CUSTOMER CHURN PREDICTION

**TEAM LEADER:** GURU VARSHENI A

**REGISTER NO:** 211521244015

**TEAM MEMBER:** DHANUSHREE.S

**REGISTER NO:** 211521244011

**TEAM MEMBER:** ISHA.K

**REGISTER NO:** 211521244018

**TEAM MEMBER:** POOJA.S

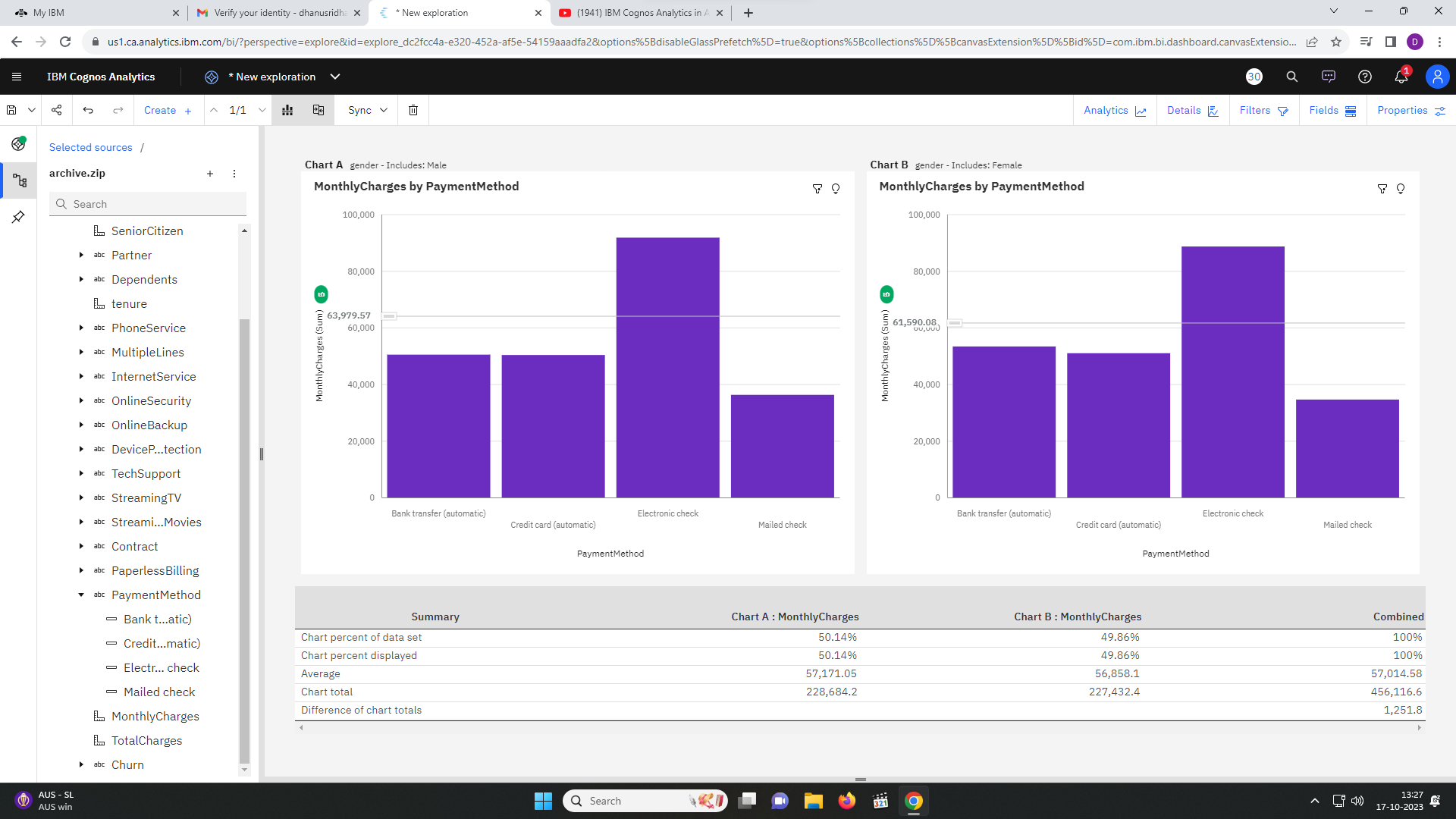
**REGISTER NO:** 211521244039

***CREATING VISUALIZATIONS USING IBM COGNOS:***

Visualizations communicate comparisons, relationships, and trends. They emphasize and clarify numbers. IBM Cognos Analytics provides analytic insights that help you to detect and validate important relationships and meaningful differences based on the data that is presented by the visualization. It also provides a number of recommended visualizations based on the data that you are working with.

Given below is the representation of **MonthlyCharges** by **PaymentMethod** in the form of column charts. We take PaymentMethod as the X-axis and the MonthlyCharges as the Y-axis.

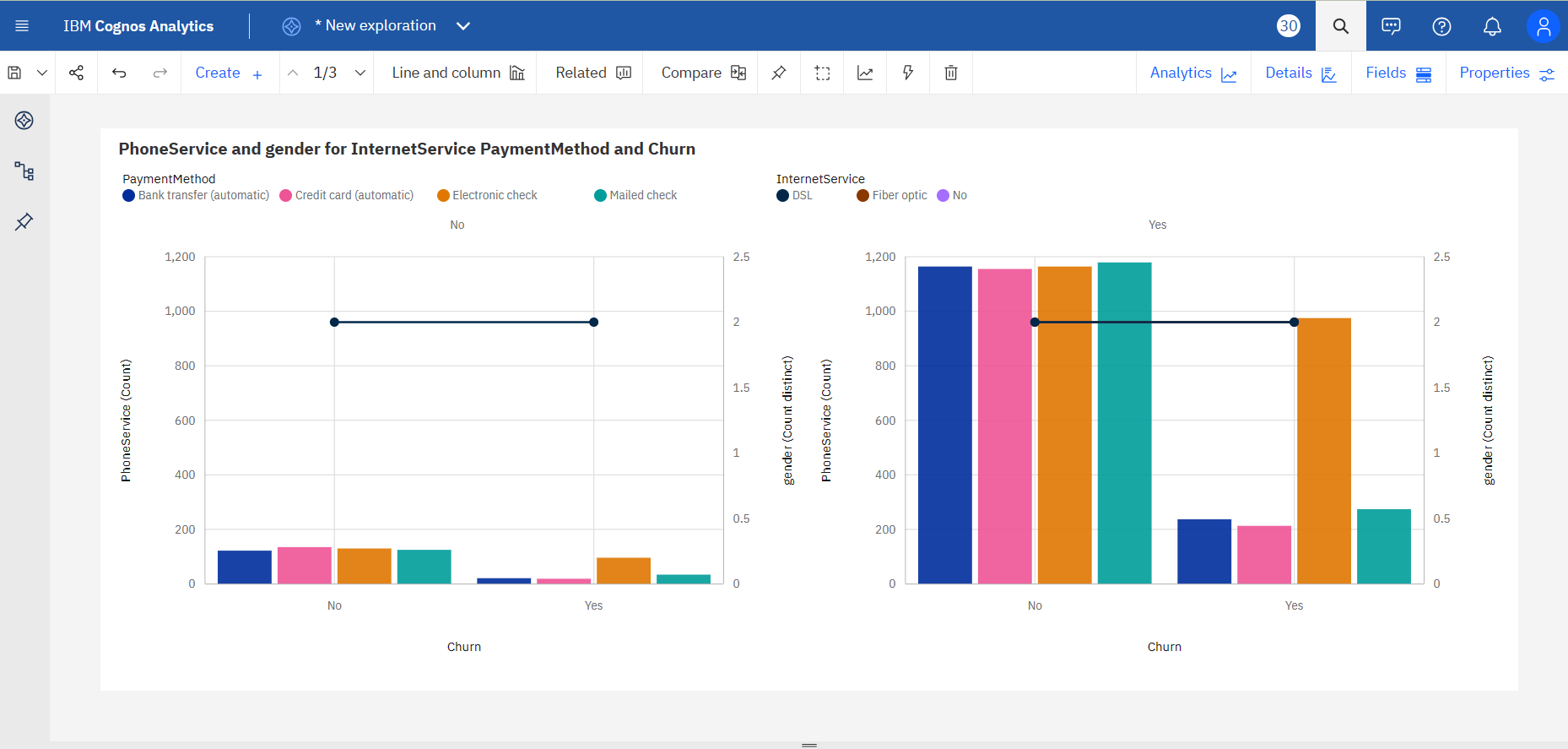
In IBM Cognos we can create your own comparison to analyze the data between two visualizations or start with a recommended comparison. In either case, a summary of key information and differences between the two visualizations is generated.

We compare two charts with the filter of gender where the right side includes only the male gender and the left one includes only the female gender.

Next, we have a Line and Column chart which shows the gender and PhoneService by Churn. The total number of results for **PhoneService**, across all **Churn**, is over seven thousand.

No is the most frequently occurring category of **Churn** with a count of 5174 items with **PhoneService** values (73.5 % of the total).Yes is the most frequently occurring category of **PhoneService** with a count of 6361 items with **PhoneService** values (90.3 % of the total).

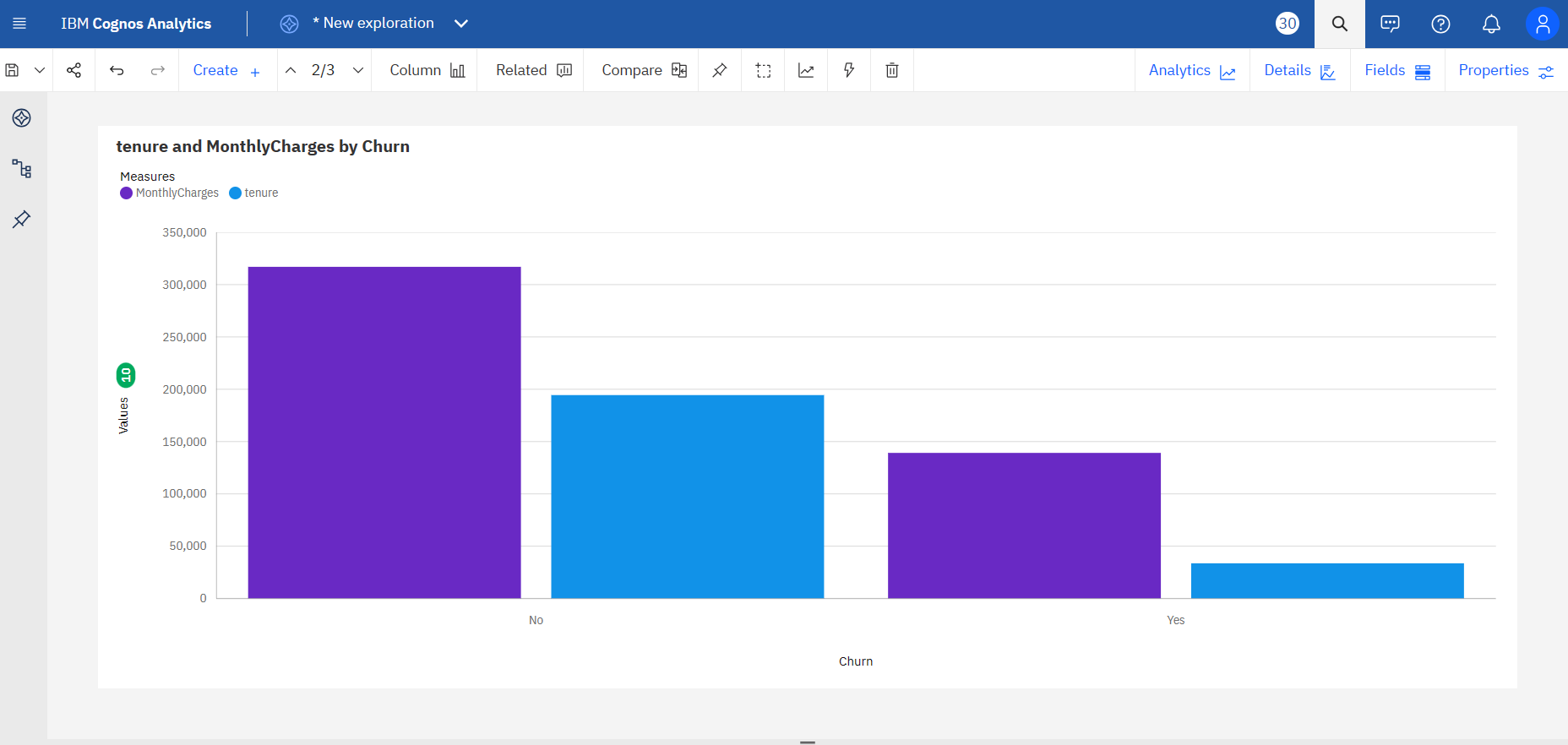
The overall number of results for **gender** is over seven thousand. No is the most frequently occurring category of **Churn** with a count of 5174 items with **gender** values (73.5 % of the total). Yes is the most frequently occurring category of **PhoneService** with a count of 6361 items with **gender** values (90.3 % of the total).



Next, we have a Column chart that represents the **tenure and MonthlyCharges by Churn.** The total number of results for tenure, across all Churn, is over seven thousand. Over all values of Churn, the average of tenure is 32.37. The total number of results for MonthlyCharges, across all Churn, is over seven thousand. Over all values of Churn, the average of MonthlyCharges is 64.76.

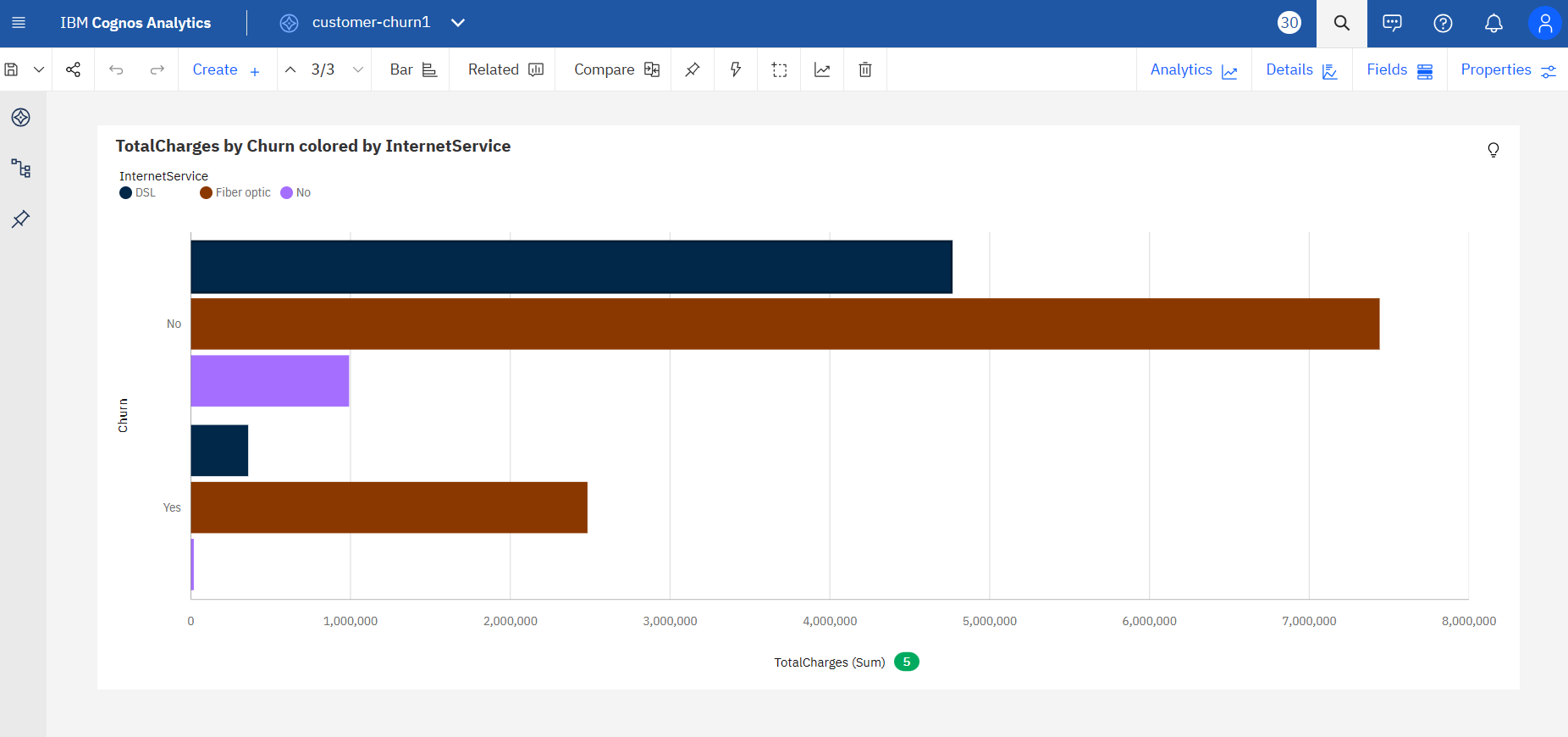
**MonthlyCharges** ranges from over 139 thousand, when **Churn** is Yes, to nearly 317 thousand, when **Churn** is No. **tenure** ranges from almost 34 thousand, when **Churn** is Yes, to over 194 thousand, when **Churn** is No.

No is the most frequently occurring category of **Churn** with a count of 5174 items with **tenure** values (73.5 % of the total). No is the most frequently occurring category of **Churn** with a count of 5174 items with **MonthlyCharges** values (73.5 % of the total).



This visualisation shows a bar chart representing the **TotalCharges by Churn colored by InternetService.** Across all values of Churn and InternetService, the sum of TotalCharges is over sixteen million. The summed values of TotalCharges range from nearly 20 thousand to over 7.4 million. TotalCharges is unusually high when the combination of Churn and InternetService is No and Fiber optic.

**TotalCharges** is unusually high when **InternetService** is Fiber optic. For **TotalCharges**, the most significant value of **Churn** is No, whose respective **TotalCharges** values add up to over thirteen million, or 82.2 % of the total. For **TotalCharges**, the most significant value of **InternetService** is Fiber optic, whose respective **TotalCharges** values add up to over 9.9 million, or 61.8 % of the total.



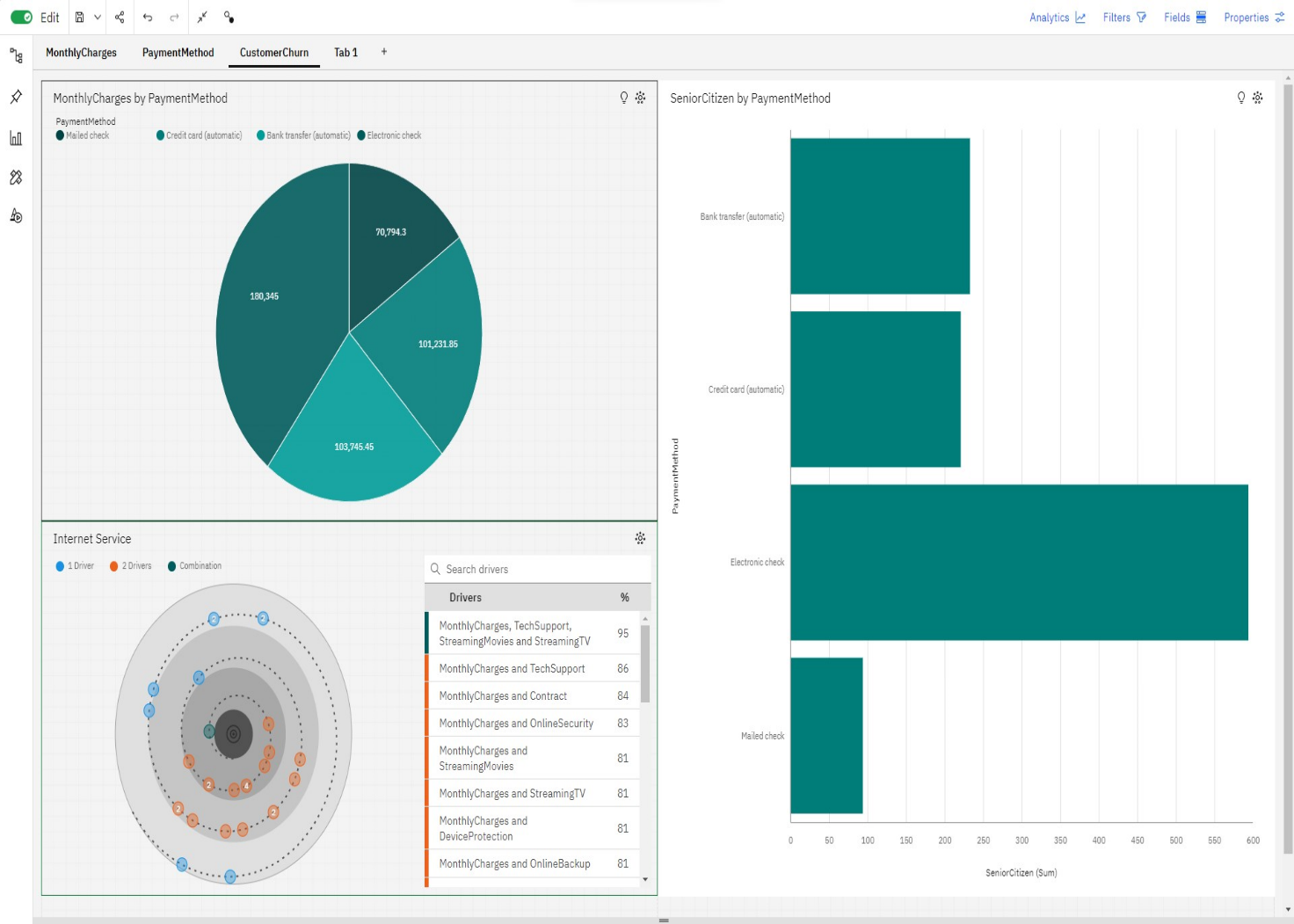
**DASHBOARDS IN IBM COGNOS:**

IBM Cognos Analytics is a business intelligence tool that offers many features, including dashboards. A dashboard is a collection of visualizations that provide insight into data. IBM Cognos Analytics allows you to create dashboards with powerful visualization capabilities that help you discover patterns and relationships in your data. You can also share these dashboards with others.

Given below is a dashboard which has three visualisations. The first one is a pie chart. Pie charts are useful for highlighting proportions. They use segments of a circle to show the relationship of parts to the whole. This pie chart represents MonthlyCharges by PaymentMethod.

Next, we have the spiral visualisation which shows the Internet Services. A spiral visualization shows you the key drivers, or predictors, for a given target. The closer the driver is to the centre, the stronger that driver is.

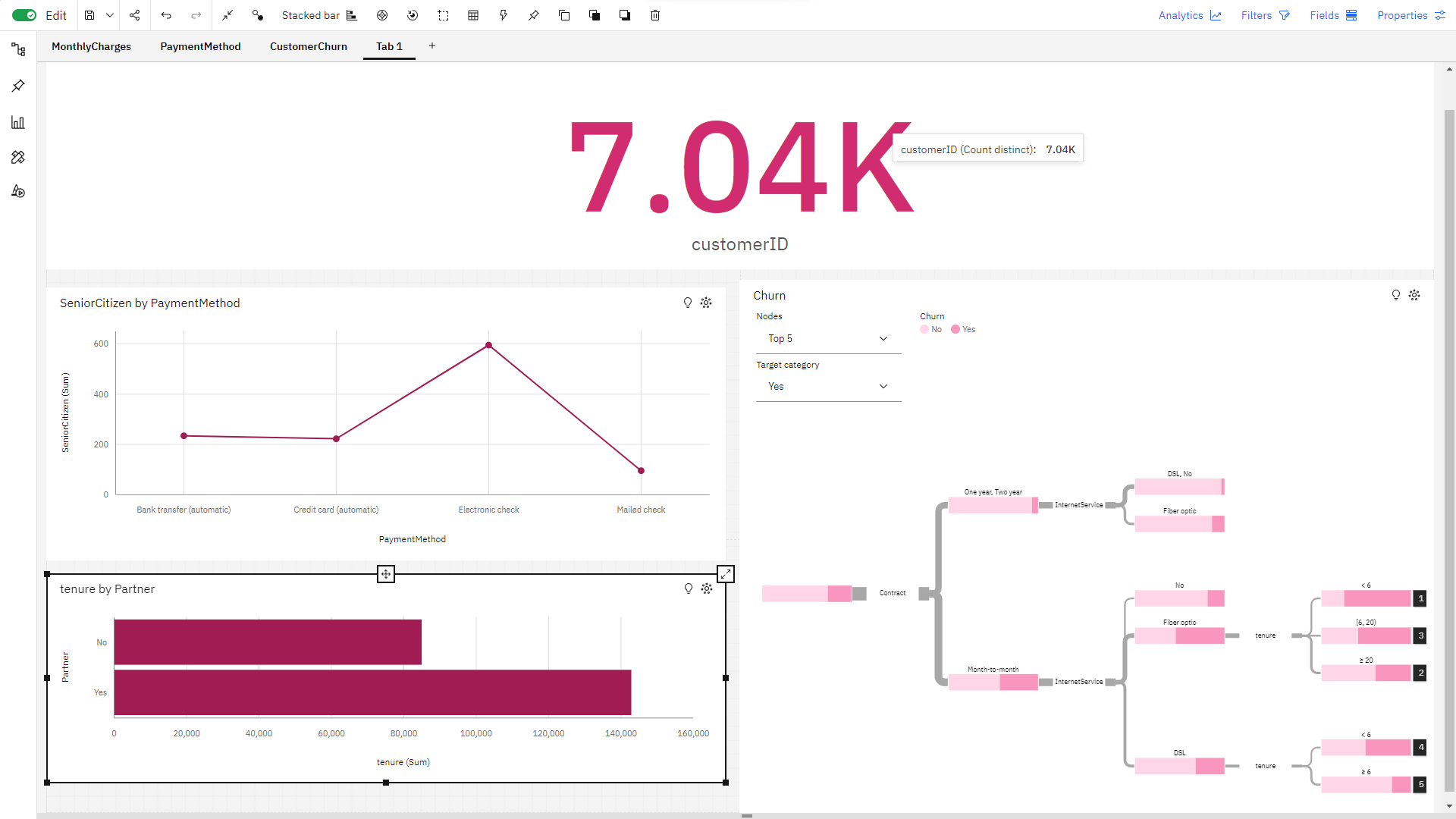
Lastly, we have SeniorCitizen by PaymentMethod in the Y-axis and the X-axis respectively. It shows that SeniorCitzen mostly use the Electronic check as the payment method.

****

Given below is another dashboard which shows the count of the distinct customerID which is estimated to be 7.04K. Also, we have three other visualisations, one of which is the line chart which represents SeniorCitizen by PaymentMethod in the Y-axis and the X-axis respectively. It shows that SeniorCitzen mostly use the Electronic check as the payment method.

Next, we have a bar chart that shows tenure by Partner in the Y-axis and the X-axis respectively. It shows that people who have partners have more tenure than those who don’t.

Finally, we have a decision tree which shows the top 5 nodes and the the category is set to yes.

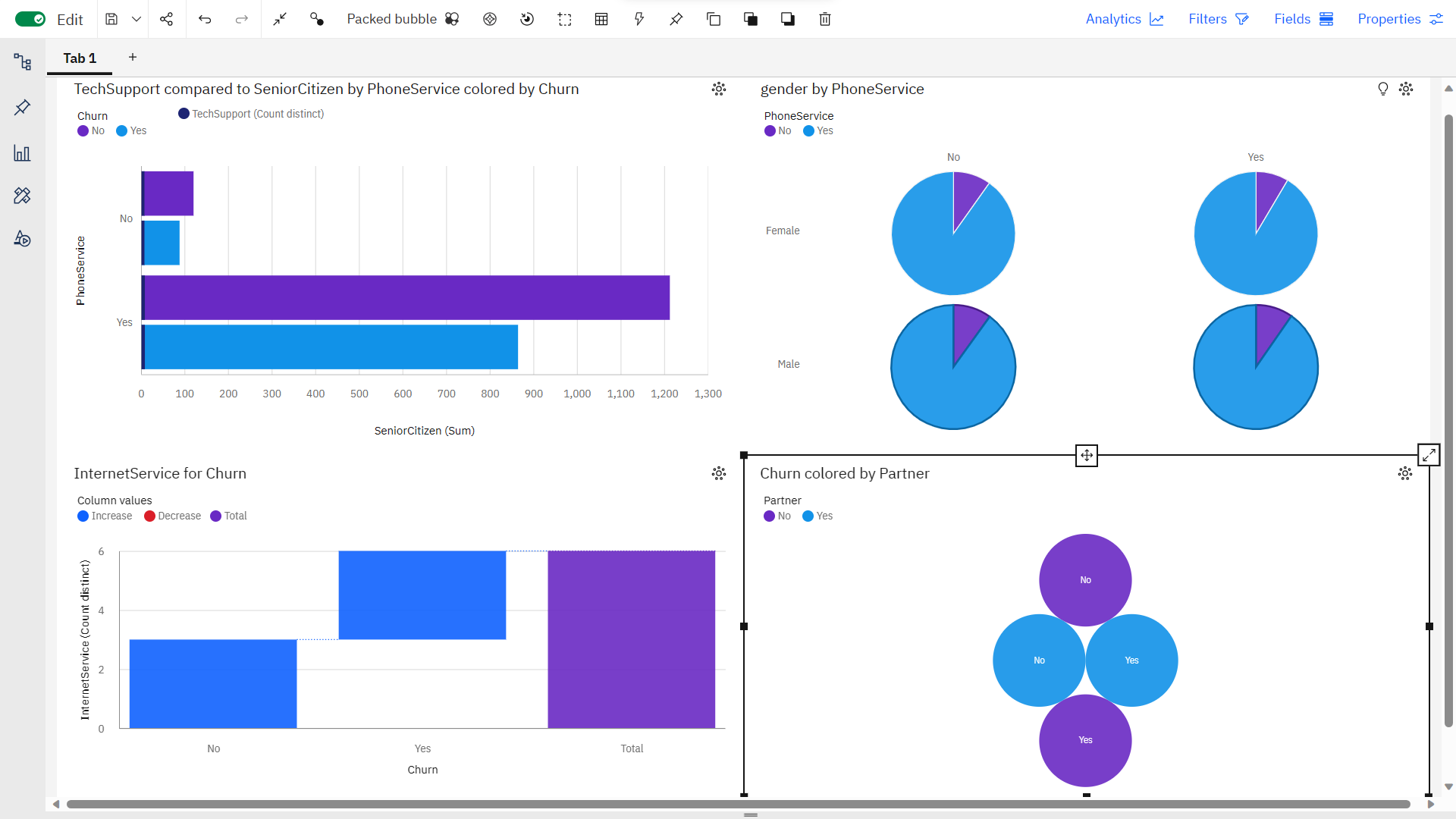


Given below is another dashboard which contains a bar chart, pie chart, a waterfall chart and a packed bubble chart representing various key factors. The bar chart represents the TechSupport compared to SeniorCitizen by PhoneService colored by Churn. Churn No has the highest values of both TechSupport and TotalCharges. Over all values of PhoneService and Churn, the sum of SeniorCitizen is almost 2500.

The pie chart represents gender by PhoneService. Churn No has the highest values of both gender and TotalCharges. No is the most frequently occurring category of Churn with a count of 10,348 items with gender values (73.5 % of the total).Yes is the most frequently occurring category of PhoneService with a count of 12,722 items with gender values.

The waterfall chart represents the InternetService for Churn. Churn No has the highest values of both InternetService and TotalCharges.No is the most frequently occurring category of Churn with a count of 10,348 items with InternetService values (73.5 % of the total). The total number of results for InternetService, across all Churn, is over fourteen thousand.

Finally, the packed bubble chart represents the Churn colored by Partner. Churn No has the highest total TotalCharges due to Partner Yes.Partner Yes has the highest TotalCharges at almost 21 million, out of which Churn No contributed the most at over 17 million.



**MACHINE LEARNING ALGORITHMS:**

## **Evaluating the model Results**

acc\_results =[]auc\_results =[]names = []result\_col = ["Algorithm", "ROC AUC Mean", "ROC AUC STD", "Accuracy Mean", "Accuracy STD"]model\_results = pd.DataFrame(columns = result\_col)i=0 *# K- fold cross validation***for** name, model **in** models: names.append(name) kfold = model\_selection.KFold(n\_splits=10, random\_state=0) cv\_acc\_results = model\_selection.cross\_val\_score(model, X\_train, y\_train, cv = kfold, scoring="accuracy") cv\_auc\_results = model\_selection.cross\_val\_score(model, X\_train, y\_train, cv = kfold, scoring="roc\_auc") acc\_results.append(cv\_acc\_results) auc\_results.append(cv\_auc\_results) model\_results.loc[i] = [name, round(cv\_auc\_results.mean()\*100,2), round(cv\_auc\_results.std()\*100,2), round(cv\_acc\_results.mean()\*100,2), round(cv\_acc\_results.std()\*100,2)] i+=1model\_results.sort\_values(by = ['ROC AUC Mean'], ascending=False)

Algorithm ROC AUC Mean ROC AUC STD Accuracy Mean \9 Voting Classifier 84.93 1.39 80.23 8 Gradient boost classifier 84.72 1.42 79.72 7 Adaboost 84.55 1.25 80.09 0 Logistic Regression 84.39 1.47 74.38 1 SVC 82.99 2.07 79.11 6 Random Forest 82.75 2.01 78.67 4 Gaussian NB 82.32 1.28 75.38 2 Kernel SVM 79.65 2.12 79.26 3 KNN 77.14 1.43 75.90 5 Decision Tree Classifier 66.67 1.07 73.73  Accuracy STD 9 1.89 8 1.95 7 1.77 0 1.94 1 2.01 6 1.98 4 1.23 2 1.67 3 2.01 5 1.12

fig = plt.figure(figsize=(25,15))ax = fig.add\_subplot(111)plt.boxplot(acc\_results)ax.set\_xticklabels(names)plt.ylabel('ROC AUC Score\n',horizontalalignment="center",fontstyle = "normal",fontsize = "large", fontfamily = "sans-serif")plt.xlabel('\n Baseline Classification Algorithms\n',horizontalalignment="center",fontstyle = "normal",fontsize = "large", fontfamily = "sans-serif")plt.title('Accuracy Score Comparison \n',horizontalalignment="center", fontstyle = "normal",fontsize = "22", fontfamily = "sans-serif")plt.xticks(rotation=0, horizontalalignment="center")plt.yticks(rotation=0, horizontalalignment="right")plt.show()

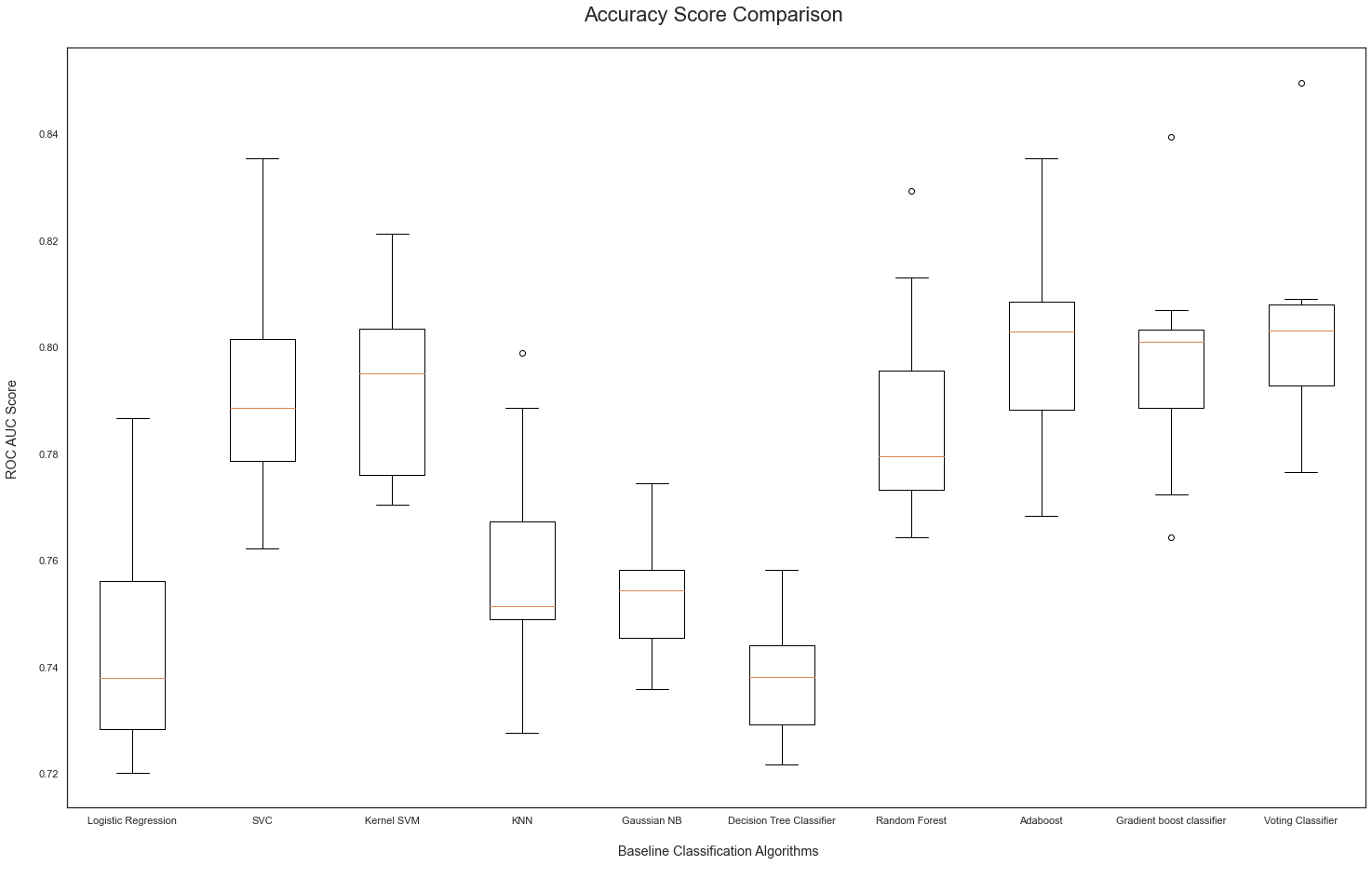
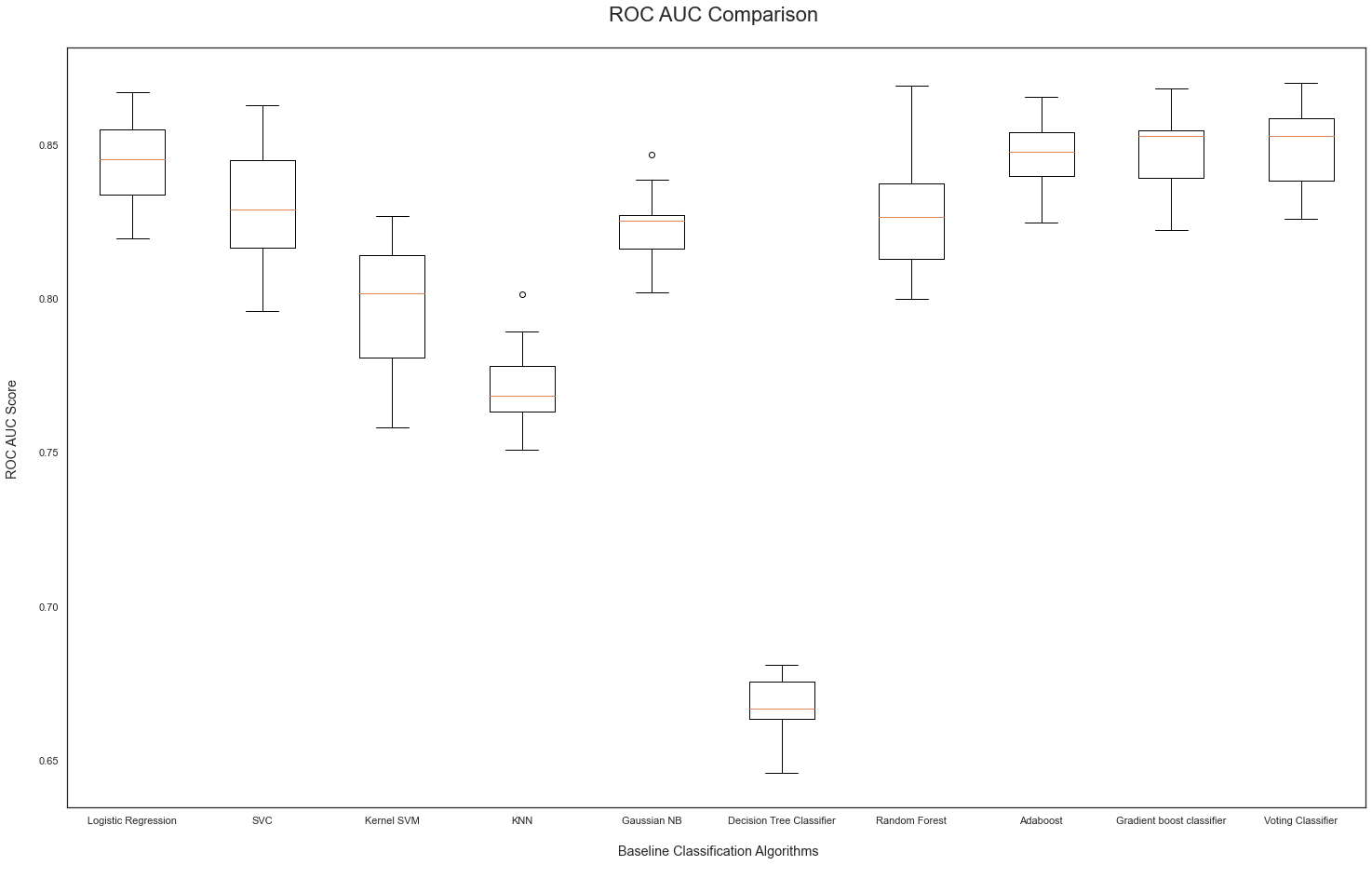
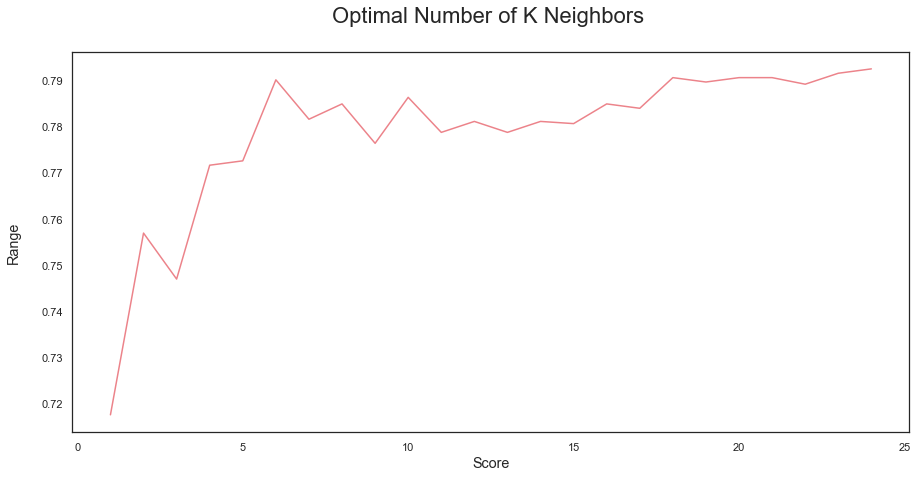


fig = plt.figure(figsize=(25,15))ax = fig.add\_subplot(111)plt.boxplot(auc\_results)ax.set\_xticklabels(names)plt.ylabel('ROC AUC Score\n',horizontalalignment="center",fontstyle = "normal",fontsize = "large", fontfamily = "sans-serif")plt.xlabel('\n Baseline Classification Algorithms\n',horizontalalignment="center",fontstyle = "normal",fontsize = "large", fontfamily = "sans-serif")plt.title('ROC AUC Comparison \n',horizontalalignment="center", fontstyle = "normal",fontsize = "22", fontfamily = "sans-serif")plt.xticks(rotation=0, horizontalalignment="center")plt.yticks(rotation=0, horizontalalignment="right")plt.show()



**KNN**

fig = plt.figure(figsize=(15, 7))plt.plot(range(1,25),score\_array, color = '#ec838a')plt.ylabel('Range\n',horizontalalignment="center",fontstyle = "normal", fontsize = "large", fontfamily = "sans-serif")plt.xlabel('Score\n',horizontalalignment="center",fontstyle = "normal", fontsize = "large", fontfamily = "sans-serif")plt.title('Optimal Number of K Neighbors \n',horizontalalignment="center", fontstyle = "normal",fontsize = "22", fontfamily = "sans-serif") *#plt.legend(loc='top right', fontsize = "medium")*plt.xticks(rotation=0, horizontalalignment="center")plt.yticks(rotation=0, horizontalalignment="right")plt.show()



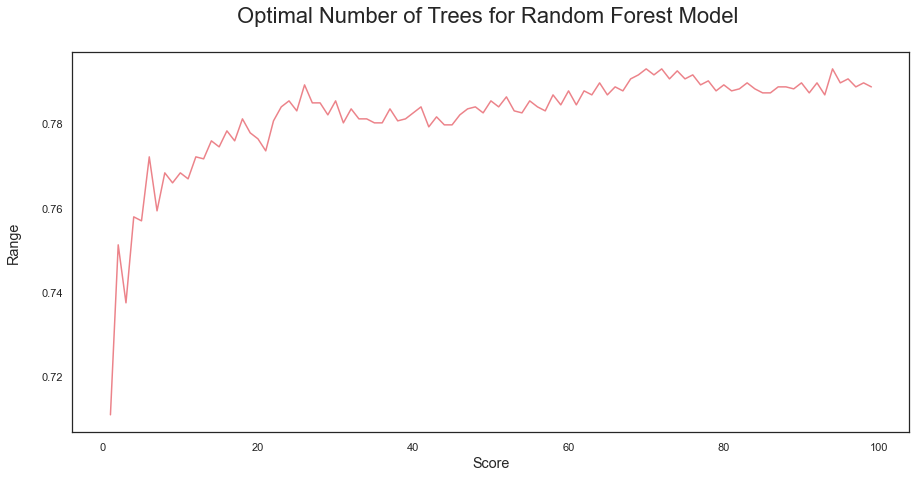
**Random Forest**

score\_array = []**for** each **in** range(1,100): rf\_loop = RandomForestClassifier(n\_estimators = each, random\_state = 1) rf\_loop.fit(X\_train,y\_train) score\_array.append(rf\_loop.score(X\_test,y\_test))

**for** i,j **in** enumerate(score\_array): print(i+1,":",j)

1 : 0.71090047393364932 : 0.75118483412322283 : 0.73744075829383884 : 0.75781990521327015 : 0.75687203791469196 : 0.77203791469194327 : 0.75924170616113748 : 0.76824644549763039 : 0.765876777251184910 : 0.768246445497630311 : 0.766824644549763112 : 0.772037914691943213 : 0.77156398104265414 : 0.775829383886255915 : 0.774407582938388616 : 0.778199052132701517 : 0.775829383886255918 : 0.78104265402843619 : 0.777725118483412320 : 0.77630331753554521 : 0.773459715639810422 : 0.780568720379146923 : 0.783886255924170624 : 0.785308056872037925 : 0.782938388625592426 : 0.789099526066350727 : 0.784834123222748828 : 0.784834123222748829 : 0.781990521327014230 : 0.785308056872037931 : 0.780094786729857832 : 0.783412322274881633 : 0.78104265402843634 : 0.78104265402843635 : 0.780094786729857836 : 0.780094786729857837 : 0.783412322274881638 : 0.780568720379146939 : 0.78104265402843640 : 0.782464454976303341 : 0.783886255924170642 : 0.779146919431279743 : 0.781516587677725144 : 0.779620853080568745 : 0.779620853080568746 : 0.781990521327014247 : 0.783412322274881648 : 0.783886255924170649 : 0.782464454976303350 : 0.785308056872037951 : 0.783886255924170652 : 0.786255924170616153 : 0.782938388625592454 : 0.782464454976303355 : 0.785308056872037956 : 0.783886255924170657 : 0.782938388625592458 : 0.786729857819905259 : 0.784360189573459860 : 0.787677725118483461 : 0.784360189573459862 : 0.787677725118483463 : 0.786729857819905264 : 0.789573459715639965 : 0.786729857819905266 : 0.788625592417061667 : 0.787677725118483468 : 0.79052132701421869 : 0.791469194312796270 : 0.792890995260663571 : 0.791469194312796272 : 0.792890995260663573 : 0.79052132701421874 : 0.792417061611374475 : 0.79052132701421876 : 0.791469194312796277 : 0.789099526066350778 : 0.790047393364928979 : 0.787677725118483480 : 0.789099526066350781 : 0.787677725118483482 : 0.788151658767772583 : 0.789573459715639984 : 0.788151658767772585 : 0.787203791469194386 : 0.787203791469194387 : 0.788625592417061688 : 0.788625592417061689 : 0.788151658767772590 : 0.789573459715639991 : 0.787203791469194392 : 0.789573459715639993 : 0.786729857819905294 : 0.792890995260663595 : 0.789573459715639996 : 0.79052132701421897 : 0.788625592417061698 : 0.789573459715639999 : 0.7886255924170616

fig = plt.figure(figsize=(15, 7))plt.plot(range(1,100),score\_array, color = '#ec838a')plt.ylabel('Range\n',horizontalalignment="center",fontstyle = "normal", fontsize = "large",fontfamily = "sans-serif")plt.xlabel('Score\n',horizontalalignment="center",fontstyle = "normal", fontsize = "large",fontfamily = "sans-serif")plt.title('Optimal Number of Trees for Random Forest Model \n',horizontalalignment="center", fontstyle = "normal", fontsize = "22", fontfamily = "sans-serif") *#plt.legend(loc='top right', fontsize = "medium")*plt.xticks(rotation=0, horizontalalignment="center")plt.yticks(rotation=0, horizontalalignment="right")plt.show()



**2nd Iteration**

*#evaluation of results***def** model\_evaluation(y\_test, y\_pred, model\_name): acc = accuracy\_score(y\_test, y\_pred) prec = precision\_score(y\_test, y\_pred) rec = recall\_score(y\_test, y\_pred) f1 = f1\_score(y\_test, y\_pred) f2 = fbeta\_score(y\_test, y\_pred, beta = 2.0) results = pd.DataFrame([[model\_name, acc, prec, rec, f1, f2]], columns = ["Model", "Accuracy", "Precision", "Recall", "F1 SCore", "F2 Score"]) results = results.sort\_values(["Precision", "Recall", "F2 Score"], ascending = False) **return** results

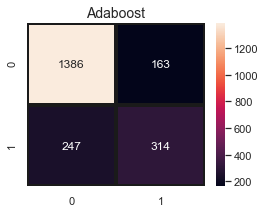
*# Logistic regression*classifier = LogisticRegression(random\_state=0)classifier.fit(X\_train, y\_train)y\_pred = classifier.predict(X\_test) *#SVC*classifier2 = SVC(kernel = 'linear', random\_state = 0)classifier2.fit(X\_train, y\_train)y\_pred2 = classifier2.predict(X\_test) *#knn*classifier3 = KNeighborsClassifier(n\_neighbors=22, metric="minkowski", p=2)classifier3.fit(X\_train, y\_train)y\_pred3 = classifier3.predict(X\_test) *#Kernel SVM*classifier4 = SVC(kernel="rbf", random\_state =0)classifier4.fit(X\_train, y\_train)y\_pred4 = classifier4.predict(X\_test) *#Naive Bayes*classifier5 = GaussianNB()classifier5.fit(X\_train, y\_train)y\_pred5 = classifier5.predict(X\_test) *#Decision tree*classifier6 = DecisionTreeClassifier(criterion="entropy", random\_state=0)classifier6.fit(X\_train, y\_train)y\_pred6 = classifier6.predict(X\_test) *#Random Forest*classifier7 = RandomForestClassifier(n\_estimators=72, criterion="entropy", random\_state=0)classifier7.fit(X\_train, y\_train)y\_pred7 = classifier7.predict(X\_test) *#Adaboost*classifier8 = AdaBoostClassifier()classifier8.fit(X\_train, y\_train)y\_pred8 = classifier8.predict(X\_test) *#Gradient Boost*classifier9 = GradientBoostingClassifier()classifier9.fit(X\_train, y\_train)y\_pred9 = classifier9.predict(X\_test) *#Voting Classifier*classifier10 = VotingClassifier(estimators=[('gbc', GradientBoostingClassifier()), ('lr', LogisticRegression()), ('abc', AdaBoostClassifier())], voting='soft')classifier10.fit(X\_train, y\_train)y\_pred10 = classifier10.predict(X\_test)

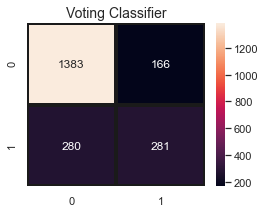
lr = model\_evaluation(y\_test, y\_pred, "Logistic Regression")svm = model\_evaluation(y\_test, y\_pred2, "SVM (Linear)")knn = model\_evaluation(y\_test, y\_pred3, "K-Nearest Neighbours")k\_svm = model\_evaluation(y\_test, y\_pred4, "Kernel SVM")nb = model\_evaluation(y\_test, y\_pred5, "Naive Bayes")dt = model\_evaluation(y\_test, y\_pred6, "Decision Tree")rf = model\_evaluation(y\_test, y\_pred7, "Random Forest")ab = model\_evaluation(y\_test, y\_pred8, "Adaboost")gb = model\_evaluation(y\_test, y\_pred9, "Gradient Boost")vc = model\_evaluation(y\_test, y\_pred10, "Voting Classifier")

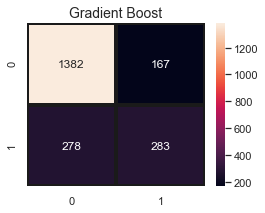
eval\_ =lr.append(svm).append(knn).append(k\_svm).append(nb).append(dt).append(rf).append(ab).append(gb).append(vc).sort\_values(["Precision","Recall", "F2 Score"], ascending = False).reset\_index().drop(columns = "index")eval\_

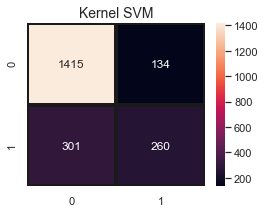
Model Accuracy Precision Recall F1 SCore F2 Score0 Adaboost 0.812322 0.687927 0.538324 0.604000 0.5628031 Voting Classifier 0.808531 0.675615 0.538324 0.599206 0.5611302 Gradient Boost 0.805213 0.672018 0.522282 0.587763 0.5466423 Kernel SVM 0.793839 0.659898 0.463458 0.544503 0.4927984 Logistic Regression 0.805687 0.658281 0.559715 0.605010 0.5769945 Random Forest 0.796682 0.650685 0.508021 0.570571 0.5313206 K-Nearest Neighbours 0.789100 0.628889 0.504456 0.559842 0.5252417 SVM (Linear) 0.788626 0.628635 0.500891 0.557540 0.5221118 Naive Bayes 0.756872 0.531088 0.730838 0.615154 0.6797089 Decision Tree 0.732701 0.497364 0.504456 0.500885 0.503022

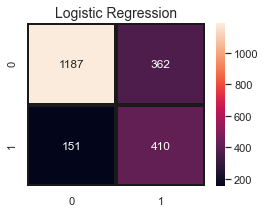
predictions = [y\_pred, y\_pred2 , y\_pred3, y\_pred4, y\_pred5, y\_pred5, y\_pred6, y\_pred7, y\_pred8, y\_pred9, y\_pred10]**for** i, j **in** zip(predictions, eval\_.Model.values): plt.figure(figsize=(4,3)) sns.heatmap(confusion\_matrix(y\_test, i), annot=True,fmt = "d",linecolor="k",linewidths=3) plt.title(j,fontsize=14) plt.show()

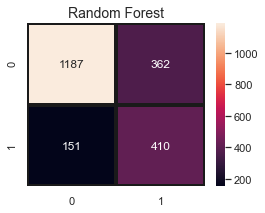


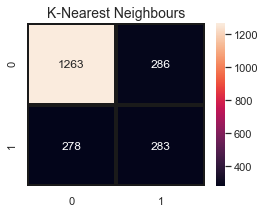


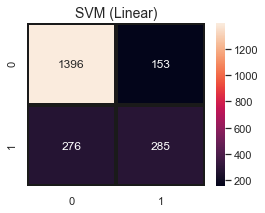


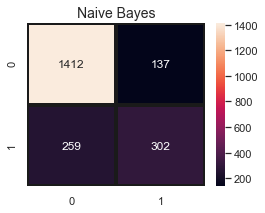


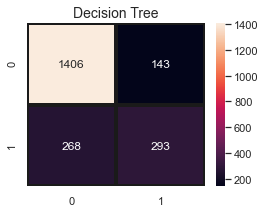












*#TODO: Model Evaluation*

**def** k\_fold\_cross\_validation(classifier\_name, name): accuracies = cross\_val\_score(estimator=classifier\_name, X=X\_train, y=y\_train, cv =10) print(name, "accuracy: %0.2f (+/- %0.2f)" % (accuracies.mean(), accuracies.std() \* 2))

k\_fold\_cross\_validation(classifier8, "Adaboost")

Adaboost accuracy: 0.80 (+/- 0.03)

k\_fold\_cross\_validation(classifier10, "Voting classifier")

Voting classifier accuracy: 0.80 (+/- 0.04)

k\_fold\_cross\_validation(classifier9, "Gradient Boost classifier")

Gradient Boost classifier accuracy: 0.80 (+/- 0.04)

k\_fold\_cross\_validation(classifier, "Logistic regression")

Logistic regression accuracy: 0.80 (+/- 0.04)

k\_fold\_cross\_validation(classifier4, "Kernel SVM")

Kernel SVM accuracy: 0.80 (+/- 0.03)