In [1]:
```
import glob
import os
import zipfile
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [937]:
```
class ExtractFiles:

    def __init__(self):
        self.dfs = {}

    def all_dataFrames(self,path):

        allFiles = glob.glob(path + "*.zip")

        for file in allFiles:
            zip_file = zipfile.ZipFile(file)
            for text_file in zip_file.infolist():
                if text_file.filename.endswith('.csv') and 'ALL PLAYS' i
n os.path.basename(text_file.filename):
                    self.dfs[text_file.filename] = pd.read_csv(zip_file.
open(text_file.filename))

        drop_columns = ['pff_PLAYID', 'pff_GAMEDATE', 'pff_GAMESEASON',
'pff_WEEK', 'pff_GSISGAMEKEY', 'pff_GSISPLAYID',
                        'pff_CATCHABLE',
                        'pff_GARBAGETIME', 'pff_NOPLAY', 'pff_OFFFORMATI
ONUNBALANCED', 'pff_OPTION', 'pff_PLAYACTION',
                        'pff_PREVIOUSPFFPLAYID',
                        'pff_SCREEN', 'pff_SNAPTIME', 'pff_SORTORDER',
'pff_STUNT', 'pff_2MINUTE', 'pff_BALLCARRIER',
                        'pff_BOXPLAYERS', 'pff_BUNCHED',
                        'pff_CENTERPASSBLOCKDIRECTION', 'pff_CHECKROUTE'
, 'pff_CHIPROUTE', 'pff_DBDEPTH',
                        'pff_DEFPLAYERS','pff_DEFPLAYERSRATINGS','pff_DE
FSUBSTITUTIONS', 'pff_DLTECHNIQUES', 'pff_DOUBLETEAM',
                        'pff_DRIVEENDPLAYNUMBER', 'pff_DROPBACKDEPTH',
'pff_DROPBACKTYPE', 'pff_FIRSTCONTACT',
                        'pff_GUNNERS', 'pff_HANGTIME','pff_HASH', 'pff_H
ASHDEF', 'pff_INJURED', 'pff_INTERCEPTION',
                        'pff_KEYPLAYERS', 'pff_KICKDIRACTUAL','pff_KICKD
IRINTENDED', 'pff_KICKER', 'pff_KICKWIDTH', 'pff_KICKZONE', 'pff_LBDEPT
H',
                        'pff_MOFOCPLAYED', 'pff_MOFOCSHOWN',
                        'pff_NEGATIVEPFFGRADE', 'pff_OFFFORMATION', 'pff
_OFFFORMATIONGROUP', 'pff_OFFODDITIES',
                        'pff_OFFPERSONNEL', 'pff_OFFPERSONNELBASIC',
                        'pff_OFFPERSONNELSKILL', 'pff_OFFPLAYERS', 'pff_
OFFPLAYERSRATINGS', 'pff_OFFSUBSTITUTIONS',
                        'pff_ONLOS', 'pff_OPERATIONTIME',
                        'pff_PASSBLOCKING', 'pff_PASSCOVERAGE', 'pff_PAS
SCOVERAGE1', 'pff_PASSCOVERAGE2',
                        'pff_PASSCOVERAGEPLAYERS', 'pff_PASSDIRECTION',
                        'pff_PASSER', 'pff_PASSPATTERN', 'pff_PASSPATTER
NBASIC', 'pff_PASSPATTERNBYPLAYER',
                        'pff_PASSRECEIVERPOSITIONTARGET',
                        'pff_PASSRECEIVERTARGET', 'pff_PASSROUTETARGET',
 'pff_PASSROUTETARGETGROUP', 'pff_PASSWIDTH',
                        'pff_PASSZONE', 'pff_PISTOL',
                        'pff_PLAYACTIONFAKE', 'pff_PLAYENDFIELDPOSITION'
, 'pff_POAACTUAL', 'pff_POACHANGEREASON',
                        'pff_POAINTENDED', 'pff_POSITIVEPFFGRADE',
```

```
                        'pff_PRESS', 'pff_PRESSUREDETAIL', 'pff_PUMPFAK
E', 'pff_PUNTRUSH', 'pff_PURSUIT', 'pff_QB',
                        'pff_QBMOVEDOFFSPOT', 'pff_QBPRESSURE',
                        'pff_QBPRESSUREALLOWED', 'pff_QBRESET', 'pff_QBS
CRAMBLE', 'pff_RBALIGNMENT', 'pff_RBDEPTH',
                        'pff_RBDIRECTION', 'pff_RBSINBACKFIELD',
                        'pff_RETDIRECTIONINTENDED', 'pff_RETURNDIRECTIO
N', 'pff_RETURNER', 'pff_RUNCONCEPT3',
                        'pff_RUNCONCEPTPRIMARY', 'pff_RUNCONCEPTSECONDAR
Y',
                        'pff_SHIFTMOTION', 'pff_SHOTGUN', 'pff_SPECIALTE
AMSTYPE', 'pff_STSAFETIES',
                        'pff_TEALIGNMENT', 'pff_UNBLOCKEDPRESSURE',
                        'pff_VISE', 'pff_WRALIGNMENT', 'pff_PLAYCLOCK',
'pff_PASSRUSHPLAYERS']

        for each_df in self.dfs.values():
            each_df.drop(drop_columns, axis=1, inplace=True)
```

In [938]:
```
class MatchLevelFacts:

    def __init__(self, data):
        self.data = data
        self.status_data = None
        self.winner = None
        self.loser = None

    # keep track of games and remove any duplicates
    def get_gameid(self):
        return self.data.pff_GAMEID.values[0]

    # grouped by scoredifferential quarter etc., and later used to find
 the match winner and loser
    def game_status(self):
        self.status_data = self.data.groupby(by=[self.data.pff_QUARTER,
                                                 self.data.pff_DEFTEAM,
                                                 self.data.pff_DEFSCORE,
                                                 self.data.pff_OFFTEAM,
                                                 self.data.pff_OFFSCORE,
                                                 self.data.pff_SCOREDIFF
ERENTIAL],
                                                 axis=0, as_index=False).siz
e().to_frame().reset_index()

    def set_winner(self):

        self.game_status()

        off_team = self.status_data.tail(1).pff_OFFTEAM.values[0]
        def_team = self.status_data.tail(1).pff_DEFTEAM.values[0]
        off_score = self.status_data.tail(1).pff_OFFSCORE.values[0]
        def_score = self.status_data.tail(1).pff_DEFSCORE.values[0]

        if off_score > def_score:
            self.winner = off_team
            self.loser = def_team
        else:
            self.winner = def_team
            self.loser = off_team

    # result of given team by end of each quarter [TAILING,LEADING,EQUA
L]
    def quarter_status(self,given_team):
        quarter_status = []
        for each_quarter in self.status_data.pff_QUARTER.unique():
            quarter_status.append(self.status_data.loc[self.status_data.
pff_QUARTER == each_quarter, :].tail(1))

        status = {}
        for each_quarter in quarter_status:
            off_team = each_quarter.pff_OFFTEAM.values[0]
            def_team = each_quarter.pff_DEFTEAM.values[0]
            score_dif = each_quarter.pff_SCOREDIFFERENTIAL.values[0]
```

```
            if (off_team == given_team and score_dif < 0) or (def_team =
= given_team and score_dif > 0):
                status[each_quarter.pff_QUARTER.values[0]] =  ['TAILING'
,score_dif]
            elif (off_team == given_team and score_dif > 0) or (def_team
 == given_team and score_dif < 0):
                status[each_quarter.pff_QUARTER.values[0]] =  ['LEADING'
,abs(score_dif)]
            else:
                status[each_quarter.pff_QUARTER.values[0]] =  ['EQUAL',0
]

        return status

    # no of successful deep passes in a game
    def deep_passes(self, given_team):
        deep_data = self.data.loc[self.data.pff_PASSRESULT == 'COMPLETE'
,:]
        deep_data = deep_data.groupby(by=[self.data.pff_OFFTEAM, self.da
ta.pff_QUARTER, self.data.pff_DEEPPASS], axis=0,as_index=False).size().t
o_frame(name = 'Count').reset_index()
        deep_data = deep_data.loc[(deep_data.pff_DEEPPASS != 0) & (deep_
data.pff_OFFTEAM == given_team),:]
        return deep_data.Count.sum()

    # percentage of successful plays by a given team in the entire match
    def success_percent(self,given_team):
        success_data = self.data.loc[self.data.pff_DEFTEAM == given_team
, :].groupby(by=[self.data.pff_DEFSUCCESS], axis=0,as_index=False).size
().to_frame('Count').reset_index()
        success_def = round(success_data.loc[success_data.pff_DEFSUCCESS
 == '1G',:].Count.values[0]/success_data.Count.sum(),4)*100

        success_data = self.data.loc[self.data.pff_OFFTEAM == given_team
, :].groupby(by=[self.data.pff_OFFSUCCESS],axis=0,as_index=False).size()
.to_frame('Count').reset_index()
        success_off = round(success_data.loc[success_data.pff_OFFSUCCESS
 == '1G', :].Count.values[0] / success_data.Count.sum(),4) * 100

        return {'def_score':success_def,'off_score':success_off}

    # average no of yards yet to cover after 2nd and 3rd down
    def average_distance(self,given_team):
        distance_data = self.data.groupby(by=[self.data.pff_OFFTEAM, sel
f.data.pff_DOWN, self.data.pff_DISTANCE], axis=0,as_index=False).size().
to_frame(name = 'Count').reset_index()

        second_down = distance_data.loc[(distance_data.pff_OFFTEAM == gi
ven_team) & (distance_data.pff_DOWN == 2), :]
        second_down['distance_covered'] = second_down.pff_DISTANCE * sec
ond_down.Count
        second_down = round(second_down.distance_covered.sum() / second_
down.Count.sum(),2)

        third_down = distance_data.loc[(distance_data.pff_OFFTEAM == giv
en_team) & (distance_data.pff_DOWN == 3), :]
        third_down['distance_covered'] = third_down.pff_DISTANCE * third
```

```
_down.Count
        third_down = round(third_down.distance_covered.sum() / third_dow
n.Count.sum(), 2)

        return [second_down,third_down]

    # drives indicate continuous plays without loosing the ball; in this
 case no of drives with more than 4
    def drive_total(self,given_team):
        drive_data = self.data[['pff_OFFTEAM','pff_DRIVE','pff_DRIVEPLA
Y','pff_DISTANCE']]
        drive_data = drive_data.loc[drive_data.pff_OFFTEAM == given_team
,:]
        drive_data = drive_data.dropna()
        drive_data = drive_data.groupby(by=[self.data.pff_OFFTEAM, self.
data.pff_DRIVE], axis=0,as_index=False).size().to_frame(name = 'Count').
reset_index()
        drive_data = drive_data.loc[drive_data.Count>4,:]
        return drive_data.shape[0]

    # punt returns when the offense team looses the ball
    def punt_return(self,given_team):
        punt_data = self.data[['pff_OFFTEAM', 'pff_DEFTEAM', 'pff_DRIVES
TARTEVENT', 'pff_DRIVEENDEVENT']]
        punt_data = punt_data.loc[(punt_data.pff_DRIVESTARTEVENT == 'PUN
T - RETURN') & (punt_data.pff_DEFTEAM == given_team),:]
        return punt_data.shape[0]

    # successful interceptions that leads to a touchdown
    def interception_touchdown(self,given_team):
         int_touch_data = self.data[['pff_DRIVESTARTFIELDPOSITION',
                                     'pff_OFFTEAM',
                                     'pff_DRIVESTARTEVENT',
                                     'pff_DRIVEENDEVENT']].loc[(self.dat
a.pff_DRIVESTARTEVENT=='INTERCEPTION')
                                                               & (self.d
ata.pff_DRIVEENDEVENT=='TOUCHDOWN')
                                                               & (self.d
ata.pff_OFFTEAM==given_team),:].drop_duplicates()

         return int_touch_data.shape[0]

    # pressure put on by defense team on offense team
    def rush_result(self,given_team):
        rush_data = self.data.loc[(self.data.pff_PASSRUSHRESULT.notnull
()) & (self.data.pff_DEFTEAM == given_team),:][['pff_DEFTEAM','pff_PASSR
USHRESULT',]]
        rush_data = rush_data.groupby(by=[rush_data.pff_PASSRUSHRESULT],
 axis=0,as_index=False).size().to_frame(name = 'Count').reset_index()

        return {'HIT':rush_data.loc[rush_data.pff_PASSRUSHRESULT == 'HI
T','Count'].values[0] if 'HIT' in rush_data.pff_PASSRUSHRESULT.unique()
else 0,
                'HURRY': rush_data.loc[rush_data.pff_PASSRUSHRESULT ==
'HURRY', 'Count'].values[0] if 'HURRY' in rush_data.pff_PASSRUSHRESULT.u
nique() else 0,
                'SACK': rush_data.loc[rush_data.pff_PASSRUSHRESULT == 'S
```

```
ACK', 'Count'].values[0] if 'SACK' in rush_data.pff_PASSRUSHRESULT.uniqu
e() else 0}

    # overall missed tackles in a game
    def missed_tackle(self,given_team):
        return self.data.loc[(self.data.pff_MISSEDTACKLE.notnull()) & (s
elf.data.pff_DEFTEAM == given_team),:][['pff_DEFTEAM','pff_MISSEDTACKLE'
]].shape[0]

    # in football blitz means an extra defensive player in near the defe
nse line to stop the quarter back
    def successful_blitz(self,given_team):
        blitz_data = self.data.loc[(self.data.pff_BLITZDOG==1)][['pff_DE
FTEAM','pff_PASSRESULT']]
        blitz_data = blitz_data.loc[blitz_data.pff_DEFTEAM==given_team]
        total_blitz = blitz_data.shape[0]
        incomplete_blitz = blitz_data[blitz_data.pff_PASSRESULT != 'COMP
LETE'].shape[0]
        return round(incomplete_blitz/(total_blitz+1),4) * 100

    def yards_after_catch(self,given_team):
        yards_data = self.data.dropna(axis=0, subset=['pff_YARDSAFTERCAT
CH'])[['pff_OFFTEAM','pff_YARDSAFTERCATCH']]
        yards_data = yards_data.loc[yards_data.pff_OFFTEAM==given_team
,:]
        return round(yards_data.pff_YARDSAFTERCATCH.mean(),2)

    def yards_after_contact(self,given_team):
        yards_data = self.data.dropna(axis=0, subset=['pff_YARDSAFTERCON
TACT'])[['pff_OFFTEAM','pff_YARDSAFTERCONTACT']]
        yards_data = yards_data.loc[yards_data.pff_OFFTEAM==given_team
,:]
        return round(yards_data.pff_YARDSAFTERCONTACT.mean(),2)

    # no of runs in the modern day games over passes
    def run_over_pass(self,given_team):
        run_data = self.data.loc[self.data.pff_OFFTEAM == given_team]
        run_data = run_data.dropna(axis=0,subset=['pff_RUNPASS'])
        total_plays = run_data.shape[0]
        run_plays = run_data.loc[run_data.pff_RUNPASS == 'R'].shape[0]
        return round(run_plays/total_plays,4) * 100
```

In [939]:
```
extract = ExtractFiles()
extract.all_dataFrames('/Users/vamsi/Downloads/PFF/')
```

In [940]:
```
cols = ['GAMEID','2nd_Q_Status','3rd_Q_Diff','No_of_DeepPass','DEF_Score
_Percent','OFF_Score_Percent','2nd_Down_Avg','3rd_Down_Avg',
           'Drives_5plus','No_of_Punt_Returns','Interception_Touchdown',
'Rush_HIT','Rush_Hurry','Rush_Sack','Missed_Tackle','Blitz_Success','Yar
ds_After_Catch','Yards_After_Contact','RUN_Percentage','Label']
df = pd.DataFrame(columns =cols)
```

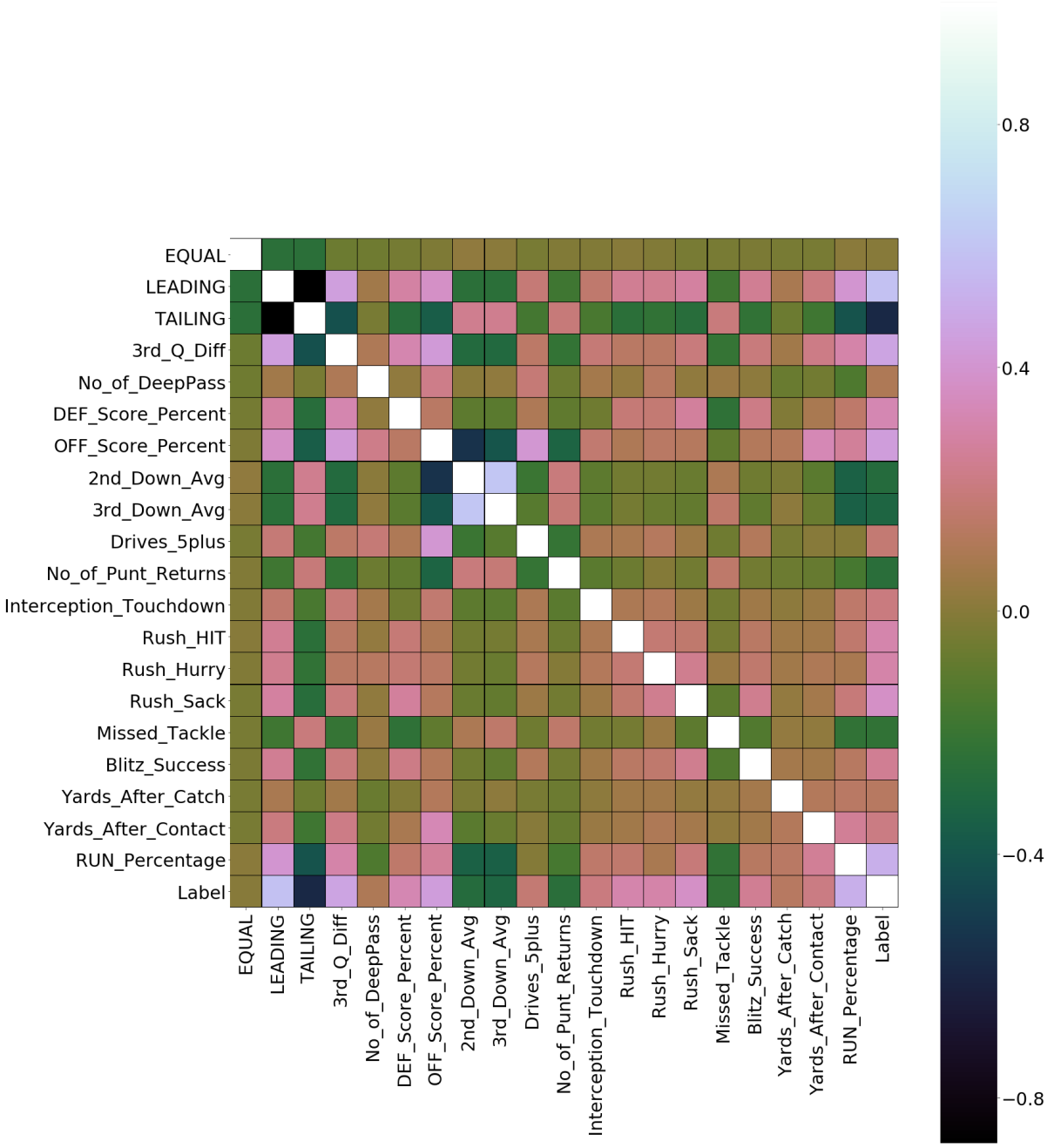
In [941]:
```
for key in extract.dfs.keys():

    match = MatchLevelFacts(extract.dfs[key])
    match.set_winner()

    winner_ =[match.get_gameid(),
            match.quarter_status(match.winner)[2][0],
            match.quarter_status(match.winner)[3][1],
            match.deep_passes(match.winner),
            match.success_percent(match.winner)['def_score'],
            match.success_percent(match.winner)['off_score'],
            match.average_distance(match.winner)[0],
            match.average_distance(match.winner)[1],
            match.drive_total(match.winner),
            match.punt_return(match.winner),
            match.interception_touchdown(match.winner),
            match.rush_result(match.winner)['HIT'],
            match.rush_result(match.winner)['HURRY'],
            match.rush_result(match.winner)['SACK'],
            match.missed_tackle(match.winner),
            match.successful_blitz(match.winner),
            match.yards_after_catch(match.winner),
            match.yards_after_contact(match.winner),
            match.run_over_pass(match.winner),
            'WON'
            ]
    df = df.append(pd.Series(winner_, index=cols), ignore_index=True)

    loser_ =  [match.get_gameid(),
               match.quarter_status(match.loser)[2][0],
               match.quarter_status(match.loser)[3][1],
               match.deep_passes(match.loser),
               match.success_percent(match.loser)['def_score'],
               match.success_percent(match.loser)['off_score'],
               match.average_distance(match.loser)[0],
               match.average_distance(match.loser)[1],
               match.drive_total(match.loser),
               match.punt_return(match.loser),
               match.interception_touchdown(match.loser),
               match.rush_result(match.loser)['HIT'],
               match.rush_result(match.loser)['HURRY'],
               match.rush_result(match.loser)['SACK'],
               match.missed_tackle(match.loser),
               match.successful_blitz(match.loser),
               match.yards_after_catch(match.loser),
               match.yards_after_contact(match.loser),
               match.run_over_pass(match.loser),
               'LOST'
               ]
    df = df.append(pd.Series(loser_, index=cols), ignore_index=True)
```

In [942]:
```
#df.to_csv('/Users/vamsi/Downloads/Matches_Trifacta.csv')
```

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:
```
#data_NU is used as test set to validate our model
data = pd.read_csv('Data/Matches_Trifacta_Final.csv')
```

In [28]:
```
f, ax = plt.subplots(figsize=(25,35))
with sns.plotting_context(font_scale=1.5):
    plot = sns.heatmap(data.corr(),ax = ax,linewidths=0.25,vmax=1.0, square=True, cmap="cubehelix", linecolor='k', annot=False)
    fig = plot.get_figure()
    fig.savefig('Correlation_Heatmap2.png')
```



In [5]:
```
# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
import collections
```

In [6]:
```
from sklearn.preprocessing import Normalizer
#X_train = data.drop(['Label','TAILING','EQUAL'], axis=1)
#X_test = data_NU.drop(['Label','TAILING','EQUAL'], axis=1)
#y_train = data['Label']
#y_test = data_NU['Label']
X = data.drop(['Label','EQUAL','No_of_DeepPass','Yards_After_Catch'],axi
s=1)
y= data['Label']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=42)

normalizer = Normalizer().fit(X_train)
X_train = normalizer.transform(X_train) #normalize(X_train)

# Turn the values into an array for feeding the classification algorithm
s.
y_train = y_train.values
y_test = y_test.values

classifiers = {
    "LogisiticRegression": LogisticRegression(),
    "KNearest": KNeighborsClassifier(),
    "SupportVectorClassifier": SVC(),
    "DecisionTreeClassifier": DecisionTreeClassifier(),
    "RandomForestClassifier": RandomForestClassifier(),
    "GBoosting": GradientBoostingClassifier()
}
```

In [7]:
```
#base models with default parameters
from sklearn.model_selection import cross_val_score

for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=10
)
    print("Classifiers: ", classifier.__class__.__name__, "Has a trainin
g score of", round(training_score.mean(), 2) * 100, "% accuracy score")
```

```
Classifiers:  LogisticRegression Has a training score of 77.0 % accurac
y score
Classifiers:  KNeighborsClassifier Has a training score of 77.0 % accur
acy score
Classifiers:  SVC Has a training score of 76.0 % accuracy score
Classifiers:  DecisionTreeClassifier Has a training score of 78.0 % acc
uracy score
Classifiers:  RandomForestClassifier Has a training score of 84.0 % acc
uracy score
Classifiers:  GradientBoostingClassifier Has a training score of 85.0 %
accuracy score
```

In [8]:
```
from sklearn.model_selection import GridSearchCV
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10
, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train,y_train)
log_reg = grid_log_reg.best_estimator_
```

In [29]:
```
print(grid_log_reg.best_score_)
print(grid_log_reg.best_params_)
```

```
0.8541666666666666
{'C': 10, 'penalty': 'l1'}
```

In [9]:
```
knears_params = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['auto'
, 'ball_tree', 'kd_tree', 'brute']}
grid_knears = GridSearchCV(KNeighborsClassifier(), knears_params)
grid_knears.fit(X_train, y_train)
knears_neighbors = grid_knears.best_estimator_
```

In [30]:
```
print(grid_knears.best_score_)
print(grid_knears.best_params_)
```

```
0.7615740740740741
{'algorithm': 'auto', 'n_neighbors': 3}
```

In [39]:
```
svc_params = {'C': [6, 7, 8], 'gamma':[.87, .9, .93],'kernel': ['rbf','p
oly', 'sigmoid', 'linear'], 'coef0': [1.5, 2, 2.5]}
#svc_params = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly', 'sigmo
id', 'linear']}
grid_svc = GridSearchCV(SVC(probability=True), svc_params)
grid_svc.fit(X_train, y_train)
svc = grid_svc.best_estimator_
```

In [40]:
```
print(grid_svc.best_score_)
print(grid_svc.best_params_)
```

```
0.8518518518518519
{'C': 7, 'coef0': 2, 'gamma': 0.9, 'kernel': 'poly'}
```

In [11]:
```
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range
(2,4,1)), "min_samples_leaf": list(range(2,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train, y_train)
tree_clf = grid_tree.best_estimator_
```

In [32]:
```
print(grid_tree.best_score_)
print(grid_tree.best_params_)
```

```
0.8240740740740741
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 4}
```

In [45]:
```
rf_params = {'n_estimators': [200, 500],'max_features': ['auto', 'sqrt',
 'log2'],'max_depth' : [4,5,6,7,8],'criterion' :['gini', 'entropy']}
grid_rf = GridSearchCV(RandomForestClassifier(),rf_params)
grid_rf.fit(X_train, y_train)
rf = grid_rf.best_estimator_
```

In [46]:
```
print(grid_rf.best_score_)
print(grid_rf.best_params_)
```

```
0.8587962962962963
{'criterion': 'entropy', 'max_depth': 7, 'max_features': 'auto', 'n_est
imators': 200}
```

In [13]:
```
gboost_params = {"loss":["deviance"], "learning_rate": [0.075,0.1],"max_
depth":[8],"max_features":["log2","sqrt"],"criterion": ["friedman_mse",
 "mae"],
                 "subsample":[0.5, 0.8,1.0], "n_estimators":[10]}
grid_boost = GridSearchCV(GradientBoostingClassifier(),gboost_params)
grid_boost.fit(X_train, y_train)
boost = grid_rf.best_estimator_
```

In [34]:
```
print(grid_boost.best_score_)
print(grid_boost.best_params_)
```

```
0.8425925925925926
{'criterion': 'friedman_mse', 'learning_rate': 0.1, 'loss': 'deviance',
'max_depth': 8, 'max_features': 'log2', 'n_estimators': 10, 'subsampl
e': 0.5}
```

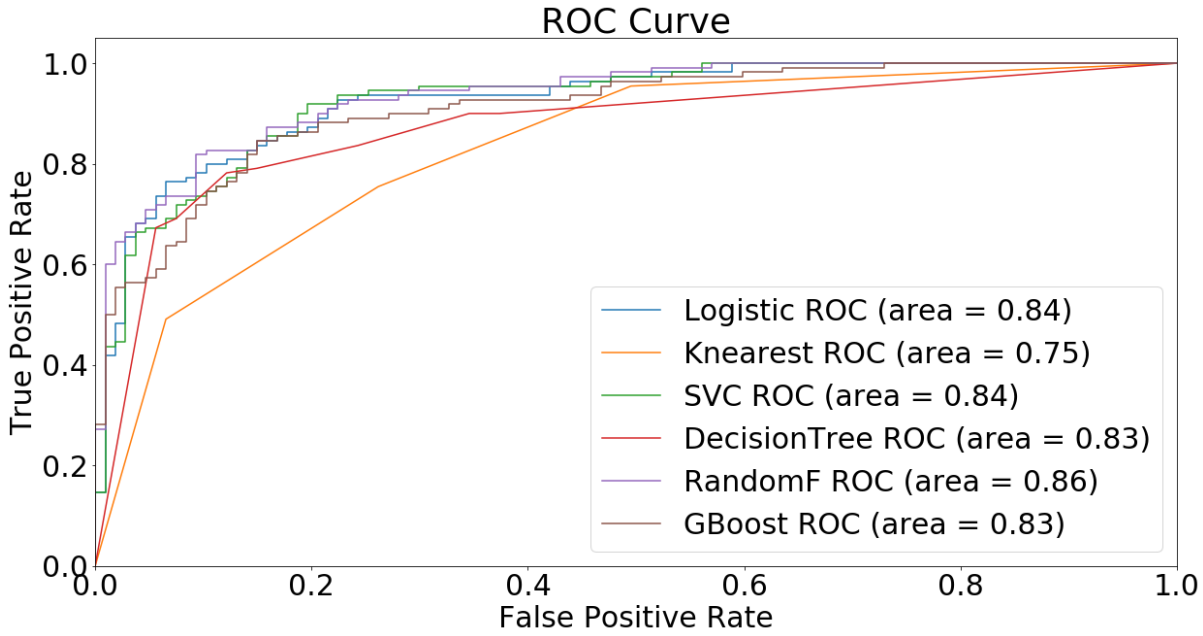
In [47]:
```
log_reg_score = cross_val_score(log_reg, X_train, y_train, cv=5)
print('Logistic Regression Cross Validation Score: on train %f' %round(l
og_reg_score.mean() * 100, 2))

svc_score = cross_val_score(svc, X_train, y_train, cv=5)
print('Support Vector Classifier Cross Validation Score on train %f' %ro
und(svc_score.mean() * 100, 2))

tree_score = cross_val_score(tree_clf, X_train, y_train, cv=5)
print('DecisionTree Classifier Cross Validation Score on train %f' %roun
d(tree_score.mean() * 100, 2))

knears_score = cross_val_score(knears_neighbors, X_train, y_train, cv=5)
print('Knears Neighbors Cross Validation Score on train %f' %round(knear
s_score.mean() * 100, 2))

rf_score = cross_val_score(rf, X_train, y_train, cv=5)
print('RandomForest Classifier Cross Validation Score on train %f' %roun
d(rf_score.mean() * 100, 2))

boost_score = cross_val_score(boost, X_train, y_train, cv=5)
print('GradientBoosting Cross Validation Score on train %f' %round(boost
_score.mean() * 100, 2))
```

```
Logistic Regression Cross Validation Score: on train 85.430000
Support Vector Classifier Cross Validation Score on train 84.730000
DecisionTree Classifier Cross Validation Score on train 80.340000
Knears Neighbors Cross Validation Score on train 76.860000
RandomForest Classifier Cross Validation Score on train 85.900000
GradientBoosting Cross Validation Score on train 85.550000
```

In [48]:
```
#normalize the data with training mean and std deviation
X_test  = normalizer.transform(X_test)

NU_log_reg_pred = grid_log_reg.best_estimator_.predict(X_test)

NU_knears_pred = grid_knears.best_estimator_.predict(X_test)

NU_svc_pred = grid_svc.best_estimator_.predict(X_test)

NU_tree_pred = grid_tree.best_estimator_.predict(X_test)

NU_rf_pred = grid_rf.best_estimator_.predict(X_test)

NU_gb_pred = grid_boost.best_estimator_.predict(X_test)

from  sklearn.metrics import accuracy_score
print('Logistic Regression accuracy on test : ',accuracy_score(y_test, N
U_log_reg_pred))
print('KNears Neighbors accuracy on test : ', accuracy_score(y_test, NU_
knears_pred))
print('Support Vector Classifier accuracy on test : ', accuracy_score(y_
test, NU_svc_pred))
print('Decision Tree Classifier accuracy on test : ', accuracy_score(y_t
est, NU_tree_pred))
print('Random Tree Classifier accuracy on test : ', accuracy_score(y_tes
t, NU_rf_pred))
print('Gradient Boosting accuracy on test : ', accuracy_score(y_test, NU
_gb_pred))
```

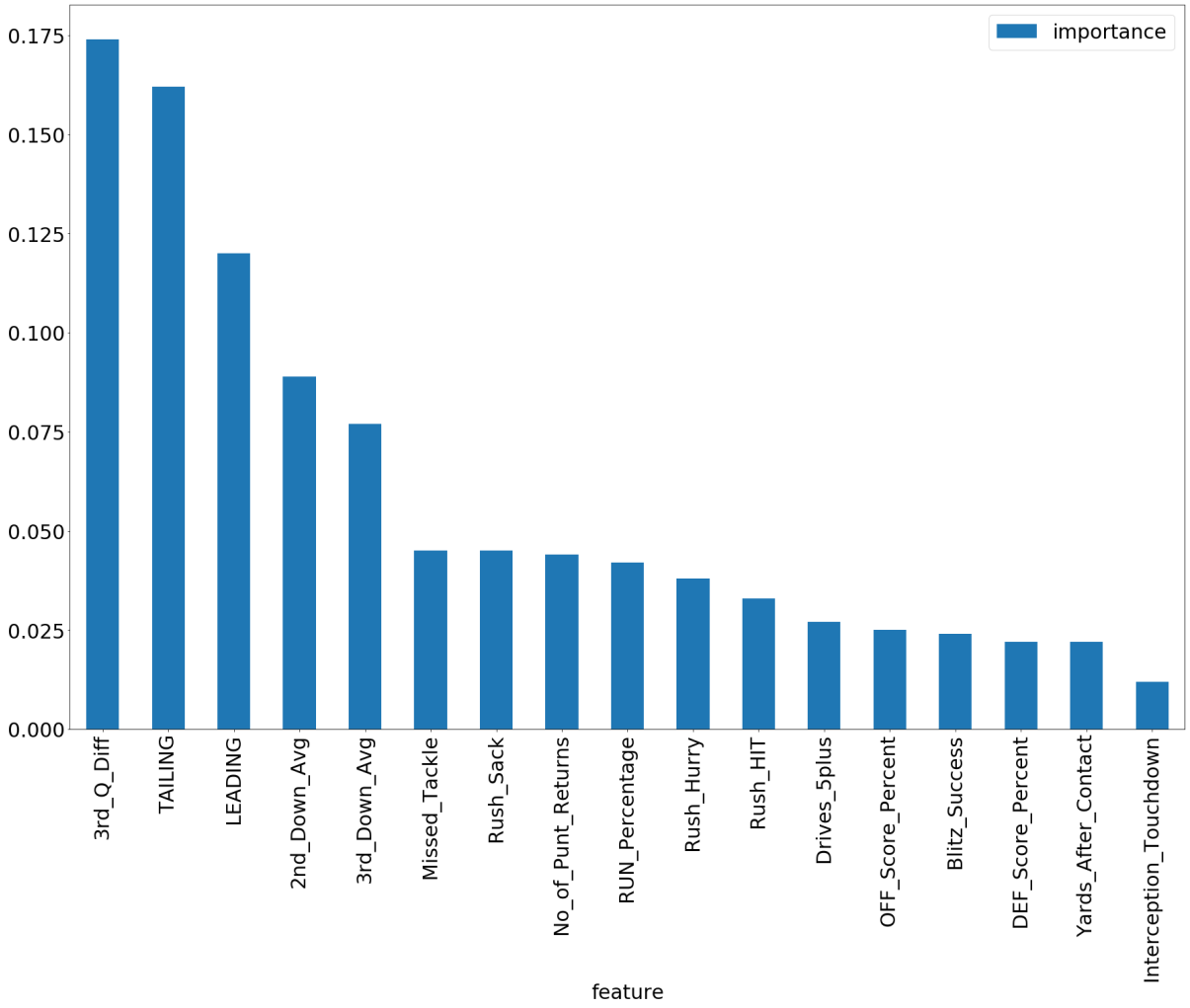
```
Logistic Regression accuracy on test :  0.8387096774193549
KNears Neighbors accuracy on test :  0.7465437788018433
Support Vector Classifier accuracy on test :  0.8433179723502304
Decision Tree Classifier accuracy on test :  0.8294930875576036
Random Tree Classifier accuracy on test :  0.8571428571428571
Gradient Boosting accuracy on test :  0.8248847926267281
```

In [52]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

plt.figure(figsize=(20,10))
models = {'Logistic':grid_log_reg,'Knearest':grid_knears,'SVC':grid_svc,
'DecisionTree':grid_tree,'RandomF':grid_rf,'GBoost':grid_boost}
for key,value in models.items():
    fpr, tpr, thresholds = roc_curve(y_test, value.predict_proba(X_test)
[:,1])
    auc = roc_auc_score(y_test,value.predict(X_test))
    plt.plot(fpr, tpr,label='%s ROC (area = %0.2f)' % (key, auc))

plt.plot()
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.savefig("ROC_Plot.png", bbox_inches='tight')
plt.show()
```



In [50]:
```
def win_lose_predict(game_fact, choosen_param,prev_prob = -1):

    columns= ['LEADING', '3rd_Q_Diff', 'No_of_DeepPass', 'DEF_Score_Perc
ent',
       'OFF_Score_Percent', '2nd_Down_Avg', '3rd_Down_Avg', 'Drives_5plu
s',
       'No_of_Punt_Returns', 'Interception_Touchdown', 'Rush_HIT',
       'Rush_Hurry', 'Rush_Sack', 'Missed_Tackle', 'Blitz_Success',
       'Yards_After_Catch', 'Yards_After_Contact', 'RUN_Percentage']
    game_pred = pd.DataFrame(columns =columns)
    game_pred = game_pred.append(pd.Series(game_fact, index=columns), ig
nore_index=True)

    normalize_game_fact = normalizer.transform(game_pred)
    predict_result = grid_rf.predict(normalize_game_fact)
    predict_prob = grid_rf.predict_proba(normalize_game_fact)

    return [predict_result,predict_prob[0][1]]
```

In [51]:
```
importances = pd.DataFrame({'feature':data.drop(['Label','EQUAL','No_of_
DeepPass','Yards_After_Catch'], axis=1).columns,'importance':np.round(rf
.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).set_
index('feature')
importances
plt.rcParams.update({'font.size': 30})
importances.plot.bar(figsize=(30,20))
plt.savefig("Importances.png", bbox_inches='tight')
```



In [53]:
```
averages = {}
for col in X.columns:
    if col not in ['Label','LEADING','TAILING'] :
        averages[col] = round(X[col].mean(),2)
averages
```

Out[53]:
```
{'3rd_Q_Diff': 5.93,
 'DEF_Score_Percent': 45.41,
 'OFF_Score_Percent': 33.56,
 '2nd_Down_Avg': 7.92,
 '3rd_Down_Avg': 7.13,
 'Drives_5plus': 6.87,
 'No_of_Punt_Returns': 8.46,
 'Interception_Touchdown': 0.36,
 'Rush_HIT': 3.22,
 'Rush_Hurry': 7.42,
 'Rush_Sack': 2.16,
 'Missed_Tackle': 8.64,
 'Blitz_Success': 47.51,
 'Yards_After_Contact': 2.76,
 'RUN_Percentage': 47.42}
```

In [54]:
```
importances
```

Out[54]:

feature	importance
3rd_Q_Diff	0.174
TAILING	0.162
LEADING	0.120
2nd_Down_Avg	0.089
3rd_Down_Avg	0.077
Missed_Tackle	0.045
Rush_Sack	0.045
No_of_Punt_Returns	0.044
RUN_Percentage	0.042
Rush_Hurry	0.038
Rush_HIT	0.033
Drives_5plus	0.027
OFF_Score_Percent	0.025
Blitz_Success	0.024
DEF_Score_Percent	0.022
Yards_After_Contact	0.022
Interception_Touchdown	0.012