = LabelEncoder()
df.Disease = le.fit_transform(df.Disease)
df.head()
```

	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
0	0	1	1	0	1	25	0	2	2	1
1	0	1	0	0	1	28	0	3	2	1
2	1	0	0	0	0	29	0	1	2	0
3	5	1	1	1	1	29	1	2	2	1
4	3	1	0	0	0	30	1	2	2	0

```
fig = px.histogram(df,
                x = 'Age',
                title='Age-Disease Distribution',
                color= 'Disease'
                  )
fig.update_layout(bargap=0.2)

fig.show()
```



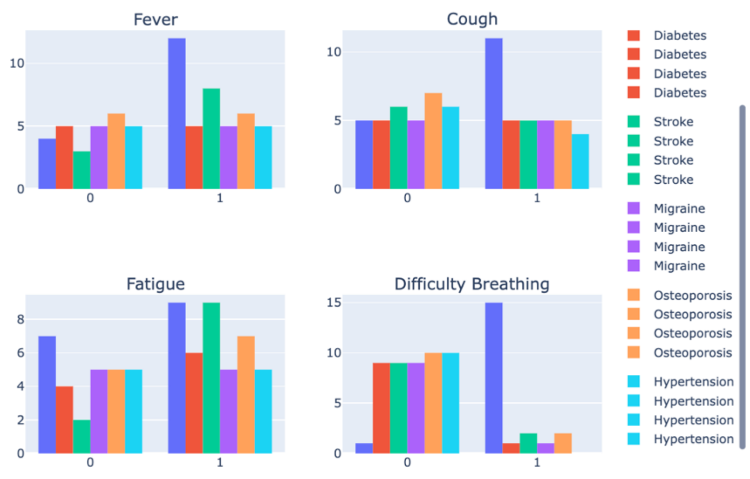
From our univariate analysis of the 'Age' variable, we observe some interesting patterns.

1. If the age is greater than 80, the disease is likely to be a stroke. This aligns with the general understanding that the risk of stroke increases with age.
2. Migraine and Hypertension are not present in ages between 20 and 30. This could suggest that these conditions are more prevalent in older age groups.
3. Hypertension and Osteoporosis appear more frequently as the age increases, indicating a potential correlation between these diseases and age.

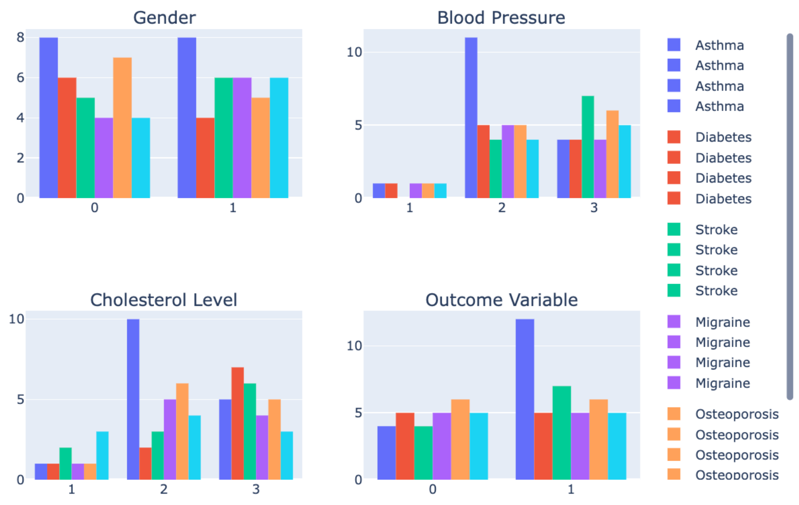
These observations suggest that age is a valuable feature for predicting certain diseases. However, it's important to note that our dataset has limited samples, especially for ages greater than 80. This could make predicting new values in this age range challenging.

Next, let's analyze how the other variables interact with different diseases. This will help us understand their potential as predictors and identify any patterns or correlations.

```
import plotly.subplots as sp

def subplots(df, columns):
    fig = sp.make_subplots(rows=2, cols=2, subplot_titles= columns)
    for idx, column in enumerate(columns):
        i = idx // 2 + 1
        j = idx % 2 + 1
        fig_express = px.histogram(df, x=column, title=column + '-Disease Distribution', color='Disease')
        for trace in fig_express.data:
            fig.add_trace(trace, row=i, col=j)
    fig.update_layout(height=600, width=800, title_text="")
    fig.show()
subplots(df, df.columns[1:5])
```



```
subplots(df, df.columns[6:])
```



Upon visual inspection of the other variables, we can observe significant differences in disease prediction based on each feature's values. For instance, whether a person has high, normal, or low cholesterol levels can significantly influence the prediction of a disease. This is consistent with real-world observations where these variables often vary among different diseases.

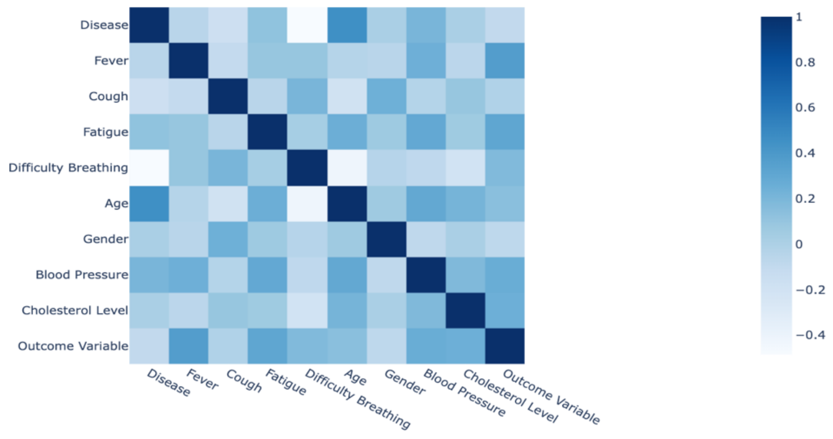
Some valuable insights we can glean from this analysis include:

1. A person with low blood pressure does not have a stroke. This could be a crucial factor in stroke prediction.
2. Fatigue, cholesterol level, and blood pressure are the features that show the most variation among different values. These could potentially be strong predictors in our model.

These observations underscore the importance of these variables in predicting diseases.

Next, let's examine how these variables correlate with each other. Understanding the relationships between different variables can help us identify patterns and potential multicollinearity, which could influence our model's performance.

```
corr_df = df.corr()
fig = px.imshow(corr_df, color_continuous_scale= px.colors.sequential.Blues)
fig.show()
```



From the correlation graph, we can observe that none of the variables strongly correlate with the 'Disease' variable. The most correlated variables are 'Age' and 'Difficulty Breathing', but even these only score 0.4 and -0.4 respectively.

In situations where we have multiple variables with low correlation scores, machine learning can be a viable alternative for prediction tasks. However, it's important to note that machine learning algorithms, especially deep learning ones, typically require large amounts of data to perform optimally.

In our case, we have only 69 data points, which is relatively small. Furthermore, we are dealing with a multi-class problem with few examples for each disease, which adds to the complexity.

Given these constraints, we will try two machine learning algorithms that can perform well without a lot of data: K-Nearest Neighbors (K-NN) and Support Vector Machines (SVM). We will fine-tune these models and select the one that performs best.

Let's proceed with data preprocessing and fit our data into these algorithms.

Model Selection

```
X = df.drop(['Disease'], axis= 1).values
y = df.Disease.values
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size= 0.4, shuffle= True, stratify= y, random_state=30)
X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size= 0.5, shuffle= True, stratify= y_val, random
```

```
svc_pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC(class_weight= 'balanced'))])
knn_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier())])
```

```
svc_pipe.fit(X_train, y_train)
knn_pipe.fit(X_train, y_train)
```

```
▸ Pipeline
  ▸ StandardScaler
▸ KNeighborsClassifier
```

```
ysvc_pred = svc_pipe.predict(X_val)
yknn_pred = knn_pipe.predict(X_val)
```

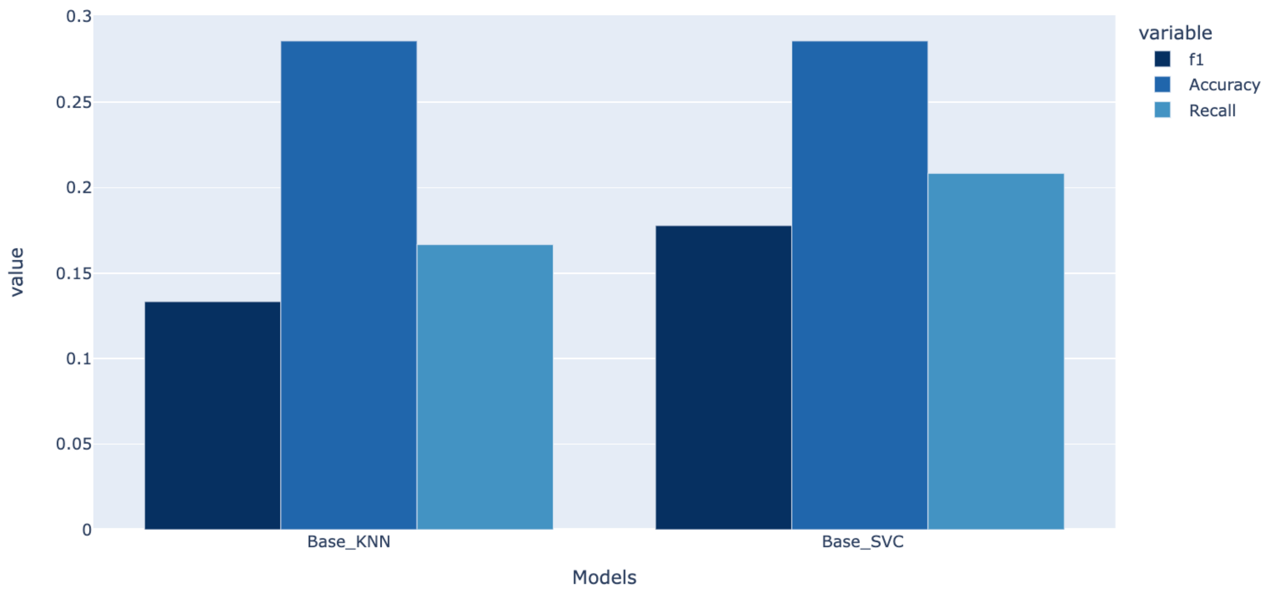
Given our problem of multi-class classification with imbalanced classes, the F1 score (macro-averaged) is an appropriate choice. The F1 score is the harmonic mean of precision and recall, and it gives a better measure of the incorrectly classified cases than the accuracy metric.

The macro-averaged F1 score calculates the F1 score for each class independently and then takes the average. This treats all classes equally, regardless of their imbalance, which is exactly what we need for our problem.

Our goal is to maximize this F1 score.

```
models = ['Base_KNN', ' Base_SVC']

f1 = [f1_score(y_val, yknn_pred, average= 'macro', zero_division= 0), f1_score(y_val, ysvc_pred, average= 'macro', z
accuracy = [accuracy_score(y_val, yknn_pred), accuracy_score(y_val, ysvc_pred)]
recall = [recall_score(y_val, yknn_pred, average= 'macro'), recall_score(y_val, ysvc_pred, average= 'macro')]

metrics_df = pd.DataFrame({'Models': models, 'f1': f1, 'Accuracy': accuracy, 'Recall': recall})
```

```
fig = px.bar(metrics_df, x='Models', y= ['f1', 'Accuracy', 'Recall'], barmode= 'group', color_discrete_sequence= px.
fig.show( )
```



After training our K-Nearest Neighbors (K-NN) and Support Vector Machines (SVM) models, we observe that SVM significantly outperforms K-NN in terms of the macro-averaged F1 score.

Given this performance difference, it makes sense to focus our efforts on the SVM model. We will proceed with fine-tuning this model to see if we can further improve the F1 score

Fine-tuning

```python
f1_scorer = make_scorer(f1_score, average='macro', zero_division=0)
parameters = {
    'svc__C': [0.1, 1, 10],
    'svc__kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'svc__gamma': ['scale', 'auto'],
    'svc__shrinking': [True, False],
    'svc__decision_function_shape': ['ovo', 'ovr']
}

grid_search = GridSearchCV(svc_pipe, parameters, cv=5, scoring= f1_scorer)

grid_search.fit(X_train, y_train)

print("Best Score: ", grid_search.best_score_)
print("Best Params: ", grid_search.best_params_)

best_clf = grid_search.best_estimator_
```

```
Best Score:  0.32111111111111107
Best Params:  {'svc__C': 1, 'svc__decision_function_shape': 'ovo', 'svc__gamma': 'scale', 'svc__kernel': 'rbf', 'sv
c__shrinking': True}
```

```python
results = pd.DataFrame(grid_search.cv_results_)
results = results[['param_svc__C', 'param_svc__kernel', 'param_svc__gamma', 'param_svc__shrinking',
                  'param_svc__decision_function_shape', 'mean_test_score']]
results
```

	param_svc__C	param_svc__kernel	param_svc__gamma	param_svc__shrinking	param_svc__decision_function_shape	mean_test_score
0	0.1	linear	scale	True	ovo	0.164444
1	0.1	linear	scale	False	ovo	0.164444
2	0.1	rbf	scale	True	ovo	0.079899
3	0.1	rbf	scale	False	ovo	0.079899
4	0.1	poly	scale	True	ovo	0.074074
...	...	...	...	...	...	...
91	10	rbf	auto	False	ovr	0.266667
92	10	poly	auto	True	ovr	0.215556
93	10	poly	auto	False	ovr	0.215556
94	10	sigmoid	auto	True	ovr	0.256667
95	10	sigmoid	auto	False	ovr	0.256667

96 rows × 6 columns

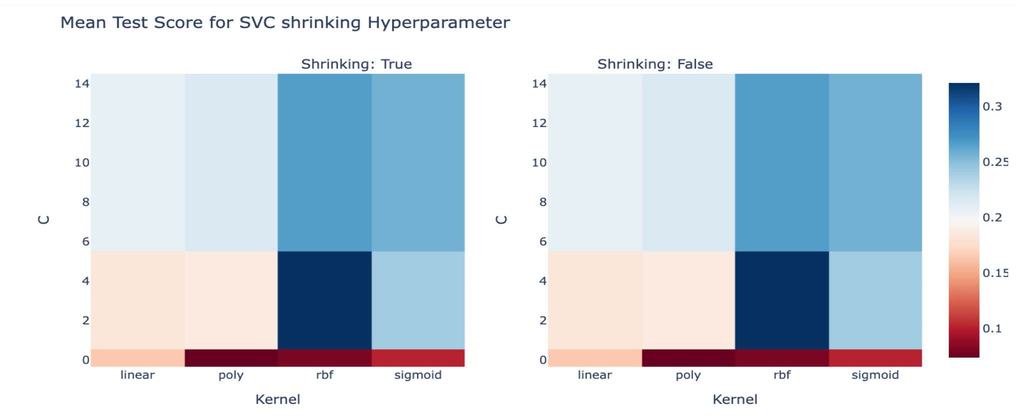
In addition to the kernel and C parameters, we will analyze the significance of other hyperparameters in achieving the highest F1 score. While the kernel and C parameters are known to have a significant impact on SVM performance, it's important to consider the influence of other hyperparameters as well.

We will specifically investigate the impact of hyperparameters such as the shrinking and the decision function shape.

Analyzing the relationship between these hyperparameters and the F1 score will provide us with a more comprehensive understanding of the model's behavior.

Let's proceed with the hyperparameter analysis and determine the optimal values for achieving the highest F1 score

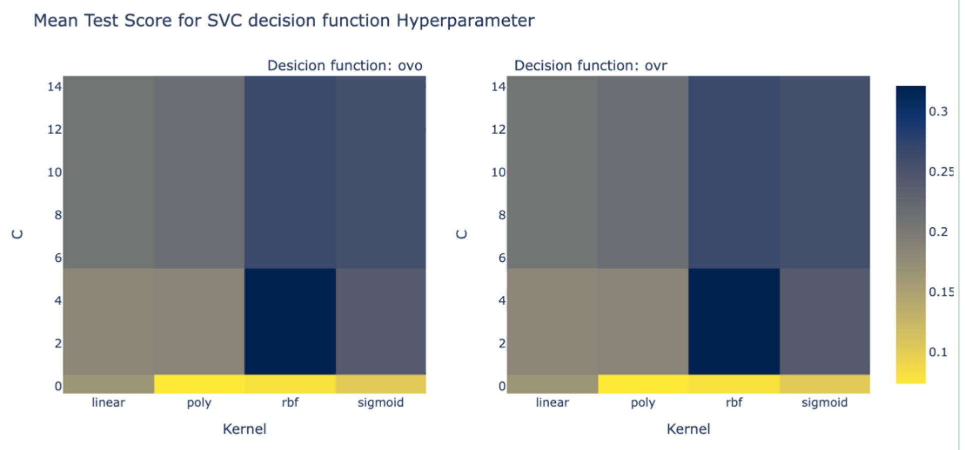
```python
grouped = results.groupby(['param_svc__C', 'param_svc__kernel', 'param_svc__shrinking'])['mean_test_score'].mean().r

pivot_table_true = grouped[grouped['param_svc__shrinking'] == True].pivot(
    'param_svc__C', 'param_svc__kernel', 'mean_test_score')

pivot_table_false = grouped[grouped['param_svc__shrinking'] == False].pivot(
    'param_svc__C', 'param_svc__kernel', 'mean_test_score')

fig = make_subplots(rows=1, cols=2)

fig.add_trace(
    go.Heatmap(z=pivot_table_true.values, x=pivot_table_true.columns,
               y=pivot_table_true.index, colorscale='RdBu', showscale=False),
    row=1, col=1
)

fig.add_trace(
    go.Heatmap(z=pivot_table_false.values, x=pivot_table_false.columns,
               y=pivot_table_false.index, colorscale='RdBu'),
    row=1, col=2
)

fig.update_layout(
    height=500, width=1000, title_text="Mean Test Score for SVC shrinking Hyperparameter",
    annotations=[
        go.layout.Annotation(
            text="Shrinking: True",
            xref="paper", yref="paper",
            x=0.25, y=1.07, showarrow=False,
            font=dict(size=14,)
        ),
        go.layout.Annotation(
            text="Shrinking: False",
            xref="paper", yref="paper",
            x=0.75, y=1.07, showarrow=False,
            font=dict(size=14,)
        )
    ]
)
fig.update_xaxes(title_text="Kernel", row=1, col=1)
fig.update_xaxes(title_text="Kernel", row=1, col=2)
fig.update_yaxes(title_text="C", row=1, col=1)
fig.update_yaxes(title_text="C", row=1, col=2)

fig.show()
```



Here we can see that there is no meaningful difference between using the shrinking hyperparameter or not. Here, the kernel and C variants determine the mean test score with no differences, whether the parameter is true or false. Now let's see if the decision function shape makes a difference or not.

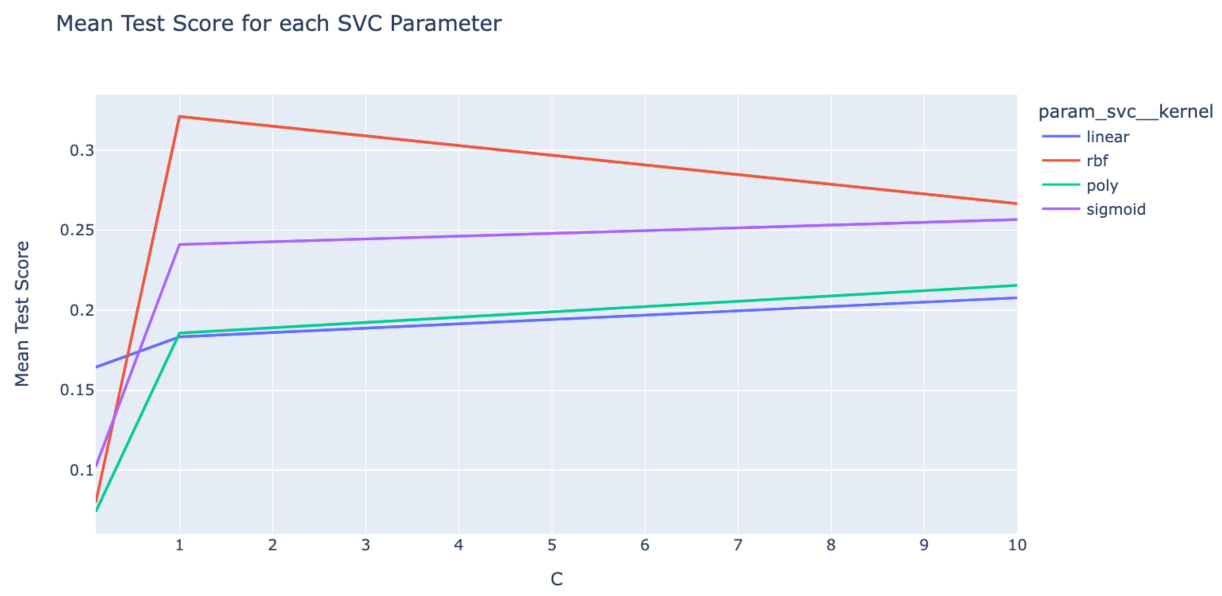
```
grouped = results.groupby(['param_svc__C', 'param_svc__kernel', 'param_svc__decision_function_shape'])['mean_test_sc

pivot_table_true = grouped[grouped['param_svc__decision_function_shape'] == 'ovo'].pivot(
    'param_svc__C', 'param_svc__kernel', 'mean_test_score')

pivot_table_false = grouped[grouped['param_svc__decision_function_shape'] == 'ovr'].pivot(
    'param_svc__C', 'param_svc__kernel', 'mean_test_score')

fig = make_subplots(rows=1, cols=2)

fig.add_trace(
    go.Heatmap(z=pivot_table_true.values, x=pivot_table_true.columns,
               y=pivot_table_true.index, colorscale= px.colors.sequential.Cividis_r, showscale=False),
    row=1, col=1
)

fig.add_trace(
    go.Heatmap(z=pivot_table_false.values, x=pivot_table_false.columns,
               y=pivot_table_false.index, colorscale= px.colors.sequential.Cividis_r),
    row=1, col=2
)
```

```
fig.update_layout(
    height=500, width=1000, title_text="Mean Test Score for SVC decision function Hyperparameter",
    annotations=[
        go.layout.Annotation(
            text="Desicion function: ovo",
            xref="paper", yref="paper",
            x=0.25, y=1.07, showarrow=False,
            font=dict(size=14,)
        ),
        go.layout.Annotation(
            text="Decision function: ovr",
            xref="paper", yref="paper",
            x=0.75, y=1.07, showarrow=False,
            font=dict(size=14,)
        )
    ]
)
fig.update_xaxes(title_text="Kernel", row=1, col=1)
fig.update_xaxes(title_text="Kernel", row=1, col=2)
fig.update_yaxes(title_text="C", row=1, col=1)
fig.update_yaxes(title_text="C", row=1, col=2)

fig.show()
```



Here we can observe a similar result as with the shrinking hyperparameter - there is no effect on the score when we use OVO (One-vs-One) or OVR (One-vs-Rest) for the decision function. Now, let's plot how the score changes when we modify the kernel and the C parameter.

```
fig = px.line(results, x="param_svc__C", y="mean_test_score",
              color="param_svc__kernel",
              line_group="param_svc__shrinking",
              hover_name="param_svc__decision_function_shape",
              labels={"mean_test_score": "Mean Test Score", "param_svc__C": "C"},
              title="Mean Test Score for each SVC Parameter")
fig.show()
```



We can observe a significant effect of the kernel and C parameter on the mean test scores. The range of scores varies widely, ranging from 0.06 to 0.255. This emphasizes the importance of these parameters in the SVC (Support Vector Classifier).

Lastly, we will use the best model to predict our test data and report the final results. Additionally, we will examine the samples that were classified incorrectly. Let's proceed with these tasks.

Model Testing

```
y_pred_test = best_clf.predict(X_test)
print('Test score with best model: ', f1_score(y_test, y_pred_test, average= 'macro', zero_division= 0))
```

```
Test score with best model:  0.7611111111111111
```