Name: Betül Biçer

ID: 201101055

Course: 470 Proje: ATP Tenis Maçı Tahmini

LIBRARIES

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, precision_s
from sklearn.preprocessing import label_binarize

from itertools import cycle
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

In [ ]:
```
import pandas as pd
import numpy as np
import os
import glob
import matplotlib.pyplot as plt
import seaborn as sns
import shap
from sklearn.tree import DecisionTreeClassifier

shap.initjs()
```

```
IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.re
adthedocs.io/en/stable/user_install.html
```

Exploratory Data Analysis (EDA)

In [ ]:
```
data_dir = "tennis_atp/"
csv_files = glob.glob(os.path.join(data_dir, "atp_matches_2*.csv"))#directorydeki b
dataframes = [pd.read_csv(file) for file in csv_files]
d = pd.concat(dataframes, ignore_index=True)
```

In [ ]:
```
pd.set_option('display.max_columns', None)
d.info(), d.head()
```

```
print(d.describe())
d
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73247 entries, 0 to 73246
Data columns (total 49 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   tourney_id          73247 non-null  object
 1   tourney_name        73247 non-null  object
 2   surface             73194 non-null  object
 3   draw_size           73247 non-null  int64
 4   tourney_level       73247 non-null  object
 5   tourney_date        73247 non-null  int64
 6   match_num           73247 non-null  int64
 7   winner_id           73247 non-null  int64
 8   winner_seed         30444 non-null  object
 9   winner_entry        9229 non-null   object
 10  winner_name         73247 non-null  object
 11  winner_hand         73240 non-null  object
 12  winner_ht           71351 non-null  float64
 13  winner_ioc          73247 non-null  object
 14  winner_age          73240 non-null  float64
 15  loser_id            73247 non-null  int64
 16  loser_seed          16858 non-null  object
 17  loser_entry         15007 non-null  object
 18  loser_name          73247 non-null  object
 19  loser_hand          73205 non-null  object
 20  loser_ht            69626 non-null  float64
 21  loser_ioc           73247 non-null  object
 22  loser_age           73235 non-null  float64
 23  score               73247 non-null  object
 24  best_of             73247 non-null  int64
 25  round               73247 non-null  object
 26  minutes             65309 non-null  float64
 27  w_ace               66778 non-null  float64
 28  w_df                66778 non-null  float64
 29  w_svpt              66778 non-null  float64
 30  w_1stIn             66778 non-null  float64
 31  w_1stWon            66778 non-null  float64
 32  w_2ndWon            66778 non-null  float64
 33  w_SvGms             66779 non-null  float64
 34  w_bpSaved           66778 non-null  float64
 35  w_bpFaced           66778 non-null  float64
 36  l_ace               66778 non-null  float64
 37  l_df                66778 non-null  float64
 38  l_svpt              66778 non-null  float64
 39  l_1stIn             66778 non-null  float64
 40  l_1stWon            66778 non-null  float64
 41  l_2ndWon            66778 non-null  float64
 42  l_SvGms             66779 non-null  float64
 43  l_bpSaved           66778 non-null  float64
 44  l_bpFaced           66778 non-null  float64
 45  winner_rank         72681 non-null  float64
 46  winner_rank_points  72681 non-null  float64
 47  loser_rank          71794 non-null  float64
 48  loser_rank_points   71794 non-null  float64
dtypes: float64(27), int64(6), object(16)
memory usage: 27.4+ MB
```

	draw_size	tourney_date	match_num	winner_id	winner_ht	\
count	73247.000000	7.324700e+04	73247.000000	73247.000000	71351.000000	
mean	55.236146	2.011267e+07	98.603779	109981.878575	186.174153	
std	40.250977	7.068866e+04	131.327812	20777.643103	6.800307	
min	2.000000	2.000010e+07	1.000000	100644.000000	163.000000	
25%	32.000000	2.005070e+07	11.000000	103507.000000	183.000000	
50%	32.000000	2.011050e+07	29.000000	104433.000000	185.000000	
75%	64.000000	2.017070e+07	201.000000	105379.000000	190.000000	
max	128.000000	2.024052e+07	1701.000000	212721.000000	211.000000	

	winner_age	loser_id	loser_ht	loser_age	best_of	\
count	73240.000000	73247.000000	69626.000000	73235.000000	73247.000000	
mean	26.286645	110012.437970	185.626404	26.395834	3.448783	
std	3.969063	20740.605653	6.757514	4.082390	0.834368	
min	14.900000	100644.000000	163.000000	14.500000	3.000000	
25%	23.400000	103470.000000	183.000000	23.400000	3.000000	
50%	26.100000	104417.000000	185.000000	26.200000	3.000000	
75%	29.000000	105550.000000	190.000000	29.200000	3.000000	
max	43.600000	212722.000000	211.000000	46.000000	5.000000	

	minutes	w_ace	w_df	w_svpt	w_1stIn	\
count	65309.000000	66778.000000	66778.000000	66778.000000	66778.000000	
mean	107.038448	6.901674	2.634925	77.900012	47.963775	
std	41.298295	5.512144	2.282820	29.140833	18.911849	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	77.000000	3.000000	1.000000	56.000000	34.000000	
50%	99.000000	6.000000	2.000000	73.000000	45.000000	
75%	130.000000	9.000000	4.000000	94.000000	58.000000	
max	1146.000000	113.000000	26.000000	491.000000	361.000000	

	w_1stWon	w_2ndWon	w_SvGms	w_bpSaved	w_bpFaced	\
count	66778.000000	66778.000000	66779.000000	66778.000000	66778.000000	
mean	36.291653	16.597742	12.503437	3.460631	5.028947	
std	13.548832	6.959677	4.216202	3.072624	4.024405	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	27.000000	12.000000	9.000000	1.000000	2.000000	
50%	34.000000	16.000000	11.000000	3.000000	4.000000	
75%	43.000000	20.000000	15.000000	5.000000	7.000000	
max	292.000000	82.000000	90.000000	24.000000	30.000000	

	l_ace	l_df	l_svpt	l_1stIn	l_1stWon	\
count	66778.000000	66778.000000	66778.000000	66778.000000	66778.000000	
mean	5.110156	3.353829	80.927012	48.546872	32.396493	
std	4.873529	2.527758	29.111572	19.180063	14.347089	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	2.000000	60.000000	35.000000	22.000000	
50%	4.000000	3.000000	76.000000	45.000000	30.000000	
75%	7.000000	5.000000	97.000000	59.000000	40.000000	
max	103.000000	26.000000	489.000000	328.000000	284.000000	

	l_2ndWon	l_SvGms	l_bpSaved	l_bpFaced	winner_rank	\
count	66778.000000	66779.000000	66778.000000	66778.000000	72681.000000	
mean	14.923867	12.294284	4.769235	8.605304	79.611645	
std	7.181560	4.216964	3.268548	4.139202	138.857016	
min	0.000000	0.000000	0.000000	0.000000	1.000000	
25%	10.000000	9.000000	2.000000	6.000000	18.000000	

50%	14.000000	11.000000	4.000000	8.000000	45.000000
75%	19.000000	15.000000	7.000000	11.000000	85.000000
max	101.000000	91.000000	27.000000	38.000000	2101.000000

	winner_rank_points	loser_rank	loser_rank_points
count	72681.000000	71794.000000	71794.000000
mean	1599.348894	117.905507	971.456152
std	1998.955200	186.296583	1120.245003
min	1.000000	1.000000	1.000000
25%	575.000000	36.000000	429.000000
50%	935.000000	68.000000	707.000000
75%	1721.000000	114.000000	1100.000000
max	16950.000000	2159.000000	16950.000000

Out[ ]:

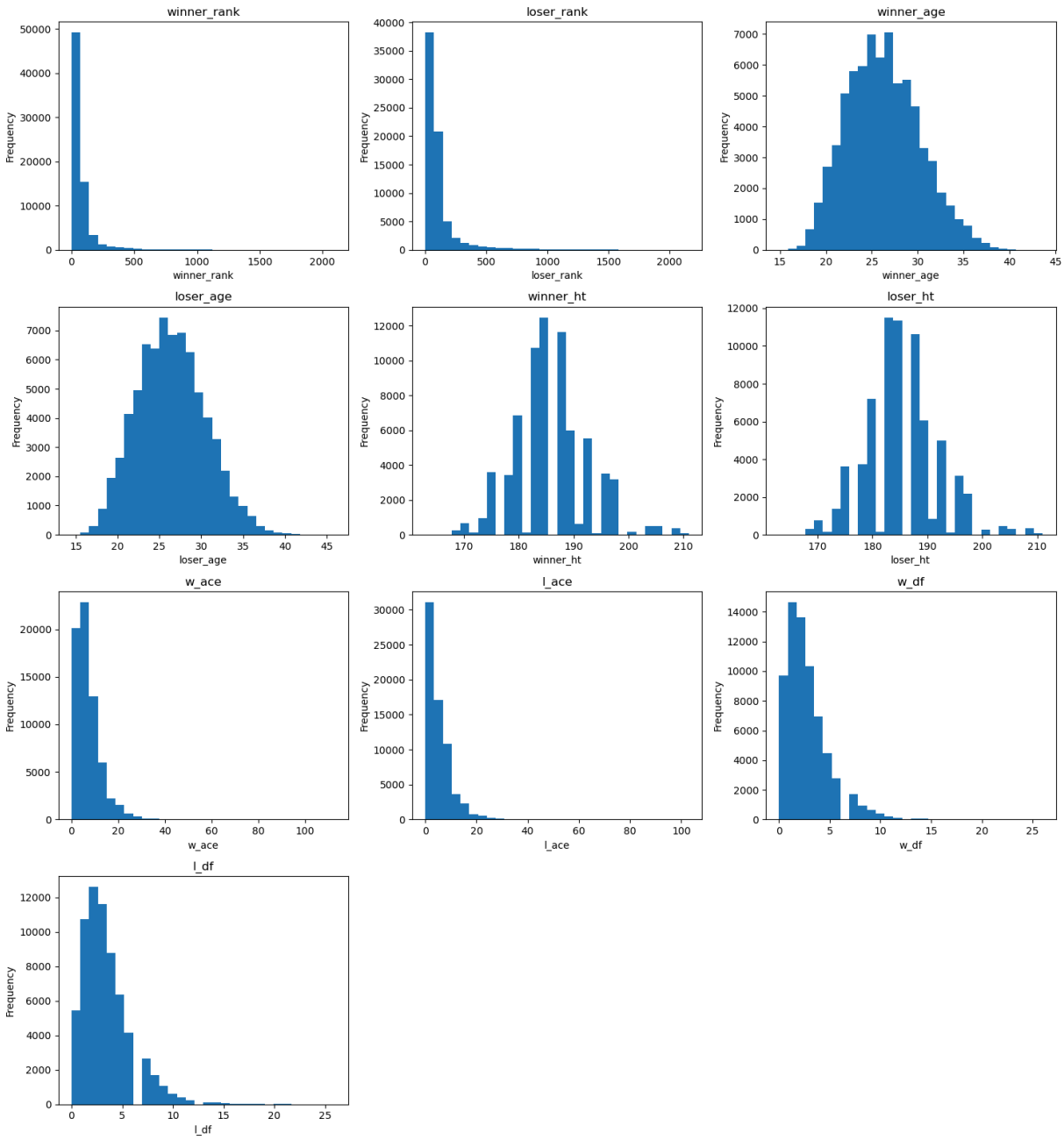
	tourney_id	tourney_name	surface	draw_size	tourney_level	tourney_date	match_
0	2000-301	Auckland	Hard	32	A	20000110	
1	2000-301	Auckland	Hard	32	A	20000110	
2	2000-301	Auckland	Hard	32	A	20000110	
3	2000-301	Auckland	Hard	32	A	20000110	
4	2000-301	Auckland	Hard	32	A	20000110	
...	...	...	...	...	...	...	
73242	2024-M-DC-2024-WG2-PO-URU-MDA-01	Davis Cup WG2 PO: URU vs MDA	Clay	4	D	20240203	
73243	2024-M-DC-2024-WG2-PO-VIE-RSA-01	Davis Cup WG2 PO: VIE vs RSA	Hard	4	D	20240202	
73244	2024-M-DC-2024-WG2-PO-VIE-RSA-01	Davis Cup WG2 PO: VIE vs RSA	Hard	4	D	20240202	
73245	2024-M-DC-2024-WG2-PO-VIE-RSA-01	Davis Cup WG2 PO: VIE vs RSA	Hard	4	D	20240202	
73246	2024-M-DC-2024-WG2-PO-VIE-RSA-01	Davis Cup WG2 PO: VIE vs RSA	Hard	4	D	20240202	

73247 rows × 49 columns

In [ ]:

```
features = ['winner_rank', 'loser_rank', 'winner_age', 'loser_age', 'winner_ht', 'l
num_plots = len(features)
```

```
cols = 3
rows = (num_plots // cols) + (num_plots % cols > 0)

plt.figure(figsize=(cols * 5, rows * 4))

for i, feature in enumerate(features):
    plt.subplot(rows, cols, i + 1)
    d[feature].plot(kind='hist', bins=30, title=feature)
    plt.xlabel(feature)

plt.tight_layout()
plt.show()
```



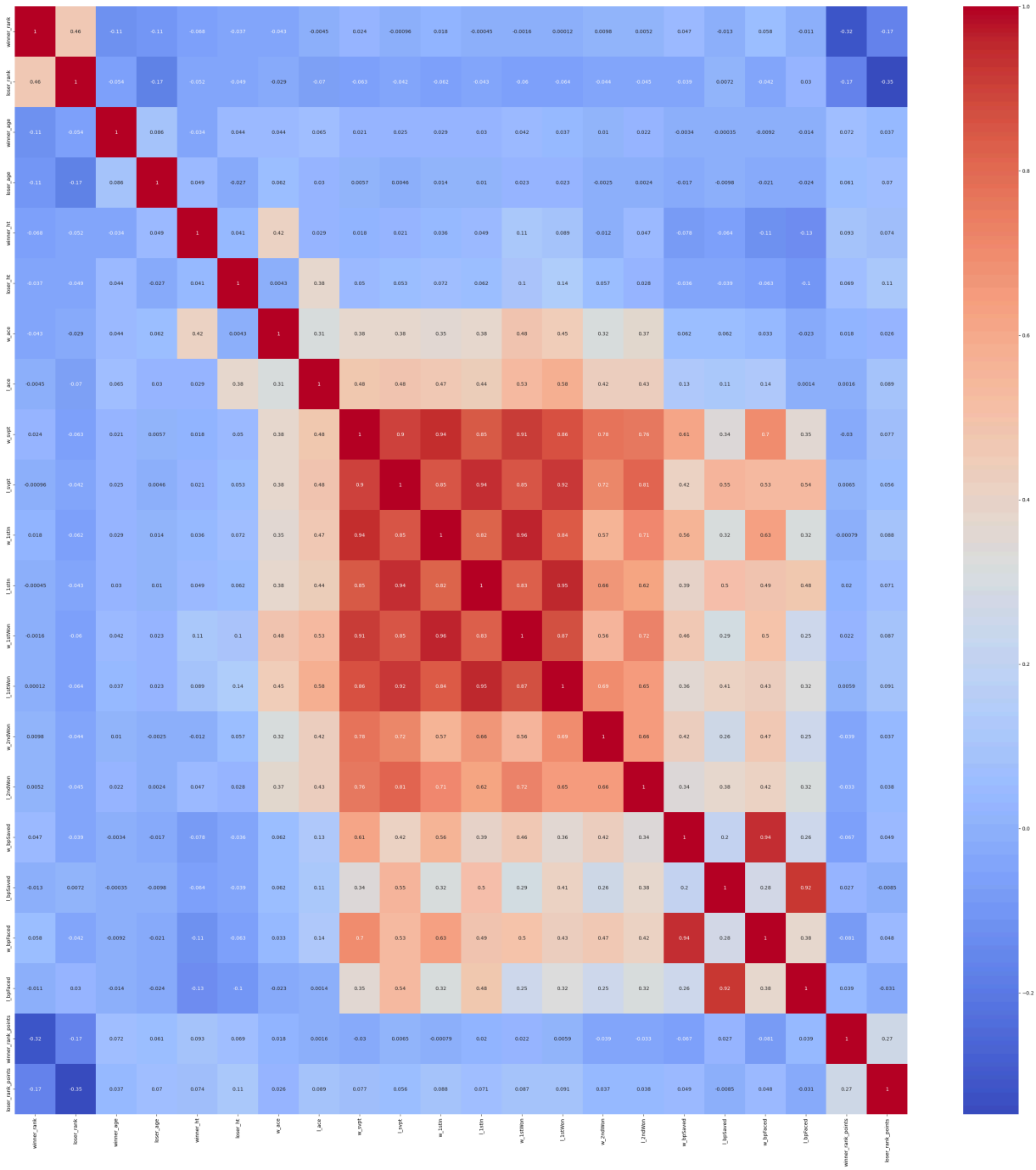
In [ ]:
```
categorical_features = ['surface', 'tourney_level', 'winner_hand', 'loser_hand']
for feature in categorical_features:
    print(f"{feature} degeri:")
```

```
    print(d[feature].value_counts())
    print("\n")
```

```
surface degeri:
surface
Hard      39893
Clay      23887
Grass      7375
Carpet     2039
Name: count, dtype: int64


tourney_level degeri:
tourney_level
A    40068
M    13738
G    12192
D     6842
F      407
Name: count, dtype: int64


winner_hand degeri:
winner_hand
R    63777
L     8962
U      495
A        6
Name: count, dtype: int64


loser_hand degeri:
loser_hand
R    62387
L     9633
U     1180
A        5
Name: count, dtype: int64
```

In [ ]:

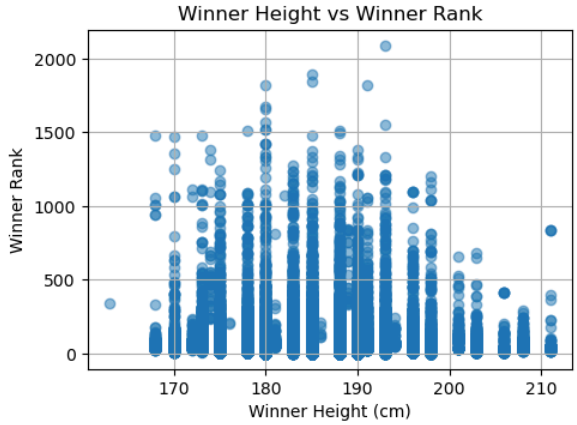
```
predictors = [
    'winner_rank', 'loser_rank', 'winner_age', 'loser_age', 'winner_ht', 'loser_ht'
    'w_ace', 'l_ace', 'w_svpt', 'l_svpt', 'w_1stIn', 'l_1stIn', 'w_1stWon', 'l_1stW
    'w_2ndWon', 'l_2ndWon', 'w_bpSaved', 'l_bpSaved', 'w_bpFaced', 'l_bpFaced',
    'winner_rank_points', 'loser_rank_points',
]

corr_matrix = d[predictors].corr()

plt.figure(figsize=(40, 40))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

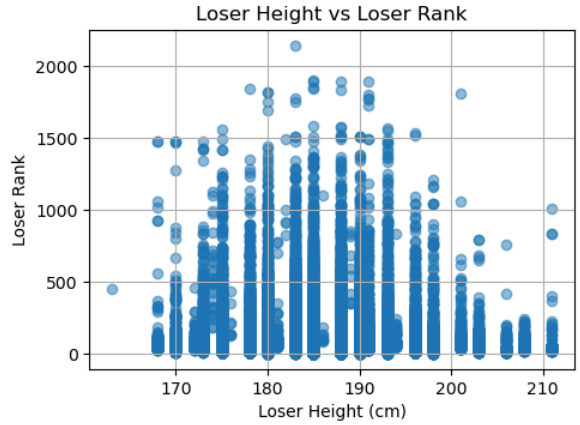


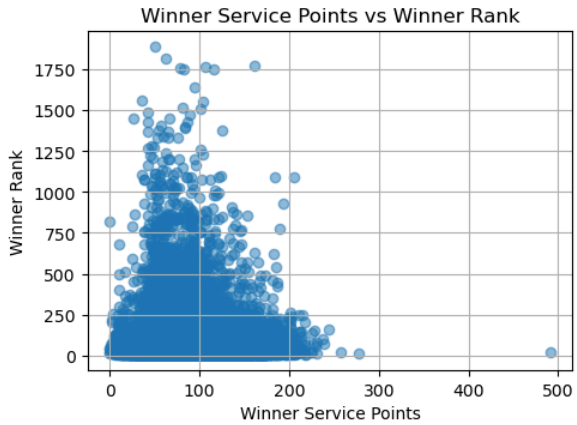
In [ ]:

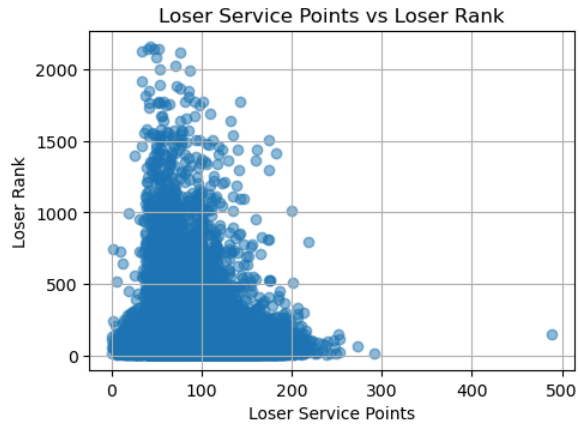
```
fig, axs = plt.subplots(4, 2, figsize=(12, 18))
fig.subplots_adjust(hspace=0.3, wspace=0.3)

axs[0, 0].scatter(d['winner_ht'], d['winner_rank'], alpha=0.5)
axs[0, 0].set_title('Winner Height vs Winner Rank')
axs[0, 0].set_xlabel('Winner Height (cm)')
axs[0, 0].set_ylabel('Winner Rank')
axs[0, 0].grid(True)

axs[0, 1].scatter(d['loser_ht'], d['loser_rank'], alpha=0.5)
axs[0, 1].set_title('Loser Height vs Loser Rank')
axs[0, 1].set_xlabel('Loser Height (cm)')
axs[0, 1].set_ylabel('Loser Rank')
axs[0, 1].grid(True)
```

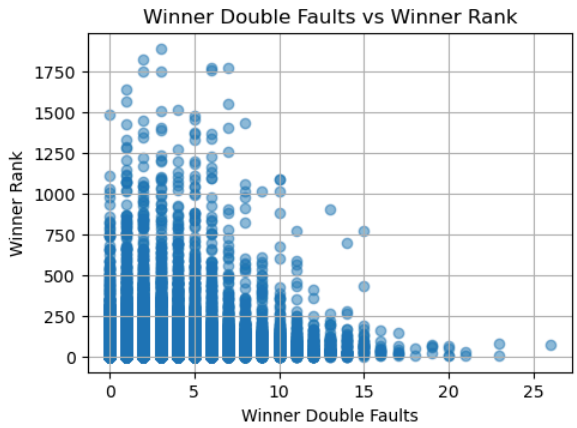
```
axs[1, 0].scatter(d['w_svpt'], d['winner_rank'], alpha=0.5)
axs[1, 0].set_title('Winner Service Points vs Winner Rank')
axs[1, 0].set_xlabel('Winner Service Points')
axs[1, 0].set_ylabel('Winner Rank')
axs[1, 0].grid(True)

axs[1, 1].scatter(d['l_svpt'], d['loser_rank'], alpha=0.5)
axs[1, 1].set_title('Loser Service Points vs Loser Rank')
axs[1, 1].set_xlabel('Loser Service Points')
axs[1, 1].set_ylabel('Loser Rank')
axs[1, 1].grid(True)

axs[2, 0].scatter(d['w_df'], d['winner_rank'], alpha=0.5)
axs[2, 0].set_title('Winner Double Faults vs Winner Rank')
axs[2, 0].set_xlabel('Winner Double Faults')
axs[2, 0].set_ylabel('Winner Rank')
axs[2, 0].grid(True)

axs[2, 1].scatter(d['l_df'], d['loser_rank'], alpha=0.5)
axs[2, 1].set_title('Loser Double Faults vs Loser Rank')
axs[2, 1].set_xlabel('Loser Double Faults')
axs[2, 1].set_ylabel('Loser Rank')
axs[2, 1].grid(True)

axs[3, 0].scatter(d['winner_age'], d['winner_rank'], alpha=0.5)
axs[3, 0].set_title('Winner Age vs Winner Rank')
axs[3, 0].set_xlabel('Winer Age')
axs[3, 0].set_ylabel('Winner Rank')
axs[3, 0].grid(True)

axs[3, 1].scatter(d['loser_age'], d['loser_rank'], alpha=0.5)
axs[3, 1].set_title('Loser Age vs Loser Rank')
axs[3, 1].set_xlabel('Loser Age')
axs[3, 1].set_ylabel('Loser Rank')
axs[3, 1].grid(True)

plt.show()
```

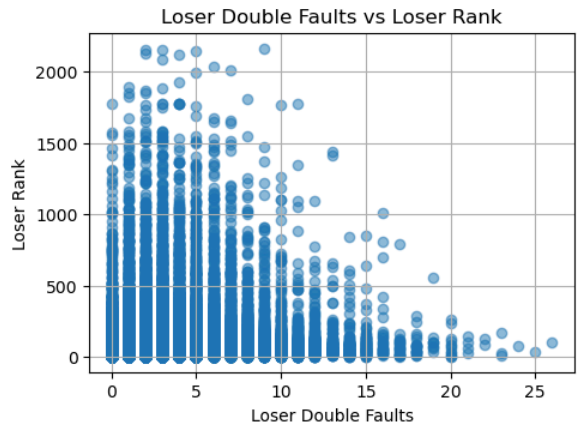


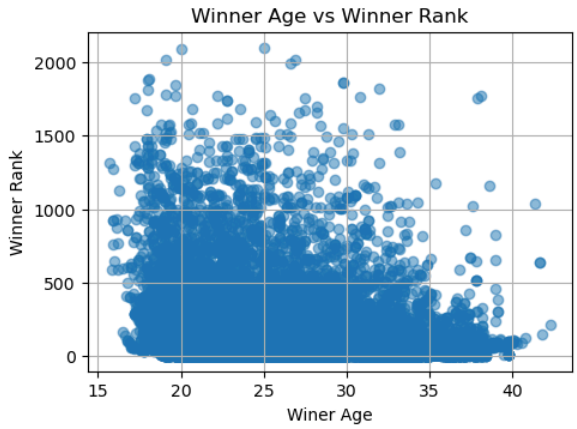












Preprocessing

In [ ]:
```
predictors = [
    'winner_id','loser_id','winner_rank', 'loser_rank', 'winner_age', 'loser_age',
    'w_ace', 'l_ace', 'w_svpt', 'l_svpt', 'w_1stIn', 'l_1stIn', 'w_1stWon', 'l_1stW
    'w_2ndWon', 'l_2ndWon', 'w_bpSaved', 'l_bpSaved', 'w_bpFaced', 'l_bpFaced','w_d
]

d = d[predictors]

print(d.columns)
```

```
Index(['winner_id', 'loser_id', 'winner_rank', 'loser_rank', 'winner_age',
       'loser_age', 'winner_ht', 'loser_ht', 'w_ace', 'l_ace', 'w_svpt',
       'l_svpt', 'w_1stIn', 'l_1stIn', 'w_1stWon', 'l_1stWon', 'w_2ndWon',
       'l_2ndWon', 'w_bpSaved', 'l_bpSaved', 'w_bpFaced', 'l_bpFaced', 'w_df',
       'l_df', 'winner_hand', 'loser_hand', 'surface', 'tourney_date'],
      dtype='object')
```

In [ ]:
```
print('Null degerlerin kontrolu:')
print(d.isnull().sum())


d['winner_age'].fillna(d['winner_age'].median(), inplace=True)
d['loser_age'].fillna(d['loser_age'].median(), inplace=True)


d['winner_ht'].fillna(d['winner_ht'].median(), inplace=True)
d['loser_ht'].fillna(d['loser_ht'].median(), inplace=True)


match_stats = ['w_ace', 'l_ace', 'w_df', 'l_df', 'w_svpt', 'l_svpt', 'w_1stIn', 'l_

for stat in match_stats:
    d[stat] = d.groupby('winner_id')[stat].transform(lambda x: x.fillna(x.median())

for stat in ['l_ace', 'l_df', 'l_svpt', 'l_1stIn', 'l_1stWon', 'l_2ndWon', 'l_bpSav
    d[stat] = d.groupby('loser_id')[stat].transform(lambda x: x.fillna(x.median()))

print('null degerlerini oyuncu bazinda mean veya medianla dolduruldaktan sonra 0 mı
print(d.isnull().sum())

match_stats = ['w_ace', 'l_ace', 'w_df', 'l_df', 'w_svpt', 'l_svpt', 'w_1stIn', 'l_
for stat in match_stats:
    d[stat] = d[stat].fillna(d[stat].median())



d['winner_rank'] = d['winner_rank'].fillna(d['winner_rank'].median())
d['loser_rank'] = d['loser_rank'].fillna(d['loser_rank'].median())

print('null degerlerin genel mean veya medianla dolduruldaktan sonra 0 mı kontrolu:
print(d.isnull().sum())
```

```
Null degerlerin kontrolu:
winner_id        0
loser_id         0
winner_rank      0
loser_rank       0
winner_age       0
loser_age        0
winner_ht        0
loser_ht         0
w_ace            0
l_ace            0
w_svpt           0
l_svpt           0
w_1stIn          0
l_1stIn          0
w_1stWon         0
l_1stWon         0
w_2ndWon         0
l_2ndWon         0
w_bpSaved        0
l_bpSaved        0
w_bpFaced        0
l_bpFaced        0
w_df             0
l_df             0
winner_hand      0
loser_hand       0
surface          0
tourney_year     0
tourney_month    0
dtype: int64
```

```
A value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


A value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


A value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


A value is trying to be set on a copy of a DataFrame or Series through chained assig
nment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.
```

```
null degerlerini oyuncu bazinda mean veya medianla dolduruldaktan sonra 0 mı kontrol
u:
winner_id        0
loser_id         0
winner_rank      0
loser_rank       0
winner_age       0
loser_age        0
winner_ht        0
loser_ht         0
w_ace            0
l_ace            0
w_svpt           0
l_svpt           0
w_1stIn          0
l_1stIn          0
w_1stWon         0
l_1stWon         0
w_2ndWon         0
l_2ndWon         0
w_bpSaved        0
l_bpSaved        0
w_bpFaced        0
l_bpFaced        0
w_df             0
l_df             0
winner_hand      0
loser_hand       0
surface          0
tourney_year     0
tourney_month    0
dtype: int64
null degerlerin genel mean veya medianla dolduruldaktan sonra 0 mı kontrolu:
winner_id        0
loser_id         0
winner_rank      0
loser_rank       0
winner_age       0
loser_age        0
winner_ht        0
loser_ht         0
w_ace            0
l_ace            0
w_svpt           0
l_svpt           0
w_1stIn          0
l_1stIn          0
w_1stWon         0
l_1stWon         0
w_2ndWon         0
l_2ndWon         0
w_bpSaved        0
l_bpSaved        0
w_bpFaced        0
l_bpFaced        0
w_df             0
```

```
l_df             0
winner_hand      0
loser_hand       0
surface          0
tourney_year     0
tourney_month    0
dtype: int64
```

In [ ]:
```
d['winner_hand'] = d['winner_hand'].fillna('U')
d['loser_hand'] = d['loser_hand'].fillna('U')
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]:
```
original_size = len(d)
print(f"Data orjinal size: {original_size}")

d = d.dropna(subset=['surface'])

new_size = len(d)
print(f"Surface feature null degerleri cikarinca: {new_size}")
print(f"Droplanan row sayisi: {original_size - new_size}")
```

```
Data orjinal size: 73247
Surface feature null degerleri cikarinca: 73194
Droplanan row sayisi: 53
```

In [ ]:
```
numeric_columns = ['winner_rank', 'loser_rank', 'winner_age', 'loser_age', 'winner_
                   'loser_ht','w_ace', 'l_ace', 'w_svpt', 'l_svpt', 'w_1stIn', 'l_
                   'w_2ndWon', 'l_2ndWon', 'w_bpSaved', 'l_bpSaved', 'w_bpFaced',
                  ]
d[numeric_columns] = d[numeric_columns].astype(float)
```

In [ ]:
```
d.tourney_date

d['tourney_year'] = d.tourney_date.astype(str).str[:4].astype(int)
d['tourney_month'] = d.tourney_date.astype(str).str[4:6].astype(int)

d = d.drop(columns=['tourney_date'])
```

In [ ]:
```
d
```

Out[ ]:

	winner_id	loser_id	winner_rank	loser_rank	winner_age	loser_age	winner_ht	lose
0	103163	101543	11.0	63.0	21.7	31.1	188.0	1
1	102607	102644	211.0	49.0	24.5	24.3	190.0	1
2	103252	102238	48.0	59.0	21.3	26.5	175.0	1
3	103507	103819	45.0	61.0	19.9	18.4	183.0	1
4	102103	102765	167.0	34.0	27.3	23.7	180.0	1
...	...	...	...	...	...	...	...	
73242	212051	209943	1109.0	740.0	18.8	21.8	185.0	1
73243	122533	202475	554.0	748.0	26.9	23.2	185.0	1
73244	144748	144775	416.0	68.0	27.3	26.4	185.0	1
73245	122533	144748	554.0	416.0	26.9	27.3	185.0	1
73246	202475	144775	748.0	68.0	23.2	26.4	185.0	1

73194 rows × 29 columns

Feature Engineering

In [ ]:

```
df1 = d.copy()
df1['first_player_id'] = df1['loser_id']
df1['second_player_id'] = df1['winner_id']
df1['first_player_hand'] = df1['loser_hand']
df1['second_player_hand'] = df1['winner_hand']
df1['first_player_age'] = df1['loser_age']
df1['second_player_age'] = df1['winner_age']
df1['first_player_ht'] = df1['loser_ht']
df1['second_player_ht'] = df1['winner_ht']
df1['first_player_rank'] = df1['loser_rank']
df1['second_player_rank'] = df1['winner_rank']
df1['first_player_ace'] = df1['l_ace']
df1['second_player_ace'] = df1['w_ace']
df1['first_player_svpt'] = df1['l_svpt']
df1['second_player_svpt'] = df1['w_svpt']
df1['first_player_1stIn'] = df1['l_1stIn']
df1['second_player_1stIn'] = df1['w_1stIn']
df1['first_player_1stWon'] = df1['l_1stWon']
df1['second_player_1stWon'] = df1['w_1stWon']
df1['first_player_2ndWon'] = df1['l_2ndWon']
df1['second_player_2ndWon'] = df1['w_2ndWon']
df1['first_player_bpSaved'] = df1['l_bpSaved']
df1['second_player_bpSaved'] = df1['w_bpSaved']
df1['first_player_bpFaced'] = df1['l_bpFaced']
df1['second_player_bpFaced'] = df1['w_bpFaced']
df1['first_player_df'] = df1['l_df']
```

```
df1['second_player_df'] = df1['w_df']
df1['tourney_year'] = df1['tourney_year']
df1['surface'] = df1['surface']
df1['label'] = 0

df2 = d.copy()
df2['first_player_id'] = df2['winner_id']
df2['second_player_id'] = df2['loser_id']

df2['first_player_hand'] = df2['winner_hand']
df2['second_player_hand'] = df2['loser_hand']
df2['first_player_age'] = df2['winner_age']
df2['second_player_age'] = df2['loser_age']
df2['first_player_ht'] = df2['winner_ht']
df2['second_player_ht'] = df2['loser_ht']
df2['first_player_rank'] = df2['winner_rank']
df2['second_player_rank'] = df2['loser_rank']
df2['first_player_ace'] = df2['w_ace']
df2['second_player_ace'] = df2['l_ace']
df2['first_player_svpt'] = df2['w_svpt']
df2['second_player_svpt'] = df2['l_svpt']
df2['first_player_1stIn'] = df2['w_1stIn']
df2['second_player_1stIn'] = df2['l_1stIn']
df2['first_player_1stWon'] = df2['w_1stWon']
df2['second_player_1stWon'] = df2['l_1stWon']
df2['first_player_2ndWon'] = df2['w_2ndWon']
df2['second_player_2ndWon'] = df2['l_2ndWon']
df2['first_player_bpSaved'] = df2['w_bpSaved']
df2['second_player_bpSaved'] = df2['l_bpSaved']
df2['first_player_bpFaced'] = df2['w_bpFaced']
df2['second_player_bpFaced'] = df2['l_bpFaced']
df2['first_player_df'] = df2['w_df']
df2['second_player_df'] = df2['l_df']
df2['label'] = 1

combined_df = pd.concat([df1, df2], ignore_index=True)

combined_df = combined_df.sample(frac=1).reset_index(drop=True)
combined_df
```

Out[ ]:

	winner_id	loser_id	winner_rank	loser_rank	winner_age	loser_age	winner_ht	lo
0	104604	105812	925.0	740.0	31.3	25.5	185.0	
1	104026	103507	11.0	63.0	22.6	24.9	198.0	
2	105137	104523	157.0	102.0	21.4	24.4	183.0	
3	108594	108740	45.0	68.0	31.9	28.8	185.0	
4	105023	104368	36.0	463.0	21.2	24.6	198.0	
...	...	...	...	...	...	...	...	
146383	105916	105023	80.0	13.0	25.9	30.2	188.0	
146384	210506	106005	74.0	104.0	19.4	31.7	185.0	
146385	208363	105948	113.0	85.0	22.9	31.9	185.0	
146386	207989	132283	2.0	46.0	20.6	28.6	185.0	
146387	104386	103174	127.0	196.0	21.6	27.8	180.0	

146388 rows × 56 columns

In [ ]:

```
columns_to_drop = ['winner_id', 'loser_id', 'winner_hand', 'loser_hand', 'winner_ag
                   'winner_ht', 'loser_ht', 'winner_rank', 'loser_rank',
                   'w_ace', 'l_ace', 'w_df', 'l_df', 'w_svpt', 'l_svpt', 'w_1stIn',
                   'w_2ndWon', 'l_2ndWon', 'w_bpSaved', 'l_bpSaved', 'w_bpFaced', '

combined_df.drop(columns=columns_to_drop, inplace=True)
combined_df
```

Out[ ]:

	surface	tourney_year	tourney_month	first_player_id	second_player_id	first_playe
0	Clay	2017	2	104604	105812	
1	Hard	2005	2	103507	104026	
2	Clay	2009	9	104523	105137	
3	Hard	2012	2	108594	108740	
4	Hard	2009	1	105023	104368	
...	...	...	...	...	...	
146383	Hard	2018	1	105916	105023	
146384	Hard	2024	2	106005	210506	
146385	Clay	2024	2	105948	208363	
146386	Hard	2024	1	132283	207989	
146387	Hard	2006	2	104386	103174	

146388 rows × 30 columns

In [ ]:
```
combined_df = pd.get_dummies(combined_df, columns=['surface', 'first_player_hand',

print('Cleaning ve preprocessingden sonra datanin son hali: ', combined_df.shape)
```

```
Cleaning ve preprocessingden sonra datanin son hali:  (146388, 39)
```

MODELLER

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

y = combined_df['label']
df_X = combined_df.drop(columns='label')

X_train_1, X_test_1, y_train, y_test = train_test_split(df_X, y, test_size=0.2, ran
```

In [ ]:
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train_1)

X_test = scaler.transform(X_test_1)
```

RANDOM FOREST

In [ ]:
```
RF_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

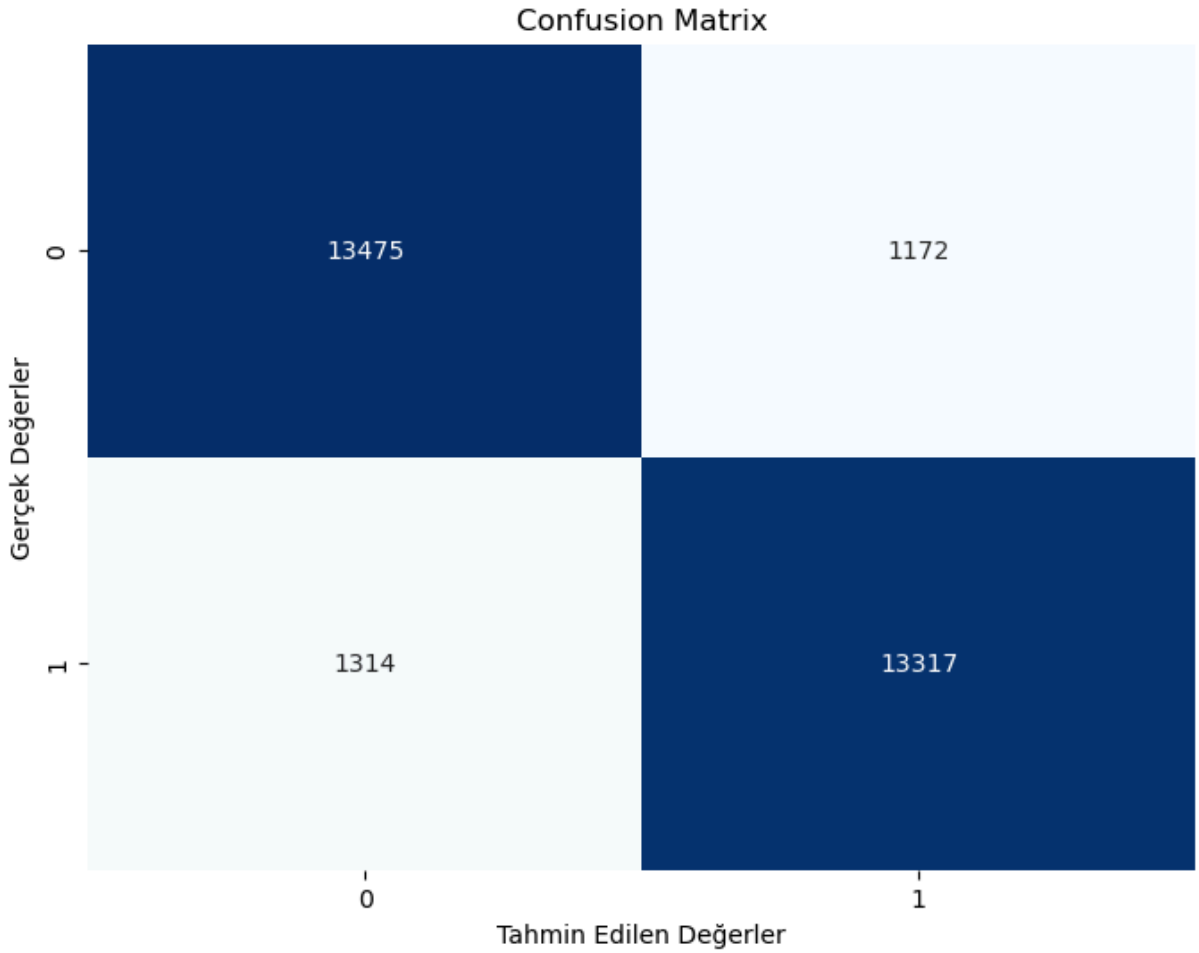
```
RF_classifier.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [ ]:

```
RF_predictions = RF_classifier.predict(X_test)

rf_confmatrix = confusion_matrix(y_test, RF_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(rf_confmatrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Tahmin Edilen Değerler')
plt.ylabel('Gerçek Değerler')
plt.title('Confusion Matrix')
plt.show()

precision = precision_score(y_test, RF_predictions, average='macro')
print('Macro Precision:', precision)

recall = recall_score(y_test, RF_predictions, average='macro')
print('Macro Recall:', recall)

f1 = f1_score(y_test, RF_predictions, average='macro')
print('Macro F1 Score:', f1)

print('Accuracy:', accuracy_score(y_test, RF_predictions))
```




```
Macro Precision: 0.915130614474561
Macro Recall: 0.9150871526952952
Macro F1 Score: 0.9150873556563017
Accuracy: 0.9150898285402008
```

In [ ]:
```
scores = cross_val_score(RF_classifier, df_X, y, cv=5, scoring='accuracy', n_jobs=-
print("Accuracy scores for each fold:")
print(scores)
print("\nAverage Cross-Validation Accuracy:", scores.mean())
```

```
Accuracy scores for each fold:
[0.91348453 0.91601202 0.91430426 0.91232025 0.91594084]

Average Cross-Validation Accuracy: 0.9144123796782043
```

XGBOOST

In [ ]:
```
XGB_classifier = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

XGB_classifier.fit(X_train, y_train)
```

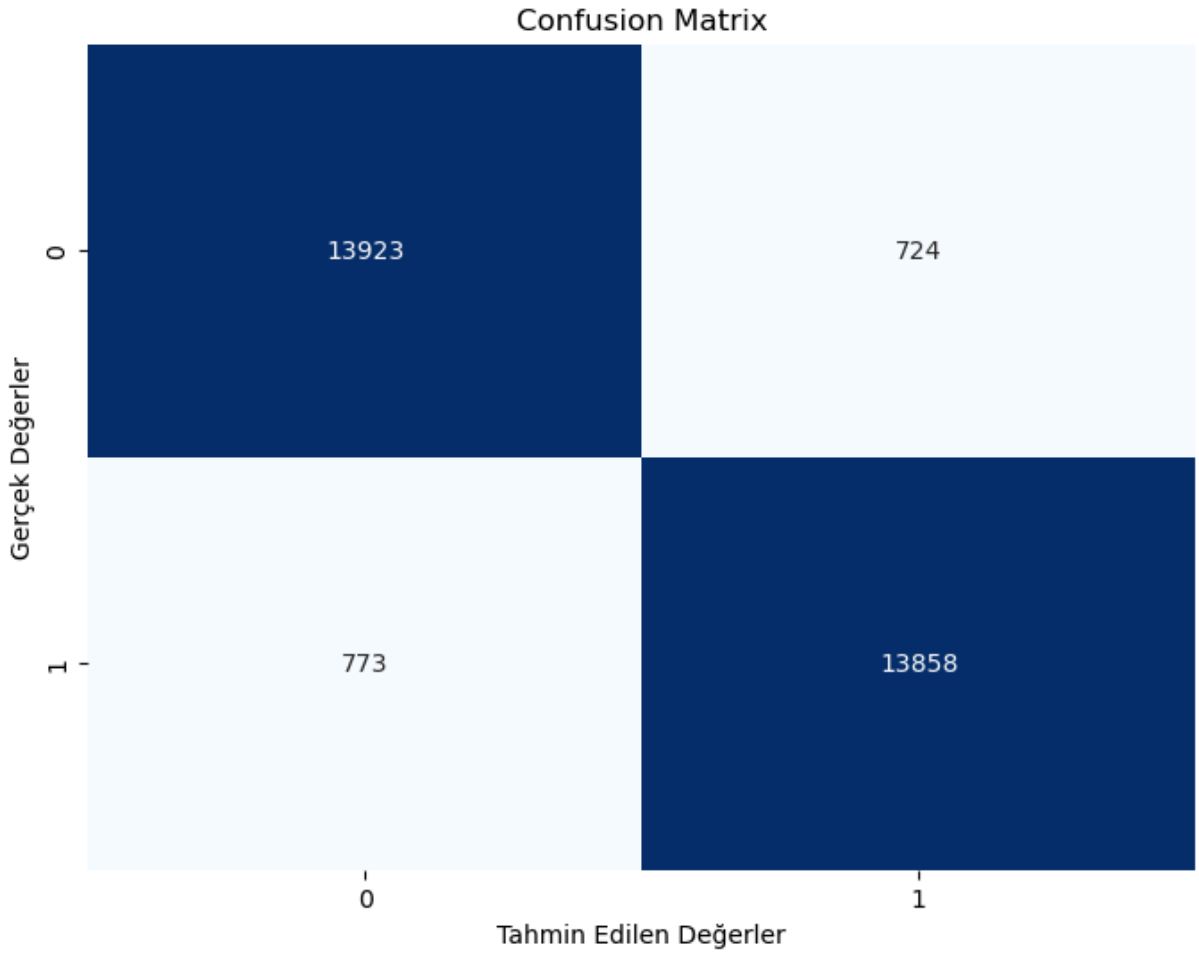
Out[ ]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds
=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=No
```

In [ ]:

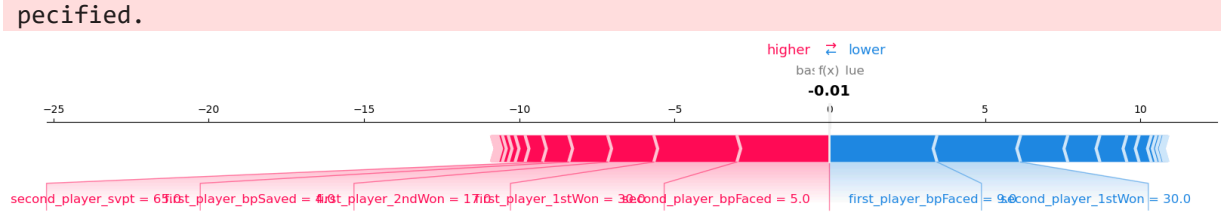
```
XGB_predictions = XGB_classifier.predict(X_test)

xgb_confmatrix = confusion_matrix(y_test, XGB_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(xgb_confmatrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Tahmin Edilen Değerler')
plt.ylabel('Gerçek Değerler')
plt.title('Confusion Matrix')
plt.show()

precision = precision_score(y_test, XGB_predictions, average='macro')
print('Macro Precision:', precision)

recall = recall_score(y_test, XGB_predictions, average='macro')
print('Macro Recall:', recall)

f1 = f1_score(y_test, XGB_predictions, average='macro')
print('Macro F1 Score:', f1)

print('Accuracy:', accuracy_score(y_test, XGB_predictions))
```



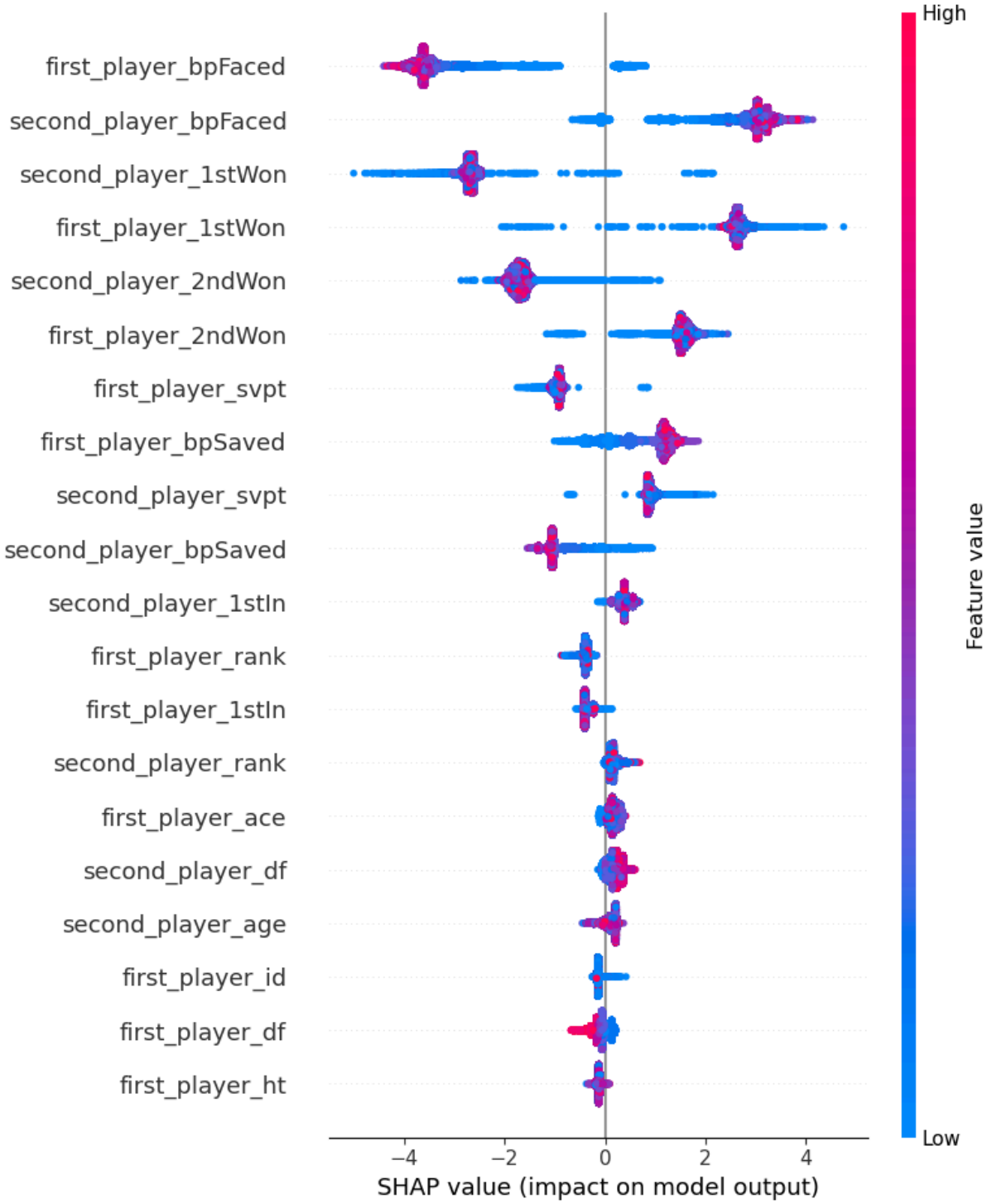

```
Macro Precision: 0.9488751997507461
Macro Recall: 0.9488685284217835
Macro F1 Score: 0.9488692062813062
Accuracy: 0.9488694582963317
```

In [ ]:
```
explainer = shap.TreeExplainer(XGB_classifier)
shap_values = explainer.shap_values(X_train_1)
shap.force_plot(explainer.expected_value, shap_values[0,:], X_train_1.iloc[0,:], ma
shap.summary_plot(shap_values, X_train_1)
```

```
[04:53:24] WARNING: C:\b\abs_0fh_d4x2ng\croot\xgboost-split_1713973188995\work\cpp_s
rc\src\c_api\c_api.cc:1240: Saving into deprecated binary model format, please consi
der using `json` or `ubj`. Model format will default to JSON in XGBoost 2.2 if not s
pecified.
```




```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
```



In [ ]:
```
scores = cross_val_score(XGB_classifier, X_train, y_train, cv=5,
                         scoring='accuracy', n_jobs=-1)

print("Accuracy scores for each fold:")
print(scores)
print("\nAverage Cross-Validation Accuracy:", scores.mean())
```

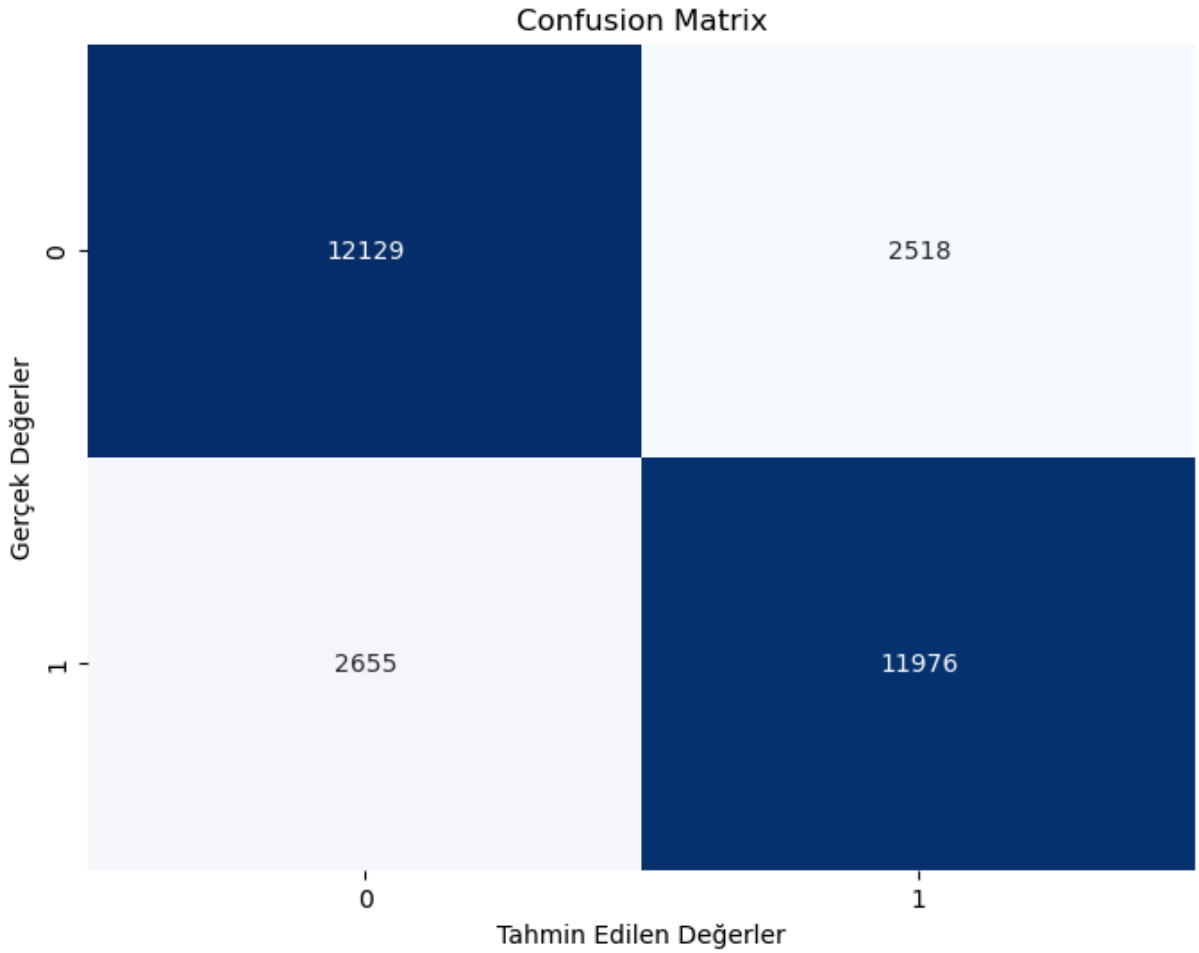
```
Accuracy scores for each fold:
[0.94633251 0.94466741 0.94650329 0.94671676 0.94573478]

Average Cross-Validation Accuracy: 0.9459909486807275
```

DESICION TREE

In [ ]:
```
DT_classifier = DecisionTreeClassifier(max_depth=5, min_samples_split=20, min_sampl
DT_classifier.fit(X_train, y_train)
```

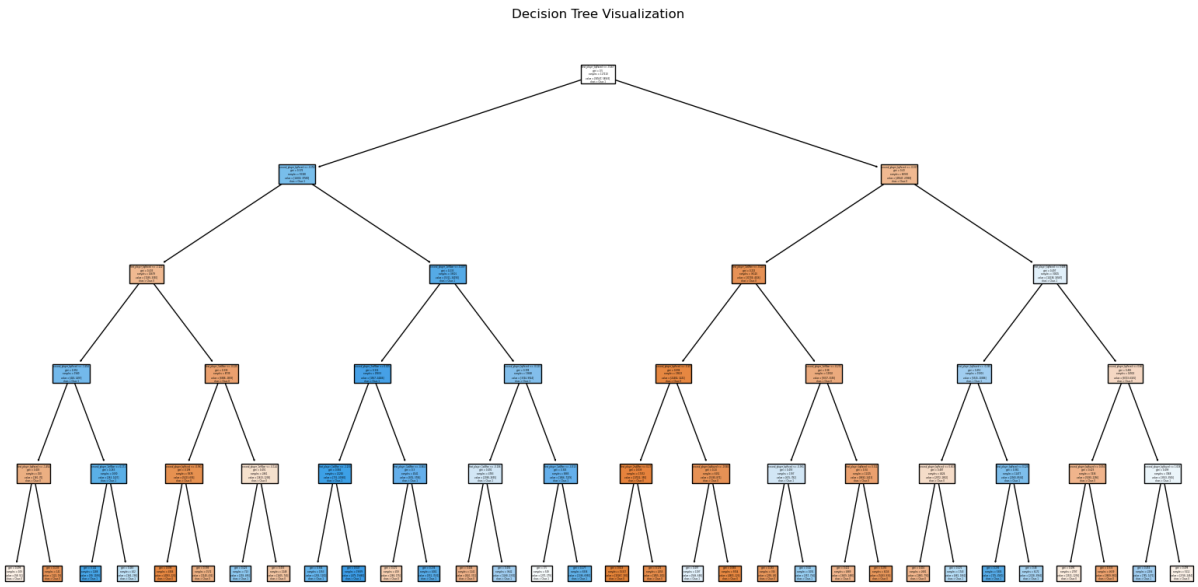
Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_leaf=10, min_samples_split
=20,
                       random_state=42)
```

In [ ]:
```
dt_predictions = DT_classifier.predict(X_test)

dt_confmatrix = confusion_matrix(y_test, dt_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(dt_confmatrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Tahmin Edilen Değerler')
plt.ylabel('Gerçek Değerler')
plt.title('Confusion Matrix')
plt.show()

print('Classification Report:')
print(classification_report(y_test, dt_predictions))

print('Accuracy:', accuracy_score(y_test, dt_predictions))
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.82     14647
           1       0.83      0.82      0.82     14631

    accuracy                           0.82     29278
   macro avg       0.82      0.82      0.82     29278
weighted avg       0.82      0.82      0.82     29278

Accuracy: 0.8233144340460414
```

In [ ]:
```
plt.figure(figsize=(20,10))
plot_tree(DT_classifier, filled=True, feature_names=combined_df.drop(columns='label
plt.title('Decision Tree Visualization')
plt.show()
```



In [ ]:
```
scores = cross_val_score(DT_classifier, X_train, y_train, cv=5, scoring='accuracy',

print("Accuracy scores for each fold:")
print(scores)
print("\nAverage Cross-Validation Accuracy:", scores.mean())
```

```
Accuracy scores for each fold:
[0.82277346 0.82405431 0.82307233 0.82115105 0.82691487]

Average Cross-Validation Accuracy: 0.8235932029715652
```

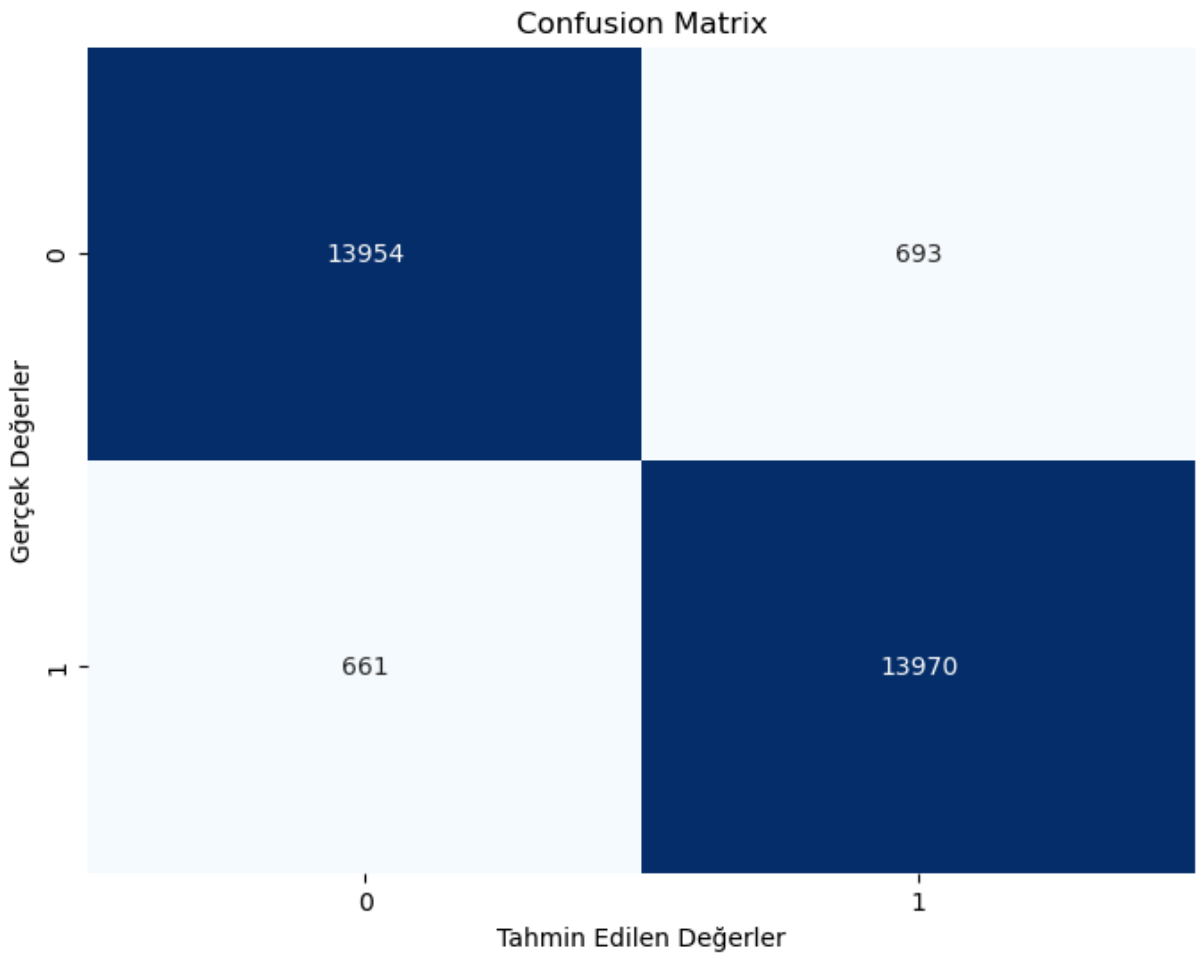
SVM MODELİ

In [ ]:
```
svm_classifier = SVC(C=1.0)

svm_classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC()
```

In [ ]:
```
svm_predictions = svm_classifier.predict(X_test)

svm_confmatrix = confusion_matrix(y_test, svm_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(svm_confmatrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Tahmin Edilen Değerler')
plt.ylabel('Gerçek Değerler')
plt.title('Confusion Matrix')
plt.show()

print('Classification Report:')
print(classification_report(y_test, svm_predictions))
```

```
print('Accuracy:', accuracy_score(y_test, svm_predictions))
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     14647
           1       0.95      0.95      0.95     14631

    accuracy                           0.95     29278
   macro avg       0.95      0.95      0.95     29278
weighted avg       0.95      0.95      0.95     29278

Accuracy: 0.9537536716988866
```

In [ ]:
```
pipeline = make_pipeline(StandardScaler(), SVC())

param_grid = {
    'svc__C': [1, 10],
    'svc__kernel': ['rbf']
}

grid_search = GridSearchCV(pipeline, param_grid, cv=3, verbose=2, n_jobs=-1)

grid_search.fit(X_train, y_train)
```

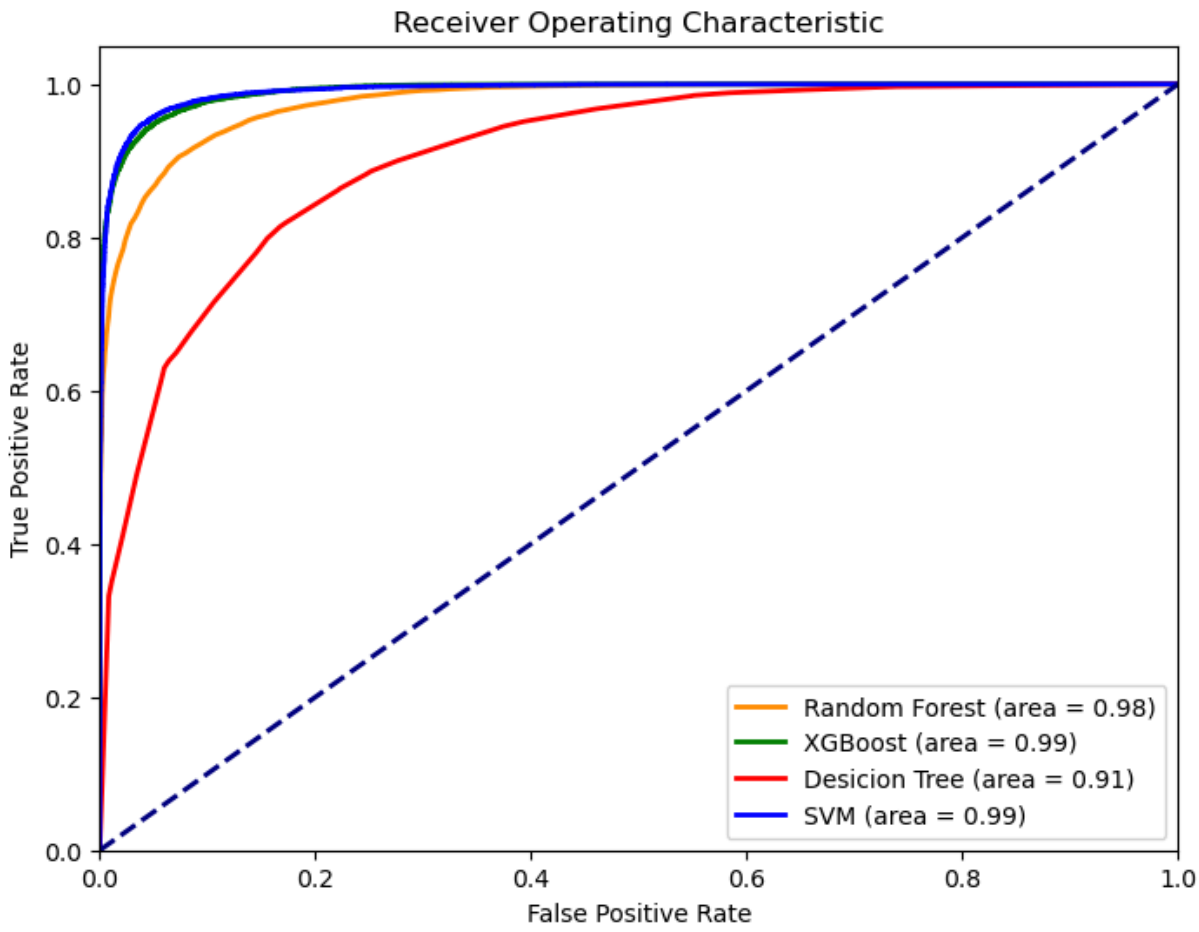
```
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
Best parameters: {'svc__C': 1, 'svc__kernel': 'rbf'}
Best cross-validation score: 0.95
```

ROC CURVELER

In [ ]:
```
RF_probabilities = RF_classifier.predict_proba(X_test)[:, 1]

fpr_rf, tpr_rf, _ = roc_curve(y_test, RF_probabilities)
roc_auc_rf = auc(fpr_rf, tpr_rf)

XGB_probabilities = XGB_classifier.predict_proba(X_test)[:, 1]

fpr_xgb, tpr_xgb, _ = roc_curve(y_test, XGB_probabilities)
roc_auc_xgb = auc(fpr_xgb, tpr_xgb)

DT_probabilities = DT_classifier.predict_proba(X_test)[:, 1]

fpr_dt, tpr_dt, _ = roc_curve(y_test, DT_probabilities)
roc_auc_dt = auc(fpr_dt, tpr_dt)

svm_scores = svm_classifier.decision_function(X_test)
fpr_svm, tpr_svm, _ = roc_curve(y_test, svm_scores)
roc_auc_svm = auc(fpr_svm, tpr_svm)
```

In [ ]:
```
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange',
         lw=2, label='Random Forest (area = %0.2f)' % roc_auc_rf)
plt.plot(fpr_xgb, tpr_xgb, color='green',
         lw=2, label='XGBoost (area = %0.2f)' % roc_auc_xgb)
plt.plot(fpr_dt, tpr_dt, color='red',
         lw=2, label='Desicion Tree (area = %0.2f)' % roc_auc_dt)
plt.plot(fpr_svm, tpr_svm, color='blue', lw=2, label='SVM (area = %0.2f)' % roc_auc

plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Random Forest (area = 0.98)
XGBoost (area = 0.99)
Desicion Tree (area = 0.91)
SVM (area = 0.99)