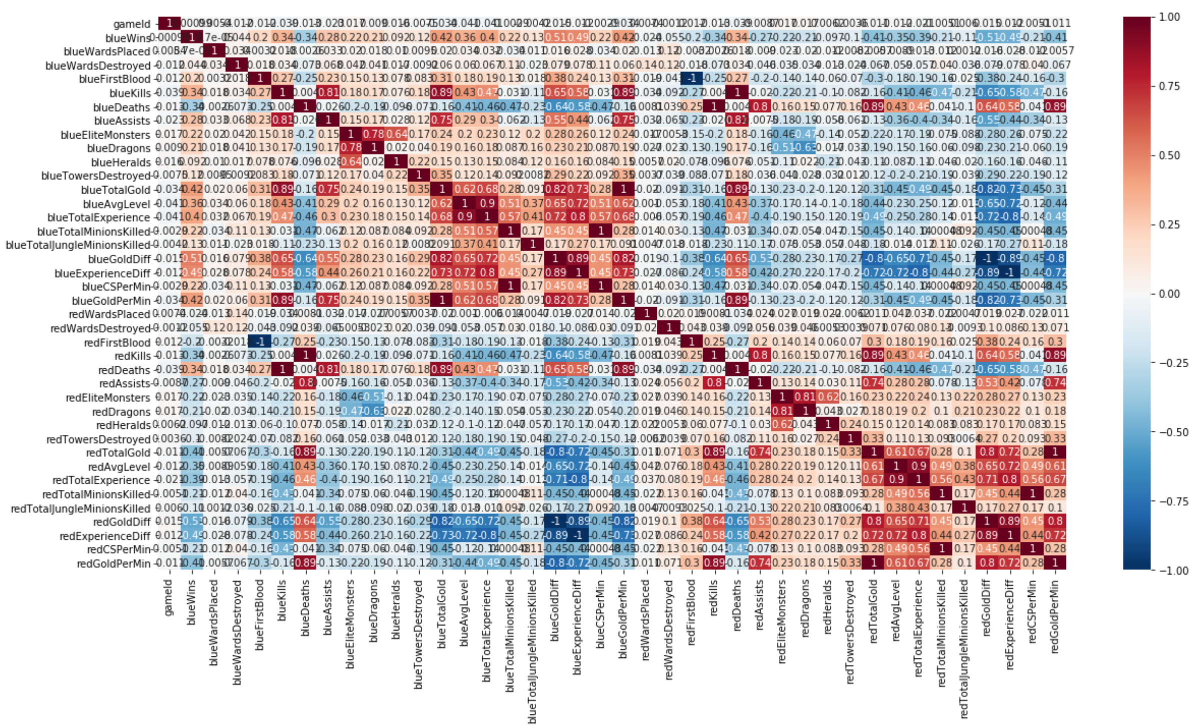
In [55]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

In [56]:
```
data = pd.read_csv('../datasets/high_diamond_ranked_10min.csv')
```

In [57]:
```
plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),cmap='RdBu_r',annot=True)
```

Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x14b94b1ee10>



In [ ]:

In [58]:
```
data = data.drop(columns=['gameId', 'redGoldPerMin', 'redKills', 'redDeaths', 'blueGoldPerMin',
                          'blueCSPerMin','redCSPerMin', 'redFirstBlood', 'redGoldDiff',
                          'redExperienceDiff', 'blueTotalGold', 'redTotalGold', 'blueTotalExperience',
                         'redTotalExperience'])

data['blueWardsPlacedDiff'] = data['blueWardsPlaced'] - data['redWardsPlaced']
data['blueWardsDestroyedDiff'] = data['blueWardsDestroyed'] - data['redWardsDestroyed']
data['blueAvgLevelDiff'] = data['blueAvgLevel'] - data['redAvgLevel']
data['blueAssistsDiff'] = data['blueAssists'] - data['redAssists']
data['blueTotalMinionsKilledDiff'] = data['blueTotalMinionsKilled'] - data['redTotalMinionsKilled']
data['blueTotalJungleMinionsKilledDiff'] = data['blueTotalJungleMinionsKilled'] - data['redTotalJungleMinionsKilled']
data['blueEliteMonstersDiff'] = data['blueEliteMonsters'] - data['redEliteMonsters']
data['blueDragonsDiff'] = data['blueDragons'] - data['redDragons']
data['blueHeraldsDiff'] = data['blueHeralds'] = data['redHeralds']
data['blueTowersDestroyedDiff'] = data['blueTowersDestroyed'] - data['redTowersDestroyed']
```

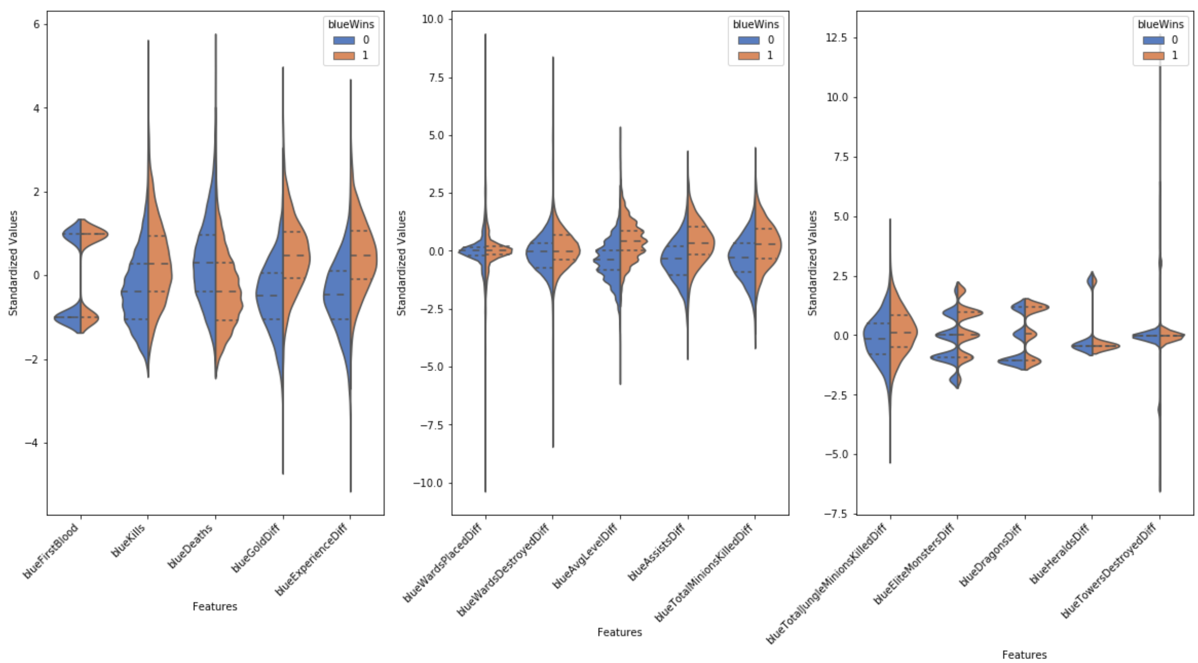
In [59]:
```
data = data.drop(columns= ['blueWardsPlaced','redWardsPlaced', 'blueWardsDestroyed', 'redWardsDestroyed',
                          'blueAvgLevel', 'redAvgLevel', 'blueAssists','redAssists', 'blueTotalMinionsKilled',
                          'redTotalMinionsKilled','blueTotalJungleMinionsKilled', 'redTotalJungleMinionsKilled',
                          'blueEliteMonsters', 'redEliteMonsters', 'redDragons', 'blueDragons', 'blueHeralds',
                          'redHeralds', 'blueTowersDestroyed','redTowersDestroyed'])
data.head()
```

Out[59]:

	blueWins	blueFirstBlood	blueKills	blueDeaths	blueGoldDiff	blueExperienceDiff	blueWardsPl
0	0	1	9	6	643	-8	
1	0	0	5	5	-2908	-1173	
2	0	0	7	11	-1172	-1033	
3	0	0	4	5	-1321	-7	
4	0	0	6	6	-1004	230	

In [60]:
```
def plot_violinplot(df,ax_key):
    df = pd.melt(df, id_vars='blueWins', var_name='Features', value_name='Standardized Values')
    sns.violinplot(x='Features', y='Standardized Values', hue='blueWins', data=df, split=True,
                   inner='quart', ax=ax[ax_key], palette='muted')
    fig.autofmt_xdate(rotation=45)

fig, ax = plt.subplots(1,3,figsize=(20,10))
df = data.loc[:, data.columns != 'blueWins']
df_std = StandardScaler().fit_transform(df)
df_std = pd.DataFrame(data = df_std, columns = df.columns)
df = pd.concat([data.blueWins, df_std.iloc[:, 0:5]], axis=1)
plot_violinplot(df,0)
df2 = pd.concat([data.blueWins, df_std.iloc[:, 5:10]], axis=1)
plot_violinplot(df2,1)
df3 = pd.concat([data.blueWins, df_std.iloc[:, 10:]], axis=1)
plot_violinplot(df3,2)
plt.show()
```



In [61]:
```
X = data.loc[:, data.columns != 'blueWins']
y = data['blueWins']
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.3, r
andom_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train2, y_train2, test_s
ize=0.3, random_state=43)
X_train
```

Out[61]:

	blueFirstBlood	blueKills	blueDeaths	blueGoldDiff	blueExperienceDiff	blueWardsPlacedDif
3189	1	6	7	457	-383	(
5249	1	10	5	1680	1254	-1
785	0	3	3	-5	-269	-21
8903	1	10	8	-1125	-1326	3
1092	1	10	5	4534	1707	7(
...	...	...	...	...	...	..
4300	0	10	8	496	187	(
7619	1	6	6	-695	-683	3
5857	0	7	6	1835	393	2
5409	1	12	7	2473	565	-4
3368	0	0	9	-4912	-4457	23

4840 rows × 15 columns

In [ ]:

In [62]:
```
from sklearn.metrics import classification_report
from matplotlib.colors import ListedColormap
```

In [63]:
```
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class examples
    for idx, cl in enumerate(np.unique(y)):
         plt.scatter(x=X[y == cl, 0],
                     y=X[y == cl, 1],
                     alpha=0.8,
                     c=colors[idx],
                     marker=markers[idx],
                     label=cl,
                     edgecolor='black')
    plt.xlabel('feature 1')
    plt.ylabel('feature 2')
    plt.legend()
```

In [64]:
```
clfL = DSELinearClassifier(activation='Logistic')
```

In [65]:
```
from sklearn.base import BaseEstimator
from sklearn.base import ClassifierMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.base import clone
from sklearn.pipeline import _name_estimators
import numpy as np
import operator
class MajorityVoteClassifier(BaseEstimator,
                             ClassifierMixin):
    """ A majority vote ensemble classifier

    Parameters
    ----------
    classifiers : array-like, shape = [n_classifiers]
      Different classifiers for the ensemble

    vote : str, {'classlabel', 'probability'}
      Default: 'classlabel'
      If 'classlabel' the prediction is based on
      the argmax of class labels. Else if
      'probability', the argmax of the sum of
      probabilities is used to predict the class label
      (recommended for calibrated classifiers).

    weights : array-like, shape = [n_classifiers]
      Optional, default: None
      If a list of `int` or `float` values are
      provided, the classifiers are weighted by
      importance; Uses uniform weights if `weights=None`.

    """
    def __init__(self, classifiers,
                 vote='classlabel', weights=None):

        self.classifiers = classifiers
        self.named_classifiers = {key: value for
                                  key, value in
                                  _name_estimators(classifiers)}
        self.vote = vote
        self.weights = weights

    def fit(self, X, y):
        """ Fit classifiers.

        Parameters
        ----------
        X : {array-like, sparse matrix},
            shape = [n_examples, n_features]
            Matrix of training examples.

        y : array-like, shape = [n_examples]
            Vector of target class labels.

        Returns
        -------
        self : object
```

```
        """
        if self.vote not in ('probability', 'classlabel'):
            raise ValueError("vote must be 'probability'"
                             "or 'classlabel'; got (vote=%r)"
                             % self.vote)
        if self.weights and len(self.weights) != len(self.classifiers):
            raise ValueError("Number of classifiers and weights"
                             "must be equal; got %d weights,"
                             "%d classifiers"
                             % (len(self.weights),
                             len(self.classifiers)))
        # Use LabelEncoder to ensure class labels start
        # with 0, which is important for np.argmax
        # call in self.predict
        self.lablenc_ = LabelEncoder()
        self.lablenc_.fit(y)
        self.classes_ = self.lablenc_.classes_
        self.classifiers_ = []
        for clf in self.classifiers:
            fitted_clf = clone(clf).fit(X,
                                  self.lablenc_.transform(y))
            self.classifiers_.append(fitted_clf)
        return self

    def predict(self, X):
        """ Predict class labels for X.

        Parameters
        ----------
        X : {array-like, sparse matrix},
            Shape = [n_examples, n_features]
            Matrix of training examples.

        Returns
        ----------
        maj_vote : array-like, shape = [n_examples]
            Predicted class labels.

        """
        if self.vote == 'probability':
            maj_vote = np.argmax(self.predict_proba(X), axis=1)
        else: # 'classlabel' vote

            # Collect results from clf.predict calls
            predictions = np.asarray([clf.predict(X)
                                      for clf in
                                      self.classifiers_]).T

            maj_vote = np.apply_along_axis(lambda x: np.argmax(
                                           np.bincount(x,
                                           weights=self.weights)),
                                           axis=1,
                                           arr=predictions)
        maj_vote = self.lablenc_.inverse_transform(maj_vote)
        return maj_vote
```

```
    def predict_proba(self, X):
        """ Predict class probabilities for X.

        Parameters
        ----------
        X : {array-like, sparse matrix},
            shape = [n_examples, n_features]
            Training vectors, where
            n_examples is the number of examples and
            n_features is the number of features.

        Returns
        ----------
        avg_proba : array-like,
            shape = [n_examples, n_classes]
            Weighted average probability for
            each class per example.

        """
        probas = np.asarray([clf.predict_proba(X)
                             for clf in self.classifiers_])
        avg_proba = np.average(probas, axis=0,
                               weights=self.weights)
        return avg_proba

    def get_params(self, deep=True):
        """ Get classifier parameter names for GridSearch"""
        if not deep:
            return super(MajorityVoteClassifier,
                         self).get_params(deep=False)
        else:
            out = self.named_classifiers.copy()
            for name, step in self.named_classifiers.items():
                for key, value in step.get_params(
                        deep=True).items():
                    out['%s__%s' % (name, key)] = value
            return out
```

In [66]:

```
clfL.fit(X_train, y_train,learning_rate=0.001)
y_pred = clfL.predict(X_test)
print(classification_report(y_test, y_pred))
#plot_decision_regions(X_test, y_test, clfL)
```

```
              precision    recall  f1-score   support

           0       0.75      0.76      0.75      1073
           1       0.74      0.73      0.74      1002

    accuracy                           0.75      2075
   macro avg       0.75      0.75      0.75      2075
weighted avg       0.75      0.75      0.75      2075


C:\Users\XChen\Anaconda3\lib\site-packages\ipykernel_launcher.py:71: RuntimeW
arning: divide by zero encountered in log
```

In [33]:
```
from sklearn.pipeline import Pipeline

clf1 = LogisticRegression(random_state=42)
clf2 = DecisionTreeClassifier(random_state=42)
clf3 = KNeighborsClassifier()
clf4 = RandomForestClassifier(random_state=42)
clf5 = SVC(random_state=42, probability = True)

pipe1 = Pipeline([['sc', StandardScaler()], ['clf', clf1]])
pipe3 = Pipeline([['sc', StandardScaler()], ['clf', clf3]])
pipe5 = Pipeline([['sc', StandardScaler()], ['clf', clf5]])
mv_clf = MajorityVoteClassifier(classifiers=[pipe1, clf2, pipe3, clf4, pipe5])

clf_labels = ['Logistic regression', 'Decision tree', 'KNN', 'RandomForestClas
sifier', 'SVM', 'mv_clf']

print('10-fold cross validation:\n')
for clf, label in zip([pipe1, clf2, pipe3, clf4, pipe5, mv_clf], clf_labels):
    scores = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=10, scori
ng='roc_auc')
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
10-fold cross validation:

ROC AUC: 0.80 (+/- 0.03) [Logistic regression]
ROC AUC: 0.62 (+/- 0.03) [Decision tree]
ROC AUC: 0.73 (+/- 0.03) [KNN]
ROC AUC: 0.78 (+/- 0.03) [RandomForestClassifier]
ROC AUC: 0.78 (+/- 0.03) [SVM]
ROC AUC: 0.77 (+/- 0.03) [mv_clf]
```

In [ ]: