In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

data= pd.read_csv("2021-2022.csv")
data
```

Out[1]:

	Div	Date	Time	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	...	AvgC<2.5	AHCh	B365CAHH	B365CAHA	PCAHH	PCAHA	MaxCAHH	MaxCAHA	AvgCAHH	AvgCAHA
0	E0	13-08-21	20:00	Brentford	Arsenal	2	0	H	1	0	...	1.62	0.50	1.75	2.05	1.81	2.13	2.05	2.17	1.80	2.09
1	E0	14-08-21	12:30	Man United	Leeds	5	1	H	1	0	...	2.25	-1.00	2.05	1.75	2.17	1.77	2.19	1.93	2.10	1.79
2	E0	14-08-21	15:00	Burnley	Brighton	1	2	A	1	0	...	1.62	0.25	1.79	2.15	1.81	2.14	1.82	2.19	1.79	2.12
3	E0	14-08-21	15:00	Chelsea	Crystal Palace	3	0	H	2	0	...	1.94	-1.50	2.05	1.75	2.12	1.81	2.16	1.93	2.06	1.82
4	E0	14-08-21	15:00	Everton	Southampton	3	1	H	0	1	...	1.67	-0.50	2.05	1.88	2.05	1.88	2.08	1.90	2.03	1.86
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
375	E0	22-05-22	16:00	Crystal Palace	Man United	1	0	H	1	0	...	2.04	0.25	1.68	2.15	1.74	2.23	1.88	2.25	1.74	2.16
376	E0	22-05-22	16:00	Leicester	Southampton	4	1	H	0	0	...	2.63	-0.75	1.83	2.07	1.88	2.03	1.94	2.26	1.87	2.01
377	E0	22-05-22	16:00	Liverpool	Wolves	3	1	H	1	1	...	3.28	-2.50	2.02	1.77	2.06	1.83	2.19	1.99	2.07	1.80
378	E0	22-05-22	16:00	Man City	Aston Villa	3	2	H	0	1	...	3.36	-2.25	2.06	1.84	2.05	1.86	2.09	2.03	2.01	1.87
379	E0	22-05-22	16:00	Norwich	Tottenham	0	5	A	0	2	...	2.40	1.50	1.95	1.95	1.96	1.94	2.16	1.98	1.99	1.89

380 rows × 106 columns

In [2]:
```python
data= data.drop(['Div','Date','Time','Referee','AHh','AHCh'],axis=1)
data
```

Out[2]:

	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	HS	AS	...	AvgC>2.5	AvgC<2.5	B365CAHH	B365CAHA	PCAHH	PCAHA	MaxCAHH	MaxCAHA	AvgCAHH	AvgCAHA
0	Brentford	Arsenal	2	0	H	1	0	H	8	22	...	2.33	1.62	1.75	2.05	1.81	2.13	2.05	2.17	1.80	2.09
1	Man United	Leeds	5	1	H	1	0	H	16	10	...	1.67	2.25	2.05	1.75	2.17	1.77	2.19	1.93	2.10	1.79
2	Burnley	Brighton	1	2	A	1	0	H	14	14	...	2.34	1.62	1.79	2.15	1.81	2.14	1.82	2.19	1.79	2.12
3	Chelsea	Crystal Palace	3	0	H	2	0	H	13	4	...	1.90	1.94	2.05	1.75	2.12	1.81	2.16	1.93	2.06	1.82
4	Everton	Southampton	3	1	H	0	1	A	14	6	...	2.24	1.67	2.05	1.88	2.05	1.88	2.08	1.90	2.03	1.86
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
375	Crystal Palace	Man United	1	0	H	1	0	H	6	10	...	1.80	2.04	1.68	2.15	1.74	2.23	1.88	2.25	1.74	2.16
376	Leicester	Southampton	4	1	H	0	0	D	12	7	...	1.49	2.63	1.83	2.07	1.88	2.03	1.94	2.26	1.87	2.01
377	Liverpool	Wolves	3	1	H	1	1	D	29	7	...	1.34	3.28	2.02	1.77	2.06	1.83	2.19	1.99	2.07	1.80
378	Man City	Aston Villa	3	2	H	0	1	A	24	4	...	1.33	3.36	2.06	1.84	2.05	1.86	2.09	2.03	2.01	1.87
379	Norwich	Tottenham	0	5	A	0	2	A	9	19	...	1.58	2.40	1.95	1.95	1.96	1.94	2.16	1.98	1.99	1.89

380 rows × 100 columns

In [3]:
```python
data['HomeTeam'] = data['HomeTeam'].map({'Charlton':1,'Chelsea':2,'Coventry':3, 'Derby':4, 'Leeds':5, 'Leicester':6,'Liverpool':7, 'Sunderland':8, 'Tottenham':9, 'Man United':10, '…
    'Blackburn':22, 'Fulham':23, 'Birmingham':24, 'Middlesboro':25, 'West Brom':26,
    'Portsmouth':27, 'Wolves':28, 'Norwich':29, 'Crystal Palace':30, 'Wigan':31,
    'Reading':32, 'Sheffield United':33, 'Watford':34, 'Hull':35, 'Stoke':36,
    'Burnley':37, 'Blackpool':38, 'QPR':39, 'Swansea':40, 'Cardiff':41, 'Bournemouth':42,
    'Brighton':43, 'Huddersfield':44, 'Brentford':45}).astype('int')
```

In [4]:
```python
data['AwayTeam'] = data['AwayTeam'].map({'Man City':1, 'West Ham':2, 'Middlesbrough':3, 'Southampton':4, 'Everton':5,
    'Aston Villa':6, 'Bradford':7, 'Arsenal':8, 'Ipswich':9, 'Newcastle':10,
    'Liverpool':11, 'Chelsea':12, 'Man United':13, 'Tottenham':14, 'Charlton':15,
    'Sunderland':16, 'Derby':17, 'Coventry':18, 'Leicester':19, 'Leeds':20,
    'Blackburn':21, 'Bolton':22, 'Fulham':23, 'West Brom':24, 'Middlesboro':25,
    'Birmingham':26, 'Wolves':27, 'Portsmouth':28, 'Crystal Palace':29, 'Norwich':30,
    'Wigan':31, 'Watford':32, 'Sheffield United':33, 'Reading':34, 'Stoke':35, 'Hull':36,
    'Burnley':37, 'Blackpool':38, 'Swansea':39, 'QPR':40, 'Cardiff':41, 'Bournemouth':42,
    'Huddersfield':43, 'Brighton':44, 'Brentford':45}).astype('int')
```

In [5]:
```python
data['FTR'] = data['FTR'].map({'H':2,'A':1,'D':0}).astype('int')
data['FTR'].unique()
```

Out[5]: array([2, 1, 0])

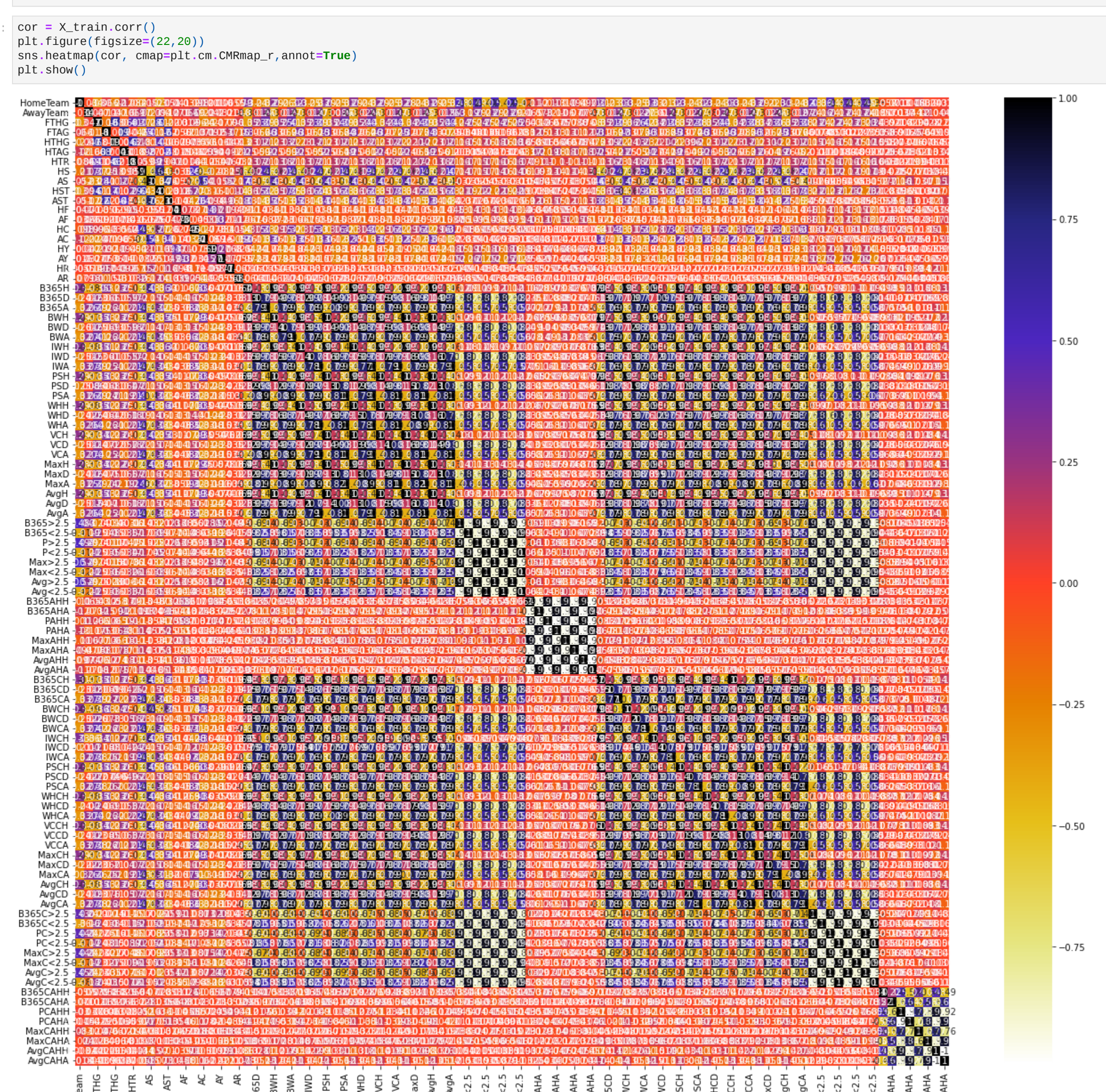
In [6]:
```python
data['HTR'] = data['HTR'].map({'H':2,'A':1,'D':0}).astype('int')
data['HTR'].unique()
```

Out[6]: array([2, 1, 0])

In [7]:
```python
y = data['FTR']
#Load X Variables into a Pandas Dataframe with columns
X = data.drop(['FTR'], axis = 1)
#y is dependent variable and X is independent variable.
```

In [8]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=100,test_size=0.3)
```

In [9]:
```python
cor = X_train.corr()
plt.figure(figsize=(22,20))
sns.heatmap(cor, cmap=plt.cm.CMRmap_r,annot=True)
plt.show()
```



In [10]:
```python
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

In [11]:
```python
corr_features = correlation(X_train, 0.7)
corr_features
```

Out[11]:
```
{'AST',
 'Avg<2.5',
 'Avg>2.5',
 'AvgA',
 'AvgAHA',
 'AvgAHH',
 'AvgC<2.5',
 'AvgC>2.5',
 'AvgCA',
 'AvgCAHA',
 'AvgCAHH',
 'AvgCD',
 'AvgCH',
 'AvgD',
 'AvgH',
 'B365<2.5',
 'B365>2.5',
 'B365A',
 'B365AHA',
 'B365C<2.5',
 'B365C>2.5',
 'B365CA',
 'B365CD',
 'B365CH',
 'BWA',
 'BWCA',
 'BWCD',
 'BWCH',
 'BWD',
 'BWH',
 'HTAG',
 'IWA',
 'IWCA',
 'IWCD',
 'IWCH',
 'IWD',
 'IWH',
 'Max<2.5',
 'Max>2.5',
 'MaxA',
 'MaxAHA',
 'MaxAHH',
 'MaxC<2.5',
 'MaxC>2.5',
 'MaxCA',
 'MaxCAHA',
 'MaxCAHH',
 'MaxCD',
 'MaxCH',
 'MaxD',
 'MaxH',
 'P<2.5',
 'P>2.5',
 'PAHA',
 'PAHH',
 'PC<2.5',
 'PC>2.5',
 'PCAHA',
 'PSA',
 'PSCA',
 'PSCD',
 'PSCH',
 'PSD',
 'PSH',
 'VCA',
 'VCCA',
 'VCCD',
 'VCCH',
 'VCD',
 'VCH',
 'WHA',
 'WHCA',
 'WHCD',
 'WHCH',
 'WHD',
 'WHH'}
```

In [12]:
```python
X_train.drop(corr_features,axis=1)
X_test.drop(corr_features,axis=1)
X_train
```

Out[12]:

	HomeTeam	AwayTeam	FTHG	FTAG	HTHG	HTAG	HTR	HS	AS	HST	...	AvgC>2.5	AvgC<2.5	B365CAHH	B365CAHA	PCAHH	PCAHA	MaxCAHH	MaxCAHA	AvgCAHH	AvgCAHA
26	7	12	1	1	1	1	0	24	6	7	...	2.02	1.82	2.10	1.83	2.11	1.83	2.16	1.88	2.08	1.81
175	17	13	1	1	1	0	2	13	13	8	...	1.63	2.31	1.73	2.08	1.76	2.20	1.87	2.23	1.77	2.12
35	18	2	0	0	0	0	0	11	13	3	...	1.69	2.21	1.90	2.03	1.89	2.03	1.99	2.13	1.87	2.02
296	9	10	5	1	1	1	0	19	8	6	...	1.78	2.07	1.97	1.93	1.97	1.95	2.00	2.04	1.94	1.93
118	16	5	3	0	1	0	2	17	4	7	...	1.51	2.57	2.02	1.77	2.13	1.79	2.20	1.91	2.09	1.80
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
343	37	6	1	3	0	2	1	14	12	6	...	2.29	1.64	1.77	2.14	1.81	2.14	1.81	2.19	1.77	2.12
359	20	29	1	1	0	0	0	13	16	4	...	2.06	1.79	1.82	2.08	1.87	2.04	2.06	2.13	1.88	2.00
323	16	32	5	1	3	1	2	21	5	8	...	1.38	3.01	1.97	1.93	1.95	1.95	2.00	2.06	1.96	1.91
280	30	1	0	0	0	0	0	7	18	1	...	1.69	2.21	1.90	2.03	1.92	2.01	2.03	2.05	1.91	1.97
8	17	2	2	4	2	1	2	17	8	3	...	1.88	1.95	2.01	1.92	2.02	1.91	2.12	1.94	2.00	1.89

266 rows × 99 columns

In [13]:
```python
#Chi-square
#Perform chi2 test
from sklearn.feature_selection import chi2
#Calculating Fscore and p value
f_p_values=chi2(X_train,y_train)
f_p_values
```

Out[13]:
```
(array([1.53700803e+02, 7.30939221e+01, 1.39134045e+02, 1.43248856e+02,
        3.94351997e+01, 5.38429878e+01, 1.50709732e+01, 5.89663796e+01,
        9.32518451e+01, 7.33186061e+01, 1.06928244e+02, 3.98543440e+00,
        5.78481419e-01, 4.94303001e+00, 8.40732268e+00, 6.21540321e+00,
        9.24057502e+00, 8.53282234e+00, 5.74630234e+00, 7.20551204e+01,
        4.92709778e+00, 1.28246736e+02, 7.32690278e+01, 4.53214207e+00,
        1.26148191e+02, 6.65693636e+01, 4.43202890e+00, 1.11471286e+02,
        8.08685826e+01, 5.17610860e+00, 1.49311323e+02, 7.40010769e+01,
        3.94731115e+00, 1.32906219e+02, 7.71882862e+01, 5.06231924e+00,
        1.39692623e+02, 8.86609662e+01, 6.08287869e+00, 1.73956185e+02,
        7.60630400e+01, 5.05264852e+00, 1.33846164e+02, 2.43655726e-01,
        7.37322924e-01, 2.51253688e-01, 7.25914075e-01, 2.59572917e-01,
        7.53033968e-01, 2.43009461e-01, 6.48120695e-01, 7.52793225e-03,
        2.20656138e-02, 1.35330158e-02, 1.42491363e-02, 1.22323266e-02,
        1.13052645e-02, 1.07025712e-02, 1.07978459e-02, 7.03678413e+01,
        4.49462640e+00, 1.15356935e+02, 6.45597659e+01, 5.43064551e+00,
        1.23573013e+02, 6.53633558e+01, 3.47284422e+00, 1.06166434e+02,
        7.84313817e+01, 5.65098707e+00, 1.37038988e+02, 7.27953093e+01,
        4.54371533e+00, 1.33024979e+02, 7.90801169e+01, 5.18600127e+00,
        1.31709132e+02, 9.13114521e+01, 6.42030631e+00, 1.68325914e+02,
        7.32418245e+01, 5.47770645e+00, 1.26353438e+02, 2.16865538e-01,
        8.07573881e-01, 2.92305235e-01, 7.98061535e-01, 3.15555645e-01,
        8.40389721e-01, 2.87399781e-01, 7.53006617e-01, 6.40567699e-03,
        3.96314425e-02, 5.55791380e-03, 1.01116694e-02, 1.49079991e-02,
        1.54139511e-02, 1.03991169e-02, 1.11944090e-02]),
 array([4.21012124e-34, 1.34232134e-16, 6.12951309e-31, 7.83260500e-32,
        2.73372728e-09, 2.03302988e-12, 5.33801455e-04, 1.56896537e-13,
        5.63143550e-21, 1.19968428e-16, 6.03707892e-24, 1.36324501e-01,
        7.48831933e-01, 8.44568095e-02, 1.49407733e-02, 4.47035839e-02,
        9.84996370e-03, 1.40320515e-02, 5.65205399e-02, 2.25646916e-16,
        8.51322891e-02, 1.41766984e-28, 1.22979514e-16, 1.03718888e-01,
        4.04826041e-28, 3.50466383e-15, 1.09042840e-01, 6.22755853e-25,
        2.75175904e-18, 7.51661488e-02, 3.77970708e-33, 8.52845425e-17,
        1.38947992e-01, 1.37968905e-29, 1.73291246e-17, 7.95666995e-02,
        4.63588108e-31, 5.59133873e-20, 4.77660881e-02, 1.68226490e-38,
        3.04173050e-17, 7.99523650e-02, 8.62332436e-30, 8.85300748e-01,
        6.91659524e-01, 8.81943888e-01, 6.95616318e-01, 8.78282961e-01,
        6.86247462e-01, 8.85586864e-01, 7.23206597e-01, 9.96243109e-01,
        9.89027831e-01, 9.93256333e-01, 9.92900751e-01, 9.93902502e-01,
        9.94363314e-01, 9.94663007e-01, 9.94615625e-01, 5.24586862e-16,
        1.05682792e-01, 8.92400363e-26, 9.57249063e-15, 6.61835886e-02,
        1.46710509e-27, 6.40512536e-15, 1.76149518e-01, 8.83592061e-24,
        9.30768882e-18, 5.92793931e-02, 1.74727780e-30, 1.55847354e-16,
        1.03120439e-01, 1.30014815e-29, 6.72931395e-18, 7.47952699e-02,
        2.51029819e-29, 1.48583335e-20, 4.03504329e-02, 2.80861436e-37,
        1.24663662e-16, 6.46444371e-02, 3.65341896e-28, 8.97239215e-01,
        6.67786384e-01, 8.64025827e-01, 6.70970057e-01, 8.54039509e-01,
        6.56918800e-01, 8.66147648e-01, 6.86256847e-01, 9.96802285e-01,
        9.80379320e-01, 9.97224901e-01, 9.94956925e-01, 9.92573713e-01,
        9.92322647e-01, 9.94813936e-01, 9.94418431e-01]))
```

In [14]:
```python
import pandas as pd
p_values=pd.Series(f_p_values[1])
p_values.index=X_train.columns
p_values
```

Out[14]:
```
HomeTeam    4.210121e-34
AwayTeam    1.342321e-16
FTHG        6.129513e-31
FTAG        7.832605e-32
HTHG        2.733727e-09
                ...
PCAHA       9.949569e-01
MaxCAHH     9.925737e-01
MaxCAHA     9.923226e-01
AvgCAHH     9.948139e-01
AvgCAHA     9.944184e-01
Length: 99, dtype: float64
```

In [15]:
```python
p_values.sort_index(ascending=False)
```

Out[15]:
```
WHH     8.528454e-17
WHD     1.389480e-01
WHCH    1.558474e-16
WHCD    1.031204e-01
WHCA    1.300148e-29
            ...
AST     6.037079e-24
AS      5.631436e-21
AR      5.652054e-02
AF      7.488319e-01
AC      1.494077e-02
Length: 99, dtype: float64
```

In [16]:
```python
#Mutual information gain
#Importing mutual information gain
from sklearn.feature_selection import mutual_info_classif
# determine the mutual information
mutual_info = mutual_info_classif(X_train, y_train)
mutual_info
```

Out[16]:
```
array([0.07392065, 0.04444119, 0.28757347, 0.2490599 , 0.0772598 ,
       0.08767678, 0.16319506, 0.02564851, 0.18908481, 0.13687474,
       0.12054634, 0.0339447 , 0.02404628, 0.03059509, 0.        ,
       0.        , 0.        , 0.        , 0.00362953, 0.08245536,
       0.00123321, 0.12425283, 0.14962151, 0.02379332, 0.10505175,
       0.08920534, 0.01745168, 0.11609019, 0.14579029, 0.        ,
       0.10449656, 0.10788717, 0.02150313, 0.09500504, 0.11286928,
       0.02323983, 0.10601232, 0.11468576, 0.06393869, 0.06855891,
       0.12702723, 0.0534141 , 0.10076395, 0.06135073, 0.02441869,
       0.09416781, 0.07706636, 0.04766036, 0.0560466 , 0.07563864,
       0.06688183, 0.01419275, 0.00898411, 0.0446844 , 0.        ,
       0.0005554 , 0.04982156, 0.05661653, 0.02297456, 0.14364863,
       0.        , 0.14988849, 0.13603443, 0.03795137, 0.16817741,
       0.1271339 , 0.        , 0.12497207, 0.16903642, 0.02553458,
       0.15178523, 0.15424518, 0.        , 0.08539069, 0.1274914 ,
       0.03565753, 0.12303995, 0.15001767, 0.04536561, 0.1093296 ,
       0.192098  , 0.        , 0.13361784, 0.07518572, 0.08495665,
       0.05105171, 0.07305942, 0.01642163, 0.05018737, 0.03256263,
       0.03894157, 0.03742308, 0.06053066, 0.        , 0.        ,
       0.06271725, 0.02438882, 0.        , 0.00152827])
```

In [17]:
```python
#Representing in list form
mutual_info = pd.Series(mutual_info)
mutual_info.index = X_train.columns
mutual_info.sort_values(ascending=False)
```

Out[17]:
```
FTHG       0.287573
FTAG       0.249060
AvgCH      0.192098
AS         0.189085
PSCH       0.169036
             ...
PCAHH      0.000000
PCAHA      0.000000
PAHA       0.000000
AvgCAHH    0.000000
WHCD       0.000000
Length: 99, dtype: float64
```

In [18]:
```python
mutual_info.sort_values(ascending=False).plot.bar(figsize=(20, 8))
```

Out[18]: <AxesSubplot:>

In [19]:
```python
from sklearn.feature_selection import SelectKBest
#No we Will select the top 10 important features
sel_five_cols = SelectKBest(mutual_info_classif, k=10)
sel_five_cols.fit(X_train, y_train)
X_train.columns[sel_five_cols.get_support()]
```

Out[19]:
```
Index(['FTHG', 'FTAG', 'HTR', 'B365A', 'BWH', 'BWCA', 'PSCH', 'PSCA', 'WHCH',
       'AvgCH'],
      dtype='object')
```

In [20]:
```python
columns = ['HomeTeam', 'AwayTeam', 'AST','HST','HS','AS']
X = data[columns]
Y = data['FTR']
```

In [21]:
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
print(X.shape, X_train.shape, X_test.shape)
```

(380, 6) (304, 6) (76, 6)

In [22]:
```python
import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

LOGISTIC REGRESSION

In [23]:
```python
modelLogisticRegression = LogisticRegression()
modelLogisticRegression.fit(X_train, Y_train)
X_train_prediction = modelLogisticRegression.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:",round(training_data_accuracy*100,2),'%')
X_test_prediction = modelLogisticRegression.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:",round(test_data_accuracy*100,2),'%')
```

```
Training Accuracy: 62.17 %
Testing Accuracy: 68.42 %
C:\Users\Acer\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Logistic Regression Hyperparameter tuning

In [33]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l2']
}

# Create the Logistic Regression model
modelLogisticRegression = LogisticRegression(max_iter=1000,penalty='l1', solver='liblinear')

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=modelLogisticRegression, param_grid=param_grid, cv=5)
grid_search.fit(X_train, Y_train)

# Get the best model and its hyperparameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Evaluate the best model on the training data
X_train_prediction = best_model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:", round(training_data_accuracy * 100, 2), "%")

# Evaluate the best model on the test data
X_test_prediction = best_model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:", round(test_data_accuracy * 100, 2), "%")
```

```
Best Hyperparameters: {'C': 0.1, 'penalty': 'l2'}
Training Accuracy: 62.17 %
Testing Accuracy: 69.74 %
```

SVC

In [24]:
```python
from sklearn.calibration import CalibratedClassifierCV
modelSVC = SVC()
modelSVC = CalibratedClassifierCV(modelSVC)
modelSVC.fit(X_train, Y_train)
X_train_prediction = modelSVC.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:",round(training_data_accuracy*100,2),'%')

X_test_prediction = modelSVC.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:",round(test_data_accuracy*100,2),'%')
SVCTEST = round(test_data_accuracy*100,2)
```

```
Training Accuracy: 58.22 %
Testing Accuracy: 59.21 %
```

XGBOOST

In [25]:
```python
modelXGBClassifier = xgb.XGBClassifier()
modelXGBClassifier.fit(X_train, Y_train)
X_train_prediction = modelXGBClassifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Traning Accuracy:",round(training_data_accuracy*100,2),'%')

X_test_prediction = modelXGBClassifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Test Accuracy:",round(test_data_accuracy*100,2),'%')
XGBTEST = round(test_data_accuracy*100,2)
```

```
Traning Accuracy: 100.0 %
Test Accuracy: 65.79 %
```

Xgboost Hyperparameter tuning

In [34]:
```python
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300]
}

# Create the XGBoost classifier
modelXGBClassifier = xgb.XGBClassifier()

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=modelXGBClassifier, param_grid=param_grid, cv=5)
grid_search.fit(X_train, Y_train)

# Get the best model and its hyperparameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Evaluate the best model on the training data
X_train_prediction = best_model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:", round(training_data_accuracy * 100, 2), "%")

# Evaluate the best model on the test data
X_test_prediction = best_model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:", round(test_data_accuracy * 100, 2), "%")

# Save the accuracy for later use
XGBTEST = round(test_data_accuracy * 100, 2)
```

```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}
Training Accuracy: 74.01 %
Testing Accuracy: 69.74 %
```

GAUSSIAN NAIVE BAYES

In [26]:
```python
from sklearn.naive_bayes import GaussianNB
modelGaussianNB = GaussianNB()
modelGaussianNB.fit(X_train, Y_train)
X_train_prediction = modelGaussianNB.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:",round(training_data_accuracy*100,2),'%')
X_test_prediction = modelGaussianNB.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:",round(test_data_accuracy*100,2),'%')
SVCTEST = round(test_data_accuracy*100,2)
```

```
Training Accuracy: 58.55 %
Testing Accuracy: 65.79 %
```

Gaussian Naive Bayes Hyper parameter tuning

In [36]:
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Create the Gaussian Naive Bayes model
modelGaussianNB = GaussianNB()

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7]
}

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=modelGaussianNB, param_grid=param_grid, cv=5)
grid_search.fit(X_train, Y_train)

# Get the best model and its hyperparameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Evaluate the best model on the training data
X_train_prediction = best_model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:", round(training_data_accuracy * 100, 2), "%")

# Evaluate the best model on the test data
X_test_prediction = best_model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:", round(test_data_accuracy * 100, 2), "%")

# Save the accuracy for later use
SVCTEST = round(test_data_accuracy * 100, 2)
```

```
Best Hyperparameters: {'var_smoothing': 1e-09}
Training Accuracy: 58.55 %
Testing Accuracy: 65.79 %
```

RANDOM FOREST CLASSIFIER

In [27]:
```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, Y_train)
X_train_prediction = rf.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:",round(training_data_accuracy*100,2),'%')
X_test_prediction = rf.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:",round(test_data_accuracy*100,2),'%')
```

```
Training Accuracy: 100.0 %
Testing Accuracy: 64.47 %
```

Random Forest Classifier (Hyperparameter tuning)

In [31]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

# Create the random forest classifier
rf = RandomForestClassifier(random_state=42)

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5)
grid_search.fit(X_train, Y_train)

# Get the best model and its hyperparameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Evaluate the best model on the training data
X_train_prediction = best_model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("Training Accuracy:", round(training_data_accuracy * 100, 2), "%")

# Evaluate the best model on the test data
X_test_prediction = best_model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("Testing Accuracy:", round(test_data_accuracy * 100, 2), "%")
```

```
Best Hyperparameters: {'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 100}
Training Accuracy: 76.97 %
Testing Accuracy: 68.42 %
```

In [ ]: